International Journal of Innovative Computer Science and IT Research

Volume 02, Issue 03 – May-Jun 2026 • E-ISSN: 3067-1108 | P-ISSN: 3067-1094

Serverless Architecture for Event-Driven Information Technology Applications

Prof. Ruben Smit

Professor, Eindhoven University of Technology, Netherlands

Abstract

The increasing demand for responsive, scalable, and cost-efficient digital services has encouraged organizations to adopt event-driven computing models that can dynamically respond to business and system events. Traditional application architectures frequently require continuously running servers and manually managed infrastructure, resulting in resource underutilization, operational complexity, and limited elasticity. Serverless architecture provides an alternative model in which cloud providers manage underlying infrastructure while applications execute through event-triggered functions and managed services. This paper examines serverless architecture for event-driven information technology applications and evaluates its contribution to scalability, responsiveness, resource efficiency, reliability, and operational flexibility. The study develops a conceptual framework integrating event sources, event brokers, serverless functions, managed data services, security mechanisms, monitoring systems, and automated orchestration. The framework emphasizes asynchronous event processing, elastic function execution, stateless service design, and infrastructure abstraction. The analysis evaluates serverless architecture according to scalability, execution responsiveness, cost efficiency, fault tolerance, deployment flexibility, and operational complexity.

Particular attention is given to event processing, function invocation, concurrency management, cold-start behavior, distributed communication, and data consistency. The findings indicate that serverless architectures can effectively support event-driven applications characterized by variable or unpredictable workloads because computational resources can scale dynamically according to incoming events. Organizations can additionally reduce infrastructure management responsibilities because cloud providers handle server provisioning, runtime maintenance, and resource allocation. However, challenges involving cold starts, execution limits, vendor dependency, observability, security, distributed debugging, state management, and unpredictable workloads remain important. The study concludes that serverless architecture is particularly suitable for event-driven information technology applications when workloads are variable, services can be decomposed into independent functions, and asynchronous processing is acceptable. Future research should explore intelligent function scheduling, predictive scaling, multi-cloud serverless portability, energy-efficient execution, and autonomous event-driven application management.

Keywords: Serverless Architecture, Event-Driven Computing, Cloud Computing, Function as a Service, Event Processing, Scalability, Cloud Applications, Serverless Functions, Distributed Systems, Information Technology

International Journal of Innovative Computer Science and IT Research

Volume 02, Issue 03 – May-Jun 2026 • E-ISSN: 3067-1108 | P-ISSN: 3067-1094

1. Introduction

Information technology applications increasingly operate in environments where digital events occur continuously and at highly variable rates. User requests, transactions, sensor messages, database updates, payment notifications, security alerts, and application activities can generate thousands or millions of events within short periods. Conventional server-based architectures generally require infrastructure to remain available regardless of workload intensity. This can result in underutilized resources during low-demand periods and capacity limitations during unexpected workload increases. Event-driven and serverless architectures provide an alternative approach by allowing computational activities to be initiated in response to specific events.

Serverless architecture represents a cloud computing model in which organizations do not directly manage the underlying servers required to execute application functions. Instead, cloud platforms dynamically provision execution environments according to application demand. Developers focus primarily on application logic, while infrastructure provisioning, server maintenance, runtime management, and resource scaling are largely handled by the cloud provider. Function as a Service (FaaS) is a major component of serverless computing, enabling small units of application logic to execute in response to defined triggers. This model can simplify application development and operational management.

Event-driven architecture complements serverless computing because both models emphasize asynchronous interactions and loosely coupled application components. In an event-driven system, an event represents a change or occurrence that can trigger one or more downstream actions. Event producers generate messages, while brokers or event-routing mechanisms deliver them to appropriate consumers. Serverless functions can process these events independently and generate additional events. This creates flexible application workflows in which services can respond dynamically without requiring continuously active application servers.

