POLITECNICO DI TORINO
Repository ISTITUZIONALE

Benchmarking Co-Simulation Orchestration Engines for Integrated Energy Systems: A Comparative Study of Mosaik and HELICS

Original
Benchmarking Co-Simulation Orchestration Engines for Integrated Energy Systems: A Comparative Study of Mosaik and HELICS / Chini, F., Canali, D., Mazzarino, P.R., Schiera, D.S., Barbierato, L., Bottaccioli, L., Patti, E., Margara, A.. - (2026), pp. 1-6. (4th International Workshop on Open Source Modelling and Simulation of Energy Systems (OSMSES 2026) Karlsruhe (DEU) March 23-25, 2026) [10.1109/osmses69376.2026.11457198].

Availability:
This version is available at: 11583/3010134 since: 2026-04-21T10:39:06Z

Publisher:
IEEE

Published
DOI:10.1109/osmses69376.2026.11457198

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
IEEE postprint/Author's Accepted Manuscript

©2026 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

25 November 2026

Benchmarking Co-Simulation Orchestration Engines for Integrated Energy Systems: A Comparative Study of Mosaik and HELICS

Fabio Chini*, Davide Canali*, Pietro Rando Mazzarino†, Daniele Salvatore Schiera†,
Luca Barbierato†, Lorenzo Bottaccioli†, Edoardo Patti† e Alessandro Margara*
*DEIB, *Politecnico di Milano*, Milan, Italy. Email: name.surname@polimi.it
†*Energy Center LAB*, *Politecnico di Torino*, Turin, Italy. Email: name.surname@polito.it

***Abstract*—Integrated Energy Systems (IES) are inherently complex, requiring co-simulation tools capable of integrating heterogeneous models and capturing diverse dynamics. This paper presents a comparative benchmark of two leading Co-Simulation Orchestration Engines (COEs) for IES: Mosaik and HELICS. We evaluate the performance and scalability of these engines within a containerized, distributed co-simulation platform based on Kubernetes. Following a qualitative comparison of their programming and execution models, we benchmark the execution time of each engine using a realistic district-scale IES scenario, varying the complexity of the simulation and the underlying computational resources. Our results demonstrate that HELICS exhibits superior scalability compared to Mosaik, achieving a more efficient use of resources as the complexity of the simulation increases. We analyze the impact of key architectural features to explain the observed performance differences. We also evaluate the benefits of a multi-broker topology for HELICS, assessing its resilience to overhead and its implications for future geographically distributed co-simulation workflows. Our study may guide the selection and optimization of COEs for the next generation of IES co-simulation tools.**

***Index Terms*—Integrated Energy Systems, Co-Simulation Orchestration Engines, Mosaik, HELICS, benchmark**

I. Introduction

Integrated Energy Systems (IES) are increasingly characterized by strong interdependencies among physical, cyber, and environmental domains [1]. Analyzing these systems requires simulation tools capable of capturing diverse dynamics across buildings, electrical grids, thermal networks, control layers, and human behavior. Because these systems are inherently multi-domain, their modeling often involves integrating different paradigms such as equation-based, event-based, or agent-based models. For each of them, modelers rely on specialized, domain-specific simulators to provide high-fidelity representations within their respective fields [2], [3].

As the complexity of modern energy systems grows, combining these specialized tools into unified simulation scenarios becomes critical. Co-simulation techniques offer a viable solution by enabling different simulators to interact while maintaining their internal modeling assumptions and numerical methods. Over the past decade, interoperability standards such as the Functional Mock-up Interface (FMI) [4] and the System Structure and Parameterization (SSP) [5] have made the integration of heterogeneous models increasingly accessible.

