ToolFuse: Provenance-Aware In-Flight Coalescing of Redundant Agent Tool Calls

Izzet Yildirim
Illinois Institute of Technology
Chicago, IL, USA
iyildirim@hawk.illinoistech.edu

Xian-He Sun
Illinois Institute of Technology
Chicago, IL, USA
sun@illinoistech.edu

Anthony Kougkas
Illinois Institute of Technology
Chicago, IL, USA
akougkas@illinoistech.edu

***Abstract*—Scientific workflows increasingly run as fleets of LLM agents analysing immutable datasets. Independent agents can issue equivalent expensive read-only tool calls simultaneously, before a completed-result cache has anything to reuse. ToolFuse addresses this cold-wave gap by coalescing active calls at a gateway between agents and tool handlers. Its contribution is a provenance-aware execution identity that establishes equivalence from versioned dependencies and execution context, rather than arguments alone. Under contention on an exclusive HPC node, 16 equivalent requests become one execution, reducing CPU time by 11.79× and makespan by 2.36×. Peak memory growth falls from 6.44 GiB to 0.53 GiB. Trace replay shows that sharing depends on call duration and arrival timing. For cheap calls, coalescing remains inactive with negligible overhead.**

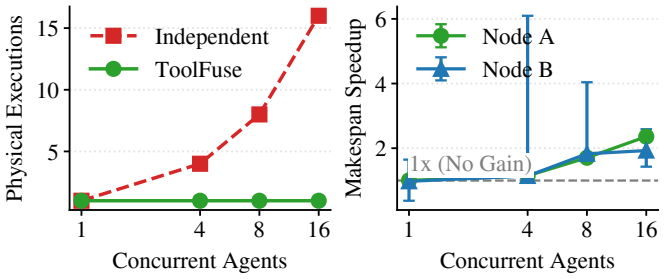


Fig. 1. Executions stay at one as agents scale. The makespan gain requires contention. Nodes reported separately, with 95% CIs.

I. Introduction

Scientific data analysis increasingly runs as fleets of LLM agents that inspect datasets, write code, and revise from observations. Isolated workspaces protect agent state, but agents working the same task reach immutable datasets independently. Without coordination at the tool boundary, they can execute identical work redundantly.

Two recorded corpora establish the duplication, each capturing every tool call on scientific data tasks. The first contains 7,134 calls across 625 sessions [1]. Of its 2,580 read-like calls, 2,485 repeat. The second records 127 agents on ten tasks, with call timestamps and durations that let us examine overlap under co-scheduling. Equivalent cross-agent calls cluster at a median gap of one second, though sessions span hours.

Suppressing duplicate work is not new. Go's `singleflight` [2] suppresses duplicate calls but keys on a caller-supplied string. Parsl [3] and Pegasus [4] memoize completed scientific work, and Parsl documents that concurrent identical cold calls still launch multiple instances [5]. Agent-focused caches such as ToolCacheAgent [6] reuse completed tool results. Completed-result reuse misses the first concurrent wave. Singleflight addresses that wave, but leaves the caller responsible for defining safe equivalence.

ToolFuse addresses this gap with a provenance-aware execution identity that establishes equivalence at the scientific tool boundary. In-flight suppression itself is standard. Our contribution is the identity and a measurement of when sharing pays: overlap increases as service time approaches the typical gap between equivalent calls.

II. Method

ToolFuse sits between agents and tool handlers, coalescing equivalent active calls. It admits only calls declared read-only, idempotent, deterministic, and closed-world, then hashes canonical arguments with versioned data dependencies, tool contract, implementation, environment, authorization scope, freshness epoch, and relevant hidden state into an execution key. The first request for an active key becomes the producer. Equivalent concurrent requests await its shielded task and receive the same immutable result. The key is deleted on success or failure, so later calls execute again. No completed result is retained, avoiding cache staleness, eviction, and storage costs. The design prefers false misses over unsafe sharing and is agnostic to the agent runtime.

We evaluate on the Ares cluster. A controlled campaign on exclusive 40-CPU nodes places the gateway before a real Parquet handler over a deterministic 262 MB, 7.3 M-row dataset. Each agent is a separate process, and no model sits on the timed path. We randomise paired conditions over ten repetitions at 1, 4, 8, and 16 agents. Discovery-call replay from the first corpus compares four reuse policies at natural and injected service times. A timing model uses the second corpus's recorded offsets and durations, followed by real execution of five tasks.

III. Results

At 16 agents under saturating concurrency, 16 equivalent requests become one physical execution with identical response digests, cutting CPU time 11.79× and peak memory growth

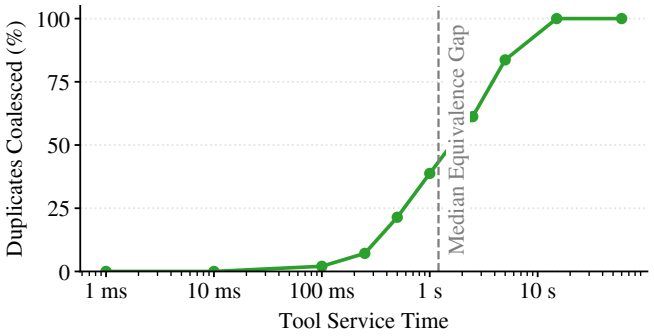


Fig. 2. Modeled discovery-call overlap increases with service time under fixed recorded arrivals.

