Assignment 4 - Hazelcast

Implement a distributed application for providing documents to users, with features such as tracking view count, comments and favorite lists.

The application will consist of a cluster of servers responsible for all the data and a client application.

In order for the application to have very fast response, all data should be kept in memory on the server. For that, use Hazelcast IMDG.

Parts of the implementation are provided. Download source code from https://d3s.mff.cuni.cz/files/teaching/nswi080/labs/sources-4.zip

Prerequisites

- Basic setup of a Hazelcast cluster
- Knowledge of the Map distributed data structure in Hazelcast, its properties, interface and configuration

Your task

The client application must provide the following functionality and satisfy the following requirements:

- The server part of the application consists of one or more members of the Hazelcast cluster. The client application is launched with a specified user name. It shows a simple command prompt and performs the commands entered by the user.

- Implement a cache for documents that are viewed by the users. The client can request a document by the document name, using the S command.

  Suppose that the application would need to do some expensive computation to generate the document. For the purpose of this task, we only simulate the long computation by waiting a few seconds. There is no need to change this code.

  The documents should be generated on the cluster (not on the clients). The documents should be stored in a cache, so subsequent accesses to the same document (from any user) should be fast. However, assume that the documents may be large, so not all documents ever generated will fit into memory at the same time. It is ok if because of this, the document has to be generated again.

- For each user, remember the name of the last document that has been shown to them (we will call this the selected document). This value should be stored in the cluster, not on the clients - that means, it will be remembered even if the user quits the client application.

- For every document, keep the number of views (number of times it has been shown). This number should be exact (under normal operation). Users can view this number by first selecting the document, and the using a I command.

- For every document, keep a list of comments. Users can use the C command to enter a comment that will be added to the list of comments about the selected document. The I command should show the user the view count and all the comments attached to the selected document. All comments are visible to all users.
For every user, keep a list of names of their favorite documents. The user can add the name of the selected document to the list by the a command and remove it by the r command. The l command will show the names in the list of favorites.

The n command can be used to quickly show documents in the favorite list. It selects and shows the next (relative to the selected document) document in the list of favorites. Using this command repeatedly will cyclically show all the favorite documents. This command should have the same effects as the s command (putting the document into cache, increasing view count, storing the name of the selected document)

You may assume the number of comments and favorites is small and the cluster will have more than enough memory to store all comments, views counts, and user data.

- Configure the distributed maps that you used to store the data. **Compare** the different requirements that the application has on the maps in terms of reliability and access speed. Explain, why the default configuration might not be the best fit when using the map as a cache for the documents, choose a configuration that might be better and explain the benefits.

**Notes**

**Application structure**

- Use client/server topology
  - we can start any number of members of the cluster - horizontal scalability.
- Documents should be generated and all data should be stored on the servers, not on clients
- If one user quits and restarts the client, it should continue where left off.

**Data**

- Implement a cache for documents.
- Store data per document:
  - access count
  - comments
- Store data per user:
  - selected document name
  - list of favorite document names
- Use multiple distributed hash maps indexed by user names and document names
- Decide what values are stored in a map (string / list / custom class type)

**Configuration**

- Hazelcast allows configuring parameters of maps:
  - backups
  - eviction
  - data format
- Cache and data have different access patterns and different requirements.
- Look through the features of Map in the Hazelcast manual

**Scope**

To keep the assignment simple, you don't need to consider:

- Security
  - not available in the open-source edition of Hazelcast
- Persistent data storage
- JCache API
  - using a Map is sufficient
- Deploying user code
  - you can use the same classpath for both servers and clients

**Advice**

Avoid race condition

- naively running `viewCountMap.put(documentName, viewCountMap.get(documentName) + 1)` on the client is not correct
  - if two clients do it at the same time, one increment may be lost

**Hazelcast manual**

The following sections may be relevant:

- 3 Overview
- 7.2 Map
- 18.1 Java client
- 10.4 Entry Processor

**Submission**

- By e-mail (deadline is on the web)
- Documentation
  - **Reasoning** about the chosen features and configuration
- The submission shall be easy to start
- Make sure it works in the lab
- Do not send any generated files (but send the build script)