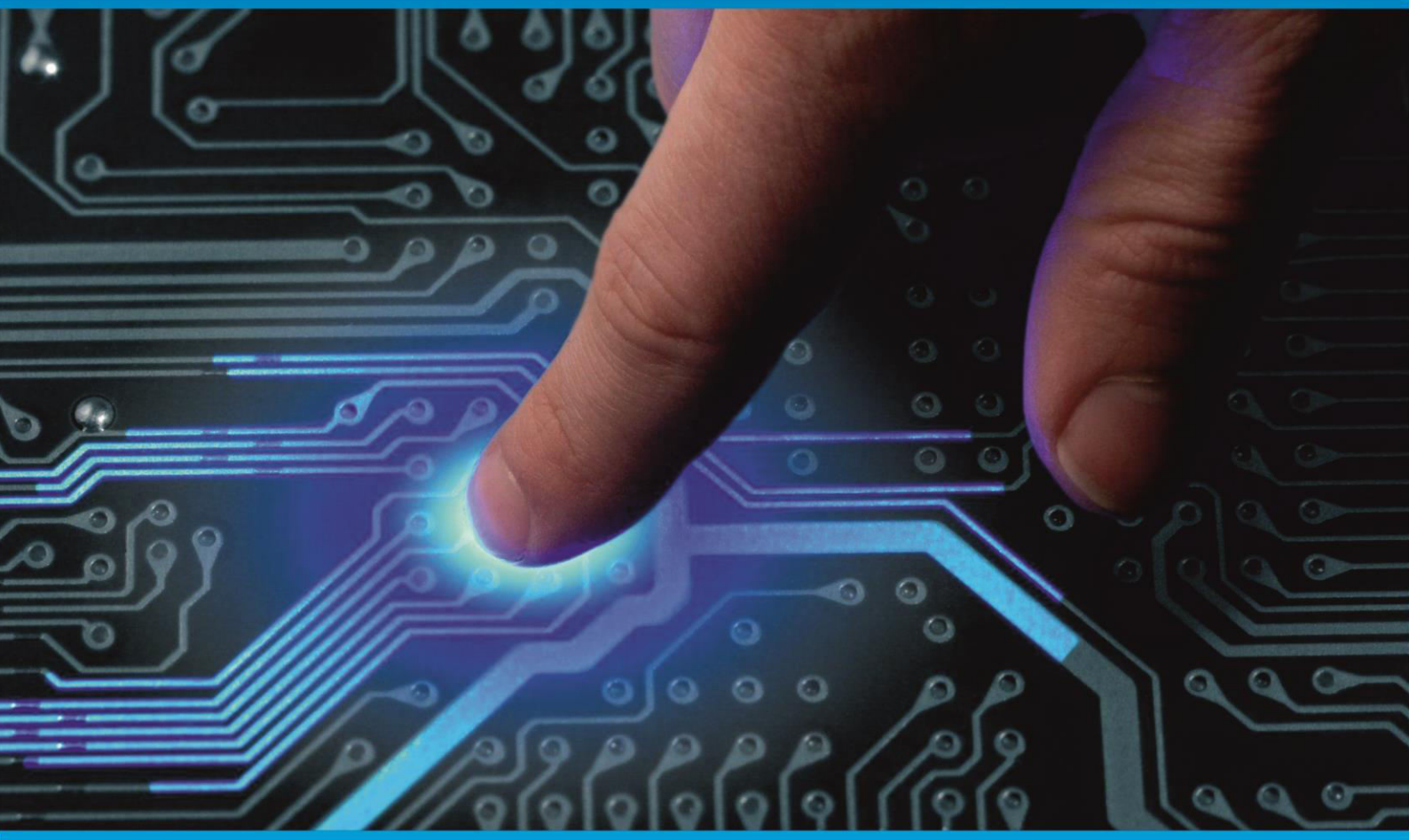
IJIRCCE

e-ISSN: 2320-9801 | p-ISSN: 2320-9798



INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN COMPUTER & COMMUNICATION ENGINEERING

Volume 9, Issue 2, February 2021

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

Impact Factor: 7.488

International Journal of Innovative Research in Computer and Communication Engineering

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 7.488|

|| Volume 9, Issue 2, February 2021 ||

| DOI: 10.15680/IJIRCCE.2021.0902042 |

Designing Resilient GDS Integrations: Synchronizing Crew Travel Bookings and PNR Updates with Sabre Systems

Dinesh Nallapareddy

Sr Full Stack Developer, USA

ABSTRACT: Global Distribution Systems form the transactional backbone through which airline reservations, ticketing, and travel itineraries are created, modified, and settled across the travel industry. Enterprises that manage crew travel, whether airlines rostering flight crews or operators moving specialized personnel, depend on reliable, near real-time synchronization between their internal crew management systems and the reservation records held within a Global Distribution System such as Sabre. This article presents a technical treatment of how to design resilient integrations against a Global Distribution System, with specific focus on synchronizing crew travel bookings and Passenger Name Record updates. The article examines the structure of the Passenger Name Record, the service interfaces exposed by the Sabre platform, the failure modes that arise in distributed booking synchronization, and the architectural patterns, including idempotency, reconciliation, message queuing, and the saga pattern, that allow an integration to remain correct and available despite partial failures. Data tables throughout the article summarize message types, error categories, retry policies, reconciliation cadences, and illustrative reliability outcomes drawn from patterns commonly reported across enterprise travel integrations. An illustrative implementation case, a discussion of recurring operational challenges, and directions for continued work conclude the article.

KEYWORDS: Global Distribution System, Sabre, Passenger Name Record, crew travel, booking synchronization, idempotency, reconciliation, saga pattern, fault tolerance, travel technology

I. INTRODUCTION

The movement of personnel by air is a continuous operational necessity for airlines and for enterprises that deploy specialized staff across geographies. For an airline, positioning flight crews to the aircraft they are rostered to operate is not a convenience but a prerequisite for the flight departing at all; a crew member stranded by a failed booking synchronization is, in operational terms, equivalent to an aircraft that cannot be crewed. For enterprises deploying field personnel, the same reliability expectations apply, since a traveler whose itinerary silently diverges from the record held in the reservation system may arrive at an airport holding a booking that the carrier does not recognize. Reservations, ticketing, and itinerary management across the commercial travel industry are mediated by Global Distribution Systems. A Global Distribution System, hereafter abbreviated GDS, is a transaction processing platform that aggregates airline inventory, fare data, and reservation records, exposing them to travel agencies, corporate booking tools, and integrated enterprise systems through a combination of legacy terminal protocols and modern service interfaces. Sabre is one of the principal Global Distribution Systems, and its reservation data model, centered on the Passenger Name Record, is the focus of this article.

