**Computer Systems Technology**

British Columbia Institute of Technology

COMP 8005 - Assignment2- TP&TR

Albert Huang &

Aiyan Ma

Feb 26, 2018

Revision Control

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Date** | **Version** | **Revised by** | **Remarks** |
| 1 | 2018-Feb-26 | V0.1 | Aiyan,Ma | The initial draft for Test plan and test case |
| 2 | 2018-Mar-03 | V1.0 | Albert Huang | Update for the test execution and report. |

# Table of Contents

[Table of Contents 2](#_Toc13407)

[1. Test Objective 3](#_Toc25756)

[2 Test Scope 3](#_Toc1334)

[3 Test case Design 3](#_Toc21729)

[1 Performance : 3](#_Toc16680)

[Java nio: 3](#_Toc23405)

[Python: 4](#_Toc23682)

[2 Scalability Maximum 4](#_Toc25095)

[Java nio 4](#_Toc3543)

[Python 4](#_Toc4412)

[4Test Tools and Test Config 5](#_Toc32446)

[1 Client Tools 5](#_Toc27287)

[2 Performance tools 5](#_Toc30622)

[3 Scalability Tools 5](#_Toc21957)

[4 Application Statics 5](#_Toc1736)

[5 Test Result 5](#_Toc16326)

[6 Conclusion 5](#_Toc25964)

[7 Further Work 6](#_Toc9354)

[8 Reference 6](#_Toc12590)

[9Appendix 6](#_Toc24249)

[1 Optimize System for best performance 6](#_Toc1892)

[2 Test Screen Capture 6](#_Toc13939)

[Java 7](#_Toc14147)

[Java Poll: 7](#_Toc12103)

# Test Objective

This is the scalability and performance test for high performance server designed in this assignment.

For each of the so called high performance server, it has two objectives:

1. Scalability: The “connection” of each server can keep
2. Performance: The through out of the high performance server can process

To measure it, the statics and the 3rd part tools may be used to collect performance data.

# 2 Test Scope

This test will cover the following:

1. Java Platform
2. Python Platform
3. Compare of the both

On Performance and Scalability .

# 3 Test case Design

## 1 Performance :

### Java nio:

|  |  |  |  |
| --- | --- | --- | --- |
| High Performance Server | Scalability  Connections | Performance  Incoming Rate(bps) | Performance  OutGoing Rate(bps) |
| EPoll |  |  |  |
| Poll |  |  |  |
| Multi-Thread |  |  |  |
| Multi-Thread-Pool |  |  |  |

### Python:

|  |  |  |  |
| --- | --- | --- | --- |
| High Performance Server | Scalability  Connections | Performance  Incoming Rate(bps) | Performance  OutGoing Rate(bps) |
| EPoll |  |  |  |
| Poll |  |  |  |
| Multi-Thread |  |  |  |

## 2 Scalability Maximum

It’s noticed that the scalability of the high performance limited by the client connections, each client, if sending multiple packets in one connection, it can only have around 4882 connections. So we modified the super-client to send only one round of packets per connection and therefore to create maximum of connections at client side.

### Java nio

|  |  |  |  |
| --- | --- | --- | --- |
| High Performance Server | Scalability  Connections | Performance  Incoming Rate(bps) | Performance  OutGoing Rate(bps) |
| EPoll |  |  |  |
| Poll |  |  |  |
| Multi-Thread |  |  |  |
| Multi-Thread-Pool |  |  |  |

### Python

|  |  |  |  |
| --- | --- | --- | --- |
| High Performance Server | Scalability  Connections | Performance  Incoming Rate(bps) | Performance  OutGoing Rate(bps) |
| EPoll |  |  |  |
| Poll |  |  |  |
| Multi-Thread |  |  |  |

# 4Test Tools and Test Config

## 1 Client Tools

1) SuperClient:

2)SuperClientls: SuperClient with loop send ability

## 2 Performance tools

Realtime through output ( incoming rate and outgoing rate) is watched by iptraf tool( <http://iptraf.seul.org/about.html)> , captured the connection and through out are stable

## 3 Scalability Tools

Connections at server side are monitored by netstat every 1 sec until it comes to stable

## 4 Application Statics

There are counters in the application, for instance the connection counter, the packets conunter, packets length to indicate the packet sent and received correctly, message ids etc. These statics data is considered as aids for the sand alone tool by Linux system and 3rd part tools, to keep the test result fair enough.

# 5 Test Result

# 6 Conclusion

# 7 Further Work

The following technology and Epoll provide can be used for further work:

1 The message middle ware such as JMS ( Java Message Service <https://en.wikipedia.org/wiki/Java_Message_Service> ) can be leveraged to further bench mark purpose.

2 Another Java EPoll provider named wizzardo <https://github.com/wizzardo/epoll> also did a lot of work on Selector Provide for epoll.

3 IBM epoll provider for java epoll provider : -Dcom.ibm.nio.rdma.EPollSelectorProvider

<https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/diag/appendixes/cmdline/dcomibmniordmaepollselectorprovider.html>

# 8 Reference

# 9Appendix

## 1 Optimize System for best performance

In Java mode, doing the following

1 increase the fd to 100K

#ulimit -n 102400

2 increase the heap memory

#java -Xss1024m -Xms4096m -Xmx5120m

## 2 Test Screen Capture

Part of the screen captured as below for demonstration purpose.

### Java

### Java Poll:

