Proactive Process-Level Live Migration in HPC Environments

Chao Wang, Frank Mueller

North Carolina State University

Christian Engelmann, Stephen L. Scott

Oak Ridge National Laboratory

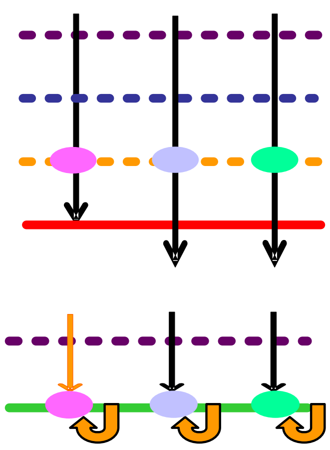
SC'08 Nov. 20 Austin, Texas

Outline

- Problem vs. Our Solution
- Overview of LAM/MPI and BLCR (Berkeley Lab Checkpoint/Restart)
- Our Design and Implementation
- Experimental Framework
- Performance Evaluation
- Conclusion and Future Work
- Related Work

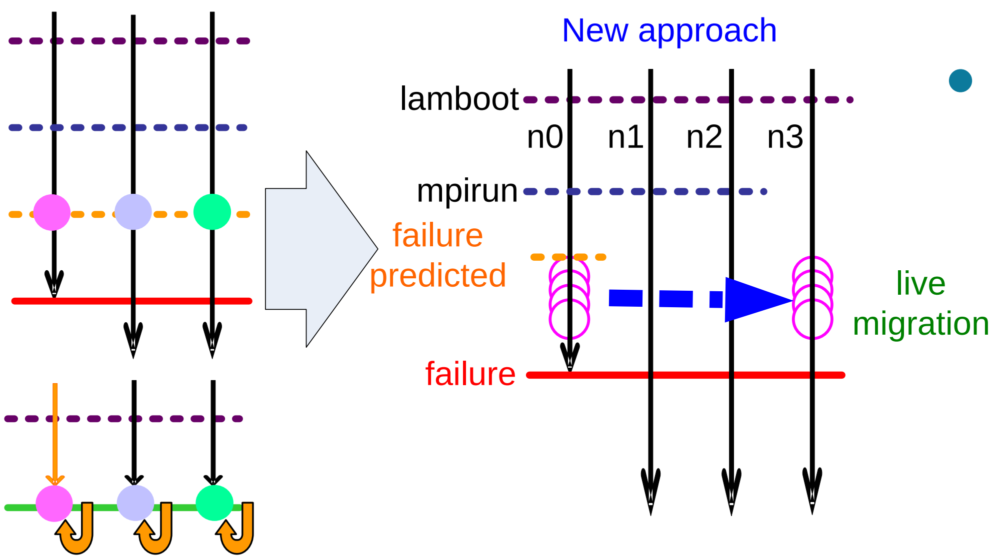
Problem Statement

- Trends in HPC: high end systems with > 100,000 processors
  - MTBF/I becomes shorter
- MPI widely accepted in scientific computing
  - But no fault recovery method in MPI standard
- Transparent C/R: (Checkpoint/Restart)
  - Coordinated: LAM/MPI w/ BLCR *[LACSI '03]*
  - *Uncoordinated*, Log based: MPICH-V *[SC 2002]*
- Non-transparent C/R: Explicit invocation of checkpoint routines
  - LA-MPI *[IPDPS 2004]* / FT-MPI *[EuroPVM-MPI 2000]*
- Frequently deployed C/R helps but…
  - 60% overhead on C/R *[I.Philp HPCRI'05]*
    - 100 hrs job -> 251 hrs
  - Must restart all job tasks
    - Inefficient if only one (few) node(s) fails
    - Staging overhead
  - Requeuing penalty



Our Solution – Proactive Live Migration

- High failure prediction accuracy with a prior warning window:
  - up to 70% reported [Gu et. Al, ICDCS'08] [R.Sahoo et.al KDD '03]
  - Active research field
  - Premise for live migration
- Processes on live nodes remain active
- Only processes on "unhealthy" nodes are lively migrated to spares



- Hence, avoid:
  - High overhead on C/R
  - Restart of all job tasks
    - Staging overhead
  - Job requeue penalty
  - Lam RTE reboot

Proactive FT Complements Reactive FT

$$T_c = \sqrt{2 \times T_s \times T_f}$$ *[J.W.Young Commun. ACM '74]*

Tc: time interval between checkpoints

Ts: time to save checkpoint information (mean Ts for BT/CG/FT/LU/SP Class C on 4/8/16 nodes is 23 seconds)

Tf: MTBF, 1.25hrs *[I.Philp HPCRI'05]*

$$T_c = \sqrt{2 \times 23 \times (1.25 \times 60 \times 60)} = 455$$

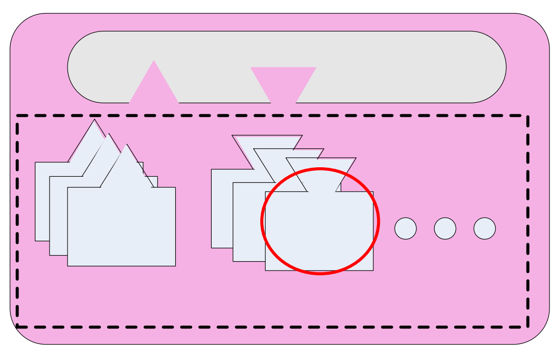
Assume 70% faults *[R.Sahoo et.al KDD '03]* can be predicted/handled proactively

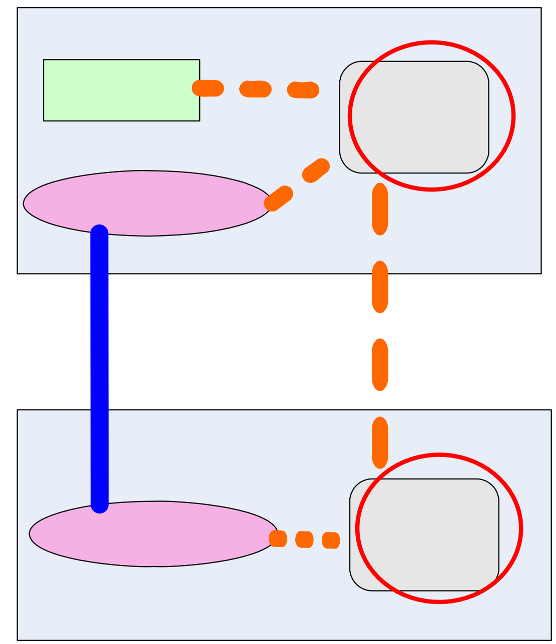
$$T_c = \sqrt{2 \times 23 \times (1.25/(1-0.7) \times 60 \times 60)} = 831$$

