Towards an I/O Tracing Framework Taxonomy

Andy Konwinski
John Bent, James Nunez, Meghan Quist
LA-UR-07-7660

Los Alamos National Laboratory

Overview

- Motivation, background
- Survey several I/O Tracing frameworks
- Define a taxonomy
  - Identify features
- Use taxonomy to classify and compare tracing frameworks

Motivation for a Taxonomy

- LANL Commitment to release I/O traces to HPC research community.
- What tool to use?
- Use existing or build our own?
- **Need a way to compare tools.**

Checkpoint

- Motivation, background
- Survey several I/O Tracing frameworks
- Define a taxonomy
  - Identify features
- Use taxonomy to classify and compare tracing frameworks

I/O Tracing Framework Survey

- Tracefs
- //TRACE (“Parallel Trace”)
- Introducing LANL-Trace

Tracefs - Overview

- Stackable File System
- Kernel module
- Advantages
  - Many advanced features (anonymization, compression, …)
  - Portable
- Disadvantages
  - Doesn’t run “out of the box” with parallel FS
  - Have to run as Root, load kernel module
  - Does not trace mpi calls or dependencies

//TRACE

- System call interposition
- Advantages
  - Focus on replayable traces
  - Built with distributed applications in mind
  - Control over time-accuracy trade-off (via sampling)
- Disadvantages
  - Potentially high overhead tracing time
  - Highly focused on replay
    - fewer features
    - Less granularity control

LANL-Trace

- Built our own tool
- Wrappers around popular `strace` and `ltrace`
- Advantages
  - Simple, built into linux, no significant installation
  - Easy to use
- Disadvantages
  - High overhead tracing time (because of `ltrace`)

More on LANL-Trace

- Unique opportunity to profile LANL-Trace as we build it.
- What are outputs
- What is overhead