However, effective co-simulation requires more than just interoperability standards; it relies on a dedicated execution engine to coordinate time advancement, manage data routing among simulators, and ensure causal consistency. This orchestration layer, often referred to as a co-simulation master, is crucial for determining the scalability, robustness, and usability of the overall system [6]. In this context, two Co-simulation Orchestration Engines (COEs) have gained notable attention: Mosaik [7] and HELICS [8]. Their underlying philosophies differ significantly. Mosaik employs a centralized orchestration model, where a single master controls the global simulation time and coordinates all interactions. In contrast, HELICS adopts a decentralized broker-based architecture, enabling simulators (federates) to exchange messages through a publish/subscribe [9] infrastructure that facilitates peer-to-peer communication and distributed time management. These architectural differences greatly impact scalability, ease of configuration, and the capability to support distributed execution.

Despite their increasing adoption, both simulation engines require significant expertise. Effective co-simulation necessitates a thorough understanding of data dependencies, time synchronization semantics, and the orchestration logic embedded in each engine. Mosaik provides a simpler user experience through its unified orchestrator, making it attractive for rapid prototyping; however, this simplicity may limit its scalability in communication-intensive scenarios. Conversely, HELICS offers greater flexibility due to its distributed design, but configuring its federation can be more complex.

This paper contributes to the field by systematically benchmarking Mosaik and HELICS within a distributed, containerized co-simulation platform [10]–[12]. We evaluate the computational scalability of these engines using a realistic IES scenario. Additionally, we analyze how their orchestration strategies impact data exchange, synchronization behavior, and overall usability. Beyond performance metrics, this study

This work is supported by COMET - Co-simulation of Multi-energy systems for Energy Transition, an Italian National funded project under PNRR M4C2, Investimento 1.4 - Avviso n. 3138 del 16/12/2021 - CN00000013 National Centre for HPC, Big Data and Quantum Computing (HPC).

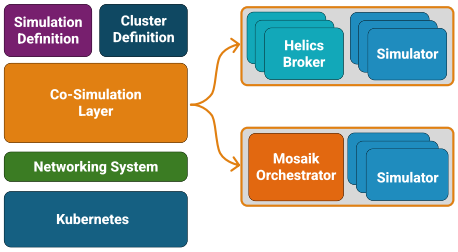

Fig. 1: Architecture of the distributed co-simulation platform.

emphasizes qualitative factors important to practitioners, such as configuration effort and the trade-offs between centralized and decentralized orchestration. Our goal is to provide a well-founded comparison that supports informed decision-making when designing co-simulation workflows for next-generation energy systems.

The paper is organized as follows. Sec. II presents the motivations behind this work, introducing the co-simulation platform we aimed to build and its requirements. Sec. III overviews the programming, deployment, and execution models of Mosaik and HELICS, and Sec. IV benchmarks their performance in a realistic simulation scenario. Finally, Sec. V draws some conclusions based on our findings and suggest future research directions.

II. Motivations and Requirements

As discussed in Sec. I, developing realistic co-simulation workflows for IES involves executing heterogeneous simulators, often characterized by different time semantics, numerical models, and communication needs, within a scalable and distributed environment. The primary objective of our research was to define a robust platform capable of handling these complexities. This development process inevitably raised the question of selecting suitable COEs to power the execution. To address this, we integrated two leading solutions, Mosaik and HELICS, assessing their programming models, execution patterns, deployment strategies, absolute performance, and scalability.

A. Platform Architecture

This work extends the architectural framework originally presented in [12], [13]. While the original system was designed exclusively around the Mosaik co-simulation engine, the architecture proposed herein has been significantly re-engineered to support the HELICS platform as well. This extension enables a fair comparison between the two engines within the same controlled environment.

Despite the transition to a multi-engine core, the platform remains grounded in three foundational design principles: (i) a *declarative specification* of both the simulation scenario and the deployment environment; (ii) strict *modularity and reusability* of simulators; and (iii) a rigorous *separation of concerns* between scenario modeling and infrastructure deployment. These principles underpin the high-level architecture illustrated in Fig. 1.

