EPA CA2

Aaron Mulvaney – X00118910

Enterprise Performance Architecture

IT Tallaght

Computing – Year 4

# Table of Contents

[Performance Analysis 2](#_Toc501036409)

[Overview 2](#_Toc501036410)

[Machine Characteristics 2](#_Toc501036411)

[Graph Characteristics/Discussion 3](#_Toc501036412)

[Graphs 4](#_Toc501036413)

[Appendix 6](#_Toc501036414)

[Appendix A 6](#_Toc501036415)

[Appendix B 7](#_Toc501036416)

[Appendix C 8](#_Toc501036417)

# Performance Analysis

Overview

The purpose of this performance analysis was to analyse throughput, utilization, service demand and response time of the CPU (Central Processing Unit) on a single core Linux virtual machine, while the number of concurrent users (N) running a load test increases from 1 to 50 inclusive (1 ≤ N ≤ 50). All values needed to calculate these metrics were gathered by executing the runtest.sh script below (see Appendix A). The script uses loadtest.sh which was generated when loadtest.C (see Appendix C) was compiled on the machine in question. When the loadtest.sh was run on the machine a file (see Appendix B) containing the number of completions for each loadtest (C0), the number of concurrent users running the loadtest (N) and the idle time of the system after the run time for each loadtest (T). All the values above were used to calculate important metrics which allow one to get an analytical view of the system over the total test period.

T = 5 seconds

Idle = idle time (Given as %)

N = 1 ≤ N ≤ 50

Ui = Utilization (Given as %)

Di = Mean Service Time (Given in milliseconds)

X0 = Throughput (Measured in tps – transactions per second)

R = Average Response Time (Given in seconds)

Machine Characteristics

The operating system I carried out this performance analysis on was a Linux Fedora (64-bit) virtual machine, composed of a single core CPU running at 2.5GHz. Using 4096Mb of RAM (Random Access Memory) and 39Gb(vmdk) of dynamically allocated storage.

Graph Characteristics/Discussion

Formula for Ui:

Ui = 100 – Idle time

Formula for X0:

X0 = C0 / T

Formula for R:

R = N / X0

Formula for Di:

Di = Ui / X0

The graphs below show the Utilization, Throughput, Mean Service Time and Response Time while the number of concurrent users increases from 1 to 50. Saturation of the CPU was achieved with Utilization reaching its peak at 99.8%. Throughout the graphs some trends have arisen, Ui Vs N, X0 Vs N and R Vs N all carry a steady rise with no extreme peaks or drops. However, there are considerable increases and decreases scattered throughout. A sharp rise of over 8% in Utilization can be seen when N is increased from 22 to 23, this may be because the throughput in that time was one of the highest increases in throughput per loadtest over the whole performance test, with 31.4 tps at N = 22 and 38.6 tps at N = 23. A similar pattern can be seen when N is increased from 24 to 25. The same trend can also be visualised in the C0 Vs N graph. Response time for these peaks was lower which suggests that the transactions were being handled at a faster rate.

