DyOrchestration: Dynamic Orchestration of Multi-Agent Service Chains

Ruichao Zhang[1,2], Lizhuang Tan[1,2], Chengxiao Yu[3], Yanyan Tan[4], Peiying Zhang[1,5]
[1]*Key Laboratory of Computing Power Network and Information Security, Ministry of Education, Shandong Computer Science Center (National Supercomputer Center in Jinan), Qilu University of Technology (Shandong Academy of Sciences), Ji'nan, China*
[2]*Shandong Provincial Key Laboratory of Computing Power Internet and Service Computing, Shandong Fundamental Research Center for Computer Science, Ji'nan, China*
[3]*Department of New Networks, Pengcheng Laboratory, Shenzhen, China*
[4]*School of Computer Science and Artificial Intelligence, Shandong Normal University, Jinan, China*
[5]*College of Computer Science and Technology, China University of Petroleum (East China), Qingdao, China*
Email: 10431250187@stu.qlu.edu.cn, tanlzh@sdas.org, yuchx@pcl.ac.cn, yytan928@sdnu.edu.cn, zhangpeiying@upc.edu.cn

***Abstract*—Complex agent tasks are often decomposed into multiple subtasks and processed by different sub-agents, forming an agent service chain. Existing methods mainly focus on logical sub-agent configuration, but often leave execution endpoints implicit. In distributed multi-server environments, feasible endpoints may differ in communication delay, queueing load, and execution capability, so capability-feasible endpoints are not necessarily latency-efficient. We propose DyOrchestration, a centralized orchestration method that jointly considers communication, queueing, and execution costs, and supports both single-task and batch-aware multi-task selection. Experiments show that, at 8 req/s, the batch-aware variant of DyOrchestration reduces average completion latency by up to 80% compared with AORCH and Nearest, while mitigating endpoint hotspots and improving completed throughput under concurrent workloads.**

***Index Terms*—Multi-Agent Systems, Agent Orchestration, Agent Service Chains, Dynamic Endpoint Selection**

I. Introduction

Complex tasks increasingly require multi-step reasoning, tool invocation, and heterogeneous model capabilities. As a result, multi-agent systems are gradually moving from single-agent designs or fixed workflows toward orchestrator-subagent architectures. Prior studies have shown that role specialization, collaborative dialogue, and specialized agents can improve complex-task solving [1], [2]. Recent agent frameworks further demonstrate the value of multi-agent interaction in complex tasks [3], [4]. We focus on distributed agent service chains, where an orchestrator decomposes a request into multiple subtasks and progressively delegates them to distributed endpoints.

Existing approaches mainly focus on logical sub-agent creation, i.e., determining the instruction, context, tools, and model capability for each subtask. These approaches address how agents can be dynamically created. However, a logical configuration only specifies the required capabilities of a subtask. It does not decide where the corresponding sub-agent should be created and executed. The same logical configuration may be supported by multiple feasible endpoints, while their network conditions, queue loads, and execution capabilities can differ. Therefore, capability feasibility does not imply latency efficiency. Prior inference-serving studies have shown that runtime resource selection and load states significantly affect service latency and throughput [5], [6]. However, they usually do not directly address agent service-chain endpoint selection after logical sub-agent configuration generation.

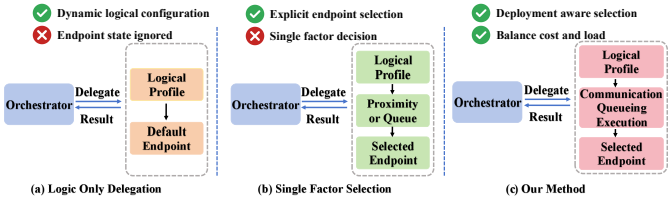


Fig. 1: Motivation of deployment-aware endpoint selection.

If multiple ready subtasks independently select endpoints at the same scheduling time, they may converge on the same endpoint and amplify queueing pressure. As shown in Fig.1, logic-only delegation ignores endpoint state, while single-factor selection explicitly introduces endpoint selection but still relies on a single factor such as proximity or queue state. Therefore, endpoint selection for agent service chains should jointly consider communication, queueing, and execution costs, and further account for assignment interactions among subtasks in the same batch [7].

