FUJITSU

shaping tomorrow with you

FEFS* Performance Evaluation on K computer and Fujitsu's Roadmap toward Lustre 2.x

Based on LUG2013 Slides

Shinji Sumimoto
Fujitsu Limited, a member of OpenSFS
Oct.17 2013 @APAC LUG 2013, Tokyo

OpenSFS

*: "FUJITSU Software FEFS"

Fujitsu Joins OpenSFS

FUJITSU

OpenSFS

About Us | Why Join | Resources | Events | News | Contact

Lustre Community

Follow Us

Search

Fujitsu Joins OpenSFS

Lustre® file system support continues to grow worldwide, OpenSFS membership expanding

Beaverton, OR – October 14, 2013 – Open Scalable File Systems, Inc. (OpenSFS), the premier non-profit organization advancing and coordinating the Lustre® file system community, is announcing Fujitsu has joined OpenSFS. Fujitsu is the world's fourth largest IT services provider and is the joint developer of the K computer, the world's fastest supercomputer in 2011. Fujitsu is joining OpenSFS at the Supporter Level, which provides organizations the ability to vote on the OpenSFS stack of software as well as participate in working groups.

Fujitsu is also a Gold Sponsor of the Lustre User Group (LUG) in Tokyo, taking place October 17, 2013.

"We are very excited to welcome Fujitsu, a perennial leader of the TOP10 supercomputing sites list," said Galen Shipman, OpenSFS Chairman. "Fujitsu pushes Lustre to extreme limits, while maintaining famously high quality standards for its users. So their membership provides even more strength and support to the growth of Lustre worldwide."

The K computer, which is jointly developed by RIKEN and Fujitsu, is part of the High-Performance Computing Infrastructure (HPCI) initiative led by Japan's Ministry of Education, Culture, Sports, Science and Technology (MEXT). Configuration of the K computer began in September 2010.

The "K" in K computer comes from the Japanese kanji letter "Kei" which means ten peta or 10 to the 16th power. And the logo for the K computer is based on the Japanese kanji letter Kei. In its original sense, "Kei" expresses a large gateway, and "it is hoped that the system will be a new gateway to computational science."

Lustre Links

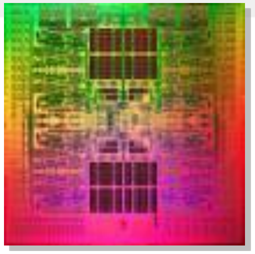
- Join a Mailing List
- Get started with Lustre
- Read Documentation
- Download Lustre
- Submit an Issue
- Lustre Wiki

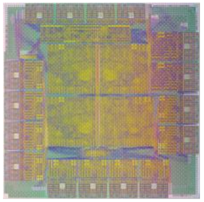
Outline

- RIKEN and Fujitsu jointly developed "K computer"
  - Now in Public Operation, and still continuing system software tuning for more suitable.
- Outline of This Talk
  - K computer and FEFS Overview
  - Performance Evaluation
  - Issues towards Exascale
  - Fujitsu's Roadmap towards Lustre 2.x

System Overview of K computer

FUJITSU



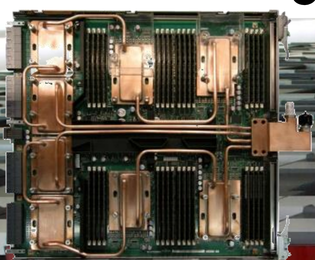


Processor: SPARC64™ VIIIfx

- Fujitsu's 45nm technology
- 8 Core, 6MB Cache Memory and MAC on Single Chip
- High Performance and High Reliability with Low Power Consumption

Interconnect Controller:ICC

- 6 dims-Torus/mesh（Tofu Interconnect）



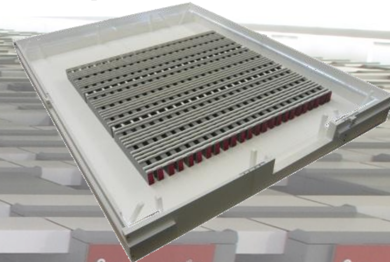
System Board: High Efficient Cooling

- With 4 Computing Nodes
- Water Cooling: Processors, ICCs etc
- Increasing component lifetime and reducing electric leak current by low temperature water cooling

Rack：High Density

- 102 Nodes on Single Rack
  - 24 System Boards
  - 6 IO System Boards
  - System Disk
  - Power Units

(10PFlops: 864 Racks)

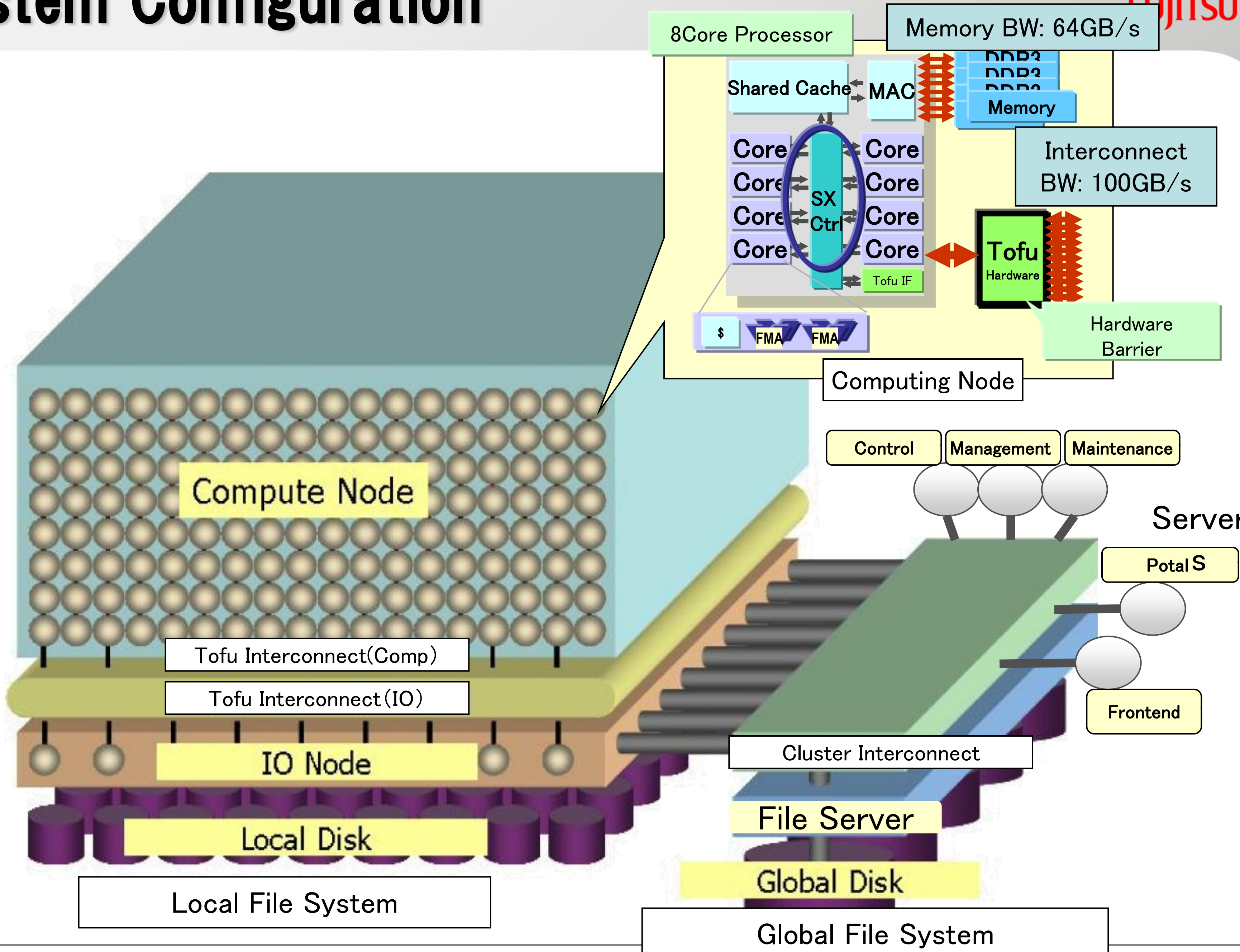


Our Goals

- Challenging to Realize World's Top 1 Performance
- Keeping Stable System Operation over 88K Node System

©RIKEN

System Configuration



IO Architecture of “K computer”

