Towards Agentic Performance Management (Extended Abstract)

Jingyuan Chen, Gongqi Huang, Amit Levy
Princeton University

Abstract

Agentic AI systems exhibit dynamic execution, evolving structures, and fluid inter-component dependencies, fundamentally challenging traditional performance management. We propose *Agentic Performance Management*, a modular and evolvable approach in which agents maintain counterfactual performance models and adapt online through coordination.

1 Introduction

Agentic AI systems introduce fundamentally new challenges for performance management. Modern agents are goal-oriented, self-evolving, and operate in open, non-deterministic environments. Individual agents are specialized to deliver distinct functions and are composed dynamically. They are assembled into pipelines whose execution structures change at runtime, where multiple interacting knobs affect performance, and where dependencies evolve as agents join or leave. This leads to *agentic* workloads with highly dynamic performance characteristics. Crucially, the set of agents relevant to any given agent's performance is itself fluid.

Such dynamism fundamentally stresses existing performance management methodologies. Traditional approaches assume fixed objectives and operate within largely static policy spaces, relying on predefined heuristics or control algorithms. When workload dynamics shift, these systems require manual recalibration of models and policies, resulting in brittle behavior. ML-based methods offer limited adaptability but still rely on relatively stable feature spaces and incur significant retraining costs under distribution shift.

A key underlying issue is the assumption of a stable system structure. Consequently, traditional methods employ monolithic, imperative control policies without explicit performance models that support counterfactual reasoning. This limits their ability to attribute root causes across components and to adapt effectively as system structure and interactions evolve.

We argue that performance management must itself become *agentic*. Rather than relying solely on monolithic controllers or static policies, each agent should maintain evolving performance objectives and models, and decide how to adapt its behavior in interaction with other agents. Because an agent's performance is coupled to both shared resources and other agents in the pipeline, such adaptation cannot be purely local. When an agent predicts that its goals will become infeasible, it should be able to coordinate with other relevant agents. In this view, performance management becomes a distributed, cooperative process spanning both the resource management layer and the agents themselves. This design enables modular, model-based, and evolvable performance management.

A key enabler of such coordination is *performance interfaces* that expose not only observed metrics but also support faithful counterfactual queries about performance behavior under changing demand and allocation [2]. Such interfaces should allow agents to reliably translate "my SLO is violated" into actionable requests (e.g., "which upstream rate reductions, peer yields, or resource reallocations would restore feasibility?"). We propose to realize this interface via an abstract simulation layer that combines analytical models (e.g., queueing models), ML-based performance predictors, and specialized approximate simulators (e.g., sampling-based scheduler simulators with explicit confidence interval guarantees), providing a flexible, extensible, and lightweight representation of performance dynamics.

2 Proposed Architecture

2.1 System Components

Each agent owns its performance objectives (e.g., latency, throughput, cost–quality tradeoffs) and controls performance-related policies such as batching, model selection, and request shaping. Subsystems that provide resource management (e.g., schedulers, rate limiters, garbage collectors, and autoscalers) may also be modeled as agents or, alternatively, expose performance models to application agents.

2.2 The Performance Interface Layer

We propose exposing a performance interface that enables actionable counterfactual reasoning via an abstract simulation layer that combines analytical models, ML-based predictors, and specialized approximate simulators, yielding an operational model of end-to-end performance dynamics. Each agent maintains local state sufficient to drive these queries, including summaries of the environment, current policies, goals, and a performance model:

$$M : I \times P \to O \times E$$

which estimates

- O: Observable performance metrics of the agent itself or other agents (e.g., average latencies/throughput);
- E: An agent's effect on the environment, such as rate of outgoing loads on downstream agents (i.e, at what rate the agent is sending requests to other agents).

under different:

- I: Environmental inputs, such as rate of incoming load from upstream agents;
- P: The agent's own control policies (e.g., scheduling or caching policies), and the set of policies of other relevant agents. For example, an agent's request execution times may depend on the scheduling policies of downstream agents; therefore, those policies must be incorporated.

This interface allows agents to reason about the impact of their own and other agents' policies for proactive coordination.

