Design and Modeling of a Non-blocking Checkpointing System

Kento Sato†1,2, Adam Moody†3, Kathryn Mohror†3, Todd Gamblin†3, Bronis R. de Supinski†3, Naoya Maruyama†4,5 and Satoshi Matsuoka†1,5,6,7

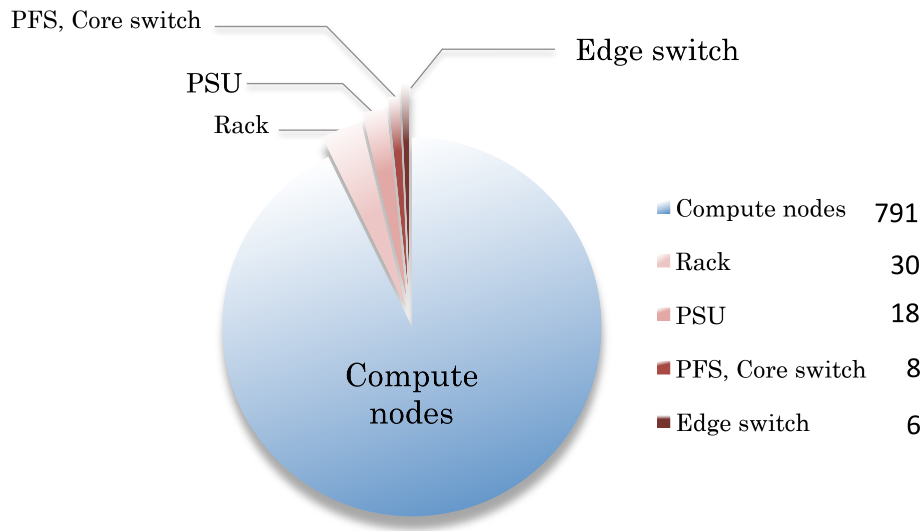
†1 Tokyo Institute of Technology
†2 Research Fellow of the Japan Society for the Promotion of Science
†3 Lawrence Livermore National Laboratory
†4 RIKEN Advanced institute for Computational Science
†5 Global Scientific information and Computing Center
†6 National Institute of Informatics
†7 JST/CREST

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52- 07NA27344. LLNL-PRES-599833-DRAFT

November 13th, 2012

Failures on HPC systems

- Exponential growth in computational power
  - Enables finer grained scientific simulations
- Overall failures rate increases accordingly
  - Due to increasing complexity and system size



Overall MTBF
13 hours

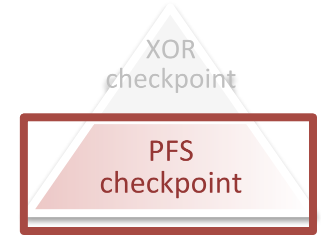
Failure analysis on TSUBAME2.0

Period: 1.5 years (Nov 1st, 2010 ~ April 6th 2012)
Observations: 962 node failures in total

TSUBAME2.0, 14th in Top500 (June 2012)

2.4 PFlops
1442 nodes
2953 CPU sockets
4264 GPUs
197 switches
58 racks

- System resiliency is becoming more important
  - Without a viable resilience strategy, applications can not run for even one day on such a large system



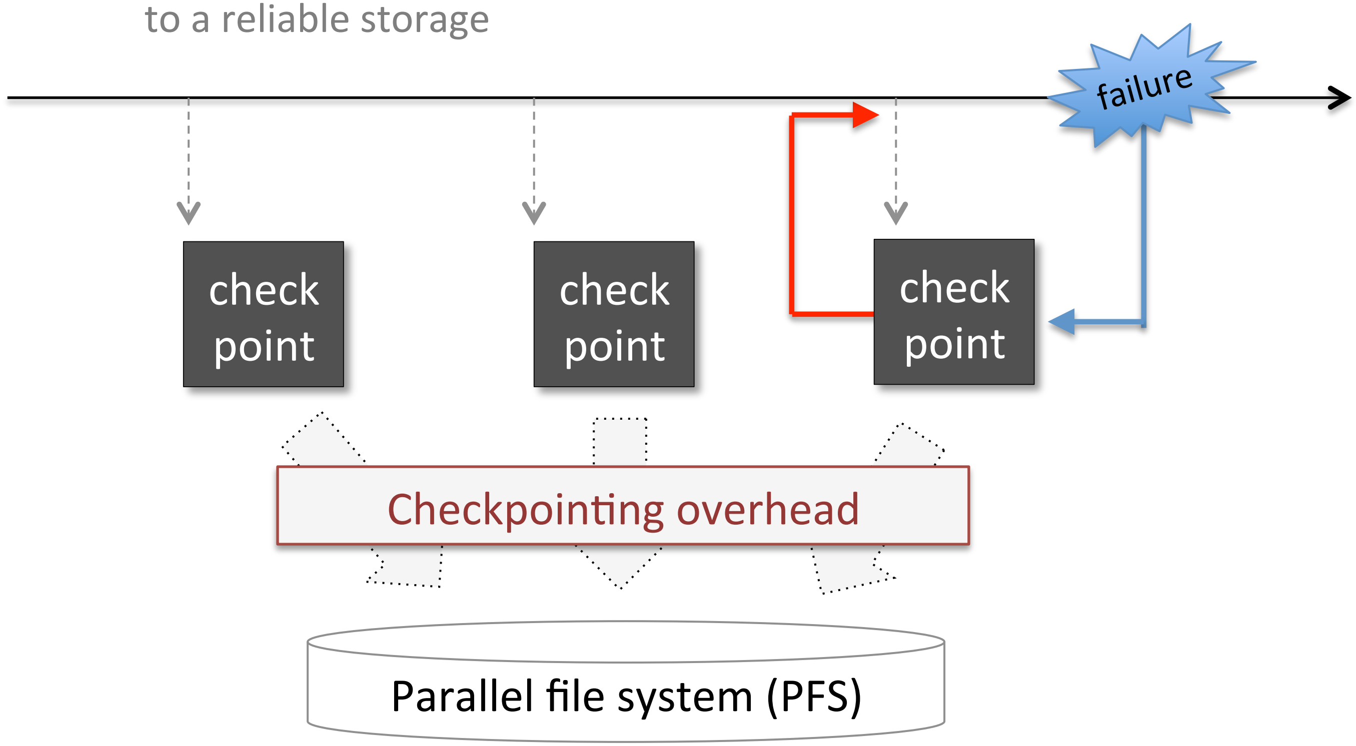
Traditional Checkpoint/Restart

Checkpoint

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

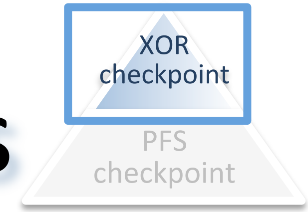
Restart

On a failure, restart the execution from the latest checkpoint

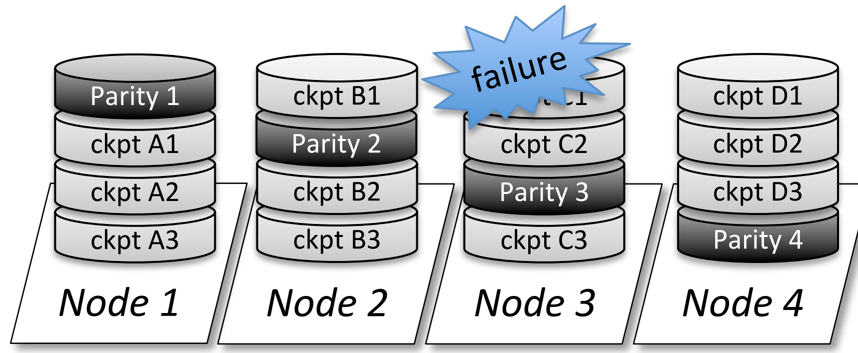


Mostly these checkpoints are stored in the most reliable storage, such as a shared parallel file system(PFS).

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

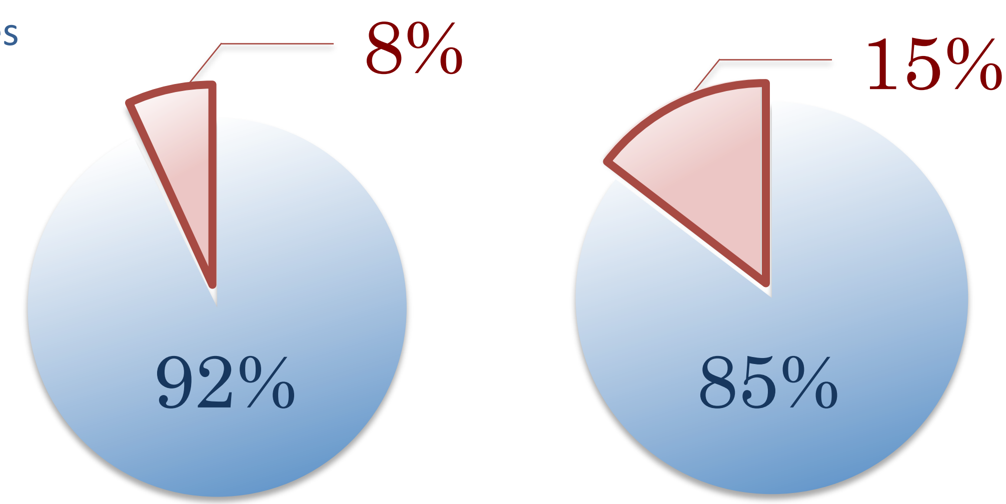


XOR encoding example

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

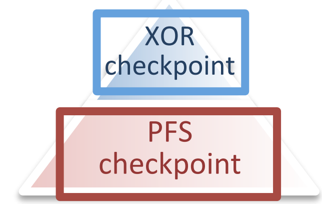
Rest of failures still require a checkpoint on a reliable PFS

LOCAL/XOR/PARTNER checkpoint
PFS checkpoint

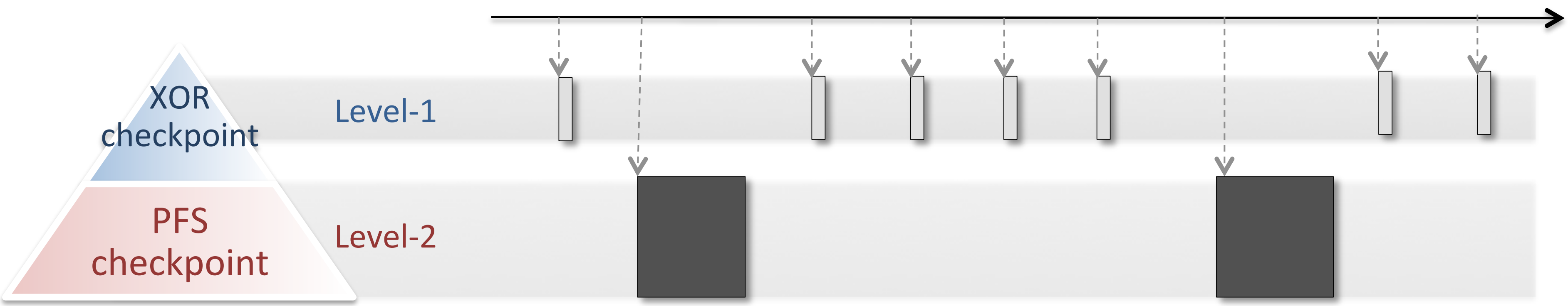


Failure analysis on TSUBAME2.0 Failure analysis on LLNL clusters

Diskless checkpoint is promising approach



Multi-level checkpointing (MLC)



