The Journal of Open Source Software

SerializableSimpy: Parallel and Serializable Discrete-Event Simulation in Python

Pierrick Pochelu [1]

1 HPC Platform, University of Luxembourg, Luxembourg

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Prashant Jha

Reviewers:

- @freifrauvonbleifrei
- @EmilyBourne

Submitted: 05 May 2025
Published: unpublished

License
Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License (CC BY 4.0).

Summary

SerializableSimpy is a Python framework for building discrete-event simulations (DES). DES modeling and analysis are useful in time-sensitive applications, such as manufacturing, logistics, and distributed systems. The framework provides core DES building blocks, including logical processes (LPs) and event causality. It also includes components for state and synchronization, such as stores, resources, and priority queues. Users focuse on business logic, not the event-processing engine. After modeling the system, users can start, pause, and resume.

SerializableSimpy extends the widely used SimPy package (Zinoviev, 2024) by introducing *serializability* and *parallel execution capabilities* (Fujimoto, 2017). Unlike SimPy, which relies on non-serializable generator code (Vassalotti, 2009), SerializableSimpy implements context switches between LPs using standard Python function calls, with an event queue of Python callables. In other words, code that would use *yield* in SimPy is replaced with functions that finish using *return*.

This design enables:

- **Effortless checkpointing and full recovery** of simulation state, including LPs and event queues, using standard serialization tools such as pickle (Foundation, 2023b). It requires no serialization logic from users.
- **Parallel Discrete-Event Simulation** (Fujimoto, 2017) using `multiprocessing` (Foundation, 2023a) or `mpi4py` (Dalcin et al., 2011), with support for inter-process event exchange through shared memory or message passing.

SerializableSimpy reuses familiar SimPy abstractions where possible, while offering a fundamentally different execution engine. It is not a drop-in replacement, it is designed for users who require reproducibility, parallel performance scaling, or integration into networked systems.

SerializableSimpy includes a tutorial (located in the `application/` folder) that reproduces selected official SimPy examples (e.g., packet-based network latency, customer queuing with reneging) to facilitate a smooth transition between the two frameworks. To support its focus on large-scale simulation, the package also provides benchmark scripts for comparing the performance of SimPy and SerializableSimpy when running on a single CPU core.

Statement of Need

SimPy is widely adopted in both academia and industry due to its simplicity and expressiveness. However, its reliance on Python generators makes it unsuitable for simulations requiring state checkpointing, multiprocessing, or execution across machines in a computer network. This limits its use in modern digital twin workflows, especially in large-scale applications.

SerializableSimpy addresses this need by providing a fully serializable DES framework that offers API similar to SimPy but avoids generators internally. By replacing generator-based flow control with callback-based logic, SerializableSimpy enables features such as:

- Running simulations in parallel with inter-process communication.
- Events portability across networked systems.
- State checkpointing of the simulation environment (and its internal event queue) using serialization.

State of the PDES Field

Over the years, a number of open-source parallel discrete-event simulation (PDES) frameworks have been developed to efficiently model systems and processes involving very large numbers of events. While they differ in scope and design, they share a common goal: to provide software developers and scientists with building blocks for constructing efficient simulators. It is important to note that these frameworks are not simulators themselves, but rather foundational components for composing and processing events.

The table below provides an overview of selected general-purpose PDES tools, focusing on their implementation languages, most recent known activity, and the core parallelization or decomposition algorithms they employ for event processing. This comparison is intended to contextualize SerializableSimpy's contributions within the broader landscape of discrete-event simulation for large-scale systems.

Although SimPy is not primarily intended for parallel simulation, it is included in the table because of its popularity, expressiveness, and simplicity. It remains a widely used framework for rapid prototyping and is often the framework of choice for computationally less demanding simulations.

Reference	Name	Language	Parallel Algorithms
(Pellegrini, 2021; Vitali et al., 2012)	ROOT-sim	C	Optimistic, anti-messages, DyMeLoR memory manager (Pellegrini et al., 2009; Toccaceli & Quaglia, 2008)
(Carothers et al., 2022; Gonsiorowski, 2016)	ROSS	C	Optimistic & conservative, LZA compression
(Santhi et al., 2015; Seshadri & others, 2023)	Simian	Lua, Python, JS	Conservative, some optimistic support, GPU, Greenlet (G. Team, 2024)
(Los Alamos National Security, 2014; Thulasidasan et al., 2014)	SimX	C++, Python	Conservative, Greenlet (G. Team, 2024)
(SST Team, 2024)	SST-core	C++	Conservative, threading, graph-based LP organization
Poshtkohi (2024)	xSim	C, Fortran	Optimistic w/o rollback; METIS, topological sort, Tarjan (Tarjan, 1972)
(W. Team, 2020)	Warped2	C++	Optimistic, Ladder Queue (Tang et al., 2005), METIS

