Real-Time Distributed Traffic Monitoring and Alert System

Distributed Systems

SOFE 4790U

Monday, December 2nd, 2024

Aqsa Asif	100755243
Rosalyn Sayim	100778284
Mohammed Fawal	100704653

Github Link: https://github.com/mohammedfawal/Distributed_FinalProject

Abstract

The Distributed Real-Time Traffic Congestion Monitoring and Alert System is a scalable, fault-tolerant, and real-time distributed system designed to monitor and manage urban traffic. This project integrates multiple traffic sensor nodes, a central server, and real-time dashboards (CLI and web-based) to collect, process, and visualize traffic data. Utilizing core distributed systems principles such as scalability, fault tolerance, and concurrency, the system addresses key challenges of centralized traffic systems, including inefficiencies, single points of failure, and high latency. Future scalability is enhanced by integrating Kafka for managing high data volumes.

Keywords

Traffic Monitoring, Distributed Systems, Scalability, Fault Tolerance, Real-Time Communication, WebSocket, Kafka.

Introduction

Urban traffic congestion is a global challenge, leading to increased pollution, longer travel times, and inefficiencies in urban mobility. Traditional centralized traffic monitoring systems, while useful, are prone to bottlenecks and single points of failure, limiting their scalability and reliability in modern, growing cities.

This project proposes a distributed approach to traffic management, leveraging real-time communication between traffic nodes and a central server. The system incorporates sensor nodes simulating traffic conditions, a central processing server for data aggregation and decision-making, and user interfaces for visualizing traffic conditions. By implementing decentralized edge processing and scalable protocols like WebSockets, the system reduces latency and ensures resilience. The proposed architecture is designed for real-world deployment, addressing key distributed systems challenges such as concurrency, fault tolerance, and scalability.

Related Work

Traffic monitoring systems have evolved from basic manual counts to advanced automated solutions. Traditional systems, such as SCATS (Sydney Coordinated Adaptive Traffic System), rely on centralized architectures, which face scalability challenges. Recent advancements include IoT-based solutions like the Intelligent Traffic Management System (ITMS), which integrates cameras and sensors to automate data collection. However, most existing systems suffer from:

1. Scalability Issues: Centralized systems struggle to handle data from rapidly growing urban areas.
2. Fault Tolerance Limitations: Single points of failure in centralized designs can disrupt operations.
3. High Latency: Centralized data aggregation delays real-time decision-making

This project builds upon distributed systems principles to address these limitations, incorporating technologies like WebSockets and Kafka to enhance real-time data processing and fault tolerance

System Overview

A. System Architecture:

The system comprises three primary components:

1. Sensor Nodes (TrafficNode):
    - Simulates real-world traffic conditions.
    - Generates data, including vehicle count, speed, and environmental conditions.
    - Transmits data to the CentralServer using WebSockets.
2. Central Server (CentralServer):
    - Aggregates and processes data from sensor nodes.
    - Serves dashboards with real-time data for visualization.
    - Maintains scalability with planned Kafka integration.
3. User Interfaces:
    - TrafficDashboard (CLI): Displays traffic data on the command line.

- TrafficDashboardWeb: Provides a web-based visualization using Thymeleaf.

B. Communication Model

The system uses WebSocket communication to achieve real-time data transfer between components. Sensor nodes establish persistent connections with the CentralServer, ensuring low-latency data updates. Future versions will integrate Kafka to handle larger data volumes and enhance fault tolerance.

Implementation

Technologies Used

1. Programming Language: Java 23
2. Build Tool: Maven
3. Communication Protocol: WebSockets
4. Frontend Framework: Thymeleaf (for TrafficDashboardWeb)
5. Future Scalability: Apache Kafka

Project Modules

1. CentralServer:

- Acts as the core processing unit.
- Receives data from multiple sensor nodes.
- Serves real-time dashboards.

2. TrafficNode:

- Simulates real-time traffic data.
- Runs as multiple instances to mimic distributed sensors.

3. TrafficDashboard:

- A CLI for basic traffic data visualization.

4. TrafficDashboardWeb:

- Web-based dashboard displaying traffic data in real time

Deployment and Usage

1. Build the project using Maven:
   - mvn clean install
2. Run modules in sequence:
   - Start CentralServer.
   - Run multiple instances of TrafficNode.
   - Start TrafficDashboard or TrafficDashboardWeb for visualization.
3. Web Dashboard Access:
   - Navigate to http://localhost:8000 and log in with default credentials.

Evaluation

The system was evaluated based on three key parameters: performance, fault tolerance, and scalability. Performance tests showed that the system could handle concurrent data streams from multiple sensor nodes with minimal latency. Fault tolerance was demonstrated by simulating node failures, where the system maintained functionality and reconnected nodes seamlessly. Scalability tests indicated that the system could accommodate additional nodes without significant performance degradation. Further enhancements, such as Kafka integration, will improve the system's capacity for large-scale deployments.

Performance

The system successfully handles real-time data updates with minimal latency. WebSocket communication ensures efficient message passing between components. Preliminary scalability tests indicate the system can support additional sensor nodes without performance degradation.

A. Fault Tolerance

The system maintains operations even if individual sensor nodes disconnect. Persistent WebSocket connections ensure that nodes can reconnect seamlessly.

B. Scalability

While the current system demonstrates scalability for small to medium deployments, Kafka integration will enable handling significantly larger data volumes.

Contributions

1. Designed a modular distributed traffic management system leveraging WebSocket communication.
2. Implemented edge processing at sensor nodes to minimize central server load.
3. Developed both CLI and web-based dashboards for real-time traffic visualization.
4. Proposed future integration with Kafka to ensure scalability for larger deployments.

Future Work

1. Integration with Kafka:
    - Transition from WebSocket to Kafka for high-throughput message streaming.
    - Enhance fault tolerance with partitioning and replication.
2. Advanced Analytics:
    - Implement machine learning models to predict traffic congestion trends.
    - Provide dynamic rerouting suggestions to users.

3. IoT Integration:

    - Replace simulated nodes with physical IoT devices for real-world deployment.

4. Improved Visualization:

    - Add map-based dashboards for enhanced data representation.

Conclusion

This project demonstrates the potential of distributed systems in addressing urban traffic challenges. By decentralizing data collection and processing, the system achieves scalability and fault tolerance, and real-time performance, laying the groundwork for future IoT-based deployments. These attributes make it a promising solution for modern smart cities, where traffic management is critical to ensuring sustainable urban mobility. Future work will focus on enhancing the system's scalability and integrating advanced analytics for predictive traffic management.

References

1. SOSP Paper: “An evaluation of the ninth SOSP submissions on how (and how not) to write a good systems paper.” ACM Digital Library.

2. Distributed Systems Principles, 3rd Edition, Andrew S. Tanenbaum.

3. Thymeleaf Documentation: https://www.thymeleaf.org/.

4. WebSocket Documentation: https://tools.ietf.org/html/rfc6455.

5. Apache Kafka Documentation: https://kafka.apache.org/.

6. OpenAI. (2023). *ChatGPT* (Oct 2023 version) [Large language model]. https://openai.com/chatgpt

(used to format/ get ideas)