2.3 The Coordination Layer

When an agent predicts that its objectives are (or will become) infeasible under current conditions, it does not merely adjust its own local policy. Instead, it can initiate coordination actions across three directions:

1. **Upstream coordination**: request upstream agents (that send demands to the agent) to reduce, redirect, or delay demands. E.g., rate limiting, batching more aggressively, reusing cached results at the cost of reduced quality, etc.
2. **Downstream coordination**: request downstream agents (that the agent sends demands to) to adjust their policies. E.g., resource reallocation, priority changes, etc..
3. **Peer coordination**: request agents sharing the same dependent agents to temporarily yield capacity, lower priority, or reduce their own rates.

Recipients of a request evaluate it against their own objectives and constraints and choose to either:

1. **Accept** the request and commit to the proposed change.
2. **Deny** the request and provide a feedback. E.g., a reason of rejection such as the existence of stringent deadlines at the recipient's side.
3. **Initiate a new coordination phase** by sending further requests to related agents to seek alternatives to satisfy the current request. For example, to satisfy a rate limiting request of a downstream agent, the current agent may further ask its upstream agents for rate limiting.

2.4 The Agentic Performance Management State Machine

Putting these pieces together, each agent participates in the following control loop:

1. **Detection.** The agent monitors its current performance and evaluates whether its objectives are violated.
2. **Plan Generation.** If a violation is detected, the agent generates candidate intervention plans (local or coordination).
3. **Plan Validation.** The agent queries the performance interface to estimate the impact of these interventions and filters or prioritizes them accordingly.
4. **Coordination/Decision.** The agent issues selected requests to the relevant agents. Recipients evaluate these requests against their own objectives and performance models and decide to accept, reject, or further coordinate.

Importantly, this loop does not assume global optimality or convergence guarantees. Its purpose is to make tradeoffs explicit, explainable, and actionable under changing objectives and system dynamics.

3 Challenges and Future Work

We sketch key challenges in realizing this architecture and outline potential directions.

A first challenge is maintaining performance models that are both sufficiently accurate to guide decisions and sufficiently extensible to accommodate changing workloads, policies, and system structures. Rather than relying on a single modeling technique, a promising direction is to combine analytical models, ML-based predictors, and specialized simulators, following hybrid approaches such as [3]. While this does not eliminate tradeoffs, it provides a practical balance between fidelity, expressiveness, and cost.

The second challenge is generating effective coordination plans. Although the architecture is agnostic to decision-making techniques, in principle, the agents could use classical optimization, control-theoretic methods, or rule-based policies to generate and evaluate negotiation requests. In practice, however, such approaches rely on fixed objectives and hand-designed strategy spaces, which reintroduce the brittleness agentic performance management seeks to avoid.

We argue that LLM agents themselves offer a compelling alternative (inspired by [1]): they can reason about tradeoffs, coordination, and adaptation under incomplete models and partial observability without committing to a narrowly predefined strategy space. To avoid the recursive "who manages the managers" problem, we decouple proposal generation from execution: stable, conventional control mechanisms remain responsible for enforcement and safety.

Importantly, LLMs are should not be placed on the critical path. Controllers should not issue costly queries to LLMs; instead, LLMs are used only in the plan generation and validation phase, where their flexibility can be leveraged without impacting runtime stability.

A third challenge is the cost of evaluating performance models during coordination. Many existing simulators are designed for offline analysis and become bottlenecks in online settings. A natural direction is to incorporate approximate computing and multi-fidelity modeling, allowing agents to trade precision for speed. For example, instead of returning a point estimate (e.g., "average latency is X"), the model may return a coarse but efficient summary (e.g., "the average latency lies in $[L, H]$ with confidence α"). Such approximations enable scalable, anytime decision-making while reserving higher-fidelity evaluation when needed.

References

[1] Audrey Cheng, Shu Liu, Melissa Pan, Zhifei Li, Shubham Agarwal, Mert Cemri, Bowen Wang, Alexander Krentsel, Tian Xia, Jongseok Park, and others. 2025. Let the Barbarians In: How AI Can Accelerate Systems Performance Research. *arXiv preprint arXiv:2512.14806* (2025).

[2] Rishabh Iyer, Katerina Argyraki, and George Candea. 2024. Automatically Reasoning About How Systems Code Uses the {CPU} Cache. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, 2024. 581–598.

[3] Arash Nasr-Esfahany, Mohammad Alizadeh, Victor Lee, Hanna Alam, Brett W Coon, David Culler, Vidushi Dadu, Martin Dixon, Henry M Levy, Santosh Pandey, and others. 2025. Concorde: Fast and Accurate CPU Performance Modeling with Compositional Analytical-ML Fusion. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, 2025. 1480–1494.