A Case for Simulation-Driven Resilience in Agentic Data Systems

Aleksey Charapko
University of New Hampshire
Durham, NH, USA
Aleksey.Charapko@unh.edu

Murat Demirbas
MongoDB
New York, NY, USA
Murat.Demirbas@mongodb.com

Akshat Vig
MongoDB
New York, NY, USA
Akshat.Vig@mongodb.com

ABSTRACT

AI agents are replacing humans as the primary clients of modern data systems, and are bringing with them a qualitative shift in workload characteristics. Agents retry aggressively with mutated queries, spawn bursty parallel sub-tasks, and hold transactions open while awaiting external LLM responses. These behaviors systematically violate assumptions baked into every layer of modern data systems (from execution control and admission policies to caching layers and concurrency control) and create fertile ground for *metastable failures*: self-sustaining states of degraded throughput that persist even after the triggering overload subsides. We argue that the community needs a simulation-driven methodology to systematically discover and prevent agent-induced failures across the data system stack *before* facing production incidents. We demonstrate this approach on Execution Control Systems (ECS), where we show how discrete-event simulation with MESSI reveals failure modes invisible to benchmarks, and outline a research agenda extending the methodology to other data system services and components.

CCS CONCEPTS

• **Information systems** → **Database management system engines**; • **Computer systems organization** → *Dependable and fault-tolerant systems and networks.*

KEYWORDS

metastability, agentic workloads, execution control, discrete-event simulation, data systems, overload control

ACM Reference Format:
Aleksey Charapko, Murat Demirbas, and Akshat Vig. 2026. A Case for Simulation-Driven Resilience in Agentic Data Systems. In *Proceedings of SAO Workshop at CAIS'26*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

AI agents are on the verge of replacing humans as the primary clients of data systems. They autonomously compose multi-step workflows that query databases, analyze results, and iteratively refine their approach [11, 15]. Solutions like the Model Context Protocol (MCP) [1] are standardizing how agents connect to data sources and accelerating the transition from human-driven to agent-driven database traffic. Unfortunately, data systems are not ready for this phase shift transition.

Every layer of a modern data system embeds assumptions about how clients behave: execution control assumes stationary arrivals, caching assumes temporal locality, and concurrency control assumes bounded hold times. Agents violate all of these simultaneously. Adapting data systems for agentic workloads is a broad challenge: recent work has proposed redesigning query interfaces, transaction semantics, and storage layers to natively support agentic speculation [8]. These are important directions, but none of them matter if the system cannot stay up.

Thus, we focus our deepest analysis on the **Execution Control System (ECS)**, the component mediating resource contention at the backend through queues, concurrency limits, and scheduling decisions. Concretely, the ECS sits between accepted client requests and the execution engine: it manages priority enforcement and compute resource allocation among executing requests. Typically, ECS uses a bounded pool of execution *tickets* to manage concurrency and resource distribution. ECS places admitted requests into one or more queues with different ticket allocation based on perceived priority and dispatches them to worker threads or storage I/O slots. The queues and per-queue ticket budgets allow prioritizing some requests over others when resources are constrained. To cope with heterogeneous workloads, production ECS designs layer on priority- or runtime-aware multi-queue scheduling [7], timeout-based load shedding for stalled work, and backpressure signals that throttle upstream admission when downstream resources saturate [3]. The ECS is the critical first domino: when it fails, every other subsystem degrades. But the threat does not stop at ECS – the same agent behaviors that destabilize execution control also poison caching layers, buffer pools, and lock managers, and the simulation methodology we develop here generalizes to each of them.

