Sciences du Nord Nature Science and Technology
Vol. 2, No. 02, December 2025: pp. 170-181
E-ISSN: 3046-6504
P-ISSN: 3047-4922
DOI: https://doi.org/10.71238/snnst.v1i02.236

Resilience-Aware Serverless Orchestration for Stateful Enterprise Workloads Across Multi-Region AWS Environments

Kedar Pradhan [1]
[1] Solutions Architect Amazon Web Services, Austin, USA
Corresponding Author: kedar_pradhan90@yahoo.com

ARTICLE HISTORY
Received July, 2025 **Revised Sept, 2025** **Accepted Dec, 2025**

ABSTRACT
Distributed applications are useful on servers, but enterprise applications like order processing and payments applications have a need for durable state, multi-step coordination, and predictable behaviour in case of failure of an AWS Region.Distributed applications work well on servers, but enterprise workloads like order processing and payments demand durable state, multi-step coordination and predictable behaviour in the case of AWS Region failures. This article examines the literature and the primary AWS documentation up to this date and investigates the possibility of resiliently orchestrating such workloads across Regions. The selected and relevant sources from a focused search into systems, database, and software-engineering venues supported the identification of mechanisms, guarantees, evaluation settings, and reported results extracted and placed into an evidence matrix of 62 sources. With respect to serverless workflows, each of the reported log-based protocols provides exactly-once execution and transactions, and is 2x or 4x faster than the unprotected version, all running within a single failure domain, and with a reported 20–40% improvement. The default configuration for AWS is to replicate business data across Regions, each of which have a claimed latency of about 1 second, and the state of executing workflows is region scoped. In multi-Region workflows, the availability model is separated into availability of the work, in addition to the availability of the recovery time and recovery points. It shows world dependence and how this impacts availability, and shows that workflow progress is not replicated but rather the availability of this progress is the effective recovery point.

Keywords: *AWS; Disaster Recovery; Exactly-Once Semantics; Multi-Region Resilience; Serverless Workflow Orchestration; Stateful Serverless Computing*

INTRODUCTION

In particular, serverless computing and its Function-as-a-Service (FaaS) form took the burden of provisioning, scaling, and much of the operational burden of distributed applications from tenants to cloud providers [1], [2]. On Amazon Web Services (AWS) the Lambda platform has claimed to scale to as many as 15,000 lambda containers per second for a single customer, serving millions of requests per second [3]. However, the research into the actual use of these platforms shows that they are not only being used as event handlers, but also in back-office systems, data pipelines, and customer-facing interfaces [4], [5].

But the majority of enterprise workloads are not stateless. The order fulfilment, payment capture, and claims processing tasks have well-defined behaviour on a failure in the middle of the task, durable state and multi-step coordination. Initial critics of FaaS platforms claimed that they didn't deal with such state well: functions are short-lived, they can't address each other directly, and they must communicate via comparatively slow external storage [6]. The services that still had the workflow for the developer included both orchestration services: AWS Step Functions and Azure

Durable Functions [7] – and the initial benchmarks of these services revealed them as underdeveloped for parallel and long running processes [8].

But there is another level: Resilience. A survey of 32 Internet services covering 597 unplanned outages revealed that the average uptime for 78% of the services (25) was less than 99.9% [9]. Metastable failures are at least part of the cause of 15 major AWS outages in the past 10 years, as failures continue to leave a system degraded even after the trigger has passed [10]. AWS considers Region as one of the prime fault isolation domains and provides a variety of disaster-recovery (DR) solutions ranging from backup/restore to multi-site active/active [11], [12].

There has been relatively little integration of these strands. There are studies that give guidance to serverless workflows with exactly-once [13] or transactional guarantees [14] within a single Region, while documentation on how to operate with several Regions is mainly contained in vendor documentation about individual data stores.

LITERATURE REVIEW

2.1 Serverless Platforms and Their Failure Model

In commercial FaaS platforms, the functions are isolated with lightweight virtual machines or containers [15], [16], [17], while in AWS Lambda, the isolation is provided by firecracker microVM monitor [18]. Measurements [19] found the placement to be opaque and trace analysis found eight orders of magnitude of cold start frequencies [20] and two to four orders of magnitude of cold-start and execution times [21]. The two most important properties for workflow design are: However, Lambda can only execute for a maximum of 900 seconds and per Region per Lambda invocation.A Lambda invocation is limited to 900 seconds and per Region, quota-limited. By the AWS Express workflows semantics correctness is guaranteed in case of re-execution, which gives at-least-on execution semantics for platforms. When the data is accessed at hyperscale, Meta's XFaaS will prefer to execute locally because the cross-region bandwidth is typically 10 times cheaper and the latency is 100 to 1000 times higher, and multiplicatively throttle when the downstream services report that they are overloaded [22].

2.2 Stateful Serverless and Exactly-Once Execution

Function instances are ephemeral so there's a need for somewhere else to store state. Early systems might be able to offer any kind of ephemeral storage [23] or application-level sandboxing [24] or shared mutable objects [25] or shared memory via WebAssembly isolation [26] or a key-value store with function-local caches [27]. They are geared towards performance and generally also inherit fault tolerance from the underlying store.

