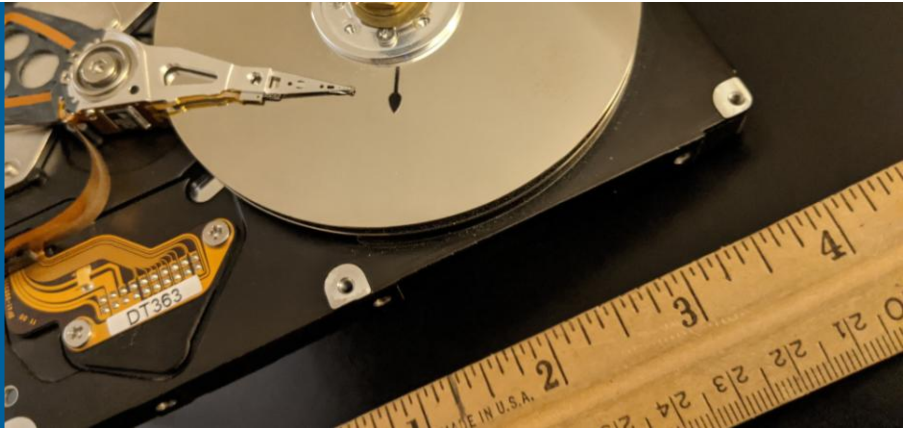
5TH INTERNATIONAL PARALLEL DATA SYSTEMS WORKSHOP

Argonne NATIONAL LABORATORY

KEEPING IT REAL

WHY HPC DATA SERVICES DON'T ACHIEVE MICROBENCHMARK PERFORMANCE



PHIL CARNS[1], KEVIN HARMS[1], BRAD SETTLEMYER[2], BRIAN ATKINSON[2], AND ROB ROSS[1]

carns@mcs.anl.gov

[1]Argonne National Laboratory

[2]Los Alamos National Laboratory

UCHICAGO ARGONNE LLC

U.S. DEPARTMENT OF ENERGY

Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.

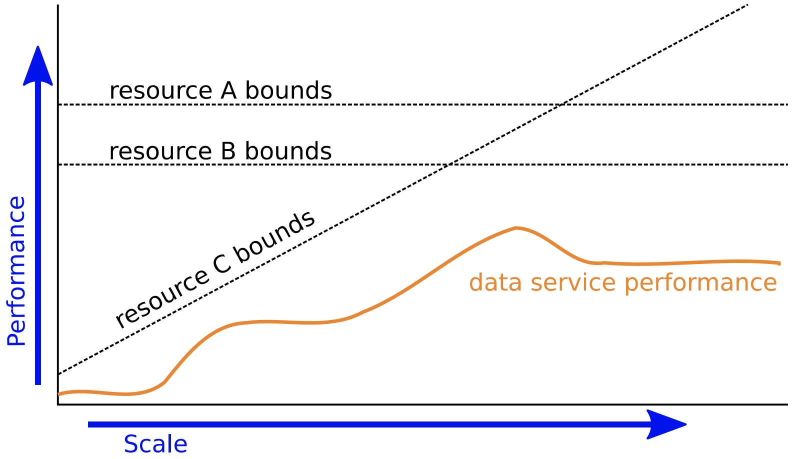
November 12, 2020 (virtual)

HPC DATA SERVICE PERFORMANCE

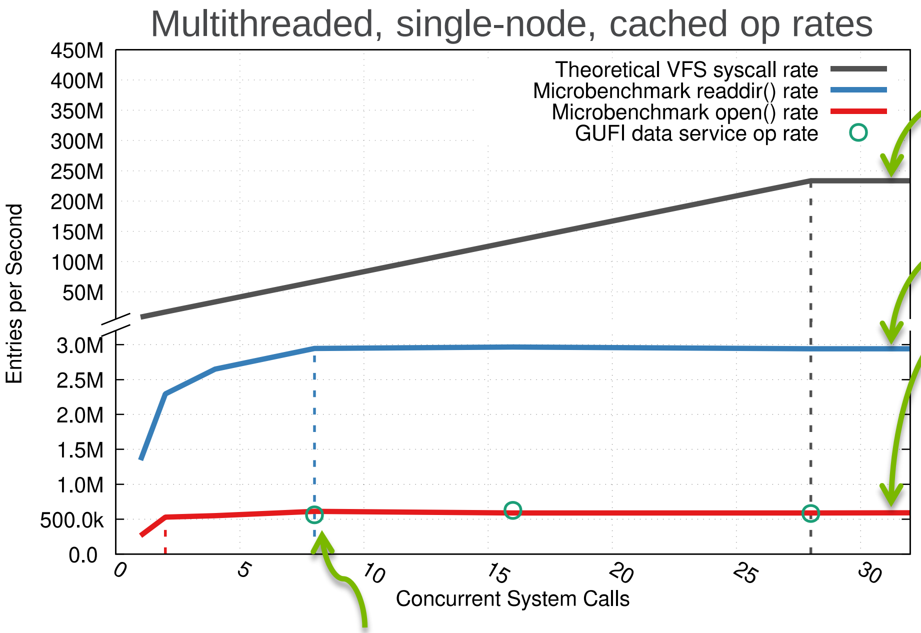
- HPC data service (e.g. file system) performance is difficult to interpret in isolation.
- Performance observations must be oriented in terms of trusted reference points.
- One way to approach this is by constructing ***roofline models for HPC I/O:***