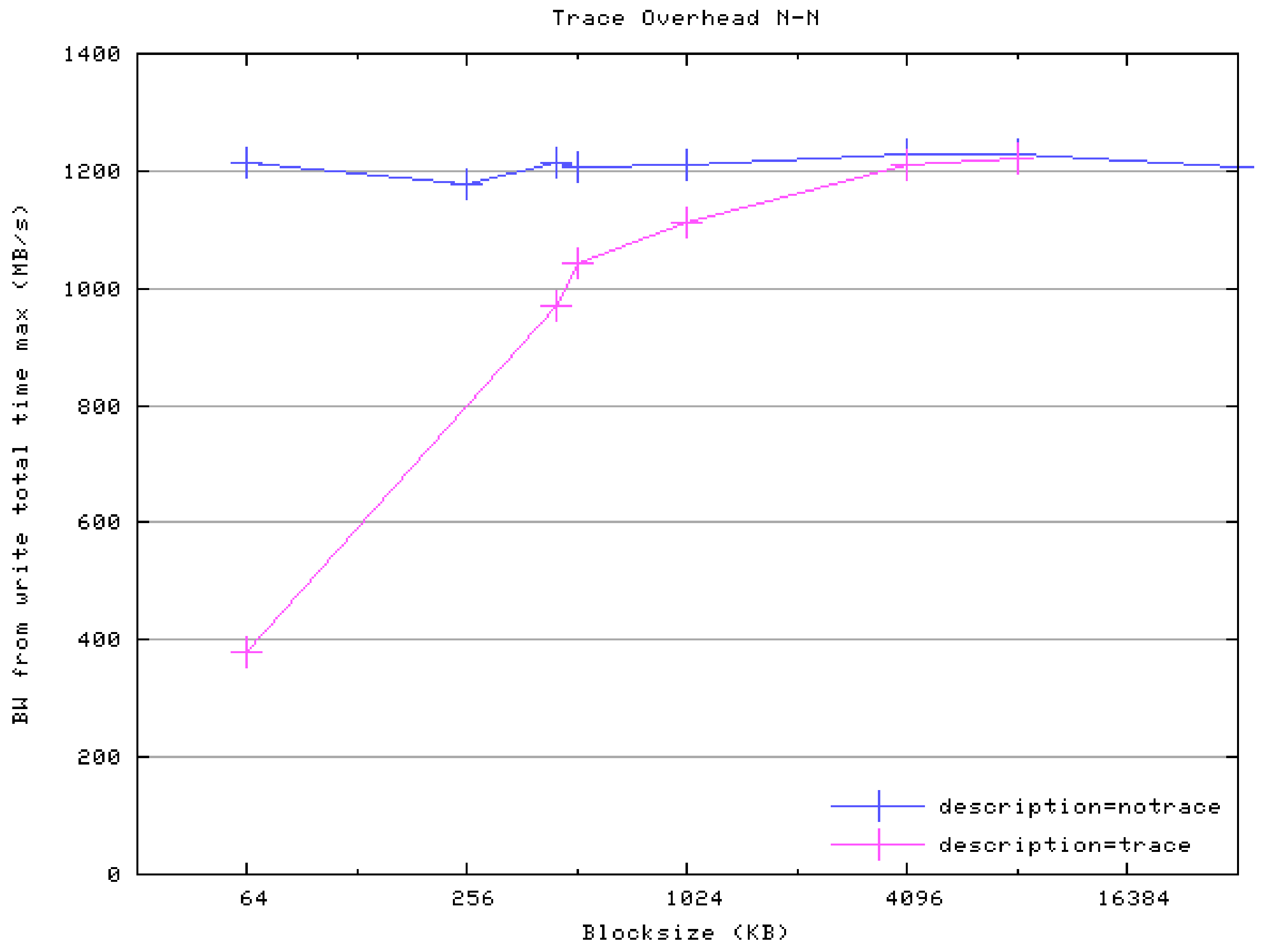
LANL-Trace :: Output

- Raw `ltrace` output
- Drift and skew timing data
