gCloudSim: A GPU Accelerated Simulator for Learning-driven Cloud Resource Optimisation

Jie Zhao and Rajkumar Buyya

Quantum Cloud Computing and Distributed Systems (qCLOUDS) Lab
School of Computing and Information Systems
The University of Melbourne, Australia

Abstract. The management of cloud resources for dynamic workloads presents a significant challenge. Learning-driven methods, including deep reinforcement learning, have emerged as promising solutions but necessitate extensive datasets for model training. Due to the practical constraints of acquiring real-world data, simulators are often used to generate the necessary training workloads. However, a critical bottleneck arises from the architectural disconnect between CPU-bound simulation environments and GPU-accelerated training processes, leading to substantial data transfer overhead. To address this inefficiency, we propose a novel GPU-accelerated simulator implemented in the Julia programming language. Our framework is designed to unify simulation, model training, and inference, allowing these phases to execute almost entirely on GPUs. This approach effectively eliminates the traditional CPU-GPU communication bottleneck. Experimental evaluations demonstrate substantial performance gains over existing simulators, achieving up to a 96% reduction in wallclock time. This work facilitates more efficient and scalable development of learning-driven cloud resource management models, offering significant potential for both research and practical applications in cloud computing.

Keywords: Cloud Resource Optimisation · GPU-Acceralated Simulation · learning-driven Resource Management.

1 Introduction

Cloud computing has revolutionised the digital landscape by enabling scalable, on-demand access to computational resources, driving advancements across sectors like business, healthcare, and scientific research[1]. However, the rapid growth of cloud services has come at a cost, particularly in terms of energy consumption and resource management challenges. Efficient resource management, which includes dynamic provisioning and scheduling of workloads, is critical to maintaining performance while minimising operational costs. Traditional resource management approaches, which rely heavily on static heuristics or manual tuning, are ill-suited to handle the growing complexity and variability of modern cloud workloads. In response, learning-based cloud resource management

approaches, particularly those using machine learning and reinforcement learning techniques, have garnered attention for their potential to adapt dynamically to real-time changes. Yet, a significant gap remains: these learning-driven approaches require vast amounts of data for training, which is challenging to generate in real-time, making cloud simulators an indispensable tool for addressing this need.

Cloud simulators, such as CloudSim[2] and recently released CloudSim 7G[3], enable the generation of synthetic workload data in controlled environments, helping researchers train learning-driven models without the high costs and resource constraints associated with real-world cloud platforms. These simulators mimic real-world cloud environments by providing metrics on CPU utilisation, memory usage, network traffic and latency, which are essential for training AI models to optimise resource management. However, existing simulators are usually CPU-bound, which has limited parallelism, slow neural network training and inference, poor scaling with high-dimensional spaces compared to GPU-based ones. In this research, we present a cloud simulator integrated with GPU acceleration to speed up the data generation process, enabling faster model training. Our approach streamlines the workflow by reducing the time and computational cost associated with training AI models for resource management tasks.

Our main contributions are summarised as follows:

- We propose a high-performance cloud simulator that can utilise GPUs. To the best of our knowledge, this work appears to be among the first to propose cloud simulation on GPUs. This work can accelerate future research in deep learning and reinforcement learning-based cloud resource management.
- We design the architecture of such a simulator and detail the core components.
- We implement the core simulator in the Julia programming language. Compared to predominant cloud simulators such as Cloudsim[2], it reduces simulation wallclock time by 96%.

2 Background and Related Work

Cloud simulators play a pivotal role in the design, development and optimisation of cloud resources, providing researchers and developers with a platform to test and validate algorithms in controlled environments. They can be used for evaluating and optimising infrastructure performance, cost efficiency, energy consumption, scheduling algorithms, and service-level objectives without incurring the high costs and complexity of real-world deployment. Several cloud simulators have emerged to meet the needs of the cloud computing community, each offering distinct features and benefits.

CloudSim[2][3] is one of the most widely used and extensible cloud computing simulators with many extensions created by the research community. Developed by the CLOUDS Lab at the University of Melbourne, it provides a comprehensive framework for modelling and simulating cloud environments, including data centres, virtual machines, and user workloads. CloudSim enables researchers to

study cloud resource management techniques, such as scheduling, load balancing, and energy-efficient provisioning, without the need for a real cloud infrastructure.

iCanCloud[4] has a highly modular and extensible model that focuses on predicting the trade-offs between performance and costs in cloud environments, helping users simulate large data centres with minimal computational resources. iCanCloud emphasises economic modelling, allowing users to predict the cost-performance ratio of different cloud setups. The main limitation of iCanCloud is its steep learning curve, as it requires a deep understanding of the underlying architecture to customise simulations.

GreenCloud[5] is a cloud simulator tailored for energy-aware cloud computing research. Its primary focus is on modelling the energy consumption of data centre networks, virtual machines, and servers. Researchers use GreenCloud to test and evaluate energy-efficient cloud computing solutions and analyse the impact of different resource management policies on power consumption.[6] While GreenCloud excels at energy modelling, it has limited support for some advanced cloud features, such as complex scheduling algorithms or the simulation of dynamic, heterogeneous workloads.

