Task 1 - Connection Pooling and Prepared Statements

First, we created a load balancer with two backend servers (slave and master). The load balancer redirects connection requests to the two backend servers depending on the request. If it is a read requests, the connection randomly chooses between slave or master. If it is a write request, the connection only goes to the master server.

To do this, we configured the context.xml and web.xml files to add and map our resources (two backend servers). Then, in our Login.java servlet, we create the DataSource to our slave and master servers and store them into a session. Whenever we need to make a connection to one our servers, we simply synchronize the sessions and request the read/write DataSource that we have stored. All files related to connection pooling are located in the directory /project2/src.

Connection Pooling w/ Slave/Master:

- Login.java
  - Lines 59 - 69 creates the context and the datasource, and also stores the two data sources (read/slave and write/master) into a session

```
59          try {
60              // the following few lines are for connection pooling
61              Context initCtx = new InitialContext();
62              Context envCtx = (Context) initCtx.lookup("java:comp/env");
63              DataSource dsRead = (DataSource) envCtx.lookup("jdbc/moviedbread");
64              DataSource dsWrite = (DataSource) envCtx.lookup("jdbc/moviedbwrite");
65
66              Connection dbcon = dsWrite.getConnection();
67
68              session.setAttribute("dsread", dsRead);
69              session.setAttribute("dswrite", dsWrite);
70
```

- EmployeeLogin.java
  - Lines 68 - 78 creates the context and the datasource, and also stores the two data sources into a session. This is repeated as in Login.java because they should be two separate sessions. MovieSearch.java
  - Lines 138 - 150 picks between slave and master for read

```
68                              try {
69                                      // the following few lines are for connection pooling
70                                      Context initCtx = new InitialContext();
71                              Context envCtx = (Context) initCtx.lookup("java:comp/env");
72                              DataSource dsRead = (DataSource) envCtx.lookup("jdbc/moviedbread");
73                              DataSource dsWrite = (DataSource) envCtx.lookup("jdbc/moviedbwrite");
74
75                              Connection dbcon = dsWrite.getConnection();
76
77                              session.setAttribute("dsread", dsRead);
78                              session.setAttribute("dswrite", dsWrite);
```

- Autocomplete.java
  - Lines 88 - 99 picks between slave and master for read

```
88              synchronized(session) {
89                  // Since we are only reading, randomly connect to slave or master instance for read
90                  int pick = (int) (Math.random() % 2);
91                  if (pick == 0) {
92                      dbcon = ((DataSource) session.getAttribute("dsread")).getConnection();
93                      System.out.println("Reading from slave");
94                  }
95                  else {
96                      dbcon = ((DataSource) session.getAttribute("dswrite")).getConnection();
97                      System.out.println("Reading from master");
98                  }
99              }
```

- AddMovie.java
  - Lines 63 - 67 gets DataSource by session.getAttribute() for write request, adding a movie to the database

```
63          try {
64              synchronized(session) {
65                  dbcon = ((DataSource) session.getAttribute("dswrite")).getConnection();
66                  System.out.println("Writing to master");
67              }
```

- AddStar.java
  - Lines 53 - 57 creates a connection to the master server for writing, adding a star to the stars table

```
53                  try {
54                          synchronized(session) {
55                                  dbcon = ((DataSource) session.getAttribute("dswrite")).getConnection();
56                                  System.out.println("Writing to master");
57                  }
```

- Checkout.java
  - Lines 83 - 97 randomly picks between master and slave to read from, fetches data to display for cart

```
83              try {
84                      synchronized(session) {
85                              // Since we are only reading from database, can randomly choose between master or slave
86                              // A better implementation would to keep count of how connections each server has and
87                              // pick the one with less
88                              int pick = (int) (Math.random() % 2);
89                              if (pick == 0) {
90                                      dbcon = ((DataSource) session.getAttribute("dsread")).getConnection();
91                                      System.out.println("Reading from slave");
92                              }
93                              else {
94                                      dbcon = ((DataSource) session.getAttribute("dswrite")).getConnection();
95                                      System.out.println("Reading from master");
96                              }
97                      }
98
```

- Dashboard.java
  - Lines 48 - 61 picks between master and slave to read from, display meta data to employee user

```
47          try {
48                  synchronized(session) {
49                          // Since we are only reading from database, can randomly choose between master or slave
50                          // A better implementation would to keep count of how connections each server has and
51                          // pick the one with less
52                          int pick = (int) (Math.random() % 2);
53                          if (pick == 0) {
54                                  dbcon = ((DataSource) session.getAttribute("dsread")).getConnection();
55                                  System.out.println("Reading from slave");
56                          }
57                          else {
58                                  dbcon = ((DataSource) session.getAttribute("dswrite")).getConnection();
59                                  System.out.println("Reading from master");
60                          }
61                  }
62
```

