

# Project 5 report

Shaoyi Huang

Yangyu Shu

## Task 1

### How did you use connection pooling?

We opened the connection pooling in context.xml, so that each time when we need to access the database, we are reusing a connection from the pool, so it saves us time to open new connection.

Firstly, we defined datasource in context.xml about how many connections can be exist in connection pooling (this part defined on context.xml line 10), so that every time we finished to use a connection, it will go back to the connection for future use.

Directory: cs122b-winter19-team-92/project1/WebContent/META-INF/context.xml : line 5

```
5      <!-- Defines a Data Source Connecting to localhost moviedb-->
6      <Resource name="jdbc/moviedb"
7          auth="Container"
8          driverClassName="com.mysql.jdbc.Driver"
9          type="javax.sql.DataSource"
10         maxTotal="100" maxIdle="30" maxWaitMillis="10000"
11         username="mytestuser"
12         password="mypassword"
13         url="jdbc:mysql://localhost:3306/moviedb?
14         useSSL=false&
15         autoReconnect=true&
16         allowMultiQueries=true&
17         useUnicode=true&
18         useJDBCCompliantTimezoneShift=true&
19         useLegacyDatetimeCode=false&
20         serverTimezone=UTC
21         &cachePrepStmts=true"/>
--
```

Then we defined db\_source class (since connection pooling setup code are repeated many times in all servlets) that determine which data source we gonna use in the each servlet, and handles different cases (like single instance version or scaled-version).

Directory: cs122b-winter19-team-92/project1/src/db\_source.java (whole file for reuse purpose)

```
20 public class db_source {
21
22     Context env;
23     String url;
24
25     public db_source(String url1) throws NamingException {
26
27         Context initCtx = new InitialContext();
28         url=url1;
29         //checking for env
30         env = (Context) initCtx.lookup("java:comp/env");
31     }
32
33
34
35     public DataSource multiple() throws NamingException {
36
37
38
39         System.out.println("ds is loadbalancer one");
40         DataSource ds = (DataSource) env.lookup("jdbc/master_inst");
41
42
43         return ds;
44     }
45 }
```

Public db\_source(String url1) is a constructor that stored current url and context to use.  
multiple() is the datasource return function for multiple instance to use

```

46     public DataSource single() throws NamingException {
47
48         //decide which ds to use
49
50         System.out.println("ds is single one");
51         DataSource ds = (DataSource) env.lookup("jdbc/moviedb");
52
53         return ds;
54
55     }
56
57     public DataSource getSource() throws NamingException {
58         if(url.contains("172.31")) {
59             return multiple();
60         }
61         else {
62             return single();
63         }
64     }
65 }
66 }

```

**single()** is the datasource return function for single instance to use

**getSource()** is the function to return a correct source to use depend on different cases.

Here we use url to determine which source to use, since only for loadbalancer case, the url will contain "172.31" will only be private IP of AWS so use it to determine multiple instances or single instance.

Finally in MovieServlet (for example), we initialized a db\_source class by request url to determine which case this is, and use **db\_source.getSource()** to get connection source for future request queries which will keep reusing the the connections in the connection pool we built.

cs122b-winter19-team-92/project1/src/MovieServlet.java : line 57-59

```

52     try {
53         //without connection pooling part
54         // Get a connection from dataSource
55         //Connection dbcon = dataSource.getConnection();
56
57         db_source dbs=new db_source(request.getRequestURL().toString());
58         DataSource ds=dbs.getSource();
59         Connection dbcon = ds.getConnection();
60
61         // Declare our statement
62
63
64         if (mode.equals("browse")) {

```

cs122b-winter19-team-92/project1/src/SingleStarServlet.java (line 43-46)

```

41
42         try {
43             db_source dbs=new db_source(request.getRequestURL().toString());
44             DataSource ds=dbs.getSource();
45             Connection dbcon = ds.getConnection();
46

```

Etc. For all of servlet we applied similar steps for connection pooling.

## How did you use Prepared Statements?

We defined in similar context.xml of data sources that allow prepared statement cache setting to true(in line 21) first.