- How does data service performance compare to platform capabilities?
- Where are the bottlenecks?
- Where should we optimize?

How do we find these rooflines?



HPC I/O ROOFLINE EXAMPLE



- Theoretical bound based on projected system call rate
- Actual bounds based on local file system *microbenchmarks*
- Microbenchmarks + rooflines:
  - Help identify true limiting factors
  - Help identify scaling limitations
  - *Might be harder to construct and use than you expect.*

GUFI: metadata traversal rate observed in a metadata indexing service (https://github.com/mar-file-system/GUFI). The open() and readdir() system calls account for most of the GUFI execution time.

THE DARK SIDE OF MICROBENCHMARKING

♫ Bum bum bummmmm ♫

Employing microbenchmarks for rooflines is straightforward in principle:

1. **Measure performance of components**.
2. Use the measurements to construct rooflines in a relevant parameter space.
3. Plot actual data service performance relative to the rooflines.

This presentation focuses on potential pitfalls in **step 1**:

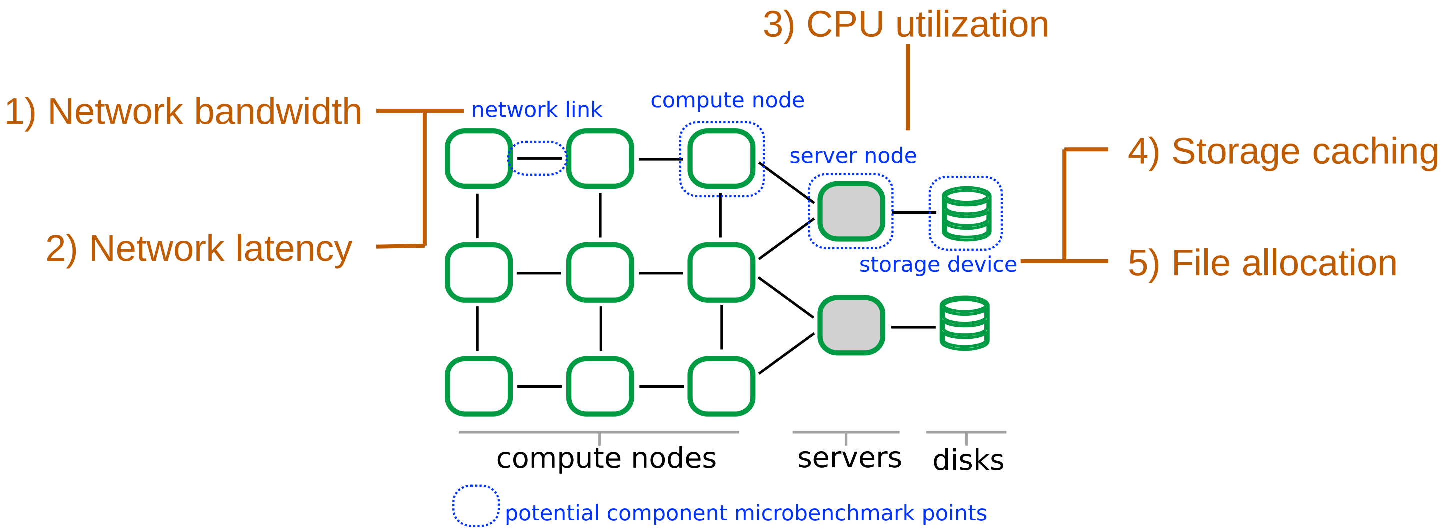
- Do benchmark authors and service developers agree on what to measure?
- Are the benchmark parameters known and adequately reported?
- Are the benchmark workloads appropriate?
- Are the results interpreted and presented correctly?

