CS 553 - Cloud Computing, Spring 2018

PA 1 - Read Me Manual

Benchmarking

Saumil Pareshbhai Ajmera | A20397303 | sajmera4@hawk.iit.edu

This document will help in running each application on cluster and helps to know how input is passed and where output is obtained. Also it includes screenshots of benchmarks ran on cluster.

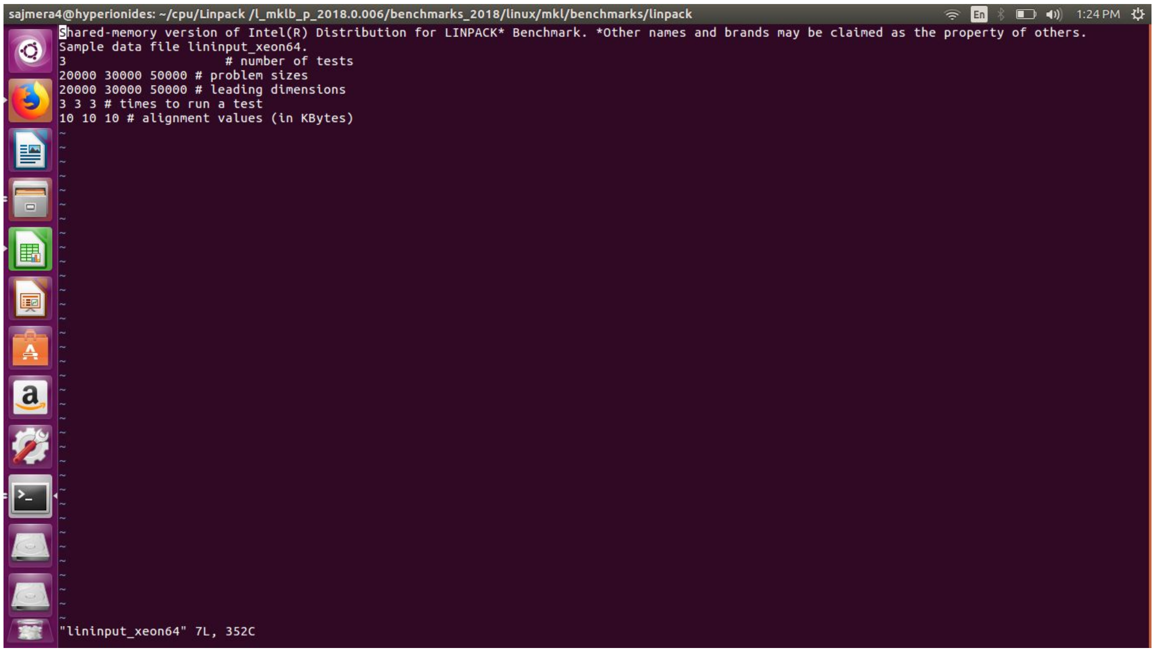
1. Processor Benchmarking

Steps to run MyCPUBench: Hyperion Cluster

1) Run the *Makefile* which will compile the *MyCPUBench.c* code with required flags into executable *MyCPUBench*
2) Run script file *create_slurms* using *bash create_slurms* command which will generate config files naming like *cpu1.slurm*, *cpu2.slurm* etc. which contains slurm commands to run exe by passing required input arguments as input DAT file and output DAT file.
3) Now I have prepared single script file per 9 config files to submit 9 jobs to different nodes. Run1.sh and Run.sh are two script files.
4) Now do *sbatch run1.sh* and after 9 files has been processed do *sbatch run2.sh* for running remaining jobs.
5) Output is written into file into output folder named *cpu_SP_1thread.out.dat* with required set of values.

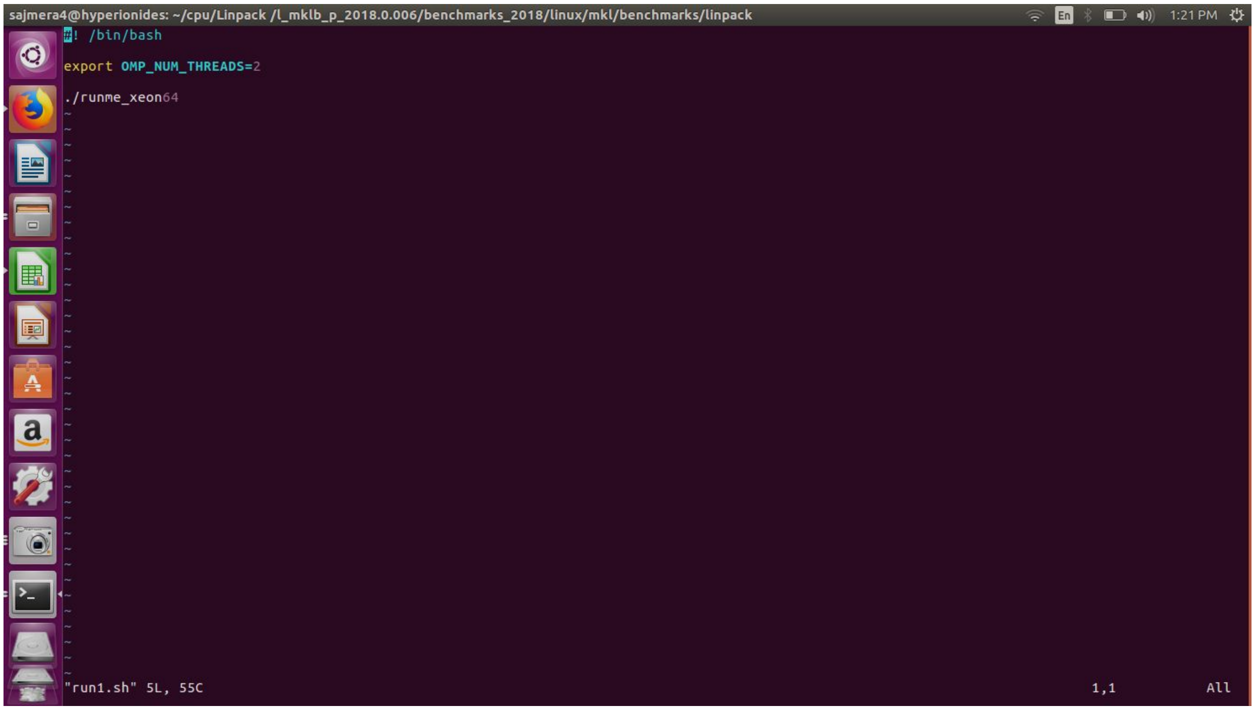
Steps to run Linpack Benchmark:

1) Go to the path mentioned below in the linpack folder after unzipping the tar file - *Linpack /l_mklb_p_2018.0.006/benchmarks_2018/linux/mkl/benchmarks/linpack*
2) Modify the *lininput_xeon64* file by number of trials, threads, problem size and ¾ of pmemory.


sajmera4@hyperionides: ~/cpu/Linpack /l_mklb_p_2018.0.006/benchmarks_2018/linux/mkl/benchmarks/linpack
En
1:24 PM

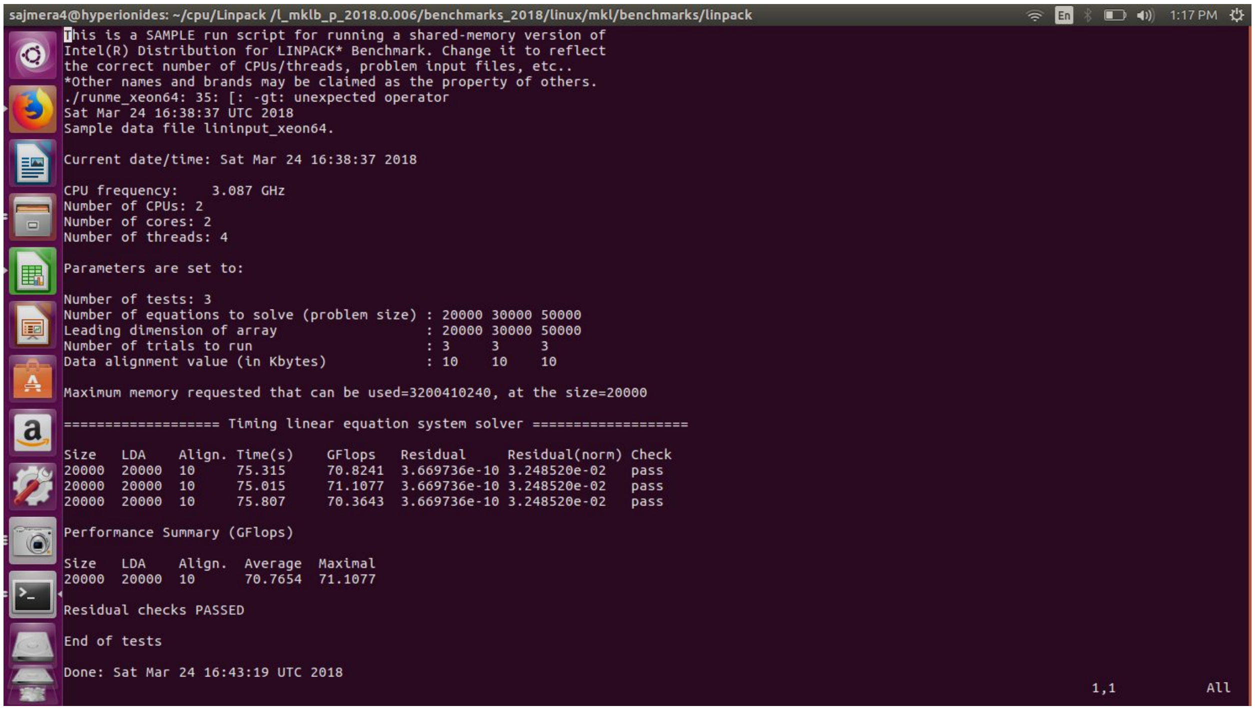
```
Shared-memory version of Intel(R) Distribution for LINPACK* Benchmark. *Other names and brands may be claimed as the property of others.
Sample data file lininput_xeon64.
3                     # number of tests
20000 30000 50000 # problem sizes
20000 30000 50000 # leading dimensions
3 3 3 # times to run a test
10 10 10 # alignment values (in KBytes)
"lininput_xeon64" 7L, 352C
```

3) Now create run1.sh file as per below screenshot to call executable runme_xeon64


sajmera4@hyperionides: ~/cpu/Linpack /l_mklb_p_2018.0.006/benchmarks_2018/linux/mkl/benchmarks/linpack
En
1:21 PM

```
#! /bin/bash

export OMP_NUM_THREADS=2

./runme_xeon64
"run1.sh" 5L, 55C                                                    1,1           All
```

4) Output is written into file into slurm.out file in the same folder with following values of Giga Ops.



sajmera4@hyperionides: ~/cpu/Linpack /l_mklb_p_2018.0.006/benchmarks_2018/linux/mkl/benchmarks/linpack
1:17 PM

```
This is a SAMPLE run script for running a shared-memory version of
Intel(R) Distribution for LINPACK* Benchmark. Change it to reflect
the correct number of CPUs/threads, problem input files, etc..
*Other names and brands may be claimed as the property of others.
./runme_xeon64: 35: [: -gt: unexpected operator
Sat Mar 24 16:38:37 UTC 2018
Sample data file lininput_xeon64.

Current date/time: Sat Mar 24 16:38:37 2018

CPU frequency:    3.087 GHz
Number of CPUs: 2
Number of cores: 2
Number of threads: 4

Parameters are set to:

Number of tests: 3
Number of equations to solve (problem size) : 20000 30000 50000
Leading dimension of array                  : 20000 30000 50000
Number of trials to run                     : 3     3     3
Data alignment value (in Kbytes)            : 10    10    10

Maximum memory requested that can be used=3200410240, at the size=20000

=================== Timing linear equation system solver ===================

Size   LDA    Align. Time(s)    GFlops   Residual     Residual(norm) Check
20000  20000  10     75.315     70.8241  3.669736e-10 3.248520e-02   pass
20000  20000  10     75.015     71.1077  3.669736e-10 3.248520e-02   pass
20000  20000  10     75.807     70.3643  3.669736e-10 3.248520e-02   pass

Performance Summary (GFlops)

Size   LDA    Align.  Average  Maximal
20000  20000  10      70.7654  71.1077

Residual checks PASSED

End of tests

Done: Sat Mar 24 16:43:19 UTC 2018
                                                                  1,1           All
```

