Report on RESTful DHT

CODE PART:-

package edu.stevens.cs549.dhts.activity:-

In DHT.java:-

In getSucc():-

```
private NodeInfo getSucc(NodeInfo info) throws Failed
{
    NodeInfo localInfo = this.getNodeInfo();
    if (localInfo.addr.equals(info.addr))
    {
            return getSucc();
    }
    else
    {
            localInfo = this.client.getSucc(info);
    }
    return localInfo;
}
```

It is basically retrieving the data from the routing logic where the Successor is stored.

In getPred():-

```
protected NodeInfo getPred(NodeInfo info) throws Failed
{
    NodeInfo localInfo = this.getNodeInfo();
    if (localInfo.addr.equals(info.addr))
    {
            return getPred();
    }
    else
    {
            localInfo = this.client.getPred(info);
    }
    return localInfo;
}
```

It is basically retrieving the data from the routing logic where the Predecessor is stored.

In closestPrecedingFinger():-

```
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)
            {
                    return this.client.findClosestPrecedeFinger(info.addr, id);
            }
            else
            {
                    return getSucc(info);
            }
    }
}
```

To lookup in the Finger table for Internal Operation.

Added closestSuccessingFinger():-

```
public NodeInfo closestSuccessingFinger(int id)
{
    return routing.closestSuccessingFinger(id);
}
```

To lookup in the Finger table for Internal Operation for finding the Successing Finger.

Added findSuccessor():-

```
private NodeInfo findSuccessor(NodeInfo closeNode, int id) throws Failed
{
    NodeInfo pred = getPred(closeNode);
    if(pred.id == id)
    {
            return closeNode;
    }
    else
    {
            return findSuccessor(pred, id);
    }
}
```

With the help of this method we can find the successor of the Node from the routing table.

Added checkSuccessor():-

```
public void checkSuccessor()
{
    NodeInfo succ = getSucc();
    boolean failed = this.client.isFailed(succ.addr);
    if(failed)
    {
        info("Begin to fix the drop node " + succ);
        NodeInfo closestNode = closestSuccessingFinger(succ.id);
        try
        {
            if(closestNode.id == succ.id)
            {
                setSucc(info);
                setPred(null);
                this.stabilize();
                return;
            }
            closestNode = findSuccessor(closestNode, succ.id);
        }
        catch (Failed e)
        {
        info("checkSuccessor: Closest Node has failed (id=" + closestNode.id + ")");
        closestNode = null;
        }
        if(closestNode != null)
        {
            setSucc(closestNode);
            try
            {
            TableRep db = this.client.notify(closestNode, state.extractBindings());
            notifyContinue(db);
            }
            catch (Failed e)
            info("checkSuccessor: notify failed (id=" + closestNode.id +")");
        }
    }
}
```

The checkSuccessor method would help to find out the successor and attain the node information which would set it as the successor of a particular node.

In get():-

```
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
    {
            return this.client.getKey(n.addr, k);
    }
}
```

Get the key value at the specified node. If this is the current node then go to the local state.

In add():-

```
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
            this.client.putKey(n.addr, k, v);
    }
}
```

Add function helps in adding a value to a key.

In delete():-

```
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
    {
            this.client.deleteKey(n.addr, k, v);
    }
}
```

To the delete a value under a key.

In join():-

```
public void join(String uri) throws Failed, Invalid
{
    setPred(null);
    NodeInfo info = getNodeInfo();
    NodeInfo succ;
    this.state.clear();//clear Bindings
    int id = info.id; // find nodeId here by uri
    URI targetUri =  UriBuilder.fromUri("http://" + uri).path("dht").build();
    succ = this.client.findSuccessor(targetUri, id);
    setSucc(succ);  // set current succ
    this.stabilize();// notify succ that I'm yours predecessor
}
```

Clear any local bindings first of all, to maintain consistency of the ring
We start then with the bindings that are transferred from the new successor. Then perform a stabilize().

