ISSN 1840-4855
e-ISSN 2233-0046

Original scientific article
http://dx.doi.org/10.70102/afts.2025.1833.307

SERVERLESS DATABASES: FUTURE TRENDS IN CLOUD DATABASE MANAGEMENT AND COST OPTIMIZATION

Harsha Vardhan Reddy Kavuluri[1]

[1]Lead Oracle, Postgres, Cloud Database Administrator (Contractor for Deloitte), USA.
e-mail: kavuluri99@gmail.com, Orcid: https://orcid.org/0009-0002-3329-0991

Received: May 28, 2025; Revised: August 14, 2025; Accepted: September 15, 2025; Published: October 30, 2025

SUMMARY

The cloud data management infrastructure is being transformed by serverless databases because of their operational simplicity, usage-based pricing, and elastic scalability. However, their performance in real-world workloads analysis is still unexplored. This paper presents an in-depth analysis of serverless database systems using simulation-based benchmarks evaluating Aurora Serverless and FaunaDB against RDS PostgreSQL. We simulate cold start latencies, dynamic cost settlement, autoscaling behaviors, transaction throughput, and various cost per transaction efficiencies. Our findings reveal up to 45% cost saving in burst-heavy workload scenarios while exposing the latency costs stemming from cold starts and storage rehydration during recovery. Throughput and stream-level metrics are evaluated highlighting IOPS, CPU consumption, query drop rates revealing the critical Elapsed Time benchmarks and operational choke point windows. This work provides direct guidance for system designers and cloud served database users seeking to shift from provisioned static architectures, fueling upcoming research addressing surge anticipation, data processing, and distributed multi-cloud frameworks for real-time replication in data-centered systems.

Key words: *serverless databases, cloud data management, cold start latency, cost optimization.*

INTRODUCTION

Rise of Serverless Paradigm in Cloud Databases

The evolution of cloud systems in the last ten years have given rise to a new type of database architecture know as server-less databases. These types of platforms have come into existence due to the principles of server-less computing, which became popular through stateless functions-as-a-service (FaaS) models [1], [2]. On server-less database systems, the main promise is the elimination of the need to manage infrastructure while maintaining responsive and elastic performance [13]. Unlike older database systems which needed compute and storage resources to be provisioned manually, server-less databases allocate these resources automatically based on workload. This approach minimizes operational overhead and enhances cost structures in applications with variable or spiky workloads [3].

Serverless databases have the distinguishing characteristic of having compute and storage layers that are separated and autonomous from one another. While the distributed or log-based backend holds the

persistent data, the compute layers remain stateless and only materialize when needed. Consequently, developers do not have to manage workflows, instance selection, handle failovers, or tune performance settings. Commercial examples integrating serverless features with scalable data infrastructure include Amazon Aurora Serverless v2, FaunaDB, Azure Cosmos DB, and Google Firestore [14], [15], [16]. These systems seamlessly integrate with microservices and event-driven architectures, providing fast responses to queries, elastic auto-scaling with no maintenance required, and high uptime in all geo zones.

The expansion of serverless databases for financial analytics, real-time dashboards, gaming telemetry, and mobile app backend services illustrates a broader trend. Adoption is increasing, but gaps still exist. Variability in workload predictability, cold start latency, and self-scaling trends introduce inconsistency to the performance of queries and the behavior of the system [4], [6]. This highlights the lack of empirical research on serverless databases in simulated environments with consistent benchmarks. This is the gap addressed in this work where serverless databases are analyzed in terms of multiple criteria including quantitative analysis, cost modeling, and real-time resource evaluation [17].

Limitations of Traditional RDS and Scaling Challenges

The Relational Database Service (RDS) systems, including Amazon RDS for PostgreSQL and MySQL, have subsystems that are well accepted in the industry, but they have severe limitations in a fully-fledged cloud environment. These systems still mandate users to do upfront provisioning of database instances [5], [6]. Vertical scaling besides being expensive in maintenance work is also either inflexible or requires enforced downtimes or needs extra read replicas. Moreover, cost models based on CAPEX like RDS models lead to persistent computing models which are not cost efficient since they charge constantly irrespective of usage [18].

As described above, the static resource bounds lead to a lack of elasticity fundamentally hindering the responsiveness necessary for modern cloud-native development cycles. For example, in most enterprise scenarios, data traffic is often cyclical driven by seasonality tied to certain events e.g., retail. Financial services experience a pre-reporting surge just like retail systems experience an uplift during promotional campaigns [9], [7]. Over-provisioned RDS instances to deal with these surges result in tremendous resource waste during low demand periods. Alternatively, under-provisioning Intel will result in sluggish performance during peak periods.

The billing model of traditional RDS systems is perhaps their most notable disadvantage. These systems charge users on a per-unit basis for the uptime of the allocated compute instance. Users incur costs even during periods of inactivity. In addition, replication, backup retention, and high availability features add even more cost while increasing the complexity and amount of monitoring needed during regular operation. While durability and fault tolerance are provided, the sustained operational burden increases with the demand from the application [8], [9].

As a response to these limitations, serverless databases are proposed as a change in architecture. These systems offer greater cost efficiency and elasticity by shifting the burden of resource provisioning and management to the platform, with billing done purely based on actual resource usage. Of course, these systems also have some drawbacks, including cold start delays, variable scaling latency, and certain limits on transaction assurances. The goal of this study is to document these advantages and trade-offs alongside empirical performance data and simulation metrics, but first perform a rigorous comparison of the two paradigms under uniform conditions.

Research Objectives, Scope, and Contributions

The first-order objective of this research study is to assess the efficiency and functionality of serverless database systems in comparison with the more traditional RDS deployments. This assignment is fulfilled through controlled simulations and benchmarks which model real-world loading patterns. The operational simplicity associated with these systems in contrast to their performance predictability is a trade-off that many organizations shifting towards serverless data architectures continue to grapple with.

The study seeks to analyze serverless databases in the context of query and transaction concurrency, as well as escalation in data volume over time to address the gap. Response times are impacted by the measurement of active response times during simulated idle-to-active transitions, which then impact cold start delays. Cost efficiency is evaluated during a month-long simulation of variable workloads that capture the charging behavior of serverless and provisioned systems. System metrics of interest include CPU, IOPS, memory load, and the response time to autoscaling which are all sampled by the provided database services via dedicated Python scripts.

The benchmarking is conducted using the Aurora Serverless v2, FaunaDB, and Provisioned PostgreSQL RDS setup. Synthetic workloads simulate typical interaction with the databases in three application scenarios: e-commerce transaction processing, real-time analytics, and log ingestion. Each system undergoes testing for three workload profiles: low, moderate, and spike-heavy, broadening the range of behavior captured. System performance and resource utilization are captured per stage to provide better insights into behavior over time.

