Experiments

Code is available on Github[1].
In some experiments with `gcc -O3` a blackhole[2] value (garbage) is used to make sure that code is not optimized out (dead code).
In the following section "use `abc`" defines the build target (`abc`) in the Makefiles.

CPU

Measurement overhead

Use `loop_time`. The basic idea is to run the operation many times and divide the total runtime by the number of operations. Std::chrono::stead_clock and rdtscp (based on intel's paper[3]) instructions are measured.

i. A simple std::chrono (steady_timer) timer.
ii. Compiler will replace std::chrono with kernel provided `clock_gettime`.
    1. Accessible in user space program memory. Offers some optimizations such as VDSO.
iii. No need to worry about CPU frequency scaling. Modern CPUs take care of that (constant_tsc)

Use `loop_overhead`. A loop with no body is run (`gcc -O0`) and used Compiler Explorer[4] to make sure the correct assembly is created and measured.

Procedure Overhead

Use `rdtscp-overhead`. Used `gcc -O0` to make sure code is optimized out. Used the same variable for all args to make sure no memory/cache penalty is paid for different methods.

System Call

Use `syscall`

Task Creation

Use `thread`

[1] https://github.com/esihaj/CSE221-OS-Benchmarks
[2] https://javadoc.io/static/org.openjdk.jmh/jmh-core/1.23/org/openjdk/jmh/infra/Blackhole.html
[3] How to Benchmark Code Execution Times on Intel® IA-32 and IA-64 Instruction Set Architectures (https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ia-32-ia-64-benchmark-code-execution-paper.pdf)
[4] https://godbolt.org

Context Switch

Use `context-switch` and `context-switch-thread`.
Run context-switch-thread with `taskset -c 5`
The idea is to pin process/threads to a single core and then force them to do a context-swtich which is carried out by doing a ping-pong message passing with pipes (block on pipe)

Memory

Latency

Use `linked-list.out`. Interestingly `random-access.out` did not produce results.
Idea: allocate a contiguous memory for linked lists. Randomly connect them together and then traverse it multiple times. This random access pattern will break the prefetcher.

Bandwidth

Use `parallel_memset.out`. Tried many different ideas. Most of them did not work as expected. (check the bandwidth dir). Idea: multiple threads (over 2 NUMA nodes are run to do multiple iterations of `memset`). Assembly code is checked to make sure the memset is not optimized out.

The code is modified to loop over packet sizes and then it is run with the best packet size (16KB).

Page Fault

Use `pagefault.out`. Idea: clear file cache, and use gnu time (`/usr/bin/time -v`) to measure major page faults.

Network

RTT

Use `client.out` and `server.out` to run a ping-pong over TCP and measure the latency.
Use linux `ping` command to measure ping ICMP latency.
Usage (same for the rest too)
`./server.out port`
`./client ip port`

Bandwidth

Use `bw-client.out` and `bw-server.out` to measure. Idea: send packets in a loop without waiting). Used Asynchronous IO (ASIO Lib)

Connection Overhead

Use `c-connection-client.out` and `c-connection-server.out`. This one is implemented in pure c[5] as the one implemented with ASIO reported really large latencies (~400 microseconds for local connection).

File System

File Cache

Use `file_cache.out`. Idea: drop the all file cache. read the file for a specific size. And the start the measurement (bandwidth of reading the file for the specified size). Run the experiment for different sizes to determine file cache size.
Run with `sudo`
To reduce experiment size free/available memory size is artificially reduced (`fake-mem-consumer.sh`).

File Read

Use `read.out`. Idea: file cache is dropped in each iteration.
Run with `sudo`.

Remote Read

Use `read.out` but edit the code of `read.cpp` to point to remote (locally mounted) dir.

System Specifications

The servers used in these experiments are bare-metals and are instantiated on CloudLab[6]. They are `c6220`[7] models available on CloudLab's APT datacenter.

OS

[5] https://www.cs.cmu.edu/afs/cs/academic/class/15213-f00/www/class24code/echoclient.c
[6] https://www.cloudlab.us
[7] https://www.apt.emulab.net/portal/show-nodetype.php?type=c6220

description	Ubuntu 22.04.01 LTS
kernel	5.15.0-46-generic

System Preparation

1. CPU governer is set to `performance`
2. Hyper-Threading is disabled
3. When applicable tests are run using `taskset -c [core-number] ./program` to pin them to a single cpu core.