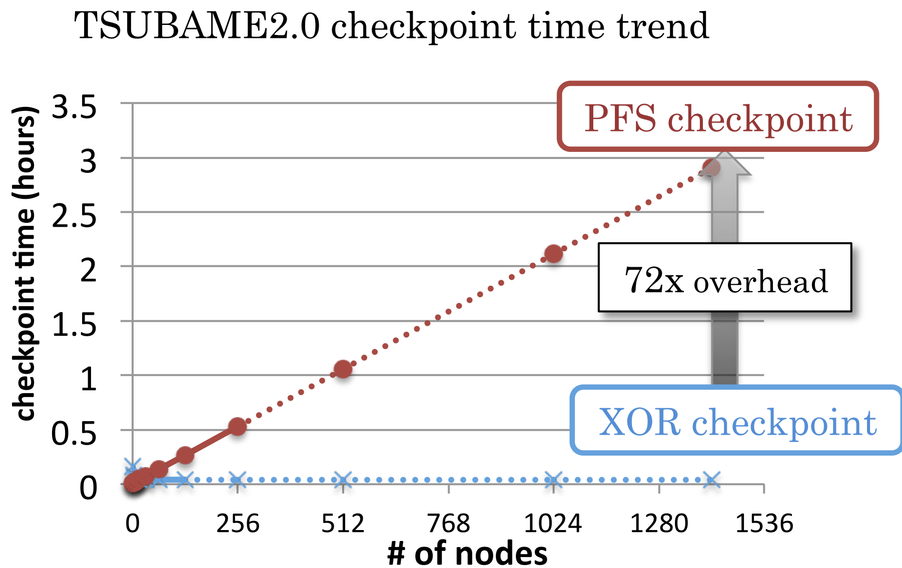
- Use storage levels hierarchically
  - XOR checkpoint: Frequently
    - for one node or a few node failure
  - PFS checkpoint: Less frequently
    - for multi-node failure
- 8x efficiency improvement
  - With MLC implementation called SCR(Scalable Checkpoint/Restart) library developed in LLNL
  - Compared to single-level checkpointing

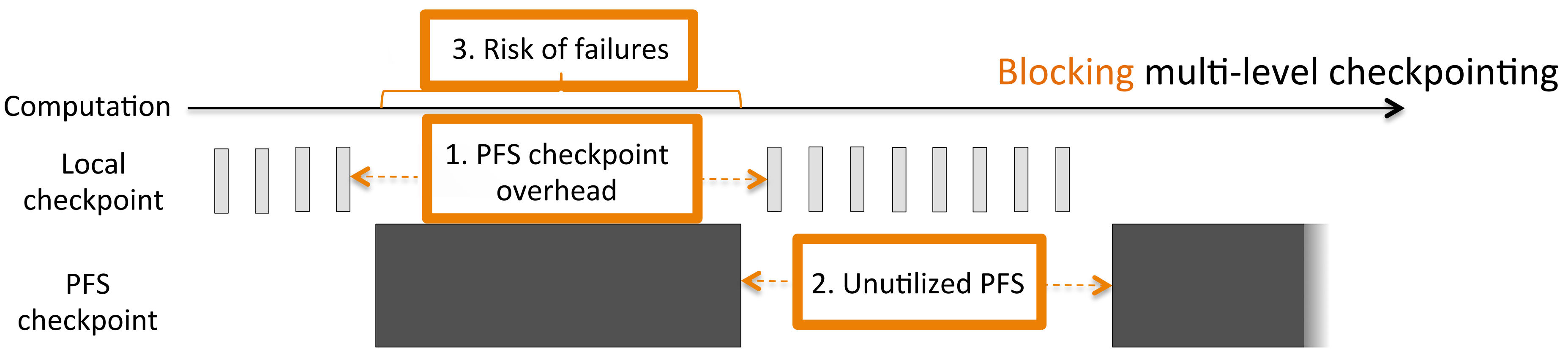
Source: A. Moody, G. Bronevetsky, K. Mohror, and B. R. de Supinski, "Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System," in Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC 10).

MLC Problems on Petascale or larger

Three 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
3. Failure during PFS checkpoint
   – At scale, prolonged PFS checkpointing has a risk of failures during checkpointing





Objective, Proposal and Contributions

- **Objective**: More efficient MLC
  - Minimize PFS checkpoint overhead
  - Improve PFS utilization
  - Reduce a risk of failure during PFS checkpoint

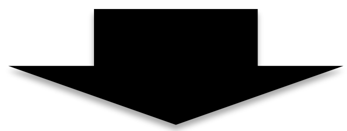
- **Proposal & Contributions**:
  - Developed an non-blocking checkpointing system as an extension for SCR library
    - PFS checkpoint with 0.5 ~ 2.5% overhead
  - Modeled the non-blocking checkpointing
    - Determine optimal multi-level checkpoint configuration
    - 1.1 ~ 1.8x efficiency on current and future systems

Outline

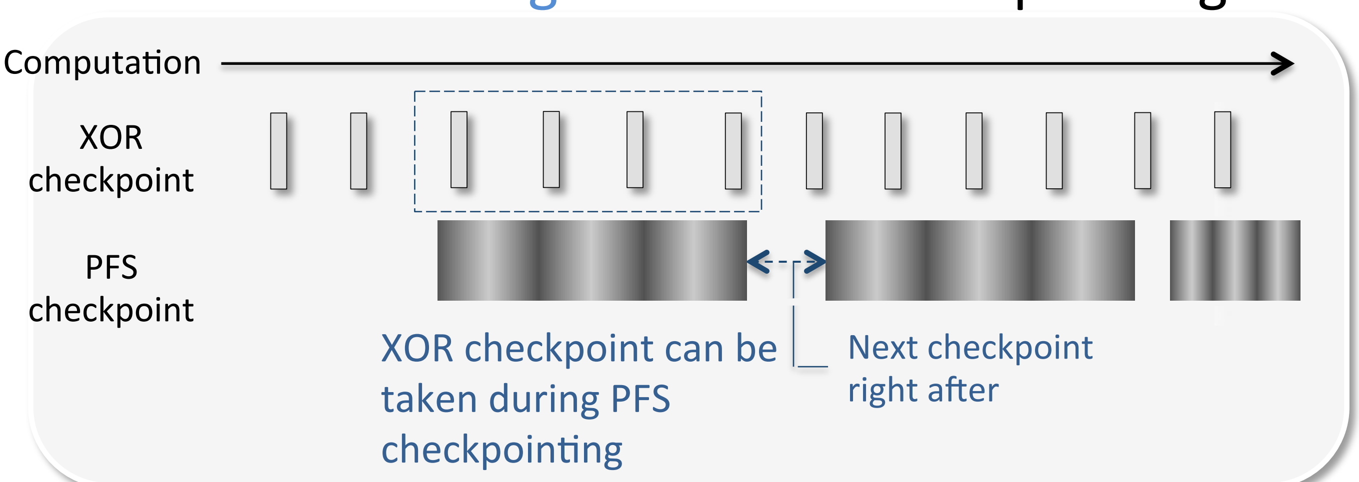
- Introduction
- Design of a Non-blocking checkpointing system
- Modeling of the Non-blocking checkpointing
- Evaluation
- Summary

Non-blocking checkpointing overview

Blocking multi-level checkpointing



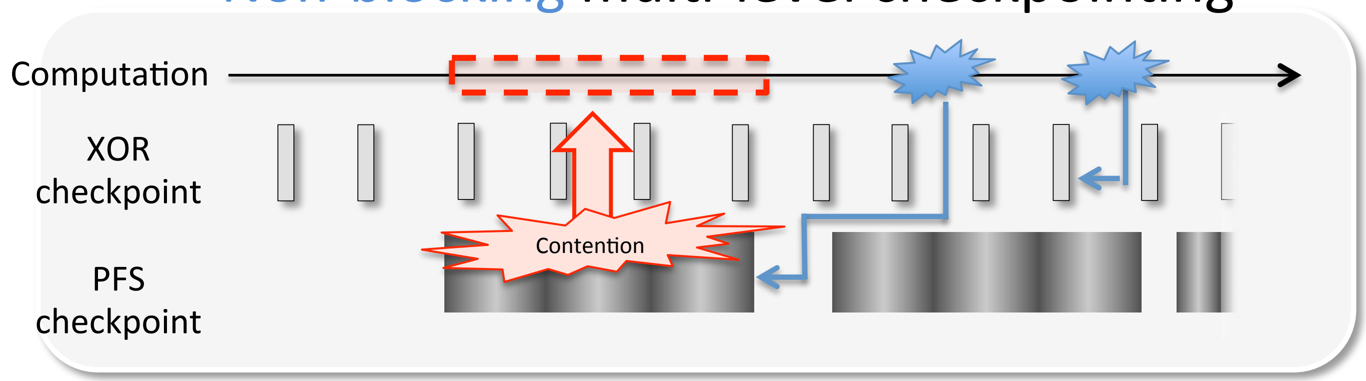
Non-blocking multi-level checkpointing



- Write PFS checkpoint in the background, minimize overhead
- By initiating next ckpt right after previous one, increase utilization
- Reduce impact of failures requiring XOR checkpoint

