Assigenment-2: DHT Standalone – Documentation Report

Dhruvit Patel (CWID: 10404032)

Abstract

In this Documentation, I demonstrated that what I have done to make peer-to-peer system in working implementation. I mentioned each implementation (REST API and the business logic for the DHT) with code snippet

In the second portion of Documentation, I tested the working abilities of DHT System with three locally created nodes and also in Amazon EC2 Web Instances. Here, I mentioned the command to set up Distribute Hash Table system.

Snapshots for DHT.java

Snapshot 1

Placing a call web service call to the closest preceding finger in the finger table. The call is placed with the help of client object. And this function returns the NodeInfo of the respective preceding node, and it take the NodeInfo and id of current node as its parameters to determine its closest node.

DHT.java | WebClient.java | NodeResource.java | NodeService.java | State.java

```java

    /*
     * Perform a Web service call to get the closest preceding finger in the
     * finger table of the argument node.
     */
    protected NodeInfo closestPrecedingFinger(NodeInfo info, int id) throws Failed {
        NodeInfo localInfo = this.getNodeInfo();
        if (localInfo.equals(info)) {
            return closestPrecedingFinger(id);
        } else {
            if (IRouting.USE_FINGER_TABLE) {
                /*
                 * TODO: Do the Web service call to the remote node. Add by Dhruvit
                 */
                return client.closestPrecedingFinger(info,id);
            } else {
                /*
                 * Without finger tables, just use the successor pointer.
                 */
                return getSucc(info);
            }
        }
    }

```

Snapshot 2

This webservice call would return the NodeInfo of the predecessor node of the node calling the method.

DHT.java | WebClient.java | NodeResource.java | NodeService.java | State.java

```java
     * Make a special case for when this is the local node, i.e.,
     * info.addr.equals(localInfo.addr), otherwise get an infinite loop.
     */
    protected NodeInfo getPred(NodeInfo info) throws Failed {
        NodeInfo localInfo = this.getNodeInfo();
        if (localInfo.addr.equals(info.addr)) {
            return getPred();
        } else {
            /*
             * TODO: Do the Web service call. Add by Dhruvit
             */
            return client.getPred(info);
        }
    }

```

Snapshot 3

This webservice call would return the NodeInfo of the successor node of the node calling the method.

DHT.java | WebClient.java | NodeResource.java | NodeService.java | State.java

```java
     * finger table of the argument node.
     */
    protected NodeInfo closestPrecedingFinger(NodeInfo info, int id) throws Failed {
        NodeInfo localInfo = this.getNodeInfo();
        if (localInfo.equals(info)) {
            return closestPrecedingFinger(id);
        } else {
            if (IRouting.USE_FINGER_TABLE) {
                /*
                 * TODO: Do the Web service call to the remote node. Add by Dhruvit
                 */
                return client.closestPrecedingFinger(info,id);
            } else {
                /*
                 * Without finger tables, just use the successor pointer.
                 */
                return getSucc(info);
            }
        }
    }
```

Snapshot 4

This webservice call would return the NodeInfo of the node calling the get(n,k) method.

DHT.java | WebClient.java | NodeResource.java | NodeService.java | State.java

```java

    /*
     * Get the values under a key at the specified node. If the node is the
     * current one, go to the local state.
     */
    protected String[] get(NodeInfo n, String k) throws Failed {
        if (n.addr.equals(info.addr)) {
            try {
                return this.get(k);
            } catch (Invalid e) {
                severe("Get: invalid internal inputs: " + e);
                throw new IllegalArgumentException(e);
            }
        } else {
            /*
             * Retrieve the bindings at the specified node.
             *
             * TODO: Do the Web service call.
             */
            return client.get(n,k);

        }
    }

    /*
     * Retrieve values under the key at the current node.
     */
```

Snapshot 5

This webservice call would delete the value of the Node calling the delete(n,k,v) method, here n is the nodeinfo, k is the key and v is the value assigned to that key, which would be deleted.

DHT.java | WebClient.java | NodeResource.java | NodeService.java | State.java

```java
    /*
     * Delete value under a key.
     */
    public void delete(NodeInfo n, String k, String v) throws Failed {
        if (n.addr.equals(info.addr)) {
            try {
                delete(k, v);
            } catch (Invalid e) {
                severe("Delete: invalid internal inputs: " + e);
                throw new IllegalArgumentException(e);
            }
        } else {
            /*
             * TODO: Do the Web service call.
             */
            client.delete(n,k,v);

        }
    }

```

Snapshot6

This webservice call would add the Node to the finger table using the add(n,k,v) method.

