Article

A Framework for Enhancing Multi-Agent Systems: Leveraging Microservices and DLT to Optimize Message Overhead and Load Balancing

Ching Han Chen [1], Ming Fang Shiu [1,*]

[1] Department of Computer Science and Information Engineering National Central University, Taiwan; pierre@g.ncu.edu.tw (C.H.C); 108582003@cc.ncu.edu.tw (M.F.S.)
* Correspondence: 108582003@cc.ncu.edu.tw; Tel.: +886-3-4207151 ext.35211

Abstract: Advancements in AI and computing hardware have facilitated the adoption of deep learning networks in cloud and edge computing, integrating distributed intelligent nodes into unified services. This paper introduces a practical framework for distributed computing using Multi-Agent Systems (MAS) and its publish-subscribe architecture, enhancing collaboration among distributed agents. MAS, with its robustness and scalability, suits large-scale systems but faces challenges like message overhead and load balancing. These are addressed through Microservices and Distributed Ledger Technology, which improve message efficiency and secure interactions. The framework effectively manages communications within a decentralized MAS model, optimizing performance and resource usage.

Keywords: Multi-Agent Systems, Distributed Artificial Intelligence, Publish-Subscribe Architecture, Decentralized Computing

1. Introduction

Advancements in AI and computing hardware have enabled the integration of deep learning networks in cloud and edge computing, centralizing Multi-Agent Systems (MAS) as a key element of Distributed Artificial Intelligence (DAI) [1], leveraging autonomous agents to achieve common goals [2]. This paper introduces a practical framework for distributed computing environments using MAS's publish-subscribe architecture, demonstrating its effectiveness in fostering collaboration among distributed intelligent agents. MAS technology, with its parallel computation, robustness, scalability, cost-effectiveness, and reusability, is ideal for large-scale systems [3]. However, challenges like message overhead and load balancing require strategies such as message compression, QoS adjustments, and consensus algorithms [4].

Message overhead is common in Many-to-One communications, where multiple client agents communicate with a single agent, leading to unnecessary message traffic [5]. Load balancing also poses challenges in One-to-Many scenarios, such as optimizing response efficiency and fault tolerance within a cluster. These challenges are addressed through a dynamic decentralized framework [6].

Microservices refine MAS technology by offering a scalable, adaptable architecture compatible with holonic and publish-subscribe models [7]. Breaking down complex systems into smaller services mitigates message overhead and facilitates load balancing via distributed processing, enhancing robustness and resource management. Microservices support various communication patterns and service discovery mechanisms, effectively addressing MAS's design challenges.

Distributed Ledger Technology (DLT) enhances MAS by promoting autonomy, scalability, and secure transactions without centralized control [8]. Adopting decentralized consensus principles, DLT enables autonomous decision-making and adaptation, boosting system resilience and efficiency. DLT's immutable record-keeping and distributed consensus algorithms reduce communication overhead and facilitate local data processing, marking a strategic advancement for dynamic coordination, secure interactions, and efficient decentralized operations in MAS.

Our approach develops a Multi-Agent Systems model based on the holonic agent concept, integrating Microservices and Distributed Ledger Technology to address communication challenges. For Many-to-One communication, Microservices improve scalability and manageability, ensuring efficient message distribution. DLT tackles One-to-Many challenges by ensuring secure, consensus-driven interactions among agents. Microservices for flexibility and DLT for secure aggregation validate the framework's efficiency in managing many-to-many communications within a decentralized MAS model.

The thesis explores integrating Multi-Agent Systems in distributed computing, detailing MAS model development, emphasizing publish-subscribe architecture and holonic agents, and rigorously testing system performance. The discussion analyzes outcomes and challenges, while the conclusion summarizes findings and future research directions.

2. Design

The design chapter integrates Multi-Agent Systems in distributed computing, addressing message overhead and load balancing. It uses Microservices for adaptability and Distributed Ledger Technology for decentralized interactions. Class diagrams and algorithms detail the framework's structure and dynamics. Focused experiments assess its effectiveness in reducing message overhead and enhancing load balancing, linking theory to practice.

