Benchmarking and profiling I/O

- Stefano Cozzini
- CNR-IOM and eXact lab srl

Agenda

- I/O performance
- Introduction to benchmarking
- I/O benchmarking
- I/O benchmarking tools
  - Iozone
  - IOR
  - mdtest
- I/O profiling
  - Darshan tool
- Optimization techniques: MPI_hints
- Conclusions

I/O performance

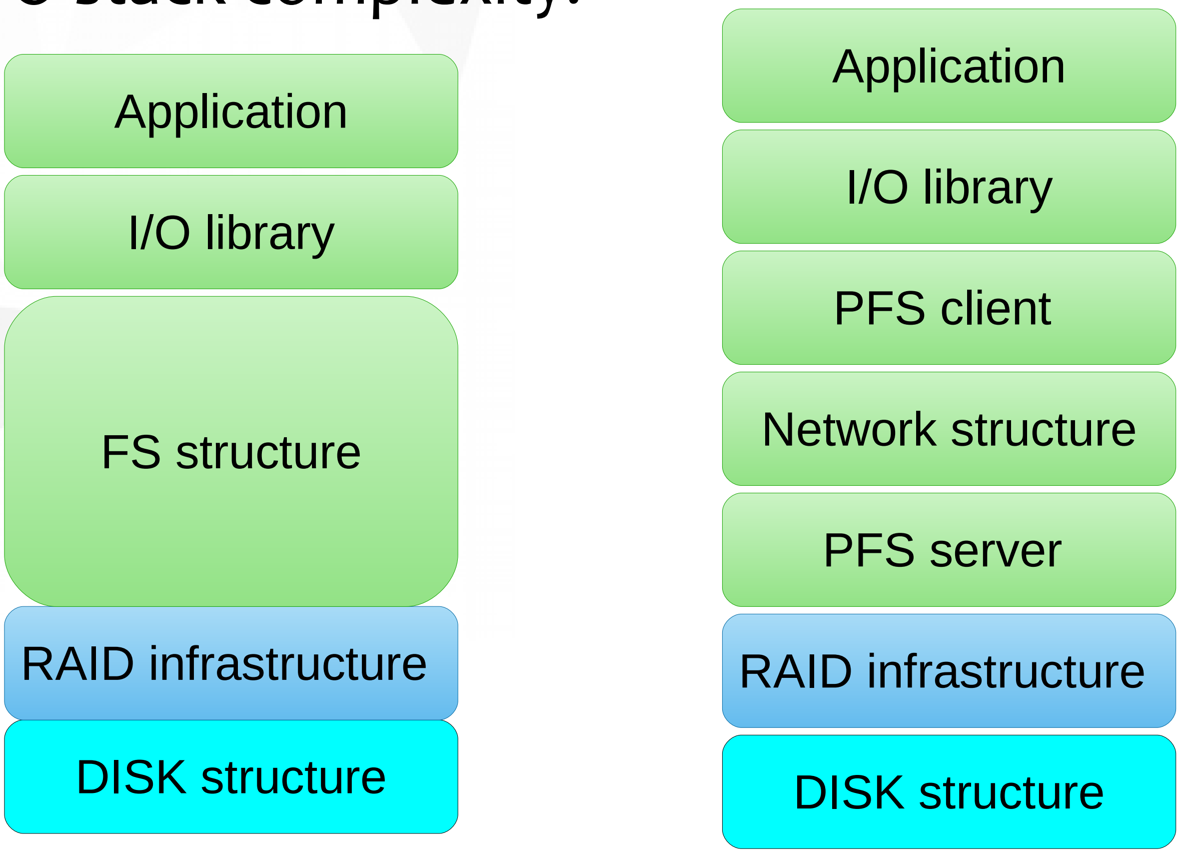
- There is no “One Size Fits All” solution to the I/O problem.
- Many I/O patterns work well for some range of parameters.
- Bottlenecks in performance can occur in many locations. (Application and/or File system)
- Going to extremes with an I/O pattern will typically lead to problems.
- Increase performance by decreasing number of I/O operations (latency) and increasing size (bandwidth).

What is a benchmark ?

- Standardized way to compare performance of different systems
- Properties of a good benchmark
  - *Relevant:* captures essential attributes of real application workload
  - *Simple:* Provides an understandable metric
  - *Portable & scalable*
  - *Consistent & repeatable results* (on same HW)
  - *Accepted by users & vendors*
- Types of benchmark
  - Microbenchmark
  - Application-based benchmark
  - Synthetic workload

I/O benchmarking

- It is becoming more and more important
- I/O performance tends to be trickier than CPU/memory ones due to The I/O stack complexity:



How to test a complex I/O infrastructure

- Benchmark all the single component of the infrastructure
- Compare simple component Peak performance with measured numbers
- Combine all numbers together to get a performance model and some expected value
- Perform the high level benchmark and compare against what you evaluated.

When benchmarking...

- Plan What and How You Are Going To Test
  - *Decide what type of workload test to perform (reads, writes, I/O size, random, sequential, response time)*
- Decide/Select the appropriate Benchmarking Tools
- Benchmark the benchmarking tools
  - *Be sure you are able/understand how to use them*
- Focus on the metric that matters you
  - *IOPS vs Bandwidth ?*
  - *Metadata vs data ?*
  - *Cpu load / Network load ?*

Application I/O benchmarks..