DHT.java | WebClient.java | NodeResource.java | NodeService.java | State.java

```java
    }

    /*
     * Add a value under a key.
     */
    public void add(NodeInfo n, String k, String v) throws Failed {
        if (n.addr.equals(info.addr)) {
            try {
                add(k, v);
            } catch (Invalid e) {
                severe("Add: invalid internal inputs: " + e);
                throw new IllegalArgumentException(e);
            }
        } else {
            /*
             * TODO: Do the Web service call.
             */
            client.add(n, k, v);
        }
    }

```

Snapshots for WebClient.java

Snapshot6

The below mentioned method of putRequest is designed in a way to fetch the details of the node entered under the add command. This method is automatically called when the ADD command is executed, this method takes the path of the key entered as the parameter.

```java

    private Response putRequest(URI uri, Entity<?> entity) {
        // TODO
        try {
            Response cr = client.target(uri)
                    .request(MediaType.APPLICATION_XML_TYPE)
                    .header(Time.TIME_STAMP, Time.advanceTime())
                    .put(entity);
            processResponseTimestamp(cr);
            return cr;
        } catch (Exception e) {
            error("Exception during PUT request: " + e);
            return null;
        }


    }

```

Snapshot7

The below mentioned method of delRequest is designed in a way to fetch the details of the node entered under the delete command. This method is automatically called when the delete command is executed, this method takes the path of the key entered as the parameter i.e. URI.

DHT.java | *WebClient.java | NodeService.java | State.java

```java
    }

    private Response delRequest(URI deletePath) {
        // TODO Auto-generated method stub
        try {
            Response cr = client.target(deletePath)
                    .request(MediaType.APPLICATION_XML_TYPE)
                    .header(Time.TIME_STAMP, Time.advanceTime())
                    .delete();
            processResponseTimestamp(cr);
            return cr;
        } catch (Exception e) {
            error("Exception during DELETE request: " + e);
            return null;
        }
    }

```

DHT.java | *WebClient.java | NodeService.java | State.java

```java

    public void add(NodeInfo n, String k, String v) throws Failed {
        // TODO Auto-generated method stub: Added by dhruvit Patel
        UriBuilder ub = UriBuilder.fromUri(n.addr).path("add");
        URI addPath = ub.queryParam("key", k).queryParam("value", v).build();
        TableRep tablerep = new TableRep(null, null, 1);
        tablerep.entry[0] = new TableRow(k, new String[]{v});
        info("client add(" + addPath + ")");
        Response response = putRequest(addPath, Entity.xml(tablerep));
        if (response == null || response.getStatus() >= 300) {
            throw new DHTBase.Failed("PUT /add?id=ID");
        }

    }

    public void delete(NodeInfo n, String k, String v) throws Failed {
        // TODO Auto-generated method stub: Added by dhruvit Patel
        UriBuilder ub = UriBuilder.fromUri(n.addr).path("delete");
        URI deletePath = ub.queryParam("key", k).queryParam("value", v).build();
        info("client delete(" + deletePath + ")");
        Response response = delRequest(deletePath);
        if (response == null || response.getStatus() >= 300) {
            throw new DHTBase.Failed("DELETE /delete?id=ID");
        }

    }

```

Snapshots for NodeService.java

Snapshot8

The below generated methods-: getSucc() is for getiing the next node in the finger table, findClosestPrecedingFinder(id)-: it is for returning the closest preceeding node in respect to the id mentioned in the parameter, getValue(key) -: Determines the value assigned to the respective key mentioned in parameter, add(key,value)-: This method takes two arguments, one is the key and assigns a respective value to it.