Declarative Configuration. User interaction is mediated through a *declarative interface*. Users define the co-simulation scenario via a *simulation definition*: a set of YAML documents that enumerate the simulators, their configuration parameters, and the topology of their interconnections. Orthogonally, users describe the target computing resources in a separate *cluster configuration* file. This decoupling allows scenarios to be hardware-agnostic, enabling seamless migration across different infrastructures.

Simulation and Execution Management. The *co-simulation layer* automates the instantiation of (i) the individual *simulators* required by the scenario, and (ii) the necessary orchestration components (i.e., Mosaik orchestrator or HELICS brokers). Both simulators and brokers or orchestrator are distributed as Docker[1] containers. This containerization ensures that each simulator operates in an isolated environment meeting its specific runtime dependencies, while providing the flexibility to inject custom configurations as needed.

Distributed Infrastructure. The execution is supported by a robust, multilayered infrastructure designed for horizontal scalability. At the *networking system layer*, inter-container communication is managed by Flannel[2], which enforces fine-grained control over networking policies. At the *infrastructure control level*, Kubernetes[3] governs the distribution of containers across physical hosts, providing mechanisms for automated scheduling, load balancing, and failover.

Operational Workflow. The platform functions as a coordination system that synthesizes these layers into a cohesive runtime environment. Upon receiving the user's definitions, the system dynamically generates Kubernetes manifests to dictate the optimal placement of simulator and broker containers. Simultaneously, it injects specific context data, providing simulators with scenario parameters and brokers with the connection topology.

B. Requirements

To effectively evaluate the suitability of Mosaik and HELICS for large-scale IES co-simulation, we identified three key categories of requirements. These guided both the integration effort and the subsequent performance benchmarking:

Programming Model and Usability. The integration of diverse simulators into the platform must require a reasonably low effort. A simple and expressive programming model is essential to facilitate the extension of the platform to new simulators, thereby increasing flexibility regarding supported usage contexts and scenarios. Furthermore, a streamlined integration process accelerates experimentation by reducing the development time required for alternative simulator versions or configurations.

Absolute Performance. The COE must ensure high performance to support complex simulation scenarios. As IES

[1]https://www.docker.com/
[2]https://github.com/flannel-io/flannel
[3]https://kubernetes.io/

models grow to include district-scale aggregations with tight coupling between domains (e.g., power grids and thermal networks), the overhead introduced by the orchestration layer must remain minimal to allow for viable execution times.

Scalability. Beyond baseline performance, the COE must exhibit strong scalability in two dimensions. First, it should scale efficiently as the dimension of the simulation scenario increases. Second, it must effectively exploit available computational resources; we require the software to leverage additional hardware (e.g., adding nodes to the cluster) to manage larger scenarios proportionally, avoiding bottlenecks that prevent the utilization of the distributed infrastructure.

III. Orchestration Engines Under Test

In this section, we analyze Mosaik and HELICS focusing on their programming, deployment and execution models.

A. Programming Models

From a user's perspective, Mosaik and HELICS offer distinct paradigms for configuring and organizing co-simulations, differing primarily in abstraction level and control flow management.

Mosaik. Mosaik utilizes a component-based design with a compact abstraction. Simulators are wrapped in a Python-based component class that exposes metadata regarding inputs, outputs, and temporal behavior. Users structure experiments through a Python script that instantiates simulators, connects interfaces based on data-flow rules, and specifies the global timeline. In Mosaik, the control flow is implicit, as the interaction follows an inversion of control pattern: Developers implement a standard interface (API), and the Mosaik engine manages the simulation loop externally by triggering lifecycle methods (callbacks), such as `step()`, on the simulator instance. This approach abstracts the low-level communication logic, allowing users to focus on the model behavior.

