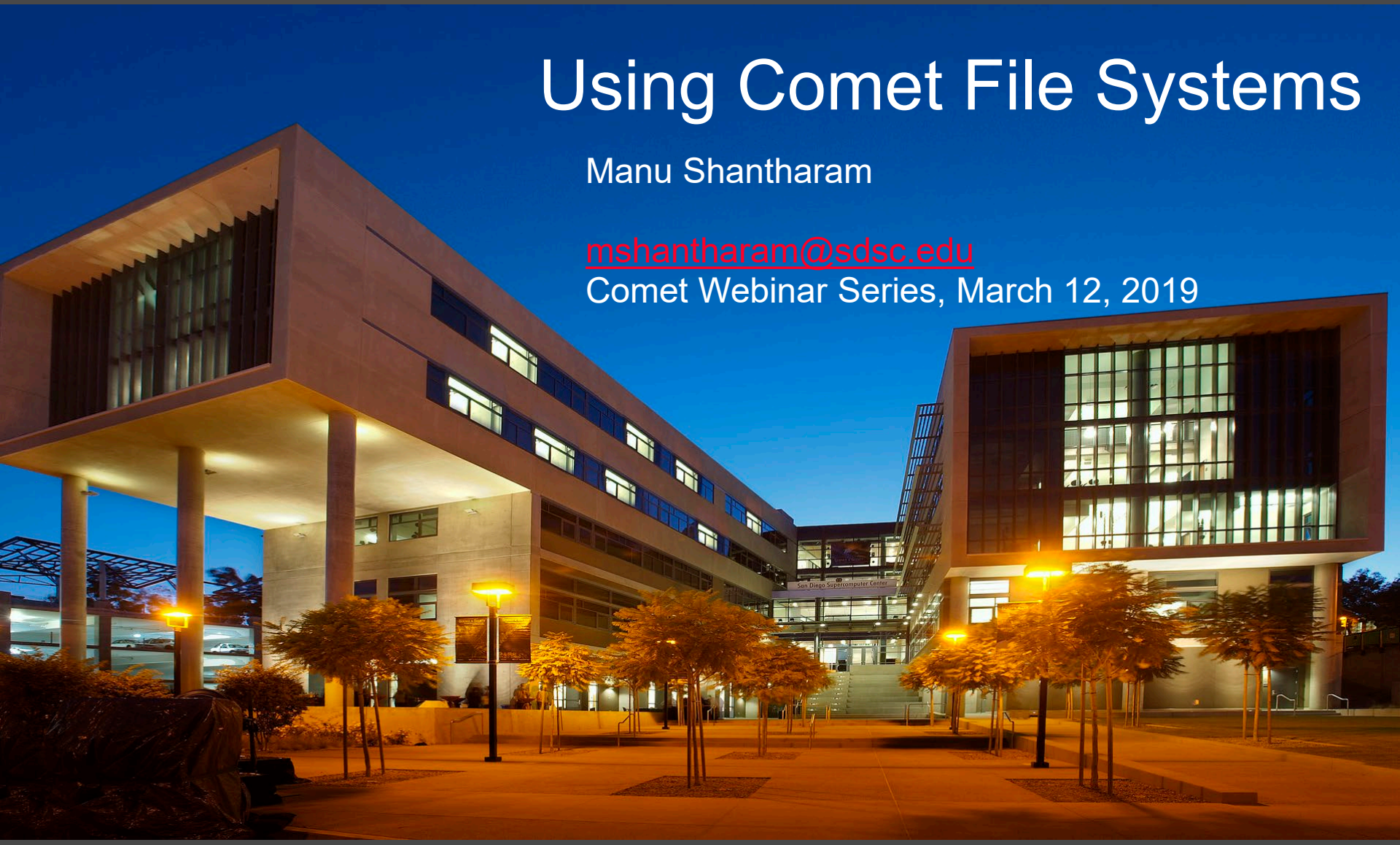
Using Comet File Systems

Manu Shantharam

mshantharam@sdsc.edu

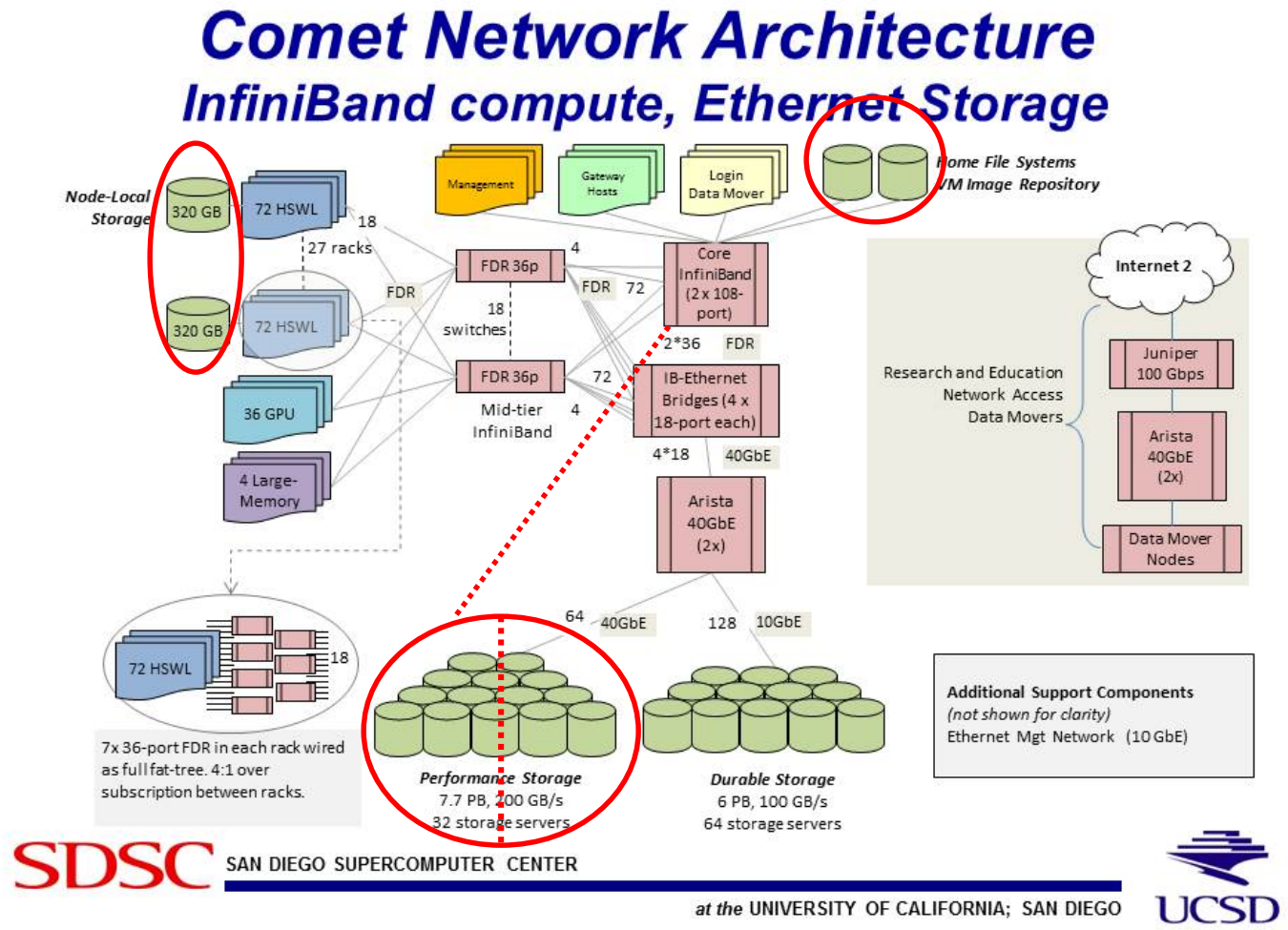
Comet Webinar Series, March 12, 2019



Why File Systems

- Out-of-core calculations generating 1000s of temporary files – genome sequencing
- Large shared file dumps due to checkpointing – weather forecasting codes
- Large number of files per process

Comet File System Overview



Why Various File Systems

- Performance
- Shared access across nodes
- Backup / long-term
- Quota

Comet File Systems: $HOME

- Location of the home directory – when you login to comet
- Network File System (NFS) storage
  - Typically used to store source codes, important files…
  - Storage limit around 100 GB
- Backup / long-term