SimGrid[7] provides a versatile framework for simulating distributed computing infrastructures, including clouds, grids, and peer-to-peer systems. It is known for its flexibility and accuracy in simulating large-scale environments and allows the study of distributed systems in both controlled and real-world conditions. However, its complexity and steep learning curve can be a challenge for users unfamiliar with distributed systems.

GroudSim[8] is a simulation framework for both grid and cloud computing environments. It is designed to evaluate job scheduling algorithms, resource allocation policies, and the behaviour of distributed applications under different conditions. GroudSim integrates cloud and grid simulation, allowing users to model hybrid environments that combine cloud and grid resources. This makes it particularly useful for research in areas that overlap both paradigms, although it may not be as specialised as other simulators in purely cloud-based scenarios.

Simulator	Focus Area Extensibility	Strengths Language	Limitations Workload type
CloudSim[2]	General cloud simulation Highly extensible	Widely used, flexible, comprehensive framework Java	Limited scalability for large-scale environments Batch, interactive
iCanCloud[4]	Performance-cost modelling Extensible for specific cloud providers	Detailed economic modelling, modular C++	Steep learning curve, requires deep customisation Batch, Real-time
GreenCloud[5]	Energy efficiency Moderate extensibility	Focus on energy modelling in data centres C++ (with NS2/NS3)	Limited support for complex scheduling Energy-efficient workload
SimGrid[7]	Distributed systems (cloud, grid, HPC) Extensible across distributed paradigms	Accurate large-scale simulation, flexible C	High complexity, steep learning curve large-scale distributed systems, batch processing, real-time distributed applications
GroudSim[8]	Grid and cloud computing Supports hybrid environments	Hybrid cloud/grid simulation Java	Not specialised for pure cloud environments Hybrid cloud/grid workloads, batch jobs
This Work	General cloud simulation Support integration with other language	High performance, flexible, comprehensive framework Julia	Less popular programming language Batch, interactive

Table 1: Comparison of Cloud Simulators

Table 1 compares the differences of related work in the focus area, strengths, limitations, extensibility, programming language and workload type. While these simulators provide a robust platform for experimentation, one common limi-

tation is that they are CPU-bound, which creates bottlenecks when training learning-driven models that require extensive data, as deep learning typically leverages GPU acceleration. Additionally, many cloud simulators lack functionalities such as real-time metric collection, which makes it difficult to integrate into dynamic, real-world cloud environments. The integration of GPU-based simulation and real-time metric generation is a growing area of interest, as these enhancements could significantly improve the utility of simulators for learning-driven cloud resource management research. Therefore, our proposed new GPU-driven simulation system is a timely contribution.

3 Architecture and Core Components

To overcome the aforementioned shortfalls in existing simulators, we propose a novel cloud simulation framework. This section details its architecture design and system components. Figure 1 demonstrates the core components of the pro-

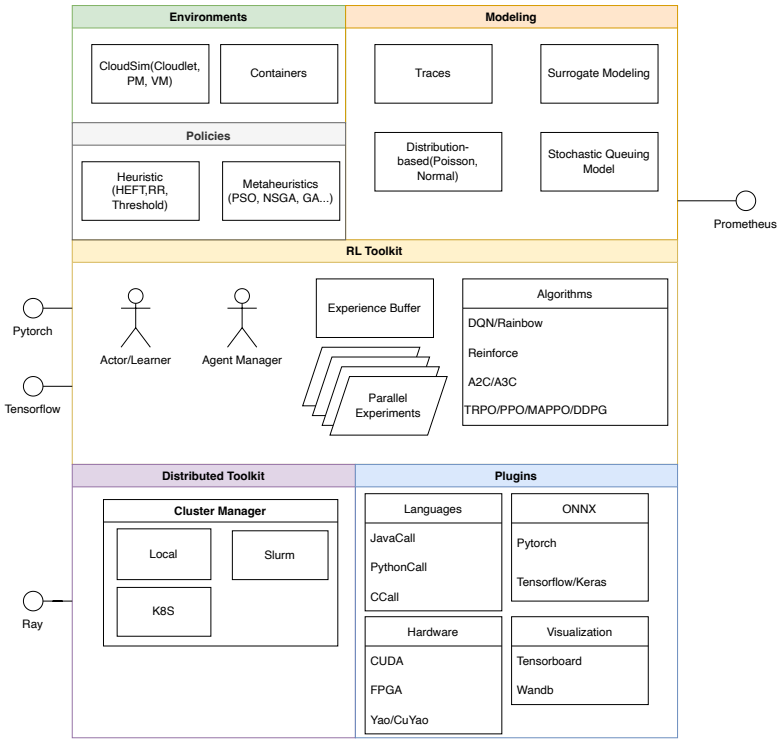

Fig. 1: *gCloudSim* Components Diagram

posed simulation framework. The simulator consists of six main components: *Environments* The environments module provides a configurable and extensible simulation substrate for evaluating resource management strategies in cloud and edge computing infrastructures. It includes support for CloudSim-style abstractions such as Cloudlets, Physical Machines (PMs), and Virtual Machines (VMs), as well as more lightweight container-based deployments. By modelling resource heterogeneity and dynamic workload arrivals, researchers can simulate a wide range of scenarios.