Scalability is a major advantage of serverless architecture for event-driven applications. Function execution can increase or decrease according to incoming event volume, allowing applications to respond to workload changes without requiring administrators to manually provision servers. This elasticity is especially useful for applications with unpredictable traffic patterns. However, rapid scaling can also create concurrency management challenges, database connection pressure, downstream service overload, and increased execution costs. Effective architectural design must therefore consider the complete event-processing chain rather than function scaling alone.

Serverless architectures also introduce important reliability, security, and operational considerations. Functions are generally designed to be stateless, requiring persistent information to be maintained through external storage or managed databases. Distributed execution makes application behavior more difficult to observe and debug because a single business transaction may involve multiple functions, queues, APIs, and data services. Security additionally requires careful management of function permissions, event sources, secrets, APIs, and data access. These challenges demonstrate that serverless computing reduces infrastructure management but does not eliminate architectural responsibility.

This paper examines serverless architecture for event-driven information technology applications and proposes a conceptual framework integrating event generation, event routing, function execution, managed data services, security, monitoring, and orchestration. The study evaluates serverless

architecture according to scalability, responsiveness, resource efficiency, reliability, deployment flexibility, and operational complexity. It also identifies major implementation challenges and future research opportunities related to intelligent orchestration, multi-cloud portability, predictive scaling, and energy-efficient serverless computing.

2. Objectives

2.1 Primary Objective

The primary objective of this study is to examine the role of serverless architecture in developing scalable, responsive, and reliable event-driven information technology applications. The study focuses on event-triggered function execution, cloud-managed infrastructure, asynchronous communication, automatic scaling, managed data services, security, and monitoring as core elements of serverless application architecture.

2.2 Evaluation of Scalability and Resource Efficiency

The study evaluates how serverless architecture supports dynamic scalability and efficient resource utilization in event-driven applications. It examines automatic function provisioning, concurrency management, workload elasticity, execution-based resource allocation, and event-processing mechanisms to determine how serverless systems can respond effectively to fluctuating application demand.

2.3 Examination of Reliability and Operational Management

Another objective is to examine reliability and operational management requirements in serverless environments. The study considers fault isolation, retry mechanisms, event durability, monitoring, logging, security, function execution controls, and distributed observability as essential mechanisms for maintaining dependable event-driven IT applications.

3. Review of Literature

3.1 Serverless Computing and Function-Based Execution

Serverless computing extends cloud abstraction by allowing developers to execute application logic without directly managing server infrastructure. Function-based execution models enable applications to respond to events such as HTTP requests, database changes, file uploads, scheduled tasks, and message arrivals. Functions are generally short-lived and stateless, allowing cloud platforms to dynamically allocate execution environments. This model can reduce infrastructure management requirements and support highly elastic workloads.

The serverless paradigm has influenced application architecture by encouraging developers to divide functionality into smaller execution units. Such decomposition can improve modularity and independent deployment, although excessive decomposition may increase operational complexity. Serverless systems therefore require careful consideration of function granularity, invocation frequency, execution duration, dependency management, and communication patterns.

3.2 Event-Driven Information Technology Applications

Event-driven architecture enables systems to respond asynchronously to events generated by users, applications, devices, or external systems. Event producers and consumers are often decoupled through messaging infrastructure, allowing producers to continue operating without waiting for downstream processing. This model supports flexible workflows and can improve responsiveness when event volumes fluctuate.

Event-driven applications are particularly suitable for real-time notifications, data processing, automation, monitoring, transaction processing, IoT systems, and integration workflows. Serverless functions can act as event consumers that process incoming messages and trigger subsequent actions. However, event ordering, duplicate messages, retries, delivery guarantees, and eventual consistency must be carefully managed to maintain correct application behavior.

3.3 Scalability, Performance, and Reliability

Serverless platforms provide automatic scaling by creating execution environments according to demand. This capability reduces the need for manual capacity planning and enables applications to handle sudden increases in events. Nevertheless, performance can be influenced by function startup delays, runtime selection, dependency size, memory allocation, network configuration, and downstream service latency. Cold starts are therefore an important performance consideration for latency-sensitive applications.

