**Enterprise Performance Architecture**

**CA2 – Load Testing**

**Dónal McManus X00099816**

**Overview – with appendix A ………………………… Page 1**

**Machine Statistics ………………………………………… Page 2**

**Utilisation – with appendix B ……………………….. Page 2**

**Service Demand – with appendix C ………………. Page 3**

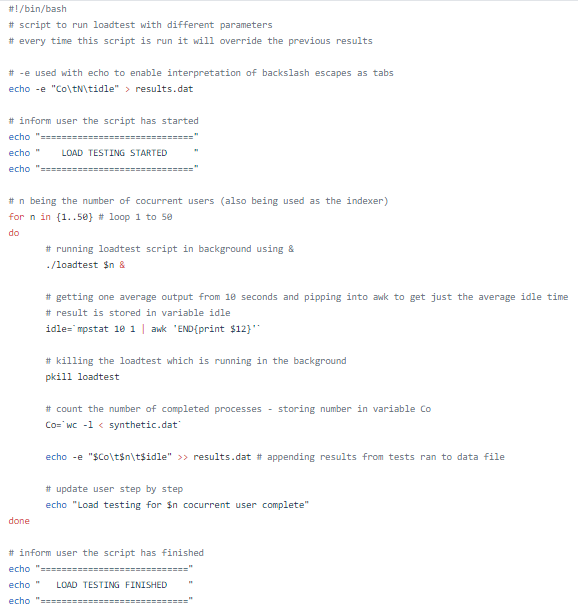
**Throughput – with appendix D …………………….. Page 3**

**Response Time – with appendix E ………………… Page 4**

**Overview**

The goal of this assignment was to analyse the throughput (Xo), utilization (Ui), service demand (Di) (i referring to the number of CPU’s in this case 1) and response time (R) of a single CPU as the number of concurrent users increases from 1 to 50. Testing is done by running a script that I wrote which in turn runs a load test which was written in C, the load test takes a parameter which is the number of concurrent users we want to simulate (this is out N value) this is run in the background, so we can perform of testing. Utilization is gathered from mpstat. The parameters added to mpstat was 10 and 1. The ten as that runs for ten seconds gathering information the entire time. The 1 that follows gives one output which averages the information gathered from the ten seconds. This was then pipped into awk. awk scans the output for the field I asked for being 12 (which is the idle value) the which must be prefixed with the dollar sign. I also use ‘END’ in my command to get the last line of the output which is being scanned in this case that is the average result of the 10 seconds. pkill then kills the background task loadtest. The load test writes to synthetic.dat while its running, using wc -l will count the number of transactions complete (Co) using the < argument will give us the number without the filename. These results are then appended to results.dat.

This is the script used below. **Appendix A**



**Machine Statistics**

The virtual machine I used for this assessment was ran using virtual box. The operating system was Fedora (64-bit). I assigned 5027 MB for RAM ensure for this case only one CPU was given to the VM. Video Memory was put up to 128 MB with Disk storage using the SATA controller with storage being of virtual size 17.61 GB with actual size being 10.07 GB, this is dynamically allocated storage. GHZ??

**Utilisation Appendix B**

**Ui formula;**

**Ui = 100 – idle time**

The above graph was created from plotting the values Ui and N. As the number of concurrent users increase the CPU utilisation maxes out at 100 when N gets to ~21 and remains at full utilisation until the end (N = 50). From N=1 => N=~21 the graph indicates a logarithmic pattern up until this point and then levels off as utilisation is maxed out at 100%. This is an expected result as the more simulated users are added the more stress the CPU would be under to complete all requests leaving no idle time until it can’t perform any faster which we can see with no idle time. In theory the CPU will always reach maximum utilisation as long as N increases, but the rate at which Ui approaches 100% utilisation is going to be down to the capability of the CPU power.

**Service Demand Appendix C**

**Di formula;**

**Di = Ui / Xo**

Service demand is graphed here using the results from Di – in milliseconds and N. Service demand is the average time spent by a request in obtaining the resource it requested in our case the single core CPU. The time taken for the load test to access the CPU is ~15 milliseconds. There are two notable spikes in this graph when N=~25 (33% increase) and at N=~36 which is an 85% increase in time, this indicates that during these tests it took longer to get CPU time perhaps another process running on the VM used of some CPU time restricting access for the load test. Ideally there would be no spikes which would mean the test had uninterrupted access to the CPU which would evaluate to a relatively flat line. The most straightforward way to reduce this number would be to improve the CPU, further improving the script for better performance could also achieve this but not at the same scale of improving the hardware.

**Throughput Appendix D**

**Xo formula;**

**Xo = Co / T (where T is time in this case 10 seconds)**

Here is where we graphed the throughput. This is equal to the number of completed requests (Co) divided by the observation interval which was 10 seconds. I noticed that after graphing the results there is a logarithmic curve at the beginning just like in **[Appendix B]** leading me to believe that with the rise in CPU utilisation so with it rose the throughput. The expected result would be a rise as see as N is between 1 – 18. For this period of testing there is a general pattern of increase. After this comes and unexpected decrease in throughput, the biggest drop was when N=~36 with a 83% dip in throughput and looking at **[appendix C]** there is a correlating spike here to leading me to believe that due to the load test not getting the CPU time as quickly this has the knock on effect to the throughout as no transactions can be completed while the load test is waiting on the CPU.

**Response Time Appendix E**

**R formula;**

**R = N / Xo – using Little’s Law**

Average response time is measured by the number of concurrent users divided by the throughput. Where N is between 1 and 33 the graph is linear although there is a minor spike where N = 24 which can be correlated to **[appendix C]** and **[appendix D]**, in **[appendix C]** there was a spike in service demand meaning the test could not use the CPU time which would in turn affect the response time, the effect is a jump of ~25% in **[appendix E]** taking into account the linear progression compared to the jump of 33% in **[appendix C]** my assumption would be that the variance here is caused due to R being the average response time and the averaging diluted the affect somewhat. The major spike when N=36 is seen on both **[appendix C]** and **[appendix D]** it is also noted that utilisation reached 100% at this time. My remarks on this jump in longer response time would be the same as the minor spike, the service demand was higher and the throughout was considerably lower at N=36 shown in both **[appendix C]** and **[appendix D]** respectively leading to a increase in response time of 100%.