Performance of File System

Operations and Computer Technologies Div.
RIKEN Center for Computational Science

Contents

Introduction

- **Please refer to the Users Guide for overview and configuration of file system of Fugaku.**

- **Describes performance information of file I/O in terms of metadata and data accesses.**
  - **Metadata Access :**
    Accessing metadata of files. Accessing a large number of files increases the load on the server that processes the metadata access and it becomes a bottleneck.
    open (2), close (2), stat (2), unlink (2), mkdir (2), … : Mainly metadata access
  - **Data Access :**
    Accessing contents of files. Accessing a large file increases the load on the server that processes the data access and it becomes a bottleneck.
    read(2), write(2), truncate(2), ... : Mainly data access

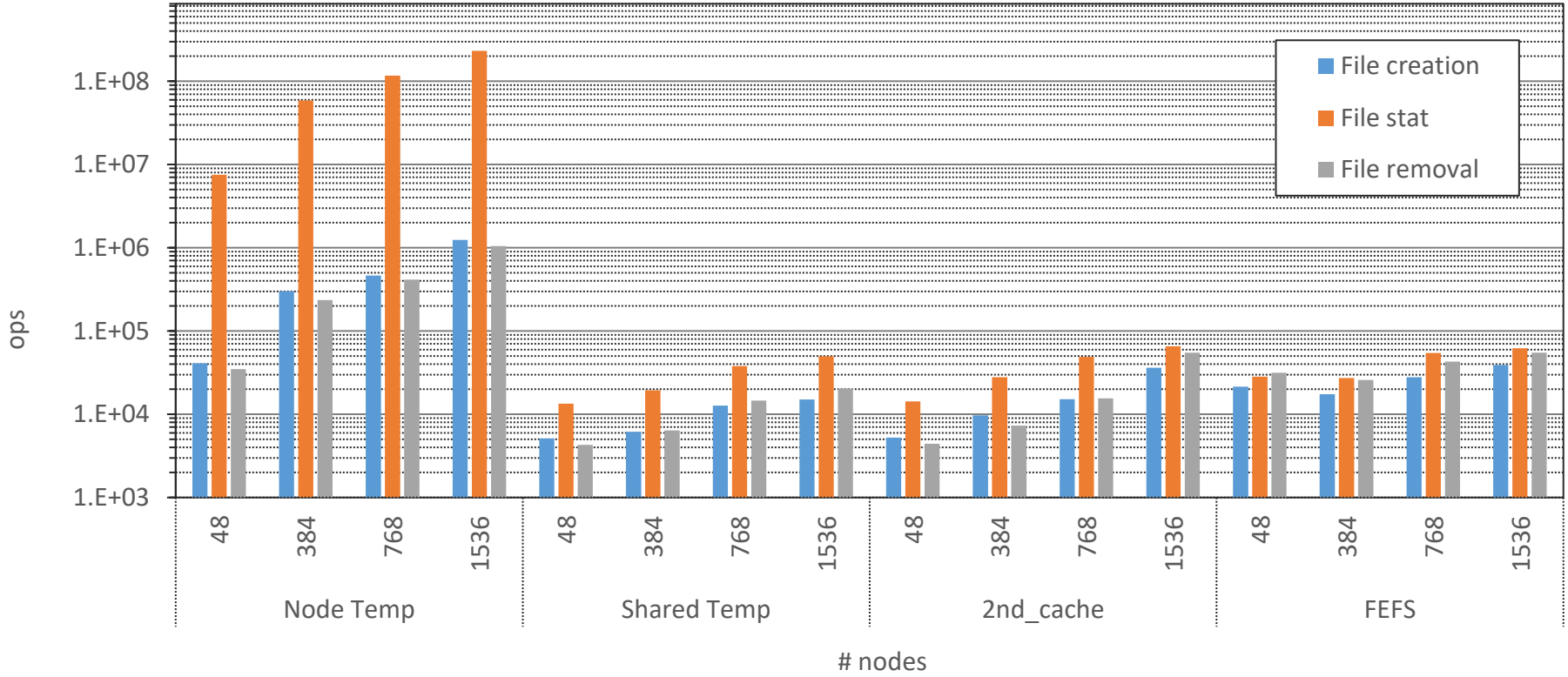
- **It may be lower than the results shown in this report because sharing the I/O bandwidth with other jobs.**

