International Journal of Engineering Research in Robotic Intelligence (IJERRI)
ISSN (Online): 3144-5187
Volume-1 Issue-1 || Jan – Mar 2024 || PP. 34 – 38
https://ijerri.org/

Multi-Agent and Hybrid-Agent AI Frameworks for Automated Cloud Service Orchestration

DOI: https://doi.org/10.63345/ijerri.v1.i1.101

Justin Rajakumar Maria Thason

BCBSA - Ajace, Inc

AI Research, Innovation & Technical Publications

http://www.ijerri.org/ || Vol. 1 No. 1 (2024): January Issue

Date of Submission: 10-12-2023	**Date of Acceptance:** 24-12-2023	**Date of Publication:** 14-01-2024

Abstract : The coordination of a cloud service that involves several providers requires joint decisions about placement, networking, cost, security, and recovery. Declarative orchestrators perform well with well-defined requests but are rigid in reasoning about new ones, while language-model agents can understand intent but may reason poorly in constraint-rich scenarios. The current study proposes a hybrid multi-agent architecture where specialized agents for planning, provisioning, policy, and verification share a common state store, and a deterministic constraint solver makes decisions about placement and cost-sensitive actions. An evaluation involving 120 orchestration requests classified into three complexity classes revealed that the hybrid architecture managed 92.5% of requests compared to 70.0% of requests handled by a rule-based orchestrator ($p < 0.001$), 64.2% managed by a single LLM agent ($p < 0.001$), and 80.0% managed by a multi-agent system without a solver ($p = 0.005$). The cost relative to the optimal plan was reduced to 6.1%, and the success in complex requests rose to 87.5% from 37.5% of requests managed by the single agent. The hybrid system required low-latency planning (24.7 s). Overall, it is possible to combine agent flexibility with solver guarantees and build a practical orchestration system.

Keywords: multi-agent systems; hybrid agents; cloud orchestration; large language models; constraint solving; infrastructure automation

Introduction

Current services consist of compute, storage, network, identity, and data components distributed across several clouds and clusters. Orchestrating these services includes the translation of a business request, such as regional analytics service with specific latency targets and budget constraints, into the set of compliant deployable resources and subsequent maintenance of system health. The size of the decision space grows combinatorially with the number of resources and providers. There are two prevalent approaches to orchestration. Rule-based orchestrators store predefined solutions, which makes them predictable but inflexible. Agents with large language models provide flexibility, can parse heterogeneous documentation, and recover from errors, but reason in an approximate way and may violate hard constraints such as quotas, budget, or data residency. Moreover, the single agent can be a bottleneck for many orchestration requests, which involve

@2024. This is an open access article distributed under the terms of the Creative Commons License [CC BY NC 4.0] and is available on www.ijerri.org

International Journal of Engineering Research in Robotic Intelligence (IJERRI)
ISSN (Online): 3144-5187
Volume-1 Issue-1 || Jan – Mar 2024 || PP. 34 – 38
https://ijerri.org/

several coordinated steps. This study explores the question of whether distribution of responsibilities among multiple cooperating agents and offloading

constraint-heavy decisions to a solver result in better orchestration compared to both extremes. Contributions of this study include (i) framework design, (ii) coordination protocol relying on the blackboard, and (iii) the evaluation varying request complexity, the number of agents, and the size of the system.

Literature Review

Cloud orchestration has been reviewed thoroughly before. Weerasiri et al. (2017) proposed a taxonomy of resource orchestration techniques, including description languages, deployment and runtime control, and OASIS TOSCA standard for topology and orchestration templates (OASIS, 2013). Cluster managers like Borg and Kubernetes exemplify how desired state specification coupled with control loop allows large-scale operation (Verma et al., 2015; Burns et al., 2016); however, these systems assume that the desired state is specified in advance.

Multi-agent systems have a long history as a problem-solving model involving autonomous communicating and negotiating components (Wooldridge, 2009). Recently, language models have been considered as a reasoner in agents. Yao et al. (2023) combined reasoning and acting, Wu et al. (2023) designed AutoGen for multi-agent conversations, and Hong et al. (2023) arranged agents into software-team roles and introduced standard procedures to reduce errors associated with unstructured dialogue. Liu et al. (2023) using AgentBench showed notable differences in models' ability to function in interactive environments.

