TEXAS ADVANCED COMPUTING CENTER

WWW.TACC.UTEXAS.EDU

TEXAS
The University of Texas at Austin

An Innovative Tool for IO Workload Management on Supercomputers

Lei Huang and Si Liu

HUST 2018
Nov 11, 2018

Team Members

Lei Huang
huang@tacc.utexas.edu
Texas Advanced
Computing Center

Si Liu
siliu@tacc.utexas.edu
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 a couple of weeks, TACC supercomputer administrators have to temporarily block some users on TACC systems due to the filesystem issues raised by too intense IO work!

Issues of Parallel Shared Filesystem

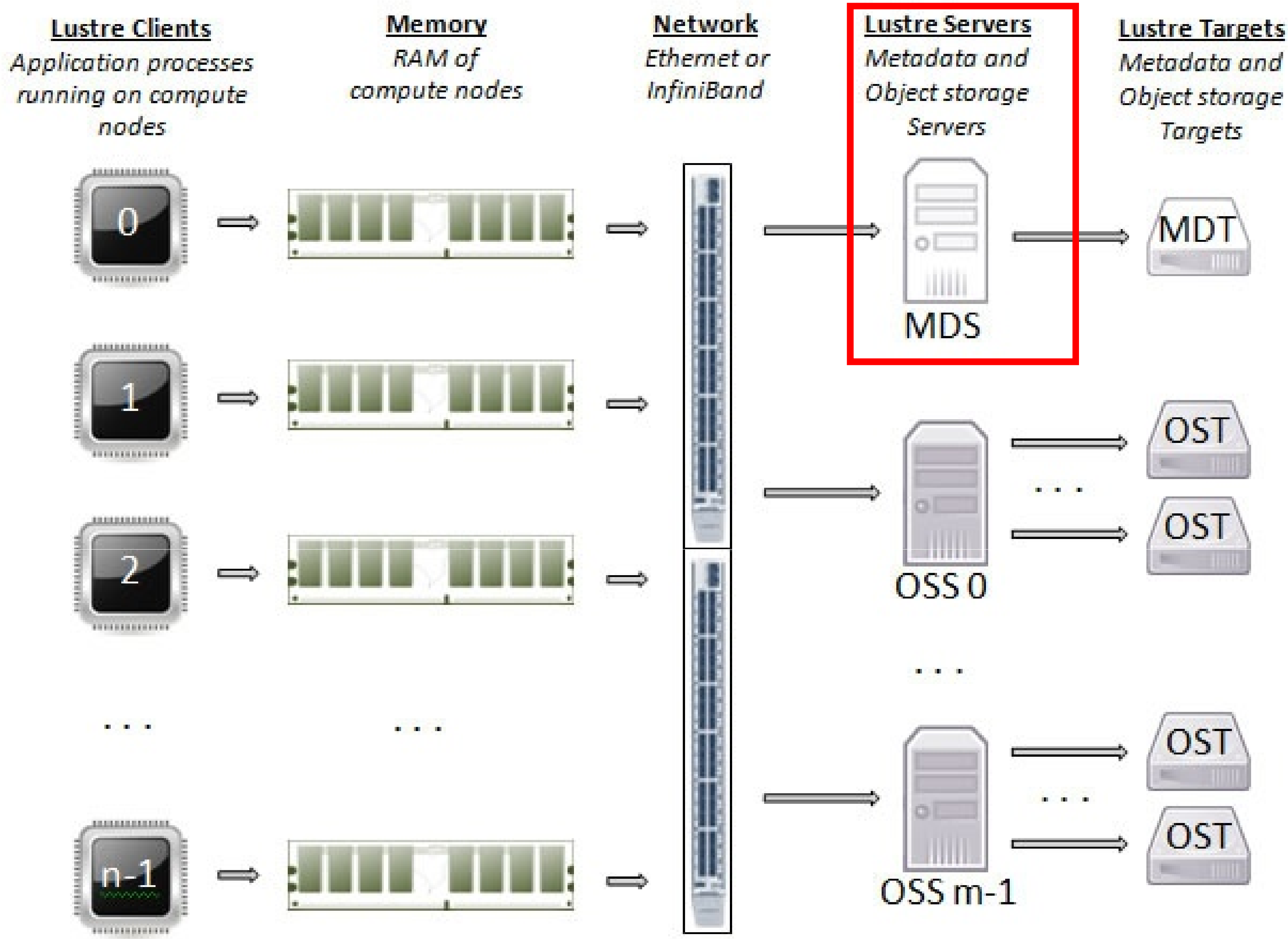
- Achilles' heel of HPC: filesystem is shared by all users on all nodes (even crossing multiple clusters). It is a weak point of modern HPC.
- Overloading metadata server results in global filesystem performance degradation and even unresponsiveness.
- Many practical applications (in computational fluid dynamics, quantum chemistry, machine learning, etc.) raise a huge amount of IO requests in a very short time.
- There is no strict enforced IO resource provisioning in production (e.g. metadata sever throughput, bandwidth) on user level or node level.

