AgentServe: Online Who–Where Adaptation for Open-World, Geo-Distributed Stateful Multi-Agent Systems

Demonstration Track

Toshimi Okubo
NTT, Inc.
Tokyo, Japan
toshimi.okubo@ntt.com

Takuya Fukushima
NTT, Inc.
Tokyo, Japan
takuya.fukushima@ntt.com

Akira Sakamoto
NTT, Inc.
Tokyo, Japan
akira.sakamoto@ntt.com

Reina Hoshino
NTT, Inc.
Tokyo, Japan
reina.hoshino@ntt.com

Noriaki Inoue
NTT, Inc.
Tokyo, Japan
noriaki.inoue@ntt.com

Kensuke Yokota
NTT, Inc.
Tokyo, Japan
kensuke.yokota@ntt.com

Yoshimi Ichiyanagi
NTT, Inc.
Tokyo, Japan
yoshimi.ichiyanagi@ntt.com

ABSTRACT

We demonstrate AgentServe, a framework for open-world, geo-distributed multi-agent systems where interaction locality and execution conditions change online. In real deployments, systems must continuously adapt *who* coordinates with whom as locality shifts, while also adapting *where* stateful agents run as latency, bandwidth, and load vary across heterogeneous sites. AgentServe adapts Who by computing dynamic communication graphs from agent state, and adapts Where via adaptive placement across tiers with live migration. The demo shows continuous agent behavior while connectivity and placement adapt online under changing locality and resource conditions.
Demonstration video: https://youtu.be/d56kDFyv8qA.

KEYWORDS

multi-agent systems; dynamic graphs; stateful migration

ACM Reference Format:
Toshimi Okubo, Takuya Fukushima, Akira Sakamoto, Reina Hoshino, Noriaki Inoue, Kensuke Yokota, and Yoshimi Ichiyanagi. 2026. AgentServe: Online Who–Where Adaptation for Open-World, Geo-Distributed Stateful Multi-Agent Systems: Demonstration Track. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Paphos, Cyprus, May 25 – 29, 2026*, IFAAMAS, 3 pages. https://doi.org/10.65109/NWCW1919

1 INTRODUCTION

Many multi-agent systems (MAS) prototypes implicitly assume a closed and stationary setting: a fixed set of agents, a stable environment model, and coordination over a mostly fixed interaction structure. Real deployments are often the opposite. They are open-world (agents and workloads appear and disappear), non-stationary (local context and objectives drift), and geo-distributed (execution spans heterogeneous sites such as edge locations and regional clouds). In such settings, both the interaction structure and the execution substrate become moving targets.

A representative example is V2X-style cooperative perception. Vehicles and roadside units continuously enter and leave the scene, and the set of "relevant peers" changes with proximity, occlusions, and changing traffic context. At the same time, the best execution site also changes as latency, bandwidth, and server load fluctuate across sites. Similar dynamics arise in city-scale sensing, multi-robot teams, and long-running AI agent swarms. These systems require continuous coordination while both locality and resource conditions shift online.

We frame the core challenge as two coupled questions that must be answered continuously: Who should communicate with whom as locality shifts, and Where should stateful agents run as latency and load vary across sites. At scale, "connect everyone" is infeasible, and fully centralized control becomes brittle. Moreover, restart-based relocation breaks long-lived state, interrupting online adaptation precisely when conditions change.

Where-side adaptation (placement and relocation of stateful actors) has been explored in virtual-actor runtimes such as Orleans [3] and in systems work on actor locality optimization (e.g., ActOp [6]). AgentServe is motivated by open-world, non-stationary MAS and aims to treat both Who and Where as first-class controls for long-running agents. To provide an AI-friendly, Pythonic platform for implementing long-running agents and their control logic, we build AgentServe on Ray [5] and its abstractions for heterogeneous resources.