- **Please refer to the "Job Operation Software Glossary" on the Fugaku website for terms used in this document.**

Metadata access performance

- **Benchmark**
  - mdtest-3.4.0+dev
  - Options : -i 3 -n 25 -u -F -d *OUT_DIR*
    - Measurement of the maximum value
- **Measurement conditions**
  - nodes: {48:$_{2x3x8}$, 384:$_{4x6x16}$, 768:$_{4x6x32}$, 1536:$_{8x6x32}$}:strict-io
  - 4 processes/node
  - rank-map-bynode
  - llio-async-close on
  - sio-read-cache off
  - llio stripe-count 1
  - Measured with no other jobs in the whole system

Results of Metadata access performance



Node Temp : Node temporary area
Shared Temp : Shared temporary area
2nd_cache : Second-layer storage cache area
FEFS : Second-layer storage area

Evaluation of Metadata Access

- **Node temporary area**
  - Best performance
  - Performance is improved in proportion to node scale (# of SIOs)
- **Shared temporary area, second-layer storage cache area and FEFS**
  - Metadata server is shared among these areas and becomes a bottleneck
  - Performance is degraded as the number of nodes increases.
    - For the second-layer storage cache area, “llio_transfer” copies files on all SIOs and the access performance (metadata and data) is improved.

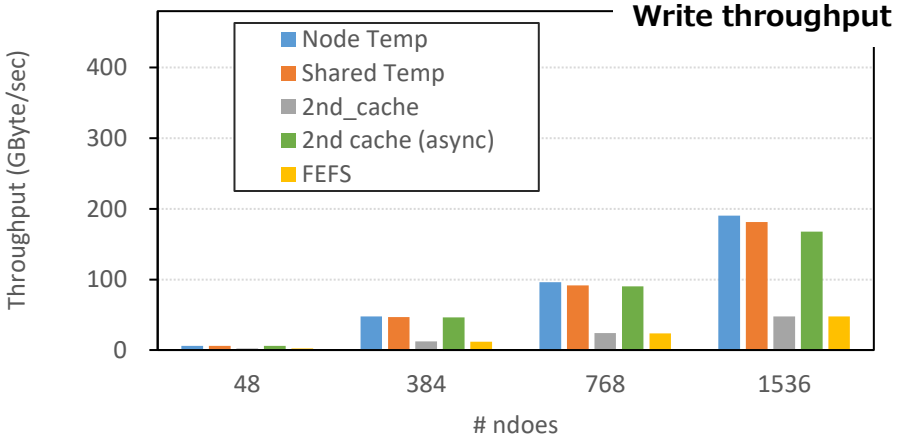
Function	Access Path	Server Name	#
Node temporary area	$PJM_LOCALTMP	SIO	9936
Shared temporary area	$PJM_SHAREDTMP	MDS	2
Second-layer storage cache area	/vol000x	MDS (SIO)	2 (9936)
Second-layer storage （FEFS）	/2ndfs	MDS	4