DHT.java | *WebClient.java | *NodeService.java | State.java

```java
    public Response getSucc() {
        // TODO Auto-generated method stub
        advanceTime();
        info("getSucc()");
        return response(dht.getSucc());
    }

    public Response findClosestPrecedingFinger(int id) {
        // TODO Auto-generated method stub
        advanceTime();
        info("findClosestPrecedingFinger()");
        return response(dht.closestPrecedingFinger(id));
    }

    public Response getValue(String key) throws Invalid {
        // TODO Auto-generated method stub
            advanceTime();
            info("getValue()");
            return response(new TableRow(key, dht.get(key)));
    }

    public Response add(String key, String value) throws Invalid {
        // TODO Auto-generated method stub
            advanceTime();
            info("add()");
            dht.add(key, value);
            return response();
    }
```

```java
    public Response delete(String key, String value) throws Invalid {
        // TODO Auto-generated method stub
            advanceTime();
            info("delete()");
            dht.delete(key, value);
            return response();
    }

}
```

Snapshots for State.java

Snapshot 9-: Setting and getting of finger respectively, and assigning node information to it.

DHT.java | *WebClient.java | *NodeService.java | State.java

```java
    }

    public synchronized void setFinger(int i, NodeInfo info) {
        /*
         * TODO: Set the ith finger.
         */
        finger[i] = info;
    }

    public synchronized NodeInfo getFinger(int i) {
        //return backupSucc;
        /*
         * TODO: Get the ith finger.
         */
        return finger[i];
    }
```

Snapshot 10-: Getting the closest Preceding Finger in respect to the current mentioned in the parameter of the method and returning its respective NodeInfo.

DHT.java | *WebClient.java | *NodeService.java | *State.java

```java

    public synchronized NodeInfo closestPrecedingFinger(int id) {
        //return backupSucc;
        /*
         * TODO: Get closest preceding finger for id, to continue search at that
         * node. Hint: See DHTBase.inInterval()
         */

        int RoutingFingers = IRouting.NFINGERS;

        while (RoutingFingers > 0) {

            int RoutingKeys = IRouting.NKEYS;
            int Upper = finger[RoutingFingers].id % RoutingKeys;
            int Lower = finger[RoutingFingers - 1].id % RoutingKeys;

            if (id == Upper) {
                return getFinger(RoutingFingers -1);
            }

            if (Lower < Upper) {
                Upper = (Upper - Lower);
                Lower = (id - Lower);
                if (id > 0 && Upper > id) {
                    return getFinger(RoutingFingers - 1);
                }
            } else if (Lower > Upper) {
                Upper = (Upper + RoutingKeys - Lower) % RoutingKeys;
```

Snapshots for NodeResource.java

Snapshot 11-: Determining the annotations and adding responsive methods for getting the successor of the node, finding the closest preceding finger, getting value, adding key and value, deleting the value for that key. This methods determine the response.

DHT.java | *WebClient.java | *NodeService.jav | *State.java | *NodeResource.ja

```java
    @GET
    @Path("succ")
    @Produces("application/xml")
    public Response getSucc() {
        return new NodeService(headers, uriInfo).getSucc();
    }

    @GET
    @Path("findClosestPrecedingFinger")
    @Produces("application/xml")
    public Response findClosestPrecedingFinger(@QueryParam("id") String index) {
        int id = Integer.parseInt(index);
        return new NodeService(headers, uriInfo).findClosestPrecedingFinger(id);
    }

    @GET
    @Path("getValue")
    @Produces("application/xml")
    public Response getValue(@QueryParam("key") String key) throws Invalid {
        return new NodeService(headers, uriInfo).getValue(key);
    }

    @PUT
    @Path("add")
    @Consumes("application/xml")
    public Response add(TableRep tablerep) throws Invalid {
        return new NodeService(headers, uriInfo).add(tablerep.entry[0].key, tablerep.entry
    }
```

Commands and Utilization

Add: - We use this command to create key pair value in the chord system. We can add more values to the variable.

- add foo first
- add foo2 second
- add foo2 third

Get: - To get the value of variable.

- get foo
- ➢ {first}
- get foo2
- ➢ {second, third}

Del: - Delete the value of a variable using the Del command.

- del foo2 third
- get foo2
- ➢ {second}

Bindings: - With the help of bindings, we check all the variables created by us and also check the values which are bindings with different variables.

Join: - With the join command, new node join to the already established chord ring, after then it will access the share data.

Routes: - We can view the finger table using routes. We can also check the successor and predecessor for the current node.