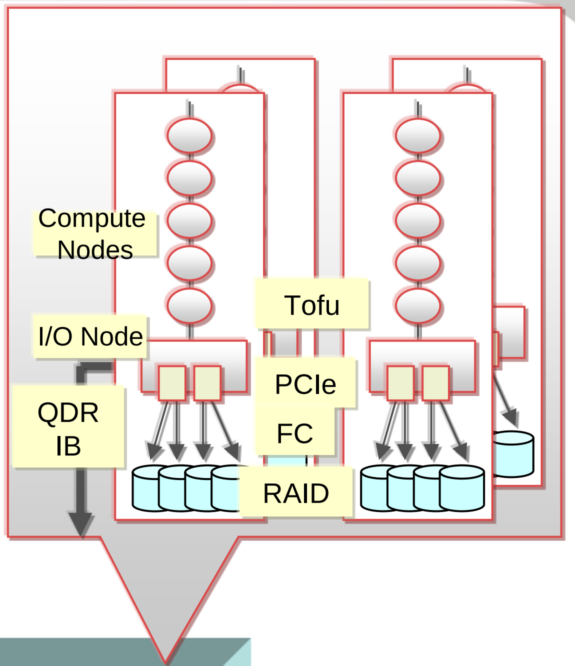
FUJITSU

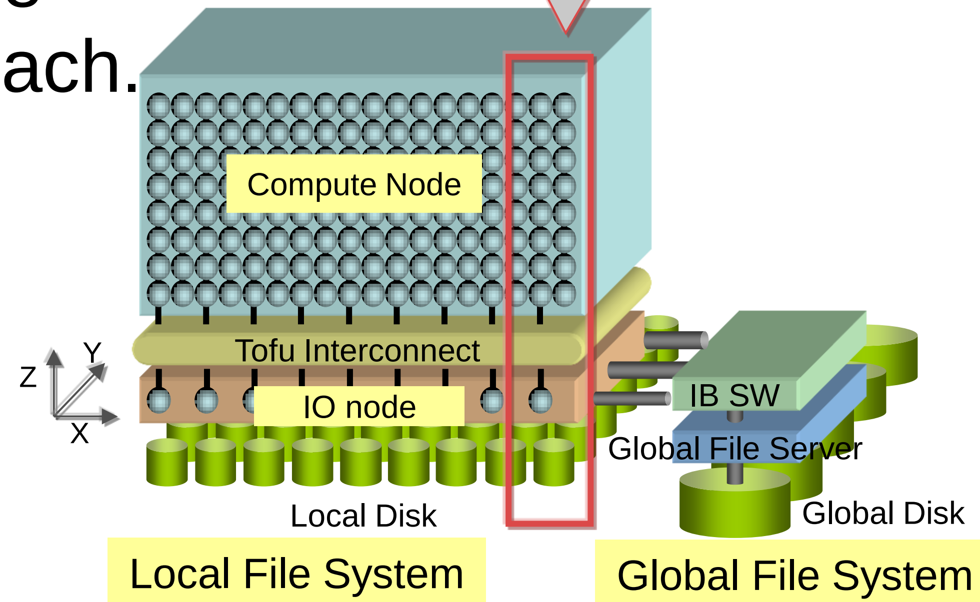
IO System Architecture

- Local Storage for JOB Execution: ETERNUS(2.5inch, RAID5)
- Global Storage for Shared Use: ETERNUS(3.5inch, RAID6)

Configurations of each file system is optimized for each.

- Local File System: Over 2,400-OSS (for Highly Parallel)
- Global File System: Over 80-OSS (for Big Capacity)





FEFS Requirement of K computer

- Extremely Large
  - Extra-large volume (100PB~1EB).
  - Massive number of clients (100k~1M) & servers (1k~10k)
- High Performance
  - Throughput of Single-stream (~GB/s) & Parallel IO (~TB/s).
  - Reducing file open latency (~10k ops).
  - Avoidance of IO interferences among jobs.
- High Reliability and High Availability
  - Always continuing file service against component failures
- Low Resource Usage
  - System Software including MPI runtime, file cache and OS limits its memory usage within 10% of physical memory.

Design Issues for Ultra Large Scale File System

- Keeping user's available memory over 90% of physical memory
  - Clients requires o(# of Servers) memory statically
- Minimizing impact of OS jitter to application performance
  - llpings among all clients and OSSs are terrible
- Parallel IO performance
  - Leveled I/O and Communication Performance among Servers and Network Links
- RAS
  - Recovery Performance

Keeping user's available memory over 90% of physical memory

- **Strategy:**
  - Limiting file buffer cache for dirty buffers
  - Minimizing local buffer usages
  - Minimizing Number of OSTs
- **Issue:**
  - System Software including MPI runtime and OS limits its memory usage within 10% of physical memory.
  - Basic Memory Allocation Policy of Lustre is pre-allocation for max size system.

Memory Issue: Request Buffer (1)

FUJITSU

Issue

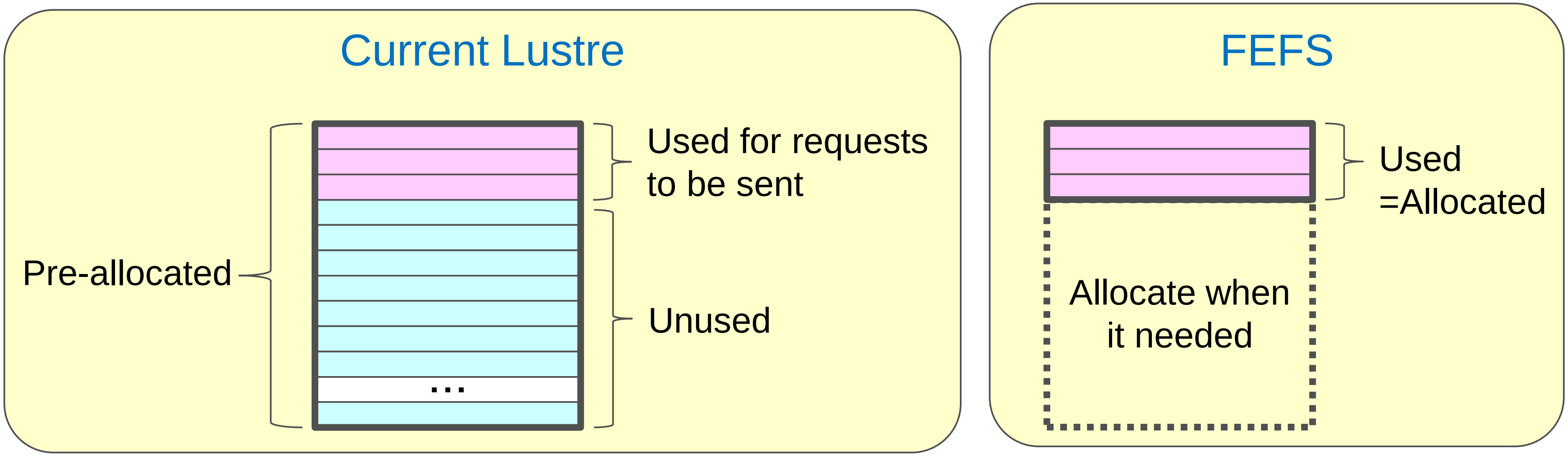
- Request buffer on client is **pre-allocated by #OSTs** in Lustre.
  - Buffer size = **8KB x 10 x #OSTs / request**

    #OST=1,000 ⇒ 80MB / request

    #OST=10,000 ⇒ 800MB / request

Our Approach

- On demand allocation: Allocate request buffer when it required.



Memory Issue: Request Buffer (2)

Issue

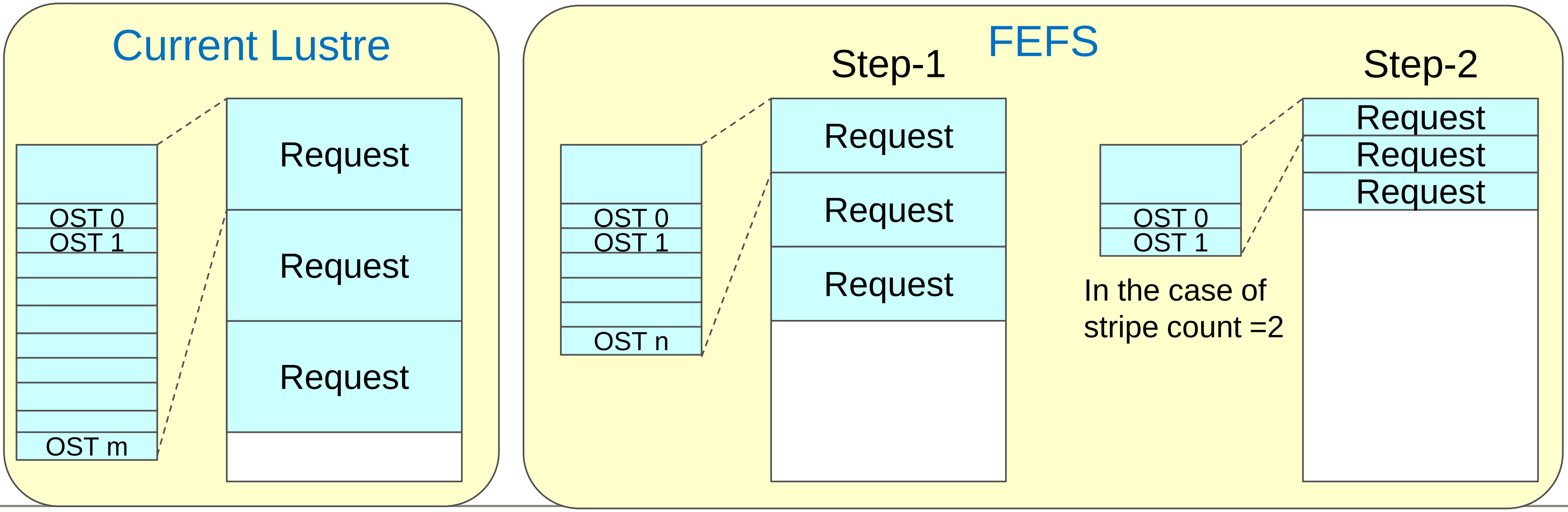
- When create a file, client allocates "**24B x Max. OST index**" size of request buffer. (to store message sent from MDS)

  OST index = 1,000 ⇒ 23KB / request

  OST index = 10,000 ⇒ 234KB / request

Our Approach

- Step-1: Reduce buffer size to "**24B x #Existing OSTs**". (done)
- Step-2: Minimize to "**24B x #Striped OSTs**".



Request Size: OST data sent in close

Issue

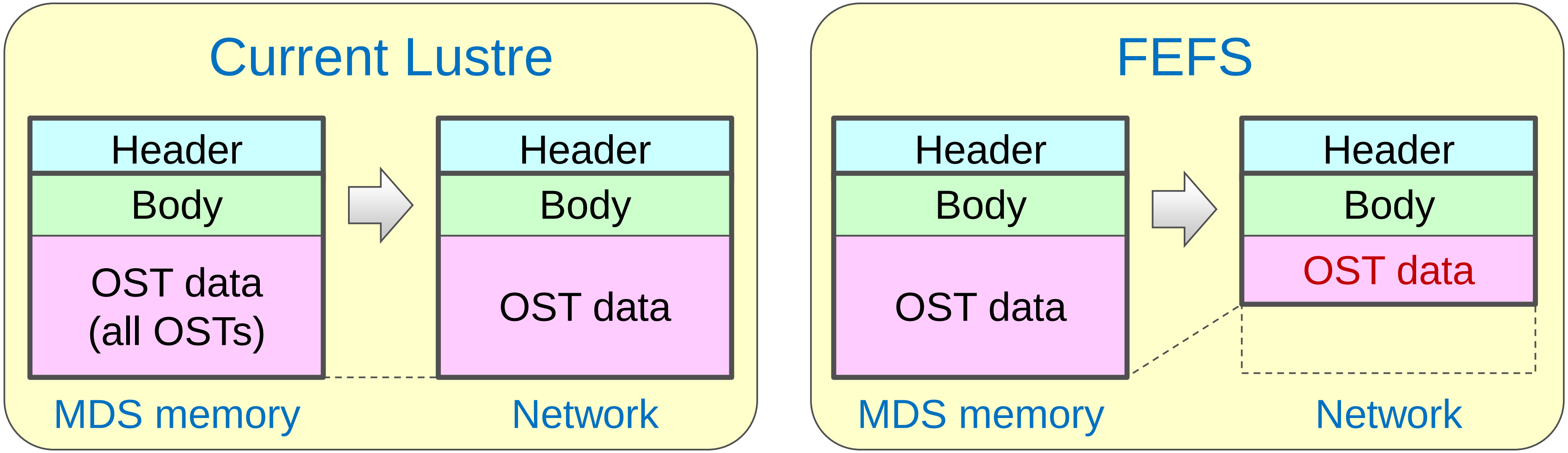
- When a client closes a file, OST data (including all OST information) is transferred from MDS to the client. ⇒Increase in proportion to #OSTs
  - OST data size = “**32B x #OST**”.

    1,000 OSTs ⇒ 31KB / close

    10,000 OSTs ⇒ 312KB / close

Our Approach

- Only send striped #OST data instead of ALL OSTs.
  - ex. Stripe count=1 ⇒ 1 OST data is sent.



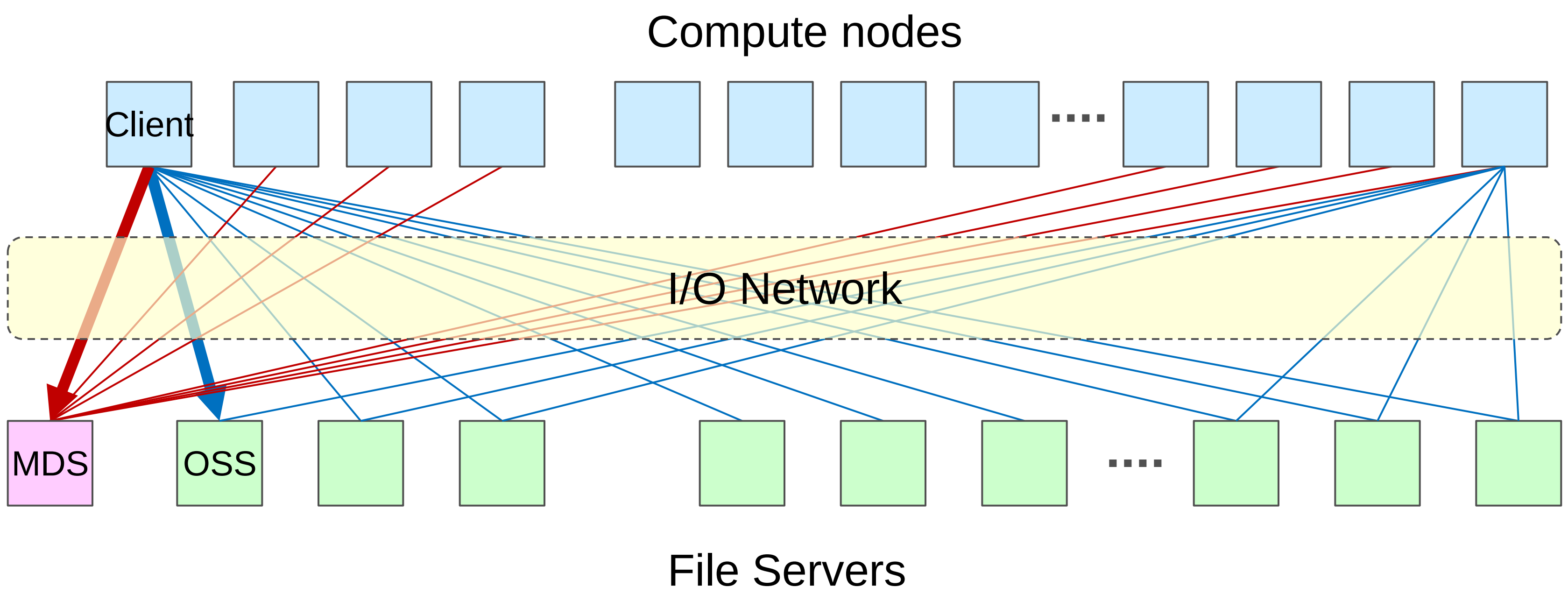
Improving Application Performance: Minimizing OS jitter

- **ll_ping Problem:**
  - All clients broadcast monitoring pings to all OSTs (not OSS) at regular intervals of 25 seconds. **100K Clients x 10K OSTs ⇒ 1M pings every 25 seconds.**
    - Vast amount of pings cause performance degradation of MPI and application.
  - Our Solution: Stopping broadcasting pings on clients.
    - Other pings, such as for recovery and for I/O confirmation, etc., are kept
- **ldlm_poold Problem:**
  - Operation time of *ldlm_poold* on client increases in proportion to the number of OSTs. *It* manages the pool of LDLM locks. It wakes up regular interval of 1sec.
  - Our Solution: Reduce the processing time per operation of *ldlm_poold* by divide the deamon's internal operation.

