Performance Analysis

Parallel Storage Systems

2023-06-26

Jun.-Prof. Dr. Michael Kuhn
michael.kuhn@ovgu.de

Parallel Computing and I/O
Institute for Intelligent Cooperating Systems
Faculty of Computer Science
Otto von Guericke University Magdeburg
https://parcio.ovgu.de

Outline

Optimizations

- What is the difference between write-behind and write-through caching?
  1. Write-behind writes to the device first and to the cache afterwards
  2. Write-behind writes to the cache and the device at the same time
  3. Write-through writes to the cache and the device at the same time
  4. Write-through only writes to the device and circumvents the cache

- What does data sieving do?
    1. Data sieving turns contiguous accesses into non-contiguous ones
    2. Data sieving turns non-contiguous accesses into contiguous ones
    3. Data sieving allows having holes in derived data types

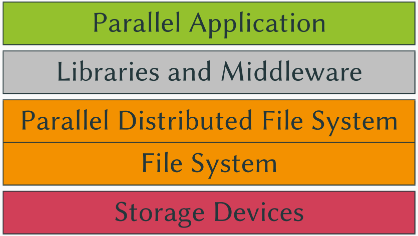
Optimizations

- What does the Two Phase optimization do?
  1. Split up file into domains and coordinate I/O operations among processes
  2. Perform I/O on one process and distribute data to other processes
  3. Read data after a write operation to check whether write was successful

Outline

Introduction

- Performance analysis can be hard to perform
  - Software and hardware get more complex
  - Many layers are involved and interact
- Performance analysis consists of two parts
  - Performance measurement and assessment
- Measurement gives indication of actual performance
  - Measuring correctly is a topic of its own
- Assessment to determine potential performance
  - Important when buying a new storage system etc.



Introduction...

- Performance measurement
  - How to measure performance?
  - How long do measurements have to be?
  - How often do measurements have to be repeated?
  - Is it possible to eliminate external influences?
- Performance assessment
  - Which performance can we potentially achieve?
  - Which performance can we expect in practice?

Outline

Measurements

- Measuring performance is a complex process
  - Performance is influenced by caching, network, I/O etc.
  - Which components are involved and have to be measured?
  - Which performance can we expect on a given system?
- Our goal is to collect metrics quantitatively
  - Metrics include runtime, throughput, latency and more
  - The metrics to collect depend on the software and hardware
- Published measurements should be scientifically sound
  - Other scientists should be able to reproduce your findings
  - Measurements of metrics have errors that have to be accounted for

Quiz

- Application A runs for 4.274 s, application B for 4.176 s. Which one is faster?
  1. Application A
  2. Application B
  3. Difference is negligible, performance is the same
  4. Not enough information

Measurements…

- Single measurements are more or less random
  - Processor might be busy with something else
  - Some other application is currently occupying the network
  - There is a certain variability for each component
- It is never enough to do a single measurement
  - Always repeat measurements at least three times
  - If you talk to physicists, they will probably say 30 times
- Averaging the metrics is also not enough
  - There are important derived metrics, such as standard deviation etc.

Measurements...

```
Benchmark #1: ./sincos-O2
Time (mean +- sig):   4.192 s +-  0.033 s   [User: 4.181 s, System: 0.001 s]
Range (min .. max):   4.160 s ..  4.274 s   10 runs

Benchmark #2: ./sincos-O3
Time (mean +- sig):   4.191 s +-  0.016 s   [User: 4.179 s, System: 0.001 s]
Range (min .. max):   4.176 s ..  4.221 s   10 runs

Summary
'./sincos-O3' ran
    1.00 +- 0.01 times faster than './sincos-O2'
```

- Application A and B have the same performance
  - Both previous results were extreme values (minimum and maximum)

- There are two kinds of errors
  1. Random errors
     - Might be caused by operating system activity in the background
     - Performance of most hardware varies a bit
     - Larger variations are also possible due to hardware defects, load balancing etc.
  2. Systematic errors
     - Might be caused by wrong methodology/implementation
     - For instance, you want to measure disk speed but hit the cache

- Which errors can we get rid of by repeating measurements?
    1. Random and systematic errors
    2. Random errors
    3. Systematic errors
    4. None

Measurements. . .