HELICS. HELICS adopts a federated, general-purpose approach. It operates as a federation of independent executables (federates) that communicate via a common interface. Configuration is typically decentralized or defined via JSON/TOML configuration files for each federate, though a broker hierarchy must be established to manage complex federations. In HELICS, the control flow is explicit. Unlike Mosaik's inversion of control, HELICS enforces an imperative execution model where the simulator drives the lifecycle. The operational logic typically follows an *initialize-loop-finalize* pattern entirely managed by the user code. HELICS, as its native purpose is a middleware with some APIs to be used directly, integrating them in the simulator's code. This allows strong model independence. Indeed, each simulator must be modified to be responsible for orchestrating the step execution: it must actively invoke API primitives to request time advancement (blocking until granted), retrieve subscribed inputs from the middleware, and publish computed results. While this explicit handling increases the integration effort, it grants extensive control over synchronization points and supports complex iteration schemes. To mitigate the implementation effort, we have designed some wrapping classes that standardize possible federate templates in order to cover the basic data exchange and synchronization needs.

B. Deployment and Execution Models

The programming models described above translate into fundamentally different runtime architectures. Understanding these differences is essential to interpreting the performance benchmarks analyzed later in this study.

Mosaik employs a strictly centralized architecture. A single orchestration process (the Master) holds the complete dependency graph and coordinates the entire execution.

Execution Logic - Mosaik. The centralized nature enables implicit time management but imposes a polling-based execution model. The orchestrator linearly queries simulators to trigger steps and retrieve data. As the number of simulators (N) grows, the scheduling overhead per time step scales as $O(N)$.

Data Exchange and Overhead - Mosaik. Communication is routed internally by the Master. However, the standard implementation relies on TCP sockets with JSON serialization. In high-frequency exchange scenarios, the CPU overhead required for parsing and serializing JSON messages can become a significant bottleneck. Furthermore, being a pure Python implementation, the orchestrator is constrained by the Global Interpreter Lock (GIL), which inherently limits its ability to exploit multi-core architectures for parallel processing.

HELICS relies on a partially distributed architecture based on a publish-subscribe [9] paradigm. There is no single central master for data routing; instead, a hierarchy of Brokers coordinates the message passing between distributed Cores attached to each federate.

Execution Logic - HELICS. Synchronization is event-driven. Each federate's local HELICS Core manages its own time, updating it based on the timestamps of incoming messages and the global consensus provided by the brokers. This minimizes idle time, as federates only block when necessary for causal consistency. This Logic makes it harder to define co-simulation scenarios with complex data exchange, leading to inconsistencies.

Data Exchange and Overhead - HELICS. Communication in HELICS is topic-based and managed through a hierarchical Broker architecture. Unlike Mosaik's polling mechanism, HELICS adopts a publish–subscribe model, in which federates publish data on named topics and subscribe to those of interest, without direct knowledge of data producers or consumers. Each federate interfaces with the federation through a local Core, which forwards data messages and synchronization signals to the Broker layer. Brokers resolve subscriptions and route messages to the appropriate destination Cores based on topic matching. While this routing is logically centralized, Brokers can be organized hierarchically, enabling distributed execution and scalable message propagation. The Core and Broker layers are implemented in C++ and use efficient binary

communication, substantially reducing serialization overhead compared to Mosaik's Python-based JSON parsing. Regarding network traffic, we observed that most exchanged messages are related to time coordination (time requests and grants) rather than payload data, resulting in a network load profile that differs markedly from Mosaik.

IV. Performance Results

Following the qualitative comparison of the programming and execution models, we now focus on the quantitative evaluation of the two orchestration engines. We conducted a series of experiments to benchmark the absolute performance and scalability of Mosaik and HELICS within the distributed platform, aiming to understand how their architectural differences impact runtime efficiency under increasing complexity.

