CS 5300
The Architecture of Large Scale Information Systems

Project 1b: Scalable and Available Website

SUBMITTED BY

Karteeka Gosukonda (kg399)
Shruthi Srinivasan(ss3354)
Nandini Nagaraj(nn262)

TABLE OF CONTENTS

Project Overview:

The available and scalable website uses the AWS Elastic Bean Stalk together with UDP Networking, to build a distributed, scalable, fault tolerant version of the website built in Project 1.a.
We have also implemented k-Resiliency - Extra Credit.

When the session is new:

- When the cookie == null, it indicates either the first time cookie is going to be generated or the cookie had expired.
- If cookie and the session have expired, then the session data is removed from the table.
- If the session is still active and cookie has expired, then the session is garbage collected.

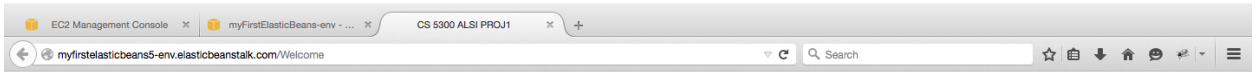
A session object is created and its details such as version number, expiry time are written into the session table. The server to which the request is sent becomes the primary server and the backup server is chosen by looping over primary's view table. This session is added locally into the primary's session table. The primary then attempts to store the session in backups. In the very first run, backup servers are all SERVER_ID_NULLs and so the session is replicated only on the primary. However, if it is a new session but not the first run, then there might be some servers up and running, whose information could have got into this primary's view table through gossip. In that case, the primary attempts to find backups by looping over its view table, and sending RPC session_write requests. Whichever server responds, the primary adds it to it's list of backups. Whichever server doesn't respond, the primary sets its status to DOWN in its ViewTable. The cookie is updated with all the values related to the session. And cookie value contains the list of backups that primary built up.

When the session already exists:

Based on the session ID present in the Cookie, its details are fetched from the session table. If the request is sent to the primary or backup server, it handles the request immediately by looking within its session table. If it is sent to some other server, then based on the location metadata<primary server, backup servers>, the request is now directed to the primary and backup servers(mentioned as a part of location metadata).
Whichever responds, we take the corresponding session values, and the value of the cookie is updated within the session table to contain the new values. In all cases, if the primary and backup servers do not send a response, the cookie is invalidated.

Our UI Behavior:

Condition 0: On page load, a new session is created and the following page is displayed:



Hello User!

Replace
Refresh
Logout

Cookie Value SessionID_VersionNum_locationmetadata: 1-54.186.118.199_0_54.186.118.199,54.187.216.128,SERVER_ID_NULL,SERVER_ID_NULL

Session Expiration time : Wed Apr 01 02:53:28 UTC 2015

Session Discard time : Wed Apr 01 02:53:28 UTC 2015

My Server ID : 54.186.118.199

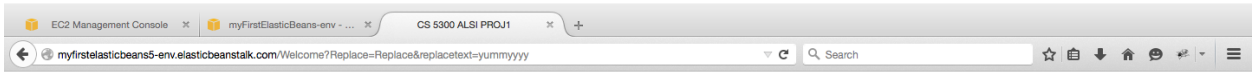
Fetched session from : 54.186.118.199

My View Table:

54.186.118.199	UP	1427856499780
54.187.216.128	UP	1427856452443

Session Table:
1-54.186.118.199 0 Hello User! Wednesday, Apr 01, 2015 02:53:28 AM

Condition 1: When ***Replace*** button is clicked, if cookie is not null and session is active, the session data is fetched if server_local who received the request is same as primary/backup servers in cookie value. Otherwise, server_local issues RPC session_read to primary and backup servers in cookie value. The fetched session's message is updated with new value entered in replace text box, if any. If no new message, the expiration time, version number is incremented. The values get updated in backup server also if any.



yummyyyy

Replace
Refresh
Logout

Cookie Value SessionID_VersionNum_locationmetadata: 1-54.186.118.199_1_54.187.48.47,54.186.118.199,SERVER_ID_NULL,SERVER_ID_NULL