Just like many other components of data systems, today's ECS designs are built on assumptions that agents systematically violate: stationary arrival processes, bounded service-time distributions, graceful client backoff, and independent requests. Although agents interact with data systems through driver libraries, connection pools, and middleware that impose some client-side discipline, they still exhibit characteristics that are qualitatively different from traditional clients:

- **Aggressive retries with mutation.** When a query fails, an agent's reasoning loop treats this as a problem to solve, retrying the query, rewriting it, or spawning alternatives. Even with standard retry middleware, the agent's tendency to *alter* the query on each attempt produces a stream of novel requests that evade per-query deduplication.
- **Non-deterministic query generation.** Agents generate SQL or API calls via LLM inference. Two invocations of the same

SAO Workshop at CAIS'26, May 26, 2026, San Jose, CA, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

logical task can produce radically different queries with different resource profiles, making *a priori* task classification nearly impossible.

- **Bursty parallelism.** Agentic frameworks routinely "fan out" work to sub-agents. A single user request can spawn dozens of concurrent database operations in milliseconds, then go idle.
- **External blocking.** Agents may pause mid-transaction to invoke external LLM APIs, dispatch sub-agents, or call other tools, holding database resources (tickets, connections, locks) across reasoning steps. Effective hold time becomes query-execution time *plus* inference latency multiplied by the number of reasoning steps spent inside the transaction, breaking pool-sizing and ticket-budget assumptions calibrated for query execution alone.

These characteristics (documented in early empirical studies of agentic database access [8]) combine to produce sudden bursts of unpredictable, resource-intensive load that we call *agentic spikes*. Liu et al. [8] report that at Neon, agents created 20× more branches and performed 50× more rollbacks compared to humans, confirming that the volume and volatility of agent-generated operations far exceeds what human-paced scaling heuristics anticipate. Agentic spikes are also qualitatively different from traditional load spikes. They are discontinuous, self-reinforcing, and correlated through the agent's reasoning loop. This combination is what pushes ECS overloads from transient spikes that drain after the trigger subsides into *metastable collapse*: a self-sustaining degraded regime in which the very mechanisms designed to protect the system (retries, load shedding, queue management) close feedback loops with the agent and keep the system stuck [5].

In sum, data systems were designed for a world where clients behave predictably, and agents do not. We argue that agentic workloads constitute a *phase change* in database workload characteristics, a qualitative shift that invalidates assumptions embedded across the data system stack. Patchwork fixes will not suffice. We propose *simulation-driven resilience*: systematic exploration of the agent-database boundary using discrete-event simulation to discover and prevent these failures before production incidents force reactive fixes. We demonstrate this approach on ECS (speculation-aware scheduling and admission control co-design) and outline how the same simulation framework can be applied to caching, buffer management, and more. Finally, we propose empirically-grounded workload generators built on a *stateful frustration model* that captures how agents escalate from well-behaved clients to adversarial cache-busters.

2 AGENT-INDUCED METASTABILITY

Metastability is a failure mode in which a distributed system becomes trapped in a self-sustaining state of degraded performance: unlike transient overload, metastable failures persist even after the initial trigger is removed, because resilience mechanisms (retries, queues, timeouts) turn inward and sustain the degraded state [5]. We argue that AI agents amplify every known metastability trigger and introduce entirely new feedback loops. Figure 1 illustrates the core loops that arise when agentic clients interact with an ECS.

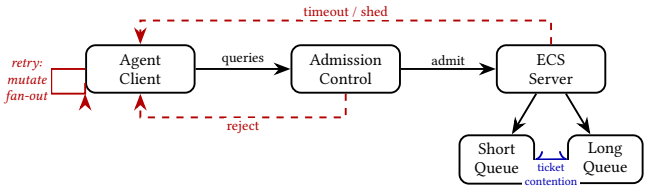

Figure 1: Agent-ECS feedback loops. Solid arrows show the request path. Dashed red arrows show the feedback that drives metastability: rejected or timed-out requests return to the agent, which retries with mutated queries or parallel fan-out, amplifying load. Ticket contention between queues (blue) creates an internal feedback loop within the ECS.