A. Simulation Scenario

The benchmark scenario represents an evolution of the integrated urban energy system model introduced in our previous work [13]. While the original model focused on energetic feasibility, this iteration has been specifically extended to stress-test the orchestration engines under realistic, communication-intensive conditions. The core setup models a district-scale energy community composed of multiple residential buildings. Each building is simulated by a dedicated set of components: an FMU-based thermodynamic model (EnergyPlus), a behavioral model for occupants (HBM), rooftop photovoltaic generation (PV), an electric heat pump (EHP), and a battery storage system (BT). These components interact through continuous data exchanges, capturing heat flows, electricity consumption, generation, and storage dynamics. To stress-test the orchestration engines under realistic, communication-intensive conditions, we configured the scenario to include a Power Grid simulator that aggregates electrical demand from all buildings to compute the global grid state. This topology establishes a global coupling among all building-level units. Consequently, every time step requires the exchange of aggregated signals between every building and the grid. This enforces a strict synchronization lock-step across the entire cluster, preventing simulators from "drifting" ahead in time and significantly increasing the communication load on the orchestrator.

B. Hardware and Software Configurations

The scenario is executed on the distributed platform described in Sec. II. Each simulator runs inside a containerized environment deployed on a Kubernetes cluster. This setup enables us to scale the number of buildings, and thus the number of simulators, in a controlled manner while distributing the computational load across multiple nodes. The Kubernetes cluster is hosted on five virtual machines: (1) one *leader node* (16-core 2GHz Xeon CPU, 64GiB memory) hosts the Kubernetes Control Plane; (2) four *worker nodes* (32-core 2GHz Xeon CPU, 128GiB memory each) serve as the runtime environment for simulation pods.

We evaluated the total execution time across three configurations: i) *Mosaik.* The centralized Mosaik orchestrator, managing the entire simulation from a single process. ii) *HELICS (Single Broker).* Helics orchestrator using a centralized topology where a single broker manages all federates. iii) *HELICS (Multi-Broker).* Helics orchestrator using a hierarchical topology where each Kubernetes Pod contains a local broker managing its internal simulators, connected upstream to a global cluster broker. These configurations were tested on cluster sizes of 1, 2, and 4 worker nodes, varying the number of buildings (and thus simulators) on a logarithmic scale. All experiments were repeated 5 times; we report the average and standard deviation of the measurements.

C. Performance and Scalability Analysis

1) Absolute Performance Comparison: Our primary comparison focuses on the total execution time as the scale of the simulation increases (Fig. 2). The results demonstrate a significant performance gap between the two engines. In general, HELICS outperforms Mosaik, especially in complex scenarios, where the gap is more than an order of magnitude.

The Mosaik configuration exhibits a steep, super-linear increase in execution time as the number of simulators grows. Notably, for the largest scenarios involving 128 and 256 buildings, the Mosaik-based simulation failed to complete within a reasonable timeframe (timeout set at 1 hour), regardless of the number of worker nodes allocated. This suggests that the bottleneck lies within the centralized orchestration logic rather than infrastructure capacity.

As detailed in Sec. III, the divergence between Mosaik and HELICS is driven by two architectural factors: serialization and scheduling. Mosaik's reliance on JSON creates substantial CPU overhead for parsing, whereas HELICS's binary protocol drastically reduces traffic and processing time. Furthermore, Mosaik's polling-based scheduling ($O(N)$) becomes a limiting factor for large N, whereas HELICS's event-driven approach maintains efficiency under higher load.

2) HELICS Scalability Analysis: Focusing on the HELICS performance curves (Fig. 3), we observe a highly efficient scalability profile. The system demonstrates near-linear scalability with respect to the available hardware resources. Specifically, we observed that the platform can simulate 64 buildings on a single worker node, 128 buildings on two nodes, and 256 buildings on four nodes with a comparable execution time (approximately 100 seconds).

The shape of the curves further highlights how the bottleneck shifts from software to hardware. When the number of simulators is lower than the available physical CPU cores, the execution time remains nearly constant, indicating that the orchestration overhead is negligible and the platform effectively parallelizes the load. As the number of simulators exceeds the physical core count, the slope increases linearly due to resource contention (CPU saturation). However, as shown by the comparison across cluster sizes, adding new worker nodes effectively flattens the curve again, confirming that HELICS allows the simulation capacity to grow proportionally with the infrastructure.

