Checkpointing and Lustre

Kento Sato

Tokyo Institute of Technology

LLNL-PRES-644916

APAC LUG 2013

October 17th, 2013

Outline

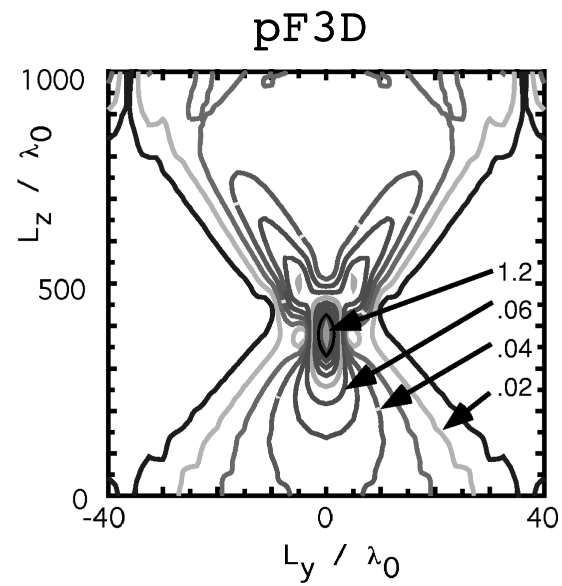
- Failures on HPC systems
- Challenges on Checkpoint/Restart
- Two approaches
  - Multi-level Checkpoint/Restart
  - Storage design
- Summary

Failures on HPC systems

- System resiliency is critical for future extreme-scale computing
- 191 failures out of 5-million node-hours
  - A production application: Laser-plasma interaction code (pF3D)
  - Hera, Atlas and Coastal clusters @LLNL

Estimated MTBF (If no hardware reliability improvement)

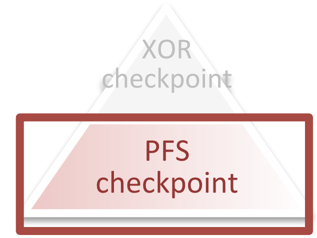
	1,000 nodes	10,000 nodes	100,000 nodes
MTBF	1.2 days	2.9 hours	17 minutes



Sourece: Berger, R. L., Still, C. H., Williams, E. A. and Langdon, A. B.: On the Dominant and Subdominant Behavior of Stimulated Raman and Brillouin Scattering Driven by Nonuniform Laser Beams (Physics of Plasmas 1998)

Lawrence Livermore National Laboratory

- Difficult to continuously run for a long time without fault tolerance



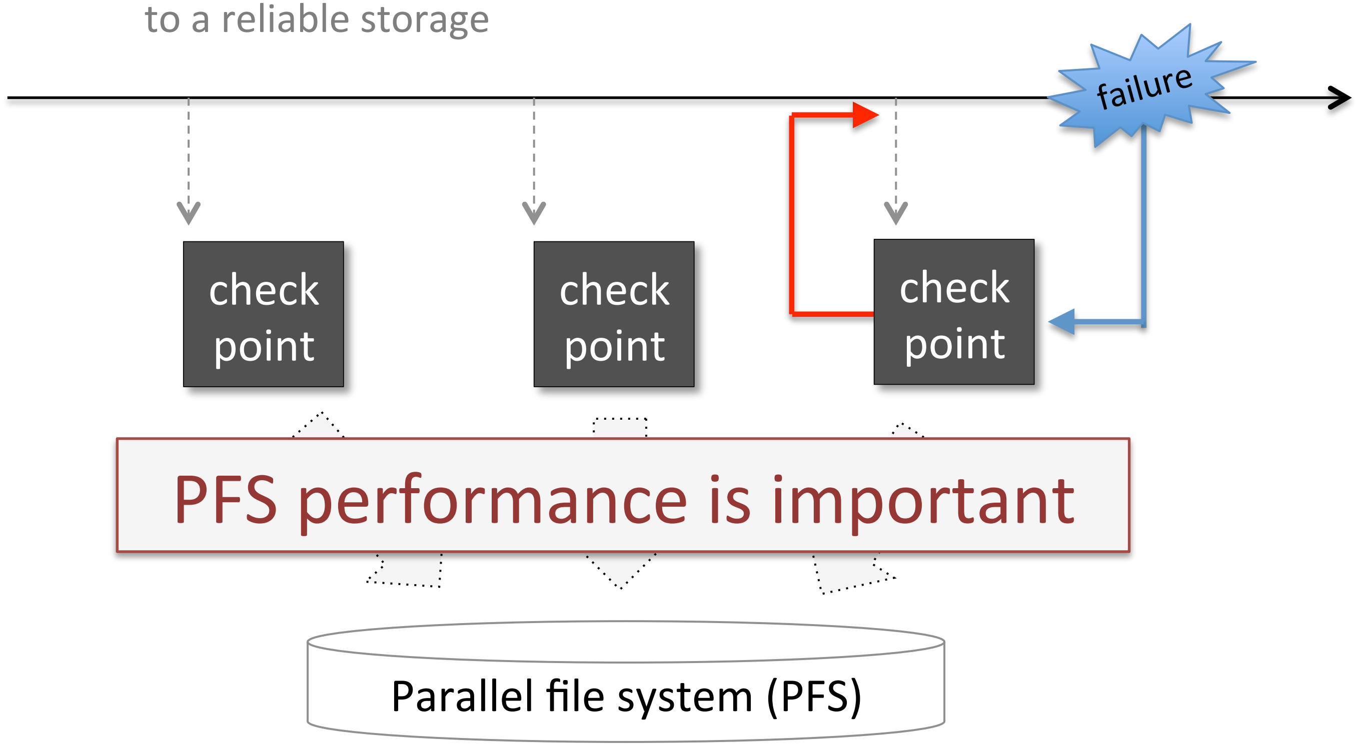
Checkpoint/Restart

Checkpoint

Periodically save a snapshot of an application state to a reliable storage

Restart

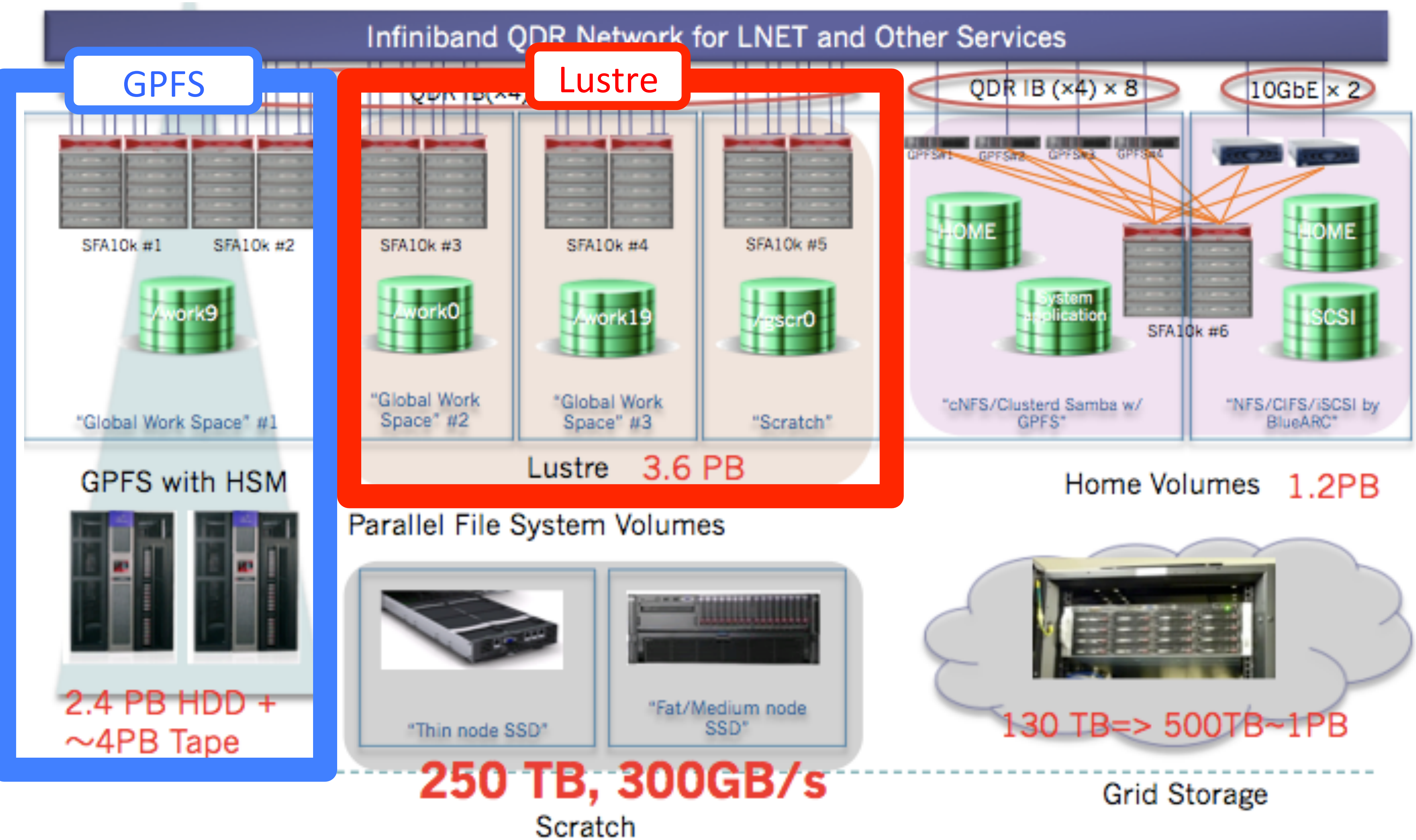
On a failure, restart the execution from the latest checkpoint