Reference	Name	Language	Parallel Algorithms
(Liu, 2020; Simulus Team, 2019)	Simulus	Python	YAWNS sync protocol (Nicol, 1993), Greenlet (G. Team, 2024)
(Bergero & Kofman, 2011; P. Team, 2020)	PowerDEVS	C++	Conservative, hierarchical model construction, real-time target
(team, n.d.; Zinoviev, 2024)	Simpy	Python	N/A
(Pochelu, 2025)	SerializableSimpy	Python	Conservative

Experimental highlights (token-ring benchmark):

- **Familiar, rich API:** 19 classes at publication time, inspired by SimPy's 35 classes. Other frameworks include Simian (7 classes) and Simulus (15 classes).
- **Compact event semantics for scale:** ~1M events in the benchmark vs ~4M for SimPy, ~1M for Simian, ~2M for Simulus — fewer queue operations overall.
- **Fastest runtime:** best initialization and main-loop times, driven by fewer event operations and efficient heapq-based priority queues.
- **Parallel-ready:** supports multiprocessing and MPI backends.

Detailed scripts and results: https://gitlab.com/uniluxembourg/hpc/research/cadom/serializable-simpy/-/blob/main/application/pdes_compare/README.md

Past and Ongoing Research Use

SerializableSimpy was developed as part of a research collaboration between the University of Luxembourg and the Goodyear Company. It is currently being integrated into workflows for decomposed, large-scale discrete-event manufacturing simulations, enabling execution through parallel logical processes on multi-core or networked environments. Its simple Python API and minimal software dependencies facilitate rapid prototyping and experimentation. These lightweight dependencies are compatible with Python Just-In-Time (JIT) compilers and have been tested with (Bolz et al., 2009), enhancing performance for many multi-core and distributed workloads.

While SerializableSimpy provides examples and tools to facilitate parallel execution with minimal user effort, users are still required to manually decompose simulations and assign tasks to processing cores. To reduce this burden, ongoing research explores graph-based decomposition techniques (Karypis & Kumar, 1998), with the goal of automating partitioning and enhancing scalability, evaluated on large-scale manufacturing simulations.

Acknowledgment

SerializableSimpy contributor(s) would like to express their gratitude for the fruitful discussions and the collaboration that made this project possible between Goodyear company and the University of Luxembourg.

Supported by the Luxembourg National Research Fund 17941664

Supported by the Ministry of Economy (MECO) 17941664

DRAFT

SerializableSimpy contributor(s) would like to thank the EuroHPC Joint Undertaking and LuxProvide for granting access to the MeluXina supercomputer.

References

Bergero, F., & Kofman, E. (2011). PowerDEVS: A tool for hybrid system modeling and real-time simulation. *SIMULATION*, *87*, 113–132. https://api.semanticscholar.org/CorpusID:15790985

Böhm, S., & Engelmann, C. (2011). xSim: The extreme-scale simulator. *2011 International Conference on High Performance Computing & Simulation*, 280–286. https://doi.org/10.1109/HPCSim.2011.5999835

Bolz, C. F., Cuni, A., Fijalkowski, M., & Rigo, A. (2009). Tracing the meta-level: PyPy's tracing JIT compiler. *Proceedings of the 4th Workshop on the Implementation, Compilation, Optimization of Object-Oriented Languages and Programming Systems*, 18–25. https://doi.org/10.1145/1565824.1565827

Carothers, C. D., Perumalla, K. S., & Fujimoto, R. M. (2022). *Rensselaer's optimistic simulation system*. https://ross-org.github.io/.

Dalcin, L. D., Paz, R. R., Kler, P. A., & Cosimo, A. (2011). Parallel distributed computing using python. *Advances in Water Resources*, *34*(9), 1124–1139. https://doi.org/10.1016/j.advwatres.2011.04.013

Foundation, P. S. (2023a). *Multiprocessing — process-based parallelism*. https://docs.python.org/3/library/multiprocessing.html.

Foundation, P. S. (2023b). *Pickle — python object serialization*. https://docs.python.org/3/library/pickle.html.

