A cloud-based visualization interface for gama to simplify the simulation of complex socio-environmental systems

Nguyen Tuan Thanh Le[1], Xuan Truong Nguyen[2*], Minh Thu Dao[2], Linh Manh Pham[2]

Abstract: This paper presents a novel cloud-based architecture for multi-agent simulations, built on top of the GAMA platform, to address computational limitations of the traditional desktop-based version. We propose a distributed system leveraging the cloud computing model, pub/sub messaging, and containerized deployment to enable scalable, parallel execution of complex socio-environmental simulations, especially for pig farming. Experimental results demonstrate significant performance improvements, with the system supporting concurrent simulations across multiple worker nodes. The solution reduces infrastructure costs by 40% compared to physical implementation while providing researchers with an accessible web interface for scenario execution. This work establishes a reusable framework for cloud-based agent-based modeling, with particular applicability to smart agriculture and epidemic management.

Keywords: Multi-agent modeling and simulation, cloud computing, GAMA platform, distributed systems.

1. Introduction

Agent-based modeling and simulation (ABMS) has become indispensable for simulating complex systems in ecology, epidemiology, and agriculture. Among ABMS's platforms, the GAMA (GIS Agent-based Modeling Architecture) platform (Taillandier et al., 2019) provides a robust open-source environment. However, it remains constrained by desktop-based execution, limiting scalability for large-scale simulations. Traditional GAMA implementations face three critical limitations: 1) Hardware dependency requiring high-end local workstations; 2) Sequential processing preventing concurrent simulation execution; 3) Accessibility barriers for non-technical researchers. These constraints impede large-scale simulations.

Concurrently, cloud computing is a model of providing computing resources over the Internet, allowing users and organizations to access and use hardware, software, and services through a cloud infrastructure without having to invest in physical infrastructure. This model allows businesses and individuals to save on initial investment costs and system maintenance costs, while providing flexible scalability, easy access from anywhere and on any device with an Internet connection. Cloud computing offers unlimited computational resources, yet integration with ABM frameworks remains underexplored.

Therefore, in this paper, we present a cloud-based architecture for the multi-agent GAMA platform to address computational limitations of the traditional desktop-based version. The contributions of this paper are as follows: (1) A cloud-native architecture for distributed GAMA simulations implemented on OpenStack; (2) A user-friendly web interface enabling the execution of simulation experiments in a cloud environment; (3) Two agricultural case studies demonstrating the integration capability of the distributed system; and (4) Performance benchmarks of the implementation of distributed simulations on the proposed architecture with different cloud infrastructures. The rest of the paper is organized into seven sections. Section 2 reviews related works. Section 3 details the architecture of our proposed system. Section 4 presents implementation of the proposed system. Then, on top of it, Section 5 presents two case studies and Section 6 evaluates performance of the models, simulated on cloud. Finally, Section 7 concludes with future directions.

2. Related works

2.1. Agent-Based Simulation Platforms

Modeling complex systems like transportation networks, tsunamis, or epidemics can be challenging because of their many components and non-linear behavior. Traditional analytical models are often too difficult or impossible to handle these challenges. As a result, agent-based modeling and simulation (ABMS) has emerged as a bottom-up alternative. This approach treats a complex system as a group of independent agents that can observe their surroundings and act accordingly (Russell et al. 2021). To support modellers to design their models, several ABMS platforms are developed. Among them, GAMA (Taillandier et al., 2019) is a well-known open-source platform. It enables spatially explicit ABM through its GAML language, supporting complex behaviors and inheritance hierarchies. While powerful, its desktop-bound execution contrasts with cloud-native solutions

[1]Thuyloi University
[2]VNU University of Engineering and Technology
[*]Corresponding author
Received 6th Sep. 2025
Accepted 14th Nov. 2025
Publication date 31st Dec. 2025

like NetLogo's distributed variants (Wilensky & Rand, 2015) or AnyLogic's cloud integration (Grigoryev, 2016).

2.2. Cloud Computing for Simulation

Deploying traditional applications requires businesses to invest in physical infrastructure, including servers, storage, networks, and other hardware. Businesses must manage and maintain this hardware, which is not only expensive but also requires significant human resources and time to maintain. Cloud-based simulation frameworks (e.g., AWS Batch, Azure CycleCloud) offer elasticity but lack ABM-specific optimizations. Research by (Jacobsen, 2009) established pub/sub patterns for distributed systems, yet few studies apply these to agent-based models. The recent work by (Nguyen et al. (2022)) implemented cloud-based precision feeding simulations; however, their system did not provide distributed execution capabilities. Therefore, this study aims to enhance and propose a new system architecture that addresses this limitation.