- Always use a well-defined hardware/software environment
  - Document the setup, including version numbers etc.
- Minimize external influence to keep random errors low
  - Use resources exclusively if possible
  - Do not run anything intensive in the background
- Increase measurement time and repeat measurements
  - This helps canceling out random errors
- Compare results with expected performance
  - "My application finishes in two hours. Could it finish in one?"
  - This typically involves some kind of performance modeling

Measurements…

- There is a wide range of benchmarks available
  - For processors, caches, main memory, network etc.
- There are also many I/O benchmarks, each with a different focus
  - IOzone, Bonnie, Bonnie++, PostMark, b_eff_io, FLASH I/O and many more
  - We will look at three examples: fio, IOR and mdtest
- Benchmarks typically only cover certain access patterns
  - This leads to many different benchmarks for different use cases

Example: fio [Axboe, 2021]

- fio is a flexible I/O tester
  - The main author is Jens Axboe, maintainer of Linux's block layer
    - He is also responsible for the cfq, noop and deadline schedulers
    - Developed the `blktrace` tool and the `splice` system call
- fio is able to measure arbitrary workloads
  - Typically requires many different specialized tests
- Usage is supported by so-called job files
  - Users can set common and job-specific parameters
  - Everything can also be controlled using the command line
- Limitation: Parallelism is only supported locally via processes/threads

Example: fio…

- Operation types
  - Read/write/mixed as well as sequential/random
  - Buffered, direct or fsync to include or exclude cache's influence
- Block size and total data size
  - Single values as well as ranges
  - File and thread counts for parallel workloads
- I/O engine
  - Synchronuous, asynchronous, memory mapping and null
  - Queue depth for asynchronous engines
- Preallocation and optimizations using `fallocate` and `fadvise`
  - Focus on block allocation or certain optimizations

Example: fio…

- Locks and alignment
  - None, exclusive and non-exclusive read
  - I/O can be aligned to stripes etc.
- Throughput limit
  - To simulate background load
- Compressibility and deduplicatibility
  - Current SSDs and file system compress data transparently
- Verification
  - Check whether read data matches the written data

Example: fio… [Axboe, 2021]

- Randomly read from 128 MiB large files
  - Files are created automatically for the test
- Two processes `job1` and `job2` are used
  - File names are also generated automatically
- Can also be specified using the command line
  - `fio --rw=randread --size=128m --name job1 --name job2`

```
[global]
rw=randread
size=128m

[job1]

[job2]
```

Example: fio… [Axboe, 2021]

- Asynchronous I/O with a depth of 4
  - Four asynchronous I/O operations are pending at once
  - Might be necessary to achieve full performance
- Four processes write randomly using buffered I/O
  - Process-local 64 MiB files with an access size of 32 KiB
- CLI: `fio --name=random-writers ...`

```
[random-writers]
ioengine=libaio
iodepth=4
rw=randwrite
blocksize=32k
direct=0
size=64m
numjobs=4
```

Example: fio…

- fio also supports trace replay
  - That is, fio can execute access patterns recorded in a log
  - Makes it possible to generate I/O load without application
  - Easier to compare systems with different software environments
- Especially useful for complex real-world applications
  - Many dependencies, hard to compile and execute
- Supports blktrace and its own format
  - blktrace format is binary
  - fio format is plain text and can be generated easily
    - `write_iolog` and `read_iolog` can be used for logging

Example: IOR [IOR Developers, 2021]

- IOR supports parallel I/O across different nodes
  - fio only allows multiple processes on a single node
  - Parallel distributed file systems require multiple nodes
- IOR supports multiple backends
  - Dummy, HDF5, HDFS, IME, mmap, MPI-IO, Parallel-NetCDF, POSIX, RADOS, S3 etc.
- There is supports for different I/O modes
  - Shared or process-local files
  - Processes can be reordered to circumvent the cache
    - For instance, client X writes data, client X+n reads data

Example: IOR. . .