Comet File Systems: Lustre Scratch

- Location: /oasis/scratch/comet/$USER/temp_project
- Lustre File System (LFS) performance storage
  - Typically used to store input / output data, large files…
  - Allows distributed access
  - Storage limit around 500 GB
- No Backup

Comet File Systems: Lustre Projects

- Location: /oasis/projects/nsf/…
- Lustre File System (LFS) performance storage
  - Typically used to store input / output data, large files…
  - Project specific data
  - Allows distributed access
  - Storage limit around 2.5 PB
- No Backup

Comet File Systems: Node Local Storage

- Location: /scratch/$USER/$SLURM_JOB_ID…
- Node local SSD storage
  - Typically used to store large number of files…
  - Fast node-local access
  - Storage limit around 210 GB on compute nodes
  - Only accessible from a compute node
- No Backup

Comet File Systems

Path	Purpose	User Access Limits	Lifetime
$HOME	NFS storage; Source code, important files	100 GB	Backed-up
/oasis/scratch/comet/$USER/temp_project	Global/Parallel Lustre FS; temp storage for distributed access	500 GB	No backup
/oasis/projects/nsf	Global/Parallel Lustre FS; project storage	~2.5 PB total	No backup
/scratch/$USER/$SLURM_JOB_ID	Local SSD on batch job node fast per-node access	210 GB per compute node, 286GB on GPU, Large memory nodes	Purged after job ends

Comet File Systems - Guidelines

```
[2] Filesystems:

     (a) Lustre scratch filesystem : /oasis/scratch/comet/$USER/temp_project
         (Preferred: Scalable large block I/O)
            *** Meant for storing data required for active simulations
            *** Not backed up and should not be used for storing data long term
            *** Periodically clear old data not required for active simulations

     (b) Compute/GPU node local SSD storage: /scratch/$USER/$SLURM_JOBID
         (Meta-data intensive jobs, high IOPs)

     (c) Lustre projects filesystem: /oasis/projects/nsf

     (d) /home/$USER : Only for source files, libraries, binaries.
         *Do not* use for I/O intensive jobs.
```

Order of Magnitude Guide

Storage	file/directory	file sizes	BW
Local HDD	1000s	GB	100 MB/s
Local SSD	1000s	GB	500 MB/s
RAM FS	10000s	GB	GB/s
NFS	100s	GB	100 MB/s
Lustre	100s	TB	100 GB/s

Local file systems are good for small and temporary files
(low latency, modest bandwidth)

Network file systems very convenient for sharing data between systems
(high latency, high bandwidth)

Application Focus

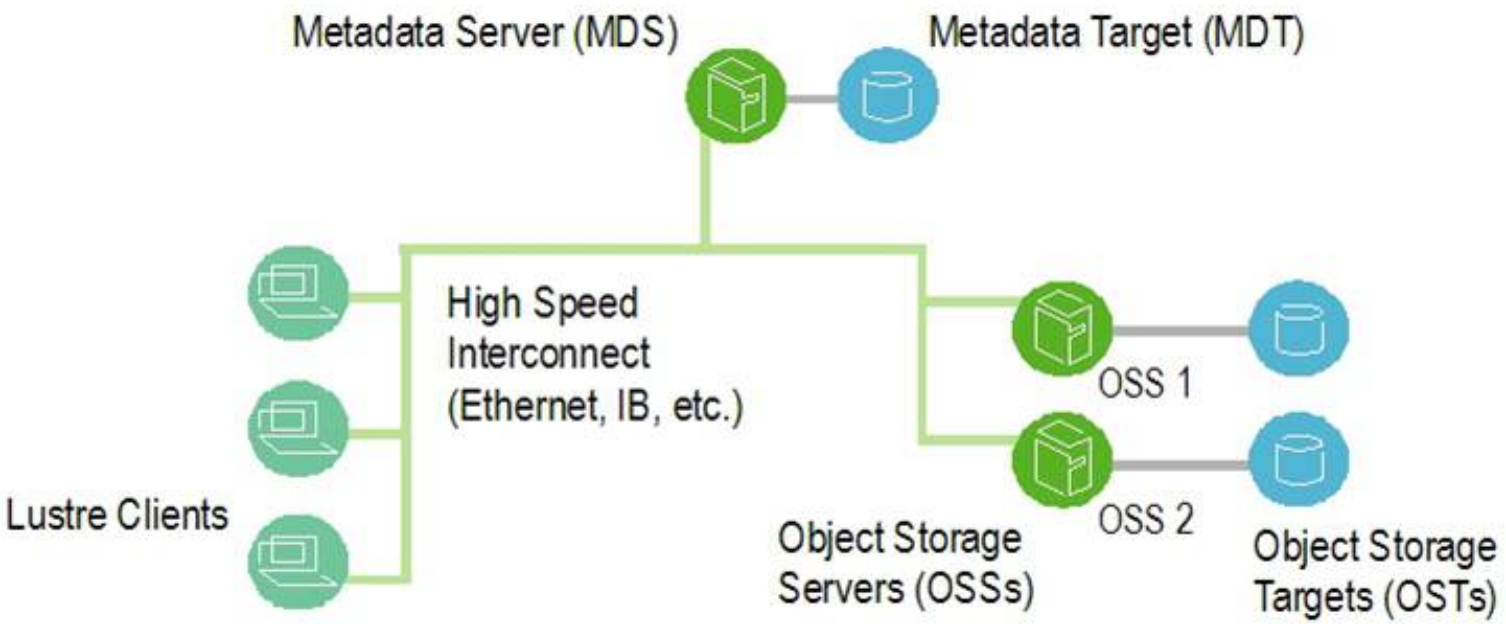
Storage choices should be driven by application need, not just what's available.

