Workshop 3: Concurrency, Parallelism, and Distributed Databases for AI-Powered Predictive Trading Analytics Platform

David Alexander Colorado Rodríguez (20211020031)
Andres Felipe Martín Rodríguez (20201020137)

June 19, 2025

Course: Data Bases II
Professor: Carlos Andres Sierra Virgüez
Institution: Francisco José de Caldas District University, Bogotá, Colombia

Contents

Abstract

This document presents a comprehensive analysis and design for enhancing the database architecture of an AI-powered predictive trading analytics platform, focusing on concurrency control, parallel processing, and distributed database systems. Building on the hybrid architecture established in Workshop 2—comprising PostgreSQL for transactional data, MongoDB for real-time market data, and Snowflake for analytical workloads—this report addresses the challenges of concurrent data access, scalability, and performance optimization in a high-frequency trading environment. Through detailed concurrency analysis, a distributed database design, performance improvement strategies, and refinements to prior deliverables, we propose a robust, scalable, and efficient system tailored to the platform's big data, multi-location, and high-availability requirements. This work aligns with both technical and business objectives, ensuring data integrity, regulatory compliance, and rapid decision-making capabilities.

1 Introduction

The rapid evolution of financial markets has necessitated advanced technological solutions capable of processing vast datasets, executing real-time analytics, and supporting global operations. The AI-powered predictive trading analytics platform, developed as part of the Data Bases II course at Francisco José de Caldas District University, exemplifies such a solution. Designed to integrate high-frequency trading, predictive modeling, and business intelligence, the platform relies on a hybrid database architecture comprising PostgreSQL, MongoDB, and Snowflake. However, the demands of concurrent access, large-scale data processing, and geographic distribution require a deeper exploration of concurrency mechanisms, parallel processing, and distributed systems.

This report, prepared for Workshop 3, aims to enhance the platform's database system by addressing four key areas: (1) concurrency analysis to ensure data consistency under simultaneous access, (2) parallel and distributed database design to support scalability and multi-location operations, (3) strategies to improve performance through parallelism and distribution, and (4) improvements to the deliverables from Workshop 2 based on assumed feedback. Each section is developed with academic rigor, leveraging insights from prior deliverables and extending them to meet the platform's technical and business needs.

2 Database System Entries

2.1 Datos de Mercado en Tiempo Real

The system continuously receives OHLC (Open, High, Low, Close) price data for multiple financial instruments, including stocks, currencies, cryptocurrencies, and commodities. This information comes from sources such as Yahoo Finance and Alpha Vantage and is processed at frequencies ranging from seconds to minutes depending on the instrument type. The estimated volume reaches approximately 1GB per day for each monitored exchange.

Los datos incluyen información de volúmenes de transacciones, spreads bid-ask y métricas de liquidez. La captura se realiza mediante APIs REST y WebSocket, garantizando la menor latencia posible para decisiones de trading críticas. El sistema implementa validación automática de integridad para detectar anomalías en los datos recibidos.

2.2 Información de Usuarios y Configuraciones

The platform manages differentiated user profiles with their respective customized strategy settings. Each user maintains specific trading preferences, risk tolerance parameters, and alert

settings. The system stores complete authentication and activity history to comply with regulatory requirements.

Subscriptions are managed with different service levels, from free to enterprise plans, each with specific usage limits and access to advanced features. Billing information and payment methods are directly integrated with user management systems.

2.3 Operation Execution Data

The system captures all trading orders generated by the AI algorithms, including execution confirmations from brokers such as XM and MetaTrader 5. Detailed metrics on latency, slippage, and transaction costs are recorded for further performance analysis.

The results of each trade are stored with complete information on profit/loss, applied commissions, and market conditions at the time of execution. This information is essential for calculating performance metrics and continuously optimizing strategies.

2.4 Technical Analysis Information

Technical indicators are calculated in real time using market data as input. The system generates trading signals based on machine learning algorithms, storing confidence levels and parameters used in each calculation. Backtesting results are retained for historical validation of strategies.

Performance metrics include Sharpe ratios, maximum drawdown, win rate, and profit factor. This information is used both for user reporting and for automatic optimization of strategy parameters.

