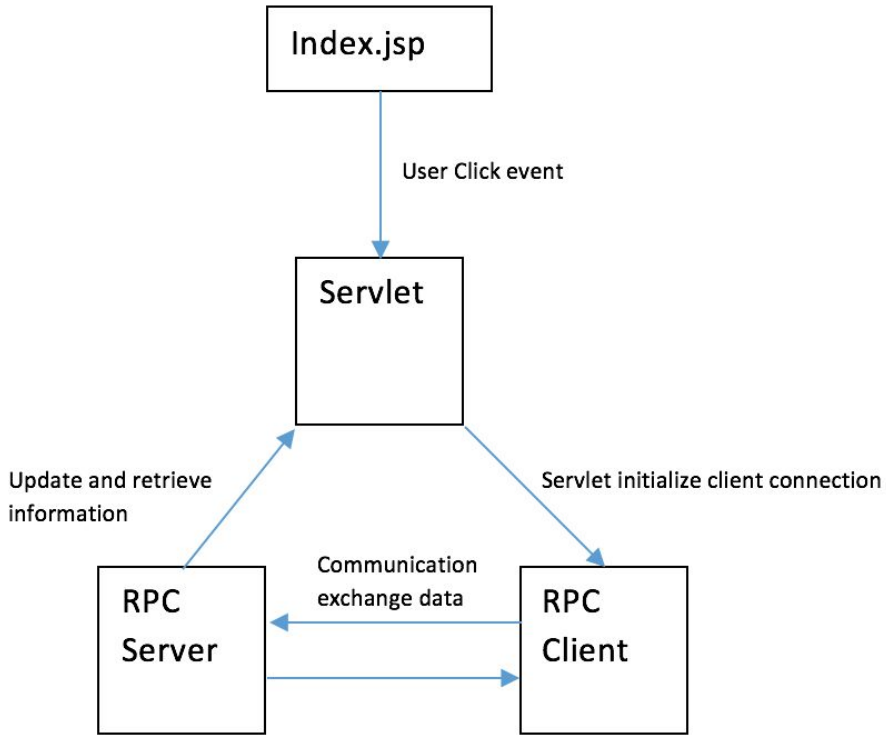
CS5300 Project1b

0. Important Test Information for the convenience of Professor and TA
(1)URL format to access the application :
http://server0.**cs2238**.bigdata.systems:8080/**Project1b/session**
(2)Cookie expire time: **5 minutes(**This is to ensure the cookie is still alive when iinstance fully recover from reboot **)**
(3)run **launch.sh** to start up
When setting the value of N and F, please ensure that **the value of N in launch.sh is the same as the value of N in install.sh**. Also, please ensure **N >= 2*F+1**
(4) The war file and scripts are put under the same folder in order to start the application by running the lauch.sh.

1.Application Structure



The application structure is shown above. Index.jsp file is for user interface, servlet is to handle user input and event from jsp file. Servlet will use RPC client class when servlet needs to communicate with other servers. RPC client will exchange data with other server's RPC server class through UDP protocol. RPC server will update and retrieve information from servlet once it receives instruction from other server.

2.Shell Scripts

Shell script

2.1. General:
There are three script in total: launch.sh, installation.sh, reboot.sh
The launch.sh is mainly used to launch instances. It also triggers the installation.sh so that the installation.sh is executed on each instance when an instance is launched. The reboot.sh is a "virtual" script which should be executed after an instance is rebooted manually.

2.2. launch.sh
This script fulfills three missions:
first, it copies the local file to aws s3 bucket and gives permission to all aws users. It also copies the installation.sh to aws s3 so that when we reboot and connect to the instance, we can run the installation.sh again.
Second, it deletes any domain in simpleDB with the same name and creates a new simpleDB domain to put instances' ServerIDs and IPs.
Third, it launches N instances with specific type and run installation.sh on each launched instance.
We also need to configure aws credentials at the start so that we could run relevant aws services like s3 and simpleDB.

2.3. install.sh
The install.sh is triggered by launch.sh every time an instance is launched. It runs on every single instance and executes inside command lines.
This install.sh is designed to be able to be run more than once to handle installation failure for 5.2. This can be achieved by check if some files already exist before download, upload or create them.
This shell script realizes functionalities as follows:
(1) Configure aws credentials so that aws account and service can be used.
(2) Change the java version installed on instance from java7 to java8 so that the compiled java version is compatible with the java version on instance.
(3) Install tomcat8 on instance and copy the .war from s3 to tomcat server if the .war file does not exist.
(4) If there was not a file indicating N and F, save N (number of launched instances) and F (fault-tolerance) to a file so that they can be read by Java codes and set variables correspondingly.
(5) If there was not a file indicating this instance's ServerID and IP, save this instance's ServerID and IP to a file and upload that file to simpleDB.
(6) If there was not a file indicating all of instances' IPs, wait until all instances' IPs have been uploaded to simpleDB. This can be achieved by comparing the number of items and N.
Once all of instances' IPs have been uploaded, download the whole IP information from simpleDB, and save it in a file.
(7) Start tomcat server.