- Run real application on real data set, measure time
  - Best predictor of application performance on your cluster
  - Requires additional resources (compute nodes, etc.)
- Difficult to acquire when evaluating new gear
  - Vendor may not have same resources as their customers
  - Can be hard to isolate I/O vs. other parts of application
  - Performance may depend on compute node speed, memory size, interconnect, etc.
- Difficult to compare runs on different clusters

  Time consuming – realistic job may run for days, weeks
- May require large or proprietary dataset
- Hard to standardize and distribute

I/O microbenchmarks to play

- Measures one fundamental operation in isolation
  - – Read throughput, write throughput, creates/sec, etc.
- Good for:
  - Tuning a specific operation
  - Post-install system validation
  - Publishing a big number in a press release
- Not as good for:
  - Modeling & predicting application performance
  - Measuring broad system performance characteristics
- Example to play
  - IOR: https://github.com/hpc/ior
  - iozone (www.iozone.org)
  - Mdtest (included in the IOR)

Platforms&FS to test..

- ULYSSES SISSA (/scratch /home)
- C3HPC (/lustre /local_scratch)

IOZONE (1)

Compilation: trivial

```
wget http://www.iozone.org/src/current/iozone3_429.tar
tar -xvf iozone3_429.tar
cd iozone3_429/src/current
make
make linux
./iozone
```

IOZONE (2)

- Test to run:
  - Iozone -a (basic testing)
  - Large file (large than memory to avoid caching effects)

    ```
    iozone -i 1 -i 0 -s 32g -r 1M -f ./32gzero2
    ```
  - On lustre try to stripe the file over different OST
  - Iozone with multiple threads…

- Short introduction of basic flags:
  http://www.thegeekstuff.com/2011/05/iozone-examples/

IOZONE on different OSTs..

- 1. create directories with appropriate stripe

```
mkdir ost0
lfs setstripe  -c 1 -i 0 ost0
```

- 2. run the code

```
IO_test/iozone3_429/src/current/iozone -i0 -i
1 -s 20m -r 1m -f ost0/poo
```

IOZONE multiple threads

- Options to explore/use:
- -t # Number of threads or processes to use in throughput test
- -+m Cluster_filename   Enable Cluster testing

IOZONE multiple threads (2)

```
# Example: With two copies of Iozone on each of the two clients
# (format 4 fields)
# client1  /home/user/tmp /home/user/tmp/iozone /tmp/foo1
# client1  /home/user/tmp /home/user/tmp/iozone /tmp/foo2
# client2  /home/user/tmp /home/user/tmp/iozone /tmp/foo3
# client2  /home/user/tmp /home/user/tmp/iozone /tmp/foo4
b11  /lustre/exact/exact /u/exact/exact/bin/iozone /lustre/exact/exact/tmp1
b12  /lustre/exact/exact /u/exact/exact/bin/iozone /lustre/exact/exact/tmp2
b13  /lustre/exact/exact /u/exact/exact/bin/iozone /lustre/exact/exact/tmp3
B14  /lustre/exact/exact /u/exact/exact/bin/iozone /lustre/exact/exact/tmp4
```

```
~/bin/iozone  -+m ./cosint_list  -t 4 -i 0 -i 1 -s 80g -r 1m
```

NOTE: set the RSH environment variable to /usr/bin/ssh

I/O performance measurements

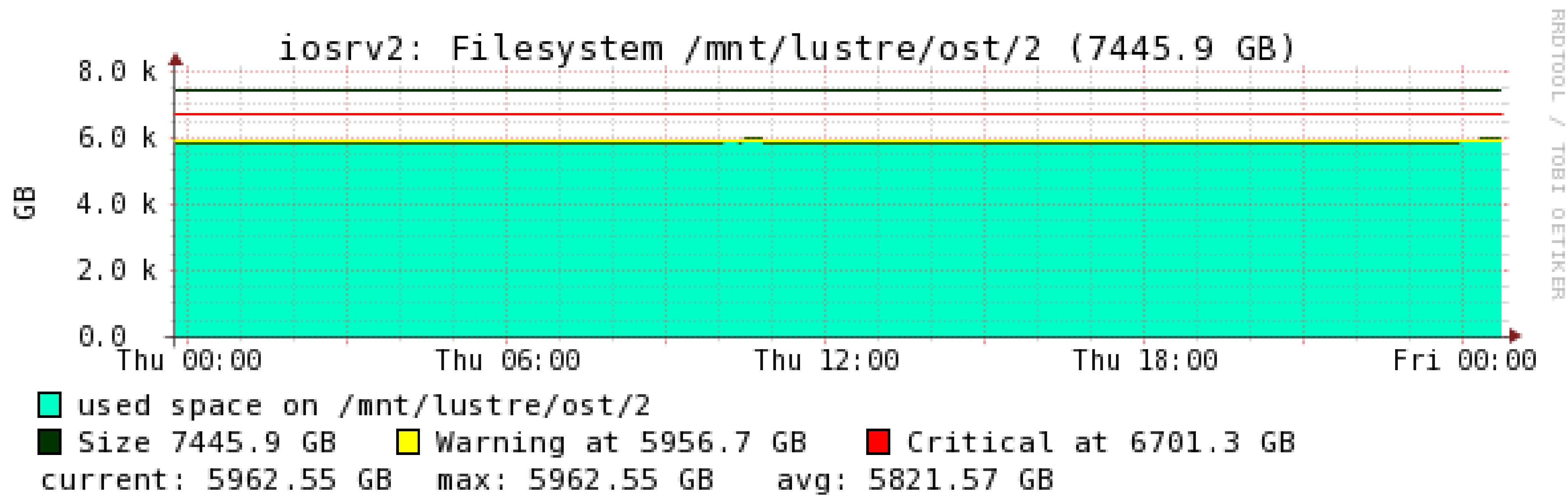
- Lots of different performance metrics
  - Sequential bandwidth, random I/Os, metadata operations
  - Single-threaded vs. multi-threaded
  - Single-client vs. multi-client
  - N-to-N (file per process) vs. N-to-1 (single shared file)
- Ultimately a method to try to estimate what you really care about
  - "Time to results", aka "How long does my app take?"
- Benchmarks are best if they model your real application
  - Need to know what kind of I/O your app does in order to choose appropriate benchmark
  - Similar to CPU benchmarking – e.g., LINPACK performance may not predict how fast your codes run

