Real-Time Analytics Dashboard for a Payment App

Using Apache Cassandra and Real-Time Data Pipelines

Academic Year: 2023-2027

Institute:

RAMAIAH INSTITUTE OF TECHNOLOGY, BENGALURU

Guide:

Akshata Kamath

Team Members:

Ojasvi Poonia
1MS23CS125

Prithvi Kiran
1MS23CS140

Praneel Sindhole
1MS23CS133

Abstract

Modern payment applications generate a massive velocity and volume of data, with millions of transactions occurring concurrently. Traditional relational database systems are not equipped to handle this high write throughput and fail to provide the real-time analytical insights (e.g., fraud detection, spending trends) that are critical for business intelligence and user engagement. This project proposes a **Real-Time Analytics Dashboard for a Payment App**, built on a distributed NoSQL architecture.

The core of this system is **Apache Cassandra**, chosen for its masterless architecture, linear scalability, and exceptional write performance, making it ideal for ingesting a relentless stream of transaction data. The system will ingest, process, and store payment data with low latency, enabling a dashboard to query and visualize analytics in near real-time. This project demonstrates a robust, scalable, and fault-tolerant data architecture designed for the high-stakes, high-throughput world of financial technology.

Keywords: Apache Cassandra, NoSQL, Real-Time Analytics, Big Data, Data Modeling, Payment Systems, FinTech, System Architecture, React, Node.js

1. Introduction

In the rapidly expanding digital economy, payment applications are ubiquitous. The competitive advantage for these apps no longer lies just in the ability to transfer money, but in the value-added services they provide. This includes instant notifications, fraud alerts, and insightful analytics on spending habits. However, providing these features in real-time is a significant engineering challenge.

Traditional systems, often built on relational databases, typically run analytics on data warehouses. This process is slow, with data latency measured in hours or even days. For a payment app, this is unacceptable. A fraudulent transaction must be flagged in seconds, not hours. A user must be able to see their updated spending categories instantly. This project tackles this challenge by designing a system from the ground up for "write-intensive" operations and real-time queries.

2. Problem Statement

Conventional payment processing systems, often built on monolithic, SQL-only architectures, face several critical performance and scalability issues:

1. **High Write Latency & Lock Contention:** The sheer volume of concurrent transactions can overwhelm a relational database, leading to locking and high latency on new transactions.
2. **Poor Scalability:** A centralized SQL database is difficult and expensive to scale horizontally to meet a growing global user base and data volume.

3. **Lack of Real-Time Analytics:** Analytics are traditionally run via batch jobs on a separate database (OLAP), making it impossible to provide "live" insights on a user's dashboard.
4. **Delayed Fraud Detection:** The inability to query and analyze data in real-time means that fraudulent activity is often detected long after it has occurred, leading to financial loss.
5. **System Downtime:** In a traditional master-slave architecture, if the master database fails, the entire system can experience downtime.

3. Objectives

The primary objective of this project is to design, implement, and analyze a scalable backend and dashboard for a payment app that overcomes these limitations. The specific aims are:

1. To design and implement a robust, query-first database schema in **Apache Cassandra** optimized for time-series data (transactions) and analytical queries (spending by category, merchant, etc.).
2. To build a backend system (e.g., in Node.js or Python) capable of ingesting a high throughput of payment transactions and writing them to the Cassandra cluster with low latency.
3. To develop a responsive **React-based dashboard** that visualizes key analytics for a user in near real-time, such as live transaction feeds, spending by category, and location-based data.
4. To implement a basic real-time fraud detection rule (e.g., "flag transactions from two different countries for the same user within 5 minutes") by leveraging Cassandra's fast read capabilities.
5. To empirically analyze and demonstrate the system's performance, specifically its write throughput (transactions per second) and read query latency (dashboard load time).

4. Literature Survey

The management of high-velocity data has seen a significant evolution from traditional models.