3 Database System Processes

3.1 Data Ingestion and Processing

The ingestion process implements a robust ETL pipeline that validates, normalizes, and stores incoming market data. Apache Kafka handles real-time data streaming, ensuring that critical information is not lost during spikes in market volatility. Integrity validation is performed automatically, rejecting inconsistent or corrupted data.

Temporal partitioning is automatically applied, organizing data by financial instrument and timeframe. This strategy optimizes historical queries and facilitates database maintenance. Data is compressed using time-series-specific algorithms, significantly reducing storage requirements.

3.2 Algorithmic Calculation and Analysis

Artificial intelligence algorithms continuously process historical data to generate trading signals. The system runs machine learning models trained on historical patterns, automatically adjusting to changing market conditions. Alerts are generated based on user-configurable confidence thresholds.

Strategy performance is continuously evaluated, comparing actual results with predictions. This feedback loop allows for automatic parameter adjustment and early identification of model accuracy degradation.

3.3 User Management and Security

The system implements mandatory multi-factor authentication for all users, with secure session management and full activity tracking. Subscription limits are automatically applied, ensuring

fair use of system resources. Strategy customization is managed through intuitive interfaces that allow for granular adjustments without compromising system stability.

Activity monitoring includes detection of anomalous usage patterns, triggering automatic alerts for potential security breaches. Audit logs are maintained immutably to meet regulatory requirements.

3.4 Optimization and Maintenance

Query optimization runs automatically, analyzing usage patterns to identify opportunities for improvement. Intelligent partitioning adjusts data distribution based on observed access patterns. Geo-replication ensures low latency for global users while maintaining data consistency.

Automatic cleansing processes manage the data lifecycle, archiving historical information according to configurable retention policies. This approach balances compliance requirements with storage cost optimization.

4 Database System Controls

4.1 Security and Data Protection

The implementation of AES-256 encryption protects all sensitive data both at rest and in transit. The system requires multi-factor authentication for access to critical functionalities, complemented by an auditing system that logs all operations performed. Data segregation is implemented based on access levels, ensuring that users can only access authorized information.

The database firewall implements strict rules that block unauthorized connections, while continuous monitoring detects anomalous access attempts. SSL/TLS certificates are automatically renewed to maintain secure communications.

4.2 Integrity and Consistency

ACID transactions ensure the consistency of critical operations, especially in trade processing and portfolio updates. Checksums automatically verify the integrity of stored data, detecting any corruption early. Schema versioning allows for controlled updates without interrupting service.

Automatic rollback mechanisms are triggered when transaction errors are detected, maintaining system integrity. Constraint validation is performed in real time, preventing the insertion of inconsistent data.

4.3 Performance and Availability

Continuous latency monitoring ensures that critical alerts are delivered in less than 100 milliseconds. CPU and memory thresholds trigger automatic alerts when predefined limits are exceeded, enabling proactive intervention. The load balancer efficiently distributes queries across multiple database instances.

The intelligent caching system keeps the most frequently accessed data in memory, significantly reducing response times. Throttling mechanisms prevent system overload during exceptional peak demand.

4.4 Regulatory Compliance

GDPR compliance implementation protects European user data through automatic anonymization and consent management. Immutable audit logs provide full traceability for regulatory audits. Compliance reports are automatically generated, simplifying compliance with multiple jurisdictions.

Data retention policies are configured according to region-specific requirements, ensuring information is retained as long as necessary without privacy breaches. Data anonymization is automatically applied for statistical analyses that do not require personal identification.

5 Database System Outputs

5.1 Real-Time Alerts and Communications

The system generates instant trading alerts via WebSocket, ensuring users receive critical information without delay. Mobile push notifications complement web alerts, ensuring traders can respond quickly to market opportunities. Performance reports are automatically distributed based on each user's preferences.

Risk alerts are automatically triggered when predefined limits, including maximum drawdown and exposure per position, are exceeded. Execution confirmations are sent immediately after each trade, providing complete transparency in the trading process.

5.2 Reports and Advanced Analysis

Interactive dashboards present key metrics in real time, including strategy performance, risk allocation, and market trends. Backtesting reports provide detailed analysis of historical performance, including risk-return metrics and comparisons with market benchmarks.

Portfolio analysis includes sophisticated metrics such as correlation between strategies, risk contribution, and capital optimization. Historical comparisons allow users to evaluate the relative effectiveness of different trading approaches.

