# A Job Pause Service under LAM/MPI+BLCR for Transparent Fault Tolerance

Chao Wang, Frank Mueller

*North Carolina State University*

Christian Engelmann, Stephen L. Scott

Oak Ridge National Laboratory

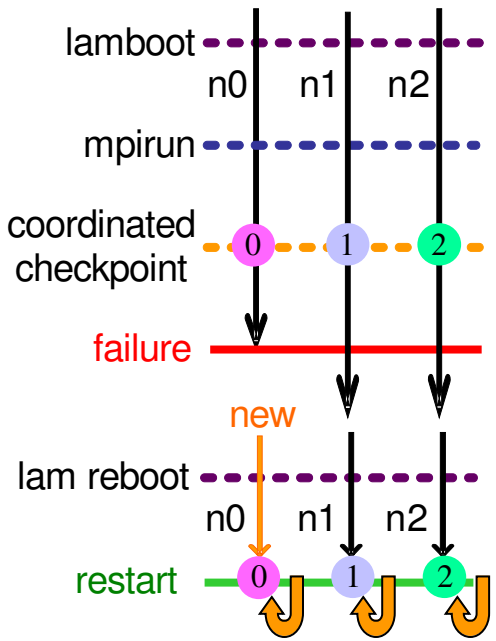
# Outline

- Problem vs. Our Solution
- Overview of LAM/MPI and BLCR
- Our Design and Implementation
- Experimental Framework
- Performance Evaluation
- Related Work
- Conclusion

# Problem Statement

- Trends in HPC: high end systems with thousands of processors
  - Increased probability of node failure: MTTF becomes shorter
- MPI widely accepted in scientific computing
  - But no fault recovery method in MPI standard

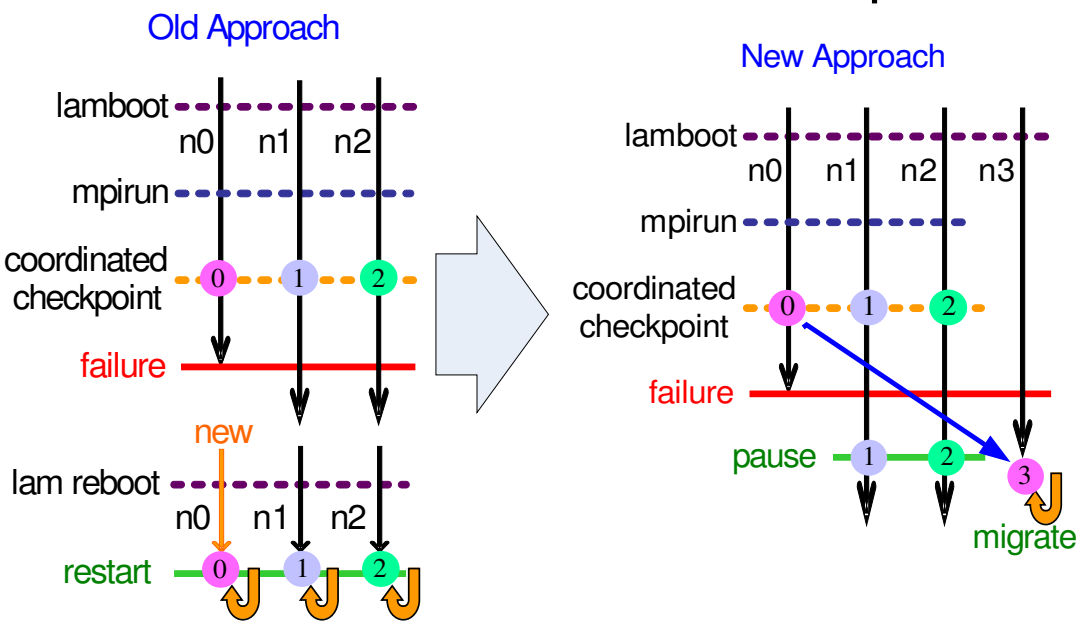
Today's MPI Job C/R




- Extensions to MPI for FT exist but…
  - Cannot dynamically add/delete nodes transparently at runtime
  - Must reboot LAM RTE
  - Must restart entire job
    - -Inefficient if only one/few node(s) fail
    - -Staging overhead
  - Requeuing penalty

# Our Solution - Job-pause Service

- Integrate group communication
  - Add/delete nodes
  - Detect node failures automatically
- Processes on live nodes remain active (roll back to last checkpoint)
- Only processes on failed nodes dynamically replaced by spares
  - resumed from the last checkpoint