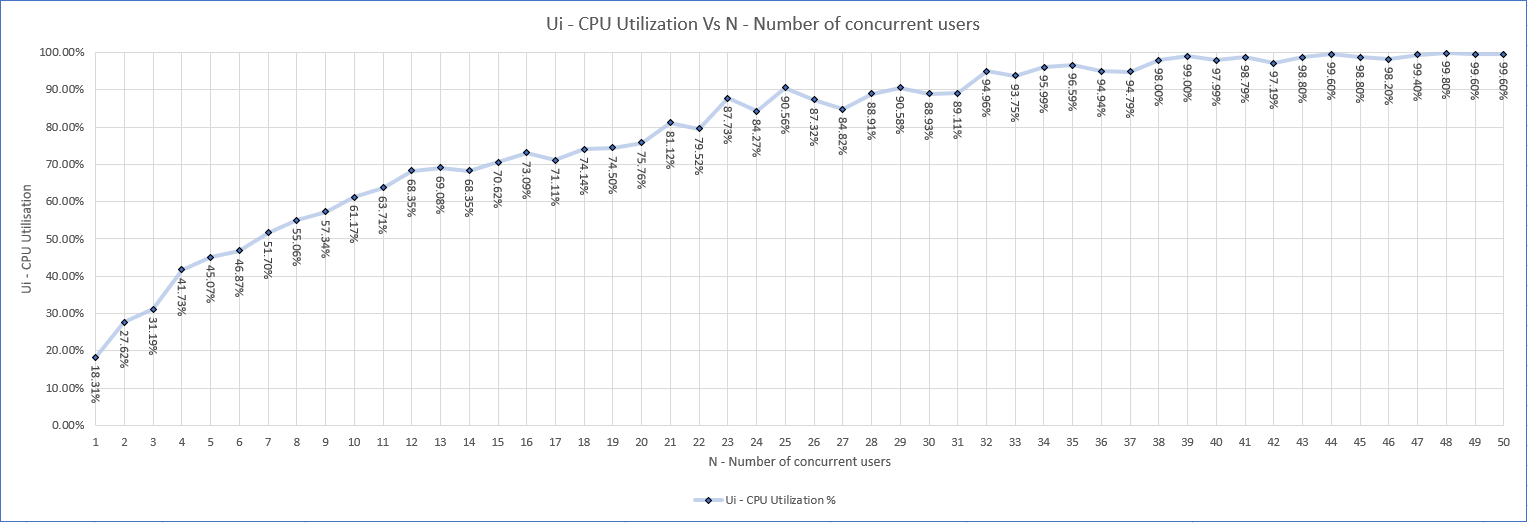
Contrary to the sharp rises, dips are notable in between each of the peaks mentioned above, for example when N is increased from 23 to 24 a ripple effect takes place affecting all metrics, the Utilization and Throughput decreased while the Response time increased showing that the reason the decrease happened was because the transactions weren’t being responded to as quick as they were in the loadtest prior and after.

With regards to Di or Service Demand, there is only one plummet in the graph, for the first user N, the service demand(Di) was 4.16 milliseconds. In comparison to when N is 2 the Di was 0.73 milliseconds, this is a massive jump if you compare this difference to the rest of the graph. It suggests that the first loadtest that was run took time to get access to the CPU in order to utilize its resources as Di is defined as the average time spent by a request waiting to obtain a service from the CPU in this case. For the rest of the Di Vs N graph there were no anomalies to note, this may propose that the interaction between the loadtest and the CPU was uninterrupted for the duration of the performance test, it looks like no other request bore a heavy load on the CPU at that time.

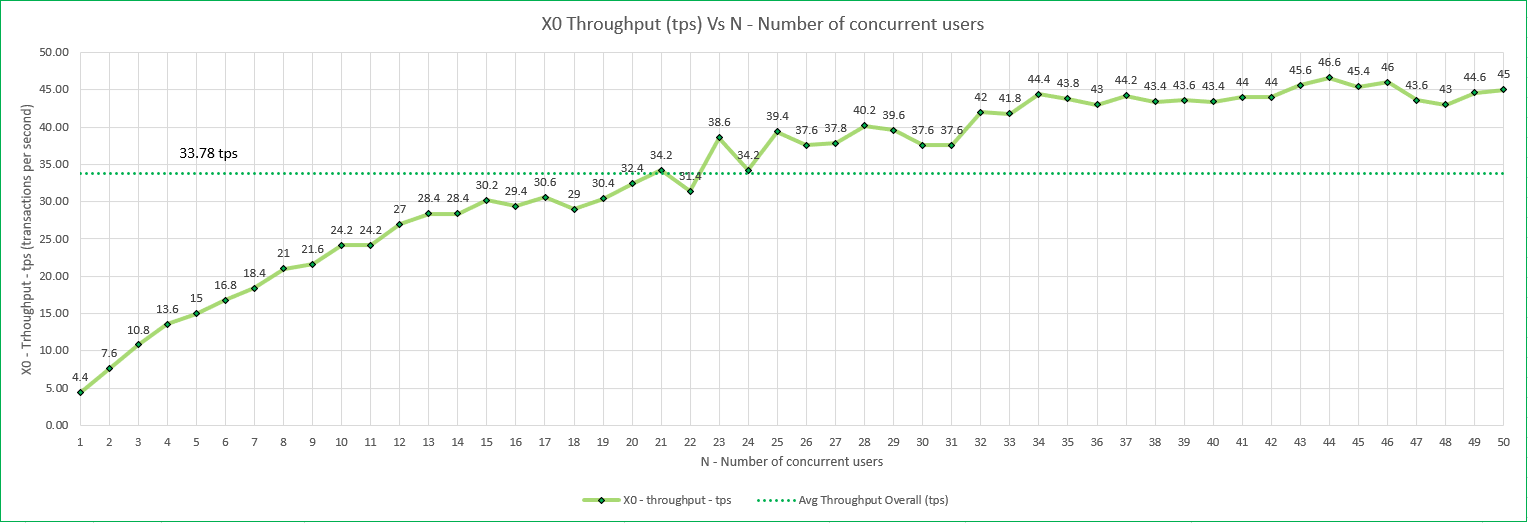
Some subtle trends between the X0 Vs N and R Vs N graphs can be identified. For instance, the last 4 stages of the performance test, 46 ≤ N ≤ 50. The graphs are almost the opposite in this case, the Throughput declines whilst the Response time increases showing a direct correlation between the two. The contrast can also be detected at 22 ≤ N ≤ 25, this time it is also visible in the Ui Vs N graph, again highlighting interaction between the 3 metrics.

