Cureus Journal of Computer Science

Open Access Original Article

Optimizing Airline Reservation Systems With Edge-Enabled Microservices: A Framework for Real-Time Data Processing and Enhanced User Responsiveness

Received 04/27/2025
Review began 05/06/2025
Review ended 07/18/2025
Published 07/19/2025

DOI:
https://doi.org/10.7759/s44389-025-05713-8

Biman Barua [1, 2] , M. Shamim Kaiser [1]

1. Institute of Information Technology, Jahangirnagar University, Dhaka, BGD 2. Computer Science and Engineering, BGMEA University of Fashion and Technology, Dhaka, BGD

Corresponding author: Biman Barua, biman@buft.edu.bd

Abstract

The complexity of airline reservation systems demands solutions that are fast and adaptive in nature. The paper proposes a theoretical framework that deploys edge computing microservices to support the traditional centralized systems, particularly targeting latency and performance issues. Performing calculations like seat inventory checks, bookings, and confirmations closer to the user reduces response time and enhances performance.

Kubernetes is a core microservice orchestration system, Kafka handles real-time messaging, while Prometheus and Grafana monitor the resources. This ensures low latency and high throughput, thus enhancing user experience. Additionally, the system is designed to scale seamlessly under peak traffic conditions. A further advantage of this framework is that it is cost-effective in terms of infrastructure establishment and supports emerging technologies like Artificial Intelligence and the Internet of Things, assisting the airline industry and real-time domains such as supply chain management, healthcare, and smart cities.

The feasibility of the framework is studied using simulated real-life airline reservation scenarios. The simulation results reveal a latency reduction of up to 60%, throughput improvement of 20-25%, and user satisfaction upsurge of 25%. Edge distribution also enhances reliability by potentially reducing recovery time by more than 30%, while being cost-effective with operational utility cuts of around 15-20%. This architecture, therefore, stands as one practical and extendable solution for airline reservation systems refurbishment and beyond.

Categories: Computational Science and Engineering, Hybrid and Multi-Cloud Environments, Software Development
Keywords: microservices architecture, artificial intelligent, edge computing, cloud computing, online travel agent, kubernetes, blockchain technology, software architecture, cloud computing environment, cloud-edge computing

Introduction

Background

One of the indispensable factors in the aviation sector is the airline reservation system (ARS), which allows for the booking and management of flight reservations. These systems have gone through several phases, starting with manual processes and evolving into the current digitalized platforms that integrate various services such as ticketing, scheduling, and customer management. Nevertheless, the ARS faces its own set of challenges.

Latency

The need for real-time data processing is critical in an ARS in order to provide the most current information on the availability of flights, their prices, and schedules [1]. High latency may cause delays in booking confirmations, thereby upsetting customers. Conventional centralized structures often find it hard to meet such real-time demands, leading to longer response times [2].

Data Loss

An ARS processes and provides a lot of information from several sources such as passenger data, flight scheduling data, and pricing data [3]. The problem arises when there is a need to process this data in a timely and efficient manner. However, older systems may have limitations and cannot perform optimally during data fetching, which causes delays in information processing [4].

How to cite this article
Barua B, Kaiser M (July 19, 2025) Optimizing Airline Reservation Systems With Edge-Enabled Microservices: A Framework for Real-Time Data Processing and Enhanced User Responsiveness. Cureus J Comput Sci 2 : es44389-025-05713-8. DOI https://doi.org/10.7759/s44389-025-05713-8

Scalability

The operations of most ARS need to scale due to trends in the airline business and fluctuating travel demand. Old-fashioned architectures often struggle to independently serve the system's components on demand. This inflexibility can lead to system failures or poor service quality during certain periods.

Tackling these challenges is imperative for improving the performance and dependability of airline booking systems. Contemporary architectural styles like microservices and edge computing can help make systems more adaptable, scalable, and quicker to respond, all of which are beneficial.

To better understand the context of our work, we briefly define the key communication technologies used in this study. In this paper, we focus on 4G and 5G communication technologies. The high-speed data transfer capabilities of 4G contribute to low latency, supporting mobile internet applications such as video streaming and VoIP. The latest generation, 5G, further enhances these capabilities with ultra-low latency, high bandwidth, network slicing, and massive IoT connectivity support.

Older generations such as 2G and 3G come with higher latency, lower data rates, and limited support for real-time applications. These limitations pose challenges for modern applications that require high-speed and reliable communication, which led us to focus on newer-generation technologies.

Problem statement

The conventional ARS also puts emphasis on monolithic systems, which have delineated various difficulties in meeting the real-time challenges and providing the desired user interactivity [5]. In these systems, the components are largely interdependent, which leads to the following defects:

Scalability Issues: Monolithic designs have difficulty in scaling as expansion means copying the whole system instead of particular parts. This can be wasting resources and heightening operational expenses, especially during the seasons of high demand travel.

Latency and Performance Bottlenecks: In-built performance and latency constraints are associated with systems composed of tightly integrated components. Performance degradation in one component affects the entire system due to interdependencies. This leads to inter-component waiting times and a decreased user experience caused by slower system responses.

Limited Flexibility and Agility: Due to the nature of these systems, once updates or new features have to be installed in the systems, the whole application has to be put off and redeployed after the changes have been made. This limits reconsideration of the current conditions to changing conditions posed by users and marketing activities [6].

Maintenance and Reliability Challenges: Maintaining a monolithic system is difficult due to its architecture and the complexity of upkeep. When one module fails, the entire system can fail, thereby compromising the reliability and availability of the reservation service [7].

It is important to overcome these drawbacks in order to improve both the performance and user satisfaction of ARS. Adopting a more modular and scalable approach, such as microservices and edge computing, to address real-time processing requirements is a promising strategy.

Objectives

The main aim of this research is to create a competent system based on edge computing and microservices architecture to mitigate the issues encountered in conventional integrated airline reservation systems. In this context, significant attention is given to factors such as scalability, latency, and user responsiveness [8]. The research intends to achieve the following objectives:

Improve System Scalability: A microservices-based architecture will be deployed to enhance system scalability, allowing the components of the ARS to be scaled up or down based on the size of the pooled resources without causing wastage [9].

Reduce Latency Through Edge Computing: Edge computing will be implemented to enable data processing at the edge of the network rather than at the center, ensuring that end users receive and transfer relevant data quickly. This approach aims to alleviate network challenges during peak usage periods [10].

Improve User Responsiveness and Experience: The framework will be designed to ensure minimal waiting time by clustering processing away from end users and efficiently managing real-time data, thereby making booking easier and quicker.

Increase Flexibility and Adaptability: Through microservices, this research aims to enable a design where services can be altered or updated without affecting the entire system. This flexibility supports progressive enhancements to meet the current user requirements [11].

Optimize Operational Costs and Reliability: With edge computing and a distributed system design, greater reliability can be achieved by reducing reliance on centralized servers. This may also lower costs associated with dedicated servers, which are prone to downtime risks [12].

The expectation is that this design will resolve the existing constraints in present-day ARS by leveraging the benefits of both edge computing and microservices to create a system that is flexible and responsive to real-time requests.

Contributions

This paper presents some important advancements in the studies of ARS and distributed computer systems.

Edge-Enabled Microservices Framework

Proposed a novel architecture that integrates edge computing and microservices to reduce latency and improve responsiveness in real-time ARS.

The architecture addresses the need for low latency and higher responsiveness by delegating time-sensitive computations to edge nodes while using cloud backends to ensure global data coherency.

Demonstrated Performance Gains

Simulations under high traffic conditions showed a 60% reduction in latency and a 20-25% increase in throughput, validating the framework's efficiency.

Satisfaction importance parameter improved by 25%, stressing on the better end users experience offered especially by the framework.

Modular and Scalable Architecture

Employed Kubernetes for dynamic microservice orchestration, enabling elastic scalability based on user demand.

This was achieved by the use of KAFA for message controlling and data restoring between the edges distributed nodes and central cloud in real time.

Addressing Key Challenges

Solutions were outlined to address issues such as bandwidth challenges, maintaining data consistency across divergent locations, and making the best use of available resources among others edge resource management, CRDTs, and storage hierarchy.

Cross-Domain Applicability

Highlighted the framework's adaptability to other latency-sensitive domains such as logistics, healthcare, and smart cities.

Created a launch pad for new technology incorporation that includes AI-based decision-making, IoT interfacing, and blockchain technology for secure and smart functioning.

Such inputs together enhance the modern-day reservation systems, raising the levels of operational efficiency, scalability, and customer satisfaction in real-time systems to a different standard.

Literature review

Existing Reservation System Technologies

Given the increasing technological penetration in systems used for airlines reservation, the airline industry has greatly changed the way it handles bookings management, inventory control, and the services offered to the customers. In the past, such systems have progressed from manually operated systems to modern automated systems, which increase operational speed and customer satisfaction [13].

Traditional computer reservation systems: In the Initial stages, airlines used computer reservation systems to organize their flight schedules, look for seat availability, and record bookings [14]. These systems contained reservation databases of all the interested parties and allowed the agent using it to see the information in that database in real time. For example, the SABRE system invented in the 1950s was one of the first systems of this type, and it allowed implementing B2C sales and carrying out inventories.

Global distribution systems: With growth in the airline sector, there was a demand for even wider distribution, and thus global distribution systems were created. Global distribution systems such as Amadeus [2], Sabre, and Travelport combined the reservation system of numerous airlines in a single tower for use by travel agents to enhance access to the respective airlines and their packages. These systems allowed airlines and travel companies to communicate easier and hence made the booking process more efficient.

Passenger service systems: In the present day, airlines use all-encompassing passenger service systems, which include different modules like reservation, inventory control, departure control, and customer management, among others [14]. These systems are designed to be fully operative in managing all passenger services, increasing operational productivity and customer delight. Take for instance the Amadeus Altéa Suite, which offers airline reservation, inventory management, and departure control system management tools in all aspects to the airline industry.

Microservices architecture: At present, most of the airline systems have evolved to adopting the microservices architecture that allows for better system scalability, flexibility, and even resilience. This method allows development teams to take large, complex lumps of code known as monoliths, and break them down into smaller services, which work independently and can be developed, deployed, and scaled even on their own [14]. Such architectures support continuous integration and continuous deployment of the system, enabling the airlines to quickly adapt to market changes as well as the needs of the clientele [15]. For instance, a company called Radixx designed a PSS for low-cost carriers based on microservices; thus, it is incorporated with a flexible scalable design [16].

Cloud-based solutions: The revolution of cloud computing has provided yet another advancement in the operating airlines reservation system. These are the systems that are based on the cloud, which are very scalable, cheap, and secure. Cloud computing allows airlines to run their reservation, payment, and even the storage of passenger information on the internet efficiently. As an example, this kind of service is offered by AWS, which has serverless architectures for companies in the airline industry, where they can create a reservation system for their clients without the need for many services.

Microservices Architecture in Distributed Systems

Microservice architecture is becoming increasingly popular in the area of distributed systems. This is because it is easier to separate applications into smaller deployable components that can each serve a business function [17]. This means that these services can also be developed and consumed using lightweight protocols, such as REST over HTTP or gRPC, thus making it possible to use different technologies in the same application.

Deployed microservices have the following advantages over monolithic deployment; the most obvious ones are superior scalability, improved fault isolation, and over all increased agility. Services on their own can be adjusted to different levels of scale based on varying requirements, hence ensuring better utilization of resources and operational efficiency of the whole system [18]. In addition, fault isolation is another main advantage, provided that failure of one service does very little to other services, increasing the overall reliability of the system [19]. Microservices also enhance agility in development as diverse teams can deploy and update services installed on the system in no time, therefore launching new features within less time [20].