Reliability in serverless systems depends on the interaction between functions, event brokers, data services, and cloud infrastructure. Retry mechanisms can improve resilience but may also produce duplicate processing if functions are not designed to be idempotent. Dead-letter mechanisms can isolate repeatedly failing events. Observability platforms can monitor function execution, errors, latency, and event flows. Consequently, serverless reliability requires coordinated management across the complete event-processing architecture.

4. Methodology

4.1 Research Design and Serverless Framework Development

This study adopts a conceptual and analytical research methodology for examining serverless architecture in event-driven information technology applications. The framework integrates event sources, event brokers, serverless functions, managed storage, APIs, security mechanisms, monitoring systems, and orchestration components. These elements are analyzed according to their contribution to scalability, responsiveness, reliability, resource efficiency, and operational flexibility. The methodology emphasizes architectural relationships rather than dependence on a specific cloud provider.

The framework begins with event producers generating business or system events. Events are routed through an event broker, message queue, or event-streaming mechanism to appropriate serverless functions. Functions perform application-specific processing and communicate with managed databases, storage services, APIs, or additional event consumers. Monitoring systems record execution behavior and failures, while orchestration mechanisms manage retries, concurrency, and recovery. This architecture creates a loosely coupled processing pipeline capable of responding dynamically to changing event volumes.

4.2 Event Processing and Function Execution

Event processing is evaluated according to event generation, routing, delivery, processing, and completion. The methodology considers both synchronous and asynchronous event patterns, with particular emphasis on asynchronous processing because it reduces direct temporal dependencies between components. Event brokers are evaluated according to durability, delivery behavior, scalability, ordering requirements, and integration capabilities. Function execution is assessed based on invocation frequency, execution duration, concurrency, memory requirements, and dependency complexity.

Serverless functions are analyzed as stateless processing units that perform specific application operations. Persistent state is maintained through external databases, object storage, caches, or other managed services. Functions should be designed to tolerate repeated execution because event systems may retry failed or unacknowledged events. Idempotency, validation, error handling, and controlled retry mechanisms are therefore incorporated into the conceptual framework to improve reliability and prevent incorrect duplicate operations.

4.3 Scalability, Reliability, and Security

Scalability is assessed by examining the ability of serverless functions to respond to changing event volumes without manual infrastructure provisioning. Concurrency controls and throttling mechanisms are considered important for preventing sudden function scaling from overwhelming downstream databases and APIs. Reliability is evaluated through retry strategies, dead-letter handling, redundant services, health monitoring, and failure isolation. These mechanisms collectively determine whether event-driven workflows can continue operating during partial service failures.

Key methodological evaluation points include:

- Event-driven function invocation
- Asynchronous message processing
- Automatic function scaling
- Concurrency and throttling management
- Idempotent function design
- Retry and dead-letter mechanisms
- Managed database and storage integration
- Identity and access management
- Encryption and secret protection
- Centralized logging and monitoring
- Distributed tracing
- Automated deployment and rollback

Security assessment focuses on function-level permissions, API authentication, encrypted communication, secret management, event-source validation, and least-privilege access. These controls are considered necessary because serverless applications may contain large numbers of independently executed functions and event endpoints.

4.4 Analytical Evaluation Criteria

The proposed serverless architecture is evaluated across six dimensions: scalability, performance, reliability, resource efficiency, operational flexibility, and security. Scalability measures the ability to

accommodate changing event volumes. Performance considers execution latency, function startup behavior, and communication overhead. Reliability examines event durability, retries, failure isolation, and recovery. Resource efficiency evaluates whether computing capacity corresponds to actual execution requirements.

Operational flexibility considers deployment independence, infrastructure abstraction, and integration with managed cloud services. Security examines identity, access control, encryption, secrets, and event validation. The analysis recognizes that serverless computing can reduce infrastructure management but introduces additional application-level responsibilities. Therefore, the framework evaluates the complete application lifecycle rather than focusing only on function execution.

