UDC 004.75
SRSTI 50.47.31

https://doi.org/10.55452/1998-6688-2026-23-1-185-196

[1]*Kumalakov B.,
PhD, Associate Professor, ORCID ID: 0000-0003-1476-9542,
*e-mail: bolatzhan.kumalakov@astanait.edu.kz
[1]Tsoy D.,
Junior Researcher, ORCID ID: 0009-0001-2711-5233,
e-mail: tsoy.daniil.an@gmail.com
[1]Astana IT University, г. Астана, Қазақстан

DEVELOPMENT OF AN AGENT BEHAVIOR FOR AD-HOC GRID COMPUTING

Abstract

This study addresses the problem of inefficient task scheduling and limited fault tolerance in ad-hoc grid computing environments, where traditional systems rely on centralized control and stable infrastructure. To overcome this, a decentralized agent behavior was developed using the Java Agent DEvelopment Framework, enabling autonomous task redistribution among heterogeneous Worker agents. The proposed Scheduler–Worker architecture allows dynamic coordination and failure recovery without centralized orchestration. Experiments on five devices show that increasing the number of agents from one to three reduces total execution time by 1.98–3.25, while the best performance is achieved with four agents, providing a 2.99 speedup for 100 tasks. However, using six agents on fewer devices reduces efficiency to 2.34 due to resource contention and communication overhead. The study is limited by a single network topology and a small-scale testbed. Nevertheless, the results demonstrate the practical potential of agent-based decentralized scheduling for resilient distributed machine learning systems.

Keywords: multi-agent system, ad-hoc grid computing, JADE, distributed computing, fault tolerance, task scheduling, agent consensus.

Received: January 12, 2026; revised: March 3, 2026; accepted: March 12, 2026

Introduction

Distributed system is a collection of autonomous computers that appear to users as a single, coherent system. Such systems enable resource sharing, fault tolerance, and scalability by coordinating multiple networked nodes. Classical literature distinguishes several major categories of distributed systems, including distributed computing systems, distributed information systems, and distributed pervasive systems [1, 2]. Distributed computing systems emphasize coordinated execution of computational tasks across multiple machines; distributed information systems focus on consistency, availability, and reliability of shared data; and distributed pervasive systems address mobility, context awareness, and dynamic participation of devices that may frequently join or leave the system.

At the same time, the increasing heterogeneity of hardware resources - from laptops and desktops to cloud and edge devices – poses challenges for traditional, centrally orchestrated frameworks. Many established platforms assume relatively stable cluster configurations and rely on centralized schedulers, which may be less effective in dynamic environments where nodes frequently join and leave or where resource availability changes over time. Empirical studies of large-scale frameworks such as Hadoop and Spark show that, while they are highly effective for certain workloads, their default scheduling strategies often struggle in heterogeneous or rapidly changing environments, and many proposed improvements remain largely analytical rather than experimentally validated [6, 7].

Multi-agent systems (MAS) offer an alternative paradigm for building distributed applications under such conditions. An agent is a software entity situated in an environment, capable of autonomous action to meet its design objectives and able to interact with other agents. Agents are commonly characterized by autonomy, social ability, reactivity, and proactiveness [3]. In the Belief-Desire-Intention (BDI) model, an agent's internal state is structured into beliefs about the environment, desires representing objectives, and intentions corresponding to committed plans, which together guide behavior and decision-making [4]. A multi-agent system consists of multiple interacting agents that collectively solve problems that are difficult or inefficient for a single agent to handle, using mechanisms such as coordination, negotiation, and cooperation [3].

The system developed in this work builds on these MAS concepts and positions itself at the intersection of distributed computing systems and distributed pervasive systems. It is a JADE-based MAS deployed over a wireless local area network (WLAN), where multiple heterogeneous nodes cooperate to execute machine-learning (ML) workloads. Agents can dynamically join or leave the environment, reflecting characteristics of pervasive systems, while still supporting distributed computation across devices with differing central processing units (CPU), graphics processing unit (GPU), random-access memory (RAM), and operating system configurations. The architecture follows a Scheduler–Worker model implemented using the JADE framework, which provides agent communication, lifecycle management, and service discovery via a Yellow Pages mechanism [5]. Python-based ML executables, specifically XGBoost, Random Forest, and CatBoost, are dispatched through Spring Boot services to different devices for training.

