CS549_Report_of_a4_DHT

Renjie Weng

➢ Describe how you completed the code to obtain a working implementation.

1. /dht/notify?failover=true

As described in guide of assignment 4, “When a node detects that its successor has failed, it should contact the successor of its successor, to have it take over from the failed node with the bindings that it has backed up.”

So, the predecessor should detect its successor’s failure, and then call the /dht/notify?failover=ture Web service method on its original succ*2.

In the method of notify_failover on the succ*2, two actions should be done:

First, add its backup bindings to its current bindings; second, replace its original backup bindings with the new backup bindings retrieved from /dht/notify?failover=ture call of its new predecessor.

- In **dht.getSucc()**, I will add codes to detect the failure of its successor:

```
    /*
     * This version gets the local successor from RMI server.
     */
    // WebMethod
    public NodeInfo getSucc() throws Error {
        try {
            // TODO
            // ADD, Ranger, Nov 29
            //System.out.println("Now start succCheck....");
            NodeInfo succ = routing.getSucc();
            if (succ != null) {
//              System.out.println("CheckPredecessor: Predecessor's (id=" + pred.id
//                      + ")");
                try {
                    getPred(succ);
                } catch (Failed e) {
                    info("CheckSuccessor: Successor has failed (id=" + succ.id
                            + ")");
                    try {
                        setSucc(state.getBackupSucc());
                        info("Set succ's succ="+state.getBackupSucc()+" succeeded.");
                        succ=getSucc();
                        try {
                            client.notifyFailover(state.getBackupSucc(), state.extractBindings());
                        } catch (Failed e1) {
                            System.out.println("Notify failover to succ="+succ.id+" failed!");
                            e1.printStackTrace();
                        }
                    } catch (RemoteException e1) {
                        e1.printStackTrace();
                        System.out.println("Failed: setPred(state.extractBindings().getSucc());");
                    }
                }
            }
            // END-ADD
            return routing.getSucc();
        } catch (RemoteException e) {
            severe("GetSucc: RMI error in getSucc: " + e);
            throw new Error(e);
        }
    }
```

- **notifyFailover** executes the function of /dht/notify?failover=true:

```
80      @PUT
81      @Path("notifyFailover")
82      @Consumes("application/xml")
83      /*
84       * Trigger failover in succ of succ
85       */
86      public Response notifyFailover(TableRep predDb) {
87          /*
88           * See the comment for WebClient::notify (the client side of this
89           * logic).
90           */
91          //if(failover==true){
92          new NodeService(uriInfo).notifyFailover(predDb);
93          //}
94          return Response.notModified().build();
95      }
```

I cannot name this function as @PUT @Path(“notify”) because the RESTFUL API doesn’t allow two Web service method share same VERB and PATH.

- **Dht.notifyFailover(TableRep predDb)**:

```java
// WebMethod
public void notifyFailover(TableRep predDb) throws Error {
    info("Notify Failover is running....");
    try {
        // add original backup bindings to visible bindings
        state.failoverBindings();
        // replace original bindings with new predecessor's bindings
        state.backupBindings(predDb);
    } catch (RemoteException e) {
        log.severe("Remote exception while backing up bindings: " + e);
    }
}
```

- **state.failoverBindings()**:

```java
public synchronized void failoverBindings() throws RemoteException, Error{
    checkFailed();
    // add backup to dict
    Enumeration<String> keys = backup.keys();
    while (keys.hasMoreElements()) {
        String k = keys.nextElement();
        List<String> v = backup.get(k);
        dict.put(k, v);
    }
    // clear backup
    backup = Persist.newTable();
}
```