Potential Solutions

- System level
  - A strong parallel filesystem that can handle any kind of IO requests from all users without losing efficiency, e.g., upgrade hardware of MDS to achieve better IO throughput
    - Impractical, expensive or limited improvement
  - Burst buffer
    - Needs extra hardware and software, even changes in user code
- Application level
  - A well-designed workflow with reasonable IO workload
    - Recommended way
    - Expertise required
- User level
  - Users give up planned IO work to avoid heavy IO requests or decrease the number of jobs
    - A compromise rather than a solution

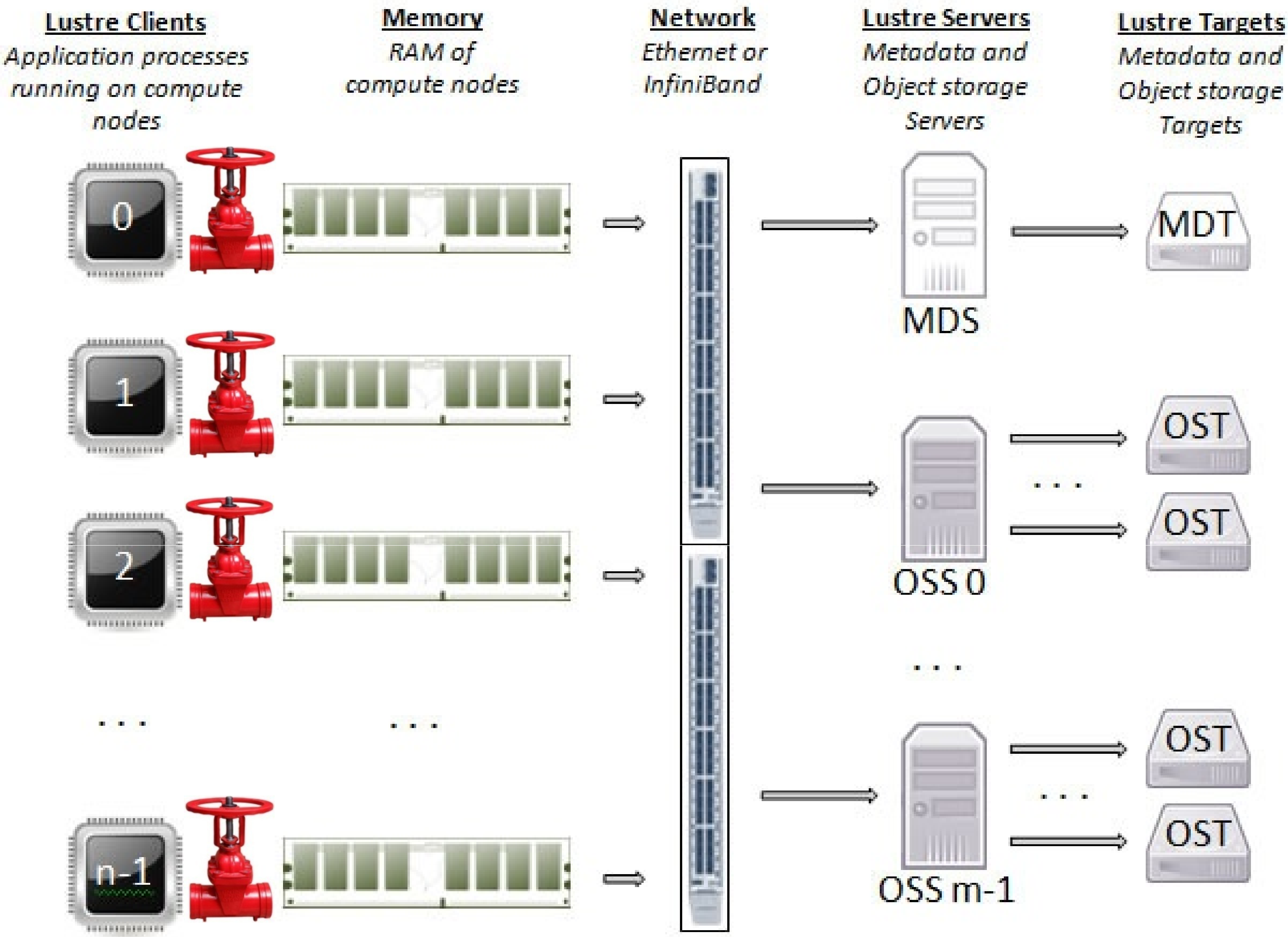
Potential Solutions

- System level
  - o A strong parallel filesystem that can handle any kind of IO requests from all users without losing efficiency, e.g., upgrade hardware of MDS to achieve better IO throughput
    - Impractical, expensive or limited improvement
  - o Burst buffer
    - Needs extra hardware and software, even changes in user code
- 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 or decrease the number of jobs
    - A compromise rather than a solution
  - o An optimal system that makes heavy IO work under control
    - Without rewriting users'code