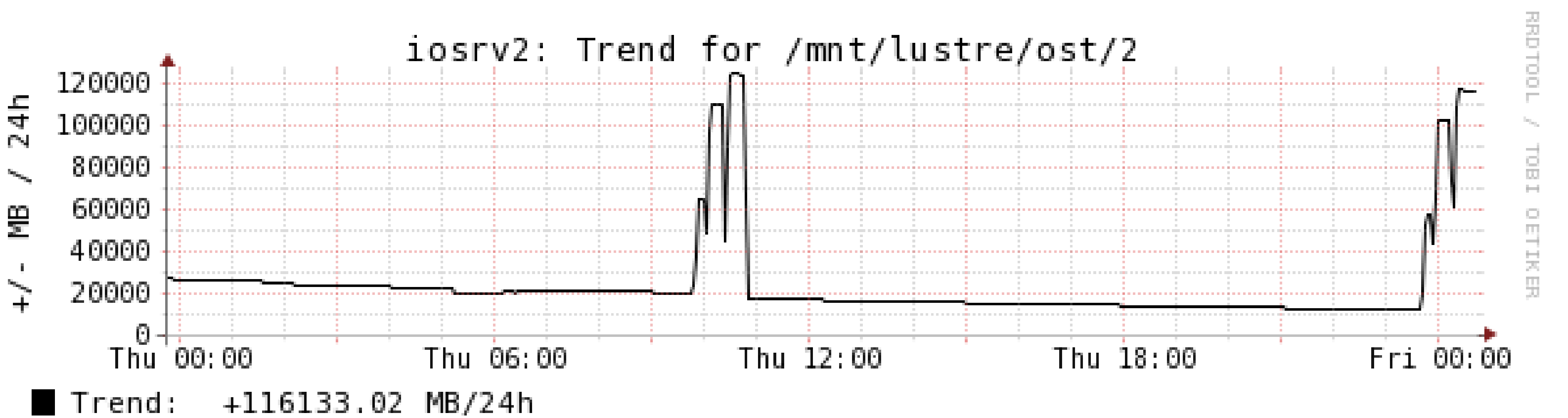
Numbers to be collected IOZONE (1)

- Statistics (at least 3 independent runs) for all OSTes on
  - Ulysses (12 scratch) (2 home)
  - C3HPC(4)

Basic script prepared by us on your github account. PLEASE IMPROVE IT

Benchmarking activities in progress..





IOR : the de-facto I/O benchmark for HPC

HPC IO Benchmark Repository build error

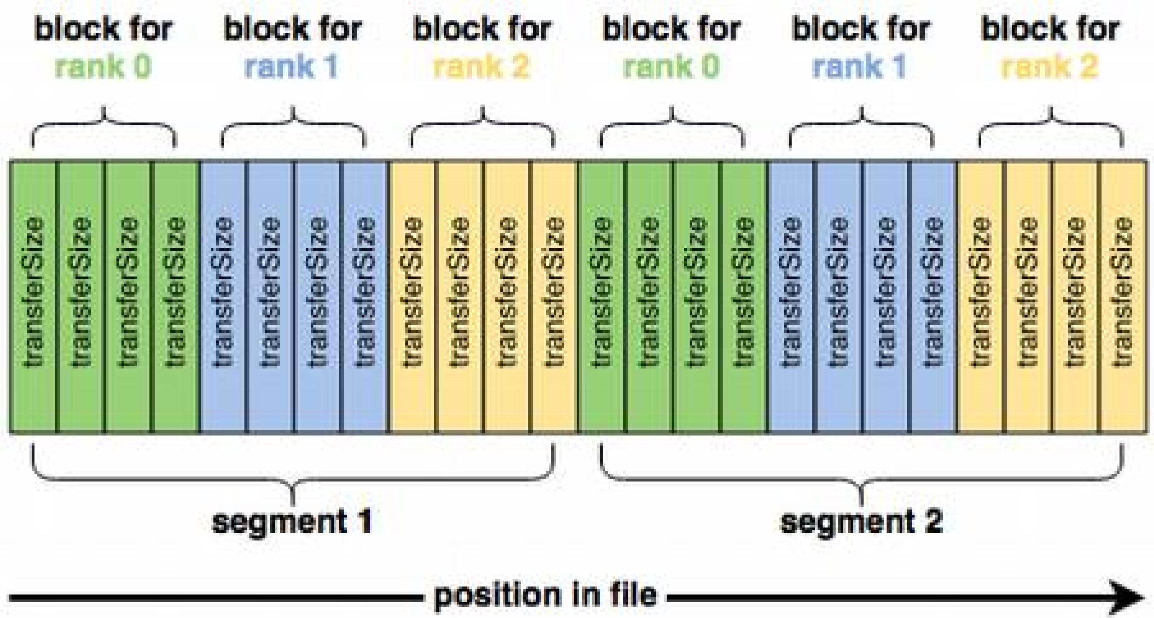
This repository contains the IOR and mdtest parallel I/O benchmarks. The official IOR/mdtest documention can be found in the `docs/` subdirectory or on Read the Docs.

