Word Count Using Spark Structured Streaming In Distributed System

Chris Cao

Introduction: This project will be implemented based on Project 1. Besides what has been implemented in the first project, this project will pursue a higher standard of high availability and high concurrency.

In the previous implementation, it is too difficult to retrieve the word count results, therefore, in order to implement a more flexible and user-friendly retrieval system, we propose to integrate Redis as a lightweight, high-performance in-memory data store to persist the intermediate and final word count results computed by the Spark backend. This not only improves query efficiency but also decouples data storage from computation, enhancing the system's overall responsiveness and scalability. Meanwhile, we may also integrate Redis clusters into this project, for lower latency.

Besides, in order to ensure real-time and structured interaction between different system components, and prevent over-stuffed storage on single machine, we introduce a gRPC-based API service that enables external clients to query word count results, subscribe to updates, or manage processing tasks. Compared to traditional RESTful APIs, gRPC provides lower latency, efficient binary serialization, and native streaming support.

Method: This project builds on the architecture of Project 1 by enhancing the system's retrieval capability, modularity, and resilience. The method consists of three major components: data ingestion, stream processing, and real-time result retrieval, with new layers integrated to support high availability and flexible query access.

- **System Workflow**
  - Data Ingestion Layer (Go + Kafka)
    - As in Project 1, a Go application reads lines from a file or network input and pushes them into a Kafka topic.
    - Go routines and channel-based concurrency are leveraged to enable high-throughput ingestion and avoid blocking operations.
    - Kafka acts as a decoupled message queue, ensuring durability and retry support in case of transient failures in downstream components.
  - Stream Processing Layer (Apache Spark)
    - A Spark Structured Streaming job subscribes to the Kafka topic and continuously consumes messages.
    - Each line is tokenized into words, which are then aggregated using a groupBy().count() operation over a sliding window or append mode depending on the query requirement.
    - The computed word counts are not written to file or console, but instead persisted to Redis, allowing fast and on-demand access by external systems.
  - Result Retrieval Layer (gRPC + Redis)
    - A standalone gRPC service, implemented in Go, acts as the entry point for user queries and client applications.

    - Compared to REST APIs, gRPC offers more efficient binary transmission, built-in streaming, and better performance in high-frequency query scenarios.
- **Simplified Workflow:**
    - Client -> RPC Input Ingestion API -> Kafka -> Spark Word Counter -> Redis -> RPC Result Retrieval API
- **Parameters**
    - This implementation first ensures high-concurrency and high-availability, therefore, it should behave stable under high-concurrency scenarios, it is expected to out-perform plain implementation(with no Kafka, Redis or gRPC) in response time and results accuracy.
- **Interesting Point**
    - We use Redis to support more flexible and user-friendly way in results retrieval, such as using ZSET to rank word counts, or using HASH to build inverted index.
    - gRPC's bi-directional streaming (optional) could be used in future work for live word count subscriptions

References:

- https://github.com/IBM/sarama (Kafka library)
- https://github.com/gin-gonic/gin (Gin Framework library)
- https://github.com/grpc/grpc-go (gRPC-Go library)
- https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#quick-example (Spark Structured Streaming examples)
- https://github.com/WenbinZhu/mit-6.824-labs (Distributed System Implementation based on Spark)
- "*Spark: Cluster Computing with Working Sets"* - Matei Zaharia

Source Code: Code has been attached as a zip file.