Graphs

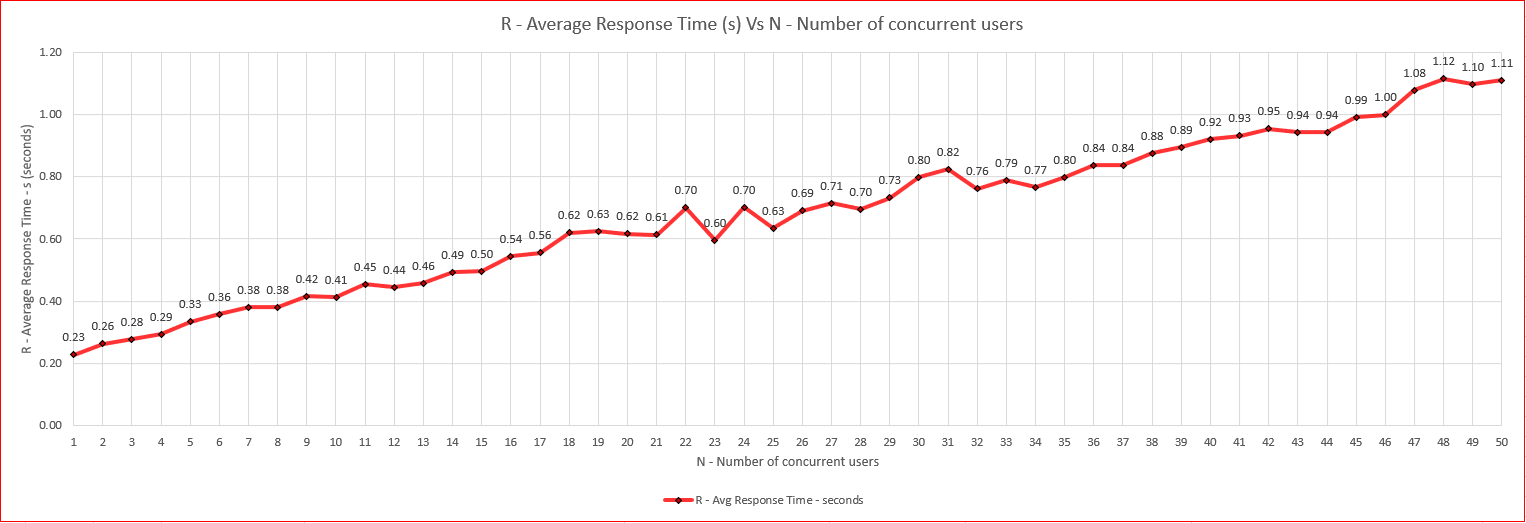
Ui – CPU Utilization Vs N – Number of Concurrent users