2.4. Reboot.sh
The reboot.sh is a “virtual” script. When an instance fails, reboot it in console as instructed. Connect to the rebooted instance, execute the command in reboot.sh to restart tomcat server on this instance.
Generally, it provides aws credentials, gets the installation script and re-run it.

3.Servlet

Java servlet file for session management. The functionality of this file is to handle HTTP GET and POST request from frontend jsp file.

When it receive a GET method, it means either the user enter http://localhost:8080/CS5300Project1/session. Then it will check from user’s local cookie to see if there is an existing cookie with the key "cs5300project1", if exists such cookie, retrieve the sessionID information from the cookie value. Then it will call sessionReadClient() along with the location data retrieved from cookie in the RpcClient class to check if there exists a corresponding session object in the session table on the server in the location data stored in the cookie. If it does, retrieve the session from corresponding servers and refresh this session and write it back with an updated version number to W random servers and expiry time; if it does not, render a new session. Call sessionWriteClient() to write the new session to W random servers, with WQ valid reponses.

Then the doPost() method handles the post request from the index.jsp. There are two events handled by doPost() method, first one is message replace click. We retrieve the current session information from cookie or create a new session if the current session is timed out through sessionReadClient() and sessionWriteClient() method inside of the RpcClient class, then set the message field inside of the session object from the text input field of index.jsp file. The second event is the logout event. It will redirect the user to the logout.jsp page and terminate current active session from the session table.

createCleanupThread(): it is used to create a daemon thread that will be activated every 5 minutes to clean up the session table, remove all time-out session.

initializeRpcServer(): it is used to initialize RPC server thread that listen to port 5300 for any incoming packet. It will be called once the instance launches the project.

restoreServerInfo(): it is used to read in the local text file which stores the server information: serverID and rebootNumber. It is called every time the servlet initializes.

saveServerInfo(): it is used to save the current server information into a local text file in case of crash. It will update the reboot number every time the servlet starts. It is called every time the servlet initializes.

restoreServerMapping(): it is used to restore the serverID and IP mapping information from local text file. And load it into servlet built-in hashmap to allow searching IP address from given input server ID. It is called every time the servlet initializes.

4.RPC

In rpc package, we implemented four classes to help build the RPC mechanism.

4.1 RpcClient.java

RpcClient represents the client stub which provide interface for SessionServlet to send read/write message to

There are mainly two methods in this class.

First method: sessionReadClient. This method takes 3 arguments representing session ID, version number and an array of destination IP. A socket is created in this method to send a reading request info to the IP addresses contained in the InetAddress[] argument. The format of the message sent to RPC server is: **callID_OperationCode_sessionID_versionNumber**.

After the sending operation, the socket starts listening to packets sent back from the RPC server. Then it compares the received callID and versionNumber with the callID and versionNumber sent away. Once these two numbers are correct, this response will be regarded as a good one. Then listening is stopped. A Response object is sent to Session Servlet with the following parameters being set: resStatus(indicating the result of RPC reading), resMessage(the message read from storage), serverID(the server from which the data was read).

Second method:sessionWriteClient. This method takes 5 arguments representing sessionID, version number, message to be stored, expire time and destination IP addresses. The message sent to RPC server follows the format:
callID_OperationCode_sessionID_versionNumber_message_expireTime.
After sending this to all the IP in IP array, the socket starts to listen to responses. It will compare the compare the callID and version Number contained in the response from RPC server. Each time a qualified packet arrived, a counter would increase itself. When the number of counter reach WQ, then the method would return a Response object to the Session Servlet containing writing result, updated message and location metadata.

4.2 RpcServer.java

rpcCallRequestProcessor(): this method is running as a background thread to receive requests from RPC Clients and call sessionRead() or sessionWrite() method. It has started running at the very first moment.

sessionRead(String readInfo): this method is used to search session and read data from session given the sessionID and version number provided by readInfo. The return information contains callID, session ID, version Number, message and server ID.

sessionWrite(String writeInfo): this method is used to update session message or create new session given the sessionID, version number and message provided by writeInfo. The return information contains callID, server ID, session ID and version Number.