2. Memory Benchmarking

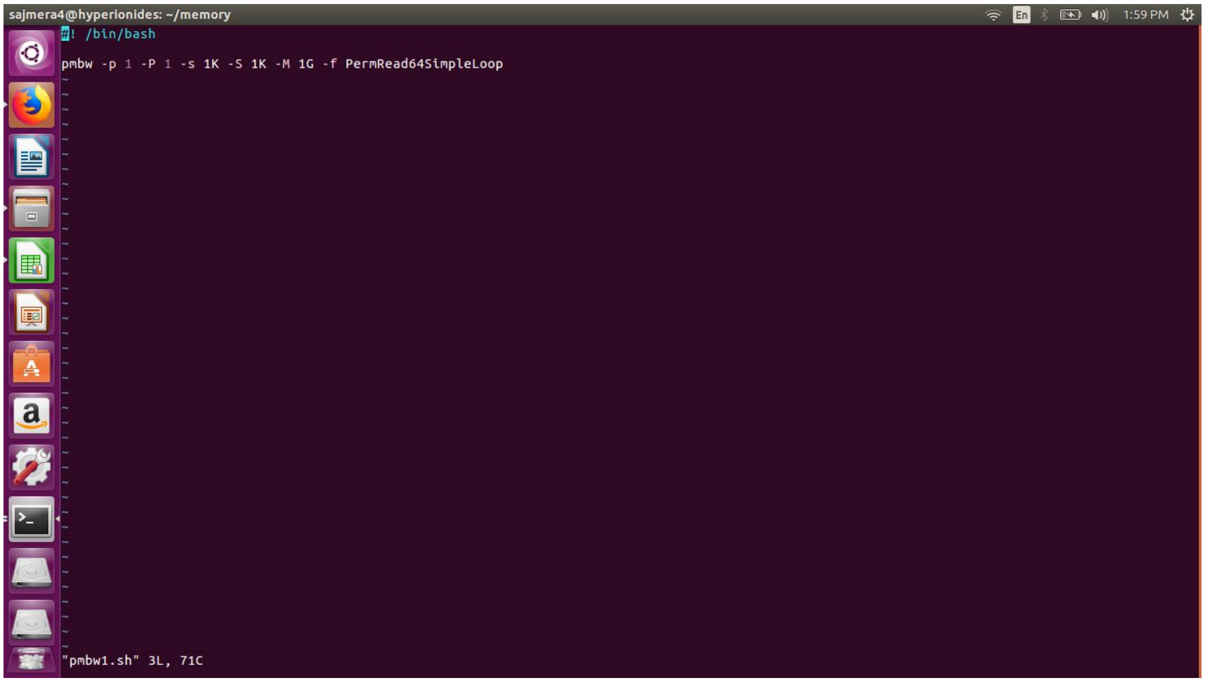
Steps to run MyRAMBench: - Hyperion Cluster

1) Run the *Makefile* which will compile the *MyRAMBench.c* code with required flags into executable *MyRAMBench*
2) Run script file *create_slurms* using *bash create_slurms* command which will generate config files naming like *mem1.slurm*, *mem2.slurm* etc. which contains slurm commands to run exe by passing required input arguments as input DAT file and output DAT file.
3) Now I have prepared single script file per 9 config files to submit 9 jobs to different nodes. Run1.sh, Run2.sh and Run3.sh script files.
4) Now do *sbatch run1.sh* and after 9 files has been processed do *sbatch run2.sh* for running remaining jobs. Similarly for other script files
5) Output is written into file in the output folder named *memory-RWR-1-1thread.out.dat* with required set of values. Output for latency is written into file in the output folder name *memory-latency.out.dat* with required set of values.

Steps to run PMBW Benchmark:

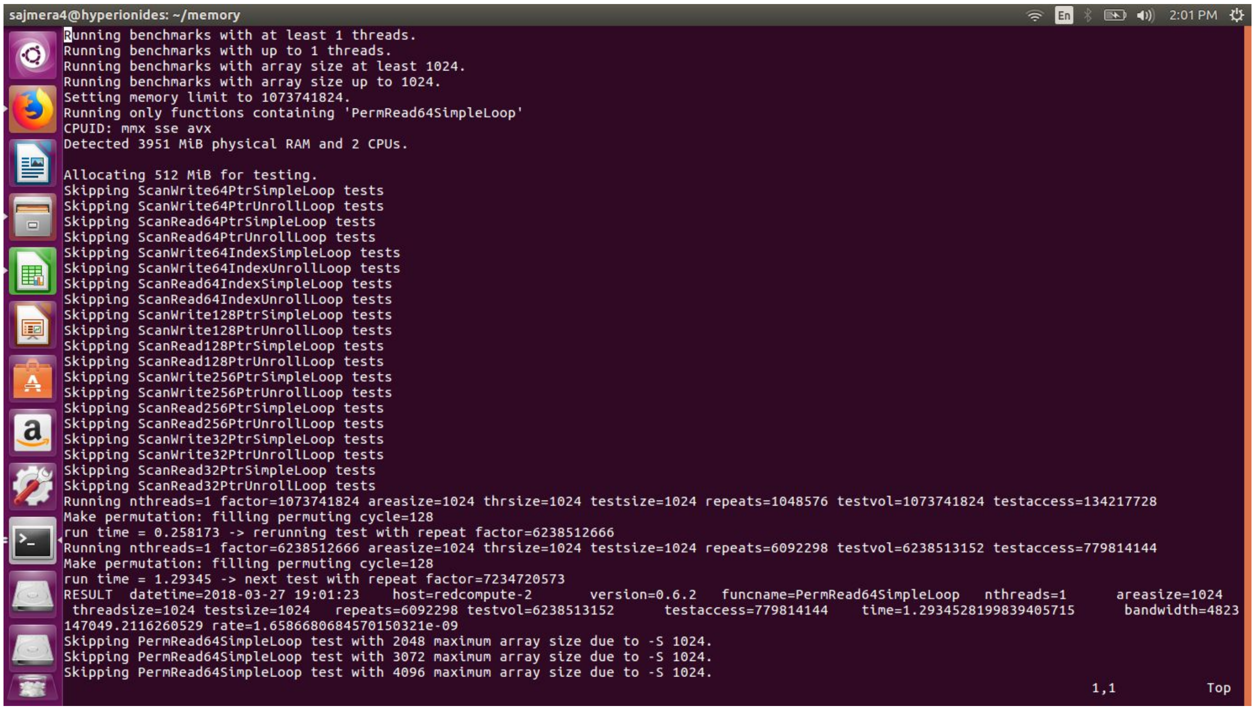
1) I have made one file pmbw1.sh which contains command to call executable of pmbw by passing certain parameters for our criteria.

