Protect Shared Filesystem from Overload on Supercomputers

ECSS Symposium
June 19, 2018

PRESENTED BY:

Si Liu
Texas Advanced Computing Center
Team Members

Lei Huang
Texas Advanced Computing Center

Si Liu
Texas Advanced Computing Center
User Blocked By Administrators

Unhappy user: *Ooops!* My account is blocked!

User access log (Stampede2 at TACC, early 2018)

#user A, 2018-01-08, excessive MDS activity, running more than 48 tasks per node
#user B, 2018-02-15, running multiple IOR jobs and impacting other users of /scratch
#user C, 2018-03-13, beating up on the /scratch filesystem and impacting other users
#user D, 2018-04-10, causing excessive MDS activity to /work and /home1

Every few days, TACC supercomputer administrators have to temporarily block some users on TACC systems due to the filesystem issues raised by improper IO work!
Early ECSS Work

OpenFOAM

- Popular in the XSEDE community
- An open source Computational Fluid Dynamic software
- Use IOStream (C++) to read and write updated parameters throughout the application code (more than a million lines of C++)
- Separate files for each parameter, each time step, each task!

Early project in 2014

- XSEDE users from Pittsburg State University and University of Colorado at Boulder
- ECSS experts:
  Dr. Anirban Jana (Pittsburgh Supercomputing Center)
  Dr. Si Liu (Texas Advanced Computing Center)
- Months of hard work to rewrite a few IO plugins of OpenFOAM with HDF5/NetCDF format
Issues of Parallel Shared Filesystem

• Achilles' heel of HPC: filesystem is shared by all users on all nodes (even crossing multiple clusters). It is a weak point of modern HPC.

• Overloaded metadata server results in global filesystem performance degradation and even unresponsiveness.

• Many practical applications (in computational fluid dynamics, quantum chemistry, machine learning, etc.) involve non-optimal IO works raising a huge amount of IO requests in a very short period of time.

• There is no strict enforced IO resource limitation in production (e.g. metadata server throughput, bandwidth) on user level or node level.
Potential Solutions

• System level
  o A strong parallel filesystem that can handle any kind of IO requests from all users without losing efficiency
    ➢ Impractical and expensive
  o A self-protect filesystem

• Application level
  o A well-designed workflow with reasonable IO workload
    ➢ Recommended way
    ➢ Expertise required

• User level
  o Users give up planned IO work to avoid heavy IO requests
    ➢ A compromise rather than a solution
Potential Solutions

• System level
  o A strong parallel filesystem that can handle any kind of IO requests from all users without losing efficiency
    ➢ Impractical and expensive
  o A self-protect filesystem

• Application level
  o A well-designed workflow with reasonable IO workload
    ➢ Recommended way
    ➢ Expertise required

• User level
  o Users give up planned IO work to avoid heavy IO requests
    ➢ A compromise rather than a solution
  o An optimal system that makes the heavy IO work accepted
    ➢ Without rewrite the code
Lustre Architecture (NICS website)
https://www.nics.tennessee.edu/computing-resources/file-systems/lustre-architecture
Lustre Architecture (NICS website)

https://www.nics.tennessee.edu/computing-resources/file-systems/lustre-architecture
Network Attached Storage

Client node

Interconnect Fabric (10GE, InfiniBand)

File System Server

RAID Controller

Storage Device

Client node

File System Server

RAID Controller

Storage Device

GPFS Storage Topology (Virginia Tech)

https://www.slideshare.net/GabrielMateescu/sonas-44390281
Network Attached Storage

Client node

InfiniBand Fabric (10GE, Infini)

File System Server

RAID Controller

Storage Device

Storage Device

Client node

File System Server

RAID Controller

Storage Device

Storage Device

GPFS Storage Topology (Virginia Tech)

https://www.slideshare.net/GabrielMateescu/sonas-44390281
Our Proposed Solution

• Intercept IO related functions (system calls `open()`, `stat()`, `close()`, etc.) within applications and keep a record of
  o Real-time IO operation time (response time)
  o Real-time IO operation frequency (recent counts)

• Evaluate filesystem status (busy/modest used/free)
  o Responding time per operation

• Evaluate IO workloads (recent IO request count and frequency)
  o Node based or user based

• Insert decent delays when necessary
Optimal Overloaded IO Protection System (OOOPS)

- An innovative IO workload managing system that optimally controls the IO workload from the users' side.
- Automatically detect and restrict improper IO workload from supercomputer users to protect parallel shared filesystems.
Function Interception

Without OOOPS loaded

User application

write_data() {
  ...
  MPI_File_open(parameters);
  ...
}

libc version of open()
Defined in libc.so

With OOOPS loaded (LD_PRELOAD ooops library)

