Lustre2.5 Performance Evaluation: Performance Improvements with Large I/O Patches, Metadata Improvements, and Metadata Scaling with DNE

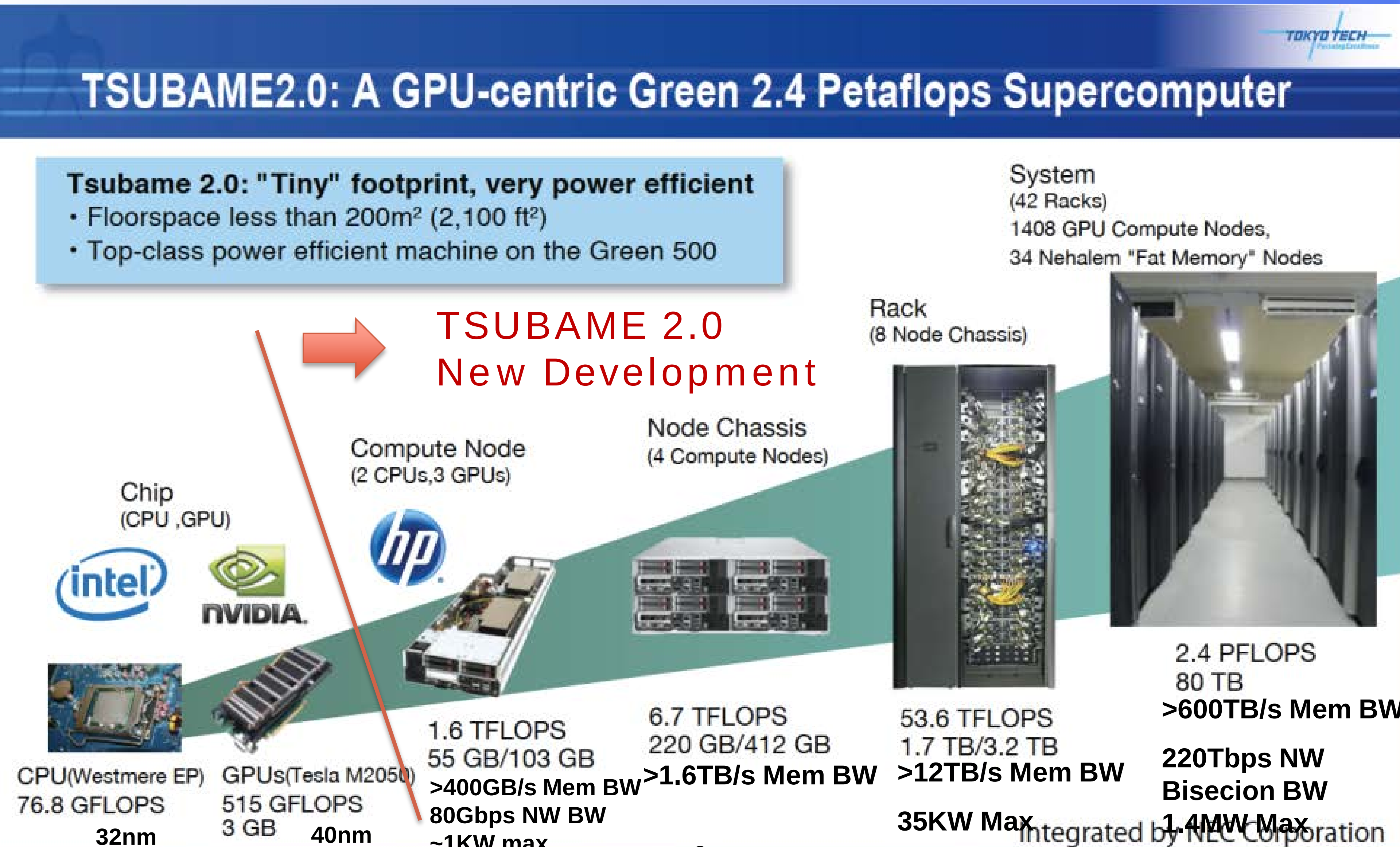
Hitoshi Sato[*1], Shuichi Ihara[*2], Satoshi Matsuoka[*1]