Policies Module For simple scheduling problems, such as VM-to-PM mapping or container-to-VM mapping, basic heuristics and meta-heuristics are implemented as baselines. This component implements a wide range of baseline and advanced scheduling policies, spanning rule-based heuristics (e.g., HEFT, Round-Robin), metaheuristics (e.g., Genetic Algorithms, Particle Swarm Optimisation, NSGA-II), and learned strategies. These policies can be used individually or as part of hybrid ensembles for comparative evaluation. The availability of standard benchmarks enables fair comparisons and ablation analyses. Researchers can also prototype novel policy paradigms and plug them into the experimental pipeline. The policy module is the locus of control decisions and central to the simulator's research utility.

The Workload Modelling component captures the probabilistic and empirical behaviour of workloads and system dynamics. It integrates surrogate modelling techniques to approximate performance and energy metrics, thereby reducing the computational burden of detailed simulation. The component also supports stochastic queuing models and statistical workload distributions (e.g., Poisson arrivals, Normal durations) to emulate realistic system behaviour under uncertainty. This flexibility enables controlled scenario generation and validation against established analytical baselines. The module is critical for training learning agents under both abstract and trace-driven assumptions. The *RL Toolkit* implements several baseline algorithms that researchers can use to compare and evaluate their own new algorithms. It supports running parallel experiments and managing experience buffers and logs. A few baseline algorithms such as Rainbow DQN, Reinforce[9], [10], A2C[11], A3C[12], TRPO[13], PPO[14], DDPG[15] are implemented and provided as baselines.

Distributed Toolkit stores agent-environment interaction tuples—comprising state, action, reward, and next state—for later reuse during training. It supports both online (on-policy) and offline (off-policy) learning, depending on the algorithm employed. Replay mechanisms enhance sample efficiency and training stability, particularly for deep neural networks. The buffer can be partitioned or prioritised based on importance sampling or temporal difference error to accelerate convergence. It plays a crucial role in decoupling data collection from gradient updates, thereby facilitating parallelism and robustness.

The plugin architecture allows seamless interoperability with external programming languages and ML toolkits. It supports JavaCall, PythonCall, and CCall interfaces, facilitating cross-language composition and reuse of established libraries. These are Julia's native ways to interface with those languages. This

enables integration of learning models written in TensorFlow[16], PyTorch[17], or exported via ONNX. By leveraging these mature ecosystems, the simulator benefits from advanced neural architectures and pretrained components. The plugin system ensures extensibility and lowers the barrier to adopting the simulator in diverse research contexts.

The monitoring subsystem provides real-time telemetry collection for both simulation and infrastructure layers. Prometheus integration enables time-series analysis of CPU, memory, and energy metrics to support feedback-driven learning and profiling. These insights can be used to inform reward design, identify resource bottlenecks, and validate the effectiveness of policies. Monitoring data can also support closed-loop control when integrated with online learning algorithms.

4 Design and Implementation

To design and develop a novel learning-driven cloud simulator, the first choice to make is the programming language to implement it with. As discussed in Table 1, existing cloud simulators are mainly implemented in JAVA or C/C++, although Python has become the de facto programming language in deep learning, with PyTorch and Tensorflow as the primary DL frameworks. Those frameworks are in fact implemented with highly optimised C/C+ code[16], [17]. As a scripting language with a global interpreter lock (GIL), Python is known to be slower than other languages. This slowness becomes a problem in simulator or reinforcement learning-based policy research, because these use cases require billions of simulation steps to be executed. To tackle this problem, we chose Julia over Python for developing a learning-driven cloud resource management simulator for the following reasons: *Performance and Speed* Julia is designed for high-performance computing with Just-In-Time (JIT) compilation via LLVM, which allows Julia code to run as fast as C or Fortran in many cases. This is crucial for simulators that require fast execution of complex numerical calculations or large-scale simulations, as seen in cloud resource management tasks.

Ease of Writing High-Performance Code In Julia, high-level code often performs on par with low-level languages like C, without needing the programmer to drop to lower-level languages for optimisation. This makes it easier to develop and maintain high-performance code, particularly for cloud simulation models that handle resource scheduling, machine learning, and optimisation algorithms in real-time.

Support for Machine Learning and Numerical Computing Julia provides native support for machine learning and numerical computing through libraries such as Flux.jl for machine learning and DifferentialEquations.jl for simulation and differential equation modelling. These tools are part of Julia's ecosystem, allowing seamless integration between high-performance simulations and learning-driven models.

Parallelism and Distributed Computing Julia has built-in support at the language level [18] for parallelism and distributed computing with simple syntax.

These features are handy for cloud simulators, where tasks need to be scheduled across distributed systems or simulated on multiple cores or machines. Julia's straightforward handling of concurrency enables efficient scaling for extensive simulations.

