The Eastasouth Journal of Information System and Computer Science
Vol. 3, No. 02, December 2025, pp. 354-365
ISSN: 3025-566X, DOI: 10.58812/esiscs.v3i02

Event-Safe Agent-to-Cloud Fabrics for Transactionally Consistent Autonomous Enterprise Systems

Shivakrishna Vangala
Principal Architect, T-Mobile USA Inc, Bellevue, USA

Article Info

Article history:

Received December, 2025
Revised December, 2025
Accepted December, 2025

Keywords:

Agent-To-Cloud;
Autonomous Agents;
Event-Safe Fabrics;
Eventual Consistency;
Idempotency;
Replicated Logs;
Saga;
Systematic Review;
Transactional Consistency

ABSTRACT

The more and more autonomous software agents perform business operations against the shared cloud state, the more duplicated, misordered and irrecoverable effects are experienced by enterprises in the event of failure and retry. A systematic review of 26 peer-reviewed sources over 46 years was carried out to see how transactional consistency can be ensured from the agent to the cloud. A rubric for eight coordination mechanisms was developed with six criteria, and sources were coded into five domains. The synthesis demonstrates that crash tolerance is 2f + 1 replicas, Byzantine tolerance is 3f + 1, and that update-anywhere replication is N^3 times as many conflicts, where N is the number of nodes, or that deterministic sequencing has been reported at about 500,000 transactions per second. A composite event-safe fabric, by combining a durable intent log, per domain sequencing, idempotent effects and sagas, as well as convergent agent memory, achieved a composite score of 2.45, compared to a mean of 1.94 for single mechanisms. The ranking assumes a certain level of weight for the effect safety, and the conclusions are based on ordinal coding instead of measured agent workloads. Architectural guidelines for designing agent enabled enterprise platforms and directions for empirical research are provided.

This is an open access article under the CC BY-SA license.

Corresponding Author:

Name: Shivakrishna Vangala
Institution: Principal Architect, T-Mobile USA Inc, Bellevue, USA
Email: shivakrishna.vangala@gmail.com

1. INTRODUCTION

The construction of autonomous enterprise systems, where software agents perceive the operational environment, make decisions and act based on the shared state of the enterprise business with minimal human intervention is a growing trend. The weak notion of agency is defined by autonomy, social ability, reactivity and pro-activeness and was formalised in the mid-1990s and is the standard definition of an agent [1]. In enterprise environments, these four properties give rise to four different consistency exposures: autonomy – unsupervised writes, social ability – concurrent writers, reactivity – event bursts, pro-activeness – speculative actions, that can be undone.

As more advancements are made in large language model (LLM) based agents, this exposure has been broadened. These agents are grouped into profile, memory, planning, and action modules that relate to model output and external tools and services [2]. Model outputs are stochastic, so it is

possible that if the same request is made twice, the arguments may be different the second time and the second call may change enterprise records. In this review, event safety refers to the property where all the external action of an agent is applied at most once, is done in a defendable order, and is done in a recoverable outcome.

The size of the coordination is seen in simulated agent societies. The emergent social behavior from individual decisions was found with a sand box of 25 generative agents with their own memory stream, reflection and planning [3]. In the enterprise, it means a number of independent writers operating on components of shared state via cloud interfaces. The research question to be answered is how to organize the agent-to-cloud path to maintain transactional consistency while utilizing the elasticity that drives cloud adoption. The synthesis is guided by three research questions: which guarantees are possible, what is the corresponding coordination cost and what is the composition of mechanisms to achieve these guarantees.

2. REVIEW METHODOLOGY

2.1 Review Protocol

The construction of autonomous enterprise systems, where software agents perceive the operational environment, make decisions and act based on the shared state of the enterprise business with minimal human intervention is a growing trend. The weak notion of agency is defined by autonomy, social ability, reactivity and pro-activeness and was formalised in the mid-1990s and is the standard definition of an agent [1]. In enterprise environments, these four properties give rise to four different consistency exposures: autonomy – unsupervised writes, social ability – concurrent writers, reactivity – event bursts, pro-activeness – speculative actions, that can be undone.

As more advancements are made in large language model (LLM) based agents, this exposure has been broadened. These agents are grouped into profile, memory, planning, and action modules that relate to model output and external tools and services [2]. Model outputs are stochastic, so it is possible that if the same request is made twice, the arguments may be different the second time and the second call may change enterprise records. In this review, event safety refers to the property where all the external action of an agent is applied at most once, is done in a defendable order, and is done in a recoverable outcome.