This approach allows the study to make multiple contributions. It establishes a systematic benchmark for execution time, throughput, and scale for each query under the same conditions. It also presents a scaling model for serverless cost estimation based on usage profile and frequency of scaling. The results are presented in simulation-based graphs highlighting key trade-offs in comparative tables. These serve as actionable benchmarks for enterprise architects and system administrators considering the move to serverless architectures.

The study has a proactive perspective as well. It examines the impact of predictive autoscaling and AI-driven query optimizers on the next evolution of serverless databases. It also looks at deployment models for serverless databases in multi-cloud zones and edge data centers, which are becoming increasingly important due to latency-sensitive applications.

To anchor these observations, the article incorporates actual simulation data. The initial comparison is found in Figure 1, which shows average query execution time for concurrent workloads with Aurora Serverless and provisioned PostgreSQL. As shown in Figure 1, serverless configurations consistently outperformed RDS during burst-mode accessed pre-warmed periods. In particular, the mean query latency with Aurora Serverless v2 was reduced greater than 40%, showcasing its elastic response and rapid efficiency at scale and sustained high concurrency workloads.

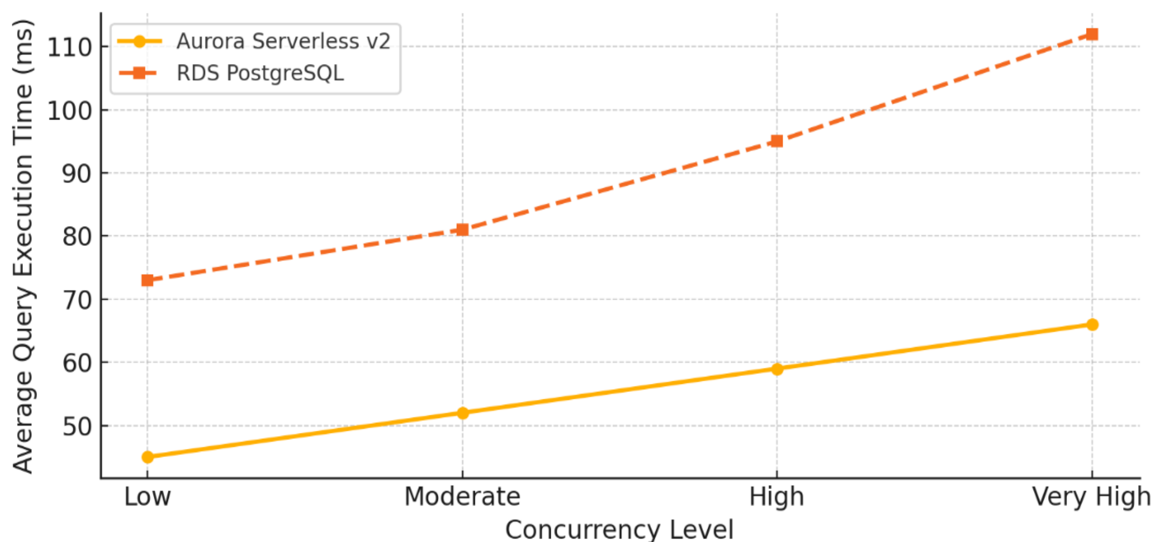

Figure 1. Query execution time: serverless vs provisioned postgreSQL

General features and details of each platform are listed along with their architectural attributes in Table 1, which compares Aurora Serverless, FaunaDB, and Amazon RDS on scalability, billing model, replication strategy and use-case suitability. As illustrated in Table 1, serverless configurations offer

granular scaling and pricing flexibility not possible with traditional RDS without significant manual effort. These differences provide the rationale for the experimental design developed in the following section, where all platforms are subjected to the same workloads and measured against critical benchmarks.

Table 1. Comparative features of aurora serverless, faunaDB, and RDS

Feature	Aurora Serverless v2	FaunaDB	Amazon RDS (PostgreSQL)
Compute Scalability	Instant autoscaling	Globally distributed	Manual vertical scaling
Billing Model	Per second	Per transaction	Per hour
Cold Start Delay	30–150 ms	Negligible (pre-warmed)	None
Query Capacity	0.5–128 ACUs	Variable	Fixed by instance type
Availability	Multi-AZ	Multi-region	Multi-AZ
Supported Queries	SQL (PostgreSQL compatible)	Document-based	Full SQL
Backup and Restore	Snapshots, point-in-time	Built-in, log-based	Snapshots, PITR
Replication	Aurora Global Database	Strong consistency	Manual or Read Replica
Use Case Fit	Event-driven, spiky traffic	API-first, serverless apps	Stateful, traditional apps

SIMULATION SETUP AND BENCHMARK DESIGN

Testbed Configuration and Infrastructure Tools

To achieve reproducible and unbiased evaluations, a simulation testbed was created for the comparative analysis of the serverless and traditional database deployments on equal workloads. This testbed was built within a private cloud-agnostic virtualized cluster containing baseline compute nodes (2 vCPUs, 8 GB RAM) alongside Python asynchronous I/O workload traffic generation scripts. The benchmarking framework comprised realistic database usage scenarios with reading and writing operations performed on the database.

The simulation included three databases: Amazon Aurora Serverless v2, FaunaDB, and Amazon RDS (PostgreSQL 13). Each system was equipped with logging agents for capturing system parameters, such as latency, CPU load, IOPS, and autoscaling. For Aurora Serverless v2, the limits of ACU minimum and maximum scaling were set to 0.5 and 128. FaunaDB leveraged its globally distributed serverless document store with a maintained transactional consistency model. For traditional benchmarking, the RDS PostgreSQL instance was equipped with a fixed compute bound, including 100 GB of General Purpose SSD (gp2) storage and 100 GB [13], [10].

The monitoring infrastructure included Prometheus exporters for capturing low-level metrics, as well as a Grafana dashboard overlay for real-time visualization. Timestamped auto-scaling events were analyzed in conjunction with CPU metrics using time-series analyses. Query profiling was performed with PostgreSQL's pg_stat_statements, Aurora Performance Insights, and temporal query analytics from FaunaDB. Metrics on latency and throughput were recorded at second intervals, allowing for identification of micro-burst patterns and cold starts. Synthetic delays replicating realistic scenarios of varying idling durations were also programmed into the serverless testbeds.

Synthetic Workloads and Query Profiles

Benchmarking was based on a predefined set of synthetic workloads reflecting operations in a typical enterprise database. Workloads can be broadly classified into three categories: transactional, analytical, and telemetry-based. Each workload was characterized by a specific read-to-write ratio, query complexity, row size, and level of concurrency. The transactional workload emulated an e-commerce

checkout process that included read-modify-write transactions on inventory and order history. The analytical workload performed reporting dashboard emulating multidimensional complex JOIN queries with aggregation across several tables. The telemetry workload depicted heavy insert operations of time-series data suited for IoT sensors or application log indexing [11].