[*1] Tokyo Institute of Technology

[*2] Data Direct Networks Japan

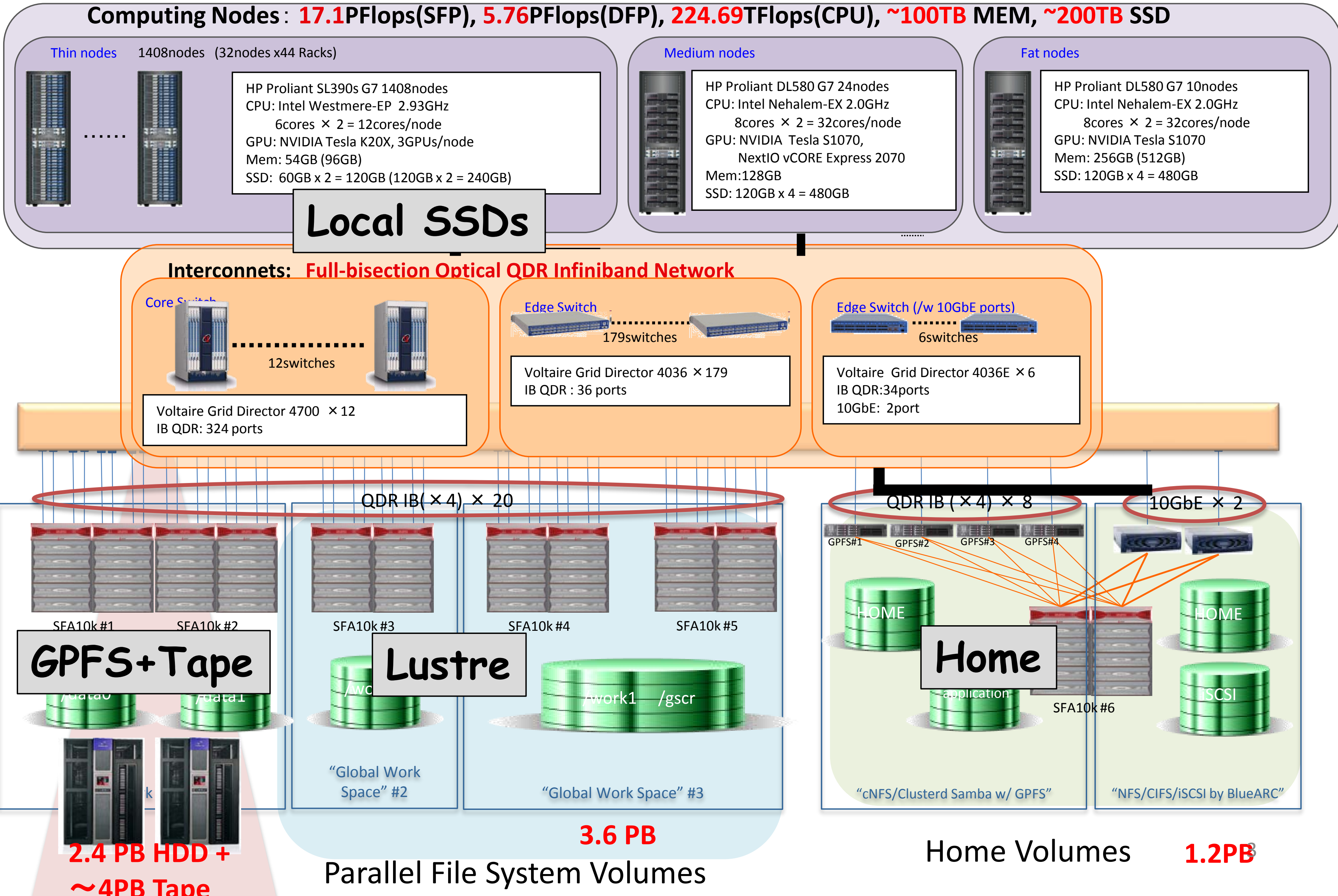
Tokyo Tech's TSUBAME2.0 Nov. 1, 2010

"The Greenest Production Supercomputer in the World"



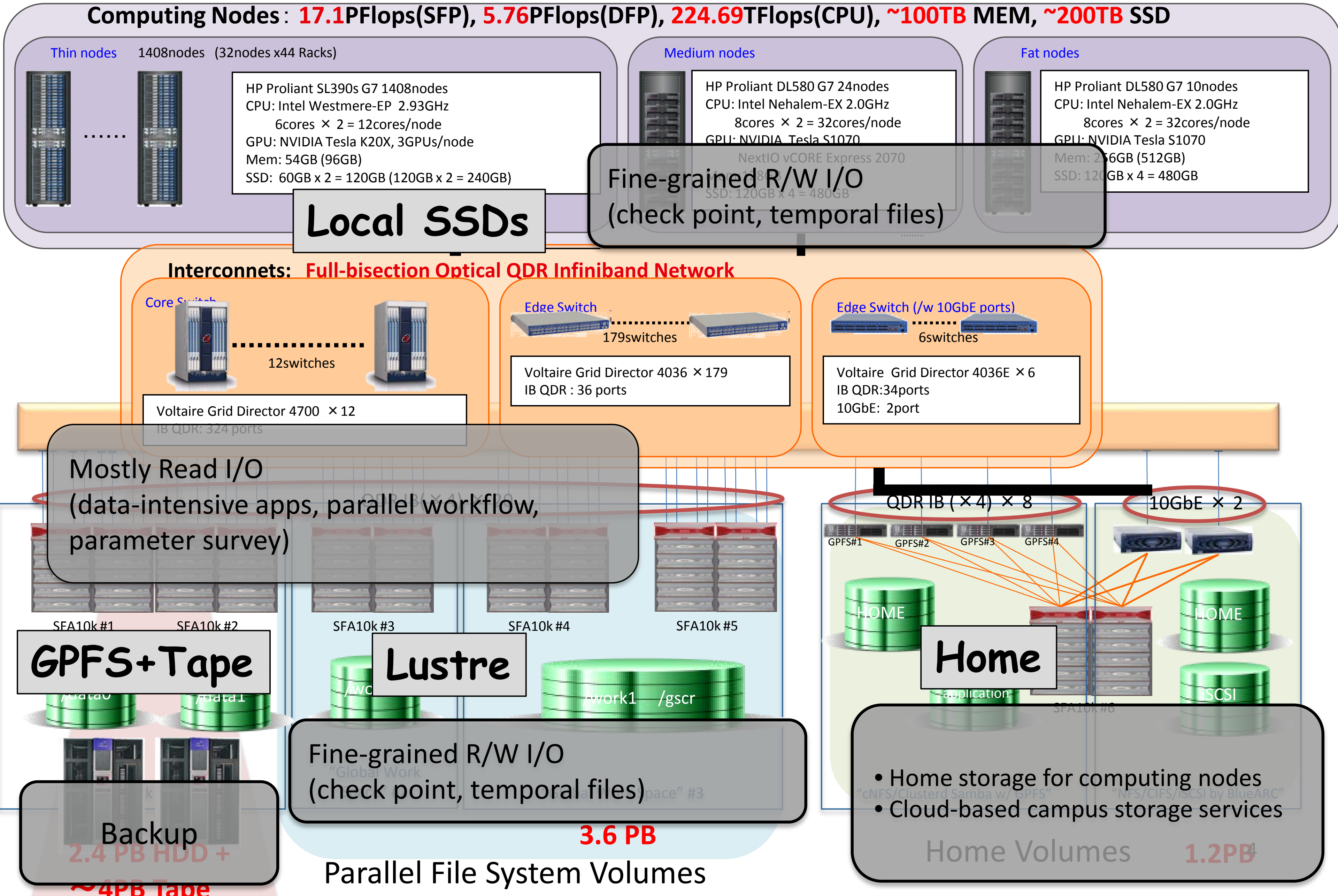
TSUBAME2 System Overview

11PB (7PB HDD, 4PB Tape, 200TB SSD)



TSUBAME2 System Overview

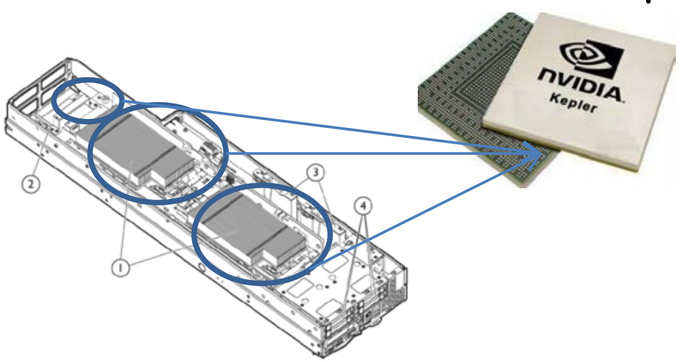
11PB (7PB HDD, 4PB Tape, 200TB SSD)



Towards TSUBAME 3.0

Interim Upgrade TSUBAME2.0 to 2.5 (Sept.10th, 2013)

Upgrade the TSUBAME2.0s GPUs NVIDIA Fermi M2050 to Kepler K20X



TSUBAME2.0 Compute Node
Fermi GPU 3 x 1408 = 4224 GPUs

SFP/DFP peak
from 4.8PF/2.4PF
=> 17PF/5.7PF

TSUBAME-KFC

A TSUBAME3.0 prototype system with advanced cooling for next-gen. supercomputers.