Added getFinger():-

```
public NodeInfo getFinger(int index)
{
    return this.state.getFinger(index);
}
```

Add node information in the finger table.

In IDHTBackground.java:-

Added checkSuccessor():-

public void checkSuccessor() throws Error;

Check Successor function to check the check if there is a successor or not to the Node.

In IDHTResource.java:-

Added getFinger():-

public NodeInfo getFinger(int index);

Lookup in the Finger table for a particular value

package edu.stevens.cs549.dhts.main:-

In WebClient.java:-

Creation of Client Instance

```
protected Client client;
public WebClient()
{
	client = ClientBuilder.newClient();
}
```

Added putRequest():-

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

We notify the node of a new predecessor node, specified in the input element

Added deleteRequest():-

```
private Response deleteRequest(URI uri)
{
```

```
try
{
        Response cr = client.target(uri)
                        .request()
                        .header(Time.TIME_STAMP, Time.advanceTime())
                        .delete();
        processResponseTimestamp(cr);
        return cr;
        }
        catch (Exception e)
         {
                error("Exception during POST request: " + e);
                return null;
        }
}
```

We can delete the key value of the Key with the help of this function.

Added getSucc():-

```
public NodeInfo getSucc(NodeInfo node) throws DHTBase.Failed
{
    URI succPath = UriBuilder.fromUri(node.addr).path("succ").build();
    info("client getSucc(" + succPath + ")");
    Response response = getRequest(succPath);
    if (response == null || response.getStatus() >= 300)
    {
            throw new DHTBase.Failed("GET /succ");
    }
    else
    {
            NodeInfo succ = response.readEntity(nodeInfoType).getValue();
            return succ;
    }
}
```

It is basically going to the routing logic where the Successor is stored.

Added getFinger():-

```
public NodeInfo getFinger(URI uri, int index) throws DHTBase.Failed
{
    URI fingerPath = UriBuilder.fromUri(uri).path("finger").queryParam("index",
    index).build();
    info("client getFinger(" + fingerPath + ")");
    Response response = getRequest(fingerPath);
    if (response == null || response.getStatus() >= 300)
    {
            throw new DHTBase.Failed("GET /finger");
    }
    else
```

```
    {
            NodeInfo succ = response.readEntity(nodeInfoType).getValue();
            return succ;
    }
    }
```

Get the finger value from the finger table.

Added findClosestPrecedeFinger():-

```
public NodeInfo findClosestPrecedeFinger(URI addr, int id) throws Failed
{
    URI succOfIdURI = UriBuilder.fromUri(addr).path("closestfinger").queryParam("id",
    id).build();
    info("client findClosestPrecedeFinger(" + succOfIdURI + ")");
    Response response = getRequest(succOfIdURI);
    if (response == null || response.getStatus() >= 300)
    {
            throw new DHTBase.Failed("GET /pred");
    }
    else
    {
            NodeInfo precedeFinger = response.readEntity(nodeInfoType).getValue();
            return precedeFinger;
    }
}
```

To lookup in the Finger table for internal operation.

Added findSuccessor():-

```
public NodeInfo findSuccessor(URI addr, int id) throws Failed
{
    URI succOfIdURI = UriBuilder.fromUri(addr).path("find").queryParam("id", id).build();
    info("client getPred(" + succOfIdURI + ")");
    Response response = getRequest(succOfIdURI);
    if (response == null || response.getStatus() >= 300)
    {
            throw new DHTBase.Failed("GET /pred");
    }
    else
    {
            NodeInfo successor = response.readEntity(nodeInfoType).getValue();
            return successor;
    }
}
```

With this operation you could find the successor in the dht.

Added getKey():-

```
public String[] getKey(URI addr, String key) throws Failed
{
    URI succOfIdURI = UriBuilder.fromUri(addr).queryParam("key", key).build();
    info("client getKey(" + succOfIdURI + ")");
    Response response = getRequest(succOfIdURI);
    if (response == null || response.getStatus() >= 300)
    {
            throw new DHTBase.Failed("GET /getKey");
    }
    else
    {
            String[] values = response.readEntity(stringArray).getValue();
            return values;
    }
}
```

This operation only happens locally and does not involve searching the network. It basically retrieves the bindings for a key at a node.