Session Expiration time : Wed Apr 01 02:53:57 UTC 2015

Session Discard time : Wed Apr 01 02:54:02 UTC 2015

My Server ID : 54.187.48.47

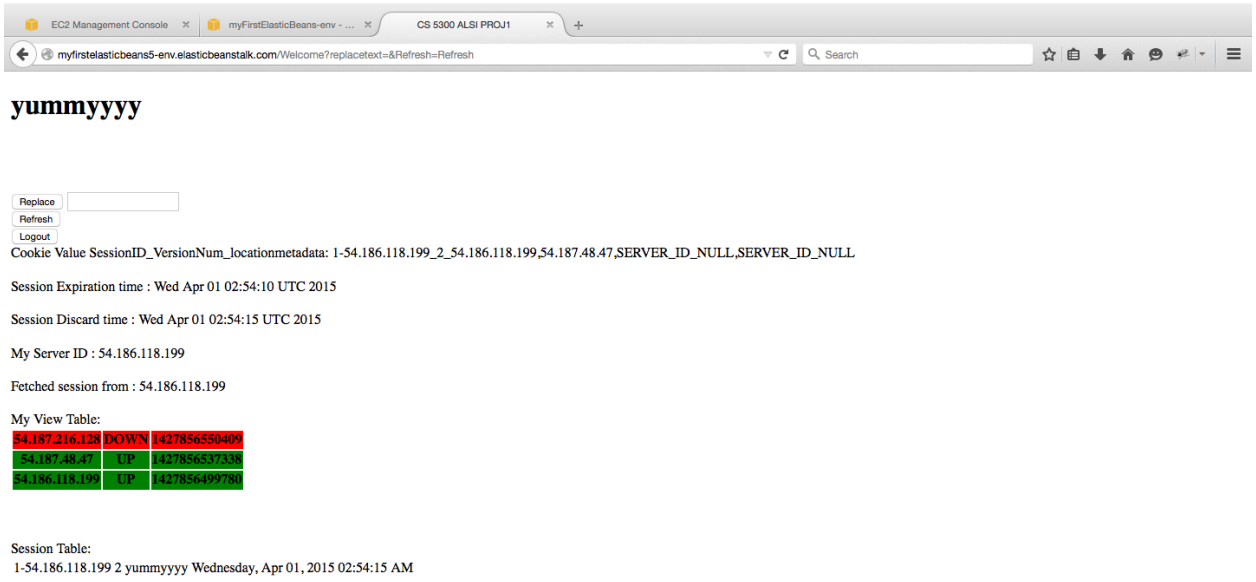
Fetched session from : 54.186.118.199

My View Table:

54.187.216.128	DOWN	1427856537336
54.187.48.47	UP	1427856533758
54.186.118.199	UP	1427856499780

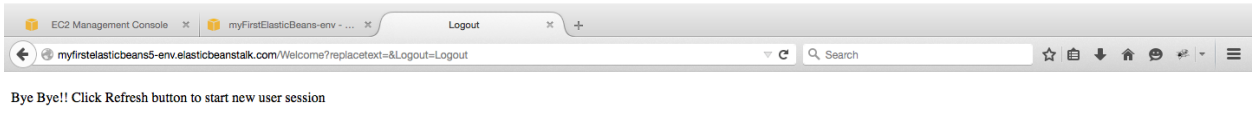
Session Table:
1-54.186.118.199 1 yummyyyy Wednesday, Apr 01, 2015 02:54:02 AM

Condition 2: When ***Refresh*** button is clicked, session data is fetched in the same manner as mentioned above in Condition Replace. The fetched session's timestamp is updated and the session is stored with replication. The values get updated in the backupserver also.