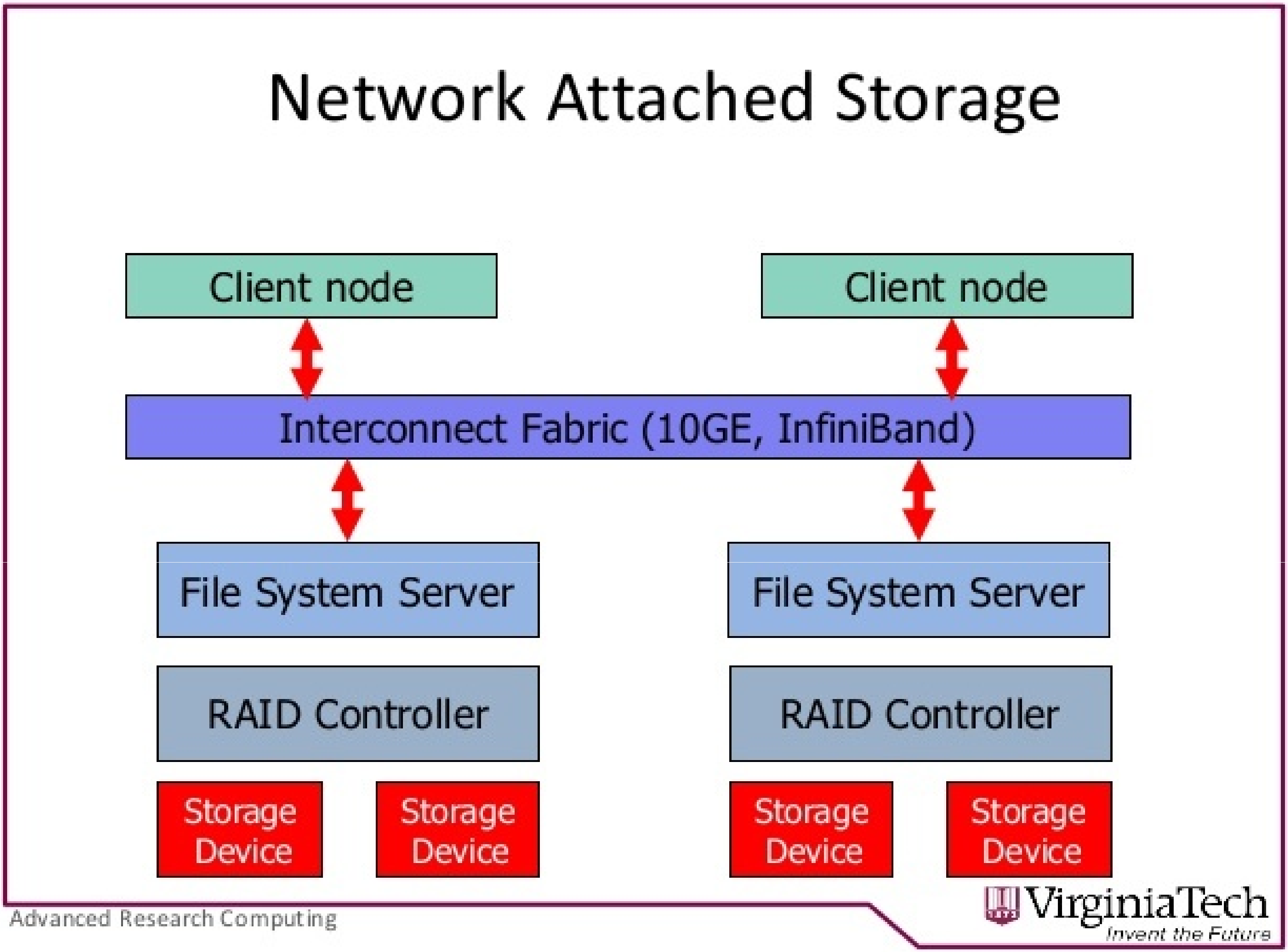
Lustre Architecture (NICS website)

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

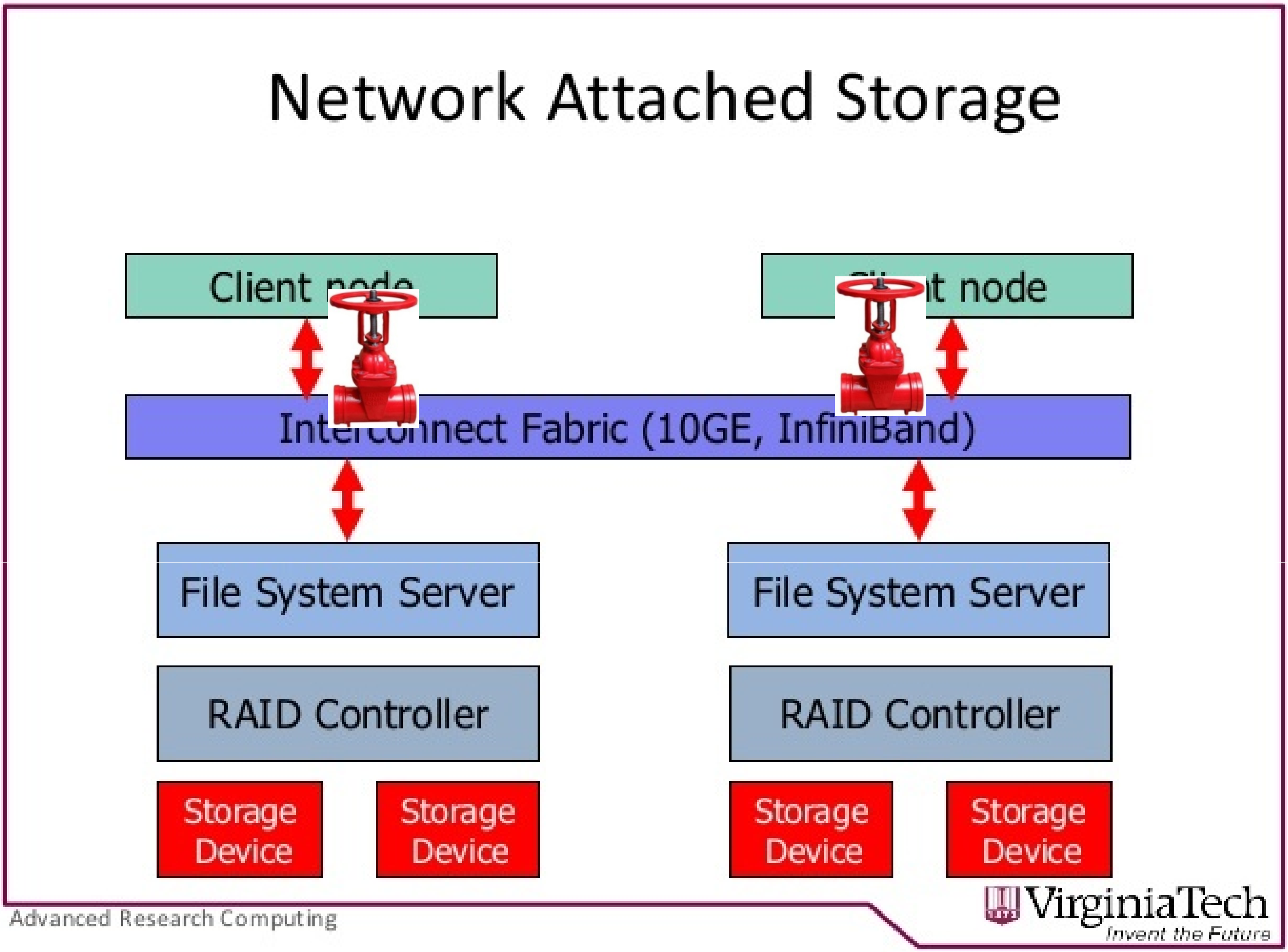


Lustre Architecture (NICS website)