Challenges on Non-blocking checkpointing

Non-blocking multi-level checkpointing



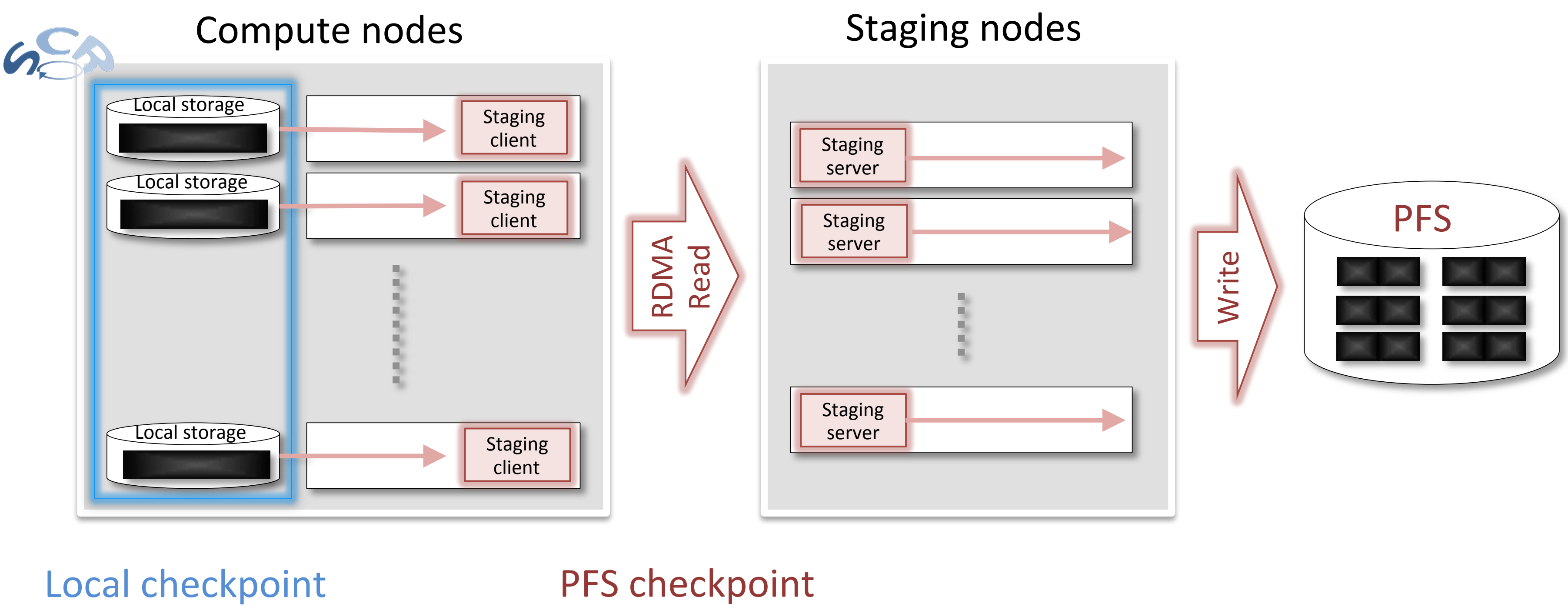
- Utilize local SSDs for the additional space
  - Write PFS checkpoint in the background which requires additional storage spaces
- Minimize resource contention
  - PFS checkpointing is running in the background, inflate the runtime due to resource contention

  ⇒ Implementation: Use RDMA with checkpoint dedicated nodes
- Optimize configuration (e.g. checkpoint interval)
  - On a failure requiring PFS , need "complete PFS checkpoint"
  - On a failure requiring XOR, need to restore both XOR & PFS ckpt being written

  ⇒ Modeling: Model a non-blocking multi-level checkpoint

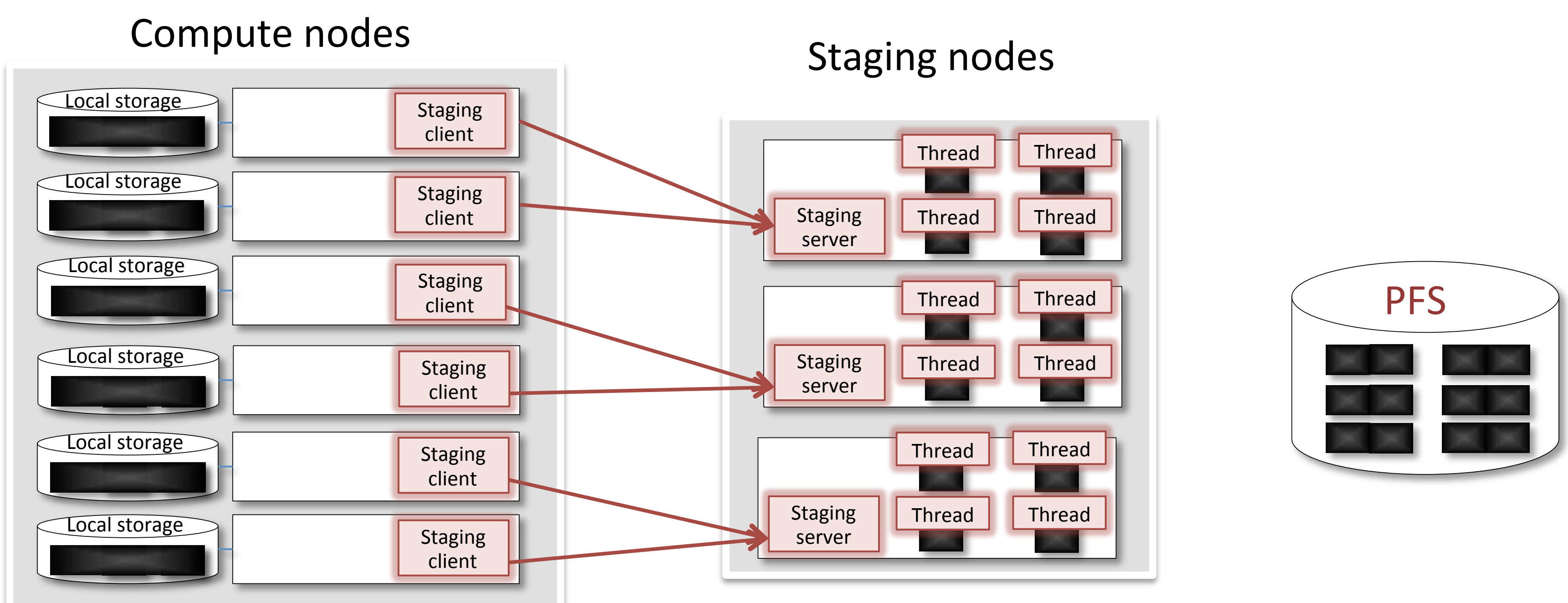
Non-blocking checkpointing overview

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



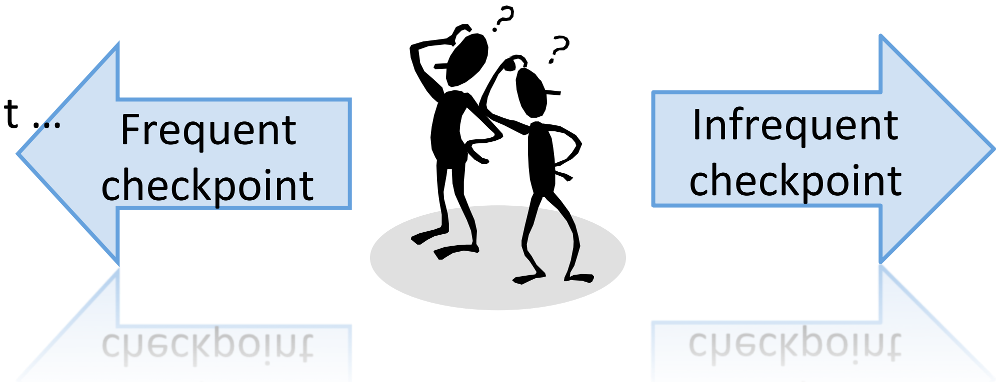
Non-blocking checkpointing using RDMA

1. Local storages to Local memory
   – After SCR writes checkpoint to a local storage, staging clients running on compute nodes read chunks of the checkpoint from the local storage to a buffer memory
2. Local memory to Remote memory
   – Send RDMA Read requests to a mapped staging server running on a staging node, staging server read the checkpoints from the buffer using RDMA
3. Remote memory to PFS
   – Data writer threads running on Staging nodes write checkpoint chunks to PFS in parallel



Modeling of Non-blocking checkpoint

more resilient but …
huge overhead



Low overhead but …
less resilient

Outline

- Introduction
- Design of Non-blocking checkpointing system
- Modeling of Non-blocking checkpointing
- Evaluation
- Summary

more resilient but …
huge overhead

Frequent checkpoint

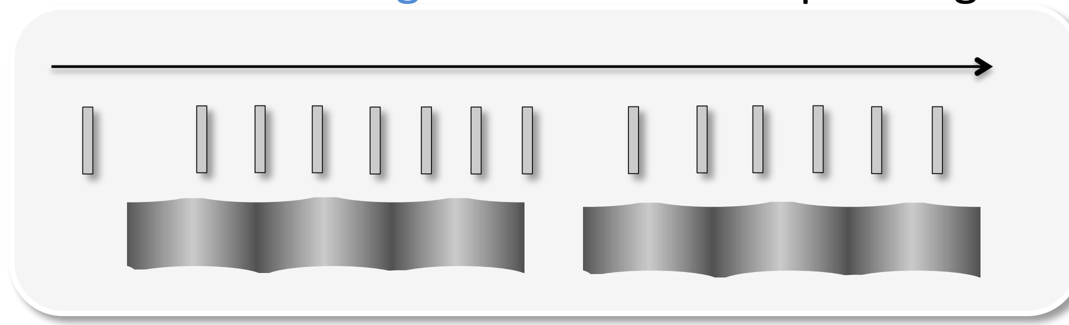
Infrequent checkpoint

Low overhead but …
less resilient

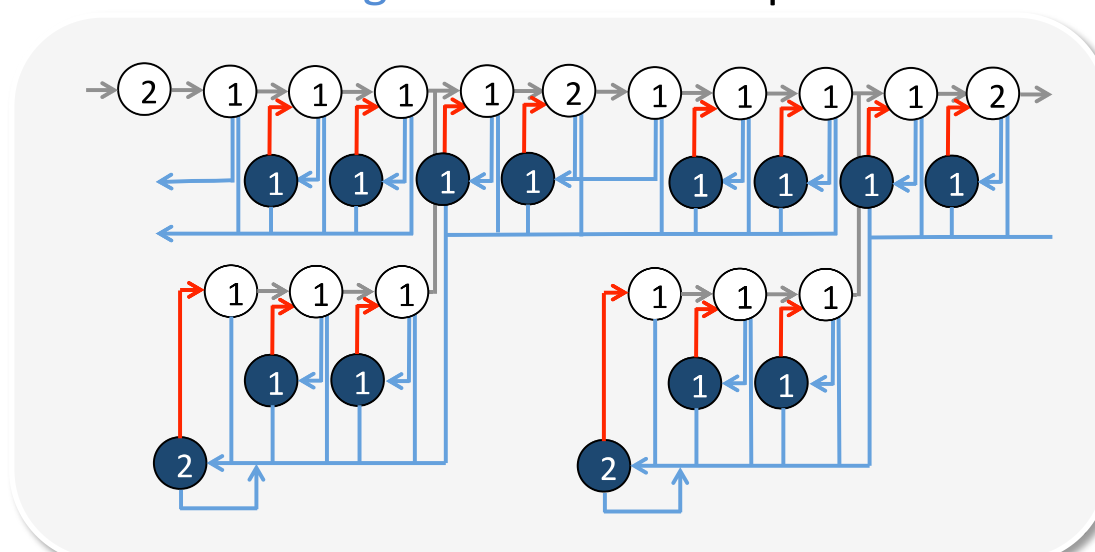
Non-blocking MLC model overview

- Describe an application's state transitions as Markov model
- Input (each level of ..)
  - Checkpoint time
  - Restart time
  - Failure rate
  - Interval
- Output
  - Expected runtime
- Find checkpoint intervals that minimize runtime