GPU Integration Julia integrates seamlessly with GPUs through packages like CUDA.jl, which allows you to write GPU kernels directly in Julia, without needing a separate language like CUDA-C.

Numerical Accuracy and Mathematical Libraries Julia is built with high numerical accuracy in mind, particularly for scientific and mathematical computing, ensuring that floating-point precision and other numerical operations are handled efficiently and accurately.

Scalability As a learning-driven cloud resource management simulator scales in complexity (e.g., larger workloads, more intricate machine learning models), Julia's performance does not degrade as significantly as Python's in heavy computational scenarios. The ability to run simulations, train models, and perform inference all within Julia's ecosystem, without switching between languages or environments, ensures more streamlined scalability.

To implement a complex cloud simulator, building from scratch is a challenging task, if not infeasible. Hence, we surveyed and built upon some open source libraries that are useful for our simulator. The selected libraries with their description and relevance are listed in Table 3 in the appendix section.

5 Core Simulation Framework Using GPUs

GPUs are particularly well-suited for processing matrices and vectorised environments due to their architecture, which is optimised for parallel computations and data-level parallelism (DLP): 1) They have numerous cores (thousands, compared to CPUs' few cores) that can handle many operations in parallel. This makes them highly effective for vectorised computations where the same operation needs to be applied to multiple data points, such as matrix operations in machine learning or simulations. 2) The architecture of GPUs is designed for Single Instruction, Multiple Data (SIMD) execution, where a single instruction operates on multiple data elements simultaneously, improving the efficiency of vectorised operations. 3) Libraries like CUDA and cuBLAS allow developers to leverage GPUs for efficient matrix and tensor operations. These libraries are optimised for performing mathematical computations, such as matrix multiplication or convolutions, which are core operations in many vectorised workloads.

5.1 Modelling Cloud Environment as Vectorised Environment

Inspired by CloudSim[2], we model cloud environments as a collection of entities: Data Centre, Hosts, VMs. For each physical host at time t, denoted as H_t, consider both capacity and current utilisation, which is given by:

$$\mathbf{H_t} = \begin{bmatrix} H_{cpu}^{u(t)} & H_{mem}^{u(t)} & H_{iops}^{u(t)} & H_{bandwidth}^{u(t)} & \cdots \\ H_{CPU}^{c(t)} & H_{mem}^{c(t)} & H_{iops}^{c(t)} & H_{bandwidth}^{c(t)} & \cdots \end{bmatrix} \tag{1}$$

Where $H_R^{u(t)}$ reprenents resource utilisation of host H at time t, and $H_R^{c(t)}$ denotes resources capacity of Host H. In the initial implementation, resource $R = [CPU, mem, iops, bandwidth, ...]$. This entity can be easily extened with Julia's multi-dispatch paradigm if a new type of resources needs to be considered. Then, a data centre DC consisting of n hosts can be represents as a three-dimensional matrix.

$$\mathbf{DC} = \begin{bmatrix} H_1 \\ H_2 \\ \cdots \\ H_n \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} H_{cpu}^{u(t)} & H_{mem}^{u(t)} & H_{iops}^{u(t)} & H_{bandwidth}^{u(t)} \cdots \\ H_{CPU}^{c(t)} & H_{mem}^{c(t)} & H_{iops}^{c(t)} & H_{bandwidth}^{c(t)} \cdots \end{bmatrix} \\ \begin{bmatrix} H_{cpu}^{u(t)} & H_{mem}^{u(t)} & H_{iops}^{u(t)} & H_{bandwidth}^{u(t)} \cdots \\ H_{CPU}^{c(t)} & H_{mem}^{c(t)} & H_{iops}^{c(t)} & H_{bandwidth}^{c(t)} \cdots \end{bmatrix} \\ \cdots \\ \begin{bmatrix} H_{cpu}^{u(t)} & H_{mem}^{u(t)} & H_{iops}^{u(t)} & H_{bandwidth}^{u(t)} \cdots \\ H_{CPU}^{c(t)} & H_{mem}^{c(t)} & H_{iops}^{c(t)} & H_{bandwidth}^{c(t)} \cdots \end{bmatrix} \end{bmatrix} \tag{2}$$

Many data centres are typically organised with different physical tiering, for instance, multiple floors, server rooms and racks. In such a case, those entities can be further extended by adding additional dimensions, e.g. to add a server rack dimension, DC becomes a four-dimensional matrix instead of a three-dimensional one.

