## Contents

## 1.Introduction

This report provides test results and analysis of the performance of the Online Book Store application which is developed in assignment 2 of COMP9321 based on assignment 1. It is divided into five sections. In the first part, it is described that what functions are designed and how they are built to achieve the requirement. Next part is about how to set up the test for producing required and measureable results. Then, the values of test result requested in the assignment specification is stated in the third part. The fourth part involves the performance analysis of application based on previous values. Finally, some statements are provided to conclude the whole report.

## 2. Application Details

### 2.1 Customer Side

### 2.1.1 User Registration

The user does registration through adding personal information including username, password, email address etc. Some detection such as username repetition checking are involved in this part. After completing register, an account active email will be sent to user's email. User can not login until active the account by clicking the active link.

The registration process is shown in Figure 1. First the information of user will be added to database and attribute 'actived' is set to 0 which means user is not active yet. After user click the active link in the email, a call is made to database to change user's attribute 'actived' from 0 to 1. Now the user is active and can log in the application.

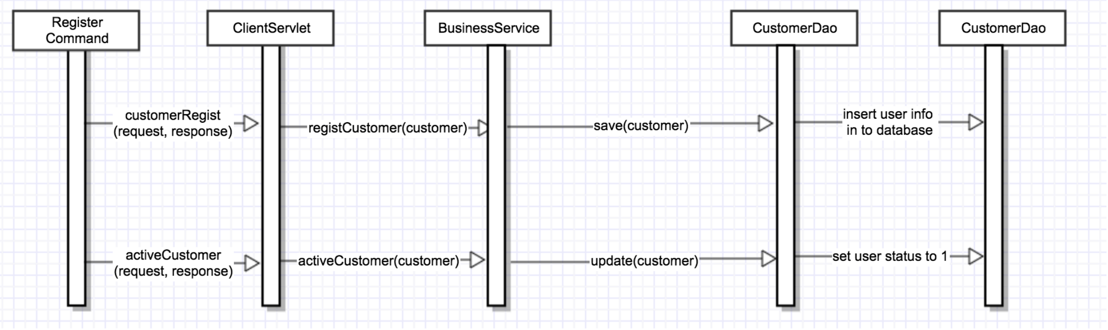


Figure 1

### 2.1.2 User Login

The user inputs username and password for login. If the user information matches one in the database and user is not banned, the login will be successful. Otherwise, if the user information does not match the data in database or user is banned, user can not log in the application.

The login process is shown in Figure 2. A call is made to the database to find the matched user by username and password. If the user is found, user's information is return to application from database. Second check the user is banned or not. If not banned, log in successful, otherwise refused.

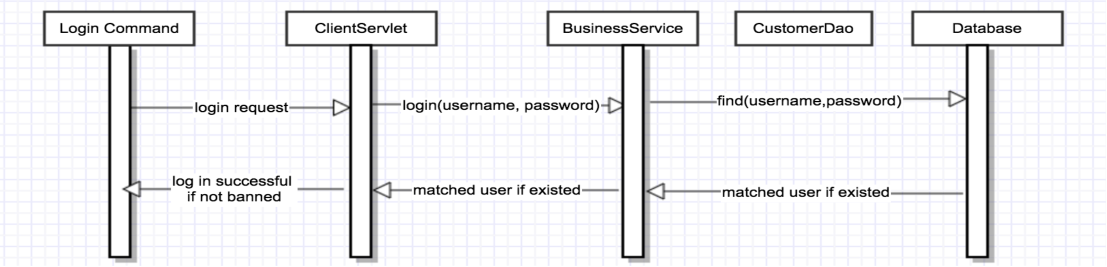


Figure 2

### 2.1.3 Search

There are two search choices which are standard search and advanced search. User can use standard search to select title, description and type. The advanced search can be used to select author, year and venue as additional options. When user click search button, the calls are made to the database to display the matched results.

The search process is shown in Figure 3. A call is made to the database to find all matched items whose attribute 'status' is 1.(status=1 means the book is selling,0 is paused) and return the results to application for display.

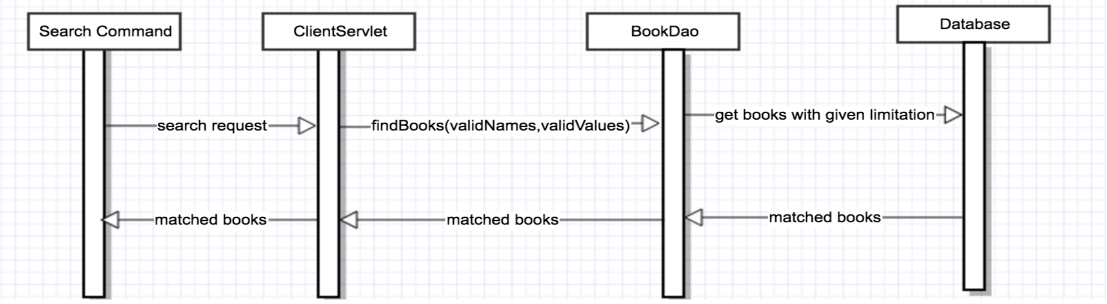


Figure 3

### 2.1.4 Add book

The user who has logged in can sell book in the add book function. User can add book's information such as title, description, price and price. After user inputs all the information and click add button, the book will be added to database.

The add book process is shown in Figure 4. when user clicks the add button, .a call is made to database and add the book info into the database.

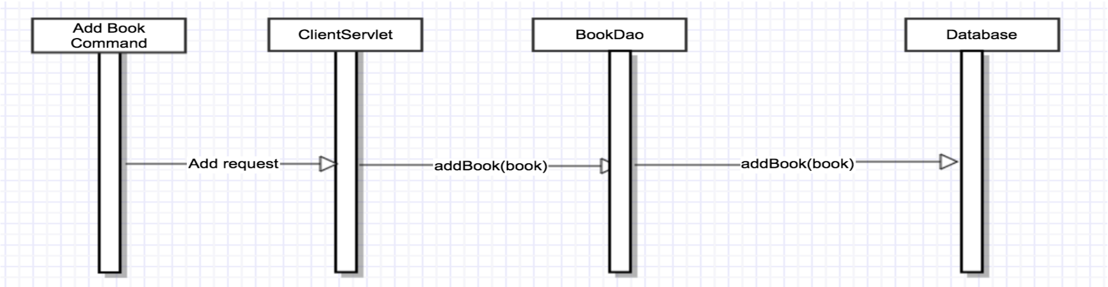


Figure 4

### 2.1.5 Sale Management

The user can view and manage his selling book in this function, including pause book sale and relist books. When user click the pause button, the selected book will stop selling and can not be searched. After relist button is clicked, the book will be on sale again and can be searched.

The sale management process is shown in Figure 5, when user clicks the sale management button, a call is made to dataset and find the books sold by this user through matching the attribute 'code' of book table with the one of user. Pause function is achieved by make a call to database and set the attribute 'status' to 0. Similarly, relist function is to set the attribute 'status' to 1.