40 compute nodes are oil-submerged.
160 NVIDIA Kepler K20x
80 Ivy Bridge Xeon
FDR Infiniband

Total 210.61 Flops
System 5.21 TFlops

Storage design is also a matter!!

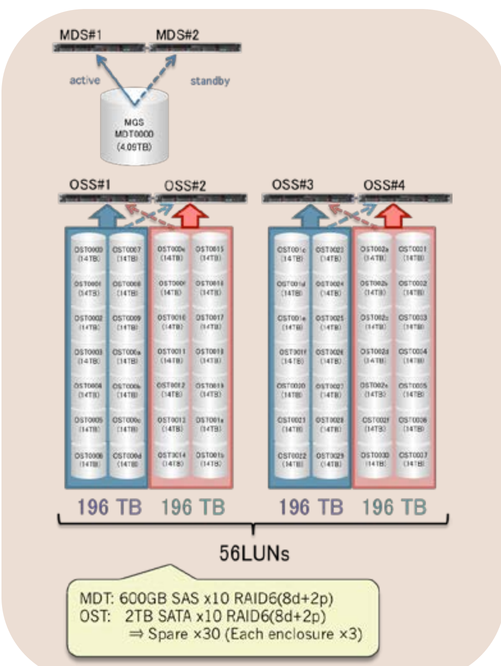
Existing Storage Problems

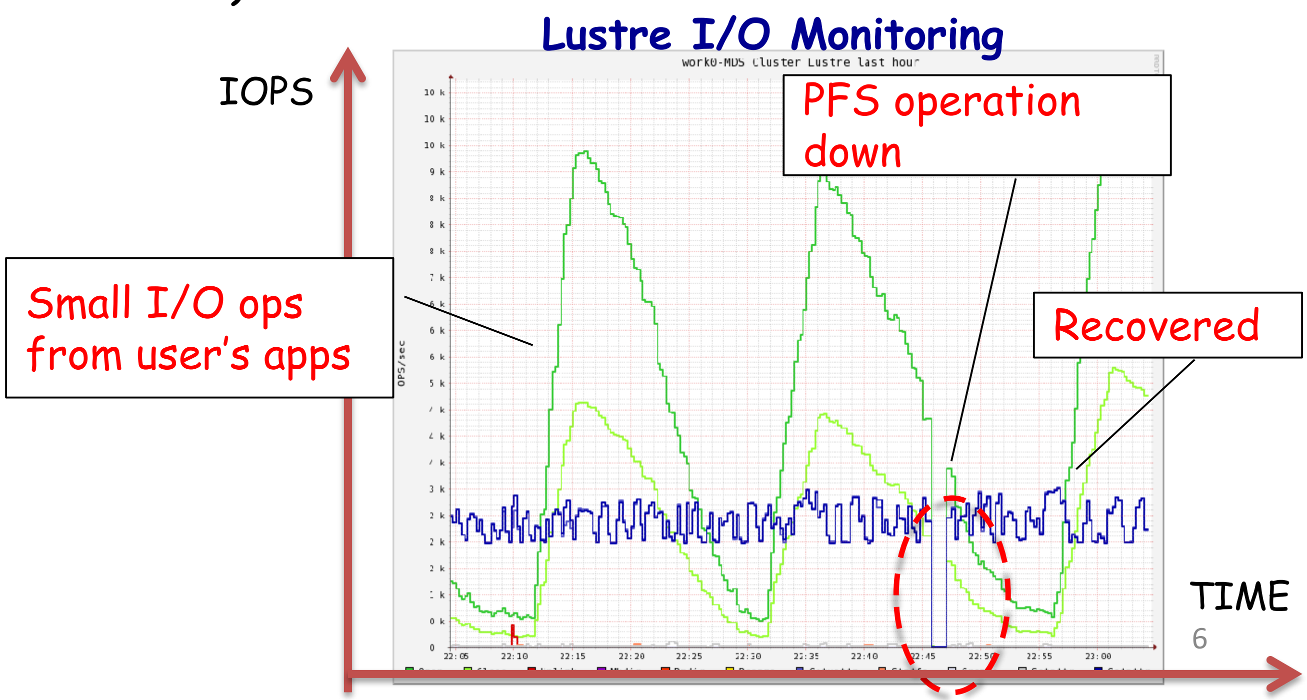
Small I/O

- PFS
  - Throughput-oriented Design
  - Master-Worker Configuration
    - Performance Bottlenecks on Meta-data references
    - Low IOPS (1k – 10k IOPS)

I/O contention

- Shared across compute nodes
  - I/O contention from various multiple applications
  - I/O performance degradation





Requirements for Next Generation I/O Sub-System 1

- Basic Requirements
  - TB/sec sequential bandwidth for traditional HPC applications
  - Extremely high IOPS for to cope with applications that generate massive small I/O
    - Example: graph, etc.
  - Some application workflows have different I/O requirements for each workflow component
- Reduction/consolidation of PFS I/O resources is needed while achieving TB/sec performance
  - We cannot simply use a larger number of IO servers and drives for achieving ~TB/s throughput!
  - Many constraints
    - Space, power, budget, etc.
- Performance per Watt and performance per RU is crucial

Requirements for Next Generation I/O Sub-System 2

- New Approaches
  - Tsubame 2.0 has pioneered the use of local flash storage as a high-IOPS alternative to an external PFS
  - Tired and hybrid storage environments, combining (node) local flash with an external PFS
- Industry Status
  - High-performance, high-capacity flash (and other new semiconductor devices) are becoming available at reasonable cost
  - New approaches/interface to use high-performance devices (e.g. NVMexpress)