5. Analysis

5.1 Components of Serverless Event-Driven Architecture

A serverless event-driven architecture consists of multiple interconnected components that collectively manage event generation, processing, storage, and response. Event sources initiate application workflows, while event brokers route messages to appropriate consumers.

Serverless functions execute application logic and interact with managed databases, storage systems, and external APIs. Monitoring and security mechanisms provide operational and protection capabilities throughout the environment.

Table 1: Core Components of Serverless Event-Driven Applications

Component	Primary Function	Contribution
Event Source	Generates application or system events	Initiates automated workflows
Event Broker	Routes and manages events	Decouples producers and consumers
Serverless Function	Executes application logic	Provides event-triggered processing
API Gateway	Manages application interfaces	Supports controlled external access
Managed Database	Stores application information	Provides persistent state
Object Storage	Stores files and unstructured data	Supports scalable data management
Identity Management	Controls access permissions	Strengthens application security
Monitoring	Tracks execution and failures	Improves operational visibility
Distributed Tracing	Tracks event workflows	Supports troubleshooting
Orchestration	Coordinates processing and recovery	Improves workflow reliability

Interpretation: The table demonstrates that serverless applications depend on an ecosystem of managed services rather than isolated functions. Event brokers provide decoupling, functions perform computation, and managed storage maintains persistent information. Security and observability mechanisms ensure that application operations remain controlled and measurable.

Orchestration additionally supports complex workflows and recovery procedures. Therefore, the effectiveness of a serverless architecture depends on the integration and coordination of all major components.

5.2 Comparative Evaluation of Serverless Characteristics

Serverless architecture provides several benefits for event-driven IT applications, particularly where workloads are variable and processing can be divided into independent functions.

Automatic scaling and infrastructure abstraction reduce operational effort, while event-driven execution can improve resource utilization. Nevertheless, function startup latency, distributed debugging, execution constraints, and cloud-provider dependency can affect application suitability.

Table 2: Evaluation of Serverless Architecture Characteristics

Characteristic	Serverless Capability	Enterprise Benefit	Main Limitation
Scalability	Automatic function scaling	Handles variable workloads	Concurrency can overload dependencies
Resource Efficiency	Execution-based resource use	Reduces idle infrastructure	Cost can increase with high invocation volume
Deployment	Independent function deployment	Supports rapid releases	Large function ecosystems require governance
Reliability	Retries and managed infrastructure	Supports fault recovery	Duplicate processing can occur
Performance	On-demand execution	Effective for event-driven tasks	Cold starts may affect latency
Maintenance	Provider-managed infrastructure	Reduces server administration	Less infrastructure-level control
Flexibility	Event and service integration	Supports diverse workflows	Provider-specific services can create lock-in
Observability	Function-level monitoring	Enables execution analysis	Distributed debugging is complex

Interpretation: The comparison shows that serverless architecture is highly suitable for variable and event-driven workloads because resources can scale according to execution demand. It also reduces infrastructure maintenance responsibilities. However, these benefits are accompanied by limitations involving startup latency, observability, concurrency, cost predictability, and provider dependency.

Therefore, serverless architecture should be selected according to workload characteristics, application latency requirements, execution patterns, and organizational operational capabilities.

5.3 Event Processing and Scalability

Event processing represents the central operational mechanism of a serverless application. Events can originate from user interactions, database updates, file uploads, monitoring systems, IoT devices, or external applications. Event brokers route these events to functions according to predefined rules. As event volumes increase, additional function instances can be created automatically. This provides elastic capacity without requiring administrators to manually provision application servers.

However, unrestricted scaling can produce secondary problems. A sudden event burst may trigger thousands of concurrent function executions, potentially overwhelming databases or external APIs. Concurrency limits, throttling, queue buffering, and controlled backpressure are therefore essential. Serverless scalability must be evaluated across the entire application dependency chain rather than simply measuring how quickly functions can be created.