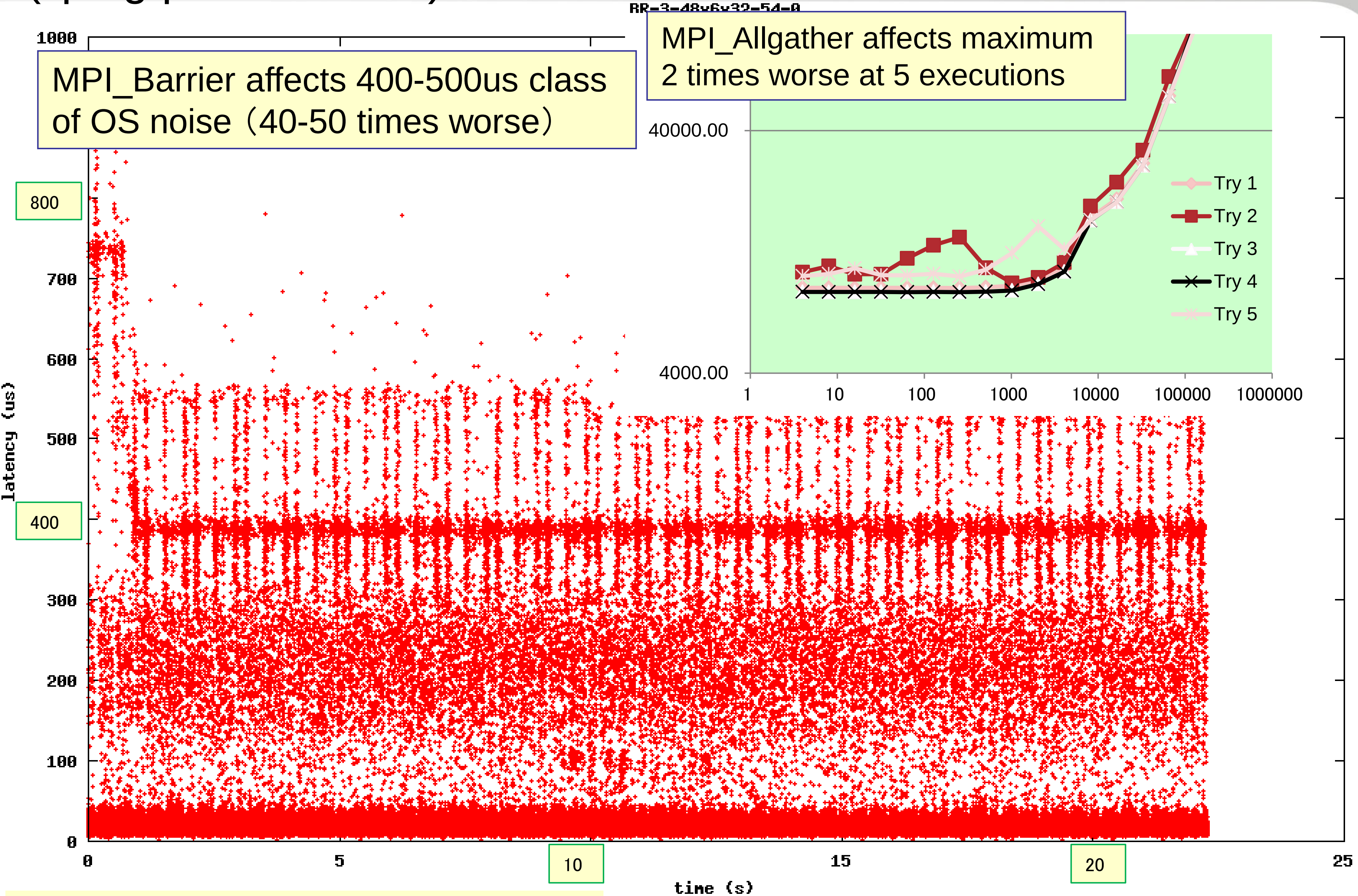
Improving Application Performance: Minimizing Network Traffic Congestion by ll_ping

- ll_ping Problem: Network congestion and request timeout cause: #of monitoring pings ∝ "#of clients x #of servers"
  - MPI and file I/O communication degradation.
  - Application performance degradation by OS jitter.

- Our Solution: Stopping interval ll_ping

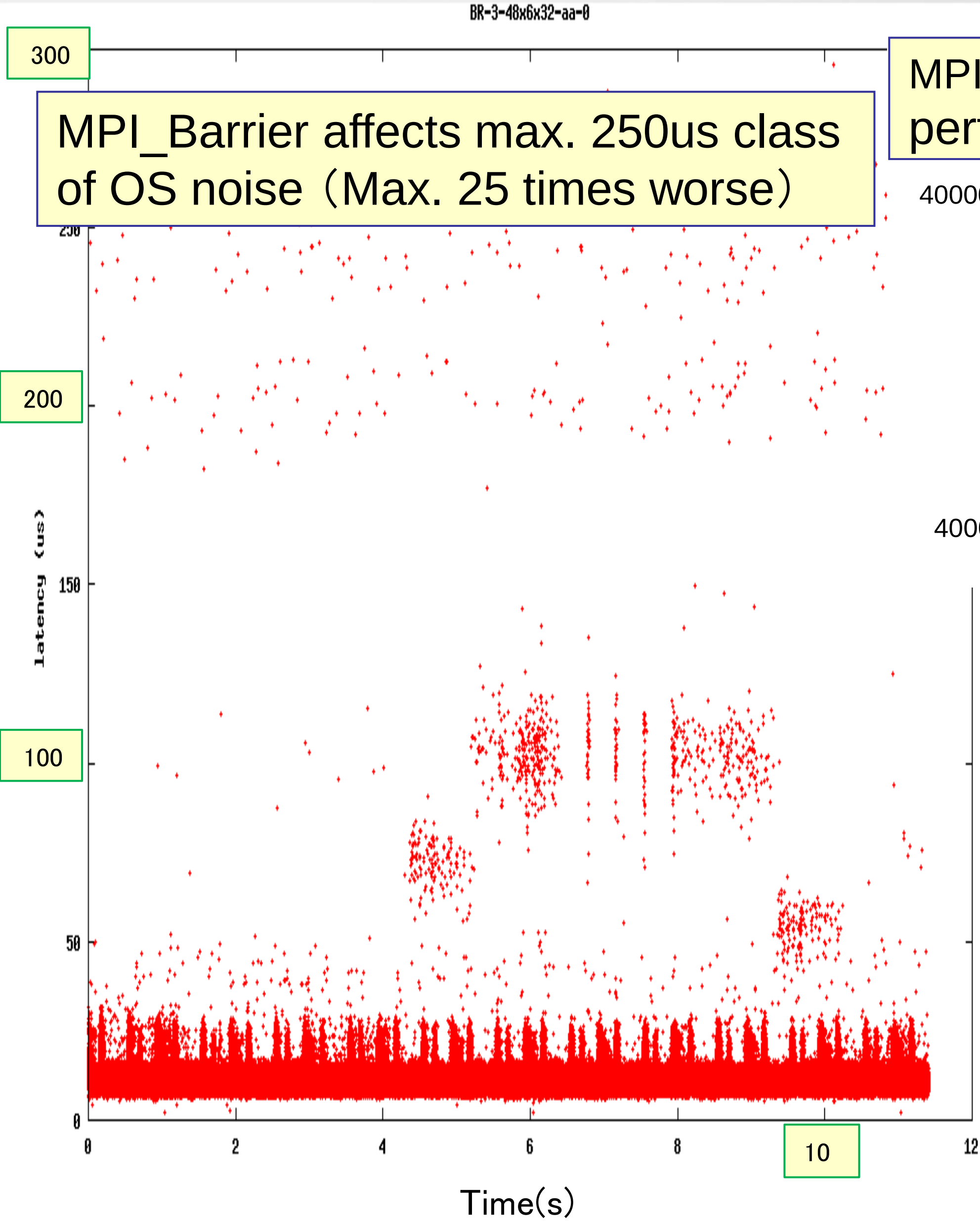


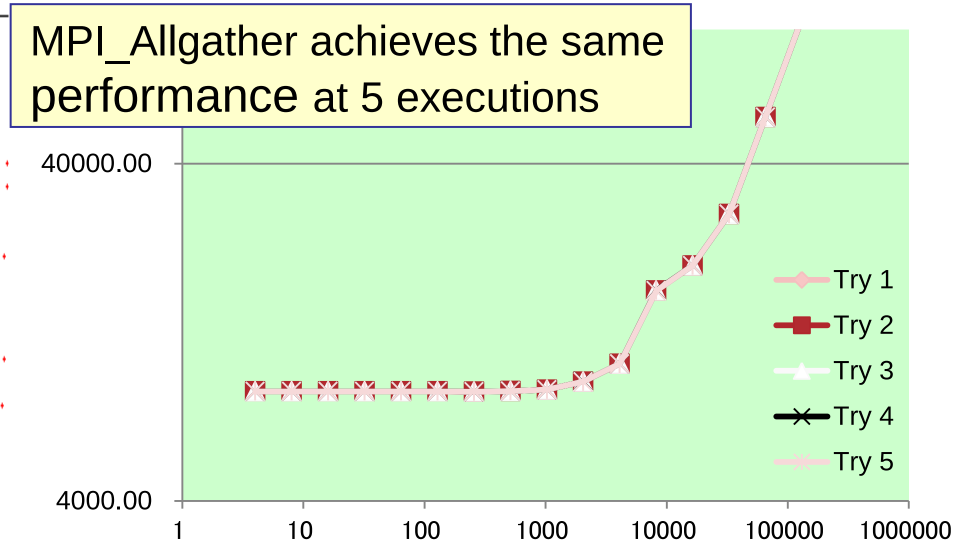
llping Impacts for MPI Performance on 1PF System (llping period 20min)