Added putKey():-

```
public void putKey(URI addr, String key, String value) throws Failed

{

    URI succOfIdURI = UriBuilder.fromUri(addr).queryParam("key",
    key).queryParam("value", value).build();
    info("client putKey(" + succOfIdURI + ")");
    Response response = putRequest(succOfIdURI);
    if (response == null || response.getStatus() >= 300)
    {
            throw new DHTBase.Failed("GET /putKey");
    }
}
```

Updating a binding for a key at a node.

Added deleteKey():-

```
public void deleteKey(URI addr, String key, String value) throws Failed
{
    URI succOfIdURI = UriBuilder.fromUri(addr).queryParam("key",
    key).queryParam("value", value).build();
    info("client deleteKey(" + succOfIdURI + ")");
    Response response = deleteRequest(succOfIdURI);
    if (response == null || response.getStatus() >= 300)
    {
            throw new DHTBase.Failed("GET /putKey");
    }
}
```

Deleting the value under the key.

package edu.stevens.cs549.dhts.resource:-

In NodeResource.java:-

Operation for the Predecessor.

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

Operation for the Successor.

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

Operation for adding key value in the finger table.

```
@GET
@Path("finger")
@Produces("application/xml")
public Response getFinger(@QueryParam("index") int index)
{
    return new NodeService(headers, uriInfo).getFinger(index);
}
```

Operation for getting the key value.

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

Operation for putting the key values for a specified ID.

```
@PUT
@Produces("application/xml")
public Response putKeyBinds(@QueryParam("key") String key, @QueryParam("value")
String value) throws Invalid
{
    return new NodeService(headers, uriInfo).put(key, value);
}
```

Operation for deleting the key value for a specified ID.

```
@DELETE
@Produces("application/xml")
public Response deleteKeyBinds(@QueryParam("key") String key, @QueryParam("value")
String value) throws Invalid
{
    return new NodeService(headers, uriInfo).delete(key, value);
}
```

Operation to find the internal operation in the Finger Table.

```
@GET
@Path("closestfinger")
@Produces("application/xml")
public Response getClosestPrecedingFinger(@QueryParam("id") int id) throws Invalid
{
    return new NodeService(headers, uriInfo).findClosestPrecedingFinger(id);
}
```

Operation to get the routing logic where the successor is stored.

```
public Response getSucc()
{
    advanceTime();
    info("getSucc()");
    return response(dht.getSucc());
}
```

Operation to get the Finger info from the Finger Table.

```
public Response getFinger(int id)
{
    advanceTime();
    info("getSucc()");
    return response(dht.getFinger(id));
}
```

Operation to get the key value from the Routing table.

```
public Response get(String key) throws Invalid
{
    advanceTime();
    info("getKey()");
    return response(dht.get(key));
}
```

Operation to update the key value in the finger table.

```
public Response put(String key, String value) throws Invalid
{
    advanceTime();
    info("putKeyValue()");
    dht.add(key, value);
    return response();
}
```

Operation to delete the key value from the finger table.

```
public Response delete(String key, String value) throws Invalid
{
    advanceTime();
    info("deleteKeyValue()");
    dht.delete(key, value);
    return response();
}
```

Operation you can figure out the internal operation in the Finger Table

```
public Response findClosestPrecedingFinger(int id)
{
    advanceTime();
    info("findClosestPrecedingFinger()");
    NodeInfo node = dht.closestPrecedingFinger(id);
    return response(node);
    }
```

package edu.stevens.cs549.dhts.state:-

In IRouting.java:-

In closestSuccessingFinger():-

public NodeInfo closestSuccessingFinger (int id);

Added closest successing finger in the finger table.

In IState.java:-

Added getFinger():-

public NodeInfo getFinger(int index);

The operation for adding node information to get the information about the finger table.