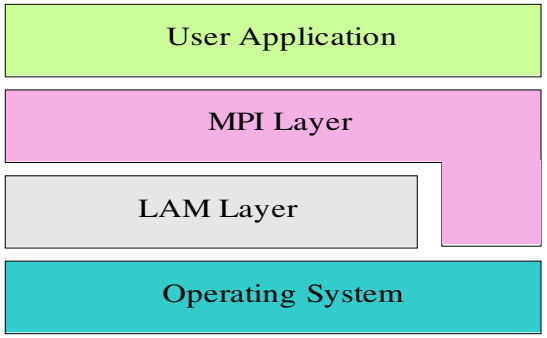

- Hence:
  - no restart of entire job
  - no staging overhead
  - no job requeue penalty
  - no Lam RTE reboot

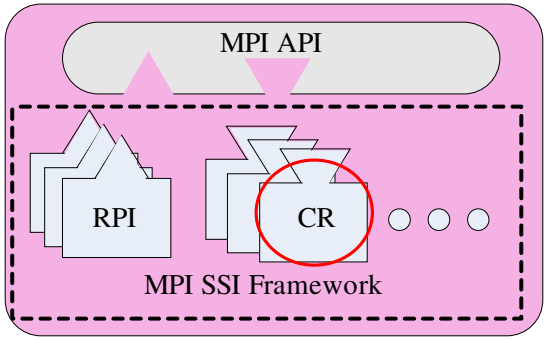
# Outline

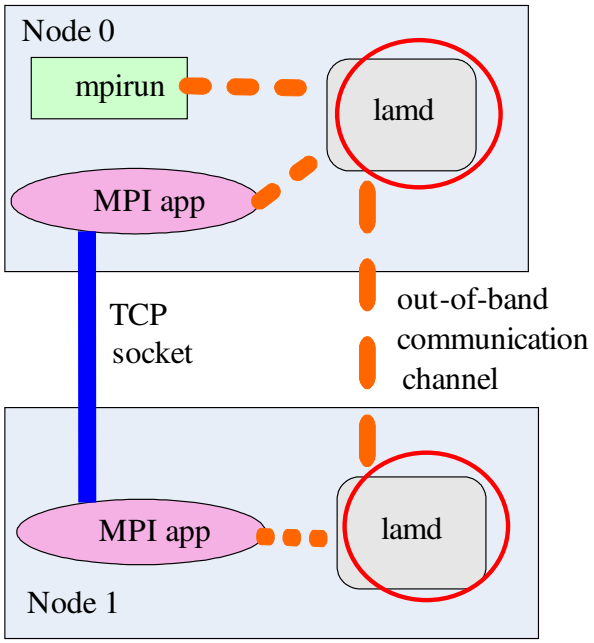
- Problem vs. Our Solution
- Overview of LAM/MPI and BLCR
- Our Design and Implementation
- Experimental Framework
- Performance Evaluation
- Related Work
- Conclusion

# LAM-MPI Overview

- Modular, component-based architecture
  - 2 major layers
  - Daemon-based RTE: lamd
  - "Plug in" C/R to MPI SSI framework:
  - Coordinated C/R & support BLCR




Ex: 2-node MPI job

RTE: Run-time Environment
SSI: System Services Interface
RPI: Request Progression Interface

# BLCR Overview

- Process-level C/R facility: for single MPI application process
- Kernel-based: saves/restores most/all resources
- Implementation: Linux kernel module
  - allows upgrades & bug fixes w/o reboot
- Provides hooks used for distributed C/R: LAM-MPI jobs

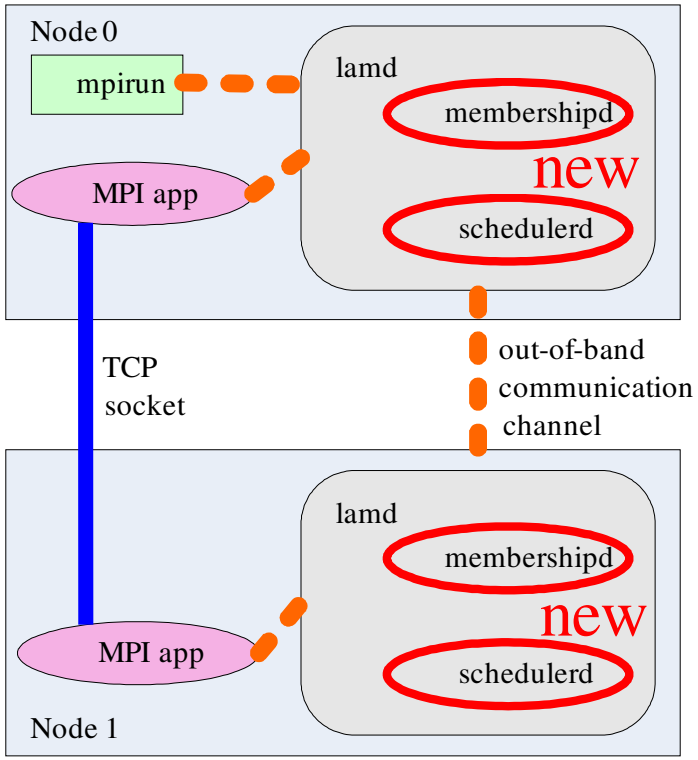
# Outline

- Problem vs. Our Solution
- Overview of LAM/MPI and BLCR
- Our Design and Implementation
- Experimental Framework
- Performance Evaluation
- Related Work
- Conclusion

