

Real-Time Flight Commerce at Scale: Architectural Patterns for Low-Latency, High-Reliability Airline Integrations

Ravi Teja Thutari

Independent Researcher, USA

raviteja.thutari@gmail.com

<https://orcid.org/0009-0007-3839-2846>

Seshendranath Balla Venkata

University of Central Missouri, Warrensburg, MO, USA

seshendra.ballaven@gmail.com

Author ID: 493890588655356

Suresh Kurapati

Independent Researcher, USA

Sureshkurapati12345@gmail.com

Abstract

The global airline distribution ecosystem processes over \$800 billion in annual transactions, connecting hundreds of airlines with thousands of travel agencies, online travel agents (OTAs), and metasearch platforms through a complex web of legacy and modern distribution systems. As consumer expectations shift toward real-time pricing, instant booking confirmation, and seamless ancillary purchasing, flight commerce platforms face unprecedented challenges in delivering sub-second response times while maintaining transactional integrity across heterogeneous airline systems. This paper presents comprehensive architectural patterns for building high-performance flight commerce platforms that achieve 99.99% availability and sub-500 millisecond end-to-end latency at scale. We introduce a multi-tier caching architecture that reduces airline system load by 78% while maintaining fare accuracy within configurable staleness bounds. Our distributed transaction coordination framework ensures booking consistency across airline reservation systems exhibiting diverse reliability characteristics, achieving successful booking rates exceeding 99.7% even when individual airline endpoints experience degraded performance. Through extensive production deployment serving 2.3 billion annual search requests and 180 million bookings across 450 airline integrations, we demonstrate that our approach reduces average search latency by 64% and booking failures by 43% compared to conventional integration patterns. We further present novel techniques for predictive content refresh, intelligent request routing, and graceful degradation that maintain service quality during airline system outages and traffic spikes. Our findings provide actionable guidance for architects building mission-critical travel commerce systems.

Keywords: Flight commerce, airline distribution, GDS integration, low-latency systems, distributed transactions, caching strategies, NDC, high availability, travel technology

1. Introduction

The airline industry operates one of the world's most complex real-time commerce ecosystems, facilitating the sale and distribution of billions of passenger journeys annually [1]. This ecosystem has evolved over six decades from proprietary airline reservation systems to Global Distribution Systems (GDS) and, more recently, to direct connect and New Distribution Capability (NDC) architectures. Today's flight commerce platforms must navigate this heterogeneous landscape, integrating with legacy EDIFACT-based GDS connections, modern REST/JSON NDC APIs, and proprietary airline interfaces—each exhibiting distinct performance characteristics, reliability patterns, and content capabilities.

Consumer expectations have fundamentally transformed the performance requirements for flight commerce systems. Metasearch platforms and OTAs compete on response speed, with empirical studies indicating that even modest increases in latency can measurably reduce user engagement and conversion rates [2]. Users expect instant availability confirmation, real-time seat maps, and immediate booking acknowledgment—capabilities that strain traditional airline distribution infrastructure designed for batch processing and human-speed interactions. Simultaneously, the rise of ancillary revenue (now exceeding \$100 billion annually industry-wide) demands real-time access to baggage options, seat selections, and bundled offers that were historically unavailable through distribution channels.

Building reliable flight commerce systems presents unique technical challenges distinct from conventional e-commerce. Airline inventory is perishable (unsold seats have zero value post-departure) and highly contested (popular flights may receive thousands of simultaneous booking attempts). Fare rules exhibit extraordinary complexity, with individual itineraries potentially subject to dozens of combinable fare components, each with distinct validity conditions, change penalties, and combinability restrictions. Regulatory requirements mandate specific data retention, passenger information handling, and financial settlement procedures that constrain architectural choices.

This paper addresses these challenges through a comprehensive architectural framework developed and refined through production operation at scale. Our contributions include: (1) A multi-tier caching architecture with intelligent invalidation that balances content freshness against system load and latency requirements; (2) A distributed transaction coordination framework that ensures booking consistency across unreliable airline endpoints; (3) Predictive content refresh mechanisms that anticipate demand and pre-warm caches for high-value content; (4) Adaptive request routing strategies that optimize for latency, cost, and reliability across heterogeneous distribution channels; (5) Graceful degradation patterns that maintain partial service availability during upstream failures; and (6) Extensive empirical evaluation demonstrating significant improvements in latency, reliability, and operational efficiency.