We made the prepared statements if user is using searching (there are other modes like browse) then we update the statement with parameter passed in the url

cs122b-winter19-team-92/project1/src/MovieServlet.java : line 333 and line 205

Making prepared statements

```

333     public String updateBySearch(HttpServletRequest request) {
334         String query="select m.id,title, year, director, rating\n"
335             +"from movies m, ratings r, stars s, stars_in_movies sm\n"
336             +"WHERE m.id=r.movieId and sm.movieId=m.id and sm.starId=s.id";
337         String title = request.getParameter("title");
338         String year = request.getParameter("year");
339         String director = request.getParameter("director");
340         String star = request.getParameter("stars");
341         if(title!=" " &&!title.equals("null")) {
342
343             //or edth(title, ?, ?))
344
345             query+=" and ((MATCH (title) AGAINST ( ? IN BOOLEAN MODE)) or title like ? ) ";
346         }
347
348         if(year!=" " &&!year.equals("null")) {
349             query+=" and year=? \n";
350         }
351         if (director!=" " &&!director.equals("null")){
352             query+=" and director LIKE ? \n";
353         }
354         if(star!=" " &&!star.equals("null")) {
355             query+=" and s.name LIKE ? \n";
356         }
357
358
359
373         query+="Group by m.id,title, year, director, rating\n";
374
375         return query;
376
377     }

```

Like for line 345 , line 349, line 352, line 355 we built the prepared statement.

## Binding user-inputs

```

205         else {
206             String title = request.getParameter("title");
207             String year = request.getParameter("year");
208             String director = request.getParameter("director");
209             String star = request.getParameter("stars");
210             if(title!=" " &&!title.equals("null")) {
211                 String query="";
212                 String [] arrOfStr = title.split(" ");
213                 for (String i:arrOfStr) {
214                     query+=" "+i+"* ";
215                 }
216                 result.setString(index++, query);
217                 //title like ? or edth(title, ?, ?))
218                 String likeOperator="%" + title + "%";
219                 result.setString(index++, likeOperator);
220                 //result.setString(index++, title);
221                 //result.setInt(index++, (int)Math.round(0.4*title.length()) );
222             }

```

```

224         if(year!="&&!year.equals("null")) {
225             result.setInt(index++, Integer.parseInt(year));
226         }
227         if (director!="&&!director.equals("null")){
228             result.setString(index++, "%"+director+"%");
229         }
230         if(star!="&&!star.equals("null")) {
231             result.setString(index++, "%"+star+"%");
232         }

```

· For codes after we gather user request from front end, and put it into our prepared statement. Like Line 219, line 225, line 228 and line 231.

## Task 2

Address of AWS and Google instances

AWS:

Instance1(load balancer):18.188.100.51

Instance2(master):18.217.51.132

Instance3(slave):18.188.151.135

Google:

Instance: <http://35.196.87.125:80>

(if you want to access project, enter /Project1/login.html after public ip)

Eg. <http://35.196.87.125:80/Project1/login.html>

They are all accessible and open already.

## Explain how connection pooling works with two backend SQL (in your code)?

Similarly, we defined a jdbc replicationDriver in context.xml first and we enable the connection pooling setting on (line 27). We put two replication urls in URL(line 30), the

first one is the master instance url and the second one is slave instance url. We also enable round robin balance (line 46) to true and SetReadOnly(line 48) to true since round robin only works on read only status, which work as a load balancer that evenly split task to two database connection pool. For other connection pooling process, it act similarly as the case above.

Directory cs122b-winter19-team-92/project1/WebContent/META-INF/context.xml : line 23

```
23     <Resource name="jdbc/master_inst"
24         auth="Container"
25         driverClassName="com.mysql.jdbc.Driver"
26         type="javax.sql.DataSource"
27         maxTotal="1000" maxIdle="30" maxWaitMillis="10000"
28         username="mytestuser"
29         password="mypassword"
30         url="jdbc:mysql:replication://172.31.45.222:3306,172.31.39.120:3306/moviedb?
31         useSSL=false&
32         autoReconnect=true
33         &
34         allowMultiQueries=true
35         &
36         useUnicode=true
37         &
38         useJDBCCompliantTimezoneShift=true
39         &
40         useLegacyDatetimeCode=false
41         &
42         serverTimezone=UTC
43         &
44         cachePrepStmts=true
45         &
46         roundRobinLoadBalance=true
47         &
48         setReadOnly=true"/>
```