2. PUT & DELETE /dht/backup?key=KEY&val=VAL

- Add codes of addBackup in **dht. add(String k, String v)** :

```
public void add(String k, String v) throws Error, Invalid {
    try {
        /*
         * Validate that this binding can be stored here.
         */
        int kid = DHTBase.NodeKey(k);
        NodeInfo info = getNodeInfo();
        NodeInfo pred = getPred();

        System.out.println("k's hashcode: "
                + Math.abs(k.hashCode() % IRouting.NKEYS));

        if (pred != null && inInterval(kid, pred.id, info.id, true)) {
            /*
             * This node covers the interval in which k should be stored.
             */
            state.add(k, v);
            // TODO, add backup
            NodeInfo succ = this.getSucc();
            try {
                addBackup(succ,k,v);
            } catch (Failed e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        } else if (pred == null && info.equals(getSucc())) {
            /*
             * Single-node network.
             */
            state.add(k, v);
            // TODO, add backup
            NodeInfo succ = this.getSucc();
            try {
                addBackup(succ,k,v);
            } catch (Failed e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        } else if (pred == null && info.equals(getSucc())) {
            severe("Add: predecessor is null but not a single-node network.");
        } else {
            throw new Invalid("Invalid key: "+k+" (id="+kid+")");
        }
    } catch (RemoteException e) {
        severe("Add: RMI error: " + e);
        throw new Error(e);
    }
}
```

- The difference between add(k,v) and addBackup(k,v) is addBackup will do a Web service call to its successor, and then add the key-value pairs to state.bakup parameter. Show the codes of **state. addBackup(String k, String v)** :

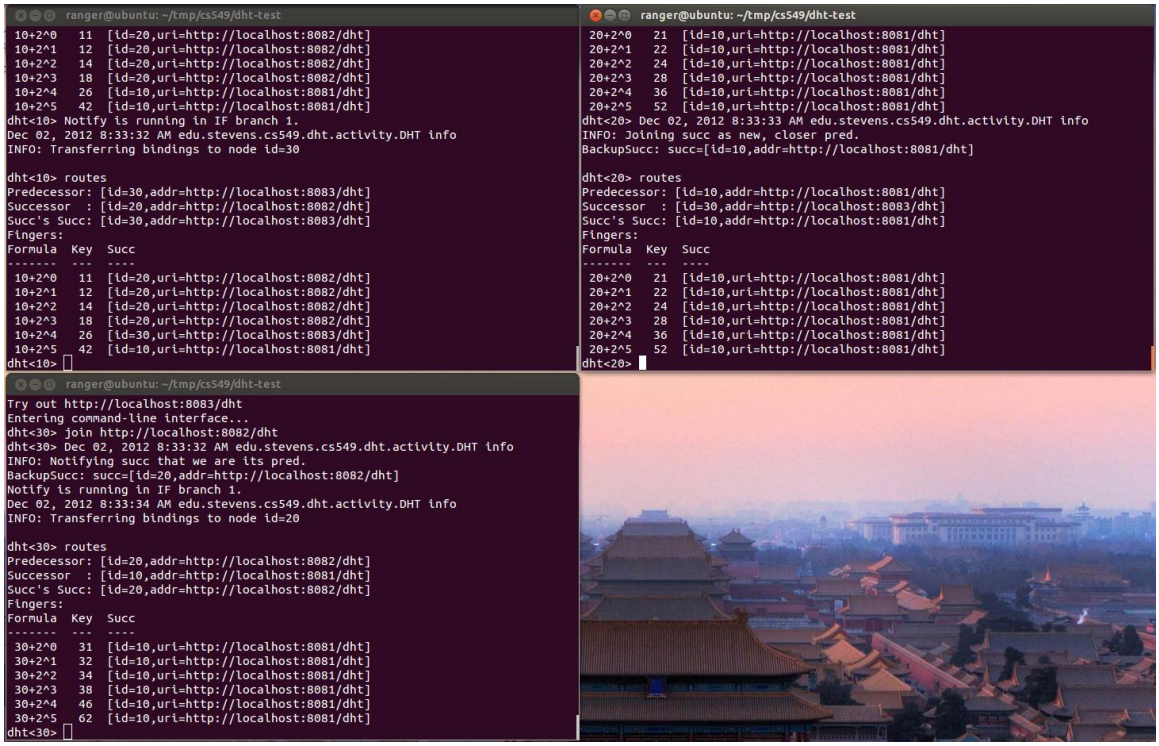
```java
    public synchronized void addBackup(String k, String v) throws RemoteException, Error {
        checkFailed();
        List<String> vl = backup.get(k);
        if (vl == null) {
            vl = new ArrayList<String>();
        }
        vl.add(v);
        backup.put(k, vl);
    }
```

➢ Describe how you tested the code.