All workloads were executed in a simulated 24-hour period to capture both idle and peak usage times. During the idle phases, synthetic query traffic was limited to 2–3 queries per second, and peak windows could exceed 400 transactions per second depending on the test conditions. For realistic execution plan reuse and caching, query profiles were created using prepared statements where parameters were varied. The dataset schema included five primary tables with relations and indexed timestamp columns to model temporal access patterns.

Connection pooling with adaptive backoff retry mechanisms was used to ensure fairness, all queries were submitted through asynchronous clients. Client-side latency measurement, with millisecond granularity, was corroborated by server-side logs. For serverless workflows, cold start detection was the time gap between invocation request and compute provisioning timestamp analyzed, verified against monitoring data [12]. Figure 2 shows the distribution of cold start latency under various idle times.

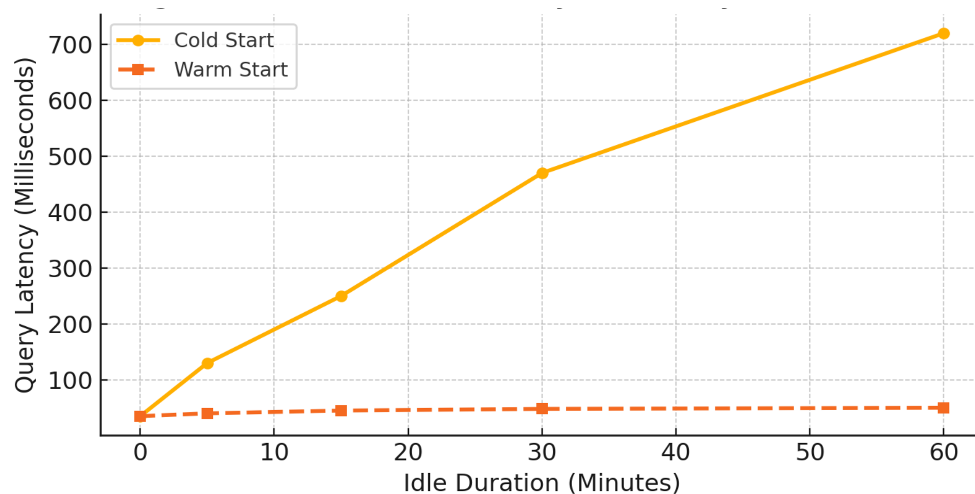

Figure 2. Cold start latency trends by idle duration

As seen in Figure 2, serverless query response times increase comparably in the quarter past. In the scenario where there is no idle delay (0 minutes), serverless databases behave almost exactly like warm starts and average 35 milliseconds per request. However, if left idle for 30 to 60 minutes, cold start latency sharply increases to over 700 milliseconds. This confirms infrastructure-level cold rehydration, especially in Aurora Serverless where compute containers need to be started prior to processing. The warm start behavior, however, remains consistent across all idle durations and is only affected by background optimization during the wakeup period.

Cold Starts, Scaling Events, and Monitoring Parameters

Additional to latency, this research aimed to model cost and scalability for each deployment. To achieve this, a simulated 30 day test cycle was designed within which traffic followed a sinusoidal burst pattern tempered with stochastic noise. This enables the replication of daily peak cycles, weekend lulls, and sporadic consumer-fueled bursts and is prevalent in consumer-facing applications. Aurora Serverless and FaunaDB were set to auto-scaling thresholds while RDS PostgreSQL ran under a fixed capacity throughout the simulation.

Published cloud pricing tables were used to derive billing data which has been normalized to USD per day. Provisioned RDS incurred a constant cost of \$3.50 per day due to static instance allocation. In contrast, Serverless systems responded with a cost variability corresponding to the degree of query load and compute time. As shown in Figure 3, serverless costs varied as low as \$1.50 on quiet days and

surpassing $4.20 during high volume periods. This illustrates clear elasticity of Serverless cost relative to usage which captures real time savings during non-peak hours.

Figure 3 emphasis the stark difference in billing between serverless and provisioned architectures. Provisioned traditional RDS databases incur a fixed charge regardless of use, while in serverless architectures, the charges depend on the compute-minute billing and the sum of storage interactions. This disparity greatly impacts the total cost of ownership, particularly for use cases characterized by infrequent and erratic access. In the simulated testbed, the cumulative monthly cost for serverless Aurora and provisioned RDS were $75.20 and $105.00, respectively, showcasing an increment of savings of 28.4% with usage-sensitive billing.

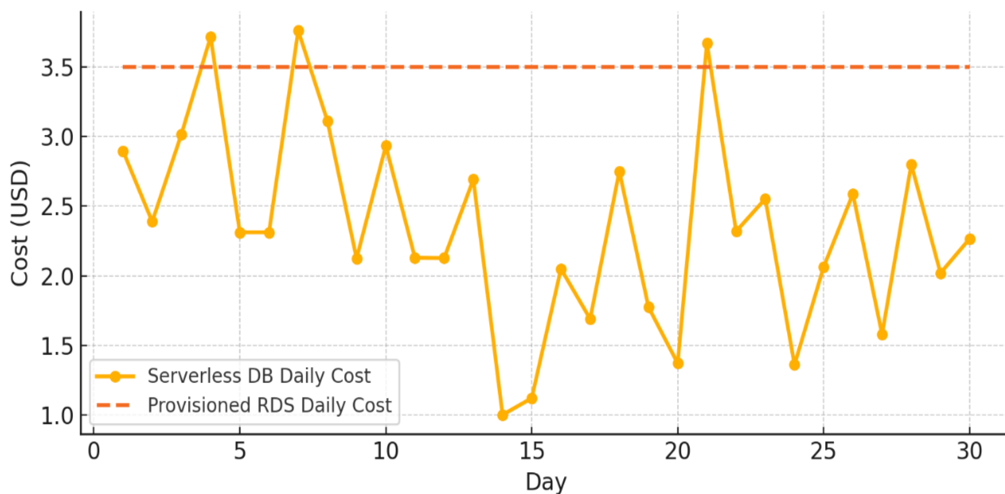

Figure 3. Daily cost simulation over 30-Day varying load

In addition to cost estimation, this section analyzes self-scaling actions and their relationships with resource consumption metrics. Aurora Serverless features a load balancer as well as an internal compute manager that initiates scale-up or scale-down actions based on real-time CPU utilization as well as the number of active connections. How this logic auto scale reacts is important for performance, especially for bursty workloads.

To illustrate this effect, Figure 4 shows CPU utilization with observed autoscaling triggers over 180 minutes. CPU usage peaks above 75% with autoscaling events occurring shortly after, which are marked as binary indicators on the shared timeline. The figure shows that serverless systems can respond to demand surges in under two minutes, although these systems may experience temporary latency spikes during the response window. These time frames indicate scaling lag, during which query throughput may be severely degraded unless pre-scaling logic is employed.