Collaborative work with RIKEN on K computer

Removing llping Affects for MPI Performance on 1PF System

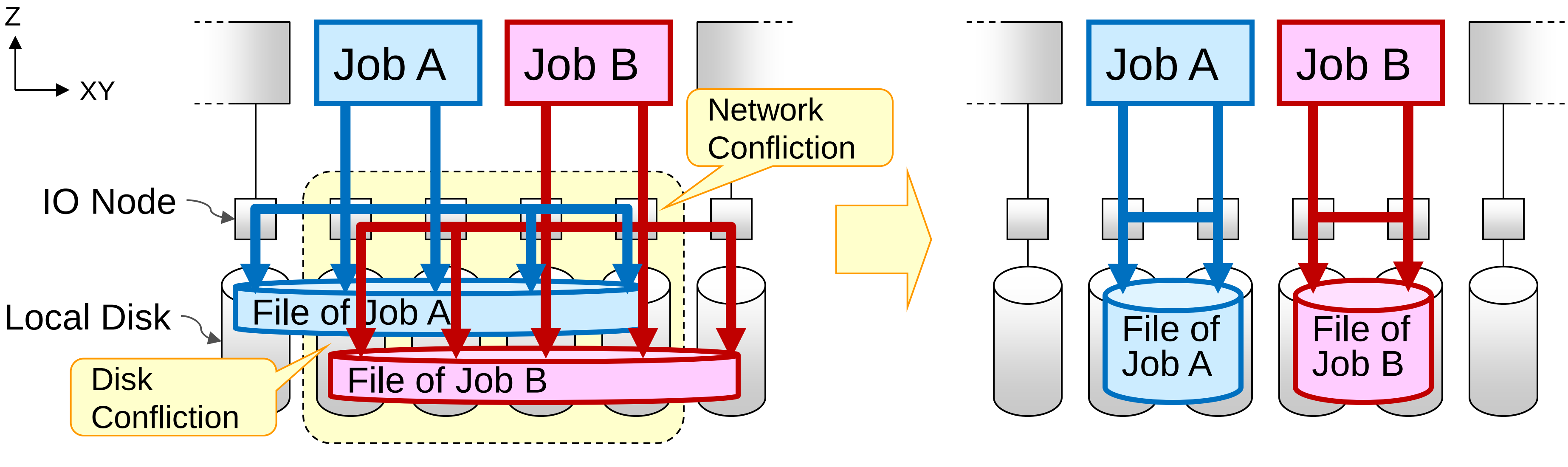




Collaborative work with RIKEN on K computer

I/O Zoning: I/O Separation among Jobs

- Issue: Job's I/O conflicts on hardware.
  - Sharing disk volumes, network links among jobs cause I/O performance degradation because of their confliction.
- Our Approach: Separate hardware among jobs.
  - Separating of disk volumes, network links among jobs as much as possible.



No-good: w/ I/O Confliction

Good: w/o I/O Confliction

FEFS Performance Evaluation on K computer

- Environment: Full system of K computer 864 Racks

- Target: Local File System:
  - OSS：IO Node（Memory 16GB） x 2,592
    - OST：ETURNUS（RAID5+0｛（4D+1P）x2｝ x2 set） x 2,592 （5,184 OST）
  - MDS： Xeon 2.00 GHz x2，Memory 64GB
    - MDT：ETRUNUS（RAID1+0（4D+4M） x4 set） x1

- Benchmark Programs:
  - IOR： w/,w/o IO Zoning
  - mdtest: MDS Performance Evaluation

Local FS I/O Performance on 10PF without I/O Zoning

FUJITSU

- IOR Results using 2,575 OSSs (w/o slow 17 OSSs)

	POSIX File/Proc	POSIX Shared File	MPI-IO Shared File
Write	965 GB/s	929 GB/s	659 GB/s
Read	1,486 GB/s	983 GB/s	847 GB/s

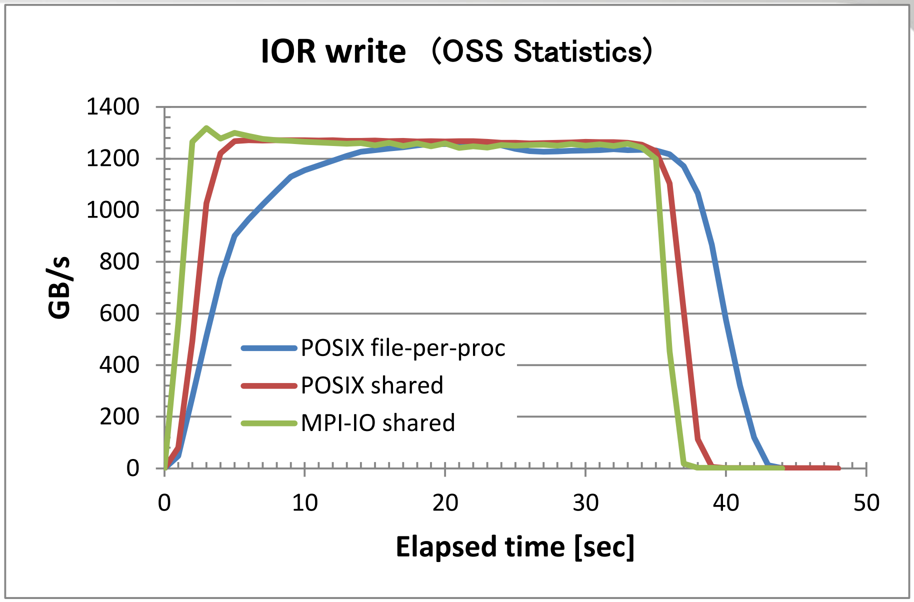
Collaborative work with RIKEN on K computer

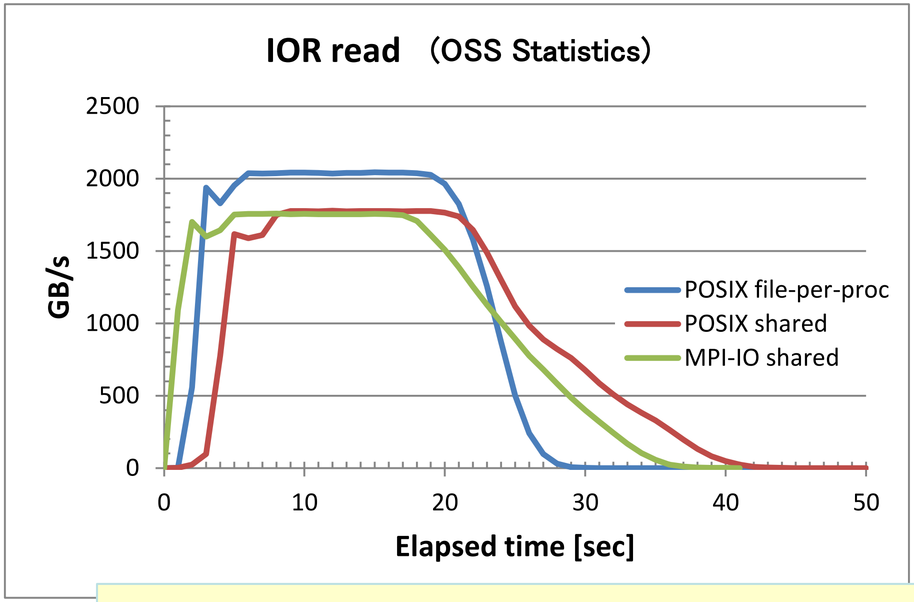
- Two Issues
  - POSIX shared file: Slow Read Performance
  - MPI-IO shared file: Slow Read/Write Performance

IOR Performance Statistics Analysis by Collectl

FUJITSU

- **Write:**
  - 1.2 TB/s Sustained Performance
  - No reason for slow MPI-IO
- **Read:**
  - Sustained Over 2.0 TB/s Performance on File/Proc
  - Sustained 1.75 TB/s on Shared/MPI-IO
    - Shared: Slow Startup/Ending
    - MPI-IO: Fast Startup/ Slow Ending
      Seems to be serialized by something.