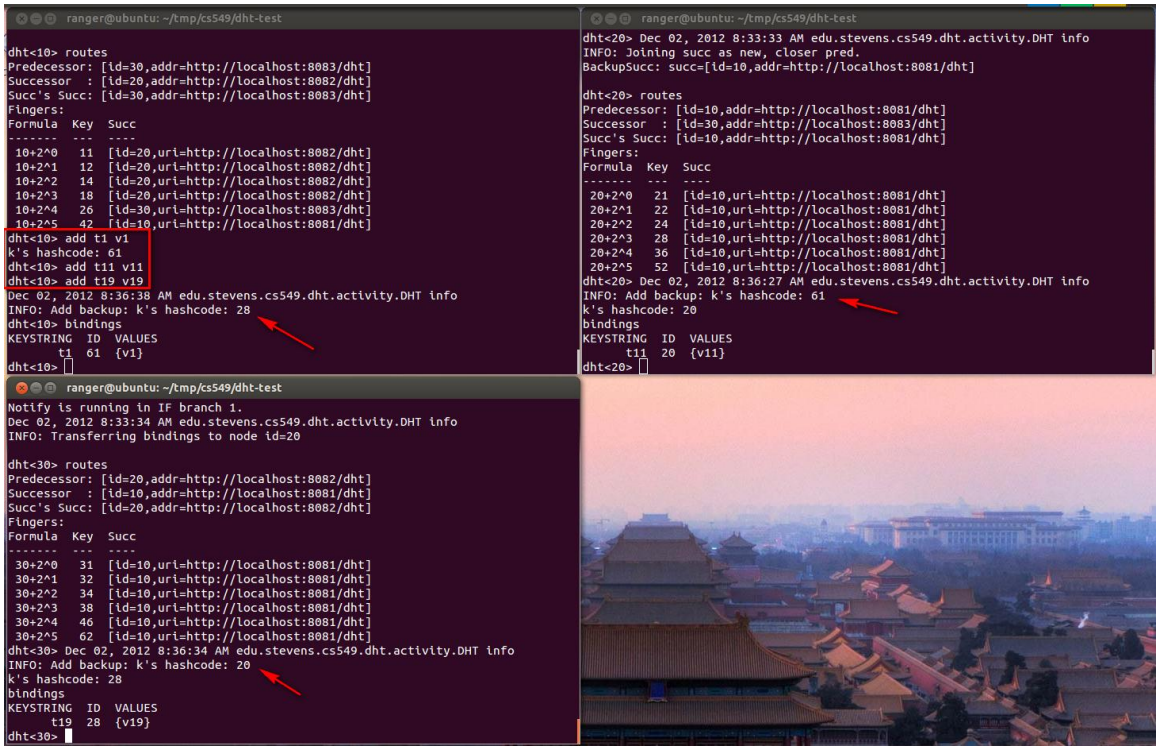
I have two tests, the 1st one is among three nodes, and the 2nd one is adding a fourth node to the three nodes test after the second node has failed in the first node. Then fail the 3rd node.