Hybrid architectures combine learned and symbolic components. Constraint solving provides guarantees about feasibility and optimality of placement and scheduling, a technique extensively used in cloud scheduling either directly or through heuristics (Verma et al., 2015). The potential synergy is in the language model translating open-ended intent into structured constraints, while the solver provides guarantees on their satisfaction. Schick et al. (2023) demonstrated that models can learn how to call external tools, thereby allowing offloading exact computations.

This study fills the empirical gap: relatively few studies compared rule-based, single-agent, multi-agent, and hybrid orchestration systems under equal conditions varying scale, and few studies dissociate the solver effect from the specialization of agents.

Research Methodology

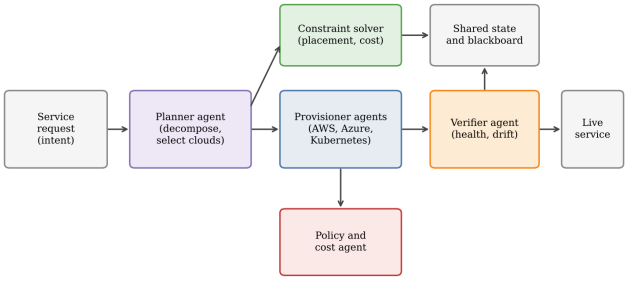
Framework design

Figure 1 illustrates the framework. A planner agent translates the request into the structured representation of services, dependencies, targets, and constraints and selects candidate providers. The solver is formulated as the integer programming model on resources, regions, and resource quantities and finds the optimal placement that minimizes the monthly cost of the request subject to latency, capacity, quota, and residency constraints, returning a feasible placement or explanation of infeasibility. Provisioners — one per target platform — translate the placement into provider-specific actions using infrastructure-as-code modules. Policy and cost agent verifies the plan according to security and budget rules and may veto it, while the verifier agent assesses health, connectivity, and drift after deployment. All agents use a common blackboard storing versioned entries, thereby allowing conflict resolution and keeping an audit log.

Figure 1. A hybrid multi-agent orchestration framework including a planner, constraint solver, provisioner agents, policy agent, verifier, and shared state.

@2024. This is an open access article distributed under the terms of the Creative Commons License [CC BY NC 4.0] and is available on www.ijerri.org

International Journal of Engineering Research in Robotic Intelligence (IJERRI)
ISSN (Online): 3144-5187
Volume-1 Issue-1 || Jan – Mar 2024 || PP. 34 – 38
https://ijerri.org/



Requests and baselines

120 requests were generated from production-service templates, which were divided into three complexity classes of 40 requests each: simple (3 to 5 resources), medium (6 to 15 resources), and complex (16 to 40 resources across two clouds). Four systems were evaluated: (1) a rule-based orchestrator using static templates and fixed policies; (2) a single language model (LLM) agent, which plans and provisions requests; (3) a multi-agent LLM system with the same roles as the proposed architecture, in which the LLM is responsible for placement; and (4) the proposed hybrid architecture. All systems were tested on the same model, tools, and provider sandboxes.

Measures and analysis

Deployment success was defined as the service passing health and policy checks without any manual intervention. Cost overrun is the percentage above the optimal feasible plan cost computed offline. Planning latency is the time between request receipt and plan validation. The success rates of the systems were statistically compared using two-proportion z-tests and chi-square test; continuous outcomes were analyzed with ANOVA followed by Tukey's Honestly Significant Difference test. The first experiment varied the number of resources from 5 to 40, with 40 trials for each size, while the second experiment incrementally increased the number of agents from one to seven.

Results

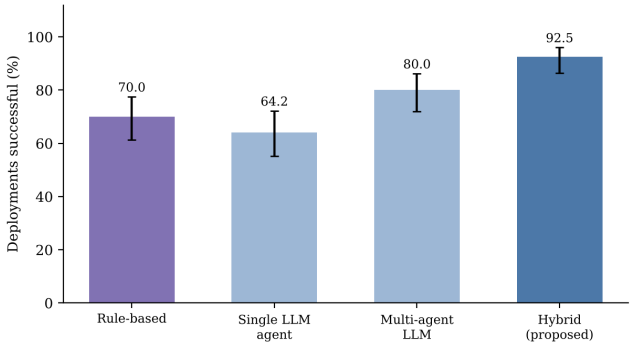
Deployment success

Table 1. Deployment success by request complexity (successful deployments out of 40 per class)