HPC STORAGE SYSTEM COMPONENTS

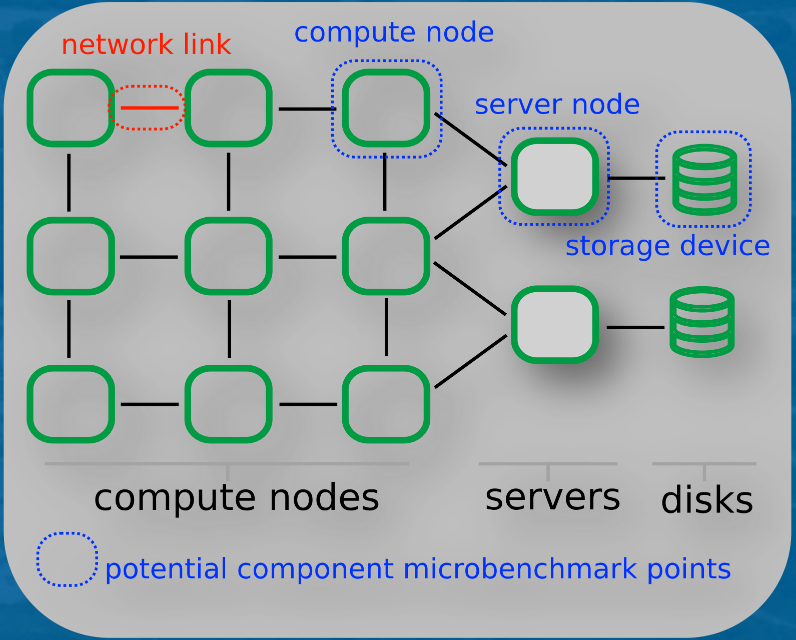
Illustrative examples

We will focus on 5 examples drawn from practical experience benchmarking OLCF and ALCF system components:

Please see Artifact Description appendix (and associated DOI) for precise experiment details.



NETWORK CASE STUDIES



CASE STUDY 1: BACKGROUND

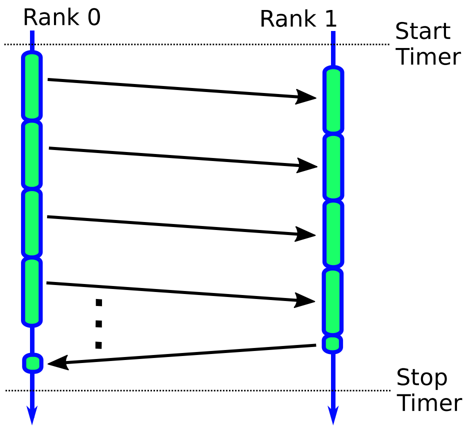
Network Bandwidth

- Network transfer rates are a critical to distributed HPC data service performance.
- What is the best way to gather empirical network measurements?
  - MPI is a natural choice:
  - Widely available, portable, highly performant, frequently benchmarked
  - It is the gold standard for HPC network performance.
- Let’s look at an **osu_bw** benchmark example from the OSU Benchmark Suite (http://mvapich.cse.ohio-state.edu/benchmarks/).

CASE STUDY 1: THE ISSUE

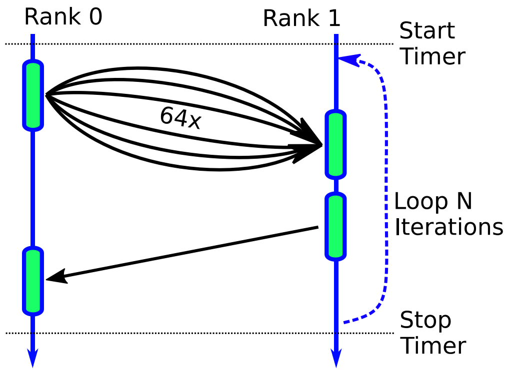
Does the benchmark access memory the way a data service would?

Example network transfer use case in HPC data services (e.g., developer expectation)



Incrementally iterate over a large data set with continuous concurrent operations.

