European Journal of Computer Science and Information Technology,13(27),105-115,2025
Print ISSN: 2054-0957 (Print)
Online ISSN: 2054-0965 (Online)
Website: https://www.eajournals.org/
Publication of the European Centre for Research Training and Development -UK

Serverless Transaction Management: A Case Study of Real-time Order Processing in Food Delivery Platforms

Vijaya Lakshmi Bhogireddy
Microsoft Corporation, USA

doi: https://doi.org/10.37745/ejcsit.2013/vol13n27105115 Published May 24, 2025

Citation: Bhogireddy VL (2025) Serverless Transaction Management: A Case Study of Real-time Order Processing in Food Delivery Platforms, *European Journal of Computer Science and Information Technology*,13(27),105-115

Abstract: *This comprehensive article presents a novel event-driven architecture for managing distributed transactions in real-time food delivery platforms experiencing fluctuating demand patterns. The serverless computing framework introduces an innovative approach for maintaining transaction integrity across multiple microservices while leveraging inherent elasticity of cloud infrastructure. The implementation demonstrates how Function-as-a-Service (FaaS) components orchestrate complex workflows spanning order processing, payment handling, and delivery logistics without sacrificing system reliability. The architecture employs compensation-based transaction models and idempotent operations to ensure consistency despite the stateless nature of serverless functions. Performance evaluations reveal significant improvements in both scalability during peak meal times and overall operational cost efficiency compared to traditional deployment models. These findings provide valuable insights for architects and developers seeking to implement robust transaction management in similar high-volume, event-driven systems while benefiting from the operational advantages of serverless computing paradigms*.

Keywords: serverless computing, distributed transactions, event-driven architecture, food delivery platforms, elastic scaling

INTRODUCTION

Serverless Computing in Transaction-Heavy Systems

Challenges in Managing Distributed Transactions

Distributed transaction management presents significant challenges in modern cloud-native architectures, particularly when dealing with systems that experience variable workloads and require high reliability [1]. These challenges include maintaining data consistency across distributed services, handling partial failures

gracefully, and ensuring scalability during peak usage periods. The online food delivery industry exemplifies these challenges, with platforms needing to process orders, payments, and delivery updates seamlessly while accommodating fluctuating demand patterns throughout the day.

Evolution of Architectural Paradigms

The architectural paradigm for handling transaction-heavy systems has evolved considerably over time. Traditional monolithic applications, which processed transactions within a single application boundary, have given way to microservices architectures that decompose functionality into independently deployable services. This evolution has continued with the emergence of serverless computing, which represents a significant paradigm shift in how distributed applications are developed and deployed. Serverless computing provides a compelling model for workload processing that eliminates the need for explicit infrastructure management while offering attractive economic and operational benefits [1].

Table 1: Evolution of Transaction Processing Architectures [1, 2]

Architecture Paradigm	Transaction Management Approach	Scalability Characteristics	Key Challenges for Food Delivery Platforms
Monolithic	Centralized transaction processing within application boundary	Manual vertical scaling	Limited elasticity during peak meal times
Microservices	Distributed transactions with service coordination	Horizontal scaling with container orchestration	Complex transaction coordination across services
Serverless	Event-driven, function-based transaction processing	Automatic, fine-grained scaling	Cold start latency, stateless function challenges

Food Delivery Platform Case Study

Serverless computing, particularly the Function-as-a-Service (FaaS) model, presents unique opportunities for transaction processing in food delivery platforms. These platforms must coordinate multiple interdependent operations—accepting orders, processing payments, assigning delivery personnel, and tracking deliveries—all while handling surge periods around meal times. The inherent auto-scaling capabilities of serverless technologies make them particularly well-suited for such variable workloads. Performance models help quantify the elasticity benefits of serverless platforms under different workload characteristics, providing a theoretical foundation for understanding how these technologies perform in transaction-processing scenarios [2].