# Our Design & Implementation – LAM/MPI

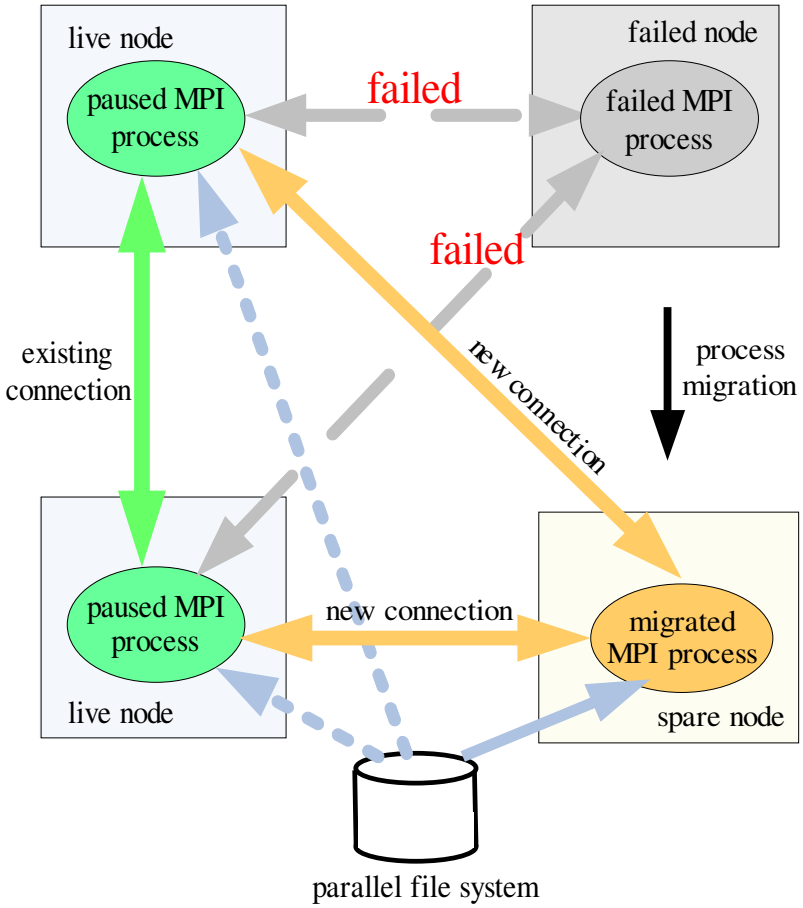
- Decentralized scalable Membership and failure detector (ICS'06)
  - Radix tree → scalability
  - dynamically detects node failures
  - NEW: Integrated into lamd

- NEW: Decentralized scheduler
  - Integrated into lamd
  - Periodic coordinated checkpointing
  - Node failure → trigger
  4. process migration (failed nodes)
  5. job-pause (operational nodes)



# New Job Pause Mechanism – LAM/MPI & BLCR

- Operational nodes: Pause
  - BLCR: reuse processes
  - restore part of state of process from checkpoint
  - LAM: reuse existing connections
- Failed nodes: Migrate
  - Restart on new node from checkpoint file
  - Connect w/ paused tasks

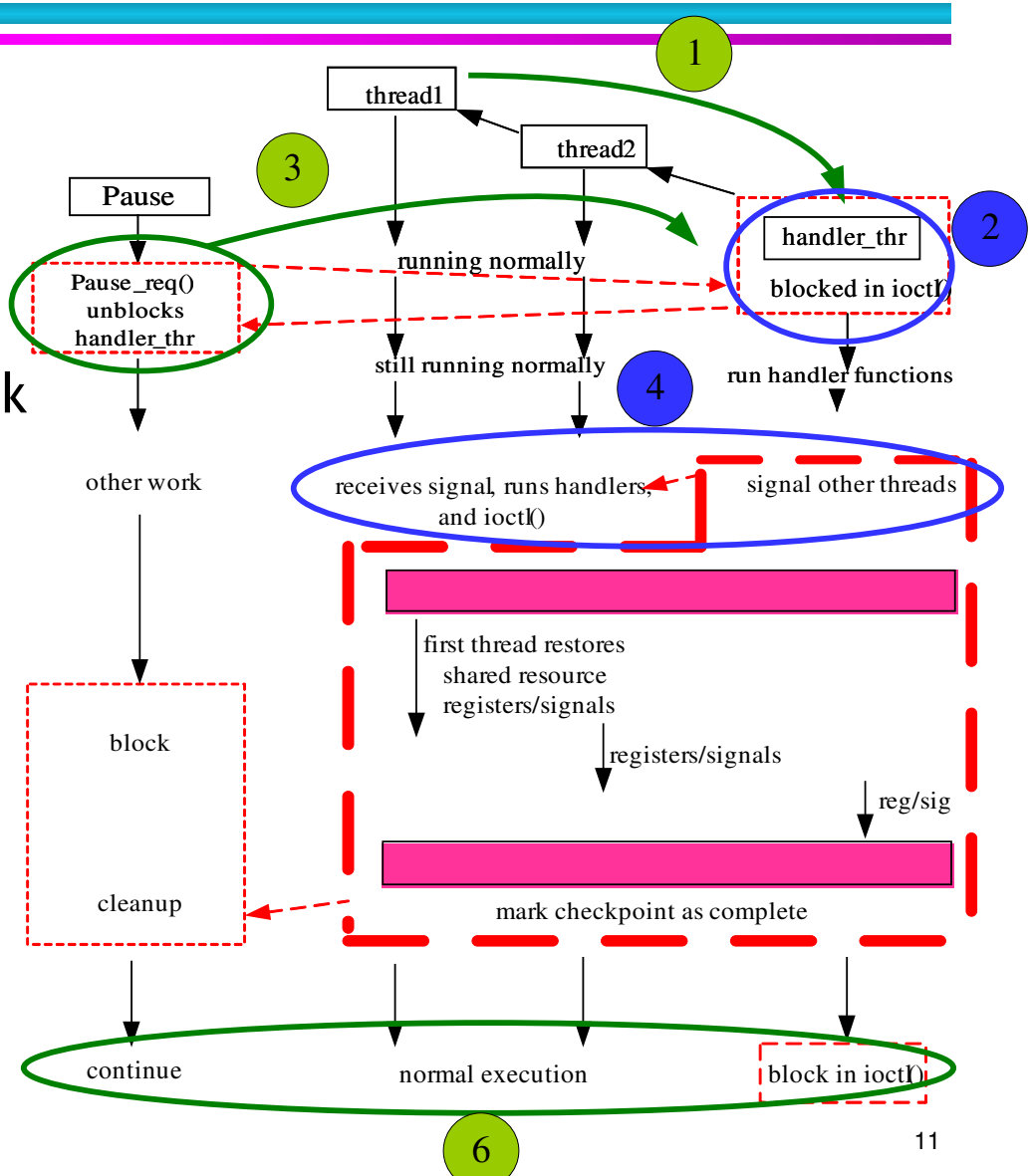


# New Job Pause Mechanism - BLCR

Call-back kernel thread: coordinates user command process and app. process

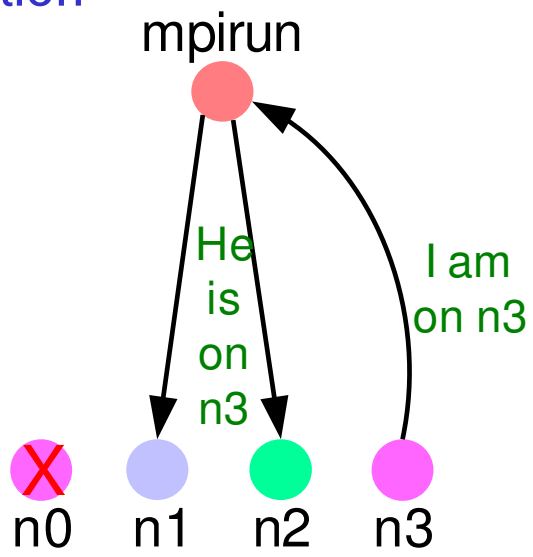
(In kernel: dashed lines/boxes)

1. app registers threaded callback → spawns callback thread
2. thread blocks in kernel
3. pause utility calls ioctl(), unblocks callback thread
4. All threads complete callbacks & enter kernel
5. New: All threads restore part of their states
6. Run regular application code from restored state



# Process Migration – LAM/MPI

- Change addressing information of migrated process
  - in process itself
  - in all other processes
- Use node id (not IP) for addressing information
- Update addressing information at run time
  1. Migrated process tells coordinator (mpirun) about its new location
  2. Coordinator broadcasts new location
  3. All processes update their process list




- No change to BLCR for Process Migration

# Outline

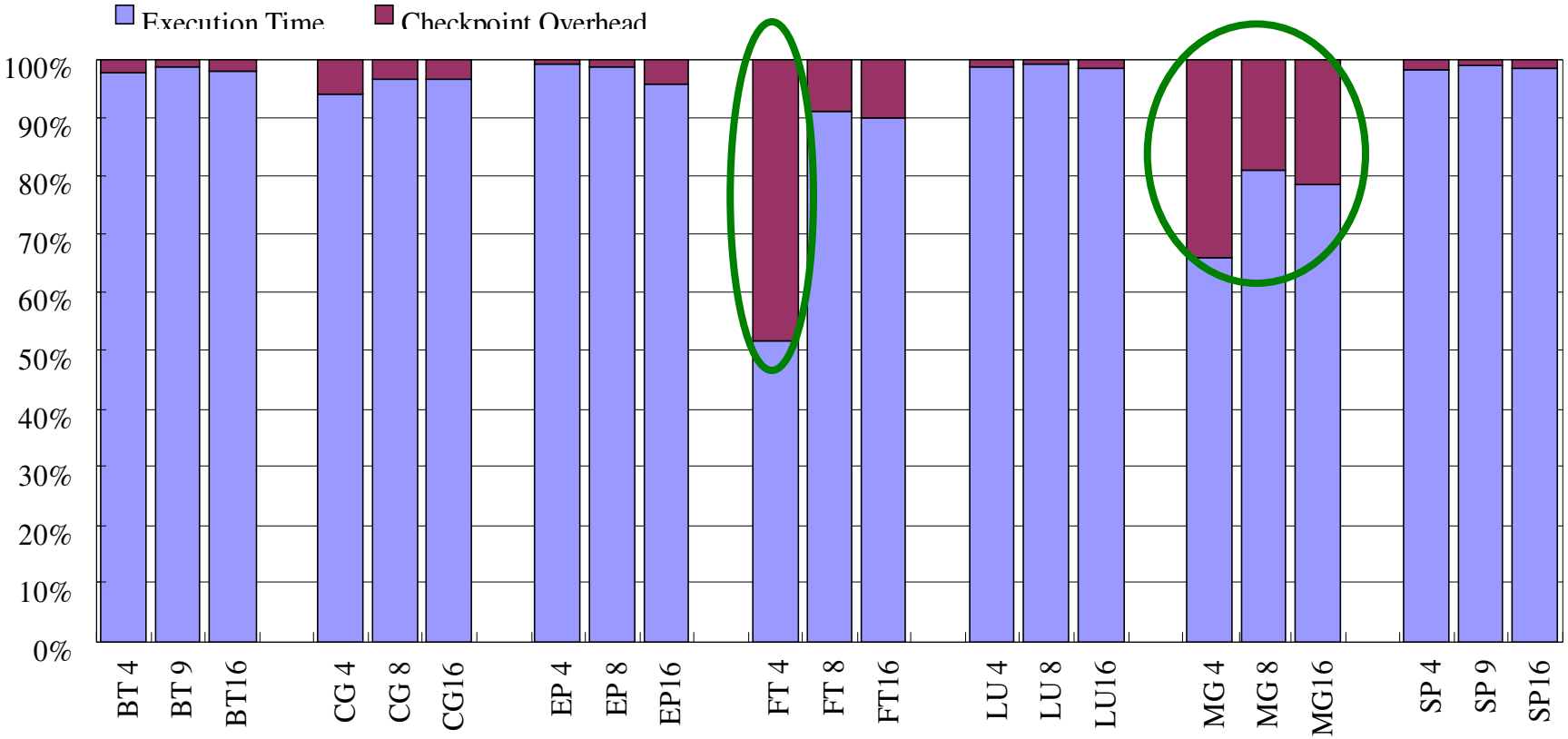
- Problem vs. Our Solution
- Overview of LAM/MPI and BLCR
- Our Design and Implementation
- Experimental Framework
- Performance Evaluation
- Related Work
- Conclusion

# Experimental Framework

- Experiments conducted on
  - Opt cluster: 16 nodes, 2 core, dual Opteron 265, 1 Gbps Ether
  - Fedora Core 5 Linux x86_64
  - Lam/MPI + BLCR w/ our extensions