Non-blocking multi-level checkpointing

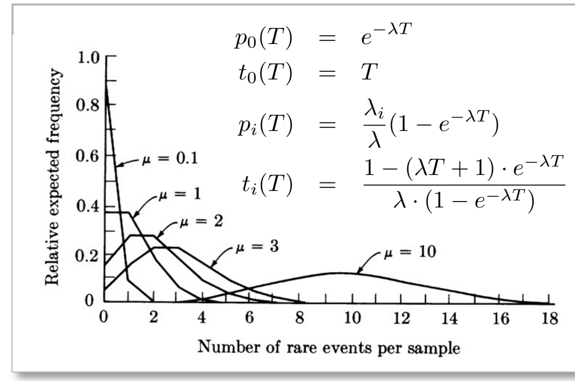


Non-blocking multi-level checkpoint model



Assumptions on the model

- Independent and identically distributed failure rate & Poisson distribution
  - One failure does not increase the probability of successive failures



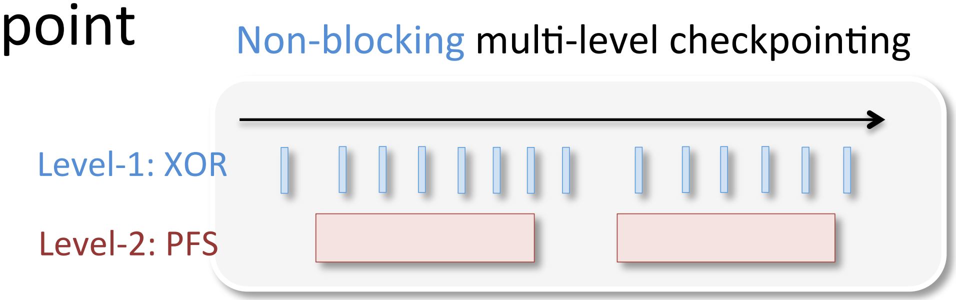
- Stable write & read performance
  - Checkpoint/Restart time significantly does not change during overall the runtime

- Failure on Level-k recovery => Level-($k+1$) checkpoint
  - Another one node failure during XOR recovery requires a PFS checkpoint
  - Assume PFS checkpoint can retry infinitely

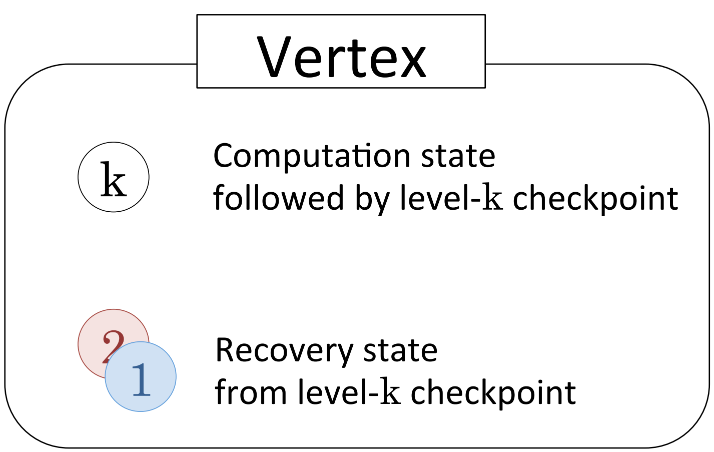
- Saved checkpoints are never lost on non-failed nodes and a PFS
  - Guarantee failed job can restart from the latest checkpoint

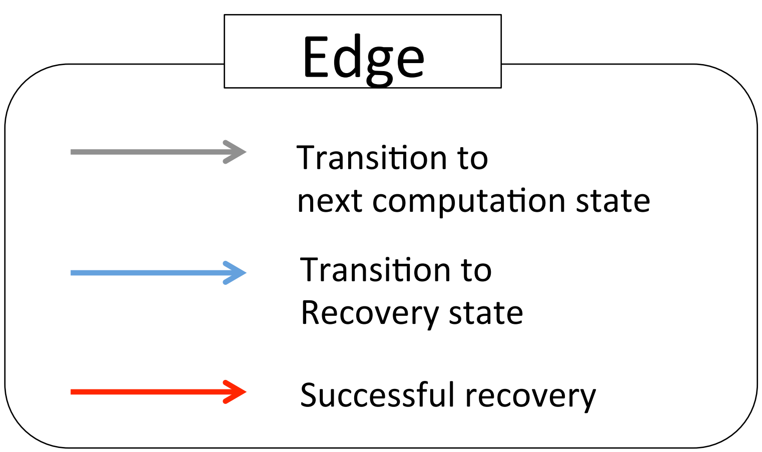
Two-level checkpoint example

- For simplicity, two-level checkpoint
  - Level-1: XOR checkpoint
  - Level-2: PFS checkpoint



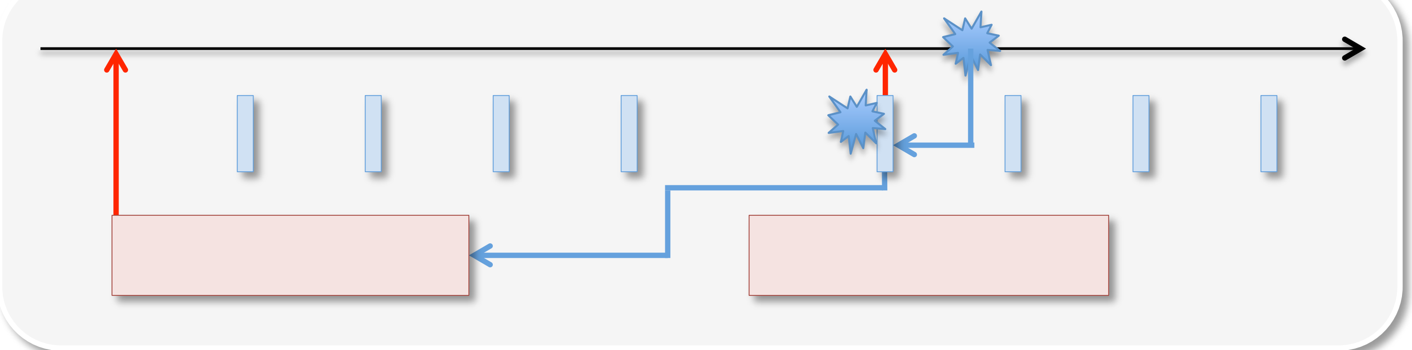
- Describe state transitions as Markov model





