DISTRIBUTED SYSTEMS – LAB 2 REPORT

Distributed Systems – Lab 2 Remote Method Invocation Report

Submitted: October 14, 2014
By: Devan Shah 100428864
Submitted to: Weina Ma

Lab Report Questions:

1. What did you use in your implementation to ensure that the Election service records a vote whenever a client thinks they have cast a vote?

   When the voter initializes the client in the form of casting votes the voter is asked for the candidate that they want to vote for and their voter number. Once the client has this data it sends it directly to the vote function which records the vote in a HashMap, if a duplicate of voter number is detected the voter is notified that they have already casted a vote. The vote function returns a Boolean to the client and the client depending on the Boolean value determine if the vote was successful or failed and prints it to the client accordingly.

2. How did you achieve the guarantee that all votes are safely stored even when the server process crashes?

   I incrementally write the votes to a file when the vote function is called by the client, this way I can guarantee that all votes are safely stored even when the server process crashes. The HashMap that stores the votes that are casted is written to a file "ElectionResultsRawData.ser" in the form of serialized data (secure). The server is able to read the serialized data from the file and restore the HashMap of votes casted.

3. Outline your implementation for ensuring that the records remain consistent when it is concurrently accessed by multiple clients. You should include snippets of your program to help with your explanation.

   My implementation is using HashMap <Integer, String> to store the votes that were casted, also uses HashMap <String, Integer> to store the results of the election by candidate and a Vector <Object> to store the serialized version of the results in a vector form. Please refer to *Figure 1* and *Figure 2* for snippet from source code. Vectors is a data types that use auto synchronization for accessing and updating operations. Therefore, the records are always consistent when it is accessed concurrently by multiple clients. Furthermore, I made the vote and results functions synchronized as this will help to make sure that HashMaps are accessed in a synchronized manner, this way only one thread (client) can make changes to the HashMaps therefore this will avoid any consistently issues. Please refer to Figure 3 and Figure 4 for snippet from source code.

```
/*
 * Stores the votes that are casted will contain:
 *     Integer - voters ID
 *     String - the candidate's name they voted for.
 */
public Map <Integer, String> votesCasted ;
```

Figure 1 Shows the HashMap that is used to store the votes that are casted by the users. Which is updated in the vote function which is synchronized.

```
// Declare vector and HashMap
Vector <Object> candidateResults
Map <String, Integer> candidateResultsHolder
```

Figure 2Shows the HashMap and the Vector that is used to store the results of the election. Which are in the result functions which is synchronized.

```
@Override
public synchronized boolean vote ( String candidateName, int voterNumber )
{
```

Figure 3 Shows the vote function being synchronized to make sure that there are no issues when accessed by multiple clients.

```
@Override
public synchronized Vector<Object> result () throws RemoteException
{
```

Figure 4 Shows the result function being synchronized to make sure that there are no issues when accessed by multiple clients.

Following are the example and snippets from each of the tasks this is how the program was tested in the conditions mentioned. Feel Free to take a look at this, a readme file is provided to explain how to run the programs for each task.

Task #1:

Define the interface to the Election service in Java EMI.

A. The Election Interface can be found in the attached folder **DistributedSystems - Lab 2 - Task 1 Define Election Interface** under **src**. (~/DistributedSystems - Lab 2 - Task 1 Define Election Interface\src\ElectionInterface.java). The Election Interface contains function definitions for *vote* and *result* functions. The vote function takes in a string (candidate's name) and int (voter number), also returns a Boolean to identify if the vote was casted successfully or not. The result() function takes in no parameters but returns a vector of objects (ElectionResults objects). I have an ElectionResults class that is used to store the election results as a serialized object in a vector. The ElectionResults.java file can be found in folder **DistributedSystems - Lab 2 - Task 1 Define Election Interface** under **src** (~/DistributedSystems - Lab 2 - Task 1 Define Election Interface\src\ElectionResults.java) More details regarding the files and functions can be found in the source files ElectionInterface.java and ElectionResults.java.

Task #2:

Implement the Election service in Java RMI. Your implementation should ensure that a vote is recorded whenever a client thinks they have sent in a vote.
Write a client program and run multiple instances of the client to case votes. A client may also query the server for the results and display them.
The Server can be started with command: "java ElectionServer localhost"
The Client can be started with command: "java ElectionClient vote localhost 1099" for voting
The Client can be started with command: "java ElectionClient results localhost 1099" for results
The source can be found under: **DistributedSystems - Lab 2 - Task 2 Implement Election\src** in the submitted zip file.
Following are outputs from the client and server when votes are casted/results are retrieved:

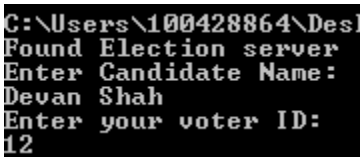

Figure 9Client 1 casting vote

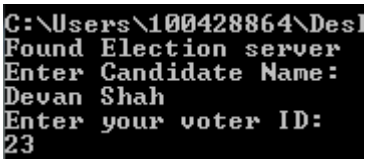

Figure 8Client 2 casting vote

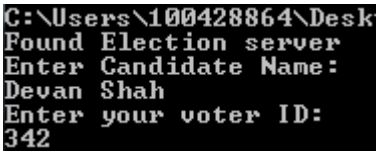

Figure 7Client 3 casting vote

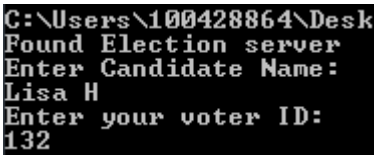

Figure 6Client 4casting vote

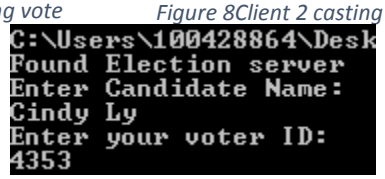

Figure 5Client 5 casting vote

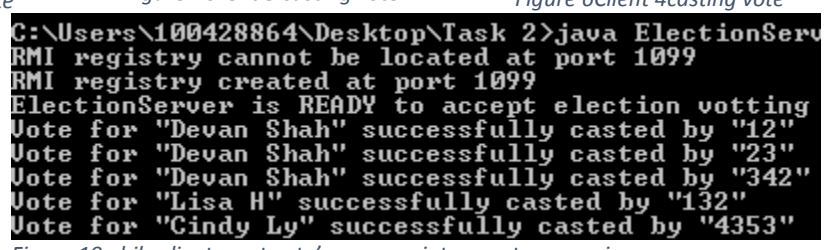

Figure 10while clients cast vote's server prints as votes come in

Figure 11Client querying the results from the server

Task #3:

Now modify, if necessary, your implementation to ensure that the votes are recorded even when the server process crashes.

Using the same votes that the 5 clients casted in Task 2, the server stored the casted results in a file “ElectionResultsRawData.ser” and supports restoring the results when server is started again.

The Server can be started with command: “java ElectionServer localhost”
The Client can cast a vote with command: “java ElectionClient vote localhost 1099”
The Client can retrieve results with command: “java ElectionClient results localhost 1099”
The Server can be killed with ctrl + z
Restart the Server and the server will ask if you would like to restore using the file.
Enter yes and then use the client to retrieve results, the results will be the same as that were retrieved previously.

The source can be found under: **DistributedSystems - Lab 2 - Task 3 Save Results\src** in the submitted zip file.

Following is the sample output following the procedure mentioned above:

```
C:\Users\100428864\Desktop\Task 2>java ElectionServer localhost
RMI registry cannot be located at port 1099
RMI registry created at port 1099
ElectionServer is READY to accept election votting .....
Vote for "Devan Shah" successfully casted by "12"
Vote for "Devan Shah" successfully casted by "23"
Vote for "Devan Shah" successfully casted by "342"
Vote for "Lisa H" successfully casted by "132"
Vote for "Cindy Ly" successfully casted by "4353"
The total number of candidates being returned to client: 3

C:\Users\100428864\Desktop\Task 2>^Z
```

Figure 14 Server was terminated using ctrl + z this has saved the results that were previously sent to the server.

```
C:\Users\100428864\Desktop\Task 2>java ElectionServer localhost
RMI registry cannot be located at port 1099
RMI registry created at port 1099
Restore File "ElectionResultsRawData.ser" was found. Would you like to restore E
lection results form this file? ( yes or no )
yes
Restoring the Election Server back to it’s previous state before crash ... Pleas
e Wait.
ElectionServer is READY to accept election votting .....
```

Figure 13 After server is terminate and server process is restarted, it will be restored if user wants.

```
C:\Users\100428864\Desktop\Task 2>java ElectionClient
Found Election server
Current Election Results are:
   Candidate "Devan Shah" has accumulated: "3" votes.
   Candidate "Cindy Ly" has accumulated: "1" votes.
   Candidate "Lisa H" has accumulated: "1" votes.
```

Figure 12Client querying the server for results after server was restored

Task #4:

Modify your implementation, if necessary, to ensure that the records remain consistent when it is concurrently accessed by multiple clients.

My current implementation is using HashMap <Integer, String> to store the votes that were casted, also uses HashMap <String, Integer> to store the results of the election by candidate and a Vector <Object> to store the serialized version of the results in a vector. HashMap and Vectors are data types that use auto synchronization for access and update operations of the data types. Therefore, the records are always consistent when it is accessed concurrently accessed by multiple clients. Furthermore, I made the vote and results functions synchronized as this will help to make sure that the vote and results functions are synced when they are called and accessing the same data types.