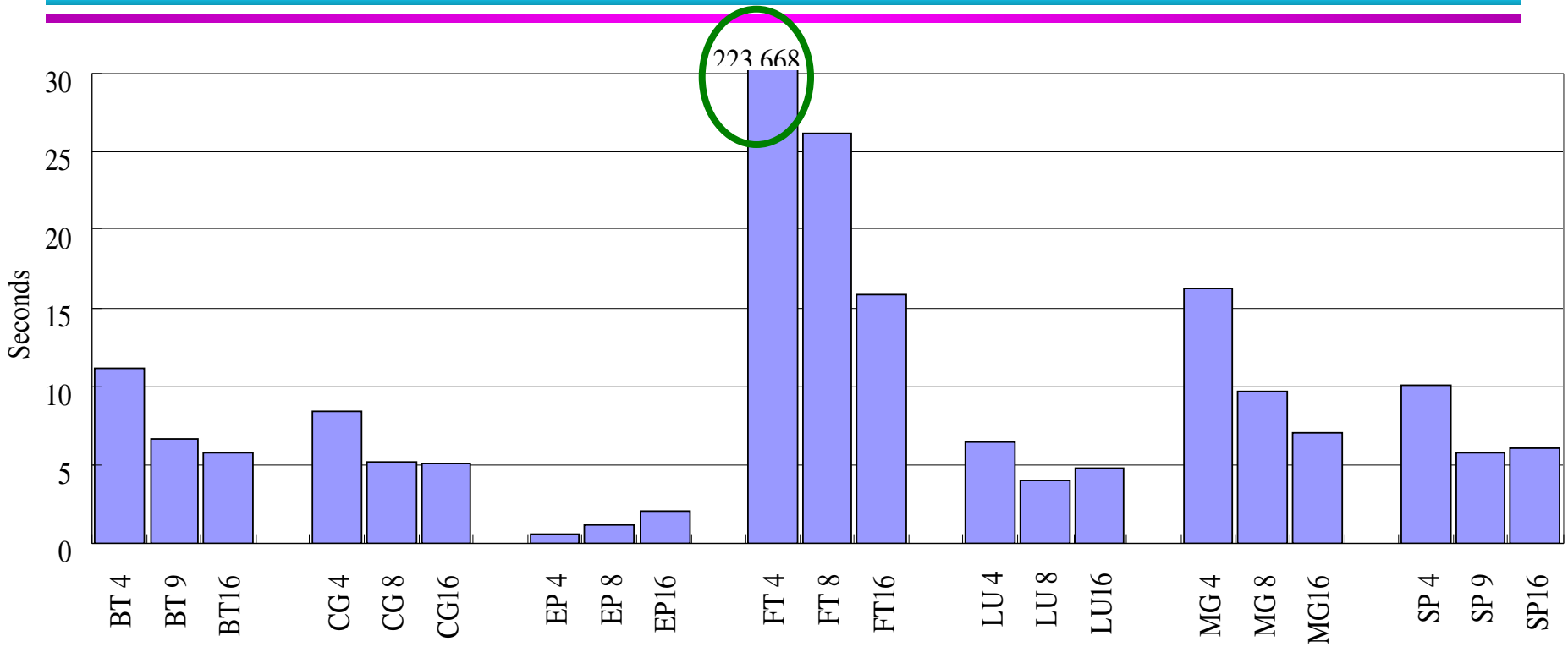
- Benchmarks
  - NAS V3.2.1 (MPI version)
    - run 5 times, results report avg.
    - Class C (large problem size) used
    - BT, CG, EP, FT, LU, MG and SP benchmarks
    - IS run is too short

# Relative Overhead (Single Checkpoint)



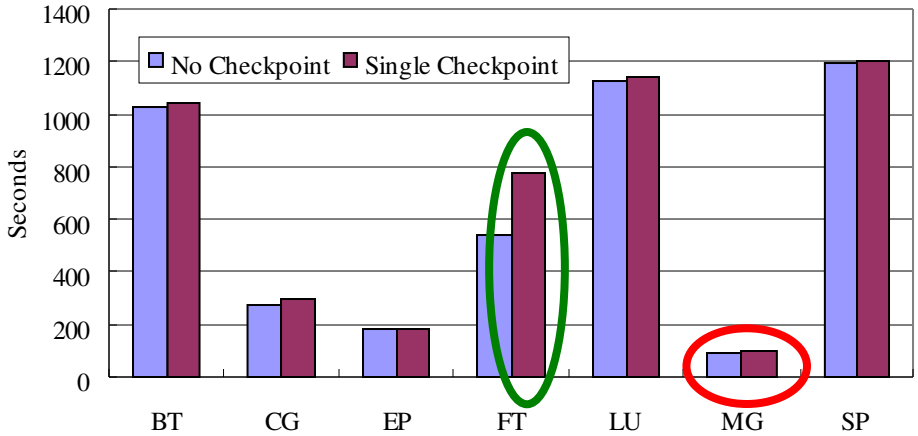

- Checkpoint overhead < 10%
- Except FT, MG (explained late)