Fujimoto, R. M. (2017). Parallel discrete event simulation: The making of a field. In W. K. V. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, & E. Page (Eds.), *Proceedings of the 2017 winter simulation conference* (pp. 262–276). IEEE Press. https://doi.org/10.1109/WSC.2017.8247783

Gonsiorowski, E. (2016). *Enabling extreme-scale circuit modeling using massively parallel discrete-event simulations* [Electronic thesis, Rensselaer Polytechnic Institute]. https://hdl.handle.net/20.500.13015/1686

Karypis, G., & Kumar, V. (1998). METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. *University of Minnesota*, *102*. https://conservancy.umn.edu/items/2f610239-590c-45c0-bcd6-321036aaad56

Liu, J. (2020). Simulus: Easy breezy simulation in python. *2020 Winter Simulation Conference (WSC)*, 2329–2340. https://doi.org/10.1109/WSC48552.2020.9383886

Los Alamos National Security, L. (2014). *SimX: Parallel discrete-event simulation library with python frontend and c++ backend*. https://github.com/sim-x/simx.

Nicol, D. M. (1993). The cost of conservative synchronization in parallel discrete event simulations. *J. ACM*, *40*(2), 304–333. https://doi.org/10.1145/151261.151266

Pellegrini, A. (2021). *ROOT-Sim: The ROme OpTimistic Simulator*. https://root-sim.github.io/core/.

Pellegrini, A., Vitali, R., & Quaglia, F. (2009). Di-DyMeLoR: Logging only dirty chunks for efficient management of dynamic memory based optimistic simulation objects. *2009 ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation*, 45–53. https://doi.org/10.1109/PADS.2009.24

DRAFT

Pochelu, P. (2025). *Serializable-simpy*. https://gitlab.com/uniluxembourg/hpc/research/cadom/serializable-simpy.

Poshtkohi, A. (2024). *Pdes: Parallel discrete event simulation framework*. https://github.com/poshtkohi/pdes

Santhi, N., Eidenbenz, S., & Liu, J. (2015). The simian concept: Parallel discrete event simulation with interpreted languages and just-in-time compilation. *2015 Winter Simulation Conference (WSC)*, 3013–3024. https://doi.org/10.1109/WSC.2015.7408405

Seshadri, P., & others. (2023). *Simian: A fast parallel discrete event simulation engine*. https://github.com/pujyam/simian.

Tang, W. T., Goh, R. S. M., & Thng, I. L.-J. (2005). Ladder queue: An o(1) priority queue structure for large-scale discrete event simulation. *ACM Trans. Model. Comput. Simul.*, *15*(3), 175–204. https://doi.org/10.1145/1103323.1103324

Tarjan, R. (1972). Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, *1*(2), 146–160. https://doi.org/10.1137/0201010

Team, G. (2024). *Greenlet documentation*. https://greenlet.readthedocs.io/en/latest/

Team, P. (2020). *CIFASIS/power-devs: PowerDEVS is an integrated tool for hybrid systems modeling and simulation based on the DEVS formalism*. https://github.com/CIFASIS/power-devs.

team, S. (n.d.). *SimPy*. https://gitlab.com/team-simpy/simpy/.

Team, Simulus. (2019). *Simulus - a discrete-event simulator in python*. https://github.com/liuxfiu/simulus

Team, SST. (2024). *SST core: Structural simulation toolkit parallel discrete event core and services*. https://github.com/sstsimulator/sst-core.

Team, W. (2020). *Warped2: warped simulation kernel*. https://github.com/wilseypa/warped2/.

Thulasidasan, S., Kroc, L., & Eidenbenz, S. (2014). Developing parallel, discrete event simulations in python - first results and user experiences with the SimX library. *2014 4th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH)*, 188–194. https://doi.org/10.5220/0005042701880194

Toccaceli, R., & Quaglia, F. (2008). DyMeLoR: Dynamic memory logger and restorer library for optimistic simulation objects with generic memory layout. *2008 22nd Workshop on Principles of Advanced and Distributed Simulation*, 163–172. https://doi.org/10.1109/PADS.2008.23

Vassalotti, A. (2009). *Why you cannot pickle generators*. https://peadrop.com/blog/2009/12/29/why-you-cannot-pickle-generators/.

Vitali, R., Pellegrini, A., & Cerasuolo, G. (2012, March). Cache-aware memory manager for optimistic simulations. *Proceedings of SIMUTools 2012 - 5th International Conference on Simulation Tools and Techniques*. https://doi.org/10.4108/icst.simutools.2012.247766

Zinoviev, D. (2024). *Discrete event simulation: It's easy with SimPy!* https://arxiv.org/abs/2405.01562

DRAFT