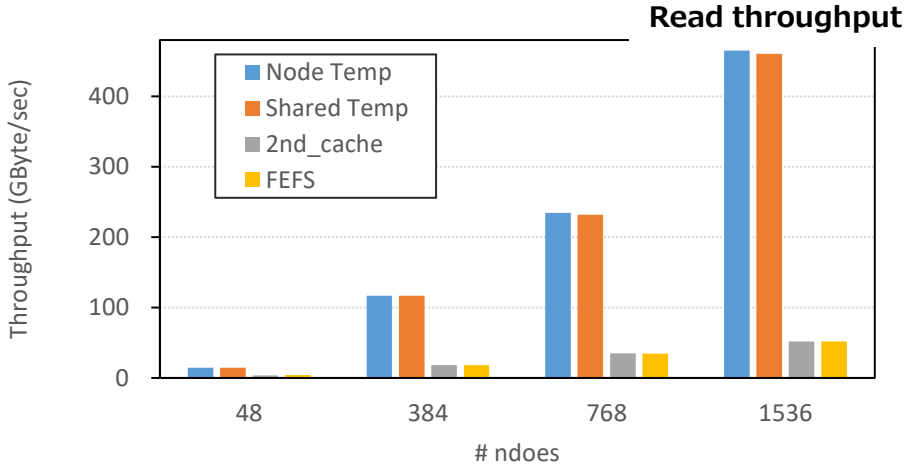
Data Throughput

- **Benchmark**
  - IOR-3.4.0+dev
  - Option: -a POSIX -b 1g -t 16m -i 3 -e -F -v -o *dir/out.dat* --posix.odirect
    - One file per process
    - Measurement of the maximum value

- **Measurement condition**
  - nodes: {48:2x3x8, 384:4x6x16, 768:4x6x32, 1536:8x6x32}:strict-io
  - 4 processes/node
  - rank-map-bynode
  - llio-async-close on
  - sio-read-cache off
  - llio stripe-count 24
  - Measured with no other jobs in the whole system

Results of Data Throughput





Node Temp:
Node temporary area

Shared Temp:
Shared temporary area

2nd_cache:
Second-layer storage cache area

2nd_cache(async):
Remove the IOR options:
-e, --posix.odirect

FEFS:
Second-layer storage area
(/2ndfs)

Evaluation of Data Throughput

- **Node temporary area**
  - Best performance
  - Throughput performance is improved in proportion to node scale (# of SIOs)
- **Shared temporary area, second-layer storage cache area and FEFS**
  - Throughput performance is improved in proportion to node scale.
  - 2nd_cache/FEFS
    - Storage server is a bottleneck
    - About 250 GB/s for 2nd_cache and 150 GB/s for FEFS is limits even if the node scale is increased.
  - Second-layer storage cache area
    - When using async mode, writing is faster

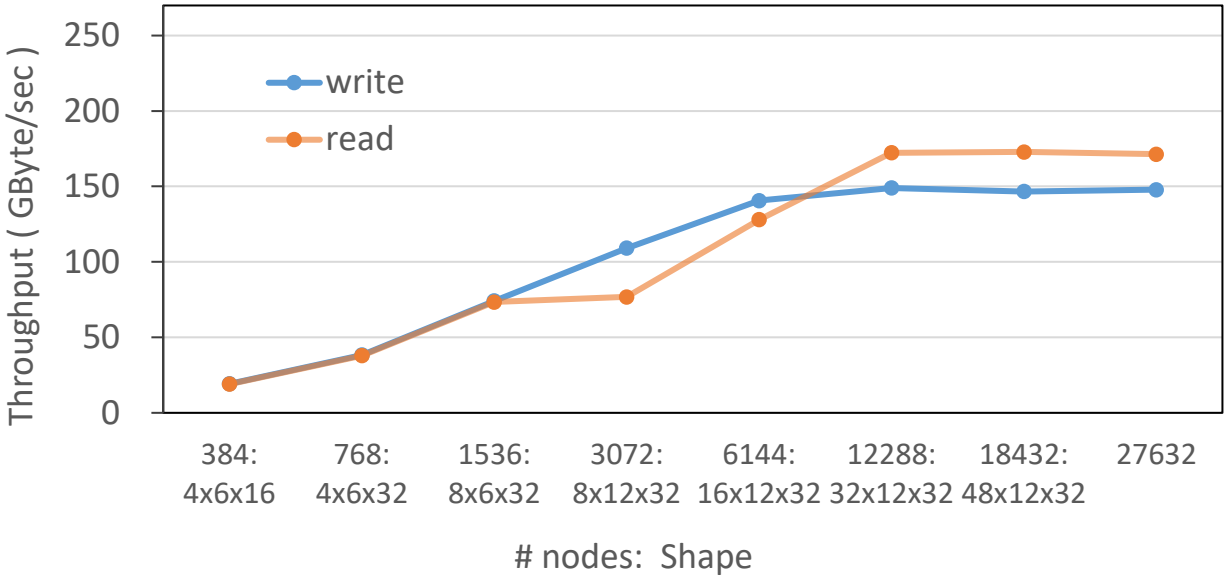
Function	Access Path	Server Name	#
Node temporary area	$PJM_LOCALTMP	SIO	9936
Shared temporary area	$PJM_SHAREDTMP	SIO	9936
Second-layer storage cache area	/vol000x	SIO (OSS)	9936 (20)
Second-layer storage area （FEFS）	/2ndfs	OSS	12