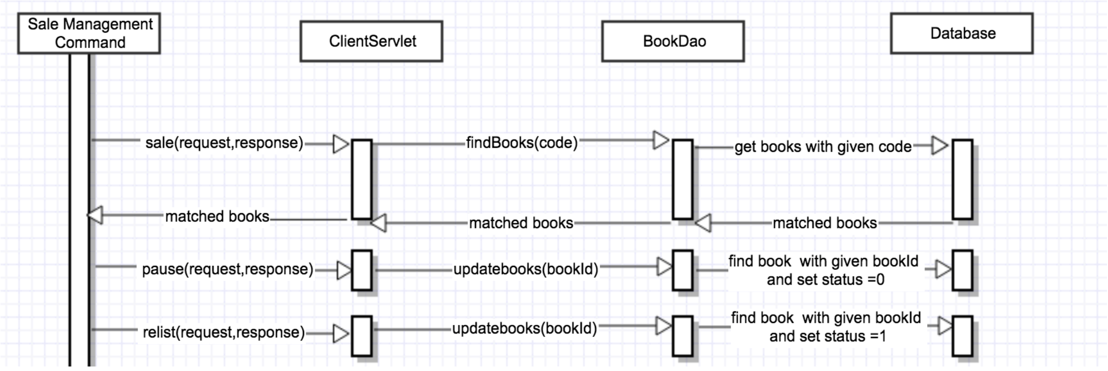


Figure 5

### 2.1.6 Add To Cart

The user can add selected book to shopping cart and the single book can be added repeatedly.

The add to cart process is shown in Figure 6, first a call is made to database and find the selected book by book id. Then a second call is made to database to add the selected book info into cart table.

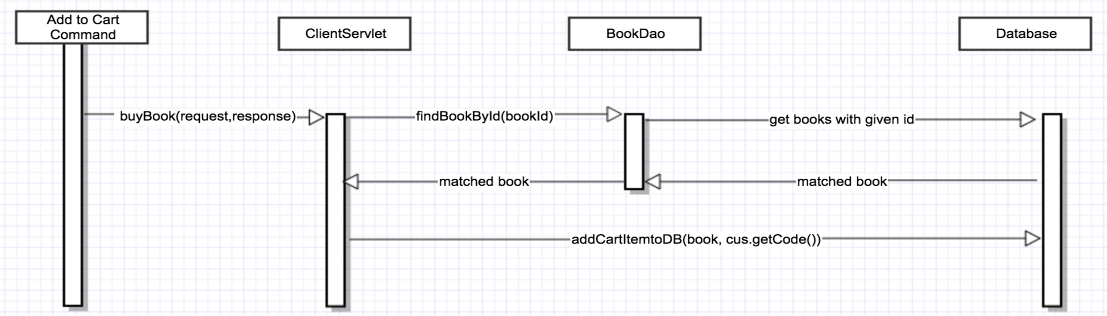


Figure 6

### 2.1.7 Shopping Cart

The user can view the items he ordered in this function. Besides, user can also remove the ordered books or confirm to purchase.

The shopping cart process is shown in Figure 7. When user click the shopping cart button, a call is made to database and find the ordered item from cart table by user's attribute 'code'. If user do remove operation, a call is made to database and delete the selected ordered item from cart table by book id and user's code. After check out button is pressed, a call is made to database and add all the ordered item into order table.

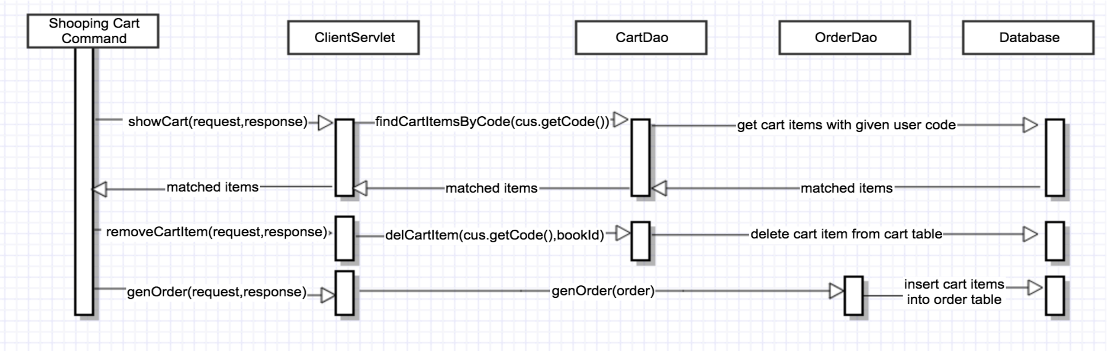

Shooping Cart Command
ClientServlet
CartDao
OrderDao
Database
showCart(request,response)
findCartItemsByCode(cus.getCode())
get cart items with given user code
matched items
matched items
matched items
removeCartItem(request,response)
delCartItem(cus.getCode(),bookId)
delete cart item from cart table
genOrder(request,response)
genOrder(order)
insert cart items into order table

Figure 7

### 2.1.8 Order statement and payment

The user can view his order info and do the payment operation in this function. When user click the pay button, the items confirm to be purchased and seller will get an email notification.

The order statement and payment process is shown in Figure 8, when user click the ‘My Order’ button, a call is made to database and find all the user’s order info from order table by user’s attribute ‘id’. When ‘pay’ button is clicked, a call is made to database and set attribute ‘status’ of order table to 1.Besides, an email will be sent to seller.

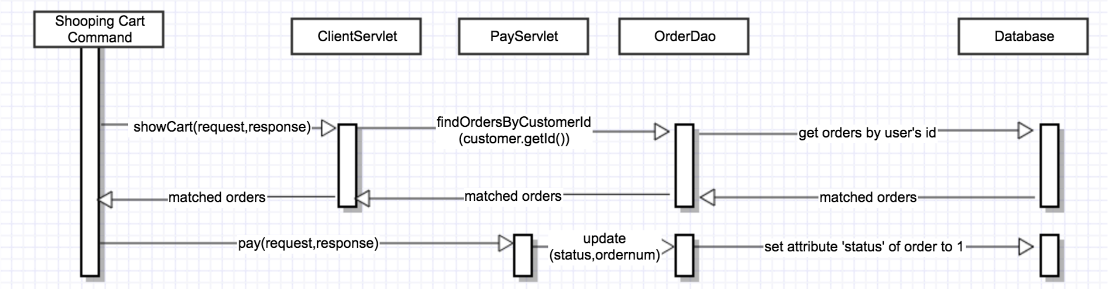


Figure 8

### 2.1.9 Personal Information update

The user can user this function to view and update their information such as password, email address, nickname ,etc.

The personal information update process is shown in Figure 9. First the values in session attribute ‘customer’ is called and display them in this page. After user complete the change and click ’update information’ button. A call is made to database and replace the original user information with new ones.

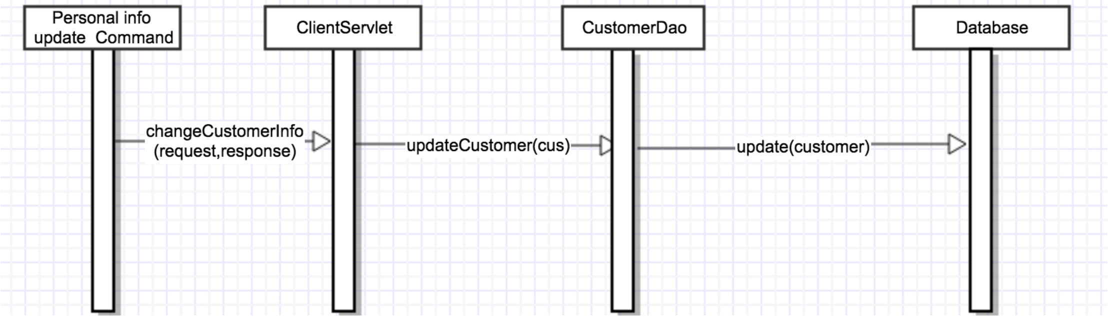


Figure 9

### 2.1.10 Detail

The user can click the link on book title to see the detail information of the book. The detail process is shown in Figure 10, a call is made to database and find the book info by book id.