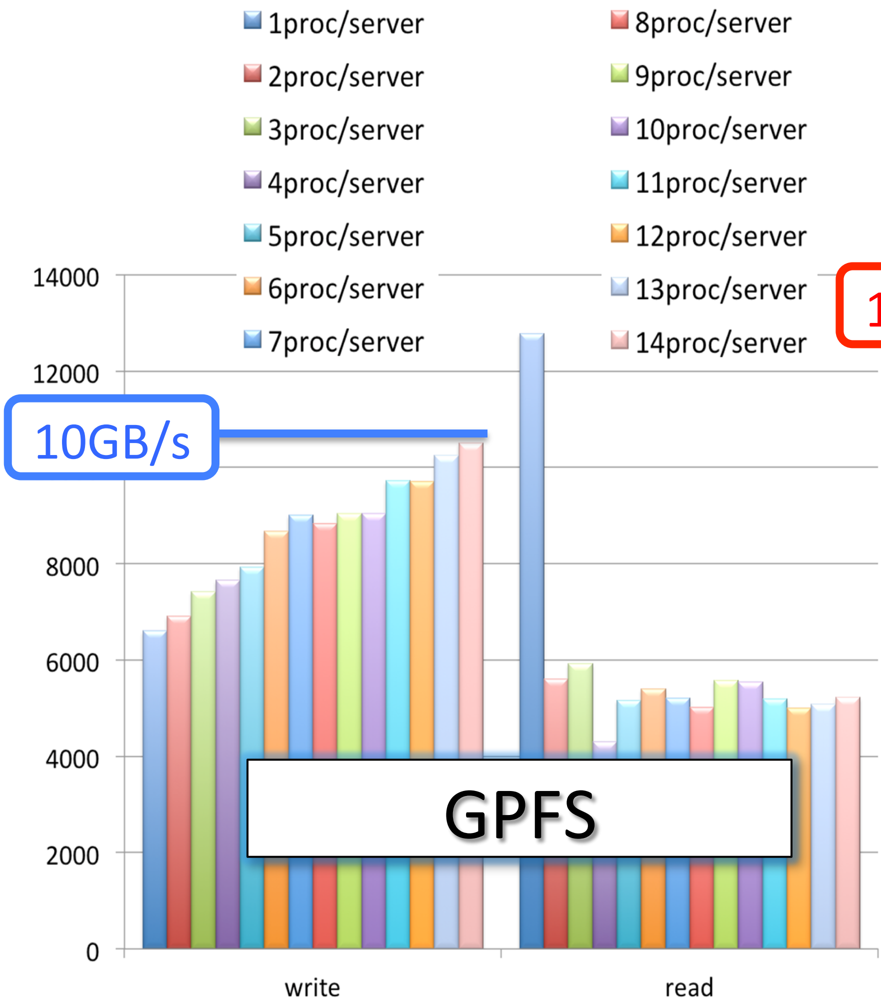
Mostly these checkpoints are stored in a PFS

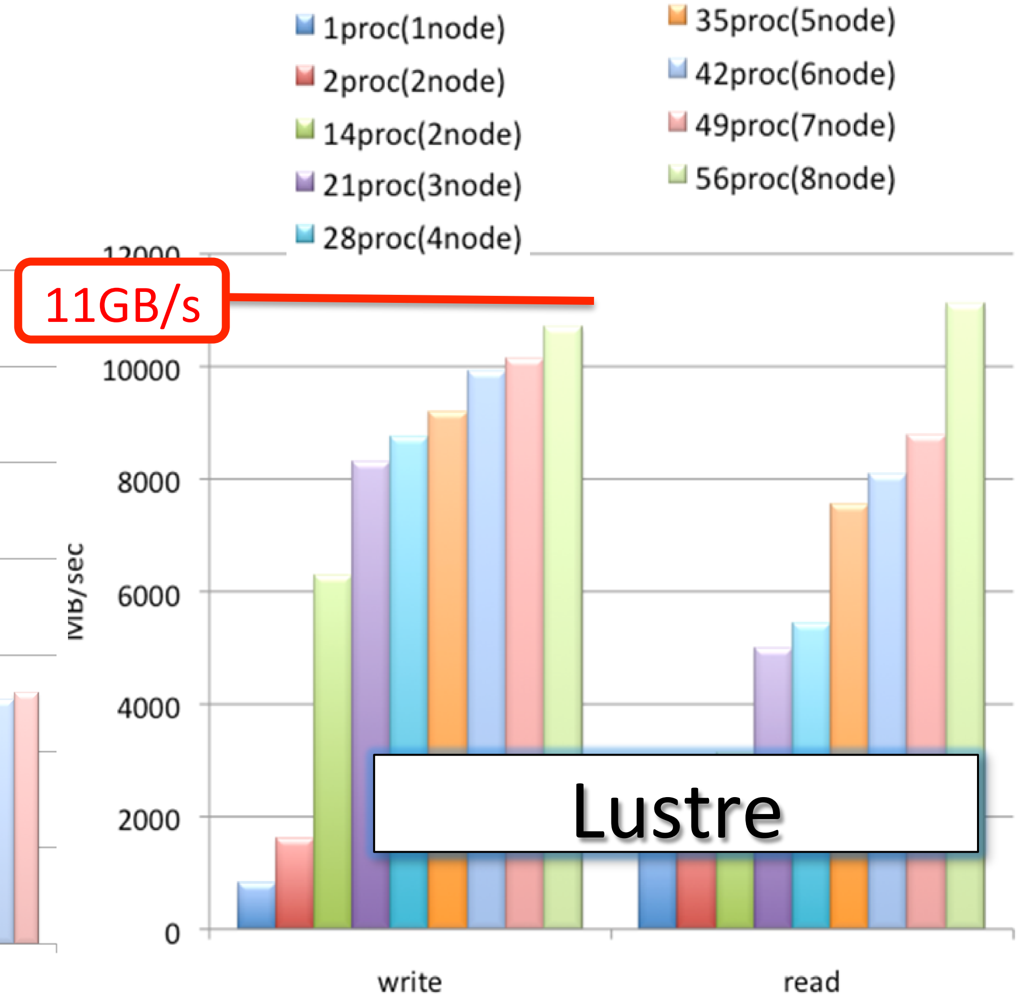
TSUBAME2.0/2.5 Storage Overview

TSUBAME2.0 Storage 11PB (7PB HDD, 4PB Tape)



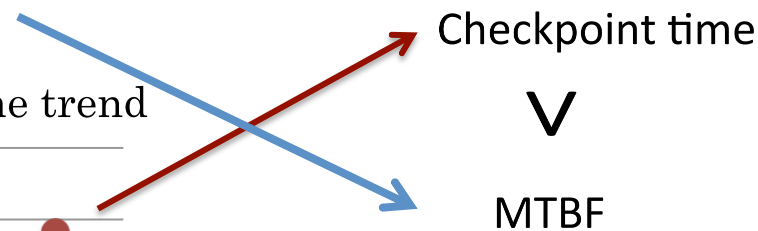
TSUBAME2.0 PFS Performance
- checkpoint & restart -

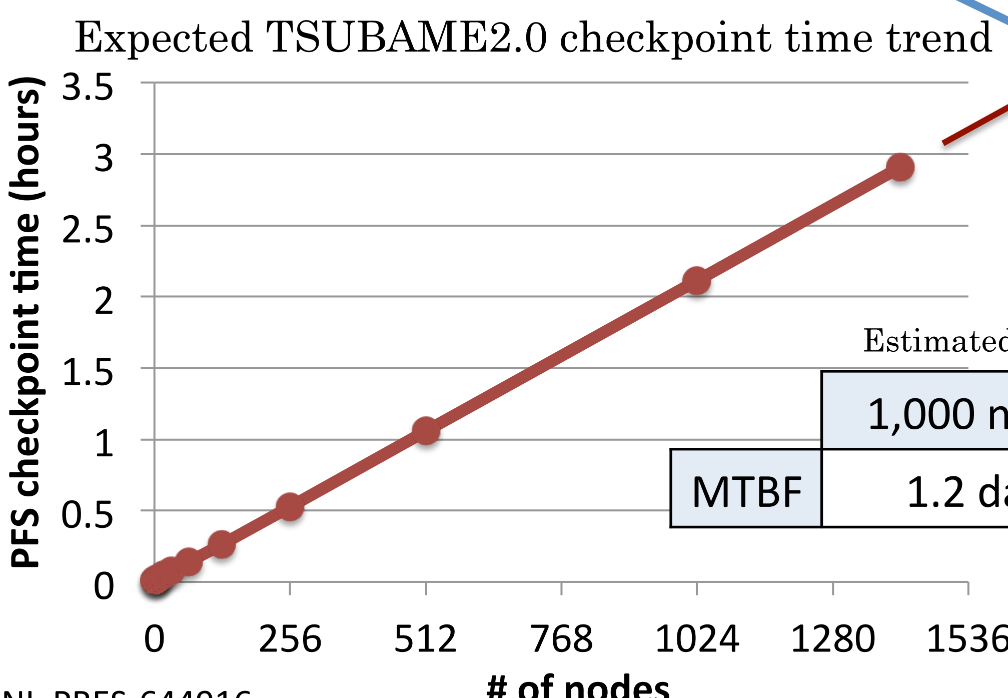




HPC applications require more bandwidth

- We scale out the system, Both checkpointing time and failure rate increases





Estimated MTBF (If no hardware reliability improvement)

	1,000 nodes	10,000 nodes	100,000 nodes
MTBF	1.2 days	2.9 hours	17 minutes

For fast checkpointing

- Buy many & fast PFSs

10 Lustre file systems at LLNL

Lawrence Livermore National Laboratory

- DOE applications sometimes run for days or weeks

1TB/s

70PB

OCF File System	Bandwith (GB/s)	Capacity (PB)	OSS Nodes	OSTs
lscratchrzb	18	1.2	16	16
lscratchc	40	1.8	32	480
lscratchd	50	2	40	600
lscratche	18	1.2	16	16
lscratchv	106	6.7	96	96
SCF File System	**Bandwith (GB/s)**	**Capacity (PB)**	**OSS Nodes**	**OSTs**
lscratch1	850	53	768	768
lscratch2	70	2.7	56	840
lscratch4	60	2.3	48	720
lscratch5	80	3.4	64	960
lscratch6	32	2.4	32	96

10 Lustre file systems at LLNL

Lawrence Livermore National Laboratory

OCF Maximum Lustre Bandwidths (GB/s)				
OCF System (CZ)	lscratchc	lscratchd	lscratche	lscratchv*
Ansel	12	12	18	10
Aztec	.125	.125	.125	.125
Cab	40	20	18	10
Edge	10	10	18	10
Herd	1.25	1.25	1.25	1.25
OSLIC	1.25	1.25	1.25	1.25
Sierra	40	20	18	10
Vulcan	–	–	–	106

SCF Maximum Lustre Bandwidths (GB/s)					
SCF System	lscratch1*	lscratch2	lscratch4	lscratch5	lscratch6
Coastal	40	15	40	40	32
CSLIC	1.25	1.25	1.25	1.25	1.25
Graph	15	20	15	15	15
Inca	.125	.125	.125	.125	.125
Juno	40	15	40	40	32
Muir	40	15	40	40	32
Sequoia	850	–	–	–	–
Zin	100	15	60	80	32

OCF System (RZ)	lscratchrzb
RZMerl	18
RZCereal	10
RZGPU	12
RZSLIC	1.25
RZuSeq	12
RZZeus	10