Correctness under failure was dealt with directly. To ensure read-atomic isolation, despite retrial of functions, AFT inserted a shim between functions and storage [28]. Beldi enhanced latency of write operations to AWS Lambda and AWS Dynamo (DDB) by adding operation logs, an intent table and transactions; it claimed to deliver median latency up to 100 percent better than the unprotected baseline up to about 400 requests per second (rps) and up to 240 percent to 330 percent faster at saturation [13]. Boki implemented a shared log, instead of per operation database logging, and achieved performance speeds of 4.3-4.7 times faster than Beldi on the workflow library [29] for the same runtime. Halfmoon showed that logging does not have to come with the overhead of writing, and it reported latency to be 20-40% less than Boki [30] and the logging overhead to be 1.5-4 times less. Both Kappa [31] and ExoFlow [32] checkpoint long running functions for resumption if they get timeouted and ExoFlow stores the semantics for exactly-once workflows as a DAG.

2.3 Orchestration Models

Orchestration services will make sure that a progress continues even if a step fails. The new Netherite engine uses event sourcing and deterministic replay, requiring orchestrator code to be deterministic [7], [8] and it has been reported that it could be up to 18.6 times faster than the original implementation in terms of throughput [33] and up to 16 times faster in terms of median latency. With AWS Step Functions, you can use two types of workflow. The default workflows run and execute for one year, and Express workflows execute and run for at-least-once semantics (asynchronous) and at-most-once semantics (synchronous) up to 5 minutes. Step Functions' median orchestration overhead was determined to be 0.1 - 0.2 s for sequential workflows [34] and a study revealed that Azure is superior to Azure DevOps for inferences pipelines but inferior to parallel processing [35]. Orchestrator-free designs [36] and trigger-based orchestration question the necessity of the central service, and Unum's runtime is two times as fast and nine times as affordable as Step Functions' [37].

2.4 Geo-Distribution, Consistency, and Multi-Region Data

Multi-Region state continues to have the CAP and PACELC trade-offs of availability, latency and consistency [38], [39]. Amazon's Dynamo selected eventual consistency [40] and its replacement DynamoDB, reported to have been able to handle 89.2 million requests per second at a sales event [41]. By default, DynamoDB global tables replicate asynchronously (usually within a second), promptly resolve concurrent writes using last-writer-wins, and support atomicity only in the Region where the transactions are issued. AWS thus categorize their modes as write to any Region, write to one Region, and write to your region based on whether they are written idempotently. A preview of a synchronous, strongly consistent multi-Region mode with ZERO recovery point objective (RPO) was released.

Aurora pushes redo-log processing to distributed storage, with sub-second replication guaranteed by its Global Database, and a one-second RPO, sub-minute recovery [42]. In addition to AWS, other solutions have proved it is possible to achieve externally consistent global transactions at the cost of introducing high commit latencies across the network [43], weaker but partition-tolerant variants of application state, including causal consistency [44] and conflict-free replicated data types [45] and saga-style compensation [46], [47].

2.5 Resilience Engineering and Federated Execution

In accordance with the dependability theory, the difference between faults, errors and failures exists, and the concept of availability is defined as being ready in the correct service [48]. The failures that have occurred in the cloud have highlighted gray failures that pass health checks [49] and also metastable states maintained by retry amplification [10], [50]. AWS practice many small independently failing units of state with static reliability [51], and a preference for operations on the data plane over the control plane during recovery, as described in their reliability guidance [11], [12] - and explicitly testing these in production [52]. The most similar academic model to multi-Region orchestration is federated serverless research. With several FaaS providers, xAFCL achieved better Success Rate of choreographies by 53.45% and higher number of Functions called by 3.94% when compared to AWS Step Functions [53], [54], and with a virtual serverless provider (VSP) across clouds, xAFCL reduced the number of SLO (service level objective) violations in preliminary experiments by 98.8% [55].

METHODS

This study is an evidence based literature synthesis and not a systematic review. The searches were performed against search engines, general search engines, the publisher search, proceedings search (USENIX, ACM Digital Library, VLDB Endowment, IEEE, arXiv), and on AWS documentation. The words serverless, Function-as-a-Service, stateful, workflow, orchestration, exactly-once, fault tolerance, multi-Region, geo-replication, disaster recovery, and AWS were all included in the search strings, and the lists of retrieved papers were then searched back for these terms. Given the engine type, no attempt is made to provide a reproducible export and counts of screens will not be reported nor is a claim for complete coverage made.

Any sources, peer-reviewed papers, archival technical reports or preprints from a clearly defined research group, or AWS serverless documentation were included.A source was deemed if it was peer-reviewed, archival technical reports or preprints from a research group, or AWS serverless documentation related to serverless platform behaviour, state management or fault tolerance in serverless execution, workflow orchestration, geo-replicated data management or cloud resilience practice [17]. Not included are blogs or marketing, and AWS features that were announced after the review period. The final corpus consisted of 62 sources, 53 academic and 9 vendor documents (Figure 1).

