DAOS PERFORMANCE AND I/O CHALLENGES

Adrian Jackson

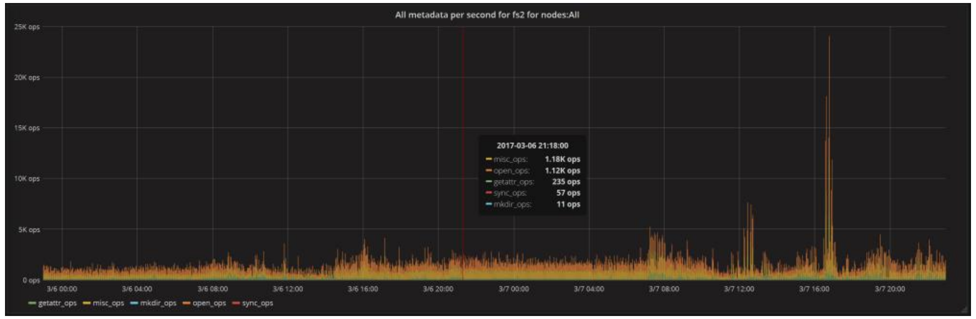
a.jackson@epcc.ed.ac.uk

Nicolau Manubens

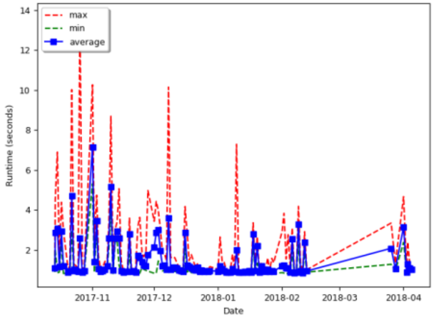
ECMWF

I/O in HPC

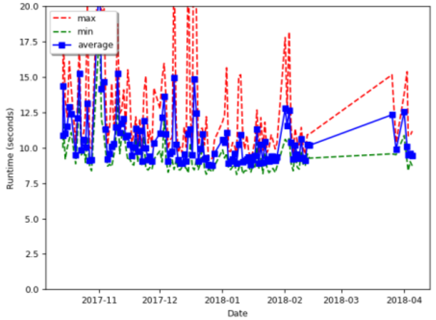
- MPI I/O performance and functionality
  - Long recognition that for a subset of applications I/O is a non-trivial overhead



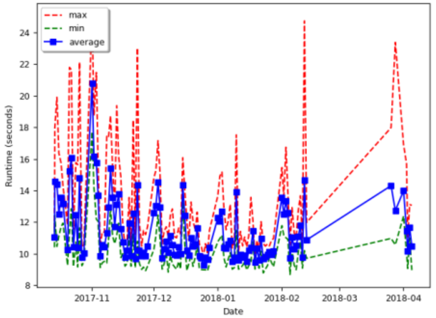
- I/O formats and functionality
  - Domain users also desire more than just bits per second functionality



MPI-I/O



HDF5



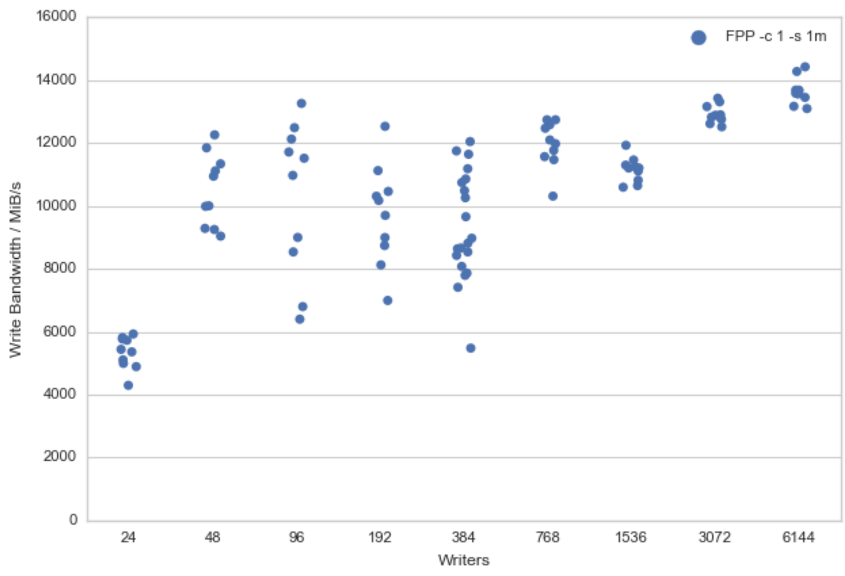
NetCDF

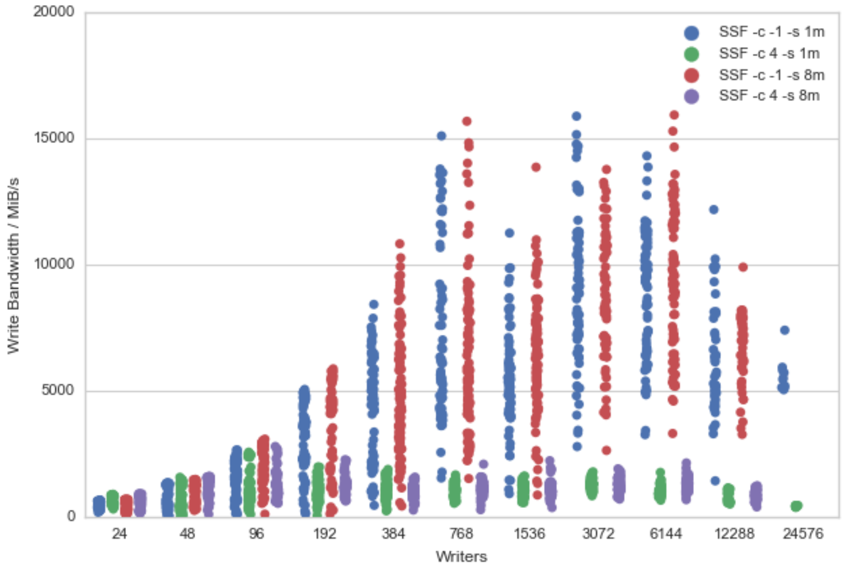
THE UNIVERSITY of EDINBURGH

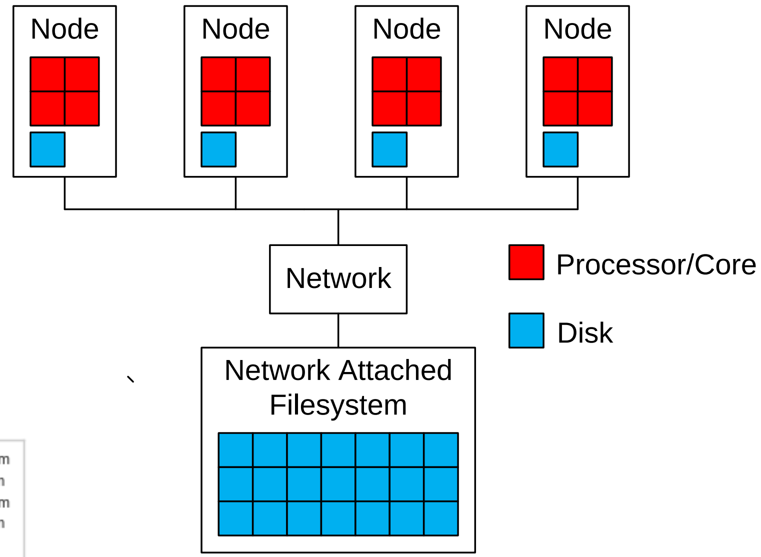
epcc

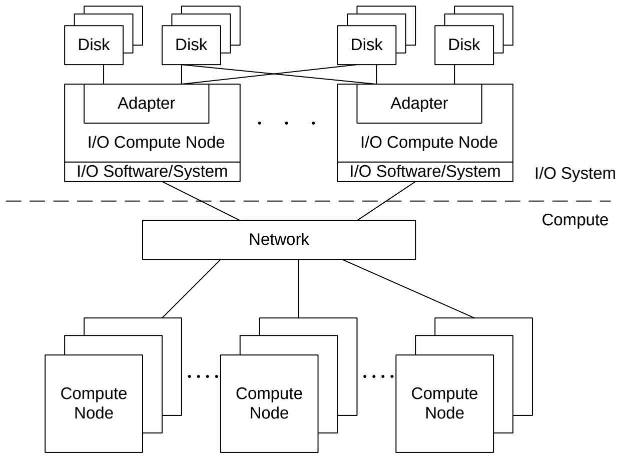
I/O at EPCC

- Complexity of the hardware and software layers
  - POSIX issues
  - Shared resources
  - Multiple requirements





