

# CS 553: Cloud Computing - HA1

Saptarshi Chatterjee  
CWID: A20413922

March 28, 2018

## 1 Hardware Specification

Experiments are run on 2 different clusters -

- **Hyperion** : Intel Xeon E312xx (Sandy Bridge). MemTotal: 4046452 kB  
. Ethernet interface product: Virtio network device
- **Prometheus** Disk- is Micron 5100 PRO 2.5" 480GB, SATA, 6Gb/s, 3D NAND, 7mm, 1.5D WPD2,  
NIC- Super Micro Computer Inc Ethernet Controller 10-Gigabit X540-AT2, CPU - Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz , 8GiB DIMM  
Synchronous 2400 MHz (0.4 ns, 64bit)

## 2 CPU Benchmarking (Run on Hyperion)

We were asked to perform 1 trillion arithmetic (quarter precision, half precision, single precision, double precision) using 1, 2 and 4 threads.

$$\begin{aligned}\text{FLOPS} &= \text{sockets} \times \frac{\text{cores}}{\text{socket}} \times \frac{\text{cycles}}{\text{second}} \times \frac{\text{FLOPs}}{\text{cycle}} \\ &= 2 * 1 * 2.299 * 8 \\ &= 36.78(\text{For DP})(\text{Sandy Bridge -8 DP FLOPs/cycle}) \\ &= 73.56(\text{for SP})(16 \text{ SP FLOPs/cycle: 8-wide AVX addition} + 8\text{-wide AVX multiplication}) \\ &= 147.136(\text{for QP}) \\ &= 147.136(\text{for HP})\end{aligned}$$