2) -p and -P are min and max threads, -s and -S are min and max block size , -M for main block of memory and *PermRead64SimpleLoop* and *ScanRead64PtrSimpleLoop* are used for RWR and RWR throughput calculation. Here bandwidth term in generated output file gives the throughput value.



```
#! /bin/bash

pmbw -p 1 -P 1 -s 1K -S 1K -M 1G -f PermRead64SimpleLoop
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"pmbw1.sh" 3L, 71C
```

3) For latency calculation *ScanRead32PtrSimpleLoop* is used to find latency for 1B of data, its value is stored in rate format in the generated output file.



```
Running benchmarks with at least 1 threads.
Running benchmarks with up to 1 threads.
Running benchmarks with array size at least 1024.
Running benchmarks with array size up to 1024.
Setting memory limit to 1073741824.
Running only functions containing 'PermRead64SimpleLoop'
CPUID: mmx sse avx
Detected 3951 MiB physical RAM and 2 CPUs.

Allocating 512 MiB for testing.
Skipping ScanWrite64PtrSimpleLoop tests
Skipping ScanWrite64PtrUnrollLoop tests
Skipping ScanRead64PtrSimpleLoop tests
Skipping ScanRead64PtrUnrollLoop tests
Skipping ScanWrite64IndexSimpleLoop tests
Skipping ScanWrite64IndexUnrollLoop tests
Skipping ScanRead64IndexSimpleLoop tests
Skipping ScanRead64IndexUnrollLoop tests
Skipping ScanWrite128PtrSimpleLoop tests
Skipping ScanWrite128PtrUnrollLoop tests
Skipping ScanRead128PtrSimpleLoop tests
Skipping ScanRead128PtrUnrollLoop tests
Skipping ScanWrite256PtrSimpleLoop tests
Skipping ScanWrite256PtrUnrollLoop tests
Skipping ScanRead256PtrSimpleLoop tests
Skipping ScanRead256PtrUnrollLoop tests
Skipping ScanWrite32PtrSimpleLoop tests
Skipping ScanWrite32PtrUnrollLoop tests
Skipping ScanRead32PtrSimpleLoop tests
Skipping ScanRead32PtrUnrollLoop tests
Running nthreads=1 factor=1073741824 areasize=1024 thrsize=1024 testsize=1024 repeats=1048576 testvol=1073741824 testaccess=134217728
Make permutation: filling permuting cycle=128
run time = 0.258173 -> rerunning test with repeat factor=6238512666
Running nthreads=1 factor=6238512666 areasize=1024 thrsize=1024 testsize=1024 repeats=6092298 testvol=6238513152 testaccess=779814144
Make permutation: filling permuting cycle=128
run time = 1.29345 -> next test with repeat factor=7234720573
RESULT  datetime=2018-03-27 19:01:23    host=redcompute-2       version=0.6.2   funcname=PermRead64SimpleLoop   nthreads=1      areasize=1024
 threadsize=1024 testsize=1024   repeats=6092298 testvol=6238513152      testaccess=779814144    time=1.29345281998394057l5      bandwidth=4823
147049.2116260529 rate=1.6586680684570150321e-09
Skipping PermRead64SimpleLoop test with 2048 maximum array size due to -S 1024.
Skipping PermRead64SimpleLoop test with 3072 maximum array size due to -S 1024.
Skipping PermRead64SimpleLoop test with 4096 maximum array size due to -S 1024.
                                                                                                                     1,1           Top
```