5.2 Modelling the network behaviour

Modelling the network behaviour in a cloud environment should also be an essential consideration. While data centres can use different network topologies in their environments, for instance, fat-tree, star topologies, mesh topology, etc., the QoS parameters such as latency can be modelled as a multi-dimensional adjacency matrix. A multi-dimensional adjacency matrix extends the concept to represent relationships between nodes across multiple dimensions or layers. In a multi-layered network, a graph consists of various layers, and each layer has its own adjacency matrix. The relationships between the same nodes across different layers (or dimensions) can also be captured. Hence, a multi-dimensional adjacency matrix can be seen as a tensor (multi-dimensional array) where each "slice" represents the adjacency matrix for a different layer, dimension, or type of connection.

$$A_{ij}^{(k)} = \begin{cases} l & \text{if there is a connection between } i,j \text{ in layer } k, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Where $A^{(k)}$ represents the adjacency matrix for layer k, and i and j are nodes (hosts) in the graph, l is the latency between node i and j. While it's physically impossible to have zero latency in real life, we can use 0 to denote that there is no direct connection between i and j (i.e. inter-connection between i and j has to go through additional hops in the same or different layer.)

For example, in a 3D adjacency matrix:

1. The first dimension represents nodes.
2. The second dimension represents other nodes (as in the traditional adjacency matrix).
3. The third dimension (or depth) represents the different layers or types of relationships.

$$A = \left[A^{(1)}\ A^{(2)}\ \ldots\ A^{(n)}\right] \tag{4}$$

Virtual machines(VMs) and containers can be modelled similarly to hosts with tensors at time t, capturing their capacities and utilisation. With the formulation of a multi-dimensional matrix (tensor), we can then convert the simulator's behaviour and logic to tensor operations on GPUs. In the simulation of provisioning or allocations, we assume the data centre has a broker and queues requests (a request can be VM, container or job execution). As the simulator clock advances, based on the capacity of available hosts, unless a policy has been specified, jobs are scheduled to hosts in a FIFO and round-robin manner. Users of the simulator should always override this behaviour to evaluate their own policies. Remaining capacities $H_R^{r(t)}$ of a resource R on each host can be calculated as $H_R^{r(t)} = H_R^{c(t)} - H_R^{u(t)}, 0$. Here when the host's utilisation exceeds the capacity, i.e. overbooking, remaining capacity returns a negative value to show the degree or severity of overbooking. However, no further job will be assigned to the host until it has some capacity.

Due to page number constraints, we include an example training workflow and algorithm in the appendix.

6 Performance Evaluation

To evaluate the performance of *gCloudSim*, we create a data centre with sets of hosts and cloudlets and run simulations comparing CloudSim and *gCloudSim*. We use permutation of $N_{hosts} = 100, 500, 1000, 2000$ hosts and $N_{cloudlets} = 1000, 2000, 5000, 10000$ cloudlets, hence 16 experiments in total. The experiments are conducted on a Linux machine (Ubuntu 24.04 LTS) with an Intel i7-14700K CPU, nVidia RTX 3070 8GB and 192 GB of Memory. For CloudSim, JDK17 is used with the G1 garbage collector and JVM parameters -Xms8G -Xmx8G. By using a fixed JVM heap size and allocating enough memory to the JVM, it will minimise the impact of the garbage collection process. Table 2 compares the runtime of CloudSim and gCloudSim across various workloads and host configurations. For 1000 cloudlets in CloudSim, the times are 5.84s, 6.80s, 6.57s and 6.95s for 100, 500, 1000 and 2000 hosts, respectively. The increase in hosts does not impact the runtime much since the decision time mainly depends on the number of cloudlets. Interestingly, the runtime is higher when the number of hosts is low and the number of cloudlets is high. We suspect it takes more time to make a placement decision when all the hosts are overloaded, e.g. some cloudlets need to wait in the queue until some hosts have free capacity. As the number of cloudlets increases—from 1,000 to 10,000—the runtime of CloudSim rises

No. of Hosts	No. of Cloudlets	Runtime CloudSim(s)	Runtime *gCloudSim* (s)	Improvement
100	1000	5.84	1.43	75.50%
100	2000	6.75	1.24	81.59%
100	5000	9.92	1.42	85.73%
100	10000	36.55	3.15	91.37%
500	1000	6.80	1.24	81.71%
500	2000	7.11	1.63	77.14%
500	5000	9.68	1.04	89.23%
500	10000	33.93	1.19	96.48%
1000	1000	6.57	1.38	79.02%
1000	2000	7.25	1.26	82.63%
1000	5000	9.90	1.41	85.76%
1000	10000	31.97	3.28	89.73%
2000	1000	6.95	1.24	82.17%
2000	2000	7.20	1.58	78.08%
2000	5000	10.09	1.16	88.52%
2000	10000	31.89	3.37	89.42%

Table 2: Performance Comparison between CloudSim and Our work

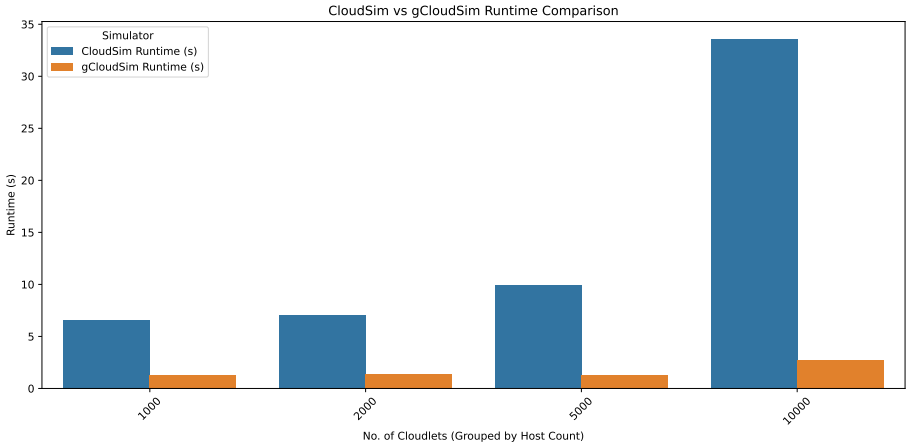

Fig. 2: Runtime comparison between CloudSim and gCloudSim under varying cloudlet counts.