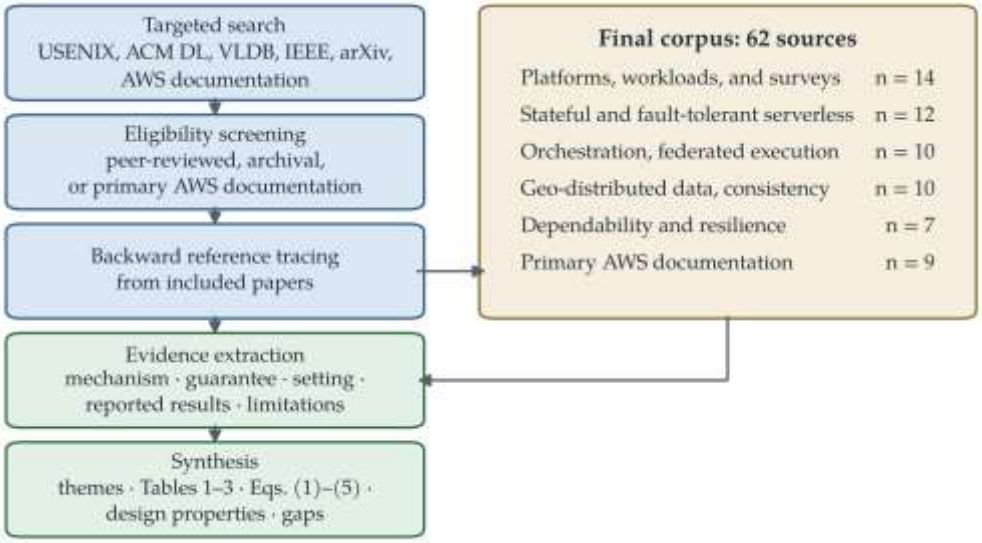


Figure 1. Review protocol and composition of the final corpus

Source: Author's elaboration from the review process; counts refer to the reference list of this article.

RESULTS AND DISCUSSION

4.1 Evidence Matrix of Fault-Tolerance Mechanisms

The main systems are summarized in table 1. There are three key findings. However, it has been demonstrated that the cost of executing exactly-once is achievable on commodity serverless platforms, and that the cost has been improved as designs have evolved: Beldi added nearly 2x latency at moderate load [13], Boki improved the latency by more than 4x [14] and Halfmoon improved the cost by 20–40% [30]. The direction is consistent but the magnitude can't be added because of different baseline and infrastructures used in each study. Secondly, none of these systems were tested Regionally. Beldi assumes that it has strongly consistent databases which are not

available during partitions [13] (in contrast to the default global tables, which are asynchronously replicated). Thirdly, only systems that extend across failure domains offer provider-level failure domain availability, but offer no guarantees of shared-state transactions [53], [54], [55].

Table 1. Evidence matrix of fault-tolerance and orchestration mechanisms for stateful serverless workflows

System	Mechanism	Guarantee	Evaluation setting	Reported result (own baseline)	Cross-Region scope
AFT [28]	Shim between functions and storage	Read-atomic isolation under retries	Serverless functions over existing storage engines	Minimal overhead; thousands of req/s (qualitative)	None reported
Beldi [13]	Linked operation logs, intent table, callbacks, garbage collection	Exactly-once; transactions	AWS Lambda + DynamoDB; three applications	Median ≈2× baseline to ≈400 req/s; 2.4–3.3× at saturation	None
Boki [14]	Shared log (metalog, LogBook API)	Exactly-once workflows; transactions	Research cluster on Nightcore [29]	Workflows 4.3–4.7× faster than Beldi	None reported
Halfmoon [30]	One-sided logging of reads or writes	Exactly-once; log-optimal	Compared against Boki	20–40% lower latency; 1.5–4.0× lower logging overhead	None reported
Durable Functions / Netherite [7], [33]	Event sourcing, deterministic replay; partitioned group commit	Durable orchestration progress	Azure Functions	Up to 18.6× throughput; up to 16× median latency vs. original engine	Not addressed
Step Functions Standard	Managed persistence of state transitions	Exactly-once steps (unless Retry is set)	Managed AWS service	Median orchestration overhead 0.1–0.2 s, sequential [34]	Region-scoped
Unum [37]	Orchestration compiled into functions	Fault-tolerant orchestration without central service	Serverless platforms	Up to 2× faster, 9× cheaper than Step Functions	None reported
rAFCL [54]	Alternative functions across providers	Choreography availability	Three providers; three applications	+53.45% success vs. Step Functions; +3.94% invocations	Multi-provider
Virtual serverless provider [55]	Cross-cloud request routing	SLO compliance	Preliminary multi-cloud experiments	−98.8% SLO violations; −54% invocation cost	Multi-provider

Source: Compiled from the cited studies; results are reported against each study's own baseline and are not directly comparable.

