Phoenix: Parallel Event-Driven Simulation in PeerSim for Intelligent Workloads

Susana Vitória Marques, Cláudia Brito, Fábio Coelho
INESC TEC - Institute for Systems and Computer Engineering, Technology and Science, Portugal
{susana.v.marques, claudia.v.brito, fabio.a.coelho}@inesctec.pt

***Abstract*—Simulating large-scale distributed systems with realistic concurrency and heavy per-node computation is increasingly important for studying intelligent workloads. Gossip learning and related decentralized learning approaches are a well-established use case for large-scale simulation, and PeerSim has been a natural platform for such studies because of its support for overlay protocols and experimentation. However, PeerSim's event-driven engine executes sequentially, which becomes a bottleneck when each node performs nontrivial computation, as in decentralized learning and other intelligent distributed workloads. This work presents Phoenix, a multi-threaded extension of PeerSim's event-driven core that executes independent node events in parallel while preserving the original protocol semantics. We report primary experimental results using a canonical averaging protocol and synthetic CPU-bound workloads, measuring CPU utilization and memory behavior on overlays of up to one million nodes. Phoenix reduces simulation time by up to 80% compared to single-threaded PeerSim, while maintaining stable memory usage under fixed seeds. These results show that a multi-threaded engine can significantly extend the practical range of PeerSim for the simulation of large, compute-intensive distributed systems, providing a useful platform for performance and scalability studies in advanced artificial intelligence.**

***Index Terms*—P2P, parallelization, event-driven simulation**

I. Introduction

Peer-to-peer (P2P) simulation has become an important tool not only for studying overlay protocols, but also for evaluating decentralized learning methods in which data and computation remain distributed across nodes. A prominent example is gossip learning [1], where models move, update, and combine over a P2P network without central coordination.

PeerSim [2] is a natural foundation for this line of work because it provides scalable support for P2P overlays and cycle-based and event-driven simulation modes [2]. Despite this, PeerSim's event-driven engine executes all events sequentially on a single thread. This design is effective for lightweight protocols, but limits scalability and realism when simulating systems in which nodes perform heavy computations or model intelligent agents [3], such as gossip-based learning, or CPU-bound tasks within each node. In these cases, the single-threaded event loop turns into a bottleneck: wall-clock time grows, and running large overlays or more demanding workload configurations becomes hard in practice.

This paper presents Phoenix, a multi-threaded extension of PeerSim's event-driven core motivated by decentralized learning solutions such as gossip learning, where the simulator must capture both P2P interaction and nontrivial local computation. Phoenix retains PeerSim's modeling abstractions but parallelizes independent interactions at the same simulation moment, improving runtime while preserving the semantics of the original protocol. It introduces a parallel execution scheme in which simulation time is partitioned into discrete moments, and within each moment, node interactions that do not share participants are processed concurrently by worker threads.

We evaluate Phoenix on overlays of up to one million nodes using a canonical averaging protocol and synthetic compute-intensive tasks, measuring runtime and memory behavior. Across these configurations, Phoenix achieves reductions in simulation time of up to roughly 80% compared to single-threaded PeerSim on a six-core machine, while producing deterministic outcomes under fixed random seeds and maintaining stable memory usage. These results indicate that a characterized multi-threaded engine can substantially extend the practical range of PeerSim for simulating large, compute-intensive distributed systems, providing useful infrastructure for performance and scalability studies of modern workloads.

II. State of the Art

P2P simulators bridge the gap between analytical models, which often rely on simplifying assumptions, and real-world prototypes, which are costly and difficult to control at scale [4], [5]. Prior work has explored a range of P2P simulation frameworks [6]–[8] and abstraction levels, including simulators for specific overlays as well as more general-purpose platforms such as OverSim [9] or PeerSim [2].

Recent research has also moved beyond classic centralized learning toward P2P and gossip-based protocols that remove the need for a central coordinator. This has made decentralized federated learning and gossip learning [1] an active research area, especially for settings where data stay at the edge, and communication must remain fully distributed [10]–[12]. For example, GLow [13], introduced in 2025, extends a learning framework to simulate custom gossip-learning systems over arbitrary network topologies; and P2PFL [14] provides a modular decentralized learning library for P2P networks.