- **Traditional Relational Models:** Systems like MySQL or PostgreSQL are built on ACID principles, ensuring transactional integrity. While excellent for banking ledgers (the "system of record"), they are not designed for the write-intensive, high-volume, and low-latency read requirements of a real-time analytics feed. Their rigid schemas and difficulty in horizontal scaling make them unsuitable for this project's primary goal.
- **The Rise of NoSQL:** NoSQL databases emerged to address the scalability and performance limitations of RDBMS. These are broadly categorized, but for this use case, the most relevant are column-family stores.
- **Wide-Column Stores (Apache Cassandra):** Cassandra is a distributed, masterless NoSQL database. Its key strengths are its **linear scalability** (you add more nodes to get more power), **high availability** (data is replicated, so there is no single point of failure), and **exceptional write performance**. It is designed to ingest massive amounts of data very quickly, making it a standard in FinTech, IoT, and messaging applications where

write performance is paramount. This project leverages this strength for its core data ingestion.

5. Methodology

The project will be executed by integrating database design, a streaming pipeline, a backend API, and a frontend dashboard.

1. **Cassandra Data Modeling:** This is the critical first step. Unlike SQL, Cassandra data modeling is "query-first." We will identify the queries our dashboard needs to make (e.g., "get all transactions for user X," "get total spending by category for user X this month") and design tables (Column Families) specifically for each query to ensure blazingly fast reads.
2. **Data Ingestion:** A backend API (e.g., Node.js) will provide an endpoint to receive new transactions. For a more advanced setup, a message queue like **Apache Kafka** would be used to create a streaming data pipeline, decoupling the app from the database and handling back-pressure, before writing to Cassandra.
3. **Backend API (Data Access):** The same backend will expose a REST API for the frontend. These API endpoints will run the pre-determined queries against Cassandra to fetch data for the dashboard.
4. **Frontend Dashboard:** A React application will be developed. It will use libraries like **Chart.js** or **D3.js** to visualize the JSON data received from the backend API, presenting it as interactive graphs and live-updating tables.

6. System Architecture

The system will be composed of several decoupled components:

1. **React Frontend:** The client-side dashboard that handles user interaction and data visualization.
2. **Backend API (Node.js/Python):** The server that acts as the "brain." It handles API requests, business logic, and communication with the data layer.
3. **(Optional) Streaming Layer (Apache Kafka):** A message broker that sits between the app and the database, queuing incoming transactions to ensure data is never lost and is processed in order.
4. **Data Layer (Apache Cassandra):** The persistent, distributed NoSQL database that stores all transaction and analytics data.

7. Database Design (Cassandra)

A detailed design of the Cassandra keyspace is critical. All tables are designed based on their query.

- **Table: `transactions_by_user`**
    - **Query:** Get all transactions for a user, in reverse chronological order.
    - **Schema:** `CREATE TABLE transactions_by_user ( user_id uuid, transaction_time timestamp, transaction_id uuid, amount decimal, category text, merchant text,`

    - PRIMARY KEY (user_id, transaction_time)) WITH CLUSTERING ORDER BY (transaction_time DESC);
- **Table: spending_by_user_category_month**
    - **Query:** Get the total spending for a user by category for a given month (for a pie chart).
    - **Schema:** CREATE TABLE spending_by_user_category_month (user_id uuid, year_month text, // e.g., "2025-11" category text, total_spent counter, // A Cassandra counter for atomic updates PRIMARY KEY ((user_id, year_month), category));
    - **Note:** When a transaction comes in, we update both this table (using the counter) and the transactions_by_user table. This is a common Cassandra pattern (denormalization for read performance).

8. Expected Outcome

The successful implementation will yield a highly responsive and scalable analytics dashboard. We expect a dramatic, measurable improvement in write throughput compared to a relational model, capable of handling thousands of transactions per second. The read latency for dashboard queries is expected to be in the sub-second range, providing a "live" feel to the user. This project will serve as a clear, practical demonstration of Cassandra's power in building real-time analytics systems for write-intensive applications.

9. Conclusion

This project proposes the design of a modern, real-time analytics system that addresses the key limitations of traditional database architectures in the payment industry. By leveraging Apache Cassandra, the system is designed for high throughput, horizontal scalability, and high availability. It isolates the high-velocity write-intensive workload, enabling the "system of engagement" (the dashboard) to be fast and responsive.

Future Scope: The completed platform is a strong foundation for more advanced features. This could include integrating **Apache Spark** for more complex, batch-based analytics (e.g., "compare your spending to users like you") or feeding the real-time data into a Machine Learning model for more sophisticated, adaptive fraud detection.