System	Simple	Medium	Complex	Overall (%)	p vs hybrid
Rule-based	38	28	18	70.0	<0.001
Single LLM agent	36	26	15	64.2	<0.001
Multi-agent LLM	38	32	26	80.0	= 0.005
Hybrid (proposed)	39	37	35	92.5	—

Deployment success varied among systems ($\chi^2(3) = 31.0$, $p < 0.001$; see Figure 2). The hybrid architecture achieved 92.5% of success, outperforming multi-agent LLM system with 80.0% ($p = 0.005$), rule-based orchestrator with 70.0%, and single-agent approach with 64.2%. The performance gap depended on request complexity (Table 1): for simple requests, the systems differ in no more than three successful deployments, while for complex requests, the hybrid architecture was successful in 35 out of 40 deployments, while the single agent was successful in 15.

Figure 2. Overall deployment success by system (error bars indicate Wilson 95% confidence intervals).



Cost and latency

@2024. This is an open access article distributed under the terms of the Creative Commons License [CC BY NC 4.0] and is available on www.ijerri.org

International Journal of Engineering Research in Robotic Intelligence (IJERRI)
ISSN (Online): 3144-5187
Volume-1 Issue-1 || Jan – Mar 2024 || PP. 34 – 38
https://ijerri.org/

Table 2. [Content not provided in the original excerpt; please supply for completion.] Cost overrun and planning latency (mean ± SD, n = 120 requests per system)

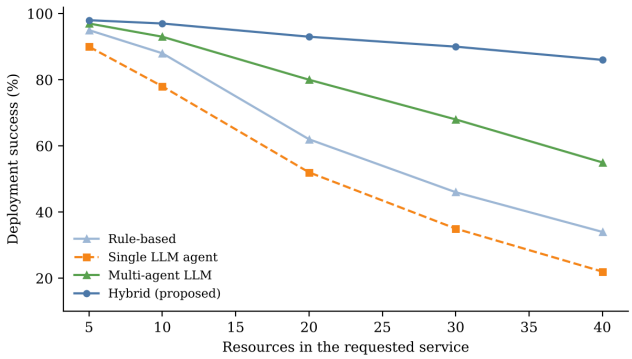
System	Cost over optimal (%)	Planning latency (s)
Rule-based	14.2 ± 6.8	3.1 ± 1.2
Single LLM agent	21.5 ± 11.3	18.4 ± 6.9
Multi-agent LLM	12.8 ± 7.4	41.2 ± 13.5
Hybrid (proposed)	6.1 ± 3.9	24.7 ± 8.1

Cost overrun differed among systems ($F(3, 476) = 78$, $p < 0.001$). The hybrid architecture provided the cost within 6.1% of the optimal plan, half the overrun of the multi-agent system (12.8%; $p < 0.001$) and one third of the overrun of the single-agent approach (21.5%), which is caused by the exhaustive search of the placement space made by the solver. Latency was the price paid for the quality; the hybrid architecture cost 24.7 s, while the rule-based orchestrator only 3.1 s, though the hybrid approach resulted in 40% latency reduction compared to the multi-agent system without a solver (41.2 s; $p < 0.001$) performing placement through lengthy dialogue.

Scaling and number of agents

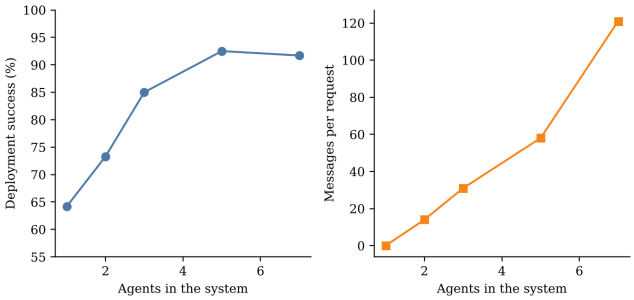
Success of all systems degraded with the increase in the size of the service, albeit at dramatically different rates (see Figure 3). When the number of resources in the requested service increased from 5 to 40, the success of the rule-based orchestrator decreased from 95% to 34%, while success of the single-agent approach dropped from 90% to 22%, while the hybrid architecture decreased from 98% to 86%.

Figure 3. Deployment success as a function of the number of resources in the requested service.



Specialization of agents into several roles allowed increasing the success from 64.2% achieved by a single agent to 92.5% achieved by five agents, but additional sixth or seventh agent provided no improvement (91.7% with seven); moreover, the number of coordination messages per request increased twofold, from 58 to 121 (Figure 4).