4.2 AWS Primitives and Their Recovery Semantics

Table 2 illustrates common AWS building blocks and scope and replication behaviour. What is important to notice is that there is an asymmetry in the business data replicating across Regions using managed services, but not for the execution state of the workflow. The service doesn't provide any built-in support for replications of the state of an execution, or for replications of tables with the claimed lags of approximately one second as Step Functions does not provide such support. Therefore, in-flight executions need to be restarted (or reconstructed in the remaining Region from the externally persisted checkpoints) after a Region failover. Last-writer-wins semantics of default global tables, combined with the fact that they are not transactional means that a restarted workflow might observe part of the results of the original execution.

Table 2. Recovery-relevant semantics of AWS building blocks for stateful serverless workflows

Service	Scope	Replication and semantics	Vendor-stated recovery figures	Implication for workflows
AWS Lambda	Regional; quotas per Region	Stateless; 900 s maximum duration per invocation	Not applicable	Re-executed steps must be idempotent or logged [13]
Step Functions Standard	Regional	Exactly-once steps; up to one year; history retained 90 days	No cross-Region replication of executions	In-flight executions lost on Regional failover unless checkpointed externally
Step Functions Express	Regional	At-least-once (async) or at-most-once (sync); up to five minutes	None	Requires idempotent steps
DynamoDB global tables, eventual mode	Multi-Region, active-active	Asynchronous; last-writer-wins; transactions atomic only in origin Region	Replication typically ≤1 s; RPO equals replication delay	Write mode must match idempotence; conditional writes need a single writer
DynamoDB global tables, strong mode	Three US Regions (preview)	Synchronous replication	RPO of zero	Preview; outside production evidence
Aurora Global Database [42]	Primary and secondary Regions	Storage-level asynchronous replication; one primary (writer) Region	Lag <1 s; RPO 1 s; RTO <1 min	Failover by promoting a secondary Region
Route 53 health checks / ARC [11], [12]	Global data plane	Health-check-driven DNS failover	Not stated	Preferred failover path in AWS DR guidance

Source: AWS documentation as cited; figures are vendor statements, not independent measurements.

4.3 Analytical Model of Multi-Region Workflow Resilience

The presented 'standard' reliability relations are not reported results, but rather used as analytical tools for the reviewed architecture. If a workflow is n steps in software in series with independent service dependencies, then its availability, based on the definition from the dependability literature [48] is given by Eq. (1) where Ai is the steady state availability of the services used by step i.

$$A_{\text{wf}} = \prod_{i=1}^{n} A_i \tag{1}$$

If, for each Region i, the availability of the Region is Ar, then the result is given by Eq. (2), with the shared dependencies having joint availability Ag, Eq. (3) etc. are examples of such dependencies.If each Region i has availability Ar, then the result is given by Eq. (2), where the shared dependencies have joint availability Ag, etc. are examples of such dependencies. This is an upper bound since failures are correlated, gray failures, and imperfect detection [9], [49].

$$A_{\text{sys}} = A_{\text{g}}[1 - (1 - A_{\text{r}})^{k}] \tag{2}$$

It is performed here for illustrative values not measured, as shown in figure 2, for the evaluation in Eq. (2). When Ar = 0.999 and k = 2, there is a cap on availability, which is 0.999999, not Regional redundancy. This is in line with AWS guidelines to de-couple control-planes from recovery paths [11], [12].

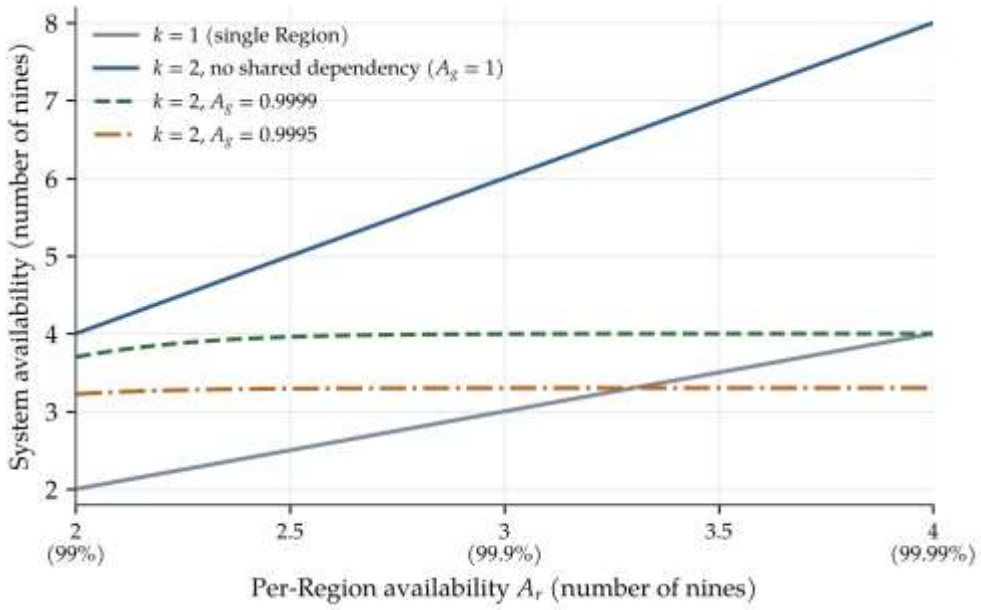


Figure 2. Illustrative evaluation of Eq. (2): system availability versus per-Region availability
Source: Calculated from Eq. (2) with assumed parameter values; the curves are not measured data.

