Quality Attribute Analysis

for

Fernwood Farmers' Market Booth Scheduling System

or

FaBS

Table of Contents

- 1 Quality Attribute Scenarios
  - 1.1 Scenario 1: Edit Profile Information
  - 1.2 Scenario 2: View Schedule
- 2 Measurements
  - 2.1 Methods
    - 2.1.1 Scenario 1: Edit Profile Information
    - 2.1.2 Scenario 2: View Schedule
  - 2.2 Results
    - 2.2.1 Scenario 1 Results
    - 2.2.2 Scenario 2 Results
- 3 Revisions
  - 3.1 Improvement Tactics
    - 3.1.1 Reduce Overhead
    - 3.1.2 Increase Resource Efficiency
    - 3.1.3 Maintain multiple copies of data
    - 3.1.4 Bound Execution Times
  - 3.2 Architecture changes
  - 3.3 Expected Results

1 Quality Attribute Scenarios

1.1 Scenario 1: Edit Profile Information

Scenario	Values
Source	Users request
Stimulus	Edit vendor profile information
Environment	Normal operating conditions
Artifact	Mongolab
Response	Vendor account profile page
Response Measure	Update confirmation received within 2 seconds

1.2 Scenario 2: View Schedule

Scenario	Values
Source	User request
Stimulus	Load schedule page
Environment	Overloaded
Artifact	Mongolab
Response	Schedule/Page information
Response Measure	Market schedule fully loaded within 3 seconds

2 Measurements

2.1 Methods

We used an online load testing tool called “Load Impact” to locate and identify possible performance issues with our web application. “Load Impact” allows users to record a particular use-case and then simulate the scenario across a significant number of concurrent virtual users. The end result is a graph showing the system response times over an increasing number of concurrent virtual users. For our testing purposes and limitations, the maximum number of concurrent users were capped to 100, with the test running for 5 minutes. The same process was replicated across multiple computers simultaneously to simulate a larger concurrent user base, essentially resembling an overloaded operating environment when required.

2.1.1 Scenario 1: Edit Profile Information

Throughout this scenario, a virtual user is attempting to log-in to the website, access profile information, and then make a series of changes or edit their profile information. “Load Impact” calculates how long it takes for a virtual user to successfully accomplish the scenario with the response measure being a confirmation pop-up being presented to the user no later than two seconds after they have confirmed profile changes. The graph below shows how our system behaves (for this scenario) over a constantly increasing number of concurrent users over a fixed period of time, within the limitations enforced by “Load Impact”.