To make this control user-friendly, AgentServe cleanly separates reusable runtime mechanisms from application-specific control policies. The runtime provides name-based communication, state publication, and state-preserving relocation, while users supply pluggable discovery and placement policies to adapt Who and Where online. This design lets developers keep agent logic stable and focus on policy design under non-stationarity, rather than re-implementing recurring distributed-systems plumbing. The next section summarizes the architecture and key components.

2 ARCHITECTURE AND KEY COMPONENTS

AgentServe is a policy-driven framework that adapts both Who and Where online for long-running, stateful agents (Figure 1). A

Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus. © 2026 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). https://doi.org/10.65109/NWCW1919

key design choice is to keep platform services simple and reactive, while embedding essential runtime capabilities inside each agent so scalability follows the agent population. In practice, the control plane stays lightweight and policy-driven, and most runtime mechanisms live with the agents. This section summarizes the system boundaries and then describes the core components that enable online adaptation of interaction structure and execution placement.

System boundaries. AgentServe interfaces with the outside world through a lifecycle API used by clients and external systems to spawn and despawn agents in response to events and workload changes (Figure 1, via the API/Controller). It also runs on an execution substrate on Ray (optionally Kubernetes [7]) that abstracts heterogeneous resources for distributed execution across tiers. AgentServe adds a control layer on top.

Agent runtime. Each agent is long-running and stateful, and includes built-in runtime services that remove recurring engineering overhead from application code. First, agents self-register in a directory for name resolution, avoiding manual endpoint bookkeeping. They also publish dynamic state (e.g., position and attributes) to the discovery registry, which serves neighbor queries. Second, agents support live migration with their own state transfer: when migration is triggered, a replica is spawned at the destination, the agent transfers its state to that replica, and the directory entry is updated so subsequent communication follows the migrated instance. This makes the migration execution largely agent-driven (while the decision to migrate is made by the scheduler), keeping the platform reactive and avoiding heavy centralized orchestration for state movement.

Discovery (Who). Discovery maintains a state registry of published agent state and serves on-demand queries. The scheduler, described next, can also read this registry and derived edges to make placement and migration decisions. A pluggable discovery policy computes a neighbor set or an edge map from the latest registered state, enabling dynamic locality (e.g., proximity-based interaction) without global synchronization among all agents. For example, a spatial indexing policy partitions the space (e.g., grid/tiles) and returns nearby candidates efficiently as agents move. Discovery is intentionally passive. Agents push updates and Discovery responds to queries, so it scales with the number of agents without a central coordinator.

Scheduler (Where). The scheduler places agents across heterogeneous tiers under a pluggable policy, and triggers migration when conditions change (e.g., load and fragmentation) or when coordination objectives shift. Policies represent different operational goals: a consolidation-oriented policy reduces active tiers, a balance-oriented policy reduces hotspots, and a locality-oriented policy co-locates frequently interacting agents to reduce cross-tier edges. Migration is executed via live state transfer, preserving long-running state and continuity while adapting placement over time.

3 IMPLEMENTATION

AgentServe is designed for minimal user code while supporting long-running, stateful execution at scale.

User-facing API hooks. As illustrated in the right panel of Figure 1, users define agents as ordinary Python classes annotated with `@agent`. The decorator injects built-in runtime services, including

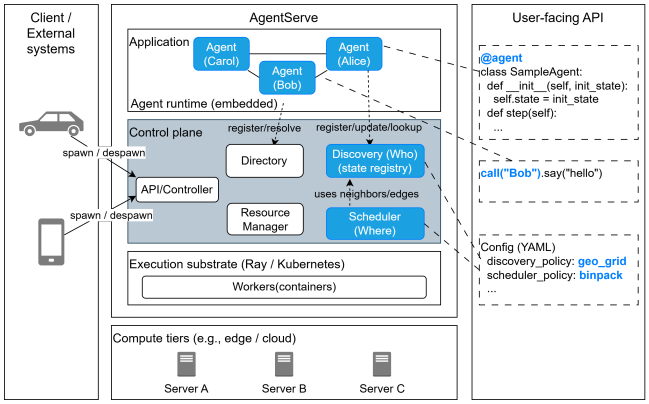

Figure 1: Overview of AgentServe.

migration support, self-registration, and access to the directory and discovery interfaces, so developers can focus on agent behavior.