A Region's recovery time after an evacuation can be broken down as shown in Eq. (3): Tdet is the time needed to detect a failure in the Region, Tdec is the time needed to make a decision to fail over to the other Region, Tshift is the time needed to reroute the traffic to the other Region, Tcap is the time needed to acquire capacity in the other Region, and Tres is the time needed to resume in-

flight work. Pre-provisioning [11] has an impact on Tcap, data-plane health checks have an impact on Tshift [12] and gray failures have an impact on Tdet [49]. Existing research does not consider Tres for stateful workflows.

$$\mathrm{RTO} = T_{\mathrm{det}} + T_{\mathrm{dec}} + T_{\mathrm{shift}} + T_{\mathrm{cap}} + T_{\mathrm{res}} \tag{3}$$

The delay for the business data and delay for the last time a workflow progressed are also stored in a persistent store, which can survive a Regional failure; namely Ldata and Lwf respectively. With Step Functions and global tables, Ldata is on the order of a second while with Lwf, the execution state is not replicated and can be as old as the whole execution. The difference leads to recovery-point skew, as described in this review: the surviving data does not contain any information about what (if anything) this surviving orchestrator does not know that was done.

$$\mathrm{RPO}_{\mathrm{eff}} = \max(L_{\mathrm{data}}, L_{\mathrm{wf}}) \tag{4}$$

If a workflow is re-restarted, after performing its side effects at rate λs, the expected number of times that the side effect is performed again after re-restart of the workflow is λs/2, according to Eq. (5). All of those effects should be idempotent, removable and/or compensable [46], [47]. The lower bound on E[R] is also lowered as in log-based designs [13], [14] where the same Regions are deployed and in this design, an additional write per step is added.

$$E[R] = \lambda_{\mathrm{s}} \cdot L_{\mathrm{wf}} \tag{5}$$

4.4 Design Properties for Resilience-Aware Orchestration

Combining Tables 1 and 2 with Eqs. (1)–(5) yields six design properties. They are derived from the evidence, not validated experimentally; Figure 3 shows them in a two-Region deployment.

P1, idempotent effects. Every step that changes external state carries an idempotency key checked in a replicated store, so that at-least-once execution yields exactly-once effects [13].

P2, replicated progress. Workflow checkpoints are written to the same replicated store as business data, which bounds L_{wf} in Eq. (4) [14], [30].

P3, write-mode alignment. Each entity uses a write mode matched to its idempotence; non-idempotent entities have a single home Region and use conditional writes.

P4, data-plane failover. Traffic shifts to pre-provisioned capacity through health-check-driven routing rather than control-plane calls [11], [12].

P5, overload containment. Retries are bounded and back-pressure is propagated, because recovery traffic can sustain metastable failure [10], [22].

P6, continuous verification. Failover is exercised regularly through fault injection, as chaos-engineering practice and AWS reliability guidance recommend [52].

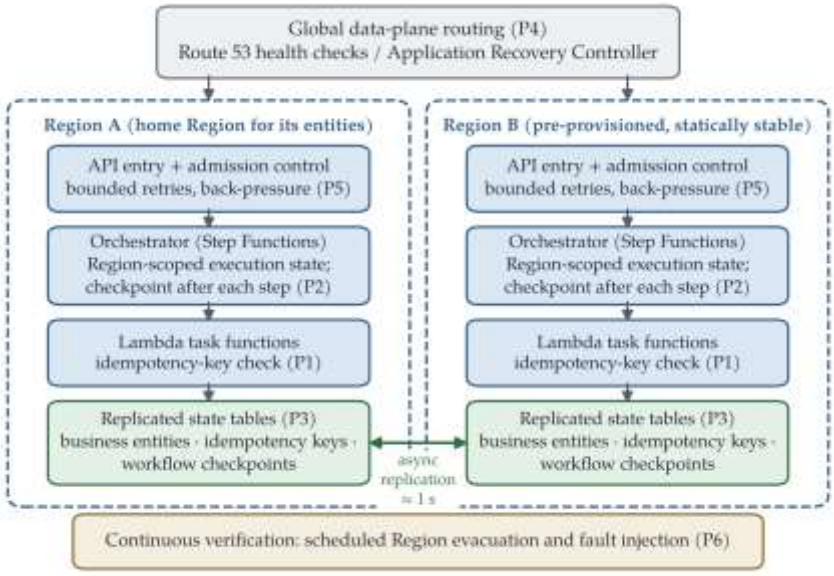


Figure 3. Conceptual two-Region architecture combining design properties P1–P6

Source: Author's synthesis from [11], [12]; conceptual, not an evaluated system.