3. System architecture

Our architecture, as shown in Figure 1, comprises five core components, as following: (1) **Web frontend**: the user interface for simulation configuration and monitoring; (2) **Master node**: the central coordinator managing task distribution; (3) **Worker nodes**: the GAMA execution units for running simulations; (4) **Redis**: the pub/sub broker for inter-node communication; and (5) **MySQL**: the persistent storage for simulation metadata and results

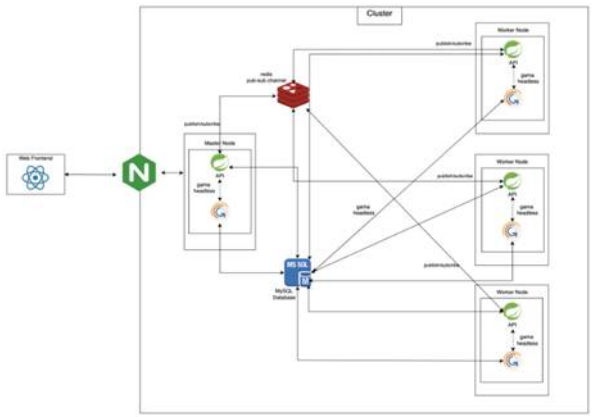

Figure 1. The architecture of our proposed system - a distributed architecture with master-worker pattern and pub/sub messaging

3.1. Master Node

The master node functions as the central orchestrator of the system, responsible for receiving simulation requests through the REST API. Upon receiving a request, it distributes tasks to worker nodes via Redis channels, leveraging real-time messaging to optimize task allocation. The worker nodes subsequently execute experiments in the headless mode of GAMA, based on the parameters specified in the XML configuration files. Once completed, the output data from the workers is aggregated by the master node and stored in a MySQL database for subsequent analysis and interpretation. Furthermore, to ensure fault tolerance and maintain system continuity, the master node is equipped with a failover mechanism, enabling it to temporarily assume the role of worker nodes during periods of overload.

3.2. Worker Node

Each worker consists of four core elements: an API Layer that manages task reception and status updates, GAMA Headless for running simulations efficiently without the resource demands of a graphical interface, a Storage Module to handle input and output file operations, and a Project Repository that stores GAML models along with their necessary dependencies. This structure ensures seamless task processing, simulation execution, data management, and model organization within each worker.

3.3. Communication Protocol

We implement a Redis pub/sub workflow to orchestrate simulation tasks, following the mechanisms described by (Redis Ltd. (2022)). The master node publishes new tasks to the simulation_queue channel, where workers subscribe to receive them. To ensure exclusive task ownership and prevent conflicts, workers rely on atomic operations to claim tasks, allowing each worker to process only one task at a time. During execution, workers periodically publish progress updates to the status_updates channel, enabling real-time monitoring of task advancement. Upon completion, a final notification is issued, triggering the aggregation of results across all workers. This structured workflow ensures efficient task distribution, transparent progress tracking, and centralized result compilation.

4. Implementation

4.1. Cloud Infrastructure

We deployed our approach on OpenStack. The system comprises a Master Node (configured as c8-r16g with 8 vCPUs and 16 GB RAM) and three Worker Nodes (each configured as c4-r8g with 4 vCPUs and 8 GB RAM). All nodes run Ubuntu 22.04 and utilize Docker containerization, following the deployment model described by (Docker Inc. (2023)), to ensure streamlined and consistent application execution across the cluster.

4.2. GAMA Headless Integration

A core element of GAMA is the GAML modeling language, which enables the specification of agents, behaviors, and environments in a structured and expressive manner. Although the platform provides an intuitive graphical user interface for model development, many experimental settings benefit from executing simulations without a GUI. This capability is offered by the GAMA Headless mode, which supports

command-line or script-based execution and is well suited for large-scale deployments on servers or computing clusters.

In this study, we employ GAMA Headless an integrated feature of the GAMA platform to execute predefined simulation scenarios directly on distributed nodes, thereby eliminating the need to launch the full graphical application. While headless execution enables high automation and improves computational efficiency, it also introduces challenges related to recording and managing simulation outputs. To accommodate the requirements of each specific model, we customize the input parameter set and dynamically generate simulation configurations based on the selected model and experiment. Furthermore, to address the limitations associated with headless execution, we refine XML configuration files and extend the result-storage mechanisms within our distributed simulation framework.