Server

description	Rack Mount Chassis
product	PowerEdge C6220 II
vendor	Dell Inc.

This server / CPU / motherboard use PCI-Express 3.

CPU

CPU details in the following table are taken from `lscpu` command. They can also be found on [intel][8], [cpuagent.com][9], and [cpu-world][10].

Architecture	x86_64
CPU op-mode(s)	32-bit, 64-bit
Address sizes	46 bits physical, 48 bits virtual
Byte Order	Little Endian
CPU(s)	32
On-line CPU(s) list	0-15
Off-line CPU(s) list	16-31

[8] https://www.intel.com/content/www/us/en/products/sku/75269/intel-xeon-processor-e52650-v2-20m-cache-2-60-ghz/specifications.html

[9] https://www.cpuagent.com/cpu/intel-xeon-e5-2650-v2/specs/nvidia-geforce-rtx-2080-ti?res=1&quality=ultra

[10] https://www.cpu-world.com/CPUs/Xeon/Intel-Xeon%20E5-2650%20v2.html

Vendor ID	GenuineIntel
Model name	Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz
CPU family	6
Model	62
Thread(s) per core	1
Core(s) per socket	8
Socket(s)	2
Stepping	4
CPU max MHz	3400
CPU min MHz	1200
Flags	fpu vme de pse **tsc** msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb **rdtscp** lm **constant_tsc** arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 cx16 xtpr pdcm pcid dca sse4_1 sse4_2 x2apic popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm cpuid_fault epb pti ssbd ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid fsgsbase smeperms xsaveopt dtherm ida arat pln pts md_clear flush_l1d
Virtualization	VT-x
Caches (sum of all)	
L1d	512 KiB (16 instances x 32 KiB)
L1i	512 KiB (16 instances x 32 KiB)
L2	4 MiB (16 instances x 256 KiB)
L3	40 MiB (2 instances x 20 MiB)
NUMA node(s)	2
NUMA node0 CPU(s)	0-7
NUMA node1 CPU(s)	8-15
Vulnerabilities	
Itlb multihit	KVM
L1tf	Mitigation; PTE Inversion; VMX conditional cache flushes, SMT disabled
Mds	Mitigation; Clear CPU buffers; SMT disabled
Meltdown	Mitigation; PTI
Mmio stale data	Not affected
Retbleed	Not affected

Spec store bypass	Mitigation; Speculative Store Bypass disabled via prctl and seccomp
Spectre v1	Mitigation; usercopy/swapgs barriers and __user pointer sanitization
Spectre v2	Mitigation; Retpolines, IBPB conditional, IBRS_FW, RSB filling
Srbds	Not affected
Tsx async abort	Not affected

Memory

Data is taken from `dmidecode -t 17` command.

Configuration

1	Size	8 GB
2	Size	No Module Installed
3	Size	8 GB
4	Size	No Module Installed
5	Size	8 GB
6	Size	No Module Installed
7	Size	8 GB
8	Size	No Module Installed
9	Size	8 GB
10	Size	No Module Installed
11	Size	8 GB
12	Size	No Module Installed
13	Size	8 GB
14	Size	No Module Installed
15	Size	8 GB
16	Size	No Module Installed

Memory Device

Array Handle	0x001B

Error Information Handle	0x0033
Total Width	72 bits
Data Width	**72 bits**
Size	**8 GB**
Form Factor	DIMM
Set	None
Locator	DIMM_B4
Bank Locator	CPU2
Type	**DDR3**
Type Detail	Synchronous Registered (Buffered)
Speed	**1866 MT/s**
Manufacturer	AD00B300AD00
Serial Number	7000CF5C
Asset Tag	1421363
Part Number	HMT41GR7AFR4C-RD
Rank	1
Configured Memory Speed	1866 MT/s

Network

Public IP:

description	Ethernet interface
product	I350 Gigabit Network Connection
vendor	Intel Corporation
s/mellize	1Gbit/s

Other Networking NICs:
They are not set up and don’t have an ip associated with them.