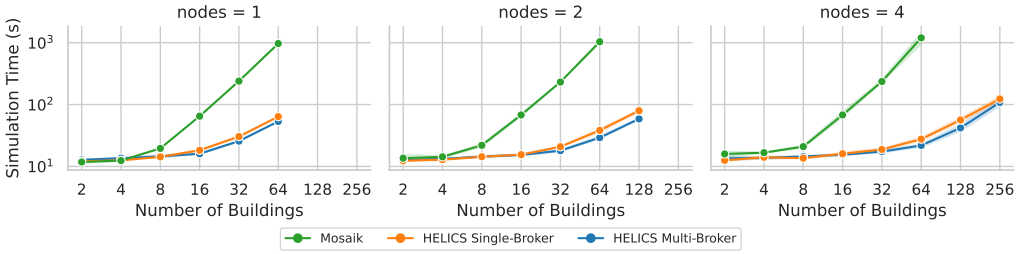

Fig. 2: Execution time comparison of Mosaik and HELICS with 1, 2, and 4 worker nodes.

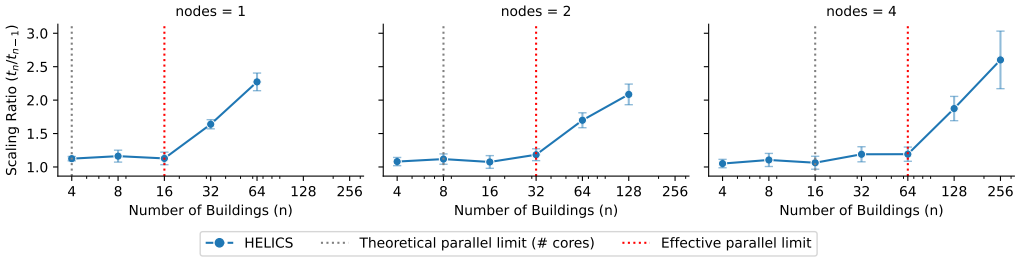

Fig. 3: Execution time scaling ratio between consecutive building counts

TABLE I: Comparative Analysis of Simulation Data Packets

Metric	HELICS	Mosaik
Total number of packets	27,632	9,735
Number of data packets	10,124	4,964
Avg. payload size	67.01 bytes	68.20 bytes
Duration	12.72 s	16.16 s
Avg. Throughput	426.67 kbps	167.59 kbps

D. Network Traffic Analysis

To quantify the overhead of the distributed time synchronization system of HELICS and the JSON-Python serialization protocol used by Mosaik, we performed a comparative analysis using a simulation of two buildings, where we measure the packets produced and their average size (Table I) for a two-days simulation. To ensure an accurate representation of the communication load generated by the simulation logic, our analysis focuses specifically on the messages originating from the individual simulators and directed towards the central orchestration entity (i.e., the HELICS broker or the Mosaik orchestrator). We explicitly excluded acknowledgment packets (ACKs) and the subsequent forwarding of messages from the orchestrator to destination simulators. This filtering approach prevents double-counting, as the communication inherently involves a two-step delivery proce ss where the same logical payload traverses the network twice: first to the broker, and then to the recipient.

In the HELICS environment, we measured 27,632 messages sent from the simulators to the broker, representing 30.5% of the total network traffic. Out of them, 10,124 were actual data packets (type `cmd_pub`), while the others were related to time-management: 9,092 were time requests (type `cmd_time_request`), and 8,416 time grants (type `cmd_time_grant`). This distribution confirms that time-management signals constitute a significant portion of the communication overhead. In contrast, Mosaik generated 9,735 messages sent to the orchestrator, accounting for 35.4% of its total traffic. Out of them, 4,964 were actual data packets.