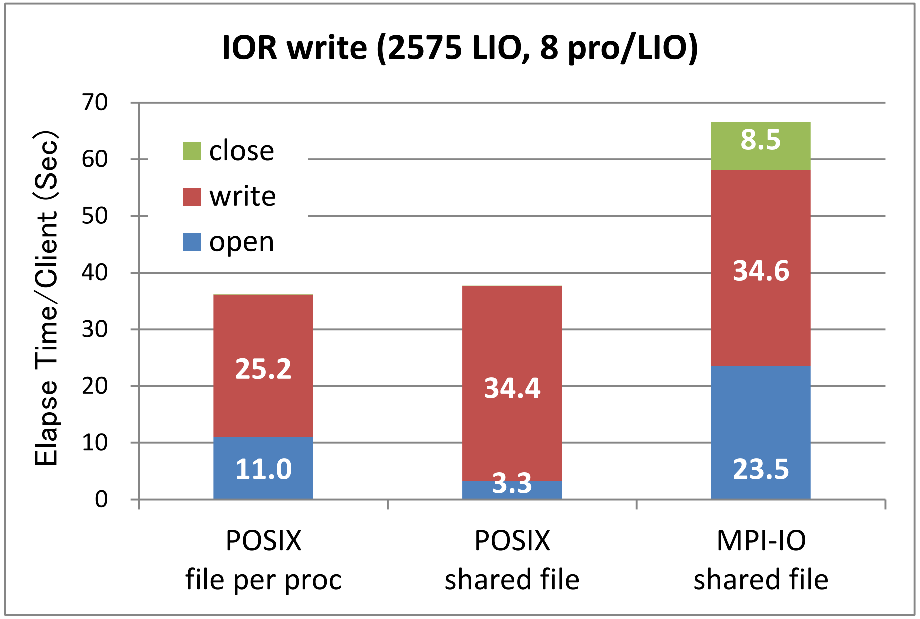


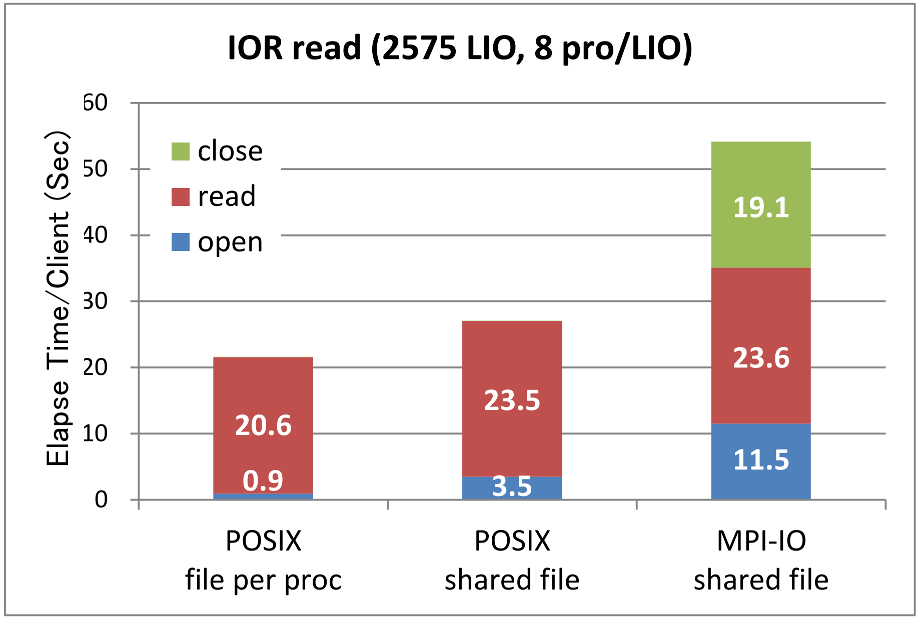


Collaborative work with RIKEN on K computer

MPI-I/O：Performance Degradation Analysis

- Measured Elapsed Time of Open-Read/Write-Close
  - The same level time of POSIX File/Proc and Shared File
  - The open and close time of MPI-IO are the reason for performance degradation.

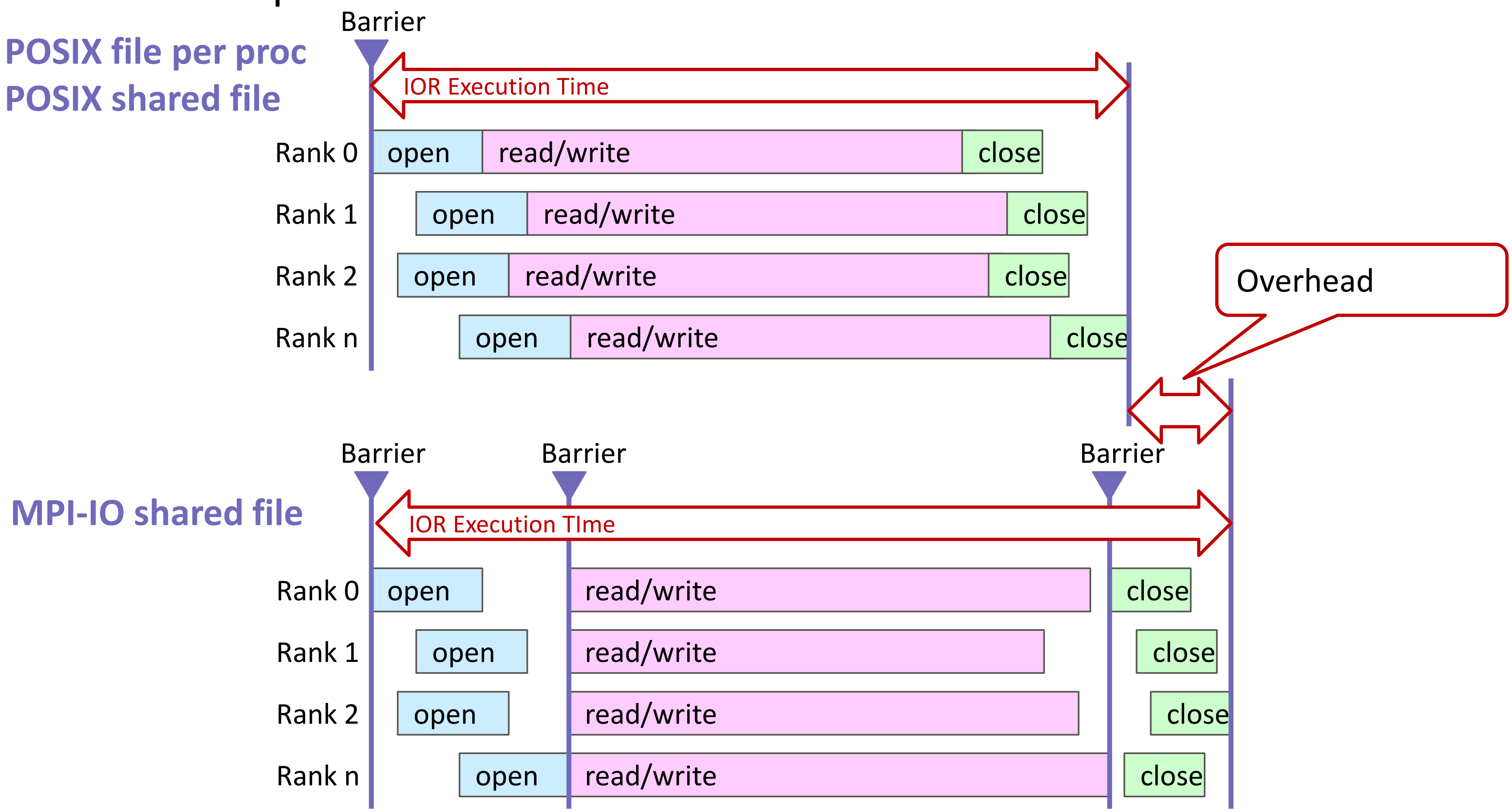




Collaborative work with RIKEN on K computer

MPI-IO: The Reason for Performance Degradation

- MPI-IO library uses barrier on open and close file
  - This means 2GB I/O is too small to realize enough bandwidth for K computer.



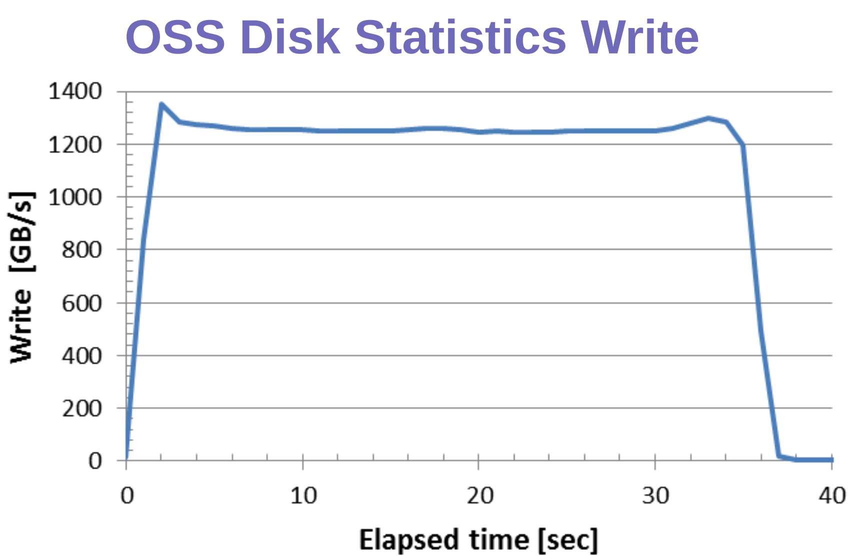
Local FS MPI I/O Performance on 10PF with I/O Zoning

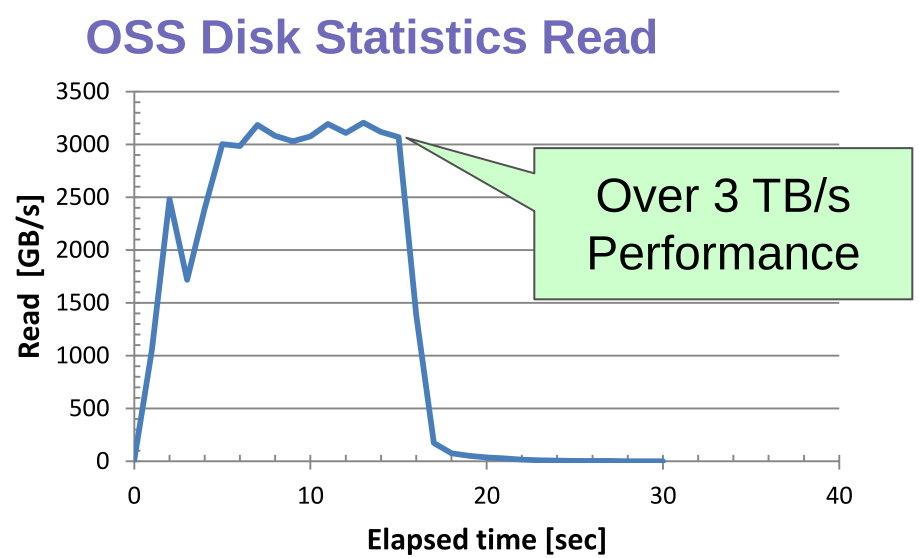
- Same Level Performance to File/Proc on Read Performance
  - I/O and Network Congestion Reduces IOR Performance
  - POSIX Shared File Performance will also speed up w/ I/O Zoning.