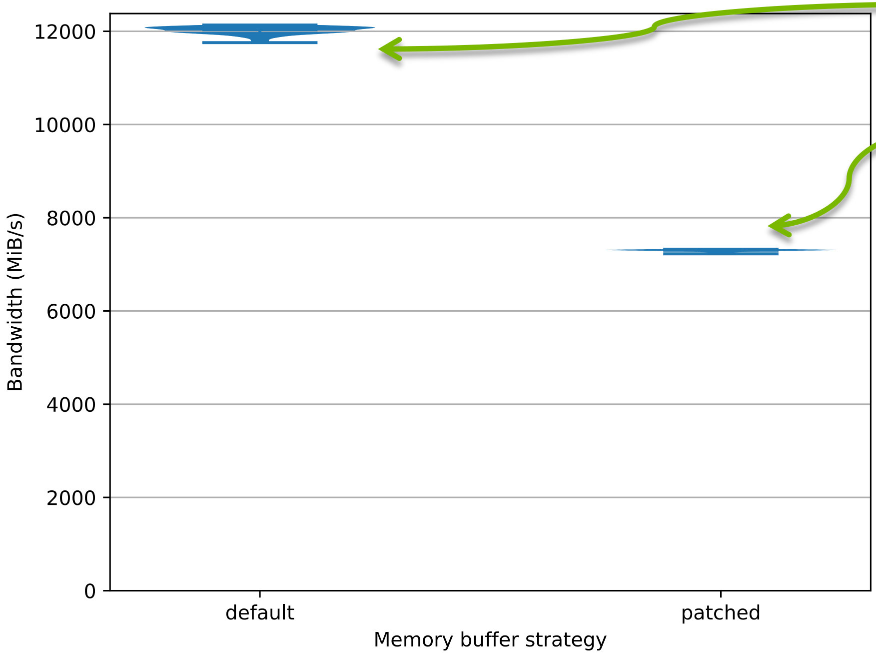
Pattern measured by the osu_bw benchmark (e.g., benchmark author intent)



All transfers (even concurrent ones) transmit or receive from a single memory buffer, and concurrency is achieved in discrete bursts.

CASE STUDY 1: THE IMPACT

Does this memory access pattern discrepancy affect performance?



- The stock osu_bw benchmark achieves 11.7 GiB/s between nodes.
- The modified version iterates over a 1 GiB buffer on each process while issuing equivalent operations.
- **40% performance penalty**
- Implications: understand if the benchmark and the data service generate comparable workloads.

CASE STUDY 2: BACKGROUND

Network Latency

- Network latency is a key constraint on metadata performance.
- MPI is also the gold standard in network latency, but is it doing what we want?
  - Most MPI implementations busy poll *even in blocking wait operations*.
  - Can transient or co-located data services steal resources like this?
- Let's look at an **fi_msg_pingpong** benchmark example from the libfabric fabtests (https://github.com/ofiwg/libfabric/tree/master/fabtests/).
  - Libfabric offers a low-level API with more control over completion methods than MPI.

CASE STUDY 2: THE ISSUE

How do potential completion methods differ?

```
# check completion queue (blocking)
fi_cq_sread(…)
# repeat until done
```

Fabtest default completion method

- Loop checking for completion
- Consumes a host CPU core
- Minimizes notification latency

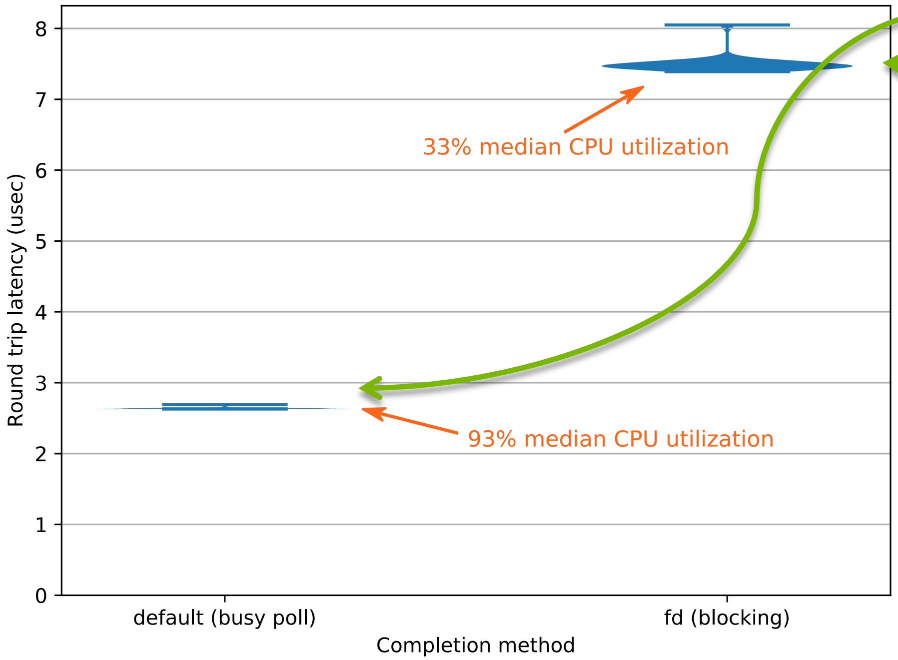
Fabtest “fd” completion method

- Poll() call will suspend process until network event is available
- Simplifies resource multiplexing
- Introduces context switch and interrupt overhead

```
# is it safe to block on this queue?
fi_trywait(…)
# allow OS to suspend process
poll(…, -1)
# check completion queue (nonblocking)
fi_cq_read(…)
# repeat until done
```