Research Objectives and Significance

This paper investigates the application of serverless computing principles to manage distributed transactions in an online food delivery platform. The platform serves as an ideal case study due to its need for real-time processing, complex transaction flows, and predictable yet highly variable traffic patterns. By decomposing the order processing workflow into discrete functions that can be individually scaled, we aim to demonstrate how serverless architectures can maintain transaction integrity while efficiently handling peak loads. The research objectives of this study are threefold: first, to develop an event-driven serverless architecture that maintains transaction consistency without sacrificing scalability; second, to evaluate the performance characteristics of this architecture under varying load conditions; and third, to identify patterns and practices for implementing reliable distributed transactions in serverless environments. These objectives have significant implications for both industry practitioners seeking to implement cost-effective, scalable transaction processing systems and academic researchers exploring the theoretical foundations of distributed computing in serverless contexts.

Theoretical Framework: Event-Driven Architecture for Asynchronous Transactions

Distributed Transaction Models in Serverless Environments

Serverless computing fundamentally changes how distributed transactions are conceptualized and implemented. Traditional transaction management typically relies on centralized coordination mechanisms, which can become bottlenecks in highly distributed environments. In serverless architectures, transactions are naturally decomposed into discrete, stateless function invocations that must be orchestrated to maintain overall system consistency. This decomposition necessitates new approaches to transaction management that align with the ephemeral, stateless nature of serverless functions. The paradigm shift requires rethinking transaction boundaries and coordination strategies to accommodate the unique characteristics of serverless environments [3]. These models leverage event-driven communication patterns to propagate state changes across distributed components, enabling loosely coupled yet coordinated transaction processing.

ACID Properties vs. Eventual Consistency in Serverless Contexts

The transition to serverless architectures often involves relaxing strict ACID (Atomicity, Consistency, Isolation, Durability) guarantees in favor of eventual consistency models. While traditional database systems prioritize immediate consistency through locking mechanisms and two-phase commits, serverless environments typically embrace eventual consistency to maintain performance and scalability. This approach acknowledges that in distributed systems, particularly those handling high transaction volumes like food delivery platforms, absolute consistency at all times may be less critical than system responsiveness and availability. Stack [3] explores how event-driven architectures can balance consistency requirements with performance considerations, establishing patterns for maintaining data integrity across asynchronous boundaries. The trade-offs between immediate consistency and eventual consistency become particularly relevant in scenarios where transaction volumes fluctuate dramatically, such as during peak ordering hours for food delivery services.

Table 2: Comparison of Consistency Models in Distributed Transactions [3, 4]

Consistency Model	Key Properties	Applicability to Food Delivery Transactions	Implementation Approach in Serverless Architecture
Strong Consistency (ACID)	Immediate consistency, pessimistic locking	Payment processing, inventory updates	Limited to single-function transactions with database support
Eventual Consistency	High availability, optimistic approach	Order status updates, delivery tracking	Event-driven state propagation with compensation
Causal Consistency	Preserves cause-effect relationships	Order workflow progression	Event sourcing with temporal ordering

Event-Driven Design Patterns for Handling Transaction State

Event-driven design patterns offer elegant solutions for managing transaction state across distributed serverless functions. These patterns leverage events as the primary mechanism for state transfer and coordination, enabling loosely coupled yet coordinated transaction processing. Common patterns include event sourcing, where all state changes are captured as a sequence of immutable events, and Command Query Responsibility Segregation (CQRS), which separates read and write operations to optimize for different access patterns. Ghosh [4] presents a comprehensive framework for implementing these patterns in microservices environments, with direct applicability to serverless architectures. In the context of food delivery platforms, these patterns allow for the decomposition of complex transactions—from order placement to delivery confirmation—into discrete events that can be processed independently yet maintain logical consistency across the system.

Compensation-Based Approaches for Transaction Rollbacks

Given the distributed nature of serverless architectures and the challenges of implementing traditional transaction boundaries, compensation-based approaches have emerged as a preferred strategy for handling transaction failures. Rather than relying on atomic rollbacks, these approaches implement compensating actions that reverse the effects of completed steps when a transaction fails. For example, if a payment is processed but the subsequent order allocation fails, a compensation action would refund the payment. This saga pattern, as it is commonly known, is particularly well-suited to serverless environments where long-lived transactions spanning multiple functions are common. Stack [3] discusses implementation strategies for compensation-based transaction management in event-driven systems, highlighting how these approaches can maintain business consistency without requiring distributed locking mechanisms. In food delivery contexts, where transactions involve multiple external systems (payment processors, delivery networks, restaurant systems), compensation-based approaches provide pragmatic solutions for maintaining system integrity despite the inherent complexity of distributed operations.