No failure & Level-"1" failure case

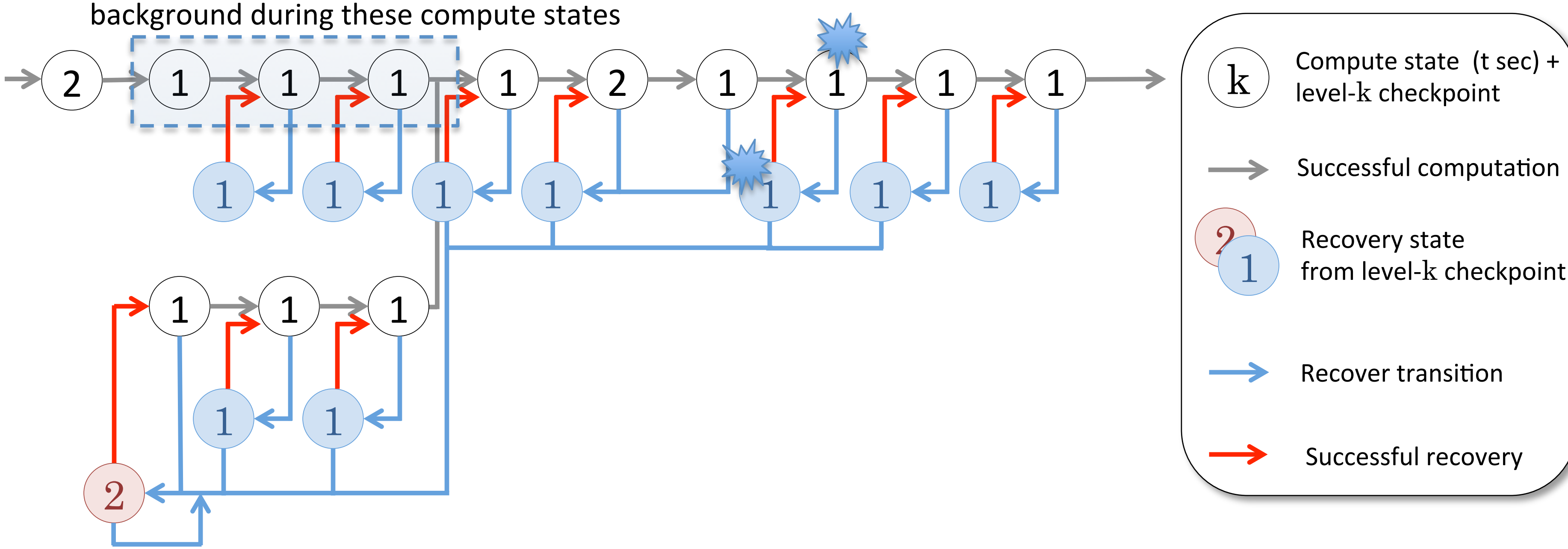
Level-"1" failure



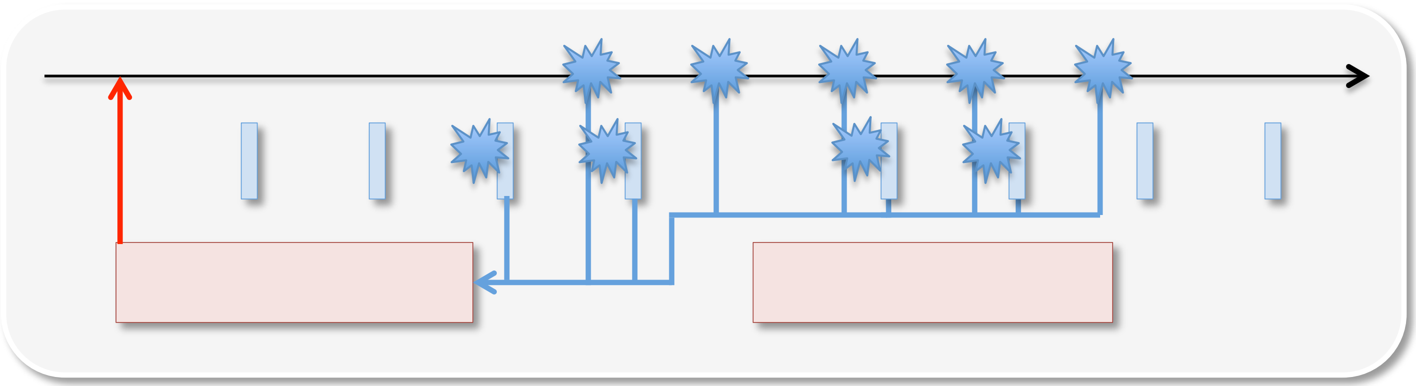
L-2 ckpt: L-1 ckpt

= 1: 4

PFS checkpointing is running in the background during these compute states

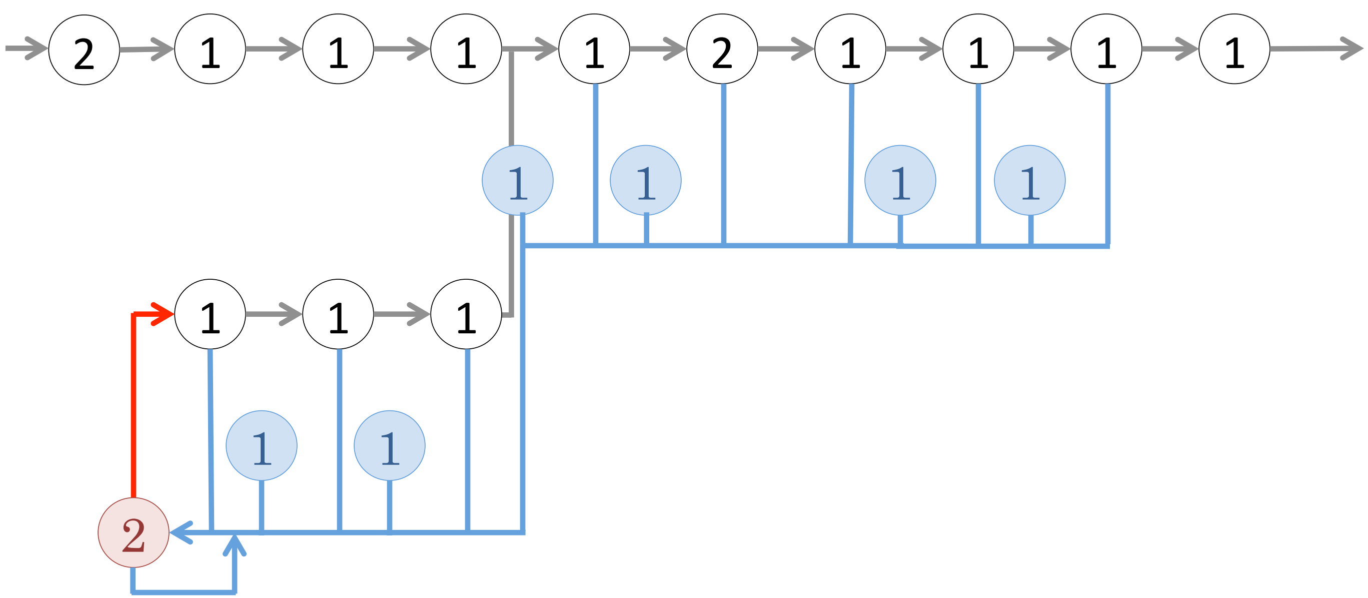


Level-"2" failure case



L-2 ckpt: L-1 ckpt

= 1: 4



k Compute state (t sec) + level-k checkpoint

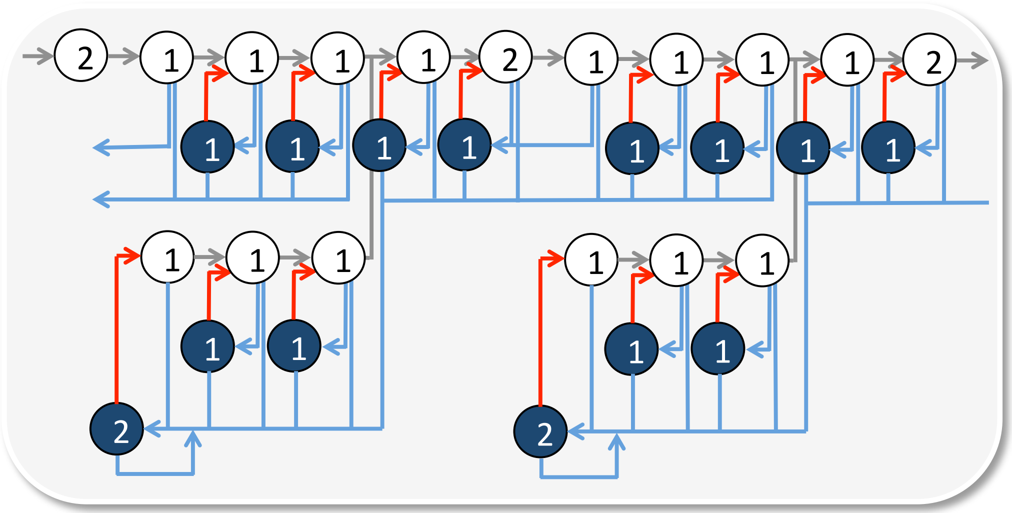
Successful computation

Recovery state from level-k checkpoint

Recover transition

Successful recovery

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

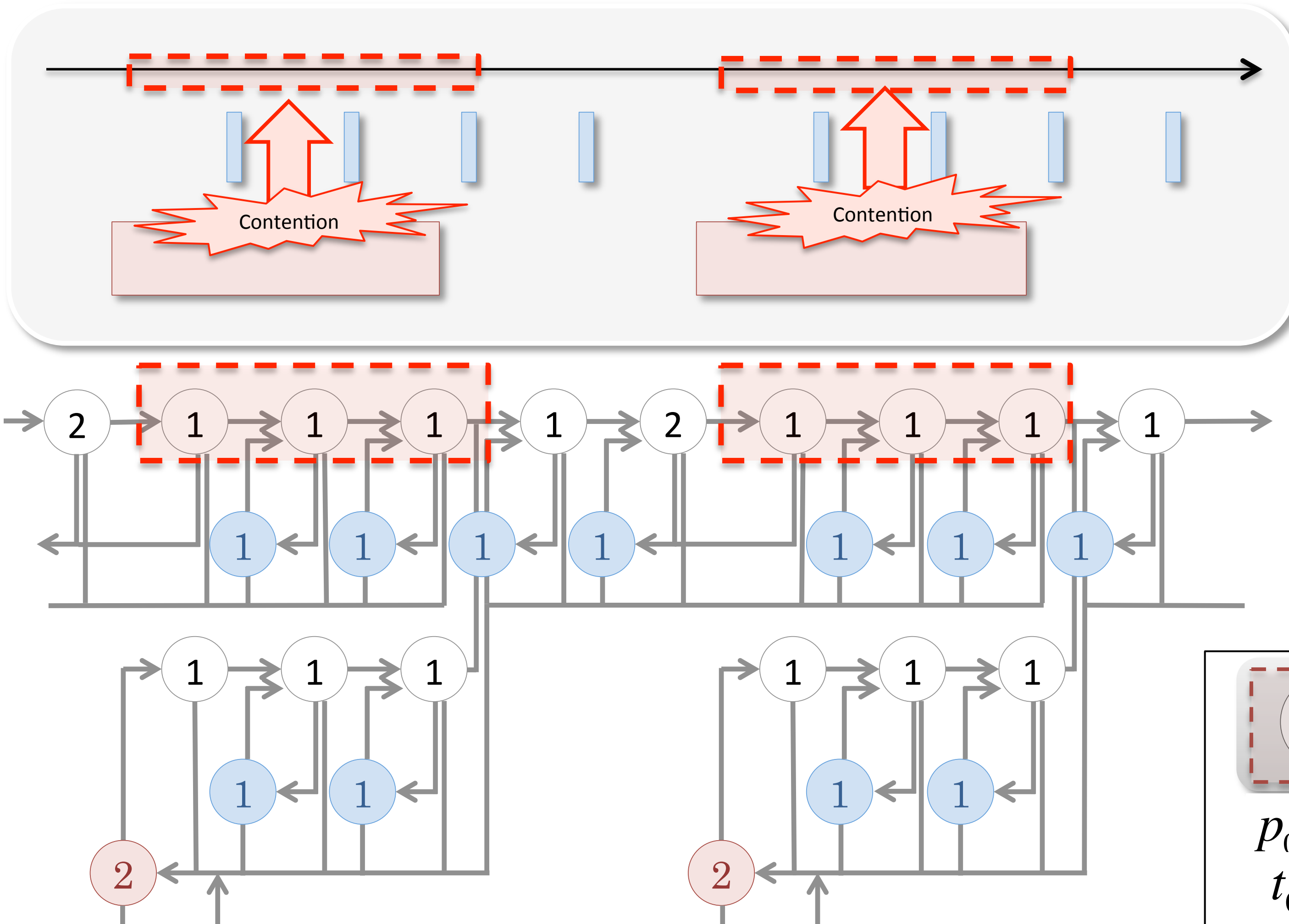
$\lambda = \sum \lambda_i$

- $p_0(T)$: No failure for T seconds
- $t_0(T)$: Expected time when $p_0(T)$

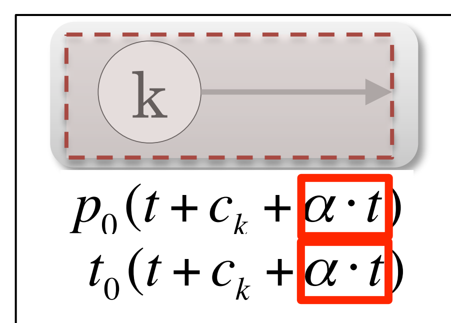
- $p_i(T)$: i - level failure for T seconds
- $t_i(T)$: Expected time when $p_i(T)$