These enhancements leverage GAMA's Database Access feature, which enables direct interaction between simulations and external data storage systems, including both relational (e.g., MySQL, PostgreSQL) and NoSQL databases. In our implementation, we integrate a MySQL backend to support reliable and flexible storage of simulation results, facilitating subsequent analysis and enabling real-time data retrieval during execution.

4.3. Container Orchestration

To automate the process of deploying the services of the proposed architecture on Docker, we used the Docker Compose tool. The code snippet in Figure 2 is an excerpt of a Docker Compose configuration file used to create a *worker* service, which has different directories to store the simulation project files and corresponding results. The worker service can connect to the *redis* service, which is already started with a given configuration file.

```
services:
  worker:
    build: ./docker/worker
    volumes:
      - ./projects:/app/projects
      - ./storage:/app/storage
    environment:
      REDIS_HOST: redis

  redis:
    image: redis:7.4
    command: redis-server /usr/local/etc/redis/redis.conf
```

Figure 2. An excerpt of the Docker Compose configuration file

4.4. Web Frontend

We developed a web-based interface to facilitate the configuration and execution of distributed experiments. As shown in Figure 3, the interface—implemented using the component-based architecture described by Meta Platforms, Inc. (2023)—provides modular input fields that allow users to specify the simulation model, select relevant experiments, allocate computational nodes, and define the execution duration in terms of simulation steps.

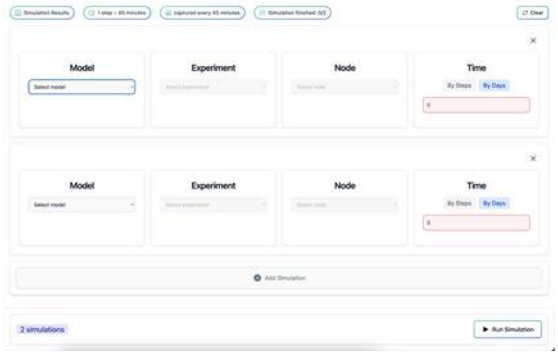

Figure 3. Web interface of our simulation system

Multiple experiments can be configured concurrently through a user-friendly feature, thereby enabling batch execution across different models or parameter configurations. Moreover, real-time system indicators displayed at the top of the interface report execution progress, including simulation status, step duration, and data capture intervals, thus supporting transparent monitoring and management of ongoing activities. Experimental results are presented in a clear manner, analogous to their representation in desktop environments, as illustrated in Figure 4. This architecture enhances research usability while ensuring scalability and efficiency in deploying agent-based simulations within cloud environments.

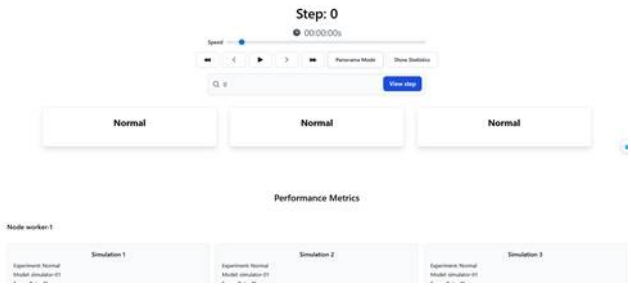

Figure 4. The user interface for displaying simulation results

5. Case studies

5.1. Precision Feeding Model

We model the Precision Feeding problem in GAMA, with the input consisting of a herd of 20 pigs. The system simulates the process in which each individual pig receives a specific amount of feed through the smart feeder, with pig growth determined by the quantity of feed consumed. We developed a

simulation model with a graphical interface on GAMA, which replicates the structure of a precision feeding system for pigs. The farm interface is shown in Figure 5, includes the following elements: (1) *Living area*: corresponding to the life area in the conceptual model, where the herd resides; (2) *Entrance gate*: equipped with a scale that records pig weights once per week, according to the weighing schedule; (3) *Feeding and watering system*: consists of five feeders, each dedicated to a single pig; and (4) *Exit gate*: allowing pigs to leave the feeding area and return to the living area.