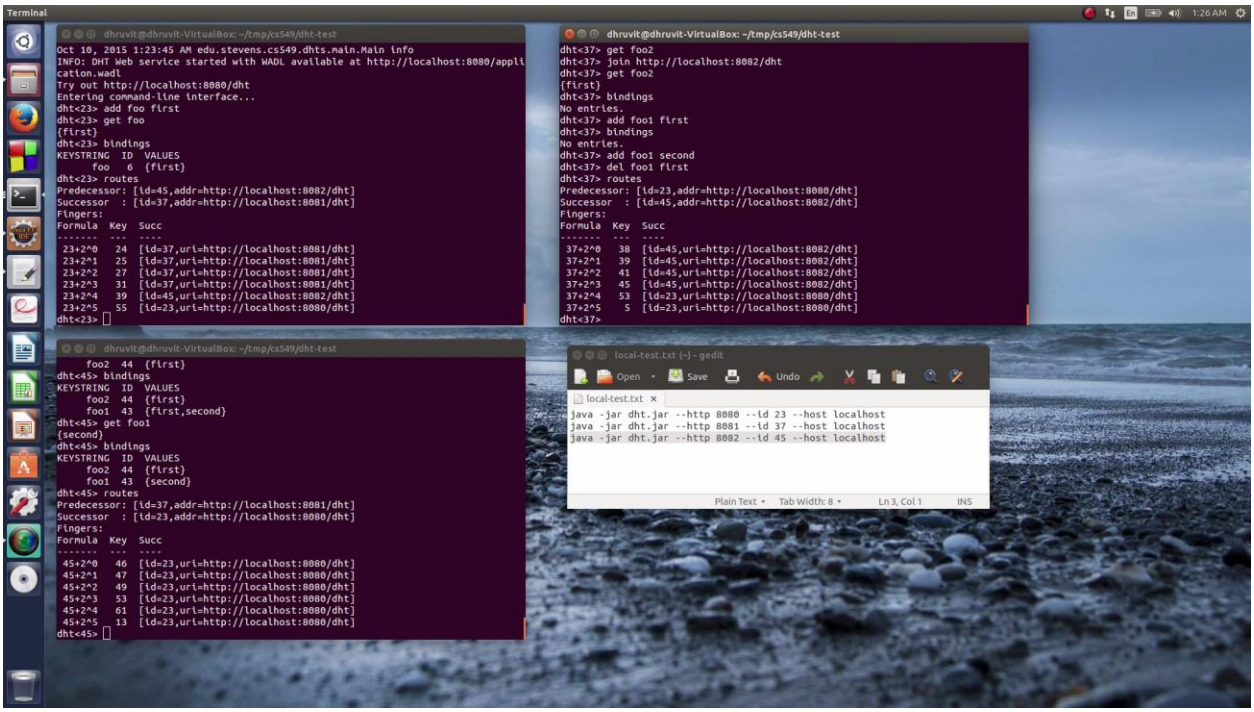
Testing

❖ Local

We use following commands to set up peer to peer network between three nodes in the system.

- java -jar dht.jar --http 8080 --id 23 --host localhost
- java -jar dht.jar --http 8081 --id 45 --host localhost
- java -jar dht.jar --http 8082 --id 37 --host localhost

I created three nodes 23, 37 and 45.



Finger table of three nodes with Predecessor and Successor.

Nodes	Predecessor	Successor
23	45	37
37	23	45
45	37	23

I used following command to test working abilities of chord ring.

- Add
- Get
- Bindings
- Del
- Routes

❖ Remote

Following command are used for login into Amazon EC2 Console of three different instances.

1. sudo ssh -i ~/AWS_EC2/dhruv_aws_ec2.pem ubuntu@ec2-54-68-9-52.us-west-2.compute.amazonaws.com
2. sudo ssh -i ~/AWS_EC2/dhruv_aws_ec2.pem ubuntu@ec2-52-89-81-227.us-west-2.compute.amazonaws.com
3. sudo ssh -i ~/AWS_EC2/dhruv_aws_ec2.pem ubuntu@ec2-54-69-96-171.us-west-2.compute.amazonaws.com

Following command are used for uploading dht.jar file into Amazon EC2 Instances.