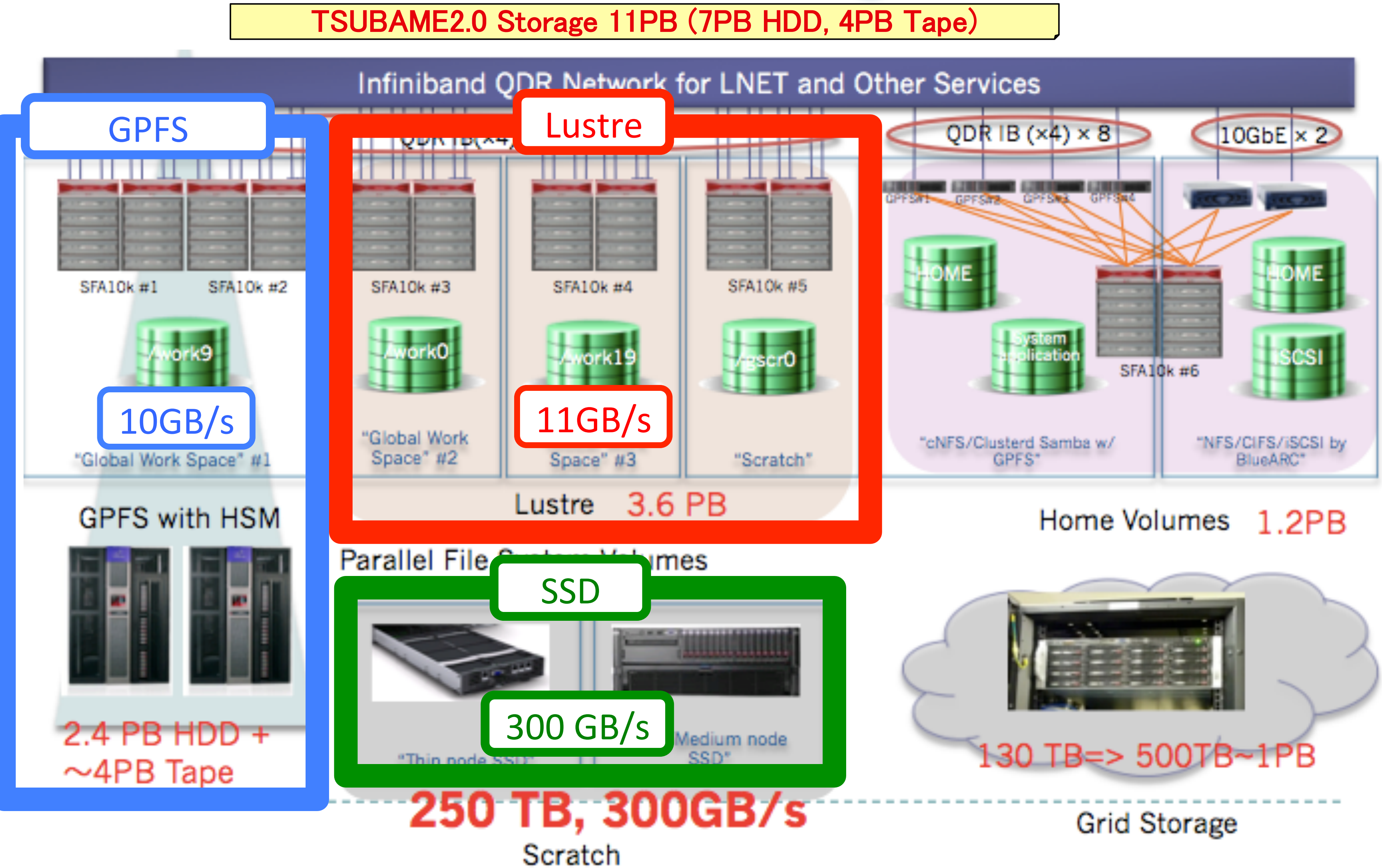
- 22 systems shares 10 Lustre
  - Unstable performance
- Sequoia checkpointing time
  - 1.5 PB memory / 850 ~= 5 hours

