Buffered Co-scheduling: A New Methodology for Multitasking Parallel Jobs on Distributed Systems

Fabrizio Petrini and Wu-chun Feng
{fabrizio,feng}@lanl.gov

Los Alamos National Laboratory

LA-UR-00-0894

W. Feng, RADIANT
Outline

• Motivation & Background
• Related Work
• Buffered Coscheduling
  – Communication Buffering
  – Strobing
  – Non-Blocking Communication
• Workload and Simulation Model
• Experimental Results
• Conclusion

W. Feng, RADIANT
Motivation

• **Goals**
  – High system throughput.
  – Fast response time for interactive & high-priority jobs.
  – Efficient resource utilization.

• **Current Commercial Solution**
  – Third-party software: LSF.
  – Problem: Efficient space sharing (> 90%) but no time sharing.
    • CPU utilization ≈ 55% (typical)
    • Network utilization ≈ 5% (typical)

  • *Result:* High system throughput, poor response time, and inefficient resource utilization.

W. Feng, RADIANT
Space Sharing vs. Time Sharing

• Space Sharing (LSF)
  - Parallel Program 1 (8 parallel jobs)
  - Parallel Program 2 (4 parallel jobs)
  - Parallel Program 3 (2 parallel jobs)
  - Parallel Program 4 (2 parallel jobs)

• Time Sharing
  - Parallel Program 1 (8 parallel jobs)
  - Parallel Program 2 (4 parallel jobs)
  - Parallel Program 3 (2 parallel jobs)
  - Parallel Program 4 (2 parallel jobs)

W. Feng, RADIANT
Motivation

• Goals
  – High system throughput.
  – Fast response time for interactive & high-priority jobs.
  – Efficient resource utilization.
• Current Commercial Solution
  – Third-party software: LSF.
  – Problem: Efficient space sharing (>90%) but no time sharing.
    • CPU utilization ≈ 55% (typical)
    • Network utilization ≈ 5% (typical)
    • Result: High system throughput, poor response time, and inefficient resource utilization.

W. Feng, RADIANT
ASCI Sweep 3D, 1 Billion-Cell Problem

Communication
Computation
Related Work

• Time Sharing (a.k.a. Coscheduling or Gang Scheduling)
  – *Explicit Coscheduling*
    • Precomputed job schedule that requires simultaneous context-switching across all processors.
    • System throughput (+), response time (---), resource utilization (+)
  – *Local Scheduling*
    • Each processor independently schedules its processes.
    • System throughput (---), response time (-), resource utilization (---)
  – *Implicit or Dynamic Coscheduling* (UC-Berkeley & MIT)
    • Each processor makes decisions that dynamically coordinate scheduling actions of cooperating processes across processors.
    • System throughput (-), response time (+), resource utilization (+)

W. Feng, RADIANT
Buffered Coscheduling
A New Methodology to Multitask Parallel Programs

• Components
  – Communication Buffering
  – Strobing for Information Exchange & Synchronization
  – Non-Blocking Communication

• Features
  – Push resource management from MPI down into the operating system.
    • Allows computation, communication, and I/O which arise from a set of parallel programs to be overlapped with the computations in those programs.
  – Amortize communication overhead.
  – Provide a framework for fault tolerance.
  – Decrease software development time for parallel programs.
  – Enable accurate performance analysis.

W. Feng, RADIANT
Communication Buffering

- **Goal**
  - Amortize communication overhead over a set of messages.

- **Approach**
  - Buffer communication over a given time slice.
  - Perform communication at the end of a time slice.
  - (Future Note: Jobs can be multitasked at any time.)

- **Result**
  - Up to 87% decrease in execution time for parallel programs.
  - Speedup of up to 7.5 times.

W. Feng, RADIANT
Strobing

- **Goal**
  - Provide a mechanism for a total exchange of control information at the end of each time slice (to fill in communication “holes”).

- **Approach**

- **Result:** Predictable (“bounded”) barrier synchronization.

- **Implication**
  - Enables global flow-control strategy as every processor knows how much information to expect & who to expect it from.

W. Feng, RADIANT
Barrier Synchronization Times

W. Feng, RADIANT
Non-Blocking Communication

W. Feng, RADIANT
Conclusion

• Future Work
  – Benchmark and compare with commercial solutions (i.e., LSF & LoadLeveler) as well as implicit/dynamic coscheduling.
  – Leverage lessons learned to implement a distributed OS to support the buffered coscheduling methodology.

• Publications