Performance Evaluation of UNSWBook

Callum Bannister Jerome Samir Peter Bishay Aaron Quan
Waseem Sajeev

October 22, 2017

Contents

1 Introduction **2**

2 Application Details **3**
2.1 User Login 3
2.2 Search 4
2.3 Adding a friend 4
2.4 Posting a message 5

3 Test Setup **6**
3.1 Server 6
3.2 Client 6
3.3 Experimental Method 7

4 Results **8**
4.1 Data 8
4.2 Analysis 9

5 Conclusion **10**

1. Introduction

Maintaining quality of service under heavy usage is essential for any web application that expects a large amount of concurrent users. As there is currently no demand for an asocial media platform, we have performed a series of experiments on a selection of functions of our UNSWBook application to obtain data on performance degradation as the number of simultaneous users increases. We present the methodology; what functions were tested and what hardware were they tested on. Then, we present results of the experiment, an analysis of the results, and the conclusions to be drawn.

2. Application Details

We looked at four basic functionalities of UNSWBook for performance under pressure. They are as follows:

1. User login
2. Searching for friends
3. Adding a friend
4. Posting a message

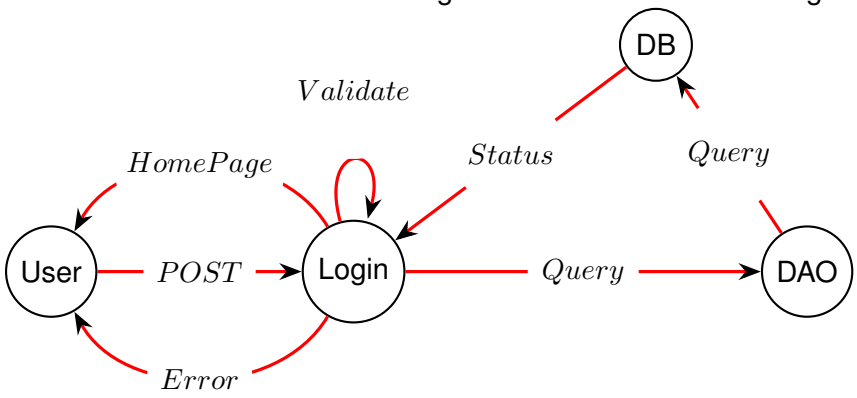
For each of these functions, we will explain the implementation and what database interactions were required, in order to give context to the performance data in Chapter 3.

2.1 User Login

The user submits their username and password with a POST request. The request body is passed to the *Login* servlet's doPost method, where we perform initial validation of the input, checking that the required fields exist.

Then, we construct the DAO object for a User. We make a single query to the database to check that a non-admin user with the name and password exists, and we retrieve their unique id and whether or not they are banned. Then, we route the user to either the login page, or their home page, returning an error message where necessary.