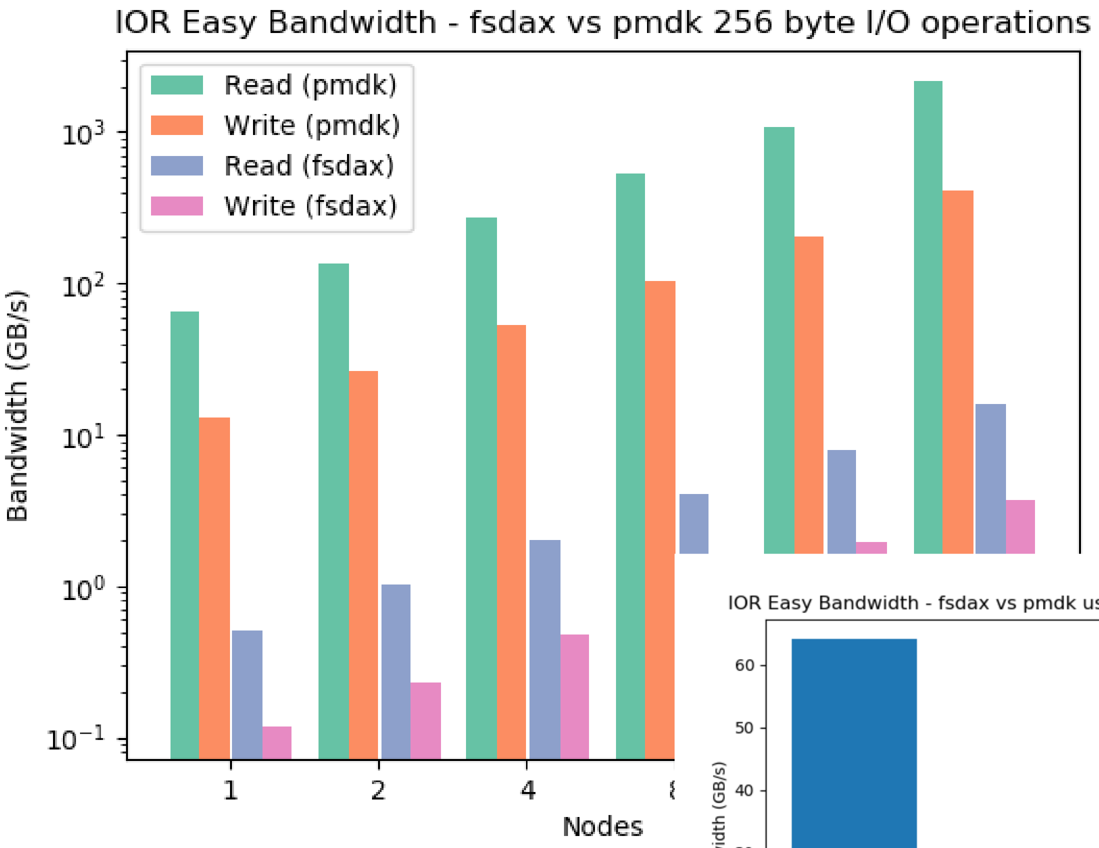


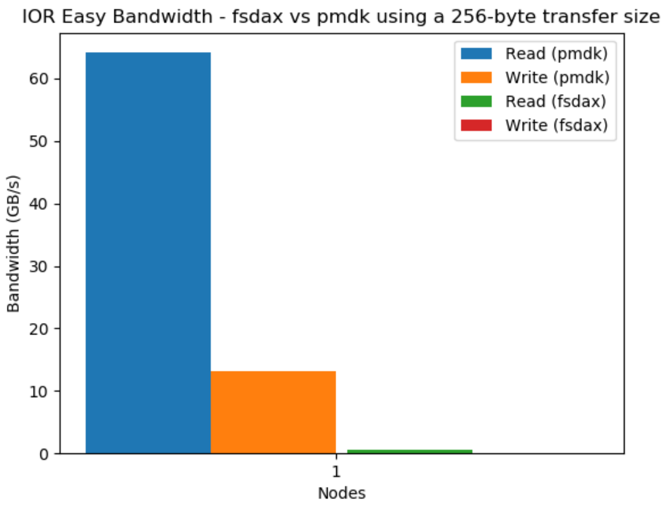


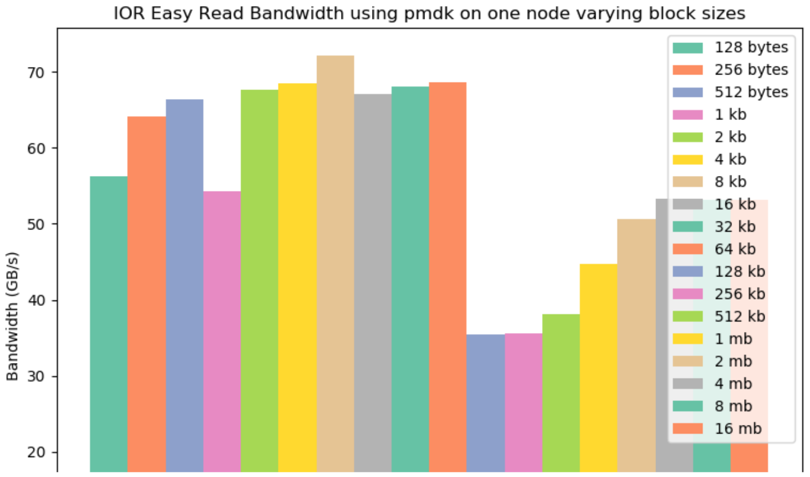
Levels of concern

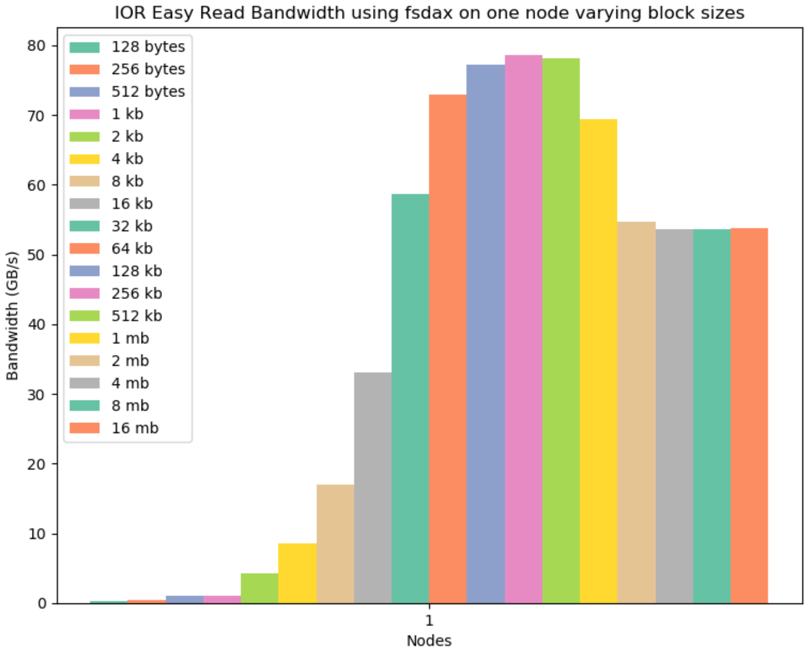
- User implementation
- API/Client interface
- Storage system software
- Hardware used