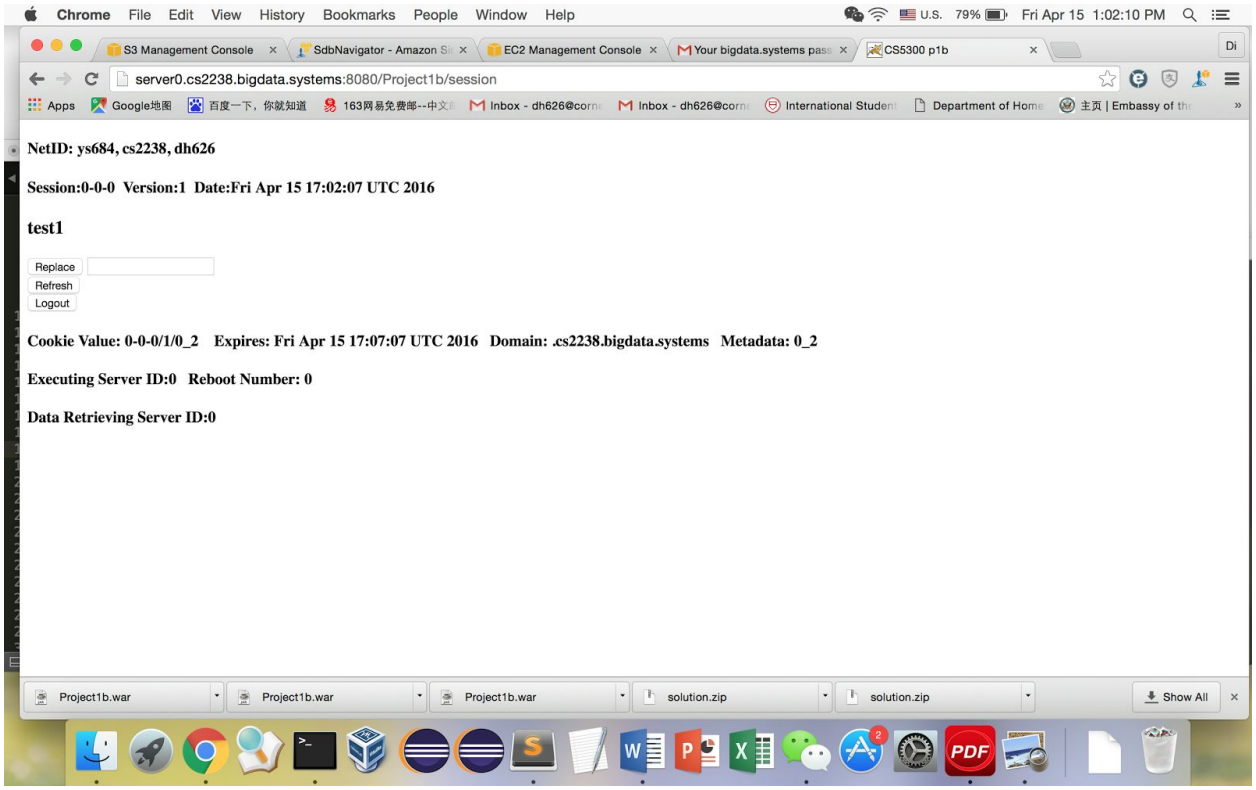
4.3 Response.java
Response is a class mainly used to pass information from RPC client to session servlet. It has 3 fields: resStatus(the result of operation), resMessage(response message), locationData and serverID.

4.4 Utils.java
This class contains all the constants that are used through the whole application. Important information: **DOMAIN_NAME** = ".cs2238.bigdata.systems"