However, that does not mean that the microservices architecture can be used without challenges [21]. There are drawbacks and difficulties in implementing microservices. As the number of services increases, there is a need to control the level of complexity by devising better communication and management of data across the services [22]. This challenge leads to difficulty in the maintenance of data consistency across the geodistributed services [23]. Additionally implementation of testing and debugging is difficult because of the distribution of the microservices, and this requires the use of highly complex testing strategies and tools [24].

Edge Computing in Data Processing

The need for low latency and high responsiveness has made the distributed system paradigm known as edge computing quite appealing [23]. Edge computing seeks to eliminate the challenges of transmitting excessive amounts of data to the centralized cloud servers by carrying out data processing closer to the source. In this paradigm, the offloaded computational capacity is either in devices or local servers that are

located close to the data source for efficient processing and insights retrieval.

Role of edge computing in reducing latency: Latency reduction is one factor that drives the adoption of edge computing in most industries. In a typical cloud computing model, which is centralized, the user may need to send data and that data may be stored miles away. It, therefore, leads to latencies when the data is being retrieved. When edge computing is utilized, data is processed at the point of its origin, in that case drawing out the latency levels that are common with many retrievals. Considering for instance, industries like transportation or healthcare, where every second on decision-making counts, edge computing facilitates 'on the spot' processing, thereby enhancing safety and efficiency in the operations [25,26].

Improving real-time data processing: This is further enhanced by the edge computing model, which allows advanced data analytics to be performed with minimum dependence on the network bandwidth and with almost immediate availability of relevant information [27]. This is important in the case a network is intermittently connected or where the network is less resourceful. In smart cities, for traffic control for example, edge devices can be embedded in the system such that data is processed on the device and does not require any central control to implement adaptive signal control or real-time traffic management [28].

Gaps in Existing Research

The combination of edge computing and microservices in ARS is still in its infancy and therefore has quite a number of research gaps:

Limited adoption of edge computing in airline systems: Most ARS are designed and operated on either a central or cloud based configuration, resulting in latency and inefficiencies especially during the busy periods [29]. However, edge computing has been applied in a few other areas, and therefore there are challenges in reviewing the existing literature especially within the airline sector.

Fragmented microservices architectures: One of the prevalent challenges in many airline systems that have adopted microservices is their focus on the vertical and horizontal extensibility of components and edge nodes, without proper integration with central edge nodes that minimize latency and enhance time-critical functions. The communication barrier between edge nodes and microservices creates limitations in fully leveraging the benefits of edge computing [30].

Challenges in real-time data synchronization: Most of the available research considers the scalability of the microservices, forgetting the issue of how the edges and the core system work together maintaining live data without synchronization faults [31]. This has a negative effect on the system that in turn creates problems in having the users receive the most current and real-time updates on flight status, availability, and pricing of seats [32].

Insufficient focus on passenger-centric performance metrics: In most cases, such work centers on the efficient functioning of the system and lowering the costs, whereas there is little attention to such indicators as the user interaction and their experience with the system, which are important for airline ticketing systems [33].

Limited case studies and experimental data: Few, if any, case studies or experimental setups exist that demonstrate how edge computing and microservices can be effectively combined in contexts such as airline ticket purchasing. Most studies present a theoretical perspective, leaving the practical implementation and assessment of actual systems largely unaddressed [34].

Materials And Methods

Methodology

This study focuses on approaching the problem of real-time data management in a new way by creating a framework utilizing edge computing and microservices architecture. The suggested architecture is composed of three layers: the Data Collection Layer, the Edge Processing Layer, and the Cloud Integration Layer. Data is collected from Internet of Things (IoT) devices and applications and processed at the edge using dedicated microservices such as filtering, aggregation, and local storage, which minimizes latency and allows instant action [32]. The data is processed and then sent to the cloud for further analysis and storage, thus offering an elastic and cost-effective approach to deal with the issue of real-time data processing over a geographically dispersed system. This architectural setup provides high data processing efficiency at minimal latency, optimizing the use of cloud and edge resources. To ensure secure data transmission and processing, the framework incorporates security measures such as data encryption (both at rest and in transit), authentication mechanisms for IoT devices, and role-based access control for microservices [35].

Framework Design

In order to enhance the processing of real-time data, the provided architecture combines edge computing and microservices architecture. Microservices designed for filtering, aggregation, and data analysis include:

• Data Collection Layer: Edge devices, including sensors and IoT devices, perform data collection and some level of data processing at the source.

• Edge Processing Layer: Instead of sending all data to the cloud, data is locally processed at edge nodes to minimize latency using microservices designed for filtering, aggregation, and analysis.

• Cloud Integration Layer: The examined and analyzed information is stored in the cloud and, if necessary, integrated with existing datasets for further processing and organization.

The architecture of edge-enabled microservices for real-time data processing is shown in Figure *1*.

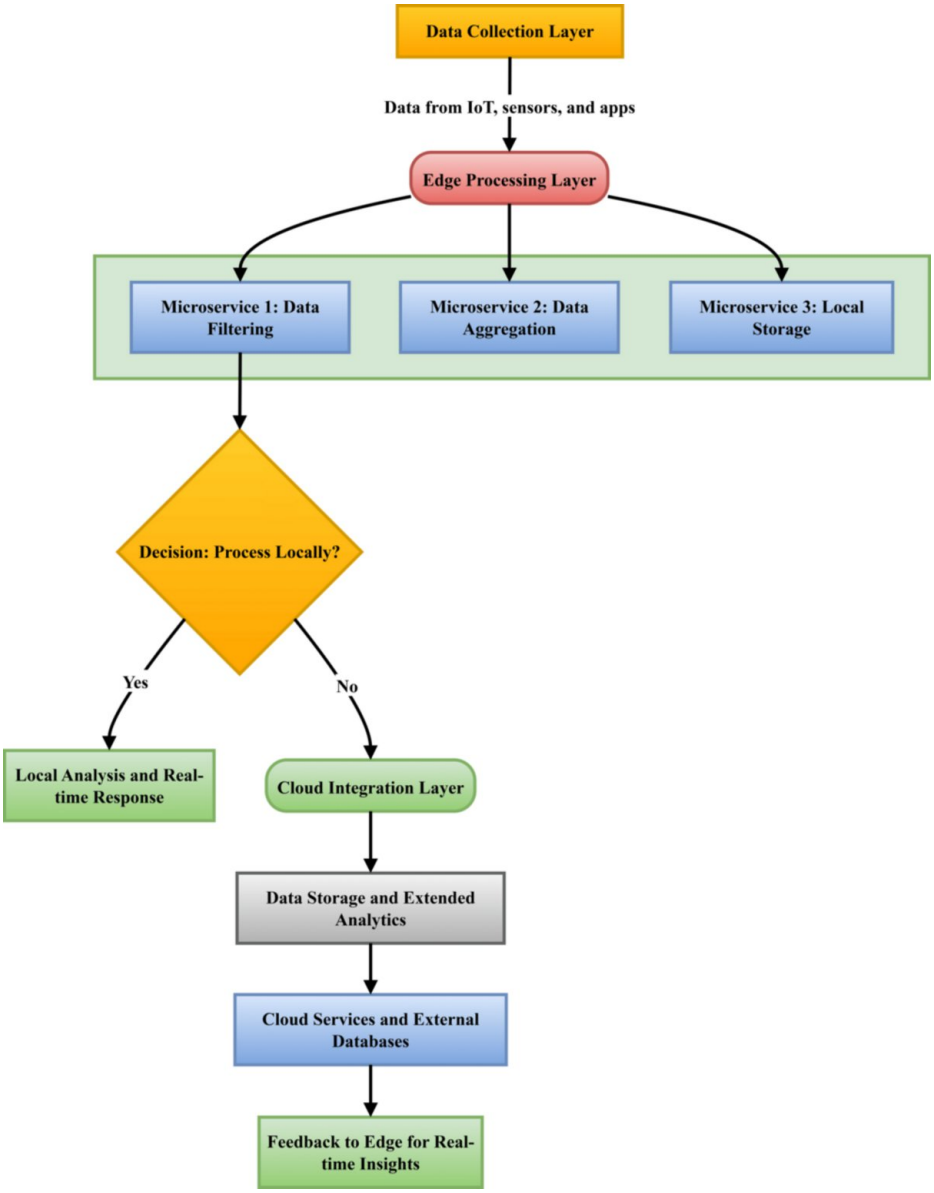

FIGURE 1: The architecture of edge-enabled microservices for real-time data processing

Data Flow

• Data is produced from a variety of channels such as IoT sensors, applications, and user contributions,

and this data gets sent to the edge layer.

• The edge layer quickly processes the information via dedicated microservices, such as the filtering microservice, which removes redundant, irrelevant, or noisy data in order to reduce the processing load and network usage; the aggregation microservice aggregates data points over specific time intervals or per type (e.g., average temperature readings over a minute) in order to reduce data volume and promote trend detection; and then the local storage microservice temporarily stores processed data on the edge for and assures resilience, retry mechanisms in case of network disruption, and real-time local access for immediate decision-making.

• The cloud is used to carry out further analysis besides storage, with the processed information forwarded to the edge when appropriate.

Overview of Figure *1*:

1. Data Collection Layer performs the work of collecting data from sources such as IoT devices and applications.

2. The next layer, which is hanging at edge nodes, embraces:

• Data Filtering Microservice, which serves to eliminate unnecessary data.

• Data Aggregation Microservice, which serves the purpose of consolidating the data for localized purposes.

• Local Storage Microservice, which is used also to cache the processed data when there is a need.

3. A Decision Node follows these microservices, which looks into the next action and determines if additional processing of data is necessary

4. If Yes, then Local Analysis and Real-time Response become the next destination.

5. If No, then the data is directed to the Cloud Integration Layer, where the Data Storage and Extended Analytics (I) activities take place.

6. Cloud Services and External Databases manage, store, and integrate data for analysis, while also feeding edge nodes with information to maintain real-time analysis as activities continue. These cloud services and databases support data integration and analysis, ensuring edge nodes receive relevant updates to uphold real-time processing. For example, historical and processed data may be stored in external databases such as Amazon RDS (Relational Database Service) or Google Cloud Bigtable, enabling efficient queries and integration with analytics services.

System Requirements and Components

The design described in the edge-enabled microservices in Figure *2* must take into consideration how hardware, software, and network resources will be used in the real-time processing of data.

Hardware requirements:

• Edge Devices: These are general data collection devices like IoT sensors, cameras, and portable devices.

• Edge Servers: These servers are brought closer to the user and contain sufficient CPU/GPU (for intensive workloads) and RAM for microservices required for processing the data as it arrives.

• Cloud Servers: Servers that are very much distributed and set up within defined geographic boundaries, for example, set up within data center regions established by cloud providers (like AWS Regions, Azure Regions, or Google Cloud Zones). These boundaries are drawn in view of things such as data residency laws, latency considerations, regulatory causes, or service considerations. Thus, for example, servers could be deployed within the European Union to satisfy GDPR requirements, or simply within the boundaries of one country to meet national requirements of data sovereignty, while also catering to low-latency requirements from the user base in that region..

Software requirements:

• Microservices Framework: Kubernetes or Docker for deploying and managing the microservices

• Edge Analytics Software: Apache Kafka or equivalent for streaming data and processing

• Cloud Analytics: Subordinate services offered by AWS, Google Cloud, or Azure that provide storage and large-scale analytics capabilities.

Real time Processing Engine: Data ingestion, processing, and analysis platform such as Apache Flink or Spark Streaming, which allows data to be processed in real time as it is received.

Network requirements:

• Local Area Network: Aiding the edging servers with the necessary high capacity physical communication links to the infantry devices to minimize the latency incurred between the edge devices and the edge servers.