- Proactive FT cuts checkpoint frequency in half!
- Future work: use 1. better fault model 2. Ts/Tf on bigger cluster to measure its complementation effect

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





- Example: A two-way MPI job on two nodes

RTE: Run-time Environment
SSI: System Services Interface
RPI: Request Progression Interface
MPI: Message Passing Interface
LAM: Local Area Multi-computer

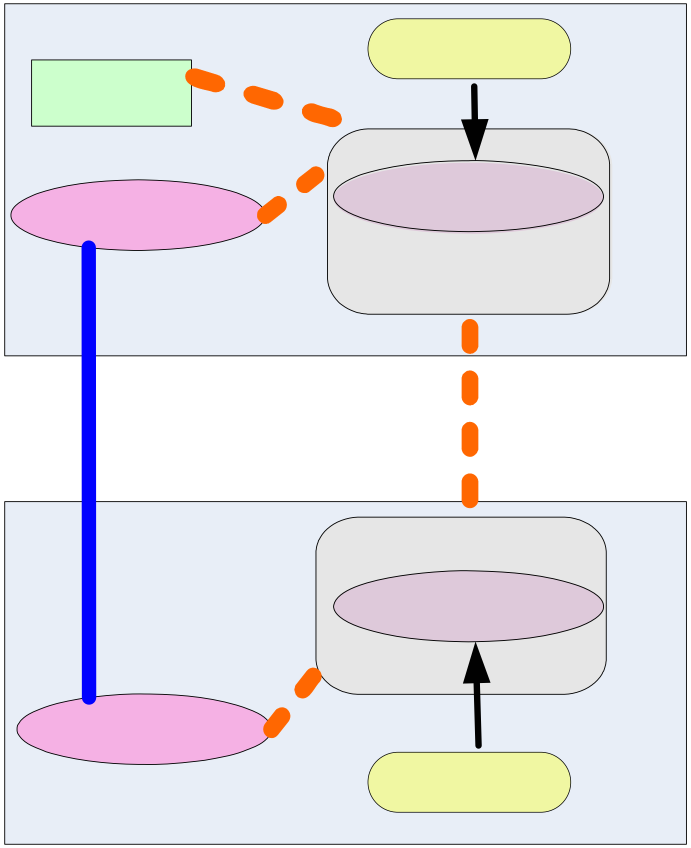
BLCR Overview

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

Our Design & Implementation – LAM/MPI

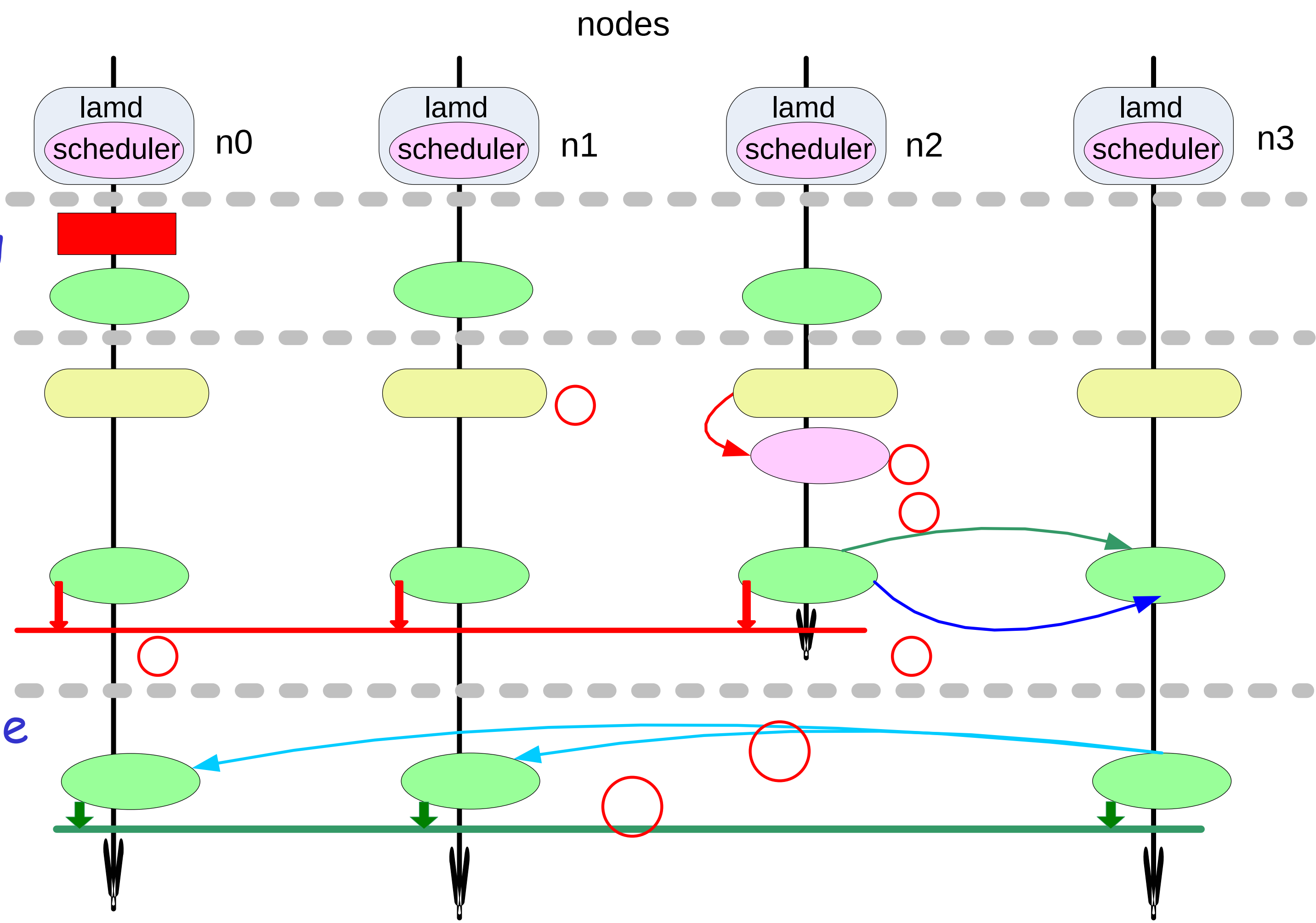
- Per-node health monitoring mechanism
  - Baseboard management controller (BMC)
  - Intelligent platform management interface (IPMI)