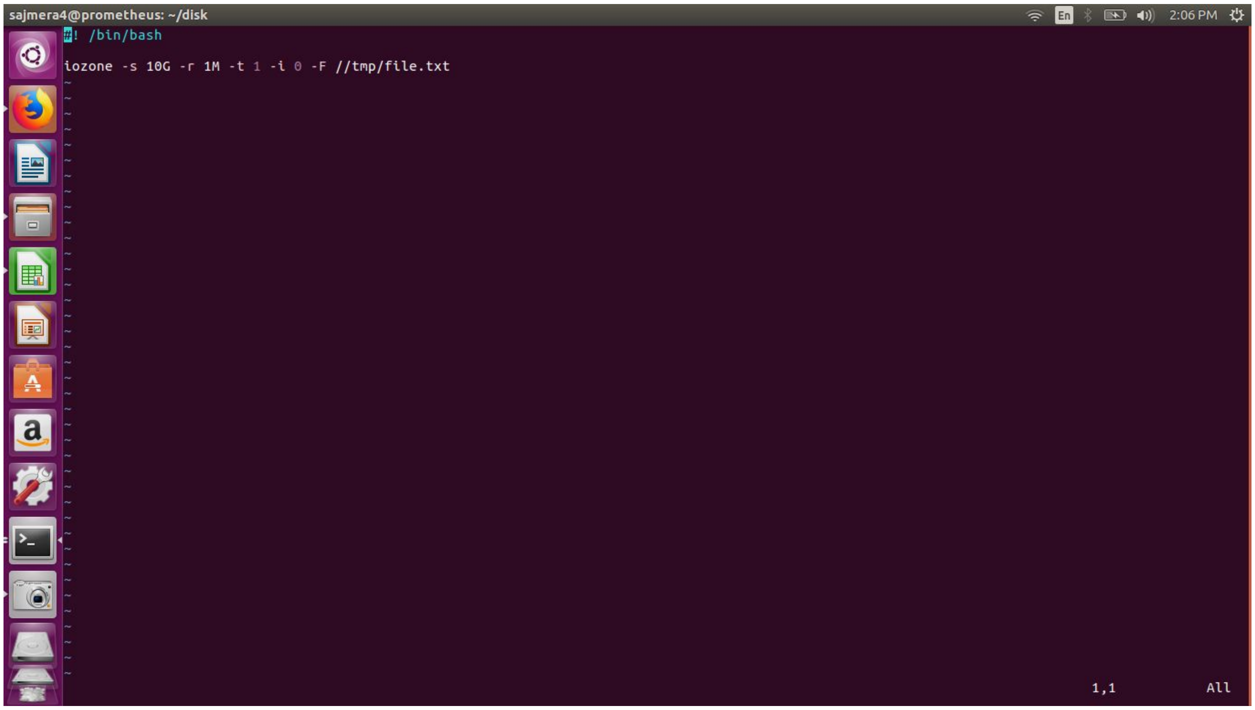
3. Disk Benchmarking

Steps to run MyDiskBench: Prometheous Cluster

1) Run the *Makefile* which will compile the *MyDiskBench.c* code with required flags into executable *MyDiskBench*
2) Run script file *create_slurms* using *bash create_slurms* command which will generate config files naming like *disk1.slurm*, *disk2.slurm* etc. which contains slurm commands to run exe by passing required input arguments as input DAT file and output DAT file.
3) Now I have prepared single script file per 9 config files to submit 9 jobs to different nodes. Here there are 52 experiments so there are 9 run.sh files containing all jobs.
4) Now do *sbatch run1.sh* and after 9 files has been processed do *sbatch run2.sh* for running remaining jobs. Similarly for other script files
5) Output is written into file in the output folder named *memory-RR-1-1thread.out.dat* with required set of values. Output for latency is written into file in the output folder name *disk-latency-IOPS.out.dat* with required set of values.

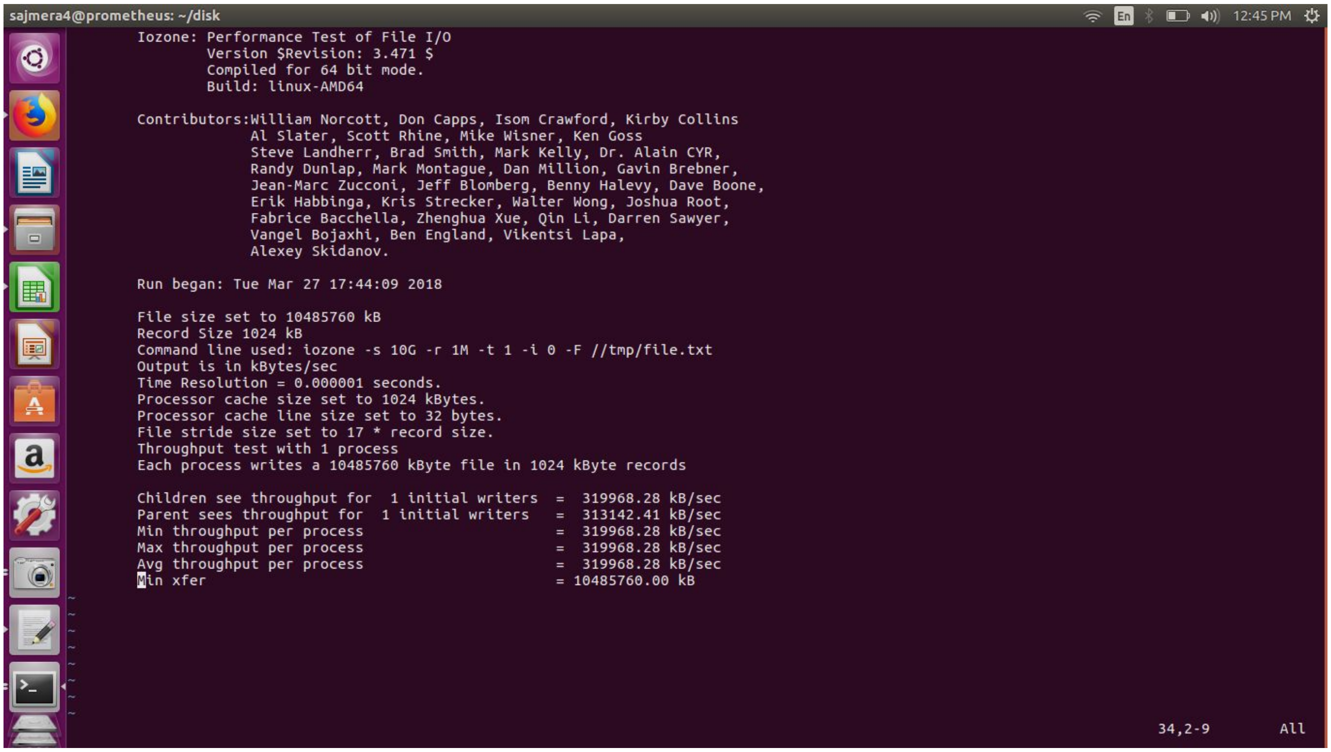
Steps to run IOZone Benchmark:

1) I have made one file runIozone.sh which contains command to call executable of iozone by passing certain parameters for our criteria.
2) -s indicates file size, -r indicates block size, -t indicates throughput in parallel threads -i indicates operation type 0,1,2 for sequence read, sequence write and random read write, -F indicates file name to be generated.



```
#! /bin/bash

iozone -s 10G -r 1M -t 1 -i 0 -F //tmp/file.txt
```

3) Here output file is generated after running sbatch runIozone.sh as slurm.out with following details.



```
        Iozone: Performance Test of File I/O
                Version $Revision: 3.471 $
                Compiled for 64 bit mode.
                Build: linux-AMD64

        Contributors:William Norcott, Don Capps, Isom Crawford, Kirby Collins
                     Al Slater, Scott Rhine, Mike Wisner, Ken Goss
                     Steve Landherr, Brad Smith, Mark Kelly, Dr. Alain CYR,
                     Randy Dunlap, Mark Montague, Dan Million, Gavin Brebner,
                     Jean-Marc Zucconi, Jeff Blomberg, Benny Halevy, Dave Boone,
                     Erik Habbinga, Kris Strecker, Walter Wong, Joshua Root,
                     Fabrice Bacchella, Zhenghua Xue, Qin Li, Darren Sawyer,
                     Vangel Bojaxhi, Ben England, Vikentsi Lapa,
                     Alexey Skidanov.

        Run began: Tue Mar 27 17:44:09 2018

        File size set to 10485760 kB
        Record Size 1024 kB
        Command line used: iozone -s 10G -r 1M -t 1 -i 0 -F //tmp/file.txt
        Output is in kBytes/sec
        Time Resolution = 0.000001 seconds.
        Processor cache size set to 1024 kBytes.
        Processor cache line size set to 32 bytes.
        File stride size set to 17 * record size.
        Throughput test with 1 process
        Each process writes a 10485760 kByte file in 1024 kByte records

        Children see throughput for  1 initial writers  =  319968.28 kB/sec
        Parent sees throughput for  1 initial writers   =  313142.41 kB/sec
        Min throughput per process                      =  319968.28 kB/sec
        Max throughput per process                      =  319968.28 kB/sec
        Avg throughput per process                      =  319968.28 kB/sec
        Min xfer                                        = 10485760.00 kB
```