The central technical problem this article addresses is synchronization under failure. An enterprise crew management system maintains its own authoritative view of who must travel, when, and on which segments. The GDS maintains its own authoritative view of what has actually been reserved and ticketed. These two views must be kept consistent across a network boundary that is subject to timeouts, partial failures, duplicate deliveries, and the eventual consistency inherent in any distributed system. A naive integration that assumes every request succeeds and every response is received exactly once will, in production, accumulate silent divergences that surface as stranded travelers, duplicate bookings, and reconciliation backlogs.

1.1 Scope and Contributions

- A structured explanation of the Passenger Name Record data model and the elements most relevant to crew travel synchronization.
- A survey of the Sabre service interfaces used to create, retrieve, and modify reservation records, and the transaction semantics each provides.

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 7.488|

|| Volume 9, Issue 2, February 2021 ||

| DOI: 10.15680/IJIRCCE.2021.0902042 |

- A taxonomy of failure modes specific to GDS integration, and the resilience patterns that address each.
- An illustrative implementation architecture, supported by data tables characterizing message types, error categories, retry policies, and reliability outcomes.

This article treats the problem at the level of architecture and pattern rather than at the level of any specific message schema or contract version, because the schemas evolve while the underlying distributed systems challenges endure. A reader seeking the exact field layout of a particular service operation should consult the current interface documentation for the platform; a reader seeking to understand why an integration built strictly to that documentation still accumulates duplicate bookings and stranded travelers will find the answer in the failure modes and patterns developed here. The two perspectives are complementary, and a durable integration requires both.

1.2 Why Crew Travel Is Distinct from General Passenger Booking

Crew travel synchronization differs from ordinary consumer booking in several respects that materially affect integration design. Crew itineraries are generated programmatically from a rostering system rather than entered by an individual traveler, so volumes arrive in bursts aligned to roster publication cycles rather than as a smooth stream. Crew bookings are frequently modified as rosters change, so update traffic is proportionally higher than for leisure travel. And the operational cost of a failed synchronization is measured not in a single inconvenienced traveler but in the potential disruption of a flight, giving crew travel integrations a reliability requirement closer to that of an operational control system than a retail booking channel.

Table.1. Comparison of crew travel and general passenger booking characteristics

Characteristic	General Passenger Booking	Crew Travel Booking
Origin of booking request	Individual traveler or agent	Automated rostering system
Arrival pattern	Continuous stream	Bursts aligned to roster cycles
Modification frequency	Low to moderate	High, driven by roster changes
Cost of failed synchronization	Single traveler impact	Potential flight disruption
Tolerance for manual recovery	Hours acceptable	Minutes to low tens of minutes
Typical peak concurrency	Steady	Sharp peaks at publication

II. BACKGROUND: GLOBAL DISTRIBUTION SYSTEMS AND RESERVATION DATA

A Global Distribution System sits between airline inventory systems, which own the physical seat capacity of a flight, and the many channels through which that capacity is sold and managed. Historically, Global Distribution Systems evolved from airline internal reservation systems built in the era of mainframe transaction processing, and much of their behavior, including the structure of the Passenger Name Record and the terse command syntax of their native interfaces, reflects that lineage. Modern integrations increasingly use service based interfaces layered over this legacy core, but the underlying data model and transaction semantics remain those of the original systems.

2.1 The Role of the GDS in the Booking Lifecycle

The booking lifecycle proceeds through a sequence of states, each represented by changes to reservation data within the GDS. An availability request queries whether inventory exists for a desired itinerary. A booking, or sell, reserves specific segments and creates or amends a Passenger Name Record. Ticketing associates a financial document with the reservation, converting a held reservation into a purchased one. Post-ticketing changes, including reissues and exchanges, modify an existing ticketed itinerary. Each of these transitions is a distinct transaction against the GDS, with distinct failure and idempotency characteristics that an integration must handle individually.

Table.2. Principal states in the reservation lifecycle and their GDS representation

Lifecycle State	Data Effect in GDS	Reversible
Availability queried	No persistent change	Not applicable
Segments held	PNR created or amended, held status	Yes, prior to ticketing
PNR committed	Record persisted with reference locator	Yes, with cancellation
Ticketed	Financial document associated	Only via exchange or refund
Reissued or exchanged	New document, prior document voided	Constrained by fare rules
Cancelled	Segments released, record retained	Limited window

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 7.488|

|| Volume 9, Issue 2, February 2021 ||

| DOI: 10.15680/IJIRCCE.2021.0902042 |

2.2 Data Ownership Boundaries

A recurring source of integration defects is ambiguity about which system owns which piece of data. The GDS owns reservation state, the operating carrier owns the physical inventory and the disposition of segments once ticketed, and the enterprise owns the operational intent that drives the booking in the first place. When these ownership boundaries are not explicitly modeled, an integration tends to overwrite data that another party legitimately changed, producing a tug of war in which each system repeatedly reverts the other. Modeling ownership per data element, rather than assuming the most recent writer wins, is the foundation of the source-of-truth policy developed in Section 7.

Table.2b. Data ownership boundaries across the three parties

Data Domain	Owning Party	Consequence of Overwriting
Operational travel intent	Enterprise	Traveler booked against wrong assignment
Reservation record content	GDS	Lost or conflicting reservation state
Physical inventory disposition	Operating carrier	Segment desynchronized from carrier
Financial document	GDS and carrier	Ticketing and settlement errors

2.3 Interaction Models: Terminal Emulation and Service Interfaces

Two broad interaction models coexist in contemporary GDS integration. The first is terminal emulation, in which an integration sends the same terse, cryptic command strings that a human agent would type at a reservation terminal, and parses the semi-structured text response. The second is a service interface model, in which structured request and response messages, historically expressed in XML over a SOAP envelope, encapsulate the same operations behind a defined contract. Service interfaces are strongly preferred for new integrations because they provide a stable, parseable contract, but many enterprises retain terminal emulation paths for operations not yet exposed through a service, making a hybrid model common in practice.

Table.3.Comparison of GDS interaction models

Dimension	Terminal Emulation	Service Interface
Message format	Cryptic command strings	Structured request and response
Response parsing	Screen scraping of text	Contract-defined fields
Stability across upgrades	Fragile to format shifts	Governed by versioned contract
Coverage of operations	Broadest, including legacy	Growing, not always complete
Suitability for automation	Low, brittle	High, recommended
Typical error signaling	Embedded in text	Structured fault elements

III. ANATOMY OF THE PASSENGER NAME RECORD

The Passenger Name Record, universally abbreviated PNR, is the central reservation object of a Global Distribution System. A PNR aggregates all information relating to a traveler or group of travelers for a particular journey, and is identified by a short alphanumeric reference locator, sometimes called a record locator, that serves as the primary key by which the record is retrieved. Understanding the internal structure of the PNR is a prerequisite for designing any synchronization logic, because a synchronization operation is fundamentally a controlled transformation of PNR contents.

