Andrew Serensits

Dr. Raunak

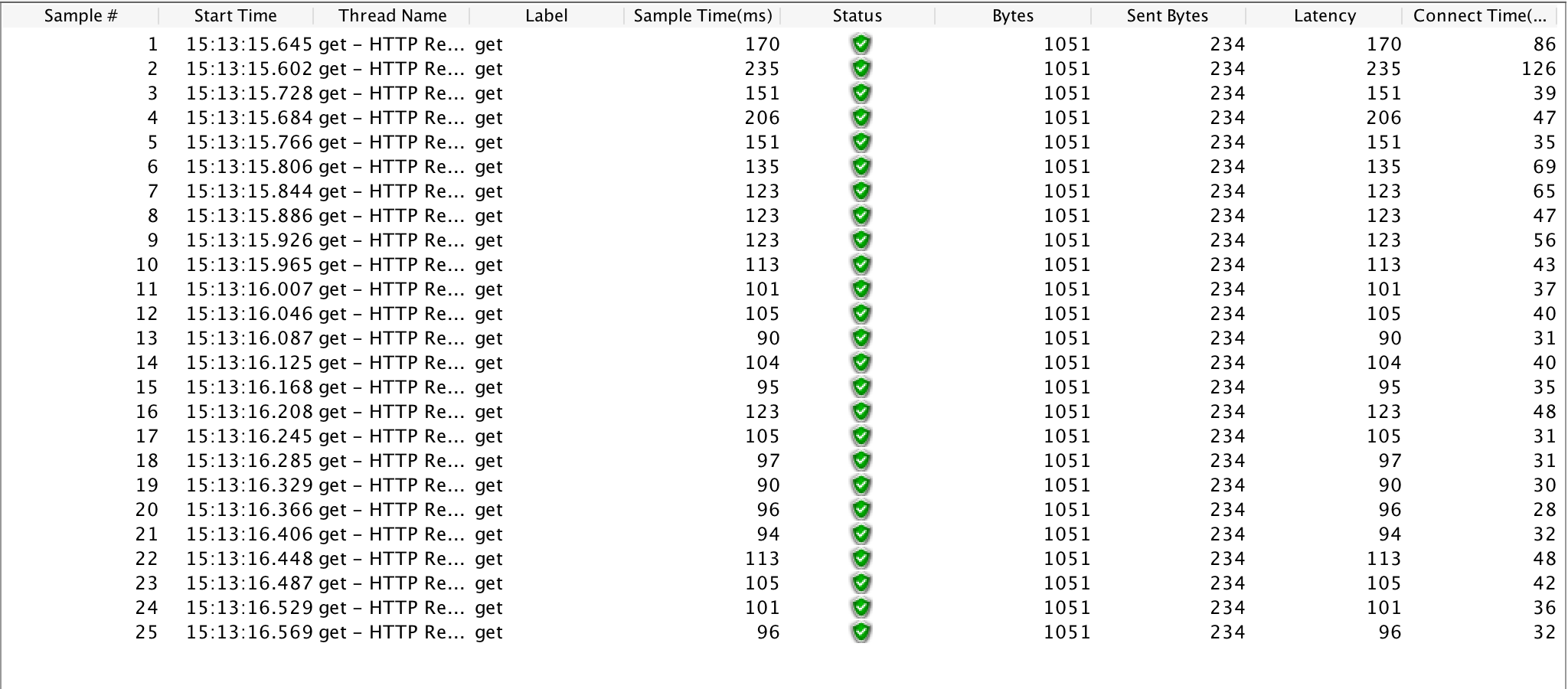
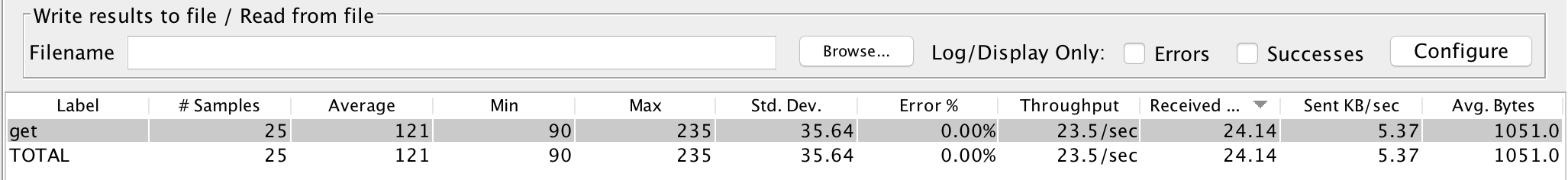
CS 483