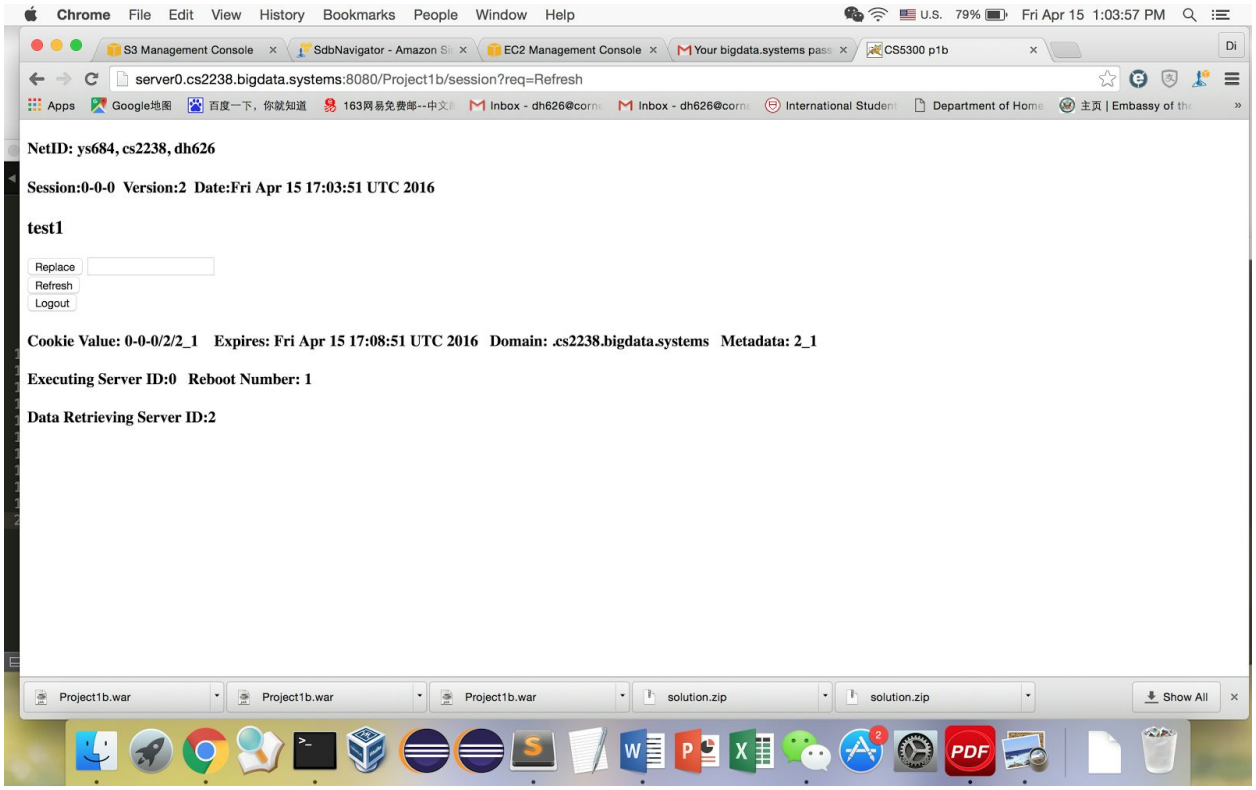
5. Session
This is the wrapper class for each session providing the following field: serverID, sessionID, versionNumber, message, expireTime and createTime.
When we initialize this class, a unique sessionID needs to be passed in, the message will be set to default "Hello User!", the createTime will be set to current time and expireTime will be set to 5 minutes from the current time.

6. Screenshots for 1 resilient system

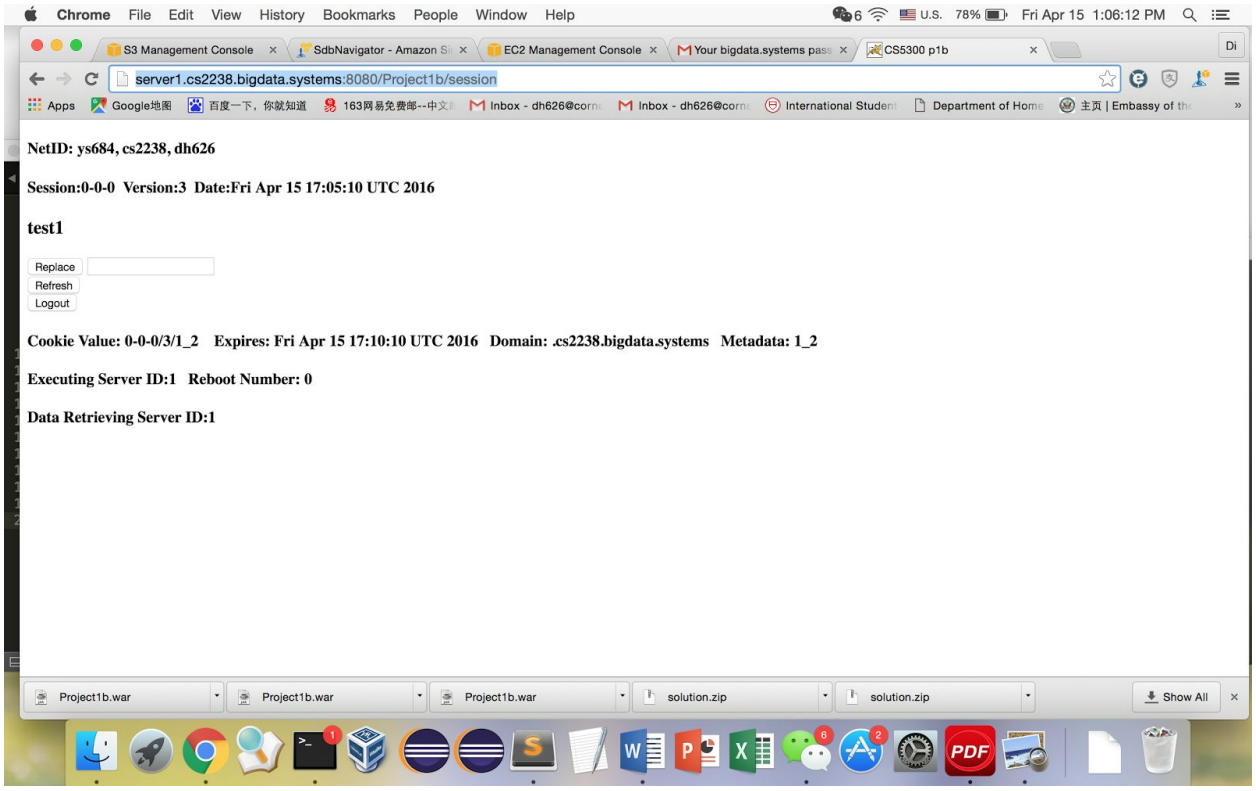
(1)Connected to Server 0, input a message and clicke replace. Version number becomes 1.