- Data is written using MPI-IO
  - Other interfaces provided by backends
  - Reading is disabled, file is deleted afterwards
- All processes access a shared file
  - Processes use an access size of 1 MiB
  - Each process is responsible for a block of 1 GiB
  - File is split up into 10 segments
  - Everything is repeated three times
- Also possible to specify using command line

```
IOR START
  api=MPIIO
  testFile=/path/to/file
  repetitions=3
  readFile=0
  writeFile=1
  filePerProc=0
  keepFile=0
  segmentCount=10
  blockSize=1g
  transferSize=1m
RUN
IOR STOP
```

Example: IOR… [Shan and Shalf, 2007]

- File structure inspired by real-world scientific applications
  - Accesses happen with transfer size
  - Processes access blocks exclusively
  - Segments represent time steps etc.
- All processes access one shared file
  - Alternatively, one file per process

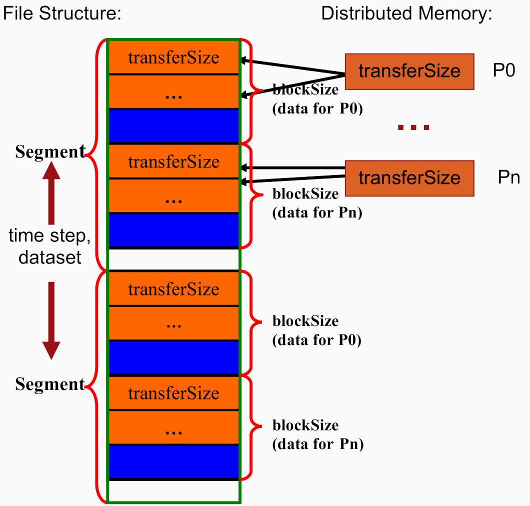


Fig. 1. The design of the IOR benchmark for shared file type. Blocks are stored in separate files for the 1-file-per-processor mode of operation.

Example: mdtest

- Most benchmarks measure data throughput
  - Metadata performance is an important factor
- mdtest uses MPI for parallel metadata access
  - Uses the same backends as IOR to perform operations
  - Supported functionality very similar to IOR
- Split up into multiple phases
  - Creating, writing, getting status, reading, removing etc.
- Uses a hierarchical directory structure
  - Multiple root directories to test several metadata servers

Tools

- Multitude of benchmarks for vastly different use cases
  - Typically focused on either data or metadata
- Results are often not easily comparable
  - Different access patterns
  - Different computation of results
  - Different behavior (synchronization, locking etc.)
- Results can be hard to interpret
  - MB vs. MiB (difference of $\approx 10\,\%$ for TB/s)

Tools...

- Benchmarks only allow us to measure the current performance
  - They cannot tell us reasons for performance problems etc.
  - Benchmarks do not necessarily use realistic I/O patterns
- Analysis and optimization require additional tools
  - We need to be able to get an insight into the inner workings
  - Tracing is often used to record all activity (Score-P)
- Abstracted performance metrics are sometimes enough to get an overview
  - For instance, we can characterize the I/O behavior (Darshan)

Example: Darshan

- Darshan is a tool to characterize I/O behavior
  - Sanskrit for “sight” or “vision”
- We want to get a useful picture of application I/O
  - This includes information about I/O patterns
  - Overhead should be as low as possible to not influence behavior
- Darshan is designed for permanent use
  - Tested with applications using more than 750,000 cores
- Solid support for MPICH
  - Developed at Argonne National Laboratory
  - Group that also develops OrangeFS, MPICH and ROMIO

Example: Darshan. . .