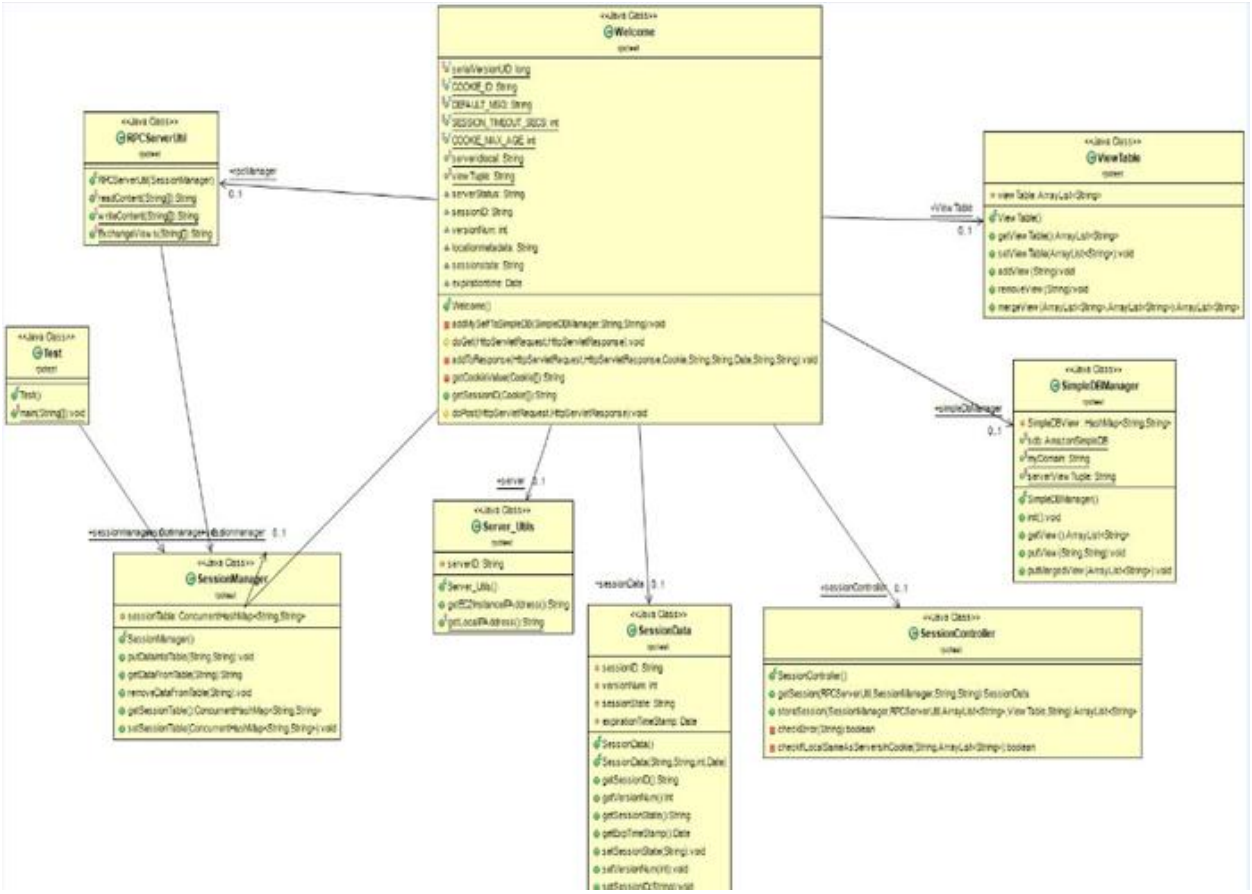


In the View Table, Red rows represent the servers which are Down and the green rows represent the up servers.

Condition 3: On ***Logout***, cookie value is set to null, and cookie's max age parameter is set to 0. The cookie is added to response, and a new display page "Logout.jsp" is displayed with the message **"Bye Bye!! Click Refresh button to start new user session"** .



The flow of code:



Elastic BeanStalk Setup Procedure:

Select the EBS ->Select the Platform (Tomcat)-> Launch Now -> Create New Environment -> Set up Environment type -> Load Balancing -> Upload the war file -> Set the No. of Instances.(Maximum Instances can be set to 4)-> New Environment is created .
Select Configuration in EBS -> Under Scaling -> Choose the Maximum and Minimum Instances -> A Default Environment URL will be shown. Add our Welcome to the URL and deploy it.
Add a security group to all instances which allows UDP traffic.

To Kill a Instance: Under EC2, select the Instance. Right Click on it and stop the instance.

EBS replaces the failed instance and that it registers itself with the membership gossip protocol : There can be maximum 4 instances running. When one instance is killed, the remaining instance under the same security group registers itself and start running.

How to run the project:

Upload the war file in EBS and follow the EBS Setup procedure. Then add the /welcome to the default environment URL and deploy it.

For Example: Default-Environment-psmi72echi.elasticbeanstalk.com/welcome

Format of Cookies:

Cookies are formed by the triplet SessionID, version and location metadata.