Focusing only on data packets, the average payload sizes are nearly identical (≈68 bytes). However, HELICS achieves a throughput that is approximately 2.6× higher than Mosaik. This disparity highlights some performance bottlenecks in the Mosaik architecture. Mosaik relies on Python-based JSON (de)serialization. Python's interpreted nature, combined with the text-based parsing of JSON, introduces significant CPU latency for every message processed by the orchestrator. The lower throughput (167.59 kbps vs. 426.67 kbps) suggests that the orchestrator spends a disproportionate amount of time on message handling and context switching within the Python Global Interpreter Lock (GIL).

These results could indicate that the communication layer is a primary candidate for optimization within Mosaik. Implementing a compiled C++ core or adopting binary serialization formats (e.g., Protobuf or FlatBuffers) could yield substantial returns in simulation speed, potentially bringing Mosaik's performance closer to the high-efficiency baseline established by HELICS.

E. Analysis of Topology (Single vs. Multi-Broker)

Comparing the HELICS Single Broker and Multi-Broker configurations reveals that performance lines are nearly identical in our test environment. This indicates that within a high-performance LAN, the overhead of introducing an additional hierarchical layer of brokers is negligible.

However, the significance of the Multi-Broker topology extends beyond LAN performance. Preliminary experiments

conducted in geographically distributed environments (not detailed in this study) suggest that this hierarchy becomes crucial when network latency is high. In such contexts, co-locating simulators that communicate frequently with each other with a local broker, thereby keeping high-frequency synchronization traffic within the same node, can yield substantial performance benefits.

V. Discussion and Conclusions

In this work, we presented a comparative evaluation of Mosaik and HELICS, two prominent orchestration engines for Integrated Energy Systems, integrated within a unified, containerized distributed platform. Our benchmark highlights critical trade-offs between usability, architectural flexibility, and raw performance, providing guidelines for researchers and practitioners in selecting the appropriate tool for their specific constraints.

1) Architectural Trade-offs and Usability: While the experimental results demonstrate HELICS's superior performance in complex scenarios, Mosaik remains a strong contender for specific use cases. Its centralized, Python-centric abstraction lowers the barrier to entry, making it ideal for rapid prototyping, educational purposes, and smaller-scale experiments where ease of configuration outweighs execution speed. However, as system complexity grows, the scheduling overhead inherent to Mosaik's centralized design introduces significant latency. Conversely, HELICS demands a steeper learning curve and a higher initial investment in configuration to manage the distributed synchronization logic. However, it rewards this effort with robustness and scalability, proving to be the necessary choice for large-scale, high-fidelity simulations. Future research in the co-simulation domain should focus on bridging this gap: enhancing HELICS's high-level abstractions to simplify user onboarding and exploring event-driven scheduling improvements for Mosaik to mitigate its performance bottlenecks.

2) Performance and Scalability Profiles: The experimental analysis revealed distinct scalability profiles for the two engines:

Vertical scalability. The Mosaik-based configuration exhibited limited vertical scalability. Execution times increased super-linearly well before hardware saturation, primarily driven by the single-threaded nature of the orchestrator (and Python's GIL) combined with the overhead of JSON serialization. In contrast, HELICS, leveraging a C++ core and efficient binary protocols, demonstrated excellent vertical scalability, maintaining a flat performance profile until physical CPU cores were fully saturated.

Horizontal scalability. The experiments validated HELICS's ability to scale horizontally. By decoupling logical simulation groups across multiple nodes, the platform successfully shifted the bottleneck from the orchestration software to the physical computing resources. Furthermore, the Multi-Broker topology introduced no measurable overhead in a LAN environment, confirming that the architecture can expand seamlessly with added hardware resources without incurring synchronization penalties.