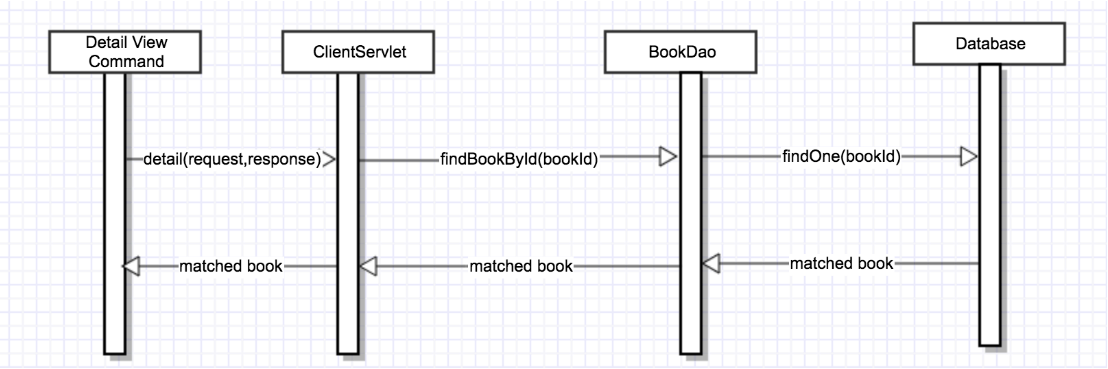


Figure 10

### 2.1.11 Graph Store and Visualization

This function can display the search result in graph form. Including the relation "paper--(authoredBy)-->author", "paper--(publishedIn)-->venue".

This process is shown in Figure 11. A call is made to database and find matched subjects from entity_store table and edges from graph_store table.

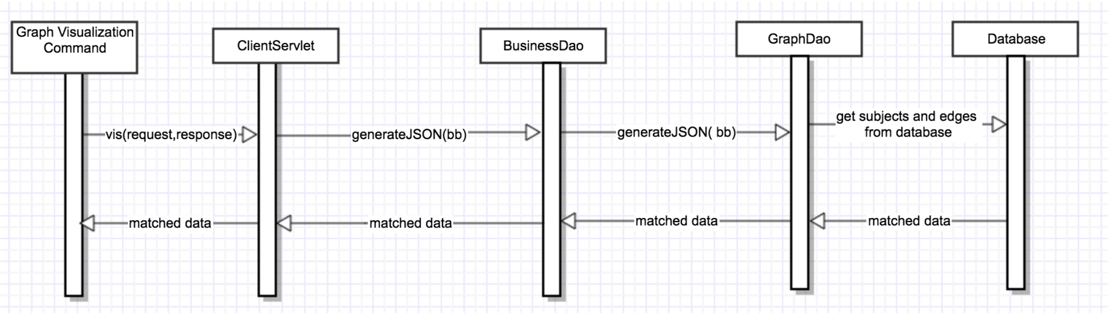


Figure 11

## 2.2 Administrator Side

### 2.2.1 Login

Administrator needs to input username and password to log into the backend management.

The login process is shown in Figure 12, a call is made to database and match the admin info with the one stored in users table. If the two are matched, log in successful, otherwise failed.

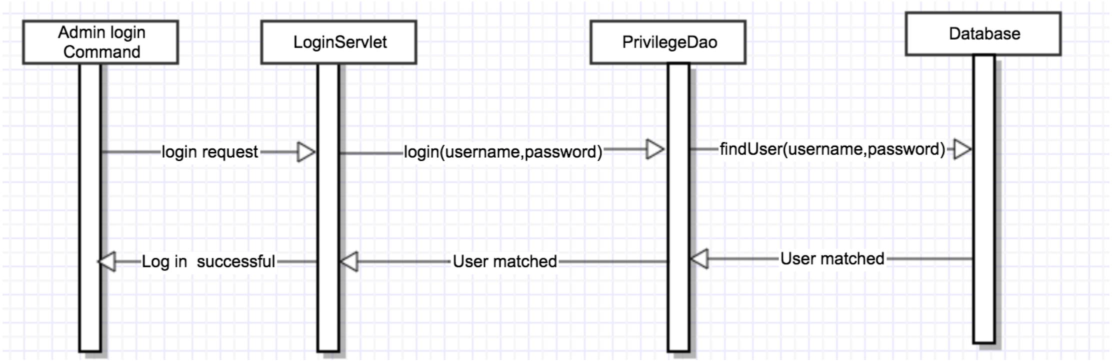


Figure 12

### 2.2.2 Book management

The administrator can view all books and remove selected books.

The book management process is show in Figure 13. First a call is made to database and return all elements stored in books table. If remove button is clicked, a called is made to database and find the selected book by book id and delete it.

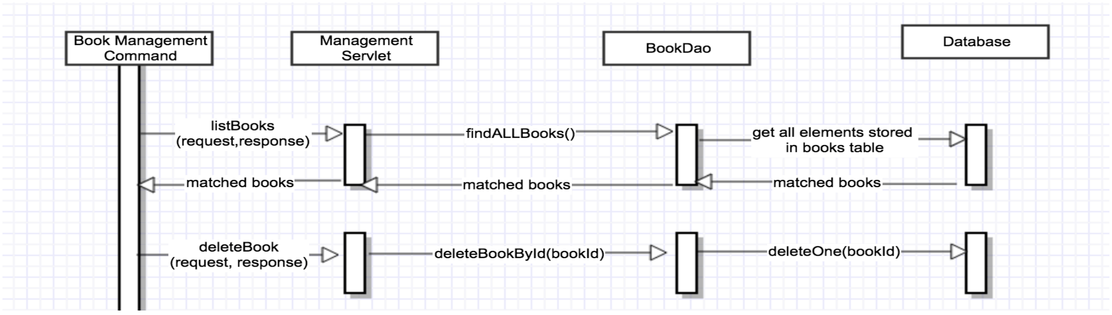


Figure 13

### 2.2.3 User Management

Administrator can view all the users info such as username, etc. Besides, operation to the users can also be made in this function, including ban user and reactive user.

User management process is shown in Figure 14, First a call is made to database and return all the users stored in customers table. If the ban button is clicked, a call will be be made to database and find the user by user id and set attribute ’ active’ to ’0’.Similarly, If the reactive button is clicked, a call will be be made to database and set attribute ’ active’ to ’1’.