But, applications need to adapt as they scale.

Writing a few small files to an NFS server is fine…
writing 1000's simultaneously will wipe out the server.

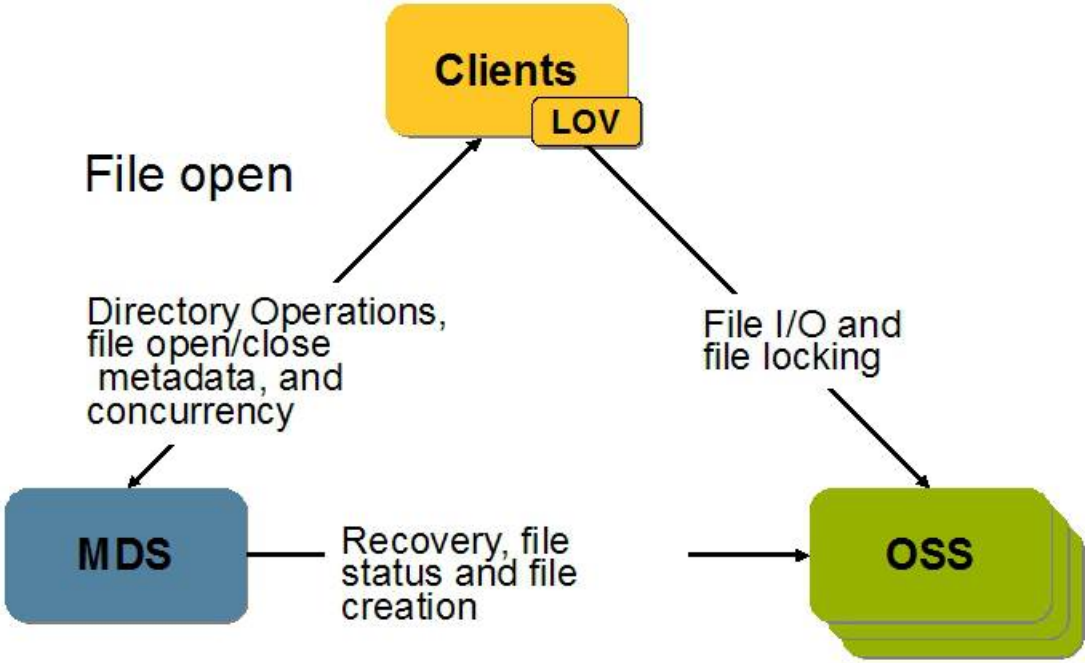
If you use binary files, don't invent your own format. Consider HDF5.