Building

1. If `configure` is missing from the top level directory, you probably retrieved this code directly from the repository. Run `./bootstrap` to generate the configure script. Alternatively, download an official IOR release which includes the configure script.
2. Run `./configure`. For a full list of configuration options, use `./configure --help`.
3. Run `make`
4. Optionally, run `make install`. The installation prefix can be changed via `./configure --prefix=...`.

IOR basic usage

- IOR writes data sequentially with the following parameters:
  - `blockSize (-b)`
  - `transferSize (-t)`
  - `segmentCount (-s)`
  - `numTasks (-n)`



Find the difference ..

```
$ mpirun -n 64 ./ior -t 1m -b 16m -s 16
...
access bw(MiB/s) block(KiB) xfer(KiB) open(s)  wr/rd(s) close(s) total(s) iter
------ --------- ---------- --------- -------- -------- -------- -------- ----
write  427.36    16384      1024.00   0.107961 38.34    32.48    38.34    2
read   239.08    16384      1024.00   0.005789 68.53    65.53    68.53    2
remove -         -          -         -        -        -        0.534400 2
```

```
$ mpirun -n 64 ./ior -t 1m -b 16m -s 16 -F
...
access bw(MiB/s) block(KiB) xfer(KiB) open(s)  wr/rd(s) close(s) total(s) iter
------ --------- ---------- --------- -------- -------- -------- -------- ----
write  33645     16384      1024.00   0.007693 0.486249 0.195494 0.486972 1
read   149473    16384      1024.00   0.004936 0.108627 0.016479 0.109612 1
remove -         -          -         -        -        -        6.08     1
```

Switching from writing to a single-shared file to one file per process using the -F (filePerProcess=1) option changes the performance dramatically !

Taken from: https://glennklockwood.blogspot.com/2016/07/basics-of-io-benchmarking.html

Further option for IOR

- Some more options to know:

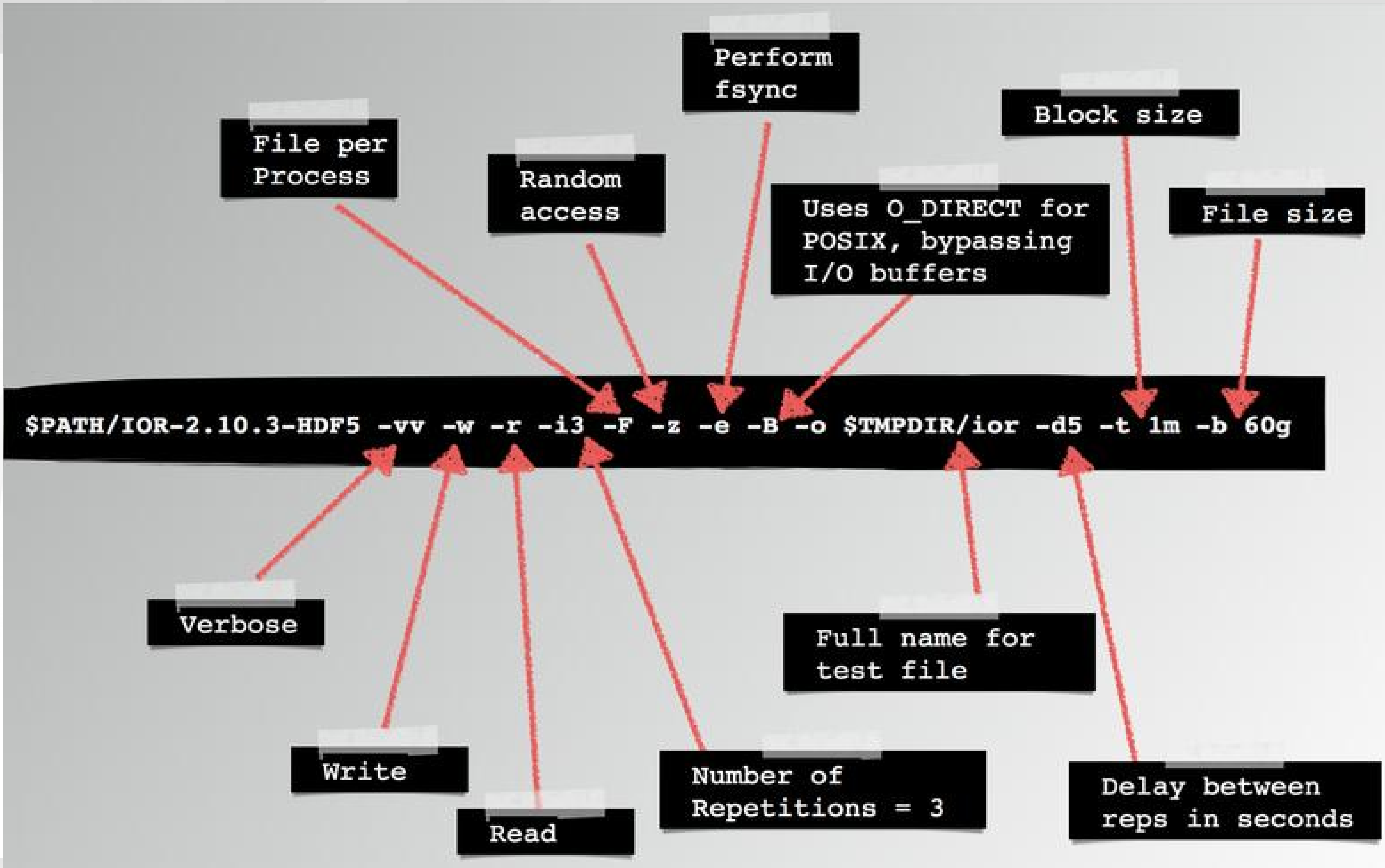
`-d N` interTestDelay -- delay between reps in seconds