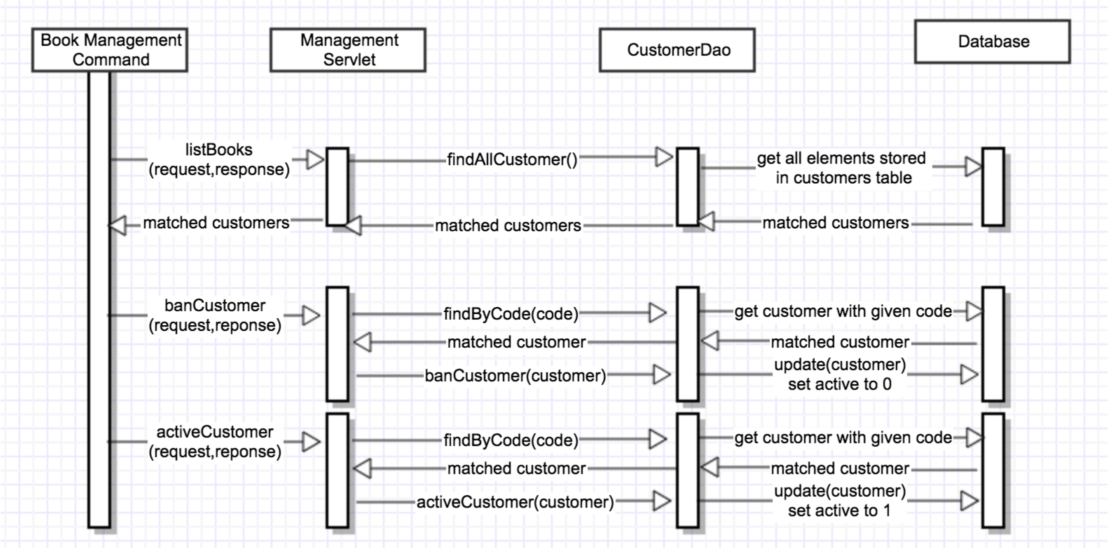


Figure 14

### 2.2.4 Log management

Administrator can view the log in this function, including purchase record and book remove record. Besides, all the logs can be deleted by click the ‘clean all logs’ button.

Log management process is shown in Figure 15. First a call is made to database and return all the content stored in log table. If the clean log button is clicked, a call will be made to database and delete all the content stored in log table.

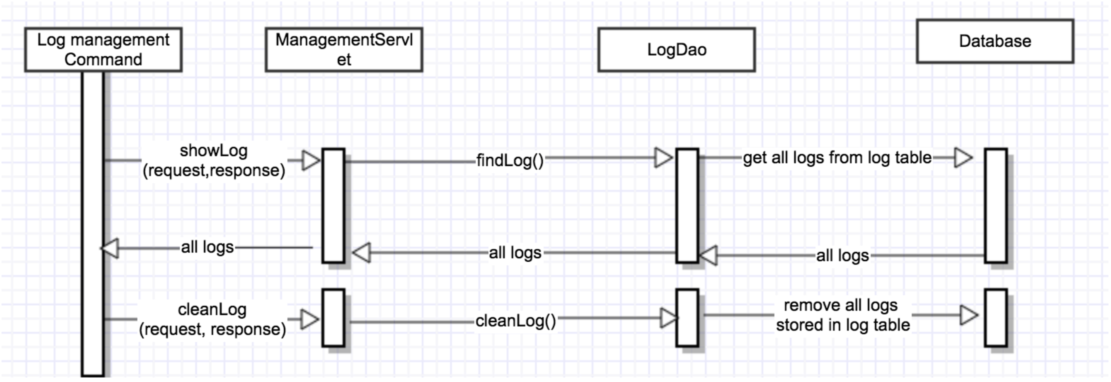

Log management Command
ManagementServlet
LogDao
Database
showLog (request,response)
findLog()
get all logs from log table
all logs
all logs
all logs
cleanLog (request, response)
cleanLog()
remove all logs stored in log table

Figure 15

## 3.Test Setup

### 3.1 Server Details

| | |
|---|---|
| CPU | I5-3230M 2.6GHz |
| RAM | 6G |
| HDD | 320G |

Table 1: Hardware details.

| | |
|---|---|
| Operating system | Windows7 64-bit version 6.1 |
| Java version | 1.7.0_51 |
| JVM Server | Java HotSpot(TM) 64-Bit Server VM |
| JVM Server version | 66-b17 |
| JVM maximum memory | 1.32G |
| Apache Tomcat Server | 7.0.56 |
| Database | MySQL 5.7 |

Table 2: Operating enviroment.

#### 3.1.1 Database

The application uses MySQL 5.7 as database server and c3p0-v0.9.5.2 as JDBC connection pool. We kept original settings for the testing though they impact the performance at an extent.

| | |
|---|---|
| Initial pool size | 100 |
| Min pool size | 100 |

| | |
|---|---|
| Max pool size | 1000 |
| Increment | 200 |

Table 3: Connection pooling.

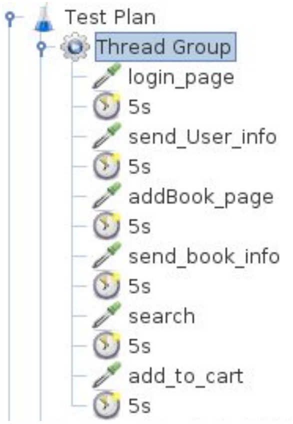
## 3.2 Client Details

| | |
|---|---|
| CPU | i5- 6200U 2.3GHz |
| RAM | 4G |
| SSD | 128G |

Table 4: Hardware details.

**This experiment was undertaken in a LAN.**

### 3.2.1 Test Plan



- Test Plan
  - Thread Group
    - login_page
    - 5s
    - send_User_info
    - 5s
    - addBook_page
    - 5s
    - send_book_info
    - 5s
    - search
    - 5s
    - add_to_cart
    - 5s

- Visit customer login page
- Submit username and password
- Go to book sale page
- Submit text-based information and a picture (multipart form data)
- Search for the book just added
- Add it to cart
- Wait for 5 seconds between each action

Because there is a redirection after adding items to cart in our back end, the last 5-sec waiting is needed.

### 3.2.2 Experiment structure

**Procedures:**

- Re-deploy the application and visit the homepage once to initialize the server whenever a test has been done.
- Keep our application's monitoring page open in Psi-probe, and wait some large response time value to show up.
- Start the test in Jmeter.
- Note down the stable average response time on the monitoring page.
- Save table data of aggregate graph in Jmeter and import them to excel to analyze

later.

- Change the thread number and login parameters in Jmeter, and clean book table in database for the next test.