Overhead factor: α

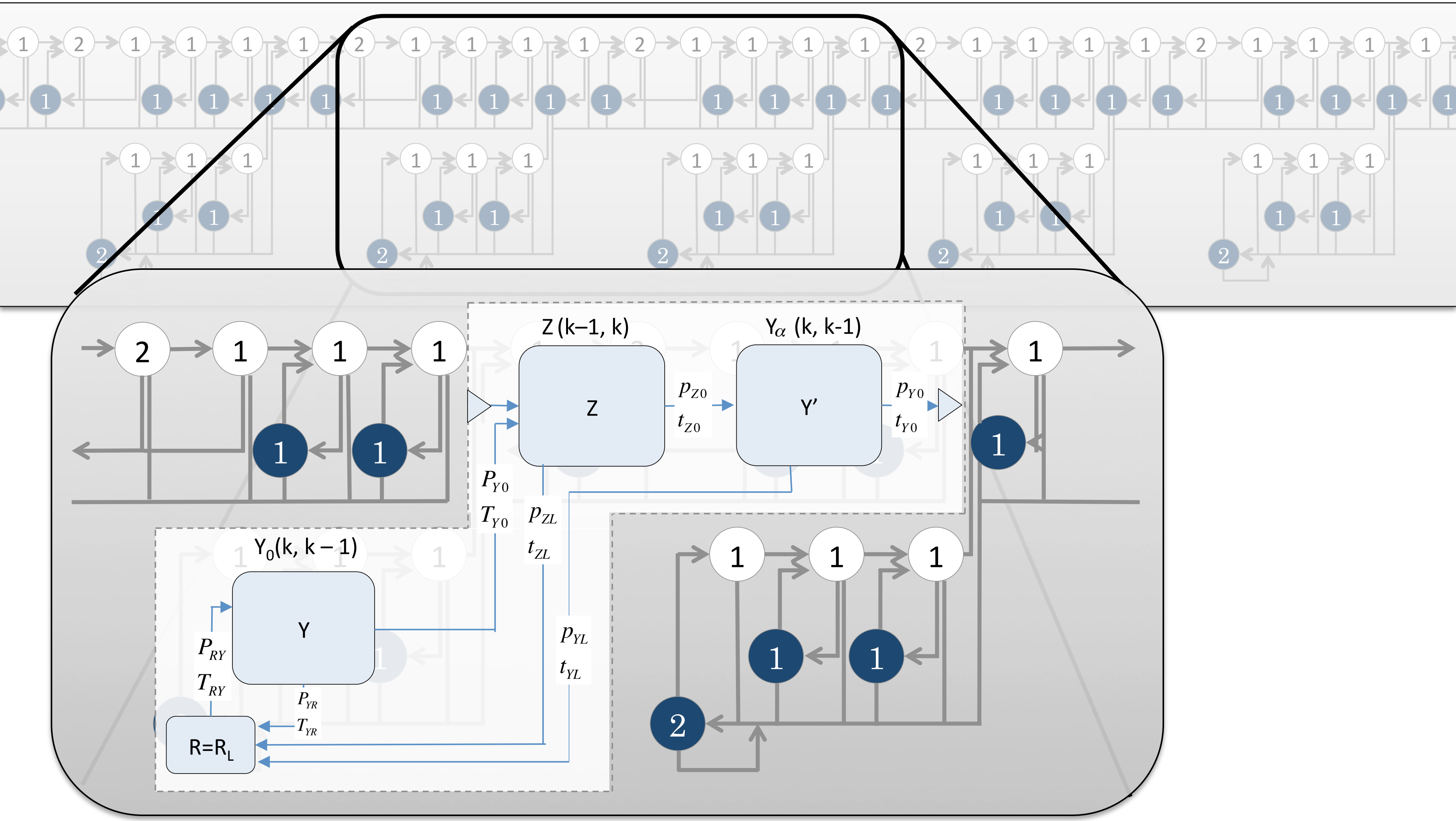
- Quantify an overhead by our proposed non-blocking checkpointing system



During these compute states PFS checkpointing is running in the background, inflate the runtime due to resource contention



Arbitrary N - level checkpointing model

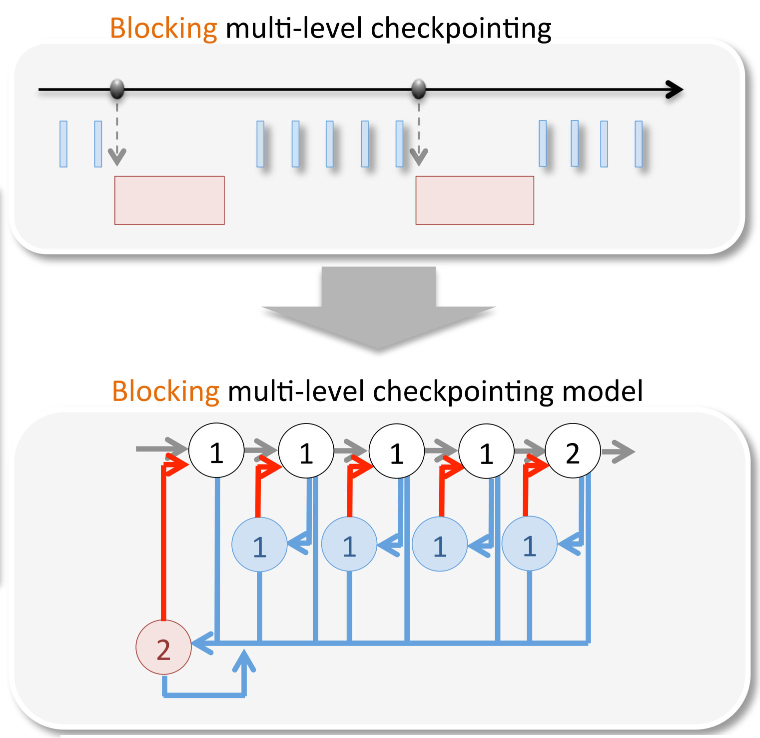


Non-blocking vs. Blocking MLC checkpointing

- **Benchmark**: Himeno benchmark
  - Stencil application solving Poisson's equation using Jacobi iteration method
- **Target System**:
  TSUBAME2.0 Thin nodes (1408 nodes)

CPU	Intel Xeon X5670 2.93GHz (6cores x 2 sockets)
Memory	DDR3 1333MHz (58GB)
Network	Mellanox Technologie Dual rail QDR Infiniband 4x (80Gbps)
Local storage	120GB Intel SSD (RAID0/60GBx2)
PFS	Lustre (/work0)

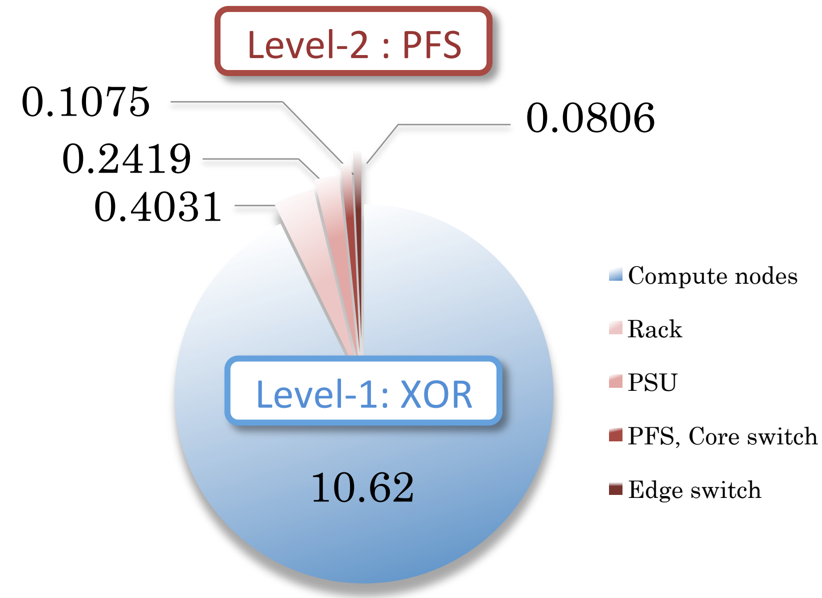
- **Checkpoint Level**: Two-level
  - Level-1: XOR using local SSD
  - Level-2: PFS using Lustre



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).

Model Parameters

- **<u>Failure rates</u>**
  - 1.5 years (Nov 1st 2010 ~ Apr 6th 2012) failure history

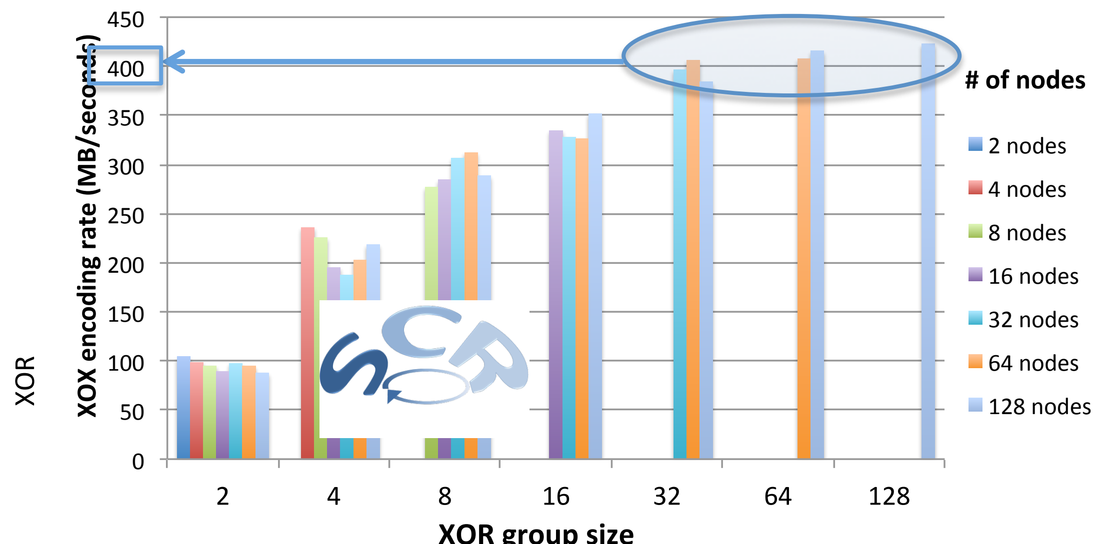


Failure rates (failures/week) on TSUBAME2.0

- **<u>Checkpoint size per node</u>**: 29GB
  - TSUBAME nodes memory: 58GB

Level-1

- **<u>XOR throughput</u>**: 400MB/s



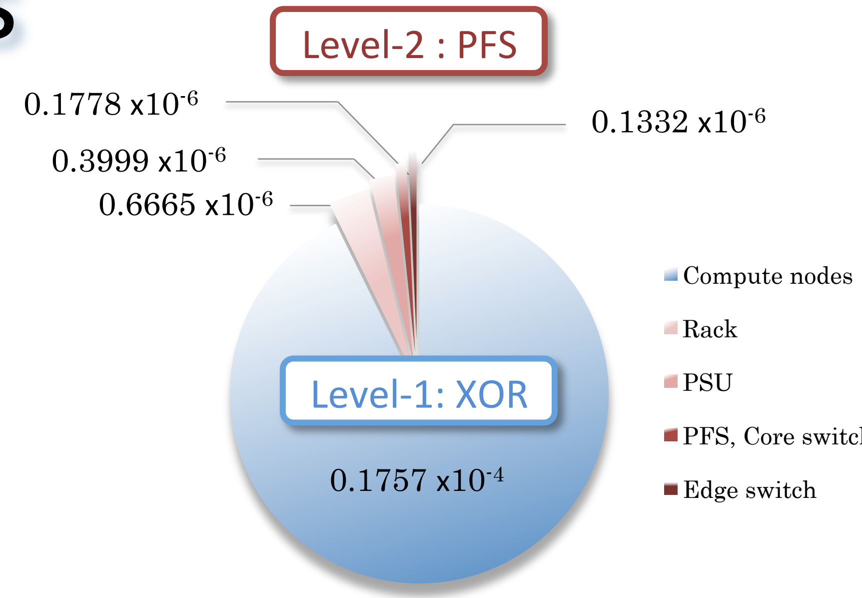
XOR encoding performance on TSUBAME2.0 using local SSDs