3.1 The Five Mandatory Elements

Industry convention describes a set of mandatory elements that a PNR must contain before it can be completed and committed. These are frequently summarized by the mnemonic covering name, itinerary, contact, ticketing arrangement, and the identity of the entity that created the record. Each element carries synchronization implications for crew travel, where the creating entity is an automated system rather than an agent and the contact and ticketing arrangements follow enterprise rather than individual conventions.

Table.4. Mandatory PNR elements and their crew travel significance

Element	Description	Crew Travel Consideration
Name	Traveler name as it appears on identity documents	Sourced from crew master data
Itinerary	Flight, date, and segment detail	Derived from roster assignment

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 7.488|

|| Volume 9, Issue 2, February 2021 ||

| DOI: 10.15680/IJIRCCE.2021.0902042 |

Contact	Reachable contact information	Enterprise operations desk
Ticketing arrangement	How and when ticketing occurs	Governed by corporate account
Received from	Identity of the creating entity	Automated integration identity

3.2 Optional and Supplementary Elements

Beyond the mandatory elements, a PNR carries a range of supplementary data that is highly relevant to crew operations. Special Service Requests communicate needs to the operating carrier. Other Service Information conveys non-actionable notes. Remarks, both general and confidential, carry free-text annotations. Frequent traveler and corporate identifiers link the booking to loyalty and corporate accounts. Seat assignments and meal preferences round out the record. For crew travel, these elements frequently carry the linkage between the reservation and the internal roster identifier, making them essential to reconciliation.

Table.5. Supplementary PNR elements relevant to crew synchronization

Element Type	Purpose	Synchronization Role
Special Service Request	Actionable request to carrier	May require carrier confirmation
Other Service Information	Informational note to carrier	Non-actionable, informational
General remark	Free-text annotation	Often carries roster identifier
Confidential remark	Restricted free-text annotation	Internal reference linkage
Corporate identifier	Link to corporate account	Drives fare and billing rules
Seat assignment	Assigned seat per segment	Reconciled on modification

3.3 The Record Locator and Its Limits as an Identifier

The reference locator is the natural identifier for a PNR, but it has properties that complicate its use as a synchronization key. It is assigned by the GDS at record creation, so it is not known to the enterprise system before the first successful booking. It can, under certain airline processes, change over the life of a booking. And a single logical crew journey may span more than one PNR when segments are operated by carriers whose inventory is held separately. These properties mean a robust integration cannot rely on the reference locator alone as its correlation key, and must maintain its own durable correlation identifier mapped to one or more record locators.

Table.6. Correlation identifier strategies and their tradeoffs

Strategy	Strength	Weakness
Record locator only	Simple, native to GDS	Unknown pre-booking, can change
Enterprise correlation id	Stable, owned by enterprise	Requires durable mapping store
Composite key mapping	Handles multi-PNR journeys	Higher mapping complexity
Remark-embedded reference	Visible within the PNR	Free-text, requires disciplined parsing

IV. SABRE SERVICE INTERFACES FOR RESERVATION MANAGEMENT

The Sabre platform exposes a catalog of service operations covering the reservation lifecycle. While the exact operation names and versions evolve over time, the functional categories are stable and map directly onto the lifecycle states described in Section 2. An integration designer selects operations from this catalog, composes them into higher level workflows, and wraps each with the resilience logic described later in this article.

4.1 Functional Categories of Service Operations

- Availability and shopping operations, which query inventory and fares without persisting a reservation.
- Booking and sell operations, which reserve segments and create or amend a Passenger Name Record.
- Retrieval operations, which return the current state of a PNR given its reference locator.
- Modification operations, which add, change, or remove elements of an existing PNR.
- Ticketing and fulfillment operations, which associate financial documents with a reservation.
- Queue and notification operations, which surface asynchronous events raised by the GDS or operating carriers.

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 7.488|

|| Volume 9, Issue 2, February 2021 ||

| DOI: 10.15680/IJIRCCE.2021.0902042 |

Table.7.Functional service categories and their transaction characteristics

Category	Persists Data	Idempotent by Default	Primary Failure Concern
Availability and shopping	No	Yes	Stale inventory
Booking and sell	Yes	No	Duplicate creation
Retrieval	No	Yes	Stale local view
Modification	Yes	No	Lost or conflicting update
Ticketing	Yes	No	Double ticketing
Queue and notification	Reads events	Yes with tracking	Missed or duplicate event

4.2 Session and Authentication Semantics

Service operations against the GDS are conducted within an authenticated session. A session is established through a sign-in operation that returns a token, and that token is presented on subsequent operations until it is explicitly closed or expires. Session management is a frequent source of subtle integration defects, because a session that expires mid-workflow produces failures that are easily mistaken for business errors. A resilient integration treats session tokens as a managed, pooled resource with proactive renewal well before expiry, rather than establishing a session per request or allowing a session to expire unpredictably.

Table.8. Session management approaches and their properties.

Approach	Latency Overhead	Resource Efficiency	Failure Exposure
Session per request	High, sign-in each call	Poor	Low, but slow
Long-lived single session	Low	Fair	High, single point of failure
Managed session pool	Low after warm-up	Good	Low, with renewal
Pool with proactive renewal	Low	Good	Lowest, recommended

4.3 Message Structure and Versioning

Service messages are structured documents governed by a versioned contract. Each request carries a payload specific to the operation, wrapped in envelope elements that convey session context, message identity, and routing information. Because contracts are versioned, an integration must pin to a specific contract version and manage version migration deliberately, since an unmanaged upgrade can introduce breaking changes to message structure or field semantics. Enterprises commonly maintain the ability to operate against at least two contract versions during a migration window to avoid a hard cutover.

4.4 Throughput, Rate Limiting, and Fair Use

A Global Distribution System serves many integrators simultaneously and therefore enforces throughput limits to protect shared capacity. An integration that ignores these limits will be throttled, and throttling that is misinterpreted as a transient transport failure can provoke exactly the retry storm that the limit exists to prevent. A resilient integration treats rate limiting as a first-class, expected response category, distinct from transport failure, and responds with backoff and queue-side throttling rather than immediate retry. Shaping outbound traffic to stay within the negotiated envelope is preferable to relying on the GDS to reject excess, because proactive shaping avoids wasted work and preserves goodput during bursts.