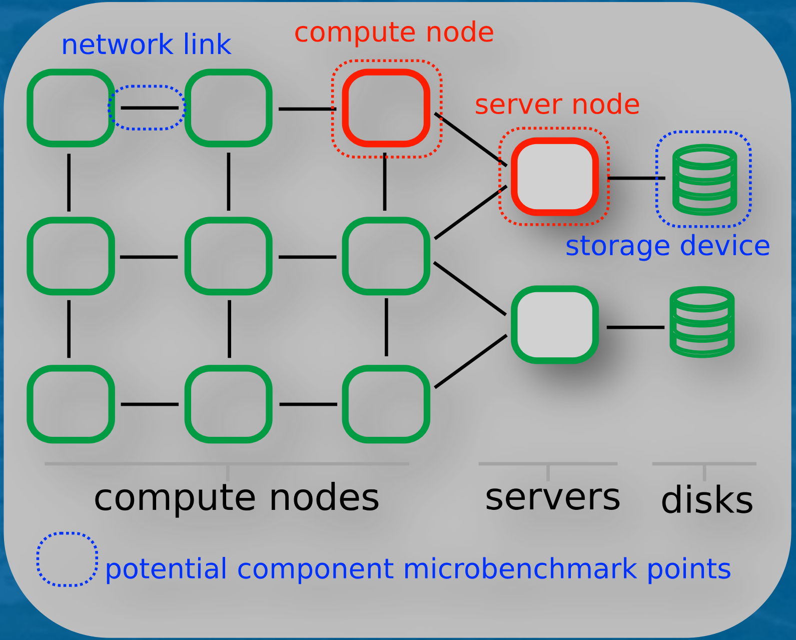
CASE STUDY 2: THE IMPACT

How does the completion method affect performance?



- The default method achieves < 3 microsecond round trip latency.
- The fd completion method suspends process until events are available.
- This incurs a **3x latency penalty.**
- This also lowers CPU consumption (would approach zero when idle).
- Implication: Consider if the benchmark is subject to the same resource constraints as the HPC data service.

CPU
CASE STUDIES



CASE STUDY 3: BACKGROUND

Host CPU utilization

- The host CPU constrains performance if it coordinates devices or relays data through main memory.
- This case study is a little different than the others:
  - Observe the indirect impact of host CPU utilization on throughput.
  - Is the data service provisioned with sufficient CPU resources?
- Let's look at a **fi_msg_bw** benchmark example from the libfabric fabtests (https://github.com/ofiwg/libfabric/tree/master/fabtests/)
- In conjunction with **aprun**, the ALPS job launcher

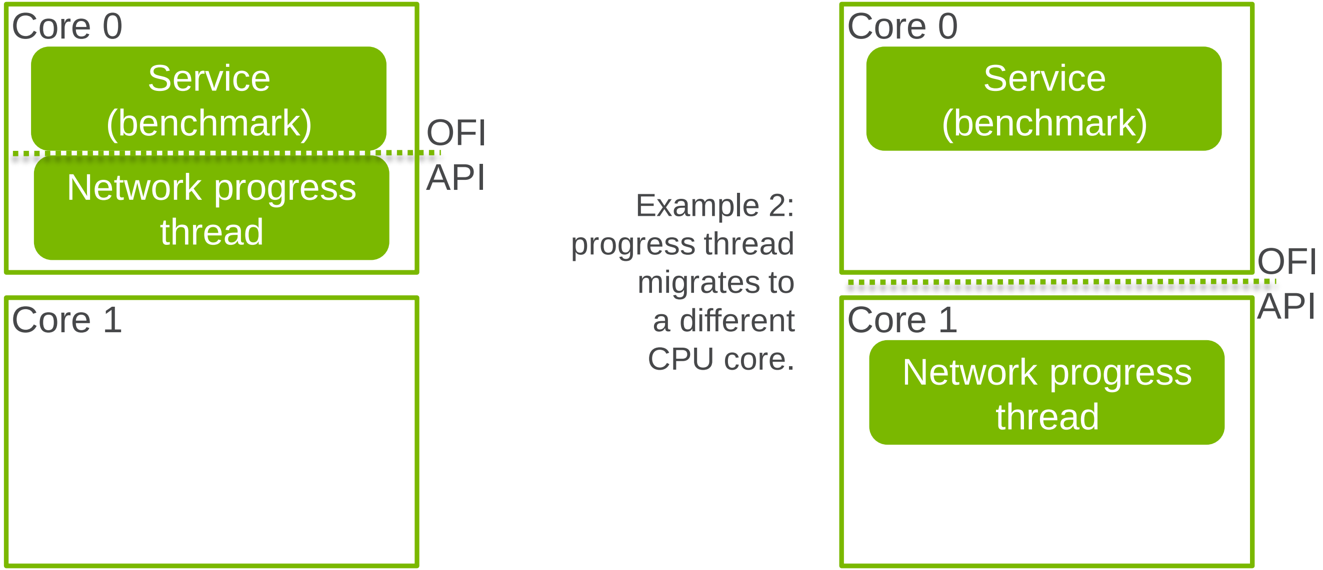
CASE STUDY 3: THE ISSUE

Do service CPU requirements align with the provisioning policy?