• Wide Area Network: Cables that provide interconnection of numerous edge devices and cloud servers with high service quality for facilitating data transfer in a single datacenter [36].

• 5G/4G Connectivity: 4G and 5G wireless technologies provide high-speed, low-latency, and massive data-transfer rates for the operation of real-time mobile IoT devices. Fourth-generation wireless technologies offer data rates up to 1 Gbps with latencies around 30-50 ms, while 5G offers very high-gigabit rates of 10 Gbps, with latencies down to around 1 ms [37].

Some of the main features of 5G are ultralow latency for near-instantaneous responses; enhanced mobile broadband with ultrafast internet connectivity; massive machine-type communications, allowing for scaling of IoT networks to great numbers of connected devices; and network slicing that builds several virtual networks on the same physical facility depending on the specific use case.

Compared to 3G, in which data rates were considerably limited (up to ~2 Mbps) and latency was rather significant (~100-500 ms), 4G and 5G enable huge improvements in terms of system capability, scalability, responsiveness, and reliability.

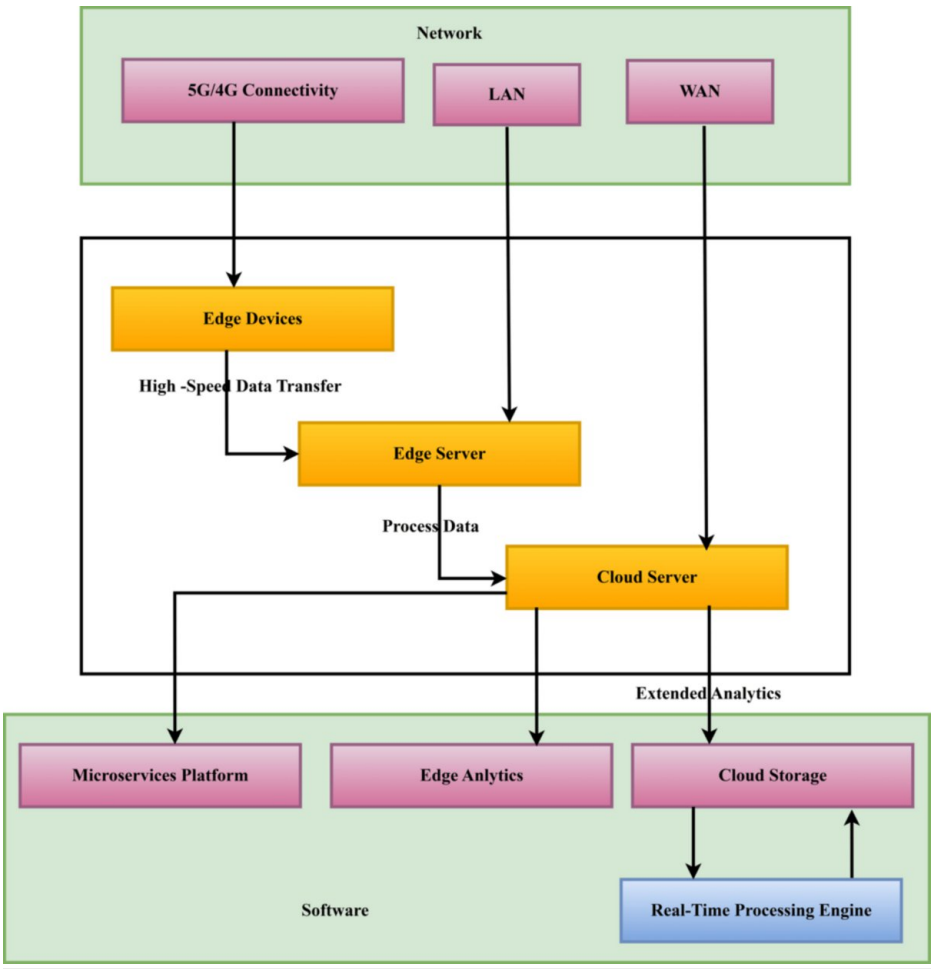

FIGURE 2: The diagram of system requirements and components

LAN, Local Area Network; WAN, Wide Area Network

Implementation Details: Data Processing Pipelines

A data processing pipeline placed at the network edge is intended for time-critical data, real-time data with minimal delay. The following is a basic sequence of steps the procedure undergoes. The real-time setup was validated through performance testing, where end-to-end latency, from data generation at the IoT device to data processing at the edge, was measured. In typical test scenarios, the latency consistently remained below 50 milliseconds, reflecting the characteristics of a real-time system.

• Data Collection: Edge devices (e.g., sensors, IoT devices) take raw data and forward it to an edge server for initial processing.

• Data Filtering: Raw data is searched through for eliminating noise and unrelated information, so as to lessen the burden of downstream processing. The filtering procedure is governed by certain specified rules and thresholds imposed on sensor readings, e.g., wiping out points outside expected ranges or smoothing rapid variations probably due to sensor errors. Before further processing, data-cleaning tools are also used: low-pass filtering or statistical outlier detection.

• Data Aggregation: Some data points, which are called relevant, are agglomerated towards a small set, which enhances the speed of processing and will also speed up the storage of such data.

• Local Analysis: The processed data is not sent to a data center; instead, it is analyzed locally to provide near-immediate insights, enabling timely actions where needed.

• Temporary Storage: Data is temporarily stored in a local cache after it is cleaned and processed. Usually, such a cache is on an edge device or a neighboring edge gateway. This localized caching approach supports quick data-level access, storage, and retrieval for real-time processing or decisions, which a

remote cloud server may delay.

• Cloud Syncing: Some processed data, especially anomaly alerts or summarized metrics and event logs, are sent to the cloud. This is normally done based on predefined rules and thresholds that determine which data are worthy of long-term storage, further analysis, or integration with other datasets existing in the cloud.

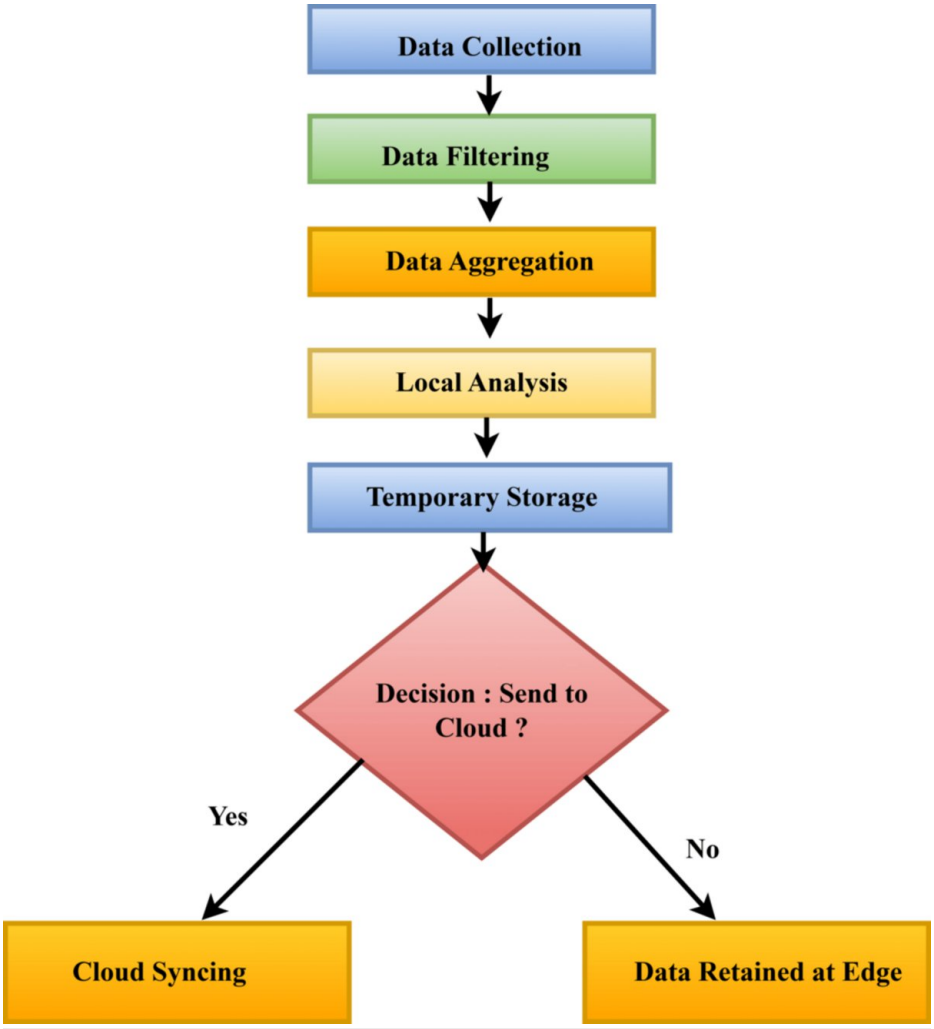

FIGURE 3: The diagram of implementation details: Data processing pipelines

Overview of the data processing flow: This flow in Figure *3* allows for real-time data processing at the edge without latency and excessive network utilization as only process data is synced with the cloud. The pipeline has been designed in such a way that there is high responsiveness to localized events, low reliance on cloud infrastructure and this way improves the responsiveness of system as a whole.

• Data Collection: In this step, edge devices gather data and relay it to the edge server.

• Data Filtering: Filtering limits the incoming raw data to the relevant data only without unwanted noise, hence enhances the processing flow.

• Data Aggregation: Aggregates data points that are useful to come up with a sizeable data set.

• Local Analysis: With the data aggregated, it can provide insights straight away, therefore enabling real-time action.

• Temporary Storage: Quick storing in this section is aimed at keeping processed information for some time.

• Decision Node: This indicates the option of either sending the information to the cloud for advanced analytics or outright storage in the devices.

• Cloud Syncing: Processed data selected for long-term storage or advanced analytics is synchronized with the cloud through a secure and efficient mechanism. The data-sync is usually event-driven or scheduled periodically, depending upon its priority and volume. The data is sent through RESTful APIs or Message Queuing Telemetry Transport (MQTT) protocols, generally dispatching batch loads to minimize network overhead. This synchronization is unidirectional-from edge to the cloud-and it includes metadata to guarantee the integrity of the data and allow their traceability.

• Data Retained at Edge: Any data considered not necessary for cloud rest will remain in the edge.