Model Parameters

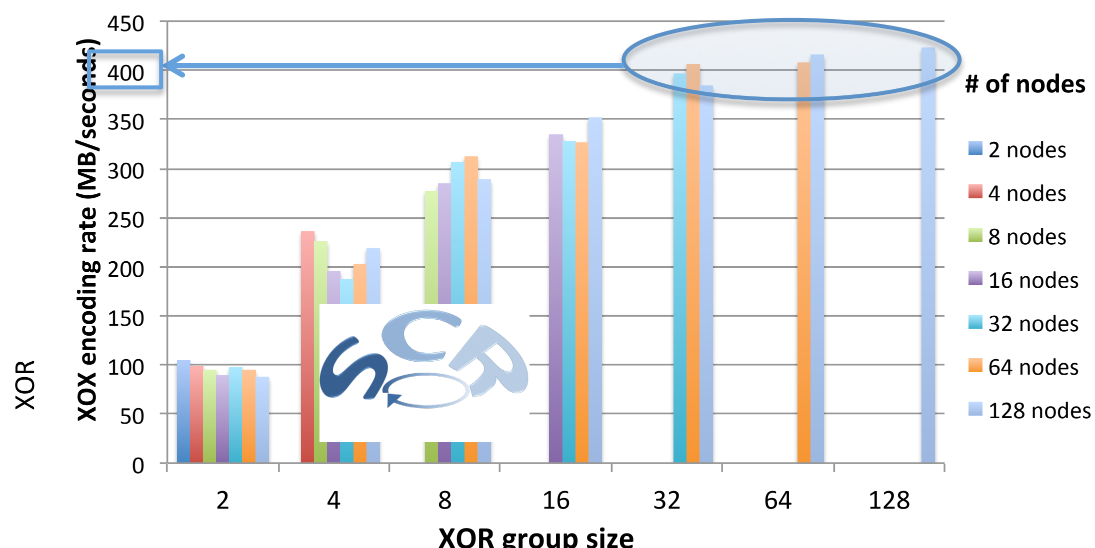
- **Failure rates**
  - 1.5 years (Nov 1st 2010 ~ Apr 6th 2012) failure history
- **Checkpoint size per node**: 29GB
  - TSUBAME nodes memory: 58GB

Level-1

- **XOR throughput**: 400MB/s



Failure rates (failures/second) on TSUBAME2.0



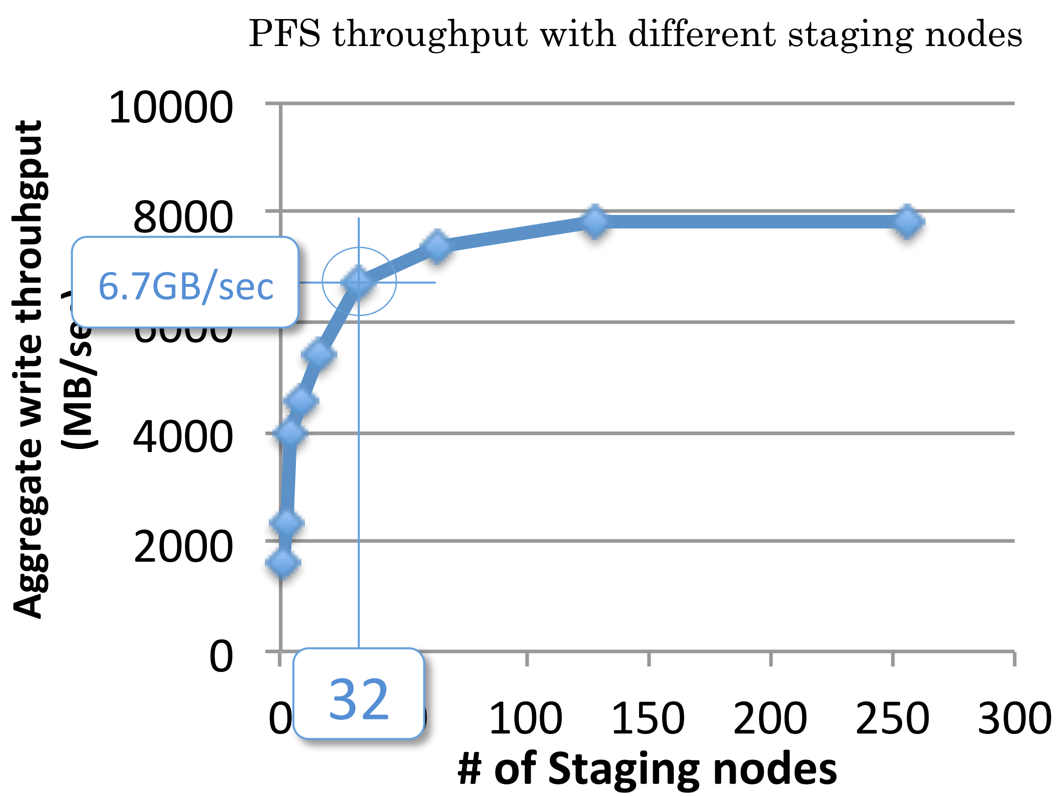
XOR encoding performance on TSUBAME2.0 using local SSDs

Staging node tuning for TSUBAME2.0

Level-2

- **# of Staging nodes**: 32 nodes
  - 2.3% of TSUBAME2.0 thin nodes (1408 nodes)
- **PFS throughput**: 6.7GB/seconds
  - 209.5 MB/seconds* per Staging node

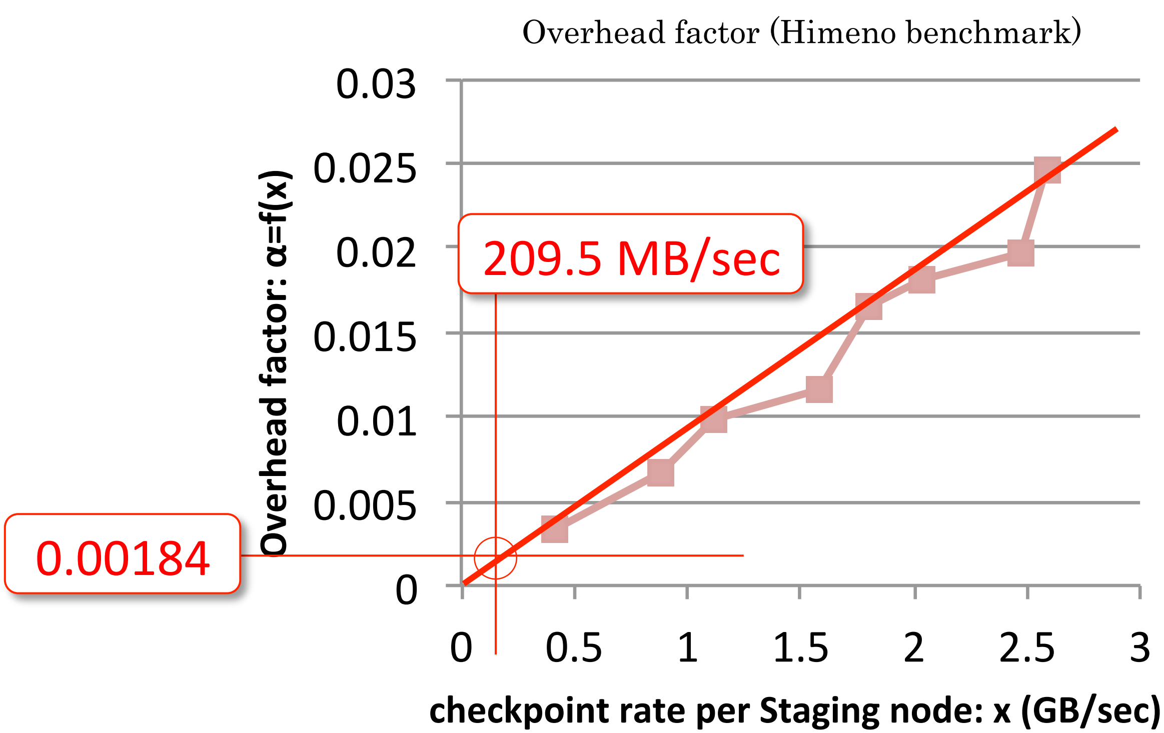
  * 6.7(GB/s) / 32(nodes) = 209.5



Overhead factor

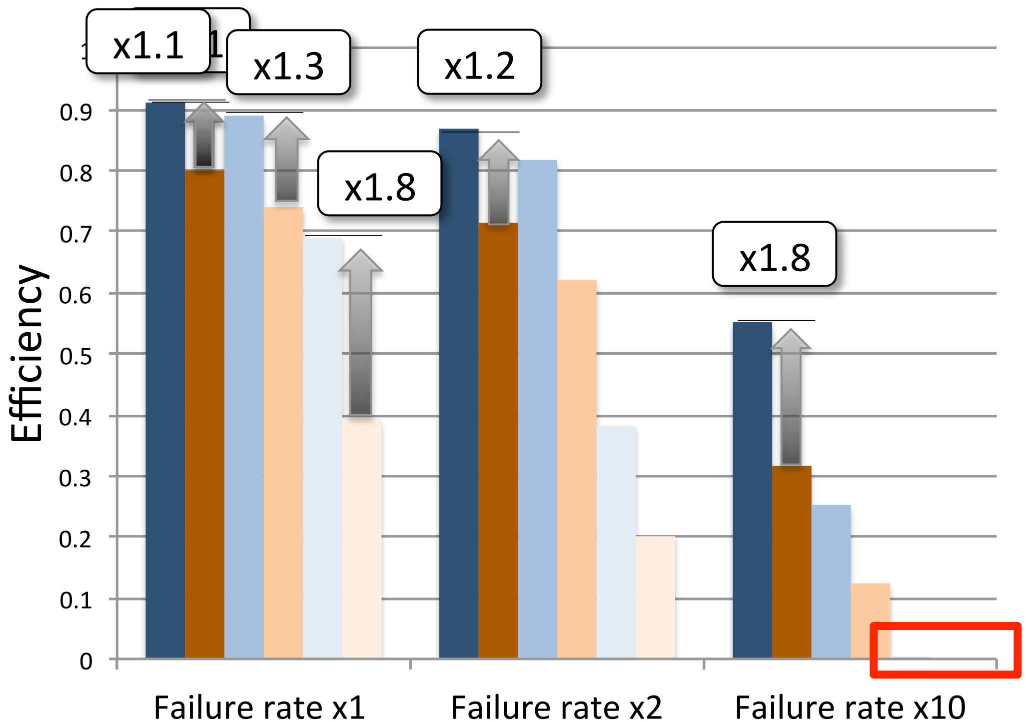
- **Overhead factor**: 0.00184 (0.184%)
  - For Himeno bechmark