- SingleStar.java
  - Lines 67 - 81 picks between master and slave to read from, display information about a star

```
67          try {
68                  synchronized(session) {
69                          // Since we are only reading from database, can randomly choose between master or slave
70                          // A better implementation would to keep count of how connections each server has and
71                          // pick the one with less
72                          int pick = (int) (Math.random() % 2);
73                          if (pick == 0) {
74                                  dbcon = ((DataSource) session.getAttribute("dsread")).getConnection();
75                                  System.out.println("Reading from slave");
76                          }
77                          else {
78                                  dbcon = ((DataSource) session.getAttribute("dswrite")).getConnection();
79                                  System.out.println("Reading from master");
80                          }
81                  }
```

For each type of search used in our website we used Prepared Statements. This includes the search/navigation bar which includes autocomplete and fuzzy search, the search function based on four fields, and the browse by genre/name feature. In each of our java servlets that handle these search queries, we first created the query string using “?” in place of our parameters. Then we store that query string into a PreparedStatement object. We add our parameters using the PreparedStatement.setString() or PreparedStatement.setInt() function to add our parameters and then execute the query.

Prepared Statements:

- MovieSearch.java
  - Lines 54 - 129 are setting up the query string
  - Lines 155 - 202 are adding the parameters to the prepared statement and executing the query

```
62          if (!movieTitle.equals("")) {
63              if (type.equals("search")) {
64                  //queryMovieSearch += " WHERE title LIKE '%" + movieTitle + "%'";
65                  queryMovieSearch += " WHERE title LIKE ?";
66              }
67              else if (type.equals("browse")) {
68                  if (movieTitle.equals("1")) {
69                      //queryMovieSearch += " WHERE title regexp '^[0-9]+' ";
70                      queryMovieSearch += " WHERE title regexp '^[0-9]+'";
71                  }
72                  else {
73                      //queryMovieSearch += " WHERE title LIKE '" + movieTitle + "%'";
74                      queryMovieSearch += " WHERE title LIKE ?";
75
76                  }
77              }
78              else if (type.equals("searchbar")) {
79                  /*
80                  queryMovieSearch += " WHERE MATCH (title) AGAINST ('";
81                  String[] titleWords = movieTitle.split(" ");
82                  for (int i = 0; i < titleWords.length; i++) {
83                      queryMovieSearch += "+" + titleWords[i] + "*";
84                  }
85                  queryMovieSearch += "' IN BOOLEAN MODE) OR edth(lower(title), '" + movieTitle + "', 3)";
86                  */
87                  queryMovieSearch += " WHERE MATCH (title) AGAINST (";
88                  String[] titleWords = movieTitle.split(" ");
89                  for (int i = 0; i < titleWords.length; i++) {
90                      queryMovieSearch += "? ";
91                  }
92                  queryMovieSearch += "IN BOOLEAN MODE) OR edth(lower(title), ?, 3)";
93
94              }
95              first = false;
96          }
97          if (!movieYear.equals("")) {
98              if (!first) { queryMovieSearch += " AND"; }
99              else { queryMovieSearch += " WHERE"; }
00
01              //auervMovieSearch += String.format(" vear = '%s'". movieYear):
```

```
100
101             //queryMovieSearch += String.format(" year = '%s'", movieYear);
102             queryMovieSearch += " year = ?";
103
104             first = false;
105         }
106         if (!movieDir.equals("")) {
107             if (!first) { queryMovieSearch += " AND"; }
108             else { queryMovieSearch += " WHERE"; }
109
110             //queryMovieSearch += " director LIKE '%" + movieDir + "%'";
111             queryMovieSearch += " director LIKE ?";
112
113             first = false;
114         }
115
116         queryMovieSearch += " GROUP BY movies.id";
117
118         if (!movieActor.equals("")) {
119             //queryMovieSearch += " HAVING actors LIKE '%" + movieActor + "%'";
120             queryMovieSearch += " HAVING actors LIKE ?";
121         }
122         if (!movieActor.equals("") && !movieGenre.equals("")) {
123             //queryMovieSearch += " AND genres LIKE '%" + movieGenre + "%'";
124             queryMovieSearch += " AND genres LIKE ?";
125         }
126         else if (movieActor.equals("") && !movieGenre.equals("")) {
127             //queryMovieSearch += " HAVING genres LIKE '%" + movieGenre + "%'";
128             queryMovieSearch += " HAVING genres LIKE ?";
129         }
```

```
154             //Statement statement = dbcon.createStatement();
155             int paramIndex = 1;
156             PreparedStatement statement = dbcon.prepareStatement(queryMovieSearch);
157             if (!movieTitle.equals("")) {
158                 if (type.equals("browse")) {
159                     if (movieTitle.equals("1")) {
160                         ;
161                     }
162                     else {
163                         statement.setString(paramIndex, movieTitle + "%");
164                     }
165                 }
166                 else if (type.equals("searchbar")) {
167                     String[] titleWords = movieTitle.split(" ");
168                     for (String word : titleWords) {
169                         statement.setString(paramIndex, "+" + word + "*");
170                         paramIndex++;
171                     }
172                     statement.setString(paramIndex, movieTitle);
173                 }
174                 else if (type.equals("search")){
175                     statement.setString(paramIndex, movieTitle);
176                     paramIndex++;
177                 }
178             }
179             if (!movieYear.equals("")) {
180                 statement.setString(paramIndex, movieYear);
181                 paramIndex++;
182             }
183             if (!movieDir.equals("")) {
184                 statement.setString(paramIndex, "%" + movieDir + "%");
185                 paramIndex++;
186             }
187             if (!movieActor.equals("")) {
188                 statement.setString(paramIndex, "%" + movieActor + "%");
189                 paramIndex++;
190             }
191             if (!movieActor.equals("") && !movieGenre.equals("")) {
192                 statement.setString(paramIndex, "%" + movieGenre + "%");
193                 paramIndex++;
194             }
195             else if (movieActor.equals("") && !movieGenre.equals("")) {
```

```
193                 paramIndex++;
194             }
195             else if (movieActor.equals("") && !movieGenre.equals("")) {
196                 statement.setString(paramIndex, "%" + movieGenre + "%");
197                 paramIndex++;
198             }
199
200             // Execute query
201             //ResultSet rs = statement.executeQuery(queryMovieSearch);
202             ResultSet rs = statement.executeQuery();
```