To this end, we propose DyOrchestration, a deployment-aware endpoint selection method for distributed multi-agent service chains. DyOrchestration retains a centralized orches-

This work was supported by the Shandong Provincial Natural Science Foundation under Grant No.ZR2024LZH006, the National Natural Science Foundation of China under Grant No.62402257, the Pilot Project for Integrated Innovation of Science, Education and Industry of Qilu University of Technology (Shandong Academy of Sciences) under Grant No.2025ZDZX01, the Key Laboratory of Computing Power Network and Information Security, Ministry of Education under Grant No.2024PY009, the Major Key Project of PCL under Grant No.PCL2025A08.

trator architecture. After logical sub-agent configuration generation, it delegates subtasks by jointly considering communication overhead, queue load, and execution cost. For a single ready subtask, it selects the endpoint with the minimum overall cost. For multiple ready subtasks in the same scheduling batch, it adopts batch-aware greedy selection and uses virtual waiting-time updates to reduce the risk that subtasks in the same batch concentrate on the same endpoint. The main contributions of this paper are as follows.

- **Problem formulation.** We formulate the deployment-aware endpoint selection problem in distributed multi-agent service chains, and incorporate creation and execution endpoint decisions after logical sub-agent configuration generation into the orchestration process.
- **DyOrchestration design.** We design DyOrchestration within a centralized orchestrator architecture. It jointly considers communication overhead, queue load, and execution cost, and supports both single-task selection and batch-aware multi-task selection.
- **Controlled evaluation.** We build a controlled agent service-chain evaluation, showing that DyOrchestration reduces average completion latency by up to 80% over logic-only and proximity-first baselines, while mitigating endpoint hotspots and improving completed throughput under concurrent loads.

II. Related Work and Motivation

We discuss related work from three aspects, including multi-agent orchestration, agent service chains, and deployment-aware endpoint selection.

A. Multi-Agent Orchestration

Multi-agent orchestration mainly studies how multiple agents are organized, coordinated, and dynamically selected. Zhang *et al.* [8] proposed Chain-of-Agents, where multiple worker agents sequentially process long-context segments and a manager agent aggregates the results. Liu *et al.* [9] proposed DyLAN, which dynamically selects an agent team and adjusts the communication structure according to task requirements. Zhuge *et al.* [10] proposed GPTSwarm, which models language agents as optimizable computational graphs and improves agent orchestration through node/edge optimization. These studies show that agent selection, role assignment, and communication topology are key factors in multi-agent collaboration. However, their adaptivity mainly lies in the logical collaboration layer. In contrast, we focus on endpoint selection after logical sub-agent configuration generation.

B. Agent Service Chains

Complex agent tasks usually cannot be completed by a single LLM call; instead, they require stepwise invocations of tools, models, APIs, functions, or sub-agents. Schick *et al.* [11] proposed Toolformer, which enables language models to learn when to call external APIs, how to pass arguments, and how to use returned results. Shen *et al.* [12] proposed HuggingGPT, where an LLM controller performs task planning, model selection, task execution, and response generation by invoking external expert models for complex multimodal tasks. Yu *et al.* [13] studied serverless function orchestration and showed that complex applications often consist of multiple function workflows, where intermediate data exchange among functions can significantly affect execution efficiency. Prior work shows that complex agent tasks often involve chained invocations and intermediate-result transfer. We therefore abstract orchestrator-generated subtasks as an agent service chain, while creation and execution endpoint selection after logical sub-agent configuration remains largely implicit.

C. Deployment-Aware Endpoint Selection

In distributed systems, execution placement directly affects communication delay, resource contention, queueing delay, and end-to-end completion time. Zhu *et al.* [14] proposed an edge-cloud workflow scheduling method for scheduling dependent workflows under dynamic workloads. Raza *et al.* [15] proposed COSE, which jointly selects configuration and placement for serverless applications. Xu *et al.* [16] proposed a placement method for stateful serverless applications, considering function/state dependencies, data size, and network latency. Tang *et al.* [17] proposed a joint decision method for resource overbooking and container scheduling, capturing the coupling between resource heterogeneity and scheduling decisions. Prior work confirms that deployment location, runtime load, and resource heterogeneity affect distributed execution performance. However, existing methods mainly target workflows, serverless functions, or containers, while we focus on capability-constrained endpoint selection for agent service-chain subtasks, jointly considering communication, queueing, execution costs, and batch-induced load.