Consider that a transport library may spawn an implicit thread for network progress:

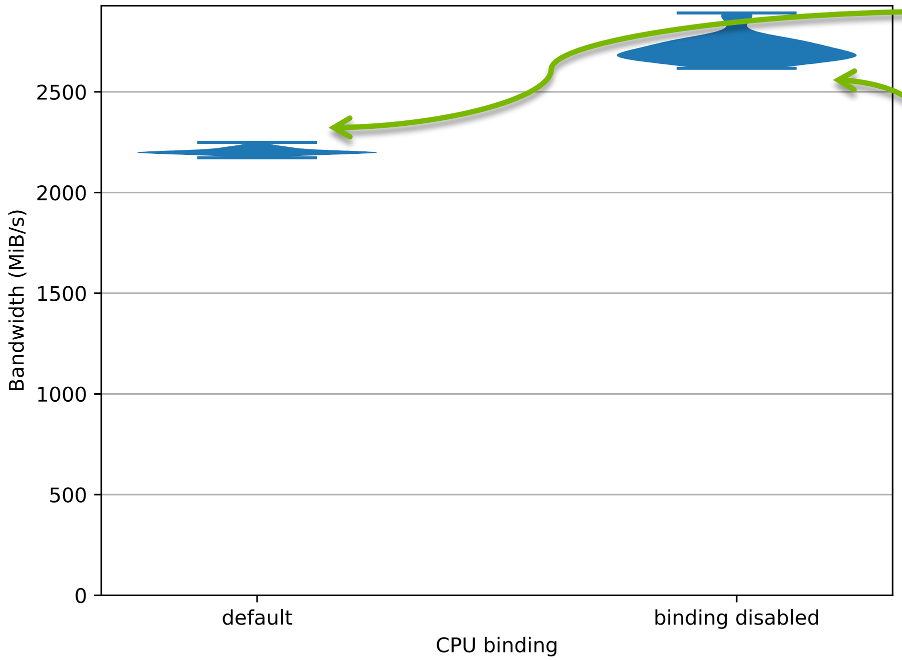
Example 1: progress thread is bound to the same CPU core as the service.

Example 2: progress thread migrates to a different CPU core.



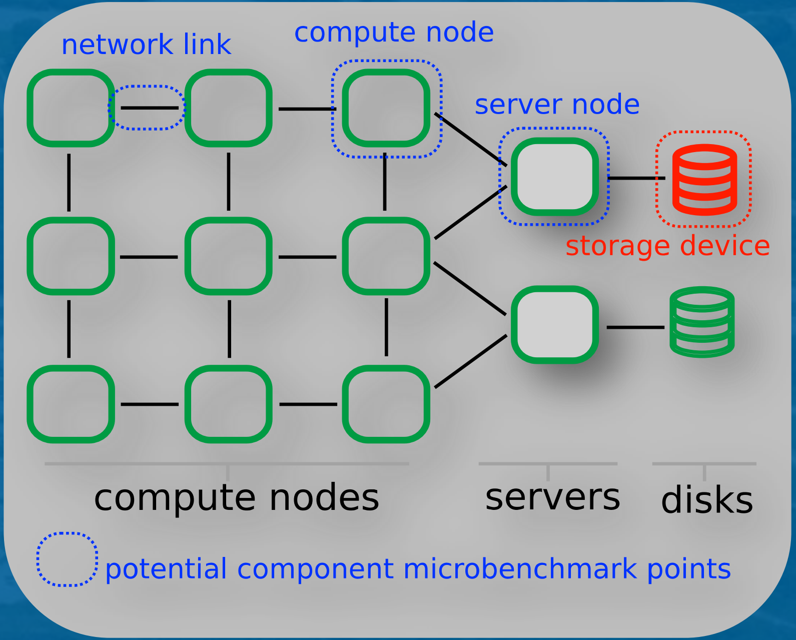
CASE STUDY 3: THE IMPACT

How does core binding affect performance?



- Default configuration achieves 2.15 GiB/s.
- The only difference in the second configuration is that launcher arguments are used to disable default core binding policy.
- **22.5% performance gain**
- Implication: Is the benchmark using the same allocation policy that your data service would?