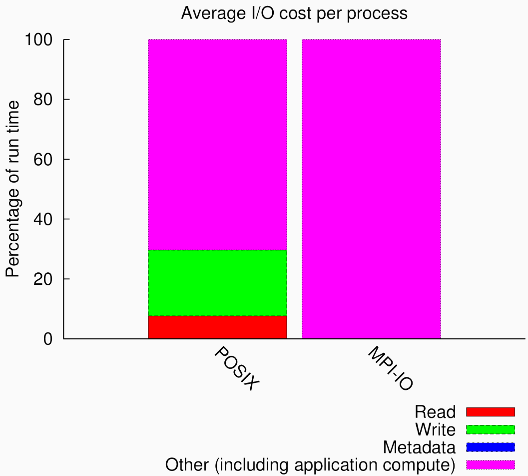
- Darshan consists of two parts
  - Runtime and analysis tools
- Runtime records the application's I/O
  - Has to be compiled for a specific MPI implementation
  - Supports options for batch schedulers and a shared log directory
  - Offers compiler wrappers and a preload library `libdarshan.so`
- Tools analyze the recorded application logs
  - `darshan-job-summary.pl`, `darshan-parser` etc.

Example: Darshan...

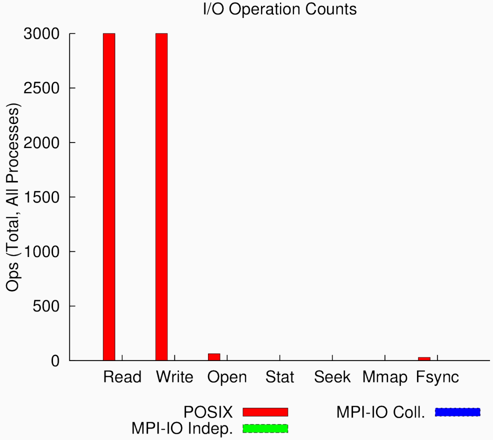
- Assume an MPI-parallelized POSIX benchmark with ten processes
  - First a write phase, followed by a read phase
    - Both phases are separated by barriers
  - Use a block size of 1 MiB
    - Write or read 100 blocks in total
  - Cache is dropped in between the phases
    - `echo 3 > /proc/sys/vm/drop_caches`
  - `fsync` is called before closing the file
    - Only after writing, file is re-opened for reading
  - The whole process is repeated three times

Example: Darshan. . .

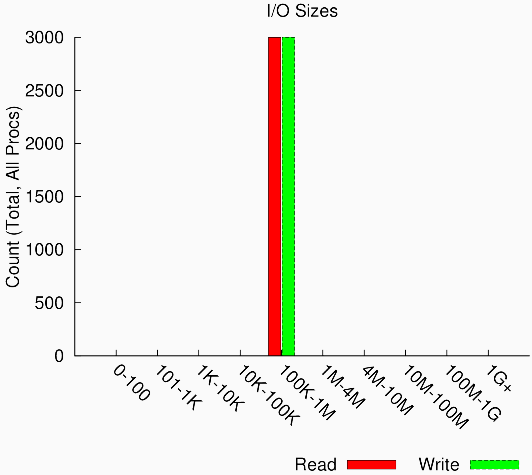
- Darshan aggregates operations
  - According to interface and operation type
- Benchmark does no computation
  - “Other” is still very high
  - Likely due to barriers etc.



- 3,000 read and write operations
  - 10 processes × 100 operations × 3 repetitions
- 60 open operations
  - 10 processes × 2 phases × 3 repetitions
- 30 sync operations
  - 10 processes × 3 repetitions



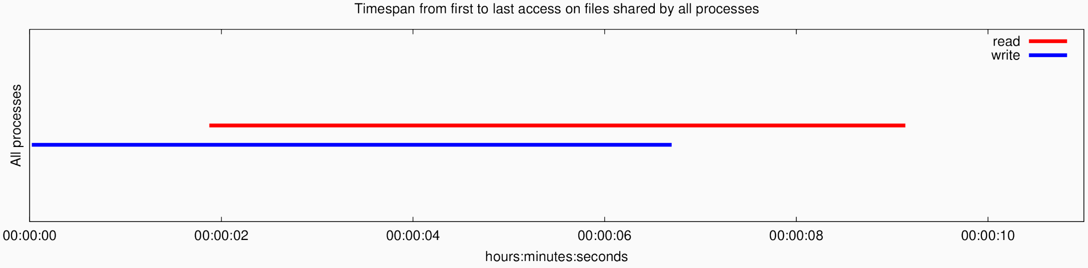
- All operations have 1 MiB size
  - No operations are split up
- Small accesses would hint at inefficient I/O



Example: Darshan…

- Darshan can differentiate access patterns
- Sequential
  - Accesses with increasing offset
- Consecutive
  - Directly adjacent to previous access

Example: Darshan...