Then in MovieServlet (for example), we initialized a db\_source class by request url to determine which case this is, and use **db\_source.getSource()** to get connection source for future request queries which will keep reusing the the connections in the connection pools we built here. The property of each connection pool is similar to above, but round robin balance the requests to each pool.

cs122b-winter19-team-92/project1/src/MovieServlet.java : line 57-59

```

52         try {
53             //without connection pooling part
54             // Get a connection from dataSource
55             //Connection dbcon = dataSource.getConnection();
56
57             db_source dbs=new db_source(request.getRequestURL().toString());
58             DataSource ds=dbs.getSource();
59             Connection dbcon = ds.getConnection();
60
61             // Declare our statement
62
63
64             if (mode.equals("browse")) {

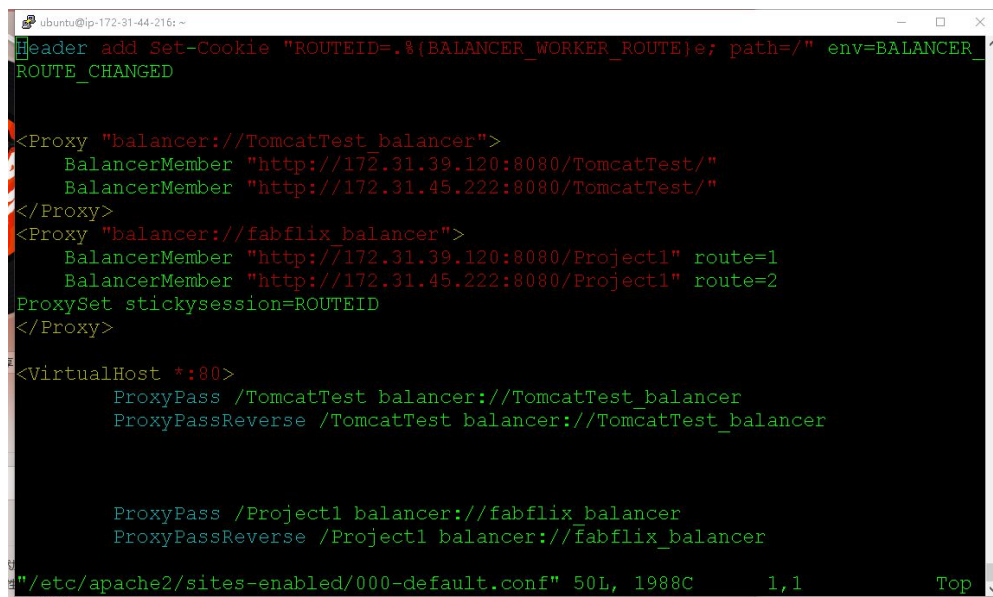
```

Servlets etc.

## How read/write requests were routed?

### Early load balancer

After request send to load balancer, instance 1 with port 80, it will work as a proxy that redirect request evenly to two different instances, for both read and write parts. We used apache 2 as a load balancer, since the Fabflix requires session, made a stick session for load balancer that, all of requests will be sent to the instance that they first time sent to. After, request sent to two instances, the read on write wil treated differently.



```

Header add Set-Cookie "ROUTEID=.%{BALANCER_WORKER_ROUTE}e; path=/" env=BALANCER_ROUTE_CHANGED

<Proxy "balancer://TomcatTest_balancer">
    BalancerMember "http://172.31.39.120:8080/TomcatTest/"
    BalancerMember "http://172.31.45.222:8080/TomcatTest/"
</Proxy>
<Proxy "balancer://fabflix_balancer">
    BalancerMember "http://172.31.39.120:8080/Project1" route=1
    BalancerMember "http://172.31.45.222:8080/Project1" route=2
ProxySet stickysession=ROUTEID
</Proxy>

<VirtualHost *:80>
    ProxyPass /TomcatTest balancer://TomcatTest_balancer
    ProxyPassReverse /TomcatTest balancer://TomcatTest_balancer

    ProxyPass /Project1 balancer://fabflix_balancer
    ProxyPassReverse /Project1 balancer://fabflix_balancer

```

### Read part



Since we already defined the context.xml above, in datasource “jdbc/master\_inst”, we put replication urls on **line 30** that can determine **master:the first url** and **slave: the second url**. After we enable round robin balance and setReadOnly to true, it will assign reading requests depend on work load of each database. Slave and master databases can be read by both instances(2 and 3).