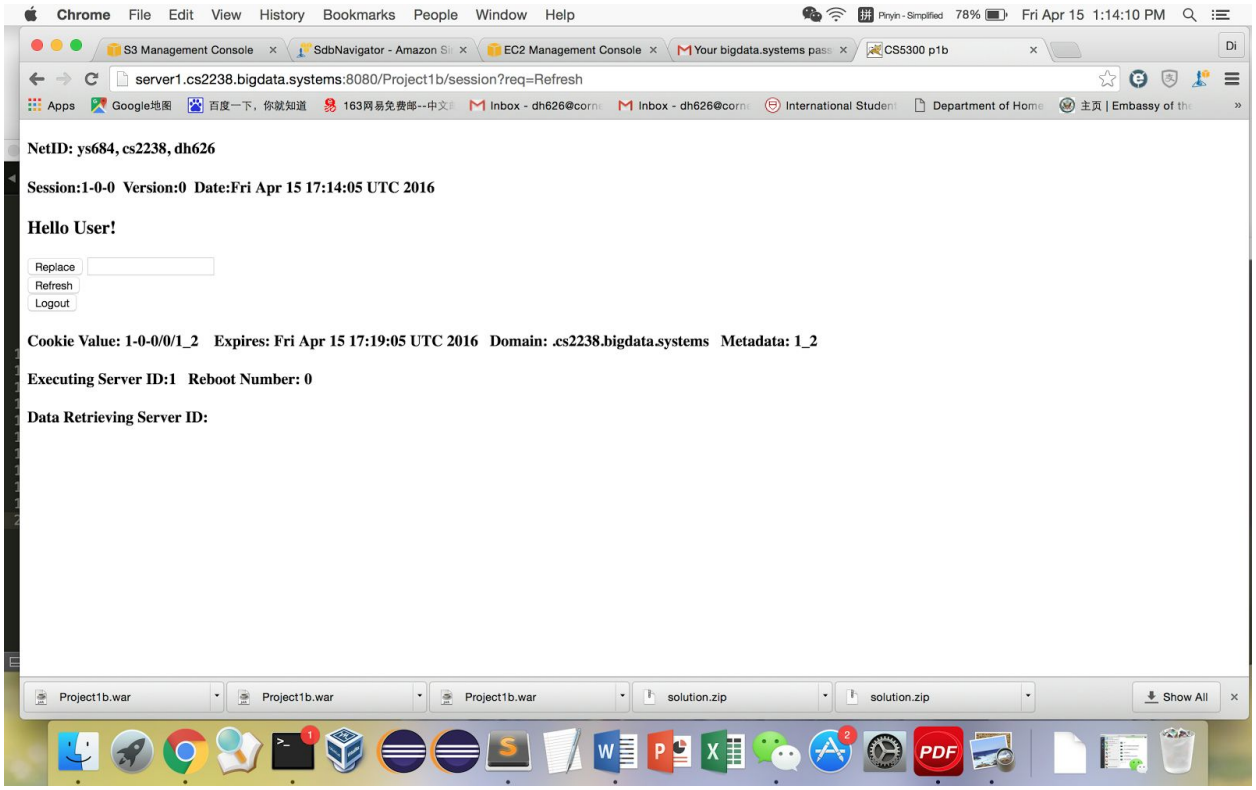
(2)Screenshots taken after rebooted server 0, ran its reboot script and clicked refresh. Then the version number increased by 1 and the message remain the same.



(3) After rebooted server 0 without running the reboot script(which means server 0 has crashed), we change the url so request goes to server 1. The screenshot shows the application still works with the version number increased by 1 and message remaining the same.