Small I/O performance

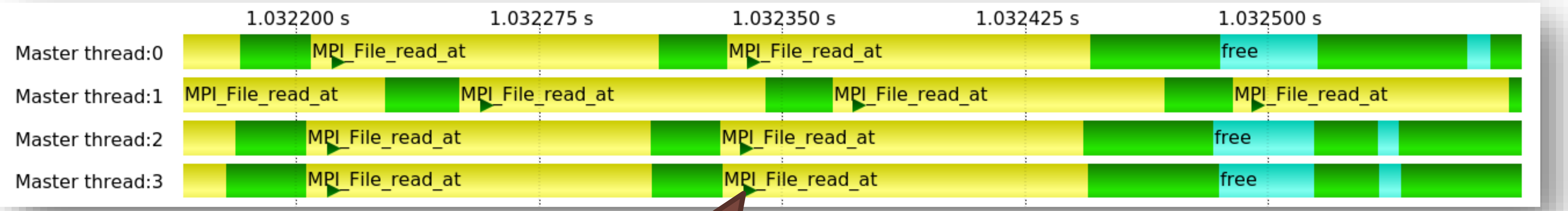




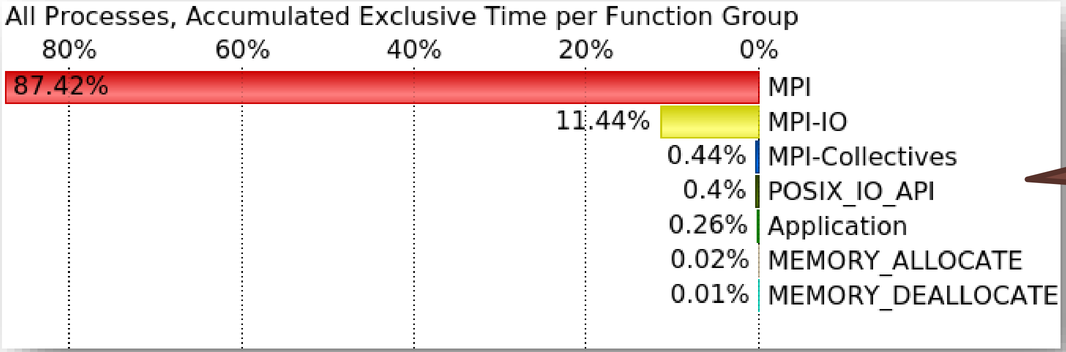




I/O application patterns

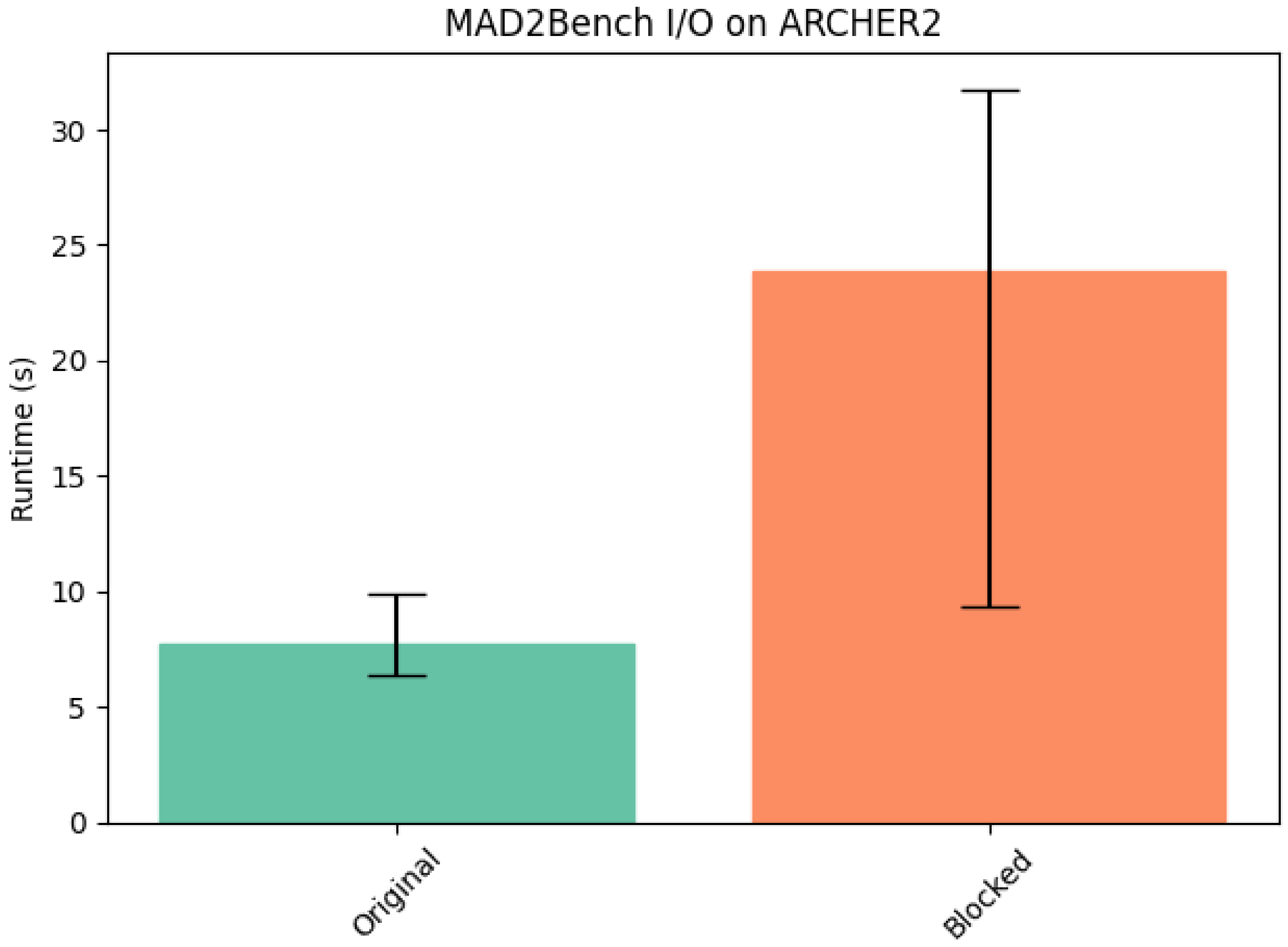


Individual I/O Operation

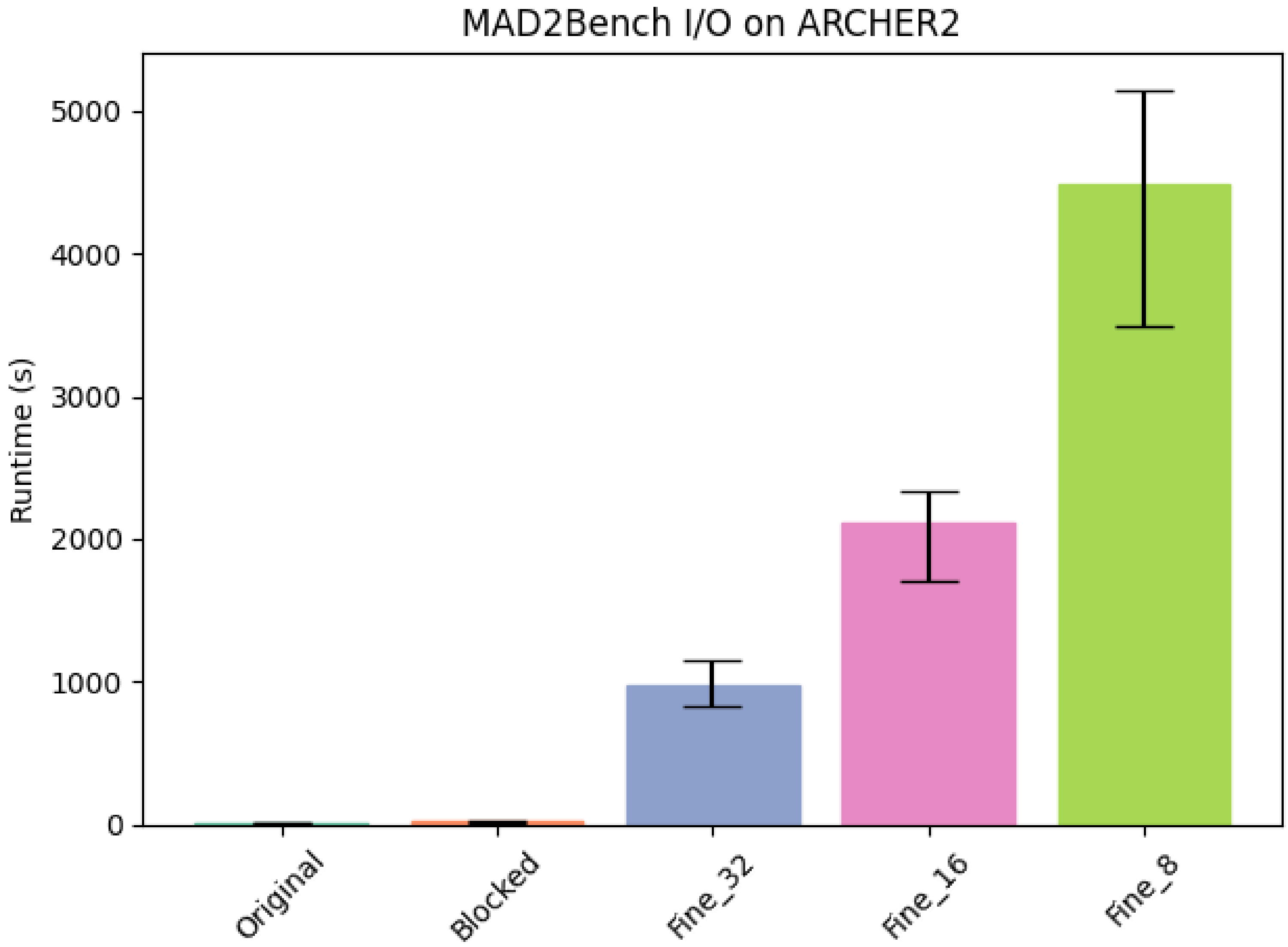


I/O Runtime Contribution

MAD2Bench

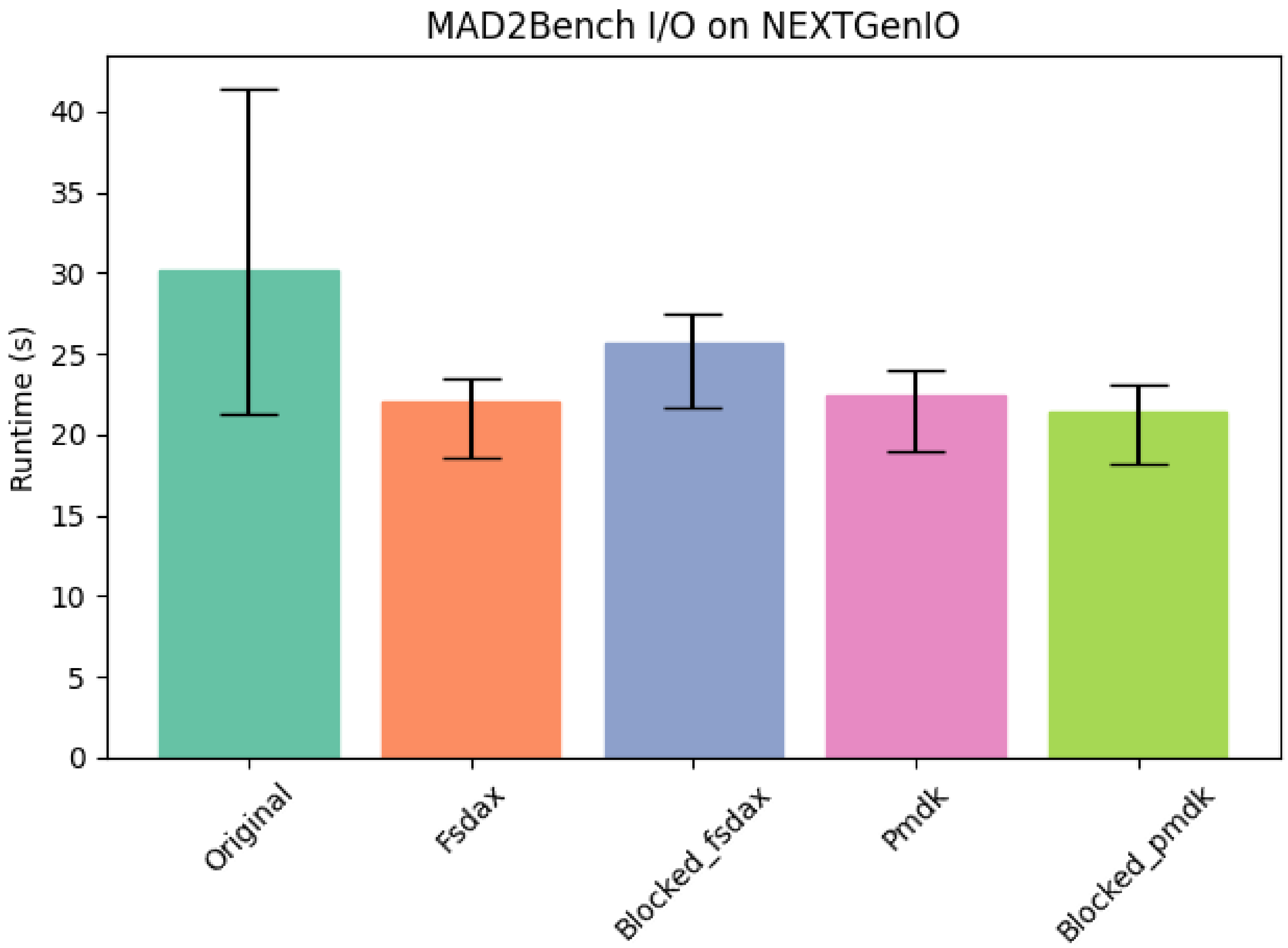

MAD2Bench I/O on ARCHER2
Runtime (s)
0
5
10
15
20
25
30
Original
Blocked

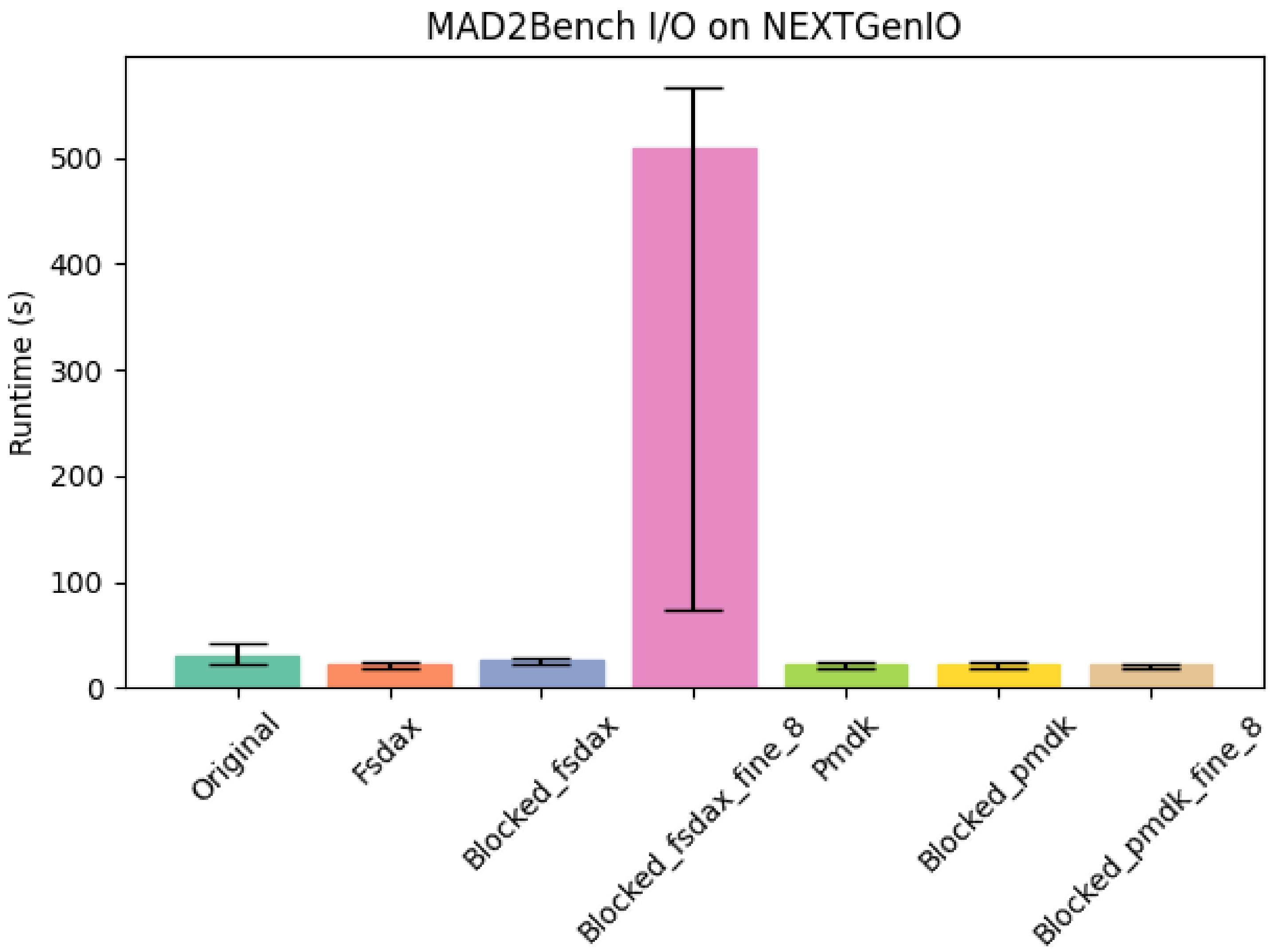
MAD2Bench



MAD2Bench I/O on ARCHER2
Runtime (s)
5000
4000
3000
2000
1000
0
Original
Blocked
Fine_32
Fine_16
Fine_8

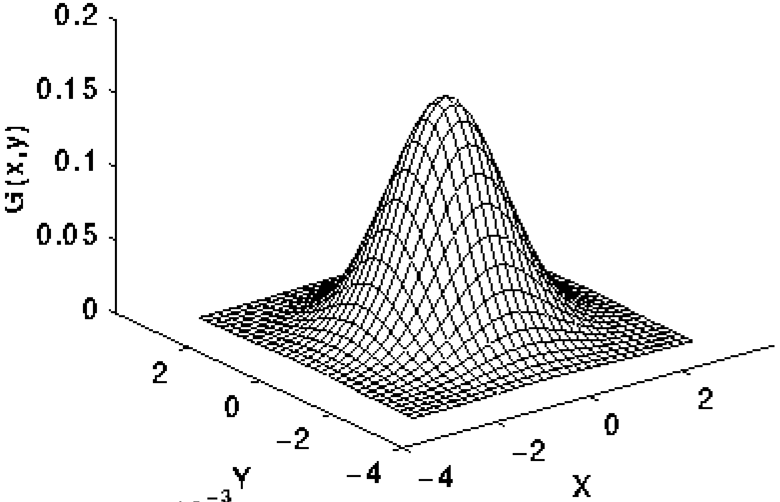
MAD2Bench

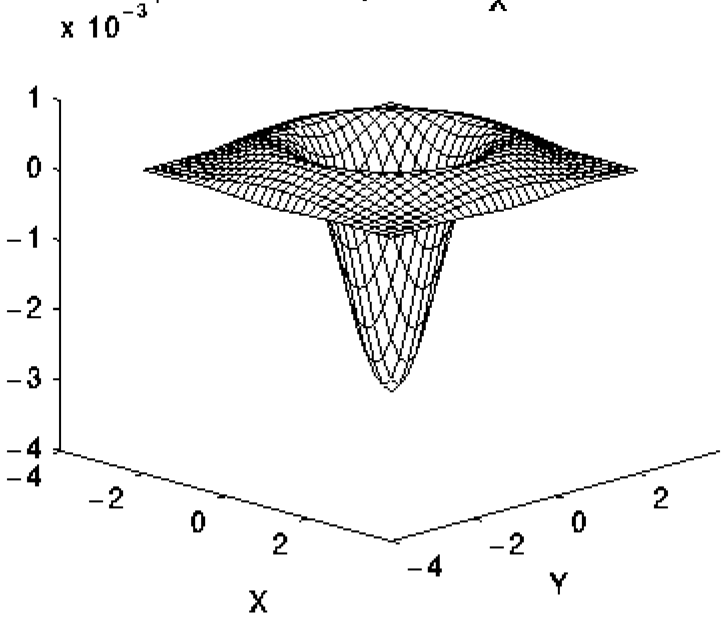

MAD2Bench I/O on NEXTGenIO
Runtime (s)
0
5
10
15
20
25
30
35
40
Original
Fsdax
Blocked_fsdax
Pmdk
Blocked_pmdk


MAD2Bench I/O on NEXTGenIO
Runtime (s)
0
100
200
300
400
500
Original
Fsdax
Blocked_fsdax
Blocked_fsdax_fine_8
Pmdk
Blocked_pmdk
Blocked_pmdk_fine_8

Multi-level memory exploitation

- Simple image sharpening stencil
  - Each pixel replaced by a weighted average of its neighbours
  - weighted by a 2D Gaussian
  - averaged over a square region
  - we will use:
    - Gaussian width of 1.4
    - a large square region
  - then apply a Laplacian
    - this detects edges
    - a 2D second-derivative ∇^2
- Combine both operations
  - produces a single convolution filter
- 4 similar sized arrays, two that are updated and two that are source data





Multi-level memory exploitation

```
address = (int **) malloc(nx*sizeof(int *) + nx*ny*sizeof(int));
fuzzy = int2D(nx, ny, address);
```

```
pmemaddr1 = pmem_map_file(filename, array_size,PMEM_FILE_CREATE|PMEM_FILE_EXCL,
                          0666, &mapped_len1, &is_pmem)
fuzzy =  int2D(nx, ny, pmemaddr1);

int **int2D(int nx, int ny, int **idata){
  int i;
  idata[0] = (int *) (idata + nx);

  for(i=1; i < nx; i++){
      idata[i] = idata[i-1] + ny;
    }
  return idata;
}
```