2.1. Holonic Structures in Multi-Agents System

Multi-Agent Systems address issues with multiple computational units [9], each designated as an agent. MAS form networks of interacting agents to tackle complex challenges beyond individual capacities. Holonic MAS are beneficial for constructing intricate AI systems [10]. A holon integrates sub-holons within a broader system, originally describing social dynamics in biological species [11], now applied to manufacturing and business [12]. In MAS, a single agent may comprise multiple subagents. Each holon is managed by a head agent capable of interaction and communication. In homogeneous MAS, head agent selection can be arbitrary, like rotating leadership in wireless sensor networks. In heterogeneous setups, selection depends on agents' distinct capabilities. Holons may merge to form superholons, as shown in Fig. 1, where agents H-21 and H-31 coordinate with H-1, while H-4 is an atomic agent without subagents.

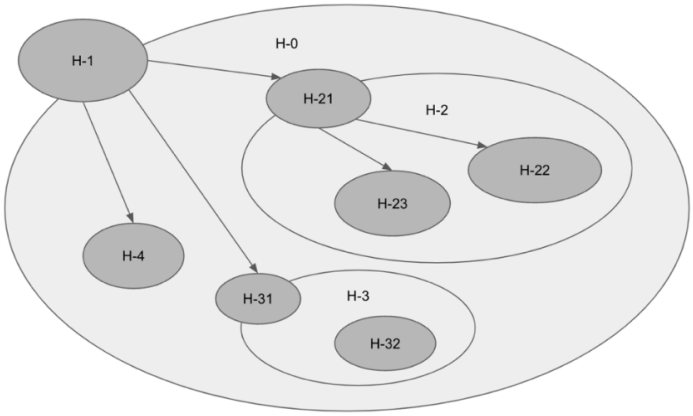


Fig. 1 Example of a holonic MAS

2.2. Foundational design

The holonic MAS architecture simulates biological tissues, using holonic MAS to design intelligent agents integrating various perceptions and actions. The resulting HolonicAgent highlights this integration.

The research encapsulates the Publish-Subscribe paradigm in the abstract MessageBroker class, ensuring flexibility with essential operations like start, stop, publish, and subscribe. Implementations like MQTT_Broker, DDS_Broker, and ROS_Broker cater to different communication frameworks. This modular approach allows interchangeable messaging systems.

To simplify HolonicAgent complexity, the BrokerNotifier interface intermediates MessageBroker interactions, promoting loose coupling and modularity. Fig. 2 illustrates these setups. The next phase will tackle advanced communication patterns.

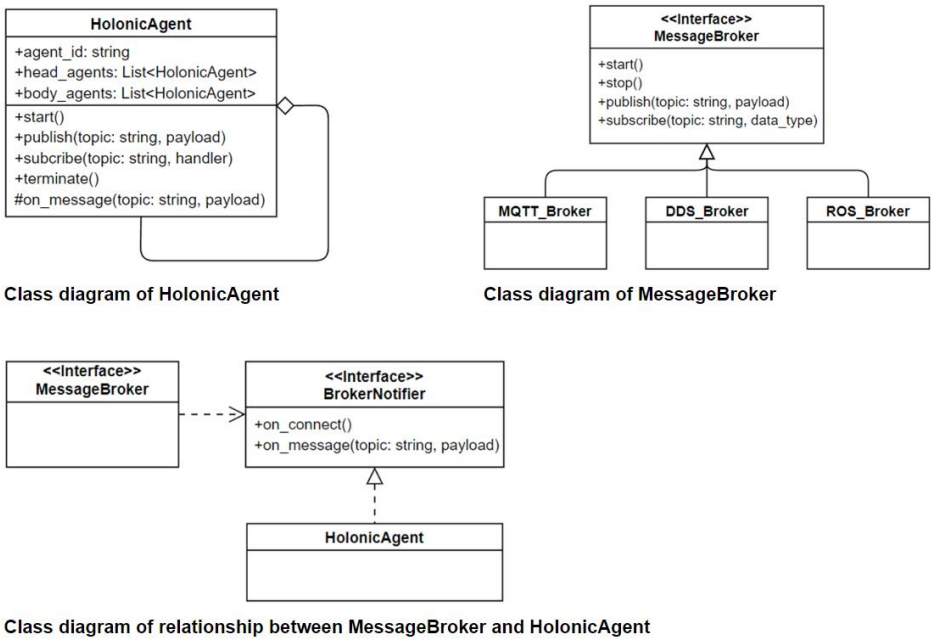
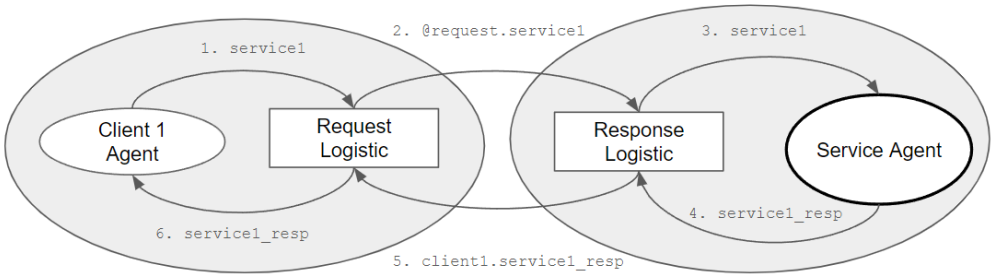