Other work [15], [16] continues this direction by proposing communication-efficient gossip variants, showing that the protocol design space is still evolving rapidly. In this context, the main question for our study is which simulation environment best supports iterative, P2P learning optimization. We choose PeerSim [2] as a particularly suitable baseline because it hosts the pioneer implementation of gossip learning [1] and provides

a modular, scalable P2P simulation environment that emphasizes distributed protocols rather than the learning model itself. This makes it possible to concentrate on improving the effectiveness of gossip and exploring optimizations within the protocols, which is where PeerSim offers the most value for our setting since it allows us to build on an existing framework rather than reconstructing the entire learning pipeline from scratch.

III. Phoenix

PeerSim [2] is a modular P2P simulator that supports both cycle-based and event-driven execution. Its cycle-based engine is highly scalable but abstracts transport details, while the event-driven engine is more realistic but less scalable [2]. This trade-off is especially relevant for decentralized learning workloads built on P2P communication [1] and related optimization methods [16], where simulation must combine overlay dynamics with substantial computation at each interaction.

To overcome these limitations, we propose **Phoenix**, a parallel event execution simulator based on PeerSim to allow scalability. Our approach introduces multi-threaded processing, allowing multiple node events to run concurrently without compromising the simulation.

A. Parallel Simulation Workflow

Figure 2 shows the engine's workflow, starting by loading the configuration file ❶ and running the provided initialization protocols ❷ by scheduling the events ❸ and finally executing the *executeNext()* function ❹, as shown in Figure 1. This function will process an event from the original queue of events sequentially. The event processed can either be a *Message* ❺ or a *Control* ⑤.

If the event is a Message, our solution will differ from the Traditional PeerSim solution by storing that message into a secondary queue: *Queue P* ❻. After the insertion, the *executeNext()* function is again executed ❼.

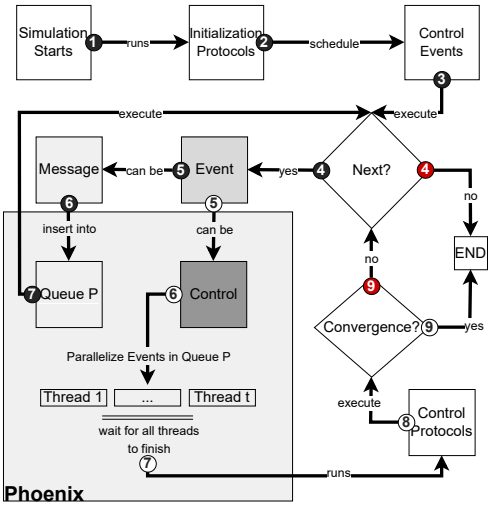


Fig. 1: System workflow.

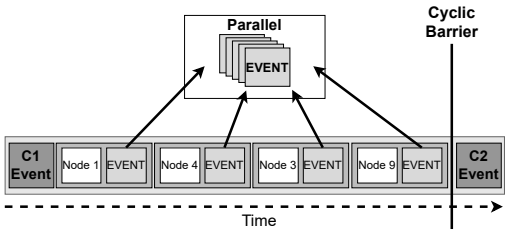


Fig. 2: Parallel execution and synchronization in *Phoenix*.

If the event is a Control, our solution will dequeue all events previously stored in *Queue P* and process them in parallel ⑥ (one event per thread). After all the events are processed, *Phoenix* will behave as the Traditional PeerSim engine and run the control protocols ⑦, coordinating any system operations or checks required for the current moment ⑧. If the ending condition is not met ❾ *e.g.,* all nodes converging to the same value, the process repeats until the queue no longer has any events ❹ or the simulation reaches its stopping criteria ⑨.

B. Concurrency Management

In modern decentralized systems, simulators play a crucial role in modeling tasks such as machine learning, where operations often require the aggregation of data, *e.g.,* model weights, across nodes. For all nodes to converge to a common aggregate value, *e.g.,* the average of the system, protocols need to guarantee mass conservation [17], which means that the total quantity being aggregated (such as the sum of all node values) remains constant throughout the process (no mass is created or lost during the computation). To address this, robust protocols such as the push-pull protocol [18] are used, which require both participating nodes to update their values atomically to ensure global correctness [17].

In our design, each node can take part in at most one interaction per simulation moment. This keeps concurrent interactions disjoint and avoids conflicts. The scheduler moves time forward in discrete steps and, within each step, ensures that no node appears in more than one event (Figure 2). If a node picks a neighbor that is already busy, it simply tries another peer or skips that step. In this way, *Phoenix* adds realistic concurrency to the execution of PeerSim events while still supporting complex P2P protocols safely.