4.5 Research Gaps and Future Directions

The gaps are highlighted in Table 3. The latter of the reviewed studies [13], [14], [30] is the key component: the reviewed studies are across one failure domain, while the key is across Regions (across exactly-once serverless workflows). Currently, there is no support for workflow-state replication in the academic system nor the AWS orchestrator. The only benchmark that gathers Regional failures and reports the recovery time, recovery point and duplicate effects is [17], [21], [34]. Despite being reported in cloud systems [10], [50] a retry-driven metastability remains unquantified for serverless orchestrators and has only been introduced in the scope of the review period for multi-Region options.

Table 3. Research gaps, supporting evidence, and suggested directions

Gap	Evidence	Consequence	Suggested direction
No multi-Region evaluation of exactly-once workflows	[13], [14], [30]	Behaviour under asynchronous replication and last-writer-wins unknown	Evaluate log-based protocols over global tables with injected Regional faults
Workflow state not replicated	[37]	Unbounded L_{wf}; recovery-point skew	Replicated checkpoints; orchestrator-free designs on replicated logs
No recovery-oriented benchmark	[17], [21], [34]	Resilience claims are not comparable	Benchmark reporting RTO, effective RPO, *E[R]*, and cost
Retry and overload dynamics unquantified	[10], [22], [50]	Failover may trigger retry storms	Model and measure retry amplification in orchestrators

Strong multi-Region consistency is new	[43]	Latency–correctness trade-off for workflow steps unknown	Compare strong and eventual modes for workflow state

Source: Author's synthesis of the reviewed evidence.

4.6 Limitations and Threats to Validity

There was not a thorough search and therefore some work may not have been identified. Abstracts: some quantitative results have been obtained from abstracts and have been represented against every study and should not be combined with each other. The statements in the above AWS documentation might be changed and the figures may be AWS vendor statements and not independently measured. The analytical model is based on the assumption of independence of failures of the Regions and neglect the failover cost resulting in bounding rather than prediction [9].

CONCLUSION

This review was dedicated to the challenges and opportunities of orchestrating stateful enterprise workloads on a serverless platform across various AWS Regions, and the published evidence. For RQ1, it's the execution and transactions of serverless workflows, where it's guaranteed to be executed exactly-once when using protocols that rely on logs, and the overhead is reduced from about twice the unprotected latency, to a variety of designs that are several times faster, but only in one failure domain. AWS second-level latency between Regions for RQ2, but its orchestration service holds the state of a workflow execution between Regions, so the recovery point of an in-flight workflow is based on the last time the state of the workflow was persisted, not just on data replication. The key deficiencies or gaps for RQ3 include the following: Multi-Region evaluations of exactly-once protocols; Replicated workflow state; Measurement of recovery behaviour in the form of benchmarks. These include problems that can be framed with the proposed model and design properties, which are yet to be theoretically tested.

Future work should focus on creating a fault-injection benchmark to evaluate the recovery time, number of points recovered (effective recovery point), side effects caused by forcing a Region out of the system during the execution of the workflows, and cost of the recovery of forcing the Region out after the recovery from globally replicated tables, and comparing the results with the other log-based exactly-once protocols, including the strongly consistent modes.

REFERENCES

[1] E. Jonas, "Cloud programming simplified: A Berkeley view on serverless computing," 2019. doi: 10.48550/arXiv.1902.03383.

[2] P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski, "The rise of serverless computing," *Commun. ACM*, vol. 62, no. 12, pp. 44–54, 2019, doi: 10.1145/3368454.

[3] M. Brooker, M. Danilov, C. Greenwood, and P. Piwonka, "On-demand container loading in AWS Lambda," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2023, pp. 315–328. doi: 10.48550/arXiv.2305.13162.

[4] S. Eismann, "The state of serverless applications: Collection, characterization, and community consensus," *IEEE Trans. Softw. Eng.*, vol. 48, no. 10, pp. 4152–4166, 2022, doi: 10.1109/TSE.2021.3113940.

[5] P. Leitner, E. Wittern, J. Spillner, and W. Hummer, "A mixed-method empirical study of Function-as-a-Service software development in industrial practice," *J. Syst. Softw.*, vol. 149, pp. 340–359, 2019, doi: 10.1016/j.jss.2018.12.013.

[6] J. M. Hellerstein, "Serverless computing: One step forward, two steps back," in *Proc. 9th Biennial Conf. Innovative Data Syst. Res. (CIDR)*, 2019. doi: 10.48550/arXiv.1812.03651.

[7] S. Burckhardt, C. Gillum, D. Justo, K. Kallas, C. McMahon, and C. S. Meiklejohn, "Durable functions: Semantics for

stateful serverless," *Proc. ACM Program. Lang.*, vol. 5, no. OOPSLA, p. 133, 2021, doi: 10.1145/3485510.

[8] P. García López, M. Sánchez-Artigas, G. París, D. Barcelona Pons, Á. Ruiz Ollobarren, and D. Arroyo Pinto, "Comparison of FaaS orchestration systems," in *Proc. IEEE/ACM Int. Conf. Utility Cloud Comput. Companion (UCC Companion)*, 2018. doi: 10.1109/UCC-Companion.2018.00049.