This design contrasts with established distributed platforms and orchestration systems. Apache Hadoop and Spark provide scalable batch, iterative, and streaming computation models, but they rely on centralized resource management and relatively stable cluster assumptions [6, 7]. Apache Kafka focuses on high-throughput, fault-tolerant data streaming rather than computation itself [8]. Distributed file systems such as Ceph, EOS, GlusterFS, and Lustre demonstrate that storage-layer design significantly affects performance under different I/O patterns, which in turn influences higher-level distributed applications [9]. Load-balancing technologies like Nginx show that algorithmic improvements at the traffic-distribution layer can substantially reduce response time and failure rates [10]. Container orchestration frameworks, including Docker Swarm and Kubernetes, introduce powerful abstractions for deployment, scaling, and fault tolerance, but they still rely on centralized control planes and assume comparatively predictable infrastructure [11, 12].

In contrast, MAS-based approaches decentralize decision-making and embed autonomy directly into the components of the system. Prior research has explored MAS for load balancing and resource management in both simulated and real environments. Ajitha [14] demonstrates improved response time and utilization over traditional scheduling algorithms using a MAS-based approach, although the evaluation is limited to simulation. Reinforcement-learning-based MAS frameworks for wireless networks and cloud–edge environments show promising results in controlled settings but often depend on stable models, extensive training, and simulation-based evaluation [15, 16]. Hybrid approaches combining classical algorithms with MAS techniques, such as LC-MAS for web servers, indicate improved performance under high load but are typically evaluated in limited, local environments [17]. Surveys of MAS in domains such as smart grids further confirm their suitability for managing complex, distributed infrastructures, while also highlighting a lack of large-scale empirical validation on heterogeneous, failure-prone hardware [18].

Distributed machine learning represents a particularly demanding application domain for such systems. Existing work on distributed ML emphasizes techniques such as distributed and local stochastic gradient descent, model parallelism, and tensor parallelism to accelerate training of large models [19]. Across these approaches, communication overhead, data movement, and scheduling efficiency are repeatedly identified as key bottlenecks. Broader analyses of emerging trends in parallel and distributed systems point toward cloud-native, edge-oriented, and AI-driven resource management solutions that can dynamically adapt to changing workloads and resource availability [20].

Within this broader context, the JADE-based MAS developed in this study serves as a small-scale, experimentally grounded platform for investigating decentralized scheduling and coordination of ML workloads. Unlike many prior studies that rely primarily on simulation, the system is deployed and evaluated on five real, heterogeneous devices connected via WLAN. This setting exposes practical issues such as communication delays, intermittent failures, agent churn, and resource contention especially when multiple agents are deployed on the same physical machine.

This study focuses on empirically evaluating how the number and placement of agents affect performance in this environment. Specifically, the following research questions are addressed: Does adding more agents always reduce training time for ML workloads? How does running multiple agents on the same physical device affect system performance and reliability?

To answer these questions, a series of experiments were conducted in which the number of Worker agents varied from one to six, and tasks consisting of training multiple instances of XGBoost, Random Forest, and CatBoost models on synthetic classification datasets were executed. Total execution time and average training time per model are measured. The results show that increasing the number of agents significantly reduces training time for small and medium workloads, but that performance gains saturate and may degrade when too many agents share the same physical resources. In particular, configurations with multiple Worker agents on the same machine experience higher communication overhead, increased error rates, and more frequent task reassignment, offsetting the benefits of additional parallelism. These findings highlight the existence of an optimal degree of parallelism and motivate MAS-based scheduling mechanisms that explicitly account for both computational and communication bottlenecks.

Table 1 – MAS and existing solutions comparison table

Platform	Dynamic participation	Autonomous agents	Heterogeneous devices	Central scheduler	ML execution	Reassign work on failure node	Cloud-scale
Proposed MAS	+	+	+	–	+	+	+
Hadoop	–	–	+	+	–	–	+
Spark	–	–	+	+	+	–	+
Kafka	+	–	+	+	–	–	+
Docker Swarm	+	–	+	+	–	–	+
Kubernetes	+	–	+	+	–	–	+

Materials and methods

System Architecture

The proposed system is implemented as a multi-agent system (MAS) using the JADE (Java Agent DEvelopment Framework) platform. The architecture follows a Scheduler–Worker model deployed over a wireless local area network (WLAN), according to the architecture scheme illustrated on Figure 1. One node hosts a Scheduler agent responsible for task coordination, while the remaining nodes host Worker agents that execute machine-learning (ML) training tasks.

Each physical device runs a JADE container and registers its agents using JADE's Yellow Pages (Directory Facilitator) service. This mechanism enables dynamic discovery of active Worker agents and supports agents joining or leaving the system at runtime. Communication between agents is performed using JADE's asynchronous message-passing model.