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



GPFS Storage Topology (Virginia Tech)
https://www.slideshare.net/GabrielMateescu/sonas-44390281



GPFS Storage Topology (Virginia Tech)

https://www.slideshare.net/GabrielMateescu/sonas-44390281

Our Proposed User-side Solution

- Intercept IO related functions (open(), stat(), etc.) within applications and keep a record of
  - IO operation time (response time)
  - IO operation frequency (calculated from saved time stamp of recent function calls)
- Evaluate filesystem status (busy/modest used/free)
  - Responding time per operation
- Evaluate IO workloads (recent IO request frequency)
  - Node based and user based
- Insert proper 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 throttle excessive IO workload from supercomputer users to protect parallel shared filesystems.

Function Interception

Without OOOPS loaded

```
write_data() {
FILE *fOut;
fOut = fopen(name, mode);
...
}
```

```
open(name, mode, ...) {
...
}
```

User application

glibc version of open() defined in libc.so

With OOOPS loaded (LD_PRELOAD OOOPS library)

```
write_data() {
FILE *fOut;
fOut = fopen(name, mode);
...
}
```

```
open(name, mode, ...) {
...
open(name, mode, ...);
...
}
```

```
open(name, mode, ...){
...
}
```

User application

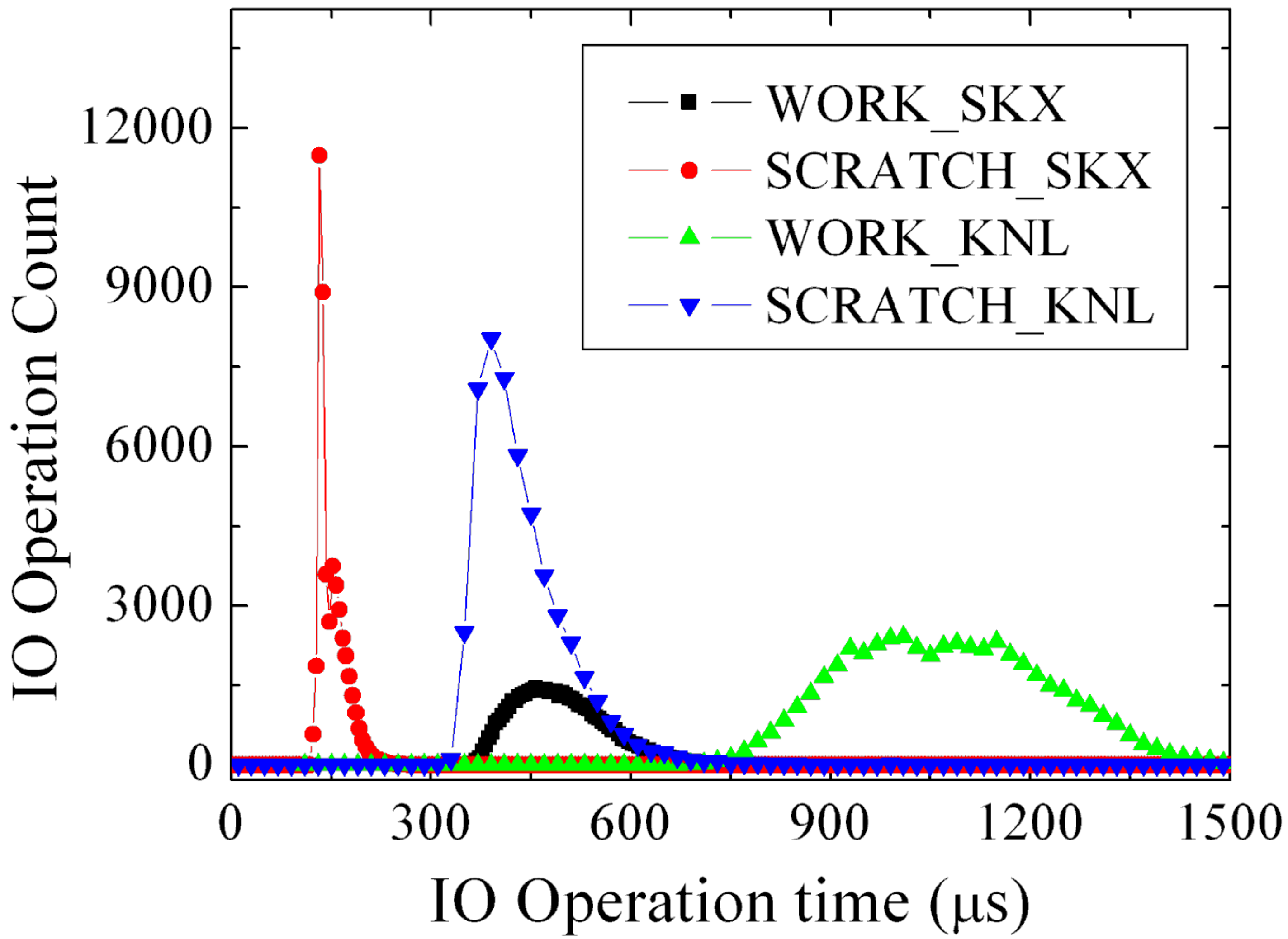
OOOPS version of open() defined in ooops.so