5.3 Integration and Connectivity

The RESTful API provides programmatic access to all system features, facilitating integration with third-party applications. WebSocket feeds deliver real-time market data with minimal latency. Native integration with MetaTrader 5 allows direct trade execution without manual intervention.

Webhooks enable automatic notifications to external systems, while structured data feeds facilitate analysis in specialized tools. Comprehensive API documentation ensures efficient integration for third-party developers.

5.4 Billing and Administrative Management

Subscription usage reports are generated automatically, providing complete transparency into resource consumption. Automatic billing is based on actual usage metrics, ensuring users pay only for the services consumed. Compliance reports are distributed monthly to meet regulatory requirements.

SLA metrics are automatically calculated and reported for enterprise customers, including availability, latency, and signal accuracy. This information is crucial to maintaining the trust of institutional customers.

6 Contingency Plan

6.1 Risk Identification and Classification

Critical risk scenarios include complete database system failure, loss of connectivity with major exchanges, security compromises, and massive data corruption. These events require immediate response due to their potential impact on users' trading operations.

High-level risks include performance degradation exceeding 500 milliseconds, AI algorithm failures generating erroneous signals, system overload during high volatility, and synchronization issues between geographic replicas. Although less critical, these events require rapid intervention to maintain service quality.

6.2 Immediate Response Strategies

Immediate response is automatically activated within the first five minutes of the incident. Backup systems are activated without manual intervention, transferring operations to the secondary database located in a different geographic region. The 24/7 operations team receives immediate notifications through multiple communication channels.

Degraded mode is automatically implemented to maintain critical operational functions, prioritizing the delivery of trading alerts and the processing of active orders. Users receive transparent notifications about the system status without creating unnecessary panic.

6.3 Recovery and Restoration

The short-term recovery process focuses on a detailed diagnosis of the problem and the implementation of temporary solutions within the first 30 minutes. Proactive communication with affected users maintains transparency and confidence in the system. Automatic escalation to the development team is triggered for issues requiring specialized intervention.

The final solution is typically implemented within the first four hours, including a complete data integrity check and full service restoration. Post-mortem analysis is systematically conducted to identify root causes and implement preventative improvements.

6.4 Recovery Resources and Objectives

The backup infrastructure includes secondary replicas in different geographic regions, utilizing AWS's Multi-AZ functionality for maximum availability. Read-only databases maintain access to historical queries during incidents affecting the primary system. The Redis distributed cache provides fast access to critical data during emergencies.

Recovery objectives establish a maximum RTO (Recovery Time Objective) of 15 minutes and a maximum RPO (Recovery Point Objective) of 1 minute of data loss. Target system availability is maintained at 99.95% uptime, with incident detection times of less than 2 minutes.

7 Additional Strategic Considerations

7.1 Scalability and Future Growth

The system's distributed architecture is designed to support 10x growth in current volume without fundamental architectural modifications. The implementation of microservices allows for independent scaling of different components based on specific demand. Horizontal sharding by geographic region distributes the load efficiently as the user base expands globally.

Auto-scaling based on load metrics automatically adjusts to fluctuating demand, optimizing costs during periods of low activity and ensuring performance during peak market volatility. This flexibility is crucial for handling unpredictable market events that can multiply the system load instantly.

7.2 Cost and Efficiency Optimization

The implementation of tiered storage enables cost-effective storage of historical data by automatically moving less-accessed information to less expensive storage media. Compression

algorithms specifically for financial time series significantly reduce storage requirements without compromising access speed.

Continuous optimization of costly queries is performed through automatic analysis of usage patterns, identifying opportunities for improvement without manual intervention. The use of reserved instances for predictable loads balances guaranteed availability with long-term cost optimization.

7.3 Innovation and Continuous Improvement

The implementation of A/B testing allows for controlled evaluation of new features without impacting the experience of existing users. Machine learning algorithms are used to automatically optimize system parameters, continuously improving operational efficiency. The feedback loop with users provides valuable information for developing new features.

Continuous benchmarking against competitors ensures that the system maintains competitive advantages in terms of performance and functionality. This constant evaluation informs investment decisions in technology and product development.

8 Success Metrics and Monitoring

8.1 Technical Performance Indicators

The average latency for alert delivery remains below 50 milliseconds, crucial for effective trading decisions. System availability exceeds 99.95%, providing exceptional reliability for users who rely on the system for their financial operations. Throughput supports more than 100,000 transactions per second, efficiently handling activity spikes during significant market events.