2. Background and Related Work

2.1 Airline Distribution Ecosystem

The airline distribution landscape comprises multiple interconnected systems with distinct historical origins and technical characteristics. Global Distribution Systems (Amadeus, Sabre, Travelport) emerged in the 1960s–1970s as centralized platforms aggregating airline inventory for travel agency access [3]. These systems communicate using EDIFACT messaging standards and proprietary terminal command interfaces,

optimized for reliability over performance. Direct connect integrations bypass GDS intermediaries, enabling airlines to offer richer content and capture distribution cost savings, but require point-to-point technical integration with each distribution partner.

The IATA New Distribution Capability (NDC) standard, introduced in 2012 and achieving significant adoption since 2018, defines XML/JSON-based APIs for airline retailing [4]. NDC enables airlines to distribute rich offers including ancillaries, branded fares, and dynamic pricing through standardized interfaces. However, NDC implementations vary significantly across airlines in terms of content availability, API performance, and operational reliability, creating integration complexity for aggregators. Low-Cost Carrier (LCC) APIs represent another integration category, with budget airlines typically offering proprietary REST APIs optimized for their specific business models.

2.2 Performance Optimization in Travel Systems

Prior research on travel system optimization has addressed specific aspects of the performance challenge. Caching strategies for airline content have been explored in the context of GDS environments, with approaches ranging from simple TTL-based expiration to more sophisticated invalidation based on booking activity patterns [5]. Query optimization techniques specific to flight search have examined index structures for temporal-geographic queries and parallel search execution strategies [6]. Revenue management literature addresses the dynamic pricing aspects that influence content volatility and caching feasibility [7].

2.3 Distributed Transaction Patterns

The challenge of coordinating transactions across heterogeneous systems has been extensively studied in distributed systems literature. The Saga pattern [8] provides a framework for managing long-running transactions through compensating actions, applicable to multi-segment bookings spanning multiple airlines. Two-phase commit protocols ensure atomicity but require participant cooperation that airline systems may not support [9]. Foundational work on idempotence in distributed systems [10] informs our approach to handling duplicate booking attempts and network failures during critical transaction phases.

2.4 High-Availability System Design

Principles for building highly available systems have been established through industry experience at scale. Circuit breaker patterns [11] prevent cascade failures when dependent services become unresponsive. Bulkhead isolation contains failures within bounded contexts. Adaptive load shedding prioritizes critical requests during overload [12]. Our work extends these patterns to the specific requirements of airline integration, where upstream systems exhibit diverse failure modes and recovery characteristics.

3. System Architecture Overview

Our flight commerce platform employs a layered architecture designed to isolate complexity, enable independent scaling, and provide multiple levels of fault tolerance. Figure 1 illustrates the high-level system structure comprising five primary layers: the API Gateway Layer, the Commerce Orchestration Layer, the Content Aggregation Layer, the Airline Integration Layer, and the Data Platform Layer.

3.1 API Gateway Layer

The API Gateway Layer serves as the entry point for all client requests, implementing authentication, rate limiting, request validation, and protocol translation. We deploy gateway instances across multiple geographic regions (North America, Europe, Asia-Pacific) to minimize network latency for global clients. Each regional deployment maintains independent capacity, preventing regional failures from affecting global availability. The gateway implements adaptive rate limiting that adjusts thresholds based on downstream system health, protecting airline integrations from traffic spikes that could trigger protective throttling.

Request routing at the gateway layer considers multiple factors including client geography, requested airlines, current system load, and A/B test assignments. For search requests, the gateway performs request normalization—standardizing date formats, airport codes, and passenger configurations—before forwarding to the orchestration layer. For booking requests, the gateway implements idempotency checks using client-provided request identifiers, preventing duplicate bookings from network retries.

3.2 Commerce Orchestration Layer

The Commerce Orchestration Layer implements the business logic for flight shopping, booking, and post-booking operations. For shopping flows, the orchestrator decomposes complex itinerary requests into component queries, dispatches to appropriate content sources in parallel, and assembles unified responses with normalized pricing and availability. The orchestrator maintains session state for multi-step booking flows, enabling price guarantee enforcement and shopping cart functionality.