description	Ethernet interface
product	Ethernet 10G 2P X520 Adapter
vendor	Intel Corporation
size	10 Gbit/s

description	Network controller
product	MT27500 Family [ConnectX-3]
vendor	Mellanox Technologies

Storage

Seagate ES.3[11]

Standard Model Number	ST1000NM0033
Spindle Speed (RPM)	7200
Max. Sustained Transfer Rate OD (MB/s)	175
Average Latency (ms)	4.16

Data taken from `lshw`

description	ATA Disk
product	**ST1000NM0033**-9ZM
physical id	0
logical name	/dev/sda
version	GA06
size	931GiB (1TB)

description	ATA Disk
product	ST1000NM0033-9ZM
physical id	1
logical name	/dev/sdb
version	GA06
size	931GiB (1TB)

[11] https://www.seagate.com/www-content/product-content/constellation-fam/constellation-es/constellation-es-3/en-us/docs/constellation-es-3-data-sheet-ds1769-1-1210us.pdf

There is also a 100 GB network file system attached (mounted from ops.apt.emulab.net:/proj/os-benchmark-PG0)

Libraries

Hdr Histogram[12]

Hdr Histogram C[13] is used for calculating percentiles and drawing percentile plots[14].

Boost[15]

Boost ASIO[16] is used for networking experiments.

NanoBench[17]

NanoBench is a microbenchmarking framework in C++ and is used in the first few experiments along with other time measuring experiments to create benchmarks.

Result Summary

CPU

Experiment			Value	Value Description
Measurement overhead	Time	std::chrono	26 ns	p99
		RDTSCP	359 cycles	p99
Loop			6 cycles	p99
Procedure Call Overhead	inline		48 cycles	p99
	0 Param		52 cycles	p99
	1 Param		52 cycles	p99
	2 Param		52 cycles	p99
	3 Param		52 cycles	p99
	4 Param		52 cycles	p99

[12] http://hdrhistogram.github.io/HdrHistogram/
[13] https://github.com/HdrHistogram/HdrHistogram_c
[14] http://hdrhistogram.github.io/HdrHistogram/plotFiles.html
[15] https://www.boost.org/
[16] https://think-async.com/Asio/
[17] https://nanobench.ankerl.com/

Experiment			Value	Value Description
	5 Param		56 cycles	p99
	6 Param		56 cycles	p99
	7 Param		56 cycles	p99
System Call	getuid()		429.5 ns/op +/- 0.0%	avg
	clock_gettime()		30.2 ns./op +/- 0.1%	avg
Task Creation	create process		75,302 ns/op +/- 0.1%	avg
	create thread and run		27,091 ms/op +/- 0.5%	avg
	create thread		14,239 ns/op +/- 0.6%	avg
Context Switch	Process		2941 cycles	p9999

Memory

Bandwidth	Theoretical		134 GB/S	
	Measured		133 GB/S	p99
Latency	L1		4 cycles	p99
	L2		15 cycles	p99
	L3		60-70 cycles	p99
Cache Size	L1		32 Kib	p99
	L2		256 KiB	p99
	L3		in range: [16,22] MiB	p99
Page Fault	File Read Time	w/ cache	27 +/- 0 (ms)	avg
		w/o cache	7098 +/- 97 (ms)	avg
	Page Fault Service Time		810 +/- 11 (μs)	avg

Network

Bandwidth	Theoretical		134 GB/S	
	Measured		133 GB/S	p99
Latency	L1		4 cycles	p99
	L2		15 cycles	p99
	L3		60-70 cycles	p99
Cache Size	L1		32 Kib	p99
	L2		256 KiB	p99
	L3		in range: [16,22] MiB	p99
Page Fault	File Read Time	w/ cache	27 +/- 0 (ms)	avg
		w/o cache	7098 +/- 97 (ms)	avg
	Page Fault Service Time		810 +/- 11 (µs)	avg
Latency	Local	TCP Ping-Pong	20.4 +/- 1.5 (µs)	avg
		ICMP Ping	12.0 +/- 2.7 (µs)	avg
	Remote	TCP Ping-Pong	88.2 +/- 4.8 (µs)	avg
		ICMP Ping	216.67 +/- 61.1 (µs)	avg
Bandwidth	Max Bandwidth for `send()` Size		Range [8, 65] KB	P99
	Local	16KB `send()`	2731 MB/s	p99
			2685 +/- 59 MB/s	avg
	Remote	16KB `send()`	114 +/- 0 MB/s	p99 & avg
	Remote	iperf3	117 MB/s	avg
Connection Overhead	Local	Set up	25 (µs)	p95 (unstable afterwards)
		Tear down	8.3 +/- 1.1 (µs)	avg
	Remote	Set up	124 (µs)	p90 (unstable afterwards)
		Tear down	6.8 +/- 1.2 (µs)	avg

