Distributed Multi-Model Analytics for E-commerce Data

Abstract

The project presents the design and implementation of a distributed multi-model analytics system for large-scale e-commerce data. By strategically combining MongoDB, HBase, and Apache Spark, the system demonstrates how different NoSQL data models and distributed processing frameworks can be leveraged to address diverse analytical requirements. Using a synthetic e-commerce dataset spanning users, products, sessions, and transactions, the project delivers scalable data models, analytical pipelines, and actionable business insights.

Contents

1 Introduction

Modern e-commerce platforms generate massive volumes of heterogeneous data, ranging from structured transactional records to semi-structured user interaction logs. No single data storage or processing technology is optimal for all analytical needs. This project explores a multi-model approach that combines MongoDB (document-oriented), HBase (wide-column), and Apache Spark (distributed analytics) to efficiently store, process, and analyze large-scale e-commerce data.
The primary objective is to demonstrate informed technology selection, schema design, and analytical integration across systems, while extracting meaningful business insights from the data.

2 System Architecture Overview

The overall system architecture follows a polyglot persistence approach:

- **MongoDB** stores rich, nested documents such as user profiles, product catalogs, and transactions.
- **HBase** stores high-volume, time-series session and product interaction data optimized for fast range scans.
- **Apache Spark** performs distributed batch analytics, SQL-based analysis, and cross-data-source integration.

Spark acts as the central analytics engine, ingesting data from JSON files and, notionally, from MongoDB and HBase for integrated queries.

3 Dataset Description

The dataset was generated using a Python-based synthetic data generator and simulates 90 days of e-commerce activity. The main entities include users, categories, products, sessions, and transactions.

3.1 Key Entities

- **Users**: demographic and registration data.
- **Categories**: hierarchical product classification.
- **Products**: pricing, inventory, and historical price changes.
- **Sessions**: detailed user browsing behavior.
- **Transactions**: completed purchases with line items and payment details.

4 Part 1: Data Modeling and Storage

4.1 MongoDB Data Model

MongoDB was selected for entities that benefit from flexible schemas and documents store for semi-structured and highly related data such as user profiles, product catalogs, and transaction records, enabling flexible schemas and powerful aggregation queries

4.1.1 Product Collection

Each product document stores core product attributes and embeds price history:

```
{
"product_id": "prod_00123",
"name": "Innovative Executive Paradigm",
"  category_id  ":        "  cat_007  ",
" subcategory_id ":    " sub_ 007 _ 01 ",
"base_price": 129.99,
"current_stock": 47,
" price_history ": [
{"price": 149.99, "date": "2024-12-20"},
{"price": 129.99, "date": "2025-02-15"}
]
}
```

This design supports efficient queries on pricing trends and inventory status.

4.1.2 User Collection

User profiles are stored as documents, optionally enriched with summary metrics such as total spend or purchase count for fast access.

4.1.3 Transaction Collection

Transactions embed purchased items as arrays, allowing complete order analysis without joins.

4.2 MongoDB Aggregation Examples

Total Revenue by Category:

```
db.transactions.aggregate([
{ $unwind: "$items" },
{ $lookup: {
from: "products",
localField: "items.product_id",
foreignField: "product_id",
as: "product"
}},
{ $unwind: "$product" },
{ $group: {
_id: "$product.category_id",
totalRevenue: { $sum: "$items.subtotal" }
}}
])
```

4.3 HBase Data Model

HBase is used as a wide-column, distributed storage layer optimized for high-volume, time-series data, particularly user session activity and product interaction logs, allowing efficient range scans and fast retrieval of recent events

4.3.1 Session Table

- **Row Key**: user$_i$*dreverse*$_t$*imestamp*

- **Column Families**:

    – meta: session metadata (device, location)
    – activity: page views, viewed products

This design enables efficient retrieval of recent sessions per user.

4.4 MongoDB vs HBase Justification

MongoDB was chosen for transactional and hierarchical data requiring flexible schemas and aggregations, while HBase was selected for sparse, high-volume time-series data optimized for range scans.

5 Part 2: Data Processing with Apache Spark

5.1 Batch Processing

Spark batch jobs were implemented using PySpark DataFrames.

5.1.1 Data Cleaning

- Standardized timestamps to ISO format.
- Handled missing session references in transactions.
- Normalized nested arrays using explode.

5.1.2 Product Affinity Analysis

A simple *"users who bought X also bought Y"* analysis was implemented by self-joining transaction items grouped by transaction ID.