The size of the coordination is seen in simulated agent societies. The emergent social behavior from individual decisions was found with a sand box of 25 generative agents with their own memory stream, reflection and planning [3]. In the enterprise, it means a number of independent writers operating on components of shared state via cloud interfaces. The research question to be answered is how to organize the agent-to-cloud path to maintain transactional consistency while utilizing the elasticity that drives cloud adoption. The synthesis is guided by three research questions: which guarantees are possible, what is the corresponding coordination cost and what is the composition of mechanisms to achieve these guarantees.

Table 1. Classification of the Reviewed Sources by Primary Domain

Domain	Sources	Share (%)	Reference numbers	Primary concern
Agency	3	11.5	[1], [2], [3]	Definition and architecture of autonomous agents
Cloud infrastructure and data processing	5	19.2	[4], [5], [6], [7], [8]	Elasticity, orchestration, storage, and stream or batch processing
Time and agreement	5	19.2	[9], [10], [11], [12], [13]	Ordering, snapshots, consensus, and fault models

Domain	Sources	Share (%)	Reference numbers	Primary concern
Transactions and correctness	6	23.1	[14], [15], [16], [17], [18], [19]	Isolation, atomicity, recovery, and sequencing
Availability and replication	7	26.9	[20], [21], [22], [23], [24], [25], [26]	Trade-offs, quorums, convergence, and replication cost
Total	26	99.9	—	Shares do not sum to 100 because of rounding

Note. Counts reflect the primary contribution of each source; each of the 26 sources is assigned to exactly one domain.

2.2 *Coding and Scoring*

Six criteria were used to code mechanisms: atomicity, isolation, failure tolerance, availability, scalability, and effect safety in retries. Consistency expectations were based on the linearizability property, where all operations seem to occur at once at some time in the execution interval between operation invocation and operation response [14]. A score of 0-3 was given for each criterion with 0 representing that the criterion was not provided and 3 representing that the criterion was fully provided. No actual measurement is made, the scoring is based on a reviewer's judgement and the results are therefore only interpreted as relative positions.

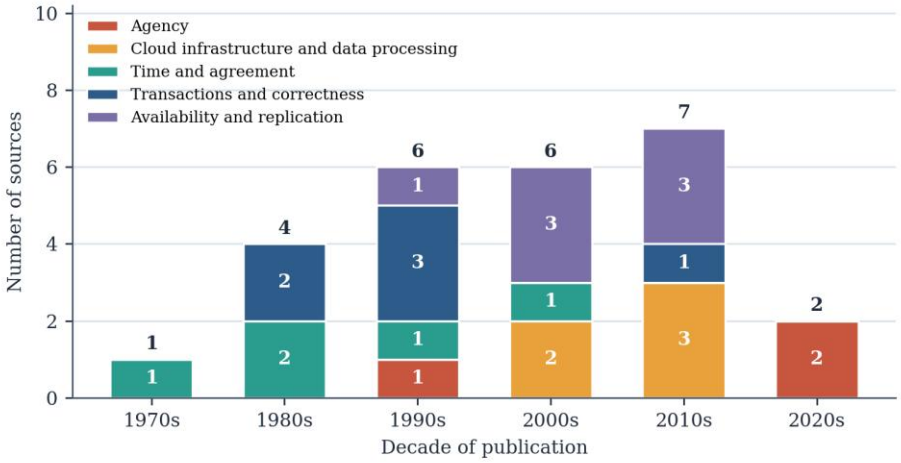


Figure 1. Distribution of Reviewed Sources by Decade and Domain

Isolation is compared to the anomaly catalogue, which expands the original ANSI definitions, differentiating dirty reads, fuzzy reads, phantoms, lost updates, and write skew, and it is demonstrated that snapshot isolation allows write skew [15]. The score of mechanism m is calculated as in Equation (1) using the criterion score s and the criterion weight w. Weights of 0.20, 0.20, 0.20, 0.15, 0.15, and 0.10 were assigned in the criterion order listed above and sum to unity.

$$S_m = \sum_{k=1}^{6} w_k s_{m,k} \quad (1)$$

3. LITERATURE SYNTHESIS

3.1 *Time, State, and Agreement*

Ordering in asynchronous systems is represented by the happened-before relation, whose clock condition is given by Equation (2). This allows for a snapshot to be taken at any given point in time throughout the world, by propagating marker messages over first in first out channels, such that the recorded state would be a consistent cut

that could have been done at the time of execution [10]. In agent fabrics, such cuts can be used for checkpointing agent state and in-flight events, without interrupting execution.

$$a \rightarrow b \Rightarrow C(a) < C(b) \quad (2)$$

Despite the fact that the agents are not synchronized, it is a fundamental result that there is no deterministic protocol that can guarantee consensus in case one agent crashes [11]. Practical systems overcome the result using timeouts, randomness, or leader election, making them more resilient than guaranteed to terminate but less resilient than guaranteed to live, depending on the assumptions. This means that an agent fabric cannot depend on unlimited waiting for the consensus of an agent, and each commit decision must be based on a protocol that does not rely on timing.