## Processor Spec

```
schatter@schatter-laptop:~$ lscpu
Architecture: x86_64
CPU op-mode(s): 32-bit, 64-bit
Byte Order: Little Endian
CPU(s): 2
On-line CPU(s) list: 0,1
Thread(s) per core: 1
Core(s) per socket: 1
Socket(s): 2
NUMA node(s): 1
Vendor ID: GenuineIntel
CPU family: 6
Model: 42
Model name: Intel Xeon E5-2630 (Sandy Bridge)
Stepping: 1
CPU MHz: 2200.000
BogoMIPS: 4609.99
Hypervisor vendor: KVM
Virtualization type: full
L1d cache: 32K
L1i cache: 32K
L2 cache: 4000K
NUMA node0 CPU(s): 0,1
Flags: fpu_eme de_pae tsc mer_pae mce cpl apic sep mtrr pae mca mwait pat pae06 cfl flush mba
      bsr sse4_0 sse4_1 ssse3 xsave xsaveopt lahf_lm constant_tsc rapr good_nop not_1gb_perf pni pclmulqdq ssse3 fma
      4b fma4 tce1 tce2 tce3 tce4 tce5 tce6 tce7 tce8 tce9 tce10 tce11 tce12 tce13 tce14 tce15 tce16 tce17 tce18 tce19
      tce20 tce21 tce22 tce23 tce24 tce25 tce26 tce27 tce28 tce29 tce30 tce31 tce32 tce33 tce34 tce35 tce36 tce37 tce38 tce39
      tce40 tce41 tce42 tce43 tce44 tce45 tce46 tce47 tce48 tce49 tce50 tce51 tce52 tce53 tce54 tce55 tce56 tce57 tce58 tce59
      tce60 tce61 tce62 tce63 tce64 tce65 tce66 tce67 tce68 tce69 tce70 tce71 tce72 tce73 tce74 tce75 tce76 tce77 tce78 tce79
      tce80 tce81 tce82 tce83 tce84 tce85 tce86 tce87 tce88 tce89 tce90 tce91 tce92 tce93 tce94 tce95 tce96 tce97 tce98 tce99
      tce100 tce101 tce102 tce103 tce104 tce105 tce106 tce107 tce108 tce109 tce110 tce111 tce112 tce113 tce114 tce115 tce116 tce117
      tce118 tce119 tce120 tce121 tce122 tce123 tce124 tce125 tce126 tce127 tce128 tce129 tce130 tce131 tce132 tce133 tce134 tce135 tce136 tce137 tce138 tce139 tce140 tce141 tce142 tce143 tce144 tce145 tce146 tce147 tce148 tce149 tce150 tce151 tce152 tce153 tce154 tce155 tce156 tce157 tce158 tce159 tce160 tce161 tce162 tce163 tce164 tce165 tce166 tce167 tce168 tce169 tce170 tce171 tce172 tce173 tce174 tce175 tce176 tce177 tce178 tce179 tce180 tce181 tce182 tce183 tce184 tce185 tce186 tce187 tce188 tce189 tce190 tce191 tce192 tce193 tce194 tce195 tce196 tce197 tce198 tce199 tce200 tce201 tce202 tce203 tce204 tce205 tce206 tce207 tce208 tce209 tce210 tce211 tce212 tce213 tce214 tce215 tce216 tce217 tce218 tce219 tce220 tce221 tce222 tce223 tce224 tce225 tce226 tce227 tce228 tce229 tce230 tce231 tce232 tce233 tce234 tce235 tce236 tce237 tce238 tce239 tce240 tce241 tce242 tce243 tce244 tce245 tce246 tce247 tce248 tce249 tce250 tce251 tce252 tce253 tce254 tce255 tce256 tce257 tce258 tce259 tce260 tce261 tce262 tce263 tce264 tce265 tce266 tce267 tce268 tce269 tce270 tce271 tce272 tce273 tce274 tce275 tce276 tce277 tce278 tce279 tce280 tce281 tce282 tce283 tce284 tce285 tce286 tce287 tce288 tce289 tce290 tce291 tce292 tce293 tce294 tce295 tce296 tce297 tce298 tce299 tce300 tce301 tce302 tce303 tce304 tce305 tce306 tce307 tce308 tce309 tce310 tce311 tce312 tce313 tce314 tce315 tce316 tce317 tce318 tce319 tce320 tce321 tce322 tce323 tce324 tce325 tce326 tce327 tce328 tce329 tce330 tce331 tce332 tce333 tce334 tce335 tce336 tce337 tce338 tce339 tce340 tce341 tce342 tce343 tce344 tce345 tce346 tce347 tce348 tce349 tce350 tce351 tce352 tce353 tce354 tce355 tce356 tce357 tce358 tce359 tce360 tce361 tce362 tce363 tce364 tce365 tce366 tce367 tce368 tce369 tce370 tce371 tce372 tce373 tce374 tce375 tce376 tce377 tce378 tce379 tce380 tce381 tce382 tce383 tce384 tce385 tce386 tce387 tce388 tce389 tce390 tce391 tce392 tce393 tce394 tce395 tce396 tce397 tce398 tce399 tce400 tce401 tce402 tce403 tce404 tce405 tce406 tce407 tce408 tce409 tce410 tce411 tce412 tce413 tce414 tce415 tce416 tce417 tce418 tce419 tce420 tce421 tce422 tce423 tce424 tce425 tce426 tce427 tce428 tce429 tce430 tce431 tce432 tce433 tce434 tce435 tce436 tce437 tce438 tce439 tce440 tce441 tce442 tce443 tce444 tce445 tce446 tce447 tce448 tce449 tce450 tce451 tce452 tce453 tce454 tce455 tce456 tce457 tce458 tce459 tce460 tce461 tce462 tce463 tce464 tce465 tce466 tce467 tce468 tce469 tce470 tce471 tce472 tce473 tce474 tce475 tce476 tce477 tce478 tce479 tce480 tce481 tce482 tce483 tce484 tce485 tce486 tce487 tce488 tce489 tce490 tce491 tce492 tce493 tce494 tce495 tce496 tce497 tce498 tce499 tce500 tce501 tce502 tce503 tce504 tce505 tce506 tce507 tce508 tce509 tce510 tce511 tce512 tce513 tce514 tce515 tce516 tce517 tce518 tce519 tce520 tce521 tce522 tce523 tce524 tce525 tce526 tce527 tce528 tce529 tce530 tce531 tce532 tce533 tce534 tce535 tce536 tce537 tce538 tce539 tce540 tce541 tce542 tce543 tce544 tce545 tce546 tce547 tce548 tce549 tce550 tce551 tce552 tce553 tce554 tce555 tce556 tce557 tce558 tce559 tce560 tce561 tce562 tce563 tce564 tce565 tce566 tce567 tce568 tce569 tce570 tce571 tce572 tce573 tce574 tce575 tce576 tce577 tce578 tce579 tce580 tce581 tce582 tce583 tce584 tce585 tce586 tce587 tce588 tce589 tce590 tce591 tce592 tce593 tce594 tce595 tce596 tce597 tce598 tce599 tce600 tce601 tce602 tce603 tce604 tce605 tce606 tce607 tce608 tce609 tce610 tce611 tce612 tce613 tce614 tce615 tce616 tce617 tce618 tce619 tce620 tce621 tce622 tce623 tce624 tce625 tce626 tce627 tce628 tce629 tce630 tce631 tce632 tce633 tce634 tce635 tce636 tce637 tce638 tce639 tce640 tce641 tce642 tce643 tce644 tce645 tce646 tce647 tce648 tce649 tce650 tce651 tce652 tce653 tce654 tce655 tce656 tce657 tce658 tce659 tce660 tce661 tce662 tce663 tce664 tce665 tce666 tce667 tce668 tce669 tce670 tce671 tce672 tce673 tce674 tce675 tce676 tce677 tce678 tce679 tce680 tce681 tce682 tce683 tce684 tce685 tce686 tce687 tce688 tce689 tce690 tce691 tce692 tce693 tce694 tce695 tce696 tce697 tce698 tce699 tce700 tce701 tce702 tce703 tce704 tce705 tce706 tce707 tce708 tce709 tce710 tce711 tce712 tce713 tce714 tce715 tce716 tce717 tce718 tce719 tce720 tce721 tce722 tce723 tce724 tce725 tce726 tce727 tce728 tce729 tce730 tce731 tce732 tce733 tce734 tce735 tce736 tce737 tce738 tce739 tce740 tce741 tce742 tce743 tce744 tce745 tce746 tce747 tce748 tce749 tce750 tce751 tce752 tce753 tce754 tce755 tce756 tce757 tce758 tce759 tce760 tce761 tce762 tce763 tce764 tce765 tce766 tce767 tce768 tce769 tce770 tce771 tce772 tce773 tce774 tce775 tce776 tce777 tce778 tce779 tce780 tce781 tce782 tce783 tce784 tce785 tce786 tce787 tce788 tce789 tce790 tce791 tce792 tce793 tce794 tce795 tce796 tce797 tce798 tce799 tce800 tce801 tce802 tce803 tce804 tce805 tce806 tce807 tce808 tce809 tce810 tce811 tce812 tce813 tce814 tce815 tce816 tce817 tce818 tce819 tce820 tce821 tce822 tce823 tce824 tce825 tce826 tce827 tce828 tce829 tce830 tce831 tce832 tce833 tce834 tce835 tce836 tce837 tce838 tce839 tce840 tce841 tce842 tce843 tce844 tce845 tce846 tce847 tce848 tce849 tce850 tce851 tce852 tce853 tce854 tce855 tce856 tce857 tce858 tce859 tce860 tce861 tce862 tce863 tce864 tce865 tce866 tce867 tce868 tce869 tce870 tce871 tce872 tce873 tce874 tce875 tce876 tce877 tce878 tce879 tce880 tce881 tce882 tce883 tce884 tce885 tce886 tce887 tce888 tce889 tce890 tce891 tce892 tce893 tce894 tce895 tce896 tce897 tce898 tce899 tce900 tce901 tce902 tce903 tce904 tce905 tce906 tce907 tce908 tce909 tce910 tce911 tce912 tce913 tce914 tce915 tce916 tce917 tce918 tce919 tce920 tce921 tce922 tce923 tce924 tce925 tce926 tce927 tce928 tce929 tce930 tce931 tce932 tce933 tce934 tce935 tce936 tce937 tce938 tce939 tce940 tce941 tce942 tce943 tce944 tce945 tce946 tce947 tce948 tce949 tce950 tce951 tce952 tce953 tce954 tce955 tce956 tce957 tce958 tce959 tce960 tce961 tce962 tce963 tce964 tce965 tce966 tce967 tce968 tce969 tce970 tce971 tce972 tce973 tce974 tce975 tce976 tce977 tce978 tce979 tce980 tce981 tce982 tce983 tce984 tce985 tce986 tce987 tce988 tce989 tce990 tce991 tce992 tce993 tce994 tce995 tce996 tce997 tce998 tce999
```