- Read-only data in DRAM

```
Calculation time was 56.175083 seconds
Overall run time was 58.261385 seconds
```

- Read-only data in B-APM

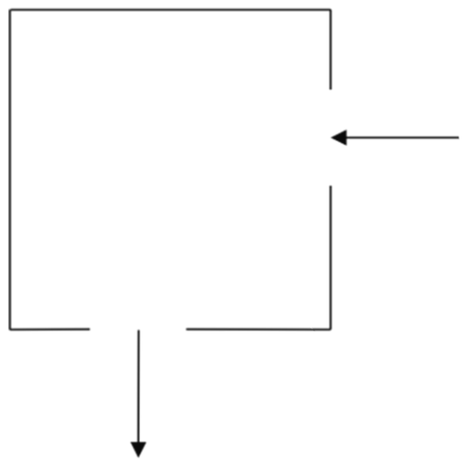
```
Calculation time was 53.992465 seconds
Overall run time was 56.385472 seconds
```

Multi-level memory exploitation

- 2D CFD Stream function kernel

$$\nabla^2\Psi = \frac{\partial^2\Psi}{\partial x^2} + \frac{\partial^2\Psi}{\partial y^2} = 0$$

$$\Psi_{i-1,j} + \Psi_{i+1,j} + \Psi_{i,j-1} + \Psi_{i,j+1} - 4\Psi_{i,j} = 0$$



- Jacobi kernel updates the grid
  - Swap update and data arrays at each iterator

```
psinew[i][j] = 0.25*(psi[i+1][j] + psi[i-1][j] +
                     psi[i][j+1] + psi[i][j-1])
```

Multi-level memory exploitation

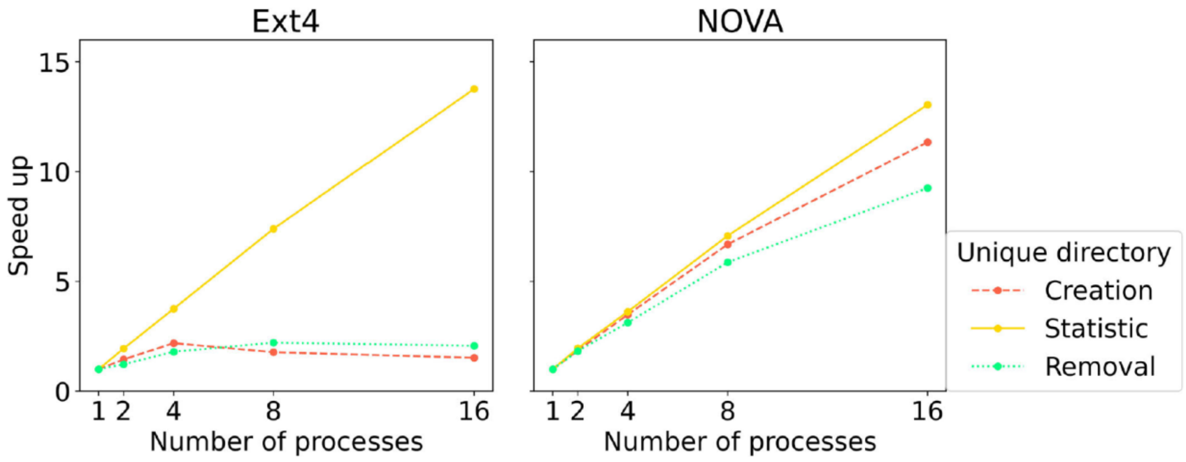
```
 totalfilename = (char *)malloc(1000*sizeof(char));

  strcpy(totalfilename,"/mnt/pmem_fsdax");
  sprintf(totalfilename+strlen(totalfilename), "%d/", socket);
  strncat(totalfilename, filename, strlen(filename));
  sprintf(totalfilename+strlen(totalfilename), "%d", rank);


  // total memory requirements including pointers

  mallocsize = nx*sizeof(void *) + nx*ny*typesize;

  if ((array2d = pmem_map_file(totalfilename, mallocsize,
                              PMEM_FILE_CREATE|PMEM_FILE_EXCL,
                              0666, mapped_len, &is_pmem)) == NULL) {
    perror("pmem_map_file");
    fprintf(stderr, "Failed to pmem_map_file for filename: %s\n",totalfilename);
    exit(-100);
  }

void swap_pointers(double*** pa, double*** pb) {
    double** temp = *pa;
    *pa = *pb;
    *pb = temp;
}
```

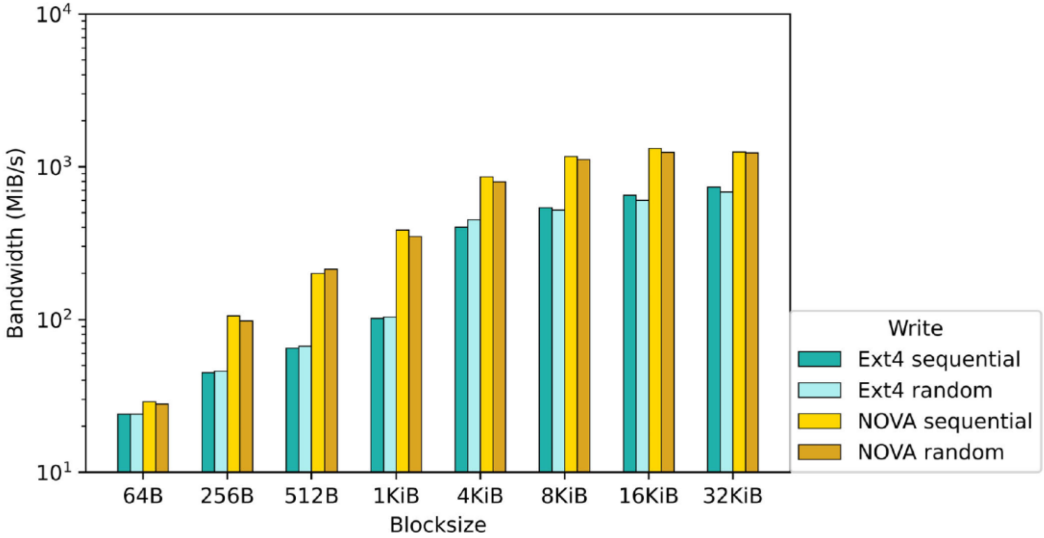
No persistence: Dram: 7.95 seconds B-APM: 9.64 seconds

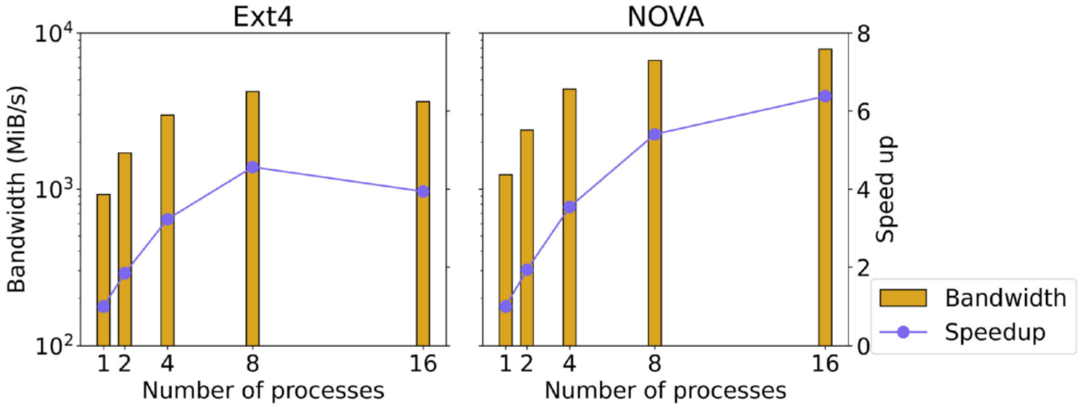
Persistence: Dram: 7.95 seconds B-APM: 10.67 seconds

Local filesystem performance

- On-node filesystems optimised for non-volatile hardware
  - Performance benefits for write operations and IOPs
  - Trade-offs in terms of capacity and other functionality
    - i.e. log append approaches, pre-allocation, wear levelling, etc...

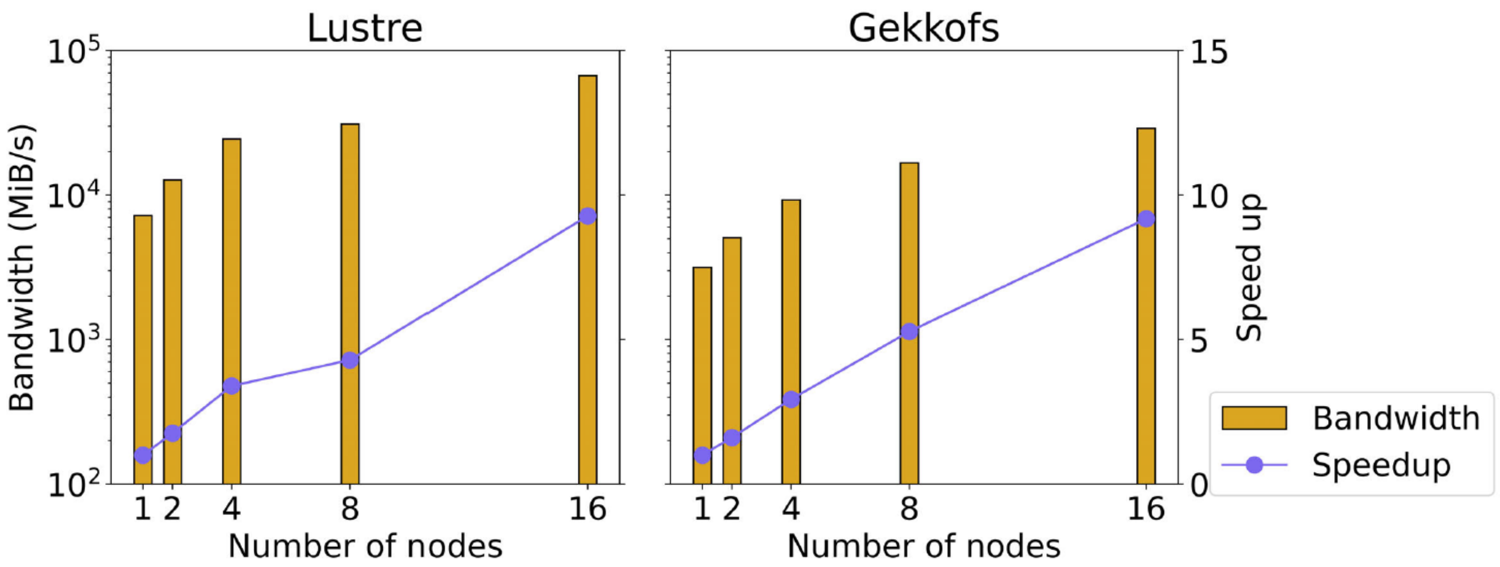






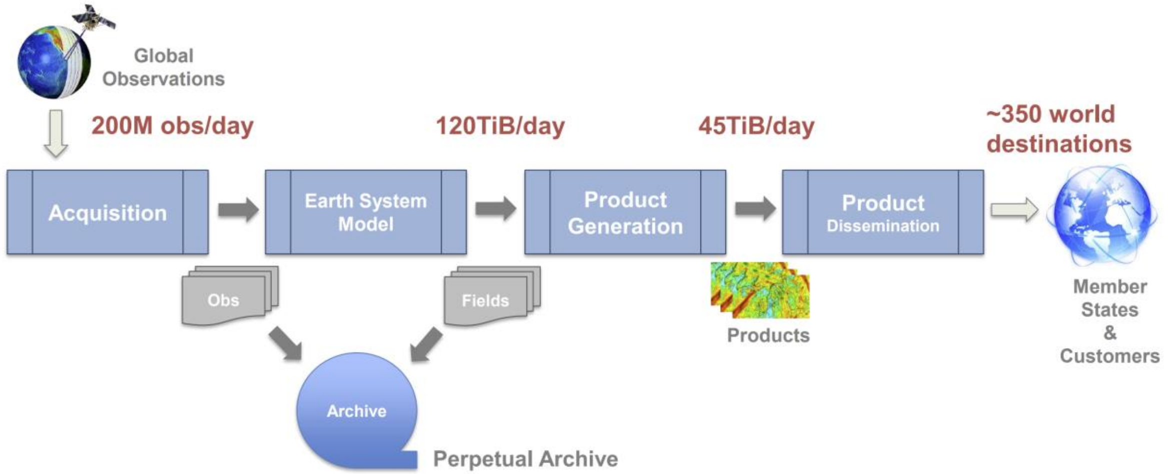
Adhoc or ephemeral filesystems

- Filesystems built using in-node storage resources on the fly
  - GekkoFS
  - CHFS
  - Simurgh
- Rocks DB for metadata
- Node-local FS or NVRAM library (i.e. PMDK) for storage
- Disaggregated resource usage