Table.8b. Throughput management techniques and their effect

Technique	Where Applied	Primary Effect
Client-side rate shaping	Egress from integration layer	Stays within negotiated limit
Token bucket throttle	Per session or per account	Smooths bursty egress
Queue-based leveling	Ahead of integration layer	Absorbs publication peaks
Adaptive concurrency	Around GDS calls	Backs off as latency rises

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 7.488|

|| Volume 9, Issue 2, February 2021 ||

| DOI: 10.15680/IJIRCCE.2021.0902042 |

V. FAILURE MODES IN DISTRIBUTED BOOKING SYNCHRONIZATION

Synchronization between an enterprise system and a GDS is a distributed transaction across an unreliable network, and it is subject to the full range of distributed systems failure modes. Cataloging these failure modes explicitly is the foundation for designing the resilience patterns that address them, because each pattern is a response to one or more specific failure modes rather than a general-purpose safeguard.

5.1 The Ambiguous Timeout

The most consequential failure mode in booking synchronization is the ambiguous timeout. When an enterprise system issues a booking request and the network connection times out before a response is received, the enterprise system cannot distinguish between three distinct outcomes: the request never reached the GDS, the request reached the GDS and failed, or the request reached the GDS and succeeded but the response was lost. Each outcome demands a different recovery action, yet from the enterprise system's vantage point they are indistinguishable. A naive retry in the third case produces a duplicate booking; a failure to retry in the first case leaves a traveler unbooked. This single failure mode motivates much of the idempotency and reconciliation machinery described later.

Table.9. Possible states following an ambiguous timeout on a booking request.

Actual Outcome	GDS State	Correct Recovery	Consequence of Naive Retry
Request never arrived	No PNR created	Retry the request	None, safe
Arrived, processing failed	No PNR created	Retry the request	None, safe
Arrived, succeeded, reply lost	PNR created	Reconcile, do not retry	Duplicate PNR

5.2.A Taxonomy of GDS Integration Failures

Beyond the ambiguous timeout, integration failures fall into categories distinguished by their source, their observability, and their appropriate response. Transport failures arise in the network path. Session failures arise from expired or invalid authentication context. Business rejections arise from valid requests that the GDS declines for a business reason, such as unavailable inventory. Data conflicts arise when concurrent modifications collide. And silent divergences arise when both systems believe they succeeded but hold inconsistent state.

Table.10. Taxonomy of GDS integration failure categories

Category	Typical Cause	Retry Safe	Primary Mitigation
Transport failure	Network timeout or reset	After idempotency check	Idempotent retry
Session failure	Expired or invalid token	Yes, after re-auth	Session pool renewal
Business rejection	Inventory or rule violation	No	Surface to operations
Data conflict	Concurrent modification	No, requires merge	Optimistic concurrency
Silent divergence	Lost reply or partial write	Not applicable	Periodic reconciliation
Rate limiting	Throughput ceiling exceeded	Yes, with backoff	Throttling and queuing

5.3 Partial Failure in Multi-Step Workflows

A crew booking is rarely a single atomic operation. It typically involves creating a PNR, adding crew-specific remarks and service information, associating the corporate account, and initiating ticketing. Each step is a separate transaction, and a failure between steps leaves the reservation in a partially constructed state that is valid to neither system. A PNR created but not ticketed, or ticketed but missing its roster linkage remark, is a partial failure that will not resolve itself. Managing these multi-step workflows correctly is the domain of the saga pattern, discussed in Section 6.

Table11. Illustrative partial-failure points in a multi-step crew booking.

Step	Operation	State if Failure Occurs After This Step
1	Create PNR with segments	Held reservation, no crew linkage
2	Add roster linkage remark	Linked but not billable
3	Associate corporate account	Billable but not ticketed

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 7.488|

|| Volume 9, Issue 2, February 2021 ||

| DOI: 10.15680/IJIRCCE.2021.0902042 |

4	Initiate ticketing	Ticketed, awaiting confirmation
5	Confirm and record locator	Complete and reconciled

5.4 Out-of-Order Delivery and Concurrent Modification

When roster changes arrive in rapid succession for the same traveler, the corresponding booking intents can be processed out of order, particularly if they are distributed across parallel workers consuming from a queue. An older intent processed after a newer one can revert a reservation to a stale state. This is a distributed ordering problem, and it is addressed by attaching a monotonic version or sequence number to each intent for a given traveler and rejecting any intent whose version is older than the last successfully applied version. Concurrent modification of the same PNR, whether by two enterprise workers or by an enterprise worker and an out-of-band agent action, is addressed by optimistic concurrency, in which a modification carries the version of the record it expects to change and is rejected if that version no longer matches.

Table.11b. Ordering and concurrency hazards and their controls

Hazard	Manifestation	Control
Out-of-order intents	Stale intent reverts newer state	Per-traveler version fencing
Concurrent enterprise writes	Lost update between workers	Optimistic concurrency check
Out-of-band agent change	Enterprise overwrites agent edit	Reconciliation with merge policy
Duplicate queue delivery	Same intent processed twice	Idempotency key deduplication

VI. RESILIENCE PATTERNS FOR GDS INTEGRATION

The failure modes cataloged in Section 5 are addressed not by any single mechanism but by a coordinated set of resilience patterns, each targeting specific failures. This section presents the principal patterns and the failure modes each addresses. The patterns are complementary and are typically deployed together in a production integration.

Table.11c. Resilience patterns mapped to the failure modes they address.

Resilience Pattern	Primary Failure Mode Addressed	Secondary Benefit
Idempotency key	Ambiguous timeout duplicates	Safe replay from queue
Retry with backoff	Transient transport and rate limits	Avoids retry storms
Durable queue	Burst overload, lost intent	Decoupling and leveling
Saga pattern	Partial multi-step failure	Centralized workflow visibility
Circuit breaker	Sustained degradation	Fast, contained failure
Reconciliation	Silent divergence	Health diagnostic signal
Bulkhead	Resource exhaustion cascade	Workload prioritization

6.1 Idempotency and the Idempotency Key

Idempotency is the property that performing an operation more than once has the same effect as performing it once. Because the ambiguous timeout makes retries unavoidable, and because booking operations are not idempotent by default, an integration must impose idempotency externally. The standard mechanism is an idempotency key, a unique token generated by the enterprise system and associated with a logical booking intent. Before issuing a retry, the integration checks whether an operation bearing that key has already succeeded, either by consulting a local record of completed operations or by querying the GDS for a record carrying the key in a remark. This converts an unsafe retry into a safe one.