Figure 2.1: Control Flow for Login

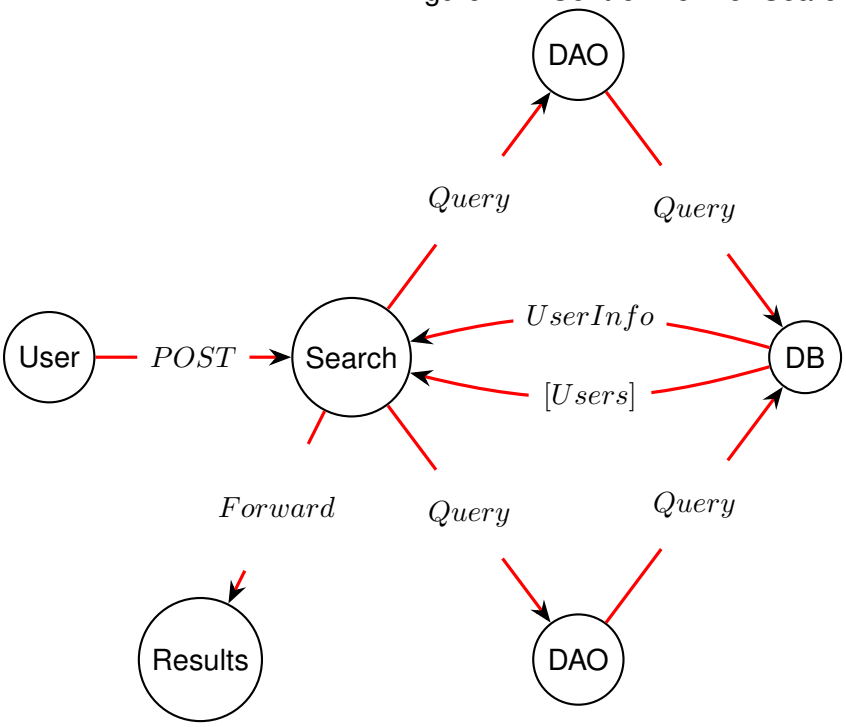


2.2 Search

In this section we describe only the basic search functionality. We expect that the advanced search to be used relatively infrequently, so optimisation of that function would be of secondary priority.

As before, the user submits the search query information via POST, getting routed to the appropriate method in the *Search* servlet. First, we construct a DAO object for the current logged in user, querying the database to obtain their information. Then, we construct a second DAO object, and use it to retrieve a list of users whose username contains the submitted string. We set this as an attribute, and forward the request onto $results.jsp$.

Figure 2.2: Control Flow for Search



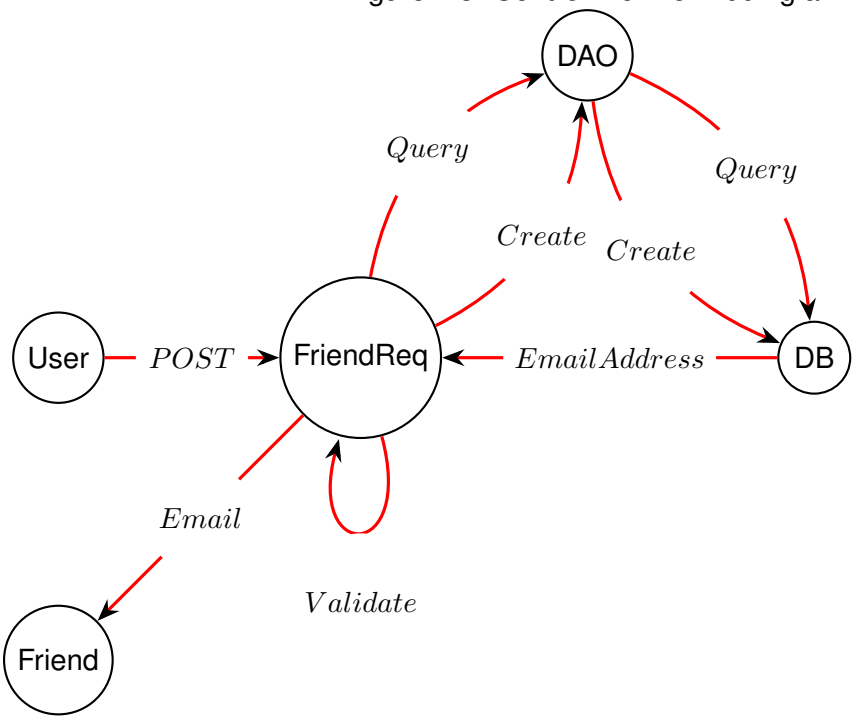
2.3 Adding a friend

Although adding a friend consists of two operations, the need for a confirmation email to link the two makes measuring them as a conjoined unit unrealistic. Instead, we present only the sending of a friend request.

The friend request submits the user id for the requester and the requestee, dispatched to the *FriendReq* servlet. As was the case for *Login*, we perform simple validation of the parameters. Then, we look up the requestee by their id, constructing the DAO and performing the query to the database through it.

We add a table to the *friends* database, with the confirmed flag set to $false$, requiring a second query. Afterwards, we extract their email, and send a message containing the appropriate link to confirm the friend request.