## Linpack output performance

```
precisions: LP, Threads 4, TotalTime = 58.147361, GigaOps= 17.197537
schatter@schatter-laptop:~$ ./exports/home/schatter-joe/iklb_p_2018.4.030/benchmarks_2018/linux/iklb/benchmarks/linpack/linpack_xeon64 /exports/home/schatter-joe/iklb_p_2018.4.030/benchmarks_2018/linux/iklb/benchmarks/linpack/linpack_xeon64
Sample data file lininput_xeon64.

Current date/time: Sun Mar 25 04:51:25 2018

CPU frequency: 2.751 GHz
Number of CPUs: 2
Number of cores: 2
Number of threads: 2

Parameters are set to:

Number of tests: 45
Number of equations to solve (problem size) : 1000 2000 5000 10000 15000 18000 20000 22000 25000 26000 27000 30000 35000 40000 45000
Leading dimension of array : 1000 2000 5000 10000 15000 18000 20000 22000 25000 26000 27000 30000 35000 40000 45000
Number of trials to run : 4 2 2 2 2 2 2 2 2 2 2 1 1 1 1
Data alignment value (in bytes) : 4 4 4 4 4 4 4 4 4 4 4 4 1 1 1
Maximum memory requested that can be used:332504416, at the size:20000

===== Timing linear equation system solver =====
Size LDA Align Time(s) OFlops Residual Residual(norm) Check
1000 1000 4 0.015 43.0410 9.394430e-13 3.203742e-02 pass
1000 1000 4 0.013 49.9420 9.394430e-13 3.203742e-02 pass
1000 1000 4 0.012 50.0427 9.394430e-13 3.203742e-02 pass
1000 1000 4 0.014 49.3332 9.394430e-13 3.203742e-02 pass
2000 2000 4 0.032 57.7435 4.085732e-12 3.554066e-02 pass
2000 2000 4 0.025 62.2829 4.085732e-12 3.554066e-02 pass
2000 5000 4 1.278 65.2630 2.262535e-11 3.454930e-02 pass
3000 5000 4 1.243 67.4014 2.262535e-11 3.454930e-02 pass
10000 10000 4 16.064 65.2633 9.487381e-11 3.237755e-02 pass
10000 10000 4 9.543 69.8812 9.487381e-11 3.237755e-02 pass
```

## \*CPU performance Table

Workload	Concurrency	MyCPUBench Measured Ops/Sec	HPL Measured Ops/Sec (GigaOPS)	Theoretical Ops/Sec (GigaOPS)	MyCPUBench Efficiency (%)	HPL Efficiency (%)
QP	1	45.393182	N/A	73.568	61.70234613	N/A
QP	2	90.078258	N/A	147.136	61.22108661	N/A
QP	4	85.05064	N/A	294.272	28.9020498	N/A
HP	1	46.200567	N/A	73.568	62.79981378	N/A
HP	2	86.669638	N/A	147.136	58.90444079	N/A
HP	4	85.924472	N/A	294.272	29.19899685	N/A
SP	1	23.579637	N/A	36.784	64.1029714	N/A
SP	2	46.652203	N/A	73.568	63.41371656	N/A
SP	4	43.583872	N/A	147.136	29.6214876	N/A
DP	1	8.508647	11.1738	18.392	46.26276098	60.75358852
DP	2	16.862083	24.9512	36.784	45.8408085	67.83166594
DP	4	17.197537	69.218	73.568	23.37638239	94.08710309

## Synopsis for CPU performance

- Getting best result for QP with 2 threads . 90.07 GIGOPS.
- GFLOPS is lowest for DP with 1 thread as we are not optimally using the available CPU power.
- I'm getting around 23% to 69% efficiency varying between different pre-

cision and concurrency using AVX. Where as Linpack gets 60%-94% efficiency

- It's not a Quad-core machine so using 4 threads actually hurts performance than 2 threads as a lot of time is spent in context switch .
- Linpack gets a best performance of 69.2180 GFLOPS for problem size 20000.

### 3 Memory Benchmarking (Run on Hyperion)

We were asked to perform random and Sequential Read/Write Ops on memory 1GB Workload. Operated over it 100X times with various access patterns (RWS, RWR) , various block sizes (1KB, 1MB, 10MB) and Concurrency (1 thread, 2 threads, 4 threads)

Theoretical Bandwidth = Base DRAM clock frequency  $\times$  Number of data transfers per clock  
 $\times$  Memory bus (interface) width  $\times$  Number of interfaces  
 $= 2133,000,000 * 2 * 64 * 1 / (8 * 1000 * 1000 * 1000)$   
 $\dots$  Assuming clock frequency 2133MHz  
 $= 34.128GBps$