III. Methodology

This section presents DyOrchestration, a deployment-aware endpoint selection method for distributed multi-agent service chains. After logical sub-agent configuration is generated, DyOrchestration selects the endpoint for creation and execution according to network state, endpoint load, and processing capability. As shown in Fig.2 (a), DyOrchestration follows a centralized orchestration process that integrates agent profiling, endpoint placement, and sub-agent execution. We then formulate the endpoint selection problem and introduce single-task selection and batch-aware multi-task selection.

A. Problem Formulation

We study agent service-chain orchestration in a distributed multi-server environment. The orchestrator decomposes a user request into subtasks and generates the corresponding logical sub-agent configurations. Existing methods mainly focus on logical requirements, such as instruction, context, tools, and model capability, but rarely consider the creation and execution endpoint. In distributed deployments, the same logical configuration may be supported by multiple feasible endpoints with different network paths, bandwidth, communication delays, queue states, and processing capabilities. Therefore, capability

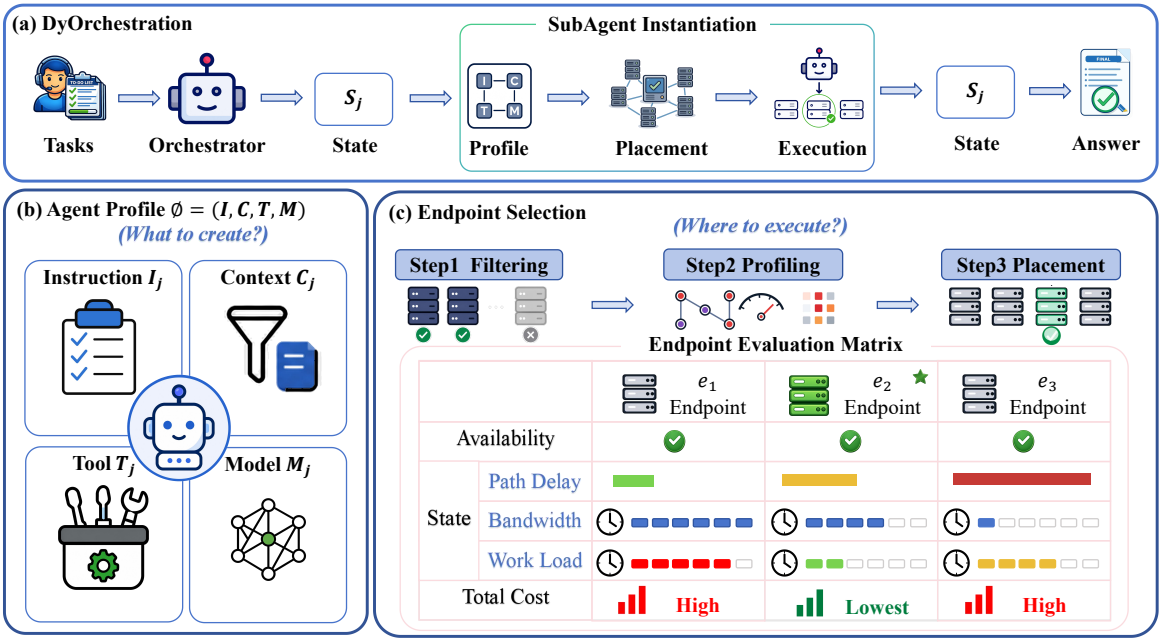


Fig. 2: Overview of DyOrchestration.

feasibility does not imply latency efficiency. We first model a user request as an agent service chain.

Let a user request be denoted by r. In the orchestrator–subagent architecture, the orchestrator decomposes the request into a set of interrelated subtasks, represented as

$$C_r = \langle v_{r,1}, v_{r,2}, \ldots, v_{r,J_r} \rangle. \tag{1}$$

Where C_r denotes the service chain corresponding to request r, $v_{r,j}$ denotes the j-th subtask, and J_r denotes the number of subtasks. We treat task decomposition and logical configuration generation as upstream orchestration results, and focus on selecting the creation and execution endpoint for each ready subtask.

For subtask $v_{r,j}$, its logical sub-agent configuration is defined as

$$\Phi_{r,j} = \langle I_{r,j}, C_{r,j}, T_{r,j}, M_{r,j} \rangle. \tag{2}$$

As illustrated in Fig2 (b), $I_{r,j}, C_{r,j}, T_{r,j}$, and $M_{r,j}$ denote the instruction, context, tool set, and model capability of the subtask, respectively. This configuration specifies the logical execution requirements of the subtask, but it is not yet bound to a specific execution endpoint.