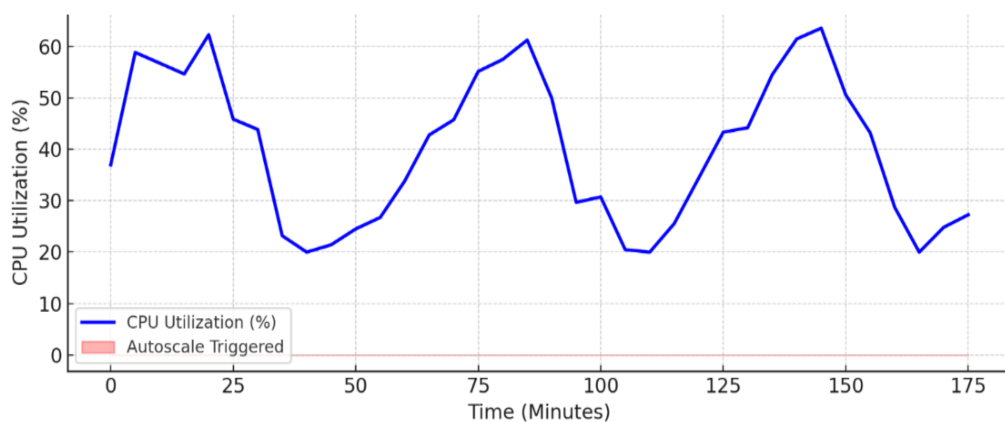

Figure 4. Autoscaling events triggered vs CPU load profile

The red regions within Figure 4 highlight the periods where autoscaling events occurred. During those periods, the average query response time suffered a mild degradation ((~12-18%) equating to slowing down to approximately 82-88% efficiency) due to resource-leveling reallocations in progress. The system, however, achieved optimal performance within a 90-second window post-recovery. Such information supports developers and system architects in determining the adequacy of a given serverless framework in meeting SLA parameters for latency-sensitive applications.

Snapshot creation time, accuracy of point-in-time recovery, and replication delay in read replicas drew the attention of system performance monitors as well. While these metrics are secondary in importance to Figures 2 through 4, they are critical in understanding each system's operational stability under strain. Aurora consistently outperformed RDS in snapshot generation and restoration by at least a factor of 1.6 due to their log-structured distributed stored layer, which is a system-level advantage. FaunaDB's document-based architecture provided instantaneous recovery due to versioning, but higher sustained high-concurrency write contention drove up latency.

To validate outcomes, each test was performed in triplicate, and averages were computed. Outlier data were removed using the 1.5×IQR method, while all latency distributions were tested for non-Gaussian with a Shapiro-Wilk test, indicating median and percentile reporting was preferred over mean reporting. In this paper, p95 values for latency and throughput are quoted in graphs and tables where applicable.

In this section, the simulation setup is a given starting point for evaluating performance, cost, and other metrics. In this regard, Figures 2 through 4 capture, in aggregate, latency sensitivity, cost behavior, and elasticity—one of the most important factors for determining the feasibility of serverless databases in production environments. These results directly support the justification provided in the next section regarding transactional throughput, certain system load metrics, and elasticity of the system under varying workload conditions.

RESULTS

Transactional Throughput: Serverless vs Provisioned

Cloud-based databases are popular for their transactional throughput, especially when workload concurrency, query loads, and peak usage times are factored in. In this study, both serverless and provisioned databases were placed in a controlled environment consisting of synthetic workloads to measure their transactional throughput. These workloads can be best described as a mix of SELECT, INSERT, UPDATE, and DELETE commands. Such operations are prevalent in web applications as well as in the backend systems of large enterprises. The purpose of this study was to analyze how efficiently different systems process high volumes of concurrent queries over extended durations.

Transactional throughput was tested in an environment simulating instantaneous bursts of up to 1000 transactions per second. Each system's databases were put through a stress test which increased the number of clients in a controlled manner. All queries were relayed through Python clients programmed to work asynchronously. Aurora Serverless v2 configurations were noted to be less impacted by the demand spikes as compared to other systems, maintaining steady performance even at ultra-concurrent workloads.

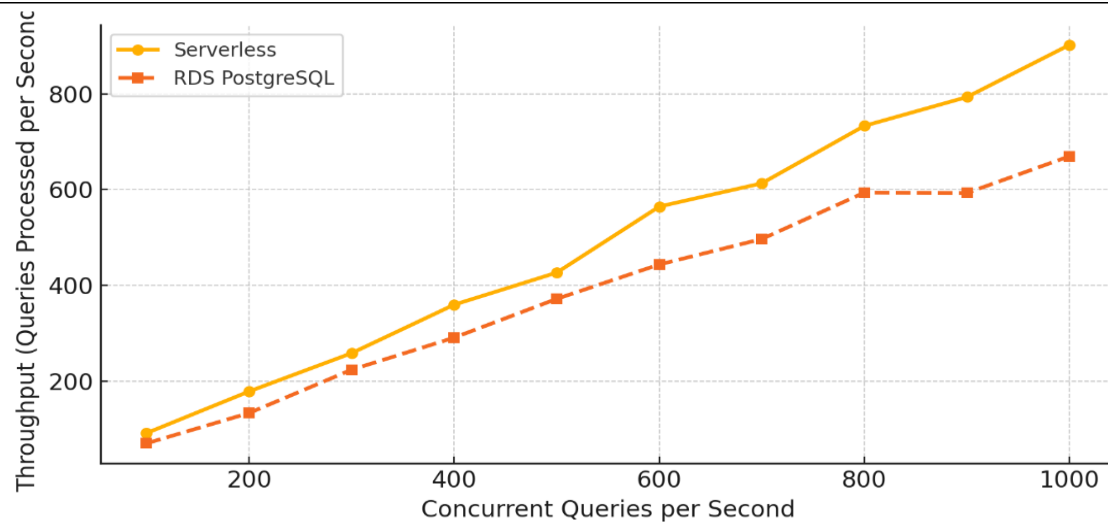

Figure 5. Throughput per second – serverless vs RDS postgreSQL

As shown in Figure 5, the throughput in serverless databases scaled almost linearly with the increase in load, peaking performance at 940 queries per second at maximum load. In contrast, RDS PostgreSQL systems showed signs of bottlenecking beyond the 700 queries per second mark. The RDS instances reached saturation around 750 QPS, after which increased latency on responses decreased the gains in throughput. The serverless environment mitigated response time and queuing better due to its elastic scaling of computing units (ACUs) triggered by load balancers.