File System

File Cache	\`Available\` memory in \`free -hm\`		5.2 GiB	-
	measured	read size = 4GiB	3095 +/- 183 MiB/s	avg
		read size = 5 GiB	266 +/- 76 MiB/S	avg
	Analysis	File Cache	in range [4,6] GiB	-
Read Time	Local Sequential	size = 0.5 GiB	172 +/- 3 MiB/S	avg
		size = 1 GiB	170 +/- 5 MiB/S	avg
		size = 2 GiB	130 +/- 38 MiB/S	avg
		size = 4 GiB	169 +/- 5 MiB/S	avg
		size = 8 GiB	174 +/- 0.4 MiB/S	avg
		size = 12 GiB	175 +/- 1 MiB/S	avg
	Remote Sequential	size = 0.5 GiB	92 +/- 3 MiB/S	avg
		size = 1 GiB	94 +/- 1 MiB/S	avg
		size = 2 GiB	92 +/- 2 MiB/S	avg
		size = 4 GiB	91 +/- 0 MiB/S	avg
		size = 8 GiB	88 +/- 2 MiB/S	avg
		size = 12 GiB	90 +/- 1 MiB/S	avg

Detailed Results

CPU

- Measurement overhead
    - Time:
        - Measurements: 100
        - Warm-Up Iters: 1'000'000
        - Iterations: 10'000'000
        - std::chrono::high_resolution_clock
            - ns
            - p99: 26, p999: 26, p9999: 26
            - mean: 26.647246647 ± 0.021858039
            - values = [26.654563100, 26.650156800, 26.641066900, 26.685132400, 26.634111700, ...] size: 100
        - RDTSCP