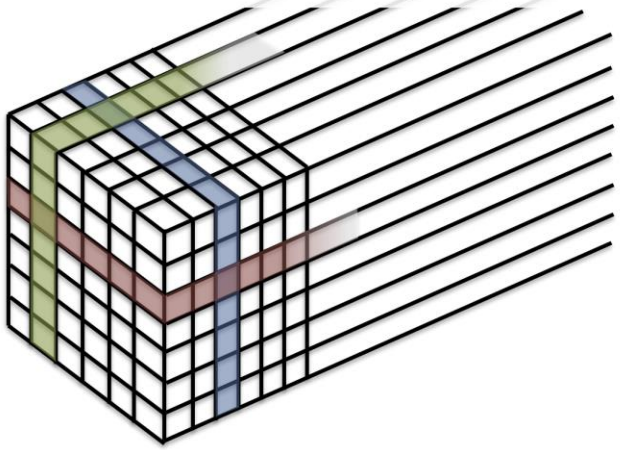
Climate/Weather domain

- Pursuing optimal I/O for applications
  - Weather forecasting workflows
- End-to-end workflow performance important
  - Simulation (data generation) only one part
- Consumption workloads different in dimension from production workloads

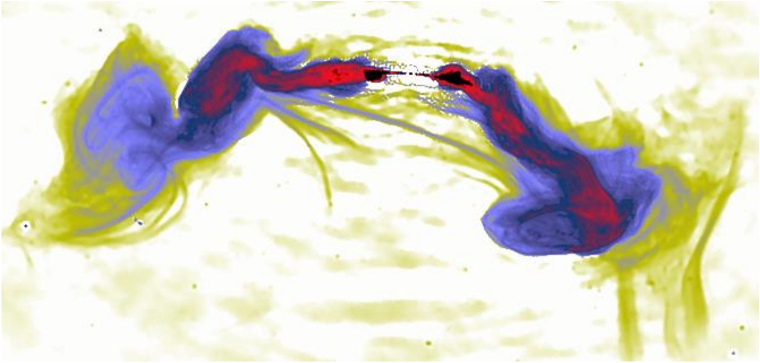


Structure free storage

Clients want to do **different** analytics across **multiple** axis



- Granular storage with rich metadata
  - Data retrieval leverages metadata
  - Build structure on the fly
- Other domains can also benefit
  - Radio astronomy
    - Data collected and stored by antenna (frequency and location) and capture time
    - Reconstruction of images done in time order
    - Evaluation of transients or other phenomenon undertaken across frequency and location

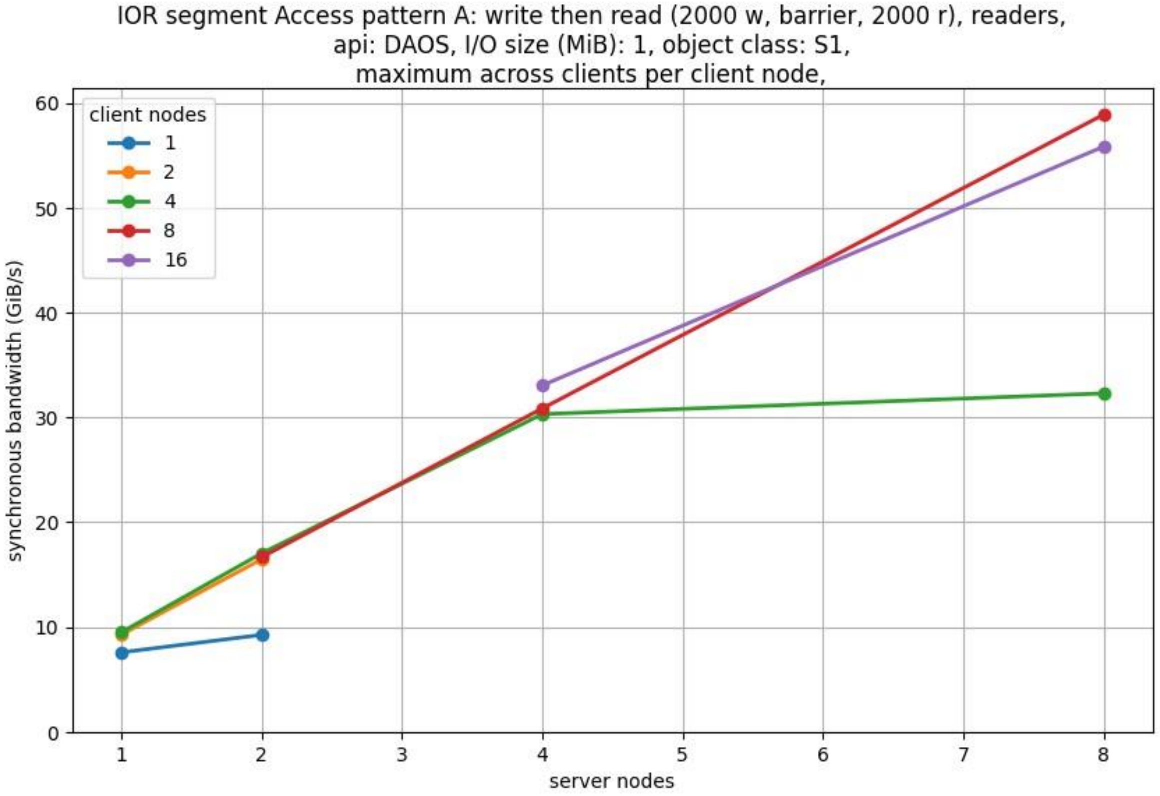


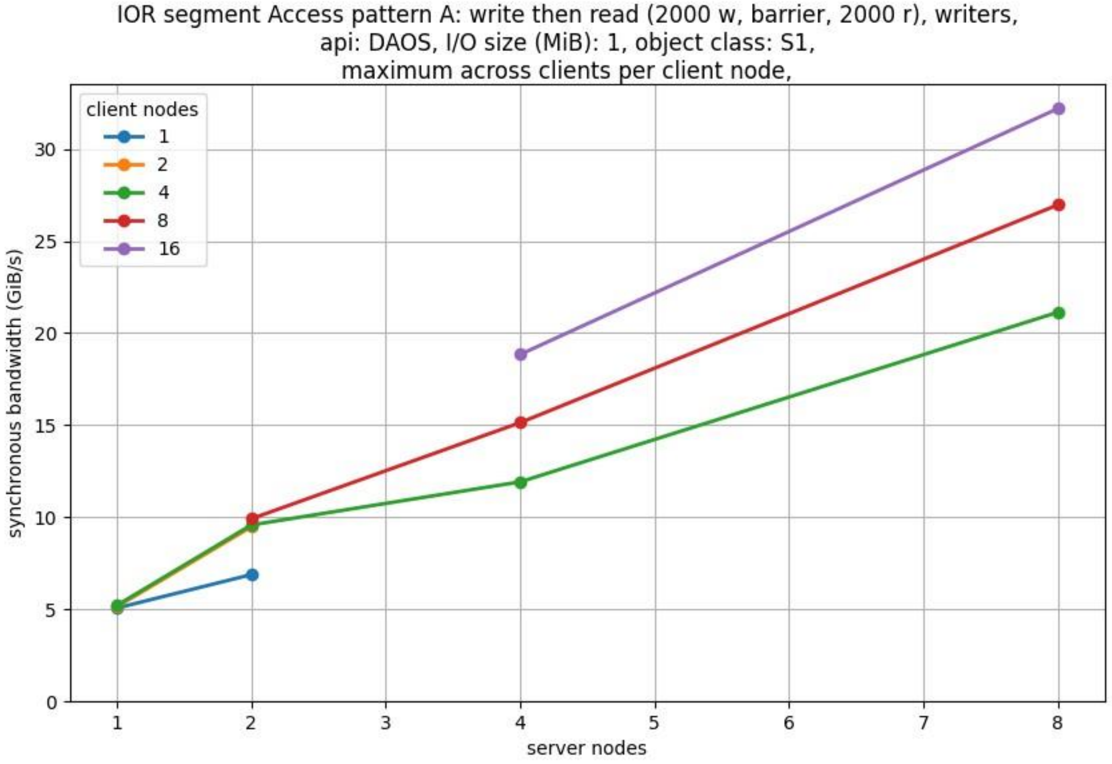
Object store approach

- Data not naturally clustered into “file” wrappers
  - Individual weather fields 1-10MB
- Object store potentially a more natural fit
  - Each weather field is an object
  - Meta data can be attached to uniquely locate them within the overall datasets
- Can object stores
  - Enable high performance I/O?
  - Enable distributed functionality?
  - Enable granular access?
  - Enable production level functionality?

DAOS

- Good bulk I/O performance

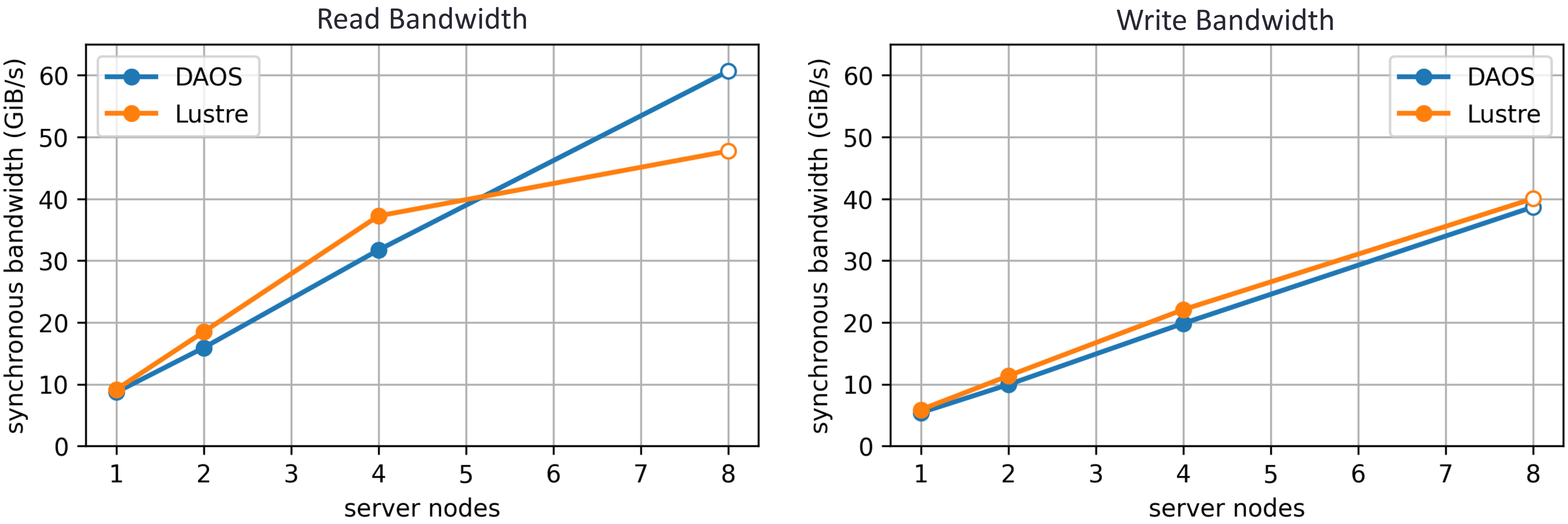




Performance Comparison Hardware configuration

- Setup compute nodes with Optane memory as DAOS server nodes or Lustre server nodes
  - Comparison of Lustre and DAOS on the same hardware
- DAOS server nodes
  - 2 DAOS engines per node (with workers)
  - PMDK/Ext4 filesystem storage backend
- Lustre nodes
  - 1 MDS with 2 targets
  - 2 OSTs per server node
  - Ext4 local storage backend