(Reference) 2ndfs throughput

- **Benchmark**
  - IOR-3.4.0+dev
  - Option: -a POSIX -b 1g -t 16m -i 3 -e -F -v -o *dir/out.dat* --posix.odirect
    - One file per process
    - Measurement of the maximum value

- **Measurement condition**
  - nodes: {384:4x6x16, 768:4x6x32, 1536:8x6x32, 3072:8x12x32, 6144:16x12x32, 12288:32x12x32, 27632}
  - 1 processes/node
  - rank-map-bynode
  - FEFS stripe-count 1
  - Measured with no other jobs in the whole system

(Reference) 2ndfs throughput



- For large jobs, the rate of increase in read throughput may not be not as good as for write due to random access to the HDD.

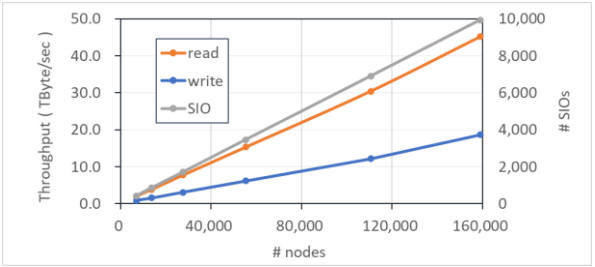
(Reference) LLIO throughput for large jobs

- **Benchmark**
  - IOR-3.4.0+dev
  - Option: -a POSIX -b 1g -t 16m -F -v -o *dir/out.dat*
    - One file per process

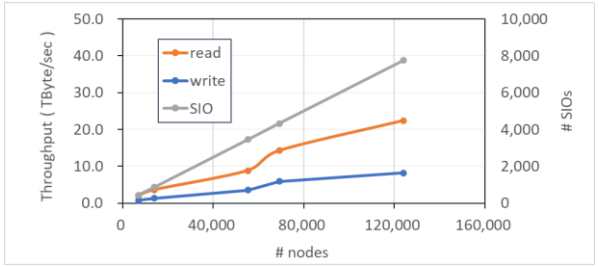
- **Measurement condition**
  - nodes: {6912, 13824, 27648, 55296, 69120, 124000, 158963}
  - 1 processes/node
  - rank-map-bynode
  - llio-async-close on
  - sio-read-cache off
  - llio stripe-count 24
  - Measured with no other jobs in the whole system

(Reference) LLIO throughput for large jobs

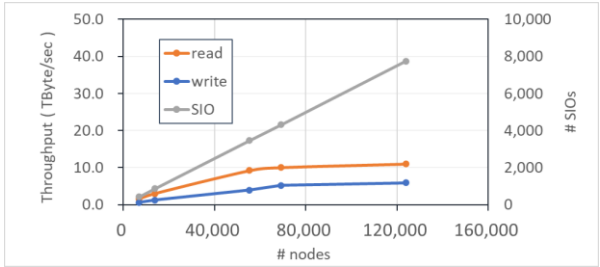
Node temporary area



Shared temporary area



Second-layer storage cache area (asnyc)



Write / SIO (MB/s)

# of ndoes	Node Temp	Shared Temp	2nd_cache
6,912	1,997	1,797	1,552
13,824	1,797	1,477	1,494
27,648	1,803		
55,296	1,785	1,009	1,142
69,120		1,363	1,210
124,000	1,764	1,053	770
158,963	1,876		