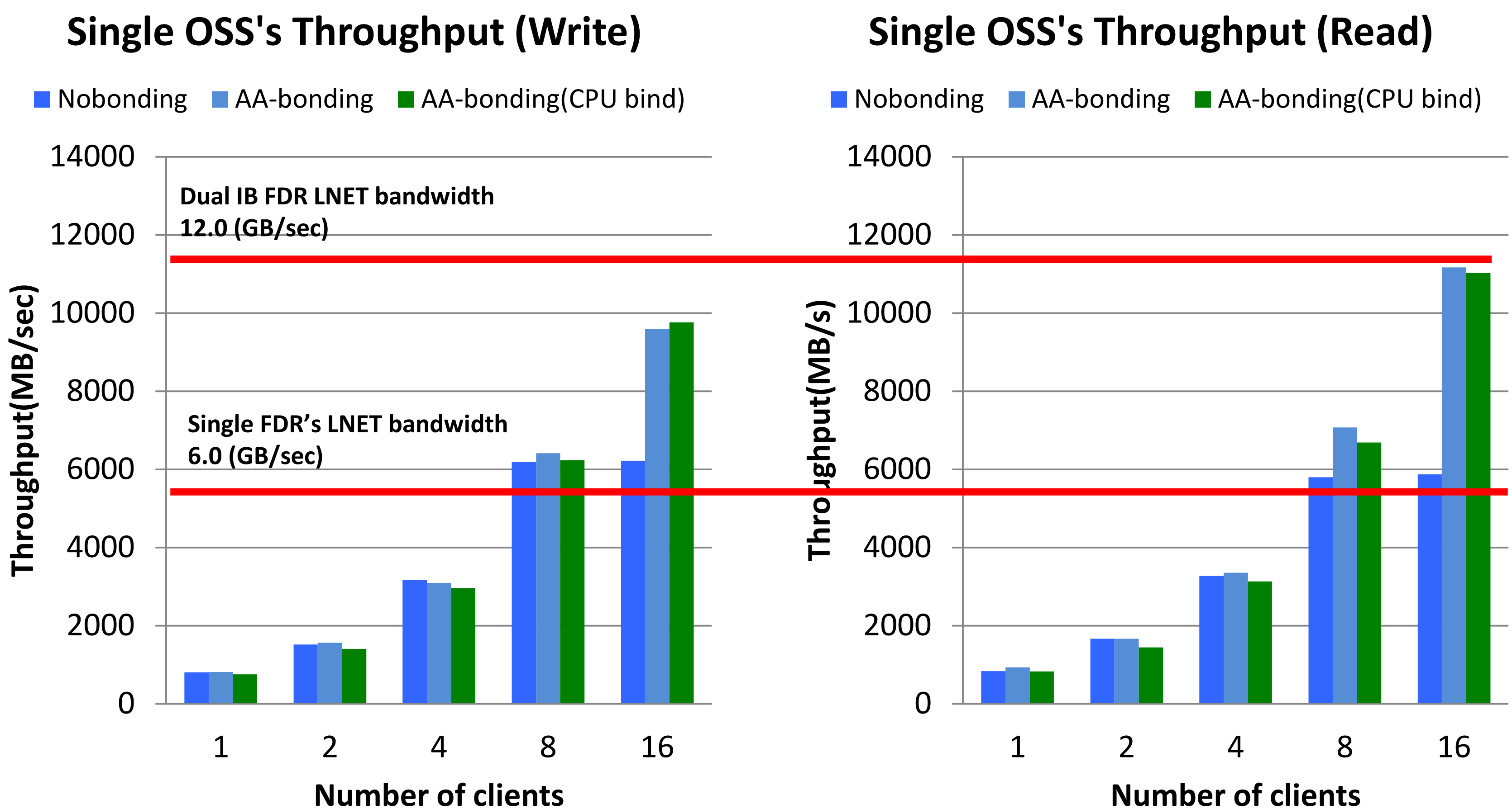
Key Requirements of I/O system

- Electrical Power and Space are still Challenging
  - Reduce of #OSS/OST, but keep higher IO Performance and large storage capacity
  - How maximize Lustre performance
- Understand New type of Flush storage device
  - Any benefits with Lustre? How use it?
  - MDT on SSD helps?

Lustre 2.x Performance Evaluation

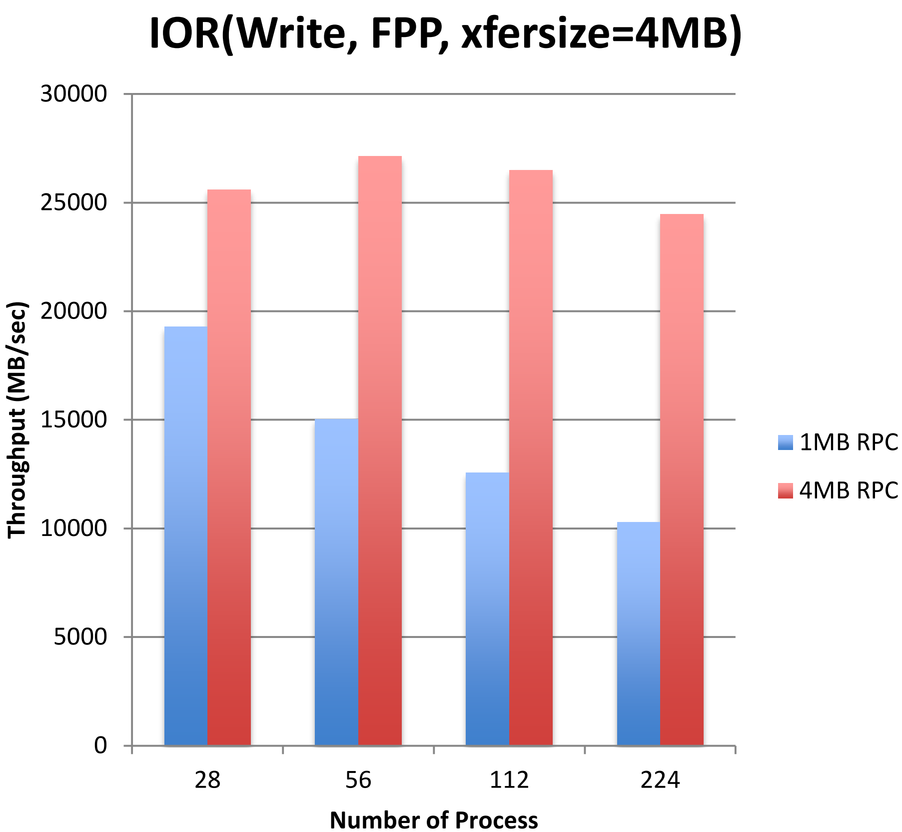
- Maximize OSS/OST Performance for large Sequential IO
  - Single OSS and OST performance
  - 1MB vs 4MB RPC
  - Not only Peak performance, but also sustain performance
- Small IO to shared file
  - 4K random read access to the Lustre
- Metadata Performance
  - CPU impacts?
  - SSD helps?