For fast checkpointing

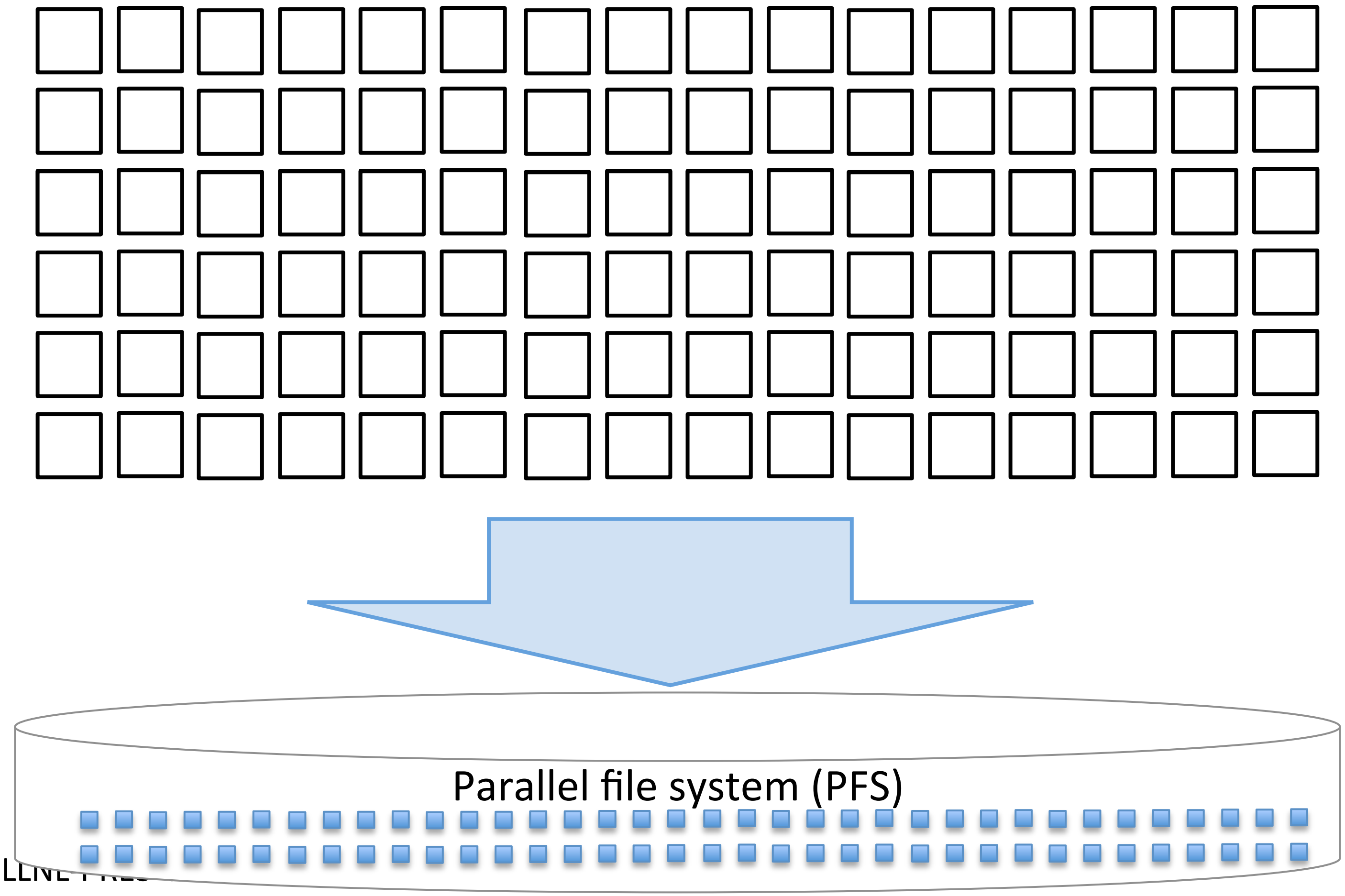
- Buy many & fast PFSs

- Local storage

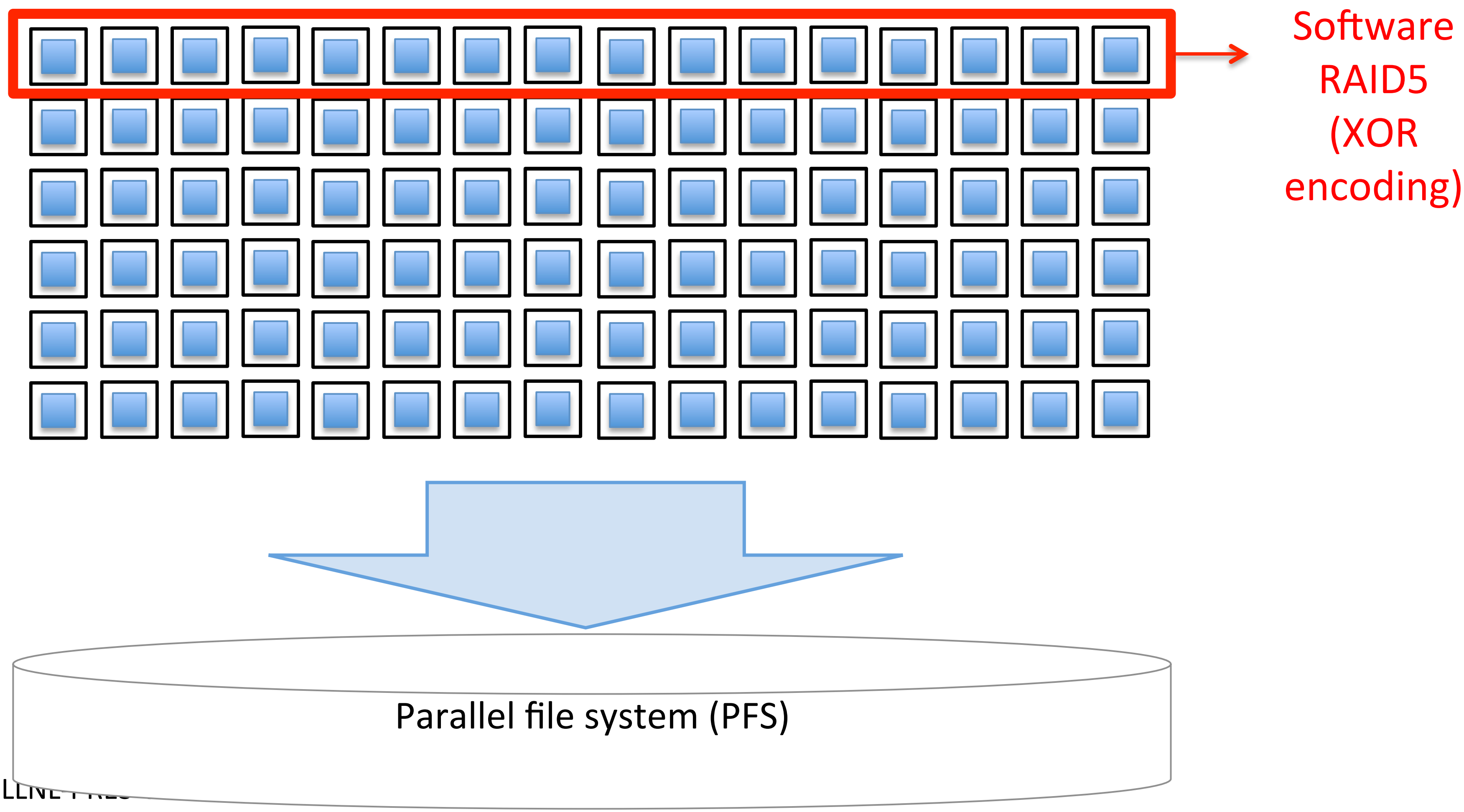
TSUBAME2.0 & 2.5 Storage Overview



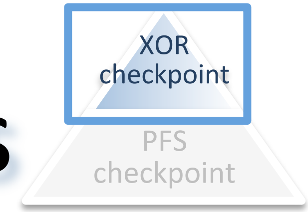
Checkpointing to Local-storage



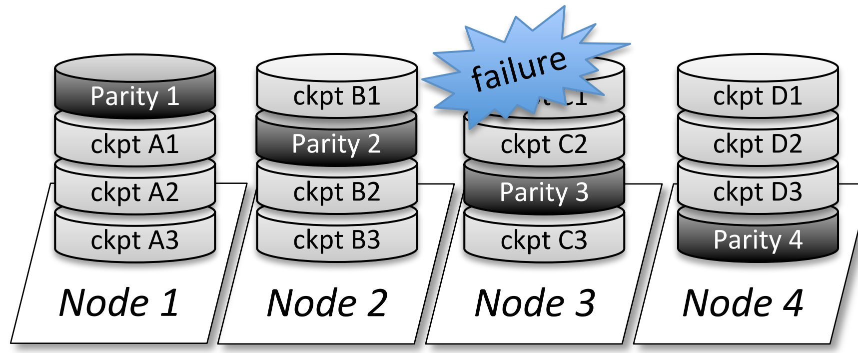
Checkpointing to Local-storage



Scalable checkpointing methods



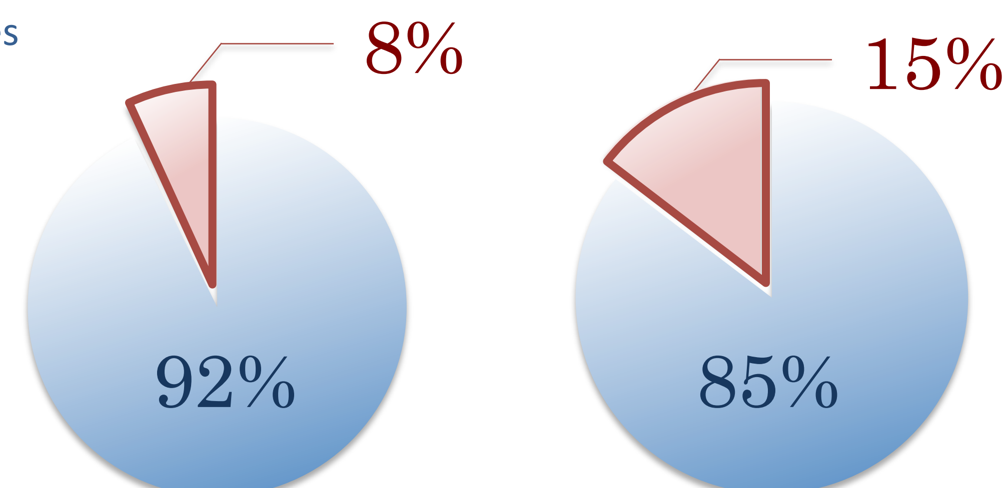
- Diskless checkpoint:
  - Create redundant data across local storages on compute nodes using a encoding technique such as XOR
  - Can restore lost checkpoints on a failure caused by small # of nodes like RAID-5



XOR encoding example

- Most of failures comes from one node, or can recover by XOR checkpoint
  - e.g. 1) TSUBAME2.0: 92% failures
  - e.g. 2) LLNL clusters: 85% failures

Rest of failures still require a checkpoint on a reliable PFS

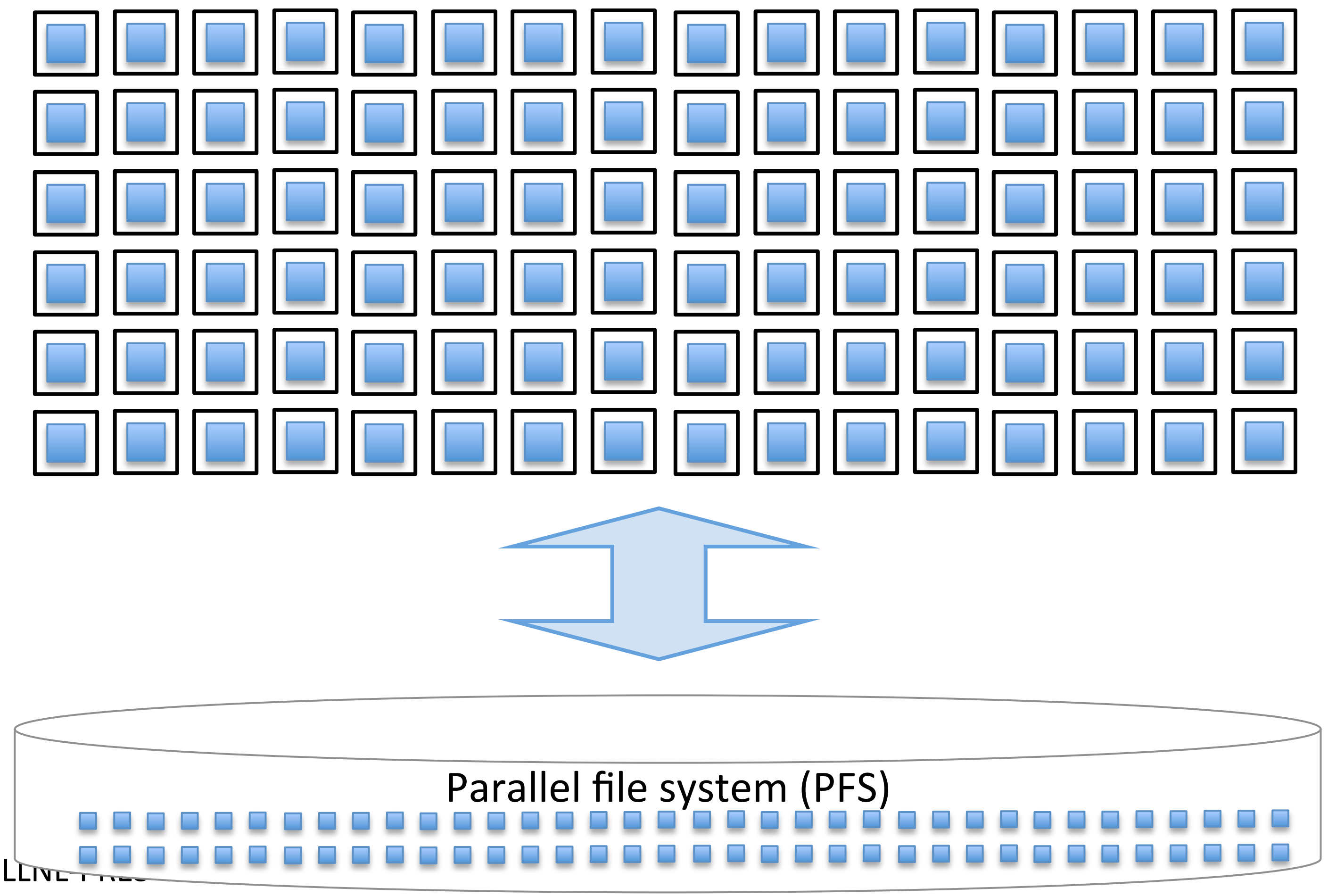


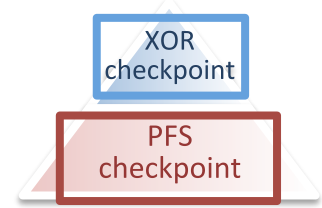
LOCAL/XOR/PARTNER checkpoint
PFS checkpoint

Diskless checkpoint is promising approach

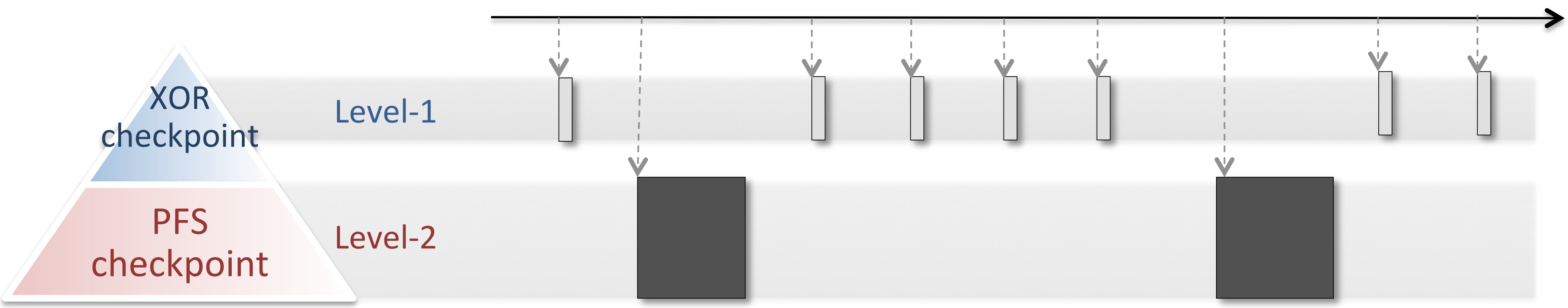
Failure analysis on TSUBAME2.0 Failure analysis on LLNL clusters

Local-storage + PFS





Multi-level checkpointing (MLC)