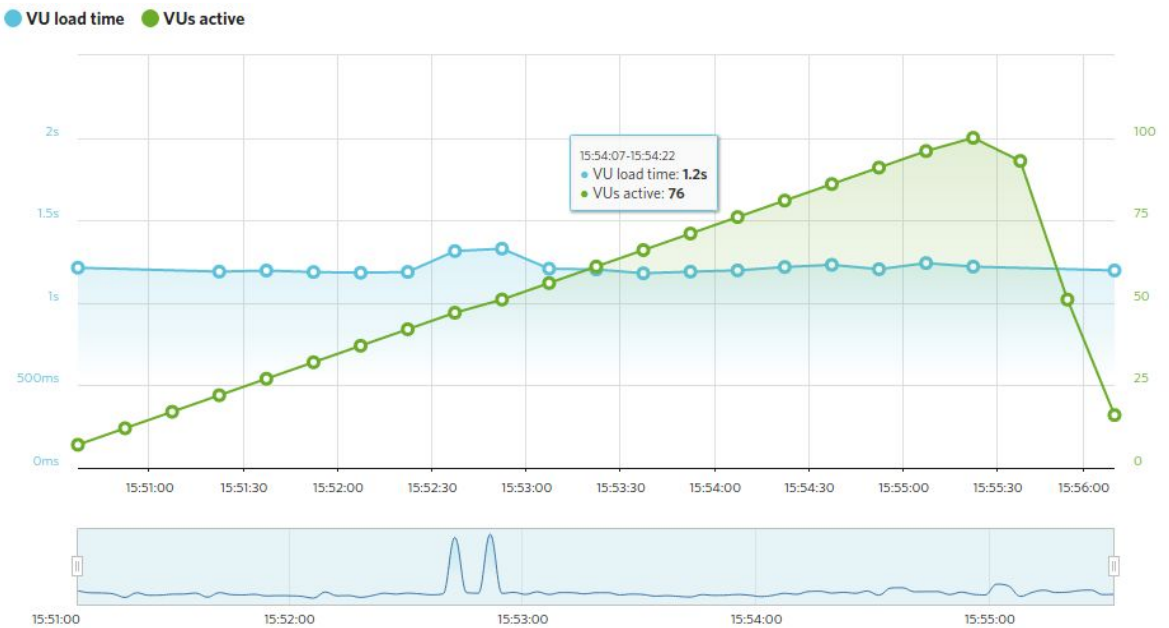

Figure 2.1.1: System response time against an Increasing number of concurrent users carrying out scenario 1 over time

2.1.2 Scenario 2: View Schedule

This scenario encompasses a virtual user attempting to log-in to the website and access/view the market schedule for the current day, and then multiple other days within the week. “Load Impact” calculates how long it takes for a virtual user to successfully accomplish the scenario with the response measure being the market schedule completely loading within the view under 3 seconds of user request. This task has been carried out in overloaded operating conditions and is expected to deviate from expected results. The graph below shows how our system behaves (for this scenario) over a constantly increasing number of concurrent users over a fixed period of time, within the limitations enforced by “Load Impact”.

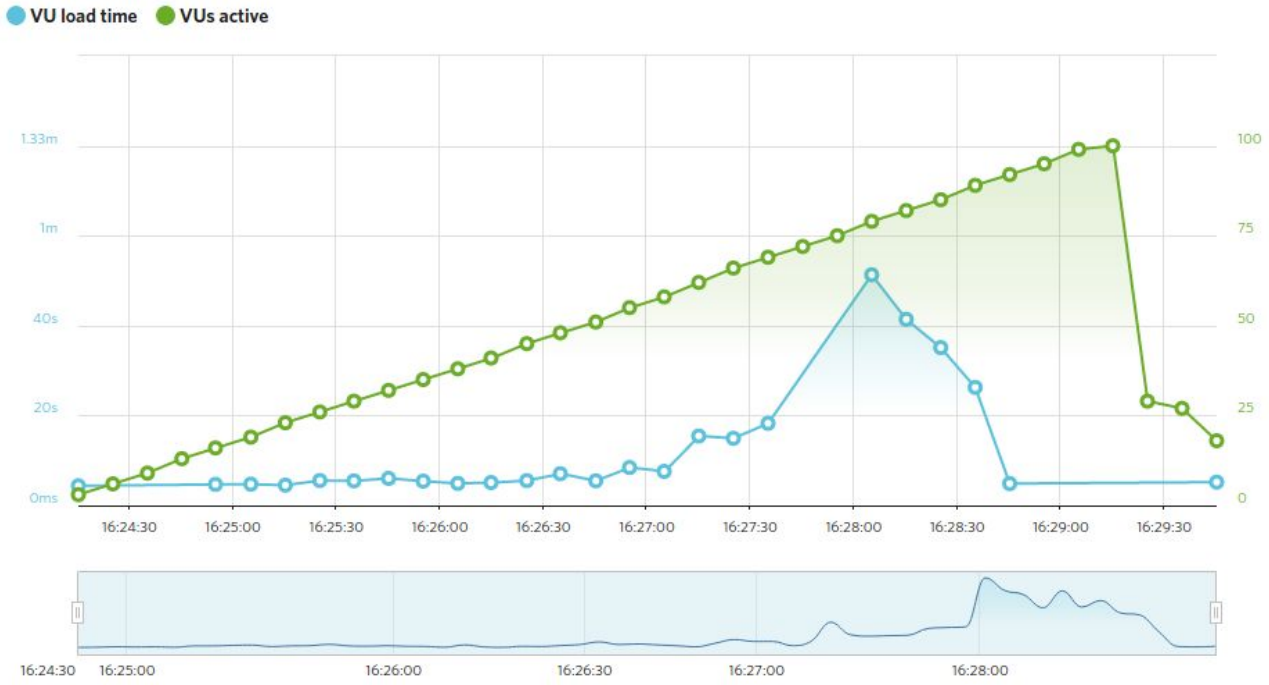

Figure 2.1.2: System response time against an Increasing number of concurrent users carrying out scenario 2 over time