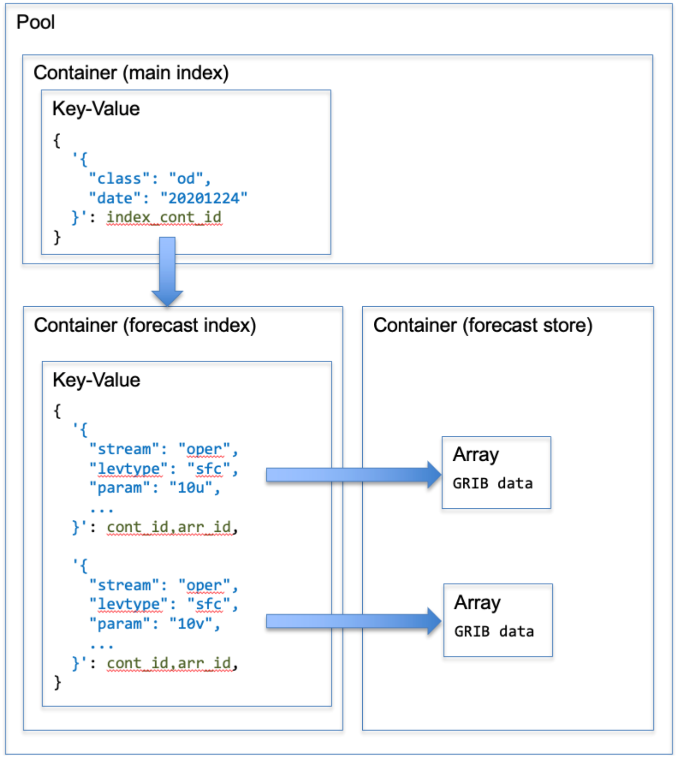
IOR bulk I/O performance comparison



- IOR (easy) benchmark: Segments mode
  - Segments: 100MB (size: 1MB Segment count: 100)
  - POSIX API for Lustre, DAOS API for DAOS

Application like benchmark: Field I/O

- DAOS Field I/O benchmark implements domain-specific object store
  - Indexing with containers and arrays for data storage
- Lustre (POSIX) port of application – object interface
  - Pools are a directory
    - Containers are sub directories within a pool
      - Key-Value objects are sub directories within a container
      - Key is index file
      - Array data separate files
- Two benchmark approaches
  - Pattern A: Separate I/O phases (write then read)
  - Pattern B: Mixed I/O phases (write and read at the same time)