1. Test One:

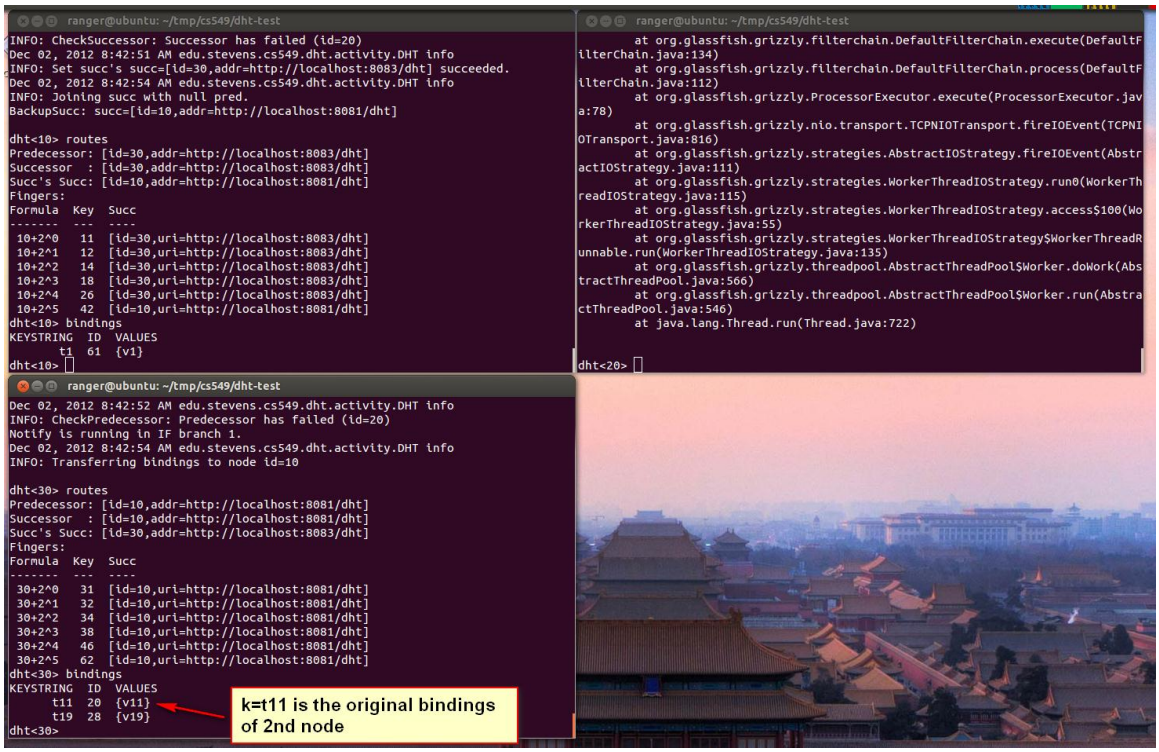
1.1. Link 1st to 3rd nodes together, show their routes.



1.2. Add three key-value pairs, as each node has one in its binding. I print both the add(k,v) and addBackup(k,v) when they are running in nodes.