5.2 Spark SQL Analytics

Spark SQL was used to compute revenue trends and user cohorts.

```
SELECT month(timestamp) AS month,
SUM(total) AS monthly_revenue
FROM transactions
GROUP BY month(timestamp)
ORDER BY month;
```

6 Part 3: Analytics Integration

6.1 Customer Lifetime Value (CLV)

Business Question: Which customers generate the highest long-term value?
Data Sources:

- MongoDB: user profiles and transactions
- HBase: session frequency and engagement metrics

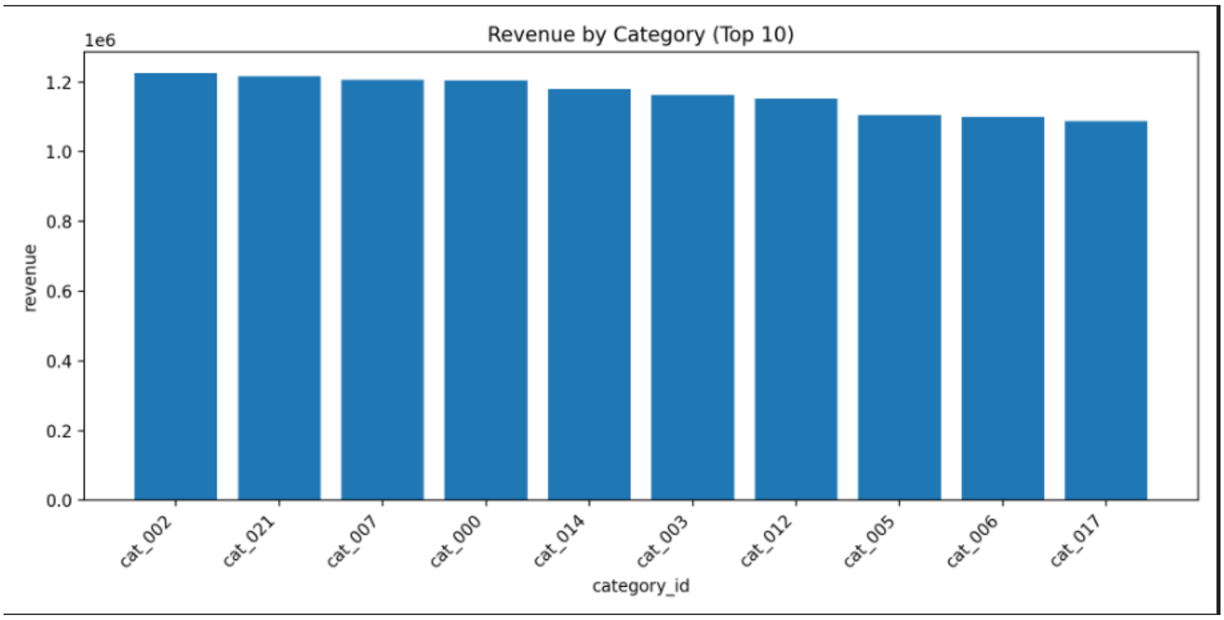
Spark integrates these datasets to compute CLV as: [CLV = $\sum_{i=1}^{n} TransactionValue_i \times EngagementFactor$]