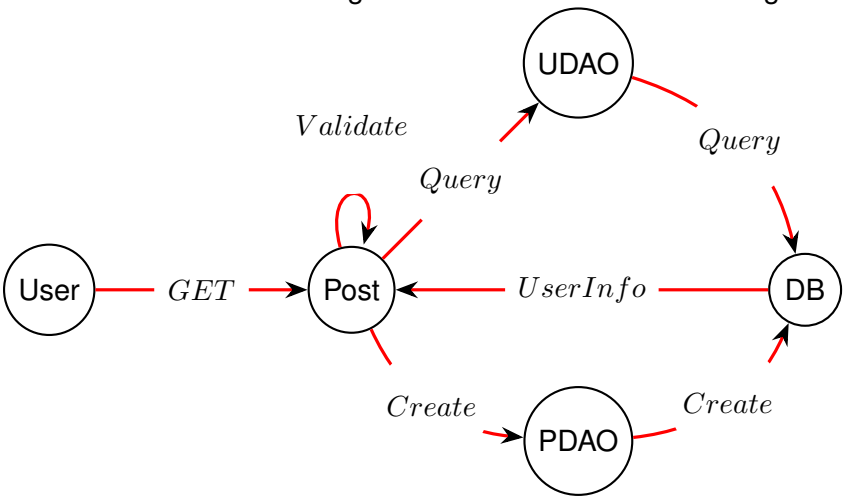
Figure 2.3: Control Flow for Adding a Friend



2.4 Posting a message

Unlike the other operations, the user posts a message by sending a GET request. This is routed to the *Post* servlet, where we once again perform simple validation on the input. We construct the DAO object and look up the currently logged in user as before. We then create a *post* DAO object, and use it to create the appropriate table in the database.

Figure 2.4: Control Flow for Posting a Message



3. Test Setup

In this chapter we present the hardware, setup, and experimental procedure that we used to obtain the data.

3.1 Server

The application server had the following hardware specifications:

Table 3.1: Server Hardware

Hardware	Specification
HDD	16 GB 1600 MHz DDR3
RAM	16GB
CPU	2.3 GHz Intel Core i7

The machine was running macOS Sierra 10.12.4, with JVM Version 1.7.0. We used Tomcat 7.0.42 as our server, and Apache Derby 10.9.1 as our database. To maximise the accuracy of our results the only user processes on the machine were the Tomcat and Derby servers.

The connection pool size was kept at the default for the jdbc driver (a *maximum* of 100 and *minimum* of 10). We did not predict that manipulating this value would lead to significantly different results.

3.2 Client

The client had the following hardware specifications:

Table 3.2: Client Hardware

Hardware	Specification
HDD	16 GB 1600 MHz DDR3
RAM	16GB
CPU	2.3 GHz Intel Core i7

As with the server machine, extraneous processes were kept to a minimum. Both the client and server were on the same LAN, minimising factors unrelated to the performance of UNSWBook from affecting the results.

3.3 Experimental Method

We constructed our experiment to simulate the following scenario:

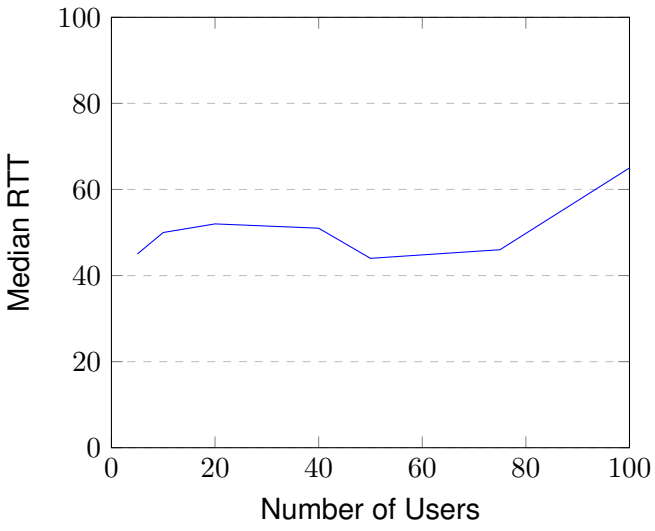
1. A user logs in, searches for a new friend, adds them, and posts a message to acknowledge their new friendship