# Absolute Overhead (Single Checkpoint)

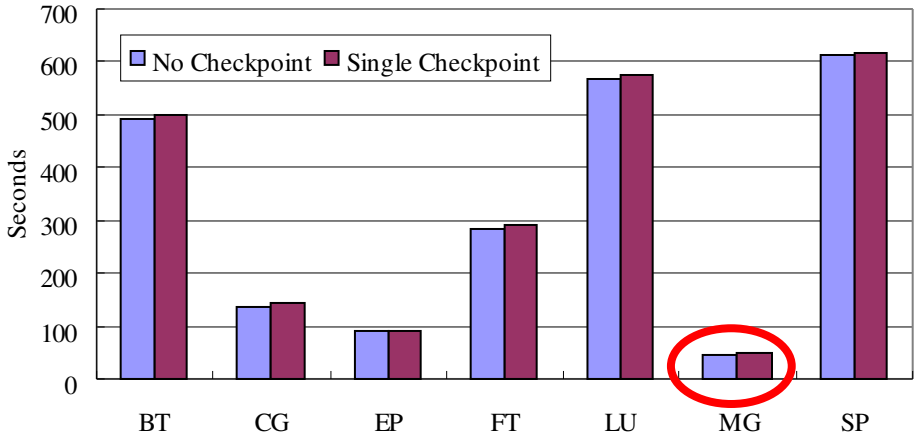



- Short: ~ 10 secs
- Checkpoint times increase linearly with checkpoint file size
- EP: small+const. chkpt file size → incr. communication overhead
- Except FT (explained next)

# Analysis of Outliers



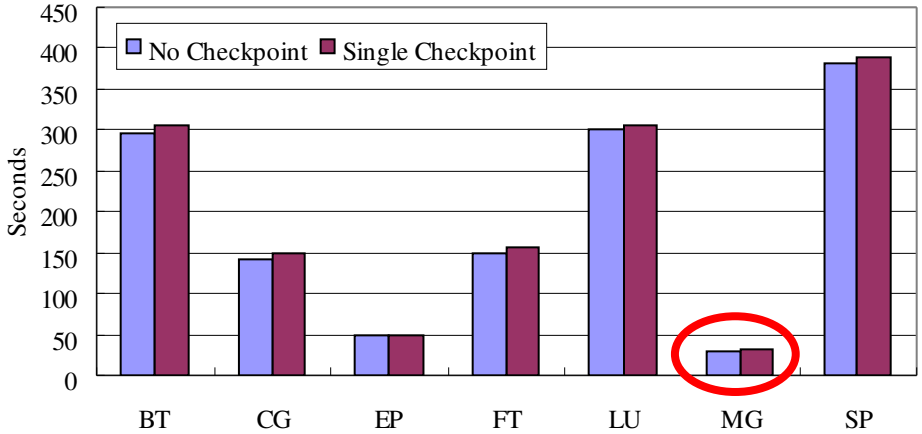

on 4 nodes




on 8 nodes

| Node # | BT | CG | EP | FT | LU | MG | SP |
|---|---|---|---|---|---|---|---|
| 4 | 406.9 | 250.88 | 1.33 | 1841.02 | 185.51 | 619.46 | 355.27 |
| 8 | 186.68 | 127.17 | 1.33 | 920.82 | 99.5 | 310.36 | 170.47 |
| 16 | 111.12 | 63.5 | 1.33 | 460.73 | 52.61 | 157.31 | 100.39 |