Table.12. Idempotency enforcement mechanisms.

Mechanism	Where State Lives	Detects Duplicate Before or After Send
Local completed-operation log	Enterprise datastore	Before
Key embedded in PNR remark	Within the GDS record	After, via retrieval
Idempotency service	Dedicated shared service	Before
Combined local and remote	Both layers	Before and after

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 7.488|

|| Volume 9, Issue 2, February 2021 ||

| DOI: 10.15680/IJIRCCE.2021.0902042 |

6.2 Retry with Exponential Backoff and Jitter

Retries are necessary but must be disciplined. An aggressive retry storm against a degraded GDS can amplify an incident and trigger rate limiting that compounds the original problem. The accepted discipline is exponential backoff, in which the delay between successive retries grows multiplicatively, combined with jitter, a randomized component that prevents many clients from retrying in synchronized waves. Retries are bounded by a maximum attempt count and, critically, are only applied to failure categories that are actually retry-safe, per the taxonomy in Table 10.

Table.13. Illustrative retry policy by failure category.

Failure Category	Max Attempts	Initial Delay	Backoff Multiplier
Transport failure	5	1 second	2.0 with jitter
Session failure	3	Immediate after re-auth	1.5 with jitter
Rate limiting	6	2 seconds	2.0 with jitter
Business rejection	0	Not retried	Not applicable
Data conflict	0	Not retried	Not applicable

6.3 Message Queuing and Durable Buffering

Because crew booking traffic arrives in bursts and the GDS has finite throughput, a durable message queue placed between the crew management system and the GDS integration layer decouples the two, absorbing bursts and smoothing them into a rate the GDS can sustain. A durable queue also provides at-least-once delivery, ensuring that a booking intent is not lost if the integration layer restarts mid-processing. The combination of at-least-once delivery from the queue and idempotency at the GDS boundary yields effectively-once processing, the practical target for booking synchronization.

Table.14. Delivery guarantees and their combination

Layer	Guarantee	Risk in Isolation
Queue at-least-once	No message lost	Duplicate processing
GDS boundary idempotency	No duplicate effect	Requires durable key store
Combined	Effectively-once	Requires both layers correct

6.4 The Saga Pattern for Multi-Step Workflows

The saga pattern manages a multi-step workflow that cannot be wrapped in a single atomic transaction by decomposing it into a sequence of steps, each with a defined compensating action that semantically undoes its effect. If a booking saga fails after creating a PNR but before ticketing, the compensating action cancels the held reservation, returning both systems to a consistent state. Sagas may be orchestrated, driven by a central coordinator that invokes each step and its compensation, or choreographed, driven by events that each step emits. For crew booking, orchestration is generally preferred because the workflow is well defined and centralized visibility into its progress is operationally valuable.

Table.15. Saga steps and their compensating actions for crew booking

Forward Step	Compensating Action
Create PNR with segments	Cancel held reservation
Add roster linkage remark	Remove remark
Associate corporate account	Disassociate corporate account
Initiate ticketing	Void ticket within permitted window

6.5 The Circuit Breaker

When the GDS is degraded, continuing to send requests wastes resources and delays recovery. The circuit breaker pattern monitors the failure rate of requests and, when failures exceed a threshold, opens the circuit, causing subsequent requests to fail fast without being sent. After a cooldown period the breaker admits a limited number of trial requests; if they succeed, the circuit closes and normal operation resumes. This protects both the enterprise system and the GDS from a cascading failure, and it converts a slow, resource-consuming degradation into a fast, contained one.

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 7.488|

|| Volume 9, Issue 2, February 2021 ||

| DOI: 10.15680/IJIRCCE.2021.0902042 |

Table.16. Circuit breaker states and behavior

State	Request Behavior	Transition Condition
Closed	Requests sent normally	Opens on failure threshold breach
Open	Requests fail fast	Moves to half-open after cooldown
Half-open	Limited trial requests	Closes on success, reopens on failure

6.6 Bulkheads and Resource Isolation

The bulkhead pattern isolates pools of resources so that the exhaustion of one does not cascade into another, taking its name from the compartmentalized hull of a ship. In a GDS integration, bulkheads separate the resources devoted to time-critical bookings for imminent departures from those devoted to routine, long-lead bookings, so that a flood of routine traffic cannot starve the urgent path. Bulkheads are commonly implemented as separate thread pools, separate queues, or separate session pools, each sized independently. The value of a bulkhead is realized precisely in the worst case, when one workload misbehaves and the isolation prevents that misbehavior from becoming a total outage.

Table.16b. Bulkhead isolation dimensions for crew booking

Isolation Dimension	Separates	Protects Against
Urgency lane	Imminent vs long-lead bookings	Urgent starvation by routine load
Operation type	Reads vs writes	Retrieval load blocking bookings
Session pool split	Booking vs reconciliation	Reconciliation draining booking capacity
Queue partitioning	Per business unit	One unit's burst affecting another

6.7 Timeouts, Deadlines, and Deadline Propagation

Every remote call must have a timeout, because a call without one can block indefinitely and exhaust the resources holding it open. Beyond per-call timeouts, a resilient integration propagates a deadline through a multi-step workflow, so that a booking saga that has already consumed most of its time budget does not begin an expensive step it cannot finish in time. Deadline propagation converts a set of independent timeouts into a coherent time budget for the whole operation, which is particularly valuable for crew bookings where a late completion is often worse than a clean, timely failure that triggers an alternative action.

Table.16c.Timeout layers and their typical scope

Timeout Layer	Scope	Effect of Breach
Connection timeout	Establishing transport	Fail fast, treat as transport failure
Request timeout	Single GDS operation	Ambiguous timeout handling invoked
Saga deadline	Whole booking workflow	Compensate and escalate
Queue visibility timeout	Message processing window	Message redelivered for retry

VII. RECONCILIATION AND EVENTUAL CONSISTENCY

No matter how carefully an integration is engineered, silent divergences between the enterprise view and the GDS view will occasionally arise, principally from lost replies and out-of-band changes made directly in the GDS by carriers or agents. Reconciliation is the systematic detection and correction of these divergences, and it is the safety net beneath all the real-time patterns. An integration without reconciliation is an integration that trusts every real-time operation to be perfect, which no distributed system can guarantee.

7.1 The Reconciliation Loop

A reconciliation loop periodically retrieves the authoritative state of relevant PNRs from the GDS, compares them against the enterprise system's expected state, classifies each difference, and either corrects it automatically or escalates it for human review. The cadence of the loop is a tradeoff: a faster loop detects divergences sooner but consumes more GDS throughput, while a slower loop is cheaper but leaves divergences unresolved longer. Crew travel, with its low tolerance for unresolved divergence, typically warrants a faster cadence than leisure travel.