A majority-quorum protocol was used for safety in case of crashes, where a value is selected once a majority of acceptors accepts it, with any two quorums having a majority intersection [12]. A set of 2f + 1 replicas can tolerate f crashes, for three crashes it is three replicas, for five crashes it is five replicas. Safety is separated from liveness: the choice delivered by competing ballots does not affect the safety of the value but progress may be blocked until a stable leader emerges.

The stronger the bound, the more arbitrary the behavior. Practical Byzantine fault tolerance was demonstrated to work with 3f + 1 replicas, and a replicated file service on this was found to have an overhead of about 3 percent compared with an unreplicated file service [13]. The two bounds are identical and are shown in Equation (3), and are compared for 1, 2, 3, 4, and 5 allowed faults in Figure 2. When agents or tools are not always trusted to act as specified, Byzantine tolerance comes into play for agent fabrics.

$$n_c \geq 2f + 1,\ n_b \geq 3f + 1 \quad (3)$$

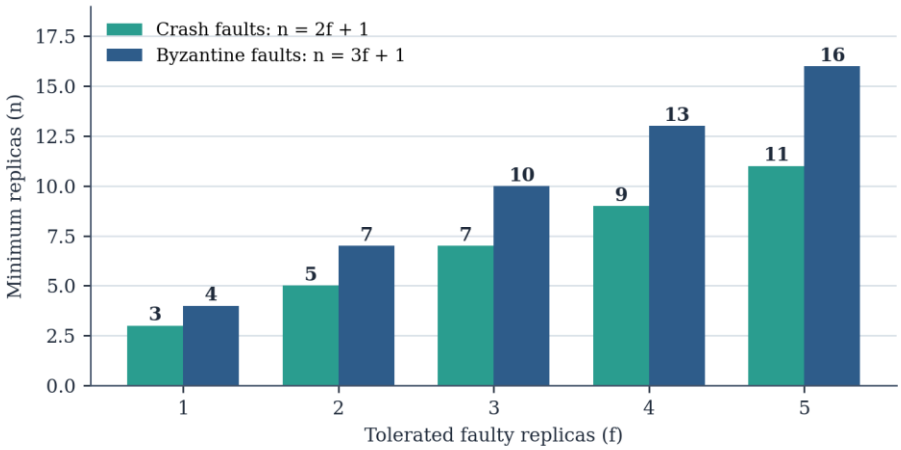


Figure 2. Minimum Replicas Required for Crash and Byzantine Fault Tolerance

3.2 *Transactional Machinery*

The properties of transactional correctness of distributed databases were formalized as serializability, the property of some serial execution being equivalent to a concurrent execution. Concurrency control techniques were separated into three groups: locking, timestamp ordering and combined methods, with an atomic commitment step across sites for each [16]. These foundations are still relevant, even if an agent is not declaring a distributed transaction, it's taking the action of updating records in multiple services.

A single node was sufficient, because write-ahead logging with analysis, redo and undo passes was able

to combine durability and atomicity. The redo and undo types are used for rolling back and logging compensation, respectively, and recover history as it happens when redoing.Recovery is redo and records compensation when undoing, allowing for fine-granularity locking and partial rollbacks [17]. The concept of an agent intent being logged before applying it to the fabric is carried over to the fabric: An agent intent to be applied to a log must be durably appended to the log before any external side effect is attempted, so that recovery can be resumed instead of being guessed.

Long-lived activities are unable to acquire locks for their lifetime. The saga model breaks down such an activity into subtransactions with compensating transactions, which are then executed in reverse order in the case of failure after the j-th subtransaction in order to repair it [18]. The execution of the resulting is written in Equation (4). Sagas allow you to have atomicity of intent, albeit with less isolation because intermediate states can be seen by other readers as the agent workflow can take minutes and cross organizational lines.

$$T_1, T_2, \dots, T_j, C_j, \dots, C_1 \quad (4)$$

The more the copies, the greater the coordination cost. An update anywhere replication analysis revealed that the number of deadlocks and the number of reconciliations are approximately proportional to the cube of the number of nodes, as indicated in Equation (5) [20]. The master-based schemes grow slower, at the square rate, and the two growth laws are compared in Figure 3. Therefore, update-anywhere designs should not be used for contended records and each record should belong to just one sequencing domain. Table 2 draws connections between the sources reviewed and the concepts included in the framework.