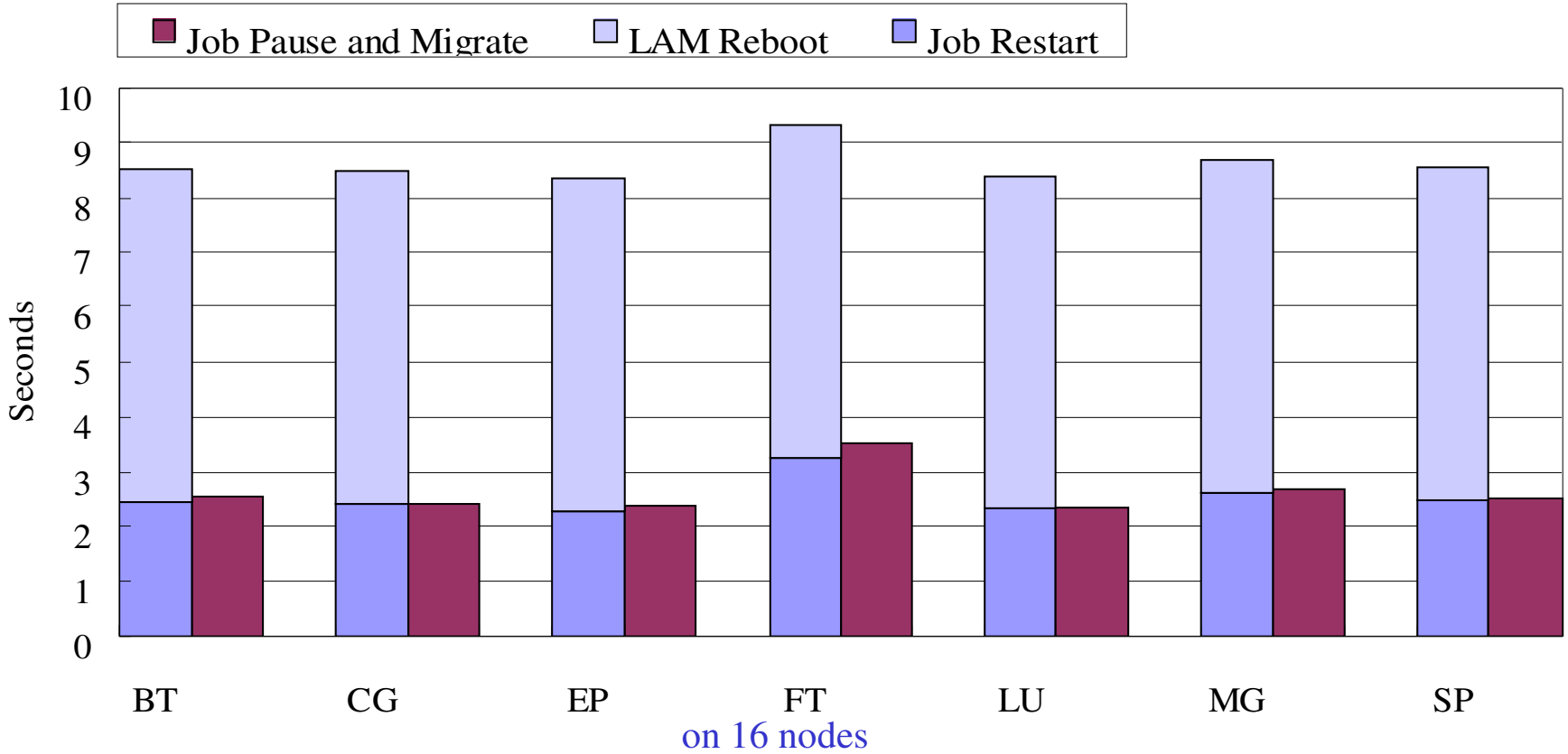
Size of checkpoint files [MB]




on 16 nodes

- Large Checkpoint files
- FT: thrashing/swap (BLCR problem)
- MG: large checkpoint files, but short overall exec time

# Job Migration Overhead




69.6% < job restart + lam reboot

- NO LAM Reboot
- No requeue penalty
- Transparent continuation of exec
  - Less staging overhead

# Related Work

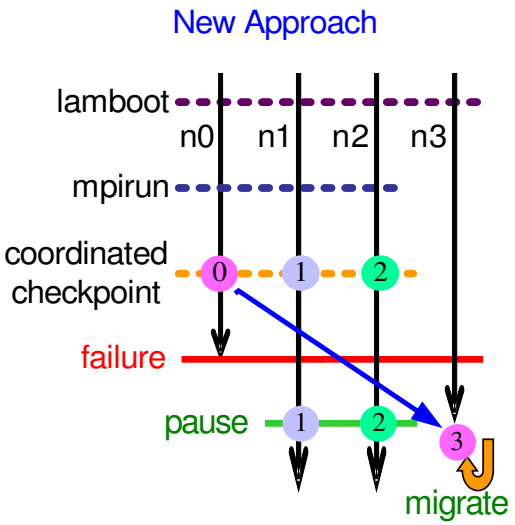
FT – Reactive approach

- Transparent
  - Checkpoint/restart
    - LAM/MPI w/ BLCR *[S.Sankaran et.al LACSI '03]*
      - Process Migration: scan & update checkpoint files *[J. Cao, Y. Li and M.Guo, ICPADS, 2005]* → *still requires restart of entire job*
    - *CoCheck [G.Stellner, IPPS ' 96]*
  - Log based (Log msg + temporal ordering)
    - MPICH-V *[G.Bosilica , Supercomputing, 2002]*
- Non-transparent
  - Explicit invocation of checkpoint routines
    - LA-MPI *[R.T.Aulwes et. Al, IPDPS 2004]*
    - FT-MPI *[G. E. Fagg and J. J. Dongarra, 2000]*

# Conclusion

## Job-Pause for fault tolerance in HPC

- Design generic for any MPI implementation / process C/R
- Implemented over LAM-MPI w/ BLCR
- Decentralized P2P scalable membership protocol & scheduler
- High-performance job-pause for operational nodes
- Process migration for failed nodes
- Completely transparent
- Low overhead: 69.6% < job restart + lam reboot
  - No job requeue overhead
  - Less staging cost
  - No LAM Reboot
- Suitable for proactive fault tolerance with diskless migration



# Questions?

**Thank you!**