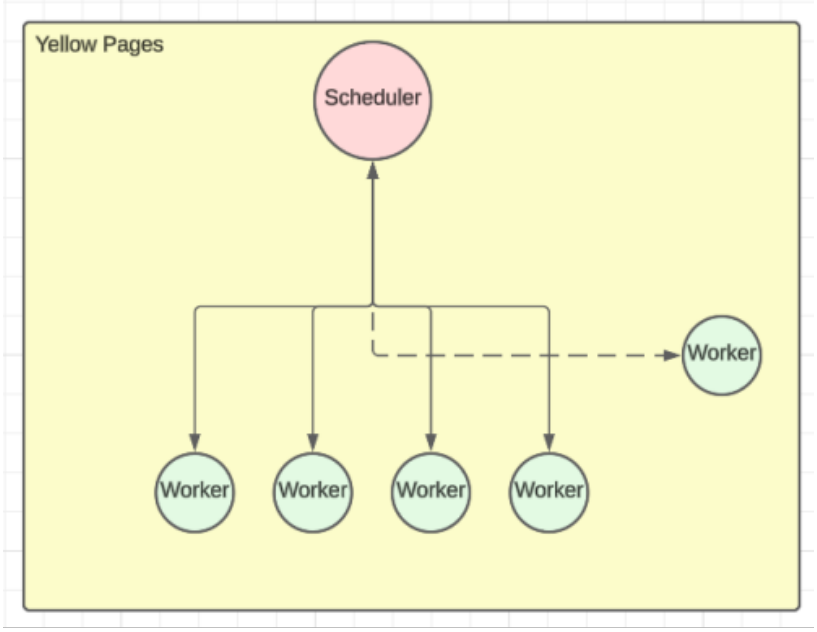

Figure 1 – Agent network architecture

Hardware and Software Environment

Experiments were conducted on five heterogeneous devices, including four laptops and one desktop computer, differing in CPU architecture, GPU availability, memory size, and operating systems. The configurations of all machines are listed in Table 2.

Table 2 – Hardware and software configurations of the experimental testbed

Device	Type	CPU	RAM	GPU	OS
1	Laptop	Intel Core i5-11400H	16 GB	NVIDIA GeForce RTX 3060 Laptop GPU (6 GB VRAM)	Windows 11
2	Laptop	Intel Core i7-12700H	16 GB	NVIDIA GeForce RTX 3050 Ti Laptop GPU (8 GB VRAM)	Windows 11
3	Laptop	Intel Core i7-13650HX	24 GB	NVIDIA GeForce RTX 4050 Laptop GPU (6 GB VRAM)	Windows 10
4	Laptop	Intel Core i5-13420H	16 GB	Intel Raptor Lake-P integrated graphics	Linux Arch
5	Desktop	AMD Ryzen 5 5600	16 GB	NVIDIA GeForce RTX 3060 (12 GB VRAM)	Windows 11

All devices were connected via WLAN and acted as independent agent hosts. The system core is implemented in Java 24.0.1. JADE is used to implement agent lifecycle management, communication, and service discovery. Spring Boot is employed to transfer executable ML training files between agents. Machine-learning models are developed in Python 3.13.5 using scikit-learn, CatBoost, XGBoost, and supporting libraries. ML models are packaged as executable files and executed locally by Worker agents on each device.

Agent Workflow

The experimental workflow is summarized in Figure 2.

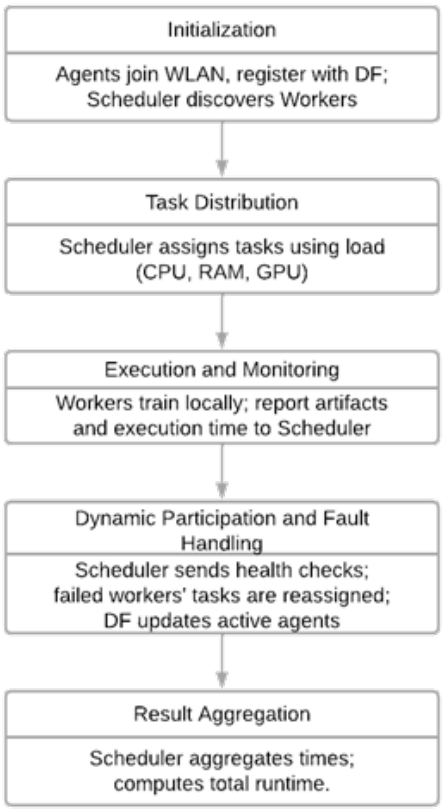

Figure 2 – Experimental workflow stages
of the proposed Scheduler–Worker system

Agent behavior definition

Task execution in the proposed system follows a leader-driven, resource-aware assignment behavior combined with heartbeat-based failure detection. The multi-agent environment contains one coordinating agent and a dynamically changing set of execution agents registered in the directory service. Communication is asynchronous and message delivery may be delayed; therefore, agents can become temporarily unreachable or permanently fail without explicit notification. Each computational job is modeled as a training task associated with an executable machine-learning model.

