San José State University

CMPE 275

Enterprise Application Development

Section 47

Spring 2024

Instructor: John Gash

Team: TechTitans

Avi Ajmera

Brahma Teja Chilumula

Steven Chang

Hasan Mhowwala

Final Project Report

Project Report

Overview

This project involves the implementation of a distributed data processing system using ingestion nodes and metadata clients. The ingestion nodes are responsible for reading data from files, processing it, and sending it to the metadata analytics leader. The metadata clients manage the nodes, including handling node discovery, elections for leadership, and coordination.

Technologies Used

- Programming Language : C++
- Framework : Qt (for network communication and threading)
- Parallel Processing : OpenMP (for parallelizing data processing tasks)
- Networking: QTcpSocket and QTcpServer from the Qt framework for network communication

- Build System : CMake

System Components

1. Ingestion Node
2. Metadata Client Node

1. Ingestion Node

The ingestion node is responsible for reading data from CSV files, processing the data, and sending it to the metadata analytics leader. It also listens for commands from the metadata client.

Key Functions and Flow

- **start()**:
  Connects to the central server.

- **startListeningForMetadataClient(quint16 port)**:
  Starts the server to listen for connections from the metadata client.

- **sendRegistrationRequest()**:
  Registers the ingestion node with the central server.

- **onServerConnected()**:
- Sends the registration request upon connecting to the server.

- **onNewMetadataClientConnection()**:
- Handles new connections from the metadata client.

- **onServerReadyRead()**:
- Processes messages from the central server.

- **onMetadataClientReadyRead()**:
- Processes commands from the metadata client.

- **processDataAndSend(const QString& folderName)**:

- Reads and processes data from CSV files in the specified folder and sends it to the metadata analytics leader if data is ready.

- **processBatch(QByteArray& batchData, QJsonArray& dataArray, int& rowIndex)**: Processes a batch of data, cleaning and converting it to JSON format.
- **sendQueryRequest(int param)**:
- Sends a query request to the metadata analytics leader.

- **processNodeDiscovery(const QJsonObject& message)**: Processes node discovery messages to update node lists and leader information.

- **sendDataToLeader()**: Sends processed data to the metadata analytics leader.

- **sendToMetadataAnalyticsLeader(const QJsonDocument& doc)**: Helper function to send data to the leader.

2. Metadata Client

The metadata client is responsible for managing node discovery, participating in leader elections, and distributing folder assignments to ingestion nodes.

Responsibilities

- Connect to the central server for node registration.
- Participate in leader election using the LCR algorithm.
- Announce leadership and distribute folder names to ingestion nodes.

Key Functions and Flow:

- **start()**: Connects to the central server.

- **sendRegistrationRequest()**:
  Registers the metadata client with the central server.

- **onServerConnected()**:
  Sends the registration request upon connecting to the server.
- **onReadyRead()**:
  Processes messages from the central server node.

- **processNodeDiscovery(const QJsonObject& message)**:
  Processes node discovery messages to update node lists and leader information.

- **processElectionMessage(const QJsonObject& message)**:
  Handles election messages to participate in leader elections using the LCR algorithm.

- **processLeaderAnnouncement(const QJsonObject& message)**:
  Handles leader announcements.

- **participateInElection()**:
  Participates in the leader election process.

- **onElectionTimeout()**:
  Starts the election process if no leader is elected after a timeout.

- **announceLeadership()**:
  Announces itself as the leader to other nodes.

- **sendHashedFolderNames()**: Distributes folder assignments to ingestion nodes based on a simple hash function.

Conclusion

This project demonstrates a distributed data processing system with robust node management, leader election, and parallel data processing capabilities. By leveraging Qt for networking and OpenMP for parallel processing, the system efficiently handles large datasets and dynamic node coordination.