- High-level timeline for I/O operations
  - Displays timelines, which can be deceptive due to three repetitions
  - Timelines also do not work well for checkpointing

- Darshan offers a coarse-grained overview of I/O costs
    - Characterizes I/O according to access counts, sizes and patterns
- Allows determining whether optimizations are necessary
    - More in-depth analyses might be necessary
    - Darshan also supports an extended tracing mode (DxT)

Outline

Introduction

- Assessing performance by modeling theoretical performance
    - Compare Rmax and Rpeak on the TOP500 list
- Requires collecting information about the system
    - Which components are involved?
    - Which performance characteristics do these components have?
- Often necessary to measure individual components
    - Requires a different set of tools

Example

- Is this performance good?

File System	Block Size	1 PPN	6 PPN	12 PPN
Lustre	1 MiB	640 MiB/s	105 MiB/s	110 MiB/s
OrangeFS	1 MiB	160 MiB/s	390 MiB/s	430 MiB/s
OrangeFS	64 KiB	250 MiB/s	115 MiB/s	180 MiB/s

Write

File System	Block Size	1 PPN	6 PPN	12 PPN
Lustre	1 MiB	1,095 MiB/s	735 MiB/s	875 MiB/s
OrangeFS	1 MiB	130 MiB/s	265 MiB/s	430 MiB/s
OrangeFS	64 KiB	505 MiB/s	140 MiB/s	195 MiB/s

Read

Example

- Is this performance good?
- Block size
  - Why is 64 KiB better than 1 MiB for 1 PPN?
- Throughput
  - Why is the maximum 1.1 GiB/s?
  - Why is write lower than read?

File System	Block Size	1 PPN	6 PPN	12 PPN
Lustre	1 MiB	640 MiB/s	105 MiB/s	110 MiB/s
OrangeFS	1 MiB	160 MiB/s	390 MiB/s	430 MiB/s
OrangeFS	64 KiB	250 MiB/s	115 MiB/s	180 MiB/s

Write

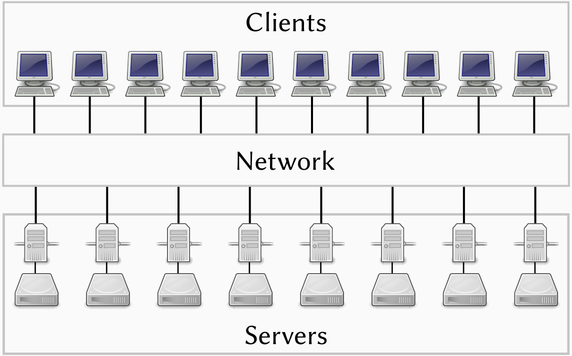
File System	Block Size	1 PPN	6 PPN	12 PPN
Lustre	1 MiB	1,095 MiB/s	735 MiB/s	875 MiB/s
OrangeFS	1 MiB	130 MiB/s	265 MiB/s	430 MiB/s
OrangeFS	64 KiB	505 MiB/s	140 MiB/s	195 MiB/s

Read

Quiz

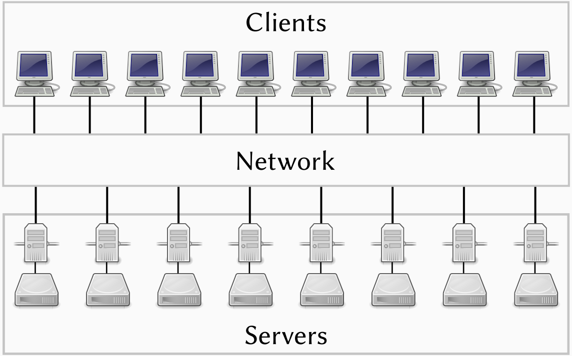
- Which components would you evaluate for a performance assessment?
    1. CPUs
    2. Main memory
    3. Network
    4. Storage devices