IOR MPI-I/O	w/ I/O Zoning	w/o I/O Zoning
Write	0.67 TB/s ⇐	0.66 GB/s
Read	1.46 TB/s ⇐	0.85 GB/s

1.35 TB/s Peak

3.2 TB/s Peak





Collaborative work with RIKEN on K computer

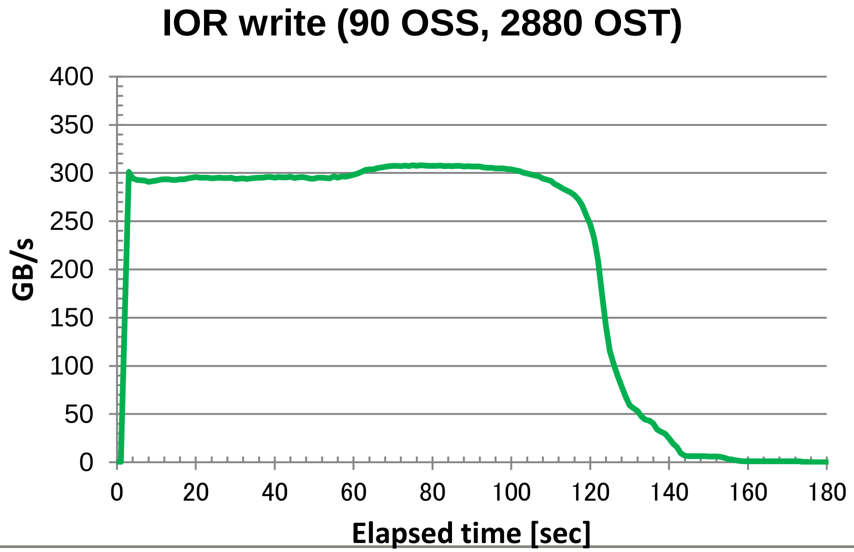
Global FS IOR Performance

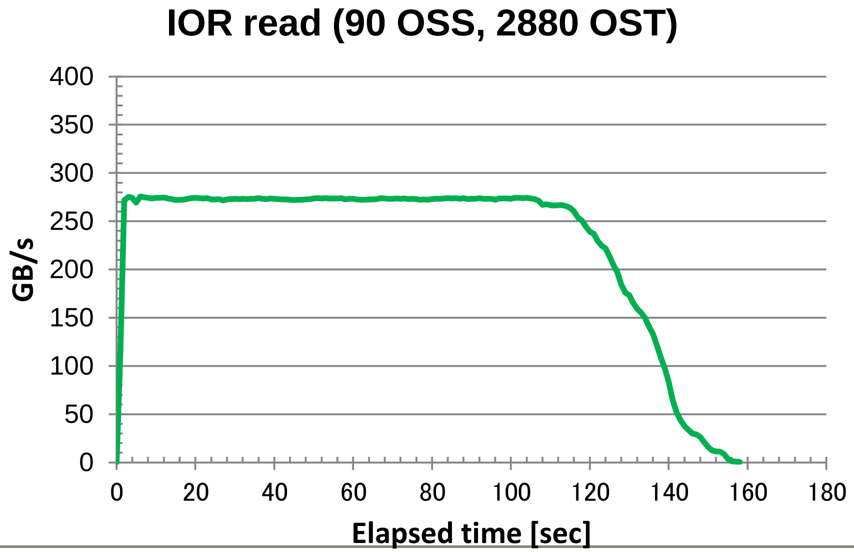
FUJITSU

- System Configuration:
  - OSS：Xeon 2.00 GHz x 2（Memory 192GB）90 Units
    - OST：ETURNUS（RAID6（6D+2P）x4 set）2800 OSTs
  - MDS：Xeon 2.00 GHz x2，Memory 64GiB
    - MDT：ETURNUS（RAID1+0（4D+4M）x4 set）2 Units

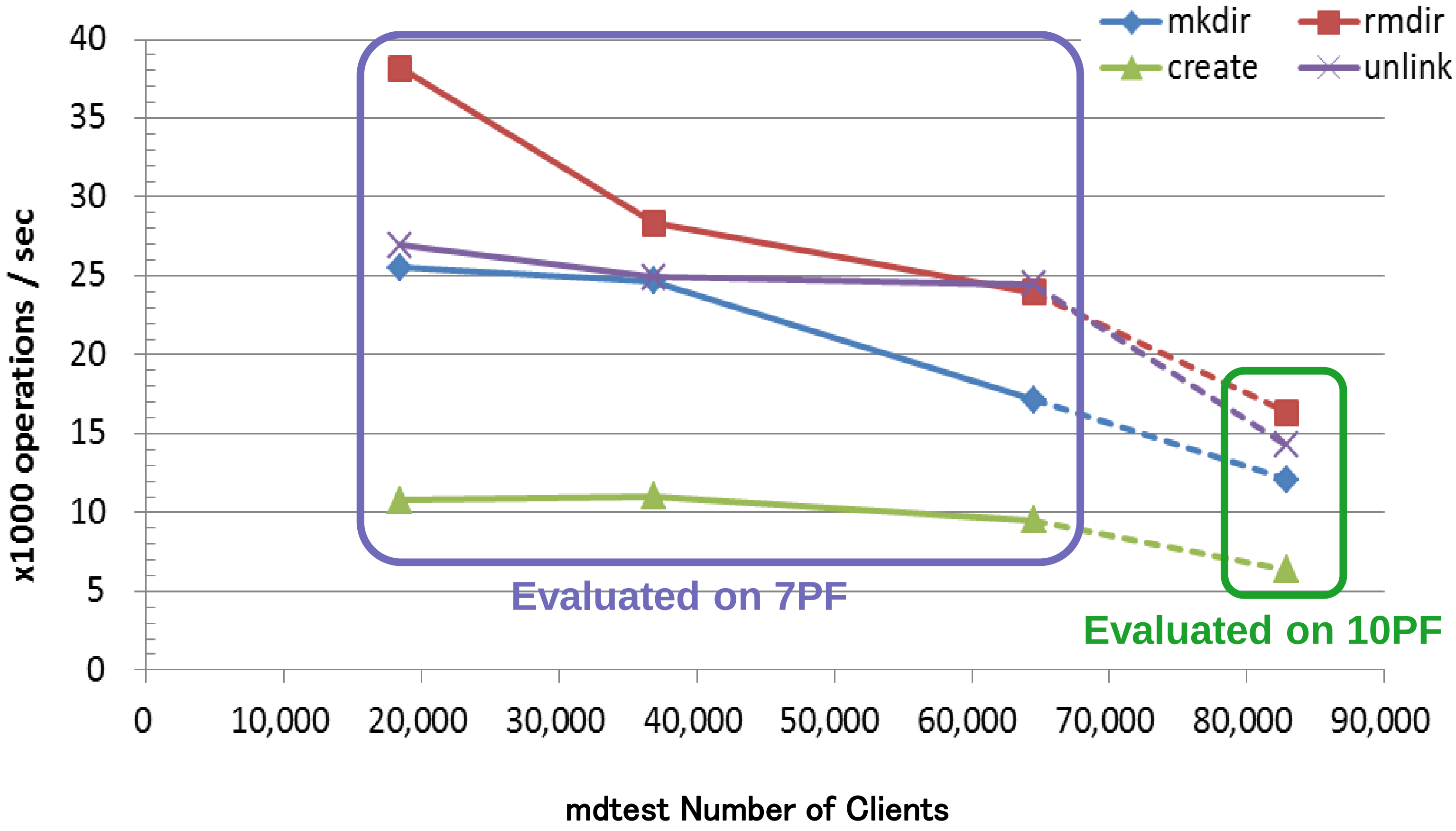
	IOR File/Proc
Write	207 GB/s
Read	235 GB/s