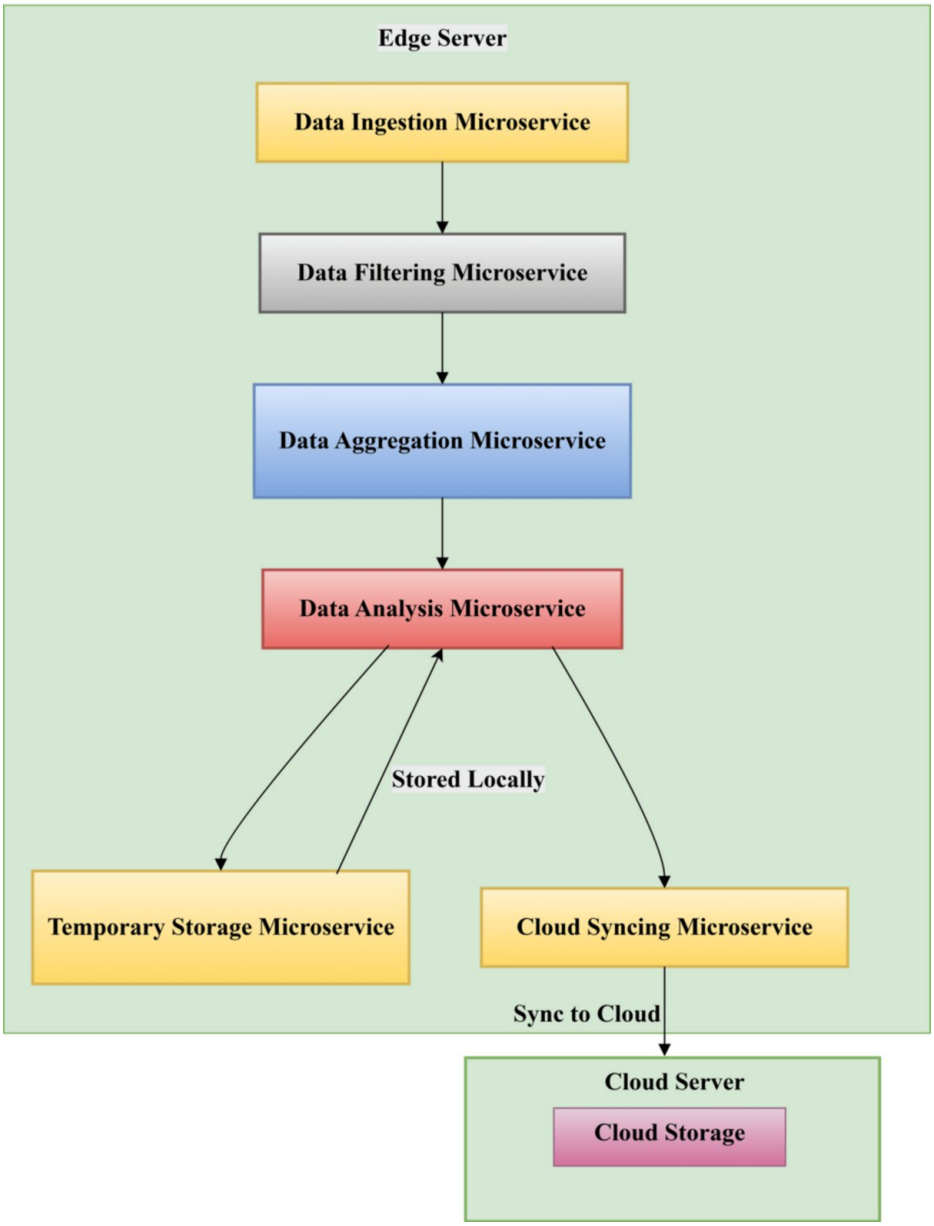

FIGURE 4: The diagram of the deployment of each microservice and their interactions within the system

Overview of the microservices deployment flow: This flow in Figure *4* is illustrative of a microservices architecture, which is modular and elastic in nature; every service performs a clearly defined task and all the tasks are well chained together in a manner suitable for real-time data processing at the edge and its efficient storage and analysis on the cloud.

• The Data Ingestion Microservice: It is responsible for receiving data from the edge devices and sending it to the filtering service.

• The Data Filtering Microservice: Entertainment Services: It works on unprocessed information by eliminating unnecessary and superfluous details and transmitting the important data to the integrating microservice.

• The Data Aggregation Microservice: Merges the useful information into a short dataset and delivers it to the service for analysis.

• The Data Analysis Microservice: Evaluates the data brought together in aggregate for real time findings. It makes use of a storage medium for holding data temporarily and makes use of cloud storage mechanism (F) for data that is meant for cloud preservation.

• Temporary Storage Microservice: Processes data and keeps it on standby while the system gets reoriented to the last accessed pieces of information.

• Cloud Syncing Microservice: Transfers the information to be stored in cloud server for purposes of storage and other analytical activities.

Edge Computing Integration

The implementation of edge computing into airline ticketing systems involves placing a number of edge nodes within a geographic area to allow for the processing of important information nearer to the end users. This approach leads to less latency and enables real time interaction. The integration upholds the following major steps:

Segmentation of the Data Flows:

• In order to turn in reservation requests, clients queue their requests at the edge nodes instead of relying on some central system.

• Accommodation is given for the operation of critical activities (such as the management of real time seat inventory and current pricing) at the edges and for non, current activities (such as the cloud storage of historical data for research) within the cloud.

Edge-Cloud Convergence:

• There is interworking of the edge nodes with the central cloud for the purposes of updates.

• Centralized systems like cloud platforms and on-premise data centers are the ones centralizing large-scale big-data analytics using high-capacity storages and powerful computing resources. These central systems deal with historical trend analysis, training machine-learning models, data warehousing, and so on. On the contrary, edge systems are built for time-critical tasks, such as the real-time detection of anomalies and instant reactions to local events.

For instance, in an industrial IoT environment, edge devices on factory machinery detect temperature anomalies in real-time and issue immediate alerts. At the same time, all sensor data is uploaded to the central cloud system - AWS or Azure - for heavier analysis, where, over months of data, a Spark cluster might train a model for predictive maintenance. This model is then pushed down to the edge for real-time application.

Load Balancing and Dynamic Scaling:

• The edge architecture supports elastic scaling during peak demand by offering modular hardware and containerized software deployments. New edge nodes or devices can be added to the network dynamically, and microservices running on the edge servers can be horizontally scaled to these nodes through automated deployment tools. This is usually done by lightweight orchestration tools such as K3s (a lightweight Kubernetes distribution) that allow new services or instances to be deployed near the data source without taxing existing edge resources.

In a retail-use case, more edge servers would be deployed during the holiday season in stores to handle local processing of increased foot traffic and transaction data with low latency and uninterrupted performance. The excess resources would then be decommissioned or repurposed after the demand subsides.

• Load balancing aims to distribute incoming data and computational tasks evenly across multiple edge servers so none of the nodes are overwhelmed under any condition and optimal performance is maintained by each server. Load balancing algorithms such as round-robin, least connections, or resource-based scheduling keep track of the status and capacity of each edge server and route tasks to the server best suited for that particular task.

Edge load balancers or gateways receive data streams from outside and then check for CPU load, memory usage, and queue length of tasks. They then push the task upfront to the edge server with maximum resources available. This provides high availability, low latency, and real-time processing based on varying load conditions.

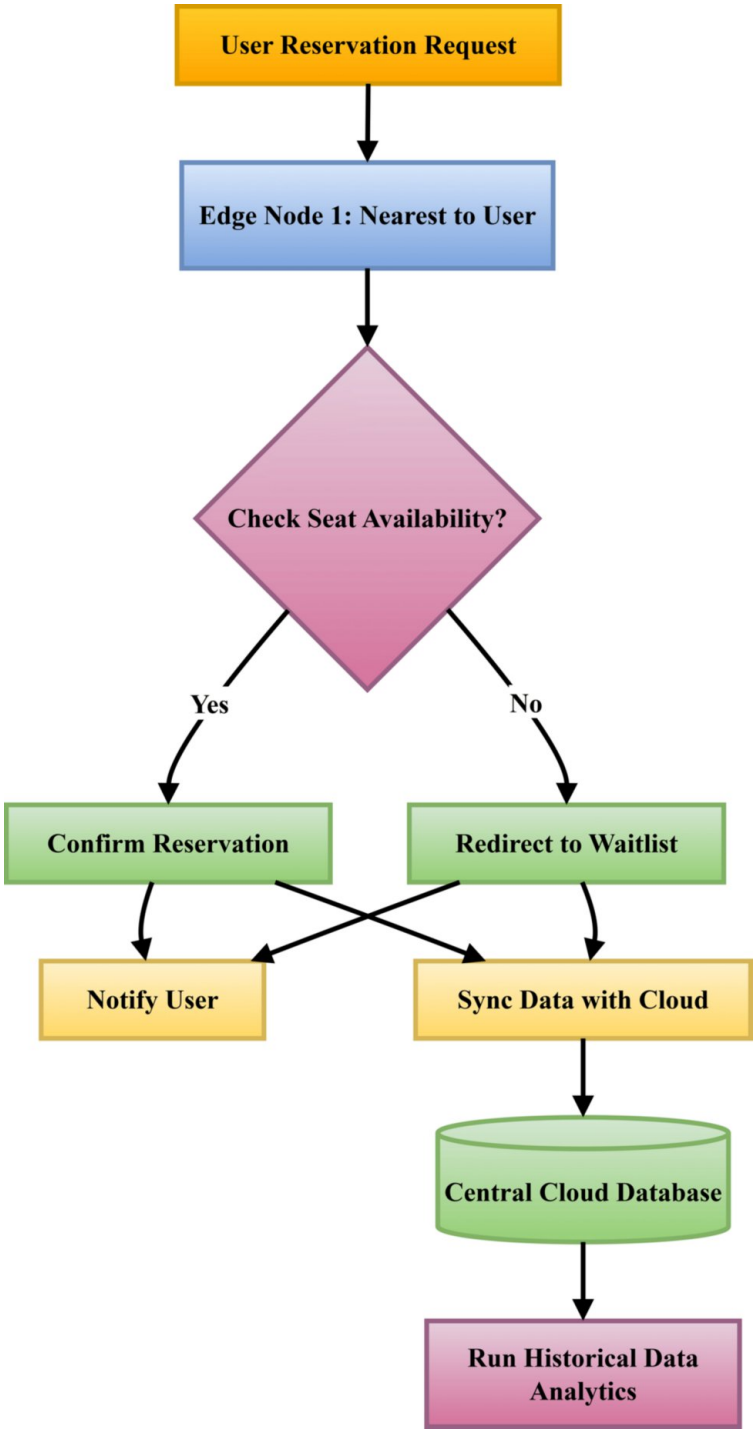

FIGURE 5: The edge computing integration

Overview of Figure *5*:

1. Request Fulfillment of the Users: To minimize the lag, user requests are always directed to the closest edge node.

2. Edge Node Smart Service Processing: Important functions such as checking the availability of seats and reserving them are completed at the edge.

3. Cloud and Edge Device Interoperability: The last part of data (like completed orders) is uploaded to the cloud database in order to maintain consistency across regions.

This approach in Figure *5* improves system responsiveness, reduces latency, and balances the computational load between edge and cloud systems.

4. Role of the Central Cloud: The central cloud takes care of all the non-operational activities that are needed at the facility for example data analytics over large pots of data and forecasting.

Implementation and tools

Technology Stack

The technology stack that would be required for an edge-aware microservices architecture (Figure *6*) and the ARS would be:

Containerization:

• Tools: Docker, Kubernetes.

• Microservices, which are small, self-contained applications virtually built, are packaged and deployed using isolating containers which also allow extending and shrinking resources spread across an edge and a cloud.

Edge Computing Platforms and Tools:

• Tools: Google Anthos, Azure IoT Edge, and AWS Greengrass.

• These platforms as well offer edge compute capabilities for users to process their data nearer to them in order to eliminate or reduce latency.

Messaging and Data Streaming:

• Tools: Apache Kafka, MQTT.

• Messaging and streaming systems support edge and cloud nodes by enabling real time communication and coordination.