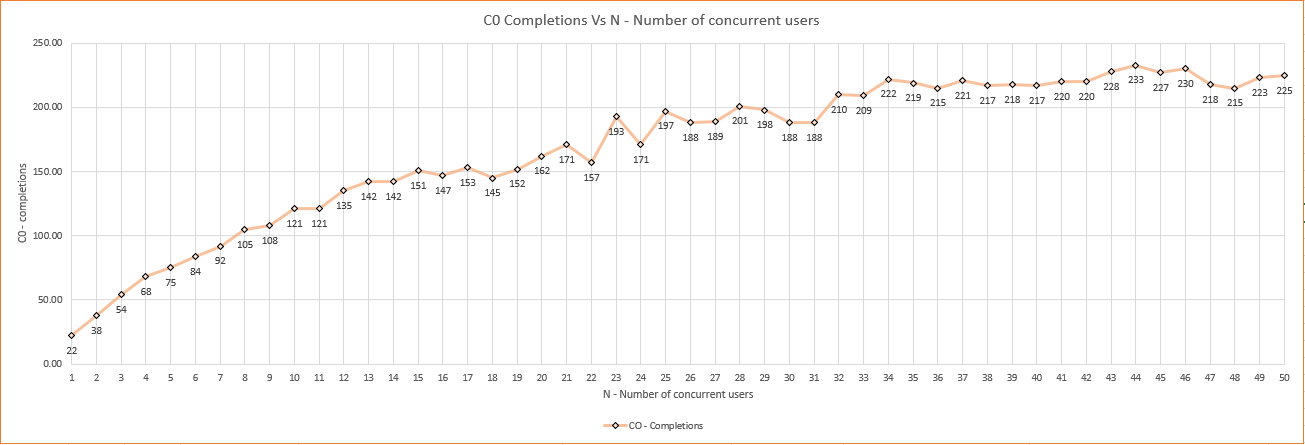
X0 – Throughput (tps) Vs N – Number of Concurrent users



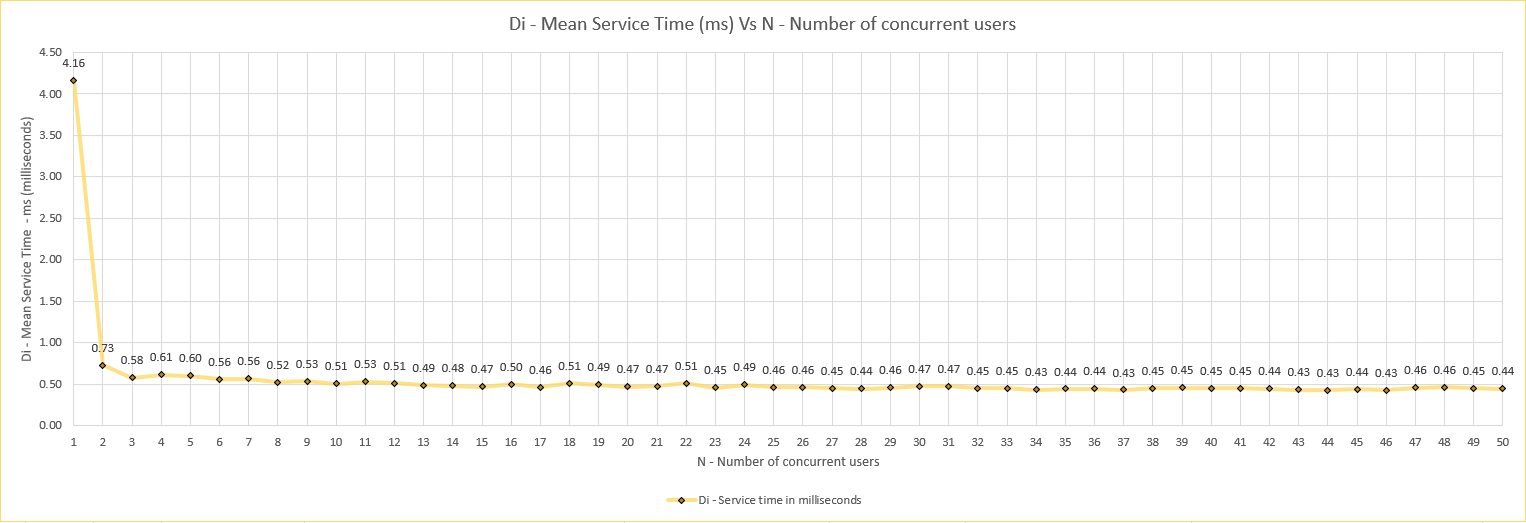
R – Response Time (s) Vs N – Number of Concurrent users



C0 - Completions Vs N – Number of Concurrent users



Di – Mean Service Time (ms) Vs N – Number of Concurrent users



# Appendix

Appendix A

**runtest.sh**

#!/bin/bash

#Echo the title for the output table to file named results.dat

echo -e "CO\tN\tidle" > results.dat

#Some output to let the user know what is happening

echo "Each loadtest will terminate after a 5 second run time"

echo "This may take a few minutes..."