Read / SIO (MB/s)

# of ndoes	Node Temp	Shared Temp	2nd_cache
6,912	4,642	4,615	3,633
13,824	4,471	4,154	3,462
27,648	4,479		
55,296	4,425	2,559	2,666
69,120		3,294	2,323
124,000	4,405	2,891	1,418
158,963	4,548		

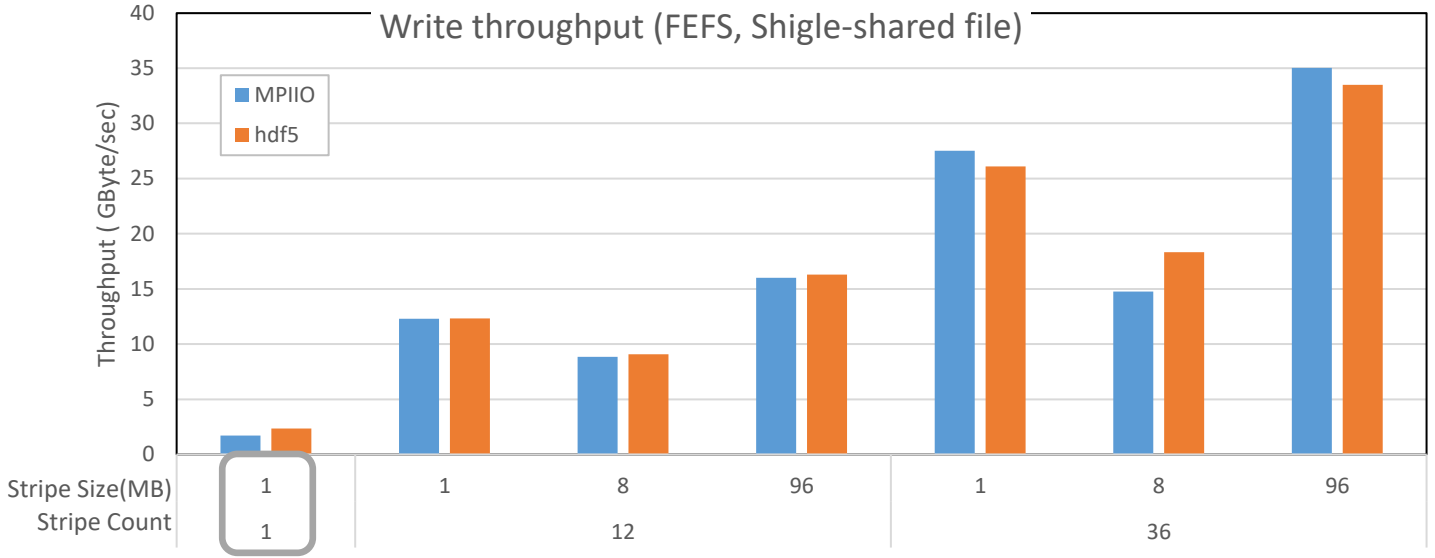
- For Shared-Temp and 2nd_cache, throughput performance is NOT improved in proportion to node scale.
- The files were created in the same directory, the cost of meta-access was high.

Note: For Second-layer storage cache area (asnyc), measured reading data already on second-layer storage cache area, and measured writing to the cache area without synchronising to the second-layer storage area.