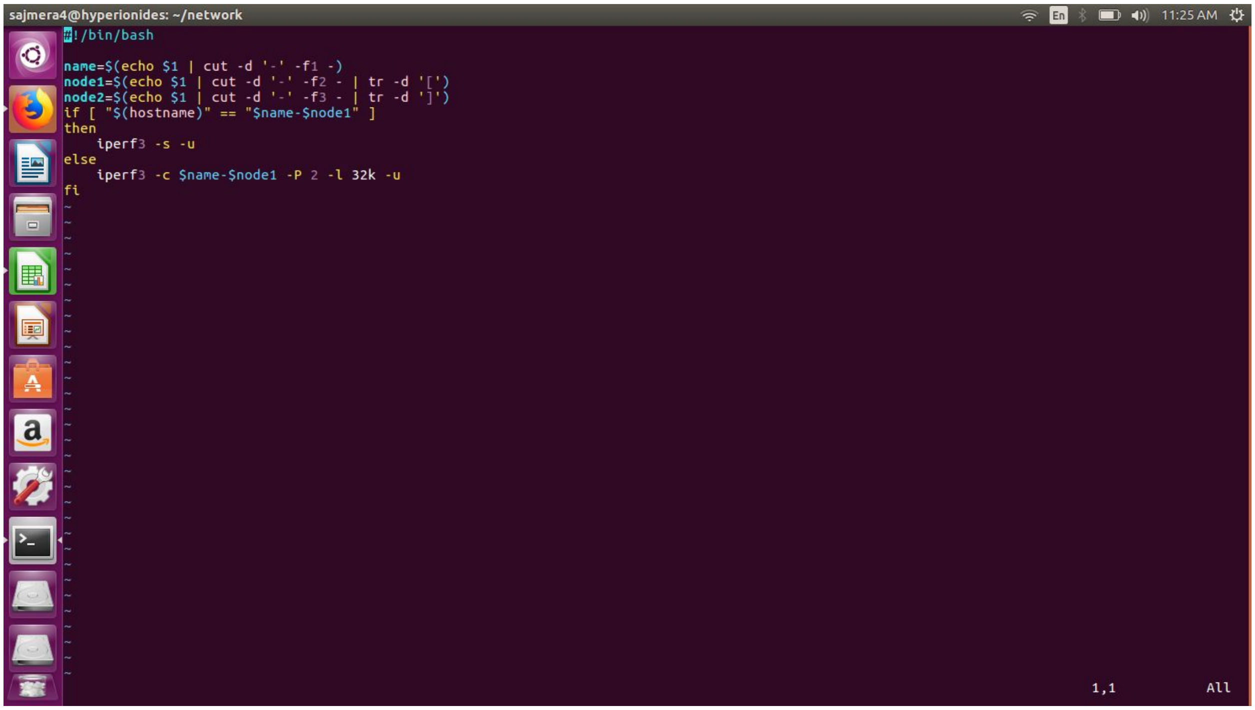
4. Network Benchmarking

Steps to run MyNETBench TCP/UDP : Hyperion Cluster

1) Run the *Makefile* which will compile the *MyNETBench-TCP.c and My NETBench-UDP.c* code with required flags into executable *MyNETBench-TCP and MyNETBench-UDP.*
2) Run script file *create_slurms* using *bash create_slurms* command which will generate config files for each configurations. which contains slurm commands to srun runUDP.sh or runTCP.sh by passing required input arguments as input DAT file and output DAT file.
3) Now manually one needs to submit each of file like sbatch TCP-1-1.slurm which will call runTCP.sh or (runUDP.sh) which will run same executable one for server and one for client by passing S or C for client server identification.
4) Note that in runTCP.sh and runUDP.sh , I have specifically mentioned path **//exports/home/sajmera4/network/MyNETBench-TCP,**
   //exports/home/sajmera4/network/MyNETBench-UDP,
   This may require changes depending upon how you are running.
5) Output for TCP and UDP for throughput is written into file in the output folder named *network-TCP-1thread.out.dat* with required set of values. Output for latency IOPS is written into file in the output folder name *network-latency-.out.dat* with required set of values.