Response times for complex queries remain below 200 milliseconds, ensuring that advanced analytics execute without perceptible delays. These metrics are continuously monitored and reported to key stakeholders to maintain transparency regarding system performance.

8.2 Business Value Indicators

Trading signal accuracy exceeds 15%, providing tangible value to users and justifying premium subscriptions. User satisfaction remains above 4.5 out of 5, reflecting the quality of service and the effectiveness of the tools provided. Subscriber retention exceeds 85%, indicating that users find continued value in the platform.

Onboarding time for new users remains under 5 minutes, minimizing barriers to entry and maximizing the conversion of potential users. These metrics directly correlate with the financial success and long-term sustainability of the platform.

This systemic analysis lays the foundation for a robust, scalable, and reliable trading platform that can withstand the demands of modern financial markets. The proposed architecture efficiently balances the performance, security, and regulatory compliance requirements necessary to operate in a global financial environment.

The implementation of comprehensive controls and detailed contingency plans ensures that the system can maintain critical operations even during adverse events. Monitoring metrics provide complete visibility into system performance, facilitating informed decision-making and continuous improvement.

The scalability and cost optimization strategy positions the platform for long-term, sustainable growth, while the focus on continuous innovation ensures it remains competitive in a rapidly evolving technology market.

9 Concurrency Analysis

Concurrency control is paramount in financial systems where multiple users, automated agents, and processes interact with shared data simultaneously. This section identifies concurrency scenarios, analyzes potential issues, proposes solutions, and justifies their applicability to the trading platform.

9.1 Scenarios with Concurrent Access

The platform's operations involve several scenarios where concurrent access to data is inevitable:

- **Trade Execution:** Multiple traders or algorithms execute trades concurrently, updating PostgreSQL tables such as `accounts`, `trades`, and `portfolios`.
- **Market Data Ingestion:** Real-time market feeds (e.g., stock prices, forex rates) are ingested into MongoDB while analytics processes and alert systems read the data simultaneously.
- **Support Ticket Management:** Customer support agents update tickets in MongoDB, often addressing the same issue concurrently during peak times.
- **User Account Management:** Users modify their profiles, preferences, or subscriptions in PostgreSQL, with potential overlap during high-traffic periods.

9.2 Potential Concurrency Problems

These scenarios introduce the following concurrency-related risks:

- **Race Conditions:** In trade execution, two transactions might read an account balance simultaneously (e.g., $10,000), each deducting funds (e.g., $6,000), resulting in an incorrect final balance (e.g., $4,000 instead of -$2,000).
- **Lost Updates:** Two support agents updating a ticket's status or notes concurrently might overwrite each other's changes, losing critical information.
- **Dirty Reads:** During market data updates, a process might read a partially updated document, leading to inconsistent analytics outputs.
- **Deadlocks:** In PostgreSQL, a transaction locking an account while another locks a trade record could result in a circular wait, halting both operations.

9.3 Proposed Solutions

To address these issues, we propose tailored concurrency control mechanisms:

- **Trade Execution (PostgreSQL):**
  - **Serializable Isolation:** Ensures transactions execute as if in serial order, preventing race conditions.
  - **Implementation:**

```
BEGIN;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
UPDATE accounts SET balance = balance - 6000 WHERE user_id = '
    user-123';
INSERT INTO trades (user_id, amount) VALUES ('user-123', 6000);
COMMIT;
```

- **Alternative:** Row-level locking with `SELECT ... FOR UPDATE`.

```
BEGIN;
SELECT * FROM accounts WHERE user_id = 'user-123' FOR UPDATE;
UPDATE accounts SET balance = balance - 6000 WHERE user_id = '
    user-123';
COMMIT;
```

- **Market Data Ingestion (MongoDB):**
  - **Optimistic Concurrency with Versioning:** Uses a version field to detect conflicts in append-heavy operations.
  - **Implementation:**

```
db.market_data.findAndModify({
    query: { _id: "data-123", version: 1 },
    update: { $push: { prices: 150.25 }, $inc: { version: 1 }
        },
    new: true
});
```

- **Support Ticket Management (MongoDB):**
  - **Versioning:** Prevents lost updates by requiring version matching.
  - **Implementation:**

```
db.support_requests.updateOne({
    _id: "ticket-456",
    version: 2
}, {
    $set: { status: "resolved" },
    $inc: { version: 1 }
});
```

- **Deadlock Prevention (PostgreSQL):**
  - **Lock Order:** Enforces a consistent sequence (e.g., accounts before trades).
  - **Implementation:** Standardize transaction logic accordingly.