3) Future Work: While this study validated the efficiency of the hierarchical architecture within a high-performance cluster, we anticipate its true potential will emerge in geographically distributed environments. In scenarios spanning multiple data centers, network latency becomes a dominant factor. Building on our findings regarding the Multi-Broker topology, our future work will move beyond engine benchmarking to focus on *deployment optimization*. We aim to develop algorithms that determine the optimal allocation of simulators and brokers across distributed nodes by co-locating highly coupled components to minimize communication latency and maximize overall system efficiency in heterogeneous infrastructure.

References

[1] J. Wu, J. Yan, H. Jia, N. Hatziargyriou, N. Djilali, and H. Sun, "Integrated energy systems," *Applied Energy*, vol. 167, pp. 155–157, 2016.

[2] C. Gomes, C. Thule, D. Broman, P. G. Larsen, and H. Vangheluwe, "Co-simulation: A survey," *ACM Comput. Surv.*, vol. 51, no. 3, May 2018.

[3] P. Palensky, A. A. Van Der Meer, C. D. Lopez, A. Joseph, and K. Pan, "Cosimulation of intelligent power systems: Fundamentals, software architecture, numerics, and coupling," *IEEE Industrial Electronics Magazine*, vol. 11, no. 1, pp. 34–50, 2017.

[4] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauß, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold *et al.*, "The functional mockup interface for tool independent exchange of simulation models," in *Proceedings of the 8th international Modelica conference*. Linköping University Press, 2011, pp. 105–114.

[5] Modelica Association Project "System Structure and Parameterization", *System Structure and Parameterization (SSP) Standard, Version 2.0*, Modelica Association, 2024, version 2.0, released December 20, 2024.

[6] J. Bastian, C. Clauß, S. Wolf, and P. Schneider, "Master for co-simulation using fmi," in *8th International Modelica Conference*, vol. 2011. Linköping University Electronic Press, Linköpings universitet Dresden, Germany, 2011.

[7] C. Steinbrink, M. Blank-Babazadeh, A. El-Ama, S. Holly, B. Lüers, M. Nebel-Wenner, R. P. Ramírez Acosta, T. Raub, J. S. Schwarz, S. Stark, A. Nieße, and S. Lehnhoff, "Cpes testing with mosaik: Co-simulation planning, execution and analysis," *Applied Sciences*, vol. 9, no. 5, 2019.

[8] T. D. Hardy, B. Palmintier, P. L. Top, D. Krishnamurthy, and J. C. Fuller, "Helics: A co-simulation framework for scalable multi-domain modeling and analysis," *IEEE Access*, vol. 12, pp. 24 325–24 347, 2024.

[9] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Comput. Surv.*, vol. 35, no. 2, p. 114–131, jun 2003.

[10] P. R. Mazzarino, M. Capone, E. Guelpa, L. Bottaccioli, V. Verda, and E. Patti, "A modular co-simulation platform for comparing flexibility solutions in district heating under variable operating conditions," *IEEE Transactions on Sustainable Computing*, vol. 10, no. 2, pp. 408–417, 2025.

[11] D. S. Schiera, L. Barbierato, A. Lanzini, R. Borchiellini, E. Pons, E. Bompard, E. Patti, E. Macii, and L. Bottaccioli, "A distributed multi-model platform to cosimulate multienergy systems in smart buildings," *IEEE Transactions on Industry Applications*, vol. 57, no. 5, pp. 4428–4440, 2021.

[12] F. Chini, D. Canali, L. Barbierato, D. S. Schiera, L. Bottaccioli, A. Margara, and E. Patti, "Streamlined deployment of a distributed and scalable co-simulation platform for integrated energy systems," in *International Conference on Intelligent Systems Applications to Power Systems*, ser. ISAP 2024. IEEE, 2024, pp. 1–6.

[13] L. Barbierato, D. Salvatore Schiera, M. Orlando, A. Lanzini, E. Pons, L. Bottaccioli, and E. Patti, "Facilitating smart grids integration through a hybrid multi-model co-simulation framework," *IEEE Access*, vol. 12, pp. 104 878–104 897, 2024.