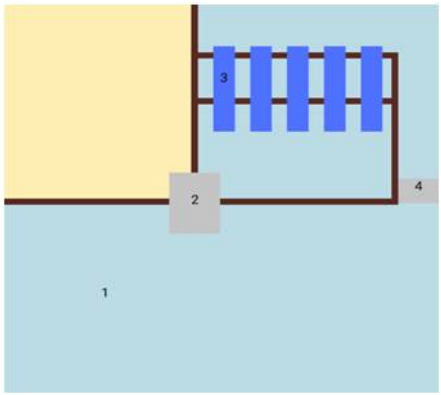

Figure 5. The graphical interface of the facility used for simulation

5.2. ASF Disease Transmission Model

The African swine fever (ASF) is a highly contagious viral disease of pigs. The infection can spread rapidly among individuals through direct contact or indirectly via contaminated hosts. In intensive farming systems, the virus can be transmitted through multiple pathways, including airborne particles, feed, water, farming equipment, and human-mediated activities. Given the severity and rapid transmissibility of ASF, research of its transmission dynamics within farms are essential for developing effective prevention and control strategies. Disease transmission between pens was modeled using a location-based contact mechanism and parameterized with an SEIR framework (Nielsen et al., 2017), that consists of four primary states: (1) *Susceptible* - Individuals not yet infected but vulnerable to the disease; (2) *Exposed* - Individuals that have been in contact with the pathogen and are in the incubation phase; (3) *Infected* - Individuals that are infectious and capable of transmitting the disease to others; and (4) *Recovered* - Individuals that have recovered and may acquire immunity.

6. Performance evaluation

6.1. Experimental Setup

To evaluate the performance of the simulation model, we conducted experiments of the epidemic spread simulator described in Section 5 on two distinct cloud platforms: (1) *OpenStack*, a private cloud deployed on the server infrastructure of the Digital University Center at the University of Engineering and Technology, Vietnam National University, Hanoi; and (2) the public cloud environment of *AWS* with equivalent configurations. Due to the limited capacity of the existing OpenStack infrastructure, the experimental deployment was implemented using one master node and three worker nodes, following the architecture illustrated in Figure 1 and configured as detailed in Table 1.

Table 1. Hardware configuration of the experimental environment

Platform	CPU		RAM		Framework
	Master	Worker	Master	Worker	
OpenStack	8 CPU (c8-r16g)	4 CPU (c4-r8g)	16GB	8GB	GAMA 1.9 + Local Storage
AWS	8 vCPUs (Intel Xeon)	4 vCPUs (Intel Xeon)	16GB	8GB	GAMA 1.9 + S3 + RDS

With OpenStack, due to limitations in infrastructure resources, we deployed the system on a relatively small scale. All modules, including the web frontend, database, and Redis, were integrated with the master node API on a single instance to optimize resource usage and simplify management. In contrast, in the AWS environment, storage and database services were deployed using available cloud-native services with configurations equivalent to those implemented on OpenStack. In the future, as the system scales, these services can be separated into dedicated instances to ensure performance, scalability, and stability. Once deployed, users can conveniently conduct simulation experiments through the web-based interface, as illustrated in Figure 2. The experimental evaluation was conducted by executing multiple simulations on two cloud platforms. Each simulation represented a pigsty consisting of 20 pigs over a simulated period of one month, with results recorded every 45 steps (equivalent to 45 minutes in simulation time). During execution, system resource metrics such as CPU and RAM utilization were monitored at 3-second intervals to collect data for performance assessment. In fact,

we carried out experiments on the two simulation models developed with GAMA, which contained nearly equivalent numbers of agents and attributes, differing only in processing logic and control algorithms. These differences did not cause significant variations in system resource consumption; thus, the results can be regarded as representative for both models. Running multiple simulations in this way served to demonstrate the capability of the cloud-based distributed architecture to integrate with different GAMA-built models.

To evaluate system performance across 6 experimental scenarios described in Table 2 (three scenarios for each simulation model), the following key factors were analyzed: (1) The analysis focused on the consumption of CPU resources during the simulation execution; (2) The duration of the simulation; (3) The RAM utilization during simulation execution; and (4) The maximum number of simulations that can be executed per node and per cluster based on configuration parameters.

6.2. Results

Table 2. Execution time of Simulations

Scenarios	Cloud	
# simulations	OpenStack	AWS
4 simulations	7 min 14 s, CPU: 30.39%, RAM: 298.11 MB / 8.00 GB	4 min 18 s, CPU: 57.11%, RAM: 336.36 MB / 8.00 GB
8 simulations	8 min 01 s, CPU: 57.04%, RAM: 740.97 MB / 8.00 GB	4 min 57 s, CPU: 98.78%, RAM: 412.34 MB / 8.00 GB
12 simulations	8 min 29 s, CPU: 78.90%, RAM: 780.80 MB / 8.00 GB	7 min 50 s, CPU: 90.99%, RAM: 414.72 MB / 8.00 GB