IV. Evaluation

A. Experimental Setup

The experiments were conducted on a machine with an Intel Core i5-9500 processor (3.00GHz, 6 cores/6 threads), 16GB DDR4-2666 memory, a 240GB NVMe SSD, a 500GB HDD, and a 10 Gbps network interface.

The experiments varied the number of threads to see how parallelism affects simulation throughput and scalability. Overlay sizes ranged from 10 to 1,000,000 nodes, so each setup probes how well the simulator copes with growing computational load under different degrees of parallel execution.

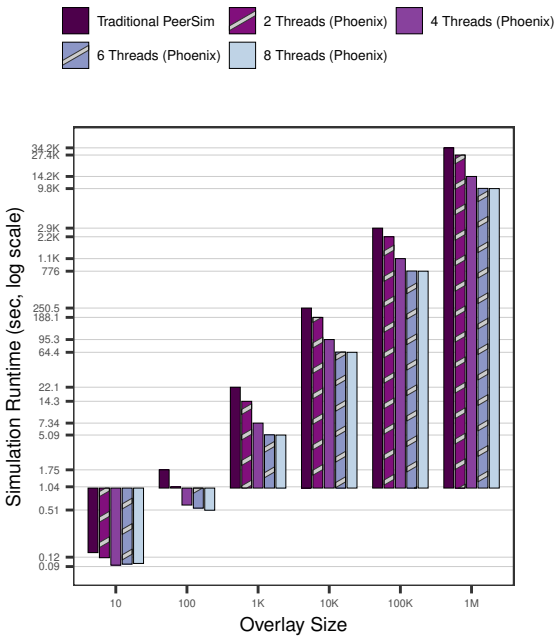


Fig. 3: Simulation runtime for Traditional PeerSim and *Phoenix* across overlay sizes ranging from 10 to 1,000,000 nodes and under varying thread counts (2, 4, 6, and 8).

B. Performance

Figure 3 presents the simulation runtimes for PeerSim (single-threaded) and *Phoenix* using 2, 4, 6 and 8 threads on different overlay sizes. The vertical axis employs a logarithmic scale to clearly convey differences in runtime spanning several orders of magnitude and to allow meaningful comparison of performance improvements at both small and large scales.

a) Single-threaded baseline: Traditional PeerSim slows down in proportion to the overlay size. In our runs it goes from 22.1 seconds at 1,000 nodes to 34,200 seconds at 1,000,000 nodes, which matches its fully sequential per-node processing.

b) Two-thread Configuration: As shown in Table I, our solution with 2 threads reduces runtimes for all overlay sizes. For 1,000 nodes, the runtime is 14.3 seconds). Across 10,000, 100,000, and 1,000,000 nodes, reductions of 25%, 24%, and 20% are observed, respectively. These results demonstrate solid parallel speedup, with the greatest benefit at the smallest scale and diminishing relative gain as overlay size increases, due to increasing serial overhead and memory bandwidth constraints (even though absolute speedups remain significant).

c) Four-thread Configuration:: With 4 threads, *Phoenix* speeds up the runs to 7.34 seconds at 1,000 nodes (67% reduction), 95.3 seconds at 10,000 nodes (62% reduction), and 14,200 seconds at 1,000,000 nodes (58% reduction). The gains are large and fairly stable across all overlay sizes, which matches the expected multi-threaded load distribution.

d) Six-thread Configuration:: In Table I, using 6 threads with our solution delivers the highest observed reduction in simulation runtime, aligning with our machine's hardware limits. Execution times drop to 5.09 seconds (77% reduction) at 1,000 nodes, 64.4 seconds (74% reduction) at 10,000 nodes, and 9,800 seconds (71% reduction) at 1,000,000 nodes. However, increasing to 8 threads yields no further improvement over the six-thread configuration. This plateau is expected, as increasing the number of threads beyond the available physical cores leads to greater CPU resource contention and synchronization overhead, which can offset the performance gains from additional parallelism [19].

Nodes	PeerSim	Phoenix: 2T	Phoenix: 4T	Phoenix: 6T
1K	22.1	14.3 (**35%**)	7.34 (**67%**)	5.09 (**77%**)
10K	250.5	188.1 (**25%**)	95.3 (**62%**)	64.4 (**74%**)
100K	2,900	2,200 (**24%**)	1,100 (**62%**)	776 (**73%**)
1M	34,200	27,400 (**20%**)	14,200 (**58%**)	9,800 (**71%**)

TABLE I: Simulation runtimes (in seconds) for Traditional PeerSim (single-thread) and *Phoenix* with different thread counts (T). Percentages show a reduction in runtime relative to PeerSim.