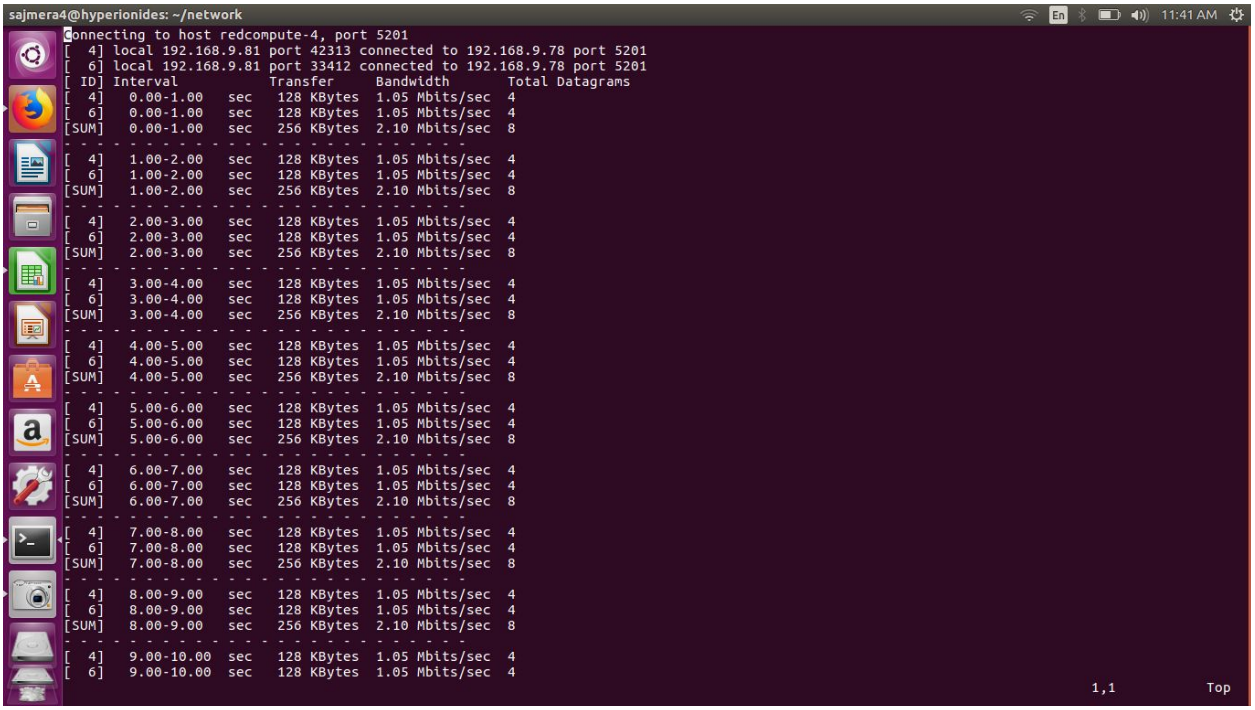
Steps to run Iperf Benchmark:

1) I have made one file *iperf.slurm* which contains command to call *iperf.sh* twice for server and client.
2) Iperf.sh contains script to extract compute node and pass as parameter to client for running client and server with -P as threads, -s and -c for Server and Client , -u for UDP, -l for block size.



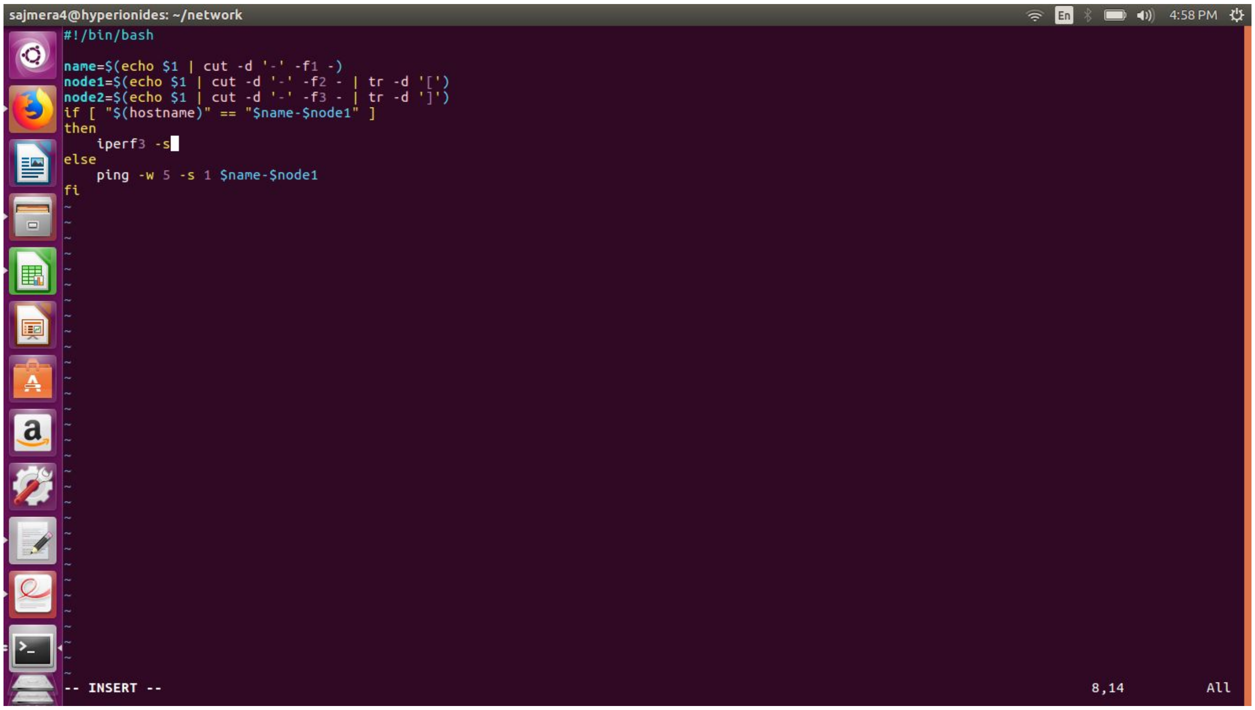
```
#!/bin/bash

name=$(echo $1 | cut -d '-' -f1 -)
node1=$(echo $1 | cut -d '-' -f2 - | tr -d '[')
node2=$(echo $1 | cut -d '-' -f3 - | tr -d ']')
if [ "$(hostname)" == "$name-$node1" ]
then
    iperf3 -s -u
else
    iperf3 -c $name-$node1 -P 2 -l 32k -u
fi
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
                                                                                 1,1           All
```

3) Here below screenshot shows the output of the script and it is stored in *iperf.out.*