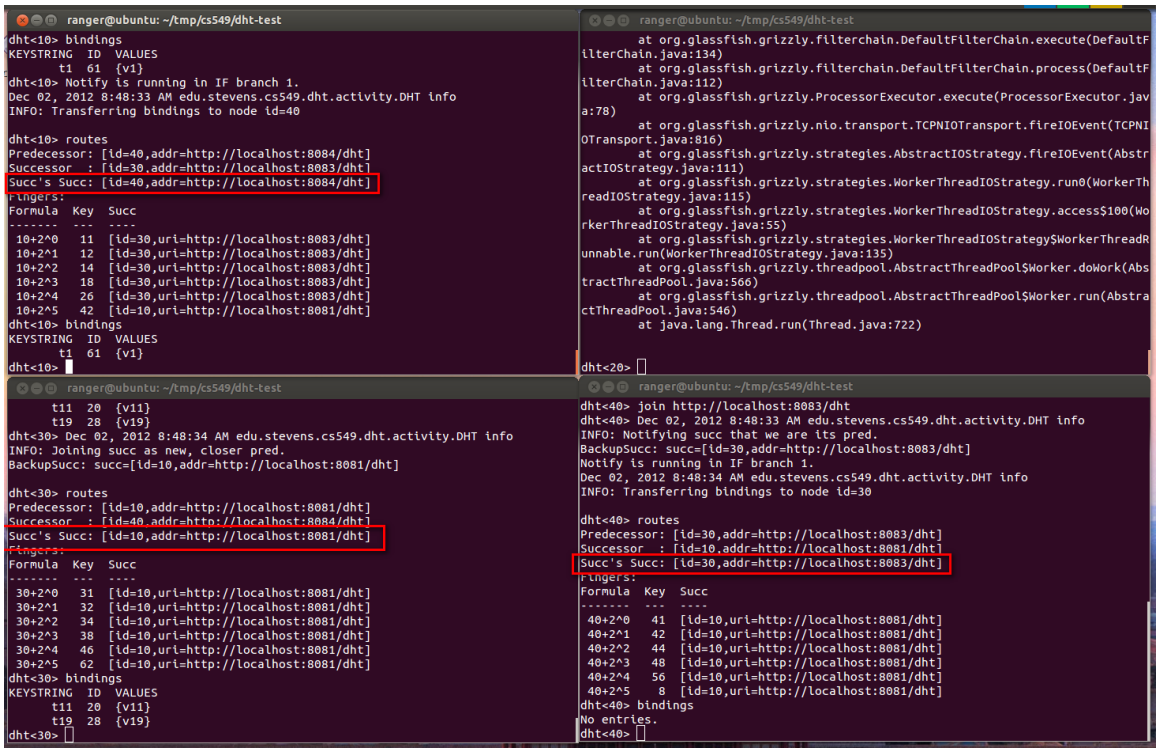


1.3. Fail@2nd node, then succ's succ would change after 2nd node has failed, and the bindings of 3rd node's backup bindings would take place of 2nd node's bindings, as well.

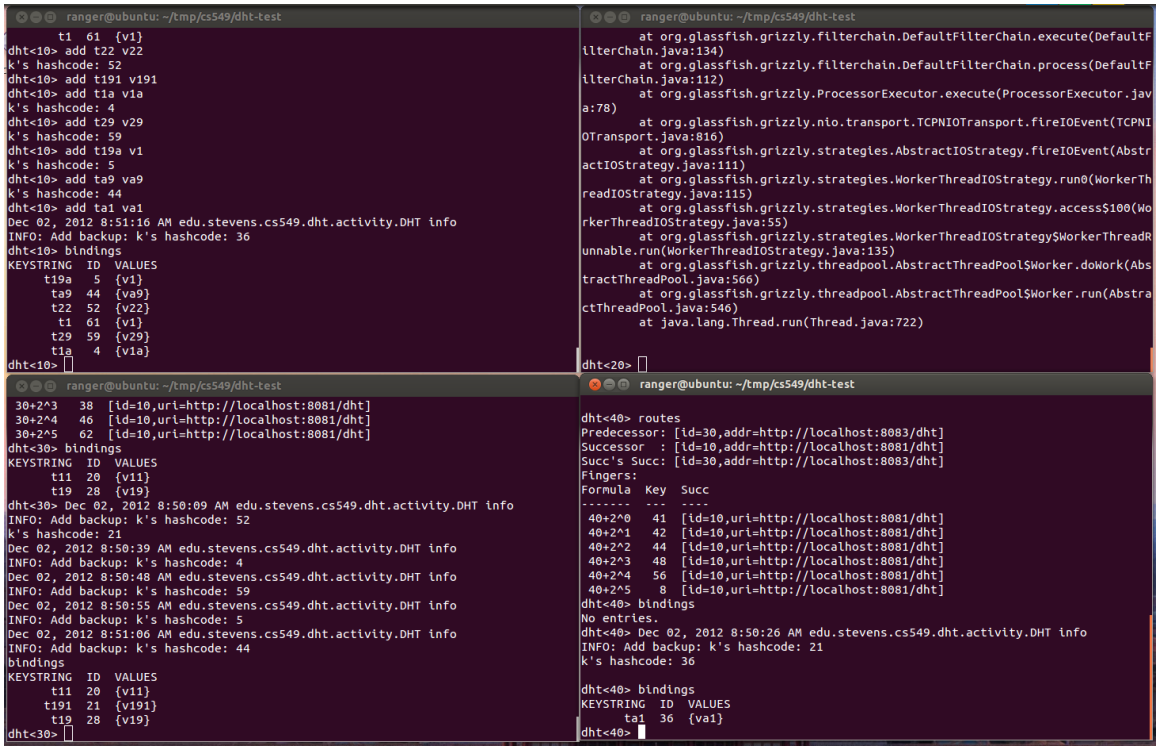


Test Two:

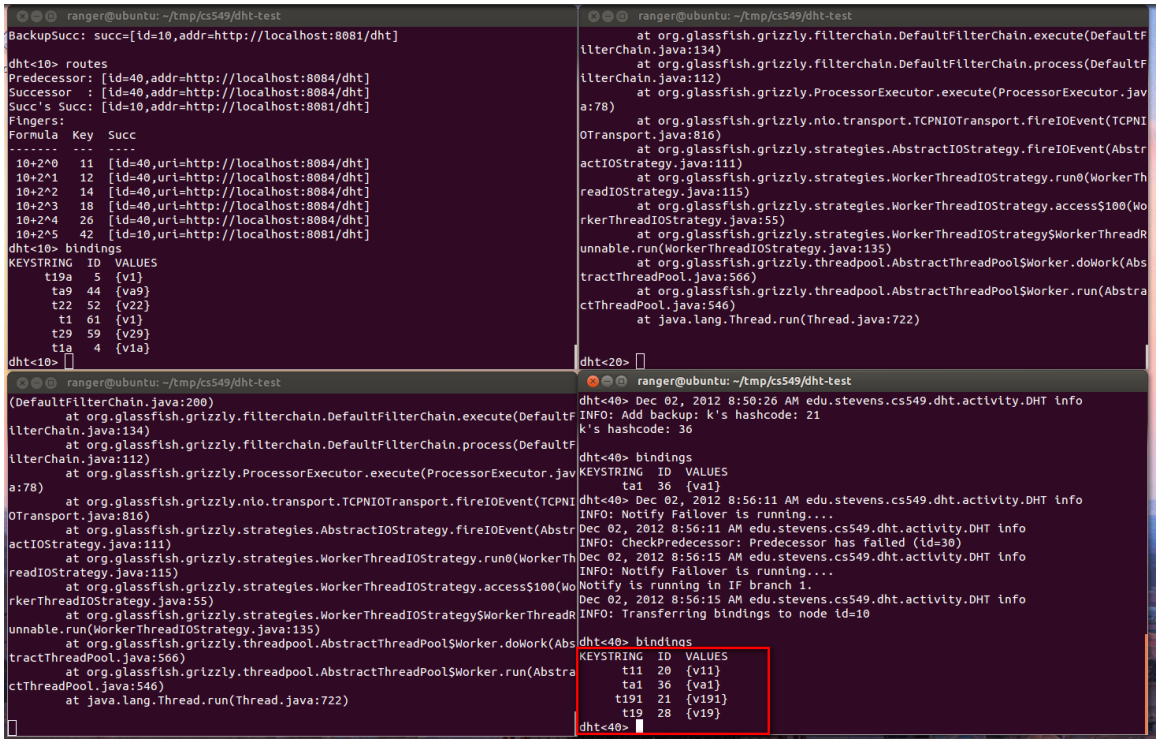
2.1. Add the 4th node into the first three nodes DHT link (actually there are only two nodes alive for 2nd has failed), show their new routes and bindings.



2.2 Add some new key-value pairs to DHT circle, show it works all the same in bindings and backup bindings after 2nd node has failed.



2.3 fail@3rd node, then the bindings of 3rd node should be shown in 4th node with the original bindings on 4th node. Because the 4th node has backed up its predecessor's bindings, and will failover them after its predecessor fails.



➢ Test Demo:

The video shows TWO tests:

First Test, three nodes and 2nd fail;

Second Test, add a 4th node to the First Test, then fail the 3rd node.

A4_Ranger_cs549_2012-02_2234 http://screencast.com/t/GsUQ32AVDtk2