Let the candidate endpoint set be $\mathcal{E} = \{e_1, e_2, \ldots, e_N\}$. Since endpoints may differ in supported models, tools, and resource states, the feasible endpoint set of subtask $v_{r,j}$ at time t is defined as

$$\mathcal{E}_{r,j}(t) = \{e \in \mathcal{E} \mid a_e(\Phi_{r,j}, t) = 1\}. \tag{3}$$

Where $a_e(\Phi_{r,j}, t) \in \{0, 1\}$ indicates whether endpoint e satisfies the execution requirements of configuration $\Phi_{r,j}$. To unify single-request and multi-request concurrent scenarios, we use u to denote a ready subtask waiting for endpoint selection, and abbreviate $\Phi_{r,j}$ and $\mathcal{E}_{r,j}(t)$ as Φ_u and $\mathcal{E}_u(t)$, respectively.

Endpoint selection depends on the current system state. We denote the runtime state at time t as

$$\Omega_t = \langle G, B(t), d(t), Q(t), \mu(t) \rangle. \tag{4}$$

Where G denotes the distributed server and network topology, $B(t)$ denotes the available path bandwidth, $d(t)$ denotes the base communication delay, $Q(t)$ denotes the endpoint queue state, and $\mu(t)$ denotes the recent effective processing rate of endpoints. These states can be observed or estimated from network monitoring, scheduler queues, and worker runtime logs.

Let l_u denote the input data location of subtask u, and let z_u denote the data size to be transferred before execution. For a candidate endpoint $e \in \mathcal{E}_u(t)$, the communication delay is defined as

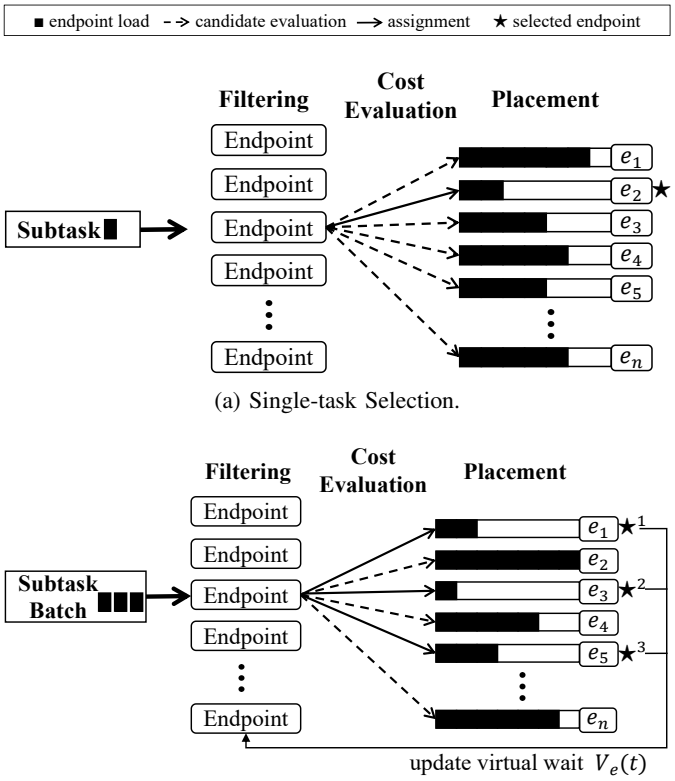
$$D_u^{comm}(e, t) = d_{l_u,e}(t) + \frac{z_u}{B_{l_u,e}(t)}. \tag{5}$$

Where $d_{l_u,e}(t)$ and $B_{l_u,e}(t)$ denote the base communication delay and available bandwidth from l_u to endpoint e, respectively.

In addition to communication delay, endpoint load and processing capability also affect completion time. For endpoint e, its estimated waiting time is defined as

$$D_e^{wait}(t) = W_e(t) = \frac{q_e(t)}{\mu_e(t)}. \tag{6}$$

Where $q_e(t)$ denotes the number of tasks currently waiting at endpoint e, and $\mu_e(t)$ denotes its recent effective processing rate. This approximation provides a lightweight estimate of endpoint waiting time.



(a) Single-task Selection.

(b) Batch-aware Multi-task Selection.

Fig. 3: Endpoint selection workflows.