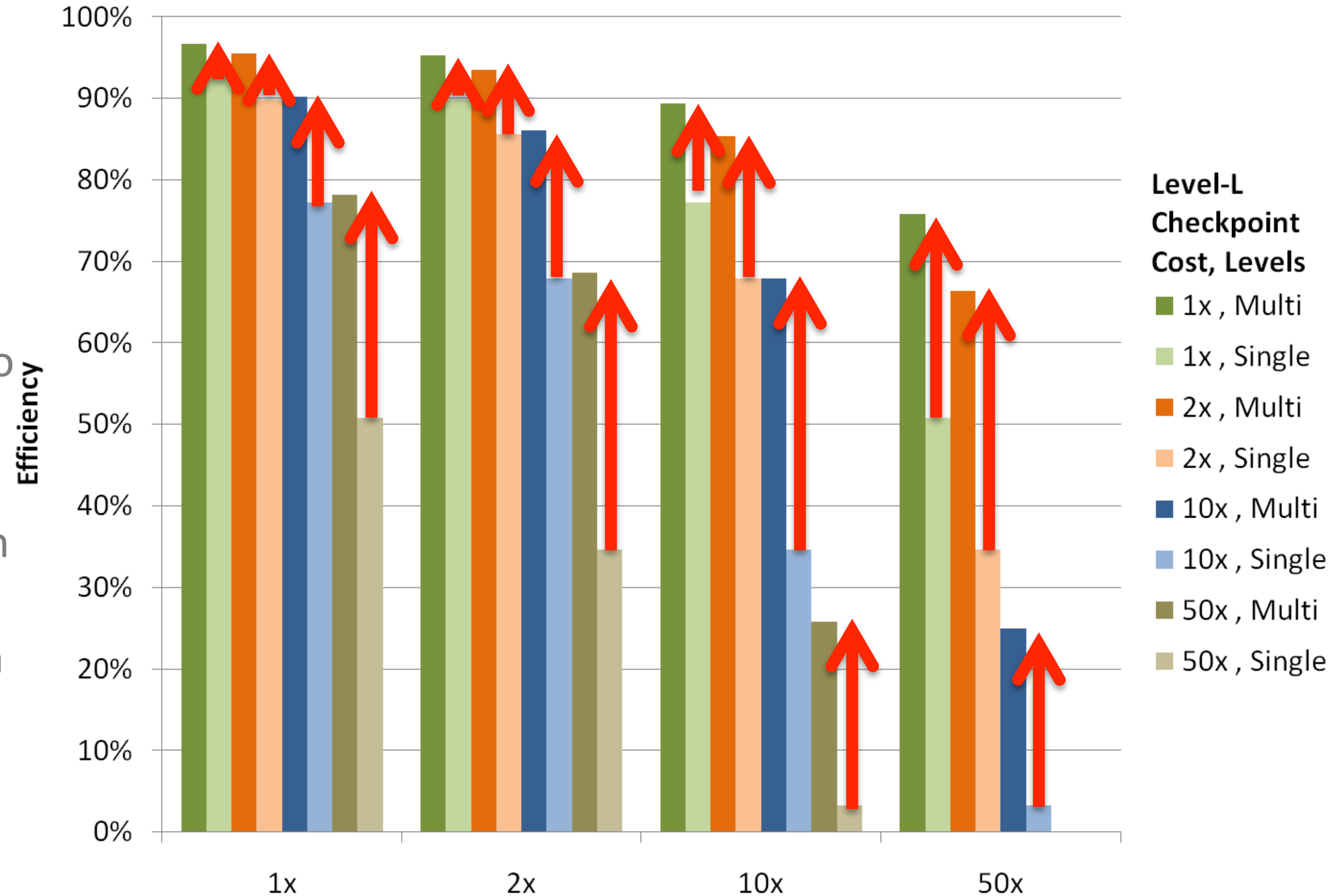


- Use storage levels hierarchically
  - XOR checkpoint: Frequently
    - for one node or a few node failure
  - PFS checkpoint: Less frequently
    - for multi-node failure

Multi-level checkpointing (MLC)

- MLC significantly improves system efficiency
  - Increase failure rate up to 50 times, but still high efficiency
  - one order of magnitude in 50 times higher failure rate

- Efficiency compared to single-level checkpointing
- Efficiency is how much ratio an application spend its computation except C/R



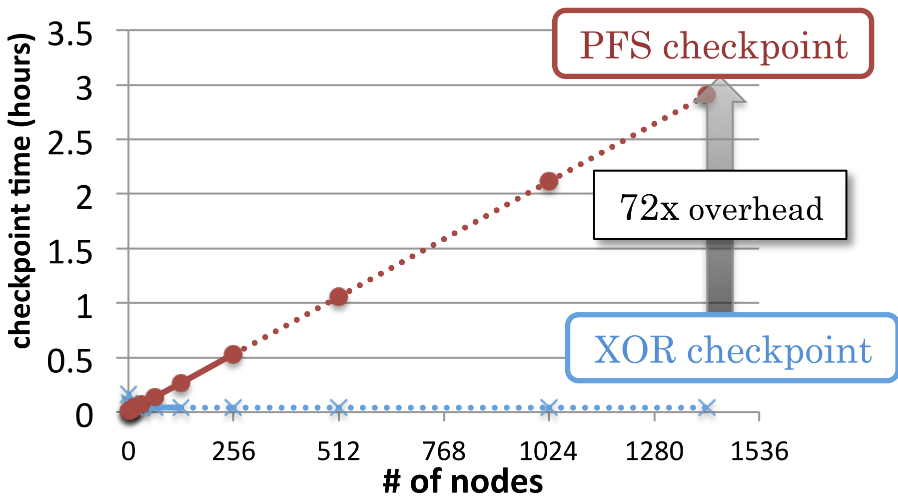
Source: A. Moody, G. Bronevetsky, K. Mohror, and B. R. de Supinski, "Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System," in Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC 10).

MLC Problems on Petascale or larger

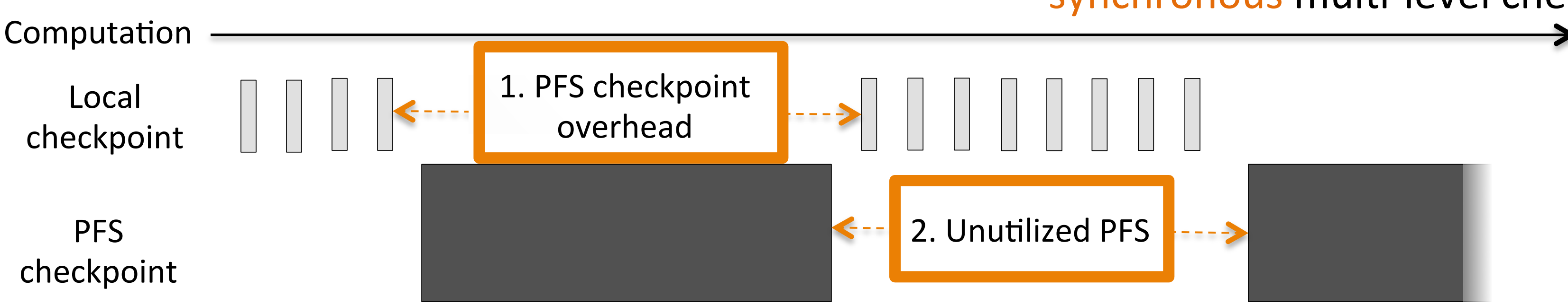
two potential problems

1. PFS checkpoint overhead
   – Even with MLC, PFS checkpoint still becomes big overhead
2. Inefficient PFS utilization
   – Time between PFS checkpoints becomes long, PFS is not utilized during XOR checkpoints

TSUBAME2.0 checkpoint time trend

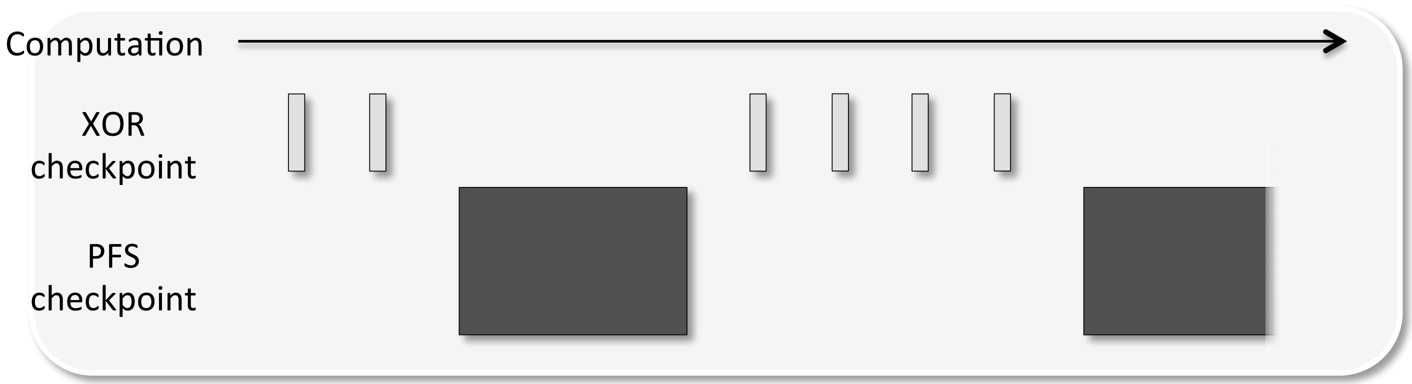


synchronous multi-level checkpointing

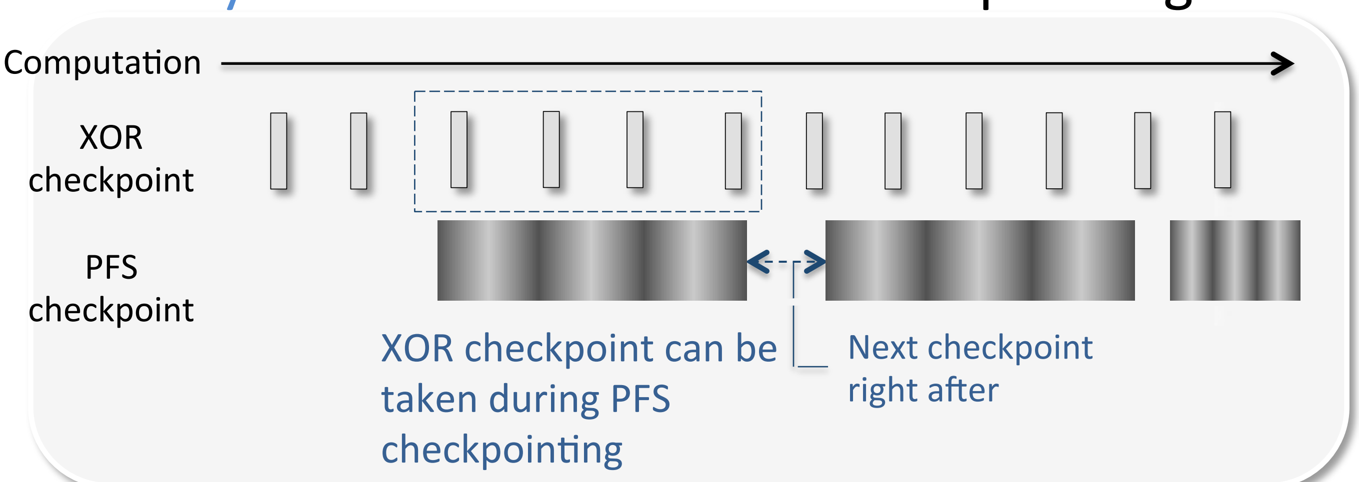


Asynchronous checkpointing overview

Synchronous multi-level checkpointing checkpointing



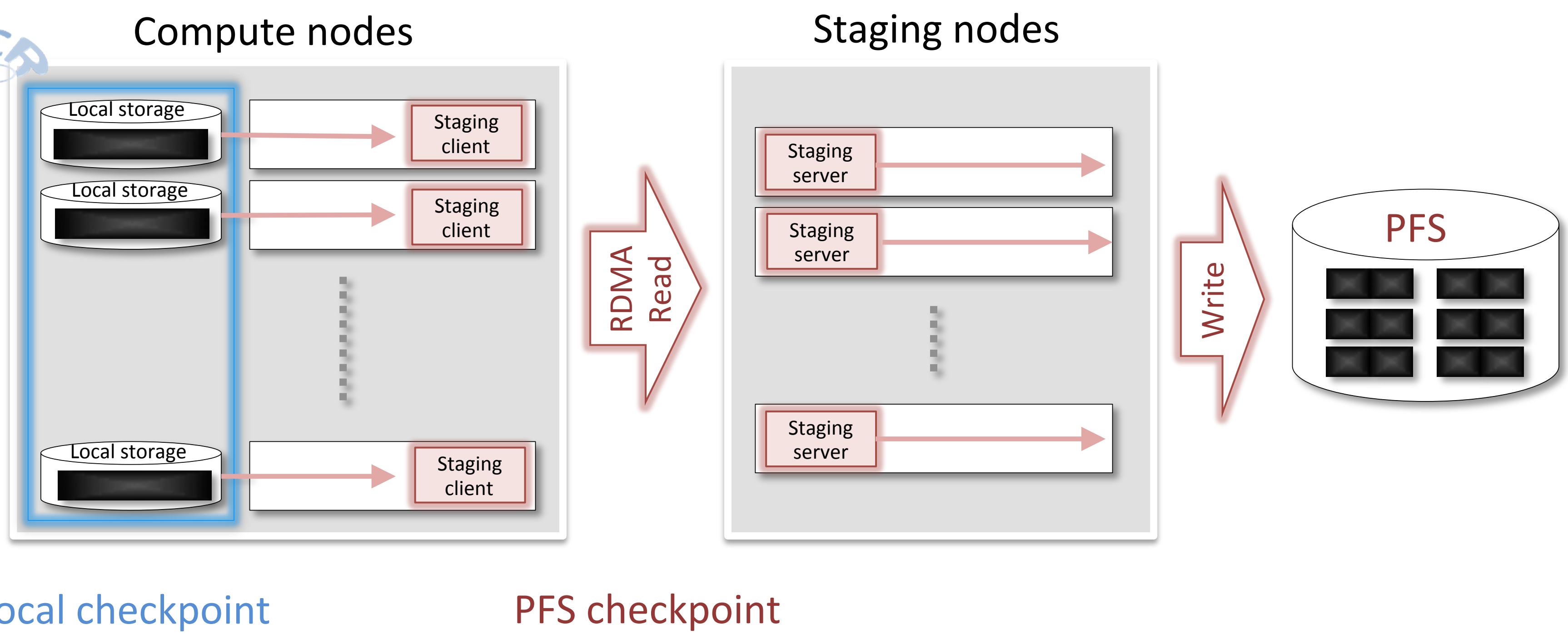
Asynchronous multi-level checkpointing



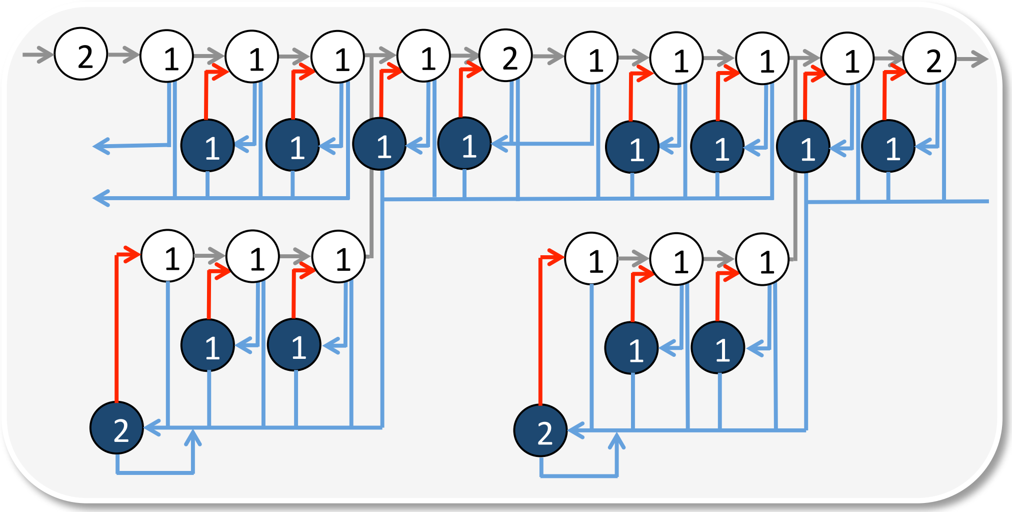
- Write PFS checkpoint in the background, minimize overhead
- By initiating next ckpt right after previous one, increase utilization

Asynchronous checkpointing system design overview

- Between compute nodes and PFS, use staging nodes
  - Dedicated extra nodes for transferring local checkpoints
  - Read checkpoints from compute nodes using RDMA, write out to a PFS



How to calculate $expected\_runtime$?



t : Interval

c_c : c-level checkpoint time

r_c : c-level recovery time

	Duration $t + c_k$		Duration r_k	
No failure	k →	$p_0(t+c_k)$ $t_0(t+c_k)$	k →	$p_0(r_k)$ $t_0(r_k)$
Failure	k → i	$p_i(t+c_k)$ $t_i(t+c_k)$	k → i	$p_i(r_k)$ $t_i(r_k)$

$$
\begin{aligned}
p_0(T) &= e^{-\lambda T} \\
t_0(T) &= T \\
p_i(T) &= \frac{\lambda_i}{\lambda}(1 - e^{-\lambda T}) \\
t_i(T) &= \frac{1 - (\lambda T + 1) \cdot e^{-\lambda T}}{\lambda \cdot (1 - e^{-\lambda T})}
\end{aligned}
$$

λ_i : i-level checkpoint time

$\lambda = \sum \lambda_i$

$p_0(T)$: No failure for T seconds

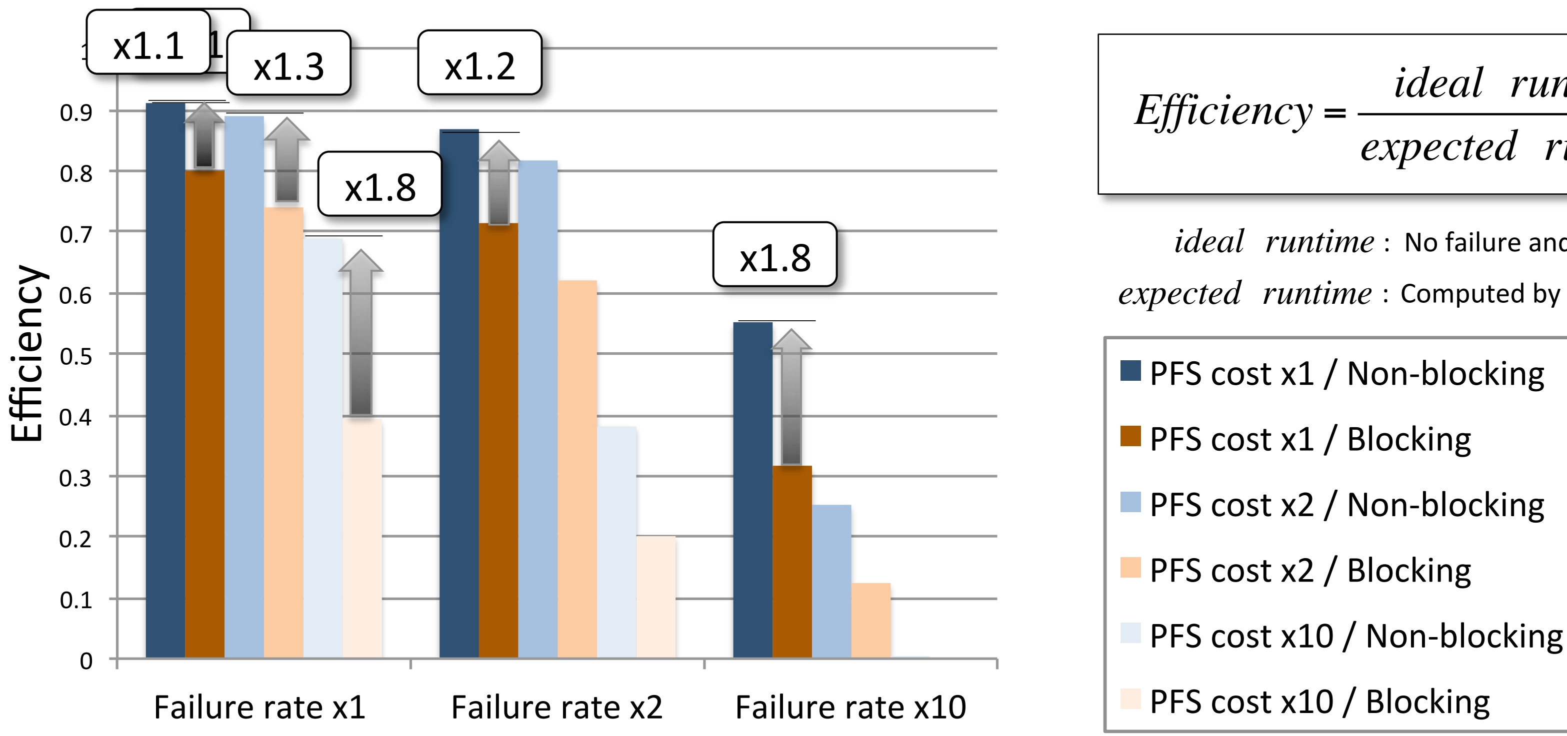
$t_0(T)$: Expected time when $p_0(T)$

$p_i(T)$: i- level failure for T seconds

$t_i(T)$: Expected time when $p_i(T)$

Efficiency: Asynchronous vs. synchronous

The asynchronous method always achieves higher efficiency than the synchronous method



$$Efficiency = \frac{ideal \quad runtime}{expected \quad runtime}$$

$ideal \quad runtime$: No failure and No checkpoint

$expected \quad runtime$: Computed by the models

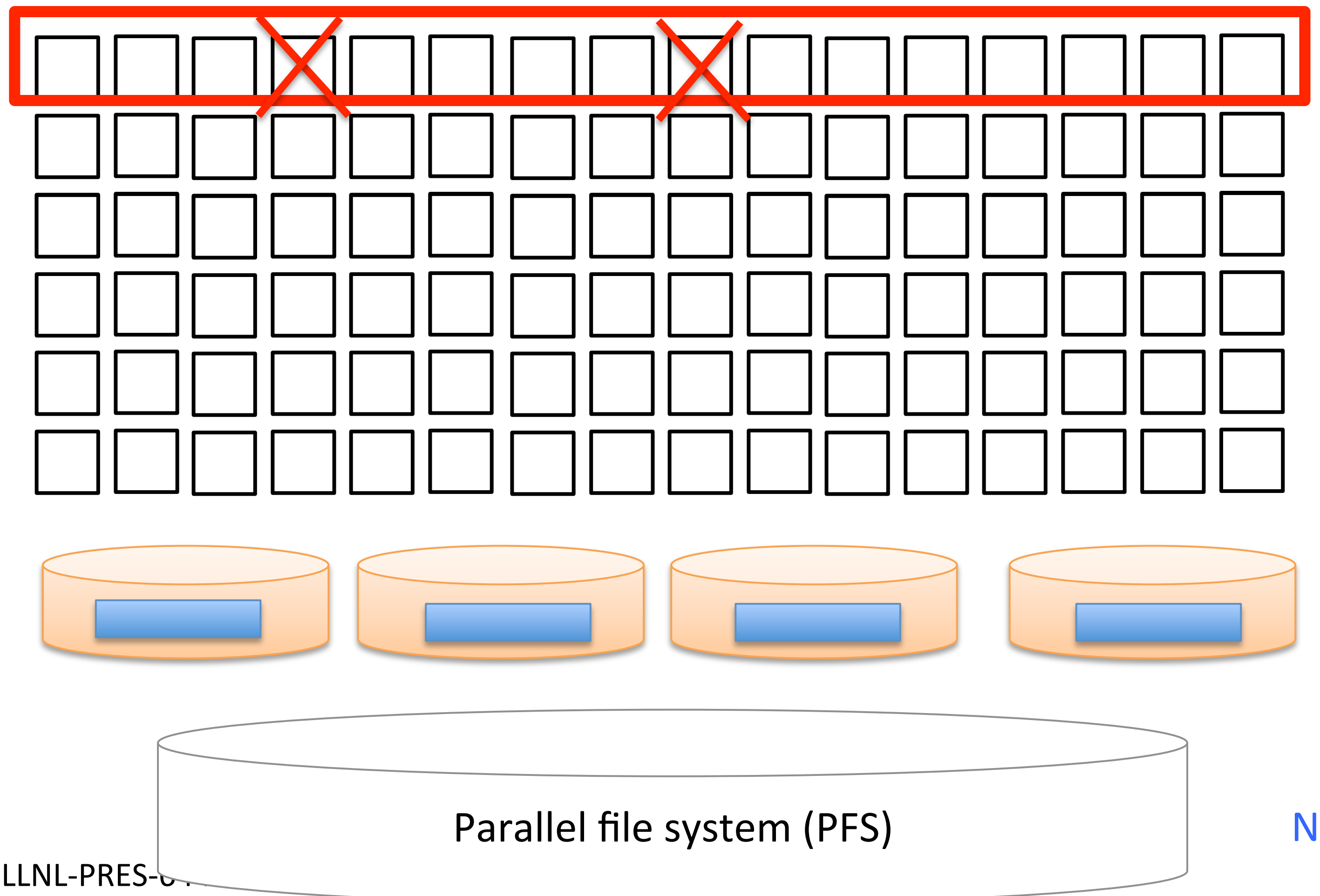
For fast checkpointing

- Buy many & fast PFSs

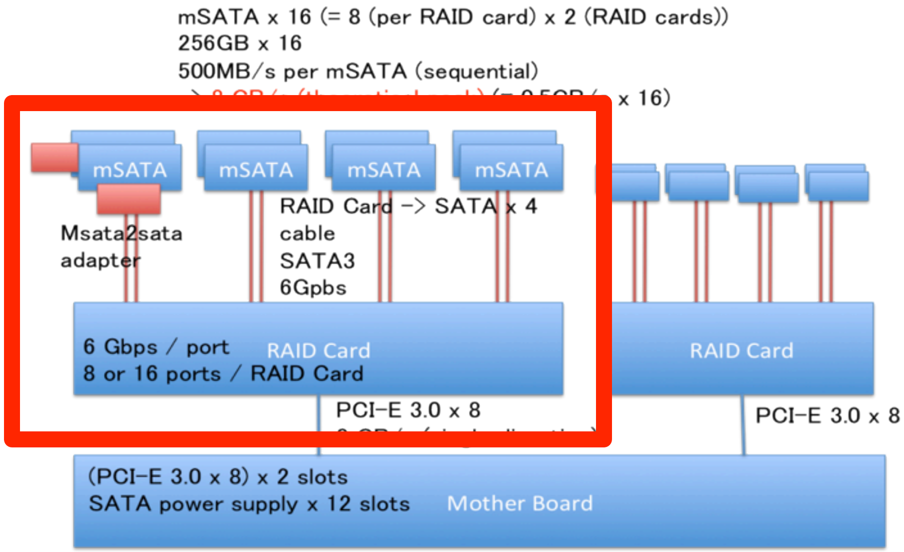
- Use of Local storage
- Storage design

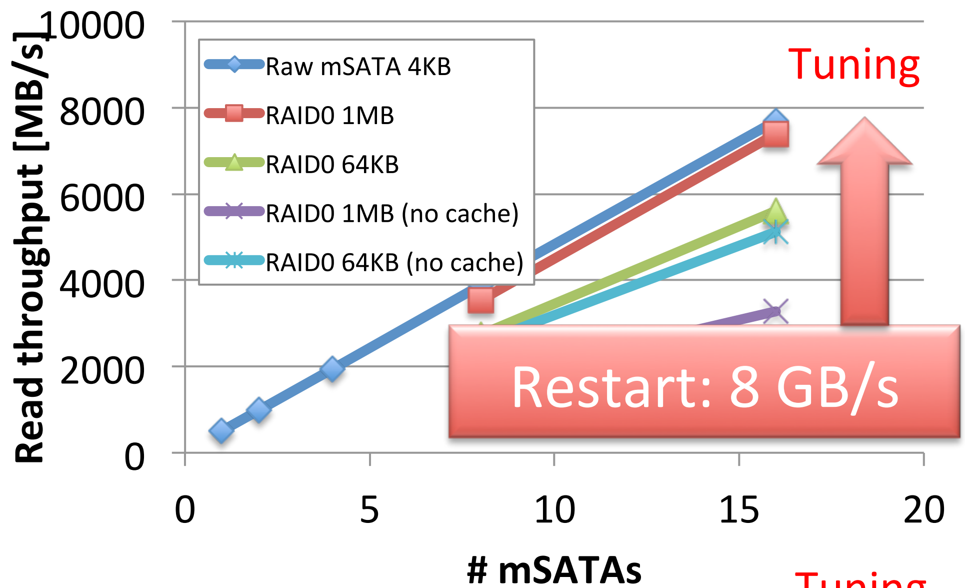
Multi-tier storage design

- Even one of checkpoint loss does not work
  - We need an additional tier of storage



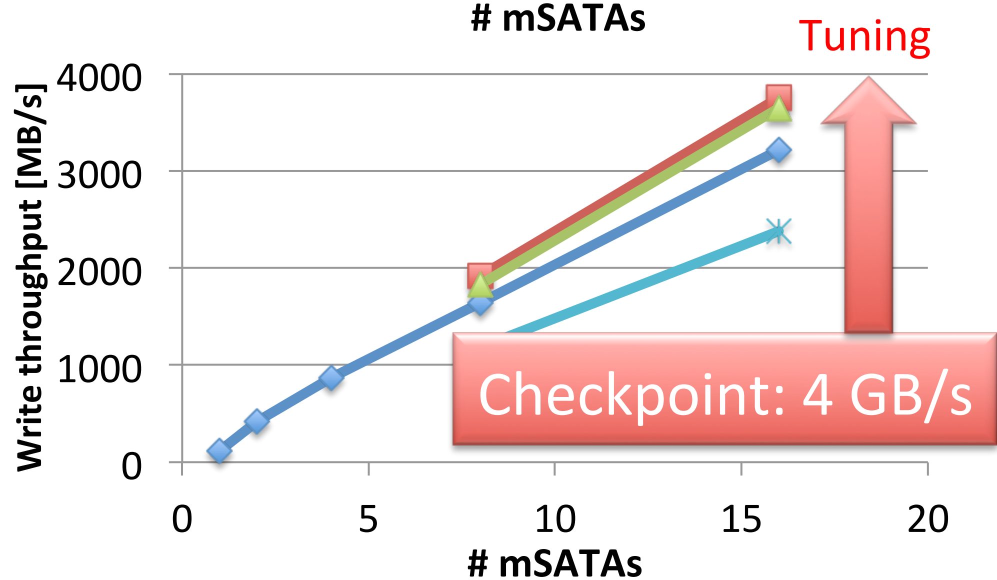
TSUBAME3.0 EBD Prototype
multi-mSATA High I/O BW, low power & cost





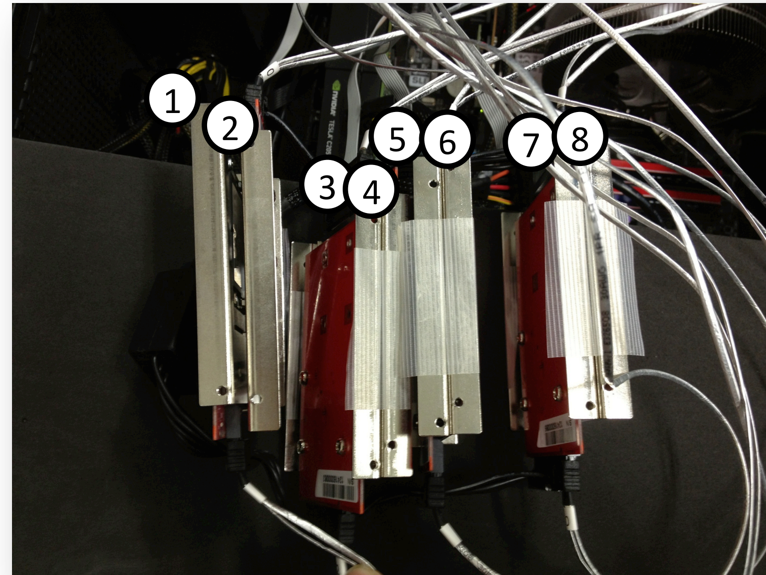
Node specification

CPU	Intel Core i7-3770K CPU (3.50GHz x 4 cores)
Memory	Cetus DDR3-1600 (16GB)
M/B	GIGABYTE GA-Z77X-UD5H
SSD	Crucial m4 msata 256GB CT256M4SSD3 (Peak read: 500MB/s, Peak write: 260MB/s)
SATA converter	KOUTECH IO-ASS110 mSATA to 2.5' SATA Device Converter with Metal Fram
RAID Card	Adaptec RAID 7805Q ASR-7805Q Single

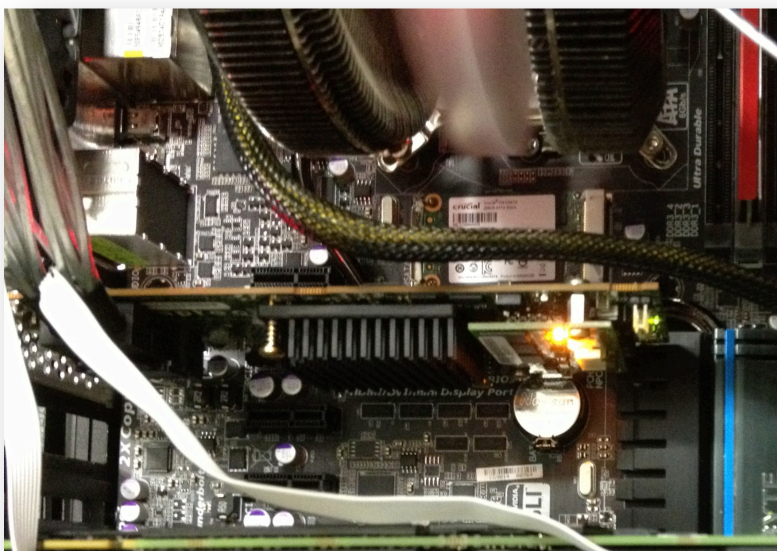


Source: Shirahata, K., Sato, H. and Matsuoka, S.: Preliminary I/O perfor- mance Evaluation on GPU Accelerator and External Memory, IPSJ SIG Technical Reports 2013-HPC-141 (2013).

