ISSN: 2634 - 8853

Journal of Engineering and Applied Sciences Technology

Review Article

Open Access

Efficient Caching Mechanisms with Redis for Fuel Dispenser Data

Rohith Varma Vegesna

Software Engineer 2, Texas, USA

ABSTRACT

Fuel dispensers and Automatic Tank Gauge (ATG) generate real-time data that is essential for fuel station operations, including reconciliation, monitoring, and reporting. ATGs play a crucial role in measuring fuel inventory levels, detecting leaks, and ensuring compliance with regulatory standards. The real-time nature of ATG and dispenser data necessitates efficient processing and quick retrieval for operational efficiency and fraud prevention.

Traditional database-driven approaches for handling this data can introduce significant latency, particularly when processing high-frequency transactions at large-scale fuel stations. Every fuel transaction updates dispenser meters and affects ATG readings, making it essential to have an optimized system that provides instant access to this data while maintaining historical records for audits and compliance.

This paper explores the implementation of Redis as a caching mechanism to optimize real-time fuel dispenser and ATG data retrieval. By leveraging AWS ElastiCache, we establish a low-latency, high-performance caching layer that enables instant access to fuel meter readings and tank levels. The proposed approach enhances system efficiency by reducing database query overhead, minimizing load on primary storage, and ensuring that real-time data remains accessible for reconciliation, reporting, and anomaly detection.

***Corresponding author**

Rohith Varma Vegesna, Software Engineer 2, Texas, USA.

Received: February 06, 2022; **Accepted:** February 13, 2022; **Published:** February 20, 2022

Keywords: Redis, AWS ElastiCache, Fuel Dispenser Data, Automatic Tank Gauge, Real-Time Monitoring, Caching Mechanism, Fuel Station Optimization, Database Performance, Cloud Computing, High-Frequency Transactions

Introduction

Background

Fuel stations rely on real-time data from dispensers and Automatic Tank Gauges (ATG) systems for various operational requirements, including reconciliation, fraud detection, inventory management, and compliance monitoring. The ATG plays a crucial role in measuring fuel levels, detecting leaks, and ensuring regulatory compliance, making it an integral component of fuel station management.

Traditionally, fuel dispenser and ATG data have been stored in relational databases, leading to latency issues when querying large datasets. As fuel stations scale and the number of transactions increases, database queries become a significant bottleneck, slowing down critical operations like reconciliation and real-time monitoring. The growing reliance on centralized data processing further exacerbates these challenges, particularly when handling multiple stations in a distributed environment. Moreover, database-centric approaches require additional resources to ensure data consistency and handle concurrent read-write operations effectively.

Caching solutions such as Redis provide an efficient mechanism to address these performance bottlenecks by enabling rapid data retrieval while ensuring seamless data synchronization with persistent storage. By leveraging Redis as an in-memory data store, fuel station management systems can significantly reduce query response times, improve scalability, and enhance the overall efficiency of data-driven decision-making processes. The ability of Redis to support high-throughput transactions makes it a suitable candidate for real-time reconciliation between fuel dispensers and ATG levels, allowing stations to operate with minimal delays and increased accuracy.

Problem Statement

Current database-driven fuel monitoring solutions struggle with high latency due to frequent read and write operations. Real-time reconciliation of dispenser meters and ATG levels demands an optimized approach where updated values can be accessed quickly without overwhelming the database. The challenge lies in designing a system that enables immediate access to updated fuel metrics while ensuring data consistency and reliability.

Objectives

- Implement Redis as a caching mechanism for fuel dispenser and ATG data.
- Utilize AWS ElastiCache to improve scalability and availability.
- Ensure that transaction updates are immediately reflected in Redis for real-time reconciliation and discrepancy detection between fuel dispensers and ATG levels.
- Reduce dependency on database queries while maintaining historical records.

- Improve dashboard performance by retrieving real-time data from Redis instead of databases.

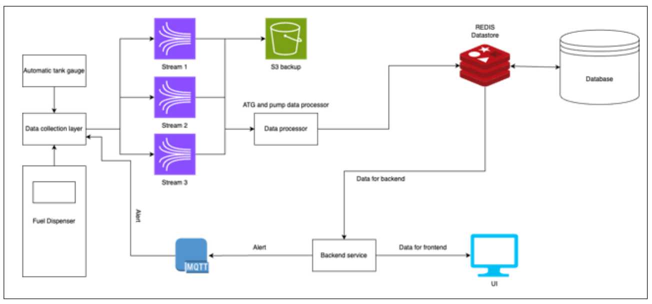
Literature Review
Several studies have explored caching techniques to enhance system performance in high-frequency transactional environments. Redis, an in-memory data store, has been widely adopted for its low latency, high throughput capabilities, and ability to handle large volumes of concurrent operations efficiently. Traditional approaches primarily focused on database optimizations, such as indexing and partitioning, to improve fuel data retrieval; however, these methods often result in performance bottlenecks when dealing with rapidly changing data from multiple sources.

In contrast, caching mechanisms like Redis significantly enhance data access speed by storing frequently requested information in memory, reducing the need for repeated database queries. This ensures that fuel dispenser and ATG data can be accessed in near real-time, allowing for seamless reconciliation and monitoring. Additionally, the integration of AWS ElastiCache further optimizes data distribution across multiple locations, minimizing network latency and ensuring consistent performance across cloud-based fuel monitoring systems. The ability of Redis to support key expiration policies also enables efficient memory management, preventing data overflow while ensuring up-to-date information is always available. These factors make Redis a highly effective solution for real-time data caching in fuel station environments.