Path: cs122b-winter19-team-92/project1/WebContent/META-INF/context.xml : line 23

```
23     <Resource name="jdbc/master_inst"
24         auth="Container"
25         driverClassName="com.mysql.jdbc.Driver"
26         type="javax.sql.DataSource"
27         maxTotal="1000" maxIdle="30" maxWaitMillis="10000"
28         username="mytestuser"
29         password="mypassword"
30         url="jdbc:mysql:replication://172.31.45.222:3306,172.31.39.120:3306/moviedb?
31         useSSL=false&
32         autoReconnect=true
33         &
34         allowMultiQueries=true
35         &
36         useUnicode=true
37         &
38         useJDBCCompliantTimezoneShift=true
39         &
40         useLegacyDatetimeCode=false
41         &
42         serverTimezone=UTC
43         &
44         cachePrepStmts=true
45         &
46         roundRobinLoadBalance=true
47         &
48         setReadOnly=true"/>
49
```

For servlet side it works no difference on read part, because we already define default setReadOnly to true. Indeed, every connection doesn't need to be changed.

cs122b-winter19-team-92/project1/src/MovieServlet.java : line 57-59

```
52     try {
53         //without connection pooling part
54         // Get a connection from dataSource
55         //Connection dbcon = dataSource.getConnection();
56
57         db_source dbs=new db_source(request.getRequestURL().toString());
58         DataSource ds=dbs.getSource();
59         Connection dbcon = ds.getConnection();
60
61         // Declare our statement
62
63
64         if (mode.equals("browse")) {
```

## Write Part

Nearly all of setting are similar to read above, but if readonly is true, it will send requests as load balancer that both master database and slave database can get the request, so before every time we want write something into databases that we will need to setReadOnly as false (line 70) then the write request only will send to the master database and the slave database will track the log and its position from master to copy the data just write.

For example in dashboard servlet we have “add Star” and “add movies” stored procedures in mysql which will write data to databases. So before execute the query, we set setReadOnly to false (line 70 and line 139). Then all of query will only send to master databases we defined in jdbc replication driver in Context.xml that already explained above.

cs122b-winter19-team-92/project1/src/\_dashboarServlet.java

```
67         db_source dbs=new db_source(request.getRequestURL().toString());
68         DataSource ds=dbs.getSource();
69         Connection conn = ds.getConnection();
70         conn.setReadOnly(false);
71         conn.setAutoCommit(false);

135         db_source dbs=new db_source(request.getRequestURL().toString());
136         DataSource ds=dbs.getSource();
137         Connection conn = ds.getConnection();
138         conn.setReadOnly(false);
139         conn.setAutoCommit(false);
140
```

### Task 3

Log file Directory: [cs122b-winter19-team-92/project1/logs](#)

The name for the log file follow the cases in the project requirement

#### Single-instance cases

1. Use HTTP, without using prepared statements, 10 threads in JMeter. [Log 1.1 txt](#)
2. Use HTTP, without using connection pooling, 10 threads in JMeter. [Log 1.2 txt](#)
3. Use HTTP, 1 thread in JMeter. [Log 1.3 txt](#)
4. Use HTTP, 10 threads in JMeter. [Log 1.4 txt](#)
5. Use HTTPS, 10 threads in JMeter. [Log 1.5 txt](#)

#### Scaled-version cases

1. Use HTTP, without using prepared statements, 10 threads in JMeter. [Log 2.1 txt](#)

2. Use HTTP, without using connection pooling, 10 threads in JMeter.[Log 2.2 txt](#)
3. Use HTTP, 1 thread in JMeter.[Log 2.3 txt](#)
4. Use HTTP, 10 threads in JMeter.[Log 2.4 txt](#)

HTML report Directory:[cs122b-winter19-team-92/project1/WebContent/jmeter\\_report.html](#)

Script Directory : [cs122b-winter19-team-92/project1/logs/parseTSTJ.py](#)

War file Directory: [cs122b-winter19-team-92/Project1.war](#)

Readme file Directory: [cs122b-winter19-team-92/Readme.md](#)

Script is written in python and it will go through the current directory of Script to check the file start from log and end with txt. We will parse a of data row by row, the first row is TJ and the second row is TS then we will written the result into a file called “data.txt” to current directory.