Fig. 2 Class diagrams of the holonic MAS architecture.

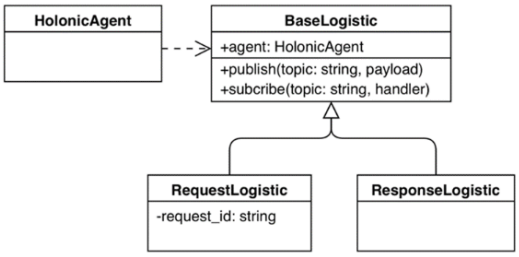
2.3. Tackling Many-to-One Communication Efficiency

The Many-to-One communication model in Multi-Agent Systems addresses the request-response issue in publish-subscribe architectures, where all clients receive every message, causing excessive data traffic. To solve this without altering agents' logic, we

introduced Logistic objects as selective couriers. These objects ensure messages reach only intended clients, preserving agents' process integrity. By utilizing a technique that subtly changes the topic so that only the intended agent receives it, they enhance communication efficiency. They integrate seamlessly with existing infrastructure, avoiding major redesigns. As intermediaries, Request and Response Logistics manage message topics, facilitating effective bidirectional communication. This method ensures messages reach recipients without broadcasting to all clients, maintaining system effectiveness with minimal agent workflow modifications.



Logistic Request-Response in Many-to-One communication model.



Class diagram of logistic.

Fig. 3 Many-to-One communication model with logistic objects.

2.4. Strategies for One-to-Many Communication Challenges

In Multi-Agent Systems, One-to-Many communications can cause conflicts when two service agents respond simultaneously, complicating coordination. Efficient load balancing and dynamic agent participation are crucial. Decentralized coordination optimizes system responsiveness and scalability. Using distributed architectures, agents can dynamically engage and disengage based on real-time demands, maintaining network equilibrium and enhancing performance.

Distributed Ledger Technology enhances decentralized coordination by maintaining a consistent record of transactions through replication and computational trust, ensuring transparency and security without a central authority. Using consensus algorithms, DLT autonomously determines the executing agent within a dynamic group, ensuring equitable load distribution and operational efficiency. A specially designed logistic object acts as an intermediary, managing task allocation to optimize load balancing across the cluster.