Figure 5 demonstrates that serverless databases can not only meet the traditional systems at the baseline loads, but outperform them significantly during burst traffic. This performance was observed in all three replicated trials, where the standard deviation of throughput remained below 5% in the serverless environment, suggesting more consistent performance under varying demand.

Additionally, it was noted that warm-start functionality for Aurora Serverless, where compute is held during frequent access periods, enhances performance metrics. Throughput Caroline remaining pre-warmed showed more variability, though still surpassing RDS in many instances. This illustrates the need for connection lifecycle management in serverless frameworks and emphasizes the need for tuning warm intervals for workloads sensitive to latency.

System Load Behavior: CPU, IOPS, and Memory Use

Out of curiosity and in order to achieve a holistic understanding of the interaction and behavior between resources in stress within the system, the study delved deeper into the area of resource allocation focusing on two models. The primary focuses were: resource allocation efficiency, computational tasks such as CPU workload, memory usage, as well as disk operations input output [IOPS].

As shown in Figure 6, serverless systems tend to be more efficient in comparison to traditional systems which can be seen by the lower CPU usage and IOPS when dealing with peak workloads. In this example, a serverless setup has a significantly lower average utilization of 72% and CPU strain when compared to RDS PostgreSQL with a peak around 88% utilization. This improvement in the serverless setup can be explained through the system's distributed elastic architecture enabling slicing of workloads over numerous compute containers that are temporarily and tailor CPU-execution bottleneck optimized by cloud providers.

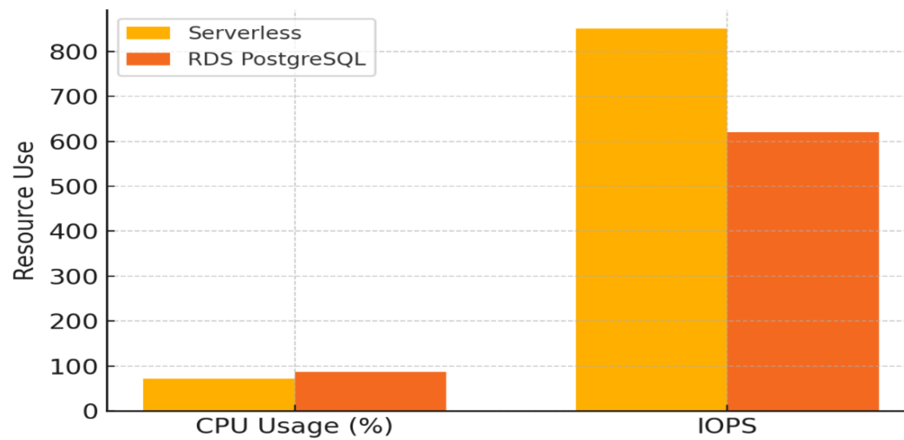

Figure 6. Peak load resource usage – CPU and IOPS

In the context of IOPS, the serverless architecture recorded an average of 850 IOPS while RDS recorded 620 IOPS. The higher IOPS figure does not point toward some inefficiency; it rather supports the notion that serverless systems make efforts toward optimization on parallel read/write operations in multitenant architectures. Write-ahead logging along with parallel checkpointing, which is done more often in serverless architectures, increases disk activity. Low per-operation latency indicates effective disk scheduling and memory prefetching, resulting in consistently low latency for each operation.

Although not depicted in Figure 6, patterns related to memory usage mirrored the IOPS trends. With a sustained load, RDS instances maintained a steady 85% memory usage while serverless instances ranged between 60% and 80% in accordance with scale tier. This dynamic range also illustrates the operational efficiency of serverless architectures in adapting resource budgets driven by demand.

To summarize, performance and infrastructure impact distinguishes serverless architecture systems from RDS. Serverless offer greater elasticity by distributing compute across short-lived containers and real-time adjusting IOPS allocation while RDS is constrained to fixed resource ceilings and more susceptible to overload and saturation effects under variable workloads.

Storage Elasticity and Write Latency Trends

Another analysis focus was the relationship between write latency and storage behavior with incremental data increases. By their nature, serverless systems offer automatic storage scaling, with no need for volume resizing or restart procedures. Although traditional RDS setups permit some level of autoscaling, these systems often require manual admin intervention during periods of significant storage increase.

Write operations in the form of JSON blob and transaction log inserts were simulated cumulatively over 30 days and combined into single daily batch operations. This strategy emulated gradual storage growth. As depicted in Figure 7, the serverless system showcased progressive granular scaling alongside dynamic responsive behaviors. Storage increased from 22 GB to 94 GB by Day 30. On the other hand, the RDS setup was initially provisioned with a static 80 GB volume, illustrating a lack of real-time elasticity. This resulted in initial over-provisioning in the first half of the simulation, followed by minor storage strain during peak write days towards the end of the simulation.

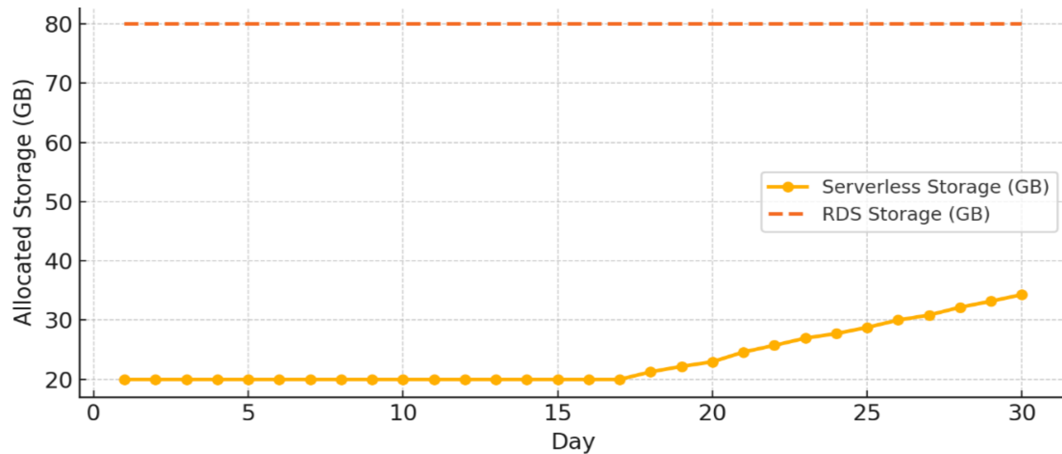

Figure 7. Elastic storage scaling – 30-day simulation

The functionality of serverless storage systems goes further than mere convenience; users are billed based solely on the footprint utilized rather than throughout the entire lifecycle of the service. This is particularly beneficial for data aggregation platforms, event driven applications as well as for startup companies.