sharply, particularly under high-load scenarios (e.g., 10,000 cloudlets), where it reaches over 36 seconds in some cases. In contrast, gCloudSim consistently maintains significantly lower runtimes, demonstrating minimal growth even as workloads scale. This trend is consistent across all host configurations (100, 500, 1000, and 2000 hosts), highlighting gCloudSim's scalability and computational efficiency. Figure 2 provides a visual summary of this trend, clearly illustrating the disparity in performance between the two simulators.

7 Conclusions and Future Work

In this research, we propose a novel cloud simulator that can utilise GPU hardware using Julia to provide high-performance simulation for learning-based cloud research. For future work and development, we can further enhance the simulator by extending the GPU-accelerated simulator to model task offloading for latency-sensitive applications across hybrid cloud and edge computing environments. Also, enhancing the simulator to model multi-tenant environments, incorporating resource contention, QoS degradation, and priority scheduling to analyse performance under competitive loads.

References

[1] B. Jamil, H. Ijaz, M. Shojafar, K. Munir, and R. Buyya, "Resource Allocation and Task Scheduling in Fog Computing and Internet of Everything Environments: A Taxonomy, Review, and Future Directions," *ACM Comput. Surv.*, vol. 54, 233:1–233:38, 11s Sep. 9, 2022, ISSN: 0360-0300.

[2] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[3] R. Andreoli, J. Zhao, T. Cucinotta, and R. Buyya, "CloudSim 7G: An Integrated Toolkit for Modeling and Simulation of Future Generation Cloud Computing Environments," *Software: Practice and Experience*, vol. 55, no. 6, pp. 1041–1058, 2025, ISSN: 1097-024X.

[4] "iCanCloud - Cloud Computing Systems." (), [Online]. Available: `https://omnetpp.org/download-items/iCanCloud.html` (visited on 09/06/2024).

[5] D. Kliazovich, P. Bouvry, Y. Audzevich, and S. U. Khan, "GreenCloud: A Packet-Level Simulator of Energy-Aware Cloud Computing Data Centers," in *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*, Dec. 2010, pp. 1–5.

[6] C. Fiandrino, D. Kliazovich, P. Bouvry, and A. Y. Zomaya, "Performance and Energy Efficiency Metrics for Communication Systems of Cloud Computing Data Centers," *IEEE Transactions on Cloud Computing*, vol. 5, no. 4, pp. 738–750, Oct. 2017, ISSN: 2168-7161.

[7] H. Casanova, A. Legrand, and M. Quinson, "SimGrid: A Generic Framework for Large-Scale Distributed Experiments," in *Tenth International Conference on Computer Modeling and Simulation (Uksim 2008)*, Apr. 2008, pp. 126–131.

[8] S. Ostermann, K. Plankensteiner, R. Prodan, and T. Fahringer, "GroudSim: An Event-Based Simulation Framework for Computational Grids and Clouds," in *Euro-Par 2010 Parallel Processing Workshops*, M. R. Guarracino, F. Vivien, J. L. Träff, *et al.*, Eds., Berlin, Heidelberg: Springer, 2011, pp. 305–313, ISBN: 978-3-642-21878-1.

[9] M. Hessel, J. Modayil, H. van Hasselt, *et al.*, "Rainbow: Combining Improvements in Deep Reinforcement Learning," Oct. 6, 2017.

[10] R. J. Williams, "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning," *Mach. Learn.*, vol. 8, no. 3–4, pp. 229–256, May 1, 1992, ISSN: 0885-6125.

[11] S. Dalton and I. Frosio, "Accelerating Reinforcement Learning through GPU Atari Emulation," p. 10,

[12] V. Mnih, A. P. Badia, M. Mirza, *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16, New York, NY, USA: JMLR.org, Jun. 19, 2016, pp. 1928–1937.

[13] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, "Trust region policy optimization," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15, Lille, France: JMLR.org, Jul. 6, 2015, pp. 1889–1897.

[14] Y. Gu, Y. Cheng, C. L. P. Chen, and X. Wang, "Proximal Policy Optimization With Policy Feedback," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 52, no. 7, pp. 4600–4610, Jul. 2022, ISSN: 2168-2232.

[15] H. Tan, "Reinforcement Learning with Deep Deterministic Policy Gradient," in *2021 International Conference on Artificial Intelligence, Big Data and Algorithms (CAIBDA)*, May 2021, pp. 82–85.