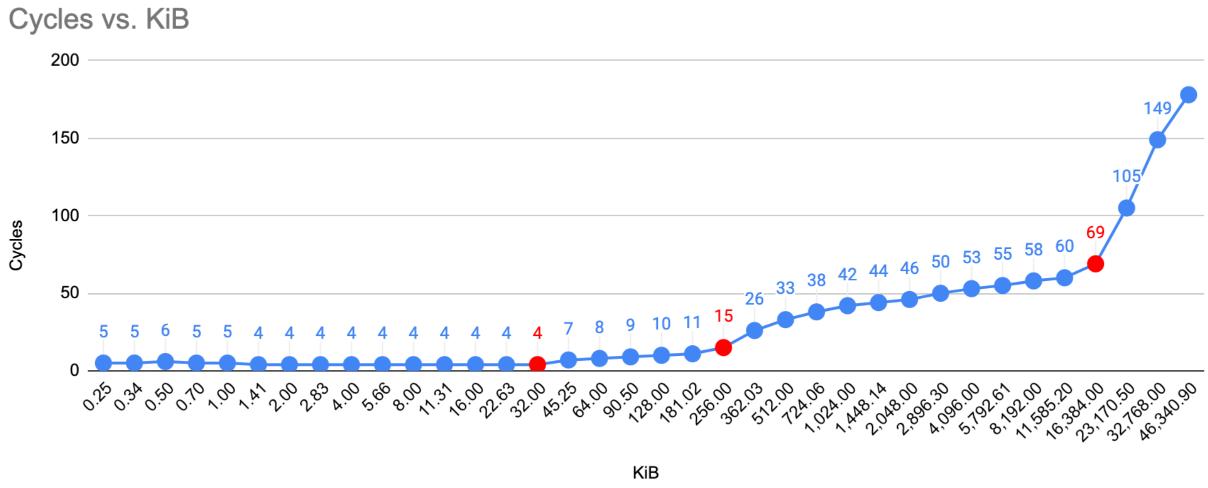
            - cycles
            - p99: 359, p999: 359, p9999: 359
            - mean: 362.390000000, std: 2.656670849, total: 100
            - values = [360.000000000, 363.000000000, 365.000000000, 365.000000000, 365.000000000, ...] size: 100
    - Loop
        - const int MAX_MEASUREMENTS = 1000;
        - const int MAX_ITERATIONS = 1'000'000;
        - Cycles
            - p99: 6, p999: 6, p9999: 6
            - mean: 6.06992 ± 0.0462562
            - values = [6.03059, 6.0913, 6.05484, 6.13522, 6.08833, ...] size: 1000
- Procedure call overhead
    - STD is too high for the measurement to be reliable. `taskset -c 5` was used to ping the process to a processor. Changing the number of the iterations did not lead to stabilization of the benchmark. Overhead seems to be close to a single cycle for each additional method parameter.
    - Results
        - cycles: inline function
        - p99: 48, p999: 48, p9999: 48
        - mean: 51.2718, std: 16.0877, total: 5000000
        -
        - cycles: 0 param function
        - p99: 52, p999: 52, p9999: 52
        - mean: 54.0266, std: 13.1513, total: 5000000
        -
        - cycles: 1 param function
        - p99: 52, p999: 52, p9999: 52
        - mean: 54.0337, std: 15.2782, total: 5000000
        -
        - cycles: 2 param function
        - p99: 52, p999: 52, p9999: 52
        - mean: 54.0277, std: 13.66, total: 5000000
        -
        - cycles: 3 param function
        - p99: 52, p999: 52, p9999: 52
        - mean: 53.2613, std: 27.9287, total: 5000000
        -
        - cycles: 4 param function
        - p99: 52, p999: 52, p9999: 52
        - mean: 54.6151, std: 384.813, total: 5000000
        -
        - cycles: 5param function

        - p99: 56, p999: 56, p9999: 56
        - mean: 57.9775, std: 17.4976, total: 5000000
        -
        - cycles: 6 param function
        - p99: 56, p999: 56, p9999: 56
        - mean: 56.2158, std: 13.9735, total: 5000000
        -
        - cycles: 7 param function
        - p99: 56, p999: 56, p9999: 56
        - mean: 56.0321, std: 14.0819, total: 5000000
        -
- System Call
    - | ns/op | op/s | err% | total | benchmark
    - |--------------------:|--------------------------:|----------:|-----------:|:---------------------------------
    - | 429.53 | 2,328,128.69 | 0.0% | 5.13 | `syscall getuid()`
    - | 30.22 | 33,090,433.19 | 0.1% | 0.36 | `syscall clock_gettime()`
- Task Creation
    - | ns/op | op/s | err% | total | benchmark
    - |----------------------:|------------------------:|----------:|-----------:|:------------------------------
    - | 75,302.89 | 13,279.70 | 0.1% | 8.99 | `create process fork()`
    - | 27,091.66 | 36,911.72 | 0.5% | 3.23 | `create and run thread`
    - | 14,239.32 | 70,228.07 | 0.6% | 1.70 | `create thread`
- Context Switch Time
    - Process
        - ./context-switch.out
        - Cycles
        - **p99: 2941**, p999: 2941, p9999: 2941
        - mean: 5348.43, std: 314198, total: 1000000
        - Seems unstable but p9999 tells a different story
    - Thread
        - taskset -c 5 ./context-switch-thread.out
        - thread context switch cycles:
        - **p99: 3041**, p999: 3041, p9999: 3041
        - mean: 5576.34, std: 319522, total: 1000000
        - values = [7200, 3264, 3136, 3156, 3140, ...] size: 1000000
        - Seems unstable but p9999 tells a different story