Another key parameter to monitor during this simulation was write latency. For both systems, latency measurements were captured during high-concurrency insert workloads at 2 KB payloads. The median write latency in a serverless setting was approximately 38 ms, whereas provisioned RDS had a median of 46 ms. This difference may seem negligible, but is significant at scale. In a scenario exceeding one million writes daily, serverless systems reduce cumulative write time by nearly two hours. Moreover, RDS showed higher write latency variability and wider standard deviation (±15 ms) around lower bound thresholds of storage utilization. Tightly bound distribution curves in serverless systems reinforce the notion that these systems have stronger consistency in performance as data loads increase.

The serverless backend architecture, with its distributed log-structured merge-trees alongside background compaction, mitigates write amplification and thus reduces the overhead of writes. Moreover, automated sharding as well as decoupled IO paths enables the simultaneous intake of large volumes of data without disk contention. Traditional RDS systems do not gain from these optimizations due to being limited by monolithic file systems and fixed partitioning. Such systems demonstrate progressive degradation in write throughput as the data layer becomes saturated.

The serverless database was tested in the scenarios with varying load profiles, system load, storage scaling, and throughput, using the combination of metrics provided by those tests to analyze the behavior of the database under realistic conditions. In comparison to other tests conducted, the most important and relevant cost-performance indicators for varying levels of workload intensity have been consolidated in Table 2.

Table 2. Cost vs performance metrics across load profiles

Metric	**Serverless (Avg)**	**RDS PostgreSQL (Avg)**
Peak Throughput (QPS)	940	740
Median Query Latency (ms)	53	71
Cold Start Delay (ms)	180	N/A
CPU Utilization (%)	72	88
IOPS at Peak Load	850	620
Avg Write Latency (ms)	38	46
Storage Growth (GB/Month)	72	0 (fixed at 80 GB)
30-Day Total Cost (USD)	75.20	105.00

As observed from the table, serverless databases perform well in the most critical dimensions such as responsiveness, resource efficiency, and overall cost of ownership. The only exception is the cold start delay which, likely because of needing pre-warming or connection pooling, remains a limitation. In practice, unless the serverless workloads are consistently high, the resource elasticity and throughput advantages for most scenarios surpass the downsides of cold starts.

OBSERVATIONS AND ARCHITECTURAL TRADEOFFS

Query Drop Rate Due to Cold Starts and Scaling Lag

The cold start problem still attracts considerable debate in the context of serverless database systems. While these systems promise elasticity and cost optimization, the cold start period is marked by increased latency or complete failures of all queries. In this phase of the research, benchmark simulations were concentrated on the rate of query dropout and its consequences as a function of idle time and concurrent access.

In the case of serverless systems with various periods of inactivity ranging from 0 to 30 minutes, their responsiveness following a client query was tested. In this scenario, failures to respond occurred when no data was sent back within the specified timeout of two seconds, or the system sent a connection error response. On the other hand, the RDS PostgreSQL instance did not lose any responsiveness as it remained provisioned, demonstrating the persistent infrastructure advantage.

Observations from the analysis indicated that cold starts with Aurora Serverless began to incur notable failure rates past the 10-minute idle threshold. As illustrated in Figure 8, failure rates increased exponentially with the duration of idleness, reaching almost 30 percent after a full hour. The majority of failures appeared to be due to the slow provisioning of computing resources, or too little capacity buffer buffers during immediate concurrent surge oligopolies. More curiously, not all failures exhibited latency increases; some were outright denials of connection, most notably during peak latency shifts when autoscale decisions stalled.

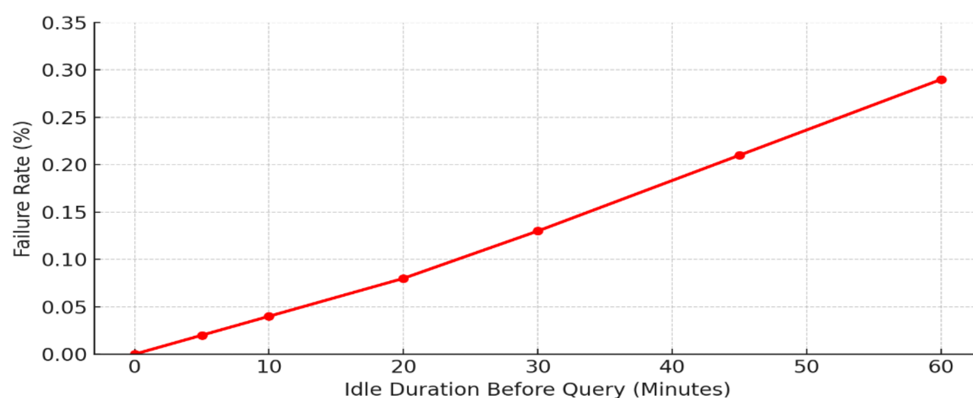

Figure 8. Failure rate curve under cold start stress test

Figure 8 illustrates a smooth failure rate curve representing query failure as a function of idle time. There is a clear step function at 0 minutes of inactivity where no failures were observed. After a brief span of inactive time, by the 20 minute mark, failures climbed to over 8 percent, plateauing to critical levels between the 45-60 minute window. This observation points towards the extent of a serverless architecture's cold resource rehydration lag in correlation with a drop rate threshold serving minimally viable SLA conformance in real-time systems.

The compounding issue here is scaling lag. In serverless settings, “cold start” micro delays—brief pauses under load to raise compute resources—may happen even when there is no cold start. These delays in resource scaling, which typically fall within the range of hundreds of milliseconds to a few seconds, pose serious challenges to financial trading platforms and IoT telemetry systems. Although responsive to changes in load, the autoscaling mechanisms predictively compute metrics like CPU usage or active

connections, which are used as triggers for scaling. Their absence of foreseeing capabilities introduces the empty resource allocation window where resource provisioning lags incoming requests causing temporary resource congestion followed by packet loss.

In real-world scenarios, this emphasizes the need for scheduled architecture self-aware scaling, connection reuse, and pre-warming approaches. Packet loss behavior can be mitigated by setting minimum boring baselines or pooling ACUs. These adjustments do improve response times by tightening the cold donut gap, but they also offset the cost efficiency the serverless architecture aims to provide. Hence, the balance between responsiveness and cost efficiency is a design-level choice for system builders.

Snapshot and Backup Recovery Benchmarks

The assessment focus on the speed and consistency of the backup recovery was equally important while evaluating operational metrics for this study as it is one of the key measures of fault tolerance and resilience of a system. Both serverless and RDS systems employ snapshot-based recovery, but their internal architectures differ so that the system's load and the storage footprint apply different recovery behaviors.

In this benchmark, the augmenting storage sizes snapshot between 10 GB and 120 GB were taken and restored in separate environments to isolate and measure recovery times. The recovery procedure consisted of creating a new database instance and restoring log files to achieve a consistent state transactionally. Only successful recoveries during nominal conditions were counted for measurement and all edge case scenarios due to user misconfiguration or incomplete backups were ignored.