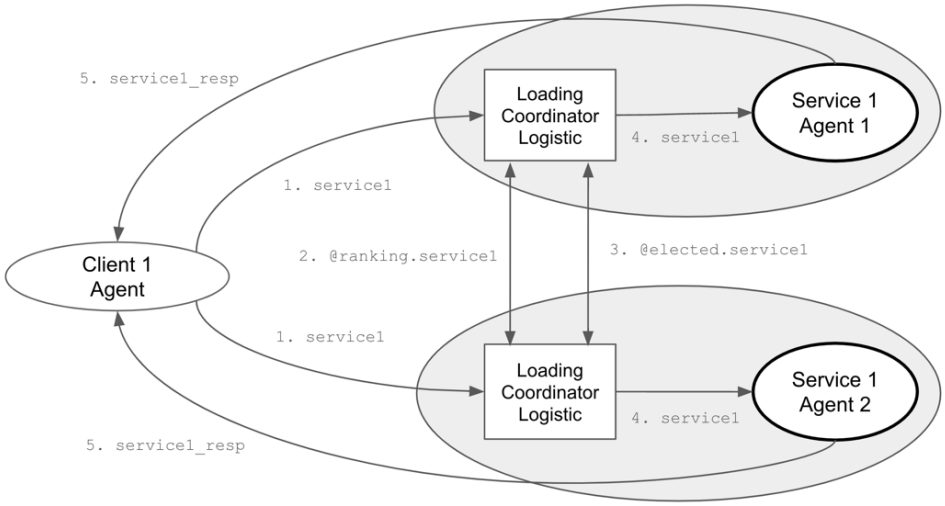


Fig. 4 Loading Coordinator Logistic in One-to-Many communication model.

Every LoadingCoordinator employs a consensus mechanism to identify the optimal executing agent. This is initiated by a coordinator who subscribes to specific topics to monitor new tasks and gather load-based rankings from various agents. Upon the arrival of a new task, the coordinator conducts an election by distributing its own ranking to assert its candidacy for handling the task. It then collects and aggregates rankings from all participating agents, which reflect each agent's current load and availability. Using a consensus algorithm, the coordinator determines the most suitable leader for the task, based on the lowest rank. If the coordinator's agent is elected, it directly undertakes the task; otherwise, it remains on standby, ready for future task assignments and elections.

Here is the consensus algorithm:

1. **Subscription Setup:** Upon initialization, the LoadingCoordinator subscribes to specific topics to listen for new tasks and ranking information.
2. **Task Arrival:** When a new task arrives, the coordinator begins an election to determine which agent should handle the task, publishing its own ranking based on its current load and a random factor.
3. **Rank Collection:** As rankings from other agents arrive, they are collected and stored.
4. **Leader Determination:** After a brief period to allow all ranks to be submitted, the coordinator determines the leader based on the lowest rank.
5. **Task Assignment:** If the responsible agent is elected, it will repeat this consensus algorithm and begin the task.

This decentralized approach ensures tasks are allocated fairly and efficiently, leveraging the system's distributed nature to optimize performance and responsiveness. The coordination process duration varies depending on the number of service agents involved. Using DLT, if other tasks arise before coordination is completed, they are recorded in a first-in, first-out manner for subsequent processing. The elected leader will initiate the retrieved tasks for a new round of work coordination.

2.5. Integration Strategies for Many-to-Many Communications

The Many-to-Many communication model in Multi-Agent Systems combines selective messaging with dynamic coordination. Using Request-Response and LoadingCoordinator logistics, it manages load balancing among agents. Request-Response handles point-to-point communication, ensuring efficient request-response matching. LoadingCoordinator orchestrates task distribution based on load and capacity,

optimizing network operations. This model benefits sectors like smart grids, healthcare, and supply chain management by improving responsiveness, operational efficiency, scalability, and flexibility through precise communication and equitable task distribution.

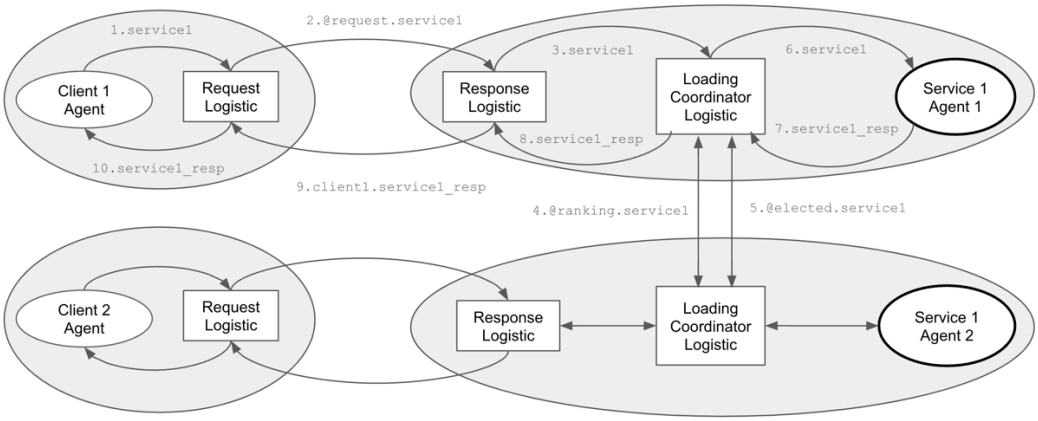


Fig. 5 Sequence Diagram of Integration

3. Experiment

In this experiment, we evaluate the efficacy of our Multi-Agent Systems architecture, integrating Microservices and Distributed Ledger Technology for optimized message distribution and load balancing. Methodical tests measure key performance indicators like message overhead and load distribution efficiency, providing empirical data to validate our design and highlight areas for future enhancement.