C. Memory Analysis

Figure 4 shows the memory consumption over time in multiple thread configurations. The data reveal that higher thread counts lead to slightly higher memory usage, with 6 and 8 threads approaching 181.1 MB and 190.7 MB, respectively. Memory usage oscillates within a stable band for all cases, reflecting garbage collection cycles [20] typical under parallel workloads in Java [20]. Configurations with higher thread counts are faster, so their traces end earlier on the time axis, reflecting higher throughput.

Memory remains bounded across all runs and thread counts. Each configuration settles into a repeating allocation–free pattern, and the sharper peaks at higher thread counts match transient thread-local buffers and synchronization overhead rather than leak-like growth.

D. Simulating Tasks

To see how parallel simulation behaves on compute-heavy workloads, we ran experiments that mimic simple machine-learning tasks on overlays with 100 nodes. We used task durations of 1, 5, and 10 seconds to cover different

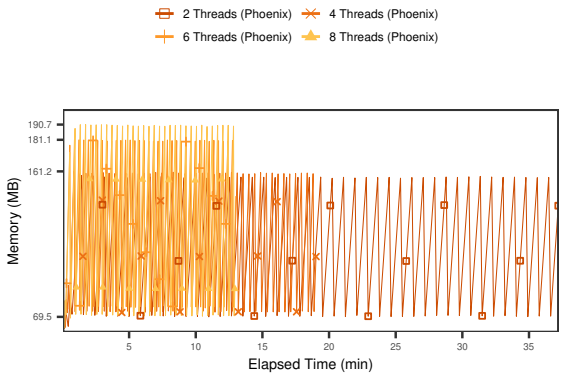
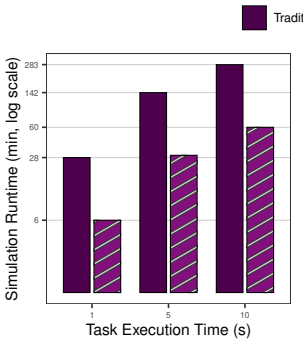


Fig. 4: Memory utilization over time for *Phoenix* running 100,000 nodes, with each simulated task taking 1 millisecond.



Task (s)	**1**	**5**	**10**
PeerSim	28	142	283
Phoenix	6	28	60
Reduction	79%	80%	79%

Fig. 5: Simulation runtime (in minutes) for 100 nodes using Traditional PeerSim and Phoenix (6 threads) with per-node task durations of 1, 5, and 10 seconds.

granularities, and capped them at 10 seconds so the sequential PeerSim runs remain feasible for comparison.

The main results are presented in Figure 5, where the graphic and the table report the overall runtime for different task durations. Across all tested durations, *Phoenix* system consistently outperformed the standard PeerSim. For example, for task durations of 10 seconds, PeerSim required 283 minutes to complete the scenario, whereas *Phoenix* completed the same simulation in 60 minutes. Results show a stable 79% reduction across all configurations, indicating effective resource utilization and favorable scaling as the per-task compute workload increases. This makes Phoenix a practical tool for stress-testing decentralized learning protocols and studying performance and resource behavior.

This experiment demonstrates that parallelization is critical for efficiently simulating modern workloads dominated by heavyweight computations, such as those seen in machine learning pipelines.

V. Conclusion

We present *Phoenix*, a parallel extension of the PeerSim simulator. Our design adds multi-threaded event-driven execution directly to the engine. By reworking the core of PeerSim, *Phoenix* introduces concurrency while maintaining the original semantics and extensibility. In our experiments, *Phoenix* reduces the execution time by up to 80% and maintains linear memory scaling for overlays of up to a million nodes.

Phoenix was motivated by decentralized learning workloads and solutions such as gossip learning, for which PeerSim provides a natural simulation foundation but whose sequential event-driven execution limits practical scalability under heavy computation. By enabling parallel execution of independent interactions while preserving PeerSim's semantics, Phoenix extends the simulator's usefulness for large-scale studies of decentralized intelligent systems.

Acknowledgment

This work is funded by national funds through FCT – Fundação para a Ciência e a Tecnologia, I.P., under the support UID/50014/2025 (https://doi.org/10.54499/UID/50014/2025), through the Carnegie Mellon Portugal Program, by scholarship reference PRT/BD/155052/2024 and DOI identifier https://doi.org/10.54499/PRT/BD/155052/2024.