A single mSATA SSD



8 integrated mSATA SSDs



RAID cards

Prototype/Test machine

Multi-level Asynchronous C/R Model

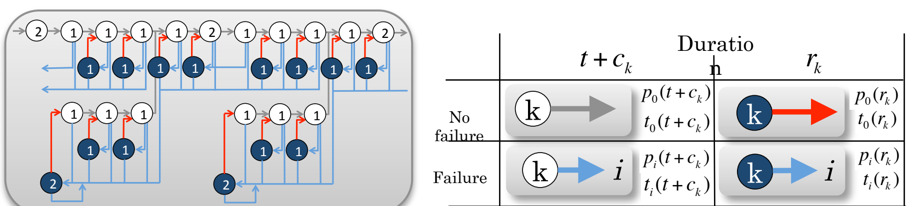
- Compute checkpoint/restart "*Efficiency*" *for C/R strategy comparison*
  - *Efficiency* : Fraction of time an application spends only in computation in optimal checkpoint interval

$$Efficiency = \frac{ideal\ \ runtime}{expected\ \ runtime}$$

$ideal\ \ runtime$: No failure and No checkpoint

$expected\ \ runtime$: Computed by the models

$$f : (L_{i=1 \ldots N},\ O_{i=1 \ldots N},\ R_{i=1 \ldots N})$$



t : Interval
c_c : c-level checkpoint time
r_c : c-level recovery time
λ_i : i-level checkpoint time

$$p_0(T) = e^{-\lambda T}$$
$$t_0(T) = T$$
$$p_i(T) = \frac{\lambda_i}{\lambda}(1 - e^{-\lambda T})$$
$$t_i(T) = \frac{1 - (\lambda T + 1) \cdot e^{-\lambda T}}{\lambda \cdot (1 - e^{-\lambda T})}$$

$p_0(T)$: No failure for T seconds
$t_0(T)$: Expected time whe $p_0(T)$
$p_i(T)$: i - level failure for T seconds
$t_i(T)$: Expected time whe $p_i(T)$

- Input: Each level of
  - L_i : Checkpoint Latency
  - O_i : Checkpoint overhead
  - R_i : Restart time
- Output: "*Efficiency*"

Efficiency

Source: Sato, K., Maruyama, N., Mohror, K., Moody, A., Gamblin, T., de Supinski, B. R. and Matsuoka, S.: Design and Modeling of a Non-Blocking Checkpointing System (SC12)

Modeling of C/R Strategies

- L_i : Checkpoint Latency
  - Time to complete a checkpoint (C_i) and encoding (E_i)

$$L_i = C_i + E_i$$

- O_i : Checkpoint overhead
  - The increased execution time of an application
    - Sync. C/R: Checkpoint overhead (O_i) = Checkpoint latency (L_i)
    - Async. C/R: Initialization time of level i C/R

$$O_i = \begin{cases} C_i + E_i & \text{(Sync.)} \\ I_i & \text{(Async.)} \end{cases}$$

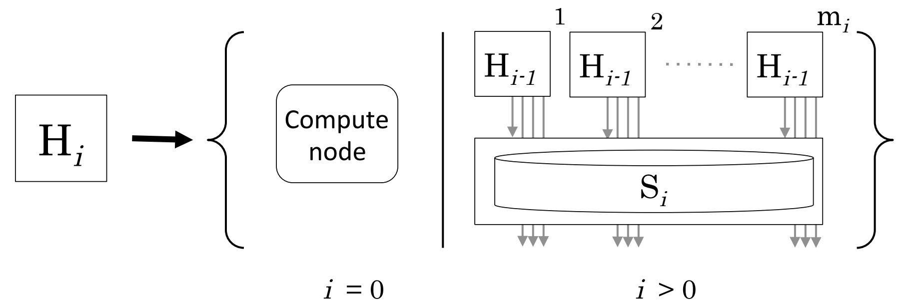
- C_i & R_i : Checkpoint/Restart time

* S_i : tier i storage

$$C_i \text{ or } R_i = \frac{\text{< C/R date size / node >} \times \text{<\# of C/R nodes per } S_i^{*} \text{ >}}{\text{< write perf. (} w_i \text{) >} \text{ or } \text{<read perf. (} r_i \text{) >}}$$

Recursive Structured Storage Model

- Generalization of storage architectures with "*context-free grammar*"
  - A tier i hierarchical entity (H_i), has a storage (S_i) shared by (m_i) upper hierarchical entities (H_{i-1})
  - $H_{i=0}$ is a compute node
  - $H_N \{m_1, m_2, \ldots, m_N\}$



Storage Model: $H_N \{m_1, m_2, \ldots, m_N\}$

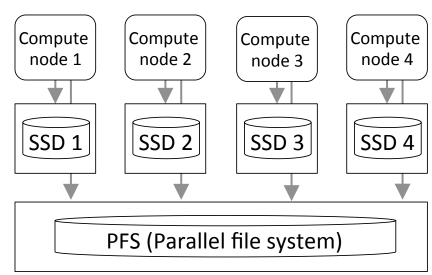
r_i	Sequential read throughput from compute nodes ($H_{i=0}$)
w_i	Sequential write throughput from compute nodes ($H_{i=0}$)
m_i	The number of a upper hierarchical entities (H_{i-1}) sharing S_i

<# of C/R nodes per S_i>

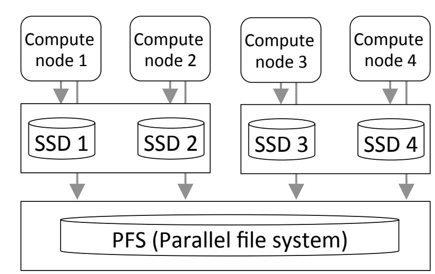
$$= \frac{K^*}{\text{<\# of } S_i\text{>} \ (= \Pi^N_{k=i+1} m_k)}$$

**K: C/R cluster size*

Example



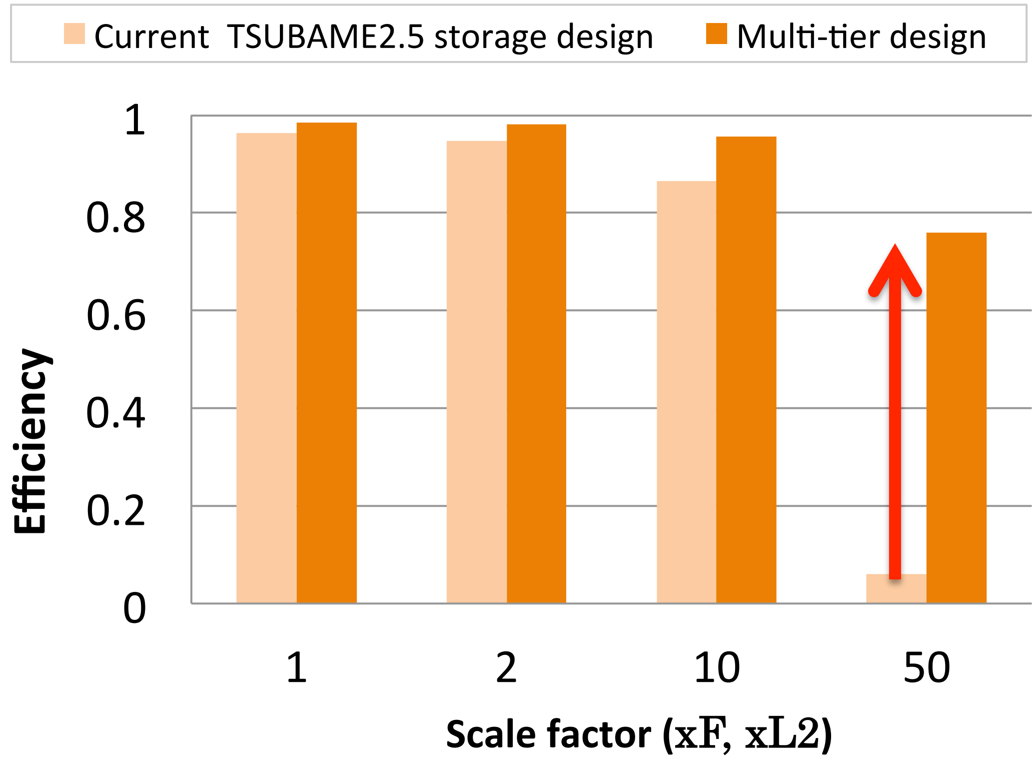
Flat buffer system: $H_2 \{1, 4\}$



Burst buffer system: $H_2 \{2, 2\}$

Efficiency with Increasing Failure Rates and Checkpoint Costs

- The burst buffer system always achieves a higher efficiency
  ⇒ Stores checkpoints on fewer nodes



- With uncoordinated checkpointing, 70% efficiency even on systems that are two orders of magnitude larger (if logging overhead is 0)
  ⇒ Partial restart can exploit the bandwidth of both burst buffers and the PFS

Summary

- Fault tolerance is important
  - Fast and Reliable checkpointing is required
- Lustre provides high bandwidth
  - Checkpointing requires more
- For fast checkpointing
  - Multi-level checkpointing
  - Multi-tier storage design

Q & A

Speaker:

Kento Sato (佐藤 賢斗)
kent@matsulab.is.titech.ac.jp
Tokyo Institute of Technology (Tokyo Tech)
Research Fellow of the Japan Society for the Promotion of Science
http://matsu-www.is.titech.ac.jp/~kent/index_en.html

Collaborators

Adam Moody, Kathryn Mohror, Todd Gamblin, Bronis R de. Supinski,
Naoya Maruyama, Satoshi Matsuoka