Collaborative work with RIKEN on K computer





Mdtest: Metadata Processing Performance



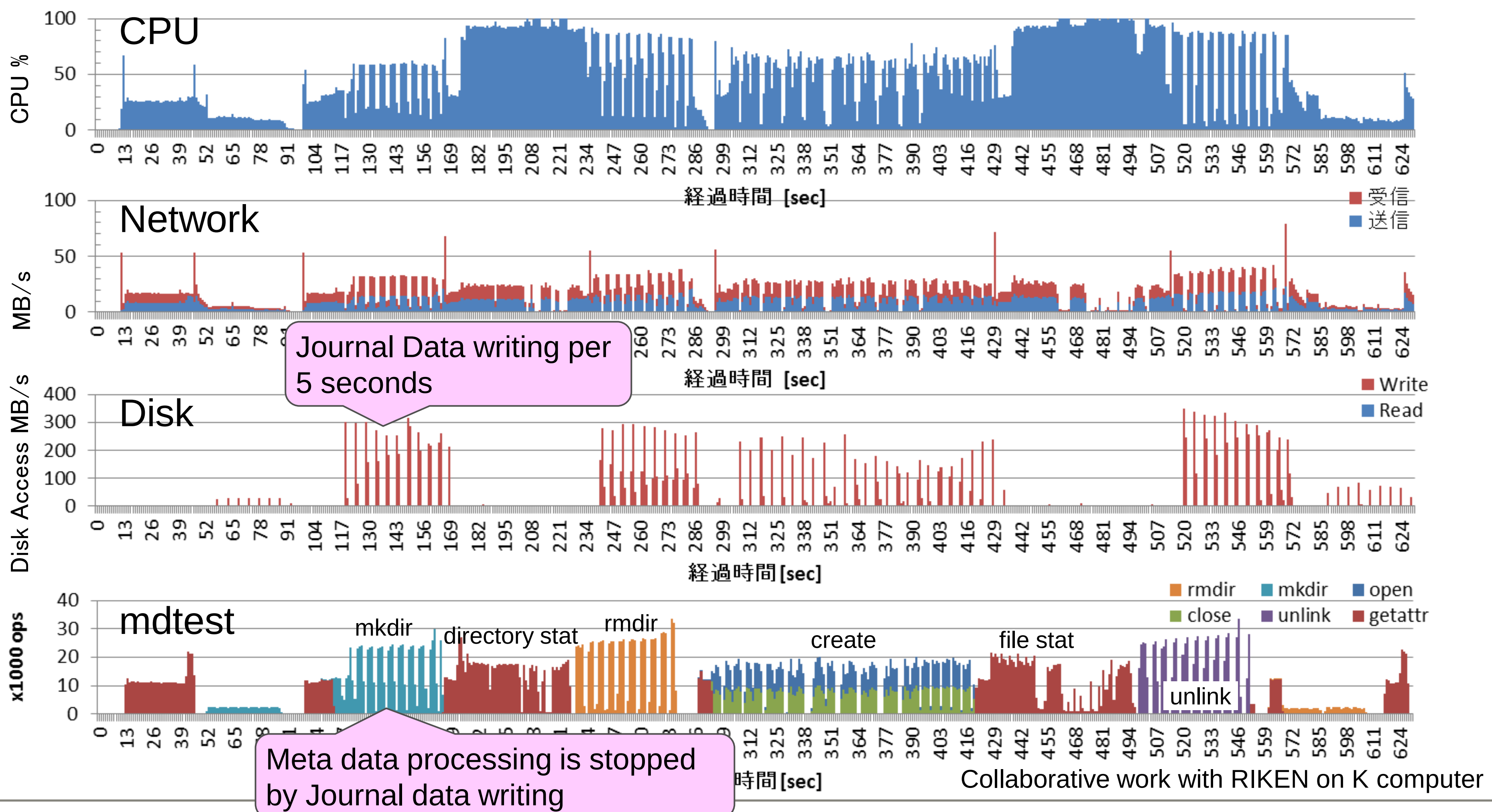
- Metadata performance degradation occurs by increasing number of clients

Collaborative work with RIKEN on K computer

Reason for Degradation of mdtest Performance

FUJITSU

- Journal data write processing per 5 seconds is the reason for mdtest performance degradation.
  - Ex: 20K ops create performance without journal writing



Collaborative work with RIKEN on K computer

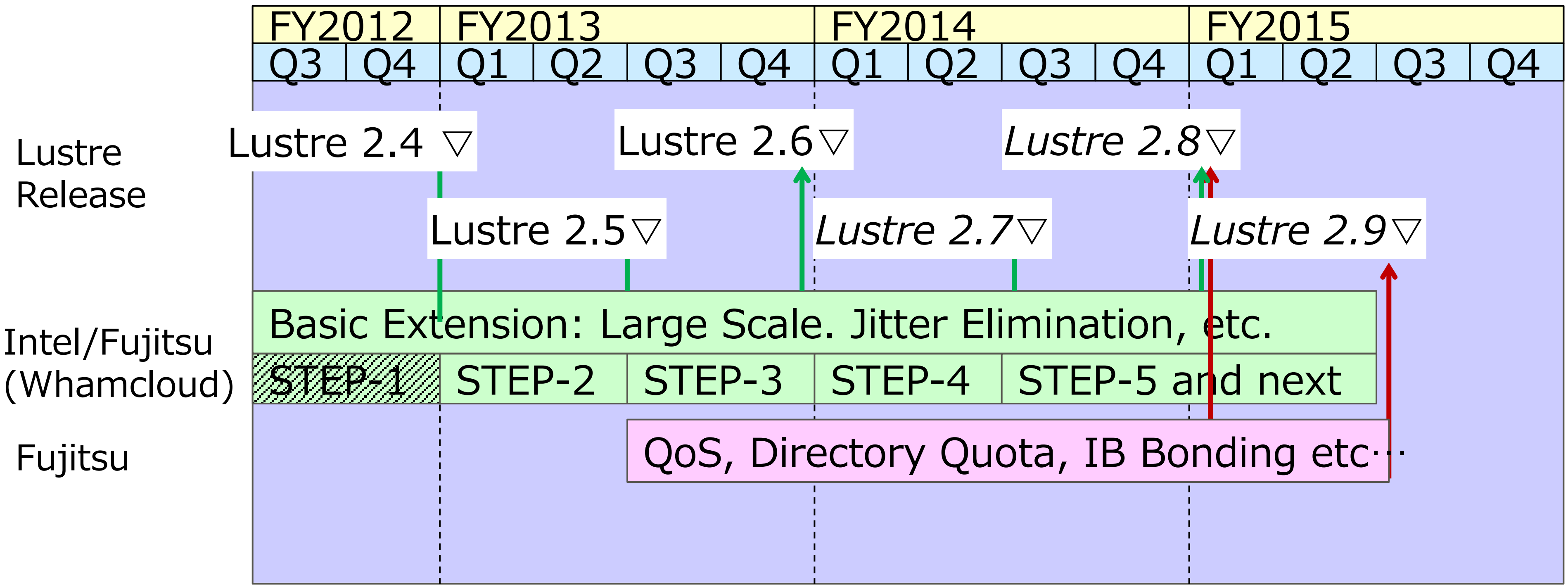
Issues Towards Exascale File System

- FEFS already has exa-byte level functions, however several problems for extra large file system.

- Resource Usage: Especially Memory
  - Client must mount whole of OSTs statically, however in MPI-IO case, a client only does file I/O into single OST. Needs to be dynamically mounted
  - Still needs to reduce memory consumption
  - Number of OSTs was reduced to half for Local FS compared with design phase
- OS Jitter
  - llping does not fit to thousands of OSTs system
- Performance Leveling among OSTs and Network Links
  - Keeping OST performance stable is very important to keep storage performance

Fujitsu's Roadmap towards Lustre 2.x

- Already started with Intel applying FEFS extension to Lustre 2.x, and will plan to finish by mid FY2015
  - Fujitsu will implement the rest of functions.



Fujitsu's Contribution Work with Intel to Lustre 2.x Roadmaps

Period	Phase	Topics
2011/12 – 2012/3	Selection of Fujitsu's Extensions to Lustre 2.x by Intel	20 of 60 items selected
2012/4 – 2012/6	Making Proposals by Intel	Three Phase Proposal
2012/9 – 2013/3	First Phase	Architecture Interoperability, LNET and OS Jitter update
2013/4 – 2013/9	Second Phase	Memory Management
2013/10 – 2014/3	Third Phase	Large Scale Performance, OST management

20 Selected Fujitsu Extensions to Lustre 2.x

FUJITSU

No	Subproject / Milestone	Category	Phase
1	LNET Networks Hashing	Performance	1
2	LNET Router Priorities	RAS	1
3	LNET: Read Routing List From File	Large Scale	1
4	Optional /proc Stats Per Subsystem	Memory Reduction	2
5	Sparse OST Indexing	Sparse OST	3
6	New Layout Type to Specify All Desired OSTs	OST selection	3
7	Reduce Unnecessary MDS Data Transfer	Meta Performance	3
8	Open/Replay Handling	Memory Reduction	2
9	Add Reformatted OST at Specific Index	OST Dynamic Addition	3
10	Empty OST Removal with Quota Release	OST Dynamic Removal	3
11	Limit Lustre Memory Usage	Memory Limit	2
12	Increase Max obddev and client Counts	Large Scale	4orF
13	Fix when Converting from WR to RD Lock	Bug Fixes (fcntl)	4orF
14	Reduce ldlm_poold Execution Time	OS Jitter	1
15	Ability to Disable Pinging	OS Jitter	1
16	Opcode Time Execution Histogram	For Debug	4orF
17	Endianness Fixes	Architecture Inter-op.	1
18	OSC Request Pool Disabling	Memory Reduction	2
19	Pinned Pages Waiting for Commit	Memory Reduction	2
20	Errno Translation Tables	Architecture Inter-op	1

Summary and Future Work

- We described performance evaluation of FEFS on 'K computer' developed by RIKEN and Fujitsu.
  - Over 1.4 TB/s performance (Over 3TB/s Read Performance except starting up and ending time)

- Future Work
  - Rebase to newer version of Lustre (2.x)
  - Continue to Contribute our extensions to Lustre Community

FUJITSU

shaping tomorrow with you