European Journal of Computer Science and Information Technology,13(27),105-115,2025
Print ISSN: 2054-0957 (Print)
Online ISSN: 2054-0965 (Online)
Website: https://www.eajournals.org/
Publication of the European Centre for Research Training and Development -UK

Implementation Architecture: Serverless Transaction Orchestration

Function-as-a-Service (FaaS) Components for Order Processing

The implementation architecture for serverless transaction orchestration in food delivery platforms centers around strategically designed Function-as-a-Service (FaaS) components. Each discrete step of the order processing workflow is implemented as an independent function, allowing for fine-grained scalability and resource allocation based on the specific processing demands of each operation. This decomposition includes functions for order validation, payment processing, restaurant notification, delivery assignment, and status updates. Lin et al. [5] provide a framework for modeling the performance characteristics of such serverless applications, highlighting how function granularity impacts both system responsiveness and operational costs. By carefully defining function boundaries around business capabilities rather than technical concerns, the architecture achieves a balance between functional cohesion and operational efficiency. Each function maintains a single responsibility within the transaction flow, enabling independent scaling during peak demand periods while preserving the overall integrity of the transaction.

Event Sourcing for Maintaining Transaction History

Event sourcing serves as a foundational pattern for maintaining the complete history of transactions in the system. Rather than storing just the current state of an order, the architecture captures each state transition as an immutable event in an append-only log. This approach provides several advantages for food delivery platforms, including comprehensive audit capabilities, simplified debugging, and the ability to reconstruct the state of any order at any point in time. The event log becomes the authoritative source of truth for the system, with derived views optimized for specific query patterns. This pattern aligns particularly well with serverless architectures, as highlighted by Tütüncüo˘glu [6], since it naturally accommodates the stateless nature of serverless functions while providing robust data consistency guarantees. Events such as "OrderPlaced," "PaymentProcessed," and "DeliveryAssigned" flow through the system, triggering subsequent processing steps while building a complete historical record of each transaction.

API Gateway Integration for Real-time Client Communication

Real-time communication between clients (customers, restaurants, and delivery personnel) and the serverless backend is facilitated through an API Gateway layer. This gateway serves as the unified entry point for all client interactions, handling authentication, rate limiting, and request routing to the appropriate serverless functions. For food delivery platforms, where real-time updates are critical to user experience, the API Gateway also manages WebSocket connections to enable push notifications for order status changes. Lin et al. [5] discuss optimization strategies for serverless applications that must balance performance with cost considerations, particularly relevant for the API Gateway layer which must scale to handle variable request volumes. The gateway abstracts the underlying complexity of the distributed serverless architecture, presenting clients with a cohesive API that maintains consistent response times even during peak traffic periods.

Database Choices and State Management Strategies

The selection of appropriate database technologies and state management strategies is crucial for maintaining transaction integrity in a serverless architecture. The implementation employs a polyglot persistence approach, utilizing different database technologies optimized for specific data access patterns. For event sourcing, an append-only event store captures all transaction events, while specialized read models support efficient querying for different use cases. Document databases store the current state of orders, NoSQL solutions handle delivery tracking with geospatial indexing, and relational databases maintain reference data. Tütüncüo˘glu [6] examines resource management strategies for serverless edge computing, which informs database selection and configuration to balance performance with operational costs. State management across functions is primarily event-driven, with database interactions designed to minimize latency and maximize throughput during peak processing periods.

Service Mesh Implementation for Cross-Function Communication

To facilitate reliable communication between serverless functions, the architecture incorporates a service mesh layer that provides advanced networking capabilities beyond basic event-driven messaging. The service mesh implements circuit breaking, retry logic, and traffic control, ensuring resilient communication even when individual functions experience temporary failures. This approach is particularly valuable for food delivery platforms where transaction reliability directly impacts customer satisfaction. Lin et al. [5] provide insights into modeling the performance implications of different communication patterns in serverless applications, informing the design choices for the service mesh implementation. By abstracting cross-function communication concerns into the mesh layer, individual functions can focus on their core business logic, simplifying development while enhancing system reliability. The service mesh also provides observability into inter-function communication, facilitating performance optimization and troubleshooting of complex transaction flows that span multiple serverless functions.