$$D_u(N) = N^3 D_1 \quad (5)$$

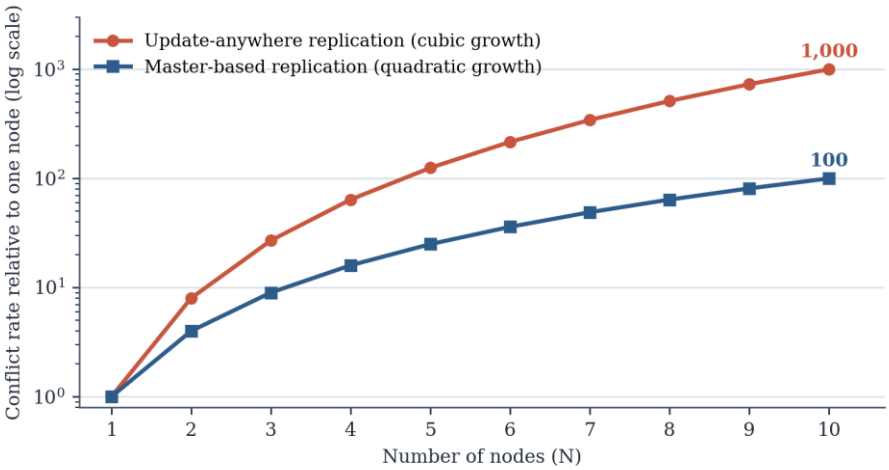


Figure 3. Growth of Replication Conflicts with Node Count

Table 2. Mapping of Reviewed Literature to Framework Concepts

Concept	Formal object	Reference numbers	Role in the fabric
Agency and agent modules	Autonomy, memory, planning, action	[1], [2], [3]	Source of concurrent and retried writes
Cloud deployment	Elastic, pay-per-use resources	[4], [5]	Placement and recovery of agent runtimes
Logical time	Happened-before relation	[9], [10]	Ordering and consistent checkpoints

Concept	Formal object	Reference numbers	Role in the fabric
Correctness conditions	Linearizability, isolation anomalies	[14], [15]	Criteria for scoring mechanisms
Consensus and fault bounds	2f + 1 and 3f + 1 replicas	[11], [12], [13]	Replicated intent log
Serializability and recovery	Concurrency control, write-ahead log	[16], [17]	Durable intent before effect
Long-lived transactions	Saga with compensations	[18]	Cross-service atomicity of intent
Replication cost	Cubic conflict growth	[20]	Single sequencing domain per record
Consistency trade-offs	Partition mode, quorums	[21], [22], [23], [24], [25]	Per-record-class posture
Storage and event time	Single-row atomicity, watermarks	[6], [7]	Idempotent effects and late events
Sequencing and convergence	Deterministic order, CRDTs	[19], [26]	Ordered effects and agent memory
Bulk processing	Restart-based fault tolerance	[8]	Replay and reconciliation

Note. Reference numbers correspond to the numbered reference list.

3.3 Availability and Convergence

The consistency vs availability vs partition tolerance trade-off proved formally: A Web service can't achieve atomic consistency, availability, and partition tolerance at the same time [21]. The proof is valid for the most consistent definition; weaker consistency definitions such as delayed consistency and partial availability were demonstrated to be attainable. Therefore, agent fabrics need to determine per record class which property emits during a partition and not take a universal approach.

A subsequent re-evaluation concluded that the two-of-three formulation is misleading as partitions occur rarely, and can be taken at a fine granularity, operation-by-operation and/or data-item-by-data-item [22]. The suggested approach is to find partitions, go into an explicit partition mode and recover by reconciling/explicitly compensating for errors. This sequence is suitable for agent fabrics that have a few modes of operation, such as read-only or approval-required, that can be set for a detected partition.

Eventual consistency was defined as a set of guarantees: read-your-writes, monotonic reads, and session consistency. When the overlap condition (6) is satisfied, quorum replication with N replicas, R=read quorum, and W=write quorum guarantees strong consistency; but smaller quorums achieve greater freshness at the cost of increased latency [23]. Figure 4 shows the overlap margin R + W − N with N = 5 with the read and write configurations overlapping. The overlap margin R + W − N is plotted in Figure 4 for N = 5, where positive margins correspond to configurations that have overlaps between read and write where every read overlaps with the latest write.