Overview



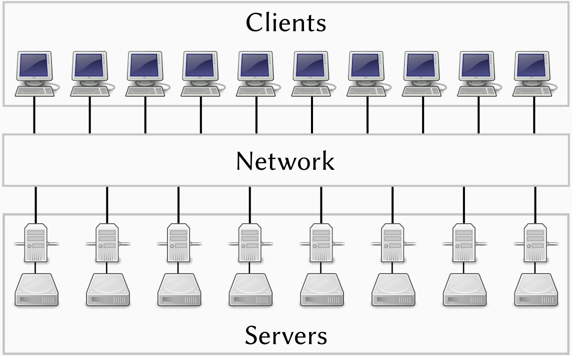
Overview

- Clients: IOPS, RAM throughput, network connection



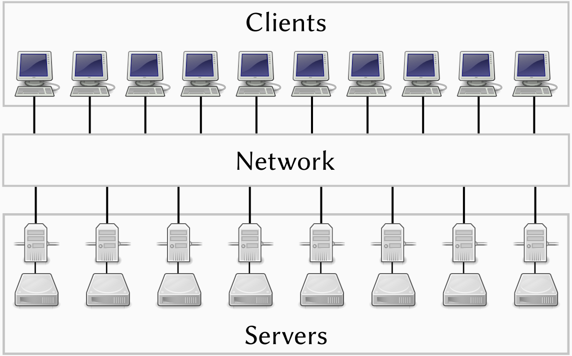
Overview

- Clients: IOPS, RAM throughput, network connection
- Network: Throughput and latency



Overview

- Clients: IOPS, RAM throughput, network connection
- Network: Throughput and latency
- Servers: Throughput and IOPS



Clients

- I/O operations per second (IOPS)
    - Context switches could limit performance
- Throughput and latency of main memory
    - Typically not a problem (if we avoid unnecessary copies)
- Estimate performance using `tmpfs` and `fio`
    - Idea: Perform many small I/O operations

```
$ mkdir /tmp/fs
$ mount -t tmpfs tmpfs /tmp/fs
$ ...
$ umount /tmp/fs
```

- Standard compute nodes

```
$ fio --name=cs                \
      --filename=/tmp/fs/foo \
      --rw=write --bs=1      \
      --size=1g --runtime=60 \
      [--numjobs=n]

$ vmstat 1

$ fio --name=bw                \
      --filename=/tmp/fs/foo \
      --rw=write --bs=1m     \
      --size=5g --runtime=60
```

Clients…

- Standard compute nodes
- ≈ 1,000,000 IOPS
  - Block size of 1 is important
  - 0 could be intercepted by libc

```
$ fio --name=cs                    \
      --filename=/tmp/fs/foo \
      --rw=write --bs=1           \
      --size=1g --runtime=60 \
      [--numjobs=n]

$ vmstat 1

$ fio --name=bw                    \
      --filename=/tmp/fs/foo \
      --rw=write --bs=1m         \
      --size=5g --runtime=60
```

Clients…

- Standard compute nodes
- ≈ 1,000,000 IOPS
  - Block size of 1 is important
  - 0 could be intercepted by libc
- ≈ 330,000 context switches

```
$ fio --name=cs                \
      --filename=/tmp/fs/foo \
      --rw=write --bs=1        \
      --size=1g --runtime=60 \
      [--numjobs=n]

$ vmstat 1

$ fio --name=bw                \
      --filename=/tmp/fs/foo \
      --rw=write --bs=1m       \
      --size=5g --runtime=60
```

Clients. . .