7 Part 4: Visualization and Insights

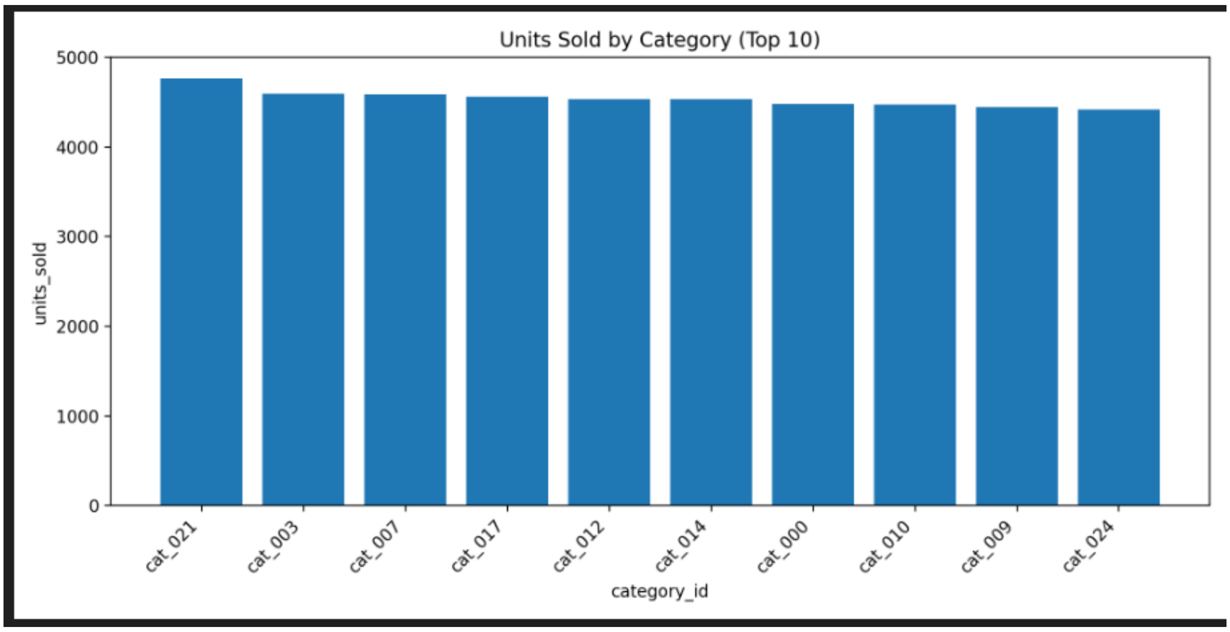
7.1 Visualizations

The following visualizations were produced using Python:

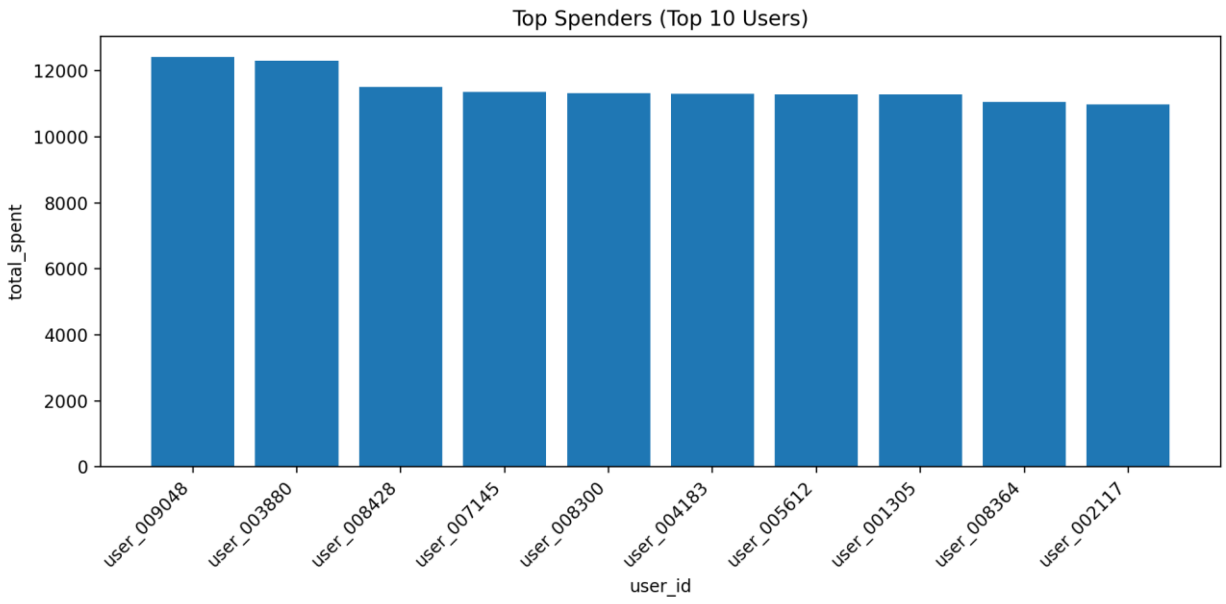
- Revenue by category (Top 10)