Booking orchestration implements the Saga pattern for multi-airline itineraries, coordinating sequential booking attempts with compensating cancellation actions if any segment fails. The orchestrator maintains a durable workflow state machine that can recover from process failures and resume incomplete transactions. Timeout management ensures that partially completed bookings are either completed or rolled back within defined SLA windows, preventing inventory from being held indefinitely in ambiguous states.

3.3 Content Aggregation Layer

The Content Aggregation Layer maintains cached airline content and implements intelligent refresh strategies. We distinguish three content categories with distinct caching characteristics: (1) Static reference data (airport information, aircraft configurations, airline profiles) cached indefinitely with manual invalidation; (2) Semi-static content (fare rules, route schedules, ancillary catalogs) refreshed daily or upon detected changes; and (3) Dynamic content (availability, pricing, seat maps) cached with short TTLs and proactive invalidation.

The aggregation layer implements a hierarchical cache structure. L1 caches (in-process, sub-millisecond access) store hot content for immediate serving. L2 caches (Redis Cluster, single-digit millisecond access) provide shared storage across service instances. L3 represents origin airline systems accessed on cache miss. Cache population follows a pull-through model for on-demand content and a push model for predictively refreshed high-value content.

3.4 Airline Integration Layer

The Airline Integration Layer encapsulates the complexity of heterogeneous airline connectivity. Each airline integration is implemented as an isolated adapter that translates between our normalized internal model and airline-specific protocols and data formats. Adapters handle connection pooling, authentication token management, request serialization, response parsing, and error classification specific to each airline's technical requirements.

The integration layer implements per-airline circuit breakers that track error rates and latency distributions, automatically reducing traffic to degraded airlines to prevent cascade failures. Retry policies are calibrated per-airline based on observed error characteristics—some airlines benefit from immediate retry while others require backoff to avoid triggering rate limits. Request queuing with priority levels ensures that booking requests (revenue-generating) receive preference over shopping requests (speculative) during capacity constraints.

3.5 Data Platform Layer

The Data Platform Layer provides analytics, machine learning, and operational intelligence capabilities. Event streaming through Apache Kafka captures all system interactions for real-time analytics and offline analysis. A feature store maintains computed attributes used by ML models including airline reliability scores, route popularity metrics, and demand forecasts. The analytics pipeline generates operational dashboards, alerting triggers, and business intelligence reports consumed by operations and commercial teams.

4. Multi-Tier Caching Architecture

4.1 Cache Strategy Design

Effective caching in flight commerce must balance multiple competing objectives: minimizing latency through cache hits, maintaining acceptable content freshness for pricing accuracy, reducing load on airline systems to avoid throttling, and controlling infrastructure costs. Our approach employs differentiated caching strategies based on content characteristics, query patterns, and business value.

For availability and pricing content, we implement adaptive TTL policies that consider route characteristics and temporal factors. High-frequency routes with volatile inventory (e.g., business shuttle routes) receive shorter TTLs (30–60 seconds) while stable leisure routes tolerate longer caching (5–15 minutes). TTLs compress during peak booking periods when inventory changes rapidly and extend during off-peak hours. Machine learning models predict optimal TTLs based on historical cache hit rates and pricing accuracy measurements.

4.2 Intelligent Cache Invalidation

Beyond TTL-based expiration, we implement event-driven invalidation triggered by signals indicating content staleness. Booking confirmations immediately invalidate cached availability for affected flights, as inventory has definitively changed. Schedule change notifications from airlines trigger invalidation of affected route/date combinations. Fare filing updates, received through industry data feeds, invalidate pricing caches for modified fare classes.

Probabilistic invalidation addresses scenarios where definitive staleness signals are unavailable. When search results from cache differ significantly from a sampled live query, we increase invalidation probability for related cache entries. This approach catches gradual pricing drift without requiring exhaustive live validation. The sampling rate adapts based on observed divergence—stable content receives infrequent sampling while volatile content triggers more frequent validation.

4.3 Predictive Content Refresh

Reactive caching alone cannot achieve optimal latency for popular content—the first request after expiration incurs full origin latency. Our predictive refresh system anticipates demand and proactively populates caches before requests arrive. Demand prediction models, trained on historical search patterns, identify high-probability queries based on factors including time of day, day of week, upcoming holidays, and trending destinations from external signals.

The refresh scheduler prioritizes content based on predicted request probability, content age, and airline system capacity. During airline system maintenance windows or degraded performance periods, refresh jobs are throttled to avoid contributing to system stress. Refresh effectiveness is continuously measured through cache hit rate improvements and latency impact, with model parameters adjusted based on observed performance.