9.4 Justification of Solutions

The chosen mechanisms align with the platform's requirements:

- **Serializable Isolation:** Guarantees consistency in financial transactions, critical for regulatory compliance, despite potential performance trade-offs.
- **Optimistic Concurrency:** Efficient for high-velocity market data with rare conflicts, optimizing throughput.
- **Versioning:** Lightweight and effective for support tickets, avoiding locking overhead.
- **Lock Order:** Proactively reduces deadlock risks, ensuring system reliability.

10 Parallel and Distributed Database Design

To support big data volumes, multi-location access, and high availability, we propose a distributed architecture integrating parallel processing and geo-distribution.

10.1 High-Level Design

- **PostgreSQL (Transactional Data):**
  - **Sharding:** Use Citus to shard by `user_id`.
  - **Replication:** Deploy read replicas per shard.
  - **Geo-Distribution:** Host shards in US, EU, and Asia regions.
- **MongoDB (Market Data):**
  - **Sharding:** Shard by `instrument_symbol`.
  - **Replication:** Use replica sets.
  - **Geo-Distribution:** Replicate across regions.
- **Snowflake (Analytics):**
  - **Distribution:** Multi-cluster architecture.
  - **Geo-Distribution:** Multi-region deployment.
- **Integration:** Apache Kafka synchronizes data across systems.

10.2 Diagrams

10.2.1 Data Distribution

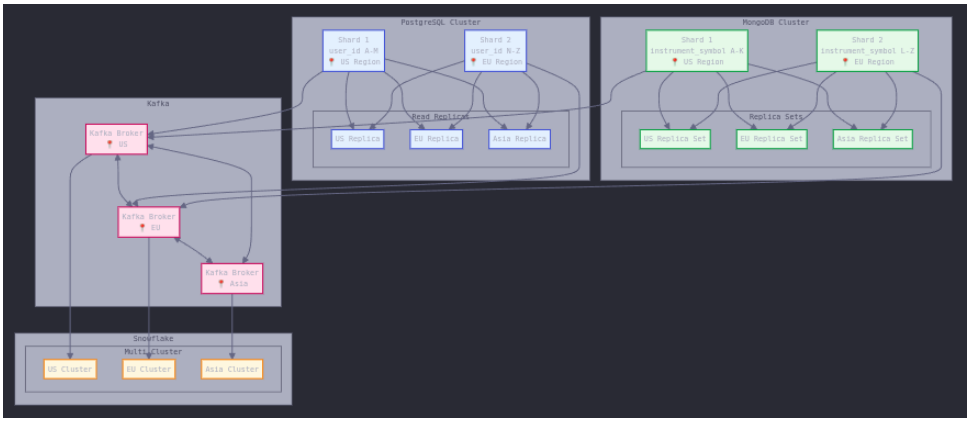

Figure 1: Data distribution diagram

10.2.2 Query Flow

10.3 Justification of Design Decisions

- **Sharding (PostgreSQL):** Co-locates user data, optimizing query performance and scalability.
- **Sharding (MongoDB):** Enhances instrument-specific query efficiency, supporting real-time analytics.
- **Geo-Distribution:** Reduces latency and ensures regulatory compliance (e.g., GDPR).
- **Replication:** Provides fault tolerance and read scalability, critical for 24/7 operations.