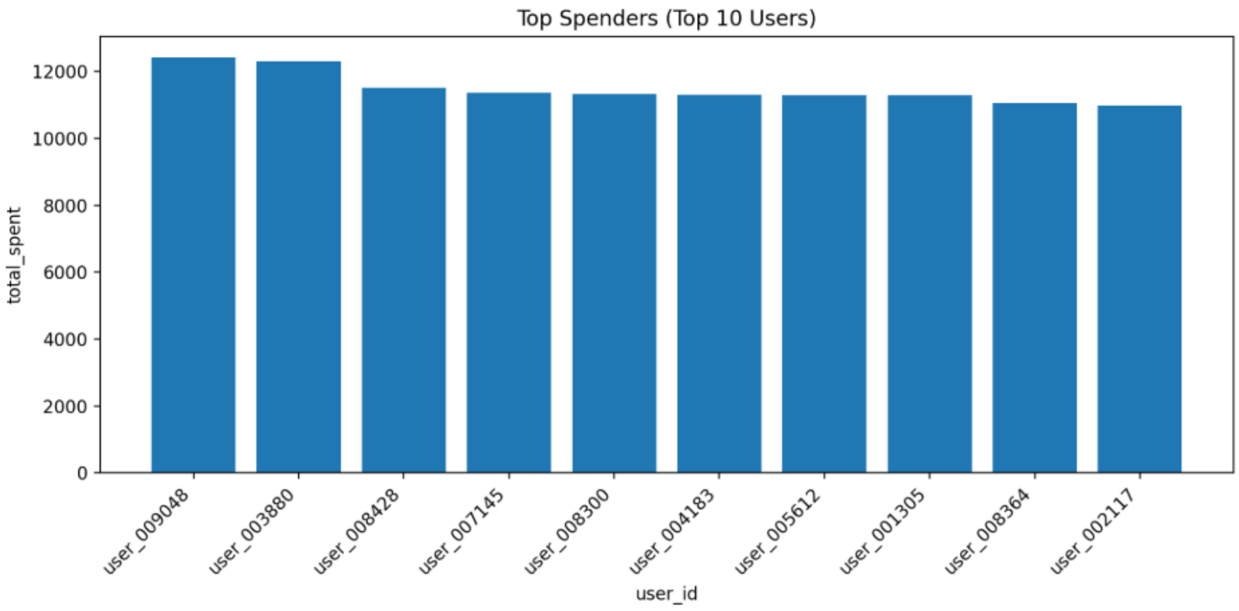


- Units sold by Category (Top 10)

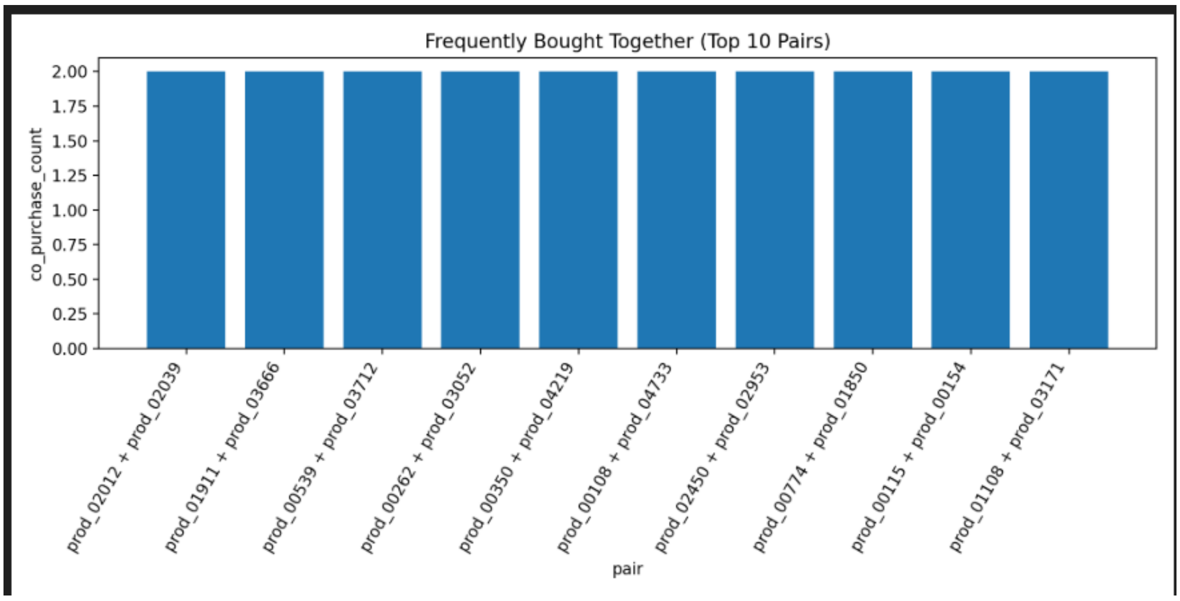


- Top-spenders





- Frequently bought together



7.2 Key Business Insights

- A small number of categories generate a disproportionate share of revenue.
- High product views do not always translate to high sales, indicating pricing or UX issues.
- Users with frequent sessions have significantly higher CLV.

8 Scalability Considerations

The architecture supports horizontal scaling through:

- Sharded MongoDB collections
- Distributed HBase regions
- Spark cluster execution for analytics

9 Conclusion

This project demonstrates how a multi-model, distributed analytics architecture can effectively address diverse e-commerce analytical needs. By aligning data models and processing engines with specific query patterns, the solution achieves scalability, flexibility, and meaningful business insights.