RDMA ⇒ No CPU cycle, No redundant memcpy
RDMA read speed ⇒ 209.5MB/s < Network & Memory bandwidth



Efficiency: Non-blocking vs. blocking

The non-blocking method always achieves higher efficiency than the blocking method



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

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

$expected\ \ runtime$: Computed by the models

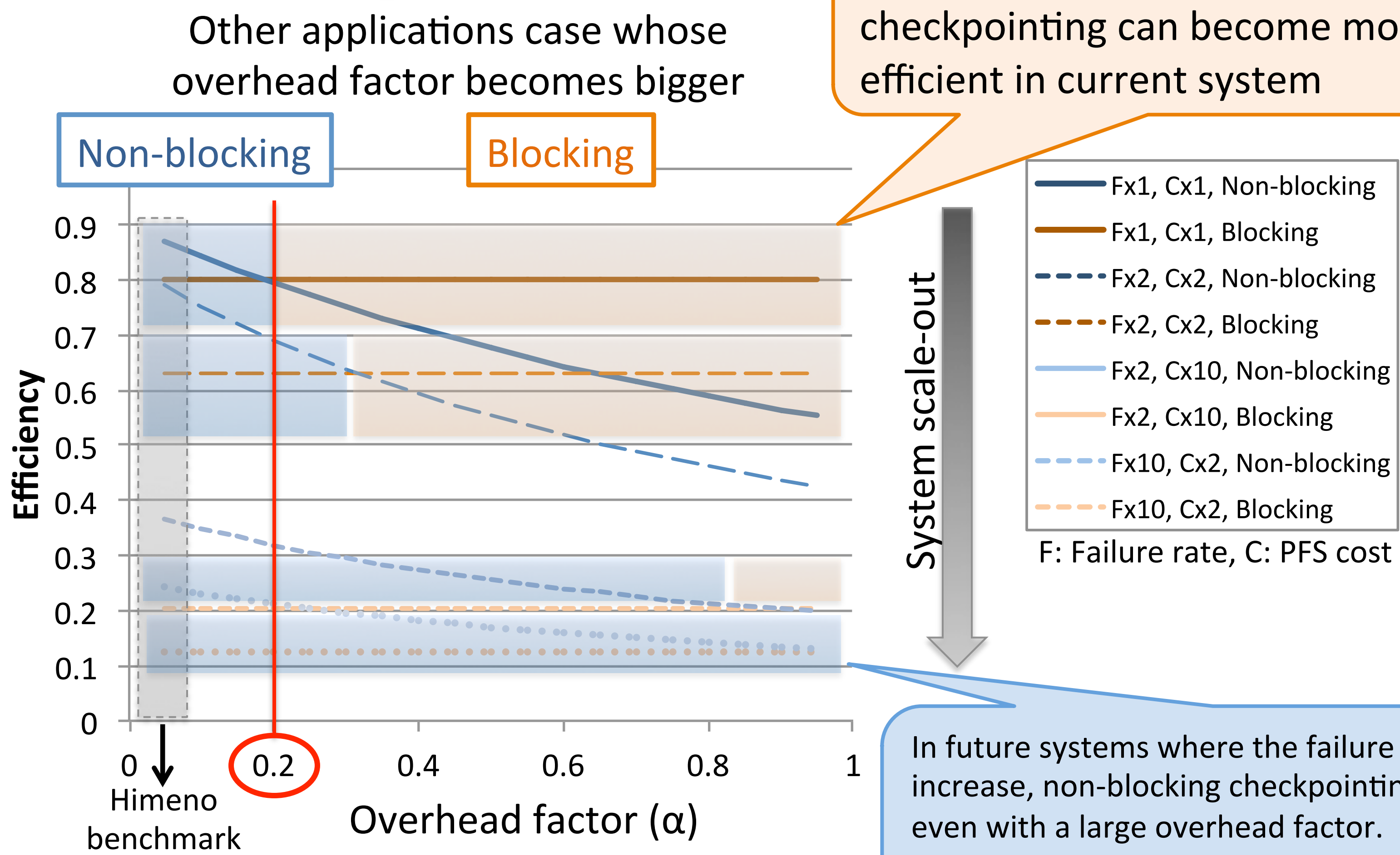
- PFS cost x1 / Non-blocking
- PFS cost x1 / Blocking
- PFS cost x2 / Non-blocking
- PFS cost x2 / Blocking
- PFS cost x10 / Non-blocking
- PFS cost x10 / Blocking

One TSUBAME2.0 node MTBF: 2.57 years
\# of Nodes: 1408 nodes

x10 scale-out →

No computation progresses !!

Overhead factor: Non-blocking vs. Blocking



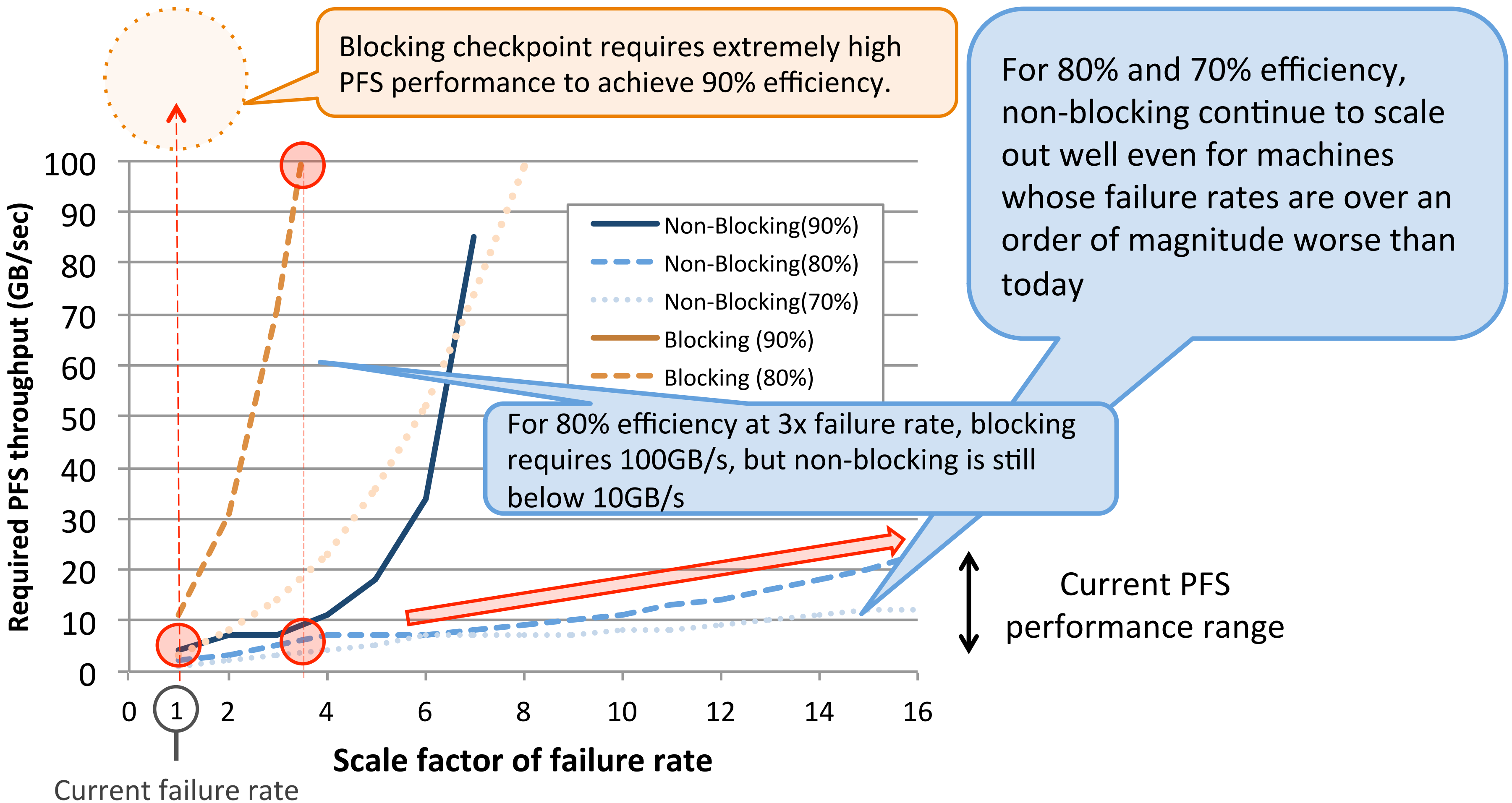
If overhead factor is over 0.2, blocking checkpointing can become more efficient in current system

In future systems where the failure rates and cost increase, non-blocking checkpointing can be effective even with a large overhead factor.
=> Blocking checkpoint overhead dominate the runtime more than overhead factor by non-blocking

Required PFS performance to meet given application efficiency

When building a reliable data center or supercomputer, two major concerns are monetary cost of the PFS and the PFS throughput required to maintain high efficiency …

=> predict required PFS performance with the models



Conclusion

- Developed an non-blocking checkpointing system
  - Write checkpoint data in the background using RDMA
- Markov model of the non-blocking checkpointing
  - Optimal multi-level checkpoint interval
  - Non-blocking v.s. Blocking checkpoint
    - Higher efficiency (1.1 ~ 1.8x) on current and future systems
  - High efficiency (up to 80%) with low PFS throughput

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

Co-authers

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

Acknowledgement

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52- 07NA27344. LLNL-PRES-599833-DRAFT.This work was also supported by Grant-in-Aid for Research Fellow of the Japan Society for the Promotion of Science (JSPS Fellows) 24008253, and Grant-in-Aid for Scientific Research S 23220003.