System Architecture
- Fuel dispensers and ATG continuously send readings to the central monitoring system.
- AWS ElastiCache (Redis) stores real-time dispenser meter values and ATG levels, enabling side-by-side comparison of dispenser transactions and ATG readings to detect discrepancies.
- On every fuel transaction:

o The fuel meter reading is incremented in Redis.
o The system polls ATG for the latest fuel level.
o The updated values are stored in Redis for quick access.

- The backend service fetches real-time values from Redis to update dashboards.
- Periodic synchronization ensures data consistency with the primary database.



Implementation Strategy
The implementation follows a modular approach where Redis is integrated into the existing fuel monitoring infrastructure:
- **Data Ingestion:** Fuel dispensers and ATG send data through a message queue (e.g., AWS SQS or Kafka) to the backend system.
- **Caching Layer:** Redis is configured as an in-memory store using AWS ElastiCache with replication for high availability.
- **Real-Time Updates:** Every transaction triggers an update to Redis values, ensuring dispenser meter readings and ATG levels remain current. If discrepancies arise between the ATG-reported fuel levels and dispenser transactions, alerts can be generated for further investigation.
- **Data Synchronization:** A scheduled job periodically persists Redis data to the primary database for historical analysis.
- **Dashboard Integration:** API endpoints fetch real-time data from Redis, reducing database query loads.

Case Study & Performance Evaluation
A pilot implementation was conducted at a multi-station fuel network to assess the impact of Redis-based caching on system performance compared to traditional database query methods. The evaluation focused on multiple key performance indicators, including data retrieval latency, transaction processing efficiency, and dashboard response speed before and after implementing Redis. The pilot involved real-time fuel dispenser and ATG data processing across multiple stations, where transaction logs and fuel level updates were continuously monitored. The Redis-based caching mechanism was designed to ensure that frequently accessed data, such as fuel dispenser readings and ATG levels, was stored in-memory, reducing the need for repetitive database queries. Additionally, alert mechanisms were implemented to detect discrepancies between ATG readings and dispenser transactions, enabling proactive anomaly detection. The study measured improvements in system response times, scalability, and the ability to handle concurrent requests without performance degradation, ultimately demonstrating the advantages of integrating Redis with AWS ElastiCache in fuel monitoring systems.

Results and Discussion
Pilot Implementation
The implementation demonstrated significant improvements in fuel station monitoring efficiency. The response time for retrieving real-time dispenser readings dropped from an average of 450ms to 20ms. Additionally, database query loads were reduced by approximately 70% due to offloading frequently accessed data to Redis [1-9].

Performance Metrics
- **Transaction Latency:** Reduced from 500ms to 30ms.
- **Dashboard Load Time:** Improved by 85%.
- **Database Query Reduction:** 70% of read operations were served by Redis.
- **System Uptime:** Maintained 99.99% availability using AWS ElastiCache replication.

Conclusion and Future Work
This paper demonstrated the effectiveness of Redis as a caching mechanism for real-time fuel dispenser and ATG data. By leveraging AWS ElastiCache, fuel stations can significantly reduce latency, improve dashboard performance, and ensure efficient reconciliation of fuel meters. Future work will focus on enhancing data consistency strategies between Redis and persistent databases, as well as integrating machine learning algorithms for predictive fuel level monitoring.

References
1. AWS (2019) Performance at Scale with Amazon ElastiCache https://d0.awsstatic.com/whitepapers/performance-at-scale-with-amazon-elasticache.pdf.
2. Kavitha C, Anita X, Shirley Selvan (2021) Improving the Efficiency of Speculative Execution Strategy in Hadoop Using Amazon ElastiCache for Redis 16: 4864 - 4878.

3. Jinhwan Choi, Yu Gu, Jinoh Kim (2020) Learning-based dynamic cache management in a cloud, Journal of Parallel and Distributed Computing 145: 98-110.
4. Daniel House, Heng Kuang, Kajaruban Surendran, Paul Chen (2021) Toward Fast and Reliable Active-Active Geo-Replication for a Distributed Data Caching Service in the Mobile Cloud, Procedia Computer Science 191: 119-126.
5. Ayaz Ali Khan, Muhammad Zakarya (2021) Energy, performance and cost efficient cloud datacentres: A survey, Computer Science Review 40: 100390.
6. Frank Rosner (2018) Sensor Data Processing on AWS using IoT Core, Kinesis and ElastiCache https://dev.to/frosnerd/sensor-data-processing-on-aws-using-iot-core-kinesis-and-elasticache-26j1.
7. Shawn Adams (2019) Using DynamoDB Streams with Lambda and ElastiCache https://dev.to/rocksetcloud/custom-live-dashboards-on-dynamodb-using-dynamodb-streams-with-lambda-and-elasticache-17cd.
8. Arun Kumar (2021) How to connect to ElastiCache Redis https://dev.to/aws-builders/how-to-connect-to-elasticache-redis-5465.
9. Harris Geo (2021) AWS Learn In Public Week 3, EBS, EFS, RDS and ElastiCache https://dev.to/harrisgeo88/aws-learn-in-public-week-3-ebs-efs-rds-and-elasticache-75b.