$$R + W > N \quad (6)$$

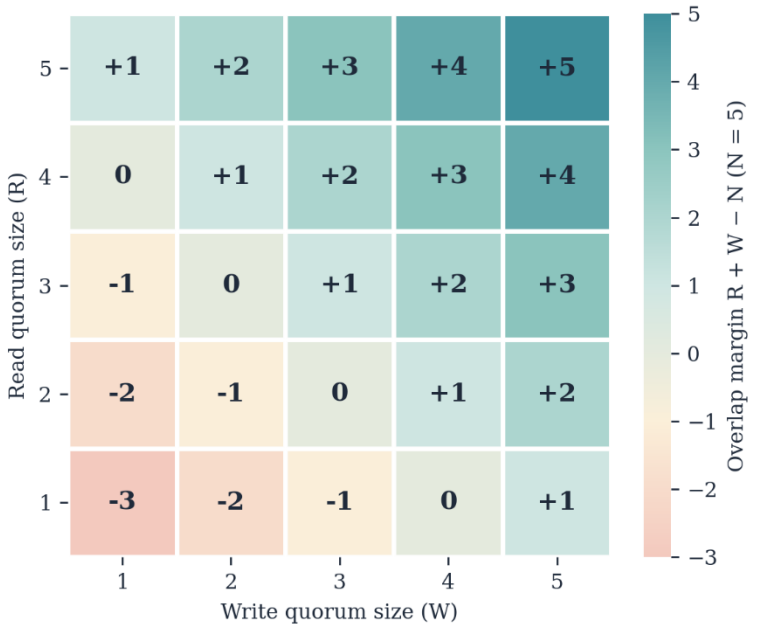


Figure 4. Quorum Overlap Margin for Five Replicas

A production key-value store implemented this approach at scale: the typical configuration of N = 3, R = 2 and W = 2 was used, with vector clocks used to reconcile versioned objects, and a 300 ms target response time at the 99.9th percentile, with a peak load of 500 requests per second [24]. The design took on the responsibility of application-level conflict resolution for the trade off of always-writable behavior. Table 3 shows some example quorum configurations and their consistency posture.

Table 3. Representative Quorum Configurations and Consistency Posture

N	R	W	R + W − N	Failures tolerated (reads / writes)	Consistency posture
3	2	2	+1	1 / 1	Strong overlap; common balanced setting
3	1	1	−1	2 / 2	Eventual; lowest latency
3	1	3	+1	2 / 0	Strong overlap; read-optimized
3	3	1	+1	0 / 2	Strong overlap; write-optimized
5	3	3	+1	2 / 2	Strong overlap; majority quorums
5	2	2	−1	3 / 3	Eventual; stale reads possible
5	1	1	−3	4 / 4	Eventual; weakest overlap

Note. Failures tolerated are computed as N − R for reads and N − W for writes; a positive overlap margin implies that each read quorum intersects each write quorum.

4. EVENT-SAFE FABRIC FRAMEWORK

4.1 Reference Architecture

The reviewed mechanisms were synthesized into a reference architecture as shown in Figure 5. The control plane is based on declarative orchestration and reconciliation loops, such as cluster managers, which started as a centralized scheduler, then were moved to an optimistic, shared-state scheduler, and are now implemented in a container platform with labels and controllers [5]. Agent runtimes are provisioned as supervised workloads with the desired state declaration, and once they've crashed, they are automatically restarted without human intervention. Based on the runtimes and the cloud services, five layers were able to be identified: intent

capture, durable log, sequencer, effect executor, and state stores.

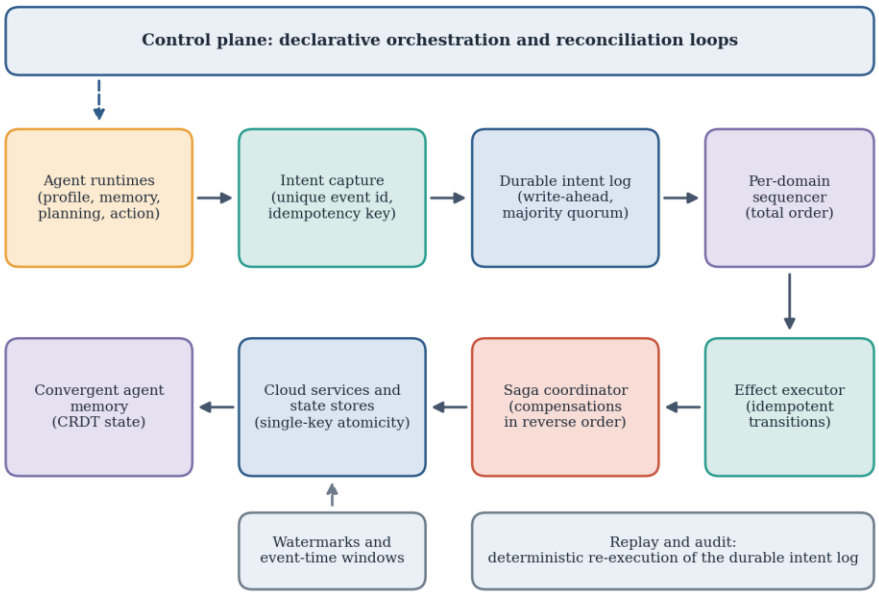


Figure 5. Reference Architecture of the Event-Safe Agent-to-Cloud Fabric

To capture intent in a durable manner, a store that has atomic single-row operations at high scale is needed. A distributed sorted-map storage system is used to support atomic reads and modifications around a single row key, and is used by over 60 products from a single operator, suggesting that single-key atomicity is adequate for many workloads [6]. Intents are therefore specified by an identifier for the particular event, while an effect is applied with an idempotent transition, as defined by Equation (7), which does not cause any changes to the state when duplication occurs due to retries.