glibc version of open() defined in libc.so

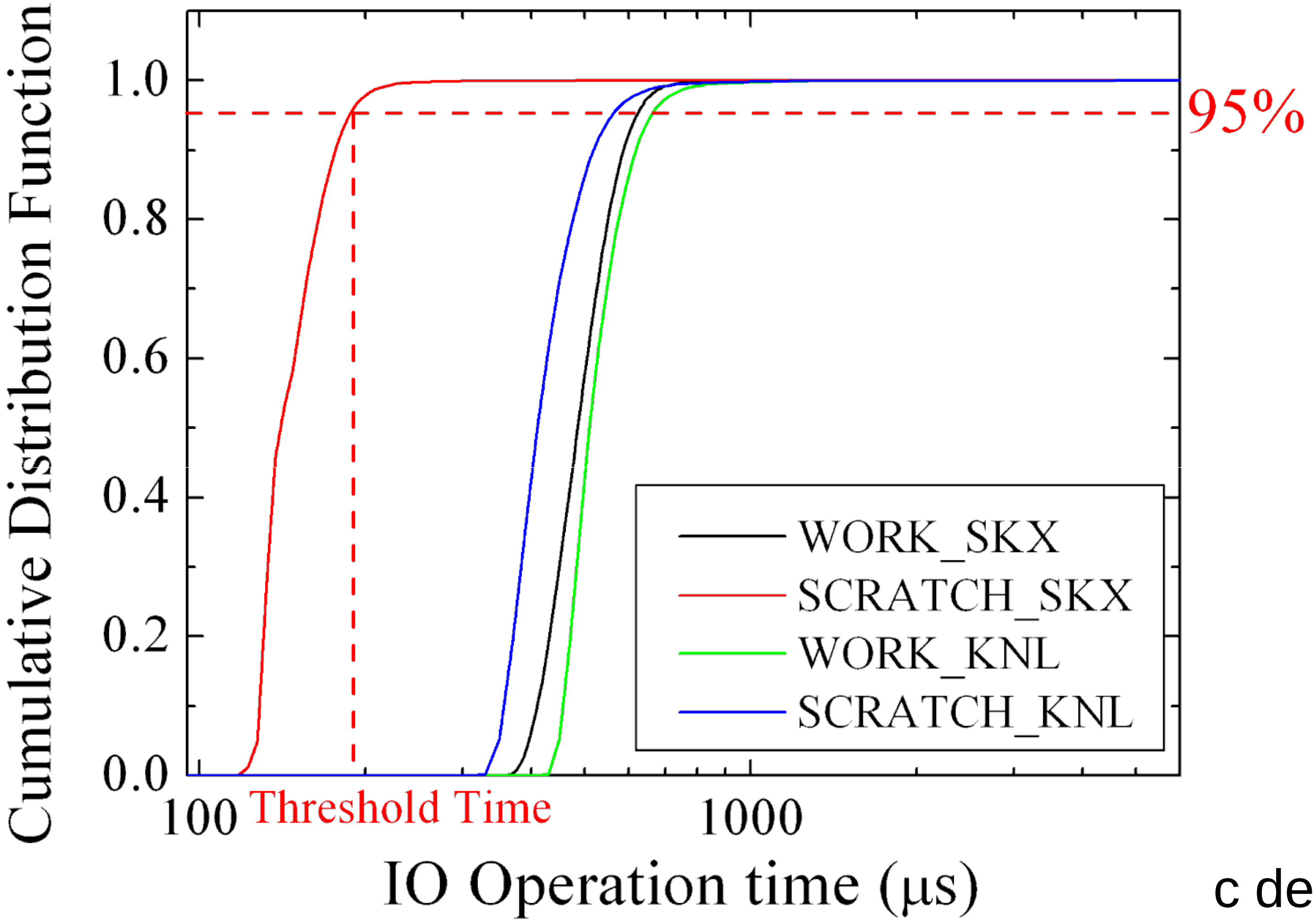
Pseudo Code in the open() in OOOPS

```
int open(name, …)
{
    call the open() function in libc;

    get_response_time_and_time_stamp();
    get_IO_request_frequency();
    evaluate_server_status_and_io_freq();

    if ( server_busy or io_frequency_too_high )
        sleep(some_time);
}
```

The Histogram of IO Operation Time



CDF of Response Time



Decide the IO response time threshold

$$MAX\_FREQ_{open/stat} = \frac{c}{< t_{open/stat} >}$$

c depends on

- Filesystem throughput
- System size
- Allocation proportion

Tentative 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

module use /work/01255/siliu/stampede2/ooops/modulefiles/
module load ooops
ibrun my-application-run   #as usual, no source code change

The Recipe to Deploy OOOPS on Other Supercomputers

Administrators:

1. Measure response time of function calls of *open*() and *stat*(), then prepare config file
2. Compile ooops.so
3. Make a module

export IO_LIMIT_CONFIG=/full_path/ooops/1.0/conf/config
export LD_PRELOAD=/full_path/ooops/1.0/lib/ooops.so

Users:

module load ooops
Run their jobs

The Recipe to Deploy OOOPS on Other Supercomputers

Administrators:

1. Measure response time of function calls of *open*() and *stat*(), then prepare config file
2. Compile ooops.so
3. Make a module

export IO_LIMIT_CONFIG=/full_path/ooops/1.0/conf/config
export LD_PRELOAD=/full_path/ooops/1.0/lib/ooops.so

Users:

module load ooops
Run their jobs

Other than on TACC resources, we also tested OOOPS on the supercomputers at NCAR and JHU.