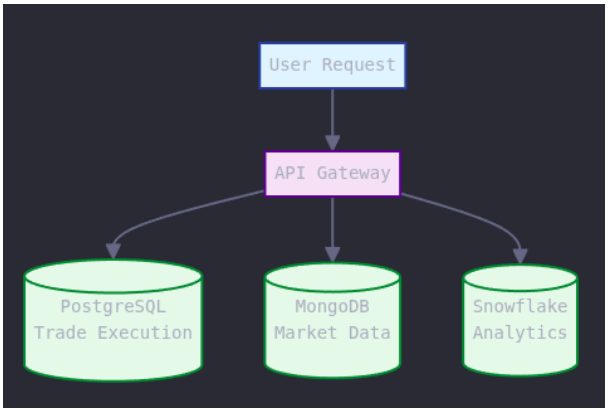

Figure 2: Query flow diagram

11 Strategies to Improve Performance

In the fast-paced domain of high-frequency trading, the performance of a database system directly influences operational success. The ability to process vast datasets, execute trades with minimal latency, and deliver rapid analytical insights is critical. To achieve these objectives, we propose three interconnected strategies: horizontal scaling, parallel query execution, and data replication. Each leverages parallelism and distribution to enhance throughput and responsiveness, while addressing the inherent challenges of complexity and consistency.

11.1 Strategy 1: Horizontal Scaling

Horizontal scaling, often termed "scaling out," involves augmenting a database system by adding additional machines or nodes to distribute workload. Unlike vertical scaling, which upgrades a single server's capacity, horizontal scaling offers a scalable and cost-effective solution for managing increasing data volumes and transaction rates. For our trading platform, this strategy is pivotal given the continuous influx of transactional data (e.g., trades, user activities) and market information (e.g., price feeds, news).

For transactional data managed in PostgreSQL, we employ Citus, an extension that enables distributed database functionality. Sharding is implemented based on the `user_id` column, partitioning user-related data across multiple nodes. For instance, users with IDs from A to M may reside on Shard 1, while those from N to Z are allocated to Shard 2. This approach ensures that queries targeting a specific user—such as retrieving trade history—are confined to a single shard, minimizing resource contention and enabling parallel processing of unrelated requests. A practical example is a query like:

```
1 SELECT * FROM trades WHERE user_id = 'user-123' ORDER BY execution_date
      DESC;
```

This query executes efficiently on the designated shard, avoiding full-table scans across the entire dataset.

In MongoDB, which handles real-time market data, we leverage its native sharding capabilities by partitioning data based on `instrument_symbol`. This distributes data for distinct financial instruments (e.g., stocks, forex pairs) across nodes, facilitating parallel ingestion and querying. During a market surge, for example, price updates for multiple instruments can be processed concurrently, ensuring real-time availability without bottlenecks.

The benefits of horizontal scaling are manifold. It allows the platform to accommodate growing datasets and transaction volumes by utilizing commodity hardware, reducing dependency on expensive high-end servers. This scalability is crucial for supporting an expanding

user base and increasing market data complexity. Furthermore, it provides a flexible framework for future growth, as additional nodes can be integrated seamlessly.

However, horizontal scaling introduces challenges, notably in maintaining data consistency across distributed nodes. In PostgreSQL, preserving ACID (Atomicity, Consistency, Isolation, Durability) properties across shards requires careful design. Cross-shard transactions, which involve data from multiple nodes, can degrade performance and are avoided by co-locating related data (e.g., a user's trades and account details) on the same shard. Additionally, operational complexity increases with the need to manage network partitions and node failures. To mitigate these, we implement robust monitoring and alerting systems, ensuring rapid detection and resolution of issues.

11.2 Strategy 2: Parallel Query Execution

Parallel query execution enhances performance by dividing complex queries into smaller, concurrently executable tasks across multiple compute nodes. This is particularly advantageous for analytical workloads, which often involve aggregations, joins, and computations over large datasets—a common requirement in our platform for tasks like backtesting trading strategies.

Snowflake, our choice for analytical processing, inherently supports this capability through its cloud-native, multi-cluster architecture. When a query is submitted, Snowflake's query engine automatically partitions the workload, distributing data segments across nodes for parallel processing. For example, backtesting a trading strategy over a year's historical data might involve calculating returns across multiple instruments and time periods. Snowflake can split this task into monthly or instrument-specific segments, processing each in parallel before aggregating the results. A simplified query might resemble:

```
SELECT instrument_symbol, AVG(profit_loss)
FROM historical_trades
GROUP BY instrument_symbol;
```

Here, each group is computed independently on separate nodes, reducing execution time significantly.

To maximize efficiency, we optimize data organization and query design:

- **Clustering Keys:** Data is clustered by `timestamp` or `instrument_symbol`, minimizing data movement between nodes.
- **Materialized Views:** Precomputed aggregations, such as daily trade summaries, are stored to accelerate frequent queries.
- **Query Structuring:** We employ window functions over subqueries to enhance parallelism and reduce computational overhead.