5. Distributed Transaction Coordination

5.1 Booking Transaction Model

Flight bookings present distinctive transaction challenges compared to conventional e-commerce. A single booking may span multiple airlines (interline or codeshare itineraries), require coordination with payment processors, and trigger downstream fulfillment actions (ticket issuance, loyalty credit, ancillary delivery). Airline reservation systems do not support distributed transaction protocols, requiring application-level coordination to ensure consistency.

We model bookings as sagas with explicit compensation actions. Each booking step (segment reservation, payment authorization, ticket issuance) records its completion status in a durable state store. If any step fails, the saga executor triggers compensating actions for previously completed steps—canceling reservations, voiding payment authorizations, refunding charges. The state machine handles partial failures, timeouts, and ambiguous responses that are common in airline integrations.

5.2 Handling Ambiguous Responses

Airline systems frequently return ambiguous responses—timeouts, generic errors, or success responses for failed operations—that complicate transaction management. Our approach implements explicit status verification for critical operations. After a booking request returns (whether success, failure, or timeout), we issue a separate status query to confirm the actual system state. This verification catches cases where the booking succeeded despite an error response, or failed despite apparent success.

For truly ambiguous states where verification is inconclusive, we implement a reconciliation process that compares our booking records against airline confirmation data received through post-booking feeds.

Discrepancies trigger alerts for manual review and automated correction where possible. This defense-in-depth approach ensures that booking state eventually converges to consistency even when real-time verification fails.

5.3 Idempotency and Duplicate Prevention

Network failures and client retries create risk of duplicate bookings—a costly error that degrades customer experience and complicates reconciliation. We implement idempotency at multiple levels. Client-facing APIs require idempotency keys that uniquely identify booking attempts; repeated requests with the same key return cached results rather than creating new bookings. At the airline integration level, adapters generate deterministic request identifiers that enable detection of duplicate submissions even when airline systems lack native idempotency support.

Duplicate detection employs probabilistic data structures (Bloom filters) for fast negative lookups, with positive matches confirmed against persistent storage. The detection window extends beyond typical retry intervals to catch delayed duplicates from queued requests or client-side bugs. Detected duplicates are logged for pattern analysis that informs improvements to retry policies and client guidance.

6. Adaptive Request Routing

6.1 Multi-Source Content Strategy

Many airlines offer content through multiple distribution channels—GDS, NDC, and direct APIs—each with distinct characteristics. GDS connections provide broad content with consistent reliability but may lack ancillaries and incur per-transaction fees. NDC channels offer richer content and potentially lower costs but exhibit more variable performance. Direct LCC APIs provide the most complete content for budget carriers but require individual integration investment.

Our routing layer dynamically selects content sources based on multiple factors: content requirements (does the request need ancillaries?), current channel performance (latency percentiles, error rates), cost considerations (GDS fees vs. NDC savings), and capacity constraints (airline-imposed rate limits). The routing decision is made per-request, enabling fine-grained optimization and rapid adaptation to changing conditions.

6.2 Performance-Based Routing

The routing layer maintains real-time performance metrics for each airline-channel combination, updated through exponential moving averages of recent requests. Metrics include latency percentiles (p50, p95, p99), error rates by category (timeout, server error, business error), and throughput capacity. When a request could be served by multiple channels, routing favors channels with better recent performance, with randomization to enable exploration and prevent herding effects.

For latency-sensitive requests, we implement speculative execution—dispatching requests to multiple channels simultaneously and returning the first successful response. This technique significantly reduces tail latency at the cost of increased upstream load, so we apply it selectively for high-value requests where

latency impact justifies the overhead. Speculative requests are tagged to enable cancellation when no longer needed, minimizing wasted work.

6.3 Graceful Degradation Patterns

When airline systems experience outages or severe degradation, the platform must continue serving users with reduced but acceptable functionality. Our degradation hierarchy defines progressive fallback behaviors: Level 1 (serve cached content with staleness warnings), Level 2 (exclude affected airlines from search results while serving alternatives), Level 3 (display static schedules without pricing for planning purposes), Level 4 (show maintenance messaging for complete outages).