#### Executing Memory benchmark

```
schutterjee@hyperionides: $ ./cs553-pat/memory_engine.sh
/exports/home/schutterjee/cs553-pat/memory/MRWBench.cpp: In function 'int main(int, char**)':
/exports/home/schutterjee/cs553-pat/memory/MRWBench.cpp:103:63: warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
    error("ERROR in creating p_thread");
    ^
/exports/home/schutterjee/cs553-pat/memory/MRWBench.cpp:113:68: warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
    error("ERROR in p_thread join");
    ^
/exports/home/schutterjee/cs553-pat/memory/MRWBench.cpp:129:55: warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
    storeInFile(reportPath, report.c_str());
    ^
Folder /slurms and /reports created..
Running Memory tests
Submitted batch job 19188
schutterjee@hyperionides: $ squeue
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
12233 compute ocl ochen41 PD 0:00 1 (PartitionTimeLimit)
12214 compute ocl ochen41 PD 0:00 1 (PartitionTimeLimit)
13005 interact1 bash awelline R 34:05 1 bluecompute-1
19108 interact1 bash med3 R 25:00 1 bluecompute-2
19110 interact1 bash asing63 R 24:19 1 bluecompute-3
18109 compute NET-UDF8 iweeli R 9:32 2 redcompute-[20-21]
18170 compute disk-slu asing63 R 8:46 1 redcompute-18
19181 compute memory.s schutter R 7:56 1 redcompute-29
```

```
42954
Threads 2, BLOCKSIZE -1030, TotalTime - 53.016736, TotalData - GB:100, AccessType -RMW, Throughput - GFS:1.8
65410
Threads 2, BLOCKSIZE -1030, TotalTime - 32.846752, TotalData - GB:100, AccessType -RMS, Throughput - GFS:3.0
44230
Threads 1, BLOCKSIZE -1030, TotalTime - 96.555199, TotalData - GB:100, AccessType -RMW, Throughput - GFS:1.0
53339
Threads 1, BLOCKSIZE -1030, TotalTime - 62.539427, TotalData - GB:100, AccessType -RMS, Throughput - GFS:1.5
99145
===== Time: Sun Mar 25 12:24:01 CDT 2018. New Execution =====
Threads 4, BLOCKSIZE -10303300, TotalTime - 33.139563, TotalData - GB:100, AccessType -RMW, Throughput - GFS
13.617297
```

#### pmbw - Parallel Memory Bandwidth Benchmark / Measurement

corresponding results re kept in /cs553-pa1/memory/pmbw.txt

$$Latency(ns) = clockcycletime(ns) \times numberofclockcycles =$$

Workload	Concurrency	Block Size	MyRAMBench Measured Throughput (GB/sec)	pmbw Measured Throughput (GB/sec)	Theoretical Throughput (GB/sec)	MyRAMBench Efficiency (%)	pmbw Efficiency (%)
RWS		1 1KB	1.63132	17.3	34.128	4.78000469	50.691514
RWS		1 1MB	1.69056	16.22	34.128	4.9535865	47.5269573
RWS		1 10MB	1.616496	15.59	34.128	4.73656821	45.6809657
RWS		2 1KB	3.151194	29.77	34.128	9.2334564	87.2304266
RWS		2 1MB	3.269755	29.92	34.128	9.58085736	87.6669484
RWS		2 10MB	2.97559	30.67	34.128	8.71891116	89.8675574
RWS		4 1KB	3.150432	26.42	34.128	9.23122363	77.4144397
RWS		4 1MB	3.234842	34.08	34.128	9.4785572	99.8593530
RWS		4 10MB	3.008679	30.71	34.128	8.81586674	89.9847632
RWR		1 1KB	1.137239	17.3	34.128	3.33227555	50.691514
RWR		1 1MB	1.53018	16.22	34.128	4.48364979	47.5269573
RWR		1 10MB	1.566664	15.59	34.128	4.59055321	45.6809657
RWR		2 1KB	2.975065	29.77	34.128	9.17870225	87.2304266
RWR		2 1MB	2.928384	29.92	34.128	8.58059072	87.6669484
RWR		2 10MB	3.030498	30.67	34.128	8.87979958	89.8675574
RWR		4 1KB	2.065833	26.42	34.128	6.05319093	77.4144397
RWR		4 1MB	2.978313	34.08	34.128	8.72688994	99.8593530
RWR		4 10MB	3.017597	30.71	34.128	8.84199777	89.9847632

Workload	Concurrency	Block Size	MyRAMBench Measured Latency (us)	pmbw Measured Latency (us)	Theoretical Latency (us)	MyRAMBench Efficiency (%)	pmbw Efficiency (%)
RWS	1	1B	0.000613001	5.78035E-05	0.004	84.67498713	98.549132
RWS	2	1B	0.00031734	3.35905E-05	0.004	92.06459524	99.1602284
RWS	4	1B	0.000317417	3.78501E-05	0.004	92.06459524	99.0537471
RWS	8	1B	0.000979323	3.78035E-05	0.004	78.0150934	98.1550152
RWR	2	1B	0.000506312	3.35905E-05	0.004	87.34218874	99.1602284
RWR	4	1B	0.000484066	3.78501E-05	0.004	87.98384515	99.0537471

- Benchmark works best 1MB Block Size and 2 cores. For block Size 10MB It's closer but not optimal.
- 10MB block , Single thread Random Access gives worst performance .
- Benchmark Provide better performance for Sequential memory access than Random Access. Sequential access is around 12% better than random for the best case ( $\frac{3.26-2.92}{2.92}$  case 1MB Block , 2 thread )

- Generally a CPU fetches a batch of RAM positions at once. It then keeps those in a cache while it's working on this. So if the data in RAM is sequential, the CPU has to wait on such fetching a lot less than if the data it needs to work on is scattered throughout RAM.
- It's not a Quad-core machine so using 4 threads actually hurts performance than 2 threads as a lot of time is spent in context switching.
- Interestingly sequential access for 10MB data 4 threads takes more time than Random Access 4 thread 10 MB . As these are first 2 operations done on the machine looks like cache is not warmed up , so this perticular test case results in anomaly.
- pmbw also gives optimal output for block size of 1MB , but instead of 2 threads , result of 4 thread is better . Which is a deviation from my code result.