References

[1] R. Ormándi, I. Hegedűs, and M. Jelasity, "Gossip learning with linear models on fully distributed data," *Concurrency and Computation Practice and Experience*, vol. 25, 02 2013.

[2] A. Montresor and M. Jelasity, "Peersim: A scalable p2p simulator," in *2009 IEEE Ninth International Conference on Peer-to-Peer Computing*, 2009, pp. 99–100.

[3] T. Gil, F. Kaashoek, J. Li, R. Morris, and J. Stribling, "p2psim, a simulator for peer-to-peer protocols," 2003.

[4] S. Naicken, A. Basu, B. Livingston, and S. Rodhetbhai, "A survey of peer-to-peer network simulators," in *Proceedings of The Seventh Annual Postgraduate Symposium, Liverpool, UK*, vol. 2, 2006, p. 13.

[5] N. Kotilainen, M. Vapa, T. Keltanen, A. Auvinen, and J. Vuori, "P2prealm-peer-to-peer network simulator," in *2006 11th International Workshop on Computer-Aided Modeling, Analysis and Design of Communication Links and Networks*. IEEE, 2006, pp. 93–99.

[6] G. F. Riley and T. R. Henderson, "The ns-3 network simulator," in *Modeling and Tools for Network Simulation*, 2010. [Online]. Available: https://api.semanticscholar.org/CorpusID:46340499

[7] W. Yang and N. Abu-Ghazaleh, "Gps: a general peer-to-peer simulator and its use for modeling bittorrent," in *13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. IEEE, 2005, pp. 425–432.

[8] N. S. Ting and R. Deters, "3ls-a peer-to-peer network simulator," in *Proceedings Third International Conference on Peer-to-Peer Computing (P2P2003)*. IEEE, 2003, pp. 212–213.

[9] I. Baumgart, B. Heep, and S. Krause, "Oversim: A flexible overlay network simulation framework," 06 2007, pp. 79 – 84.

[10] J. Gu and S. X. Chen, "Statistical inference for decentralized federated learning," *The Annals of Statistics*, vol. 52, no. 6, pp. 2931–2955, 2024.

[11] Y. Jia, M. Fang, and N. Z. Gong, "Competitive advantage attacks to decentralized federated learning," in *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. [Online]. Available: https://openreview.net/forum?id=A7WfzMiffv

[12] W.-C. Chung, C.-A. Lo, Y.-H. Lin, Z.-H. Chen, and C.-L. Hung, "Decentralized federated learning with non-iid data: Challenges, trends, and future opportunities," *ACM Comput. Surv.*, vol. 58, no. 8, Feb. 2026. [Online]. Available: https://doi.org/10.1145/3785657

[13] A. Belenguer, J. A. Pascual, and J. Navaridas, "Glow – a novel, flower-based simulated gossip learning strategy," 2025. [Online]. Available: https://arxiv.org/abs/2501.10463

[14] P. Guijas, S. Alawadi, D. Rivero, and E. Fernandez-Blanco, "P2pfl: A modular framework for decentralized federated learning," 01 2026.

[15] F. Turazza, M. Pietri, and N. S. Hadjidimitriou, "Blockchain-based framework for segmented gossip learning in decentralized systems," in *Proceedings of the 2026 International Conference on Infrastructure-as-a-Service (IaaS) and Platform-as-a-Service (PaaS) Solutions for Europe's Next-Gen Cloud Infrastructure*, ser. NOUS '26. New York, NY, USA: Association for Computing Machinery, 2026, p. 8–14. [Online]. Available: https://doi.org/10.1145/3793828.3793836

[16] A. van Elst, I. Colin, and S. Clémençon, "Robust distributed estimation: Extending gossip algorithms to ranking and trimmed means," in *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. [Online]. Available: https://openreview.net/forum?id=A4t17y2lXa

[17] P. Jesus, C. Baquero, and P. Almeida, "Dependability in aggregation by averaging," *arXiv preprint arXiv:1011.6596*, 2010.

[18] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM Trans. Comput. Syst.*, vol. 23, no. 3, p. 219–252, Aug. 2005. [Online]. Available: https://doi.org/10.1145/1082469.1082470

[19] E. A. Lee, "The problem with threads," *Computer*, vol. 39, no. 5, pp. 33–42, 2006.

[20] K. Arnold, J. Gosling, and D. Holmes, *The Java programming language*. Addison Wesley Professional, 2005.