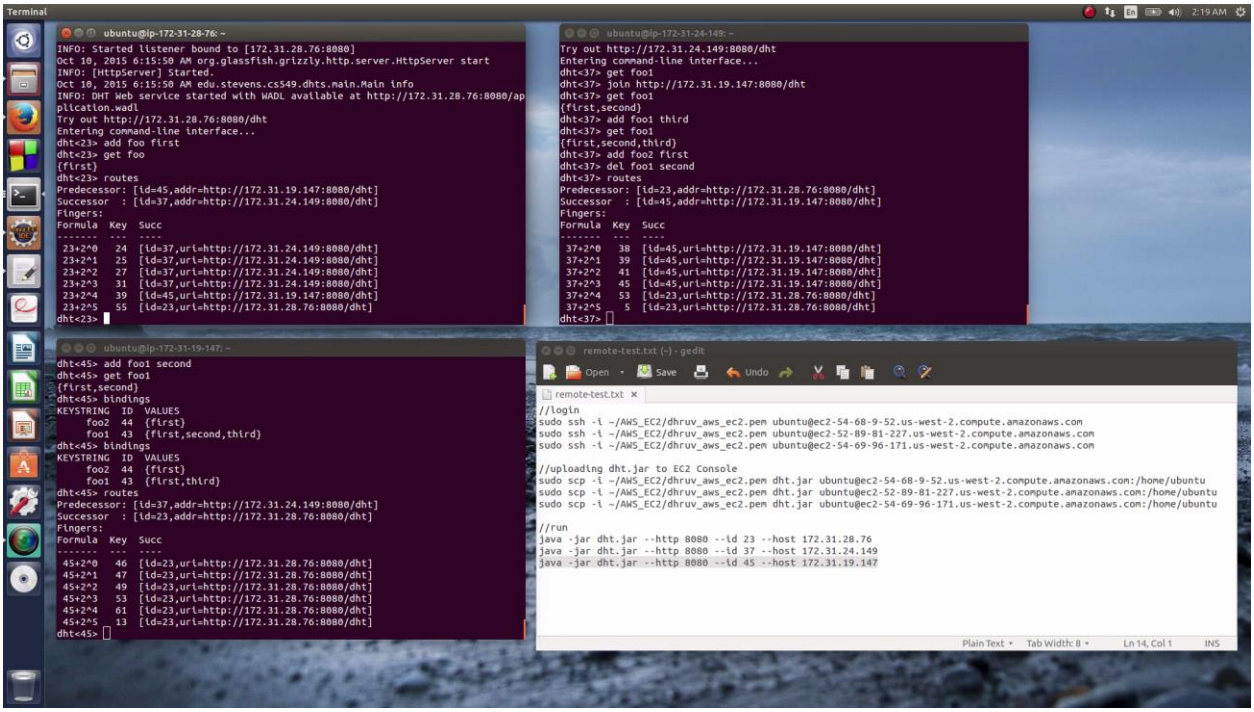
1. sudo scp -i ~/AWS_EC2/dhruv_aws_ec2.pem dht.jar ubuntu@ec2-54-68-9-52.us-west-2.compute.amazonaws.com:/home/ubuntu
2. sudo scp -i ~/AWS_EC2/dhruv_aws_ec2.pem dht.jar ubuntu@ec2-52-89-81-227.us-west-2.compute.amazonaws.com:/home/ubuntu
3. sudo scp -i ~/AWS_EC2/dhruv_aws_ec2.pem dht.jar ubuntu@ec2-54-69-96-171.us-west-2.compute.amazonaws.com:/home/ubuntu

Following command are used for set up peer-to-peer network between three Amazon EC2 Instances.

1. java -jar dht.jar --http 8080 --id 23 --host 172.31.28.76
2. java -jar dht.jar --http 8080 --id 37 --host 172.31.24.149
3. java -jar dht.jar --http 8080 --id 45 --host 172.31.19.147

Finger table of three nodes with Predecessor and Successor.