STORAGE CASE STUDIES



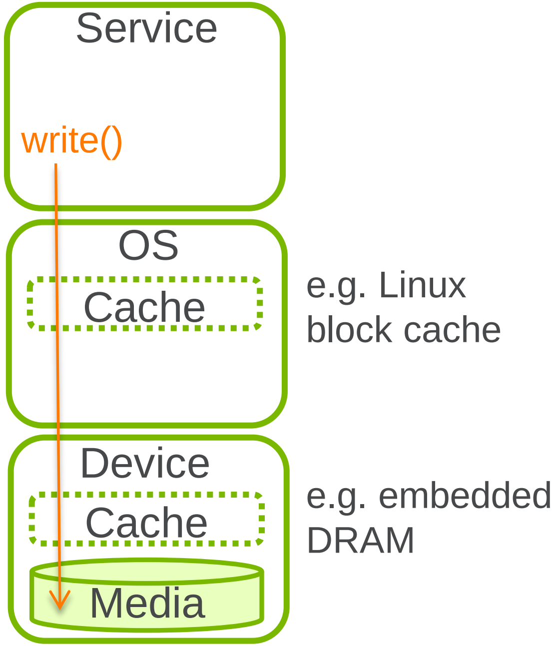
CASE STUDY 4: BACKGROUND

Storage device caching modes

- Cache behavior constrains performance in many use cases.
  - A wide array of device and OS parameters can influence cache behavior.
  - Some devices are actually *slowed down* by additional caching.
- We investigate the impact of the direct I/O parameter in this case study:
  - Direct I/O is a Linux-specific (and not uniformly supported) file I/O mode.
  - Does direct I/O improve or hinder performance for a given device?
- Let's look at an **fio** benchmark (https://github.com/axboe/fio/) example.

CASE STUDY 4: THE ISSUE

Interaction between cache layers in the write path

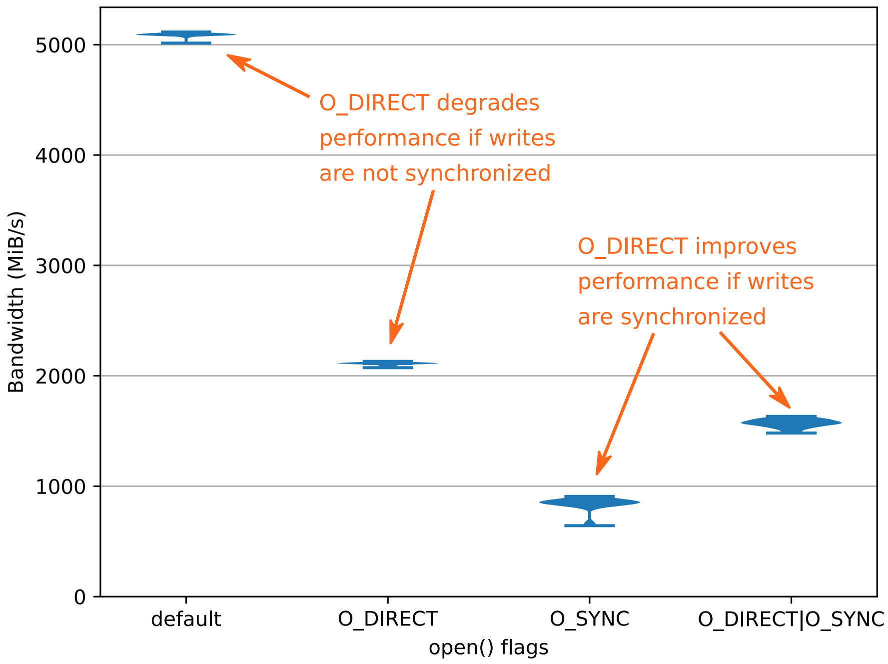


Consider two open() flags that alter cache behavior and durability:

- O_DIRECT:
  - Completely bypasses the OS cache
  - No impact on the device cache (i.e., no guarantee of durability to media until sync())
- O_SYNC:
  - Doesn’t bypass any caches, but causes writes to flush immediately (i.e., write-through mode)
  - Impacts both OS and device cache

CASE STUDY 4: THE IMPACT

Does direct I/O help or hurt performance?



- We looked at four combinations.
- **The answer is inverted** depending on whether O_SYNC is used or not.
- The write() timing in the first case is especially fast because no data actually transits to the storage device.
- Implications: the rationale for benchmark configuration (and subsequent conclusions) must be clear.