Lustre File System



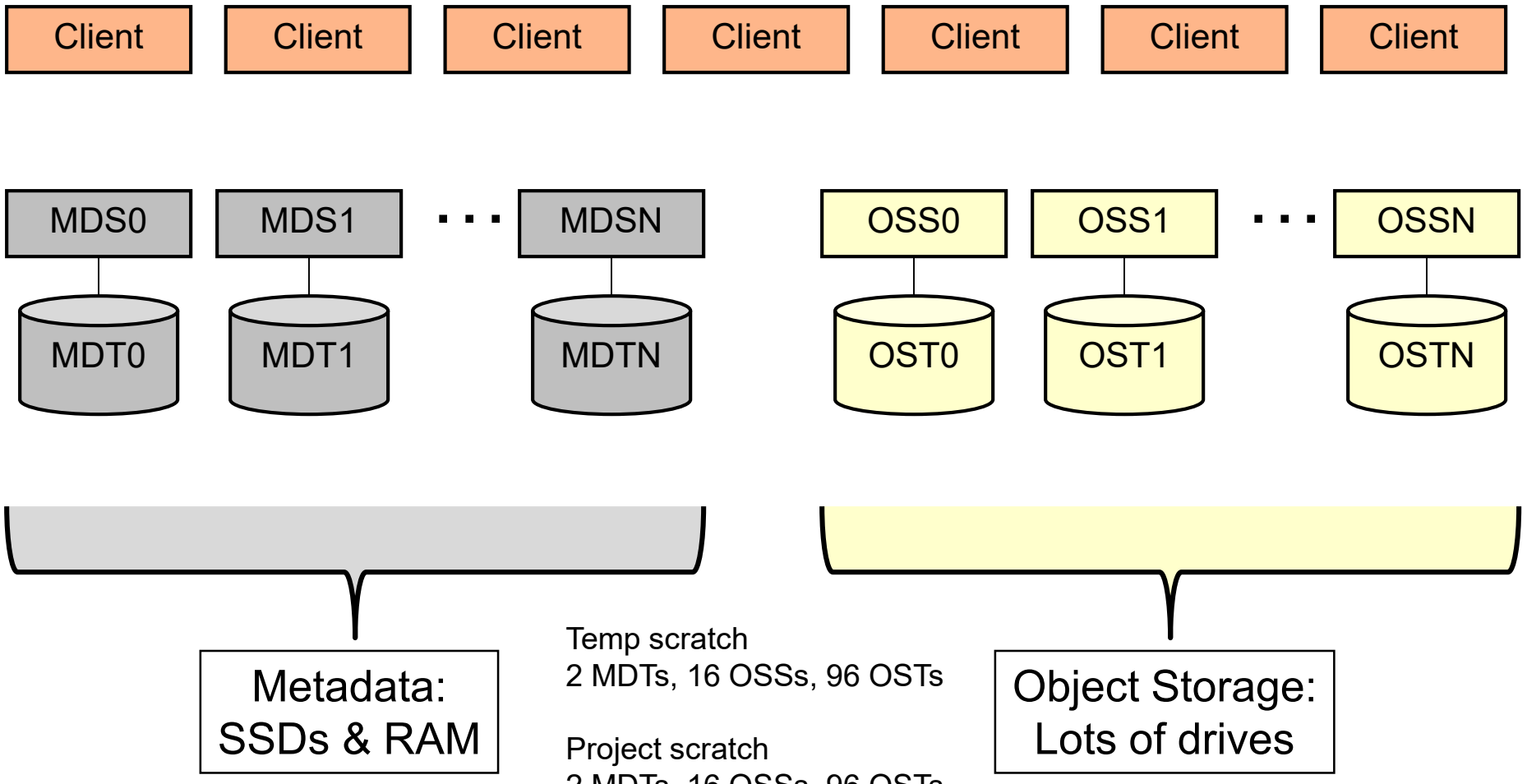
Ref: Cornell Virtual Workshop

LFS Interactions



Ref: Cornell Virtual Workshop

A Typical LFS



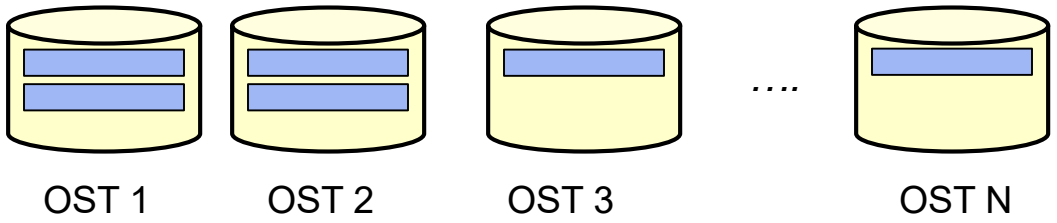
File View

Logical view of a file with N+2 segments

Stripe count = N

Stripe size = sz

Physical view of the file across OSTs



Why is striping useful?