Memory

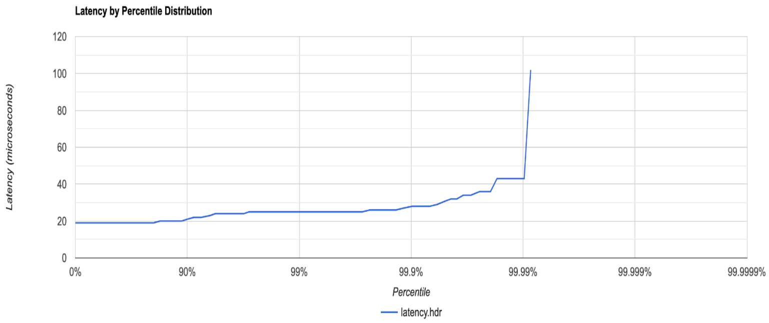
- Bandwidth
    - Theoretical:

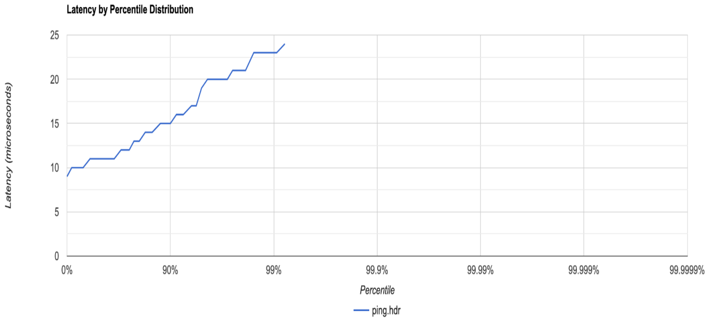
        - https://www.wolframalpha.com/input?i=1866+MHz+*+72+bits+*+8+to+GB%2Fs
        - 1866 MHz (DIMM freq) * 72 (data bus width) bits * 8 (channels) to GB/s = 134 GB/S
    - Measured:
        - Parallel Memset
        - MB/S
        - **p99: 133247**, p999: 133247, p9999: 133247
        - mean: 126851, std: 7618.61, total: 10240
- Latency
    - The data are based on P99 measurements



    - Change points are **32KiB, 256KiB, and close to 22MiB**
    - Linked list works fine but
- Page Fault Service Time
    - With dropped file cache (ms):
        - `sudo taskset -c 5  /usr/bin/time -v ./pagefault.out drop`
        - Major (requiring I/O) page faults: 8727
        - Minor (reclaiming a frame) page faults: 1713
        - Voluntary context switches: 9044
        - Measurements(ms): 7097, 7218, 7060, 7052, 7135, 7019, 7019, 7285, 7002
        - Mean: 7098 +/- 97
    - With file cache (ms):
        - `sudo taskset -c 5  /usr/bin/time -v ./pagefault.out no_drop`
        - Major (requiring I/O) page faults: 1
        - Minor (reclaiming a frame) page faults: 9828
        - Voluntary context switches: 3
        - Measurements(ms): 27, 27, 27, 27, 27, 27, 27
    - **Difference**
        - (7098 - 27) ms / 8727 (Major faults) = **810 Microseconds +/- 11**

Network

- Latency
    - round trip times in microseconds
    - Localhost Loopback device (127.0.0.1)
        - TCP Ping Pong (microseconds)
            - **p99: 25**, p999: 27, p9999: 40
            - **mean: 20.419**, std: 1.56746, **min: 18**, max: 107, total: 10240
            - 

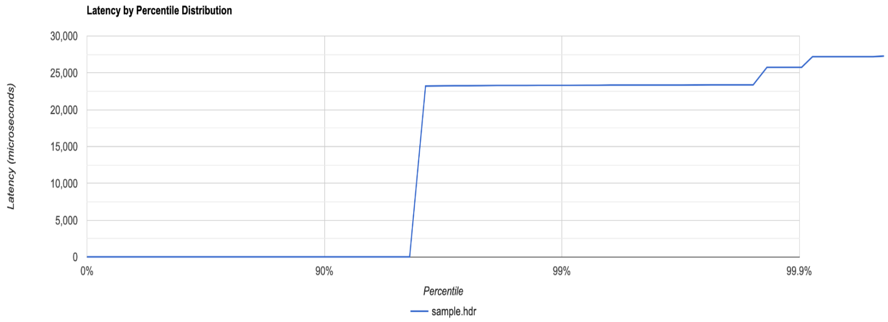
        - Ping ICMP (microseconds)
            - p99: 23, p999: 24, p9999: 24
            - mean: 11.9821, std: 2.78062, **min: 9**, max: 24, total: 112
            - 