`-i N` repetitions -- number of repetitions of test

`-k` keepFile -- don't remove the test file(s) on program exit

`-E` useExistingTestFile -- do not remove test file before write access

`-r` readFile -- read existing file


Perform fsync
Block size
File per Process
Random access
Uses O_DIRECT for POSIX, bypassing I/O buffers
File size
$PATH/IOR-2.10.3-HDF5 -vv -w -r -i3 -F -z -e -B -o $TMPDIR/ior -d5 -t 1m -b 60g
Verbose
Full name for test file
Write
Read
Number of Repetitions = 3
Delay between reps in seconds

IOR: numbers to be collected..

- Compare performance of HDF5 vs MPIIO vs POSIX..
- Run the following experiments:
  - `mpirun -np 32 IOR -a [POSIX|MPIIO|HDF5] -i 3 -d 32 -k -r -E -o yourfile_name -s 1 -b 60G -t 1m`
  - `mpirun -np 32 IOR -a [POSIX|MPIIO|HDF5] -i 3 -d 32 -k -r -E -o yourfile_name -s 1 -b 16G -t 1m`
  - `mpirun -np 32 IOR -a [POSIX|MPIIO|HDF5] -i 3 -d 32 -k -r -E -o yourfile_name -s 1 -b 4G -t 1m`

mdtest

- Example to run

```
mdtest -n 10 -i 200 -y -N 10 -t -u -d
$test_directory
```

`-n:` every process will creat/stat/remove # directories and files

`-i:` number of iterations the test will run

`-y:` sync file after writing

`-N:` stride # between neighbour tasks for file/dir stat (local=0)

`-t:` time unique working directory overhead

`-u:` unique working directory for each task

`-d:` the directory in which the tests will run

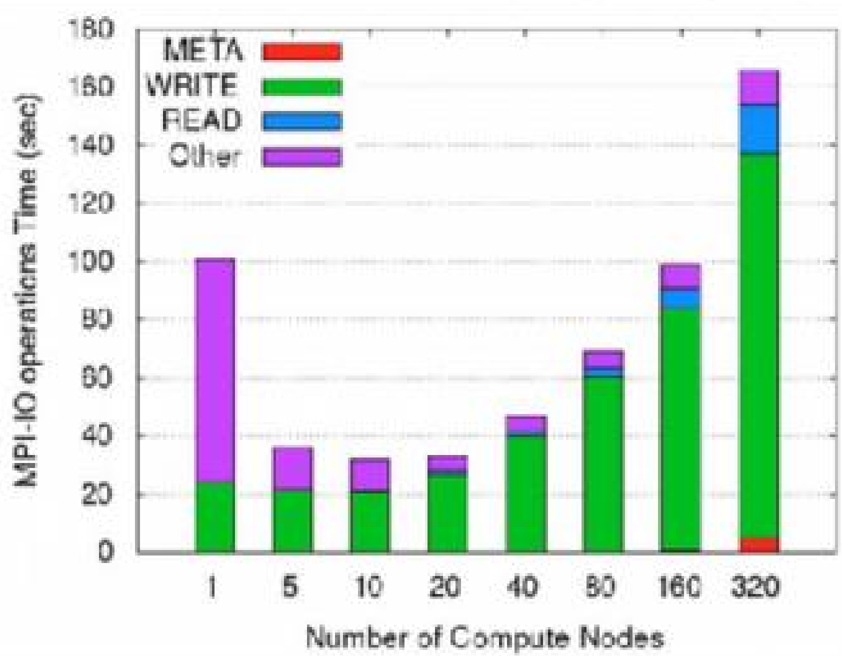
How fast the are the medata ?