Furthermore, let $D_u^{exec}(e,t)$ denote the estimated execution time of subtask u on endpoint e. The endpoint-side processing delay of subtask u on endpoint e is then defined as

$$D_u^{proc}(e,t) = D_e^{wait}(t) + D_u^{exec}(e,t). \tag{7}$$

Finally, the endpoint selection cost of delegating subtask u to candidate endpoint e is defined as

$$J_u(e,t) = D_u^{comm}(e,t) + D_u^{proc}(e,t). \tag{8}$$

This cost jointly captures network transmission overhead, endpoint waiting time, and endpoint execution time. Based on this model, the next subsection introduces DyOrchestration in single-task and multi-task endpoint selection scenarios.

B. DyOrchestration

After logical sub-agent configuration generation, DyOrchestration selects an execution endpoint among capability-feasible candidates by jointly considering communication delay and endpoint-side processing delay. As shown in Fig.2 (c), this process includes feasible endpoint filtering, runtime state profiling, and endpoint placement. As shown in Fig.3, the endpoint selection workflows are illustrated.

1) Single-task selection: As shown in Fig. 3 (a), when only one subtask u is ready at a scheduling instant, DyOrchestration performs single-task selection. This process first filters the feasible endpoint set $\mathcal{E}_u(t)$ according to the logical configuration Φ_u. It then computes the endpoint selection cost $J_u(e,t)$ for each candidate endpoint.

The objective of single-task endpoint selection is

$$e_u^* = \arg\min_{e \in \mathcal{E}_u(t)} J_u(e,t). \tag{9}$$

Where, e_u^* denotes the creation and execution endpoint selected for ready subtask u. This rule does not simply choose the endpoint with the minimum communication delay, nor does it simply choose the endpoint with the shortest queue. Instead, it selects the endpoint with the minimum overall cost $J_u(e,t)$.

After obtaining e_u^*, we combine the logical configuration with the endpoint selection result to form a deployment-aware sub-agent configuration $\Psi_u = (\Phi_u, e_u^*) = (I_u, C_u, T_u, M_u, e_u^*)$. Where, Ψ_u denotes the final delegation configuration of subtask u. It contains both the logical sub-agent configuration Φ_u and the creation and execution endpoint e_u^* selected by DyOrchestration. The orchestrator then delegates the subtask to the target endpoint according to Ψ_u, creates or invokes the corresponding sub-agent on that endpoint, and updates the service-chain context and the input location of subsequent subtasks after execution completes.

Single-task selection is the basic decision unit of DyOrchestration. It is suitable for low-concurrency scenarios or scheduling instants where only one subtask is ready. However, in multi-request concurrent scenarios, multiple subtasks may become ready at the same time. If each subtask independently applies (9), multiple subtasks may select the same endpoint that currently has the lowest cost, thereby creating new queueing pressure at that endpoint within a short period. To address this issue, DyOrchestration further introduces multi-task selection.

2) Multi-task selection: As shown in Fig. 3 (b), in multi-request concurrent scenarios or when multiple service chains advance simultaneously, multiple subtasks may become ready at the same scheduling instant. If each subtask performs single-task selection independently, multiple subtasks may select the same low-cost endpoint at the same time. This can introduce new queueing pressure. To address this issue, DyOrchestration adopts a batch-aware greedy endpoint selection strategy. It processes the ready subtasks in the current batch according to priority order and updates the virtual waiting time of the selected endpoint after each assignment.

Let the ready subtask set at time t is

$$U(t) = \{u_1, u_2, \ldots, u_K\}. \tag{10}$$

Where, K denotes the number of ready subtasks in the current scheduling batch. DyOrchestration first sorts $U(t)$ according to a lightweight priority rule. The priority mainly considers the number of feasible endpoints, ready time, and estimated workload.

To capture the impact of assignments already made within the current batch, DyOrchestration maintains a virtual waiting time for each endpoint. The virtual waiting time is initialized as the current estimated waiting time

$$V_e(t) = D_e^{wait}(t), \quad \forall e \in \mathcal{E}. \tag{11}$$

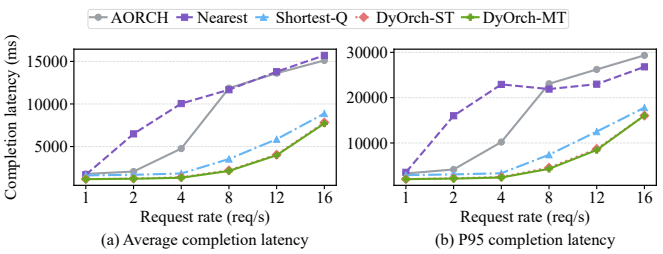


Fig. 4: Completion Latency Comparison.