- NEW: Decentralized scheduler
  - Integrated into lamd
  - Notified by BMC/IPMI
  - Migration destination determination
  - Trigger migration



Live Migration Mechanism – LAM/MPI & BLCR

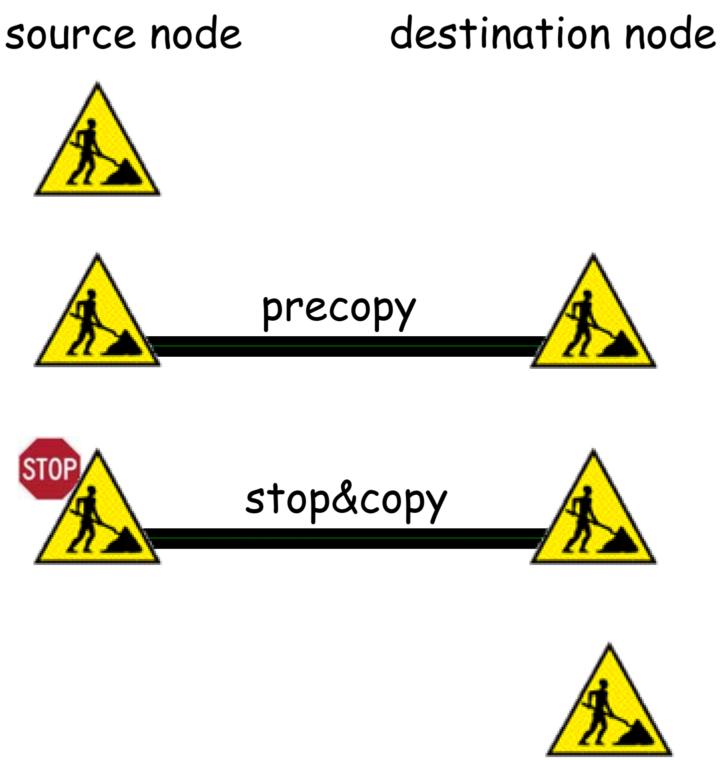
- MPI RTE setup
- MPI Job running
- Live migration
- Job exec. resume

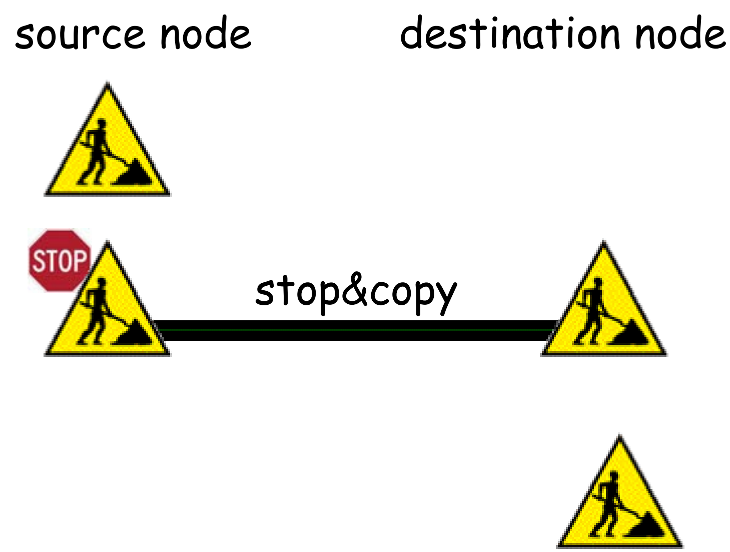


- Step 3 is optional: live migration (w/ step 3) vs. frozen (w/o step 3)

Live Migration vs. Frozen Migration

- Live migration
  - w/ precopy
- Frozen migration
  - w/o precopy
  - stop&copy-only