- SessionID is given by:

  SessionID = < session_number, SvrIDlocal>
- Version of the session is the version number of the session which is a part of the request.
- Location metadata is given by:

  Location metadata= <primary Server IP, Backup server IP>

The cookie is displayed with the cookie delimiter '_' to split the attributes of the cookie.
Cookie Format = <session_number-SvrIDlocal_Version_primaryServerIP,Backup server IP,..>

RPC Communication:

Format of RPC messages:

General format:
<call id@operation code@packetData>

<u>For a session read:</u>

Read session occurs for every button press, except Logout.
Read is issued by sending packet of the format :
callid@Utils.SESSION_READ_OPCODE@sessionID.

The server which receives this read session request, tries to find the corresponding session values for the sessionID. When an entry is not found in the hash table, only the callID is returned as response. Otherwise, callID along with details of session(message,version number, expiry time) are sent as response.
Response format : <callid@sessionid#versionnum#sessionstate#expirationtime>
Network Delimiter- '@'
the session data delimiter- '#' are used to split the attributes.

For a session write:

Write session happens every time the session needs to be replicated - for a new session, on button press of Replace and Refresh.

The request being sent consists of <call id@Utils.SESSION_WRITE_OPCODE@ sessionID#version number#message#expiry time>. On a successful write, the server which received the session write request, sends response as:

<callid@>

Exchange Views:

Exchange views is nothing but gossiping with other up servers. Whenever two servers interact, they exchange their views and then merge locally. In our exchange views, Input is a view encoded in string format. It consists of callid@opcode@viewstring. @ represents the Network Delimiter. When the two views are merged, the response is given by callid @ Mergeviewlist. The servers can exchange views until it reaches 1/view size.

Gossip:

A separate gossip thread starts to run when servlet is initialized. It sleeps for GOSSIP_SECS (60 seconds in our design) before deciding to gossip with a server or SimpleDB.

To gossip, the server executing the thread loops over its view table and chooses a server. If it chooses itself, it gossips with SimpleDB --exchanges its view with SimpleDB. Otherwise, it issues RPC ExchangeViews call to the server it chose and gossips with it. After gossiping, the server updates its view table with the merged view list.

This way, the probability of choosing a server is 1/view_size.

Group Membership and Simple DB:

Row Locking:

SimpleDB does not support row locking. Thus, two servers choosing to gossip with SimpleDB concurrently can suffer a race. We have **not** used "conditional store" because of eventual consistency. If two servers interact with simpleDB at the same time, views get overwritten and when the servers gossip with each other, while exchanging views, eventually consistency will be reached. So we are not using conditional store.

Session Timeout Implementation:

Sessions are timed out by setting cookie's MaxAge parameter to SESSION_TIMEOUT_SECS. In our design, SESSION_TIMEOUT_SECS is 5 minutes or 300 seconds.

To ensure that the session persists till its "discard_time", we calculate discard time separately for every session as :

discardTime = now + expirationTime + Delta(5 seconds in our design).

We set session data's expiration time to discard time so that while storing the session data in the session table, it is stored with its discardTime, and the session will persist until at least discardTime. This is displayed on the webpage as Session Discard Time.

Garbage Collection:

We have a separate cleanup thread that starts to run as soon as servlet begins. This cleanup thread sleeps for 305 seconds in our design. It then begins to remove sessions whose expiration time(set to discard time while storing the session) is less that current time. This way, it is guaranteed that the session is alive till its discard time.

Our Design:

The classes of our source file is explained below.

Welcome.jsp:

The JSP file is used to display the webpage with all the buttons such as Replace, Refresh, Logout and other attributes such as Cookie value, Message, Session Expiration Time, Session Discard time , View Table with list of Up and Down servers, and Session Table.

SessionData.java:

The SessionData contains the session state which is a default msg, session id,version number, expiration timestamp and the related getter and setter methods to obtain these values.

SessionManager.java:

The concurrent HashMap sessionTable is created. Sessiondata and session id can be put, get or removed from the table. Each server's session table is managed here.

SessionController.java:

The Session Controller consists of the get session and store session.

GetSession:

The session data is obtained from table if replicated locally, otherwise session is read through RPC to the servers mentioned in the cookievalue.

If local server id is same as the server id in the cookie, then the session data is constructed. If not a read session to all the resilient servers is sent. The session read consists of callID, OPcode and sessionID. The session data is constructed from the value returned by the RPC client.