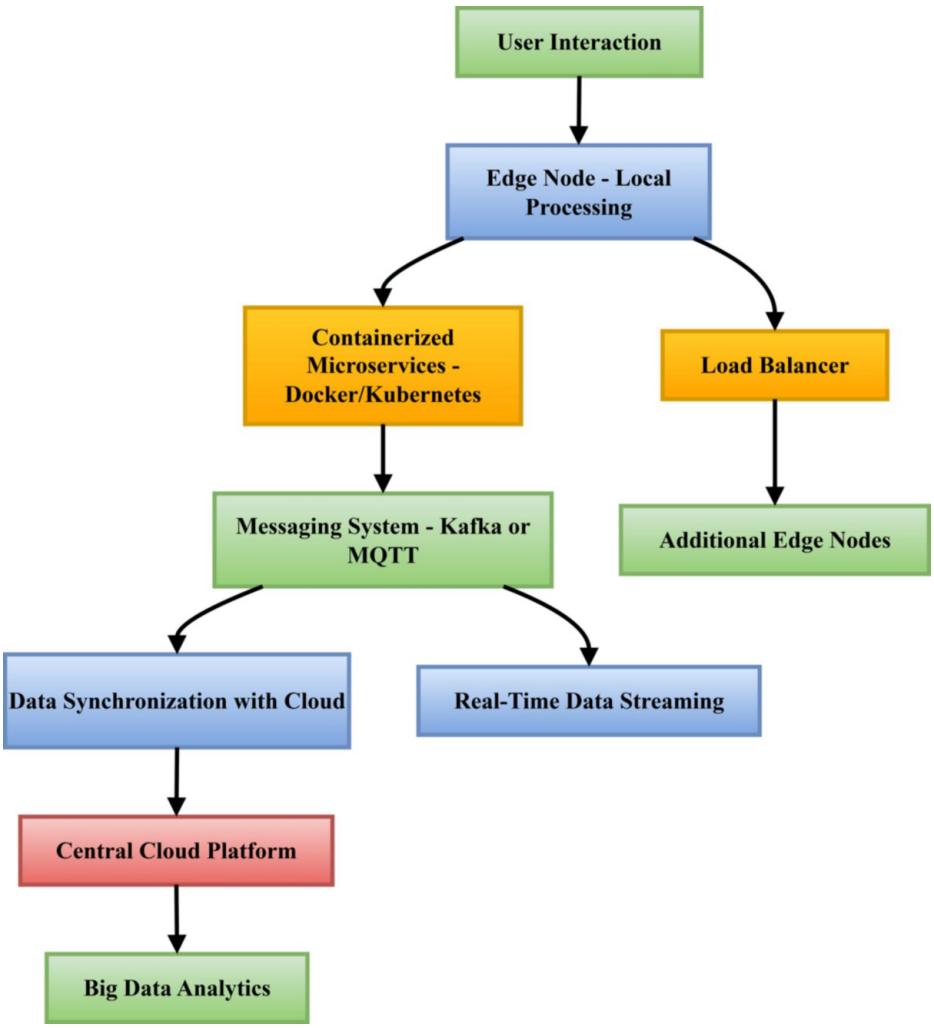

FIGURE 6: The technology stack for implementing an edge-enabled microservices architecture

Overview of Figure *6*:

1. User Interaction: Users make reservation requests or inquiries, which are sent to the closest edge node.

2. Edge Node Processing: Containers of microservices are located in edge nodes and provide real-time updates on seat availability and prices.

3. Messaging and Streaming: The data are sent to the central cloud or other edge nodes by means of Kafka or MQTT for timely updating and analytics.

4. Cloud Integration: Less important data is transferred to the cloud for reasons of big data processing and storage.

5. Load Balancer: Provides a mechanism for load balancing by distributing incoming user requests evenly across edge nodes.

Advantages of the technology stack:

• Containerization: Provides ability for quick, elastic deployment of microservices, and maintaining their homogenous environments.

• Edge Computing: Provides faster response times and improved experience for users.

• Messaging Middleware: Provides assured and timely services in communication in large scale distributed

systems.

Integration Strategies

Incorporating edge appliances with cloud infrastructures and modernizing their environment to microservices comes to a careful consideration and a step-by-step procedure. Some strategies are listed are as follows:

Associating Edge Devices and Cloud Backends

• Strategy: Implement data protocols for the edges and provide connections to the cloud.

• Steps:

1. Create Systems and APIs for them to be synced in real time.

2. Employ edges where data is regarded as processed and most importantly reduce cloud to avoid time waste.

3. Conduct a data check at intervals, and dump data into the cloud to maintain data alignment.

Upgrading from Traditional Applications to Microservices

• Strategy: Appetite for removing conventional systems a little at a time and replacing it with deploy -able microservices.

• Steps:

1. Do not begin with the critical modules but as the project is gradual, start with microservice refactoring of non-critical modules.

2. Services should be deployed independently through containerization with tools such as docker.

3. Use Kubernetes an orchestration tool to take care of deployment and scaling.

Figure *7* presents a flowchart illustrating linear parallel processes. The vertical processes represent the migration of existing legacy systems to microservices and the integration of edge devices with cloud backends.

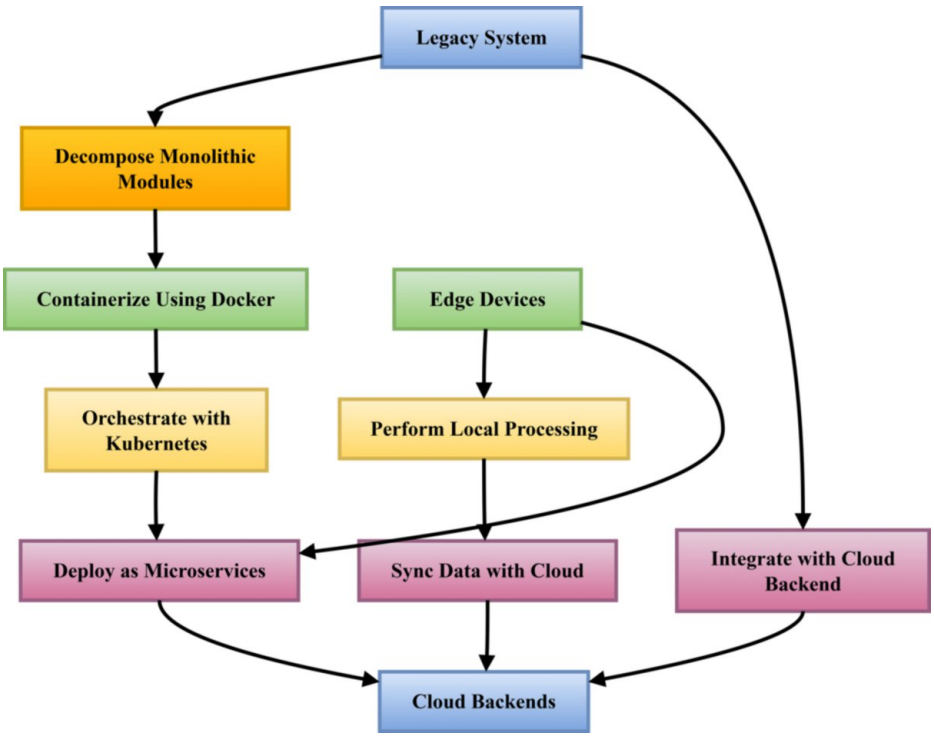

FIGURE 7: Migrating legacy systems to microservices and integrating edge devices with cloud backends

Development Workflow

The DevOps perspective is focused on, and more importantly, automating the process of software delivery and maintaining system health through CI/CD pipelines. Additionally, it incorporates monitoring and observability concepts. This approach ensures that microservices are provisioned frequently, and very little time is lost with provisioned resources as system visibility is at no point compromised.

CI/CD pipelines for frequent deployments: The process to prepare, test, and deliver or otherwise deploy a software product is articulated in a way it supports frequent software versions. The CI/CD pipeline is structured around the following steps:

• Build: Application source code is written, and containerized applications (docker) built.

• Test: Unit, integration, and end to end tests - all performed automatically.

• Deploy: Deployment and management of applications done through Kubernetes.

• Watch: Monitor the deployment constantly and initiate a rollback in case of need.

Monitoring and observability tools: Development of systems for health checks, system performance analysis, and general system production support, such as:

• *Prometheus* for metrics collection [38].

• *Grafana* for dashboards and visual representation [39].

• *ELK* (Elasticsearch, Logstash, and Kibana) for logs and data searching [40].

• *Jaeger/Zipkin* for traceability across distributed systems [41].

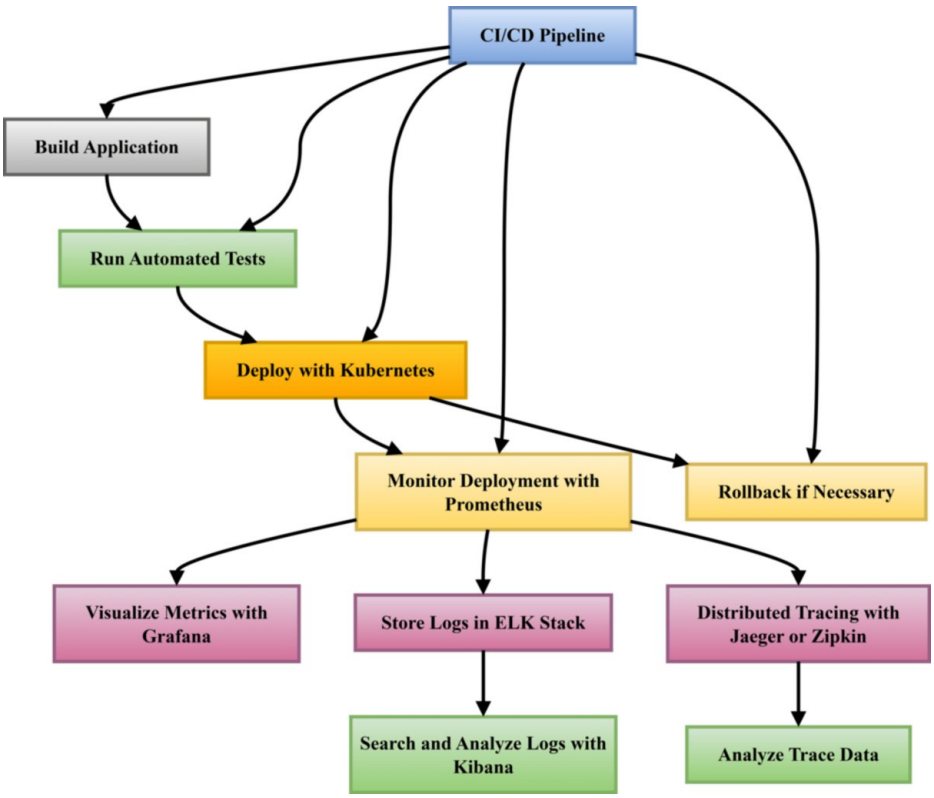

FIGURE 8: The CI/CD pipeline for continuous delivery and the observability tools

CI/CD, Continuous Integration/Continuous Delivery/Deployment

Overview of CI/CD pipeline (Figure *8*): Figure *8* illustrates the continuous delivery CI/CD pipeline and the corresponding observability tools for system health checks.

1. Build: Code is built into deployable units (e.g., Docker images).

2. Test: Automated tests (unit, integration) help to ascertain the quality of the code.

3. Deploy: Deployments across environments (staging/production) are automated by Kubernetes.

4. Monitor: Continuous Monitoring is established with Prometheus and alerting.

5. Rollback: Kubernetes deployment rollback mechanism ensures that the user is taken back to a previously stable version automatically if a deployment fails.

Monitoring and observability:

• Metrics are gathered and system status check is performed with Prometheus [38].

• Metrics with their interactive dashboards are presented by Grafana [39].

• To assist in issue resolution and system performance management, the ELK Stack is used for log monitoring [40].

• Request tracking and service performance watching is done using Jaeger/Zipkin by monitoring all requests and responses between services.

Evaluation Metrics

One can consider different metrics to assess an edge-enabled microservices framework. Latency refers to the time, which in this case is the period between the generation of information and the process of that

information, with lower latency supporting better real-time interaction. Data Throughput provides a measure of how much data is processed within a given time period and represents the ability of the system to sustain significantly high data volumes with no interruption [31]. System Response Time encompasses both the latencies of the system and the speed of its data processing thus rendering an overall rating of the system which is important in applications that require instant response. Finally, User Satisfaction includes the evaluation of the performance of the system from the eyes of the end-user which in particular are satisfaction scores and completion rates in order to ensure that the framework is able to serve its purpose [33].

