CS 549—Distributed Systems and Cloud Computing—Fall 2024
Assignment Two—Distributed Hash Tables

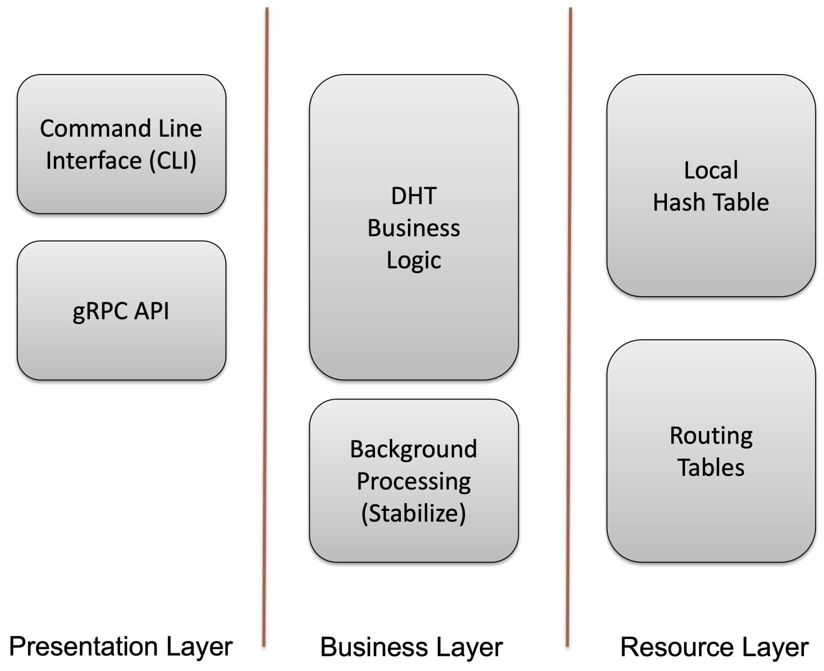
For this assignment, you will complete a simple peer-to-peer structured overlay network, a distributed hash table (DHT), to be deployed into the cloud. You are provided with the partial implementation of a DHT using gRPC for communication among nodes. Your assignment is to complete this implementation. The nodes of your network will run as standalone Java programs, using an embedded gRPC server stack.

Each node has its own "state." This is an in-process singleton object. When an application starts, it creates a state object. Concurrent Web requests synchronize their accesses to the node's state using locks in the state object.

The code is structured as a Maven project called "`dht-grpc`" with several packages:

1. `edu.stevens.cs549.dht.activity` contains business logic for the overlay network.
2. `edu.stevens.cs549.dht.main` contains the main program, as well as a CLI and a class encapsulating client requests.
3. `edu.stevens.cs549.dht.rpc` contains resources for the gRPC API.
4. `edu.stevens.cs549.dht.state` contains the state implementation.

The overall structure of the application is described by this picture:



The POM file for the main project defines several properties that are used in the project. You do not need to learn Maven or POM files, and in this case you should not need to modify this root POM.

If you build the jar file for the project (using the Maven tool window in Intellij), it is placed in

```
~/tmp/cs549/dht-test/dht.jar
```

where ~ represents your home directory. To run the program, type

```
java –jar dht.jar [--host ip-address] --http http-port --id id
```

The optional parameter `host` should be the private IP address of the EC2 instance this runs on; it defaults to `localhost`, which should be sufficient. The parameter `http` is the port number for the Web server that runs as part of the app, while `id` is the identity that this node takes on in the DHT. This will start a command line interface once it has started the gRPC server. Type "`help`" for information about the CLI commands. In contrast to the previous assignment, the application you develop is both a client and a server, each with its own CLI. You run multiple instances of the application, and they communicate with each other based on the commands you provide in the CLI. The more instances you run, the better. It should even be possible for instances belonging to different students in the class to interoperate.

To incorporate the code base into Intellij IDEA, be sure to load it as a Maven project, as you did with the previous assignment. The project includes the use of a protobuf Maven plugin that automatically downloads and runs the protoc compiler to generate Java source from the protobuf source code in the folder `src/main/proto`. The generated source is then imported by the program code. Once you have your code running locally, you should test it on several EC2 instances, each instance running a node in the DHT overlay network.

To complete the assignment, you need to complete the DHT logic, and you need to complete the gPRC API. Your application should provide the following API:

```
service DHT {

  rpc getNodeInfo (google.protobuf.Empty) returns (NodeInfo);

  rpc getPred (google.protobuf.Empty) returns (OptNodeInfo);*

  rpc getSucc (google.protobuf.Empty) returns (NodeInfo);

  rpc closestPrecedingFinger (Id) returns (NodeInfo);

  rpc notify (NodeBindings) returns (OptNodeBindings);*

  rpc getBindings (Key) returns (Bindings);

  rpc addBinding (Binding) returns (google.protobuf.Empty);

  rpc deleteBinding (Binding) returns (google.protobuf.Empty);

  rpc findSuccessor (Id) returns (NodeInfo);

}
```

where the service interface uses these message type definitions:

```
message Id {
  int32 id = 1;
}
```

```
message Key {
  string key = 1;
}

message NodeInfo {
  int32 id = 1;
  string addr = 2;
  int32 port = 3;
}

message OptNodeInfo {
  optional NodeInfo nodeInfo = 1;
}

message Binding {
  string k = 1;
  string value = 2;
}

message Bindings {
  string key = 1;
  repeated string value = 2;
}

message NodeBindings {
  NodeInfo info = 1;
  NodeInfo succ = 2;
  repeated Bindings bindings = 3;
}

message OptNodeBindings {
  optional NodeBindings nodeBindings = 1;
}
```

The semantics of these operations are described in the lectures on P2P networks, particularly for the Chord system.

Submission
Once you have your code working, please follow these instructions for submitting your assignment:

- Create a zip archive file, named after you, containing a directory with your name. E.g. if your name is Humphrey Bogart, then name the directory `Humphrey_Bogart`.
- In that directory you should provide the zip archive that contains your Intellij IDEA project, and the server jar file, `dht.jar`.
- Also include a completed copy of the rubric, recording what you have accomplished for the assignment**.**

Record one or more short videos demonstrating your code working. Make sure that your name appears at the beginning of the video. For example, display the contents of a file that provides your name. *Do not provide private information such as your email or cwid in the video.* It is very important for your grade that you do adequate testing. You should at a minimum demonstrate testing of these scenarios:

1. Joining *at least three nodes* together into a DHT network. Run these nodes on separate EC2 instances under your control.
2. Adding, retrieving and deleting bindings, at both local and remote nodes (with respect to the CLI of the instance where you are testing). Use the `bindings` and `routes` commands of the CLI to show the functioning of your network, including the states of the nodes after you perform operations on the network.
3. For example, test with the following bindings, with nodes with keys 23, 37 and 45:
```
foo -> 6
foo2 -> 44
foo3 -> 44
bar -> 19
baz -> 27
baz2 -> 55
fubar -> 36
xyz -> 25
```

Finally note any other changes you made to the materials provided, with an explanation.

Remember the format of the submission: A zip archive file, named after you, with a directory named after you. In this directory, provide three files: a zip archive of your source files and resources, a server jar file, and a completed rubric. Be sure to include videos demonstrating your app working, covering the scenarios described above