2.1 The Agentic Retry Storm

Traditional clients implement well-defined retry policies (exponential backoff, jitter, circuit breakers), and human users provide natural backpressure: they wait, switch tasks, or give up. Agents behave differently even when equipped with standard retry middleware. Agents not only retry *faster* (rate limiters can constrain that) but also retry *differently each time*. An LLM agent encountering a timeout may rewrite the query, break it into smaller queries, or escalate to a more expensive operation. Each retry appears as a novel request with an unpredictable resource profile invisible to deduplication or per-query throttling.

2.2 The Amplified Convoy Effect

ECS designs use multi-queue architectures that classify tasks by *elapsed wall-clock time*: a task exceeding a threshold (e.g., 10 ms) is demoted from the high-priority short-task queue to the lower-priority long-task queue. Agentic workloads may poison this classification.

Agents that call external LLM APIs mid-transaction hold database tickets for seconds while awaiting inference results, even though the actual database work takes only microseconds. The ECS may classify these as long-running tasks despite their relatively low CPU and IO overhead. Unfortunately, these misclassified tasks may hold locks on data needed by other operations, increasing overall data conflicts and potentially forcing other short tasks to wait longer and be misclassified as long tasks, creating a feedback loop. ECS may also load-shed long tasks to save resources; creating too many artificial long tasks increases the error rate observed by the agents, contributing to the agentic retry storms.

2.3 Novel Feedback Loops

Agentic workloads also create new feedback loops.

Retry–queue–ticket exhaustion spiral. Agent retries flood the admission queue. Queue growth exhausts the short-task ticket pool. New requests are either dropped or demoted to the long-task queue. Agents interpret drops as errors and retry *more aggressively*, closing the loop.

Query mutation–classification confusion. An agent's retry loop rewrites a failed short query as a complex join or aggregation. The ECS correctly classifies this as a long task and demotes it. The agent, receiving a timeout on the demoted query, rewrites it *again*.

The ECS is now tracking a stream of "independent" requests that are actually a single agent's frustrated retry loop, each with a different resource profile.

Multi-agent lock contention cascade. When multiple agents collaborate on shared state, they generate non-deterministic lock acquisition orders. This reduces throughput and intensifies contention through agents spawning additional parallel attempts.

The feedback loops described above represent a fundamental mismatch between the world model that current ECS designs are built for (stationary arrivals, bounded service times, graceful backoff, independent requests [12, 14]) and the threat model that agentic workloads present.

3 SIMULATING THE AGENT-DATABASE BOUNDARY

The complexity of agent-database interactions (with feedback loops, non-stationary arrivals, and correlated retry behavior) exceeds what analytical queuing models can capture. Traditional approaches like Markov chains or queueing theory assume stationary processes and independent arrivals [4]. We need tools that can model the *dynamic* interplay between agent behavior and system state.

3.1 MESSI: Discrete-Event Simulation for Metastability

In our preliminary work, we developed MESSI (MEtaStability SImulator) [2], a discrete-event simulation framework designed to explore metastability dynamics in distributed systems. MESSI models systems as directed graphs where **Logic Nodes** implement routing and state policies, and **Processors** simulate temporal resource constraints (queues, I/O delays, network latency). Individual work items (**QItems**) carry state as they traverse the graph. Leveraging this graph abstraction enables MESSI to model and investigate the metastability dynamics of any service, or composition of services, in a modern distributed data system stack.

3.2 What Simulation Reveals That Benchmarks Cannot

Static benchmarks like TPC measure steady-state throughput under fixed workload mixes, but metastable failures are *transient* phenomena that emerge from specific event sequences and may never appear in steady-state measurement. In our preliminary ECS work [2], MESSI's tick-by-tick simulation revealed that two independently reasonable policies, per-queue ticket probing for short and long queues, create a metastable feedback loop when composed: the long queue's ticket growth steals CPU from short tasks, forcing the short queue to escalate in turn, until both hit their limits. This failure mode is invisible to steady-state analysis.