In State.java:-

In setFinger():-

```
public synchronized void setFinger(int i, NodeInfo info)
{
	if(i < finger.length)
	{
		finger[i] = info;
	}
}
```

This operation sets the finger value in the finger table.

In getFinger():-

```
public synchronized NodeInfo getFinger(int i)
{
	if(i < finger.length)
	{
		return finger[i];
	}
	 else
	{
		return null;
	}
}
```

This operation gets the finger value from the finger table.

In closestPrecedingFinger():-

```
public synchronized NodeInfo closestPrecedingFinger(int id)
{
    for(int i = 0; i < finger.length - 1; i++)
    {
        // if id belong to current id to finger id
        if(DHTBase.inInterval(id, (info.id + 1 << i )%IRouting.NKEYS , (info.id + 1 <<
        (i + 1))%IRouting.NKEYS, true))
        {
            return  finger[i];
        }
    }
    return finger[finger.length - 1];
    }
```

Through this operation get closest preceding finger for id, to continue search at that node.

IMPLEMENTATION PART:-

Local Implementation:-

1. **Running the "dht.jar" file using Cygwin Terminal:-**
   The jar file is run with the help of the Cygwin Terminal and by the following command:-
   java –jar dht.jar --http ***http*** --id ***id*** --host ***localhost***
   Here ***http*** is the port number and the ***id*** is the number we want the Node to be represented with and ***localhost*** since we are running the code on the local machine.

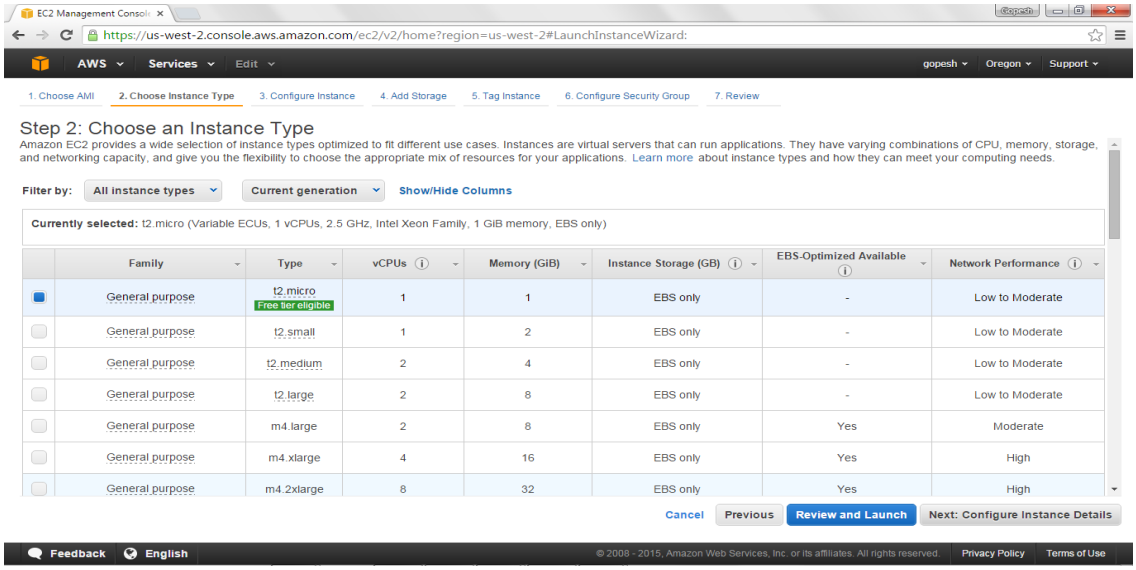
2. **Adding a Key, Getting a Key and Deleting a Key:-**
   A Key can be added for a Node with the help of the following command:-
   "add Key Value"
   Here Key is the name to the key and Value is the value assigned to the key.
   We can see the value assigned to a key with the help of the following command:-
   "get keyname"
   Here by knowing the name assigned to the key we can get the value of the key.
   We can also delete the value of the key with the help of the following command:-
   "del key value".