The coordinating agent maintains three conceptual structures: a queue of unassigned tasks, a record of which execution agent currently holds a task, and a record of the last time a response was received from each execution agent. Instead, the existence of an assignment record is interpreted as the execution agent being busy, while the absence of such a record means that the agent is free.

At periodic scheduling intervals, the coordinating agent discovers currently active execution agents through the directory service and sends a workload query to those that are not presently executing a task. Each execution agent replies with a numerical workload value (1) representing its current resource utilization. This value is computed from processor load, memory usage, and graphics-processor activity observed during the current measurement period and can be formally described as

$$L = f(CPU, RAM, GPU) \tag{1}$$

where L is the scalar workload (load) estimate; CPU, RAM, and GPU denote the measured utilization of the central processing unit, random-access memory, and graphics processing unit, respectively, during the current measurement interval; and $f(\cdot)$ is an aggregation function that combines these utilization measurements into a single load value. Lower L values correspond to higher availability.

Whenever a workload response is received and there are pending tasks, the coordinating agent assigns the next task from the queue to that execution agent. The assignment is recorded internally, and the time of the response is stored as the latest liveness indicator. Conceptually, this behavior corresponds to selecting an execution agent from the set of free agents according to equation (2)

$$w^* = arg\ min_{w \subseteq F}\ L(w) \tag{2}$$

where w denotes an execution agent; w^* is the selected agent; F is the set of agents that are currently idle (not executing a task); and $L(w)$ is the workload estimate reported by agent w. In the implementation, assignments occur immediately upon receiving availability responses, producing an online greedy approximation of this selection rule..

After receiving a task, the execution agent downloads the executable, performs local model training, and upon completion returns a message containing the execution duration and model identity. Receipt of this completion message releases the agent, allowing it to receive further tasks.

While a task is assigned, the coordinating agent periodically sends liveness requests. For each executing agent, the time elapsed since the last received message is compared to a fixed timeout threshold. An agent is considered unresponsive if the condition in equation (3) holds,

$$t_{now} - t_{last} > \Theta \tag{3}$$

where t_{now} is the current time at the coordinating agent, t_{last} is the timestamp of the most recent message received from the execution agent (workload response, heartbeat, or completion message), and Θ is the timeout threshold. In that case, the corresponding task is returned to the pending queue and the assignment record is removed. The task can then be reassigned to another available execution agent.

This behavior provides automatic recovery from failures and network interruptions. Since tasks are returned to the pending queue when an agent becomes unreachable, each task will eventually be executed provided that at least one execution agent remains available. If a previously unreachable agent later resumes after its task has been reassigned and subsequently reports completion, duplicate execution may occur. Therefore, the system offers at-least-once execution semantics rather than strict exact-once guarantees, emphasizing robustness and forward progress in unstable heterogeneous environments.

Experimental design

The experiments were designed to evaluate the impact of agent quantity and agent placement on overall ML training performance.

Three ML models, XGBoost, Random Forest, and CatBoost, were used as workloads. All models were trained on synthetic classification datasets generated using the make_classification function from scikit-learn, with fixed dataset sizes and hyperparameters to ensure experimental consistency.

Five workload sizes were evaluated: 3, 10, 20, 50, and 100 training tasks. For each workload size, experiments were run with a single Worker agent and with three Worker agents distributed across different devices. For the largest workload (100 tasks), two additional configurations were tested: four Worker agents with one agent per device, and six Worker agents with two agents running on each of three devices. For every configuration, the total execution time and the average training time per model were recorded.

Experimental Evaluation Metrics

System performance was evaluated using three metrics: total workload completion time, relative speedup compared to single-agent execution, and average training time per model. These metrics were used to assess scalability, parallelization efficiency, and the impact of resource contention when multiple agents run on the same physical device.

By varying both the number of agents and their distribution across heterogeneous devices, the methodology enables an empirical evaluation of scalability limits and communication overhead in agent-based distributed ML training. This approach highlights the trade-offs between increased parallelism and coordination costs in decentralized scheduling environments.

Results

The experimental results demonstrate that increasing the number of Worker agents generally reduces the total execution time required to train multiple machine-learning models, although the observed improvement is not linear and does not continue indefinitely.

Table 3 – Total time all task completion for each stage

Tasks	Agents	Total Time (seconds)
3	1	159
3	3	80
10	1	898
10	3	276
20	1	1381
20	3	516
50	1	3483
50	3	1536
100	1	6866
100	3	2691
100	4	2296

As shown in Table 3, for all workload sizes, scaling from one agent to three agents leads to a substantial reduction in total completion time.

Table 4 – Agents quantity to compilation time dependency

Tasks	Number of agents	Speedup
3	From 1 to 3	1.98× faster
10	From 1 to 3	3.25× faster
20	From 1 to 3	2.67× faster
50	From 1 to 3	2.26× faster
100	From 1 to 3	2.55× faster
100	From 1 to 4	2.99× faster
100	From 1 to 6 (2 agents on 3 devices)	2.34× faster

For small workloads of 3, 10, and 20 tasks, the speedup ranges from 1.98× to 3.25 according to Table 4, indicating that parallelization is highly effective when coordination overhead remains low. Similar behavior is observed for larger workloads: for 50 tasks, execution time decreases from 3483 s with one agent to 1536 s with three agents, while for 100 tasks, time is reduced from 6866 s to 2691 s. These results confirm that distributing ML workloads across multiple agents significantly improves performance compared to sequential execution. However, the performance gains begin to saturate as more agents are added. For the 100-task workload, increasing the number of Worker agents from three to four – each running on a separate physical device – further reduces total execution time to 2296 seconds, achieving the best observed speedup of 2.99× relative to single-agent execution. In contrast, further increasing the number of agents to six, implemented as two agents running concurrently on

each of three devices, results in a longer execution time of 2932 seconds, which is slower than the four-agent configuration and only marginally faster than using three agents. This behavior demonstrates that adding more agents does not always lead to better performance and that an optimal degree of parallelism exists. Table 5 summarizes the CPU, memory, and GPU utilization observed on a single Worker agent while executing the 20-task workload. The results indicate persistently high memory usage, whereas CPU and GPU utilization remain low on average with occasional short-lived peaks. Although these measurements reflect only one Worker on one device, they help explain the observed scaling limits: when multiple Workers are located on the same machine, the already-high RAM baseline makes memory contention more likely, which increases scheduling and communication overhead and reduces throughput.

Table 5 – Number of agents to compilation time dependency

Metric	Mean	Median	Min	Max
CPU (%)	0.29	0.11	0.00	8.42
RAM (%)	72.23	71.69	69.42	81.78
GPU (%)	0.43	0.20	0.05	8.44

The degradation observed in the six-agent configuration is primarily caused by resource contention and communication conflicts when multiple Worker agents operate on the same physical device. In this setup, agents compete for shared CPU and memory resources and frequently attempt to transmit training results to the Scheduler simultaneously, leading to message congestion, communication errors, agent reloads, and task reassignment. Figure 3 further supports this conclusion by showing that the average training time per model is consistently lower for the four-agent configuration than for the six-agent configuration across all model types, including XGBoost, Random Forest, and CatBoost. The higher bars for the six-agent setup indicate that the additional agents increase overhead rather than improving throughput. Overall, the results show that while increasing the number of agents initially improves system performance, excessive parallelism – especially when multiple agents are deployed on the same device - introduces coordination overhead that outweighs computational benefits. In the evaluated environment, configurations with three to four Worker agents provide the most efficient balance between parallel execution and communication cost, whereas deploying multiple agents per device leads to reduced efficiency and lower system reliability.