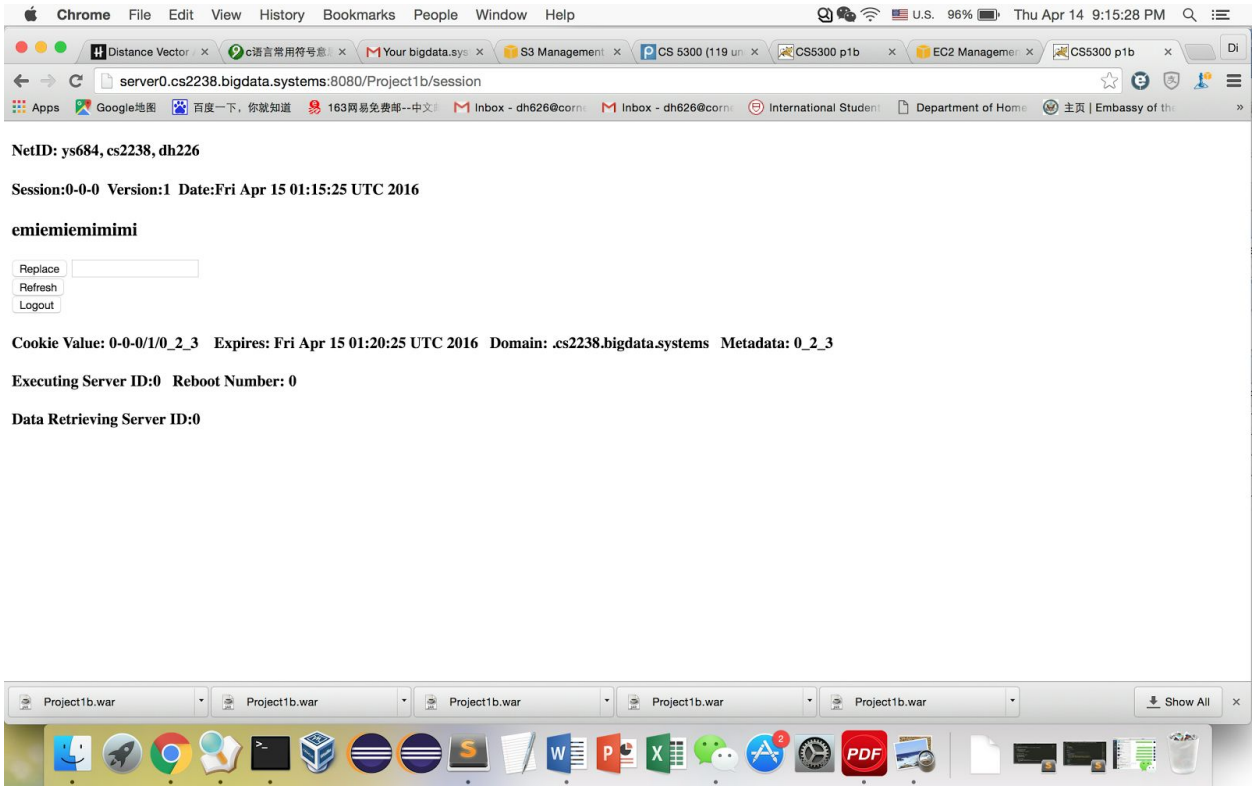


(4)Waited for 5 minutes until session timed out and clicked refresh. A new session has been created with server ID 1 as the first digit in sessionID.