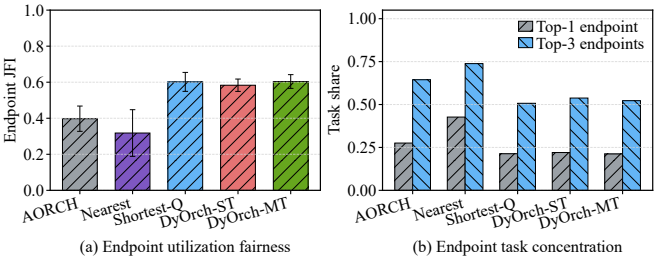


Fig. 6: Endpoint Load and Hotspots.

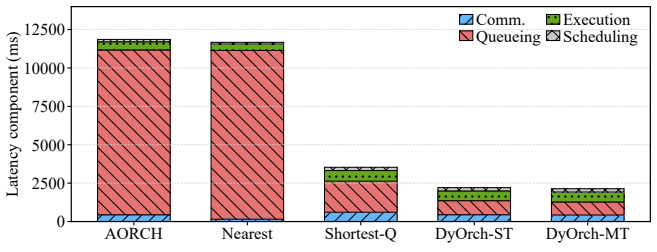


Fig. 5: Request-level Latency Breakdown.

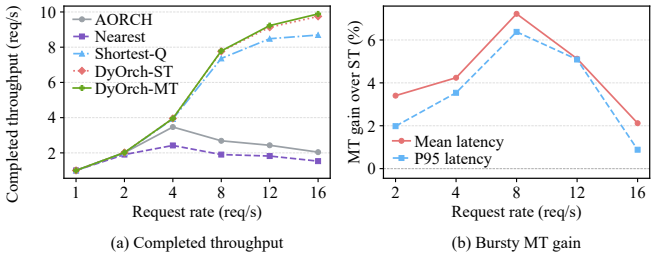


Fig. 7: Throughput and Batch-aware Gain.

Where, $V_e(t)$ denotes the virtual waiting time of endpoint e in the current batch. Initially, $V_e(t)$ is equal to the current estimated waiting time of the endpoint. When a subtask is assigned to this endpoint, $V_e(t)$ is updated to reflect the additional waiting imposed on subsequent subtasks by this assignment.

For the currently processed ready subtask u, DyOrchestration defines the batch-aware cost of assigning it to candidate endpoint e as

$$J_u^m(e,t) = D_u^{com}(e,t) + V_e(t) + D_u^{exe}(e,t). \quad (12)$$

Where, the superscript m denotes the multi-task scenario. The term $V_e(t)$ reflects the additional waiting introduced by existing assignments to endpoint e within the current batch. DyOrchestration then selects the endpoint with the minimum batch-aware cost

$$e_u^* = \arg\min_{e \in \mathcal{E}_u(t)} J_u^m(e,t). \quad (13)$$

After the assignment is completed, the virtual waiting time of the selected endpoint is updated as

$$V_{e_u^*}^{new}(t) = V_{e_u^*}^{old}(t) + D_u^{exec}(e_u^*, t). \quad (14)$$

This update allows subsequent subtasks to perceive the execution load already introduced within the current batch when they evaluate the same endpoint. It therefore reduces the risk that multiple subtasks are concentrated on the same low-cost endpoint at the same time. After endpoint selection is completed for all ready subtasks, DyOrchestration obtains the set of deployment-aware configurations

$$\{\Psi_u = (\Phi_u, e_u^*) \mid u \in U(t)\}. \quad (15)$$

The subtasks are then dispatched to their corresponding endpoints for execution.

The above multi-task selection is a batch-aware extension of single-task selection for concurrent scenarios. When $|U(t)| = 1$, the process reduces to single-task selection. When $|U(t)| > 1$, the virtual waiting-time update allows endpoint selection to perceive existing assignments within the same batch. It should be noted that this process is a lightweight online greedy heuristic rather than a globally optimal scheduling algorithm.

IV. Experimental Results and Analysis

We evaluate DyOrchestration under heterogeneous endpoint and network conditions, focusing on latency, load distribution, throughput, and the effect of batch-aware selection.

A. Experimental Setup

The workload follows a Poisson arrival process and is repeated over five random seeds. Each setting contains 300 requests, with request rates from 1 to 16 req/s. The system includes 12 heterogeneous endpoints, service-chain lengths of 3, 5, and 8, and varied network states, input sizes, and execution capabilities. Bursty arrivals are used as an ablation setting for batch-aware multi-task selection.