## 4 DISK Benchmarking (Run on Hyperion)

We were asked to Write and Read 10GB data. With various access patterns (RWS, RWR) and various block sizes (1MB, 10MB, 100MB) using 1 thread, 2 threads and 4 threads.

### IOZone benchmark

```

root@hyperion:~# ./iozone -s 10000000000 -t 1 -i 1 -l 2 -r 1024 -s 1024
iozone: Performance Test of File I/O
Version: 3.5.0
Compiled for 64 bit arch.
Build Time:

Contributors: William Norcott, Gus Dapp, Ivan Crawford, Kirby Collins,
Al Gorton, Scott Shyne, Mike Warner, Ken Owe,
Steve Laskary, David Smith, Mike Kelly, Ar. Alain Chl,
Markus Luecke, Rene Henning, Ken Hillman, Geoff Haines,
Gordon S. Brown, Jeff Kaufman, Wendy Hickey, David Re,
Eric Nieuwenhuis, John Carpenter, Michael Kitz, Stefan Kitz,
Krzysztof Raczko, Zhenghui Ren, Qin Li, Guoqin Sayer,
Ken Englund.

Run began: Sun May 20 24:30:50 2018

Auto Mode
File size set to 10000000000
Record Size: 1024 B
Record Size: 1024 B
Overall Time Used: /iozone -s 10000000000 -t 1 -i 1 -l 2 -r 1024 -s 1024
Output is in /tmp/iozone
Test Resolution is 0.000000 seconds.
Processor cache size set to 32KB bytes.
Processor cache line size set to 32 bytes.
File stride size set to 17 * record size.

      random random      read      read      write
iozone -s 10000000000 -t 1 -i 1 -l 2 -r 1024 -s 1024
iozone -s 10000000000 -t 1 -i 1 -l 2 -r 1024 -s 1024

```

corresponding results re kept in /cs553-pa1/disk/iozone.sh

### Running Disk Program

```

root@hyperion:~# ./cs553-pa1-disk-engine.sh
iozone: Performance Test of File I/O
Version: 3.5.0
Compiled for 64 bit arch.
Build Time:

Contributors: William Norcott, Gus Dapp, Ivan Crawford, Kirby Collins,
Al Gorton, Scott Shyne, Mike Warner, Ken Owe,
Steve Laskary, David Smith, Mike Kelly, Ar. Alain Chl,
Markus Luecke, Rene Henning, Ken Hillman, Geoff Haines,
Gordon S. Brown, Jeff Kaufman, Wendy Hickey, David Re,
Eric Nieuwenhuis, John Carpenter, Michael Kitz, Stefan Kitz,
Krzysztof Raczko, Zhenghui Ren, Qin Li, Guoqin Sayer,
Ken Englund.

Run began: Sun May 20 24:30:50 2018

Auto Mode
File size set to 10000000000
Record Size: 1024 B
Record Size: 1024 B
Overall Time Used: /iozone -s 10000000000 -t 1 -i 1 -l 2 -r 1024 -s 1024
Output is in /tmp/iozone
Test Resolution is 0.000000 seconds.
Processor cache size set to 32KB bytes.
Processor cache line size set to 32 bytes.
File stride size set to 17 * record size.

      random random      read      read      write
iozone -s 10000000000 -t 1 -i 1 -l 2 -r 1024 -s 1024
iozone -s 10000000000 -t 1 -i 1 -l 2 -r 1024 -s 1024

```

### Disk benchmark Table

Work- load	Concurrency	Block Size	MyDiskBench Measured Throughput (MB/sec)	IOZone Measured Throughput (MB/sec)	Theoretical Throughput (MB/sec)	MyDiskBench Efficiency (%)	IOZone Efficiency (%)
RS	1	1MB	24.240029	35.033	1200	2.020002417	29.84108333
	1	10MB	22.722776	303.734	1200	1.893564667	25.31116667
	1	100MB	17.949336	341.998	1200	1.495778	28.49983333
	2	1MB	26.432186	358.093	1200	2.202682167	29.84108333
	2	10MB	30.5996	303.734	1200	2.549966667	25.31116667
	2	100MB	31.167425	341.998	1200	2.597285417	28.49983333
	4	1MB	43.434794	358.093	1200	3.619566167	29.84108333
	4	10MB	50.746256	303.734	1200	4.228854667	25.31116667
	4	100MB	38.324509	341.998	1200	3.193709083	28.49983333
	1	1MB	43.196197	836.891	1200	3.599683083	69.74091667
	1	10MB	36.822835	688.494	1200	3.068569583	57.3745
	1	100MB	38.265487	546.319	1200	3.188790583	45.52658333
WS	2	1MB	43.337197	836.891	1200	3.611433083	69.74091667
	2	10MB	45.127668	688.494	1200	3.760639	57.3745
	2	100MB	45.08773	546.319	1200	3.757310833	45.52658333
	4	1MB	48.600181	836.891	1200	4.050015083	69.74091667
	4	10MB	38.434767	688.494	1200	3.20289725	57.3745
	4	100MB	38.099094	546.319	1200	3.1749245	45.52658333
	1	1MB	11230.61736	257.997	1200	935.8847798	21.49975
	1	10MB	3317.251299	312.902	1200	276.4376083	26.07516667
	1	100MB	3015.231274	337.239	1200	251.2692728	28.10325
	2	1MB	1019.43294	257.997	1200	84.952745	21.49975
	2	10MB	382.129398	312.902	1200	31.8441165	26.07516667
	2	100MB	106.100706	337.239	1200	8.8417255	28.10325
RR	4	1MB	7148.759976	257.997	1200	595.729998	21.49975
	4	10MB	5366.675414	312.902	1200	447.2229512	26.07516667
	4	100MB	527.133318	337.239	1200	43.9277765	28.10325
	1	1MB	232.433107	908.776	1200	19.36942558	75.73133333
	1	10MB	850.706823	968.512	1200	70.89223525	80.70933333
	1	100MB	841.7931621	851.317	1200	70.14943018	70.94308333
	2	1MB	270.136348	908.776	1200	22.51136233	75.73133333
	2	10MB	169.15893	968.512	1200	14.0965775	80.70933333
	2	100MB	193.636534	851.317	1200	16.13637783	70.94308333
	4	1MB	244.584262	908.776	1200	20.38202183	75.73133333
	4	10MB	172.471967	968.512	1200	14.37266392	80.70933333
	4	100MB	235.074646	851.317	1200	19.58955383	70.94308333

