Distributed Cache

Project Overview

Implement a simple distributed caching system using Go. This system will consist of two main services: a Cache Manager and a Data Node. These services will communicate with each other using a pub/sub mechanism and utilize Go channels for concurrent operations.

System Components

1. Cache Manager

The Cache Manager is responsible for:

- Handling client requests for data retrieval and storage
- Maintaining a mapping of keys to Data Node locations
- Coordinating with Data Nodes for data operations
- Managing cache invalidation

2. Data Node

The Data Node is responsible for:

- Storing and retrieving data
- Handling concurrent read/write operations
- Implementing a simple eviction policy (e.g. LRU)

Key Requirements

1. Use Go's standard library extensively. Avoid third-party libraries unless absolutely necessary.
2. Implement pub/sub communication between the Cache Manager and Data Nodes using a simple in-memory solution.
3. Utilize Go channels for handling concurrent operations within each service.
4. Implement basic error handling and logging.
5. Write unit tests for critical components.

Specific Tasks

1. Set up the basic structure for both services.
2. Implement the pub/sub mechanism for inter-service communication.
3. Create the Cache Manager's key-to-node mapping and request handling logic.
4. Develop the Data Node's storage mechanism with concurrent read/write support.
5. Implement a basic cache invalidation strategy.
6. Add a simple CLI or HTTP interface for interacting with the Cache Manager.
7. Write unit tests for key components.

Evaluation Criteria

- Correct use of Go's concurrency primitives (goroutines, channels)
- Effective use of the standard library
- Clean, well-organized, and documented code
- Proper error handling and logging
- Correct implementation of the pub/sub mechanism
- Ability to handle concurrent operations efficiently
- Quality and coverage of unit tests

Time Limit

You have 6 hours to complete this project. Focus on implementing the core functionality first, then add additional features or improvements if time permits.

Submission

If you make any assumptions add comments to your code describing them and include a paragraph about the choices and decisions you have made while implementing the application.

Please do not publish your code as a public repository. Instead, create a private repository and invite @akramhussein for collaboration. It is important to show your path to the solution, so please use `git` and commit code frequently, even if the code does not work.