We compare five endpoint selection strategies. AORCH [18] serves as a logic-only baseline, which delegates subtasks only based on the logical sub-agent configuration. Nearest [19] represents proximity-first endpoint selection, which selects the feasible endpoint with the lowest communication delay. Shortest-Q [20] represents queue-aware endpoint selection, which selects the feasible endpoint with the shortest estimated waiting time. DyOrch-ST jointly considers communication, queueing, and execution costs, while DyOrch-MT further introduces intra-batch virtual waiting-time updates. The evaluation metrics include completion latency, latency breakdown, endpoint load distribution, completed throughput, and MT-over-ST latency gain under the bursty setting.

B. Experimental Results

We evaluate DyOrchestration from four aspects: completion latency, latency breakdown, endpoint load distribution, and completed throughput.

As shown in Fig. 4, AORCH and Nearest exhibit rapidly increasing average and P95 completion latency as the offered load grows, because AORCH ignores runtime endpoint states while Nearest repeatedly selects nearby endpoints. DyOrch-ST and DyOrch-MT maintain lower latency by jointly considering communication, queueing, and execution costs; at 8 req/s, DyOrch-MT achieves about 2.13 s average latency, while AORCH and Nearest exceed 11 s.

As shown in Fig. 5, the latency gap mainly comes from queueing delay. Although Nearest has the lowest communication cost, endpoint concentration causes severe queueing, while DyOrch-ST and DyOrch-MT keep the latency components more balanced.

As shown in Fig. 6, AORCH and Nearest create endpoint hotspots, with lower utilization fairness and higher Top-1/Top-3 task shares. Shortest-Q improves load distribution, but its latency remains higher than DyOrch because it ignores communication and execution costs.

As shown in Fig. 7, DyOrch-ST and DyOrch-MT sustain higher completed throughput under high offered load. Under bursty arrivals, DyOrch-MT gains additional latency reduction through virtual waiting-time updates. The results show that DyOrchestration improves service-chain performance mainly by reducing queueing amplification and endpoint concentration.

Overall, the experimental results show that deployment-aware endpoint selection is a key factor affecting agent service-chain performance. By jointly considering communication, queueing, and execution costs, DyOrchestration reduces completion latency, mitigates endpoint hotspots, and improves completed throughput under concurrent workloads.

V. Conclusion

This paper studied deployment-aware endpoint selection for multi-agent service chains in distributed multi-server environments. DyOrchestration keeps a centralized orchestrator architecture and selects execution endpoints for ready subtasks by jointly considering communication, queueing, and execution costs under capability constraints. It supports both single-task and batch-aware multi-task selection to reduce endpoint concentration under concurrent batches. Experiments show that DyOrchestration reduces average completion latency by up to 80% compared with logic-only and proximity-first baselines, while mitigating endpoint hotspots and improving completed throughput.

Future work will validate DyOrchestration in real multi-server agent systems and extend it toward more accurate runtime state estimation, richer service-chain dependencies, endpoint failures, cost constraints, and energy-aware deployment.

References

[1] S. Hong, M. Zhuge, J. Chen, X. Zheng, Y. Cheng, C. Zhang, J. Wang, Z. Wang, S. K. S. Yau, Z. Lin, L. Zhou, C. Ran, L. Xiao, C. Wu, and J. Schmidhuber, "MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework," in *ICLR'24*, Vienna, Austria, May 2024, pp. 23 247–23 275.

[2] W. Chen, Y. Su, J. Zuo, C. Yang, C. Yuan, C.-M. Chan, H. Yu, Y. Lu, Y.-H. Hung, C. Qian, Y. Qin, X. Cong, R. Xie, Z. Liu, M. Sun, and J. Zhou, "AgentVerse: Facilitating Multi-Agent Collaboration and Exploring Emergent Behaviors," in *ICLR'24*, Vienna, Austria, May 2024, pp. 20 094–20 136.

[3] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, J. Liu, A. H. Awadallah, R. W. White, D. Burger, and C. Wang, "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversations," in *COLM'24*, Toronto, Canada, Oct 2024.

[4] C. Qian, W. Liu, H. Liu, N. Chen, Y. Dang, J. Li, C. Yang, W. Chen, Y. Su, X. Cong, J. Xu, D. Li, Z. Liu, and M. Sun, "ChatDev: Communicative Agents for Software Development," in *ACL'24*, Bangkok, Thailand, Aug 2024, pp. 15 174–15 186.