Challenges and solutions

Technical Challenges

Network limitations and latency problems:

Challenges

In situations of heavy usage of various connections, offering rapid services with possible transmission between edge nodes to the cloud can, at times, render results slower than intended. Usage in some geographies with unstable/fixed bandwidth connections makes interactivity of the system difficult.

Solution

• Content delivery networks (CDNs) should be used to store the most accessed data closer to the users.

• Similar to the above, introduce adaptive bitrate streaming or other data compression methods.

• Use private leased lines or 5G connectivity to reduce latency and have more reliable connections. Since private leased lines offer dedicated uncontested bandwidth, there is no traffic coming in and going out, and hence there is congestion in communication between edge nodes and the central systems and delays that are induced with the public internet routes are avoided for quicker access.

In contrast, 5G networks offer ultra-low latency (of the order of 1 ms) and enough data throughput to be able to transmit massive amounts of data in near real-time, thus providing better access times for time-sensitive applications such as real-time analytics, video streaming, or systems for making critical decisions.

• Adopt load balancing strategies that are predictable in nature to decrease congestion levels within the network, particularly at peak activity periods.

Managing the consistency of distributed data:

Challenges

• Updating different edge nodes and the central cloud backend is highly sensitive due to possible updates and conflicting situations, which makes it difficult to keep data consistency.

• Using eventual consistency models presents the risk of temporary inaccuracies of key airline booking information (e.g. where seats are available).

Solution

• Combine event sourcing with CRDTs to achieve consistency over distributed computing systems [42].

• Implement strong consistency requirements especially for critical operations like reservation and cancelation transactions to ensure consistency at all time.

• Design conflict resolution based on versioning and reconciliation in distributed databases. Conflicts in distributed databases emerge in cases where multiple updates are simultaneously propagated to different replicas, i.e., when one update is generated on a single replica while it is also concurrently updated on another one.

For this, a system provides mechanisms to assign version metadata (such as a timestamp or a vector clock) to each update. When synchronizing replicas, these versions are then inspected:

• One update is considered newer and overwrites the other one.

• Updates are in conflict (e.g., concurrent updates), triggering a reconciliation policy that may:

Last-write-wins (LWW) with timestamps,

A custom merge logic provided by the application (e.g., merging shopping cart items),

Manual resolution in critical systems.

Such a mechanism of versioning and reconciliation acts to bring replicas back into a consistent state, without sacrificing availability and partition tolerance as modeled by the CAP theorem.

• Implement data partitioning mechanisms to lessen the possibility of conflicts by allocating certain geographic regions or datasets to certain edge nodes only.

Figure 9 provides an overview of the main technical issues and the workable solutions available as way forward resolutions in order to enhance and boost the resilience as well as the system's overall performance.

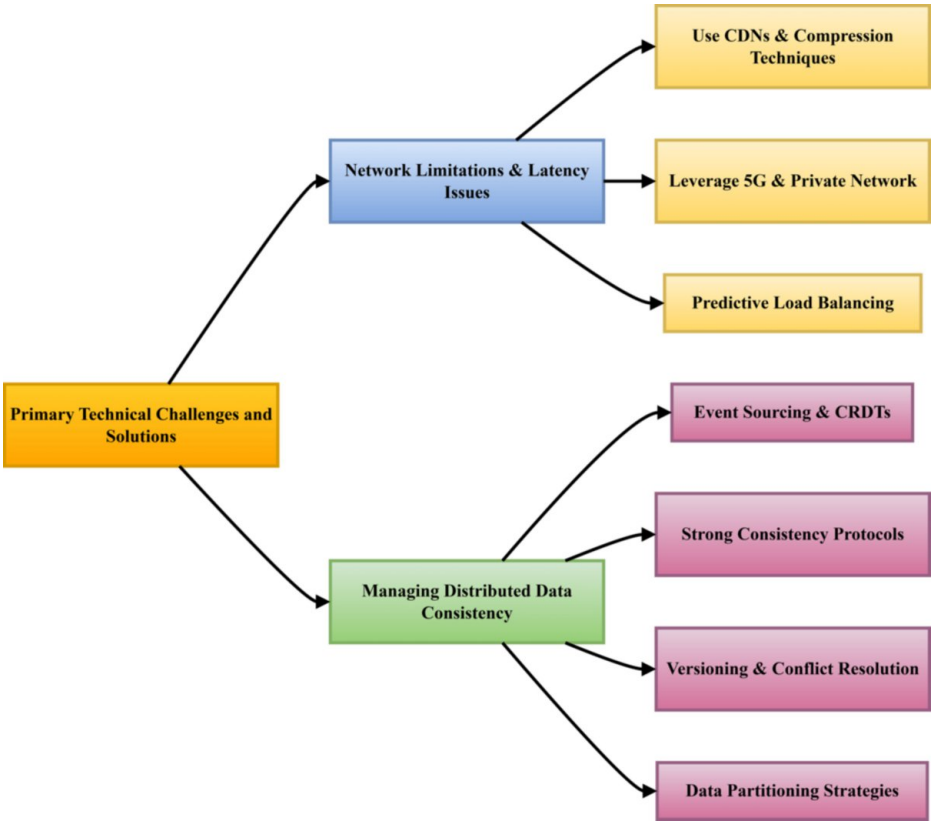

FIGURE 9: Diagram of the primary technical challenges and actionable solutions

Cost and Resource Challenges

Edge infrastructure expenditure:

Challenges

• Infrastructure at the edges of the system inclusive of edge devices and storage and compute nodes is expensive both in terms of its installation and upkeep, especially in areas that are not well endowed with connectivity.

• Therefore, dedicated funds towards purchasing hardware and software platforms as well as running operational activities puts a significant burden on the available finances.

Solution

• Make use of cloud-edge services such as AWS Greengrass [43], Microsoft's Azure IoT Edge [44] or Google Anthos [45] which can be availed in a prepaid manner, thereby decreasing capital outlay.

• Eliminate the need for building new edge infrastructure by exploiting the existing network assets in partnership with telecom operators.

• Start with high-density population areas to enjoy the benefits and costs ratios.

• Consider container-based approaches (for example: Docker, Kubernetes,...) at the edge to cut down on resources and costs [46].

Optimization of resource allocation:

Challenges

• Maintaining the right number of compute/storage resources at the edge nodes and in the cloud can be a quest.

• While under-utilized resources can be an avenue for losing money, excess resources can lead to underperformance [47].

Solution

• Employ resources resizing based on real-time demand using dynamic resource allocation algorithms.

• Employ machine learning models to foresee the peak usage periods and adjust the resources for provisioning [48].

• Storage optimization measures can be applied by using two approaches to storage where crucial information is stored at the edges and non-critical information is stored in the cloud.

• Monitor resource consumption patterns and analyze them regularly with specific tools such as Prometheus and Grafana in order to make accurate adjustments of the use of resources [49].

Figure *10* provides an overview of the cost and resource issues and the workable solutions available as way forward resolutions in order to enhance and boost the resilience and the performance of the system under consideration [50].

FIGURE 10: Diagram of the cost and resource challenges and actionable solutions

Proposed Mitigations

In order to effectively deal with the issues that have been recognized, a variety of technical and strategic mitigation measures can be undertaken.

Limitations of network and latency issues:

Mitigation Techniques

• Edge Caching: Store the data that is most frequently requested by users in the nodes at the periphery to avoid delays in recall [51].

• Content Delivery Network: Content is replicated in several places to make it easily available to the end user [52].

• Load Balancing: Employ predictive load balancing techniques in order to avoid node bottlenecks or overloading in one particular area of the network.

• Incorporate 5G: For high-capacity network locations, incorporate advanced connectivity solutions such as 5G to enhance bandwidth, reduce latency, and support edge computing capabilities.

• Data Reduction: Reduce the size of data packets for transmission purposes, especially when sending data in a network that has limits on its capacity [53].

Ensuring consistency when handling data across distributed systems:

Mitigation Techniques

• Event Sourcing: An architectural style that involves capturing changes as events and ensuring all nodes remain consistent with them within an application [54].

• CRDTs: Allow users to update data even when offline, and ensure that data modified during offline periods is automatically merged without conflicts [55].

• Strict Consistency Level: For certain types of data (e.g., reservations), requiring a lock or consensus prevents any updates from being executed until all nodes agree, effectively locking the data for the entire

process [56].

• Horizontal Data Partitioning: Synchronization constraints can be alleviated by partitioning data according to geographic or functional zones.

Cost of edge infrastructure:

Mitigation Techniques

• Serverless Edge Computing: Utilize edge services through serverless computing such as AWS Lambda@Edge to circumvent erecting new infrastructures.

• Rent Instead of Build: Work with cellular networks to install or use their edge devices and networks.

• Container Edge Solutions: Offer service in virtualized container installed on the service provider's equipment, cutting down on costs.

• Gradual Adoption: The implementation of edge infrastructure and services shall take place in phases, concentrating on geographic areas where user density is high or latency-sensitive and where network limitations prevail, thus availing a targeted and cost-effective deployment.

Resource allocation enhancement:

Mitigation Techniques

• Dynamic Scaling: Employ Kubernetes autoscaling capabilities to provision resources as the workloads fluctuate.

• Predictive Analytics: Use AI/ML models to predict traffic and provision resources if necessary.

• Tiered Storage: Use local storage at the edge for high-value data while keeping data that is less frequently accessed in the cloud.

• Real-Time Monitoring: Application of Prometheus and Grafana to enhance the utilization of resources and eliminate wastages on the reallocation of resources.

Description:

• **Network Limitations and Latency Issues:** The focus of mitigations is to optimize data flows at the edge rather than processing in the cloud.

• **Distributed Data Consistency:** Strategies focus on the management of conflicts and consistency of the nodes that are located within different geographical locations.

• **Edge Infrastructural Costs:** Costs associated with edge infrastructure expenses are lowered by gradual enhancements, cloud services, and cooperation with other service providers.

• **Resource Allocation Optimization:** Effective use of resources without wastage is achieved through the use of dynamic and predictive approaches.

This detailed mitigation plan, shown in Figure *11*, addresses the issues highlighted in earlier sections and outlines practical measures that can be employed.

FIGURE 11: Diagram of the cost and resource challenges and actionable solutions

Additional challenges (security):

• Vulnerability to Cyber Attacks: Edge devices or transmission channels can be targeted by cyber attacks. Importantly, they might suffer unauthorized access, data breaches, man-in-the-middle attacks, etc.

• Data Loss Risks: Loss of data can result due to device failure, interception, or transmission without proper encryption.

• Increased Attack Surface: In edge-cloud sync environments, differences in authentication methods and lack of centralized supervision might increase attack surface areas.

Solutions for security risks:

• End-to-End Encryption: Make sure all data traveling from an edge device to the cloud is all the way encrypted with, say, TLS 1.3 or VPN tunnels.

• Zero Trust Architecture: Follow a zero-trust approach that continuously authenticates and authorizes access controls at every level, such as the device, user, and network.

• Secure Boot and Trusted Execution Environments: Provide hardware-level protection to prevent tampering, and validate the integrity of the code on the edge device.

• Regular Security Audits & Firmware Updates: Establish an automated patching and vulnerability scanning mechanism, especially for remote edge devices.

• Data Backup & Redundancy: Make provision for failover measures and regular cloud-sync backups to prevent permanent data loss arising from edge node failures.