        - Comparison
            - P99 is not that different. But min and mean value are respectively 10 microseconds apart.
    - Remote
        - TCP (microseconds)
            - **p99: 98**, p999: 142, p9999: 210
            - **mean: 88**.2061, std: 4.86171, min: 77, max: 210, total: 1024
        - Ping (microseconds)
            - p99: 428, p999: 467, p9999: 467
            - mean: 216.676, std: 61.1859, min: 1, max: 467, total: 148
        - Comparison
            - TCP numbers look ok. A typical datacenter network without any kernel optimizations operates at 50-100 microsecond latency. But

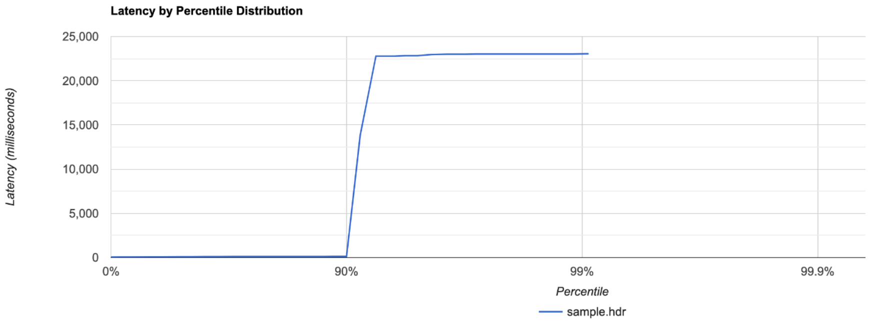
the ping numbers measured by the ping utility on linux are absurdly high.

- Peak Bandwidth
  - Local
    - With different packet sizes: Bytes -> MB/S
    - Max packet size ~= 16384-65536
      - packet: 64 = 70.3931
      - packet: 128 = 139.438
      - packet: 256 = 277.695
      - packet: 512 = 508.031
      - packet: 1024 = 910.222
      - packet: 2048 = 1424.7
      - packet: 4096 = 1985.94
      - packet: 8192 = 2340.57
      - packet: 16384 = 2730.67
      - packet: 32768 = 2730.67
      - packet: 65536 = 2730.67
      - packet: 131072 = 2621.44
      - packet: 262144 = 2730.67
      - packet: 524288 = 2621.44
      - packet: 1048576 = 2730.67
      - packet: 2097152 = 2520.62
      - packet: 4194304 = 2340.57
      - packet: 8388608 = 1820.44
      - packet: 16777216 = 1337.47
      - packet: 33554432 = 936.229
      - packet: 67108864 = 936.229
      - packet: 134217728 = 474.899
      - packet: 268435456 = 237.449
    - 100 Measurements for 16KB packets:
      - **p99: 2731**, p999: 2731, p9999: 2731
      - mean: 2685.1, std: 59.7678, min: 2520, max: 2731, total: 100
      - values = [2520.62, 2520.62, 2621.44, 2520.62, 2621.44, ...] size: 100
      - mean: 2685.04 ± 59.4739
  - Remote
    - Iperf3
      - Transferred 1.10 GBytes
      - From Sender: 943 Mbits/sec = 117.89 MB/sec
      - From Receiver: 941 Mbits/sec = 117.62 MB/sec
    - Hand written code
      - p99: 114, p999: 114, p9999: 114
      - **mean: 114**, std: 0, min: 114, max: 114, total: 100

- Connection Overhead
    - Local
        - Setup
            - **P95 = 25 microseconds**
            - p99: 23311, p999: 25759, p9999: 27295
            - mean: 1020.75, std: 4723.89, min: 20, max: 27295, total: 2048



-
        - Teardown
            - p99: 15, p999: 16, p9999: 18
            - **mean: 8.35596**, std: 1.13139, min: 8, max: 18, total: 2048
            - It is clear that the `close(socket)` is asynchronous and does not wait for the complete handshake to happen.
    - Remote
        - Setup
            - Exhibits the same sudden jump in values after a threshold (p90 here)
            - **P90: 124**
            - p99: 23023, p999: 23055, p9999: 23055
            - mean: 2058.55, std: 6303.35, min: 60, max: 23055, total: 100