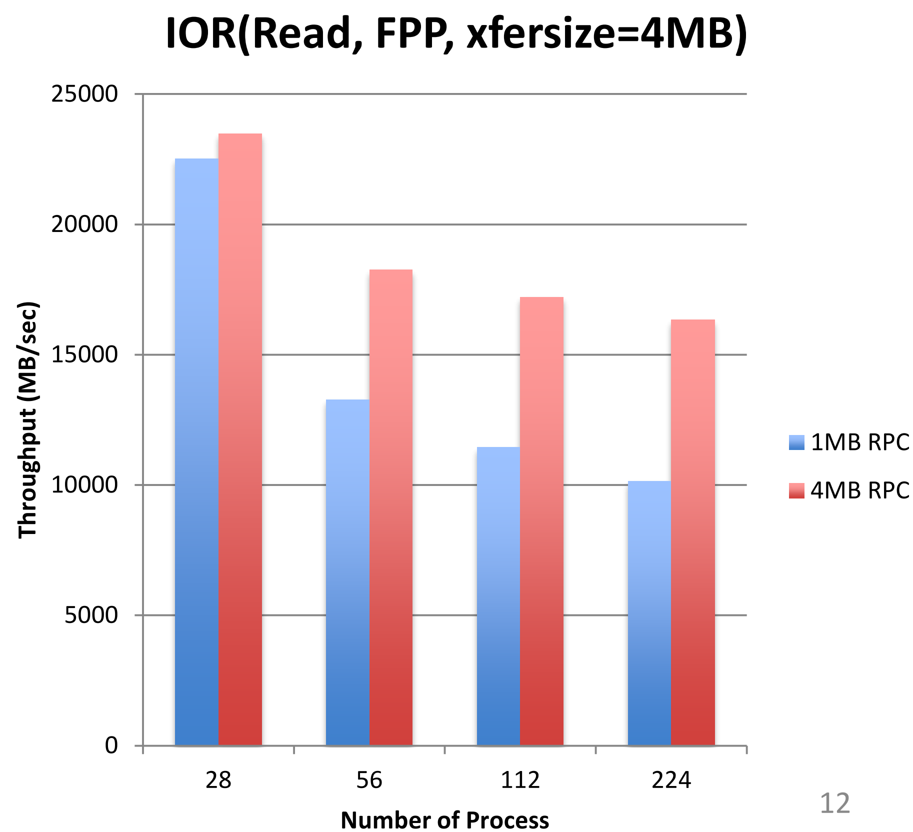
Throughput per OSS Server



Performance comparing
(1 MB vs. 4 MB RPC, IOR FPP, asyncIO)

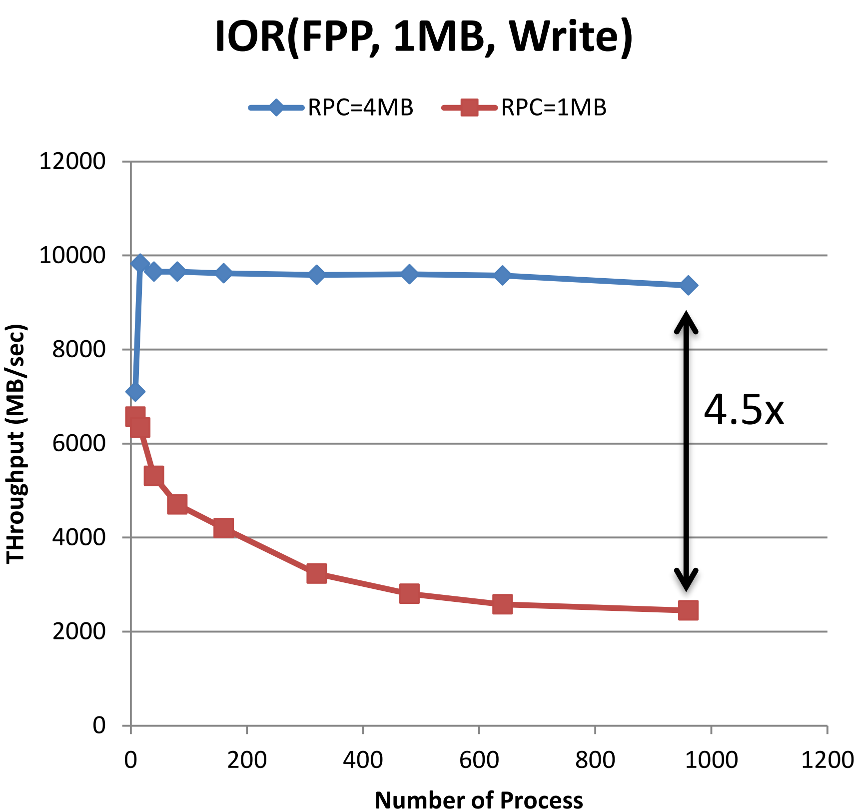
- 4 x OSS, 280 x NL-SAS
- 14 clients and up to 224 processes