6. Discussion

6.1 Scalability and Resource Optimization

Serverless architecture can provide significant scalability advantages for event-driven applications because computing resources are activated according to workload demand. Applications that experience irregular traffic can avoid maintaining continuously running servers during periods of low activity. During high-demand periods, serverless platforms can provision additional execution capacity automatically. This model is particularly appropriate for event processing, data transformation, automation, notifications, and other workloads that can be divided into independent processing tasks.

Resource optimization also results from infrastructure abstraction. Organizations do not need to estimate physical server capacity for every possible workload scenario. Instead, the cloud platform manages execution environments and allocates resources dynamically. Nevertheless, organizations must monitor invocation frequency, execution duration, memory consumption, and data-transfer requirements because high-volume workloads can generate significant operating costs. Cost optimization therefore requires continuous application-level monitoring.

6.2 Reliability and Event Processing

Serverless applications can achieve strong reliability through managed infrastructure, redundant execution environments, event durability, automated retries, and failure isolation. If a function execution fails, event-processing systems can retry the operation according to predefined policies. Dead-letter mechanisms can isolate events that repeatedly fail, preventing them from blocking successful processing. These capabilities reduce the amount of infrastructure that application teams must manually manage.

Reliability nevertheless depends on application design. Retrying an operation that has already partially succeeded may result in duplicate transactions if functions are not idempotent. Event ordering may also become difficult when multiple functions process events concurrently. Developers must therefore define appropriate delivery assumptions, idempotency mechanisms, transaction boundaries, and error-handling policies. Serverless reliability is consequently a combination of platform capabilities and application engineering practices.

6.3 Operational Flexibility and Application Development

Serverless architecture can accelerate application development by allowing teams to concentrate on business logic instead of server provisioning and operating-system management. Individual functions can be independently developed, tested, and deployed. Managed cloud services can additionally provide databases, storage, messaging, authentication, monitoring, and workflow capabilities without requiring organizations to build equivalent infrastructure.

However, excessive dependence on managed services can increase architectural complexity and vendor lock-in. A serverless application may become tightly integrated with provider-specific event formats, databases, identity services, and orchestration mechanisms. Migrating such applications to another environment can therefore be difficult. Organizations should balance the productivity advantages of managed services with portability requirements and long-term architectural governance.

7. Challenges

One important challenge is cold-start latency. Serverless platforms may need to initialize an execution environment when a function has not recently been used or when additional capacity is required. This startup process can increase response time for latency-sensitive applications. Runtime selection, dependency size, initialization logic, and workload patterns can influence startup performance. Applications requiring consistently low latency may therefore require specialized optimization techniques or alternative architectural components.

Another challenge is observability and debugging. Serverless applications distribute processing across many short-lived functions, event brokers, databases, APIs, and managed services. A single business operation can therefore generate numerous independent execution records. Identifying the source of a failure requires correlation across logs, metrics, traces, and event identifiers. Organizations must implement structured logging, distributed tracing, centralized monitoring, and consistent correlation mechanisms to maintain visibility.

Security and vendor dependency also represent significant concerns. Each function can require separate permissions and access to different resources, increasing the potential attack surface. Overly broad permissions may allow compromised functions to access unnecessary data or services. Provider-specific APIs and managed services can additionally create lock-in. Organizations should therefore implement least-privilege security, strong identity controls, encryption, secret management, dependency scanning, and portability planning.

8. Future Research

Future research should investigate artificial intelligence for predictive serverless scaling. Machine learning models could analyze event arrival patterns, function execution history, resource consumption, and application dependencies to predict future workload requirements. Predictive mechanisms could potentially reduce cold-start effects and improve resource allocation by preparing execution capacity before demand increases. Research should examine how prediction accuracy can be maintained during sudden or previously unseen workload changes.