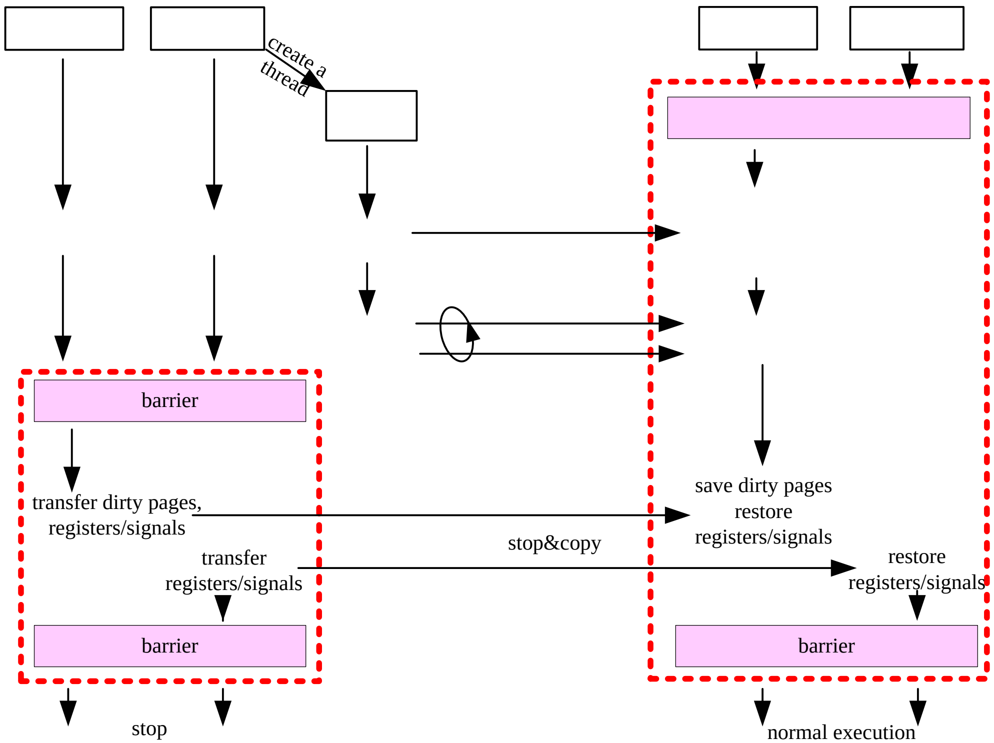


Live Migration - BLCR

New process created on destination node

Precopy: transfer dirty pages iteratively

Stop&copy

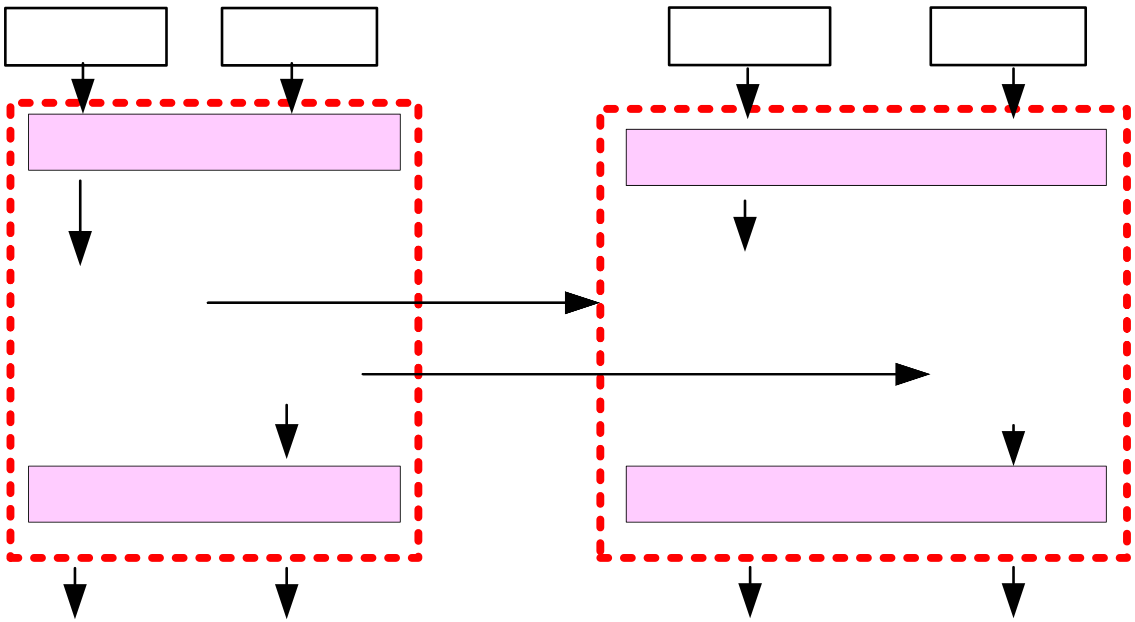


(In kernel: dashed lines/boxes)

Page-table dirty bit scheme:

1. dirty bit of PTE duplicated
2. kernel-level functions extended to set the duplicated bit w/o additional overhead

Frozen Migration - BLCR



Live vs. Frozen migration (also for precopy termination conditions):

1. Thresholds, e.g., temperature threshold
2. Available network bandwidth determined by dynamic monitoring SOUr
3. Size of write set

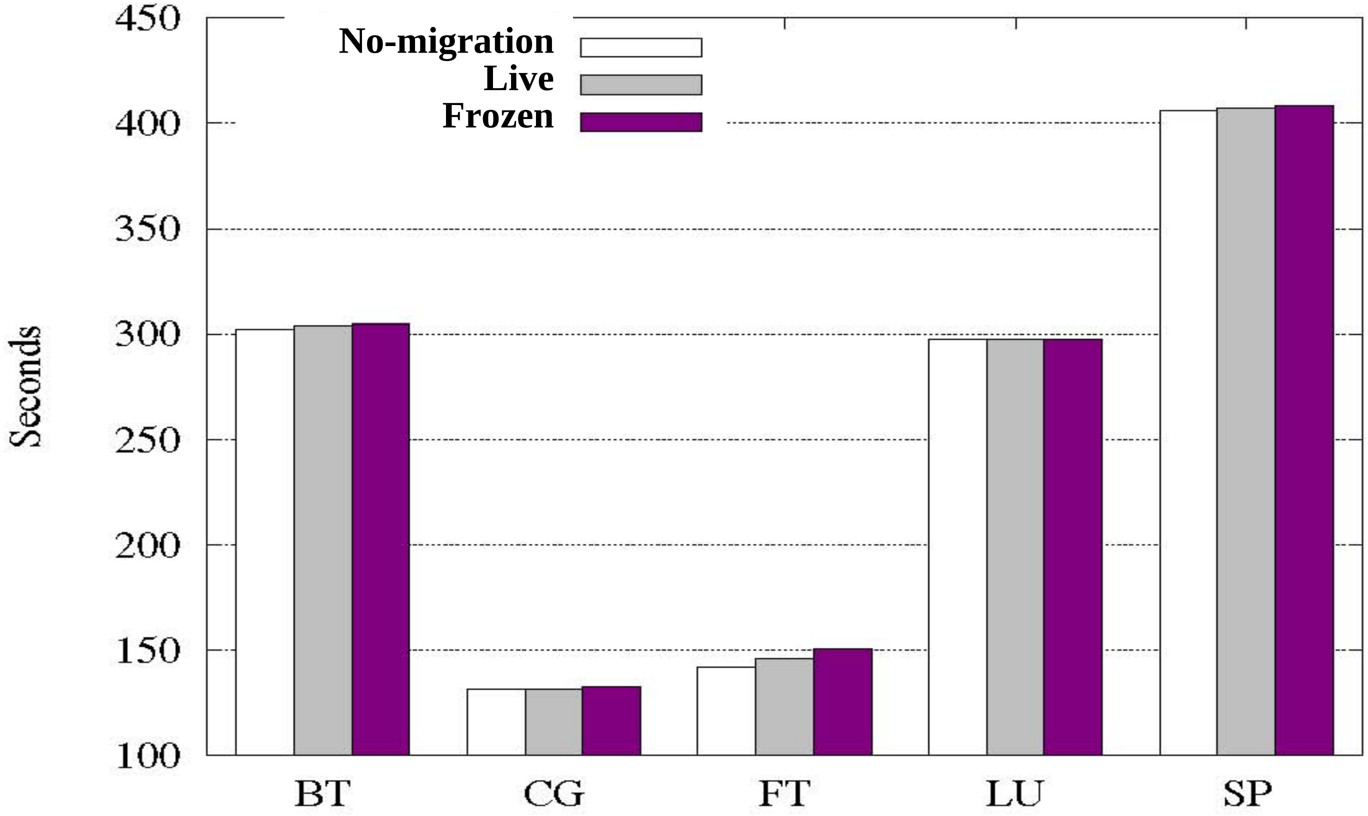
Future work: heuristic algorithm based on these conditions thread1