- a way to store a large file
- file can be accessed in parallel, increasing the bandwidth

LFS Commands

lfs help – lists all options

lfs osts – lists all the OSTs

lfs mdts – lists all the MDTs

lfs getstripe – retrieves the striping information of a file / directory

lfs setstripe – sets striping information of a file / directory

LFS Commands: getstripe

```
-bash-4.1$ lfs getstripe testout
testout
lmm_stripe_count:   1
lmm_stripe_size:    1048576
lmm_pattern:        1
lmm_layout_gen:     0
lmm_stripe_offset:  43
              obdidx                 objid                 objid                 group
                  43               8979631            0x8904af                     0

-bash-4.1$ lfs getstripe --stripe-count testout
1
-bash-4.1$ lfs getstripe --stripe-size testout
1048576
```

LFS Commands: setstripe

lfs setstripe -c 16 testout

```
-bash-4.1$ lfs getstripe testout
testout
lmm_stripe_count:   16
lmm_stripe_size:    1048576
lmm_pattern:        1
lmm_layout_gen:     0
lmm_stripe_offset:  89
              obdidx                objid                objid                group
                  89              9202813              0x8c6c7d                    0
                  45              9819070              0x95d3be                    0
                                ..............................
```

LFS Commands: setstripe

```
bash-4.1$ lfs setstripe -c -1 test1
bash-4.1$ lfs getstripe test1
test1
lmm_stripe_count:   96
lmm_stripe_size:    1048576
lmm_pattern:        1
lmm_layout_gen:     0
lmm_stripe_offset:  65
              obdidx           objid           objid           group
                  65         9738084       0x949764               0
                  41         9153699       0x8baca3               0
                                  ...
```

LFS Commands: setstripe

```
-bash-4.1$ mkdir dir
-bash-4.1$ lfs setstripe -c 4 dir
-bash-4.1$ vi dir/test
-bash-4.1$ lfs getstripe dir/test
dir/test
lmm_stripe_count:   4
lmm_stripe_size:    1048576
lmm_pattern:        1
lmm_layout_gen:     0
lmm_stripe_offset:  43
              obdidx           objid           objid           group
                  43         8979901       0x8905bd               0
                  25        10609192       0xa1e228               0
```

LFS Usage Guidelines

- Avoid certain operations
  - ls –l, ls with color, frequent file opens/closes
  - find, du, wildcards (ls *.out)
  - Why??
  - Try /bin/ls -U instead of ls -l
- Select appropriate stripe count / size
  - Best case selection is complicated
- Do not store too many files in one directory

LFS Performance Using IOR Benchmark

- IOR (Interleaved Or Random) developed at LLNL to benchmark/test parallel filesystems.
- Current version is very versatile (beyond what the name suggests).
- Test aggregate I/O rates using several I/O options including **POSIX, MPIIO, HDF5, and NCMPI.**
- Control several aspects of I/O to help mimic real applications:
  - Overall I/O Size
  - Transfer size
  - File access mode – single or file/task ….

IOR Options

IOR -h gives you all the options. Some important ones are:

-F : *write one file per task*

(without -F a single file is written)