$$\delta(\delta(s,e),e) = \delta(s,e) \quad (7)$$

4.2 Requirement Traceability

Events occur out-of-order and late. The dataflow model is based on separation of four concerns: what is computed, where in event time are the results grouped, when in processing time they are emitted, as well as how refinements relate to each other, in terms of cost, correctness and latency, with the help of windows, watermarks and triggers. Completeness of agent event streams is evaluated on event time, not arrival time, within the fabric via watermark-based completeness. Table 4 shows each requirement of the fabric and the supporting mechanism.

Table 4. Traceability from Fabric Requirements to Mechanisms

ID	Requirement	Mechanism	Refs.	Residual risk
R1	At-most-once effects	Idempotent transitions keyed by event identifier	[6]	Identifier collisions or lost deduplication state
R2	Durable intent	Write-ahead intent log	[17]	Log availability bounds progress
R3	Defensible order	Per-domain sequencer and logical clocks	[9], [19]	Sequencer throughput per domain
R4	Multi-service atomicity	Saga with compensations	[18]	Intermediate states visible to readers
R5	Late event handling	Watermarks and triggers	[7]	Heuristic watermarks may misclassify lateness

ID	Requirement	Mechanism	Refs.	Residual risk
R6	Crash tolerance	Majority-quorum log with 2f + 1 replicas	[12]	Stall without a majority
R7	Placement and recovery	Declarative orchestration and reconciliation	[5]	Control-plane dependence
R8	Partition behavior	Explicit partition mode with compensation	[22]	Operational limits during partitions

Note. Each requirement is supported by at least one mechanism from the reviewed corpus.

5. COMPARATIVE ANALYSIS

5.1 Mechanism Scoring

The scores for the eight mechanisms are shown in Table 5 and broken down by criterion in Figure 6. The highest was the composite fabric (2.45), deterministic sequencing (2.40), saga compensation (2.10) and Byzantine replication (2.00) while the lowest was quorum-based eventual consistency (1.50). The sequencing positioning is consistent with a reported throughput of approximately 500,000 TPC-C transactions per second on a commodity cluster of which the transactions are sequenced by agreeing on transaction order prior to execution, thus eliminating two phase commit [19]. The difference between the two leaders is not so great when the effect-safety criterion is removed and the remaining weights renormalized: 2.44 for sequencing and 2.39 for the composite fabric.

Table 5. Reviewer-Coded Scores and Composite Scores of Coordination Mechanisms

Mechanism	A	I	F	V	S	E	Total
Two-phase commit with locking	3	3	1	1	1	1	1.80
Majority-quorum replicated log	2	3	2	1	2	1	1.95
Byzantine replication	2	3	3	1	1	1	2.00
Saga with compensation	2	1	2	3	3	2	2.10
Quorum eventual consistency	0	1	2	3	3	0	1.50
Deterministic sequencing	3	3	2	1	3	2	2.40
CRDT-based replication	0	2	2	3	3	1	1.80
Composite event-safe fabric	3	2	2	2	3	3	2.45
Mean of seven single mechanisms							1.94

Note. Scores are ordinal values from 0 to 3 assigned by the review protocol. A = atomicity, I = isolation, F = failure tolerance, V = availability, S = scalability, E = effect safety; weights are 0.20, 0.20, 0.20, 0.15, 0.15, and 0.10. The total is the composite score computed with Equation (1).