#for loop to increase the number of concurrent users every loop starting at 1 up to 50

for i in {1..50}

do

#run the loadtest in the background

./loadtest $i &

#set variable idleTime to be equal to the system idle time after the loadtest has ran for 5 seconds

idleTime=`mpstat 5 1 -o JSON | jq '.sysstat.hosts[0].statistics[0]."cpu-load"[0].idle'`

#kill the loadtest process

pkill loadtest

#set the number of completions for this loadtest

completions=`wc -l < synthetic.dat`

#output the number of completions, number of concurrent users and the system idle time for each loadtest

echo -e "$completions\t$i\t$idleTime" >> results.dat

done

echo "Loadtest complete for 1..50 users, please see results.dat for the results of this test"

Appendix B

**results.dat**

CO N idle

22 1 81.69

38 2 72.38

54 3 68.81

68 4 58.27

75 5 54.93

84 6 53.13

92 7 48.3

105 8 44.94

108 9 42.66

121 10 38.83

121 11 36.29

135 12 31.65

142 13 30.92

142 14 31.65

151 15 29.38

147 16 26.91

153 17 28.89

145 18 25.86

152 19 25.5

162 20 24.24

171 21 18.88

157 22 20.48

193 23 12.27

171 24 15.73

197 25 9.44

188 26 12.68

189 27 15.18

201 28 11.09

198 29 9.42

188 30 11.07

188 31 10.89

210 32 5.04

209 33 6.25

222 34 4.01

219 35 3.41

215 36 5.06

221 37 5.21

217 38 2

218 39 1

217 40 2.01

220 41 1.21

220 42 2.81

228 43 1.2

233 44 0.4

227 45 1.2

230 46 1.8

218 47 0.6

215 48 0.2

223 49 0.4

225 50 0.4

Appendix C

loadtest.C

/\*

\* This is Linux ONLY code

\* Do not compile on windows

\* compile this file using the command

\* g++ -fopenmp loadtest.C -o loadtest

\* once compilation is successful run the test using

\* ./loadtest

\* Ctrl C to stop the test

\*/

#include <iostream>

#include <fstream>

#include <cmath>

#include <unistd.h>

#include <omp.h>

#include <cstdlib>

using namespace std;

void sleep\_ms(int milliseconds) ;

int main(int argc, char \*\*argv) {

srand(time(NULL));

ofstream fptr;

if (argc != 2) {

cout << "useage: loadtest N" << endl;

cout << "where N is the number of concurrent users from 0-100" << endl;

exit(1);

}

int nthreads = atoi(argv[1]);

int tid;

fptr.open("synthetic.dat", ios::trunc);

omp\_set\_num\_threads(nthreads);

omp\_lock\_t writelock;

omp\_init\_lock(&writelock);

#pragma omp parallel private (tid)

{

tid = omp\_get\_thread\_num();

#pragma omp parallel for

for (int t = 0; t < 10; t++) {

for (; ;) {

for (int i = 0; i < 1000000; i++) {

double t = rand() % 8273 + 1;

double tt = sqrt(t);

}

time\_t t = time(0);

struct tm \*now = localtime(&t);

omp\_set\_lock(&writelock);

fptr << (now->tm\_year + 1900) << '-'

<< (now->tm\_mon + 1) << '-'

<< now->tm\_mday << '-'

<< (now->tm\_hour) << '-'

<< (now->tm\_min) << '-'

<< (now->tm\_sec)

<< " UserID [" << tid << "]"

<< " Transaction Complete " << endl;

omp\_unset\_lock(&writelock);

sleep\_ms(100);

}

}

}

fptr.close();

return 0;

}

void sleep\_ms(int milliseconds)

{

struct timespec ts;

ts.tv\_sec = milliseconds / 1000;

ts.tv\_nsec = (milliseconds % 1000) \* 1000000;

nanosleep(&ts, NULL);

usleep(milliseconds \* 1000);

}