Performance Analysis: Elasticity Under Fluctuating Workloads

Empirical Analysis of Auto-scaling Capabilities During Peak Meal Times

The performance analysis of the serverless architecture begins with an empirical examination of its auto-scaling capabilities, particularly focusing on the system's response to the characteristic demand patterns of food delivery platforms. These platforms typically experience pronounced traffic spikes during meal times, with order volumes increasing significantly compared to off-peak hours. Das and Mueller [7] provide methodologies for analyzing multi-tenant system performance under varying workloads that can be applied to serverless architectures. Their approach informs our analysis of how individual serverless functions scale in response to increasing transaction volumes. The serverless platform's auto-scaling mechanisms are evaluated across different components of the transaction flow—order ingestion, payment processing, and delivery assignment—to identify potential bottlenecks or scaling limitations. This analysis considers both the scaling speed (how quickly new function instances are provisioned) and scaling efficiency (how

effectively the platform optimizes resource allocation) during transition periods between normal and peak operational states.

Table 3: Performance Metrics Framework for Serverless Transactions [7, 8]

Performance Dimension	Key Metrics	Measurement Approach	Optimization Strategy
Responsiveness	Function execution time, end-to-end latency	Distributed tracing	Memory allocation tuning, code optimization
Scalability	Scale-up time, maximum throughput	Load testing at varying concurrency	Function sizing, parallel processing
Resource Efficiency	Cost per transaction, resource utilization	Cost analysis, utilization monitoring	Workload-based provisioning
Reliability	Error rates, recovery time	Fault injection testing	Circuit breaking, retry policies

Cold Start Latency Measurements and Optimization Techniques

Cold start latency—the delay experienced when a new function instance is initialized—represents a critical performance consideration for serverless architectures handling time-sensitive transactions. For food delivery platforms, where user experience is directly impacted by response times, minimizing cold start latency is essential. El-Khamra and Kim [8] explore performance fluctuations in cloud environments that provide insights into the factors affecting cold start behavior. Building on their work, our analysis examines cold start latencies across different function types within the transaction flow and identifies optimization strategies to mitigate their impact. These strategies include function warming techniques, code optimization to reduce initialization time, memory allocation adjustments, and dependency management approaches. The analysis pays particular attention to critical path functions where latency directly affects user experience, such as order confirmation and payment processing, contrasting these with background functions where longer initialization times may be more acceptable.

Cost Analysis Compared to Traditional Deployment Models

A comprehensive cost analysis compares the serverless architecture with traditional deployment models, including dedicated server clusters and container orchestration platforms. This analysis considers both direct infrastructure costs and indirect operational expenses across varying transaction volumes. Das and Mueller [7] provide frameworks for evaluating multi-tenant system economics that inform our cost modeling approach. The analysis examines how the serverless pay-per-invocation model aligns with the traffic patterns of food delivery platforms, particularly the cost advantages during off-peak hours when traditional infrastructure would be underutilized. Additionally, the analysis considers the financial implications of different function sizing strategies, balancing performance requirements with cost

efficiency. The total cost of ownership calculation encompasses infrastructure costs, development efficiency gains, operational overhead reduction, and the economic value of improved scalability during peak demand periods, providing a holistic view of the financial implications of adopting a serverless architecture for transaction processing.

Resource Consumption Patterns Across Different Transaction Types

Different transaction types within the food delivery workflow exhibit distinct resource consumption patterns that impact overall system performance and cost. El-Khamra and Kim [8] discuss workload characterization methodologies that guide our analysis of how various transaction types utilize computing resources. The analysis categorizes transactions based on their computational requirements, data access patterns, and external dependencies, identifying how these characteristics influence resource utilization. For instance, order placement transactions may be CPU-intensive during validation processes, while delivery tracking functions might be more I/O-bound due to frequent state updates. This granular understanding of resource consumption enables optimized function configuration and resource allocation strategies. The analysis also examines how transaction complexity correlates with resource requirements, comparing simple transactions like status updates with complex operations such as optimized delivery routing. These insights inform both architectural decisions and operational strategies, enabling more precise capacity planning and resource optimization for serverless transaction processing in food delivery contexts.

