# TEST BENCH

## 1 Environment

The test is processed under the environments:

|  |  |
| --- | --- |
| Item | Description |
| Operating System | Windows 10 – CMD or PowerShell |
| Java version | JRE 1.8 |
| others | N/A |

## 2 Server Failover

This test is on local hostname 127.0.0.1, servers are distinguished by port number:

1. **Creation of a 5 nodes system**

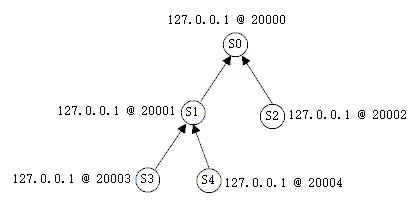


Figure 1: Sample Architecture of Test

Running instructions in 5 PowerShell Console Separately:

|  |  |
| --- | --- |
| Server | Instructions |
| S0 (ROOT) | java -jar Server.jar -lh 127.0.0.1 -lp 20000 -a 1000 -s abcde |
| S1 (🡪S0) | java -jar Server.jar -lh 127.0.0.1 -lp 20001 -a 1000 -s abcde –rh 127.0.0.1 –rp 20000 |
| S2 (🡪S0) | java -jar Server.jar -lh 127.0.0.1 -lp 20002 -a 1000 -s abcde –rh 127.0.0.1 –rp 20000 |
| S3 (🡪S1) | java -jar Server.jar -lh 127.0.0.1 -lp 20003 -a 1000 -s abcde –rh 127.0.0.1 –rp 20001 |
| S4 (🡪S4) | java -jar Server.jar -lh 127.0.0.1 -lp 20004 -a 1000 -s abcde –rh 127.0.0.1 –rp 20001 |

In the console for each server, the Father Node can be found:

e.g. The following is the Console info of node S4, whose Father Node is S1

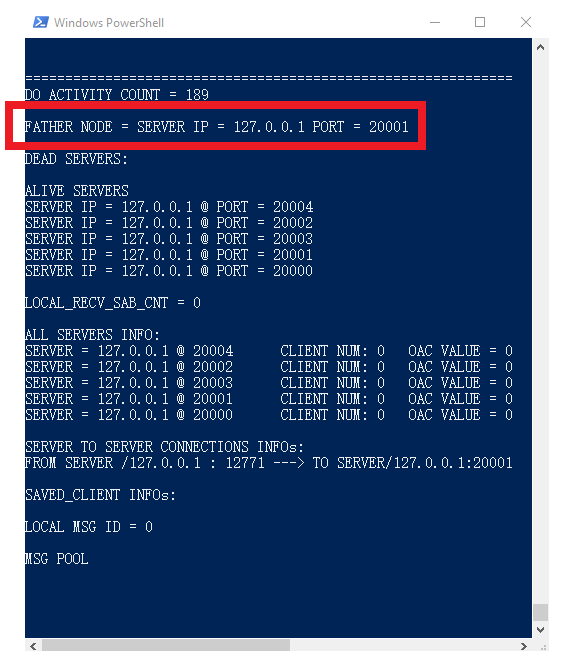


Figure 2: Console info for S4, noted with its Father Server

1. **Shutdown the ROOT node**

On shutting down the ROOT server S0, after a certain time span, each server would recognize the Failure of S0, depicted as following:

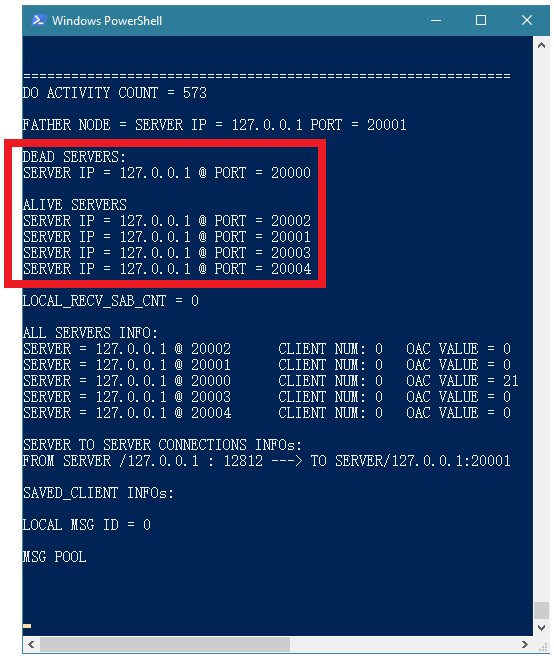
****

Figure 3: Node S2's Notices the Failure of S0

1. **Restoration Architecture**

In this test, the Restored System Architecture results in following:

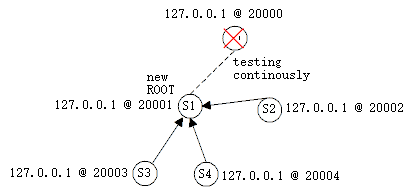
****

Figure 4: Restored System Architecture

Node S1 becomes the new ROOT for the System, S2 reconnects to S1 and set S1 as its father node. ROOT S1 once noticing the failure of S0, still keeps trying connect back to S0. Necessary information about the tree structure can be found in following Console examples:

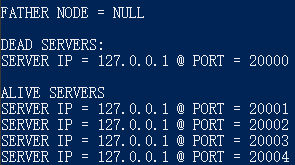
****

Figure 5: Servers Info recorded on S1

Information above tells S1 is the ROOT node of the system (Father Node == NULL)

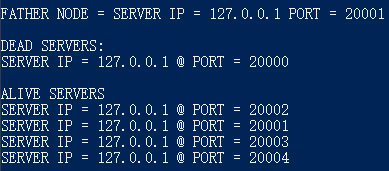
****

Figure 6: Servers Info recorded on S2

Information above tells S2’s father node is S1.

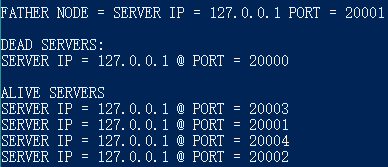
****

Figure 7: Servers Info recorded on S3

Information above tells S3’s father node is S1.

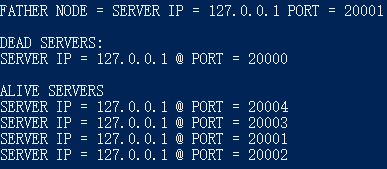
****

Figure 8: Servers Info recorded on S4

Information above tells S4’s father node is S1.

1. **\*Server S0 Restarting and going back online:**

Restart the server S0 with the following instruction in PowerShell:

java -jar Server.jar -lh 127.0.0.1 -lp 20000 -a 1000 -s abcde

The System would automatically reconnect to S0, even if S0 started without a remote hostname or port number, because, in this case, S1 keeps trying to establish the connection with S0.

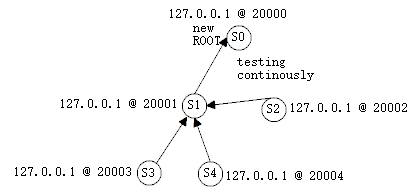


Figure 9: Architecture after S0 back online

Finally the System would form like the graph above, notice although the S0 is still the ROOT of the Tree, but the sub-connections below would be different. For more complicated architectures—more Nodes or Multi-Network-Isolations, it is hard to tell which node would be the new ROOT after the restoration process, but the architecture of Tree and functional availability could be ensured then.

## 3 Network Failover

From the previous test of Server Failure, the termination of server S0 has created a network failure between server S1 and S2. Finally the S1 and S2 reconnected together, this fact would prove the ability of reconnection over network failure.

## 4 Activity Messages

Since the Server Failure can be regarded as part of the Network Failure (central nodes’ failure may cause this), in this test, only the Server Failure is taken info consideration.