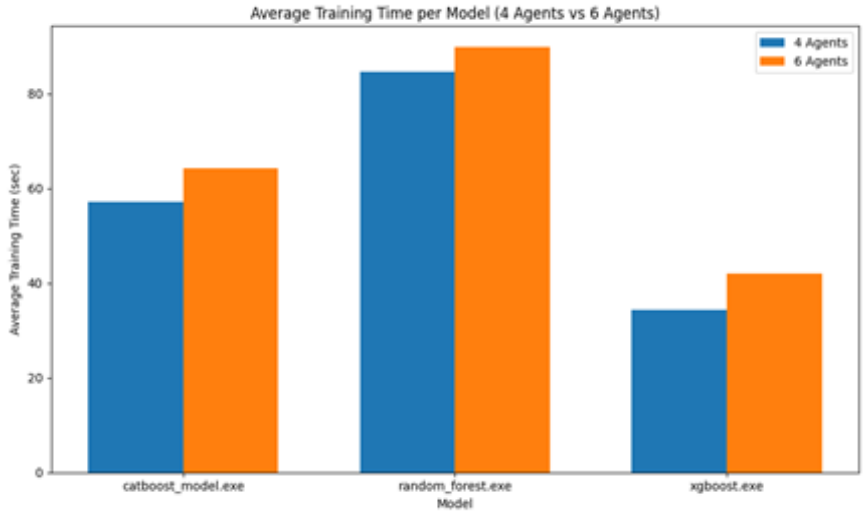

Figure 3 – Average training time per model

Discussion

The experimental results confirm several principles of distributed systems and multi-agent systems discussed in the literature, while also highlighting practical limitations that arise in real heterogeneous environments. Classical distributed systems theory emphasizes that parallelism improves performance only when coordination, communication, and resource-sharing overheads remain sufficiently low [1, 2]. The observed speedups when scaling from one to three or four Worker agents are consistent with this principle: the workload is partitioned effectively, Workers operate largely independently, and the Scheduler's coordination overhead remains manageable. This aligns with the view that scalability is achieved when computation dominates communication costs and coordination complexity remains bounded [1, 2].

However, the results also demonstrate that performance does not scale indefinitely. The degradation observed in the six-agent configuration illustrates the trade-off between concurrency and contention. When multiple agents run on the same physical device, they are no longer independent execution units; instead, they compete for shared CPU, memory, disk, and network resources. As a consequence, the effective throughput per agent decreases and overhead rises, producing diminishing returns and, eventually, worse total completion time. This behavior is consistent with findings in heterogeneous scheduling research, where parallelism beyond hardware capacity increases context switching, queueing delays, and coordination cost [6, 7].

From an agent-oriented perspective, the results reinforce core MAS concepts [3, 4]. Autonomy and asynchronous message passing enable dynamic participation and flexibility, but they also make the system sensitive to communication bursts. In the presented JADE-based design, multiple Workers may attempt to report results concurrently, and this can overload message handling on the Scheduler or cause transient failures in high-concurrency settings. The increased error rates and task reassignments observed when multiple agents share the same device reflect limitations of the current coordination and communication strategy under contention, rather than limitations of the MAS paradigm itself [5].

In the context of distributed machine learning, the findings align with established observations that communication overhead can dominate performance as parallelism increases [19]. Even though the trained models (XGBoost, Random Forest, CatBoost) run independently (without synchronized gradient exchange), the system still faces scalability limits due to coordination overhead, queueing, and result-transfer contention. This supports the practical need for resource-aware scheduling and placement strategies, especially in WLAN-based ad-hoc environments where network performance is variable.

Overall, the discussion highlights that the proposed JADE-based MAS successfully demonstrates the benefits of agent-oriented distributed computing on heterogeneous devices, particularly for small to medium workloads. At the same time, it confirms a key insight from distributed systems theory: there exists an optimal degree of parallelism, beyond which additional agents degrade performance rather than improve it. Future enhancements should therefore focus on (i) resource-aware placement (limiting concurrent Workers per device), (ii) reducing message bursts (e.g., batching, backoff, or queue-based reporting), and (iii) improving fault handling so that reassignment does not amplify congestion under load.

Conclusion

This study evaluated a JADE-based multi-agent system for distributing machine-learning training tasks across heterogeneous devices in a wireless LAN environment. Using a Scheduler–Worker architecture, the system coordinated the execution of XGBoost, Random Forest, and CatBoost workloads while dynamically reallocating tasks in case of failures. The experiments analyzed how workload size and the number of Worker agents affect overall performance.

The experimental results show that increasing the number of agents provides clear benefits up to a moderate level of parallelism. Scaling from one to three agents consistently reduced total completion time across all workload sizes, yielding speedups between 1.98 and 3.25 for smaller workloads and maintaining strong improvements for larger workloads (e.g., 50 and 100 tasks). For the 100-task queue, deploying four Worker agents on separate devices achieved the best observed performance, reducing total time to 2296 seconds and demonstrating that distributing computation across independent physical resources is the most effective scaling strategy in this environment.

At the same time, the results confirm that adding more agents does not guarantee further acceleration. The six-agent configuration, implemented as two agents per device on three machines, increased total completion time compared to the four-agent configuration and produced higher average training times per model. This degradation is primarily attributed to resource contention on shared devices and to communication conflicts when multiple Worker agents attempt to submit results to the Scheduler concurrently, leading to message delays, higher error rates, agent reloads, and task reassignment. These effects offset the expected benefits of additional parallelism and reveal a practical scalability limit for MAS deployments on heterogeneous edge-like hardware over WLAN.

The experiment demonstrates that a MAS can effectively coordinate distributed ML workloads in dynamic, heterogeneous settings, but achieving stable scalability requires explicit attention to placement and communication bottlenecks. Future work should focus on reducing coordination overhead and improving reliability under higher concurrency.