Even when an agentic benchmark does provoke a collapse on a real system, the experiment yields throughput and latency curves but does not provide a *causal explanation*. The queue depths, ticket histories, and retry trees that drive metastability are not exposed, and the instrumentation needed to see them perturbs the very dynamics under study. Every candidate fix (a different ticket policy, shedding heuristic, or admission strategy) further demands rebuilding and redeploying the production stack, making it impractical to sweep design alternatives. MESSI inverts this calculus. It exposes the full internal state of system components at every simulation tick, and enables root-cause analysis. Experiments are deterministic and replayable, so a metastable trigger discovered once can be re-run against alternate ECS designs to see which survive. The simulation environment enables users to test hypotheses, compare alternatives, and converge on a robust design before any production change is committed.

Agentic workloads bring further challenges for observability as the space of possible triggering sequences is vastly larger: an agent's retry strategy, query mutation pattern, and parallelism decisions interact with queue bounds, ticket policies, cache eviction rules, and lock scheduling heuristics. Systematically exploring this space requires the controlled repeatable experimentation that only simulation provides.

3.3 Modeling Agentic Clients

Simulation is only as good as its workload model, but here we face an empirical gap: no equivalent of TPC exists for agentic workloads. Early characterization [8] reveals key properties (retry intensity, query mutation, fan-out, external blocking) but these have not been codified into reproducible generators. To close this gap, we propose extending MESSI with a **Simulated Agent Client**: a Logic Node that emits synthetic agentic traffic exhibiting the four characteristics identified in Section 2.

The client is built around a *stateful frustration model*. It maintains a frustration index that increments on each timeout, rejection, or queue demotion and resets on success. As frustration rises, the agent escalates: at low levels it retries with backoff; at moderate levels it *rewrites* the query, drawing from a heavier service-time distribution and shifting from point lookups toward broader scans that randomize its working set and break spatial locality; at high levels it *fans out* into parallel QItems across non-overlapping data partitions. Each emitted QItem carries a trace ID linking it to the originating task, so MESSI can reconstruct the full retry tree and expose the cross-subsystem feedback loops that drive metastable collapse. Frustration thresholds, fan-out degree, service-time distributions, and external-blocking probability are all configuration parameters, so the same model spans the spectrum from well-behaved clients to adversarial cache-busters.

To ground the synthetic models in real agent behavior, we propose controlled studies in which modern LLM agents are given open-ended analytical tasks against complex reference datasets, with execution paths continuously traced. Measuring the semantic distance of query mutations, the frequency of fan-outs, and the duration of external LLM pauses will yield the parameters needed to calibrate the generator.

Finally, to validate our simulation results with that of real systems, we propose a *closed-loop validation* methodology: use MESSI to sweep the parameter space (frustration levels, fan-out degrees, ticket limits), identify configurations that trigger metastable collapse, and then replay those exact workloads against instrumented deployments of production databases such as MongoDB. By correlating simulation predictions with production telemetry (buffer pool eviction rates, lock wait queue depths, disk I/O spikes, throughput degradation), we aim to validate the model's fidelity and give

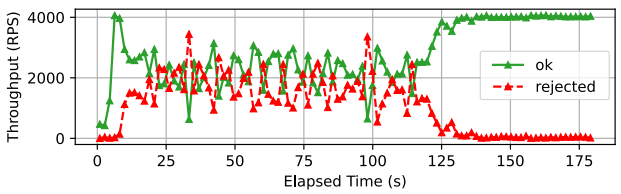

Figure 2: Admission Control and ECS can have destructive interference.

engineers the concrete reproducible evidence of agent-induced metastability needed to justify architectural changes.

4 WHAT CAN SIMULATION ATTACK?

We now outline a research agenda that applies simulation-driven resilience across the data system stack: first to ECS internals (where we have preliminary results), then to deeper subsystems.