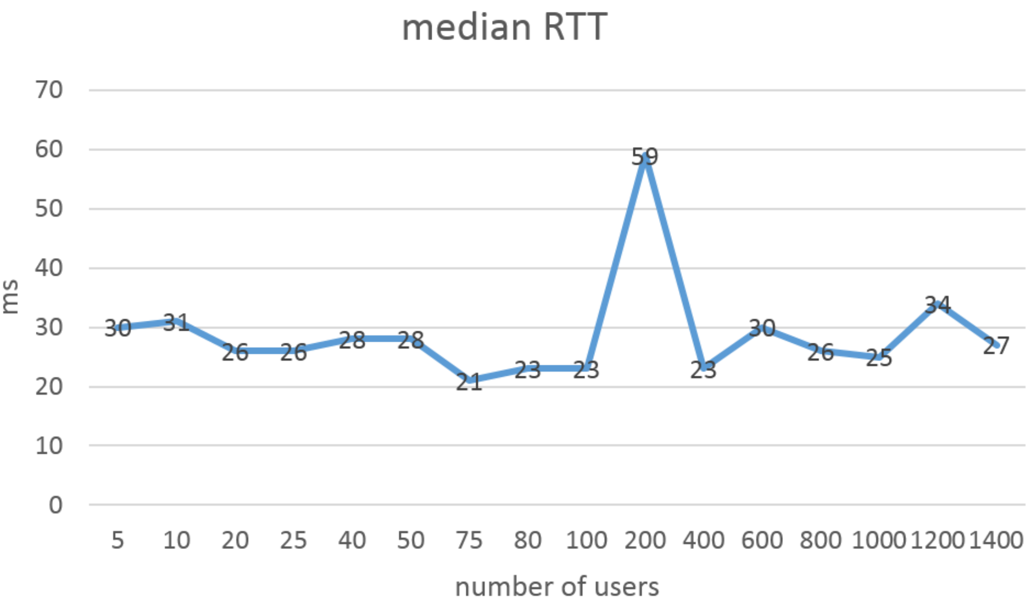
The test was intended to compare the system performances under varied number of concurrent users, and find the maximum that our configuration supports. The number of users increases from 5,10,20,25,40,50,75,80,100 to 200, and then add 200 each time until the server corrupts. The test was constructed using JMeter testing software. By using its 'Thread Group' feature we constructed a set order of HTTP requests to be sent to the server in order to replicate actions taken by actual users.
This experiment is to simulate numbers of our website's members selling and buying. To perform the real-world situation, we ran a sql script to add 1500 customer registration information into the database, and read parameters from a text file of these information to the login step letting different users login individually. And those added books are necessary to be removed between each test as enquiring performance can suffer from their accumulation.
The experiment went well with no errors until the testing number climbed to 1600. That after a half of the responses received normally, the server continuously gave "connection cannot be acquired" errors. We have run the 1600 test multiple times and confirmed that nothing can be changed unless we modify the c3p0 connection pooling configuration. However, the performance will reach the bottleneck of MySQL or tomcat somehow if we did this. Because that would lose some meaning of this experiment, we left the configuration unchanged.

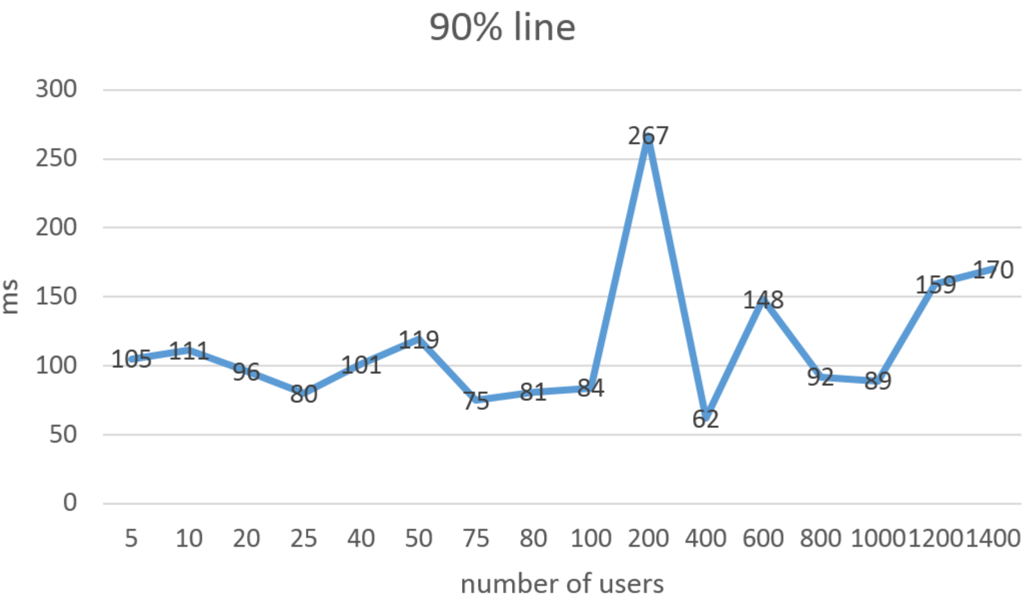
A noticeable point is that Psi-probe's response time monitor does not fit this experiment well. Because its auto-sampling interval would be 3-4 minutes, we can only start JMeter by observing the change of the average response time to "place" our testing period into that interval to obtain reasonable but yet inaccurate values. And thus there are some incorrect but truly observed results in our data which will be stated later.