[5] F. Romero, Q. Li, N. J. Yadwadkar, and C. Kozyrakis, "INFaaS: Automated Model-less Inference Serving," in *USENIX ATC'21*, Online, Jul 2021, pp. 397–411.

[6] J. R. Gunasekaran, C. S. Mishra, P. Thinakaran, B. Sharma, M. T. Kandemir, and C. R. Das, "Cocktail: A Multidimensional Optimization for Model Serving in Cloud," in *NSDI'22*, Renton, WA, USA, Apr 2022, pp. 1041–1057.

[7] X. Li, Q. Wang, W. Sha, Z. Li, and B. Zhang, "Efficient Computation Offloading for Mobile Edge Computing by Using Success-History based Adaptive DE," in *ICCEA'23*, Guangzhou, China, Apr 2023, pp. 124–127.

[8] Y. Zhang, R. Sun, Y. Chen, T. Pfister, R. Zhang, and S. O. Arı k, "Chain of Agents: Large Language Models Collaborating on Long-Context Tasks," in *NeurIPS'24*, Vancouver, BC, Canada, Dec 2024, pp. 132 208–132 237.

[9] Z. Liu, Y. Zhang, P. Li, Y. Liu, and D. Yang, "A Dynamic LLM-Powered Agent Network for Task-Oriented Agent Collaboration," in *COLM'24*, Philadelphia, PA, USA, Oct 2024.

[10] M. Zhuge, W. Wang, L. Kirsch, F. Faccio, D. Khizbullin, and J. Schmidhuber, "GPTSwarm: Language Agents as Optimizable Graphs," in *ICML'24*, Vienna, Austria, Jul 2024.

[11] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language Models Can Teach Themselves to Use Tools," in *NeurIPS'23*, New Orleans, LA, USA, Dec 2023, pp. 68 539–68 551.

[12] Y. Shen, K. Song, X. Tan, D. Li, W. Lu, and Y. Zhuang, "HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face," in *NeurIPS'23*, New Orleans, LA, USA, Dec 2023, pp. 38 154–38 180.

[13] M. Yu, T. Cao, W. Wang, and R. Chen, "Following the Data, Not the Function: Rethinking Function Orchestration in Serverless Computing," in *NSDI'23*, Boston, MA, USA, Apr 2023, pp. 1489–1504.

[14] K. Zhu, Z. Zhang, S. Zeadally, and F. Sun, "Learning to optimize workflow scheduling for an edge–cloud computing environment," *IEEE Transactions on Cloud Computing*, vol. 12, no. 3, pp. 897–912, 2024.

[15] A. Raza, N. Akhtar, V. Isahagian, I. Matta, and L. Huang, "Configuration and placement of serverless applications using statistical learning," *IEEE Transactions on Network and Service Management*, vol. 20, no. 2, pp. 1065–1077, 2023.

[16] Z. Xu, L. Zhou, W. Liang, Q. Xia, W. Xu, W. Ren, H. Ren, and P. Zhou, "Stateful serverless application placement in mec with function and state dependencies," *IEEE Transactions on Computers*, vol. 72, no. 9, pp. 2701–2716, 2023.

[17] Z. Tang, F. Mou, J. Lou, W. Jia, Y. Wu, and W. Zhao, "Joint resource overbooking and container scheduling in edge computing," *IEEE Transactions on Mobile Computing*, vol. 23, no. 12, pp. 10 903–10 917, 2024.

[18] J. Ruan, Z. Xu, Y. Peng, F. Ren, Z. Yu, X. Liang, J. Xiang, Y. Chen, B. Liu, C. Wu *et al.*, "AOrchestra: Automating Sub-Agent Creation for Agentic Orchestration," *arXiv preprint arXiv:2602.03786*, 2026.

[19] X. Wang, J. Ye, and J. C. Lui, "Decentralized Task Offloading in Edge Computing: A Multi-User Multi-Armed Bandit Approach," in *IEEE INFOCOM'22*. London, United Kingdom: IEEE, 2022, pp. 1199–1208.

[20] B. Han, V. Sciancalepore, Y. Xu, D. Feng, and H. D. Schotten, "Impatient Queuing for Intelligent Task Offloading in Multiaccess Edge Computing," *IEEE Transactions on Wireless Communications*, vol. 22, no. 1, pp. 59–72, 2023.