- Function call summary count

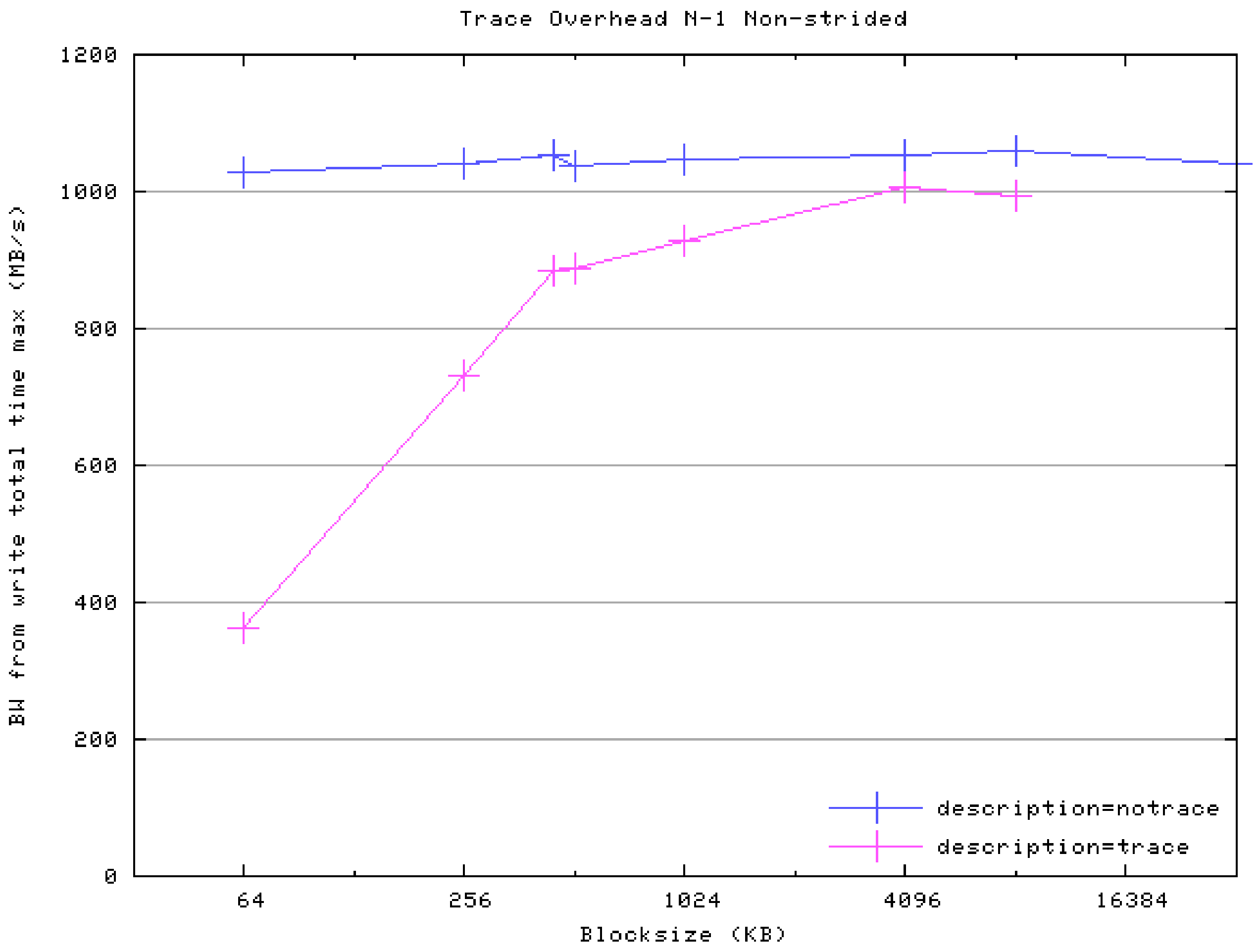
LANL-Trace :: Measuring Bandwidth Overhead