## 4.Test Results

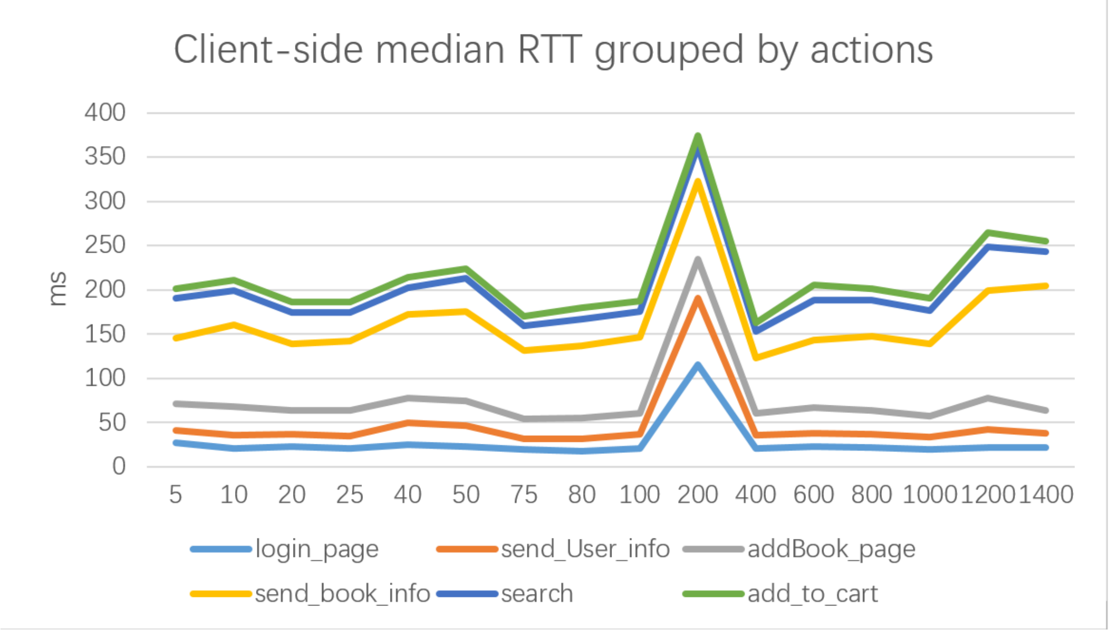
### 4.1 Client-side response times

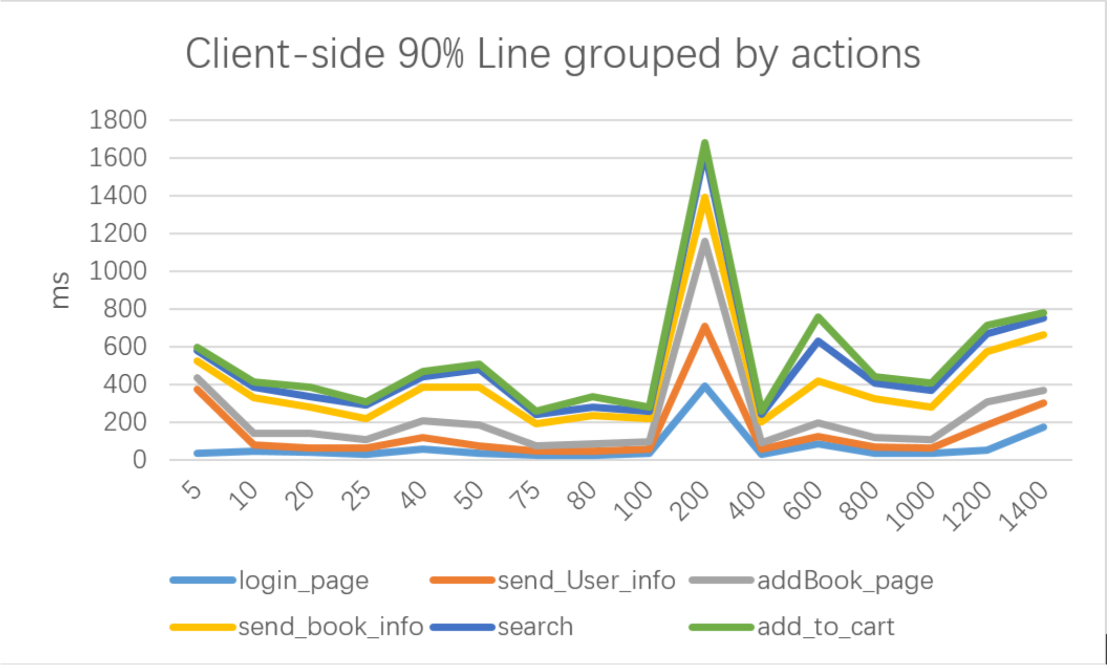
### 4.1.1 Total response times for each test



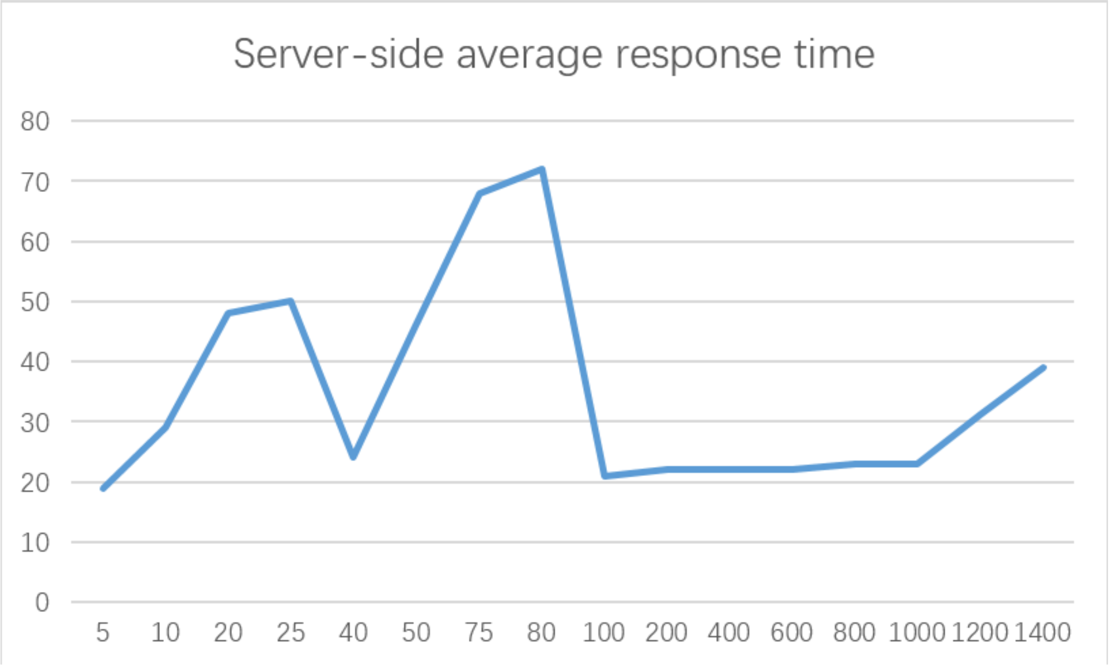


### 4.1.2 Response times grouped by actions for each test





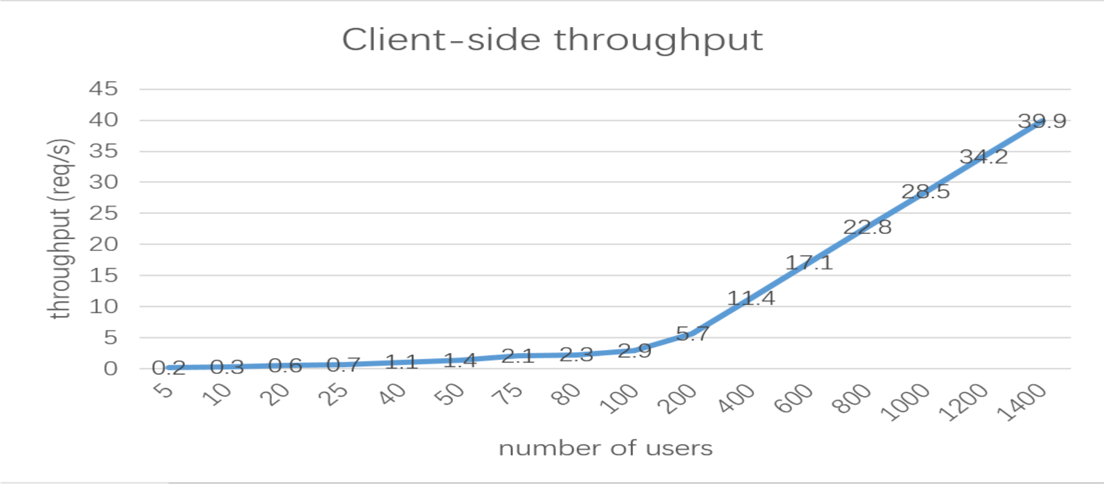
## 4.2 Server-side average response times