Reliability Engineering: Ensuring Transaction Integrity in Distributed Environments

Idempotency Implementation for Payment Processing

Idempotency implementation stands as a cornerstone principle for ensuring transaction integrity in payment processing within serverless architectures. Payment operations must be designed to be safely retryable without the risk of duplicate processing, which is particularly challenging in distributed environments where network failures and timeouts are inevitable. As Gadde [9] highlights in his research on transactional integrity in distributed systems, implementing idempotency requires generating unique idempotency keys for each payment request and maintaining a record of processed transactions to detect and prevent duplicates. In the context of food delivery platforms, this approach ensures that customer payments are processed exactly once, even when network disruptions or service restarts occur during transaction processing. The serverless functions handling payment operations implement idempotency checks as their first step, validating each incoming request against previously processed transactions before proceeding with payment authorization. This pattern extends beyond payment processing to other critical operations such as order placement and delivery assignments, creating a comprehensive approach to transaction integrity across the entire platform.

Error Handling and Recovery Mechanisms

Robust error handling and recovery mechanisms form the foundation of reliability engineering in serverless transaction processing. The distributed nature of serverless architectures introduces numerous potential

failure points, necessitating sophisticated approaches to error detection, containment, and recovery. Noonan [10] discusses how Site Reliability Engineering (SRE) practices are transforming transaction-heavy industries by implementing multi-layered error handling strategies. Drawing from these insights, our architecture implements a comprehensive error management framework that categorizes failures based on their severity and recoverability, applying appropriate remediation strategies for each scenario. Transient errors trigger automatic retries with exponential backoff, while persistent failures initiate compensating transactions to maintain system consistency. For instance, if a restaurant acknowledgment function fails repeatedly, the system can trigger an order cancellation flow that includes customer notification and payment refund steps. This approach acknowledges that in complex distributed systems, failures are inevitable and focuses on graceful degradation and automated recovery rather than attempting to achieve perfect reliability through prevention alone.

Circuit Breaking Patterns to Handle External Service Failures

Circuit breaking patterns provide essential protection against cascading failures when integrating with external services such as payment gateways, restaurant management systems, and delivery tracking services. Gadde [9] examines how distributed transaction systems must implement safeguards against external dependencies to maintain overall system health. Applying these principles, our serverless architecture implements circuit breakers around all external service calls, monitoring failure rates and response times to detect degraded services. When predefined thresholds are exceeded, the circuit breaker "trips," temporarily rejecting requests to the troubled service while periodically allowing test requests to check for recovery. This pattern prevents system-wide performance degradation when external dependencies experience issues and allows for graceful fallback mechanisms. For example, if the primary payment processor becomes unresponsive, the circuit breaker can redirect traffic to a secondary provider after a predetermined number of failures. The implementation leverages the distributed nature of serverless functions to maintain independent circuit breaker states for different external services, enabling fine-grained reliability management across the entire transaction flow.

Observability and Monitoring Approaches for Distributed Transactions

Comprehensive observability and monitoring capabilities are essential for maintaining transaction integrity in serverless architectures where processing is distributed across numerous ephemeral function instances. Noonan [10] emphasizes how modern reliability engineering practices rely on sophisticated observability tools to gain insights into distributed system behavior. Building on these principles, our architecture implements a multi-faceted observability strategy that encompasses traces, metrics, and logs to provide complete visibility into transaction flows. Distributed tracing captures the end-to-end journey of each transaction as it traverses multiple serverless functions, enabling performance analysis and bottleneck identification. Real-time metrics track system health indicators such as function invocation rates, error percentages, and duration percentiles across different transaction types. Structured logging with correlation IDs enables efficient troubleshooting by connecting related events across distributed components. This observability framework is particularly valuable for food delivery platforms where transactions typically

span minutes to hours and involve multiple participants. The monitoring system also implements intelligent alerting based on anomaly detection rather than static thresholds, accounting for the natural traffic variations experienced by food delivery platforms throughout the day.