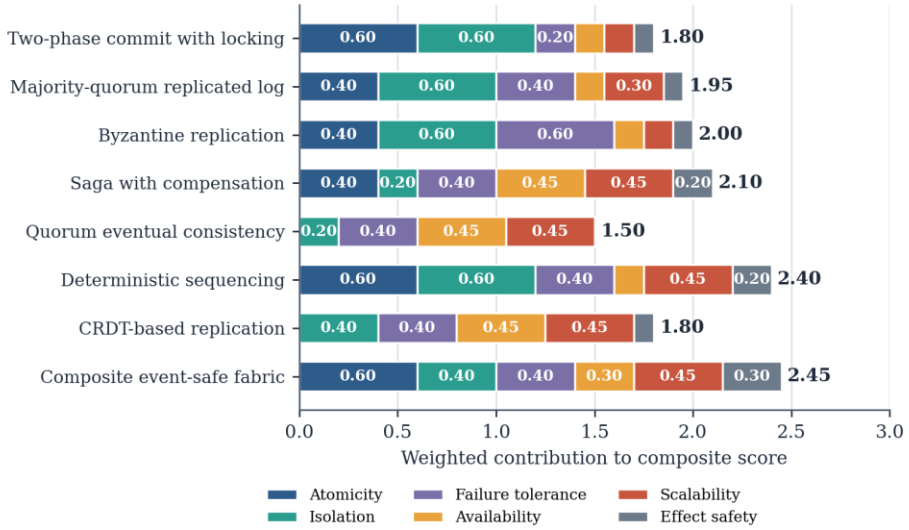


Figure 6. Weighted Criterion Contributions to Composite Mechanism Scores

5.2 *Coordination and Scale*

The second comparison is about coordination avoidance. Highly available transactions were demonstrated to be able to deliver read committed, monotonic atomic view, multiple session guarantees without synchronous coordination, while lost update and write skew can be avoided only by coordinating [25]. Table 6 compares the mechanisms based on the requirement of the replica, risk of blocking, and the evidence reported. The three coordination-free mechanisms were all given 3 points for their availability, while the four synchronous agreement-based mechanisms were given 1 point, a difference of two points for the number of round trips saved.

Table 6. Coordination Requirements and Reported Evidence by Mechanism

Mechanism	Replicas for f = 1	Blocking risk	Reported evidence
Two-phase commit with locking	Coordinator and participants	In-doubt blocking on coordinator failure	Formally surveyed; no throughput figure in the corpus
Majority-quorum replicated log	3	Stalls without majority or leader	Safety proved; liveness under assumptions
Byzantine replication	4	Stalls above f faulty replicas	About 3% overhead on a replicated file service
Saga with compensation	Not applicable	None; intermediate states visible	Compensation in reverse order after failure
Quorum eventual consistency	3 (N = 3)	None; writes remain accepted	300 ms at the 99.9th percentile, 500 requests/s
Deterministic sequencing	3 for the input log	Bounded by log replication	About 500,000 TPC-C transactions/s
CRDT-based replication	No quorum required	None	Convergence without consensus
Composite event-safe fabric	3 for the intent log	Bounded by log quorum	Synthesized; not measured

Note. Entries summarize figures reported in the reviewed sources; the composite fabric is a synthesis of this review and has no reported measurement.

6. DISCUSSION

6.1 Design Implications

The convergence problem of agent memory and shared working state poses a coordination challenge which cannot be solved economically. Conflict-free replicated data types ensure effective eventual convergence in the case of a join semilattice, where the merge operation is commutative, associative and idempotent, allowing replicas with the same updates to converge to a consistent state without consensus [26]. Notes, annotations and scratch memory of agents are best recorded using counters, sets and registers. Balances, inventory and approvals are not because there are invariants that need coordination like non-negative balances.

6.2 Limitations and Outlook

Complete the discussion by scaling and replaying. The restarts-based approach to fault tolerance proved to be practical on massive scale: the operator of one such framework has reported that it can process more than 100,000 jobs a day, and more than 20 petabytes a day [8]. Similarly, the durable intent log for the fabric enables deterministic replay for audit/reconciliation, where corrections are recomputed in bulk. The log also serves an ethical and regulatory purpose by making it possible to leave a trace of the actions taken, as they are all autonomous and can be attributed to a particular person and/or task. The small number of 26 sources, the single protocol used for scoring and the fact that agent workloads were not measured limited the scope of the results.

7. CONCLUSION

This systematic review of 26 sources found that transactional consistency along the agent-to-cloud path can be achieved with a combination of mechanisms – not just one of them – namely a durable intent log, per domain sequencing, idempotent effects, saga compensation, and convergent agent memory. The composite fabric score was 2.45, higher than the 1.94 mean for the seven single mechanisms, but the lead over deterministic sequencing (2.40) is dependent on the weight of effect of safety. The coordination cost a design must accept was bounded using quantitative anchors taken from the literature: the 2f + 1 and 3f + 1 replica bounds and cubic growth of the number of replication conflicts.

Future work should include measuring event-safe fabrics in realistic agent workloads, formally verifying idempotency and compensation properties, and tuning the scoring weights with inputs from practitioners. It is a valuable next step to compare the two approaches (sequencing-based and saga-based) in practice for long-running, cross-organizational agent workflows.

REFERENCES

[1] M. Wooldridge and N. R. Jennings, "Intelligent agents: Theory and practice," *Knowl. Eng. Rev.*, vol. 10, no. 2, pp. 115–152, 1995, doi: 10.1017/S0269888900008122.

[2] L. Wang *et al.*, "A survey on large language model based autonomous agents," *Front. Comput. Sci.*, vol. 18, no. 6, p. 186345, 2024, doi: 10.1007/s11704-024-40231-1.