Experimental Framework

- Experiments conducted on
  - Opt cluster: 17 nodes, 2 core, dual Opteron 265, 1 Gbps Ether
  - Fedora Core 5 Linux x86_64
  - Lam/MPI + BLCR w/ our extensions

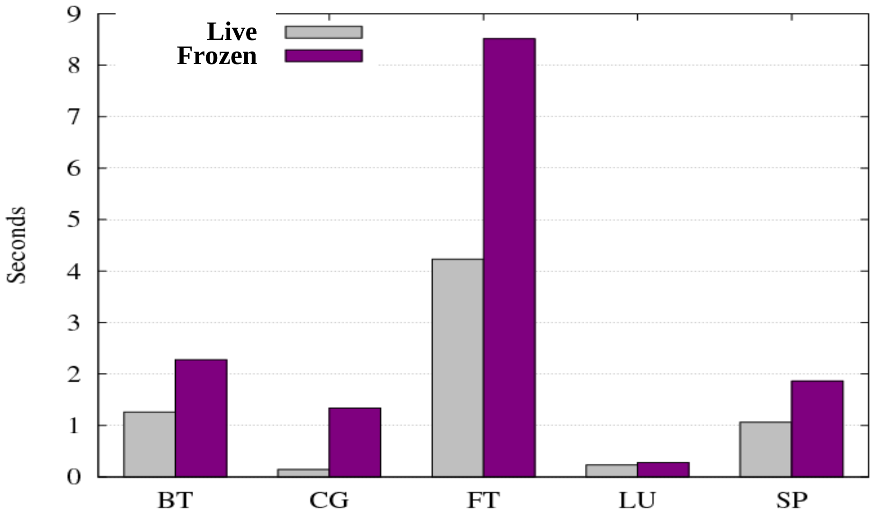
- Benchmarks
  - NAS V3.2.1 (MPI version)
    - BT, CG, FT, LU, and SP benchmarks
    - EP, IS and MG run is too short

Job Execution Time for NPB

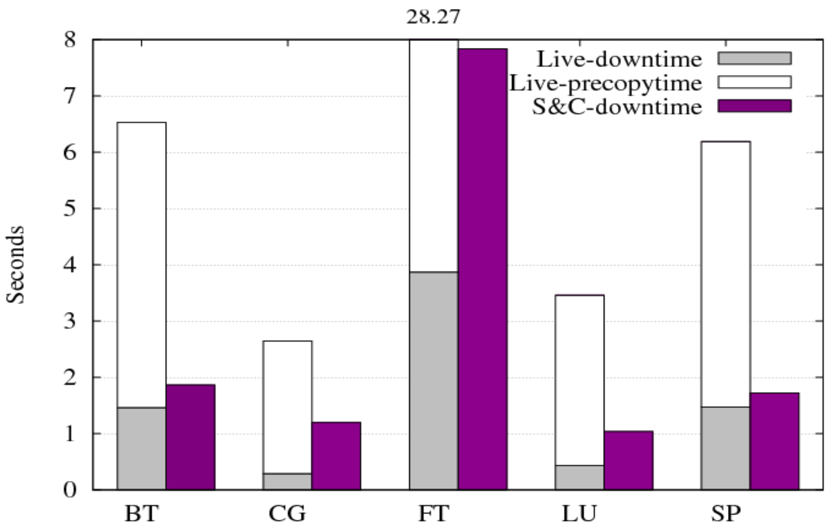


- NPB Class C on 16 Nodes
- Migration overhead: difference of job run time w/ and w/o migration

Migration Overhead and Duration



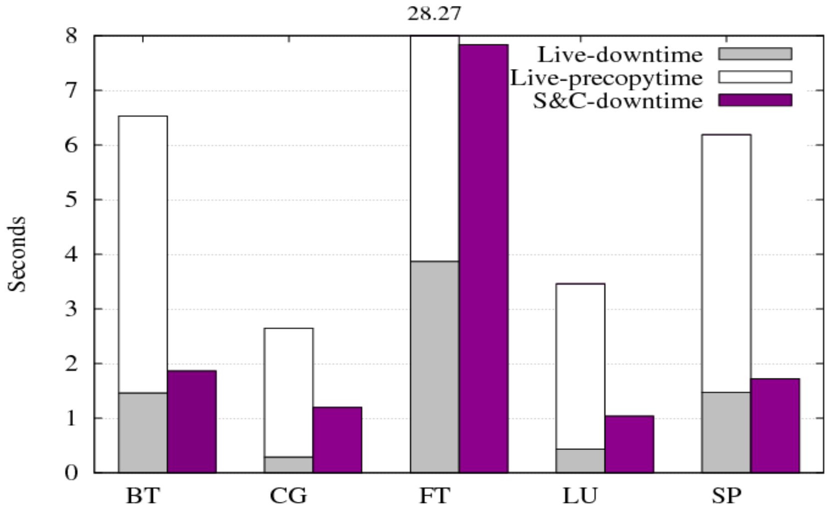
Migration Overhead



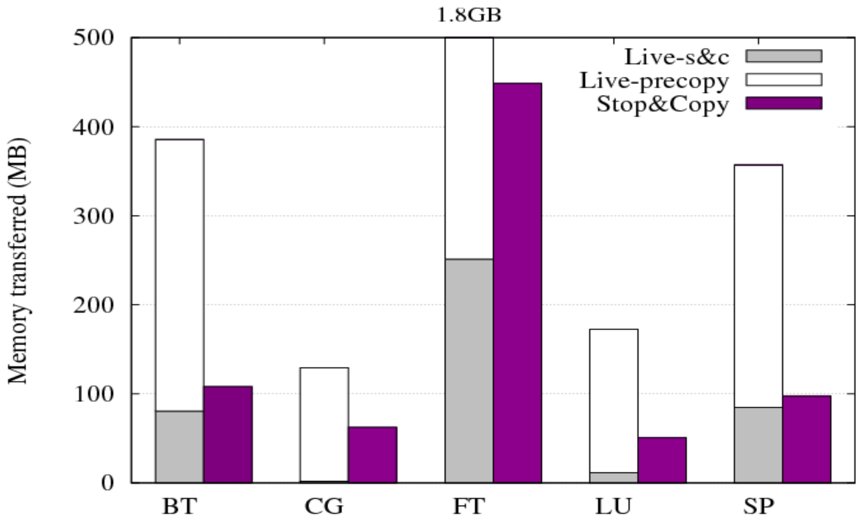
Migration Duration
(S&C = Frozen)

- Live: 0.08-2.98% overhead Frozen: 0.09-6% of benchmark runtime
- Penalty of shorter downtime of live migration: prolonged precopy
  - No significant impact to job run time, longer prior warning window required

Migration Duration and Memory Transferred



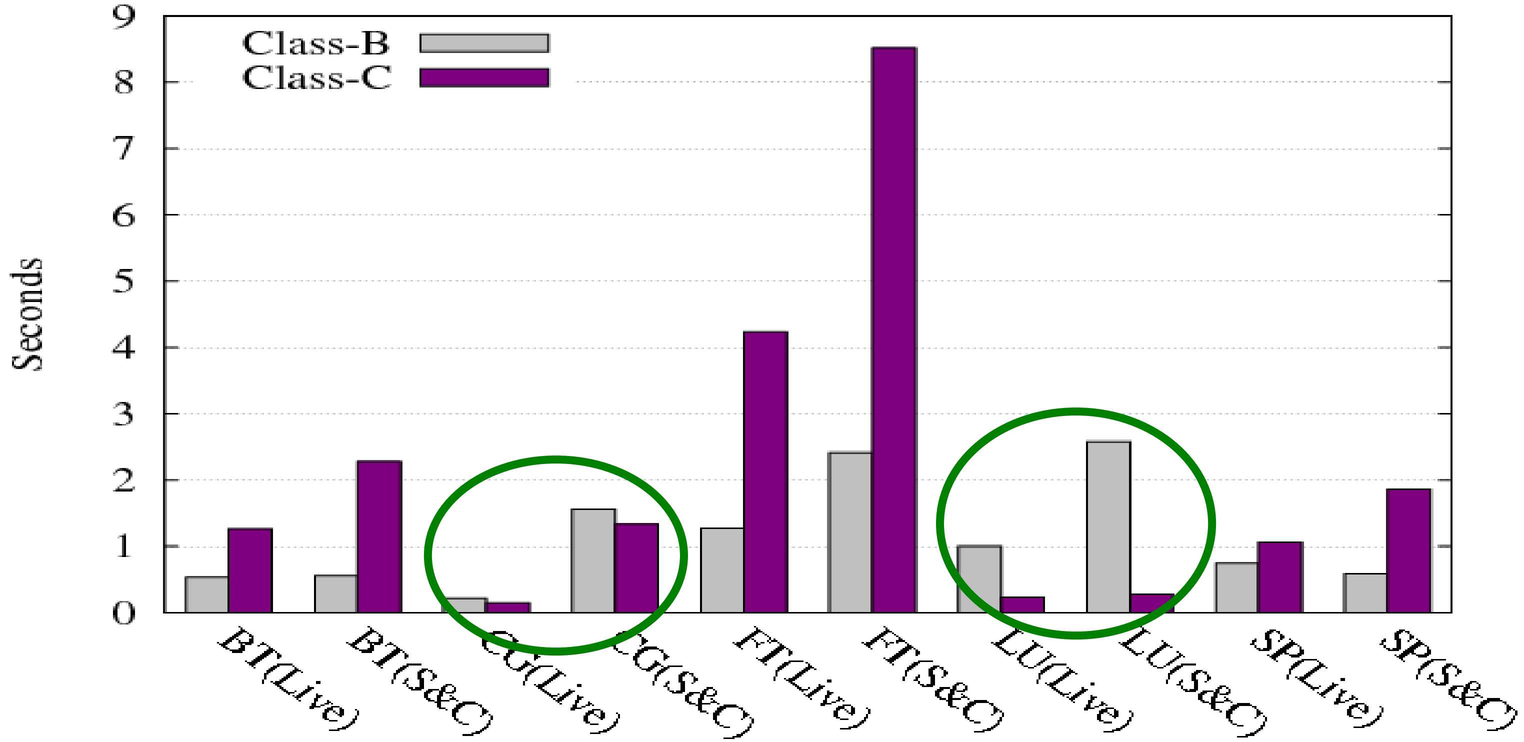
Migration Duration



Memory Transferred

- Migration duration is consistent to memory transferred

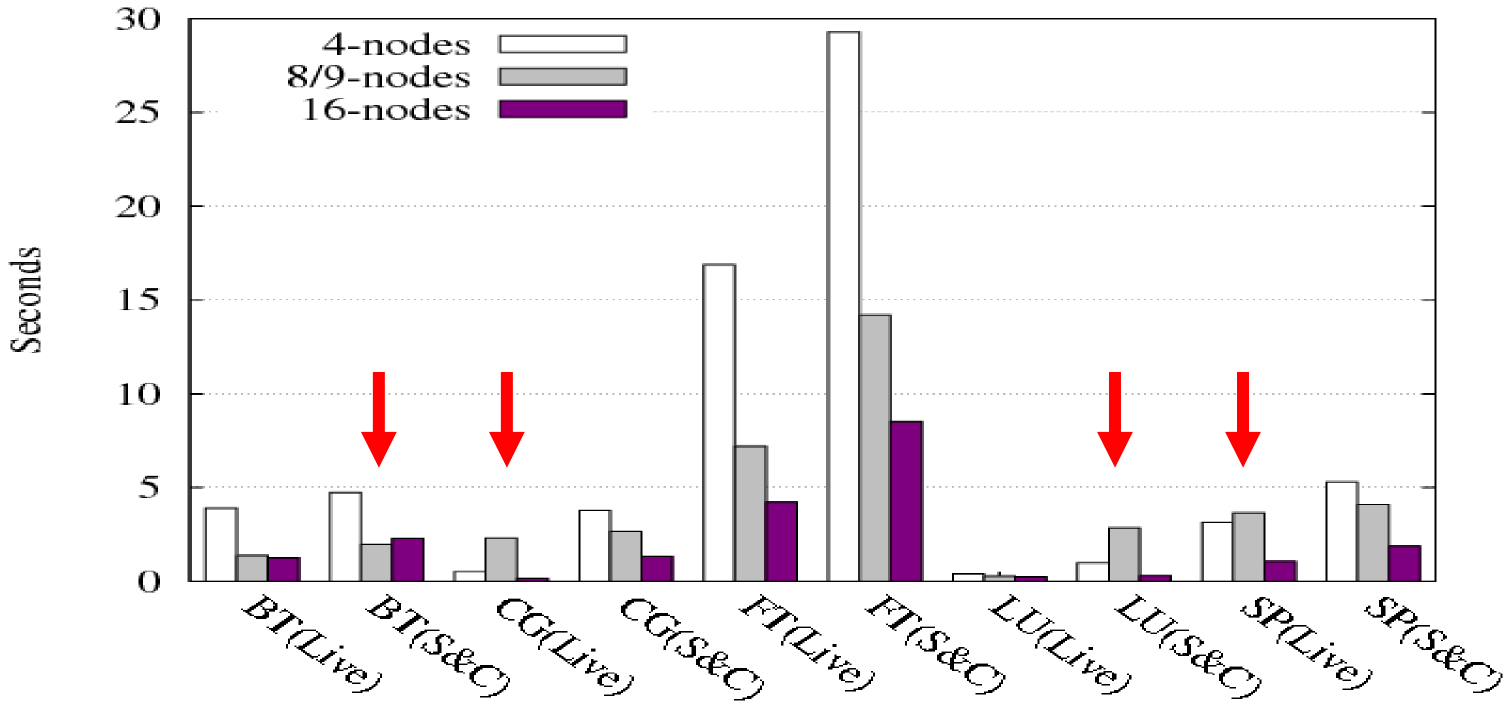