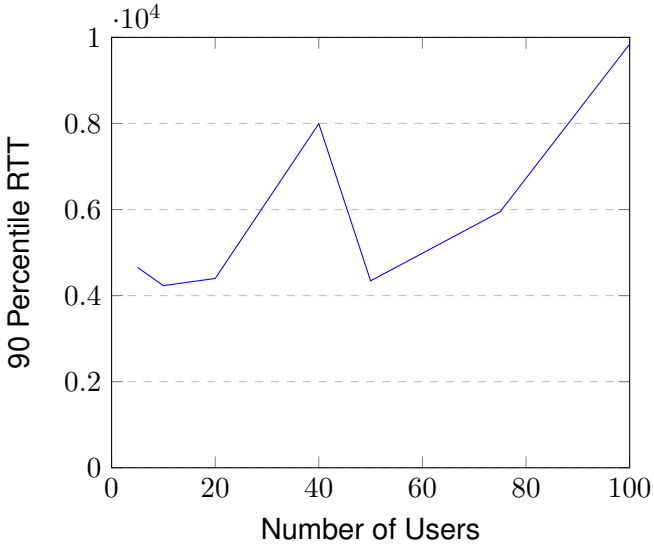
We added a delay of five seconds between each user action. We tested this scenario for increasing numbers of concurrent users: 5,10,20,40,50,75,100 and 200. Before each test attempt, we cleared the database of any existing posts and/or friends, in order to keep those factors constant.

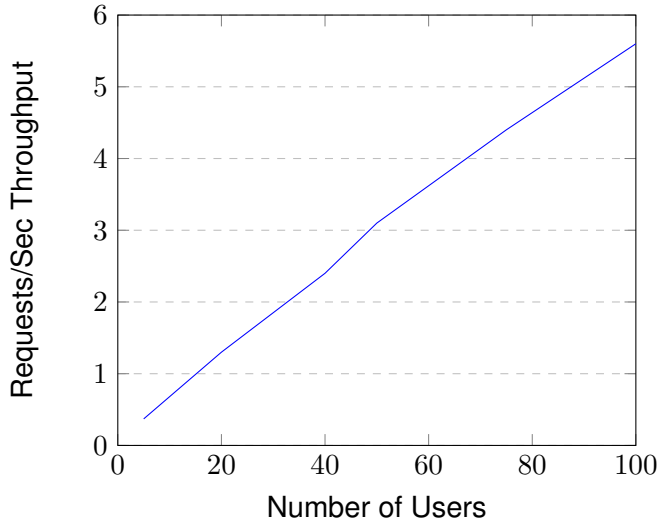
However, at 200 requests, we ran into authentication errors when trying to send out emails for friend requests, distorting the data. While this did not indicate a performance issue in the conventional sense, it demonstrated an issue that would not become apparent without placing the application under heavy load.

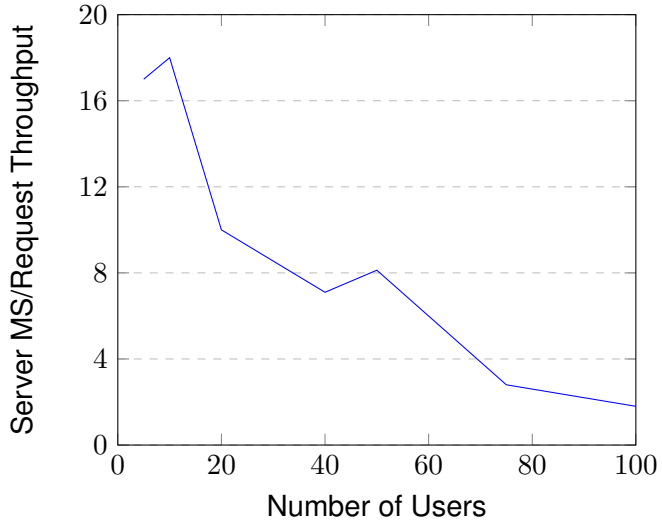
4. Results

4.1 Data









4.2 Analysis

For the bounds we were able to test, we noticed no degradation of the application performance. Aside from the email errors, it appeared as though we were able to support at least 200 users simultaneously. The throughput behaves as expected.

We observed that adding a friend caused the largest amount of delay, due to the sending of an email in order to confirm the friend request. A probable performance improvement would be to off-load the work into a queue or pool for later processing, and return to the user immediately, as the immediate sending of a friend request would be of a low priority compared to general responsiveness.

Unfortunately, we were not able to test UNSWBook on higher loads, due to the errors we found with sending friend requests.This indicates more work needs to be done on the functionality of the application under high load, rather than just responsiveness. Despite that, performance seemed good on what we tested - most operations require only one or two database calls.

5. Conclusion

We were able to gather some useful data on deploying UNSWBook under a realistic load environment. While the performance of our application held up, we found that the friend request functionality began to show errors under a high load. This indicates that the most productive area to focus on for improvement would be fixing this functionality.