Dynamical IO Request Control

An extra tool that allows users/administrators to modify parameters for individual jobs during run time

Explicitly parameter settings

set_io_param server_idx t_open max_open_freq t_stat max_stat_freq

Different levels of request control

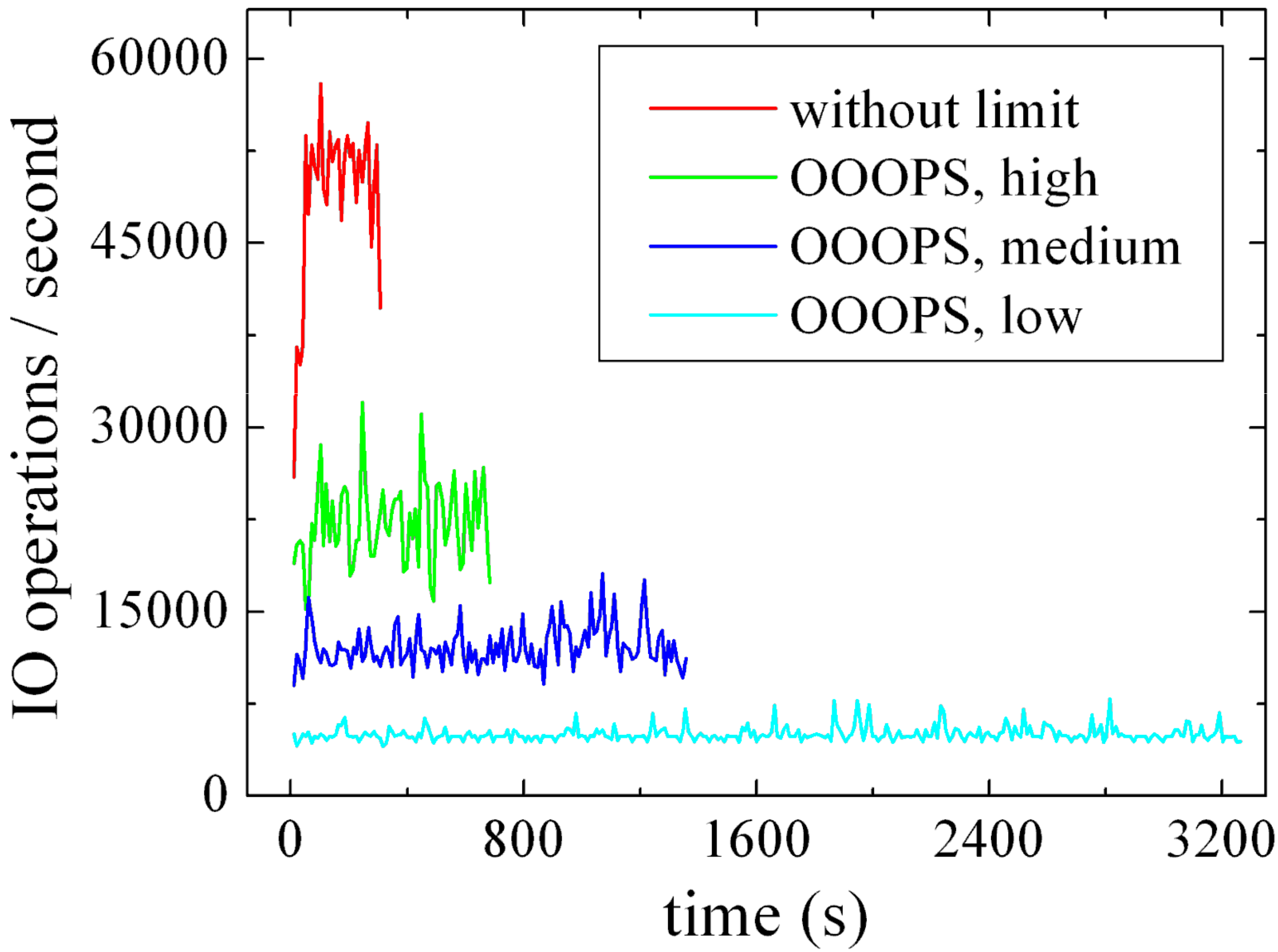
$ **set_io_param server_idx low**

$ **set_io_param server_idx medium**

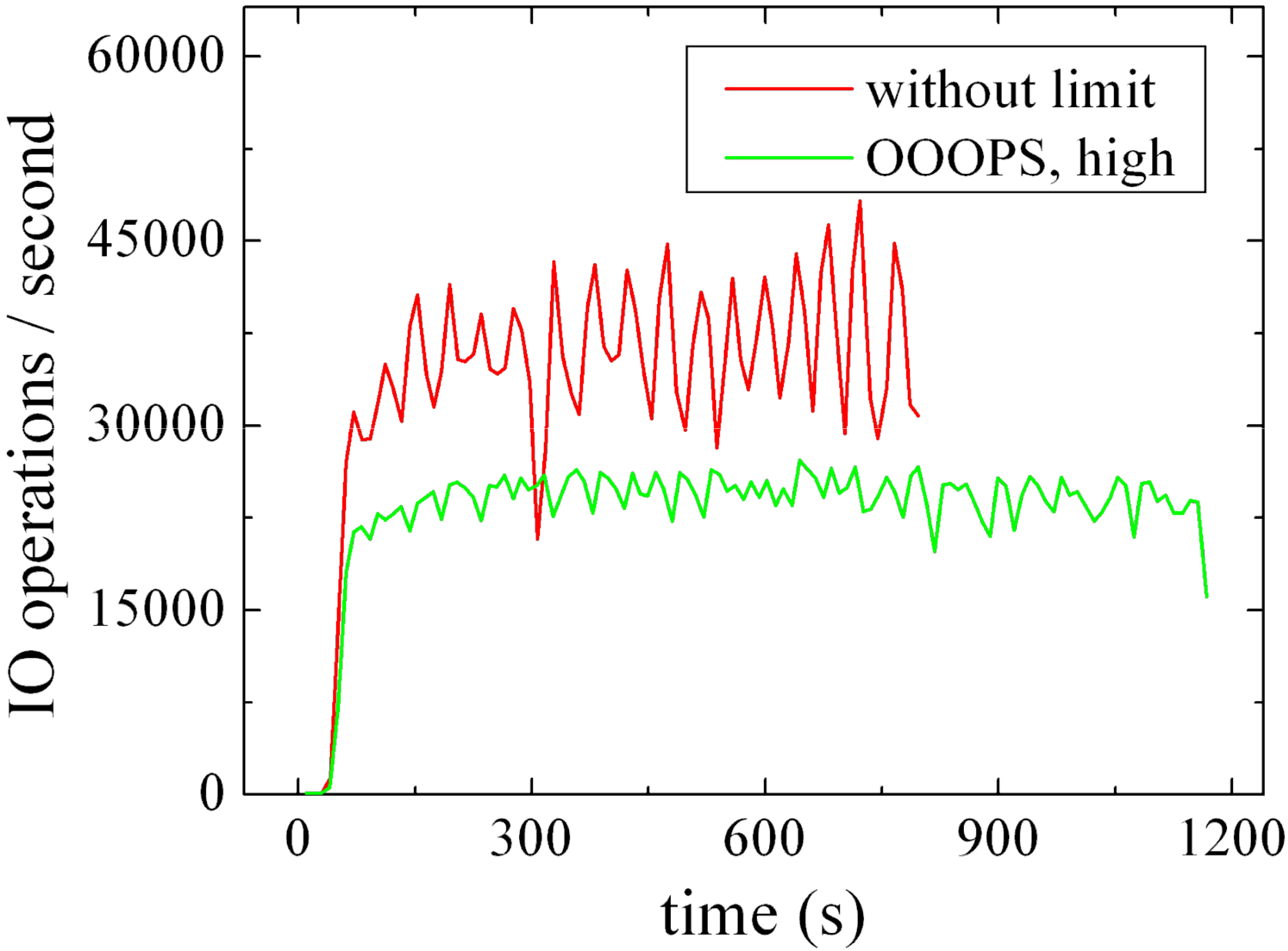
$ **set_io_param server_idx high**

$ **set_io_param server_idx unlimited**

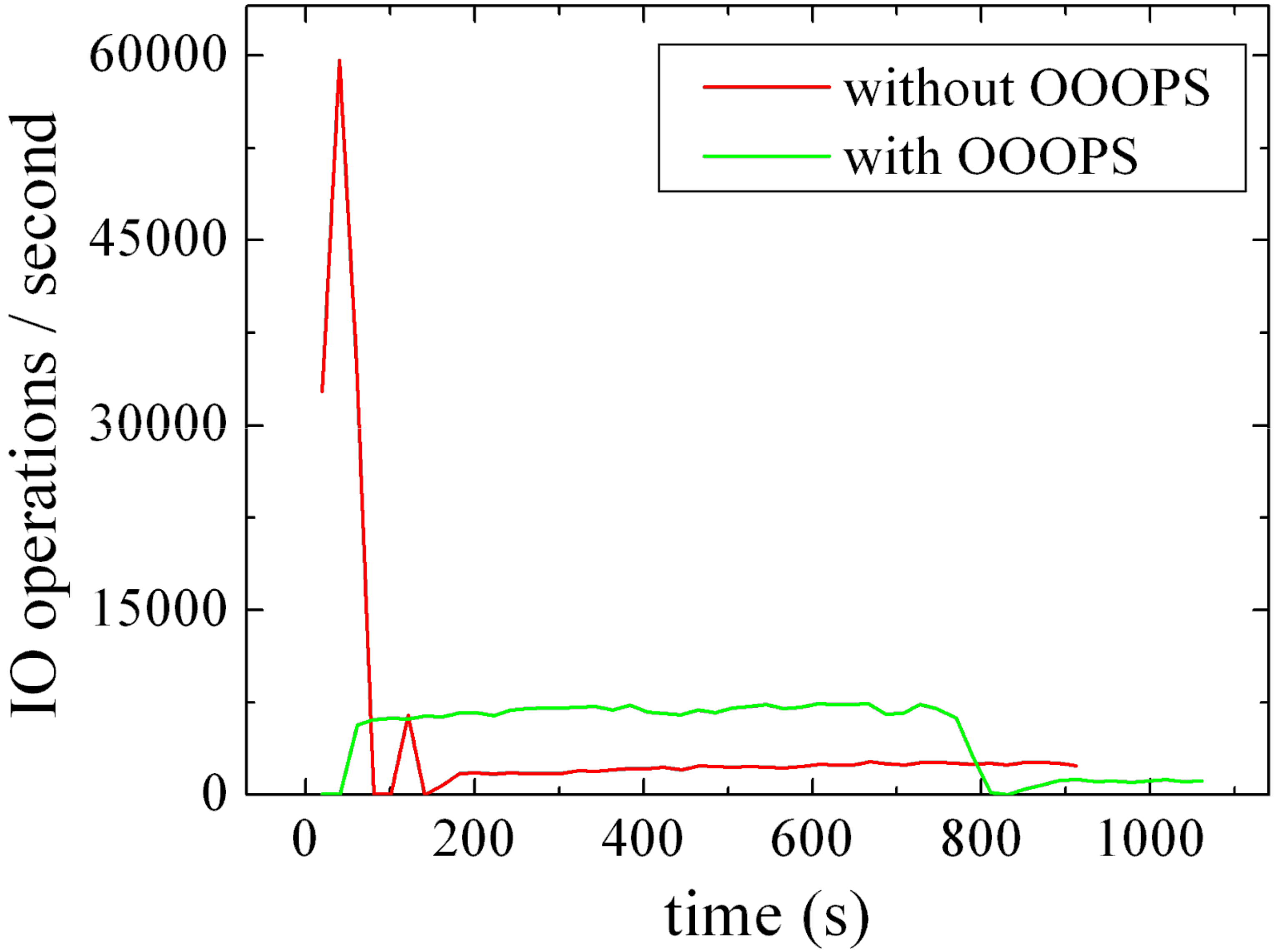
IO Requests with Different Settings



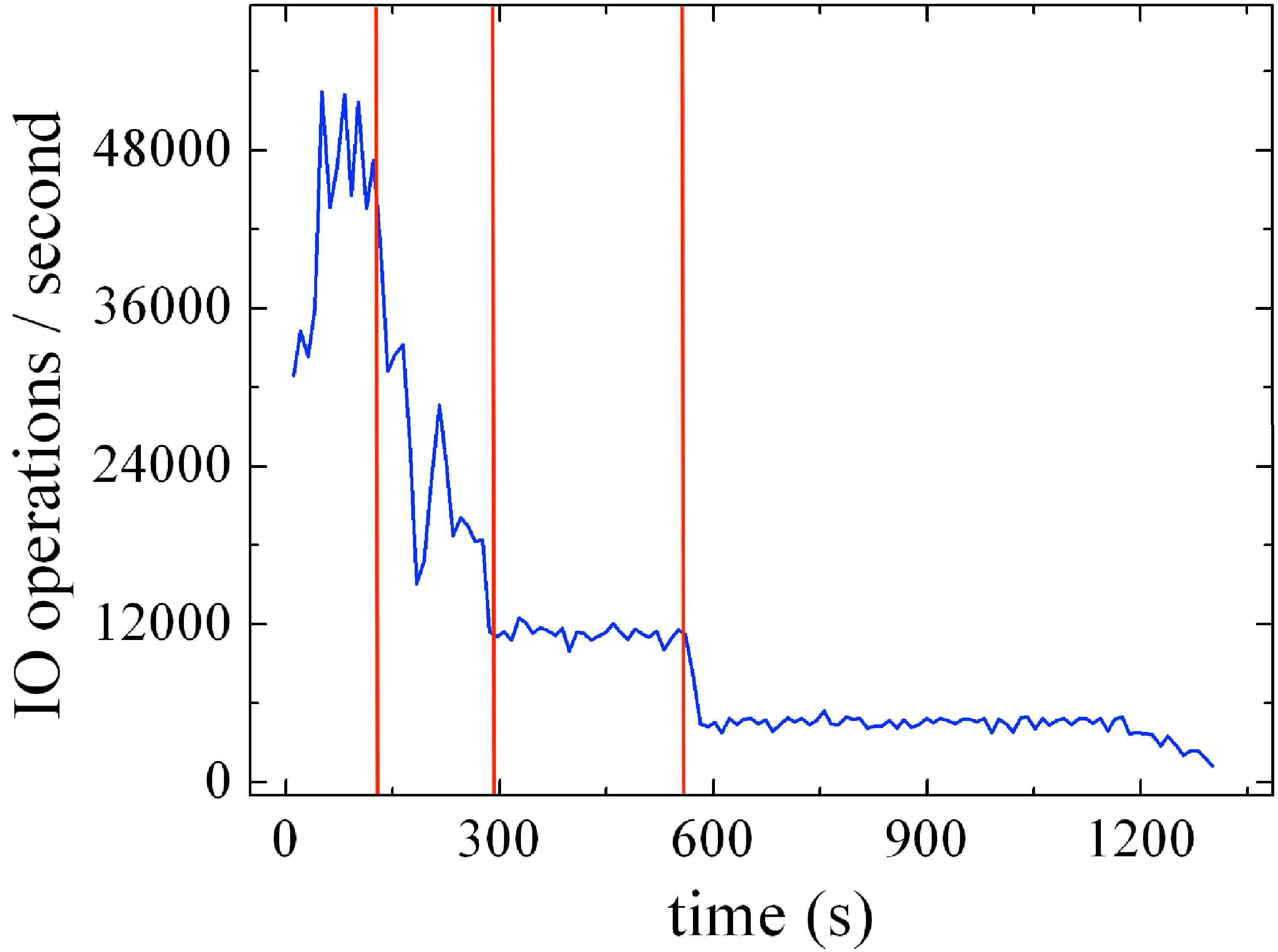
Example of Running OpenFOAM



Example of Running TensorFlow



Example of Dynamically Throttling IO Requests



OOOPS Highlights

- Convenient to HPC users
  - No source code modification at all on uses' side
  - Little/no workflow update on users' side
