Scaling Strategy for Real-Time Financial Data Feeds with TimeScaleDB

Introduction

Handling real-time financial data feeds requires a robust and scalable infrastructure. TimeScaleDB, with its focus on time-series data, provides features that can significantly enhance the performance of a real-time data pipeline. This strategy outlines key considerations for scaling the system efficiently.

1. Hypertables for Query Optimization

Hypertables in TimeScaleDB play a pivotal role in managing large volumes of time-series data efficiently. They offer a sophisticated mechanism for partitioning data into chunks, optimizing storage, and, subsequently, enhancing query performance. In our context, the following hypertables have been defined:

```
SELECT create_hypertable('trades', 'timestamp', migrate_data ⇒ true);
SELECT create_hypertable('es', 'timestamp', migrate_data ⇒ true);
SELECT create_hypertable('bz', 'timestamp', migrate_data ⇒ true);
```

Key Features:

- Time-Series Optimization: TimeScaleDB's hypertables enable effective time-series optimization, crucial for handling continuous streams of financial data. The partitioning of data into chunks facilitates faster access to relevant time intervals.
- Hyperfunctions Utilization: Hypertables enable the use of TimeScaleDB's hyperfunctions, such as `time_bucket`, which optimize query performance for time-series data. These functions simplify the process of working with time intervals, ensuring efficient and accurate calculations.

2. Continuous Aggregates and Automatic Refresh Policies

Continuous aggregates in TimeScaleDB allow for pre-aggregation of data, significantly reducing query times.
In the context of the `trades` table, consider the example where a daily High-Low-Close-Volume (HLCV) continuous aggregate (`trades_hlcv_daily`) has been defined. This aggregate captures the daily high, low, close, and volume values for each symbol:

```
CREATE MATERIALIZED VIEW trades_hlcv_daily
    WITH (timescaledb.continuous) AS
    SELECT  symbol,
        time_bucket('1 day', timestamp) AS bucket,
        MAX(high) AS high,
        MIN(low) AS low,
        LAST(close, timestamp) AS close,
        SUM(volume) AS volume
    FROM trades
    GROUP BY bucket, symbol
    WITH NO DATA;
```

By leveraging automatic refresh policies, aggregates can be updated in near real-time as new data arrives. This ensures that performance remains optimal even as the dataset grows. The automatic refresh policy guarantees that these aggregates are promptly updated whenever new trade data is ingested:

```
SELECT add_continuous_aggregate_policy( 'trades_hlcv_daily',
                                       start_offset ⇒ INTERVAL '1 day',
                                       end_offset ⇒ INTERVAL '1 minute',
                                       schedule_interval ⇒ INTERVAL '1 day');
```

Implementing these features enhances the efficiency of the real-time pipeline by providing quicker access to summarized data, especially useful for analytics and reporting.

Optimization analysis on continuous aggregates
In this optimization analysis, we examined the impact of utilizing continuous aggregates on querying performance. The comparison between running a query with and without continuous aggregates yielded compelling results:

- **Query with Continuous Aggregates:**

```
CREATE MATERIALIZED VIEW trades_hlcv_daily
WITH (timescaledb.continuous) AS
SELECT  symbol,
       time_bucket('1 day', timestamp) AS bucket,
       MAX(high) AS high,
       MIN(low) AS low,
       LAST(close, timestamp) AS close,
       SUM(volume) AS volume
FROM trades
GROUP BY bucket, symbol
WITH NO DATA;

CALL refresh_continuous_aggregate( 'trades_hlcv_daily',
                                        INTERVAL '2 years',
                                        INTERVAL '15 min');

SELECT * FROM trades_hlcv_daily;
```

Execution Time: 56.129 ms

- **Query without Continuous Aggregates:**

```
SELECT  symbol,
        time_bucket('1 day', timestamp) AS bucket,
        MAX(high) AS high,
        MIN(low) AS low,
        LAST(close, timestamp) AS close,
        SUM(volume) AS volume
FROM trades
GROUP BY bucket, symbol;
```

```
Execution Time: 1626.958 ms
```

The stark contrast in execution times serves as a first indicator of the enhanced querying performance achievable by leveraging continuous aggregates.This notable discrepancy underscores the efficiency gains attained by employing continuous aggregates when processing extensive datasets. This initial analysis highlights the potential for substantial performance improvements, especially when dealing with vast amounts of financial time-series data.

3. Additional Considerations

While Continuous Aggregates, Automatic Refresh Policies, and Hypertables serve as integral components for optimizing performance in TimeScaleDB, the following considerations contribute to a holistic strategy for scalability and efficiency:

- **Cloud Deployment for Scalability**: Leveraging cloud platforms introduces scalability opportunities, allowing dynamic resource scaling based on demand. Cloud environments provide elasticity and managed services, reducing operational overhead and focusing on performance optimization and functionality.
- **Data Partitioning**: Strategic data partitioning based on symbols, time intervals, or other dimensions enhances scalability and query performance. Thoughtful data distribution ensures efficient data management and retrieval.
- **Load Balancing**: Distributing queries across multiple nodes or replicas helps balance processing loads, preventing bottlenecks and ensuring consistent performance. Load balancing is crucial for handling varying workloads effectively.

Conclusion

The strategy for scaling real-time financial data feeds with TimeScaleDB focuses on optimizing system performance through the implementation of Hypertables, Continuous Aggregates, and Automatic Refresh Policies. The utilization of Hypertables enhances time-series data management, while Continuous Aggregates, exemplified by the daily HLCV aggregate, significantly reduces query times. Our optimization analysis demonstrates substantial efficiency gains, emphasizing the potential for improved performance, particularly with extensive financial time-series datasets. Additional considerations, including Cloud Deployment for Scalability, Data Partitioning, and Load Balancing, contribute to a comprehensive and scalable future approach for real-time data processing in dynamic financial environments.