Another important research direction is autonomous event-driven application management. Intelligent orchestration systems could automatically identify failed functions, analyze event-processing anomalies, adjust concurrency limits, reroute workloads, and initiate recovery procedures. Such

capabilities could reduce operational workloads while improving application resilience. Future research should address the safety, explainability, and reliability of automated decisions, particularly where autonomous actions influence business transactions.

Multi-cloud serverless computing also requires further investigation. Future architectures could provide portable function execution across multiple cloud providers while maintaining common event interfaces, security policies, monitoring, and data-management mechanisms. Research should examine interoperability standards, state portability, event consistency, workload migration, and cost-aware execution. Such approaches could reduce vendor dependency and allow organizations to dynamically select execution environments according to performance, availability, cost, and regulatory requirements.

9. Conclusion

Serverless architecture provides a powerful approach for developing event-driven information technology applications that require elastic scalability, flexible execution, and reduced infrastructure management. By combining event sources, brokers, serverless functions, managed data services, APIs, security controls, and monitoring mechanisms, organizations can construct applications capable of responding dynamically to changing workloads. This study examined serverless architecture from the perspectives of scalability, performance, reliability, resource efficiency, operational flexibility, and security. The analysis demonstrates that serverless computing is particularly appropriate for workloads characterized by variable event volumes, asynchronous processing, independent application functions, and short or moderate execution requirements. Its infrastructure abstraction enables organizations to focus on application logic while cloud platforms manage much of the underlying execution infrastructure.

The study further demonstrates that serverless scalability must be evaluated across the complete application ecosystem. Automatic function provisioning can rapidly accommodate increasing event volumes, but unrestricted concurrency may place excessive pressure on databases, APIs, and external services. Effective architectures therefore require throttling, queue buffering, controlled concurrency, retry policies, and backpressure mechanisms. Reliability also depends on correct application design because event retries can generate duplicate operations when functions are not idempotent. Cold-start latency, distributed observability, execution limitations, security management, and provider dependency remain important architectural considerations. These limitations demonstrate that serverless computing reduces infrastructure administration but does not eliminate the need for disciplined software architecture, performance engineering, security governance, and operational monitoring.

Overall, serverless architecture should be adopted according to application workload characteristics and organizational objectives rather than treated as a universal replacement for conventional infrastructure. Its strongest benefits occur when applications can be decomposed into event-triggered functions and when workloads fluctuate significantly over time. Successful implementation requires careful event modeling, secure function permissions, reliable data management, comprehensive monitoring, resilient communication, and systematic testing. Future serverless platforms can become more capable through predictive scaling, intelligent orchestration, autonomous recovery, energy-aware scheduling, and multi-cloud interoperability. By combining event-driven design with automated cloud execution and strong governance, organizations can develop information technology

applications that respond efficiently to dynamic workloads while maintaining reliability, security, and operational flexibility.