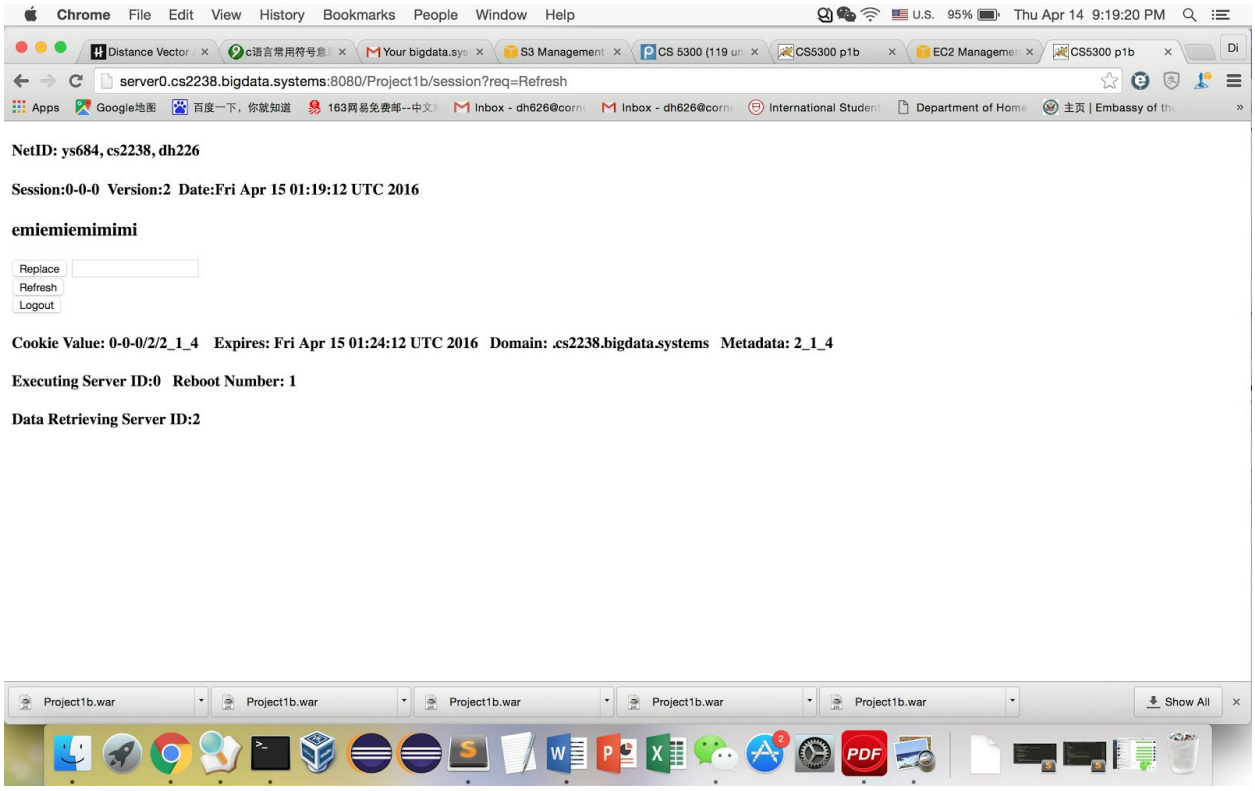


7. Screenshots for extra credit: demo for 2 resilient application

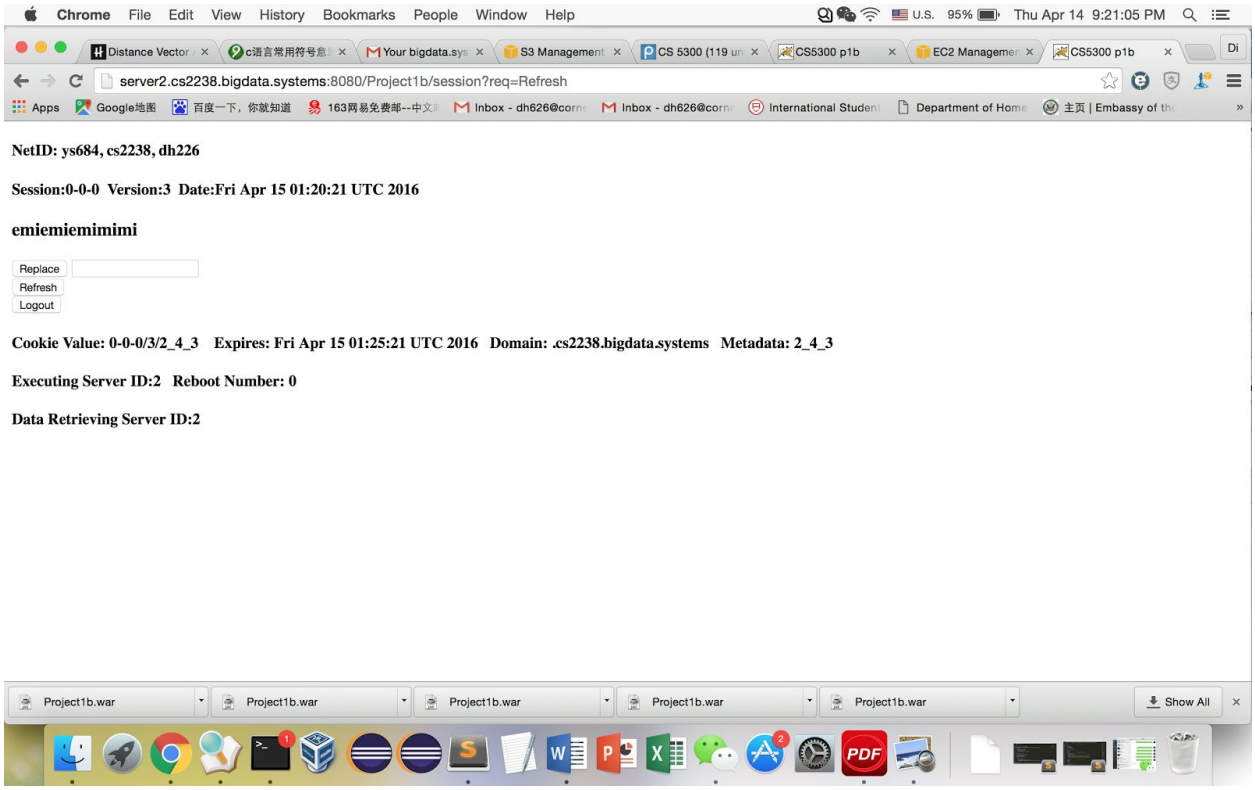
(1)Connected to server 0 and clicked replace.