4.1 ECS for Agentic Speculation

Liu et al. define *agentic speculation* [8] as agents issuing high-throughput, exploratory queries (many of them redundant) to ground themselves before formulating solutions. Only 10–20% of sub-plans are unique across parallel attempts, and queries progress through distinct phases from coarse metadata exploration to targeted solution formulation. This two-phase pattern creates a distinctive ECS challenge: exploratory queries consume all short-task tickets, and when the agent transitions to solution formulation, those heavier queries get classified as long tasks and starved. MESSI can model this by implementing agent Logic Nodes that transition between exploration and formulation modes, enabling systematic study of whether ECS policies should differentiate between phases and what signals could detect the transition.

4.2 Admission Control and ECS Co-Design

In our preliminary work [2], we found a surprising result: combining admission control with ECS can cause *destructive interference*, where admission control drops short and long tasks indiscriminately, reducing short-task goodput below what the ECS alone would achieve. We illustrate this in Figure 2, where the workload spikes at 5 seconds of elapsed time, causing immediate activation of Admission Control even though the ECS would have managed the load just fine after a short adjustment. However, with admission control dropping work, ECS cannot make the necessary adjustments to resource allocations and priorities, leaving the system stuck in reduced-goodput mode until the workload subsides. Under agentic workloads, this interaction becomes far more complex.

Admission control operates at the network edge and cannot distinguish an agent's first attempt from its fifth retry, nor whether a rejection will cause the agent to back off or escalate. MESSI's composable graph architecture can model both admission control and ECS as interacting components in the same simulation graph, enabling researchers to explore agent-aware admission policies and evaluate whether such policies improve or worsen metastability risk when composed with the ECS's internal queue management.

4.3 Beyond Execution Control: Caching and Concurrency

Buffer pool and caching layers. Frustrated agents naturally behave as cache adversaries. When an agent's initial query fails, its mutated retries abandon hot well-optimized data paths in favor of cold data subsets. This working-set randomization causes buffer pool thrashing, rapidly evicting useful pages and driving up disk I/O. By modeling the buffer pool as a MESSI Processor with LRU or clock-based eviction, we can test agent-aware cache isolation policies. For instance, we can pin stable application pages while allowing exploratory agentic queries to compete only within a bounded eviction budget.

Lock management and concurrency control. Agents generate queries non-deterministically, causing unpredictable lock acquisition across concurrent sub-tasks. Combined with long external blocking pauses (agents waiting on LLM inference while holding locks), this creates fertile ground for cascading lock contention and transaction starvation. Modeling the lock manager within MESSI enables us to observe how these non-stationary patterns interact with deadlock detection heuristics and explore whether agent-aware lock scheduling can prevent contention cascades.

5 RELATED WORK

Metastability. Huang et al. [5] established metastability as a first-class failure mode in distributed systems; Isaacs et al. [6] extended this with modeling and simulation approaches. Habib et al. [4] apply queuing-based analysis to predict metastable failures in replicated storage. Our work builds on these foundations but focuses on a new trigger: agentic workloads.

ECS and overload control. SEDA [13] introduced staged, event-driven admission control; Shenango [9] and Shinjuku [7] achieve low tail latency through kernel-bypass scheduling. Breakwater [3] applies credit-based admission for RPCs. These systems assume well-behaved clients with predictable backoff, which agents violate.

Eudoxia [10] presents a deterministic simulator designed to model the scheduling of data workloads as cloud functions within data lakehouses. It provides a low-cost experimentally driven model of the underlying system, and enables developers to safely evaluate custom scheduling algorithms across varying workload priorities.

Agent-first data systems. Liu et al. [8] propose an agent-first architecture covering query interfaces, probe optimization, and storage. Our work is complementary: we focus on the resilience and resource-management layers that underlie their proposed architecture, and advocate simulation-driven methodology for systematically discovering agent-induced failures across the stack.