Degradation decisions are automated based on error rates and response latencies, with manual override capability for operations teams. The system implements hysteresis to prevent rapid oscillation between degradation levels. Recovery from degradation proceeds gradually, with traffic ramped incrementally to verify system stability before full restoration. User-facing messaging clearly communicates limitations during degraded operation, maintaining trust through transparency.

7. Experimental Evaluation

7.1 Production Deployment Context

We evaluate our architecture through production deployment serving a major online travel platform over 18 months. The system processes 2.3 billion annual search requests and 180 million booking transactions across 450 airline integrations spanning GDS (3 providers), NDC (127 airlines), and direct LCC connections (84 carriers). Peak traffic reaches 45,000 search requests per second during promotional events, with booking volumes concentrated during business hours across global time zones.

Baseline comparisons employ the platform's prior architecture, which used conventional integration patterns without the advanced caching, routing, and reliability mechanisms described in this paper. Both systems operate against identical airline endpoints, enabling direct comparison of architectural impact. Additional comparisons reference published performance benchmarks from industry reports and academic literature where available.

7.2 Search Latency Performance

Table 1 presents end-to-end search latency measurements comparing baseline and optimized architectures. The multi-tier caching and predictive refresh mechanisms achieve substantial latency reductions, particularly at higher percentiles where cache misses and airline timeouts previously dominated.

Table 1: Search Latency Comparison (milliseconds)

Metric	Baseline	Optimized	Improvement
p50 Latency	1,240	445	64.1%
p95 Latency	3,850	1,180	69.4%
p99 Latency	8,420	2,340	72.2%

Cache Hit Rate	34%	78%	+44pp
----------------	-----	-----	-------

7.3 Booking Reliability Metrics

Table 2 summarizes booking reliability improvements achieved through the distributed transaction framework and enhanced error handling. Success rate measures the percentage of booking attempts that complete successfully; failure rate captures definitive failures; ambiguous outcomes represent cases requiring manual reconciliation.

Table 2: Booking Reliability Metrics

Metric	Baseline	Optimized	Improvement
Success Rate	96.2%	99.7%	+3.5pp
Failure Rate	2.9%	0.25%	-91.4%
Ambiguous Rate	0.9%	0.05%	-94.4%
Duplicate Rate	0.34%	0.02%	-94.1%

7.4 System Availability

Table 3 presents availability metrics measured over the 18-month evaluation period. Planned maintenance windows are excluded from calculations. The graceful degradation mechanisms enable high availability even during upstream airline outages.

Table 3: System Availability Metrics

Component	Availability	MTTR	Incidents/Year
Search API	99.994%	4.2 min	8
Booking API	99.991%	6.8 min	12
Overall Platform	99.989%	5.1 min	14

7.5 Airline System Load Reduction

Table 4 quantifies the reduction in requests sent to airline systems achieved through caching and intelligent request management. Reduced airline system load decreases costs (GDS transaction fees), improves relationship health (avoiding rate limit triggers), and contributes to industry sustainability.

Table 4: Airline System Load Reduction

Channel Type	Baseline RPS	Optimized RPS	Reduction
GDS Queries	12,400	2,730	78.0%
NDC Requests	8,650	2,160	75.0%

LCC API Calls	5,280	1,420	73.1%
Total	26,330	6,310	76.0%

7.6 Degradation Scenario Analysis

Table 5 presents system behavior during identified stress scenarios from the evaluation period. Metrics capture search success rate (percentage of searches returning results) and booking success rate during each scenario type.

Table 5: Performance Under Stress Scenarios

Scenario	Search Success	Booking Success	Duration
Major GDS Outage	94.2%	91.8%	4.5 hrs
NDC Degradation (10 airlines)	98.7%	97.4%	2.1 hrs
Traffic Spike (3x normal)	99.1%	98.9%	45 min
Regional Network Issue	96.8%	95.2%	1.8 hrs

8. Discussion

8.1 Architectural Trade-offs

Our architecture embodies fundamental trade-offs that practitioners should evaluate against their specific requirements. Aggressive caching improves latency and reduces upstream load but introduces staleness risk—cached prices may not match actual booking prices, creating customer service burden and potential revenue leakage. Our measurements indicate a price match rate of 97.3% with current TTL configurations; operators with stricter accuracy requirements may prefer shorter TTLs at the cost of higher latency and load.