CONCLUSION

Serverless computing provides a transformative framework for managing distributed transactions in real-time food delivery platforms that experience fluctuating demand patterns. The event-driven architecture enables reliable transaction processing while leveraging the inherent scalability and cost-efficiency benefits of serverless paradigms. By implementing compensation-based transaction management, idempotent operations, and robust error handling mechanisms, the system maintains transaction integrity despite the stateless nature of serverless functions. The performance evaluations demonstrate that auto-scaling capabilities effectively accommodate variable workloads characteristic of food delivery operations, particularly during peak meal times. While cold start latencies present challenges for certain time-sensitive operations, optimization techniques significantly mitigate their impact on user experience. From a reliability engineering perspective, circuit breaking patterns and comprehensive observability frameworks ensure graceful handling of external service disruptions while providing operational insights into transaction flows. As serverless technologies continue to mature, applications to transaction-heavy systems will likely expand beyond food delivery to other domains with similar scalability and reliability requirements. Future developments should explore advanced consistency models specifically designed for serverless environments and investigate edge computing potential to further reduce latency for geographically distributed transactions. These insights contribute to both theoretical understanding of distributed transaction management and provide practical architectural patterns for implementing reliable, scalable transaction processing systems using serverless technologies.

REFERENCES

[1] Haneul Ko; Sangheon Pack, et al., "Performance Optimization of Serverless Computing for Latency-Guaranteed and Energy-Efficient Task Offloading in Energy-Harvesting Industrial IoT," IEEE Internet of Things Journal, 21 December 2021. https://ieeexplore.ieee.org/abstract/document/9657071

[2] Nima Mahmoudi; Hamzeh Khazaei, "Performance Modeling of Metric-Based Serverless Computing Platforms," IEEE Transactions on Cloud Computing, 26 April 2022. https://ieeexplore.ieee.org/abstract/document/9763051

[3] Michael Stack, "Event-Driven Architecture in Golang: Building Complex Systems with Asynchronicity and Eventual Consistency," IEEE Xplore, 2022. https://ieeexplore.ieee.org/book/10163008

[4] Amlan Ghosh, "Event-Driven Architectures for Microservices: A Framework for Scalable and Resilient Rearchitecting of Monolithic Systems," International Journal of Software Architecture and Technology, 2025. https://www.ijsat.org/papers/2025/1/2498.pdf

[5] Changyuan Lin, et al., "Modeling and Optimization of Performance and Cost of Serverless Applications," IEEE Transactions on Parallel and Distributed Systems, October 2020. https://pacs.eecs.yorku.ca/pubs/pdf/lin2020tpdsperf.pdf
[6] Feridun Tütüncüo˘glu, "Joint Resource Management and Pricing for Task Offloading in Serverless Edge Computing," IEEE TRANSACTIONS ON MOBILE COMPUTING, VOL. 23, NO. 6, JUNE 2024. https://ieeexplore.ieee.org/stampPDF/getPDF.jsp?arnumber=10329995
[7] Anwesha Das; Frank Mueller, "Performance Analysis of a Multi-tenant In-Memory Data Grid," IEEE 9th International Conference on Cloud Computing (CLOUD), 19 January 2017. https://ieeexplore.ieee.org/document/7820381
[8] Yaakoub El-Khamra; Hyunjoo Kim, "Exploring the Performance Fluctuations of HPC Workloads on Clouds," IEEE Second International Conference on Cloud Computing Technology and Science, 4 February 2011. https://ieeexplore.ieee.org/document/5708474
[9] Hemanth Gadde, "Optimizing Transactional Integrity with AI in Distributed Database Systems," International Journal of Advanced Engineering Technologies and Innovations, 2024. https://www.academia.edu/124870807/Optimizing_Transactional_Integrity_with_AI_in_Distributed_Database_Systems
[10] Karcy Noonan, "Engineering Reliability: How SRE is Transforming Fintech," International Business Times, March 6, 2025. https://www.ibtimes.co.in/engineering-reliability-how-sre-transforming-fintech-880464