CS 5300
Project 1b Large Distributed Web System
Team member: Xiaoan Yan(xy254) Hengyi Huang(hh548) Yukun Jiang(yj297)

NOTE: Please replace the credentials in the <proejct directory>/src/AwsCredentials in order to test the project. After deployed on Beanstalk, the URL is in the form of

"<default-bean-stalk host addess>/MyServletPath",

for example,

"sqdkfl.elasticbeanstalk.com/MyServletPath".

Due to the congestion of network traffic, please do not click too fast. Click every two seconds maybe.

****
Running Instruction
****

Eclipse:
Unzip the project, then import the project as AWS web project in eclipse EE. To initialize bootstrap view, run InitBootStrap as java application.
Last , export the whole project in .war format.
Elastic Beanstalk:
Since view size in this project is set to be 5, we suggest to activate at least 5 instances. Upload and deploy this project on to Beanstalk. **Final, you can examine all information required in the given hosting URL followed by "/MyServletPath**".

****
How to Test The Project
****

The first time "replace" is clicked, a new session is created and users are supposed to observe that a session is stored locally and another one stored remotely in a different server. All sessions expire in 10 seconds once created or updated.

When users click on "refresh", users might notice that "Primary Server ID" and "Backup Server ID" are two different non-local IP addresses if the requested session is not stored locally.

When users click on "logout", both "Primary Server ID" and "Backup Server ID" are "NULL Server ID" since all existing sessions expire instantly.

By default, the view size is five. Therefore, users should expect five servers in the view if more than five EC2 instances are active and running simultaneously.

****
Implementation Architecture
****

MyServletContextListener serves as initialization program to control all threads in this project. SimpleDB is used to store bootstrap view and MyGossip implements gossip views exchange algorithm.

****
Cookie Value
****

The cookie values are in the following format:

< session_ID , version, primary_server_ID, backup_server_ID >

Different attributes are separated by an underscore. Everytime a new session is created, the version number is initialized as zero. Whenever a “refresh” request gets processed, the version number is incremented by 1 and the discard time gets extended.

Each session_ID is a local server IP address appended with automatically incrementing index. The reason for choosing this format is that different server IP addresses distinguish requests from different different servers from each other. The auto-incrementing index makes sure that later requests do not collide into early ones. Primary_server_ID and backup_server_ID are IP addresses of primary server and backup server.

****
Session Value
****

The construction format of session value is as follows:
<session_ID, version, session_value, expiration_time>
Different arguments in session value is separated by symbol “#”.

The meaning of each part of arguments is described in Cookie Value part.

****
RPC Client
****

A RPC client is initiated for both main and gossiping threads. The three main interfaces of a RPC client are “sessionRead”, “sessionWrite” and “getView”. The major tasks for RPC clients are to read/write session value from/to other servers as well as return the view of a server given his IP address.

Both “sessionRead” and “sessionWrite” take a destination IP and a session object as input. Then it serialize cussion information and send it over to primary/backup server using UDP protocol.

****
RPC Server
****

RPC Server is implemented as a thread to frequently listen to incoming request or message. Firstly it parses the received message and determine which operation to execute depending on the operation code.

****
SessionWrite
****

Session Write uses PRC call to write session table stored in specified servers. It’s used in both Replace and Logout functionality.

****
View
****

Each server creates and maintains its own view. In our project, each view maintain 5 ip addresses without itself. The view could achieve union, shrink, print self and other functionality.

****
Gossip
****

To better distribute views, we implements gossip protocol. The gossip protocol run as a thread and store/update the simple DB. The rule is simply described here:
1. each server starts without any view, but they can get boostrap view by communicating with simple DB. At meanwhile, simple DB’s view will be updated.
2. Each serve could request the views from others who in its view. But it only take the fresh view that it doesn’t have in this own view. The size of the view is always limited at 5. At this request, the server been requested could also get the ip address from the server send the request.
3. The ip address of this server shouldn’t appear its view.
The thread is running at a reasonable speed to fast evenly distribute the ip addresses in views in order to avoid null view.

****

Simple DB
****
Simple DB is used in this project to store backup views for any server that just starts.

****
Garbage Collection
****
The garbage collection mechanism is implemented in class MyServlet. The principle is that each time the user operates on the web page (i.e. clicking on one button), the method cleanTable will be called, and all sessions in the session table will be checked. If the expiration time stored in a session value is earlier than the present time, the session is treated out of date and is removed from the table.

Besides the garbage collection mechanism as described above, we also implements garbage collection as thread to regularly clean expired cookies. After a few try, we decide to choose above mechanism instead of thread mechanism to better control server usage which could be critical in high performance distributed web application.