-b : *blockSize – contiguous bytes to write per task*

-t : *size of transfer in bytes (e.g. 8, 4k, 2m, 1g)*

-w : *Only write a file (default is to write and read)*

-r : *Only read an existing file*

-i : *number of iterations*

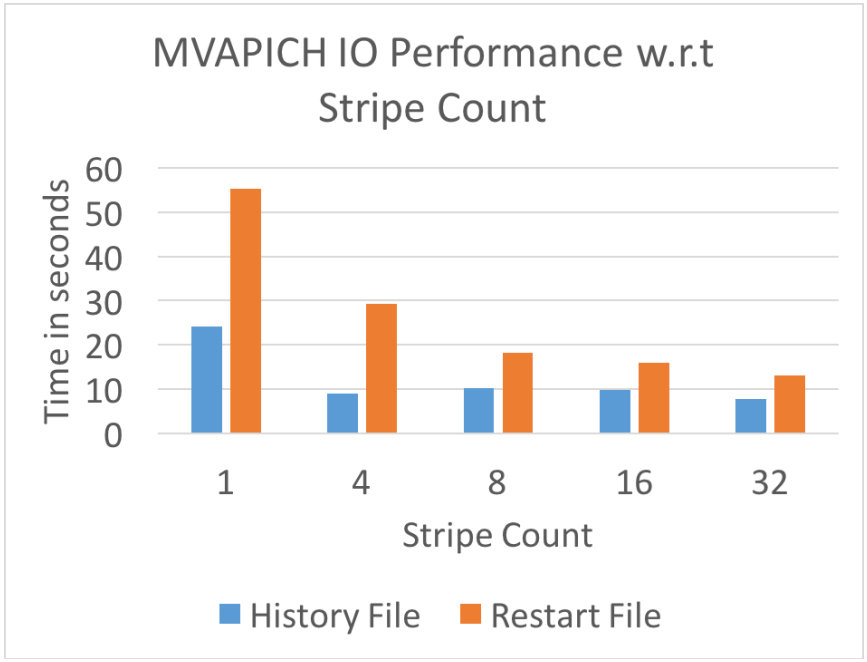
-B : *uses O_DIRECT for POSIX, bypassing I/O buffers*

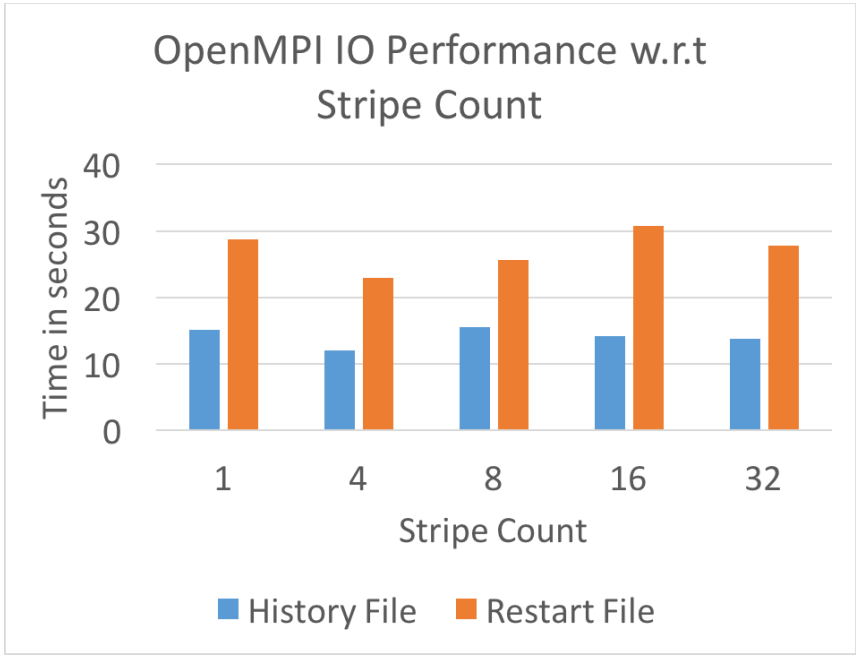
Command: mpirun -n 16 ./IOR.mpiio -a MPIIO -b 1g -w -k -t 1m -i 1 –o testfile

Case Study: WRF

- Weather Research & Forecasting (WRF) Model: collaboration between NCAR, NCEP, NRL and many other institutions
- Used for atmospheric research
- Different types of I/O within WRF
  - Reading from a large input file, writing forecast history files periodically, writing restart files periodically
- Various ways to perform I/O:
  - NetCDF file from task 0, PNetCDF, I/O Quilting – processors dedicated for output

I/O Performance vs Lustre Stripe Count
(240 cores)





Stripe count matters, need to carefully tune the libraries

Choosing

My application needs to:

Write a checkpoint dump from memory from a large parallel simulation.

I should consider:

A parallel file system and a binary file format like HDF5.

Choosing

My application needs to:

write and read 1000s of small files local to each process, store all the files across all the processes

I should consider:

a combination of local SSDs and Lustre!

Choosing

My application needs to:

Randomly access many small files, or read and write small blocks from large files.

I should consider:

RAM FS, or local scratch space.

Thank You!

Questions: help@xsede.org