[9] H. S. Gunawi, "Why does the cloud stop computing? Lessons from hundreds of service outages," in *Proc. 7th ACM Symp. Cloud Comput. (SoCC)*, 2016. doi: 10.1145/2987550.2987583.

[10] L. Huang, "Metastable failures in the wild," in *Proc. 16th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2022, pp. 73–90.

[11] Amazon Web Services, "AWS fault isolation boundaries," 2022.

[12] Amazon Web Services, "Disaster recovery of workloads on AWS: Recovery in the cloud," 2022.

[13] H. Zhang, A. Cardoza, P. B. Chen, S. Angel, and V. Liu, "Fault-tolerant and transactional stateful serverless workflows," in *Proc. 14th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2020, pp. 1187–1204. doi: 10.48550/arXiv.2010.06706.

[14] Z. Jia and E. Witchel, "Boki: Stateful serverless computing with shared logs," in *Proc. 28th ACM Symp. Oper. Syst. Principles (SOSP)*, 2021, pp. 691–707. doi: 10.1145/3477132.3483541.

[15] J. Wen, Z. Chen, X. Jin, and X. Liu, "Rise of the planet of serverless computing: A systematic review," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 5, p. 131, 2023, doi: 10.1145/3579643.

[16] H. Shafiei, A. Khonsari, and P. Mousavi, "Serverless computing: A survey of opportunities, challenges, and applications," *ACM Comput. Surv.*, vol. 54, no. 11s, pp. 1–32, 2022, doi: 10.1145/3510611.

[17] J. Scheuner and P. Leitner, "Function-as-a-Service performance evaluation: A multivocal literature review," *J. Syst. Softw.*, vol. 170, p. 110708, 2020, doi: 10.1016/j.jss.2020.110708.

[18] A. Agache, "Firecracker: Lightweight virtualization for serverless applications," in *Proc. 17th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2020, pp. 419–434.

[19] L. Wang, M. Li, Y. Zhang, T. Ristenpart, and M. Swift, "Peeking behind the curtains of serverless platforms," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2018, pp. 133–146.

[20] M. Shahrad, "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2020, pp. 205–218. doi: 10.48550/arXiv.2003.03423.

[21] A. Joosen, "How does it function? Characterizing long-term trends in production serverless workloads," in *Proc. ACM Symp. Cloud Comput. (SoCC)*, 2023, pp. 443–458. doi: 10.1145/3620678.3624783.

[22] A. Sahraei, "XFaaS: Hyperscale and low cost serverless functions at Meta," in *Proc. 29th ACM Symp. Oper. Syst. Principles (SOSP)*, 2023. doi: 10.1145/3600006.3613155.

[23] A. Klimovic, Y. Wang, P. Stuedi, A. Trivedi, J. Pfefferle, and C. Kozyrakis, "Pocket: Elastic ephemeral storage for serverless analytics," in *Proc. 13th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2018, pp. 427–444.

[24] I. E. Akkus, "SAND: Towards high-performance serverless computing," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2018, pp. 923–935.

[25] D. Barcelona-Pons, M. Sánchez-Artigas, G. París, P. Sutra, and P. García-López, "On the FaaS track: Building stateful distributed applications with serverless architectures," in *Proc. 20th Int. Middleware Conf.*, 2019, pp. 41–54. doi: 10.1145/3361525.3361535.

[26] S. Shillaker and P. Pietzuch, "Faasm: Lightweight isolation for efficient stateful serverless computing," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2020, pp. 419–433. doi: 10.48550/arXiv.2002.09344.

[27] V. Sreekanti, "Cloudburst: Stateful functions-as-a-service," *Proc. VLDB Endow.*, vol. 13, no. 11, pp. 2438–2452, 2020, doi: 10.14778/3407790.3407836.

[28] V. Sreekanti, C. Wu, S. Chhatrapati, J. E. Gonzalez, J. M. Hellerstein, and J. M. Faleiro, "A fault-tolerance shim for serverless computing," in *Proc. 15th Eur. Conf. Comput. Syst. (EuroSys)*, 2020, p. 15. doi: 10.1145/3342195.3387535.

[29] Z. Jia and E. Witchel, "Nightcore: Efficient and scalable serverless computing for latency-sensitive, interactive microservices," in *Proc. 26th ACM Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, 2021, pp. 152–166. doi: 10.1145/3445814.3446701.

[30] S. Qi, X. Liu, and X. Jin, "Halfmoon: Log-optimal fault-tolerant stateful serverless computing," in *Proc. 29th ACM Symp. Oper. Syst. Principles (SOSP)*, 2023. doi: 10.1145/3600006.3613154.

[31] W. Zhang, V. Fang, A. Panda, and S. Shenker, "Kappa: A programming framework for serverless computing," in *Proc. 11th ACM Symp. Cloud Comput. (SoCC)*, 2020, pp. 328–343. doi: 10.1145/3419111.3421277.