- Standard compute nodes
- ≈ 1,000,000 IOPS
  - Block size of 1 is important
  - 0 could be intercepted by libc
- ≈ 330,000 context switches
- ≈ 4 GiB/s throughput
  - Main memory is typically faster
  - tmpfs introduces overhead

```
$ fio --name=cs               \
      --filename=/tmp/fs/foo \
      --rw=write --bs=1       \
      --size=1g --runtime=60 \
      [--numjobs=n]

$ vmstat 1

$ fio --name=bw               \
      --filename=/tmp/fs/foo \
      --rw=write --bs=1m      \
      --size=5g --runtime=60
```

- Standard compute nodes
- ≈ 1,000,000 IOPS
  - Block size of 1 is important
  - 0 could be intercepted by libc
- ≈ 330,000 context switches
- ≈ 4 GiB/s throughput
  - Main memory is typically faster
  - tmpfs introduces overhead
- No limitations for our previous results

```
$ fio --name=cs                \
      --filename=/tmp/fs/foo \
      --rw=write --bs=1        \
      --size=1g --runtime=60 \
      [--numjobs=n]

$ vmstat 1

$ fio --name=bw                \
      --filename=/tmp/fs/foo \
      --rw=write --bs=1m       \
      --size=5g --runtime=60
```

Network

- Different performance characteristics depending on network
  - InfiniBand vs. Ethernet
- Network throughput can become a bottleneck
  - Need to be able to saturate storage devices
- Numbers of packets per second
  - Important for metadata operations
  - Limits performance for many small messages
- Measurements can be done with `ping` and `iperf`

Network…

- Between compute and storage nodes
- Round trip time ≈ 0.100 ms
- Throughput ≈ 110 MiB/s

```
$ ping -c 10000 -f $host

$ iperf --server \
        --port $port

$ iperf --client $host \
        --port $port
```

Network…

- Between compute and storage nodes
- Round trip time ≈ 0.100 ms
  - Corresponds to ≈ 10,000 messages per second
- Throughput ≈ 110 MiB/s
  - Corresponds to 1 Gbit/s Ethernet

```
$ ping -c 10000 -f $host

$ iperf --server \
        --port $port

$ iperf --client $host \
        --port $port
```

- Between compute and storage nodes
- Round trip time ≈ 0.100 ms
    - Corresponds to ≈ 10,000 messages per second
- Throughput ≈ 110 MiB/s
    - Corresponds to 1 Gbit/s Ethernet
- Both could limit our performance

```
$ ping -c 10000 -f $host

$ iperf --server \
        --port $port

$ iperf --client $host \
        --port $port
```

Storage Devices

- Performance heavily depends on storage technology
  - HDD vs. SSD
- Throughput important for data operations
  - Should be higher than network throughput to have reserves
- IOPS important for metadata operations
  - Also crucial for small random accesses
- Storage bus can be a bottleneck
  - SATA devices often support SATA 3.0 (600 MB/s)

- Unbuffered I/O to measure devices
  - Avoid page cache influences

```
$ fio --name=iops                   \
      --filename=/dev/sd?           \
      --direct=1 --rw=randread \
      --bs=4k --size=$size          \
      --runtime=60

$ fio --name=bw               \
      --filename=/dev/sd?  \
      --direct=1 --rw=read \
      --bs=1m --size=$size \
      --runtime=60
```

- Unbuffered I/O to measure devices
  - Avoid page cache influences
- HDDs
  - IOPS ≈ 60–80
  - Throughput ≈ 120 MiB/s

```
$ fio --name=iops              \
      --filename=/dev/sd?      \
      --direct=1 --rw=randread \
      --bs=4k --size=$size     \
      --runtime=60

$ fio --name=bw            \
      --filename=/dev/sd?  \
      --direct=1 --rw=read \
      --bs=1m --size=$size \
      --runtime=60
```

- Unbuffered I/O to measure devices
  - Avoid page cache influences
- HDDs
  - IOPS ≈ 60–80
  - Throughput ≈ 120 MiB/s
- SSDs
  - IOPS ≈ 15,000
    - Outlier ≈ 5,500 (might be garbage collection)
  - Throughput ≈ 270 MiB/s

```
$ fio --name=iops                  \
      --filename=/dev/sd?          \
      --direct=1 --rw=randread \
      --bs=4k --size=$size         \
      --runtime=60

$ fio --name=bw              \
      --filename=/dev/sd?  \
      --direct=1 --rw=read \
      --bs=1m --size=$size \
      --runtime=60
```