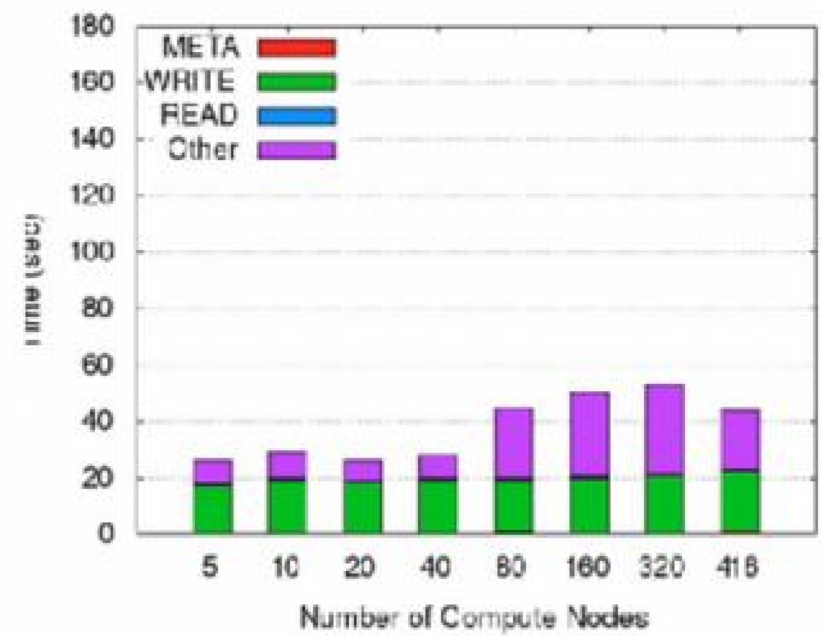
```
SUMMARY rate: (of 200 iterations)
   Operation                      Max            Min            Mean         Std Dev
   ---------                      ---            ---            ----         -------
   Directory creation        :      55501.003      26790.617      51314.024       3494.404
   Directory stat            :     432585.389     166356.183     415770.932      39806.828
   Directory removal         :      68433.152      30720.381      66283.460       4346.683
   File creation             :      91600.370      38847.635      84406.939       6310.838
   File stat                 :     437525.319     218808.354     425876.564      19175.332
   File read                 :     248538.403      95819.543     232907.018      22622.202
   File removal              :     113252.855      58348.795     109185.982       7946.754
   Tree creation             :      51850.920       8104.584      46989.748       4545.576
   Tree removal              :      49622.970      17558.931      47361.456       3847.215

SUMMARY rate: (of 200 iterations)
   Operation                      Max            Min            Mean         Std Dev
   ---------                      ---            ---            ----         -------
   Directory creation        :       3745.805        232.120       3547.295        276.796
   Directory stat            :       4975.815       3138.071       4568.473        286.241
   Directory removal         :       4160.641       2758.850       3894.362        207.197
   File creation             :       2066.019        712.113       1913.854        119.667
   File stat                 :       2005.075        809.316       1655.505        140.655
   File read                 :       2294.915       1171.782       2113.305        132.203
   File removal              :       3903.496       2569.894       3652.442        177.905
   Tree creation             :       2234.273       1204.224       2051.464        117.113
   Tree removal              :       4053.389        902.417       3824.589        268.098
```

I/O scalability

- Problem:
  - Increase the computing while increasing processor but keep constant the I/O operation, i.e. read/write the same amount of data. If I/O processes increases as the number of processors the problem does not scale anymore
- Solution:
  - Reduce the number of I/O processes



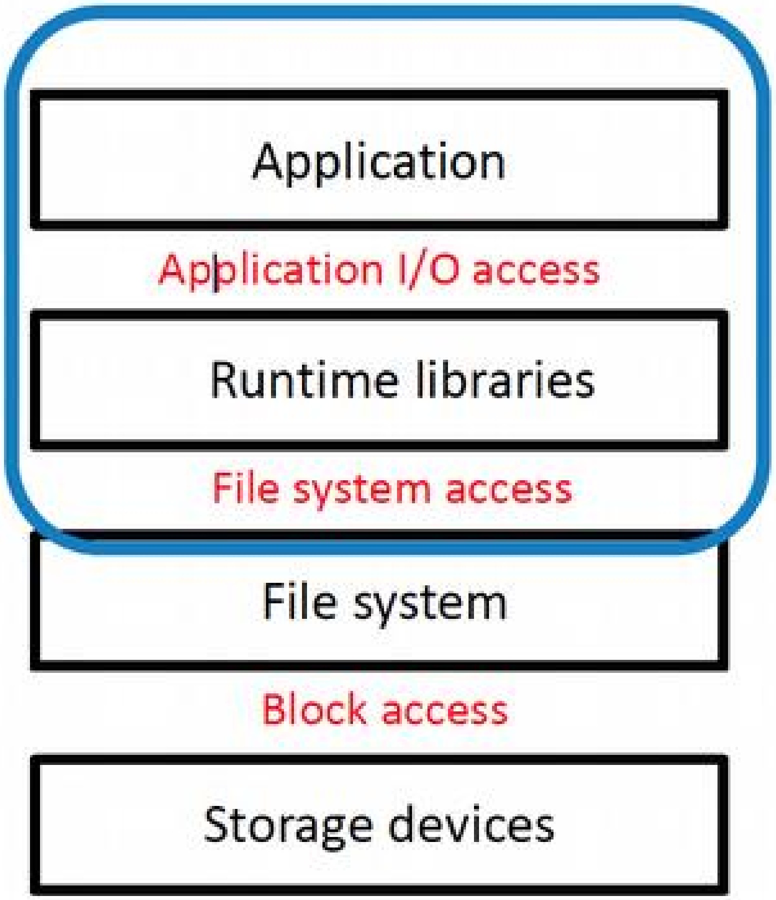


Characterizing Application I/O

How are applications using the I/O system, and how successful are they at attaining high performance?