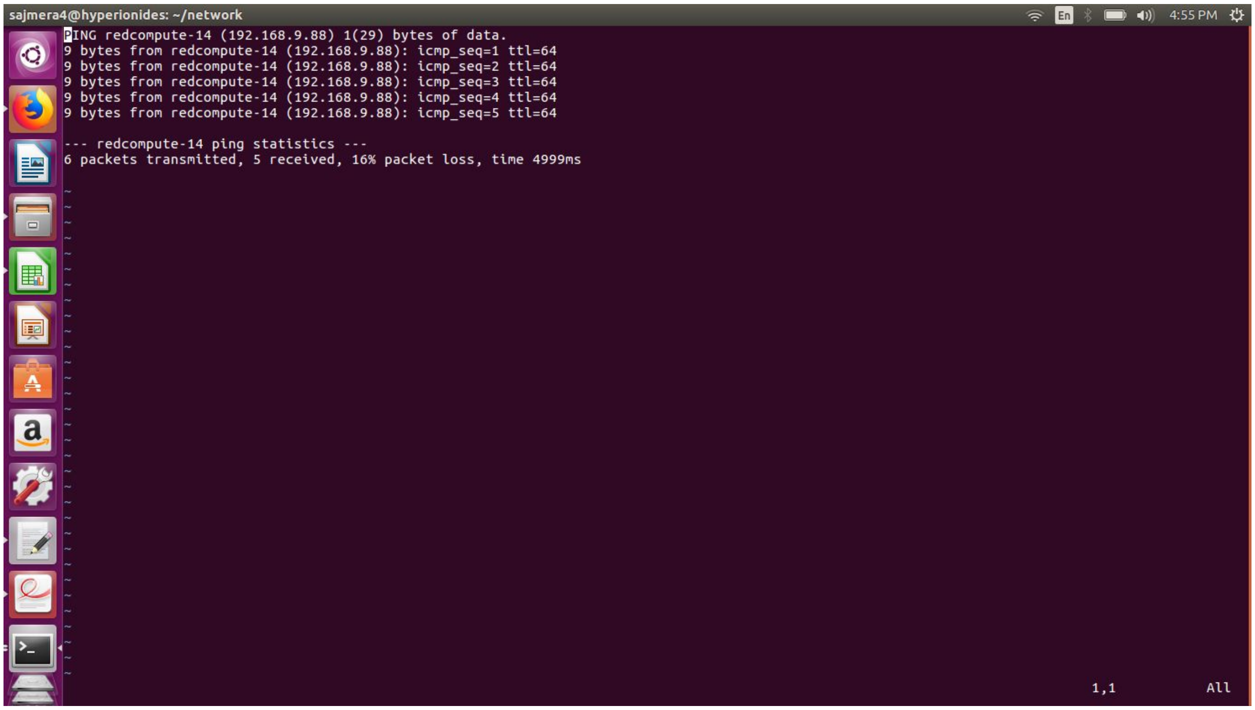
```
Connecting to host redcompute-4, port 5201
[  4] local 192.168.9.81 port 42313 connected to 192.168.9.78 port 5201
[  6] local 192.168.9.81 port 33412 connected to 192.168.9.78 port 5201
[ ID] Interval           Transfer     Bandwidth       Total Datagrams
[  4]   0.00-1.00   sec   128 KBytes  1.05 Mbits/sec  4
[  6]   0.00-1.00   sec   128 KBytes  1.05 Mbits/sec  4
[SUM]   0.00-1.00   sec   256 KBytes  2.10 Mbits/sec  8
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   1.00-2.00   sec   128 KBytes  1.05 Mbits/sec  4
[  6]   1.00-2.00   sec   128 KBytes  1.05 Mbits/sec  4
[SUM]   1.00-2.00   sec   256 KBytes  2.10 Mbits/sec  8
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   2.00-3.00   sec   128 KBytes  1.05 Mbits/sec  4
[  6]   2.00-3.00   sec   128 KBytes  1.05 Mbits/sec  4
[SUM]   2.00-3.00   sec   256 KBytes  2.10 Mbits/sec  8
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   3.00-4.00   sec   128 KBytes  1.05 Mbits/sec  4
[  6]   3.00-4.00   sec   128 KBytes  1.05 Mbits/sec  4
[SUM]   3.00-4.00   sec   256 KBytes  2.10 Mbits/sec  8
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   4.00-5.00   sec   128 KBytes  1.05 Mbits/sec  4
[  6]   4.00-5.00   sec   128 KBytes  1.05 Mbits/sec  4
[SUM]   4.00-5.00   sec   256 KBytes  2.10 Mbits/sec  8
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   5.00-6.00   sec   128 KBytes  1.05 Mbits/sec  4
[  6]   5.00-6.00   sec   128 KBytes  1.05 Mbits/sec  4
[SUM]   5.00-6.00   sec   256 KBytes  2.10 Mbits/sec  8
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   6.00-7.00   sec   128 KBytes  1.05 Mbits/sec  4
[  6]   6.00-7.00   sec   128 KBytes  1.05 Mbits/sec  4
[SUM]   6.00-7.00   sec   256 KBytes  2.10 Mbits/sec  8
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   7.00-8.00   sec   128 KBytes  1.05 Mbits/sec  4
[  6]   7.00-8.00   sec   128 KBytes  1.05 Mbits/sec  4
[SUM]   7.00-8.00   sec   256 KBytes  2.10 Mbits/sec  8
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   8.00-9.00   sec   128 KBytes  1.05 Mbits/sec  4
[  6]   8.00-9.00   sec   128 KBytes  1.05 Mbits/sec  4
[SUM]   8.00-9.00   sec   256 KBytes  2.10 Mbits/sec  8
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   9.00-10.00  sec   128 KBytes  1.05 Mbits/sec  4
[  6]   9.00-10.00  sec   128 KBytes  1.05 Mbits/sec  4
                                                                                 1,1           Top
```

4) Ping Utility script - I have used ping command to test 1 byte of data from client to server and to run it do *sbatch ping.slurm* which will call *ping.sh* twice where different conditions are set. -s for buffer size -w for timeout .



```
#!/bin/bash

name=$(echo $1 | cut -d '-' -f1 -)
node1=$(echo $1 | cut -d '-' -f2 - | tr -d '[')
node2=$(echo $1 | cut -d '-' -f3 - | tr -d ']')
if [ "$(hostname)" == "$name-$node1" ]
then
    iperf3 -s
else
    ping -w 5 -s 1 $name-$node1
fi
```

5) Output is stored in *ping.out*



```
PING redcompute-14 (192.168.9.88) 1(29) bytes of data.
9 bytes from redcompute-14 (192.168.9.88): icmp_seq=1 ttl=64
9 bytes from redcompute-14 (192.168.9.88): icmp_seq=2 ttl=64
9 bytes from redcompute-14 (192.168.9.88): icmp_seq=3 ttl=64
9 bytes from redcompute-14 (192.168.9.88): icmp_seq=4 ttl=64
9 bytes from redcompute-14 (192.168.9.88): icmp_seq=5 ttl=64

--- redcompute-14 ping statistics ---
6 packets transmitted, 5 received, 16% packet loss, time 4999ms
```