December 11, 2017

Final Project Report

The testing tool I decided to use is called Apache JMeter. It is written in 100% Java making it easily runnable on almost any machine. JMeter is open source and available to anyone who would like to use it. In my research I could not find a paid version of the application. Due to the fact that it is free, it has become commonly used among testers. I found out that it is a respected testing tool that is used by people who have been in the industry for decades, making it fairly simple to find support using the tool if need be. Finding support on the web was not something that I had to do often for JMeter. JMeter has a great website explaining how to use the product. It has a section for each possible use with an example for each use. The one thing that I ventured to the web to find out about was how to interpret JMeter’s results. There were several great explanations that I found but then I realized I did not need these explanations because JMeter provides a glossary to explain their performance metrics. Apache does a great job by making their tool easily usable. There isn’t any true installation of the program. You download the files and then run the JAR file to fire up JMeter.

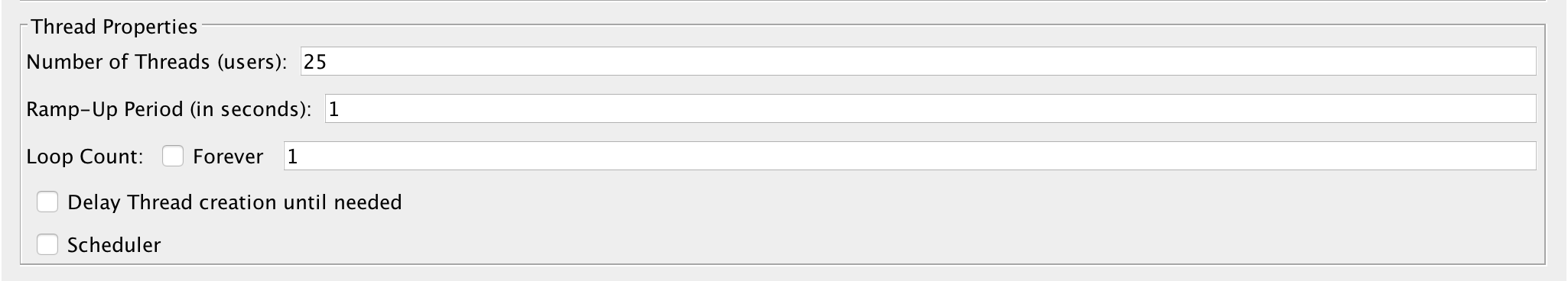
JMeter is used to load and performance test several different application/server/protocol types. There is a wide range of what JMeter is able to test, for example you could test the JDBC in order to see how well your database performs under the Java connection. You could also use to test your SMTP set up, which would basically tell you how well your machines ability to send and receive email is. The property of JMeter that I exploited was its ability to fire off HTTP requests and simulate hundreds or thousands of users. JMeter simulates X amount of users by creating X amount of threads to fire off the specific HTTP request. The one useful aspect of JMeter is also the ability to select the time frame that you want the number of requests to be sent off in. For example, you could simulate 100 users and have them all send off their requests in one second, or you could simulate 100 users sending off their requests in within a 10 second window so that 10 requests are sent per second. The one thing I discovered is that there are limitations to this. If you want all 10,000 users to send an HTTP request in under a second then you have to keep in mind your machines capabilities. I found that with even 1,000 users it was difficult to send off 1,000 HTTP requests in under a second, it always seemed to take longer.

After these requests are sent off, if properly configured, the program you are testing against will process these requests. JMeter then receives the response from the program you are testing against and gives you information about each individual request as well as information about all the requests as a whole. For each individual request it will tell you if the request was successful or not, at what time the request was sent, how long it took for the request from the time it was sent to the time the response was received, the number of bytes received in the response, the number of bytes the request itself sent, and then how long each request took to actually connect to the program you are sending the quests to. JMeter also provides metrics of these requests as a whole. It will tell you the average amount of time it took for a request to be processed, the minimum amount of time took for a request to be processed, the maximum amount of time took for a request to be processed, the percentage of error of all the requests, the average amount of bytes received from each response, the average amount of bytes sent with each request, and the throughput which is the number of requests processed per unit time. So in this case the throughput is calculated by the start of the first request to the reception of the response of the last request. These results could also be saved to CSV file, which makes it easily exportable to Microsoft Excel.