Nodes	Predecessor	Successor
23	45	37
37	23	45
45	37	23


Terminal
ubuntu@ip-172-31-28-76: ~
INFO: Started listener bound to [172.31.28.76:8080]
Oct 10, 2015 6:15:50 AM org.glassfish.grizzly.http.server.HttpServer start
INFO: [HttpServer] Started.
Oct 10, 2015 6:15:50 AM edu.stevens.cs549.dhts.main.Main info
INFO: DHT Web service started with WADL available at http://172.31.28.76:8080/application.wadl
Try out http://172.31.28.76:8080/dht
Entering command-line interface...
dht<23> add foo first
dht<23> get foo
{first}
dht<23> routes
Predecessor: [id=45,addr=http://172.31.19.147:8080/dht]
Successor : [id=37,addr=http://172.31.24.149:8080/dht]
Fingers:
Formula Key Succ
23+2^0 24 [id=37,uri=http://172.31.24.149:8080/dht]
23+2^1 25 [id=37,uri=http://172.31.24.149:8080/dht]
23+2^2 27 [id=37,uri=http://172.31.24.149:8080/dht]
23+2^3 31 [id=37,uri=http://172.31.24.149:8080/dht]
23+2^4 39 [id=45,uri=http://172.31.19.147:8080/dht]
23+2^5 55 [id=23,uri=http://172.31.28.76:8080/dht]
dht<23>
ubuntu@ip-172-31-24-149: ~
Try out http://172.31.24.149:8080/dht
Entering command-line interface...
dht<37> get foo1
dht<37> join http://172.31.19.147:8080/dht
dht<37> get foo1
{first,second}
dht<37> add foo1 third
dht<37> get foo1
{first,second,third}
dht<37> add foo2 first
dht<37> del foo1 second
dht<37> routes
Predecessor: [id=23,addr=http://172.31.28.76:8080/dht]
Successor : [id=45,addr=http://172.31.19.147:8080/dht]
Fingers:
Formula Key Succ
37+2^0 38 [id=45,uri=http://172.31.19.147:8080/dht]
37+2^1 39 [id=45,uri=http://172.31.19.147:8080/dht]
37+2^2 41 [id=45,uri=http://172.31.19.147:8080/dht]
37+2^3 45 [id=45,uri=http://172.31.19.147:8080/dht]
37+2^4 53 [id=23,uri=http://172.31.28.76:8080/dht]
37+2^5 5 [id=23,uri=http://172.31.28.76:8080/dht]
dht<37>
ubuntu@ip-172-31-19-147: ~
dht<45> add foo1 second
dht<45> get foo1
{first,second}
dht<45> bindings
KEYSTRING ID VALUES
foo2 44 {first}
foo1 43 {first,second,third}
dht<45> bindings
KEYSTRING ID VALUES
foo2 44 {first}
foo1 43 {first,third}
dht<45> routes
Predecessor: [id=37,addr=http://172.31.24.149:8080/dht]
Successor : [id=23,addr=http://172.31.28.76:8080/dht]
Fingers:
Formula Key Succ
45+2^0 46 [id=23,uri=http://172.31.28.76:8080/dht]
45+2^1 47 [id=23,uri=http://172.31.28.76:8080/dht]
45+2^2 49 [id=23,uri=http://172.31.28.76:8080/dht]
45+2^3 53 [id=23,uri=http://172.31.28.76:8080/dht]
45+2^4 61 [id=23,uri=http://172.31.28.76:8080/dht]
45+2^5 13 [id=23,uri=http://172.31.28.76:8080/dht]
dht<45>
remote-test.txt (~) - gedit
Open
Save
Undo
remote-test.txt
//login
sudo ssh -i ~/AWS_EC2/dhruv_aws_ec2.pem ubuntu@ec2-54-68-9-52.us-west-2.compute.amazonaws.com
sudo ssh -i ~/AWS_EC2/dhruv_aws_ec2.pem ubuntu@ec2-52-89-81-227.us-west-2.compute.amazonaws.com
sudo ssh -i ~/AWS_EC2/dhruv_aws_ec2.pem ubuntu@ec2-54-69-96-171.us-west-2.compute.amazonaws.com
//uploading dht.jar to EC2 Console
sudo scp -i ~/AWS_EC2/dhruv_aws_ec2.pem dht.jar ubuntu@ec2-54-68-9-52.us-west-2.compute.amazonaws.com:/home/ubuntu
sudo scp -i ~/AWS_EC2/dhruv_aws_ec2.pem dht.jar ubuntu@ec2-52-89-81-227.us-west-2.compute.amazonaws.com:/home/ubuntu
sudo scp -i ~/AWS_EC2/dhruv_aws_ec2.pem dht.jar ubuntu@ec2-54-69-96-171.us-west-2.compute.amazonaws.com:/home/ubuntu
//run
java -jar dht.jar --http 8080 --id 23 --host 172.31.28.76
java -jar dht.jar --http 8080 --id 37 --host 172.31.24.149
java -jar dht.jar --http 8080 --id 45 --host 172.31.19.147
Plain Text
Tab Width: 8
Ln 14, Col 1
INS