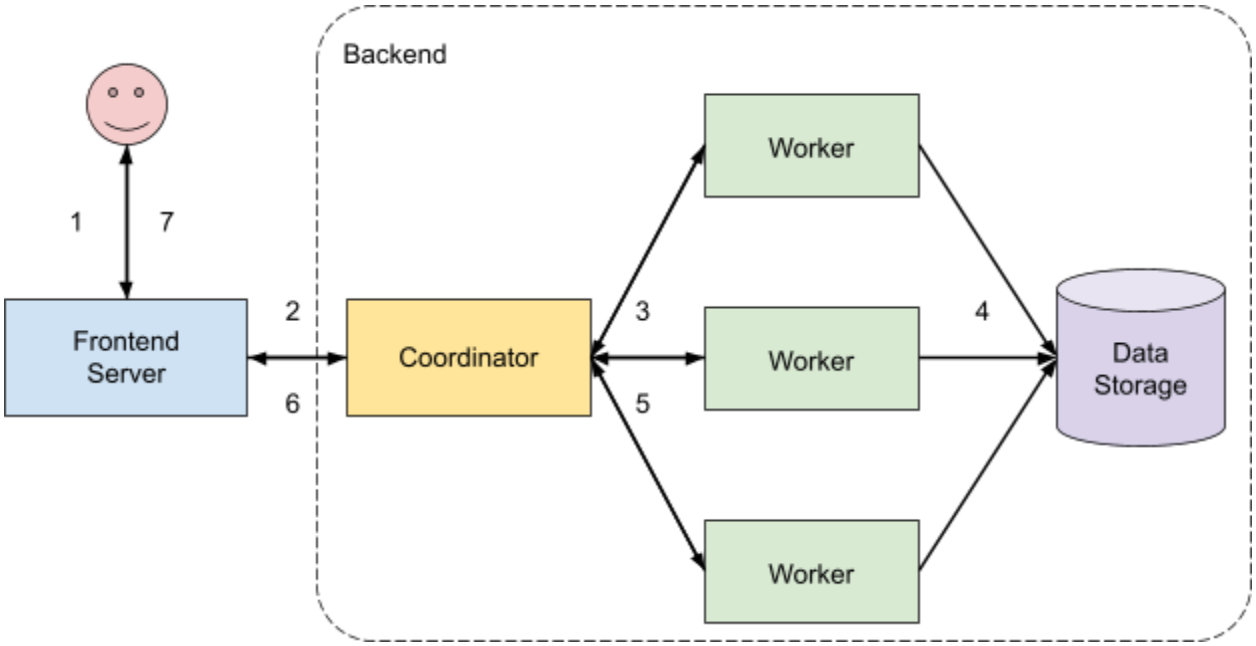
DS Final Project Executive Summary

Project Overview

The final project is a distributed document search service. The system has three major components:

- **Backend server cluster**:
  - **Coordinator**: a coordinator will be selected from the cluster through leader election. It accepts search requests from frontend and dispatches work to all the workers. Then it collects worker responses and aggregates them into the final results.
  - **Worker**: all servers that didn't win the election will be workers. Each worker will only need to handle a portion of the documents based on the coordinator's assignment.
- **Apache Zookeeper server**: zookeeper helps the backend server cluster achieve leader election and service registry.
- **Frontend server cluster**: each frontend server works independently. They receive user queries and send requests to the coordinator.

An overview of the system diagram:



The relevance model being implemented in this project is Term Frequency - Inverse Document Frequency (TF-IDF), which can be easily extended to a distributed system so that it can be executed in a parallel manner. In specific:

- **Term frequency (TF)**: the frequency of a term's occurrence in a document, normalized by the length of the document. Each worker will calculate the term frequencies for a subset of the documents.

- **Inverse Document Frequency (IDF)**: further normalize the score by how frequent the term occurs across all documents. This will be calculated by the coordinator by aggregating all worker responses.

The system relies on Apache ZooKeeper to achieve two very important functions:

- **Leader election**: each backend server will create a node in ZooKeeper to represent itself. Upon election, the server with the smallest node will be elected as the leader.
    - Every server will monitor its predecessor's node. So essentially all the servers will form a line with the coordinator as the head. When a server goes down, the successor will detect the change and monitor the new predecessor. If it was the coordinator that went down, the second server in the line will be re-elected as leader.
- **Service registration**: each server will also save its address in ZooKeeper as either coordinator or worker so that they can be discovered by other servers. In particular:
    - Frontend servers will discover a coordinator from the service registry.
    - Coordinator will discover all workers from the service registry.

Technical Impressions

Features

The features this final project provides are:

- **Availability**: both backend and frontend servers have multiple instances up running. Users can choose to access any frontend server.
- **Consistency**: document repository is replicated across backend servers so that the data is consistent.
- **Scalability**: new servers can also be added to the cluster dynamically to improve efficiency.
- **Fault-tolerance**: If one or more backend servers go down, the rest of them can still fulfill search requests.
- **Client-transparency**: frontend servers only need to talk to the coordinator and do not know about how work is dispatched and done behind the scenes.

Assumptions and design choices

There are a bunch of assumptions and the corresponding design choices that were made when implementing the system:

- **Decentralize leader election**: the leader election algorithm is designed so that each server only keeps track of its predecessor. As a result, servers essentially form a line with the leader as the head. The benefit is that upon server failure, only its successor needs to make adjustments.
    - An alternative is to let all servers monitor each other. This approach brings the herd effect, in which upon server failure, traffic volume increases exponentially with regard to the number of servers.

- **Scale with document repository**: the way coordinator dispatches work is to assign each worker a subset of the document. By doing so the system can be easily scaled up with the organic growth of the document repository.
    - An alternative to assign a subset of terms from the query to each worker. But user queries are unlikely to vary compared to the document repository. So this approach is way less scalable.
- **Codebase separation**: the project has two separate codebases for backend server and frontend server. This is mainly because of the constraint that Spring Boot projects can only have one entry point. But on the other hand, doing so makes the codebase structure cleaner.
    - The downside of this approach is that some shared code, e.g. coordinator interface proto and service registry class, need to be duplicated and kept sync across two codebases.

Enhancements

Some additional enhancements the implementation has:

- **Cloud deployment**: this project is deployed on Google Cloud Platform for reliability and availability.
- **Multiple User Interfaces**: In addition to the Web UI provided by frontend servers, the project additionally implements a client that can be used to test backend servers by taking queries from command lines.
- **Language Neutral Interface**: this feature comes as free by using gRPC and protocol buffer, which provides a language-neutral interface definition. So in theory clients / frontend servers written in C++ can also talk to our backend servers which were written in Java.