Data Throughput

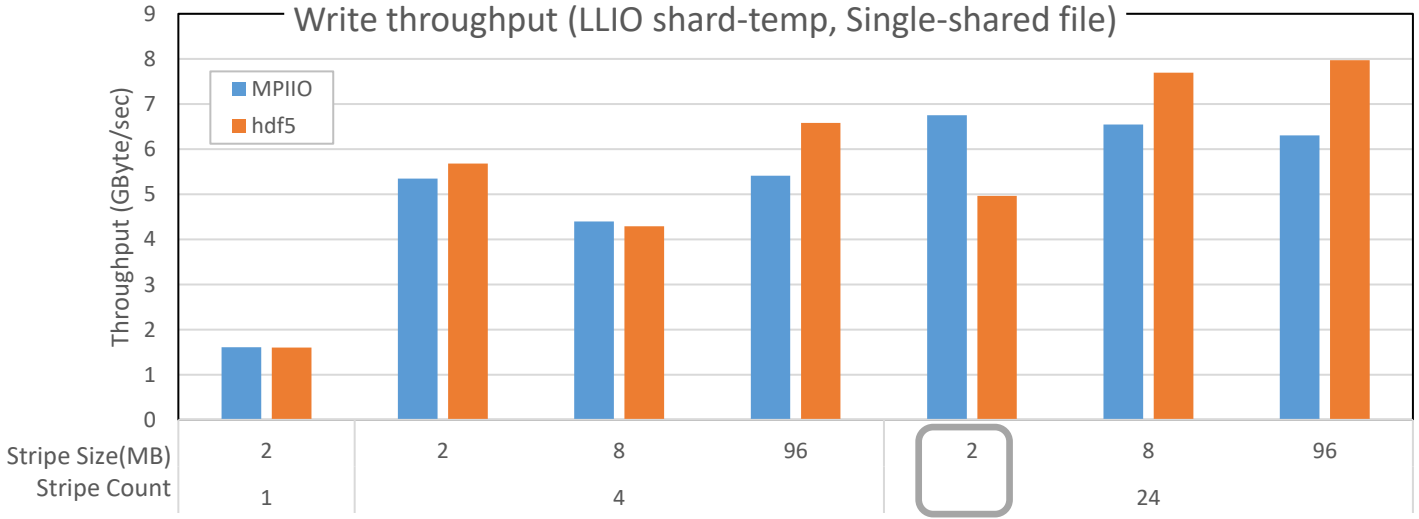
- **Benchmark**
  - IOR-3.4.0+dev
  - Option: -a MPIIO -b 96m -t 1m -i 1 -v -o *dir/out.dat*
  - Option: -a hdf5 -b 96m -t 1m -i 1 -v --hdf5.setAlignment={stripe size} -o *dir/out.dat*
    - Single-shared file
    - Specify the stripe size and stripe count for *dir/out.dat*
      - stripe size: { 1 MByte:default, 8 MByte, 96MByte }
- **Measurement condition**
  - Simple data access (no strided access)
  - nodes: 768 (4x6x32:strict-io:io-exclusive)
  - 4 processes/node
  - rank-map-bychip
  - llio-async-close on
  - Measured while other jobs are running

Results of Data Throughput



- Default stripe setting is not good for single-shared file
- Best is 36 OSTs stripe counts
- The performance is improved by matching the IO length and stripe size.