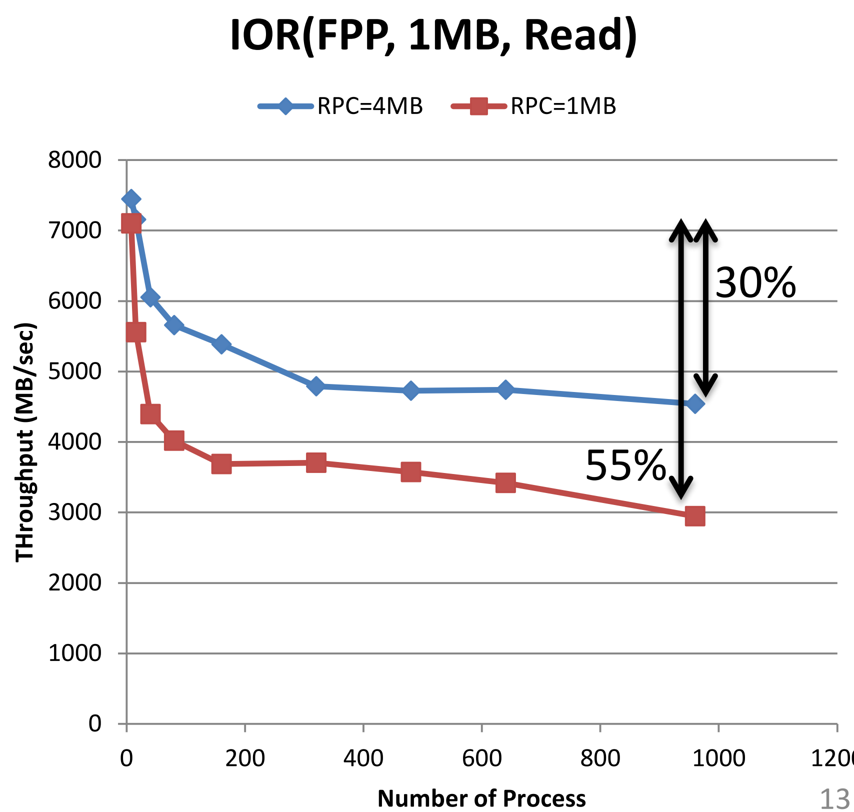




Lustre Performance Degradation with Large Number of Threads: OSS standpoint

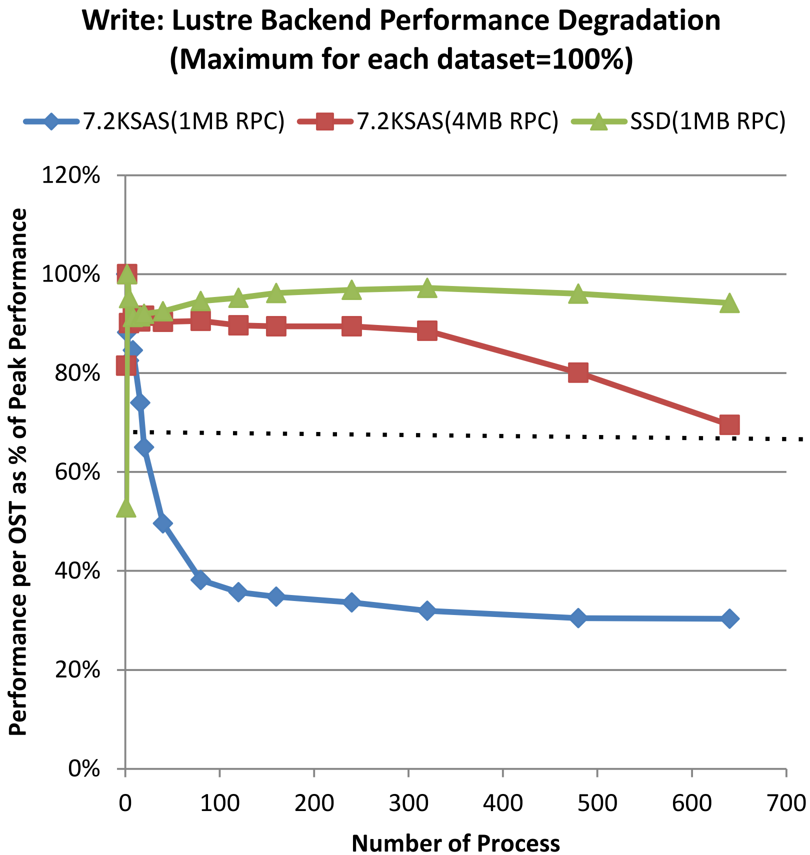
- 1 x OSS, 80 x NL-SAS
- 20 clients and up to 960 processes

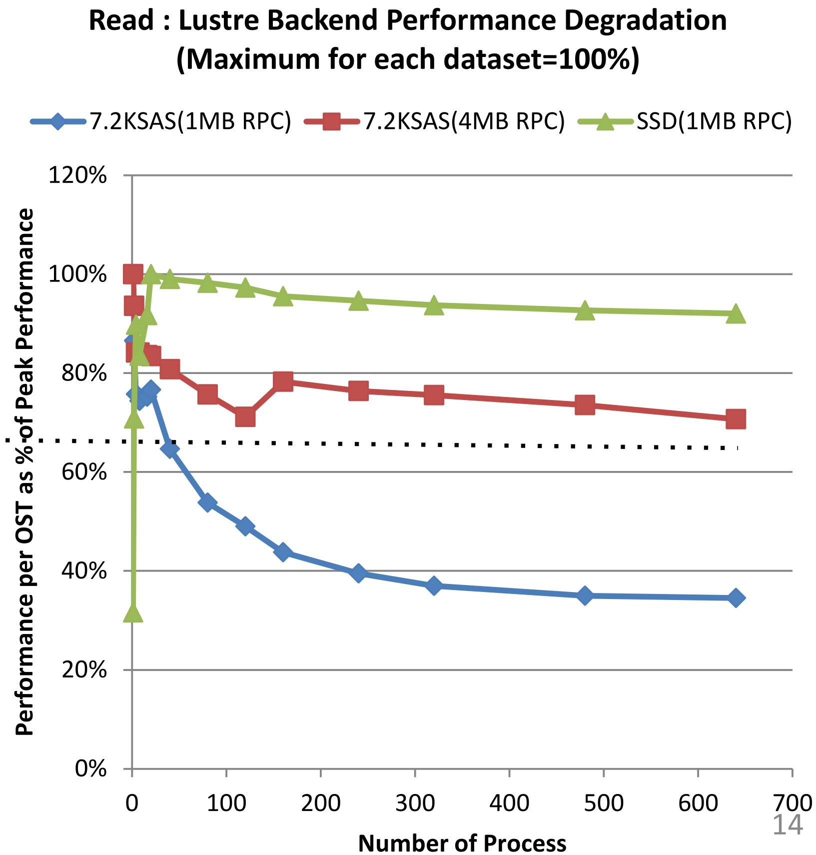




Lustre Performance Degradation with Large Number of Threads: SSDs vs. Nearline Disks