This approach drastically reduces latency for analytical tasks, enabling traders to iterate strategies rapidly. It also scales compute resources dynamically, accommodating larger datasets. However, the cost of running multiple nodes can escalate, particularly for resource-intensive queries. We address this by implementing auto-suspend policies for idle compute warehouses and refining queries to eliminate redundant computations, balancing performance with cost efficiency.

11.3 Strategy 3: Data Replication

Data replication maintains multiple synchronized copies of data across nodes or regions, enhancing read performance and system resilience. For our platform, replication serves dual purposes: distributing read loads and ensuring high availability, both critical in a 24/7 trading context.

In PostgreSQL, read replicas are deployed for each shard, offloading read-only queries (e.g., trade history retrieval) from the primary database. These replicas also serve as failover candidates, minimizing downtime if the primary fails. In MongoDB, replica sets are established for each shard, distributing read operations and providing redundancy. Geo-distribution is achieved by placing replicas in regions like the US, EU, and Asia, reducing latency for global users. For instance, a European trader querying market data accesses a local replica, improving response times.

Replication enhances scalability by allowing multiple read points and ensures fault tolerance through redundant data copies. It aligns with the platform's requirement for low-latency, multi-region access, supporting a global user base effectively.

A primary challenge is replication lag, where replicas may temporarily reflect outdated data. This can affect scenarios requiring real-time accuracy, such as account balance checks. We mitigate this by tolerating slight staleness for non-critical reads (e.g., market snapshots) and enforcing read-your-writes consistency for critical operations. Operational overhead is managed through health monitoring and automated failover mechanisms, ensuring replica synchronization and system reliability.

12 Improvements to Workshop 2

Building on Workshop 2's hybrid database architecture (PostgreSQL, MongoDB, Snowflake), we have refined the system based on performance analysis and feedback. These improvements enhance efficiency, scalability, and consistency, aligning with the platform's evolving needs.

12.1 Review and Enhancements

Workshop 2 established a foundation for managing transactional, market, and analytical data. Post-analysis revealed bottlenecks, prompting the following enhancements:

- **Schema Optimization:** In PostgreSQL, trade history queries were slow due to growing data volumes. We added a composite index on the `trades` table:

  ```
  CREATE INDEX idx_trades_user_date ON trades (user_id, execution_date DESC);
  ```

  This accelerates queries filtering by user and date, reducing execution times significantly.

- **Query Efficiency:** The real-time portfolio risk assessment query was optimized using window functions:

```
WITH current_positions AS (
    SELECT user_id, instrument_symbol,
           SUM(CASE WHEN side = 'buy' THEN quantity ELSE -
              quantity END) AS net_position,
           AVG(entry_price) AS avg_entry_price
    FROM trades
    WHERE status IN ('open', 'partial')
    GROUP BY user_id, instrument_symbol
)
SELECT user_id, SUM(net_position * avg_entry_price) AS
   total_exposure
FROM current_positions
GROUP BY user_id;
```

  This reduces complexity and leverages database optimizations for grouped calculations.

- **Concurrency Controls:** To prevent race conditions during trade execution, we adopted Serializable isolation in PostgreSQL:

```
BEGIN;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
UPDATE accounts SET balance = balance - 6000 WHERE user_id = 'user-123';
INSERT INTO trades (user_id, amount) VALUES ('user-123', 6000);
COMMIT;
```

  This ensures data integrity under high concurrency.

- **Distributed Enhancements:** Sharding and replication were integrated, as detailed in Section 3, enhancing scalability and availability.

12.2 Justification of Improvements

These changes align with key objectives:

- **Performance:** Indexes and query optimizations ensure rapid data access, critical for user experience.
- **Reliability:** Concurrency controls maintain data consistency, vital for financial accuracy.
- **Scalability:** Distributed design supports growth in users and data volume.

Together, they enhance the platform's ability to deliver real-time insights and robust operations.

12.3 Conclusion of Improvements

These refinements transform Workshop 2's deliverables into a production-ready system, capable of meeting high-frequency trading demands with improved efficiency and resilience.

13 Conclusion

This report enhances the trading platform's database architecture, ensuring it meets the demands of concurrency, scalability, and performance in a global financial context. The proposed solutions provide a foundation for future growth and reliability.