## Synopsis for Disk Benchmark

theoretical throughput = one lane rate \* 2 lane per port \* 1 port per stack  
= 600 MB/sec \* 2 \* 1 = 1200 MB/sec.

- Divided all 36 different disk throughput operations in 9 groups and parallelly run them on 9 nodes.
- Disk gives highest throughput when in Sequential Read , using 1 thread and read in 100 MB chunk.
- Creating Disk Write first followed by Read so no need to copy temp files .
- 

## 5 Network Benchmarking

We were asked to implement Server-Client software using both TCP and UDP protocols with Workload: 1GB data. We operated over it 100X times with various block sizes (1KB, 32KB) with Concurrency: 1 thread, 2 threads, 4 threads and 8 threads.

Theoretical network bandwidth calculation

Theoretical Throughput =  $\frac{RWIN}{RTT}$

Where RWIN is the TCP Receive Window and RTT is the round-trip time for the path. The Max TCP Window size in the absence of TCP window scale option is 65,535 bytes.

Iperf results UDP

```
schatterjee@compute-4: ~$ ssh
schatterjee@compute-4: ~$ iperf -c 172.16.1.4 --port 4030 --format m --len 32k --udp -i 5 -b 1703M
Client connecting to 172.16.1.4, UDP port 4030
Sending 32768 byte datagrams
UDP buffer size: 0.20 MByte (default)

[3] local 172.16.1.4 port 4744t connected with 172.16.1.4 port 4030
ID Interval Transfer Bandwidth
3) 0.0-5.0 sec 1015 Mbytes 1702 Mbits/sec
5) 5.0-10.0 sec 1015 Mbytes 1702 Mbits/sec
3) 0.0-10.0 sec 2032 Mbytes 1702 Mbits/sec
3) Sent 6405 datagrams
3) Server Report:
3) 0.0-10.0 sec 2018 Mbytes 1693 Mbits/sec 0.040 ms 303/6405 (0.50%)
3) 0.0-10.0 sec 1 datagrams received out-of-order
```

```
schatterjee@prometheus: ~$ ssh
schatterjee@prometheus: ~$ iperf -s --port 4030 --format m --len 32k --udp
Server listening on UDP port 4030
Receiving 32768 byte datagrams
UDP buffer size: 0.20 MByte (default)

[3] local 172.16.1.4 port 4030 connected with 172.16.1.4 port 4744t
ID Interval Transfer Bandwidth Jitter Lost/Total Datagrams
3) 0.0-10.0 sec 2018 Mbytes 1693 Mbits/sec 0.041 ms 303/6405 (0.50%)
3) 0.0-10.0 sec 1 datagrams received out-of-order
```

Iperf results TCP

```
schatterjee@compute-9: ~$ iperf -c 172.16.1.4 --port 4030 --format m --len 32k --run 10240M -i 10
Client connecting to 172.16.1.4, TCP port 4030
TCP window size: 0.04 MByte (default)

[3] local 172.16.1.4 port 33930 connected with 172.16.1.4 port 4030
ID Interval Transfer Bandwidth
3) 0.0-10.0 sec 4710 Mbytes 3551 Mbits/sec
5) 10.0-20.0 sec 4742 Mbytes 3578 Mbits/sec
3) 0.0-21.6 sec 10240 Mbytes 3572 Mbits/sec
schatterjee@compute-9: ~$
```

```
4) local 172.16.1.4 port 4030 connected with 172.16.1.4 port 33930
ID Interval Transfer Bandwidth
4) 0.0-10.0 sec 1307 Mbytes 1004 Mbits/sec
5) local 172.16.1.4 port 4030 connected with 172.16.1.4 port 33910
5) 0.0-6.0 sec 9853 Mbytes 1215 Mbits/sec
4) local 172.16.1.4 port 4030 connected with 172.16.1.4 port 33912
4) 0.0-72.2 sec 10090 Mbytes 1199 Mbits/sec
5) local 172.16.1.4 port 4030 connected with 172.16.1.4 port 33920
5) 0.0-21.6 sec 10240 Mbytes 3570 Mbits/sec
```