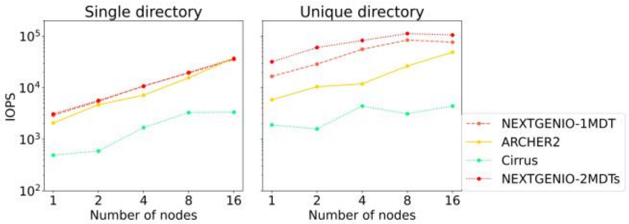


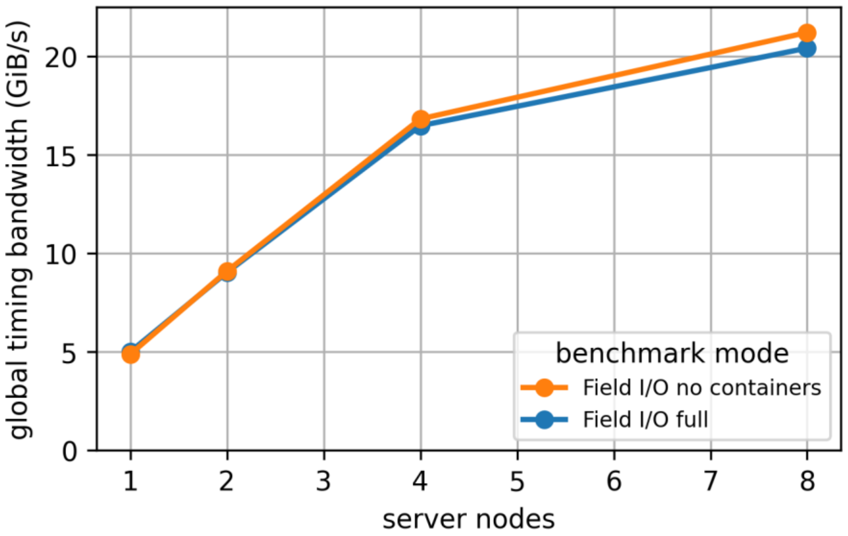
Pattern A: 1MB

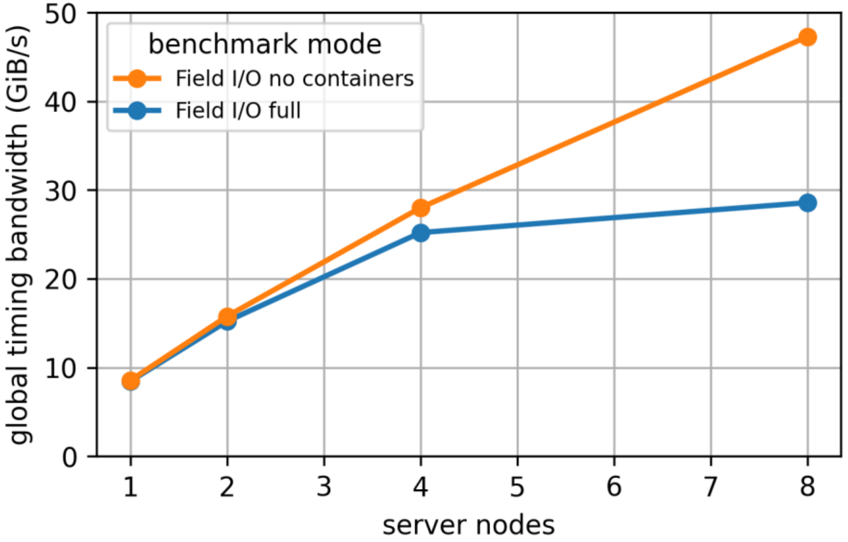
Lustre

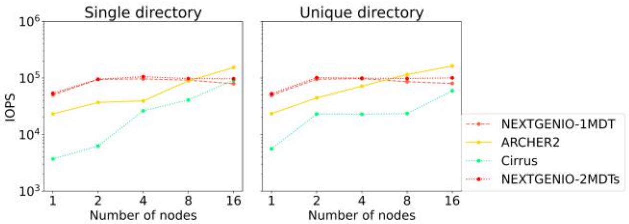
DAOS

Read

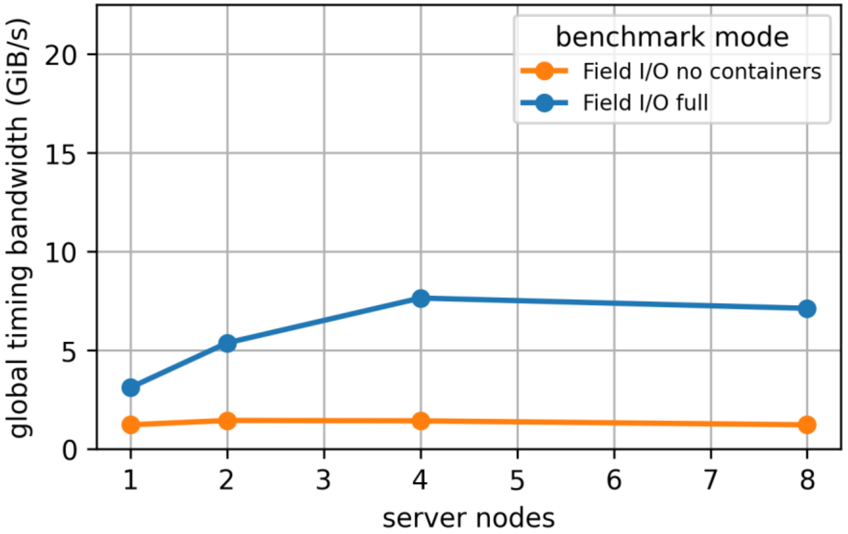


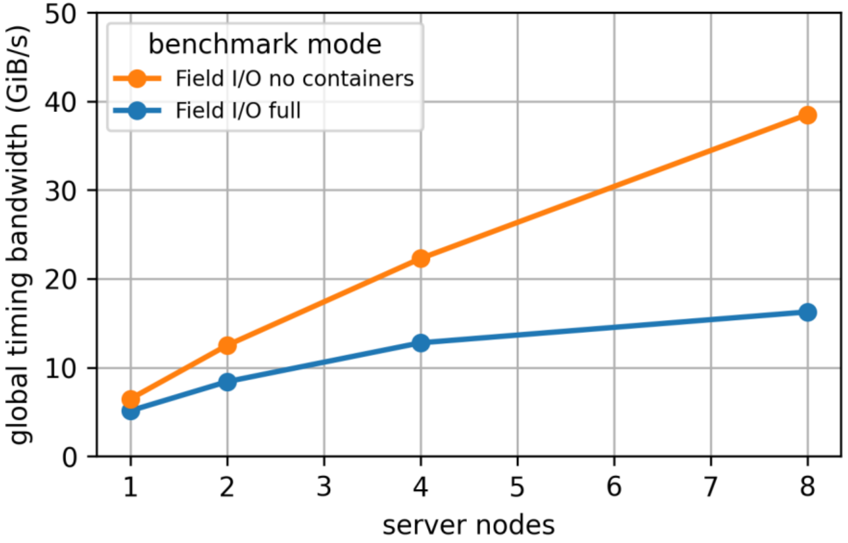




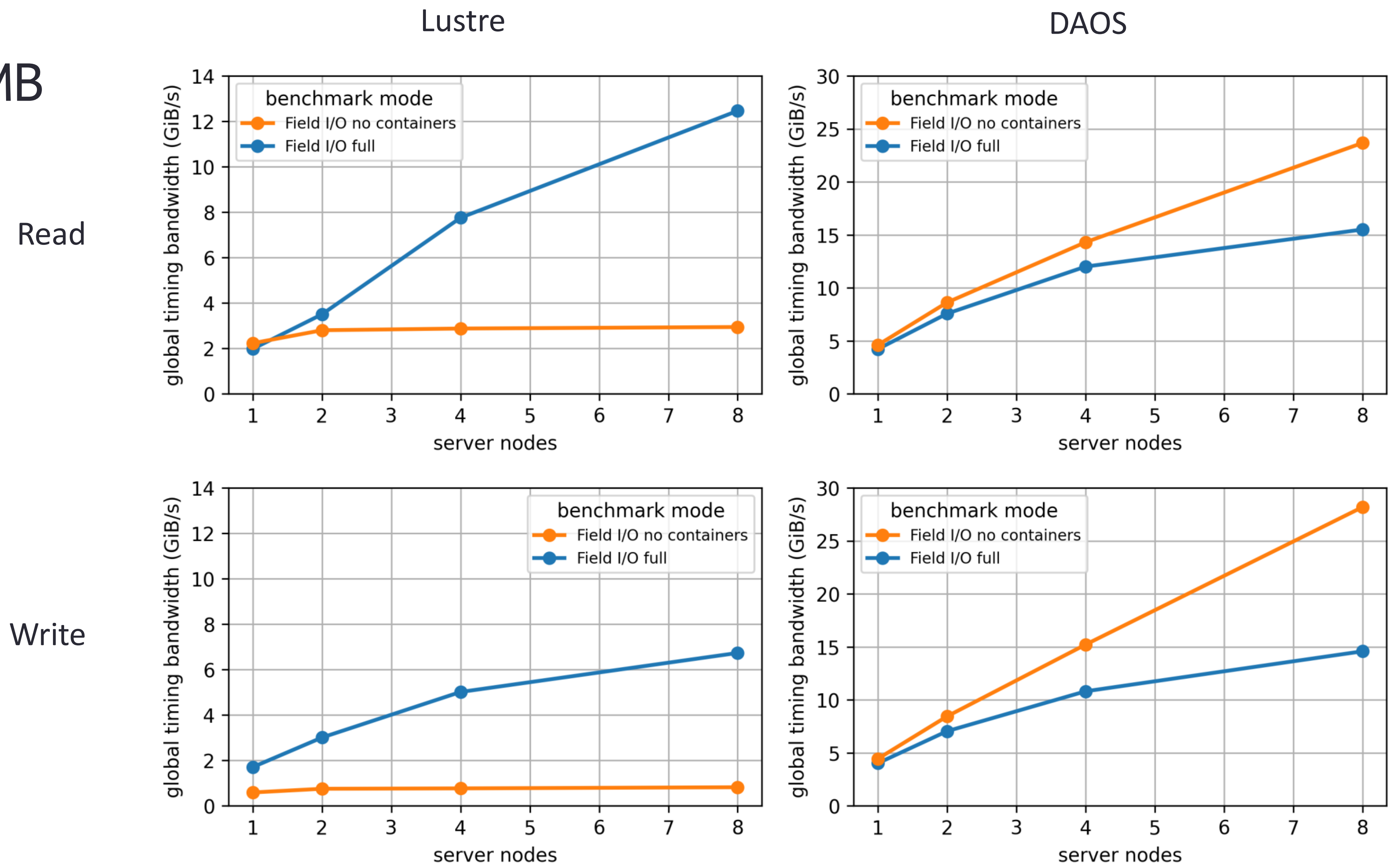


Write



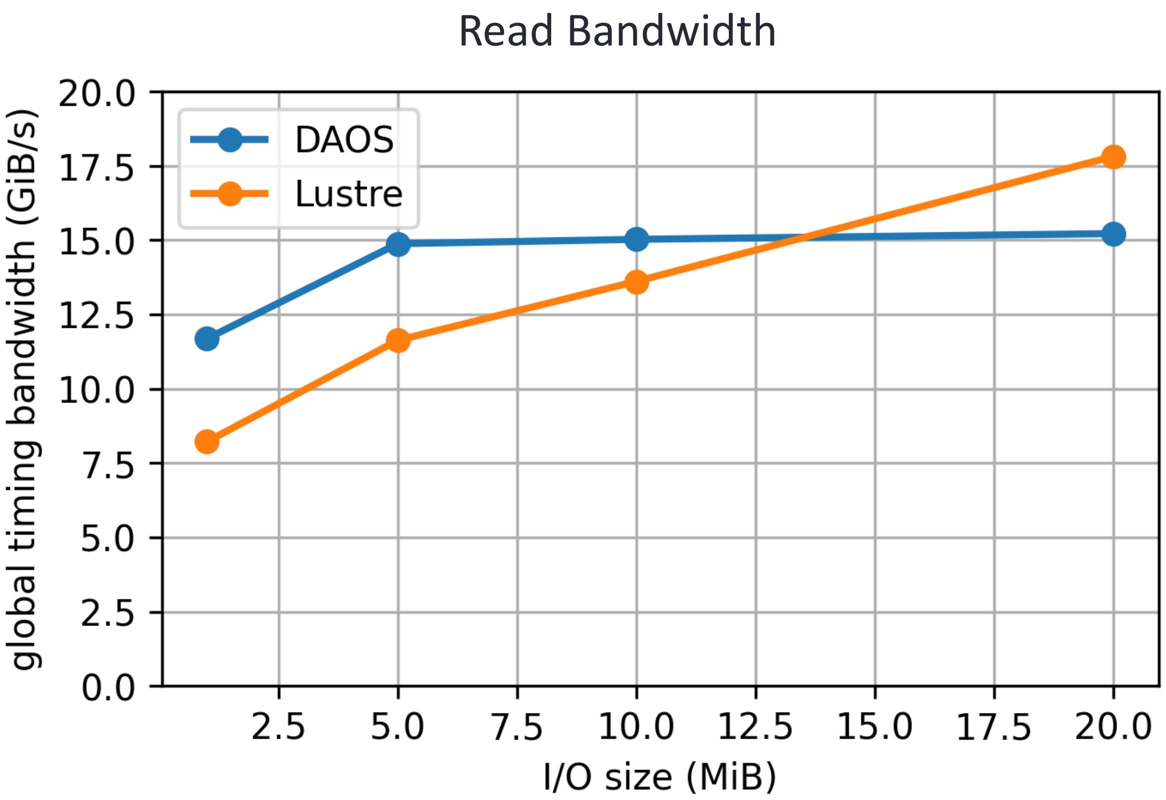


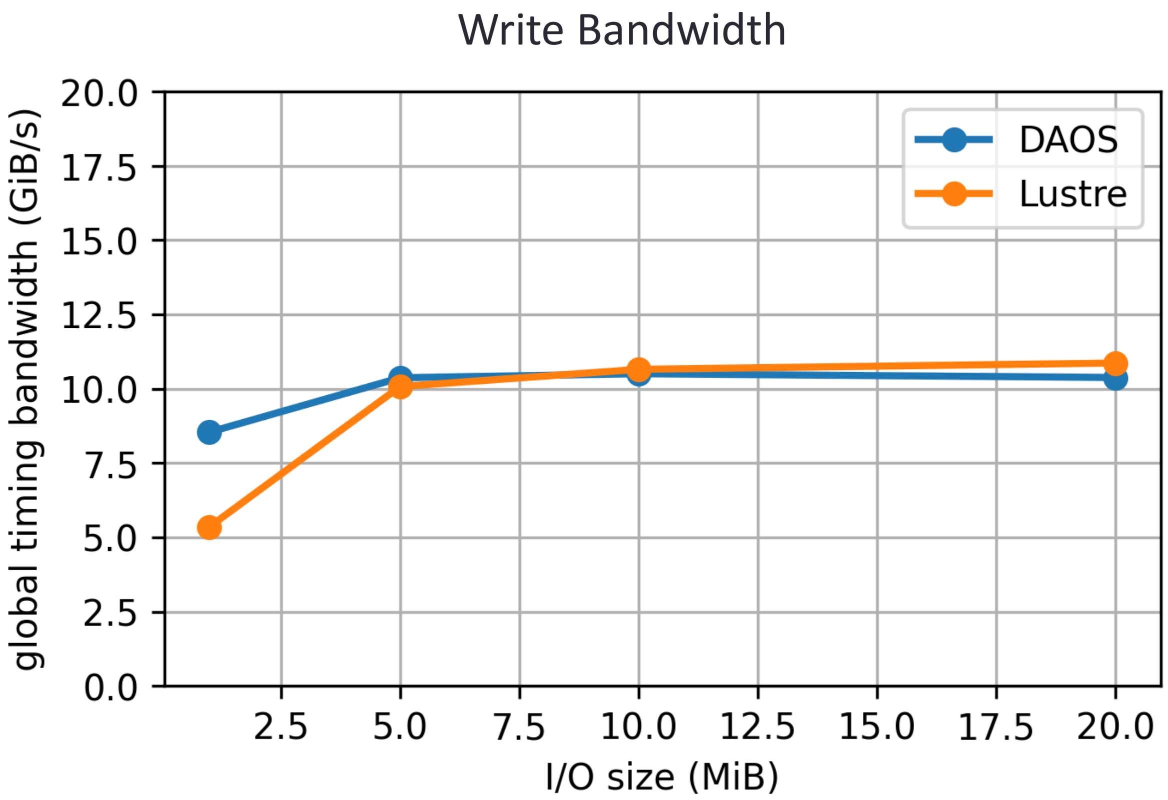
Pattern B: 1MB



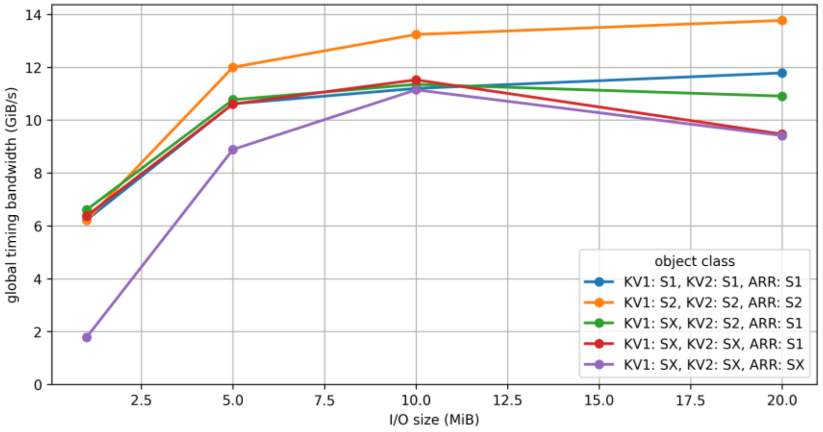
Data size

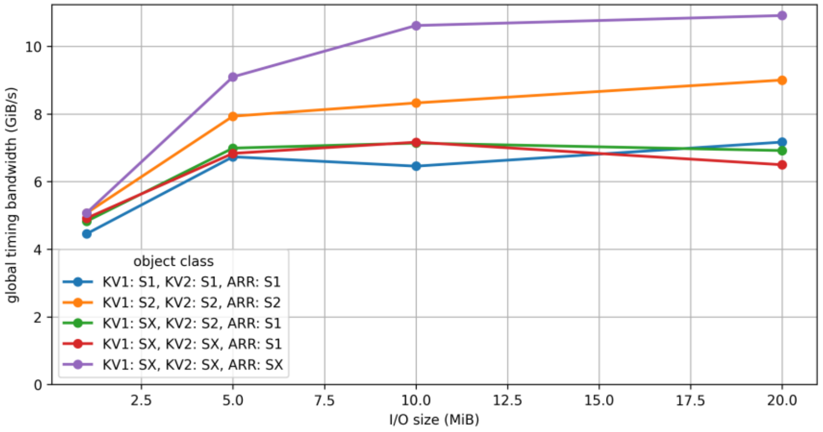
- Pattern A:
  - 2 server nodes 4 client nodes