[16] M. Abadi, P. Barham, J. Chen, *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'16, USA: USENIX Association, Nov. 2, 2016, pp. 265–283, ISBN: 978-1-931971-33-1.

[17] A. Paszke, S. Gross, F. Massa, *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 721, Red Hook, NY, USA: Curran Associates Inc., Dec. 8, 2019, pp. 8026–8037.

[18] "Distributed Computing · The Julia Language." (), [Online]. Available: `https://docs.julialang.org/en/v1/stdlib/Distributed/` (visited on 07/30/2025).

Algorithm 1 GPU-Accelerated DRL Training in Cloud Scheduling Simulator

Initialize environment $\mathcal{E}$ and agent policy π_θ
Initialize experience buffer $\mathcal{D} \leftarrow \emptyset$
Initialize parallel environments $\{\mathcal{E}_1, \ldots, \mathcal{E}_N\}$ across N simulators
Initialise GPU-based actor and learner modules
for each training iteration **do**
    for each environment $\mathcal{E}_i$ in parallel **do**
        Observe current state s_t^i
        Select action $a_t^i \sim \pi_\theta(s_t^i)$ ▷ Actor on GPU
        Execute a_t^i in $\mathcal{E}_i$, observe reward r_t^i and next state s_{t+1}^i
        Store transition $(s_t^i, a_t^i, r_t^i, s_{t+1}^i)$ in $\mathcal{D}$
    end for
    for each learner update step **do**
        Sample mini-batch $\{(s, a, r, s')\} \sim \mathcal{D}$
        Compute target $y = r + \gamma \cdot \text{Target}(s')$
        Compute loss $L(\theta) = \mathbb{E}_{(s,a,r,s')}\left[(y - Q_\theta(s,a))^2\right]$
        Update policy $\theta \leftarrow \theta - \alpha \cdot \nabla_\theta L(\theta)$ ▷ Backprop on GPU
    end for
    Log training metrics (e.g., average reward, loss)
end for

The above Fig. 1 indicates the general flow of Actor-Critic-based model training. This general algorithm can be applied to the class of algorithms that use Actor-Critic techniques. We start the training flow by initialising the environments in parallel, then run the training episodes on GPUs using CUDA kernels. Each learner collects its own experience and adds it to the experience buffer, updating the policy network accordingly. At the end of each training loop, the average reward and loss value are logged, which can be used later in visualisation.

In this appendix, we describe an example training algorithm with an Actor-Critic algorithm, more specifically, using Proximal Policy Optimisation (PPO). Let the cloud system at time t be represented by a set of container or VM states:

$$\mathcal{S}_t = \{s_t^1, s_t^2, \ldots, s_t^n\}, \quad s_t^i \in \mathbb{R}^d \tag{5}$$

where each s_t^i encodes CPU, memory, energy usage, and queue information. Jobs arrive via a stochastic model:

$$j_t \sim \mathcal{P}_{\text{arrival}}(\lambda), \quad \text{duration}(j_t) \sim \mathcal{P}_{\text{service}}(\mu) \tag{6}$$

A surrogate function estimates performance:

$$f_{\text{sur}}(j_t, s_t) \to \mathbb{R}^k \tag{7}$$

We simulate N environments $\mathcal{E}_1, \ldots, \mathcal{E}_N$ in parallel:

$$\forall i \in \{1, \ldots, N\}, \quad \text{simulate } \mathcal{E}_i \text{ independently} \tag{8}$$

Each generates transitions:

$$\tau_i = \{(s_t^i, a_t^i, r_t^i, s_{t+1}^i)\}_{t=1}^T, \quad \mathcal{D} = \bigcup_{i=1}^N \tau_i \tag{9}$$

The actor network is $\pi_\theta(a_t \mid s_t)$ and the critic is $V_\phi(s_t)$.
We compute the advantage estimate:

$$\hat{A}_t = \sum_{l=0}^{K} (\gamma\lambda)^l \delta_{t+l}, \quad \delta_t = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t) \tag{10}$$

The PPO loss:

$$\mathcal{L}^{\text{PPO}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right] \tag{11}$$

With:

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \tag{12}$$

Critic loss:

$$\mathcal{L}^{\text{V}}(\phi) = \frac{1}{2}\mathbb{E}_t \left[\left(V_\phi(s_t) - \hat{R}_t \right)^2 \right] \tag{13}$$

Parameter updates (computed on GPU):

$$\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}^{\text{PPO}}, \quad \phi \leftarrow \phi - \beta \nabla_\phi \mathcal{L}^{\text{V}} \tag{14}$$

Let the GPU have M streaming multiprocessors. CUDA streams $\{s_1, \ldots, s_M\}$ run concurrently.