CASE STUDY 5: BACKGROUND

Translating device performance to services

- Case study 4 established expectations for throughput in a common hypothetical HPC data service scenario:
  - “How fast can a server that write to a durable local log for fault tolerance?”
- We used **fio** again to evaluate this scenario, but this time:
  - We only used the O_DIRECT|O_SYNC flags (chosen based on previous experiment)
  - We wrote to a local shared file, as a server daemon would.
- Are there any other parameters that will affect performance?

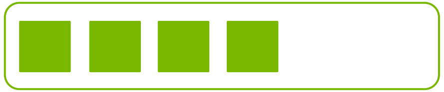
Data block

Logical file

CASE STUDY 5: THE ISSUE

A tale of three file allocation methods

Preallocate



- Use fallocate() or similar to set up file before writing
- Decouples pure write() cost from layout and allocation
- Default in fio benchmark

Append at EOF

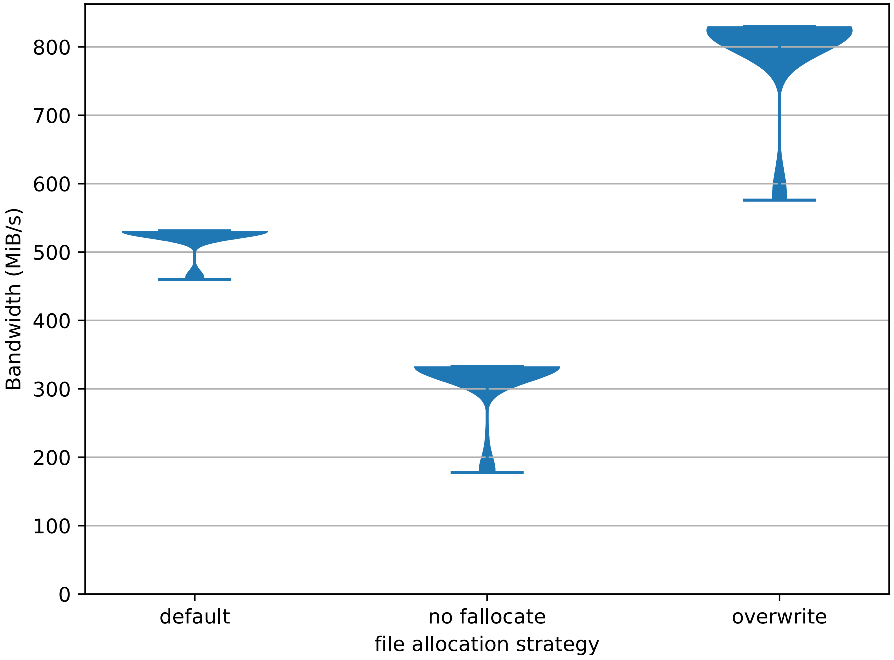
- Write data at end of file
- File system must determine block layout and allocate space in the write() path
- Natural approach for a data service or application: just open a file and write it

Wrap around at EOF

- Wrap around and overwrite original blocks at EOF
- After EOF, the file is already allocated and the layout is cached.
- Less common real-world use case, but a plausible benchmark misconfiguration

CASE STUDY 5: THE IMPACT

How do those file allocation strategy affect performance?



- Shared file, concurrent write, with O_SYNC|O_DIRECT.
- **Each file allocation method leads to markedly different write performance**.
- It was not immediately clear to the authors that fio used fallocate() by default.
- Implications: determine (and report) default benchmark parameters, and consider if the benchmark includes all relevant costs.

DISCUSSION

IMPLICATIONS FOR ROOFLINE MODELING

- Recall our original goal: **construct realistic rooflines for HPC data services to assist performance interpretation.**
- The requisite data can be surprisingly difficult to extract from a benchmark:
  - What does “realistic” mean?
  - What is the benchmark really measuring?
  - How and why were it’s configuration parameters selected?
- Resolving discrepancies may require extensive profiling effort and deep system architecture expertise.
- Alternative outcome: unrealistic expectations, misdiagnosed problems, lost development time.

HELP WANTED

How can we, as a community, improve the state of the practice?

- This study does not offer a panacea; it’s goal is to highlight examples and draw attention to the problem.
- Anecdotally, benchmarks are often designed to extract maximal hardware performance, even if the pattern that produces it is not feasible in production.
- What does this mean for our community?
  - Is there value in standardizing benchmark motifs tailored to HPC data service modalities?
  - What is the best way to report and document benchmark parameters (especially default parameters)?
  - How can we be more rigorous in reporting not only experimental results, but the rationale for experimental design?

THANK YOU!

THIS WORK WAS SUPPORTED BY THE U.S. DEPARTMENT OF ENERGY, OFFICE OF SCIENCE, ADVANCED SCIENTIFIC COMPUTING RESEARCH, UNDER CONTRACT DE-AC02-06CH11357.