- Synthetic application, mpi_io_test
- 32 nodes, Linux 2.6.14
- Interconnect: ethernet gige
- MPI library: mpich 1.2.6
- One run for each:
  - N-to-N
  - N-to-1 strided
  - N-to-1 non-strided

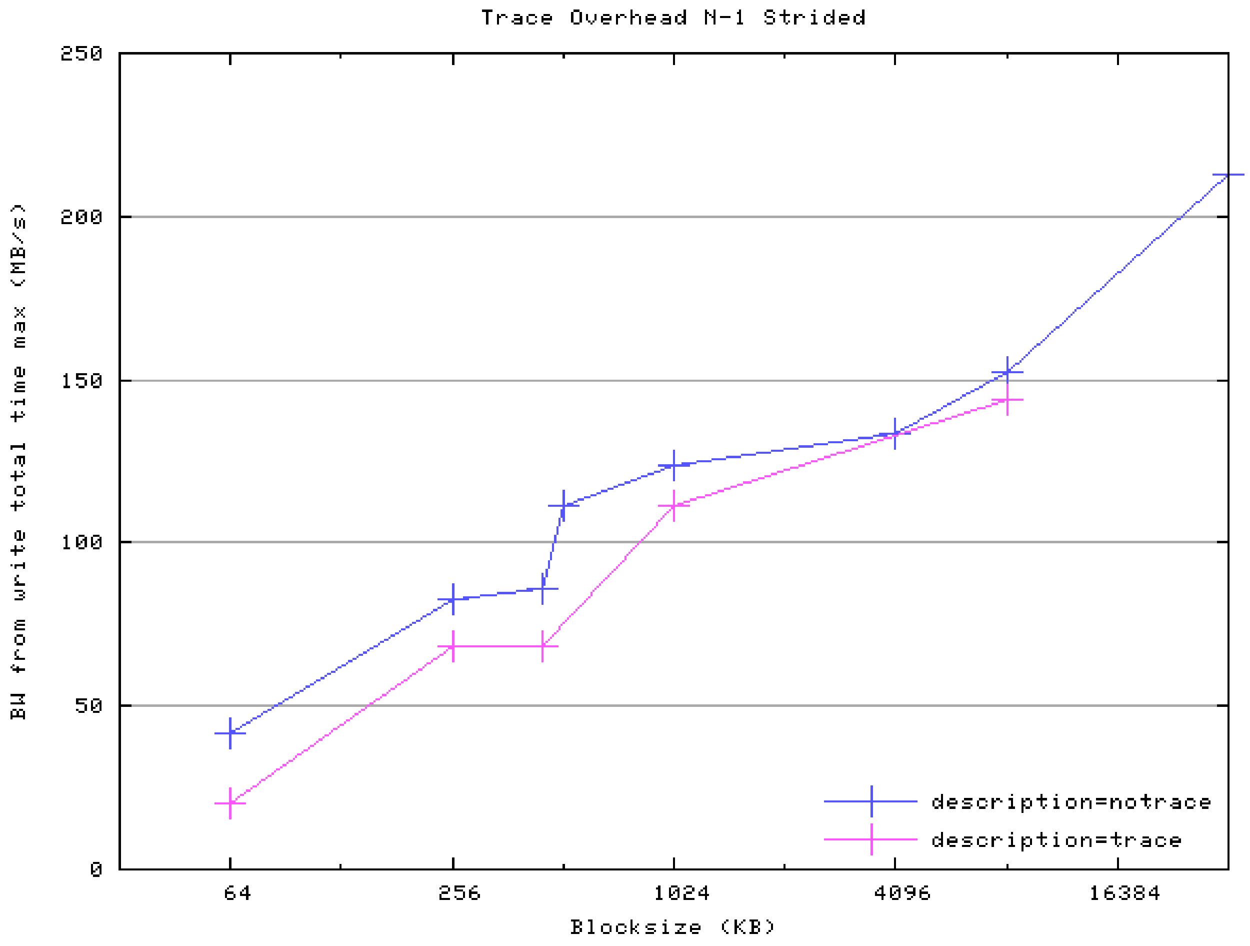
LANL-Trace Overhead N-N



LANL-Trace Overhead N-to-1 Non Strided



LANL-Trace Overhead N-1 Strided



Checkpoint

- Motivation - background
- Survey several I/O Tracing frameworks
- Define a taxonomy
  - Identify features
- Use taxonomy to classify and compare tracing frameworks

Why build a Taxonomy?

- Identify similarities and differences between frameworks
- Identify trade-offs
  - Features
  - Overheads
- Enable informed decisions:
  - Should we build our own?
  - What are the “costs” of using a currently existing one?
  - Which one should we use?

Target Users of Taxonomy

- Tracing Framework Consumers
  - application developers - Debugging
  - End users - Optimizing
  - System Administrators - Installing & maint
  - System operators - Performance monitors
  - Researchers - sharing (and all of above)
- Tracing Framework Developers
  - Guide future development
  - What is in demand
  - Where are gaps in current TF domain?

The Taxonomy
Qualitative Features

- Parallel file system compatibility
- Ease of installation
- Ease of use
- Anonymization
- Event types
- Control of trace granularity
- Replayable trace generation
- Trace replay fidelity
- Reveals Dependencies
- Intrusive vs. Passive
- Analysis tools
- Trace data format

LANL-Trace Quantitative Features

- Bandwidth overhead
- Elapsed time overhead

-- Taxonomy --
Full Summary Table

Feature	**<I/O Tracing Framework Name>**
Anonymization	[None or 1 (Simple) thru 5 (V. Advanced)]
Events types	[Systems calls, library calls, FS events]
Control of trace granularity	[Yes or No]
Replayable trace generation	[Yes or No]
Trace replay fidelity	Describe experiment results
Reveals dependencies	[Yes or No]
Intrusive vs. Passive	[1 (V. Passive), thru 5 (V. Intrusive)]
Analysis tools	[Yes or No]
Trace data format	[Binary or Human readable]
Tracing time overhead	Describe experiment results