Rest of the Iperf commands are detailed in iperf.sh

Running TCP BenchMark Codes on Hyperion

```
/mnt/home/schatterjee/iperf3-pat/network/MACBench-TCP app: In func()
in "172.16.1.4, 4030"
reports/home/schatterjee/iperf3-pat/network/MACBench-TCP app:10:60: ss
ping: [3] OK: forbids converting a string constant to 'char*' [-Werror=
trings]
if (( ${HOSTINFO} ${CUSTOMFILE} ${STR} ), host_name, "w") out << "0
/mnt/home/schatterjee/iperf3-pat/network/MACBench-TCP app:20:63: ss
ping: integer overflow in expression [-Werror=
trings]
Throughput in Mps: " + toString(100 * oneBody
/mnt/home/schatterjee/iperf3-pat/network/MACBench-TCP app:210:43: ss
ping: [3] OK: forbids converting a string constant to 'char*' [-Werror=
trings]
storeInfo([reportPath, report, cstr()]);
/mnt/home/schatterjee/iperf3-pat/network/MACBench-TCP app: In func()
in "172.16.1.4, 4030"
reports/home/schatterjee/iperf3-pat/network/MACBench-TCP app:52:20: ss
ping: ignoring return value of 'int' from 'FILE', used 'char*', ... [-W
lanned=results]
Report(file, "a", buff);
Folder /allum created.
Submitted batch job 23454
Submitted batch job 23455
Submitted batch job 23456
Submitted batch job 23457
Submitted batch job 23458
Submitted batch job 23459
Submitted batch job 23460
Submitted batch job 23461
Submitted batch job 23462
schatterjee@hyperion: ~$
```

```
===== Time: Tue Mar 27 12:18:13 CDT 2018. New Execution =====
Threads 2, BLOCKSIZE -3000, TotalTime = 1.06229, Node client, Total Data Received
1255886000, Throughput in Mps: 455.8439, Server Node: redcompute-3.
Threads 2, BLOCKSIZE -3000, TotalTime = 1.81492, Node server, Total Data Receiv
ed 1255764000, Throughput in Mps: 7.94670, Server Node: redcompute-5.
Threads 4, BLOCKSIZE -3000, TotalTime = 1.04939, Node client, Total Data Received
1255720000, Throughput in Mps: 5913.60407, Server Node: redcompute-3.
Threads 4, BLOCKSIZE -3000, TotalTime = 157.77000, Node server, Total Data Receiv
ed 1257764000, Throughput in Mps: 61.60320, Server Node: redcompute-3.
Threads 8, BLOCKSIZE -3000, TotalTime = 1.03091, Node client, Total Data Received
1255930000, Throughput in Mps: 5948.40554, Server Node: redcompute-2.
Threads 8, BLOCKSIZE -3000, TotalTime = 459.02432, Node server, Total Data Receiv
ed 1257839000, Throughput in Mps: 60.770514, Server Node: redcompute-2.
Threads 1, BLOCKSIZE -3000, TotalTime = 6.616527, Node client, Total Data Received
1255763000, Throughput in Mps: 1470.443010, Server Node: redcompute-6.
Threads 1, BLOCKSIZE -3000, TotalTime = 422.340048, Node server, Total Data Receiv
ed 1257839000, Throughput in Mps: 10.207050, Server Node: redcompute-6.
schatterjee@hyperion: ~$
```

```
1 redcompute-6 2944 compute netClique schatter R 6.47
1 redcompute-14 2942 compute netClique schatter R 6.47
1 redcompute-2 2942 compute netClique schatter R 6.47
1 redcompute-3 2945 compute netClique schatter R 6.47
2 redcompute-2 2947 compute UP-4-10 sajnera4 R 1.04
2 redcompute-2 2945 interacti bash dsalman R 0.14
1 bluescompute-2 2945 interacti bash dsalman R 0.14
schatterjee@hyperion: ~$ square
square PARTITION NWE USER ST TIME NODE
8 NODE157(6520) 2233 compute MyOsIdle cyrusale CD 15.26
1 redcompute-3 2345 compute MyOsIdle cyrusale CD 15.59
1 bluescompute-1 2345 interacti bash dsalman R 16.59
1 bluescompute-1 2343 interacti bash dsalman R 20.59
1 bluescompute-3 2947 compute UP-4-10 sajnera4 R 6.24
2 redcompute-2 2945 interacti bash dsalman R 5.31
1 bluescompute-2 2945 compute bash dsalman R 3.27
4 redcompute-4 2959 compute bash dsalman R 3.23
4 redcompute-4 2959 compute UP-4-10 sajnera4 R 0.39
2 redcompute-2 2963 compute UP-4-10 sajnera4 R 0.36
2 redcompute-15-16 2963 compute UP-4-10 sajnera4 R 0.36
schatterjee@hyperion: ~$ square[]
```

Throughput calculation Table

Protocol	Concurrency	Block Size	MyNETBench Measured Throughput (Mb/sec)	iperf Measured Throughput (Mb/sec)	Theoretical Throughput (Mb/sec)	MyNETBench Efficiency (%)	iperf Efficiency (%)
TCP	1	1KB	1.2389	564	1323.939394	0.09357679	42.6001373
TCP	1	32KB	1.6784	3626	1137.266811	0.1475819	318.834592
TCP	2	1KB	1.2389	566	1323.939394	0.09357679	42.7512016
TCP	2	32KB	1.6784	3666	1137.266811	0.1475819	322.351797
TCP	4	1KB	1.2389	562	1323.939394	0.09357679	42.449073
TCP	4	32KB	1.6784	3,524.00	1137.266811	0.1475819	309.865721
TCP	8	1KB	1.2389	563	1323.939394	0.09357679	42.5246052
TCP	8	32KB	1.6784	3585	1137.266811	0.1475819	315.229458
UDP	1	1KB	0.9765	559	1323.939394	0.07375715	42.2224765
UDP	1	32KB	0.9218	1693	1137.266811	0.08105398	148.865682
UDP	2	1KB	0.9765	892	1323.939394	0.07375715	67.3746853
UDP	2	32KB	0.9218	1761	1137.266811	0.08105398	154.84493
UDP	4	1KB	0.9765	987	1323.939394	0.07375715	74.5502403
UDP	4	32KB	0.9218	1198	1137.266811	0.08105398	105.340276
UDP	8	1KB	0.9765	959	1323.939394	0.07375715	72.4353399
UDP	8	32KB	0.9218	1399	1137.266811	0.08105398	123.014229

### Synopsis for bandwidth calculation

- Increasing No of thread does have significant +ve effect on Measured Throughput . Looks like 1 thread cant max out TCP buffer , If we use 8 threads then TCP throughput is maximal . For UDP it's 2 threads
- for TCP best result is obtained when block size is 32kb and thread count is 8.
- TCP is giving effectively higher Throughput than UDP . Each frame goes through several buffers we send it: The application buffer, The Protocol Buffer, The Software interface buffer and the Hardware interface buffer. As we start stressing the stack by sending high speed data you will fill up these buffers and TCP is performing better as TCP is optimized for high speed bulk transfers.
- Throughput is dependent on Block size as higher block size yeilds in greater utilization of the available bandwidth and less no of iteration.