3. **Joining to a Node in the Network:-**
   We can join one Node to another Node with the help of the following command:-
   "join http://localhost:<port_no.>/dht"
   Where the port no. of the Node we want to join to has to be mentioned.

4. **Displaying the Routes of the Network:-**
   The predecessor and the successor of a particular node can be determined with the help of "routes" command.

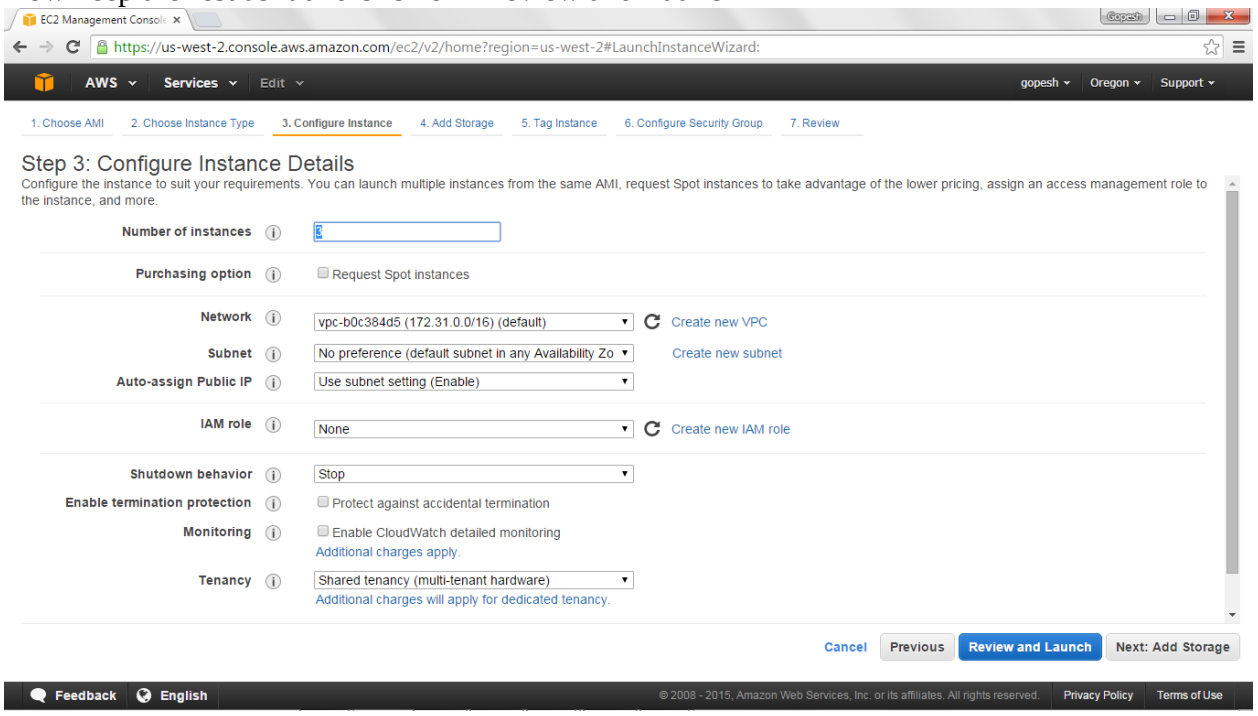
Remote Implementation:-

1. **Created a VPC for creating an EC2 instances network:-**
   I have created a Virtual Private Cloud(VPC) for the secure communication between the EC2 instances. The steps that I followed are:-
   Step 1:-
   Select the "Launch Instance" option and select the General Purpose type of Instance. Instead of clicking on "Review and Launch" select the "Configure Instance Details" tab