6 CONCLUSION

Agents are not faster humans, they are a new threat model for data systems. Their behavior violates assumptions across the data system stack and turns protective mechanisms into amplifiers of failure. We have argued that simulation-driven resilience is how the community should respond: use discrete-event simulation to find these failures before production does. We demonstrated the approach on ECS, argued that it extends to caching and concurrency control, and outlined the workload models and validation methodology needed to make progress.

REFERENCES

[1] Anthropic. 2024. Model Context Protocol. https://modelcontextprotocol.io. Open standard for connecting AI agents to data sources and tools.

[2] Aleksey Charapko, Murat Demirbas, Matt Broadstone, Daniel Gomez Ferro, and Akshat Vig. 2026. Towards Designing an Execution Control System with Metastability Resilience. (2026). To appear at IEEE International Conference on Computer Communications and Networks.

[3] Inho Cho, Ahmed Saeed, Joshua Fried, Seo Jin, and Adam Belay. 2020. Overload Control for μs-scale RPCs with Breakwater. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 299–314.

[4] Farzad Habib, Tania Lorido-Botran, Ahmad Showail, Daniel C Sturman, and Faisal Nawab. 2024. Msf-model: Queuing-Based Analysis and Prediction of Metastable Failures in Replicated Storage Systems. In *2024 43rd International Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 12–22.

[5] Lexiang Huang, Matthew Magnusson, Abishek Bangalore Muralikrishna, Salman Estyak, Rebecca Isaacs, Abutalib Aghayev, Timothy Zhu, and Aleksey Charapko. 2022. Metastable Failures in the Wild. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 73–90.

[6] Rebecca Isaacs, Peter Alvaro, Rupak Majumdar, Kiran Kumar, Muniswamy Reddy, Mahmoud Salamati, and Sadegh Soudjani. 2025. Analyzing Metastable Failures. In *Proceedings of the 2025 Workshop on Hot Topics in Operating Systems*. 172–178.

[7] Kostis Kaffes, Timothy Chong, Jack Tigar Humphries, Adam Belay, David Mazières, and Christos Kozyrakis. 2019. Shinjuku: Preemptive Scheduling for μsecond-Scale Tail Latency. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 345–360.

[8] Shu Liu, Soujanya Ponnapalli, Shreya Shankar, Sepanta Zeighami, Alan Zhu, Shubham Agarwal, Ruiqi Chen, Samion Suwito, Shuo Yuan, Ion Stoica, Matei Zaharia, Alvin Cheung, Natacha Crooks, Joseph E. Gonzalez, and Aditya G. Parameswaran. 2025. Supporting Our AI Overlords: Redesigning Data Systems to be Agent-First. *arXiv preprint arXiv:2509.00997v2* (2025).

[9] Amy Ousterhout, Joshua Fried, Jonathan Behrens, Adam Belay, and Hari Balakrishnan. 2019. Shenango: Achieving High CPU Efficiency for Latency-Sensitive Datacenter Workloads. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 361–378.

[10] Tapan Srivastava, Jacopo Tagliabue, and Ciro Greco. 2025. Eudoxia: a FaaS scheduling simulator for the composable lakehouse. *Proceedings of Workshops at the 51th International Conference on Very Large Data Bases (VLDB)* (2025).

[11] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024. A Survey on Large Language Model Based Autonomous Agents. *Frontiers of Computer Science* 18, 6 (2024), 186345.

[12] Matt Welsh and David Culler. 2003. Adaptive Overload Control for Busy Internet Servers. In *4th USENIX Symposium on Internet Technologies and Systems (USITS 03)*.

[13] Matt Welsh, David Culler, and Eric Brewer. 2001. SEDA: An Architecture for Well-Conditioned, Scalable Internet Services. In *ACM SIGOPS Operating Systems Review*, Vol. 35. 230–243.

[14] David Yanacek. 2019. Using Load Shedding to Avoid Overload. AWS Builders Library.

[15] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. *arXiv preprint arXiv:2210.03629* (2023).