Name-based communication. Agents communicate via logical identifiers rather than fixed endpoints, so users can call peers by name. Communication transparently follows a migrating agent, avoiding manual endpoint management. Discovery returns peer IDs from registered state, so "who to talk to" is realized through the same name-based interface.

Configurable and pluggable policies. Discovery and scheduling strategies are modular components selected via declarative YAML configuration (Figure 1, right). Users can provide custom discovery policies (e.g., spatial indexing for proximity-based neighbors) or custom placement heuristics without changing agent logic, enabling reuse of the same application while varying Who/Where control.

4 DEMONSTRATION

The demo runs real, stateful agents remotely on three servers (A, B, C) representing heterogeneous tiers. Agents move via a random walk, and Discovery updates communication edges online based on proximity (we fix a spatial-indexing discovery policy in this demo). We then apply representative, pluggable scheduling policies with distinct objectives: binpack for consolidation, spread for load balancing, and friends to reduce cross-tier edges by leveraging interaction locality. Across all phases, the agent population keeps changing via spawn/despawn events, while migrations are executed via live state transfer so behavior remains continuous and the system adapts both Who and Where online.

5 CONCLUSION AND FUTURE WORK

We presented AgentServe, a framework for open-world and geo-distributed MAS that adapts online (i) who agents coordinate with and (ii) where stateful agents run. The demo shows live migration with state continuity under changing locality and resource conditions. Future work includes validation in V2X-like cooperative perception and other wide-area cyber-physical workloads, and improving scalability and adaptivity via decentralized discovery and scheduling.

REFERENCES

[1] Akka Contributors. [n.d.]. Akka Documentation. https://doc.akka.io/. Accessed: 2026-01-07.

[2] Tuğba Bozkaya-Aras, Ciprian Alexandru, and Ralf Steinmetz. 2025. Optimizing Service Migration in IoT Edge Networks: Digital Twin-Based Computation and Energy-Efficient Approach. In *2025 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 1–6.

[3] Sergey Bykov, Alan Geller, Gabriel Kliot, James R. Larus, Ravi Pandya, and Jorgen Thelin. 2011. Orleans: Cloud Computing for Everyone. In *Proceedings of the 2nd ACM Symposium on Cloud Computing (SoCC '11)*. ACM. https://doi.org/10.1145/2038916.2038932

[4] Yen-Cheng Liu, Junjiao Tian, Nathaniel Glaser, and Zsolt Kira. 2020. When2com: Multi-Agent Perception via Communication Graph Grouping. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 4105–4114.

[5] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. 2018. Ray: A Distributed Framework for Emerging AI Applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association.

[6] Andrew Newell, Gabriel Kliot, Ishai Menache, Aditya Gopalan, Soramichi Akiyama, and Mark Silberstein. 2016. Optimizing Distributed Actor Systems for Dynamic Interactive Services. In *Proceedings of the 11th European Conference on Computer Systems (EuroSys '16)*. ACM. https://doi.org/10.1145/2901318.2901343

[7] The Kubernetes Authors. [n.d.]. Kubernetes: Production-Grade Container Orchestration. https://kubernetes.io/. Accessed: 2026-01-07.

[8] Michael Vögler, Johannes M. Schleicher, Christian Inzinger, Schahram Dustdar, and Rajiv Ranjan. 2016. Migrating Smart City Applications to the Cloud. *IEEE Cloud Computing* 3, 2 (2016), 72–79. https://doi.org/10.1109/MCC.2016.44

[9] Yutong Zhang, Ke Zhao, Yihong Yang, and Zhangbing Zhou. 2025. Real-Time Service Migration in Edge Networks: A Survey. *Journal of Sensor and Actuator Networks* 14, 4 (2025), 79. https://doi.org/10.3390/jsan14040079