[32] S. Zhuang, S. Wang, E. Liang, Y. Cheng, and I. Stoica, "ExoFlow: A universal workflow system for exactly-once DAGs," in *Proc. 17th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2023, pp. 269–286.

[33] S. Burckhardt, "Netherite: Efficient execution of serverless workflows," *Proc. VLDB Endow.*, vol. 15, no. 8, pp. 1591–

1604, 2022, doi: 10.14778/3529337.3529344.

[34] J. Wen and Y. Liu, "An empirical study on serverless workflow services," 2021. doi: 10.48550/arXiv.2101.03513.

[35] N. Shahidi, J. R. Gunasekaran, and M. T. Kandemir, "Cross-platform performance evaluation of stateful serverless workflows," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, 2021, pp. 63–73. doi: 10.1109/IISWC53511.2021.00017.

[36] P. García López, A. Arjona, J. Sampé, A. Slominski, and L. Villard, "Triggerflow: Trigger-based orchestration of serverless workflows," in *Proc. 14th ACM Int. Conf. Distrib. Event-Based Syst. (DEBS)*, 2020. doi: 10.1145/3401025.3401731.

[37] D. H. Liu, A. Levy, S. Noghabi, and S. Burckhardt, "Doing more with less: Orchestrating serverless applications without an orchestrator," in *Proc. 20th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2023, pp. 1505–1519.

[38] 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.

[39] D. Abadi, "Consistency tradeoffs in modern distributed database system design: CAP is only part of the story," *Computer (Long. Beach. Calif).*, vol. 45, no. 2, pp. 37–42, 2012, doi: 10.1109/MC.2012.33.

[40] G. DeCandia, "Dynamo: Amazon's highly available key-value store," in *Proc. 21st ACM Symp. Oper. Syst. Principles (SOSP)*, 2007, pp. 205–220. doi: 10.1145/1294261.1294281.

[41] M. Elhemali, "Amazon DynamoDB: A scalable, predictably performant, and fully managed NoSQL database service," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2022, pp. 1037–1048.

[42] A. Verbitski, "Amazon Aurora: Design considerations for high throughput cloud-native relational databases," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2017, pp. 1041–1052. doi: 10.1145/3035918.3056101.

[43] J. C. Corbett, "Spanner: Google's globally distributed database," *ACM Trans. Comput. Syst.*, vol. 31, no. 3, p. 8, 2013, doi: 10.1145/2491245.

[44] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen, "Don't settle for eventual: Scalable causal consistency for wide-area storage with COPS," in *Proc. 23rd ACM Symp. Oper. Syst. Principles (SOSP)*, 2011, pp. 401–416. doi: 10.1145/2043556.2043593.

[45] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "Conflict-free replicated data types," in *Proc. 13th Int. Symp. Stabilization, Safety, Security Distrib. Syst. (SSS)*, 2011, pp. 386–400. doi: 10.1007/978-3-642-24550-3_29.

[46] H. Garcia-Molina and K. Salem, "Sagas," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 1987, pp. 249–259. doi: 10.1145/38713.38742.

[47] P. Helland, "Life beyond distributed transactions: An apostate's opinion," in *Proc. 3rd Biennial Conf. Innovative Data Syst. Res. (CIDR)*, 2007, pp. 132–141.

[48] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Dependable Secur. Comput.*, vol. 1, no. 1, pp. 11–33, 2004, doi: 10.1109/TDSC.2004.2.

[49] P. Huang, "Gray failure: The Achilles' heel of cloud-scale systems," in *Proc. 16th Workshop Hot Topics Oper. Syst. (HotOS)*, 2017, pp. 150–155. doi: 10.1145/3102980.3103005.

[50] N. Bronson, A. Aghayev, A. Charapko, and T. Zhu, "Metastable failures in distributed systems," in *Proc. 18th Workshop Hot Topics Oper. Syst. (HotOS)*, 2021, pp. 221–227. doi: 10.1145/3458336.3465286.

[51] M. Brooker, T. Chen, and F. Ping, "Millions of tiny databases," in *Proc. 17th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2020, pp. 463–478.

[52] A. Basiri, "Chaos engineering," *IEEE Softw.*, vol. 33, no. 3, pp. 35–41, 2016, doi: 10.1109/MS.2016.60.

[53] S. Ristov, S. Pedratscher, and T. Fahringer, "xAFCL: Run scalable function choreographies across multiple FaaS systems," *IEEE Trans. Serv. Comput.*, vol. 16, no. 1, pp. 711–723, 2023, doi: 10.1109/TSC.2021.3128137.

[54] S. Ristov, D. Kimovski, and T. Fahringer, "FaaScinating resilience for serverless function choreographies in federated clouds," *IEEE Trans. Netw. Serv. Manag.*, 2022, doi: 10.1109/TNSM.2022.3162036.

[55] A. F. Baarzi, G. Kesidis, C. Joe-Wong, and M. Shahrad, "On merits and viability of multi-cloud serverless," in *Proc. 12th ACM Symp. Cloud Comput. (SoCC)*, 2021, pp. 600–608. doi: 10.1145/3472883.3487002.