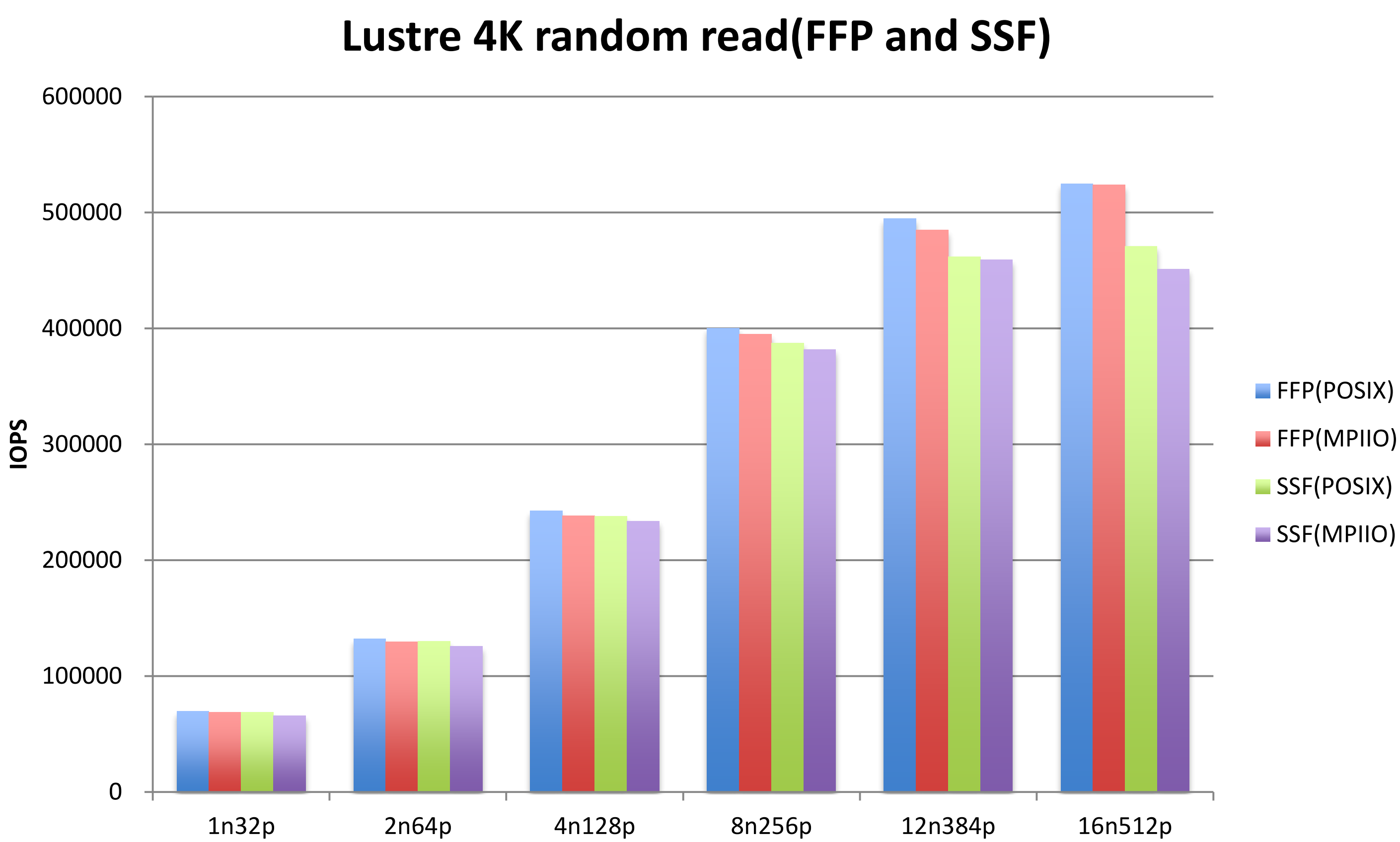
- IOR (FFP, 1MB) to single OST
- 20 clients and up to 640 processes





Lustre 2.4 4k Random Read Performance (With 10 SSDs)

- 2 x OSS, 10 x SSD(2 x RAID5), 16 clients
- Create Large random files (FFP, SSF), and run random read access with 4KB