1. **Create a System with three nodes S1🡪S0🡨S2**

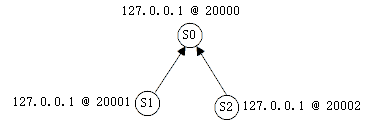


Figure 10: Architecture for Testing Message Accessibility

With the instructions as following table:

|  |  |
| --- | --- |
| Server | Instructions |
| S0 (ROOT) | java -jar Server.jar -lh 127.0.0.1 -lp 20000 -a 1000 -s abcde |
| S1 (🡪S0) | java -jar Server.jar -lh 127.0.0.1 -lp 20001 -a 1000 -s abcde –rh 127.0.0.1 –rp 20000 |
| S2 (🡪S0) | java -jar Server.jar -lh 127.0.0.1 -lp 20002 -a 1000 -s abcde –rh 127.0.0.1 –rp 20000 |

1. **Attach several clients to each of the Server**

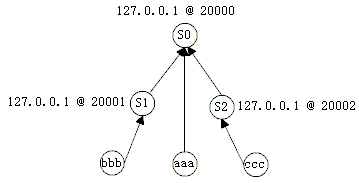
****

Figure 11: Attachment of Clients to Servers

With the instructions as following table:

|  |  |
| --- | --- |
| Client | Instructions |
| aaa | java –jar Client.jar –rh 127.0.0.1 –rp 20000 –u aaa |
| bbb | java –jar Client.jar –rh 127.0.0.1 –rp 20001 –u bbb |
| ccc | java –jar Client.jar –rh 127.0.0.1 –rp 20002 –u ccc |

1. **Terminate the Root Server S0 and bbb broadcasts a msg simultaneously (before the servers S1 and S2 notice the failure of S0)**

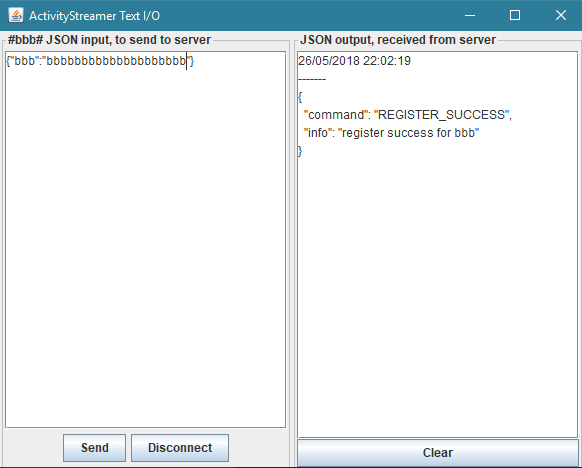
****

Figure 12:Client bbb broadcast an Activity Message

To test this, put the json string in the textbox of bbb, without hitting the Send button below. Then terminate the ROOT S0 server, sending the message by clicking on Send button in client bbb before S1 and S2 start to connect to each other.

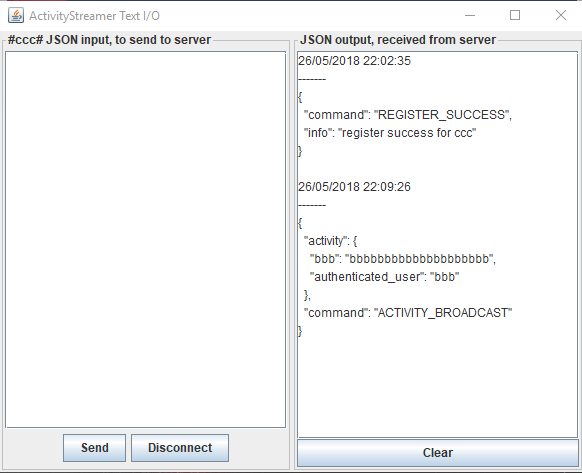
****

Figure 13: Client ccc received the Message from bbb

Finally, the other client ccc still receives the broadcast message from client bbb after the restoration of the system, which proves the Message could reach every valid clients (ccc) in the network. Also, when the number of clients mounts up, the mechanism would remain the same, here is only the simplest demonstration.

For client aaa, due to its connection to Server S0 and the S0 is terminated, client aaa would be aborted (invalid client), for in this project, there is no mechanism of client redirection while server failure.

In addition, in this test, the Network failure and Server failure appeared again, which proves the system would overcome even through a combination of failures.