Problem Scaling



Problem Scaling: Overhead on 16 Nodes (S&C = Frozen)

- BT/FT/SP: Overhead increases with problem size
- CG/LU: small downtime subsumed by variance of job run time

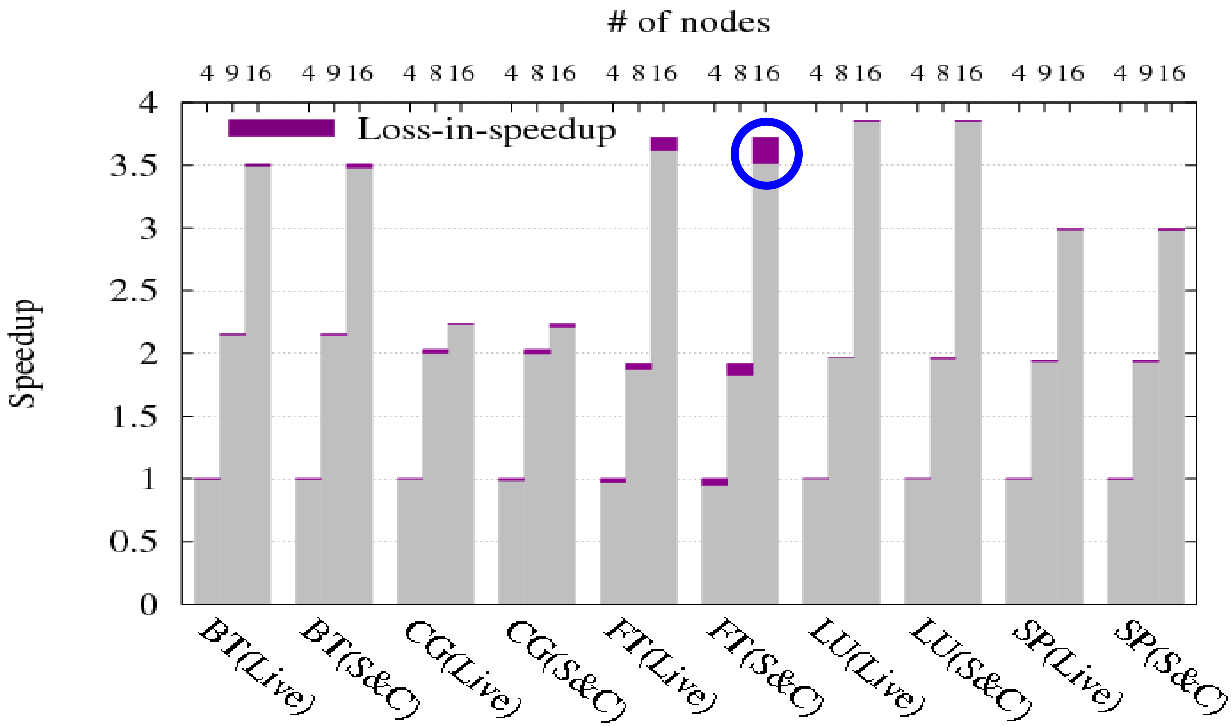
Task Scaling

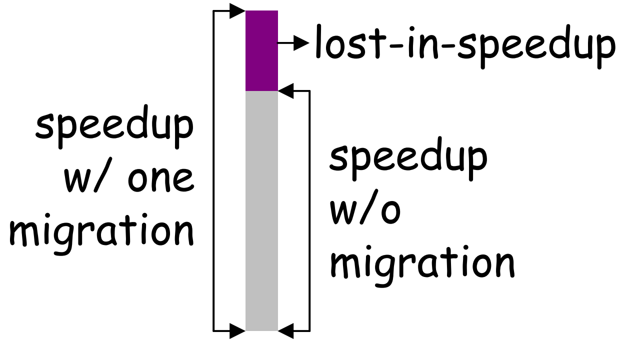


Task Scaling: Overhead of NPB Class C (S&C = Frozen)

- Most cases: Overhead decreases with task size
- No trends: relatively minor downtime subsumed by job variance