REFERENCES

1 Tanenbaum, A.S., Van Steen, M. Distributed Systems: Principles and Paradigms, 2nd ed. (Pearson Education, 2007).

2 Coulouris, G., Dollimore, J., Kindberg, T., Blair, G. Distributed Systems: Concepts and Design, 5th ed. (Addison-Wesley, 2011).

3 Wooldridge, M. An Introduction to Multi-Agent Systems, 2nd ed. (John Wiley & Sons, 2009).

4 Huhns, M.N., Singh, M.P. (editors) Readings in Agents (Morgan Kaufmann, 1998).

5 Greenwood, D., Bellifemine, F., Caire, G. Developing Multi-Agent Systems with JADE (Wiley, 2007).

6 Kalia, K., Gupta, N. Analysis of Hadoop MapReduce scheduling in heterogeneous environment. Ain Shams Engineering Journal, 12(1), 1101–1110 (2021). https:// doi.org/10.1016/j.asej.2020.06.009

7 Sewal, P., Singh, H. A Critical Analysis of Apache Hadoop and Spark for Big Data Processing. 2021 6th International Conference on Signal Processing, Computing and Control (ISPCC), 308–313 (2021). https:// doi.org/10.1109/ISPCC53510.2021.9609518

8 Raptis, T.P., Passarella, A. A survey on networked data streaming with Apache Kafka. IEEE Access, 11, 85333–85350 (2023). https://doi.org/10.1109/ACCESS.2023.3303810

9 Lee, J.-Y., Kim, M.-H., Shah, S.A.R., Ahn, S.-U., Yoon, H., Noh, S.-Y. Performance Evaluations of Distributed File Systems for Scientific Big Data in FUSE Environment. Electronics, 10(12), 1471 (2021). https://doi.org/10.3390/electronics10121471

10 Ma, C., Chi, Y. Evaluation test and improvement of load balancing algorithms of Nginx. IEEE Access, 10, 14311–14324 (2022). https://doi.org/10.1109/ACCESS.2022.3146422

11 Singh, N., Hamid, Y., Juneja, S., et al. Load balancing and service discovery using Docker Swarm for microservice based big data applications. Journal of Cloud Computing, 12, 4 (2023). https://doi.org/10.1186/s13677-022-00358-7

12 Burns, B., Beda, J., Hightower, K., Evenson, L. Kubernetes: Up and Running: Dive into the Future of Infrastructure (O'Reilly Media, 2022).

13 Pauloski, J.G., Rydzy, K., Hayot-Sasson, V., Foster, I., Chard, K. Accelerating Python applications with Dask and ProxyStore. arXiv preprint arXiv:2410.12092 (2024). https://doi.org/10.48550/arXiv.2410.12092

14 Ajitha, S. Methodology for Load Balancing in Multi-Agent System Using SPE Approach. Security Issues and Privacy Concerns in Industry 4.0 Applications, 207–227 (2021). https://doi.org/10.1002/9781119776529.ch11

15 Iturria-Rivera, P.E., Erol-Kantarci, M. Competitive Multi-Agent Load Balancing with Adaptive Policies in Wireless Networks. 2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC), 796–801 (2022). https://doi.org/10.1109/CCNC49033.2022.9700667

16 Li, Z., Yu, J., Liu, X., Peng, L. Load Balancing for Task Scheduling Based on Multi-Agent Reinforcement Learning in Cloud-Edge-End Collaborative Environments. Proceedings of the 2024 8th International Conference on Machine Learning and Soft Computing (ICMLSC ’24), 94–100 (2024). https://doi.org/10.1145/3647750.3647765

17 Rahmika, A.R., Tahir, Z., Paundu, A.W., Zainuddin, Z. Web server load balancing mechanism with least connection algorithm and multi-agent system. CommIT (Communication and Information Technology) Journal, 17(2), 245–258 (2023). https://doi.org/10.21512/commit.v17i2.8872

18 Binyamin, S.S., Ben Slama, S. Multi-Agent Systems for Resource Allocation and Scheduling in a Smart Grid. Sensors, 22(21), 8099 (2022). https://doi.org/10.3390/s22218099

19 Chatterjee, B. Distributed Machine Learning. Proceedings of the 25th International Conference on Distributed Computing and Networking (ICDCN ’24), 4–7 (2024). https://doi.org/10.1145/3631461.3632516

20 Dai, F., Hossain, M.A., Wang, Y. State of the Art in Parallel and Distributed Systems: Emerging Trends and Challenges. Electronics, 14(4), 677 (2025). https://doi.org/10.3390/electronics14040677