The best way to answer these questions is by observing behavior at the application and library level

- What did the application intend to do, and how much time did it take to do it?
- we will focus on ***Darshan***, a scalable tool for characterizing application I/O activity.



Darshan overview

- Open source runtime library
  - Instrumentation is inserted at build time (for static executables) or at run time (for dynamic executables)
  - Captures POSIX I/O, MPI-IO, and limited HDF5 and PNetCDF functions
- Minimal application impact

  Low memory consumption
  - Reduces, compresses, and aggregates data at MPI_Finalize() time
  - Instrumentation enabled via software modules, environment variables, or compiler scripts
  - No source code or makefile changes
  - No file system dependencies

How to use Darshan

- Compile a C, C++, or FORTRAN program that uses MPI
  - Run the application
  - Look for the Darshan log file
  - This will be in a particular directory (depending on your system's configuration) <dir>/<year>/<month>/<day>/<username>_<appname>*.darshan.gz
- Use Darshan command line tools to analyze the log file
- Darshan does not capture a trace of all I/O operations: instead, it reports key statistics, counters, and timing information for each file accessed by the application.

What Darshan reports

- Darshan collects per-process statistics (organized by file)
  - Counts I/O operations, e.g. unaligned and sequential accesses
  - Times for file operations, e.g. opens and writes
  - Accumulates read/write bandwidth info Creates data for simple visual representation
- To produce the report:
  - In pdf :

  darshan-job-summary.pl $DARSHAN_LOGPATH/YYYY/MM/DD/username_exe_name_idjobid.xxxxx_darshan.

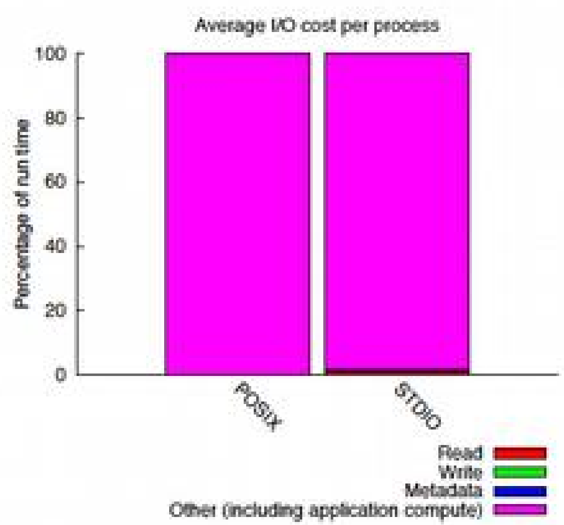
  - Summary of performance :

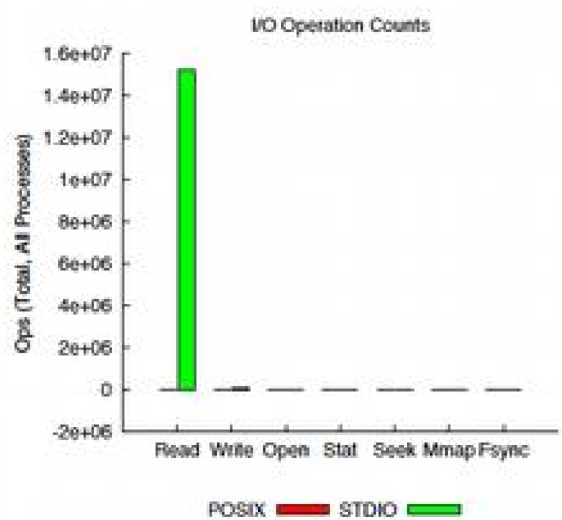
  darshan-parser –perf $DARSHAN_LOGPATH/YYYY/MM/DD/username_exe_name_idjobid.xxxxx_darshan

Darshan output..

jobid: 170515	uid: 669	nprocs: 1	runtime: 126 seconds

I/O performance *estimate* (at the STDIO layer): transferred 15.0 MiB at 8.44 MiB/s





Darshan output..

Most Common Access Sizes
(POSIX or MPI-IO)

	access size	count

File Count Summary
(estimated by POSIX I/O access offsets)

type	number of files	avg. size	max size
total opened	162	49K	488K
read-only files	46	162K	488K
write-only files	116	3.1K	193K
read/write files	0	0	0
created files	116	3.1K	193K

../../../bin/geotop .

Available darshan tools

http://www.mcs.anl.gov/research/projects/darshan/docs/darshan-util.html

Key tools:

–Darshan-job-summary.pl: creates pdf with graphs for initial analysis

–Darshan-summary-per-file.sh: similar to above, but produces a separate pdf summary for every file opened by application

–Darshan-parser: dumps all information into text format

Darshan-parser example (see all counters related to write operations):

```
darshan-parser user_app_numbers.darshan.gz | grep WRITE
```

See documentation above for definition of output fields

Optimizations

- file striping to increase IO performance
- Use I/O libraries: HDF5, NetCDF
- Serial I/O operations
- MPI-IO hints

MPI-I/O Hints

- MPI-IO hints are extra information supplied to the MPI implementation through the following function calls for improving the I/O performance
  - **`MPI_File_open`**
  - **`MPI_File_set_info`**
  - **`MPI_File_set_view`**
- Hints are optional and implementation-dependent: you may specify hints but the implementation can ignore them
  - **`MPI_File_get_info`** used to get list of hints..