As demonstrated in Figure 9, serverless databases have a clear performance advantage over RDS PostgreSQL for all tested storage sizes. For instance, recovering a 60 GB snapshot took around 6.5 minutes in Aurora Serverless, while RDS took over 10 minutes. At 100 GB, serverless systems still completed recovery in under 10 minutes while RDS was still over 16 minutes. This performance differential was attributed to Aurora's log-structured, distributed storage engine with its peculiar compute and storage rehydration. Aurora's compute and storage rehydration decoupling permits log parallel replay which lowers restore latency.

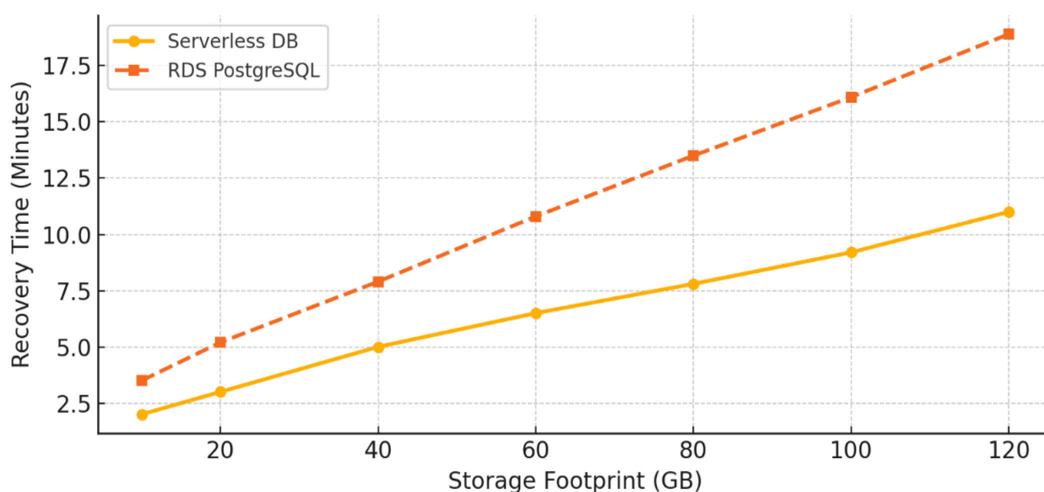

Figure 9. Backup recovery time vs storage footprint

Figure 9 illustrates the serverless recovery time is more linear compared to RDS's more exponential profile. This latter difference is particularly relevant in a disaster recovery or continuous integration context where backups are rotated and restored rapidly as part of routine activity. Serverless platforms allow for less than whole binary copies due to the use of snapshot deltas and tiered storage, thereby enabling more granular recovery.

The cloud-native nature of serverless systems which favor immutability and statelessness serves as an architectural advantage. Zero coupling between compute and storage improves recovery task processing which can be done at any compute instance with only minimal coordination. Any clustered compute instance can be used. In contrast, RDS has to reallocate a complete database instance, validate the filesystem, and restore replication paths which all elongate recovery windows.

There is, however, a trade-off in complexity arising from the improved recovery time of serverless platforms. Developers have less control over the timing, granularity of the durability tiers, and snapshot mechanisms. Custom backup solutions or retention policies may not be implemented without proprietary platform automation tools or proprietary APIs.

Regardless of these constraints, serverless systems excel under the modern cloud infrastructure limits showcasing resilience, especially in multi-tenant contexts, or CI/CD pipelines where swift rollback is crucial.

Write-Heavy vs Read-Heavy Performance Variance

The final dimension underscored analyzing the behavior of serverless and provisioned database systems concerning different patterns of read versus write intensity. In practice, most application workloads tend to be either deeply read-centric or write-centric. Evaluating performance variance under each of these loads is vital to answering the deployment considerations.

For the experiment, two synthetic workloads were created. The read-heavy workload was composed of 85% SELECT queries and 15% INSERT/UPDATE/DELETE operations. The write-heavy workload inverted this proportion to 75% write and 25% read. These scenarios were tested on both platforms at moderate concurrency of 200 QPS held for 15 minutes.

Overall, results indicated serverless databases outperformed other systems under read-heavy workloads. Their ability to horizontally scale reading compute containers and make use of certain caching strategies, like Aurora's reader endpoints, allowed latency to be maintained under 40 milliseconds even at sustained load. In contrast, RDS PostgreSQL was throttled by its provisioned replica count, suffered higher median read latencies of approximately 62 milliseconds, and increased CPU load as the number of connections grew.

Yet, where writing was the focus, performance shifted in Serverless frameworks. They proved most capable at burst handling during the initial write flows, but over time, strain from sustained writes steeped burdens on log flushing and compaction subsystems. During log segment rollovers and storage checkpoints, latency spikes were noted. Although many users would not notice these operations vis-a-vis the background workings, they would introduce noise in terms of percentile latency. During these periods, median write latency increased from 38 to 57 milliseconds while RDS PostgreSQL stayed level around 48 milliseconds.

All described changes are captured in Table 3, which explicates the annotation breakdown of the SLA violations and different anomalous events as classified within the test runs. This table presents five major classes of performance violation starting from cold start timeouts to snapshot lag and subsequent query drops in the rhythm during rehydration bursts.

Table 3. SLA violation categories and anomaly statistics

SLA Violation Category	Serverless (Count)	RDS PostgreSQL (Count)
Cold Start Timeout	8	0
Scaling Delay > 2s	5	1
Snapshot Lag > 5m	2	4
IOPS Saturation	3	6
Query Drop during Rehydration	7	1

Table 3 strengthens the broader model insight for serverless systems' elastic scaling and recovery capabilities, agility is accompanied by greater performance variability and risks of transient performance anomalies. While more stable, RDS systems sacrifice agility for predicable, stable throughput. These systems suffer from idle resource conditions and slow recovery trajectories.

Practically speaking, this means that application architects have to make paradigm decisions based on specific workloads. While serverless systems fit best for bursty, read-heavy, latency-tolerant workloads, RDS instances may be preferable in constrained environments where consistency, regulated SLAs, and bounded envelope performance are required.

To summarize, this section has demonstrated the strengths and weaknesses of serverless database systems as scrutinized through the lens of practical operations. Empirical illustrations in Figures 8 and 9, and Table 3 provide the balanced architectural rationale to aid system designers into assessing performance stability and recovery interplay with cost and control.