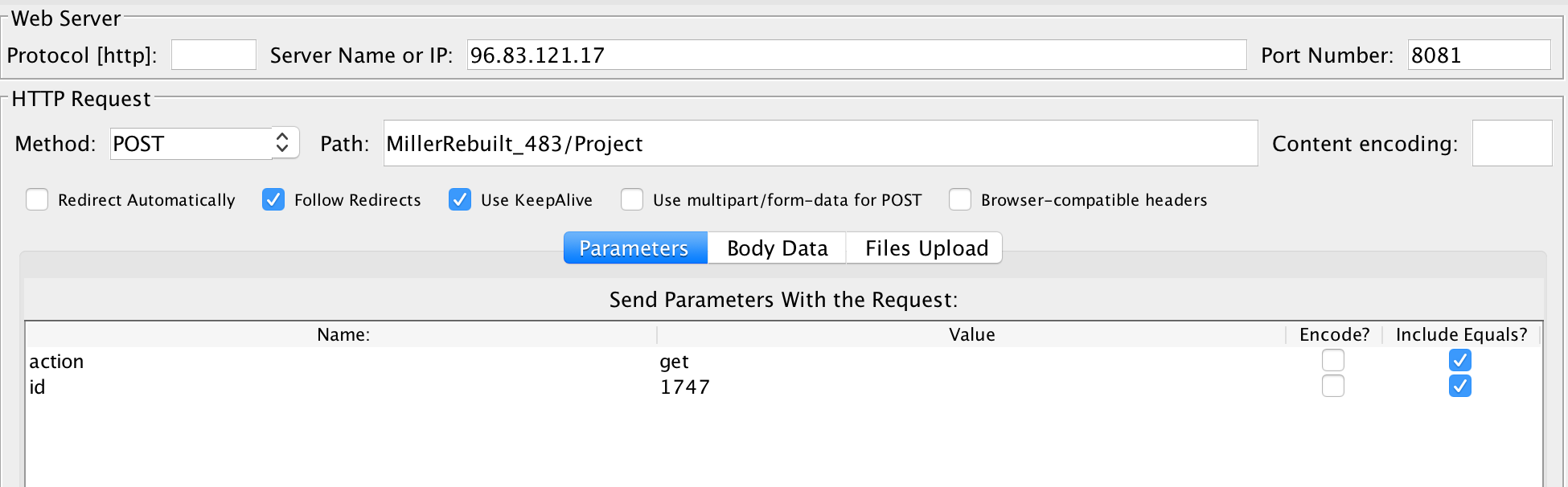
*Example summary report from Apache JMeter*

*Example results which are used to create the summary report*

The software under test was a fairly simple decision for me. In fact, the software under test drove my decision of what software testing tool I wanted to use in the first place. I am currently interning at a construction and refrigeration company and wanted to test the software out that I have been developing there. The software I have been developing there is a web application that uses a MySQL database in order to store the data. It utilizes a backend written in Java, so all of the Servlets that handle the HTTP requests are also in Java. Currently the website only has about 15 users, all from the construction side of the company, and has been performing perfectly. I know my boss’ overall goal is to eventually have the refrigeration side of the company use the website as well. Within the next year or so the amount of users on the site will most likely double or triple so I really wanted to see that the site would be able to handle the scaling. Because of this, JMeter seemed like a perfect testing tool to accomplish my goal.

The total amount of lines of code for the site is around 20,000+. A lot of the code is for the front-end part of the website which uses HTML, CSS, and JavaScript. The front-end part of the website is not testable with JMeter. However there Java backend has 60+ classes in it, with only 7 of them being Servlets. Originally only 2 of the servlets were really used, and I am trying to distribute the workload amount the other 5 servlets that I created in order to have a more intuitive design. Anyway, for this reasons I only really tested the one servlet because handles well over 90% of the requests. The servlet that I tested rigorously only contains 700 lines of code. I had to modify some of the code in order to be able to use JMeter to test it though. The only modification I made was to take out the security checking that makes sure that any request made to the servlets were from a valid user of the site. You become a valid user of the site after you log in, so any requests that JMeter initially sent received a VERIFICATION\_ERROR from the server making it impossible to test, although I was very happy to see that the verification was working properly. To compensate for this I just commented out the verification code so that any request could be processed. I actually made a copy of the company’s current database to test against as well as an insecure copy of the web application because I was uncertain about what 1,000 requests in less than one second could potentially do to a 2008 Microsoft server. I did not want to do any potential fata damage even though I was pretty positive no real harm would be done.