Results of Data Throughput

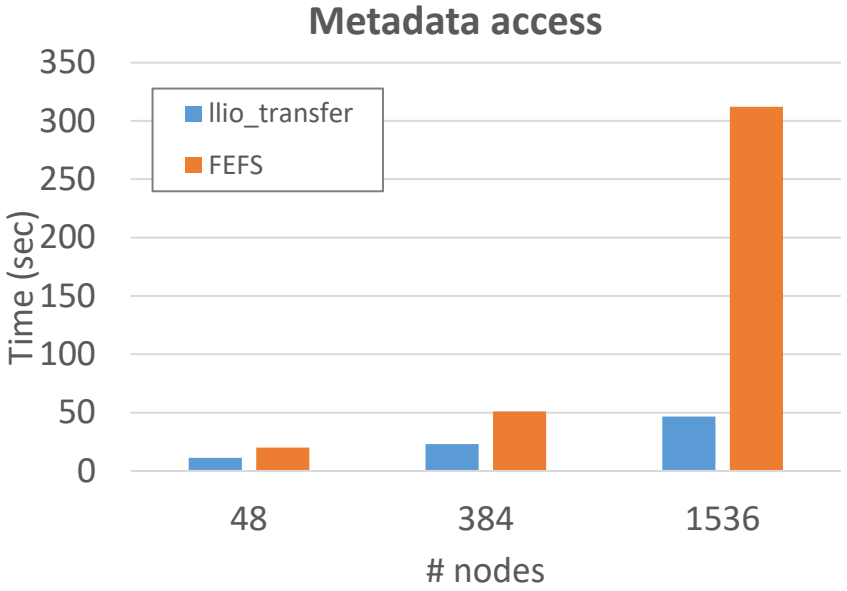


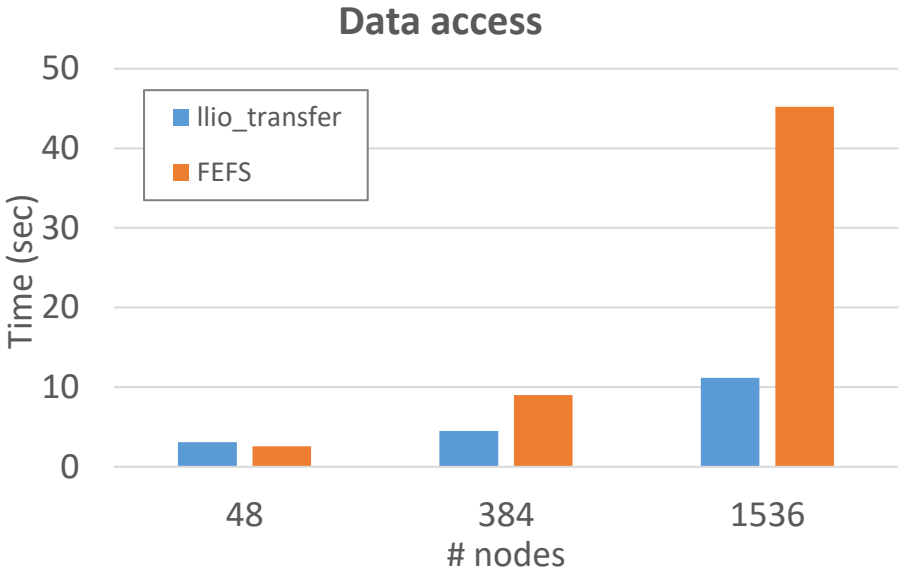
- Good performance even with default stripe setting
- Not performing as well as FEFS
- Due to LLIO limitations, using one file from multiple processes cannot be performed on a large scale.

Performance of llio_transfer

- **Overview**
  - Measured performance of metadata access and data access using common file distribution command (`llio_transfer`)
    - Access is concentrated on files (common files) read from all compute nodes, such as executable files and configuration files.
    - `llio_transfer` distributes a common read-only file on 2nd layer storage to the cache of 2nd layer storage, and balances the load of access to it and improves the performance of file I/O.
  - Compared FEFS access and access using `llio_transfer`
- **Measurement**
  - How
    - Measured mpiexec execution time
      - Metadata access : Only importing python script (About 600 files)
      - Data access : Reading 256 MiB data per process with dd
    - `llio_transfer` processing time is included.
  - Conditions
    - nodes: {48, 384, 1536:8x6x32}
      - (Node shape is not specified for 48 and 384)
    - 4 processes/node
    - rank-map-bychip
    - llio-async-close off
    - stripe-count 1
    - Measured with no other jobs in the whole system

Results of llio_transfer





- **Performance of `llio_transfer` is proportional to the node scale.**
  - `llio_transfer` eliminates the bottleneck caused by number of servers for both metadata and data accesses, and performance is improved.
  - It is recommended that you always use `llio_transfer`.

Update history

Changes	Date
1st release	15 November, 2021
2nd release p.9: Added notes where performance is FEFS > LLIO in small job. p.11: Added notes for node scale. pp.14-16: Added performance information for MPIIO and HDF5.	21 February, 2022
3rd release pp.14-16: Updated performance information for MPIIO and HDF5.	22 May, 2023
4th release pp.4-11: Updated performance information .	14 December, 2023
5th release pp.10-13: Added measurement conditions and note.	19 June, 2024