Each CUDA thread k computes:

$$\text{Thread}_k \mapsto \nabla_\theta \ell_k, \quad \text{reduce via warp-level atomic add} \tag{15}$$

Algorithm 2 PPO Training for CUDA-Accelerated Job Scheduling

Initialize environment $\mathcal{E} = \{\mathcal{E}_1, \ldots, \mathcal{E}_N\}$ on GPU
Initialize policy network π_θ and value network V_ϕ
Initialize experience buffer $\mathcal{D} \leftarrow \emptyset$
for each training epoch **do**
    for each environment $\mathcal{E}_i$ in parallel **do**
        Observe current state $s_t^i = \texttt{observe}(\mathcal{E}_i)$
        Sample action $a_t^i \sim \pi_\theta(s_t^i)$
        Store (s_t^i, a_t^i) in buffer
    end for
    Launch CUDA kernel to assign jobs and update machine loads
    Synchronise GPU and increment job indices
    for each environment **do**
        Compute reward $r_t^i = -\max(\text{machine loads})$
        Store (r_t^i, s_{t+1}^i) in buffer
    end for
    Sample minibatch $\mathcal{B} \subset \mathcal{D}$
    for each sample $(s, a, r, s') \in \mathcal{B}$ **do**
        Estimate advantage $\hat{A}(s, a)$
        Compute PPO loss:

$$\mathcal{L}^{\text{PPO}} = \min\left(r_\theta \hat{A}, \text{clip}(r_\theta, 1-\epsilon, 1+\epsilon)\hat{A}\right)$$

        Compute value loss: $\mathcal{L}^{\text{V}} = \frac{1}{2}(V_\phi(s) - \hat{R})^2$
    end for
    Update $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}^{\text{PPO}}$
    Update $\phi \leftarrow \phi - \beta \nabla_\phi \mathcal{L}^{\text{V}}$
end for

Table 3: Julia Libraries Adopted for Building a Cloud Simulator, all attributions go to the original authors

Library Name	Short Description	Relevance to Cloud Simulation/Usecase
ConcurrentSim	a discrete-event, process-oriented simulation framework	Supports discrete event simulation and allows modelling of resources and constraints
Flux	A machine learning library for Julia	Useful for building and training models to predict and optimise resource allocation in cloud environments. Supports deep learning for real-time decision-making in cloud resource management.
Reinforcement	A reinforcement learning library for Julia	Useful for building and training models to predict and optimise resource allocation using reinforcement learning in cloud environments.
JuliaDynamics	A collection of libraries for the field of dynamical systems, nonlinear dynamics and chaos theory	It supports agent-based modelling and time series analysis
CUDA	Enables CUDA programming for NVIDIA GPUs in Julia	Allows for GPU-accelerated simulations and deep learning model training in cloud simulators, making it faster to simulate large-scale environments and complex workloads.
POMDP	Partially Observable Markov Decision Process	In practice, there are often unobservable features or parameters; this library helps in those cases for modelling the environment as POMDPs
Distributed	Provides distributed computing capabilities in Julia	Essential for simulating cloud environments where workloads are distributed across multiple nodes, mimicking real-world cloud resource management scenarios.
Distributions	provides a comprehensive collection of probability distributions and statistical functions	This library allows uising simple distribution to generate workloads
HTTP	A package for handling HTTP requests and responses	Necessary for simulating API calls between services in cloud platforms, especially when testing cloud-native architectures with microservices.
JuMP	A modelling language for optimisation problems	Essential for implementing resource optimisation algorithms in cloud simulators, enabling efficient resource allocation, load balancing, and scheduling strategies.
DataFrames	Provides data manipulation tools similar to R and Python's Pandas	Important for handling large datasets generated during simulations, such as logs of resource usage or workload distribution across cloud nodes.
CSV	A package for reading and writing CSV files in Julia	Enables input/output handling for cloud simulations, allowing the simulator to import/export workload data or results in CSV format for further analysis.
Plots	A flexible plotting package for Julia	Useful for visualising simulation results, such as performance metrics, resource usage graphs, and other key outputs that help in analysing the effectiveness of cloud resource management algorithms.
Gen	A probabilistic programming system for AI applications	Can be used to model uncertainty in cloud simulations, such as unpredictable workloads or failures, helping to simulate more realistic cloud environments.
LightGraphs	A graph-based library for network and graph algorithms	Useful for simulating network topologies and communication patterns between nodes in a cloud infrastructure, helping to optimise data flow and resource usage.
Logging	Provides a logging framework for Julia applications	Helps track and record the behaviour of the cloud simulation, such as resource allocation decisions and workload distribution, for debugging and performance analysis.
TensorOperations	Provides tensor computations and contractions	Helps in performing efficient large-scale numerical simulations, especially for machine learning models used in predicting and managing cloud resources.
TimerOutputs	A package to measure and record timings of code blocks	Critical for profiling and optimising the performance of cloud simulators, identifying bottlenecks, and improving simulation runtime efficiency.
Zygote	An automatic differentiation library for Julia	Useful for implementing gradient-based optimisation in machine learning models used in cloud simulators, enabling adaptive resource management strategies.
StatsBase	Provides basic statistical functions and algorithms	Essential for analysing simulation data, including resource usage trends and workload distributions, helping to fine-tune cloud resource allocation strategies.