*Number of threads represents the number of users and the ramp up period represents the amount of time in which the users fire off their requests*



*Example HTTP Request*

I evaluated the tool by using (sort of) random HTTP requests. In class we never really went over the in depth details of performance testing. In fact, statement coverage, branch coverage, conditional coverage, context coverage, etc. were not very applicable for this type of testing. Well they might be applicable but I was unsure how to usefully apply them. In regards to the random HTTP requests, they were not 100% random. I chose to test a couple certain types of requests. I wanted to test the request that does the heaviest lifting, in this case that is the request that retrieves all of the construction projects from the database. I wanted to test the request that pulls all of the criteria from the database that the users can query projects from. I also wanted to test a request that updates a certain object in the database as well as a request that only pulls in one project from the database because this is most commonly done. Finally, I wanted to test a combination of these requests simultaneously to see how the web app’s performance changes.

*This graph shows how the throughput changes as the number of users changes. The throughput is per second and is on the Y axis and the number of users is on the X axis. All of the requests were sent within one second regardless of the number of users. The max throughput in the graph is 45.7 requests per second which was reached at 500 users which I was happy with. As you could see the throughput then began to drop after the users grew passed 500. This was because after 500 users, some requests started to fail. I took this as saying that the website should be able to withstand up to 500 users pretty well, even though wait times increase.*

*These two graphs were the best thing that came out of this project. An employee will often query projects based off of certain information such as project stage, project status, project manager, project type, project item, etc. The getAllObjects method retrieves the objects that you query off of from the database. The way I have the servlet set up is that depending on the action, the server response is written out to the log. The first graph demonstrates the throughput on the Y axis vs the number of users on the X axis. As you can see the throughput never climbs above 1. I then had the idea to have the servlet stop writing the response to the log if this certain action is required. The results were pretty cool because after removing the write to the log the throughput more than doubled which I consider a success, especially because this action is one of the most commonly invoked actions among users.*

*The first graph shows how the throughput climbs until it hits about 45 again. Again the throughput is on the Y axis and the number of users is on the X axis for the graph on the left. For the graph on the right the percent error is on the Y axis and the number of users is on the X axis. All of these requests were sent in less than a second. For this example there was a lot of error. I am unsure of the root of the error, but I suspect it has something to do with how you are trying to update the same record X amount of times in under a second. The percent error grew fairly linearly with the amount of users. This request is not incredibly realistic but it shows how the web app performs when multiple people are trying to simultaneously update the same record.*

Apache JMeter definitiely uses a white-box testing approach. There must be knowledge of the back-end of an application in order to properly use JMeter. If I did not know the code behind the software I was testing I would never have been able to run any useful tests on it using JMeter. JMeter is definitely useful for any application that communicates from computer to computer via a network because it is able to load test how well and how quickly one computer is able to send data. It is a useful tool that could definitely help someone out in figuring out how much their program can handle.

There are alteratives to Apache JMeter though, for example there is the Gatling Tool, Locust, and the Grinder. They all perform similar tasks in load testing an application. The one major difference is the fact that JMeter has been around since 1998 so it is a mature testing tool at this point. JMeter has over 500K lines of code while out of the three I listed above the Gatling Tool has the most lines of code with 50K. There are some things that the Gatling Tool does that Locust cannot do. And there are things Locust could do that Gatling cannot do. The one thing I found was that JMeter encompasses all the features of the three listed above. Another major advanatge of JMeter is that there is a GUI to make it much easier. The other tools use some sort of scripting to accomplish whatever load testing you wanted to get done.

Overall I really enjoyed using and exploring JMeter. It is incredibly simple to set up and start using. Due to the fact that it has been around for almost 20 years now, if the Apache JMeter website does not help you figure out how to use it then there are plenty of other people throughout the web that have probably encounrtered your issue before which is one benefit of using a mature software like this. I will definitely remember JMeter for the future. It can be applied to any software that handles HTTP requests, and since HTTP requests are so commonly used I would say JMeter is very effective for the amount of software it could test. I am surprised that JMeter was a free tool to use considering how much it could do. I am thrilled that I was able to do my final project on a testing tool that actually yielded me useful results for the current work I am doing. It is nice to now know that the web application I’ve been developing seems to be scalable to the future needs of my boss. In summary I would recommend JMeter to anyone that needs to do any sort of performance/load testing because chances are JMeter has a way to test your implementation.