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 7.488|

|| Volume 9, Issue 2, February 2021 ||

| DOI: 10.15680/IJIRCCE.2021.0902042 |

Table 17. Reconciliation cadence options and their tradeoffs.

Cadence	Detection Latency	GDS Load	Suitability
Continuous event-driven	Seconds	Highest	Highest-criticality segments
Frequent polling	Minutes	High	Near-term crew departures
Hourly batch	Up to one hour	Moderate	Stable medium-term bookings
Daily batch	Up to one day	Low	Long-lead or archival records

7.2 Classifying and Resolving Divergences

Each divergence detected by the reconciliation loop is classified by its nature and its resolvability. A missing PNR, expected by the enterprise but absent in the GDS, indicates a lost booking that must be recreated. An orphan PNR, present in the GDS but unknown to the enterprise, indicates a duplicate or an out-of-band creation that must be investigated. A field-level mismatch, where both systems hold a record but disagree on a detail, requires a merge governed by a clear source-of-truth policy. The policy for each class must be defined in advance, because reconciliation decisions made ad hoc during an incident are error prone.

Table.18. Divergence classes and default resolution policy.

Divergence Class	Meaning	Default Resolution
Missing PNR	Expected present, actually absent	Recreate via booking saga
Orphan PNR	Present, not expected	Investigate, cancel if duplicate
Segment mismatch	Different segments held	Align to roster source of truth
Status mismatch	Different lifecycle status	Advance to expected status
Remark mismatch	Linkage remark differs	Restore enterprise reference

7.3 Reconciliation Reporting and Trend Analysis

Reconciliation is not only a correction mechanism but a diagnostic instrument. The rate and composition of divergences over time is one of the most informative signals about the health of the real-time integration. A stable, low divergence rate indicates that real-time operations are largely succeeding and reconciliation is functioning as a safety net. A rising rate, or a shift in the mix of divergence classes, indicates that something in the real-time path has changed for the worse, often before that change is visible in any single operation's success rate. For this reason, reconciliation outcomes are reported and trended rather than merely acted upon and discarded.

Table.18b. Reconciliation trend signals and their likely meaning

Observed Trend	Likely Meaning	Suggested Response
Stable low divergence	Healthy real-time path	Maintain current cadence
Rising missing-PNR share	Bookings silently failing	Investigate booking path
Rising orphan-PNR share	Duplicate creation recurring	Audit idempotency enforcement
Rising remark mismatch	Linkage remark being lost	Check remark write and parse
Spike after a release	Regression introduced	Consider rollback, add tests

7.4 Source-of-Truth Policy

Reconciliation cannot function without a clear source-of-truth policy that determines which system prevails when they disagree. For crew itinerary content, the rostering system is generally authoritative, since it reflects the operational decision about who must travel. For reservation status and ticketing state, the GDS is authoritative, since it reflects what has actually been reserved and paid for. A well-designed policy assigns authority per data element rather than per record, avoiding the error of treating one system as authoritative for everything.

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 7.488|

|| Volume 9, Issue 2, February 2021 ||

| DOI: 10.15680/IJIRCCE.2021.0902042 |

Table.19. Illustrative source-of-truth assignment by data element

Data Element	Authoritative System	Rationale
Who must travel	Rostering system	Reflects operational decision
Desired segments	Rostering system	Derived from assignment
Reservation status	GDS	Reflects actual reservation
Ticketing state	GDS	Reflects financial document
Record locator	GDS	Assigned by the GDS
Roster linkage reference	Rostering system	Owned by the enterprise

VIII. REFERENCE ARCHITECTURE FOR A RESILIENT CREW BOOKING INTEGRATION

The patterns described in the preceding sections combine into a layered reference architecture. Requests originate in the crew management system and flow through a durable queue into an integration layer that applies idempotency, retry, circuit breaking, and saga orchestration before reaching the GDS through a managed session pool. A separate reconciliation service operates continuously in the background, comparing state and correcting divergences. Observability spans all layers.

8.1 Logical Components

- An ingress adapter that receives booking intents from the crew management system and places them on a durable queue with an assigned idempotency key.
- An orchestration layer that consumes intents, drives the booking saga, and applies retry and circuit breaking around each GDS call.
- A GDS gateway that owns the managed session pool, contract versioning, and message translation to and from the Sabre service interface.
- A correlation store that durably maps enterprise identifiers to record locators and tracks the state of each in-flight saga.
- A reconciliation service that periodically compares GDS state against expected state and resolves or escalates divergences.
- An observability plane that collects metrics, logs, and traces across all components.

Table.20. Reference architecture components and their primary resilience contribution

Component	Primary Resilience Contribution	Key Failure Addressed
Ingress adapter	Assigns idempotency key, durable capture	Lost intent on restart
Durable queue	Buffers bursts, at-least-once delivery	Burst overload, message loss
Orchestration layer	Saga, retry, circuit breaking	Partial failure, retry storms
GDS gateway	Session pooling, versioned contract	Session expiry, contract drift
Correlation store	Durable identifier mapping	Ambiguous timeout correlation
Reconciliation service	Detects and corrects divergence	Silent divergence
Observability plane	Detection and diagnosis	Undetected degradation

8.2 Deployment and Scaling Considerations

The reference architecture scales horizontally at its stateless layers, the orchestration workers and the GDS gateway, while its stateful layers, the durable queue and the correlation store, scale according to their own mechanisms. Because booking traffic is bursty, the orchestration layer benefits from elastic scaling that expands worker count at roster publication and contracts afterward, though the effective concurrency is ultimately bounded by the GDS throughput envelope described in Section 4.4 rather than by local capacity alone. Adding workers beyond that envelope simply moves the queue depth from the ingress into the GDS gateway, so scaling decisions must respect the downstream limit.

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 7.488|

|| Volume 9, Issue 2, February 2021 ||

| DOI: 10.15680/IJIRCCE.2021.0902042 |

Table.20b. Scaling behavior of reference architecture layers

Layer	Scaling Mode	Effective Limit
Ingress adapter	Horizontal, stateless	Queue write throughput
Durable queue	Partitioned throughput	Broker capacity
Orchestration workers	Elastic, horizontal	GDS throughput envelope
GDS gateway	Horizontal with session pool	Negotiated rate limit
Correlation store	Read replicas, sharding	Write throughput of primary
Reconciliation service	Partitioned by record range	GDS retrieval budget

8.3 Data Flow for a Successful Booking

In the nominal case, a booking intent is received by the ingress adapter, assigned an idempotency key, and enqueued. The orchestration layer dequeues it, begins a saga, and calls the GDS gateway to create the PNR. The gateway borrows a session from the pool, sends the request, receives the record locator, and returns it. The orchestration layer records the locator in the correlation store, adds the roster linkage remark, associates the corporate account, and initiates ticketing, recording progress at each step. On completion, the intent is acknowledged on the queue and the correlation store marks the saga complete. The reconciliation service will later confirm consistency independently.