### Latency calculation using Pingpong.cpp and 'PING' utility tool

Implemented similar Server-Client software to check latency, where server and client need to acknowledge the receipt of 1B data. And iterated it 1 million times.

Theoretical Latency: Ultimately response time over a network is limited by the speed of light. In a vacuum, light travels with a speed of 299 792 458 m / s. So, if we assume that a 'ping' travels with the speed of light, which are the best possible response times we can get ? Assuming machines are 10m apart . Latency should be  $\frac{10*2*1000}{299792458} \text{ MilliSecond} = 0.000066ms$



## Executing PingPong.cpp

```
schatterjee@hyperionides:~$ ./cs553-pat/pingpong_engine.sh 1
/exports/home/schatterjee/cs553-pat/pingpong/Pingpong.cpp:36:72: warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
int storeInFile(const char *filename, const char *data, char *mode="a+") {
/exports/home/schatterjee/cs553-pat/pingpong/Pingpong.cpp: In function 'int main(int, char**)':
/exports/home/schatterjee/cs553-pat/pingpong/Pingpong.cpp:235:68: warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
if (!storeInFile(customfile.c_str(), host_name, "w")) out << "Couldn't write IP to
/exports/home/schatterjee/cs553-pat/pingpong/Pingpong.cpp:285:43: warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
storeInFile(reportPath, report.c_str());
/exports/home/schatterjee/cs553-pat/pingpong/Pingpong.cpp: In function 'void readFromFile(const char*, char*)':
/exports/home/schatterjee/cs553-pat/pingpong/Pingpong.cpp:59:29: warning: ignoring return value of 'int fscanf(FILE*, const char*, ...)', declared with attribute warn_unused_result [-Wunused-result]
fscanf(file, "%s", buff);
Folder /slums created..
Submitted batch job 27345
Submitted batch job 27346
Submitted batch job 27347
Submitted batch job 27348
Submitted batch job 27349
Submitted batch job 27351
Submitted batch job 27352
Submitted batch job 27353
schatterjee@hyperionides:~$ {}
```

```
=> ./slums/pingpongserver.out <=
Threads 4, TotalTime = 235.005263, Mode server, Total Data Received 1000002, Latency in ms: 0.233005, Server Node redcompu
te-2, Transmission Mode: TCP...

schatterjee@hyperionides:~$ (ssh)

===== Time: Wed Mar 28 21:08:56 CDT 2018. New Execution =====
Threads 2, TotalTime = 200.66376, Mode server, Total Data Received 1000001, Latency in ms: 0.200664, Server Node redcompu
te-3, Transmission Mode: TCP...
Threads 8, TotalTime = 38.276230, Mode client, Total Data Received 1000005, Latency in ms: 0.033276, Server Node redcompu
te-4, Transmission Mode: TCP...
Threads 8, TotalTime = 224.411917, Mode server, Total Data Received 1000005, Latency in ms: 0.224412, Server Node redcompu
te-4, Transmission Mode: TCP...
Threads 1, TotalTime = 162.873105, Mode client, Total Data Received 1000000, Latency in ms: 0.162373, Server Node redcompu
te-5, Transmission Mode: TCP...
Threads 4, TotalTime = 222.120745, Mode server, Total Data Received 1000000, Latency in ms: 0.222121, Server Node redcompu
te-5, Transmission Mode: TCP...
Threads 4, TotalTime = 235.005263, Mode server, Total Data Received 1000002, Latency in ms: 0.233005, Server Node redcompu
te-2, Transmission Mode: TCP...
```

## Latency calculation Table

Protocol	Concurrency	Message Size	MyNETBench Measured Latency (ms)	ping Measured Latency (ms)	Theoretical Latency (ms)	MyNETBench Efficiency (%)	Ping Efficiency
TCP	1	1B	0.186475	0.00396	0.000066	0.035393484	1.6666667
TCP	2	1B	0.100287	0.00396	0.000066	0.065811122	1.6666667
TCP	4	1B	0.168867	0.00396	0.000066	0.039084013	1.6666667
TCP	8	1B	0.224412	0.00396	0.000066	0.029410192	1.6666667
UDP	1	1B	0.051648	0.00396	0.000066	0.127788104	1.6666667
UDP	2	1B	0.163044	0.00396	0.000066	0.04047987	1.6666667
UDP	4	1B	0.107404	0.00396	0.000066	0.061450225	1.6666667
UDP	8	1B	0.057742	0.00396	0.000066	0.114301548	1.6666667

## Synopsis for Latency calculation

- UDP is faster than TCP, and the simple reason is because its nonexistent acknowledge packet (ACK) that permits a continuous packet stream, instead of TCP that acknowledges a set of packets, calculated by using the TCP window size and round-trip time (RTT).
- UDP latencies are less compared to TCP
- For TCP we get best latency with 2 threads. And for UDP it's 1 thread.

## References

- [1] Measuring network throughput  
*[https : //en.wikipedia.org/wiki/Measuring\\_network\\_throughput](https://en.wikipedia.org/wiki/Measuring_network_throughput)*
- [2] UDP Latency  
*[https : //stackoverflow.com/questions/47903/udp-vs-tcp-how-much-faster-is-it](https://stackoverflow.com/questions/47903/udp-vs-tcp-how-much-faster-is-it)*
- [3] TCP Bandwidth  
*[https : //serverfault.com/questions/432101/why-is-udp-slower-than-tcp-on-ubuntu-server](https://serverfault.com/questions/432101/why-is-udp-slower-than-tcp-on-ubuntu-server)*