StoreSession:

The list of backupServers are fetched from the local viewtable. The sessionID of the current session is fetched from the sessionManager. Based on the sessionID, other details of the session which is stored in the hashMap are retrieved. The packet that has to be written to all the backupServers is constructed and this packet becomes a part of the session_write. A session write is performed only if the Server_ID is not null i.e there exists a valid list of backup servers and the server which has been updated is added to another list if there is no error in session_write.

RPCClient.java:

public RPCClient()	
public String ForwardtoServer(String ServerIP, String data)	Returns the response packet to the server based on the parameters passed and the operation to be performed.

RPCServerStarter.java:

Starts the thread RPCServer.java

RPCServer.java:

public RPCServer()	Initializes a new DatagramSocket
void run()	Receives the Datagram packet from the client, calls takeAction() to perform the required operation and responds with a Datagram packet
byte[] takeAction(String[] dataContent)	Performs required operation based on the operation code by calling the functions present in the RPCServer_Utils class and returns a byte array

RPCServer_Utils.java:

The necessary functions such as readcontents, writecontents and the exchange views actions that RPC server has to perform are defined in PCServer_Utils.java.

Server_Utils.java:

getEC2InstanceIPAddress()	Helps fetch the EC2 instance IP Address
getLocalIPAddress()	To get the machine's IP Address.

ViewTable.java:

setViewTable(Set**<**String**>**viewTable**)**	Sets the current view table
addView(String viewTuple**)**	Adds the new tuple to the view
removeView(String viewString**)**	Removes the view from the view table.
mergeView(String view1**,** String view2**)**	Merges the 2 views to have the updated values in both the views.

Utils.java:

The Utils class contains all the util functions such as parseReceivePacket(), parseSessionIDFromString(),parseVersionFromString(), parseSessionStateFromString(),parseCookieandGetSessionID(), checkIfversionsame(), checkSessionExpired(),ConstructFirstTimeCookie() which are called in other classes.

SimpleDBManager.java:

public void init()	Create Domain with AWS Credentials
public ArrayList<String>getView()	Adds the views to the view list and prints the

	item details such as Name and Value.
public void putView(String viewTuple, String IP)	In the putView, get attributes and replace attributes is done
public void putMergedView(ArrayList<String> set)	MergedView consists of ServerID/Status/TimeServerID/Status/Time

Gossip.java:

In Gossip class, we loop over 1/view size. When the IP fetched is the own Primary Server IP, we gossip with SimpleDB. If its a different IP, gossip among the servers.

gossipWithSimpleDB()	When server boots up register with simpleDB. 1.Gets view from SimpleDB. 2.Merges my view Table and view list from simple table. 3.Updates view table with merged view. 4.Merged view is updated back to SimpleDB.
gossipWithServer(String server)	Two servers gossips with each other and exchange their views. Constructs session data from value returned by the RPC client.

Welcome.java:

doGet(HttpServletRequest request**,** HttpServletResponse response**)throws** ServletException**,** IOException	Cookies are fetched from request The sessionID is fetched from the data within cookie If the sessionID is null, new session state is created by setting message to DEFAULT_MSG, expiration time to a value + delta, version number and location metadata and putting the sessionID into the session table Based on the Users choice of button necessary action is taken. If action= replace, SessionData is updated with new values. Else if action=refresh

	If the sessionID is present in the session table then version number and timestamp of the session are updated. Else if action = logout If the session is present in the table, expiration time and maxAge are set to null and the data related to that session is removed from the table.
privateString getCookieValue(Cookie[] cookies)	Returns the myCookieValue of the corresponding CookieID.
getSessionID(Cookie[] cookies)	Returns the sessionID of the request session present in the cookie

The discard time is calculated by Expiration Time + 5 secs in the welcome.java

CleanUpStarter.java:

It is a thread that runs over the Concurrent HashMap.

removeFromMap(ConcurrentHashMap**<**String**,** SessionData**>** sessionControlMap**)**	Removes all the sessions that are expired by comparing the expiry time which is stored in the session table with the current time.

Extra Credit: k -Resilient System

In our Utils.java class, we have defined the variable "**Resiliency Factor**" or **k** as 3.At compile time we can change the resiliency factor according to the number of backup servers needed. We are choosing backup servers until the Resiliency Factor is achieved. Similar to 1 Resilient System, when the backup server is chosen, the backup server ID's are added to the array list. If there is no back up server then SERVER_ID_NULL is displayed. This way the system is fault tolerant up to Resiliency Factor.