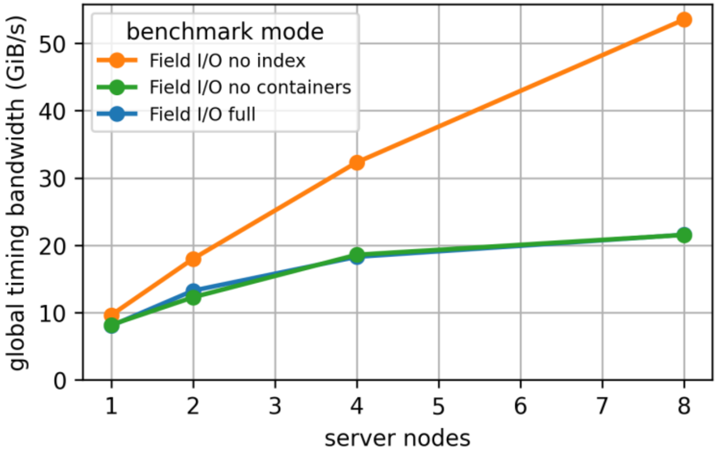


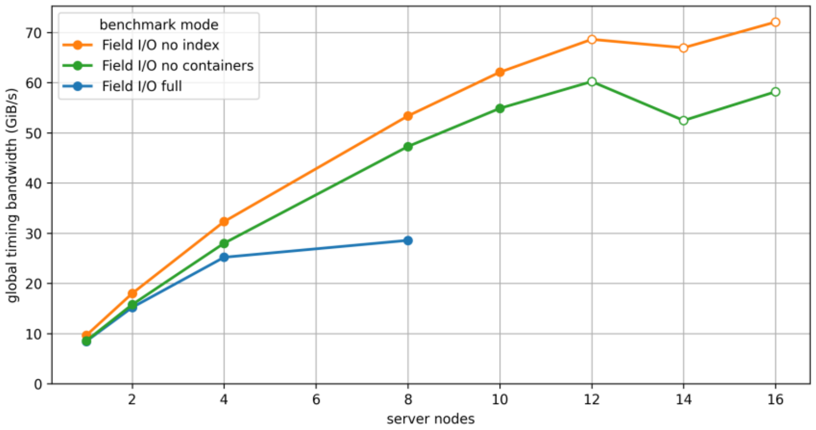


In-depth DAOS performance

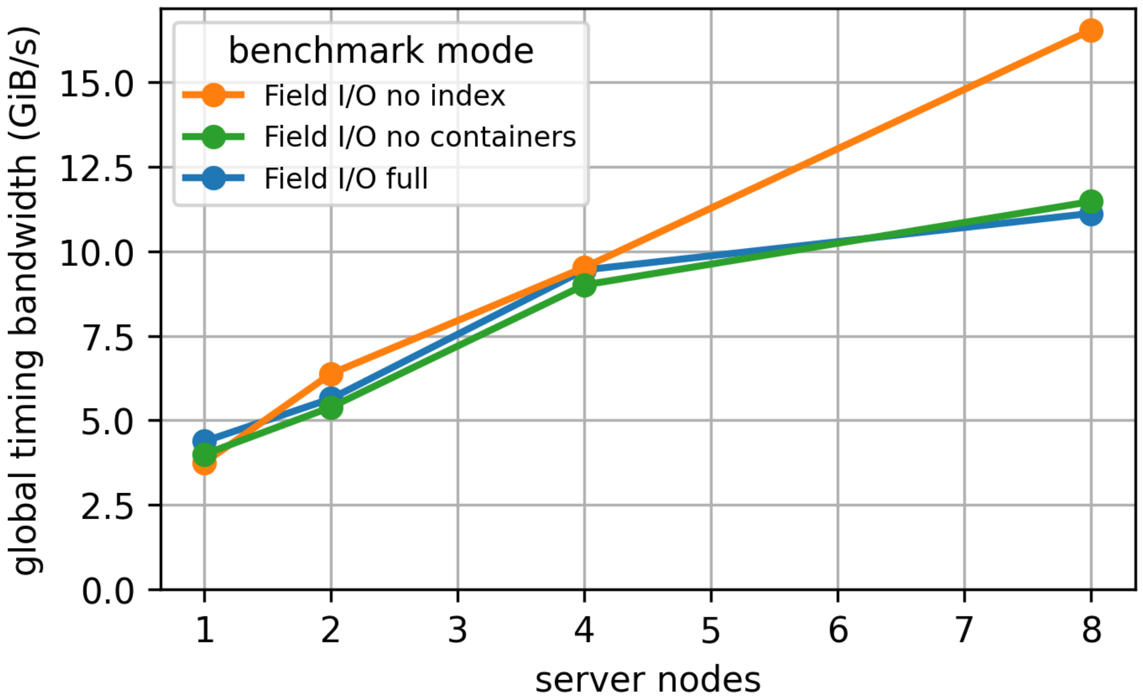


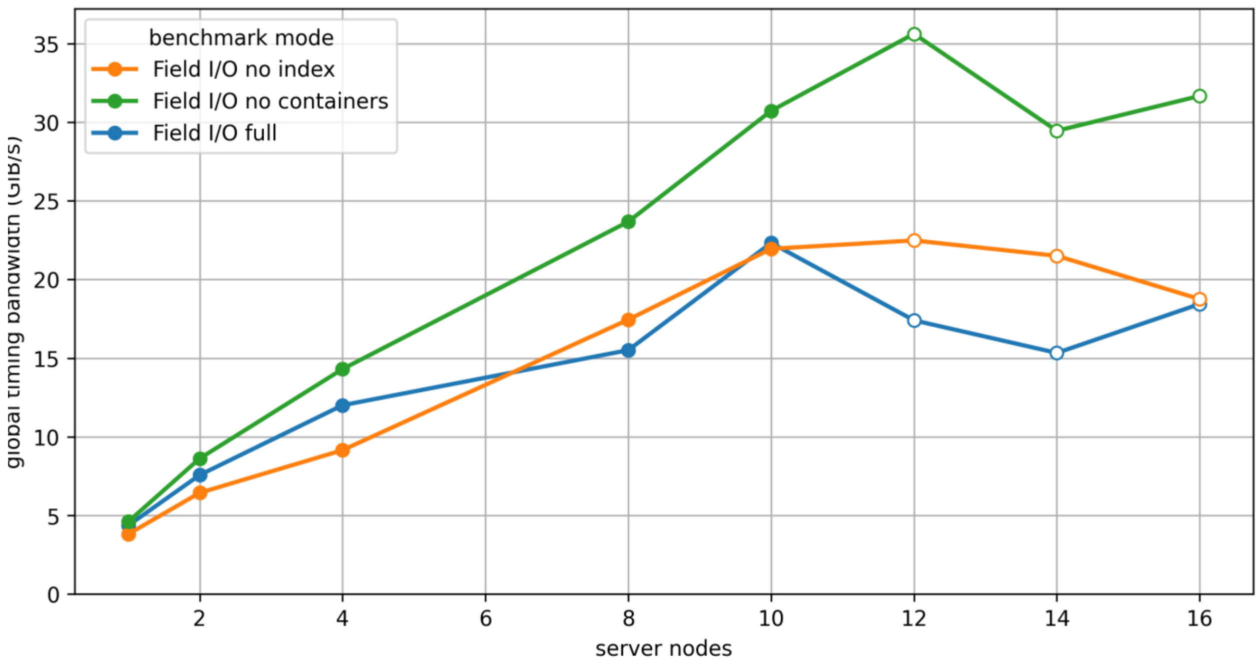




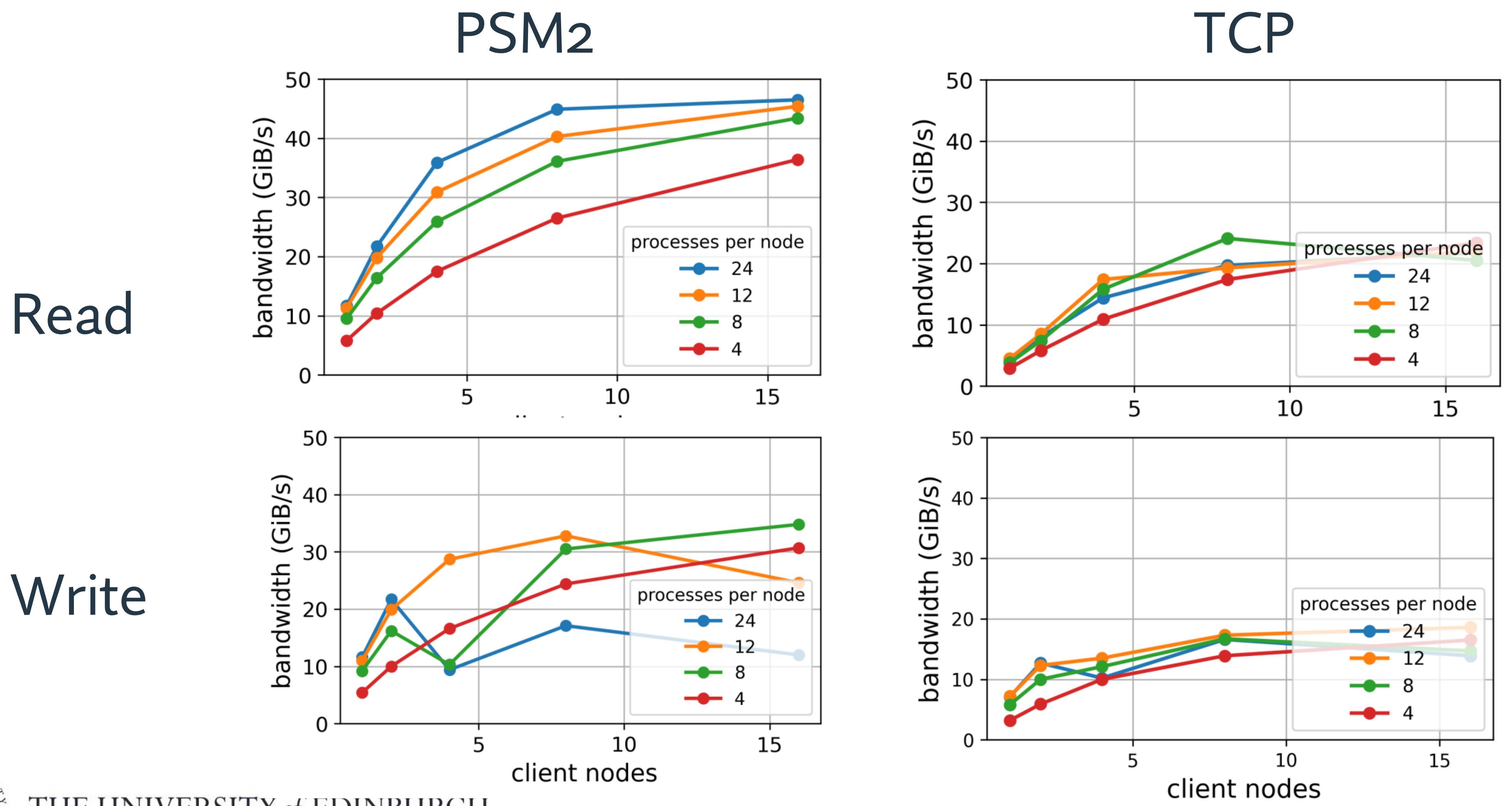


In-depth DAOS performance

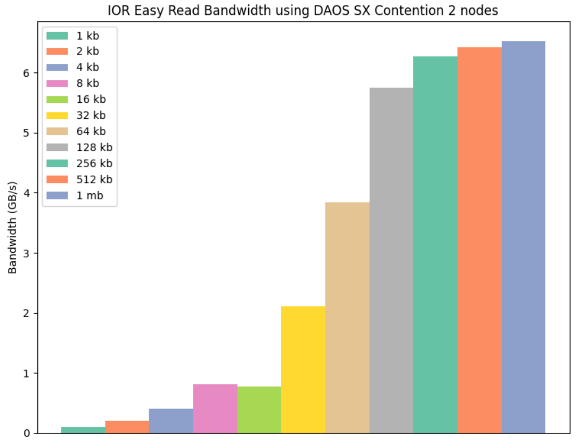


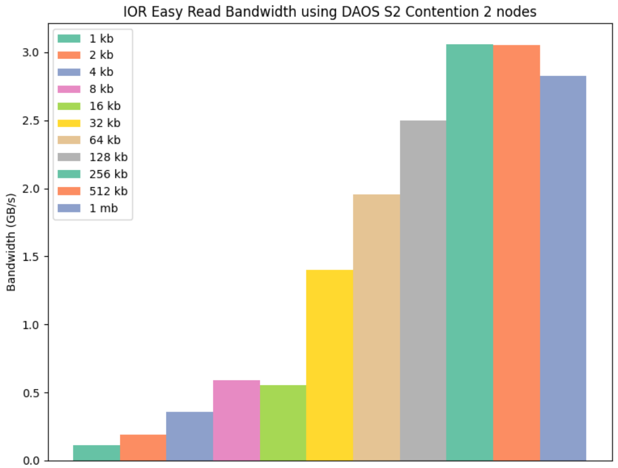


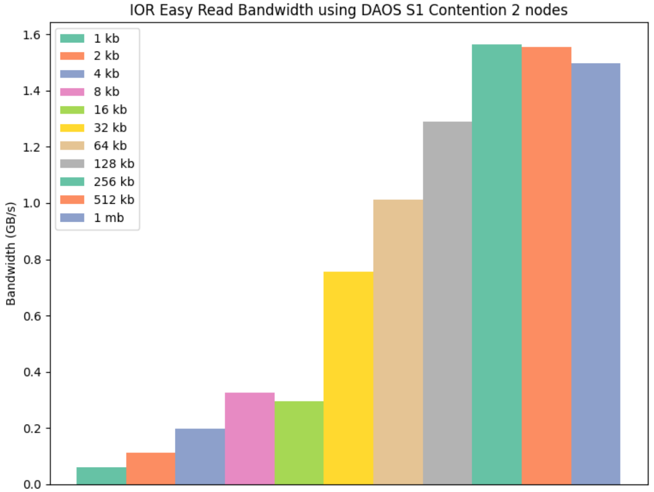
In-depth DAOS performance

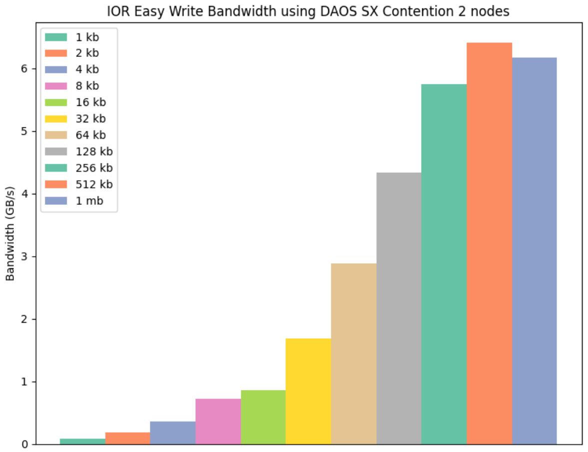


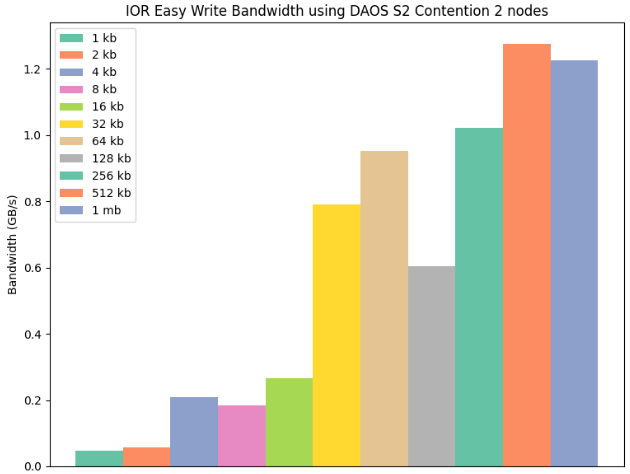
In-depth DAOS performance

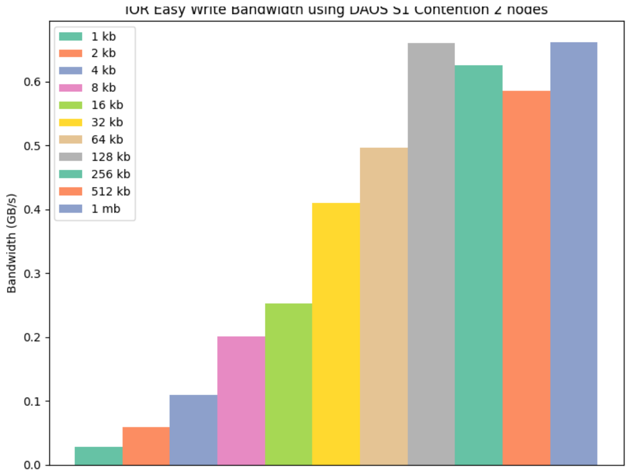












THE UNIVERSITY of EDINBURGH

Summary

- Performance impacts at all levels of I/O
  - Hard to disentangle different aspects, but important to try

- Software granularity matter but doesn't solve everything

- More complex systems are more complex

- Lots of interesting work to do