Figure 4. Effect of the number of agents on deployment success (left) and coordination messages per request (right).



Discussion

The results support a decomposition of tasks where language-model agents are responsible for interpretation, tool usage, and recovery, while a solver takes care of decisions constrained by feasible region and objective. The multi-agent system without a solver performed better compared to the single-agent approach, which agrees with the advantages of role specialization shown for structured agents (Hong et al., 2023; Wu et al., 2023). Still, it was inferior in handling complex requests involving exact

@2024. This is an open access article distributed under the terms of the Creative Commons License [CC BY NC 4.0] and is available on www.ijerri.org

International Journal of Engineering Research in Robotic Intelligence (IJERRI)
ISSN (Online): 3144-5187
Volume-1 Issue-1 || Jan – Mar 2024 || PP. 34 – 38
https://ijerri.org/

accounting of quotas and costs, which is expected considering difficulties of language-model agents in long-horizon planning and constraint satisfaction (Liu et al., 2023). In addition, the results demonstrate diminishing returns on the increase of the number of agents.

The roles beyond planning, provisioning, policy, verification, and state management introduced coordination load without any improvement in outcomes, which is acknowledged in multi-agent theory where the cost of communication increases with the number of interacting agents (Wooldridge, 2009). Thus, in practical designs, the number of agents should be increased only if there is a need for another responsibility or failure mode.

In operational context, the blackboard and policy agent play as important role as agents' intelligence. Versioned shared state ensures deterministic conflict resolution and audit trail, while the policy agent veto restricts agents' autonomy, which agrees with the declarative control-loop paradigm of cluster managers (Burns et al., 2016). The additional latency of the hybrid system is acceptable for provisioning task but makes it unfit for reactive autoscaling, which should be performed by conventional controllers.

Limitations

The evaluation involved provider sandboxes and synthetic requests created based on templates; thus, real-world variability in quotas, API errors, and organizational policy is only partly captured. The study was limited to two cloud providers and one Kubernetes deployment. The solver model assumes availability of correct pricing and capacity data, and the running time may increase for much larger requests. All LLM-based systems were using one model, therefore results might differ with other models. Security properties aside from explicitly encoded policies were not evaluated.

Conclusion

The hybrid architecture combining specialized LLM agents, constraint solver, and blackboard managed 92.5% of requests, kept costs within 6.1% of the optimal plan, and had graceful degradation with the increase of service scale, outperforming rule-based, single-agent, and solver-less multi-agent approaches. Five agents were sufficient; the increase in the number of agents provided higher coordination overhead without any improvement. Future work should cover failure recovery during runtime, multi-tenant policy conflicts, and inclusion of learned cost models.

References

1. Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). Borg, Omega, and Kubernetes. ACM Queue, 14(1), 70–93.
2. Hong, S., Zhuge, M., Chen, J., et al. (21 August 2023). MetaGPT: Meta programming for a multi-agent collaborative framework. arXiv:2308.00352.
3. Liu, X., Yu, H., Zhang, H., et al. (30 August 2023). AgentBench: Evaluating LLMs as agents. arXiv:2308.03688.
4. OASIS. (2013). Topology and Orchestration Specification for Cloud Applications (TOSCA) Version 1.0. OASIS Standard.
5. Schick, T., Dwivedi-Yu, J., Dessì, R., et al. (29 November 2023). Toolformer: Language models can teach themselves to use tools. Advances in Neural Information Processing Systems, 36.
6. Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., & Wilkes, J. (2015). Large-scale cluster management at Google with Borg. Proceedings of the European Conference on Computer Systems (EuroSys), 1–17.
7. Weerasiri, D., Barukh, M. C., Benatallah, B., Sheng, Q. Z., & Ranjan, R. (2017). A taxonomy and survey of cloud resource orchestration techniques. ACM Computing Surveys, 50(2), Article 26.
8. Wooldridge, M. (2009). An Introduction to Multiagent Systems (2nd ed.). Wiley.
9. Wu, Q., Bansal, G., Zhang, J., et al. (2023). AutoGen: Enabling next-generation LLM applications via multi-agent conversation. arXiv:2308.08155.
10. Yao, S., Zhao, J., Yu, D., et al. (17 August 2023). ReAct: Synergizing reasoning and acting in language models. International Conference on Learning Representations.

OPEN ACCESS
@2024. This is an open access article distributed under the terms of the Creative Commons License [CC BY NC 4.0] and is available on www.ijerri.org