- Unbuffered I/O to measure devices
  - Avoid page cache influences
- HDDs
  - IOPS ≈ 60–80
  - Throughput ≈ 120 MiB/s
- SSDs
  - IOPS ≈ 15,000
    - Outlier ≈ 5,500 (might be garbage collection)
  - Throughput ≈ 270 MiB/s
- Faster than network, therefore no limitation

```
$ fio --name=iops                  \
      --filename=/dev/sd?          \
      --direct=1 --rw=randread \
      --bs=4k --size=$size         \
      --runtime=60

$ fio --name=bw              \
      --filename=/dev/sd?    \
      --direct=1 --rw=read \
      --bs=1m --size=$size \
      --runtime=60
```

Example

- Let's analyze the previous results in more detail
  - Lustre and OrangeFS are compared with each other
- Different block sizes are used
  - 1 MiB and 64 KiB
  - Corresponds to the default stripe size of Lustre and OrangeFS
- Reminder: Network can do 10,000 messages per second
  - Results in maximum of 9.8 GiB/s (1 MiB) or 625 MiB/s (64 KiB) per node
- Network throughput determines maximum performance
  - We can achieve at most 1,100 MiB/s

Example...

- Processes per node (PPN)
- Performance degradation with higher PPN on Lustre
  - Shared access to OST
  - Reading unproblematic
- Fitting block size works better for one process per node
  - Results in better alignment
- Lustre with 1 PPN hits network limit

File System	Block Size	1 PPN	6 PPN	12 PPN
Lustre	1 MiB	640 MiB/s	105 MiB/s	110 MiB/s
OrangeFS	1 MiB	160 MiB/s	390 MiB/s	430 MiB/s
OrangeFS	64 KiB	250 MiB/s	115 MiB/s	180 MiB/s

Write

File System	Block Size	1 PPN	6 PPN	12 PPN
Lustre	1 MiB	1,095 MiB/s	735 MiB/s	875 MiB/s
OrangeFS	1 MiB	130 MiB/s	265 MiB/s	430 MiB/s
OrangeFS	64 KiB	505 MiB/s	140 MiB/s	195 MiB/s

Read

Example...

- Adapt block size to stripe size
  - Divide block size by PPN
- Improves performance
  - Writing for Lustre
  - Reading for OrangeFS
  - OrangeFS with 4 PPN hits network limit
- Reading anomaly with Lustre
  - Higher than network
  - Might not stand out without performance assessment

File System	Block Size	1 PPN	4 PPN	8 PPN
Lustre	1/PPN MiB	640 MiB/s	620 MiB/s	605 MiB/s
OrangeFS	64/PPN KiB	250 MiB/s	280 MiB/s	210 MiB/s

Write

File System	Block Size	1 PPN	4 PPN	8 PPN
Lustre	1/PPN MiB	1,095 MiB/s	1,800 MiB/s	525 MiB/s
OrangeFS	64/PPN KiB	505 MiB/s	655 MiB/s	455 MiB/s

Read

Outline

Summary

- Wide range of benchmarks and tools to measure performance
  - Different benchmarks cover different use cases and access patterns
- Measurements alone do not say anything about achievable performance
  - Performance assessment and modeling are necessary
- Rough performance model is often already good enough
  - Determine whether results are realistic, can be refined if necessary
- Actual performance can be unpredictable
  - Unexpected side effects such as caching, garbage collection etc.

References

[Axboe, 2021] Axboe, J. (2021). **fio - Flexible IO Tester.** http://git.kernel.dk/?p=fio.git;a=summary.

[IOR Developers, 2021] IOR Developers (2021). **IOR - Parallel filesystem I/O benchmark.** https://github.com/hpc/ior.

[Shan and Shalf, 2007] Shan, H. and Shalf, J. (2007). **Using IOR to Analyze the I/O Performance for HPC Platforms.** In *Cray User Group Conference (CUG'07)*.