Checkpoint

- Motivation - background
- Survey several I/O Tracing frameworks
- Defining a taxonomy
  - Identify features
- Use taxonomy to classify and compare tracing frameworks

Taxonomy Comparison Table

Feature	**LANL Trace**	**Tracefs**	**//TRACE**
Parallel file system compatibility	Yes	No	Yes
Ease of installation and use	2 (Easy)	4 (Difficult)	2 (Easy)
Anonymization	No	4 (Advanced)	No
Events types	Systems calls, library calls	File system operations	I/O System calls
Control of trace granularity	1 (Simple)	5 (V. Advanced)	Yes
Replayable trace generation	No	No	Yes
Trace replay fidelity	N/A	N/A	As low as 6% [1]
Reveals dependencies	No	No	Yes
Intrusive vs. Passive	1 (Passive)	1 (Passive)	1 (Passive)
Analysis tools	No	No	No
Trace data format	Human readable	Binary	Human readable
Tracing time overhead	24% - 200%+	[2]12.4%	N/A

Conclusions

- Taxonomy provide common language
  - Users to build shopping list
  - Developers to build feature lists
  - Both to find each other
- Most tracing performance highly variable by I/O access pattern
  - LANL-Trace experiment

Future Work

- Classify more tracing mechanisms
  - A few in the queue right now
- Expand taxonomy’s feature dictionary
  - Secondary features, e.g. if a TF generates replayable traces, are they accurate?
- Explore the overhead dimension.
  - Right now too apples to oranges
- Expand the Taxonomy beyond I/O to other tracing and logging tools
- Towards a common distributed application tracing API

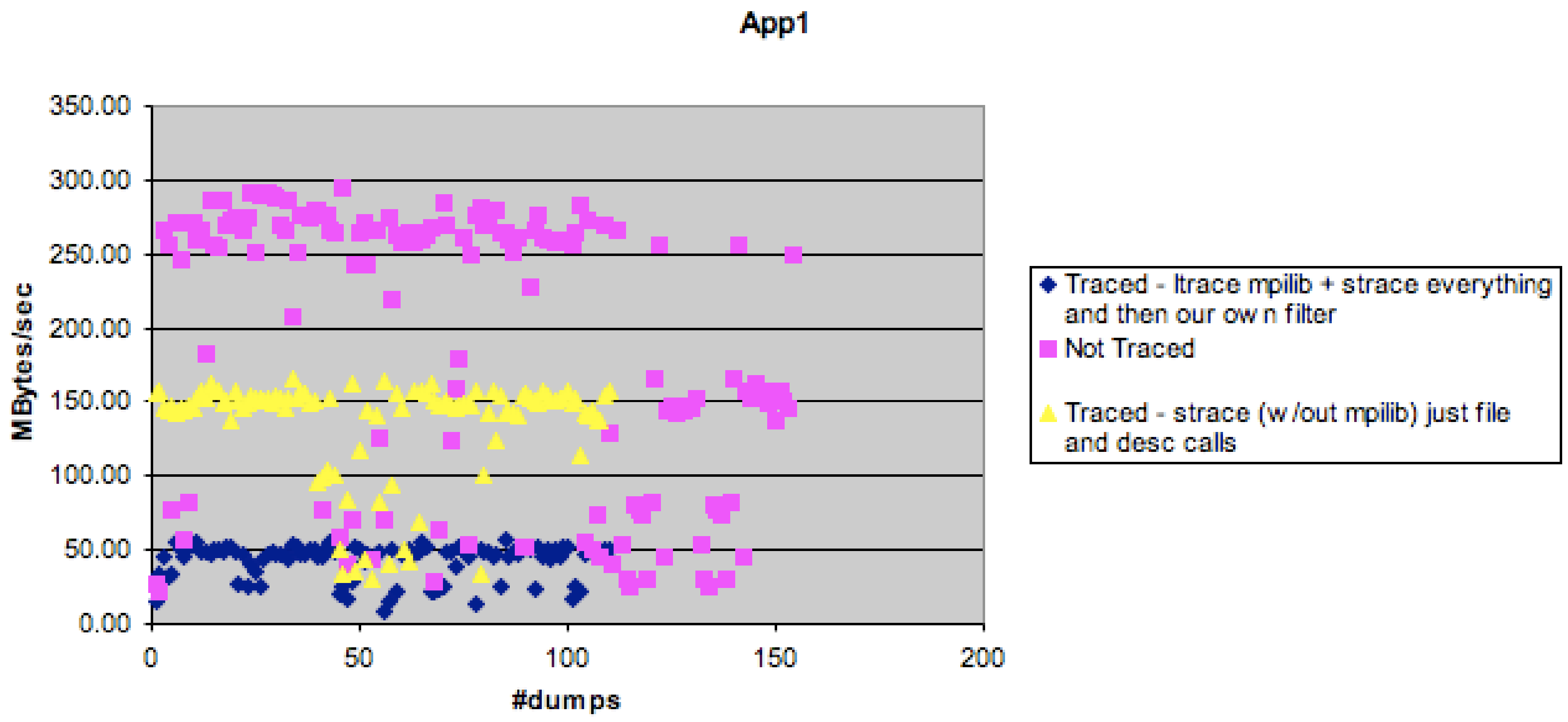
Questions?