User application

write_data() {
  ...
  MPI_File_open(parameters);
  ...
}

OOOPS version of open()
Defined in ooops.so

libc version of open()
Defined in libc.so
Simplified open() Function in OOOPS

```c
open(arguments)
{
    call the open() function in libc;
    get_response_time();
    get_IO_request_counts();
    collect_and_analyze();

    if ( server_busy or io_frequency_high )
        sleep_for(some_time);
}
```
IO Operation Time vs IO Request Count
Average Operation Response Time

\[ \text{MAX FREQ}_{\text{open/stat}} = \frac{C}{<t_{\text{open/stat}}>} \]

C depends on
- Filesystem throughput
- System size
- Allocation proportion

Decide the IO response time threshold

---

6/19/18
Tuning Parameters for Stampede2

Stampede2 SKX

FILE_SYS_TAG_0="/scratch"
T_THRESHOLD_OPEN_0=467.97
MAX_OPEN_FREQ_0=1000
T_THRESHOLD_LXSTAT_0=247.37
MAX_STAT_FREQ_0=2000

FILE_SYS_TAG_1="/work"
T_THRESHOLD_OPEN_1=907.14
MAX_OPEN_FREQ_1=500
T_THRESHOLD_LXSTAT_1=481.52
MAX_STAT_FREQ_1=1000

FILE_SYS_TAG_2="/home1"
T_THRESHOLD_OPEN_2=317.94
MAX_OPEN_FREQ_2=1000
T_THRESHOLD_LXSTAT_2=205.43
MAX_STAT_FREQ_2=2000

Stampede2 KNL

FILE_SYS_TAG_0="/scratch"
T_THRESHOLD_OPEN_0=1198.67
MAX_OPEN_FREQ_0=500
T_THRESHOLD_LXSTAT_0=821.79
MAX_STAT_FREQ_0=800

FILE_SYS_TAG_1="/work"
T_THRESHOLD_OPEN_1=1948.61
MAX_OPEN_FREQ_1=300
T_THRESHOLD_LXSTAT_1=1206.11
MAX_STAT_FREQ_1=500

FILE_SYS_TAG_2="/home1"
T_THRESHOLD_OPEN_2=1248.75
MAX_OPEN_FREQ_2=400
T_THRESHOLD_LXSTAT_2=731.82
MAX_STAT_FREQ_2=700
How to use it now (Stampede2 at TACC)

_Ooops!_
My account has been blocked due to my early IO work.

Do not worry.
Please rerun your programs with **OOOPS**.

Load the **OOOPS** module on Stampede2

```bash
module use /work/01255/siliu/stampede2/ooops/modulefiles/
module load ooops
ibrn my-application-run  #as usual, no source code change
```
This is an MPI IO job running on 4 KNL nodes with 16 MPI tasks per node. The results are collected and plotted by REMORA.
Dynamical IO Request Control

This extra command allows users/administrators to modify parameters for individual jobs during run time

```bash
set_io_param [server_idx t_open max_open_freq t_stat max_stat_freq]
```

Different levels of request control

$ set_io_param low
$ set_io_param medium
$ set_io_param unlimited

Explicitly parameter settings

$ set_io_param open_TH open_FQ stat_TH stat_FQ

This feature has not been activated in production yet!!!
Parameter Change During Runtime
(Make it happen during the IO Runs)

This is an MPI IO job running on 4 KNL nodes with 16 MPI tasks per node. The results are collected and plotted by Remora.
OOOPS Highlights

• Convenient to HPC users
  o No source code modification at all (on users’ side)
  o Little/no workflow update (on users’ side)
  o Self-driven slowdown during the IO work when necessary

• Valuable on supercomputers
  o Protect filesystem from overloaded IO requests
  o Lightweight procedure, minimal/slight influence on performance
  o Easy to deploy system wide on an arbitrary cluster
  o Little work for system administrators
  o Dynamically control running jobs’ IO requests without interruption
Conclusion

• We developed a new tool (OOOPS) to help
  ✓ users carry out heavy IO work that is originally not allowed
  ✓ administrators protect the cluster from overload

• We enforce a fair-sharing IO resource provisioning policy on the client side practically (instead of the server side)
  ✓ Treat IOPS/Metadata server throughput as a resource (More optimization for future work)
  ✓ Increase system capacity (applications with heavy IO load)
Acknowledgement

- Junseong Heo (TACC)
- Tommy Minyard, Robert McLay, and Bill Barth (TACC)
- Stampede2 early users of OOOPS
If you or your colleagues happen to have some IO overload issues on XSEDE resource …

you may try OOOPS today!

siliu@tacc.utexas.edu

Picture from https://www.dreamstime.com