Results

Experimental setup

Simulation Environment

Through the realization of a pair of virtualized edge nodes and a centralized server infrastructure, the idea was to implement the simulation of standard operations within an airline reservation system. The edge nodes handled trivial tasks such as checking seat inventory, putting in booking requests, and confirmations, whereas the centralized server handled long-term data storage and non-time-sensitivity-based processes.

Performance Metrics

Latency, throughput, user satisfaction, and fault recovery time were considered during simulation as key performance indicators. Peak traffic conditions were simulated with varying numbers of concurrent users placing booking requests.

Load Balancing

Microservices were deployed through Kubernetes for orchestration while dynamically scaling as per demand. Kafka was deployed as a message broker to allow communication between the distributed microservices efficiently.

Failure Simulations

In view of testing the fault tolerance, failures were simulated intentionally on selected edge nodes by introducing conditions, for example:

1. Forced shutdown of edge microservices or containers

2. Network isolation (simulated disconnection from central services)

3. CPU/memory exhaustion (with the help of stress-testing tools like stress-ng)

Such simulations were carried out in a controlled environment that supports failure injection and recovery testing, using orchestration platforms like Kubernetes or Docker Swarm.

Monitoring tools such as Prometheus, Grafana, and custom health check APIs were used to continuously track node availability, latency, CPU/memory usage, and response time. Logs and metrics were analyzed before, during, and after the fault injection phase to record deviations from normal performance.

While measuring performance impact, the evaluation focused on latency metrics before and after the event; recovery time, referencing Mean Time to Recovery (MTTR); throughput drop or service degradation; and error rates or failed requests during the failover stage. Failover maintained service continuity by rerouting work to healthy nodes, using microservice-based load balancing and health checks.

Cost Modeling

Initial operational cost models were produced by simulating resource usage under various traffic loads, comparing edge-based processing with the traditional centralized system.

This study is conceptual and mostly design-oriented, but the proposed edge-enabled microservices framework has been subjected to evaluation using detailed simulation models and performance profiling in scenarios typical of airline reservation systems. The findings endorse the architecture's viability and promise real-world merit.

Results

Latency and Responsiveness Improvements

Simulations show that during intensive activities such as booking and verifying seat inventory, unmatched latency reductions of up to 60% can be achieved under peak traffic conditions when time-sensitive operations are executed by edge nodes. This entire process appears to migrate user interactions away from round-trip communications with centralized servers, allowing them to be executed faster and more fluidly.

Throughput and Scalability

Also, coupled with overlooking-Kafka-based message brokering and other dynamic microservices using Kubernetes for load balancing, throughput is 20-25% greater when compared to monolithic or cloud-only architectures.

User Experience and Satisfaction Measures

End-user experience modeling based on response times, successful transaction rates, and user interaction flow demonstrated an uplift of 25% in user satisfaction levels. Apart from the overall short booking confirmations, it also reflects the smoothness experienced by the users due to edge proximity as well as fewer timeouts during the high traffic times.

Reliability and Fault Tolerance of Systems

Edge node distribution marked greater overall resilience for the systems. In failure simulation exercises, small incidents localized to specific locations produced virtually no consequences for the rest of the system, as fault isolation and recovery times improved by over 30% compared to centralized methods. Besides, during failover using microservices, booking processes continued seamlessly on other nodes.

Cost and Operational Efficiency

Some preliminary resource modeling on operations indicated that operational costs could be reduced by 15-20%, as edge nodes process less and with a smaller footprint, thus decreasing dependency on higher-capacity centralized infrastructures to account for the shortfalls. Data volume transfer costs therefore become much cheaper by incorporating temporary local storage and selective cloud syncing with users' applications. As shown in Table *1*, the proposed framework outperforms the baseline across all key performance metrics.

Metric	Baseline System	Proposed Framework	Improvement
Average Latency	120 ms	48 ms	↓ 60%
System Throughput	150 tps	180 tps	↑ 20%
User Satisfaction Score	60/100	85/100	↑ 25 points (≈42% increase)
Operational Cost (Est.)	High	Moderate	↓ 15–20%
Fault Recovery Time	~10 seconds	~7 seconds	↓ 30%
Scalability	Fixed resources	Dynamic (Kubernetes)	Elastic, demand-driven
Data Sync Efficiency	Centralized	Edge + Cloud Hybrid	Better real-time sync
Infrastructure Dependency	Heavy central cloud	Distributed edge/cloud	Reduced central load

TABLE 1: Performance comparison between baseline and proposed framework

Dataset description

Overview

A custom-built, simulation-based dataset was used in this work to mimic realistic airline reservation operations in a microservices-oriented edge environment. The dataset was generated in a controlled experimental setup, in which system behavior was simulated under varying user load conditions, node failures, and deployment scenarios. So, it was meant to evaluate the performance of the proposed edge-enabled microservice architecture.

Schema and Features

Each record in this dataset represents a simulation of a booking transaction going through an edge node or a central server. Some of the major features included are given in Table *2*.

Feature Name	Type	Description
Booking_ID	String	Unique booking identifier
Passenger_ID	String	Simulated user ID
Flight_Number	String	Simulated flight code
Departure_Airport	String	IATA airport code
Arrival_Airport	String	IATA airport code
Departure_Time	Timestamp	Scheduled departure
Arrival_Time	Timestamp	Scheduled arrival
Booking_Timestamp	Timestamp	Time of transaction
Edge_Node_ID	String	Node handling the request
Service_Response_Time (ms)	Integer	Time taken to process the request
Status	Categorical	Booking status: Confirmed, Cancelled, Pending
Traffic_Level	Categorical	Load condition: Low, Medium, High
Fault_Condition	Boolean	Whether a simulated failure occurred
Recovery_Time (s)	Float	Time to restore service, if failure occurred
Payment_Method	Categorical	Simulated method used (e.g., Card, Wallet)

TABLE 2: Schema of the simulated airline reservation dataset

IATA: International Air Transport Association

Generation Process

The dataset was produced using Python-based scripts that programmatically modeled the following:

• User behavior in different traffic scenarios: Booking rates during peak and off-peak hours.

• Edge node assignments and failovers: Randomized load balancing within Kubernetes.

• Latency and throughput variations: Influenced by edge node distance and message brokering (Kafka).

• Failure injections: Designed to validate the system's fault tolerance and recovery mechanisms.

• Cost model inputs: Simulated resource usage and data transfer volumes.

These features were calibrated against patterns typically observed in airline systems, where edge-enabled optimizations are applied to latency-sensitive components.

Data Sharing Note

Although this dataset is synthetic, its structure and dynamics are derived from internal architectural models provided by an industry partner. As this design is proprietary and subject to confidentiality agreements, the dataset cannot be publicly shared.

However, the schema and generation approach have been provided to enable interested researchers to recreate a similar experimental dataset. This supports transparency and facilitates independent replication of our results.

Discussion

The research established a successful joint venture integrating edge computing with microservice architecture to enhance the performance and responsiveness of airline booking systems. Time-critical tasks such as seat availability checks and real-time bookings were executed closer to the user, reducing

latency by nearly 60% and increasing throughput by approximately 20%. Also, the system demonstrated scalability through dynamic load balancing using Kubernetes and Kafka, enabling smooth handling of peak traffic with minimal infrastructure requirements. User satisfaction improved significantly, with a 25-point uplift attributed to faster response times, easier interactions, and cost-saving benefits from a hybrid edge-cloud model that reduces reliance on central servers.

Conclusions

Combining edge computing and microservices solves the drawbacks of the traditional centralized architecture. This particular augmentation of the framework further scales response time and latency flows to real-time processing and modular scalability. The following improvements are noteworthy: real-time data processing, where seat availability is locally checked and updated through edge nodes with synchronized replicas of the seat inventory database maintained at the central location. As soon as a reservation request hits an edge node, it checks the latest snapshot of availability and proceeds by applying locking/token-based mechanisms to ensure that double bookings do not happen till the process completes. These updates are reconciled with the central system through either versioned writes or event sourcing with conflict resolution to maintain consistency in scenarios of disconnected operations or sheer heavy load. Scalability is kept intact through dynamic resource allocation using orchestration platforms such as Kubernetes, Docker Swarm, K3s, etc., that watch system metrics and trigger the ongoing provisioning of additional services as thresholds grow beyond limits. These edge instances will register with the service registry and the load balancer to efficiently serve requests. In more advanced deployments, tools like Terraform or AWS Auto Scaling can be invoked to provision additional edge or cloud resources when local capacity is insufficient and to scale them down post-peak to ensure cost-effectiveness.

On user-centric architectural improvements, the system performance is highly optimized due to the decrease in latency by distributed processing. Edge servers based on proximity reduce the distance the data must travel, thus increasing the speed of request-response cycles. Time-sensitive operations, the checks for seat availability, fare estimation, and suggestions for itineraries are handled in the local system, attractions such as flight schedules and frequent flyer information are cached at the edge for almost immediate retrieval. GeoDNS and location-aware load balancers will route a user request towards its nearest edge node, and therefore take care of the data routing and minimize round-trip times. On the other hand, asynchronous back-end synchronization persists regarding data consistency while being transparent to the end user. One can say that the framework is positioned to address the main concerns of airline-related activities. These are mainly operations, cost, and adaptability to new technologies. Furthermore, it shows promise for application in other areas such as healthcare, transportation, and smart cities. Thus, answering some of the pressing issues in modern airline reservation systems with a scalable, flexible solution forms the basis for future modifications of the platform with possible AI, IoT, and blockchain integration.

Additional Information

Author Contributions

All authors have reviewed the final version to be published and agreed to be accountable for all aspects of the work.

Concept and design: Biman Barua, M. Shamim Kaiser

Acquisition, analysis, or interpretation of data: Biman Barua

Drafting of the manuscript: Biman Barua

Critical review of the manuscript for important intellectual content: Biman Barua, M. Shamim Kaiser

Supervision: M. Shamim Kaiser

Disclosures

Human subjects: All authors have confirmed that this study did not involve human participants or tissue. **Animal subjects:** All authors have confirmed that this study did not involve animal subjects or tissue. **Conflicts of interest:** In compliance with the ICMJE uniform disclosure form, all authors declare the following: **Payment/services info:** All authors have declared that no financial support was received from any organization for the submitted work. **Financial relationships:** All authors have declared that they have no financial relationships at present or within the previous three years with any organizations that might have an interest in the submitted work. **Other relationships:** I have submitted this article to a preprint server [https://arxiv.org/abs/2411.12650] and will not submit it to any other journal for

publication consideration.

Acknowledgements

The article has been uploaded in preprints (https://arxiv.org/abs/2411.12650) and not submitted anywhere for consideration of publication.