Step 2:-

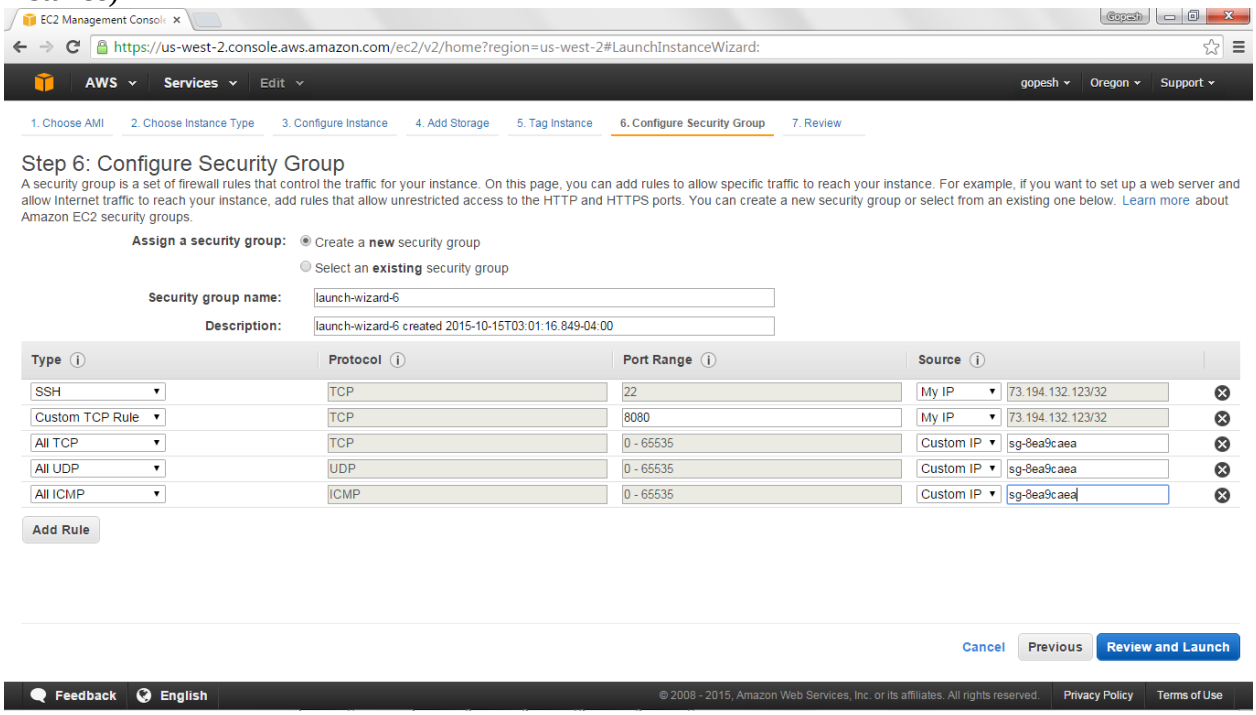
In the "Configure Instance Details" tab, select the first option which is "Number of Instances" to be 3 since we will be requiring 3 EC2 instances for networking.
In the "Network" tab you can either select the default option of the VPC or you can select the VPC you have created for eg. I have created a VPC named CS549 for this assignment.
Now keep the rest as it and click on "Review and Launch"



Step 3:-

It will redirect you to the Configure Security Groups page.
In that the default rule will be added of SSH.

Now for the communication between the EC2 instances, I have added the following rules:-
ALL TCP
ALL UDP
ALL ICMP
Custom TCP Rule(for communication between our local machine and the EC2 instance)



2. **Access and Login to each EC2 Instance Server through SSH:-**
With the help of Putty application for Windows, the EC2 Instance of the Server can be accessed by knowing the "Public DNS" and the "private key" of our Instance.

3. **Copying the Server Jar File to the EC2 Instances Server and Running the Jar File Remotely:-**
For Windows Machine, the transferring of the Jar File to the Linux Instance can be done with the help of "WINSCP" application using the "SCP protocol" and the private key with which we authenticate ourselves to transfer the file to the Instance. And then the Instance is launched remotely by running the DHT jar file on similar basis as running the program locally.