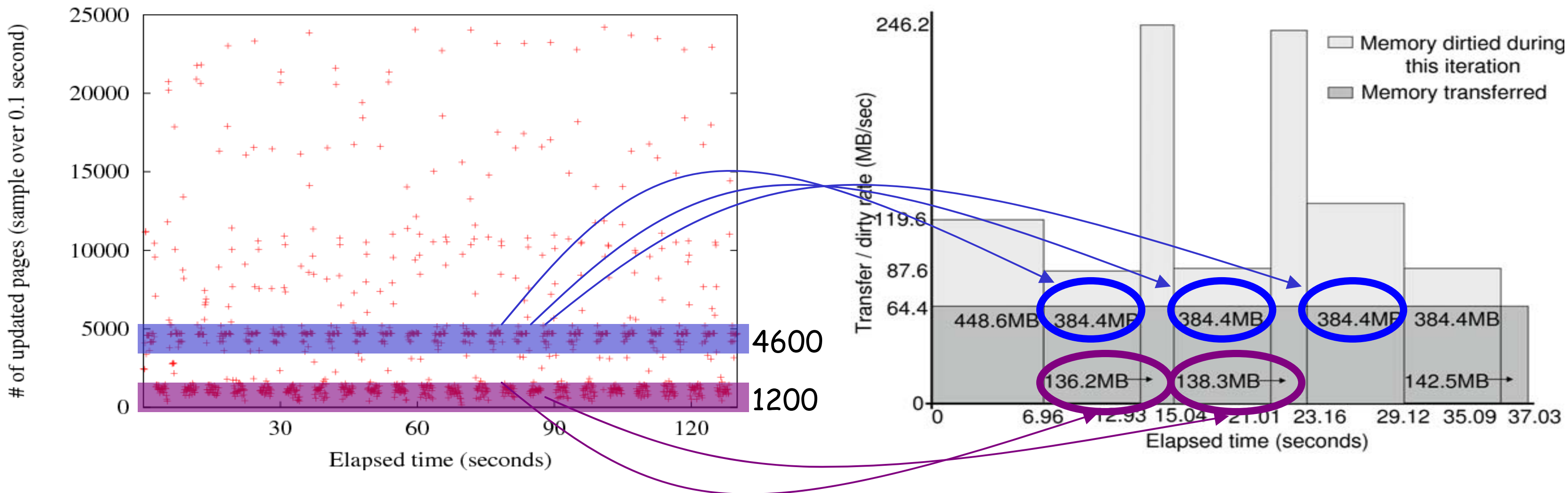
Speedup





- Normalized speedup to 4 nodes for NPB Class C
- FT 0.21 lost-in-speedup: relatively large overhead (8.5 sec) vs. short run time (150 sec)
- Limit of migration overhead: proportionate to memory footprint, limited by system hardware

Page Access Pattern & Iterative Migration



Page access pattern of FT

Iterative live migration of FT

- Page write patterns are in accord with aggregate amount of transferred memory
- FT: 138/384MB -> 1200/4600 pages/.1 second

Process-level vs. Xen Virtualization Migration

- Xen virtualization live migration *[A. B. Nagarajan & F. Mueller ICS '07]*
- NPB BT/CG/LU/SP: common benchmarks measured with both solutions on the same hardware
- Xen virtualization solution: 14-24 seconds for live migration, 13-14 seconds for frozen migration
  - Including a 13 seconds minimum overhead to transfer the entire memory image of the inactive guest VM (rather than transferring a subset of the OS image) for the transparency
  - 13-24 seconds of prior warning to successfully trigger live process migration
- Our solution: 2.6-6.5 seconds for live migration, 1-1.9 seconds for frozen migration
  - 1-6.5 seconds of prior warning (reduce false alarm rate)

Conclusion and Future Work

- Design generic for any MPI implementation / process C/R
- Implemented over LAM-MPI w/ BLCR
- Cut the number of chkpts in half when 70% faults handled proactively
- Low overhead: Live: 0.08-2.98% Frozen: 0.09-6%
  - No job requeue overhead/ Less staging cost/ No LAM Reboot

- Future work
  - Heuristic algorithm for tradeoff between live & frozen migrations
  - Back migration upon node recovery
  - Measure how proactive FT complements reactive FT

Related Work

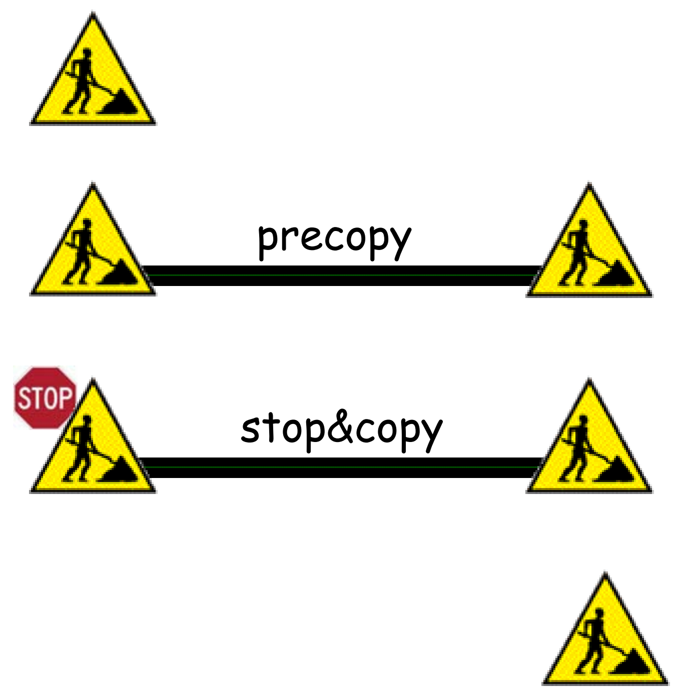
- Transparent C/R
  - LAM/MPI w/ BLCR *[S.Sankaran et.al LACSI '03]*
    - Process Migration: scan & update checkpoint files [Cao et. Al, ICPADS, 05]
      → still requires restart of entire job
  - Log based (Log msg + temporal ordering): MPICH-V *[SC 2002]*
- Non-transparent C/R: Explicit invocation of checkpoint routines
    - LA-MPI *[IPDPS 2004]* / FT-MPI *[EuroPVM-MPI 2000]*
- Failure prediction: Predictive management *[Gujrati et. Al, ICPP07] [Gu et. Al, ICDCS08] [Sahoo et. Al, KDD03]*
- Fault model: Evaluation of FT policies *[Tikotekar et. Al, Cluster07]*
- Process migration: MPI-Mitten *[CCGrid06]*
- Proactive FT: Charm++ *[Chakravorty et. Al, HiPC06], etc.*

Questions?

Thank you!

This work was supported in part by:

- NSF Grants: CCR-0237570, CNS-0410203, CCF-0429653
- Office of Advanced Scientific Computing Research
- DOE GRANT: DE-FG02-05ER25664
- DOE GRANT: DE-FG02-08ER25837
- DOE Contract: DE-AC05-00OR22725



Project websites:

MOLAR: http://forge-fre.ornl.gov/molar/
RAS: http://www.fastos.org/ras/