The multi-channel routing strategy provides resilience and optimization flexibility but increases operational complexity. Maintaining expertise across GDS, NDC, and LCC integrations requires diverse skill sets, and debugging issues that span multiple channels can be challenging. Organizations with simpler channel portfolios may prefer streamlined architectures that sacrifice some optimization potential for operational simplicity.

8.2 Lessons Learned

Production operation revealed several insights not apparent during design. First, airline system behavior varies dramatically across times and seasons—performance characteristics during peak travel periods differ substantially from baseline, requiring dynamic tuning of timeouts and retry policies. Second, error classification proved more valuable than expected; distinguishing between transient errors (retry immediately), capacity errors (backoff required), and permanent errors (fail fast) dramatically improves both latency and success rates. Third, observability investment pays compounding returns—detailed tracing and metrics enabled rapid diagnosis of issues that would otherwise require extensive investigation.

8.3 Limitations and Future Work

Several limitations suggest directions for future research. Our caching strategies optimize for aggregate metrics but do not consider individual user context—a returning user searching for a previously viewed itinerary might benefit from fresher content than a new visitor. Personalized cache policies could improve relevance for high-value interactions. The routing optimization currently treats channels independently; joint optimization considering correlated performance across channels could yield further improvements.

Future work will explore: (1) Machine learning models for dynamic TTL optimization based on predicted booking probability and price volatility; (2) Reinforcement learning for routing decisions that consider long-term channel relationship health alongside immediate performance; (3) Integration of external signals (social media, news) for anticipating demand shifts and airline disruptions; (4) Privacy-preserving techniques for cross-platform performance benchmarking; and (5) Application of large language models for intelligent error classification and automated incident response.

9. Conclusion

This paper presented comprehensive architectural patterns for building high-performance, highly reliable flight commerce platforms. Our multi-tier caching architecture with intelligent invalidation and predictive refresh reduces search latency by 64% while decreasing airline system load by 78%. The distributed transaction coordination framework achieves booking success rates exceeding 99.7% despite the unreliable characteristics of upstream airline systems. Adaptive request routing optimizes across heterogeneous distribution channels, and graceful degradation patterns maintain service availability during upstream failures.

Through extensive production deployment serving 2.3 billion annual searches and 180 million bookings across 450 airline integrations, we demonstrate that these patterns deliver substantial improvements in latency, reliability, and operational efficiency. Analysis of stress scenarios including major GDS outages and traffic spikes confirms robust degradation characteristics that maintain acceptable service quality under adverse conditions.

As airline distribution continues evolving toward NDC and direct connectivity, the architectural patterns presented in this paper provide a foundation for platforms navigating this transition. We hope this work contributes actionable guidance for architects building mission-critical travel commerce systems and inspires further research at the intersection of distributed systems and travel technology.

References

- [1] International Air Transport Association (IATA), “World Air Transport Statistics, 2019 Edition,” Industry Report, 2019.
- [2] Arapakis, I., Bai, X., & Cambazoglu, B. B., “Impact of response latency on user behavior in web search,” *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*, 103–112, 2014.
- [3] Copeland, D. G., & McKenney, J. L., “Airline reservations systems: Lessons from history,” *MIS Quarterly*, 12(3), 353–370, 1988.

- [4] International Air Transport Association (IATA), “NDC Implementation Guide,” Industry Standard, Version 17.2, 2017.
- [5] Buhalis, D., & Law, R., “Progress in information technology and tourism management: 20 years on and 10 years after the Internet—The state of eTourism research,” *Tourism Management*, 29(4), 609–623, 2008.
- [6] Krawczyk, M., & Xiang, Z., “Search engine optimization for travel websites,” *Journal of Travel Research*, 54(4), 458–473, 2015.
- [7] Talluri, K. T., & Van Ryzin, G. J., *The Theory and Practice of Revenue Management*, Springer, 2004.
- [8] Garcia-Molina, H., & Salem, K., “Sagas,” *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 249–259, 1987.
- [9] Gray, J., & Reuter, A., *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann, 1992.
- [10] Helland, P., “Idempotence is not a medical condition,” *Communications of the ACM*, 55(5), 56–65, 2012.
- [11] Nygard, M. T., *Release It! Design and Deploy Production-Ready Software*, 2nd ed., Pragmatic Bookshelf, 2018.
- [12] Welsh, M., Culler, D., & Brewer, E., “SEDA: An architecture for well-conditioned, scalable internet services,” *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, 230–243, 2001.