2.2 Results

Results obtained from the “Load Impact” website for each of our test scenarios did not deviate from general expected behaviour. *Figure 2.1.1* indicates that under normal circumstances, the application has been particularly responsive and has met our QA scenario response measures, whereas, *Figure 2.1.2* indicates that under overloaded conditions, if significant number of concurrent users trying to carry out a complex use-case, the responsiveness of the website can diminish significantly.

2.2.1 Scenario 1 Results

Upon testing our first scenario that ran with 100 concurrent users (Figure 2.1.1), our load time remained under 1.5 seconds. We are then able to say with certainty that our app allowed for at least 100 concurrent requests to run under normal operations, as the testing application used limits to 100 users. With the addition of significant number of users, it is possible that the application could see some change in responsiveness.

2.2.2 Scenario 2 Results

Our experiment revealed a constant rate of responsiveness for up to ~60 users. We can see from the graph for scenario 2 that response time greatly decreased, running up to ~50 seconds for ~75 concurrent requests, which is far out of reach of the response measure defined in our QA scenario.

3 Revisions

3.1 Improvement Tactics

3.1.1 Reduce Overhead

Our page load times were made significantly slower by the large, high resolution image (3,229.29KB) we chose for our page backgrounds. Decreasing the size of the images to be loaded on the Login splash screen would significantly speed up response time.

3.1.2 Increase Resource Efficiency

For the markethome page, the mainController is instantiated twice, as well as marketController being instantiated once. This requires the user to load request more files from the server than they necessarily need to.

3.1.3 Maintain multiple copies of data

As the user profile details within the marketHome page remains as a fixed view, they can be cached to increase response times future page refreshes.

3.1.4 Bound Execution Times

If a response time for an action is exceeding the expected loading time, a message could be provided to the user to try the service again later. This would facilitate faster response times due to possibly faster database/server calls. It would also establish an upper bound on how long a user has to wait in order to carry out a particular use-case within the application.

3.2 Architecture changes

Having only a single controller (marketController) will reduce the number of times controller are instantiated. All functions used by mainController can be copied into marketController to reduce load times. The scope of the controller can be increased to allow the whole page to use the controller without instantiating it multiple times.

Adding in a timer to our http requests will help the system from being heavily over worked during peak times. The timer would spread out the requests by creating a buffer to allow the system to deliver results consistently. Having users get a timeout error is not an ideal scenario, however during the highest peak times, timing users out will reduce the average time per response and spread out peak usage times to reduce overall load. By creating an artificial buffer for the users, we can stagger their load times and increase the potential simultaneous users, however under extreme loads, the system would still have to provide a timeout message.

If we were to compress the background image before sending it over the network, we would have to implement a compression scheme on the server side as well as a decompression scheme on the client side. Changing the file format from png to jpg would also reduce the size of the images. Another option is to simply omit a detailed background and opt for a plain colour instead. This would completely nullify the need to load megabytes per page with the sacrifice of nicer visuals.

3.3 Expected Results

By reducing the amount of controllers being instantiated, the user would not have to request as many calls to the server in comparison to previous the implementation. This would now reduce load times for the particular web page, by having to access fewer controller files from the server. When applied over several hundred concurrent users, this process would help increase responsiveness significantly.

By spreading out server load through the use of an http request timer, we would expect to see no impact on a light load use, but the rate at which the server is impacted upon a heavy load would be changed. The curve at which the response rate degrades would be pushed back and allow for more active users as they would be put into a pseudo-queue as they are being buffered back to allow for a more even server load. Upon reaching peak load on the server where the buffer simply can’t keep up with the requests, a timeout message would have to be provided for any further queries.

By applying a compression/decompression scheme, the image will take less resources to download, making the load time of the overall page be significantly reduced, but more work would be needed to implement such a scheme. By changing file types, the overall download is also reduced without the sacrifice of too much image quality. The best performance gain would be seen from the switch to a plain coloured background as there would be no image to load at all.