[1]*Кумалаков Б.,
PhD, қауымдастырылған профессор, ORCID ID: 0000-0003-1476-9542
*e-mail: bolatzhan.kumalakov@astanait.edu.kz
[1]Цой Д.,
к.ғ.қ., ORCID ID: 0009-0001-2711-5233
e-mail: tsoy.daniil.an@gmail.com
[1]Astana IT University, Астана қ., Қазақстан

AD-HOC ГРИД-ЕСЕПТЕУЛЕРІ ҮШІН АГЕНТТЕРДІҢ МІНЕЗ-ҚҰЛЫҒЫН ӘЗІРЛЕУ

Аңдатпа

Бұл зерттеу дәстүрлі жүйелер орталықтандырылған басқаруға және тұрақты инфрақұрылымға сүйенетін ad-hoc грид-есептеу орталарындағы тапсырмаларды тиімсіз жоспарлау және ақауға төзімділіктің шектеулілігі мәселесін қарастырады. Осы мәселені еңсеру үшін Java Agent DEvelopment Framework платформасын пайдалана отырып, гетерогенді Worker агенттері арасында тапсырмаларды автономды түрде қайта бөлуге мүмкіндік беретін децентрализденген агент мінез-құлқы әзірленді. Ұсынылған Жоспарлаушы–Жұмысшы архитектурасы орталықтандырылған оркестрациясыз динамикалық үйлестіруді және істен шығулардан кейін қалпына келуді қамтамасыз етеді. Бес құрылғыда жүргізілген тәжірибелер агенттер санын бірден үшке дейін арттыру жалпы орындалу уақытын 1,98–3,25 есеге қысқартатынын көрсетті, ал ең жоғары өнімділік төрт агент қолданылғанда байқалып, 100 тапсырма үшін 2,99 есе жылдамдату береді. Дегенмен, құрылғылар саны аз болған жағдайда алты агентті пайдалану ресурстар үшін бәсекелестіктің күшеюі мен байланыс шығындарының артуына байланысты тиімділікті 2,34 есе төмендетеді. Зерттеу бір ғана желілік топологиямен және шағын көлемді сынақ алаңымен шектеледі. Соған қарамастан, нәтижелер агенттерге негізделген децентрализденген жоспарлаудың ақауға төзімді үлестірілген машиналық оқыту жүйелерін құрудағы практикалық әлеуетін дәлелдейді.

Тірек сөздер: көп агентті жүйе, грид-есептеу, JADE, үлестірілген есептеу, ақауларға төзімділік, тапсырмаларды жоспарлау, агенттердің консенсусы.

[1]*Кумалаков Б.,
PhD, ассоциированный профессор, ORCID ID: 0000-0003-1476-9542,
*e-mail: bolatzhan.kumalakov@astanait.edu.kz
[1]Цой Д.,
м.н.с., ORCID ID: 0009-0001-2711-5233,
e-mail: tsoy.daniil.an@gmail.com
[1]Astana IT University, г. Астана, Қазақстан

РАЗРАБОТКА ПОВЕДЕНИЯ АГЕНТОВ ДЛЯ AD-HOC ГРИД-ВЫЧИСЛЕНИЙ

Аннотация

В этом исследовании рассматривается проблема эффективности планирования задач и ограниченной отказоустойчивости в специализированных грид-вычислительных средах, где традиционные системы полагаются на централизованное управление и стабильную инфраструктуру. Чтобы преодолеть это, было разработано поведение децентрализованных агентов с использованием платформы Java Agent DEvelopment Framework, позволяющей автономно перераспределять задачи между гетерогенными рабочими агентами. Предлагаемая архитектура Планировщик – Рабочий обеспечивает динамическую координацию и восстановление после сбоев без централизованной оркестровки. Эксперименты на пяти устройствах показывают, что увеличение числа агентов с одного до трех сокращает общее время выполнения в 1,98–3,25 раза, в то время как наилучшая производительность достигается при использовании четырех агентов, что обеспечивает ускорение в 2,99 раза при выполнении 100 задач. Однако использование шести агентов на меньшем количестве устройств снижает эффективность до 2,34 раза из-за нехватки ресурсов и увеличенной коммуникационной нагрузки. Исследование ограничено одной топологией сети и небольшим испытательным стендом. Тем не менее результаты демонстрируют практический потенциал децентрализованного планирования на основе агентов для создания устойчивых распределенных систем машинного обучения.

Ключевые слова: мультиагентная система, грид-вычисления, JADE, распределенные вычисления, отказоустойчивость, планирование задач, консенсус агентов.