References

1. Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., Ishakian, V., Mitchell, N., Muthusamy, V., Rabbah, R., Slominski, A., & Suter, P. (2017). Serverless computing: Current trends and open problems. In R. Buyya, B. Dastjerdi, & S. Dustdar (Eds.), *Research advances in cloud computing* (pp. 1–20). Springer.
2. Castro, P., Ishakian, V., Muthusamy, V., & Slominski, A. (2019). The rise of serverless computing. *Communications of the ACM, 62*(12), 44–54. https://doi.org/10.1145/3368454
3. Eismann, S., Bui, L., Grohmann, J., Grohmann, J., Herbst, N., Kounev, S., & Abad, C. L. (2021). Serverless applications: Why, when, and how? *IEEE Software, 38*(1), 32–39. https://doi.org/10.1109/MS.2020.3028302
4. Jonas, E., Schleier-Smith, J., Sreekanti, V., Tsai, C.-C., Khandelwal, A., Pu, Q., Shankar, V., Carreira, J., Krauth, K., Yadwadkar, N. J., Gonzalez, J. E., Popa, R. A., Stoica, I., & Patterson, D. A. (2019). Cloud programming simplified: A Berkeley view on serverless computing. *arXiv preprint arXiv:1902.03383*.
5. Li, Z., Guo, L., Cheng, J., Chen, Q., He, B., & Guo, M. (2022). The serverless computing survey: A technical primer for design architecture. *ACM Computing Surveys, 54*(10s), Article 220, 1–34. https://doi.org/10.1145/3508360
6. Shafiei, H., Khonsari, A., & Mousavi, P. (2022). Serverless computing: A survey of opportunities, challenges, and applications. *ACM Computing Surveys, 54*(11s), Article 239, 1–32. https://doi.org/10.1145/3510611
7. Wen, J., Chen, Z., Jin, X., & Liu, X. (2023). Rise of the planet of serverless computing: A systematic review. *ACM Transactions on Software Engineering and Methodology, 32*(5), Article 115. https://doi.org/10.1145/3579643
8. Manner, J., Rausch, T., Rausch, T., & others. (2023). A structured literature review approach to define serverless computing and Function as a Service. In *2023 IEEE 16th International Conference on Cloud Computing (CLOUD)*. IEEE. https://doi.org/10.1109/CLOUD60044.2023.00068
9. Rad, Z. S., & Ghobaei-Arani, M. (2024). Data pipeline approaches in serverless computing: A taxonomy, review, and research trends. *Journal of Big Data, 11*, Article 82. https://doi.org/10.1186/s40537-024-00939-0
10. Ghorbian, M., Ghobaei-Arani, M., & Asadolahpour-Karimi, R. (2024). Function placement approaches in serverless computing: A survey. *Journal of Systems Architecture, 157*, 103291. https://doi.org/10.1016/j.sysarc.2024.103291
11. Denninnart, M., & others. (2023). Efficiency in the serverless cloud paradigm: A survey on the reusing and approximation aspects. *Software: Practice and Experience*. https://doi.org/10.1002/spe.3233
12. Qu, C., Calheiros, R. N., & Buyya, R. (2018). Auto-scaling web applications in clouds: A taxonomy and survey. *ACM Computing Surveys, 51*(4), Article 73. https://doi.org/10.1145/3148149

13. Oakes, E., Yang, L., Zhou, D., Houck, K., Harter, T., Arpaci-Dusseau, A. C., & Arpaci-Dusseau, R. H. (2018). SOCK: Rapid task provisioning with serverless-optimized containers. In *2018 USENIX Annual Technical Conference* (pp. 57–70). USENIX Association.
14. McGrath, G., & Brenner, P. R. (2017). Serverless computing: Design, implementation, and performance. In *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)* (pp. 405–410). IEEE.
15. Fouladi, S., Romero, F., Iter, D., Li, Q., Seltzer, M., Edwards, E., S. H., & others. (2019). From laptop to Lambda: Outsourcing everyday jobs to the cloud. In *2019 USENIX Annual Technical Conference* (pp. 1–14). USENIX Association.
16. Baldini, I., Cheng, P., Fink, S. J., Mitchell, N., Muthusamy, V., Rabbah, R., Slominski, A., & Suter, P. (2017). The serverless trilemma: Function composition for serverless computing. In *Proceedings of the 2017 ACM SIGPLAN International Conference on Systems, Programming, Languages and Applications: Software for Humanity*.
17. Czentye, J., et al. (2024). Serverless application composition leveraging function fusion: Theory and algorithms. *Future Generation Computer Systems*.
18. Manner, J., Rausch, T., Rausch, T., & others. (2018). Cold start influencing factors in function as a service. In *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion*. IEEE.
19. Akkus, I. E., Chen, R., Rimac, I., Stein, M., Satyanarayanan, M., & others. (2018). SAND: Towards high-performance serverless computing with a sandboxed container architecture. In *2018 USENIX Annual Technical Conference* (pp. 923–935). USENIX Association.
20. Jain, A., James, S., & Chambers, A. (2025). *Serverless computing: A comprehensive survey*. SSRN. https://doi.org/10.2139/ssrn.5099133