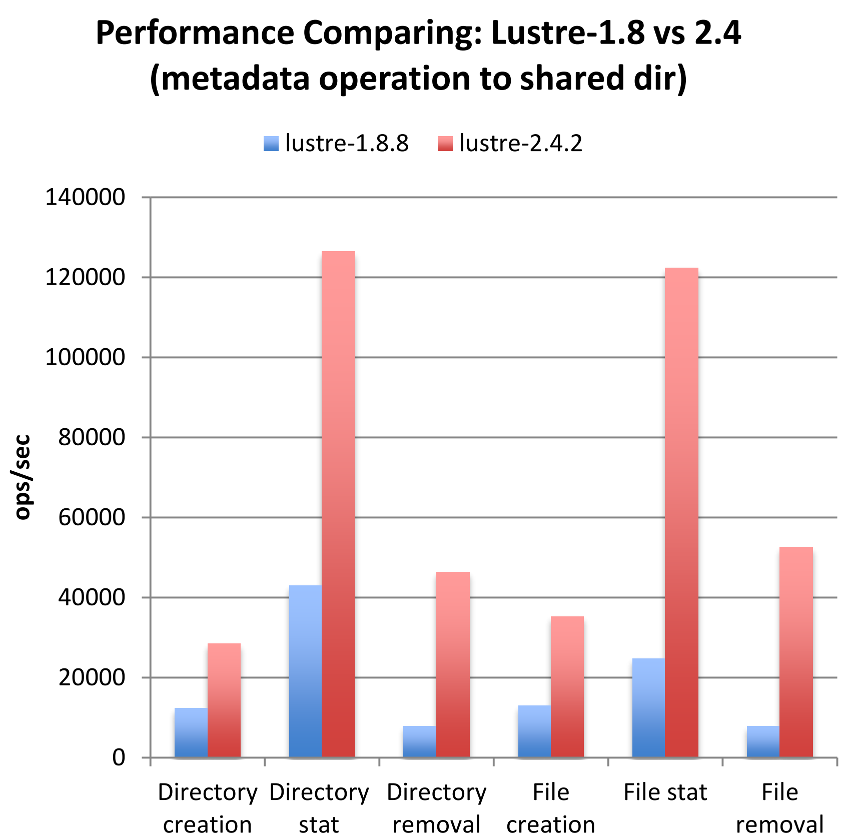


Conclusions: Lustre with SSDs

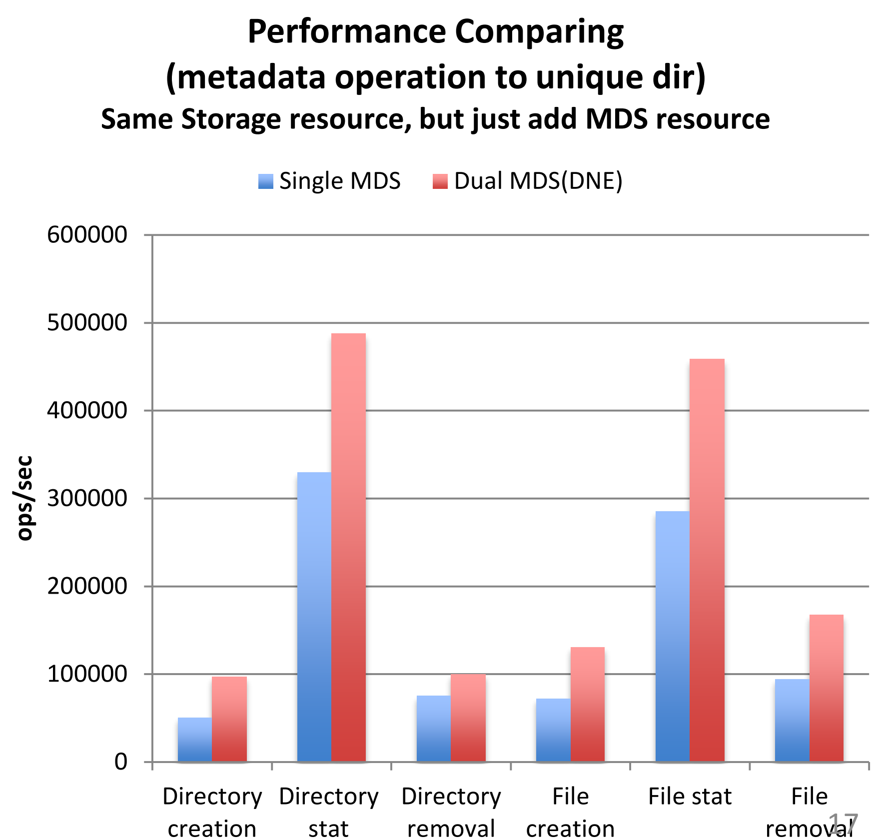
- SSDs pools in Lustre or SSDs
  - SSDs change the behavior of pools, as “seek” latency is no longer an issue
- Application scenarios
  - Very consistent performance, independent of the IO size or number of concurrent IOs
  - Very high random access to a single shared file (millions of IOPS in a large file system with sufficient clients)
- With SSDs, the bottleneck is no more the device, but the RAID array (RAID stack) and the file system
  - Very high random read IOPS in Lustre is possible, but only if the metadata workload is limited (i.e. random IO to a single shared file)
- We will investigate more benchmarks of Lustre on SSD

Metadata Performance Improvements

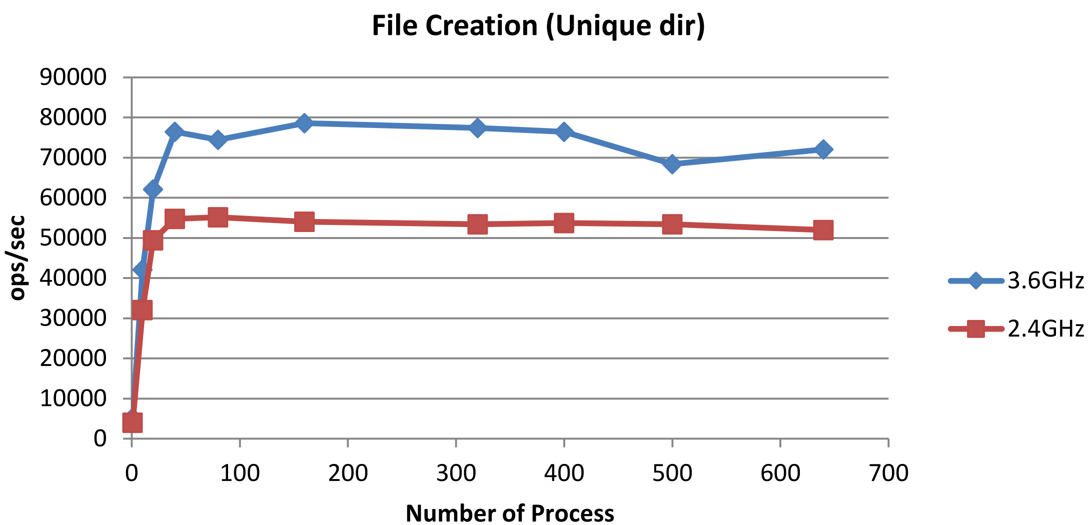
- Very Significant Improvements (since Lustre 2.3)
  - Increased performance for both unique and shared directory metadata
  - Almost linear scaling for most metadata operations with DNE

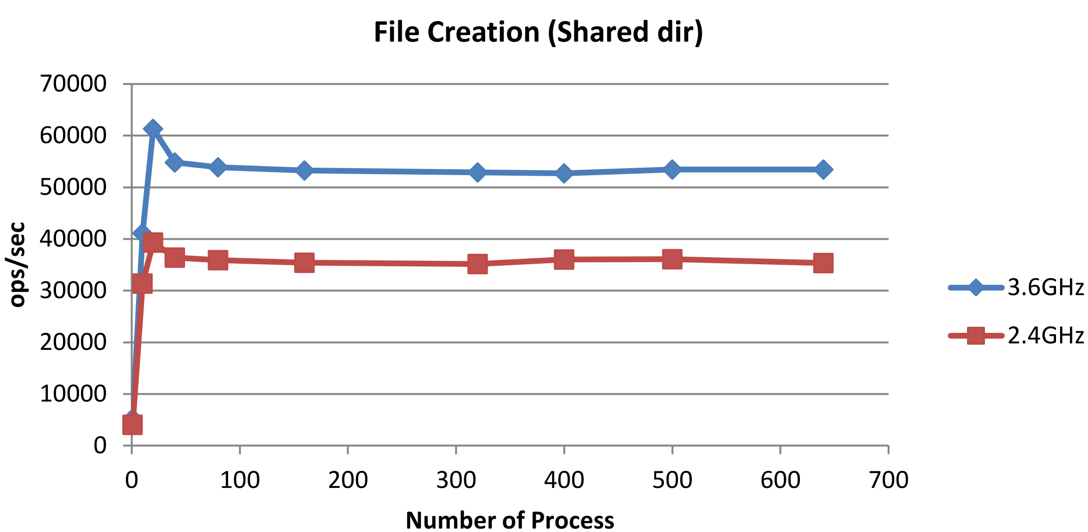


* 1 x MDS, 16 clients, 768 Lustre exports(multiple mount points)



Metadata Performance CPU Impact





- Metadata Performance is highly dependent on CPU performance
- Limited variation in CPU frequency or memory speed can have a significant impact on metadata performance

Conclusions: Metadata Performance

- Significant metadata improvements for shares and unique metadata performance since Lustre 2.3
  - Improvements across all metadata workloads, especially removals
  - SSDs add to performance, but the impact is limited (and probably mostly due to latency, not the IOPS performance of the device)
- Important considerations for metadata benchmarks
  - Metadata benchmarks are very sensitive to the underlying hardware
  - Clients limitations are important when considering metadata performance of scaling
  - Only directory/file stats scale with the number of processes per node, other metadata workloads do appear not scale with the number of processes on a single node

Summary

- Lustre Today
  - Significant performance increase in object storage and metadata performance
  - Much better spindle efficiency (which is now reaching the physical drive limit)
  - Client performance is quickly becoming the limitation for small file and random performance, not the server side!!
- Consider alternative scenarios
  - PFS combined with fast local devices (or, even, local instances of the PFS)
  - PFS combined with a global acceleration/caching layer