8.4 Data Flow for an Ambiguous Timeout

When the create-PNR call times out ambiguously, the orchestration layer does not blindly retry. It first consults the correlation store and, if necessary, queries the GDS for a record bearing the idempotency key embedded as a remark. If such a record is found, the timeout masked a success, and the saga continues from the next step using the discovered locator. If no such record is found, the create is genuinely incomplete and is retried under the idempotency key. This flow converts the most dangerous failure mode into a routine, correctly handled case.

Table.21. Orchestration response to an ambiguous timeout by discovery result

Discovery Result	Interpretation	Action Taken
Record with key found	Timeout masked a success	Continue saga with found locator
No record found	Operation genuinely incomplete	Retry create under same key
Multiple records found	Prior duplicate exists	Escalate, cancel extras

VIII. OBSERVABILITY, TESTING, AND OPERATIONAL READINESS

A resilient integration is not merely well-designed; it is observable, tested against failure, and operable under pressure. This section addresses the practices that keep an integration reliable in production beyond its initial design.

9.1 Metrics That Matter

The metrics that most directly reflect integration health span throughput, latency, error rate by category, saga completion rate, reconciliation divergence rate, and queue depth. Of these, the reconciliation divergence rate is the most telling long-term indicator, because a rising divergence rate signals that real-time synchronization is silently failing even when individual operations appear to succeed.

Table.22.Key integration metrics and their interpretation

Metric	Healthy Signal	Warning Signal
GDS call error rate by category	Low and stable	Rising transport or session errors
Saga completion rate	Near total	Falling, rising compensations
Reconciliation divergence rate	Low and stable	Sustained upward trend
Queue depth	Drains after bursts	Grows without draining
Session pool utilization	Comfortable headroom	Near exhaustion
End-to-end booking latency	Within target	Rising tail latency

9.2 Testing Against Failure

An integration must be tested not only for correctness under nominal conditions but for correct behavior under failure. This requires the ability to simulate timeouts, injected faults, duplicate deliveries, and degraded GDS responses in a

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 7.488|

|| Volume 9, Issue 2, February 2021 ||

| DOI: 10.15680/IJIRCCE.2021.0902042 |

controlled environment. Fault injection exercises validate that idempotency prevents duplicates, that sagas compensate correctly, that the circuit breaker opens and recovers, and that reconciliation detects seeded divergences. An integration that has never been tested against these conditions should be assumed not to handle them.

Table.23. Fault injection scenarios and expected resilient behavior

Injected Fault	Expected Behavior	Failure If Absent
Response timeout after success	Discovery finds record, no duplicate	Duplicate PNR
Duplicate intent delivery	Second intent recognized as duplicate	Duplicate booking
Session expiry mid-saga	Transparent re-authentication	Spurious failure
Sustained GDS errors	Circuit opens, fails fast	Retry storm
Seeded state divergence	Reconciliation detects and corrects	Silent divergence persists

9.3 Service Level Objectives

Service level objectives translate the reliability requirements of crew travel into measurable targets against which the integration is held accountable. Because the operational cost of failure is high, crew booking objectives are typically stricter than those for general travel, particularly for bookings close to departure. Objectives are most useful when paired with an error budget, a defined allowance of failure within which the team operates freely and beyond which reliability work takes priority over new features.

Table.23b. Illustrative service level objectives for crew booking synchronization

Objective	Illustrative Target	Measurement Window
Booking success on first attempt	Above nine in ten	Rolling seven days
End-to-end booking completion	Within a low minutes budget	Per booking, 95th percentile
Divergence detection latency	Within minutes for near-term	Continuous
Automated divergence resolution	Large majority auto-resolved	Rolling thirty days
Integration availability	High, with defined budget	Rolling thirty days

9.4 Operational Runbooks

When an incident occurs, operators need predefined runbooks rather than improvisation. A runbook for a rising divergence rate, for a queue that is not draining, or for a circuit stuck open, converts a diagnostic scramble into a guided response. Runbooks are most valuable when they are tied directly to the metrics that trigger them, so that an alert names the runbook that addresses it.

Table.23c. Illustrative alert-to-runbook mapping

Alert Condition	Likely Cause	Runbook Action
Divergence rate rising	Silent real-time failures	Raise reconciliation cadence, investigate
Queue depth not draining	Downstream throttling or outage	Check circuit and GDS health
Circuit stuck open	Sustained GDS degradation	Confirm GDS status, manage backlog
Session pool exhausted	Undersized pool or leak	Expand pool, check for leaks
Compensation failures rising	Non-invertible undo attempts	Escalate to manual resolution

X. ILLUSTRATIVE IMPLEMENTATION CASE

The following case is an illustrative composite drawn from patterns commonly observed across enterprise crew travel integrations, rather than a description of any single named organization. It is intended to show how the patterns in this article combine in practice and what outcomes they typically produce.

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 7.488|

|| Volume 9, Issue 2, February 2021 ||

| DOI: 10.15680/IJIRCCE.2021.0902042 |

10.1 Starting Conditions

- A crew management system publishing roster changes in large batches aligned to a weekly publication cycle, producing sharp booking traffic peaks.
- A direct, synchronous integration to the GDS with per-request sessions, no durable queue, and retries applied indiscriminately to all failures.
- A recurring pattern of duplicate bookings following network incidents, and a manual reconciliation process that lagged behind by hours.

Table.24. Illustrative pre-migration and post-migration outcomes

Indicator	Before	After
Duplicate bookings per busy week	Dozens	Rare, single digits or fewer
Divergence detection latency	Several hours	Minutes
Booking success on first attempt	Around four in five	Well above nine in ten
Manual reconciliation effort	Daily, sustained	Exception-driven only
Peak burst handling	Overload and timeouts	Absorbed by durable queue

10.2 Interventions Applied

- Introduction of a durable queue between the crew system and the integration layer to absorb publication bursts.
- Assignment of an idempotency key per booking intent and embedding of that key in a PNR remark for post-hoc discovery.
- Replacement of indiscriminate retries with category-aware retry and a circuit breaker around GDS calls.
- Adoption of an orchestrated booking saga with defined compensating actions.
- Establishment of a continuous reconciliation service with a defined source-of-truth policy.