Our Shopping List

- Parallel workloads
- Low elapsed time tracing overheads
  - V. large applications
- Workload flexibility (synthetic/non, n->1, n->n)
- High fidelity replays

LANL-Trace bandwidth on a *real code*

- Physics code (Shockwave), N->1, Strided



Case Study :: LANL-Trace

Feature	**LANL-Trace**
Parallel file system compatibility	Yes
Ease of installation and use	2 (Easy)
Anonymization	1 (Simple)
Events types	Systems calls, library calls
Control of trace granularity	1 (Simple)
Replayable trace generation	No
Trace replay fidelity	N/A
Reveals dependencies	No
Intrusive vs. Passive	1 (Passive)
Analysis tools	No
Trace data format	Human readable
Tracing time overhead (Elapsed)	234.72% to 25.65%(N-to-N)
Tracing time overhead (Bandwidth)	5.5% to 51.3% (N-to-1 strided) 6.1% to 64.7% (N-to-1 non-strided) 0.6% to 68.6% (N-to-N)

LANL-Trace Output :: Timings (capture skew and drift)

```
# Barrier before /home2/johnbent/Testing/mpi_io_test/src/mpi_io_test.caddy.x  "-type" "1" "-strided" "1" "-size" "32768" "-nobj" "1"
7: cadillac113.ccstar.lanl.gov (10378) Entered barrier at 1159808385.170918
7: cadillac113.ccstar.lanl.gov (10378) Exited barrier at 1159808385.173167
3: cadillac117.ccstar.lanl.gov (11335) Entered barrier at 1159808385.166396
3: cadillac117.ccstar.lanl.gov (11335) Exited barrier at 1159808385.168893
5: cadillac115.ccstar.lanl.gov (10373) Entered barrier at 1159808385.168842
5: cadillac115.ccstar.lanl.gov (10373) Exited barrier at 1159808385.171370
6: cadillac114.ccstar.lanl.gov (10315) Entered barrier at 1159808385.168138
6: cadillac114.ccstar.lanl.gov (10315) Exited barrier at 1159808385.170176
4: cadillac116.ccstar.lanl.gov (10272) Entered barrier at 1159808385.167178
4: cadillac116.ccstar.lanl.gov (10272) Exited barrier at 1159808385.169087
2: cadillac118.ccstar.lanl.gov (9349) Entered barrier at 1159808385.169788
2: cadillac118.ccstar.lanl.gov (9349) Exited barrier at 1159808385.172046
1: cadillac119.ccstar.lanl.gov (16609) Entered barrier at 1159808385.161409
1: cadillac119.ccstar.lanl.gov (16609) Exited barrier at 1159808385.164020
0: cadillac110.ccstar.lanl.gov (23522) Entered barrier at 1159808385.171889
0: cadillac110.ccstar.lanl.gov (23522) Exited barrier at 1159808385.174143
# Barrier after /home2/johnbent/Testing/mpi_io_test/src/mpi_io_test.caddy.x  "-type" "1" "-strided" "1" "-size" "32768" "-nobj" "1"
5: cadillac115.ccstar.lanl.gov (10436) Entered barrier at 1159808388.577588
5: cadillac115.ccstar.lanl.gov (10436) Exited barrier at 1159808388.685647
4: cadillac116.ccstar.lanl.gov (10334) Entered barrier at 1159808388.575882
…
```