-
        - Teardown
            - p99: 11, p999: 11, p9999: 11
            - **mean: 6.78,** std: 1.22947, min: 6, max: 11, total: 100

FileSystem

- File Cache
    - We use `</dev/zero head -c 56G` as a fake process to consume memory and then run the test. This is done so that the amount of memory on the server is limited. Otherwise we need to run the test to consume hundreds of GB of file cache. (slow test)
    - https://unix.stackexchange.com/a/254976/512340
    - `free -hm` before experiment
        - total used free shared buff/cache available
        - Mem: 62Gi 57Gi 327Mi 1.0Mi 347Mi **5.2Gi**
    - Read time for different sizes
        - read size: 512MB
        - p99: 3495, p999: 3495, p9999: 3495
        - mean: **3367**, std: 212.456, min: 2944, max: 3495, total: 5
        -
        - read size: 1024MB
        - p99: 3131, p999: 3131, p9999: 3131
        - mean: **3032**.2, std: 179.205, min: 2674, max: 3131, total: 5
        -
        - read size: 2048MB
        - p99: 3313, p999: 3313, p9999: 3313
        - mean: **3202**.2, std: 194.811, min: 2814, max: 3313, total: 5
        -
        - read size: 4096MB
        - p99: 3199, p999: 3199, p9999: 3199
        - mean: **3095**.4, std: 183.226, min: 2730, max: 3199, total: 5
        -
        - read size: 6144MB
        - p99: 409, p999: 409, p9999: 409
        - mean: **266**, std: 76.2863, min: 192, max: 409, total: 5
    - Analysis
        - Read bandwidth falls from 3GB/s to 266 MB/s on a 6GB read. -> the size of the file cache is between 4GB and 6 GB.
- Read Time
    - Sequential
        - read size: 512MB
        - p99: 177, p999: 177, p9999: 177
        - mean: 172.2, std: 3.86782, min: 167, max: 177, total: 5
        -
        - read size: 1024MB
        - p99: 177, p999: 177, p9999: 177
        - mean: 170.4, std: 5.60714, min: 160, max: 177, total: 5

        -
        - read size: 2048MB
        - p99: 171, p999: 171, p9999: 171
        - mean: 130.6, std: 38.5206, min: 79, max: 171, total: 5
        -
        - read size: 4096MB
        - p99: 174, p999: 174, p9999: 174
        - mean: 169.2, std: 5.26878, min: 160, max: 174, total: 5
        -
        - read size: 8192MB
        - p99: 176, p999: 176, p9999: 176
        - mean: 175.6, std: 0.489898, min: 175, max: 176, total: 5
        -
        - read size: 12288MB
        - p99: 177, p999: 177, p9999: 177
        - mean: 175, std: 1.09545, min: 174, max: 177, total: 5
    - Analysis
        - Not that dependent on the file size
        - Note that file cache is dropped on each measurement
- Remote Read Time
    - Conveniently loaded as `/proj/os-benchmark-PG0/` by cloudlab on the machine
    - Sequential
        - read size: 512MB
        - p99: 98, p999: 98, p9999: 98
        - mean: 92.4, std: 3.55528, min: 89, max: 98, total: 5
        -
        - read size: 1024MB
        - p99: 95, p999: 95, p9999: 95
        - mean: 94.2, std: 1.16619, min: 92, max: 95, total: 5
        -
        - read size: 2048MB
        - p99: 96, p999: 96, p9999: 96
        - mean: 92.2, std: 2.4, min: 90, max: 96, total: 5
        -
        - read size: 4096MB
        - p99: 92, p999: 92, p9999: 92
        - mean: 91, std: 0.894427, min: 90, max: 92, total: 5
        -
        - read size: 8192MB
        - p99: 91, p999: 91, p9999: 91
        - mean: 88.2, std: 1.93907, min: 86, max: 91, total: 5
        -
        - read size: 12288MB
        - p99: 91, p999: 91, p9999: 91

- mean: 90.2, std: 1.16619, min: 88, max: 91, total: 5