3.1. Experiment 1: Evaluating Request-Response Efficiency in Reducing Message Overhead

Objective:

Compare response time and message throughput between standard publish-subscribe and many-to-one techniques under varying loads from multiple client nodes.

Parameters:

1. Number of Client Nodes: 10, 50, 100 (to test under different scales).
2. Messages per Client: 100, 500, 1000 (to vary the load).
3. Message Size: Fixed at 1 KB to maintain consistency.
4. Message Frequency: 1 message per second per Client.
5. Service Efficiency: 100 messages per second.

Metrics to Measure:

1. Response Time: Time taken from sending a request to receiving a response.
2. Resource Usage: CPU and memory utilization of the service node.

Results and Observations:

Table 1. Comparison for both the normal and many-to-one communication

Clients/Messages	Communication Method	Response Time (ms/message)	Total Response Messages	Increased CPU Usage (%)	Increased Memory Usage (GB)

10 Clients/ 100 Messages	Normal	20.81	1000	1.42	0.54
	Many-to-One	21.57	100	1.95	0.36
50 Clients/ 500 Messages	Normal	53.23	25000	12.27	0.78
	Many-to-One	22.78	500	6.85	0.88
100 Clients/ 1000 Messages	Normal	2768.60	100000	14.02	1.67
	Many-to-One	25.24	1000	9.39	1.84

Here is the observations:

1. The "Normal" communication method shows substantial increases in Response Time due to processing bottlenecks from a high volume of total response messages (e.g., 100 clients generate 100,000 messages).
2. The "Many-to-One" method maintains stable, low response times by handling only necessary messages, reducing overhead and improving efficiency.
3. The "Many-to-One" method shows moderate CPU usage due to fewer messages but a slight increase in memory usage from the logistic mechanism, balancing resource management while effectively handling communications.

3.2. Experiment 2: Evaluating the Coordination Strategies on Response Efficiency

Objective:

Evaluate the efficiency of the LoadingCoordinator logistic model in managing high-volume requests from a single client to multiple service agents in a One-to-Many communication model.

Parameters:

1. Number of Service Agents: 1, 10, 50, 100 (testing scalability and load handling).
2. Number of Requests: Fixed at 1000 (consistent high load).
3. Request Complexity: Processing time per request fixed at 1 second.
4. Inter-Arrival Time of Requests: Fixed at 0.01 seconds.

Metrics to Measure:

1. Response Time: Time taken from sending a request to receiving a response.
2. Total Processing Time: Total time to complete all requests.
3. Resource Usage: CPU and memory utilization of the service node.

Results and Observations:

Table 2. Comparison for both the normal and one-to-many communication

Clients / Services 1000 messages	Total Processing Time (ms)	**Efficiency per Unit Service (%) *Note 1**	**Increased CPU Usage (%)**	**Increased Memory Usage (GB)**
1 Clients / 1 Service	1,005,106	99.12	0.83	0.209
1 Clients / 10 Services	118,229	84.58	2.08	0.479
1 Clients / 50 Services	34,554	58.70	7.21	1.040
1 Clients / 100 Services	56,335	17.75	14.20	2.313

Note 1:

- m = Number of messages = 1,000
- p = Service processing time per message = 1,000 milliseconds
- s = Number of services
- t = Total processing time
- Efficiency per Unit Service = $\frac{m \times p}{s \times t} \times 100\%$

Here is the observations:

1. Total Processing Time: Increasing services from 1 to 50 significantly reduces total processing time (from over 1 million ms to about 34,554 ms), highlighting improved parallel processing. However, processing time increases with 100 services, indicating potential overhead or resource contention issues.