LANL-Trace Output :: snippet (from a single proc)

```
…..
10:59:47.092996 MPI_File_open(92, 0x80675c0, 37, 0x80675a8, 0xbfdfe5e4 <unfinished ...>
10:59:47.093718 SYS_statfs64(0x80675c0, 84, 0xbfdfe410, 0xbfdfe410, 0xbd3ff4) = 0 <0.011131>
10:59:47.105818 SYS_open("/etc/hosts", 0, 0666)  = 3 <0.000034>
10:59:47.105913 SYS_fcntl64(3, 1, 0, 0, 0xbd3ff4) = 0 <0.000017>
10:59:47.105986 SYS_fcntl64(3, 2, 1, 1, 0xbd3ff4) = 0 <0.000016>
10:59:47.106055 SYS_fstat64(3, 0xbfdfde6c, 0xbd3ff4, 0x8068010, 8192) = 0 <0.000018>
10:59:47.106124 SYS_mmap2(0, 4096, 3, 34, -1)    = 0xb7f48000 <0.000024>
10:59:47.106199 SYS_read(3, "# Do not remove the following li"..., 4096) = 4096 <0.000061>
10:59:47.106461 SYS_read(3, "llac55 pink-cadillac55 pc55\n10.1"..., 4096) = 4096 <0.000032>
10:59:47.106683 SYS_read(3, "0\n10.128.204.111 cadillac111.ccs"..., 4096) = 4096 <0.000020>
10:59:47.106784 SYS_close(3)                     = 0 <0.000019>
10:59:47.106842 SYS_munmap(0xb7f48000, 4096)     = 0 <0.000031>
10:59:47.108236 SYS_umask(022)                   = 077 <0.000016>
10:59:47.108290 SYS_umask(077)                   = 022 <0.000015>
10:59:47.108352 SYS_open("/panfs/REALM1/scratch/johnbent/O"..., 32832, 0600) = 3 <0.000745>
10:59:47.109189 SYS_close(3)                     = 0 <0.000063>
10:59:47.109310 SYS_open("/panfs/REALM1/scratch/johnbent/O"..., -2147450814, 0600) = 3 <0.000564>
10:59:47.110912 <... MPI_File_open resumed> )    = 0 <0.017855>
……
```

LANL-Trace :: Output Function call summary (from a single proc)

```
#                        SUMMARY COUNT OF TRACED CALL(S)
# Function Name                     Number of Calls            Total time (s)
# ===========================================================================
MPI_Info_get_nkeys                                1                  0.000056
MPI_Init                                          1                  1.730996
SYS_mmap2                                        24                  0.000495
…

#                      SUMMARY COUNT OF NON-TRACED CALL(S)
# Function Name                     Number of Calls            Total time (s)
# ===========================================================================
SYS_getuid32                                      1                  0.000016
SYS_rt_sigaction                                 70                  0.001235
…

#            SUMMARY COUNT OF CALLS WITHIN 29 MPI_Barrier CALL(S)
# Function Name                     Number of Calls            Total time (s)
# ===========================================================================
SYS__newselect                                 1832                  2.032321
SYS_ipc                                         170                  0.002903
…
```