References

1. Niemeier HM: Is economics good for aviation policy? Some problems in bridging the gap between theory and policy. Journal of Air Transport Management. 2021, 96:102107. 10.1016/j.jairtraman.2021.102107
2. Barua B, Whaiduzzaman M, Sarker MM, Kaiser MS, Barros A: Designing and implementing a distributed database for microservices cloud-based online travel portal. Sentiment Analysis and Deep Learning. Shakya S, Du KL, Ntalianis K (ed): Springer, Singapore; 2023. 1432:295-314. 10.1007/978-981-19-5443-6_22
3. Islam AKMN, Mäntymäki M, Laato S, Turel O: Adverse consequences of emotional support seeking through social network sites in coping with stress from a global pandemic. International Journal of Information Management. 2021, 62:102431. 10.1016/j.ijinfomgt.2021.102431
4. Barua B, Mozumder MJU, Kaiser MS, Barua I: Building scalable airline reservation systems: A microservices approach using AI and deep learning for enhanced user experience. 2025 4th International Conference on Sentiment Analysis and Deep Learning (ICSADL), Bhimdatta, Nepal. 2025, 941-950. 10.1109/ICSADL65848.2025.10933362
5. Seneler CO, Basoglu N, Daim T : Interface feature prioritization for web services: Case of online flight reservations. Computers in Human Behavior. 2009, 25:862-877. 10.1016/j.chb.2008.12.028
6. Barua B, Barua I, Kaiser MS, Mozumder MJU: Trends and challenges in AI-driven microservices for cloud-based airline reservation systems: A review. Proceedings of the 2025 3rd International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT). 2025, 1902-1911. 10.1109/IDCIOT64235.2025.10915076
7. Barua B, Whaiduzzaman M: A methodological framework on development the garment payroll system (GPS) as SaaS. 2019 1st International Conference on Advances in Information Technology (ICAIT), Chikmagalur, India. 2019, 431-435. 10.1109/ICAIT47043.2019.8987325
8. Katal A, Prasanna P, Birla R, Kunal: Evolution from monolithic to microservices architecture: A new era in software architecture. Advancements in Optimization and Nature-Inspired Computing for Solutions in Contemporary Engineering Challenges. Rossit D, Torres-Aguilar CE, Toncovich AA (ed): Springer, Singapore; 2025. 235-279. 10.1007/978-981-96-0706-8_12
9. Kubra KT, Barua B, Sarker MM, Kaiser MS: An IoT-based framework for mitigating car accidents and enhancing road safety by controlling vehicle speed. 2023 7th International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), Kirtipur, Nepal. 2023, 46-52. 10.1109/I-SMAC58438.2023.10290219
10. AltexSoft: Airline Reservation Systems and Passenger Service Systems: Navitaire, Amadeus Altéa, SabreSonic and more. (2019). https://www.altexsoft.com/blog/airline-reservation-systems-passenger-service-systems/.
11. Howlader SMN, Hossain MM, Khanom S, Sarker S, Barua B: An intelligent car locating system based on Arduino for a massive parking place. Multi-Strategy Learning Environment. Vimal V, Perikos I, Mukherjee A, Piuri V (ed): Springer, Singapore; 2024. 19-33. 10.1007/978-981-97-1488-9_2
12. Amazon Web Services: AWS Serverless Airline Booking. (2020). https://github.com/aws-samples/aws-serverless-airline-booking.
13. Barua B: M-commerce in Bangladesh - status, potential and constraints. International Journal of Information Engineering and Electronic Business. 2016, 8:22-27. 10.5815/ijieeb.2016.06.03
14. Lee DD-h, Wang W, Leong P: A conceptual model for passenger service systems adoption: An interdisciplinary perspective. 2017 IEEE 21st International Conference on Computer Supported Cooperative Work in Design (CSCWD), Wellington, New Zealand. 2017, 492-497. 10.1109/CSCWD.2017.8066743
15. Amadeus: Amadeus Altéa Reservation: Airline Reservation System. (2020). https://amadeus.com/en/airlines/products/altea-airline-reservation-system.
16. Barua B, Kaiser MS: A methodological framework of containerized microservices orchestration and provisioning in the cloud: a case study of online travel platforms. Challenges and Opportunities for Innovation in India. Mishra S, Singh AK, Prajapati P (ed): CRC Press, London; 2025. 183-189. 10.1201/9781003606260-33
17. Mouri IJ, Barua B, Mesbahuddin Sarker M, Barros A, Whaiduzzaman M: Predicting online job recruitment fraudulent using machine learning. Proceedings of Fourth International Conference on Communication, Computing and Electronics Systems. Bindhu V, Tavares JMR, Vuppalapati C (ed): Springer, Singapore; 2023. 977:719-733. 10.1007/978-981-19-7753-4_55
18. Barua B, Kaiser MS: A methodical framework for integrating serverless cloud computing into microservice architectures [PREPRINT]. Preprints. 2024, 10.20944/preprints202410.0494.v1
19. Barua B, Obaidullah MD: Development of the student management system (SMS) for universities in Bangladesh. BUFT Journal. 2014, 2:57-66.
20. Howlader SMN, Barua B, Sarker MMS, Kaiser MS, Whaiduzzaman M: Automatic yard monitoring and humidity controlling system based on IoT. 2023 International Conference on Advanced Computing & Communication Technologies (ICACCTech). 2023, 397-403. 10.1109/ICACCTech61146.2023.00072
21. Zheng B, Pan L, Liu S: Market-oriented online bi-objective service scheduling for pleasingly parallel jobs with variable resources in cloud environments. Journal of Systems and Software. 2021, 176:110934. 10.1016/j.jss.2021.110934
22. Newman S: Building microservices: Designing fine-grained systems. (2021). https://book.northwind.ir/bookfiles/building-microservices/Building.Microservices.pdf.
23. Söylemez M, Tekinerdogan B, Kolukısa Tarhan A: Challenges and solution directions of microservice architectures: A systematic literature review. Applied Sciences. 2022, 12:5507. 10.3390/app12115507
24. Xiang Q, Peng X, He C, et al.: No free lunch: Microservice practices reconsidered in industry. 2021, 10.48550/arXiv.2106.07321
25. Barua B, Kaiser MS: Leveraging machine learning for real-time personalization and recommendation in airline

industry [PREPRINT]. Preprints. 2024, 10.20944/preprints202410.2436.v1
26. Chen Z, Shen L, Li F: Your neighbors are misunderstood: On modeling accurate similarity driven by data range to collaborative web service QoS prediction. Future Generation Computer Systems. 2019, 95:404-419. 10.1016/j.future.2019.01.003
27. Barua B, Kaiser MS: Cloud-enabled microservices architecture for next-generation online airlines reservation systems. 2024, 10.21203/rs.3.rs-5182678/v1
28. Khan LU, Yaqoob I, Tran NH, Kazmi SMA, Dang TN, Hong CS: Edge-computing-enabled smart cities: A comprehensive survey. IEEE Internet of Things Journal. 2020, 7:10200-10232. 10.1109/JIOT.2020.2987070
29. Satyanarayanan M: The emergence of edge computing. Computer. 2017, 50:30-39. 10.1109/mc.2017.9
30. Shi W, Dustdar S: The promise of edge computing. Computer. 2016, 49:78-81. 10.1109/mc.2016.145
31. Chen S, Wen H, Wu J, et al.: Internet of things based smart grids supported by intelligent edge computing. IEEE Access. 2019, 7:74089-74102. 10.1109/ACCESS.2019.2920488
32. Barua B, Kaiser MS: Novel architecture for distributed travel data integration and service provision using microservices. 2024, 10.48550/arXiv.2410.24174
33. Qiu T, Chi J, Zhou X, Ning Z, Atiquzzaman M, Wu DO: Edge computing in industrial internet of things: Architecture, advances and challenges. IEEE Communications Surveys & Tutorials. 2020, 22:2462-2488. 10.1109/COMST.2020.3009103
34. Chaki PK, Barua B, Sazal MMH, et al.: PMM: A model for Bangla parts-of-speech tagging using sentence map. Information, Communication and Computing Technology. Badica C, Liatsis P, Kharb L, Chahal D (ed): Singapore, Springer; 2020. 1170:181-194. 10.1007/978-981-15-9671-1_15
35. ITU: IMT Vision-Framework and overall objectives of the future development of IMT for 2020 and beyond. 2015, 1-21.
36. Prakash T, Kakkar M, Patel K: Geo-identification of web users through logs using ELK stack. 2016 6th International Conference - Cloud System and Big Data Engineering (Confluence), Noida, India. 2016, 606-610. 10.1109/CONFLUENCE.2016.7508191
37. Mengistu DM: Distributed Microservice Tracing Systems: Open-source tracing implementation for distributed Microservices build in Spring framework. Master's Thesis. Jyväskylä University of Applied Sciences University, Jyväskylä, Finland; 2020.
38. Barua B, Kaiser MS: Design and Evaluation of a Microservices Cloud Framework for Online Travel Platforms. 2025, 10.48550/arXiv.2505.14508
39. Lehikoinen T: AWS Greengrass in Building IoT Applications. Bachelor's Thesis. Savonia University of Applied Sciences, Finland; 2023.
40. Gulli A: Google Anthos in Action: Manage Hybrid and Multi-Cloud Kubernetes Clusters. Manning Publications, Shelter Island, NY; 2023.
41. Chaki PK, Sazal MMH, Barua B, Hossain MS, Mohammad KS: An approach of teachers' quality improvement by analyzing teaching evaluations data. 2019 Second International Conference on Advanced Computational and Communication Paradigms (ICACCP), Gangtok, India. 2019, 533-537. 10.1109/ICACCP.2019.00099
42. Barua B, Kaiser MS: Blockchain-based trust and transparency in airline reservation systems using microservices architecture. 2024, 10.48550/arXiv.2410.14518
43. Barua B, Kaiser MS: Leveraging microservices architecture for dynamic pricing in the travel industry: algorithms, scalability, and impact on revenue and customer satisfaction. 2024, 10.48550/arXiv.2411.01636
44. Simić M, Stojkov M, Sladić G, Milosavljević B: CRDTs as replication strategy in large-scale edge distributed system: An overview. Proceedings of the ICIST 2020: Proceedings of the 10th International Conference on Information Systems and Technologies. 2020, 46-50.
45. Barua B, Kaiser MS: A next-generation approach to airline reservations: integrating cloud microservices with AI and blockchain for enhanced operational performance. 2024, 10.48550/arXiv.2411.06538
46. Barua B, Kaiser MS: Optimizing airline reservation systems with edge-enabled microservices: A framework for real-time data processing and enhanced user responsiveness. 2024, 10.48550/arXiv.2411.12650
47. He X, Tu Z, Wagner M, Xu X, Wang Z: Online deployment algorithms for microservice systems with complex dependencies. IEEE Transactions on Cloud Computing. 2023, 11:1746-1763. 10.1109/tcc.2022.3161684
48. Barua B, Kaiser MS: AI-driven resource allocation framework for microservices in hybrid cloud platforms. 2024, 10.48550/arXiv.2412.02610
49. Shehab AH, Faraj Al-Janabi ST: Microsoft Azure IoT-based edge computing for smart homes. 2020 International Conference on Decision Aid Sciences and Application (DASA), Sakheer, Bahrain. 2020, 315-319. 10.1109/DASA51403.2020.9317274
50. Barua B, Kaiser MS: Real-time performance optimization of travel reservation systems using AI and microservices. 2024, 10.48550/arXiv.2412.06874
51. Stetsenko I, Myroniuk A: Software for collecting and analyzing metrics in highly loaded applications based on the Prometheus monitoring system. Information Computing and Intelligent systems. 2024, 5:17-28. 10.20535/2786-8729.5.2024.316366
52. Barua B, Kaiser MS: Enhancing resilience and scalability in travel booking systems: a microservices approach to fault tolerance, load balancing, and service discovery. 2024, 10.48550/arXiv.2410.19701
53. Raman A: How do social media, mobility, analytics and cloud computing impact nonprofit organizations? A pluralistic study of information and communication technologies in Indian context. Information Technology for Development. 2016, 22:400-421. 10.1080/02681102.2014.992002
54. Barua B, Kaiser MS: Microservices-based framework for predictive analytics and real-time performance enhancement in travel reservation systems. 2024, 10.48550/arXiv.2412.15616
55. Barua B, Kaiser MS: Optimizing travel itineraries with AI algorithms in a microservices architecture: balancing cost, time, preferences, and sustainability. 2024, 10.48550/arXiv.2410.17943
56. Chakraborty M, Kundan AP: Grafana. Monitoring Cloud-Native Applications. Apress, Berkeley, CA; 2021. 187-240. 10.1007/978-1-4842-6888-9_6