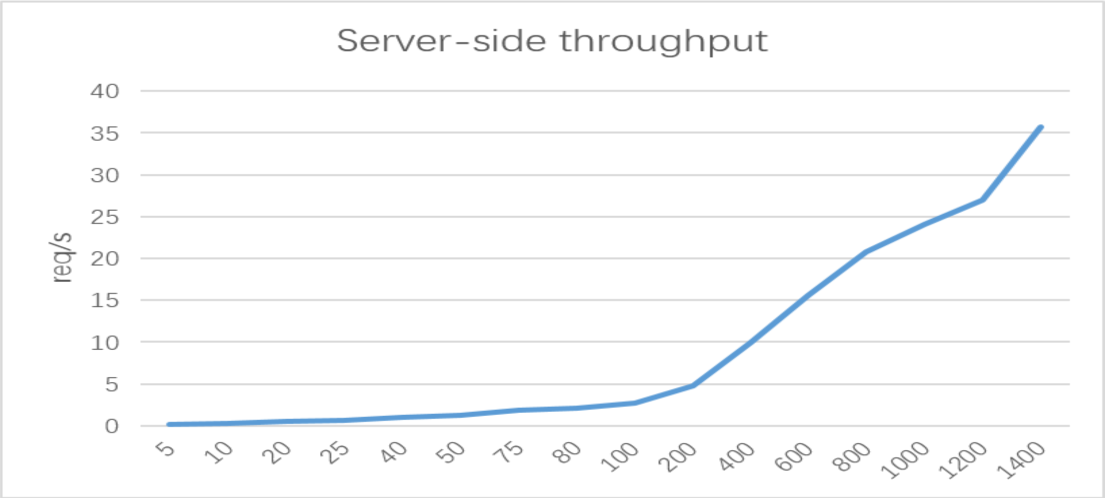


## 4.3 Throughput

| Number of users | Average response time(ms) | Number of requests |
|---|---|---|
| 5 | 19 | 32 |
| 10 | 29 | 64 |
| 20 | 48 | 124 |
| 25 | 50 | 154 |
| 40 | 24 | 244 |
| 50 | 46 | 302 |
| 75 | 68 | 452 |
| 80 | 72 | 482 |
| 100 | 21 | 604 |
| 200 | 22 | 1204 |
| 400 | 22 | 2404 |
| 600 | 22 | 3603 |
| 800 | 23 | 4801 |
| 1000 | 23 | 6000 |
| 1200 | 31 | 7189 |
| 1400 | 39 | 8402 |

Table 5 Client-side request numbers computed by Little's Law



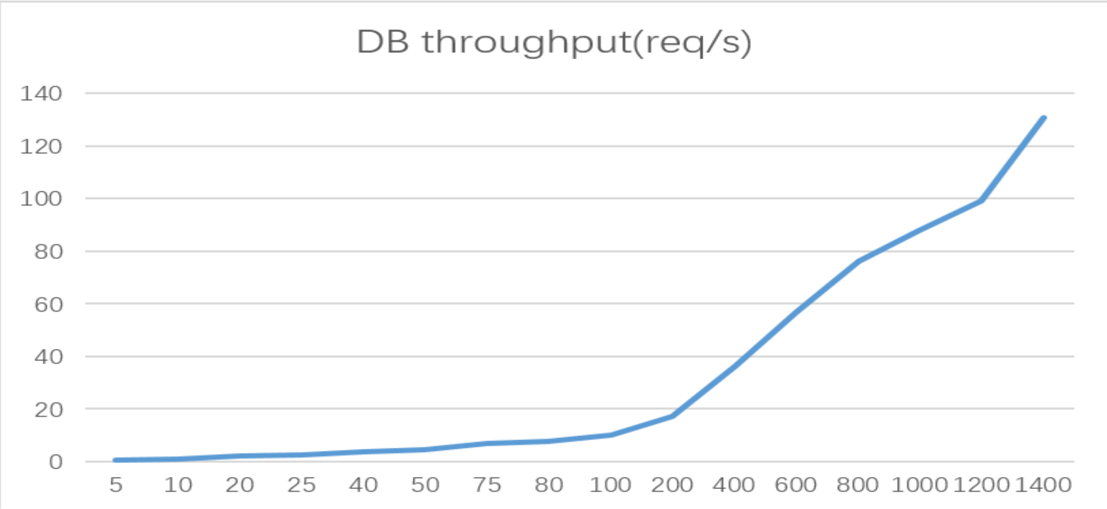


## 4.4 Database throughput

| number of users | System throughput(req/s) | avg DB visit number(visit/req) | DB throughput(req/s) |
|---|---|---|---|
| 5 | 0.14612 | 3.5625 | 0.520547945 |
| 10 | 0.28959 | 3.5 | 1.013574661 |
| 20 | 0.57143 | 3.580645161 | 2.046082949 |
| 25 | 0.66957 | 3.597402597 | 2.408695652 |
| 40 | 1.0655 | 3.62295082 | 3.860262009 |
| 50 | 1.27426 | 3.655629139 | 4.658227848 |
| 75 | 1.9234 | 3.659292035 | 7.038297872 |
| 80 | 2.06867 | 3.659751037 | 7.570815451 |
| 100 | 2.73303 | 3.649006623 | 9.972850679 |

| | | | |
|---|---|---|---|
| 200 | 4.74016 | 3.657807309 | 17.33858268 |
| 400 | 9.85246 | 3.658069884 | 36.04098361 |
| 600 | 15.5302 | 3.662225923 | 56.875 |
| 800 | 20.7835 | 3.657154759 | 76.00865801 |
| 1000 | 24.0964 | 3.657 | 88.12048193 |
| 1200 | 27.0263 | 3.663235499 | 99.0037594 |
| 1400 | 35.7532 | 3.658533682 | 130.8042553 |

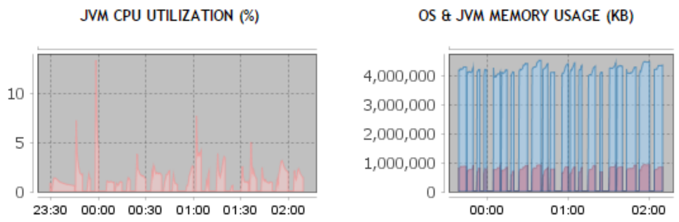
Table 6: Throughput of the database, computed using the Forced Flow Law

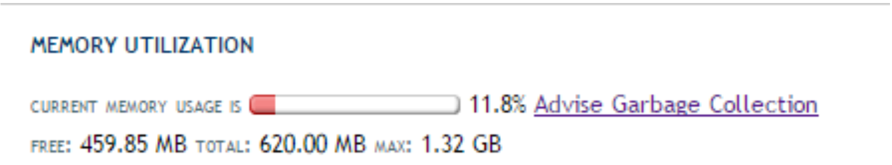


## 4.5 Server-side system utilization

| number of users | processing time (s) | observing duration (s) | Utilization (%) |
|---|---|---|---|
| 5 | 2.179 | 219 | 0.994977 |
| 10 | 2.417 | 221 | 1.093665 |
| 20 | 2.989 | 217 | 1.377419 |
| 25 | 3.442 | 230 | 1.496522 |
| 40 | 5.494 | 229 | 2.399127 |
| 50 | 6.942 | 237 | 2.929114 |

|  |  |  |  |
|---|---|---|---|
| 75 | 8.719 | 235 | 3.710213 |
| 80 | 8.813 | 233 | 3.782403 |
| 100 | 9.977 | 221 | 4.51448 |
| 200 | 20.773 | 254 | 8.178346 |
| 400 | 27.95 | 244 | 11.45492 |
| 600 | 51.473 | 232 | 22.18664 |
| 800 | 70.388 | 231 | 30.471 |
| 1000 | 89.209 | 249 | 35.82691 |
| 1200 | 155.928 | 266 | 58.61955 |
| 1400 | 219.488 | 268 | 81.89851 |




MEMORY UTILIZATION

CURRENT MEMORY USAGE IS 11.8% Advise Garbage Collection

FREE: 459.85 MB TOTAL: 620.00 MB MAX: 1.32 GB

# 5. Discussion

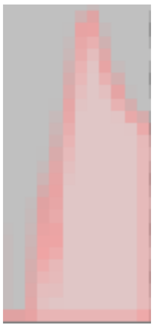
## 5.1 Performance analysis

### 5.1.1 system utilization

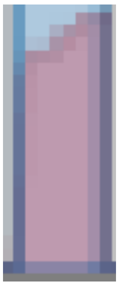
On the server machine, because our application was running on IDE embedded tomcat and database required to be clean for every test, MyEclipse, MySQL workbench, and Chrome browser with merely psi-probe page were open. This environment could consume a certain system memory mainly caused by windows7 but have no influence to the JVM memory amount and CPU utilization.