[3] J. S. Park, J. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, "Generative agents: Interactive simulacra of human behavior," in *Proceedings of the 36th annual acm symposium on user interface software and technology*, 2023, pp. 1–22. doi: 10.1145/3586183.3606763.

[4] M. Armbrust *et al.*, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010, doi: 10.1145/1721654.1721672.

[5] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, Omega, and Kubernetes: Lessons learned from three container-management systems over a decade," *Queue*, vol. 14, no. 1, pp. 70–93, Jan. 2016, doi: 10.1145/2898442.2898444.

[6] F. Chang *et al.*, "Bigtable: A Distributed Storage System for Structured Data," *ACM Trans. Comput. Syst.*, vol. 26, no. 2, pp. 1–26, 2008, doi: 10.1145/1365815.1365816.

[7] T. Akidau *et al.*, "The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing," *Proc. VLDB Endow.*, vol. 8, no. 12, pp. 1792–1803, 2015, doi:

10.14778/2824032.2824076.

[8] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008, doi: 10.1145/1327452.1327492.

[9] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, 1978, doi: 10.1145/359545.359563.

[10] K. M. Chandy and L. Lamport, "Distributed snapshots: Determining global states of distributed systems," *ACM Trans. Comput. Syst.*, vol. 3, no. 1, pp. 63–75, 1985, doi: 10.1145/214451.214456.

[11] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *J. ACM*, vol. 32, no. 2, pp. 374–382, 1985, doi: 10.1145/3149.214121.

[12] L. Lamport, "The part-time parliament," *ACM Trans. Comput. Syst.*, vol. 16, no. 2, pp. 133–169, 1998, doi: 10.1145/279227.279229.

[13] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Trans. Comput. Syst.*, vol. 20, no. 4, pp. 398–461, Nov. 2002, doi: 10.1145/571637.571640.

[14] M. P. Herlihy and J. M. Wing, "Linearizability: a correctness condition for concurrent objects," *ACM Trans. Program. Lang. Syst.*, vol. 12, no. 3, pp. 463–492, 1990, doi: 10.1145/78969.78972.

[15] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil, "A critique of ANSI SQL isolation levels," *ACM SIGMOD Rec.*, vol. 24, no. 2, pp. 1–10, 1995, doi: 10.1145/568271.223785.

[16] P. A. Bernstein and N. Goodman, "Concurrency Control in Distributed Database Systems," *ACM Comput. Surv.*, vol. 13, no. 2, pp. 185–221, 1981, doi: 10.1145/356842.356846.

[17] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz, "ARIES: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging," *ACM Trans. Database Syst.*, vol. 17, no. 1, pp. 94–162, 1992, doi: 10.1145/128765.128770.

[18] H. Garcia-Molina and K. Salem, "Sagas," *SIGMOD Rec.*, vol. 16, no. 3, pp. 249–259, 1987, doi: 10.1145/38714.38742.

[19] A. Thomson, T. Diamond, S.-C. Weng, K. Ren, P. Shao, and D. J. Abadi, "Calvin: fast distributed transactions for partitioned database systems," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, in SIGMOD '12. New York, NY, USA: Association for Computing Machinery, 2012, pp. 1–12. doi: 10.1145/2213836.2213838.

[20] J. Gray, P. Helland, P. O'Neil, and D. Shasha, "The dangers of replication and a solution," in *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, in SIGMOD '96. New York, NY, USA: Association for Computing Machinery, 1996, pp. 173–182. doi: 10.1145/233269.233330.

[21] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *Acm Sigact News*, vol. 33, no. 2, pp. 51–59, 2002, doi: 10.1145/564585.564601.

[22] E. Brewer, "CAP twelve years later: How the 'rules' have changed," *Computer (Long. Beach. Calif).*, vol. 45, no. 2, pp. 23–29, 2012, doi: 10.1109/MC.2012.37.

[23] W. Vogels, "Eventually consistent," *Commun. ACM*, vol. 52, no. 1, pp. 40–44, 2009, doi: 10.1145/1435417.1435432.

[24] G. DeCandia *et al.*, "Dynamo: Amazon's highly available key-value store," *ACM SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, pp. 205–220, 2007, doi: 10.1145/1294261.1294281.

[25] P. Bailis, A. Davidson, A. Fekete, A. Ghodsi, J. M. Hellerstein, and I. Stoica, "Highly available transactions: virtues and limitations," *Proc. VLDB Endow.*, vol. 7, no. 3, pp. 181–192, Nov. 2013, doi: 10.14778/2732232.2732237.

[26] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "Conflict-Free Replicated Data Types BT - Stabilization, Safety, and Security of Distributed Systems," in *Stabilization, Safety, and Security of Distributed Systems*, X. Défago, F. Petit, and V. Villain, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 386–400. doi: https://doi.org/10.1007/978-3-642-24550-3_29.