14 References

References

[1] PostgreSQL Documentation, *Transaction Isolation*, `https://www.postgresql.org/docs/current/transaction-iso.html`.

[2] MongoDB Documentation, *Optimistic Concurrency Control*, `https://docs.mongodb.com/manual/core/optimistic-concurrency-control/`.

[3] Snowflake Documentation, *Parallel Query Execution*, `https://docs.snowflake.com/en/user-guide/performance-query-parallelism`.

15 Glossary

ACID A set of properties (Atomicity, Consistency, Isolation, Durability) that guarantee reliable processing of database transactions.

Alert Real-time notification generated by the system when trading opportunities are detected based on strategy analysis and market conditions.

Alpha Vantage Financial data provider that offers real-time and historical market data through APIs, used as a data source for the platform.

API (Application Programming Interface) Set of protocols and tools that allow different software applications to communicate with each other.

API Gateway A service that routes and manages API calls in a microservices environment.

Backtest Process of testing a trading strategy using historical market data to evaluate its potential performance before implementing it in live trading.

BI Tools Business Intelligence tools such as Looker, Tableau, and Google Data Studio used for visualization and reporting.

Business Model Canvas Strategic management template that provides a visual chart with elements describing a firm's value proposition, infrastructure, customers, and finances.

CDC Change Data Capture; technique for tracking and propagating database changes in real-time.

Dashboard Visual interface displaying KPIs, analytics, and insights to end-users.

Entity-Relationship Diagram (ERD) Visual representation of the relationships between entities in a database system.

ETL Extract, Transform, Load; the data pipeline process for integrating and preparing data.

Financial Instrument Tradable asset such as stocks, forex pairs, cryptocurrencies, commodities, indices, or bonds.

Hexagonal Architecture Software design pattern that separates the core domain logic from external concerns through ports and adapters.

Market Data Real-time and historical information about financial markets, including price movements, volume, and other trading indicators.

Materialized Views Pre-computed views in a database that store results of complex queries for fast access.

MetaTrader 5 (MT5) Popular trading platform that provides access to financial markets and automated trading capabilities.

MongoDB A NoSQL document database used for managing time-series and support data.

MVCC Multi-Version Concurrency Control; a database feature allowing concurrent transaction processing without locking.

OLTP Online Transaction Processing; a class of systems optimized for managing transactional data (e.g., trades, user actions).

OLAP Online Analytical Processing; systems used for querying and analyzing large volumes of data.

Polyglot Persistence Architectural approach using multiple types of databases, each suited for specific tasks.

Portfolio Collection of financial investments and trading strategies managed together with allocated capital distribution.

Portfolio Manager Professional responsible for making investment decisions and managing a portfolio of securities to meet specific investment objectives.

PostgreSQL An open-source relational database system used for transactional workloads in the platform.

Primary Key (PK) Database constraint that uniquely identifies each record in a table.

Redis An in-memory data store used for caching and low-latency data access.

Risk/Reward Ratio Measure that compares the potential profit of a trade to its potential loss, used to evaluate trading opportunities.

Scalability Ability of a system to handle increased load and grow in capacity while maintaining performance.

Sharding A method of database partitioning that improves scalability by splitting data across multiple machines.

Sharpe Ratio A risk-adjusted return metric calculated from returns and volatility.

Strategy Systematic approach to trading that defines rules and conditions for entering and exiting positions in financial markets.

Technical Indicator Mathematical calculation based on historical price and volume data used to predict future market movements.

Time-Series Data Data indexed in time order, often used for market data like prices or volumes.

Trade A record of a market transaction executed by the user under specific strategies.

Trade Execution Process of completing a buy or sell order in the financial markets through a broker or trading platform.

Trading Strategy Set of rules and criteria that determine when to buy or sell financial instruments based on market analysis.

User Story Brief description of a feature from the perspective of the end user, typically following the format "As a [user], I want [goal] so that [benefit]."

UUID (Universally Unique Identifier) 128-bit identifier used to uniquely identify information in database systems.

Volatility Statistical measure of the degree of variation in a trading price series over time, indicating the level of risk.

XM Broker Online trading platform that provides access to financial markets and serves as the broker integration for trade execution.

Yahoo Finance (yfinance) Popular library and data source that provides access to financial market data, including stock prices and historical information.