- Autocomplete.java
  - Lines 61 - 73 are setting up the query string
  - Lines 108 - 127 are adding the parameters to the prepared statement and executing the query

```
61          String queryMovie = "SELECT * FROM movies WHERE MATCH (title) AGAINST (";
62          String queryStar = "SELECT * FROM stars WHERE MATCH (name) AGAINST (";
63
64          for (int i = 0; i < words.length; i++) {
65              //queryMovie += "+" + words[i] + "*";
66              //queryStar += "+" + words[i] + "*";
67              queryMovie += "? ";
68              queryStar += "? ";
69          }
70          //queryMovie += "' IN BOOLEAN MODE) OR edth(lower(title), '" + query + "', 3)";
71          //queryStar += "' IN BOOLEAN MODE) OR edth(lower(name), '" + query + "', 3)";
72          queryMovie += "IN BOOLEAN MODE) OR edth(lower(title), ?, 3)";
73          queryStar += "IN BOOLEAN MODE) OR edth(lower(name), ?, 3)";
74
```

```
107             // Declare a new statement
108             Statement statement = dbcon.createStatement();
109             PreparedStatement statementMovie = dbcon.prepareStatement(queryMovie);
110             PreparedStatement statementStar = dbcon.prepareStatement(queryStar);
111
112             int paramIndexMovie = 1;
113             int paramIndexStar = 1;
114
115             for (String word : words) {
116                 statementMovie.setString(paramIndexMovie, "+" + word + "*");
117                 paramIndexMovie++;
118                 statementStar.setString(paramIndexStar, "+" + word + "*");
119                 paramIndexStar++;
120             }
121             statementMovie.setString(paramIndexMovie, query);
122             statementStar.setString(paramIndexStar, query);
123
124
125             // Execute queries on movie title and actor name
126             //ResultSet rsMovie = statement.executeQuery(queryMovie);
127             ResultSet rsMovie = statementMovie.executeQuery();
128
```

Task 2 - Scaling Fabflix

Address:

- Google Instance: 35.230.118.71
- Instance 1 (Original): 54.215.171.42
- Instance 2 (Master): 54.193.71.63
- Instance 3 (Slave): 13.57.85.180

For connection pooling and the read/write requests, refer to Task 1 (above) as we've already answered these questions using an explanation and the provided screenshots and line numbers.

Task 3 - J Meter

The log file and HTML file have not been uploaded to the repository. While working on task3, we were unable to get JMeter to work correctly. The requests made through JMeter were all going through to our servlet and returning a status code of 200. However, we were unable to get any response data. The appropriate print statements and requests were all showing up in our logs, but some bug in configuration or code caused the servlet to not return any data when accessed

through JMeter. However, everything works correctly when accessing the load balancer through our browser.

Instead, we've provided a sample log.txt file to show that our script works. The log file contains the elapsed JDBC and Servlet time in nanoseconds for each query that was manually searched. We correctly implemented a python parser that takes in the log.txt file as input and parses through the data and returns the the average time it takes all TS's and TJ's. Inside the directory /proj5_files, you will find our python code called parser.py, our log.txt file which contains the TS and TJ of each query, and a snapshot of our parser being executed on the given text file called parse_run.jpg.

In the log.txt file, we are given entries such as this, "TS809966552 TJ808179081\n". What our parser is displayed in the order of the following steps:

1. Removing all alphabetical letters.
2. Stripping off the newline.
3. Splitting between spaces so now we have a list of lists
    a. EX. [["809966552","808179081"]]
4. Moving all items out so it is just one list and type casting them into ints.
5. Separating the list into two, TS and TJ values.
6. Taking the average of both lists and returning them.

The script is located on GitHub in the directory /proj5_files

The WAR file and README are located in the directory /proj5_files