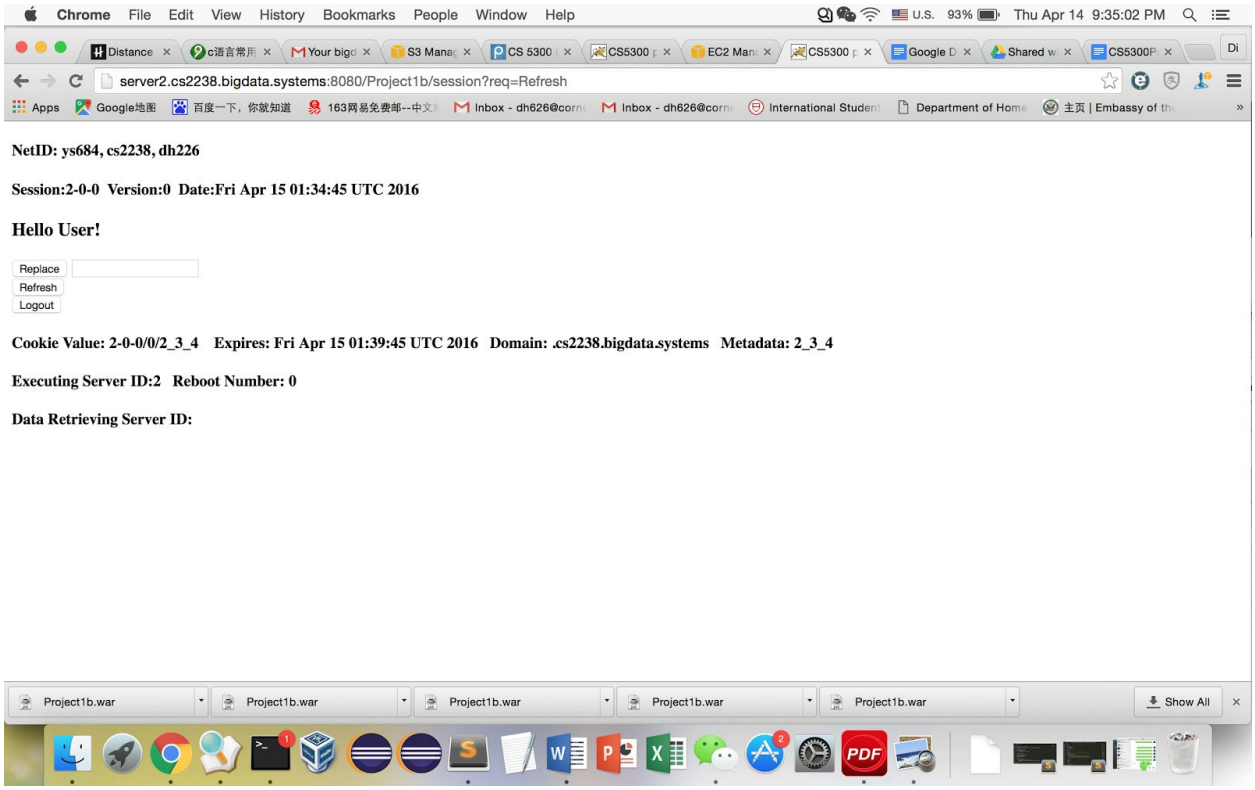
(2) Server 0 rebooted and refresh button clicked. Version number increased by one as expected and reboot number also increased by 1.



(3)Both **server 0 and server 1** were crashed. Then we connected to server 2. Version number still increased by 1, which indicates a working 2 resilient application.



(4)Waited for timeout and click refresh. A new session has been created with server ID 2 as the first digit of session ID. Version number is 0.


Chrome File Edit View History Bookmarks People Window Help
U.S. 93% Thu Apr 14 9:35:02 PM
server2.cs2238.bigdata.systems:8080/Project1b/session?req=Refresh
Apps
NetID: ys684, cs2238, dh226
Session:2-0-0 Version:0 Date:Fri Apr 15 01:34:45 UTC 2016
Hello User!
Replace
Refresh
Logout
Cookie Value: 2-0-0/0/2_3_4 Expires: Fri Apr 15 01:39:45 UTC 2016 Domain: .cs2238.bigdata.systems Metadata: 2_3_4
Executing Server ID:2 Reboot Number: 0
Data Retrieving Server ID:
Project1b.war
Project1b.war
Project1b.war
Project1b.war
Project1b.war
Show All