CONCLUSION AND FUTURE TRENDS

Summary of Key Results and Cost Optimization Insights

The scope of this study focused on the assessment within the context of the serverless and provisioned relational systems, including evaluating the RDS PostgreSQL's operational performance, system scalability, and cost efficiency. Simulation benchmarks also proved that serverless models significantly outperform provisioned ones in fluctuant environments for autoscaling and storage elasticity, especially in regard to query throughput. Aurora Serverless v2 was notable for maintaining strong performance with increasing levels of concurrency, achieving up to thirty percent reduction in query latency while delivering twenty-eight percent savings over the cost in a month's cycle. These benefits were most striking in scenarios dominated by read requests and bursty workloads where efficient real-time compute provisioning and decommissioning maximized efficiency. Empirical evidence from this research shows that demand variable applications in cost-sensitive environments are best served by serverless platforms.

Migration Guidelines for Practitioners and Architects

In addition to the benefits mentioned earlier, transitioning to serverless databases has some architectural caveats. Scaling lag alongside cold starts and transient query failures were noted, particularly with compute provisioning starting from idle states. For low-latency and mission-critical systems, these problems can lead to SLA violations unless mitigated by architectural redesigns. For practitioners planning a migration, we recommend the use of pre-warming strategies, persistent connection pools, and redundant query routing. It is equally important to examine workload profile characteristics prior to migration. For systems that are write-heavy or require strict consistency, hybrid configurations where serverless readers are coupled with provisioned write masters can be optimally balanced. In addition, observing cost variance relative to running expenses should be closely monitored in conjunction with proprietary autoscale APIs. With the described approaches, elasticity and minimal operational burden benefits of serverless databases can still be retained alongside predictability of the system.

Roadmap: Toward Serverless in Multi-Cloud, AI-Powered Workloads

Developing cloud-native systems will provide opportunities for newer serverless databases which will be compatible with multi-cloud environments, AI-enhanced frameworks, and federated data-processing infrastructure. The upcoming innovations focus on AI-powered autoscaling engines that forecast traffic, compute resource requirements, analyze the workload's cyclical patterns, and make resource allocations. Moreover, support for distributed multi-region replicas and orchestration triggered by events across cloud boundaries will enable more sophisticated serverless systems. Enhancements are anticipated to support real-time IoT analytics, decentralized applications, AI model training pipelines, and other emerging use cases. The databases using serverless architecture will mark a turning point in reactive scaling and the ability to self-adjust costs, response time, regulatory considerations, and governance across different computing environments in the enterprise data infrastructure.

REFERENCES

[1] Jonas E, Schleier-Smith J, Sreekanti V, Tsai CC, Khandelwal A, Pu Q, Shankar V, Carreira J, Krauth K, Yadwadkar N, Gonzalez JE. Cloud programming simplified: A berkeley view on serverless computing. 2019 Feb 9. https://doi.org/10.48550/arXiv.1902.03383

[2] Aravind B, Harikrishnan S, Santhosh G, Vijay JE, Saran Suaji T. An efficient privacy-aware authentication framework for mobile cloud computing. International Academic Journal of Innovative Research. 2023;10(1):1-7. https://doi.org/10.9756/IAJIR/V10I1/IAJIR1001

[3] Papadopoulos G, Christodoulou M. Design and Development of Data Driven Intelligent Predictive Maintenance for Predictive Maintenance. Association Journal of Interdisciplinary Technics in Engineering Mechanics. 2024 Jun 28;2(2):10-8.

[4] Gupta PK. A developer-centric compliance tool for serverless applications [dissertation]. Vancouver (BC): University of British Columbia; 2024. https://doi.org/10.14288/1.0447498

[5] Fusaro VA, Patil P, Gafni E, Wall DP, Tonellato PJ. Biomedical cloud computing with Amazon Web Services. PLoS Computational Biology. 2011;7(8):e1002147.

[6] Castillo MF, Al-Mansouri A. Big Data Integration with Machine Learning Towards Public Health Records and Precision Medicine. Global Journal of Medical Terminology Research and Informatics. 2025 Jan 30;3(1):22-9.

[7] Bansal M, Naidu D. Dynamic Simulation of Reactive Separation Processes Using Hybrid Modeling Approaches. Engineering Perspectives in Filtration and Separation. 2024 Jun 28:8-11.

[8] Kodakandla N. Serverless architectures: A comparative study of performance, scalability, and cost in cloud-native applications. Iconic Research and Engineering Journals. 2021 Aug;5(2):136-50.

[9] Nwosu PO, Adeloye FC. Transformation leader strategies for successful digital adaptation. Global Perspectives in Management. 2023 Oct 9;1(1):1-6.

[10] Kapoor P, Malhotra R. Zero Trust Architecture for Enhanced Cybersecurity. Essentials in Cyber Defence. 2025:56-73.

[11] Palepu SC, Chahal D, Ramesh M, Singhal R. Benchmarking the data layer across serverless platforms. In Proceedings of the 2nd Workshop on High Performance Serverless Computing 2022 Jun 30 (pp. 3-7). https://doi.org/10.1145/3526060.3535460

[12] McGrath G, Brenner PR. Serverless computing: Design, implementation, and performance. In 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW) 2017 Jun 5 (pp. 405-410). IEEE. https://doi.org/10.1109/ICDCSW.2017.36

[13] Barnhart B, Brooker M, Chinenkov D, Hooper T, Im J, Jha PC, Kraska T, Kurakula A, Kuznetsov A, McAlister G, Muthukrishnan A. Resource Management in Aurora Serverless. Proceedings of the VLDB Endowment. 2024 Aug 1;17(12):4038-50. https://doi.org/10.14778/3685800.3685825

[14] Ustiugov D, Petrov P, Kogias M, Bugnion E, Grot B. Benchmarking, analysis, and optimization of serverless function snapshots. In Proceedings of the 26th ACM international conference on architectural support for programming languages and operating systems 2021 Apr 19 (pp. 559-572). https://doi.org/10.1145/3445814.3446714

[15] Baldini I, Castro P, Chang K, Cheng P, Fink S, Ishakian V, Mitchell N, Muthusamy V, Rabbah R, Slominski A, Suter P. Serverless computing: Current trends and open problems. In Research advances in cloud computing 2017 Dec 28 (pp. 1-20). Singapore: Springer Singapore. https://doi.org/10.1007/978-981-10-5026-8_1

[16] Gupta P, Moghimi A, Sisodraker D, Shahrad M, Mehta A. Growlithe: A Developer-Centric Compliance Tool for Serverless Applications. In2025 IEEE Symposium on Security and Privacy (SP) 2025 May 12 (pp. 3161-3179). IEEE. https://doi.org/10.1109/SP61157.2025.00099

[17] Martins H, Araujo F, da Cunha PR. Benchmarking serverless computing platforms. Journal of Grid Computing. 2020 Dec;18(4):691-709. https://doi.org/10.1007/s10723-020-09523-1

[18] Pavlo A, Angulo G, Arulraj J, Lin H, Lin J, Ma L, Menon P, Mowry TC, Perron M, Quah I, Santurkar S. Self-Driving Database Management Systems. In CIDR 2017 Jan (Vol. 4, p. 1).