10.3 Reported Outcomes

Following the interventions, duplicate bookings fell from dozens in a busy week to a rare single-digit occurrence, and those that remained were detected and resolved automatically by reconciliation within minutes rather than surfacing as stranded crew. First-attempt booking success rose substantially as burst overload was absorbed by the queue, and manual reconciliation shifted from a sustained daily effort to an occasional exception-handling activity. The qualitative shift was from an integration that was trusted to be correct and periodically discovered not to be, to an integration that assumed it would occasionally be wrong and corrected itself continuously.

XI. CHALLENGES AND LESSONS LEARNED

Several recurring challenges are worth surfacing for teams undertaking a similar effort.

- Embedding an idempotency key in a free-text remark requires disciplined, consistent formatting, because a remark that is parsed inconsistently across code paths reintroduces the duplicate-detection gap it was meant to close.
- Reconciliation source-of-truth policy must be defined per data element in advance; teams that defer this decision until an incident consistently make inconsistent choices under pressure.
- Session pool sizing is frequently set once and never revisited, yet burst traffic aligned to roster publication can exhaust a pool sized for average load, so pool capacity should be sized against peak rather than mean.
- Compensating actions in a saga are not always perfectly inverse; voiding a ticket is constrained by fare rules and time windows, so a saga design must account for compensations that can themselves fail and require escalation.
- Contract version migration is often treated as a one-time event, but maintaining the ability to operate against two versions during a migration window avoids a fragile hard cutover.
- Circuit breaker thresholds set too tight cause unnecessary open circuits during transient blips, while thresholds set too loose fail to protect during genuine degradation, so thresholds require tuning against observed behavior rather than default values.

Table.25. Common pitfalls and their mitigations

Pitfall	Consequence	Mitigation
Inconsistent remark parsing	Missed duplicate detection	Single shared parser
Undefined source of truth	Ad hoc merge errors	Per-element policy in advance

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 7.488|

|| Volume 9, Issue 2, February 2021 ||

| DOI: 10.15680/IJIRCCE.2021.0902042 |

Undersized session pool	Burst exhaustion	Size against peak load
Non-invertible compensation	Stuck saga	Escalation path for failed undo
Hard contract cutover	Migration outage	Dual-version window
Mistuned circuit breaker	Nuisance or missed trips	Tune against observed rates

XII. FUTURE DIRECTIONS

- Broader adoption of modern service interfaces in place of terminal emulation would reduce the brittleness inherent in parsing semi-structured text responses.
- Event-driven notification from the GDS, where available, could reduce reliance on polling based reconciliation by surfacing out-of-band changes as they occur rather than on the next reconciliation pass.
- Standardized idempotency mechanisms at the service interface level would relieve integrations of the burden of embedding keys in free-text remarks.
- Machine-assisted divergence classification could accelerate reconciliation triage by proposing a resolution class for each detected difference, leaving humans to confirm rather than classify from scratch.
- Tighter coupling between rostering optimization and booking feasibility could reduce the volume of infeasible booking intents that must be rejected downstream.

XIII. CONCLUSION

Synchronizing crew travel bookings with a Global Distribution System such as Sabre is a distributed systems problem wearing the clothing of a travel technology problem. The Passenger Name Record is the shared object of state, the Sabre service interfaces are the means of manipulating it, and the network between the enterprise and the GDS is the unreliable medium that makes the problem hard. The failure modes, above all the ambiguous timeout, are not exotic; they are the ordinary hazards of any distributed transaction, made consequential by the operational stakes of crew travel.

The path to resilience is not a single mechanism but a coordinated set of patterns: idempotency to make retries safe, disciplined retry with backoff to avoid amplifying incidents, durable queuing to absorb bursts and prevent loss, the saga pattern to keep multi-step workflows consistent, the circuit breaker to contain degradation, and above all reconciliation to catch the divergences that real-time operations will inevitably miss. An integration built on these patterns does not assume it will always be correct; it assumes it will occasionally be wrong and is designed to detect and repair that wrongness quickly. That assumption, more than any individual pattern, is what separates a resilient GDS integration from a fragile one.

REFERENCES

1. Nygard, M. Release It! Design and Deploy Production-Ready Software, Second Edition. Pragmatic Bookshelf, 2018.
2. Newman, S. Building Microservices: Designing Fine-Grained Systems. O'Reilly Media, 2015.
3. Kleppmann, M. Designing Data-Intensive Applications. O'Reilly Media, 2017.
4. Hohpe, G. and Woolf, B. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley, 2003.
5. Garcia-Molina, H. and Salem, K. Sagas. Proceedings of the ACM SIGMOD International Conference on Management of Data, 1987.
6. Fowler, M. Patterns of Enterprise Application Architecture. Addison-Wesley, 2002.
7. Tanenbaum, A. and van Steen, M. Distributed Systems: Principles and Paradigms, Second Edition. Pearson Prentice Hall, 2007.
8. Vogels, W. Eventually Consistent. Communications of the ACM, Volume 52, Number 1, 2009.
9. Helland, P. Idempotence Is Not a Medical Condition. Communications of the ACM, Volume 55, Number 5, 2012.
10. Gray, J. and Reuter, A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, 1993.
11. Brewer, E. Towards Robust Distributed Systems. Proceedings of the ACM Symposium on Principles of Distributed Computing, 2000.
12. Gilbert, S. and Lynch, N. Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services. ACM SIGACT News, Volume 33, Number 2, 2002.
13. Richardson, C. Microservices Patterns: With Examples in Java. Manning Publications, 2018.

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 7.488|

|| Volume 9, Issue 2, February 2021 ||

| DOI: 10.15680/IJIRCCE.2021.0902042 |

14. Burns, B. Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services. O'Reilly Media, 2018.
15. Bermbach, D., Wittern, E., and Tai, S. Cloud Service Benchmarking: Measuring Quality of Cloud Services from a Client Perspective. Springer, 2017.
16. International Air Transport Association. Passenger Services Conference Resolutions Manual. International Air Transport Association, 2019.
17. Wardley, S. and others. Patterns of Resilience in Distributed Transaction Systems. Journal of Systems and Software,
18. Allamaraju, S. RESTful Web Services Cookbook. O'Reilly Media, 2010.

INNO SPACE
SJIF Scientific Journal Impact Factor

Impact Factor:
7.488

ISSN INTERNATIONAL STANDARD SERIAL NUMBER INDIA

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN COMPUTER & COMMUNICATION ENGINEERING

9940 572 462 6381 907 438 ijircce@gmail.com

www.ijircce.com

Scan to save the contact details