2. Efficiency per Unit Service: Efficiency inversely correlates with the number of services. It is highest at 99.12% with one service but drops to 18.47% with 100 services, suggesting diminishing returns due to increased complexity and coordination overhead.

3. Increased CPU and Memory Usage: As services increase from 1 to 100, CPU usage rises from 0.83% to 14.20%, and memory usage grows from 0.209 GB to 2.313 GB, indicating higher resource demands.

In conclusion, increasing services improves processing time up to a point but also increases CPU and memory usage while decreasing efficiency. Finding an optimal balance between resource usage and performance is crucial for one-to-many communication models.

4. Conclusions

This research integrates Multi-Agent Systems with Microservices and Distributed Ledger Technology to enhance distributed computing. Our framework improves scalability, robustness, and efficiency by strategically deploying these technologies. It features a holonic agent organization with a publish-subscribe model, enabling flexible system design for complex agent interactions.

Microservices minimize message overhead and enhance load balancing by breaking systems into smaller, autonomous services. This modular approach improves resource efficiency and supports dynamic scaling. Load balancing enables decentralized consensus and interactions, boosting system resilience and privacy while reducing central control risks. Experimental results show our framework effectively addresses MAS challenges in communication and responsiveness. Future research will refine these integrations, focusing on AI-driven decision-making, fault tolerance, and new consensus algorithms.

References

1. Chaib-Draa, B., Moulin, B., Mandiau, R., & Millot, P. (1992) Trends in distributed artificial intelligence. *Artificial Intelligence Review*, *6*(1), 35-66. https://doi.org/10.1007/BF00155579
2. Balaji, P. G., & Srinivasan, D. (2010) An introduction to multi-agent systems. In *Innovations in multi-agent systems and applications-1,* pp 1-27. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-14435-6_1
3. Gaud, N., Galland, S., Gechter, F., Hilaire, V., & Koukam, A. (2008) Holonic multilevel simulation of complex systems: Application to real-time pedestrians simulation in virtual urban environment. *Simulation Modelling Practice and Theory*, *16*(10), 1659-1676. https://doi.org/10.1016/j.simpat.2008.08.015
4. Van Glabbeek, R., Deac, D., Perale, T., Steenhaut, K., & Braeken, A. (2022). Flexible and Efficient Security Framework for Many-to-Many Communication in a Publish/Subscribe Architecture. In *Sensors, 22*(19), 7391. MDPI. https://doi.org/10.3390/s22197391

5. Ferraz Junior, N., Silva, A. A. A., Guelfi, A. E., & Kofuji, S. T. (2022). Performance evaluation of publish-subscribe systems in IoT using energy-efficient and context-aware secure messages. In *Journal of Cloud Computing, 11*(6). SpringerOpen. https://doi.org/10.1186/s13677-022-00278-6
6. Nandagopal, M., Uthariaraj, V.R. (2011). Decentralized Dynamic Load Balancing for Multi Cluster Grid Environment. In *Meghanathan, N., Kaushik, B.K., Nagamalai, D. (eds) Advanced Computing. Communications in Computer and Information Science*, vol 133. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-17881-8_15
7. Hannousse, A., Yahiouche, S. (2021). Securing microservices and microservice architectures: A systematic mapping study. In *Computer Science Review, 41*. Elsevier. https://doi.org/10.1016/j.cosrev.2021.100415
8. Perdana, A., Robb, A., Balachandran, V., Rohde, F. (2021). Distributed Ledger Technology: Its Evolutionary Path and the Road Ahead. *In Information & Management, 58*(3). Elsevier. https://doi.org/10.1016/j.im.2020.103316
9. Khosla, R., & Ichalkaranje, N. (2004) *Design of intelligent multi-agent systems: human-centredness, architectures, learning and adaptation* (Vol. 162). Springer Science & Business Media.
10. Rodriguez, S., Hilaire, V., Gaud, N., Galland, S., & Koukam, A. (2011) Holonic multi-agent systems. In *Self-organising Software,* pp 251-279. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-17348-6_11
11. Koestler, A. (1967) The ghost in the machine, hutchinson & co. *Publishers Ltd., London*.
12. Giret, A., & Botti, V. (2005) Analysis and design of holonic manufacturing systems. In *18th International Conference on Production Research (ICPR2005)*.