  - Self-driven slowdown IO work when necessary

- Valuable on supercomputers
  - Protect filesystem from overloaded IO requests
  - Little overhead: minimal/slight influence on performance except some jobs performing excessive IO work
  - Easy to deploy on an arbitrary cluster as long as file system is POSIX compliant
  - Scale up to any size of supercomputers
  - Little work for system administrators
  - Dynamically control running jobs' IO requests without interruption

Limitations

- The IO resource provisioning policy is too simple.
- OOOPS will lead to noticeable performance degradation for the jobs with very intensive IO for significant time.

Limitations

- The IO resource provisioning policy is too simple.
- OOOPS will lead to noticeable performance degradation for the jobs with very intensive IO for significant time.

  Transient file system based on local storage (memory, hard drive, SSD, etc.) based on MPI. "*fanstore: enabling efficient and scalable i/o for distributed deep learning*", Zhao Zhang and Lei Huang, et al.

Conclusion

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

- We enforce a fair-sharing IO resource provisioning policy on client side practically (instead of server side)
  - ü Treat IOPS/Metadata server throughput as a resource
  - ü Increase system capacity (applications with heavy IO load)

Acknowledgement

Colleagues at TACC

- Zhao Zhang
- Tommy Minyard
- Bill Barth
- Junseong Heo
- Robert McLay
- John Cazes

Stampede2 early users of OOOPS

Other HPC centers

- Davide Del Vento (NCAR)
- Kevin Manalo (JHU)

Picture from https://www.dreamstime.com

If you happen to have some IO jobs banned by administrators, or you are the administrators observing excessive load on your file system server,

you should try OOOPS!

huang@tacc.utexas.edu
siliu@tacc.utexas.edu