ROMIO Hints: FS-Related

- `striping_factor` -- Controls the number of I/O devices to stripe across
- `striping_unit` -- Controls the striping unit (in bytes)
- `start_iodevice` -- Determines what I/O device data will first be written to

MPI_hints for data sieving

- Hints for Data Sieving:
  - `ind_rd_buffer_size` controls the size (in bytes) of the intermediate buffer used when performing data sieving during read operations.
  - `ind_wr_buffer_size` Controls the size (in bytes) of the intermediate buffer when performing data sieving during write operations.
  - `romio_ds_read` determines when ROMIO will choose to perform data sieving for read. Valid values are enable, disable, or automatic.
  - `romio_ds_write` Determines when ROMIO will choose to perform data sieving for write. Valid values are enable, disable, or automatic.

How to use MPI-hints: procedure

- Create an info object with `MPI_Info_create`
- Set the hint(s) with `MPI_Info_set`
- Pass the info object to the I/O layer (through MPI_File_open, MPI_File_set_view or MPI_File_set_info)
- Free the info object with `MPI_Info_free` (can be freed as soon as passed)
- Do the I/O operations (`MPI_File_write_all`...)

Which values available here ?

```
/* query the default values of hints being used */
#include "mpi.h"
#include <stdio.h>

int main(int argc, char **argv)
{
    int i, nkeys, flag, rank;
    MPI_File fh;
    MPI_Info info_used;
    char key[MPI_MAX_INFO_KEY], value[MPI_MAX_INFO_VAL];

    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    MPI_File_open(MPI_COMM, “file",MPI_MODE_CREATE | MPI_MODE_RDWR,    MPI_INFO_NULL, &fh);

    MPI_File_get_info(fh, &info_used);
    MPI_Info_get_nkeys(info_used, &nkeys);

    for (i=0; i<nkeys; i++) {
   MPI_Info_get_nthkey(info_used, i, key);
   MPI_Info_get(info_used, key, MPI_MAX_INFO_VAL  value, &flag);
   printf("Process %d, Default: key = %s, value = %s\n", rank, key, value);
    }
    MPI_File_close(&fh);
    MPI_Info_free(&info_used);
    MPI_Finalize();
    return 0;
}
```

C3HPC openmpi hints..

```
[exact@master provided_code]$ module load openmpi
[exact@master provided_code]$ mpicc MPI_Hints.c
[exact@master provided_code]$ ./a.out

Process 0, Default: key = cb_buffer_size, value = 16777216
Process 0, Default: key = romio_cb_read, value = automatic
Process 0, Default: key = romio_cb_write, value = automatic
Process 0, Default: key = cb_nodes, value = 1
Process 0, Default: key = romio_no_indep_rw, value = false
Process 0, Default: key = romio_cb_pfr, value = disable
Process 0, Default: key = romio_cb_fr_types, value = aar
Process 0, Default: key = romio_cb_fr_alignment, value = 1
Process 0, Default: key = romio_cb_ds_threshold, value = 0
Process 0, Default: key = romio_cb_alltoall, value = automatic
Process 0, Default: key = ind_rd_buffer_size, value = 4194304
Process 0, Default: key = ind_wr_buffer_size, value = 524288
Process 0, Default: key = romio_ds_read, value = automatic
Process 0, Default: key = romio_ds_write, value = automatic
Process 0, Default: key = cb_config_list, value = *:1
```

Test hints on IOR

- IOR allows you to check (-H option) MPI_HINTS
- It also allows to set to set MPI-IO hints at runtime by parsing by parsing environment variables whose name must have the following prefix IOR_HINT__ MPI__ and as suffix the name of a valid MPI-IO hint.
- Example:

  export IOR_HINT_MPI__romio_cb_write=enable

  - activate MPI-IO hint romio cb write (collective buffer in writing mode).

Optimize IOR throughput by means of MPI_HINTS

UNIVERSITY OF TRIESTE

MASTER'S DEGREE THESIS

An I/O analysis of HPC workloads on CephFS and Lustre

Supervisor:
Dr. Stefano COZZINI

Author:
Alberto CHIUSOLE

Conclusions: a few tips

Guidelines for Achieving High I/O Performance

- Use fast file systems, not NFS-mounted home directories
- Do not perform I/O from one process only
- Make large requests wherever possible
- To access data that are noncontiguous:
- Create derived datatypes
- Define file views
- Use the collective I/O functions

Best practices for I/O; Do not write or read!

- Write/read only what is necessary and when needed/useful
- Write/read as infrequently as possible (group small operations)
- Reduce accuracy (write in single precision, for example)
- Recalculate when it’s faster

Some more tips/suggestions

- Open files in the correct mode.
  - If a file is only intended to be read, it must be opened in read-only mode because choosing the right mode allows the system to apply optimizations and to allocate only the necessary resources.
- Write/read arrays/data structures in one call rather than element per element.
  - Not complying with this rule will have a significant negative impact on the I/O performance.
- Do not open and close files too frequently because it involves many system operations.
  - The best way is to open the file the first time it is needed and to close it only if its use is not necessary for a long enough period of time.
- Limit the number of simultaneous open files because for each open file, the system must assign and manage some resources.
- Do make flushes (drain buffers) only if necessary. Flushes are expensive operations.