The experimental results reveal distinct performance characteristics between the two cloud platforms when handling an increasing number of concurrent simulations. First, execution time demonstrates a consistent advantage for AWS over OpenStack across all scenarios in Table 2. For 4, 8, and 12 simultaneous simulations, AWS completes the tasks approximately 40–50% faster than OpenStack. This indicates that AWS provides superior computational throughput, particularly under moderate workloads. However, as the number of simulations increases from 8 to 12, the performance gap narrows, suggesting that AWS approaches its upper limit of CPU utilization efficiency, while OpenStack retains more unused computational headroom. Second, CPU utilization patterns highlight different resource allocation strategies. OpenStack exhibits a more conservative utilization profile, ranging from 30.39% (4 simulations) to 78.90% (12 simulations). In contrast, AWS demonstrates more aggressive scaling, reaching nearly full CPU utilization (98.78%) at 8 simulations. This suggests that AWS's virtualization and scheduling mechanisms are optimized for maximizing resource use but also indicates a higher risk of saturation under heavy loads. OpenStack, on the other hand, may prioritize stability and workload isolation, leaving substantial computational capacity underutilized. Third, RAM consumption reveals an intriguing asymmetry. OpenStack's memory usage increases substantially with the number of simulations (from 298 MB at 4 simulations to nearly 781 MB at 12 simulations), whereas AWS maintains a relatively stable memory footprint (336–415 MB). This indicates that OpenStack allocates memory more linearly with respect to the number of running instances, while AWS appears to employ more efficient memory sharing or optimization mechanisms. Such a difference could have implications for large-scale deployments where memory contention is a limiting factor. Overall, the findings demonstrate that AWS excels in execution speed and memory efficiency, but at the cost of pushing CPU utilization to saturation, which may affect scalability and stability under extreme workloads. In contrast, OpenStack delivers slower performance with higher memory consumption but maintains more balanced CPU utilization, potentially offering more predictable stability for long-running or heterogeneous simulation workloads.

7. Conclusion

This paper proposes a cloud-based simulation architecture for the GAMA platform, leveraging large-scale data processing, parallel execution, and elastic scalability to overcome the performance limitations of traditional simulation approaches. We develop a distributed simulation framework and deploy two complex socio-environmental models on OpenStack and AWS for evaluation. Experimental results demonstrate significant improvements in computational efficiency and show that the system supports farmers in making informed decisions, reducing epidemic risks, and enhancing productivity without costly field-based interventions.

Despite these promising results, further enhancements remain necessary. Future work includes

automating the integration of experimental simulations, improving project and data management, and refining the distributed model to better support multithreading within and across computing nodes. Incorporating machine learning techniques is expected to strengthen analytical and predictive capabilities, while optimizing cloud resource usage and improving the user interface will facilitate broader practical adoption.

By extending the scale and speed of high-performance simulation, the proposed architecture enables researchers and policymakers to address complex system challenges more effectively across diverse domains, including livestock management, urban planning, traffic simulation, ecological conservation, supply-chain optimization, and disaster response.

References

Grigoryev, I. (2016). *Anylogic 7 in three days: A quick course in simulation modeling*. Ilya Grigoryev.

Nielsen, J. P., Larsen, T. S., Halasa, T., & Christiansen, L. E. (2017). *Estimation of the transmission dynamics of African swine fever virus within a swine house*. Epidemiology and infection, 145(13), 2787–2796. https://doi.org/10.1017/S0950268817001613

Nguyen, Xuan Truong and Pham, Manh Linh (2022). *Cloud-Based Simulation of Precision Feeding System for Pig Health Management*. In: The 13th International Conference on Application of Information Technology in Agriculture Asia-Pacific Region, Hanoi, Vietnam.

Russell, S. J., & Norvig, P. (2021). *Artificial intelligence: A modern approach*. Pearson.

Taillandier, P., Gaudou, B., Grignard, A., Huynh, Q.N., Marilleau, N., Caillou, P., Philippon, D. and Drogoul, A. (2019). *Building, composing and experimenting complex spatial models with the GAMA platform*. GeoInformatica, 23(2), pp. 299-322.

Wilensky, U., & Rand, W. (2015). *An introduction to agent-based modeling: Modeling natural, social, and engineered complex systems with NetLogo*. The MIT Press.

Docker Inc., *"Docker Documentation,"* Docker Docs, 2023. [Online]. Available: https://docs.docker.com/. Redis Ltd., *"Redis 7.0 Documentation,"* Redis.io, 2022. [Online]. Available: https://redis.io/docs/. Meta Platforms, Inc., *"React Documentation: React 18,"* React.dev, 2023. [Online]. Available: https://react.dev/.