As can be seen in the JVM CPU utilization figure in 4.5, the utilizations for all those test are overall less than 3%, excepting several peaks produced by the initialization of server restarts. But we can also observe a slowly climbing tendency with the increment of users. On the other hand, those purple bars in the adjacent figure stand for JVM memory usage. Compared with the maximum JVM memory amount, our tests always use around 75% of the memory. Meanwhile, this illustrates the memory utilization is quite steady with all the scales that have been tested.

Furthermore, if we inspect any of the wave shapes carefully in both CPU and memory utilization figures, it is not hard to tell which testing stage matches a particular point on that shape. Here are two particular shapes for CPU and memory usage in one test, respectively:



(a) CPU



(b)Memory

They are extracted for 1400-user group which could tell us how the JVM system are running while handling a relatively great number of visitors. On the client side, the most resource-consuming actions would be selling a book, which asks the server to store images and write a lot of data into the database. When the ramp-up period is over, all the threads are running and it is the most operation in executing. And it pushes the CPU usage to the highest point.

Nevertheless, the memory usage has seemingly not been influenced by this but the duration of the test. That even most threads and connections are over at the ending stage, there is no sign of any usage reduction. Thus, we know that memory has not been released properly, and it is predictable that the memory will run out if we add more actions to the test plan to prolong the running duration.

This conclusion could be corroborated by the table in 4.5. At the last row of that table, system utilization for the 1400-user test has not reached 100%. This is not the ceiling and the server was not under pressure at that time.

Overall, although the RAM assignment could be critical in some extended situation, hardware is not what stopped our testing number from going up.

### 5.1.2 Response time analysis

Response time would be one of the most critical factor impacting users' experience. Because our experiment was under a LAN connected with only two devices, propagation delay could be negligible. Then, consider there are 1400 concurrent users, the 90%-line figure in 4.1.1 indicates they would very likely to experience a delay lesser than 170ms on average in their total visiting period. Besides, the second graph in 4.1.1

shows the 1400 users' median response time was 27ms. It combines with average response time - 40ms(client-side) and 39ms (server-side) respectively given by the graph in 4.2 and the table in 4.3, meaning the testing group averagely experienced delays lesser than 50ms and which are fairly good performance. But ,because operations would singly be taken by individuals, they do not only evaluate a website on some average values. Hence, it is more important that when our customers do some operations that require more communication with the database, such as searching, buying, and uploading information, they may sense an about 200ms delay difference to those actions require no database accesses. (median RTT graph in 4.1.2, where "search", "submit book info", and "add an item to cart" take about 250ms while the others spend 50ms to get a response.)
Although the server was not heavy-duty and thus the line graphs do not show an explicit increasing tendency, 1200 can be seen as a turning point where experienced delays soared if we exclude 200-user results as outliers. In the table of 4.3, we can observe that client-side average response times are approximate even the numbers of requests have multiplied over 200 times. Take what we concluded in utilization analysis into consideration, we did not encounter a case that requests start queuing on the server-side. This means our server was fully competent to handle all the tested numbers.

### 5.1.3 Analysis of the different steps in the operation

The steps in the operation that takes the most time are those involving database interaction. The two graphs in 4.1.2 clearly show the difference. That the response times of the operations require more database visits, such as "searching a book", "submitting information of the book to be added", and "add a book to cart from searching results", are significantly higher than the corresponding response times of" visiting login page"," visiting book sale page", and" Submitting user's login parameters".

Furthermore, the response times of the steps involving database interaction increase in line with number of users. On the other hand, the steps involving no or very few database interactions have fairly constant response times, independent of the number of users. Nevertheless, these three actions' consumption difference can be sued to easily list them from the most consuming to the least consuming:

1. "add a book to cart from searching results"
2. "searching a book"
3. "submitting information of the book to be added"

In our design, the first action would insert book information and then update its seller's identity code later. Note that updating would query to location the record, and then rewriting it. The second action involves searching in two tables which could take O(mn). Although the last one will upload and save a picture in the server, the picture size is only 100.8kb and the experiment was under a LAN that the transmission time can be negligible. Thus, everything becomes reasonable that time spent on 1 writing +1 updating operations> that on 2 querying operations > that on 1 writing operation.

### 5.1.4 Error analysis

There was no error report until we tested 1600 group. Under that load, error rate skyrocketed to about 50% in total, causing the records invalid. It is interesting that things did not go wrong until all the 1600 threads were started and nearly half of all the actions were done. Later on, the server started to prompt that no more connections to the database can be acquired, and most of the remaining actions failed due to this.
Based on the analysis above, we confirm that it is not caused by system nor network capacities. The error is simplex, directing us to the database connection pooling. We use c3p0, and the size and increment properties in the configuration are seemingly sufficient for this 1600-user test. Because we set the *maxPoolSize* to 1000, there will be connection waiting when over a thousand database connections occur. Then, a potential factor leaded to the errors is that idle connections were not released on time that made the awaiting connections reach timeout.

## 5.2 Possible performance improvements

### 5.2.1 Connection pooling

Apparently, our c3p0 needs to be reconfigured when considering a larger number of concurrent users. We could simply set the maximum pool size to its available max value, but this will soon run out of the pool and the capacity of the website would be still limited. A more clever way is to combine c3p0 connection releasing mechanism with this, setting a reasonable *idleConnectionTestPeriod* to disconnect the idle database connections according to expected scenario.

### 5.2.2 Database interaction

Based on our analysis, we believe that performance improvements best could be achieved by improving our data access layer. This is because database interaction seems to be the costliest part of the application. We would certainly want to reduce the amount of calls made to the database, and amount of data transferred. As stated in 5.1.3, “add a book to cart from searching results” and “searching a book” require multiple querying and writing operations. The corresponding performance would greatly improve if an action that accesses database reducing from twice to once, though the database may need to be remodeled.

### 5.2.3 Memory management

As discussed in 5.1.1, our server consumes memory at a considerable speed, and it is predictable that memory crisis will show up. Because we used to program at a small scale and does not deem it as a business product, variables and instances are declared for our convenience. This project involves a huge number of instances that we gave most of them the modifier public which is easy-to-use under most circumstance but consumes more system resources though it is nearly negligible in one single invoking. But when this difference accumulates, it would partly influence the performance.

## 6. Conclusion

This report provides the analysis of the performance of the Online Book Store Application which we developed for assignment 2 in COMP9321. Our analysis found that the web application worked well up to around 1600 concurrent users. Our tests also reviled that the application made some errors when the user load increased beyond 1600 concurrent users. The most serious error was that no more connections to the database can be acquired, and most of the remaining actions failed due to this. A factor leading to the errors is that idle connections were not released on time that made the awaiting connections reach timeout. Further, we found that the bottleneck of our application is the data access layer. The steps in the operation that takes the most time are those involving database interaction. As result of this, we suggested that performance improvements best could be achieved by improving our data access layer. One of the improvements we proposed, was reducing the amount of action that accesses database . Besides, another improvements could be done is reconfiguring c3p0 when consider a larger number of concurrent users.

video link: https://youtu.be/3I6oysv3wYE