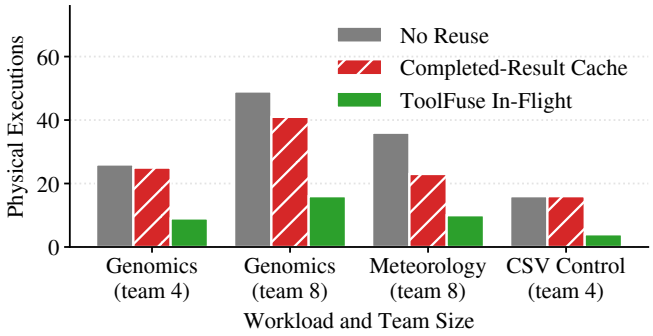


Fig. 3. With 5 s injected delay, duplicates miss the empty cache. In-flight coalescing reduces executions.

TABLE I
DISCOVERY REPLAY EXECUTIONS. CHEAP: EIGHT GROUPS AGGREGATED. EXPENSIVE: ONE EIGHT-AGENT GENOMICS GROUP WITH 5 S INJECTED DELAY.

Policy	Cheap calls (of 159)	Expensive calls (of 49)
No reuse	159	49
Completed cache	61	41
Argument singleflight	159	20
ToolFuse	159	16

from 6.44 GiB to 0.53 GiB. Makespan falls 2.36× (95% CI [2.34, 2.45]), and a second node reproduces the work reduction at 1.93× makespan [1.43, 2.59]. A live safety matrix confirms the boundary holds: differing dataset versions and authorization scopes produce different keys, and two sequential equal calls execute twice. Table I contrasts the policies.

Overlapping equivalent calls reduce work, while contention determines time savings. Sub-millisecond discovery calls show no sharing and at most 0.075% makespan overhead, while a cache absorbs most repeats (Table I). As service time increases, overlap grows (Fig. 2). At 5 s injected delay, duplicates arrive mid-execution and miss an empty cache (Fig. 3). Spreading arrivals over 500 ms creates execution waves, reducing benefit without memoisation. The second corpus models 87 of 919 calls overlapping in flight. Real replay confirms sharing on five tasks, but makespan remains unchanged with fixed arrivals and 97 to 99% idle nodes. Recorded sessions ran separately, so trace concurrency is counterfactual. Fixed replay arrivals prevent measuring whether earlier responses advance later calls.

IV. CONCLUSION

ToolFuse brings provenance-aware execution identity to in-flight sharing of scientific tool calls. Under contention, it turns 16 equivalent requests into one execution, reducing CPU time by 11.79× and makespan by 2.36×, with negligible overhead in the cheap-call control.

ACKNOWLEDGMENT

This material is based upon work supported in part by the National Science Foundation (NSF) under Awards OAC 2411318 and OAC 2313154.

REFERENCES

[1] I. Yildirim, X.-H. Sun, and A. Kougkas, "AgentStage: Exploiting LLM thinking for data staging in scientific agents," in *22nd IEEE International Conference on eScience (eScience)*, 2026, accepted. [Online]. Available: https://www.escience-conference.org/2026/papers

[2] The Go Authors, "Package singleflight," Go package documentation, golang.org/x/sync v0.23.0, accessed: September 30, 2026. [Online]. Available: https://pkg.go.dev/golang.org/x/sync@v0.23.0/singleflight

[3] Y. Babuji, A. Woodard, Z. Li, D. S. Katz, B. Clifford, R. Kumar, L. Lacinski, R. Chard, J. M. Wozniak, I. Foster, M. Wilde, and K. Chard, "Parsl: Pervasive parallel programming in Python," in *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*. ACM, 2019, pp. 25–36. [Online]. Available: https://doi.org/10.1145/3307681.3325400

[4] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, and K. Wenger, "Pegasus, a workflow management system for science automation," *Future Generation Computer Systems*, vol. 46, pp. 17–35, 2015. [Online]. Available: https://doi.org/10.1016/j.future.2014.10.008

[5] Parsl Project, "Memoization and checkpointing," Parsl 1.2.0 user guide, accessed: September 30, 2026. [Online]. Available: https://parsl.readthedocs.io/en/1.2.0/userguide/checkpoints.html

[6] Anonymous, "Tool Cache Agent: Accelerating LLM agent through intelligent tool call caching," Anonymous ICLR 2026 submission, OpenReview, 2026. [Online]. Available: https://openreview.net/forum?id=tX3YcbNa5w