**Computer Systems Technology**

British Columbia Institute of Technology

COMP 8005 - Assignment2- Report

Albert Huang&

Aiyan Ma

Mar 5, 2018

# Table of Contents

[Table of Contents 2](#_Toc1667016842)

[1. Summary 3](#_Toc874767404)

[2. Introduction 4](#_Toc2139831603)

[3. Background 5](#_Toc266633914)

[4. Discussion 6](#_Toc1577232796)

[5. Conclusion 12](#_Toc568634831)

# Summary

The purpose of this work is to design and compare the 3 models of doing high performance servers:

1. Multi-thread ( traditional way)
2. Poll ( Level ttrigger)
3. EPoll ( Edge trigger)

To archive it, Three servers are designed and compared by using java nio library.

For EPoll and Poll server, message cache, state machine with java nio are utilized with the intention to enhance the scalability and performance ( see details in the design work).

For the Multi thread server, one thread per connection model is used to accept and handle the client request.

The client, named SuperClient, also designed to connect and send packets in such a way that connections and packets are generated as much as possible. For these 3 models, the scalability(connection) and the performance both are analyzed.

To analyze and compare these three server mode, a set of performance tests were developed to help show any differences between them. Testing programs were setup on lab’s computers in attempts to test and analyze performance, notably in regards to time consuming, CPU usage etc.

My hypothesis was that the multi-threaded app would be outclassed by the my multi-processed app and the single-threaded app in turn would be outclassed by both of the multi-thread and multi-processed app. But our results proved more complicated. Time consuming were comparable between the different methods, but performance of my multi-threaded app was unable to defeat the multi-processed app. The multi-threaded app performance in some case is even worse than single-threaded one.

My discoveries were interesting. All apps slowed considerably as the calculation or IO workload went up. My multi-threaded approach show its advantage only in Internet requests works. While the multi-processed approach wins in both CPU-bound and IO-bound test, and in Internet request test both multi-threaded and multi-processed approach perform in the same level, very little difference.

# Introduction

The purpose of this assignment was to analyze and compare the performance between the multi-threaded, multi-processed method implementations. The tests were developed using Python 2.7 and were run on the Linux Fedora OS using Datacomm lab computers and Ubuntu OS using my laptop and Raspbain OS using raspberry Pi 3.

The goal of this assignment is to have a better understanding of the way different types of implementation perform on same platforms compared to each other.

# Background

To analyze and compare the performance of multi-thread, multi-processed approach, python programs were written for all three servers types as well as a common test using multi-processing and multi-threading.

The Multi-Threaded app creates a set of worker threads to manage each performance test. No data is need to receive and echoed back to main stream. When the work threads finish one test, then the main stream will print the time consuming message and generate a bunch of worker threads to perform the next test. These worker thread would experience three types of test. The first one is CPU-bound test, in this test, each worker thread would do 1,500,000 times adding calculation. The second test is IO-bound test. In this test, each thread would write millions of lines repeated words to a file, and then read this in to list. The last test is http request job, it’s just make the thread request a web page and return the web page.

The Multi-Processed app actually do the same thing as the Multi-Threaded app. The only different part is instead of using a bunch of working thread to do those test, this app generate a set of child processes. It also records those time consuming for these three type of tests.

# Discussion

My hypothesis was that the multi-threaded app would be outclassed by the my multi-processed app and the single-threaded app in turn would be outclassed by both of the multi-thread and multi-processed app.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| server type | # conn | # host | # clients/host | avg res | 2k | 4k | 4.8k |
| multi-Thread | 4k | 1 | 4800 | 652.639 | 0.859 | 165.499 | 347.516 |
|  | 10k | 2 | 4800 | N/A | N/A | N/A | N/A |
|  | 20k | 5 | 4800 | N/A | N/A | N/A | N/A |
|  | 40k | 10 | 4800 | N/A | N/A | N/A | N/A |
|  | 100k | 24 | 4800 | N/A | N/A | N/A | N/A |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| level trigger | 10k | 2 | 4800 | 0.101 | 0.2 | 0.178 | 0.162 |
|  | 20k | 5 | 4800 | 0.089 | 0.209 | 0.139 | 0.129 |
|  | 40k | 10 | 4800 | 0.087 | 0.177 | 0.156 | 0.155 |
|  | 100k | 24 | 4800 | 0.07 | 0.227 | 0.194 | 0.159 |
|  |  |  |  |  |  |  |  |
| edge trigger | 10k | 2 | 4800 | 0.157 | 0.281 | 0.206 | 0.187 |
|  | 20k | 5 | 4800 | 0.105 | 0.222 | 0.186 | 0.182 |
|  | 40k | 10 | 4800 | 0.108 | 0.297 | 0.254 | 0.257 |
|  | 100k | 24 | 4800 | 0.103 | 0.291 | 0.225 | 0.213 |

To test my hypothesis, a formal series of tests and stress tests were run with in various platforms. The time consuming and memory&CPU usage were averaged and compared. I also collect the single-threaded approach for proving.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| server type | # host | avg res | 10k | 20k | 40k | 80k | 110k |
|  |  |  |  |  |  |  |  |
| level trigger | 2 | 0.101 | 0.101 |  |  |  |  |
|  | 5 | 0.089 | 0.107 | 0.084 |  |  |  |
|  | 10 | 0.087 | 0.111 | 0.103 | 0.086 |  |  |
|  | 24 | 0.07 | 0.115 | 0.085 | 0.081 | 0.071 | 0.058 |
|  |  |  |  |  |  |  |  |
| edge trigger | 2 | 0.157 | 0.164 |  |  |  |  |
|  | 5 | 0.105 | 0.125 | 0.106 |  |  |  |
|  | 10 | 0.108 | 0.184 | 0.141 | 0.114 |  |  |
|  | 24 | 0.103 | 0.175 | 0.142 | 0.127 | 0.101 | 0.098 |

# Conclusion

From the data collected, it can be seen that Multi-processed approach is the superior choice for managing high pressure CPU-bound job and IO-bound job. The time consuming for both approach were similar for web request job test and the Multi-Threaded approach can save some memory.

The Multi-Threaded approach performed not good, even compare with the single-threaded approach in CPU-bound test and IO-bound test, Multi-Threaded approach have no much advantage. Consider about the Cpython have GIL(Global Interpreter Lock), therefore, the thread is concurrent but not parallel. So when we are running CPU-bound job, the Multi-Threaded approach perform worse than single-threaded. While in web request job, the Multi-Threaded approach is more comfortable than single-threaded.

The Multi-Processed approach perform much better in both CPU-bound and IO-bound tests. And in web request job it still good. The only problem is this approach occupy much more memory resources than other approaches. So we may choose Multi-Threaded approach more in web request task.