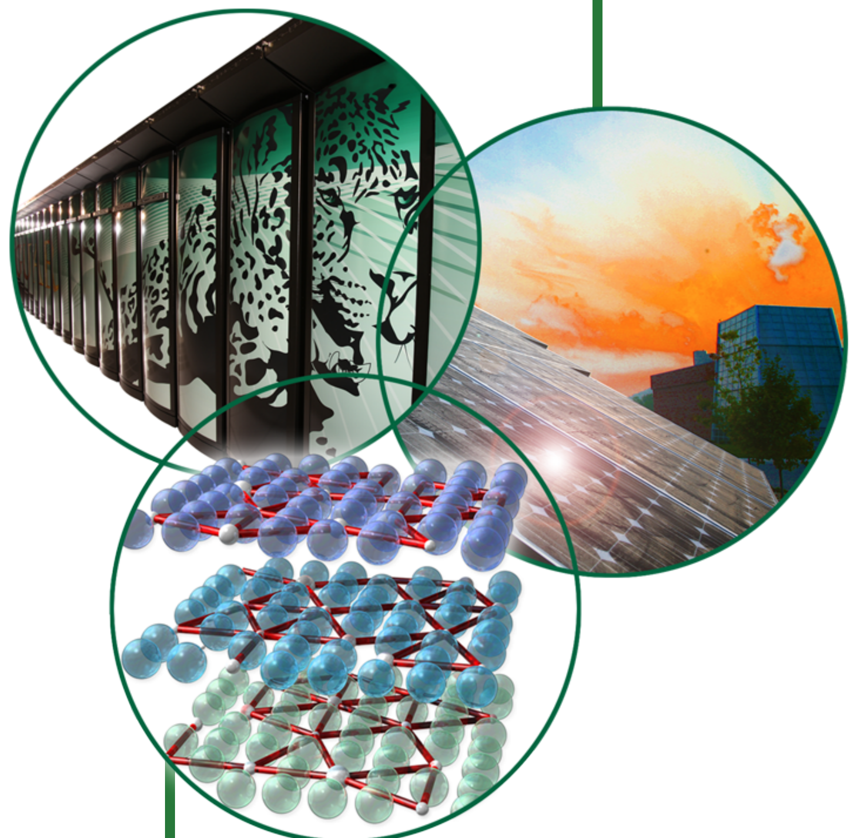
Linux Kernel Co-Scheduling For Bulk Synchronous Parallel Applications



ROSS 2011
Tucson, AZ

Terry Jones

Oak Ridge National Laboratory

Outline

- Motivation
- Approach & Research
  - Design Attributes
  - Achieving Portable Performance
  - Measurements
- Conclusion & Acknowledgements

We're Experiencing an Architectural Renaissance

ROSS 2011

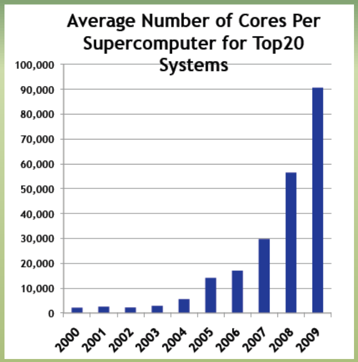
- Factors To Change
  - Moore's Law -- Number of transistors per IC double every 24 months
  - No Power Headroom -- Clock speed will not increase (and may decrease) because of Power

*Power α Voltage^2 * Frequency*

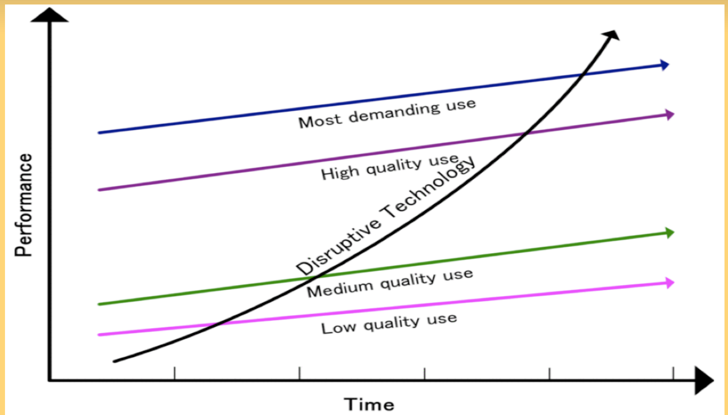
Power α Frequency

Power α Voltage^3

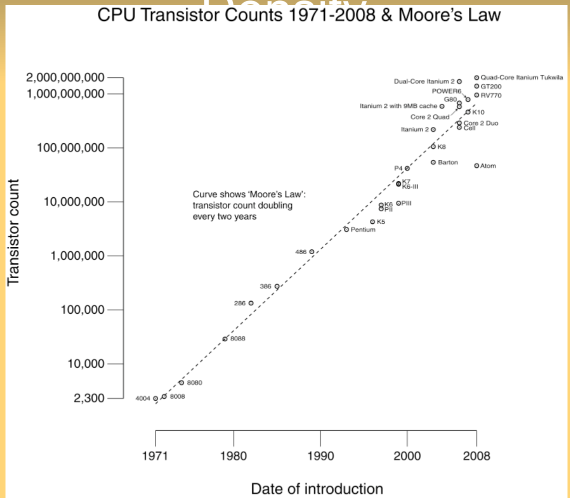
Increased Core Counts



Disruptive Technologies



Increased Transistor Density



A Key Component of the Colony Project
Adaptive System Software For Improved Resiliency and Performance

ROSS 2011

Collaborators

Terry Jones, Project PI | Laxmikant Kalé, UIUC PI | José Moreira, IBM PI

Objectives

- Provide technology to make portable scalability a reality.
- Remove the prohibitive cost of full POSIX APIs and full-featured operating systems.
- Enable easier leadership-class level scaling for domain scientists through removing key system software barriers.

Challenges

- Computational work often includes large amounts of state which places additional demands on successful work migration schemes.
- For widespread acceptance from the Linux community, the effort to validate and incorporate HPC originated advancements into the Linux kernel must be minimized.

Approach

- Automatic and adaptive load-balancing plus fault tolerance.
- High performance peer-to-peer and overlay infrastructure.
- Address issues with Linux to provide the familiarity and performance needed by domain scientists.

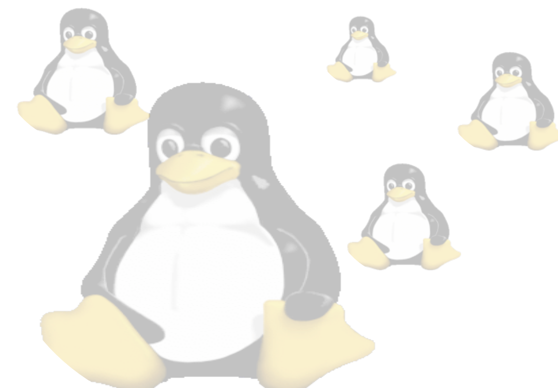
Impact

- Full-featured environments allow for a full range of programming development tools including debuggers, memory tools, and system monitoring tools that depend on separate threads or other POSIX API.
- Automatic load balancing helps correct problems associated with long running dynamic simulations.
- Coordinated scheduling removes the negative impact of OS jitter from full-featured system software.

Motivation – App Complexity

Don't Limit Development Environment

- Linux
  - -> Familiar
  - -> Open Source
  - -> Support for common system calls



- Support for daemons & threading packages
  - -> Debugging strategies
  - -> Asynchronous strategies
- Support for administrative monitoring
- OS Scalability
  - -> Eliminate OS Scalability Issues Through Parallel Aware Scheduling

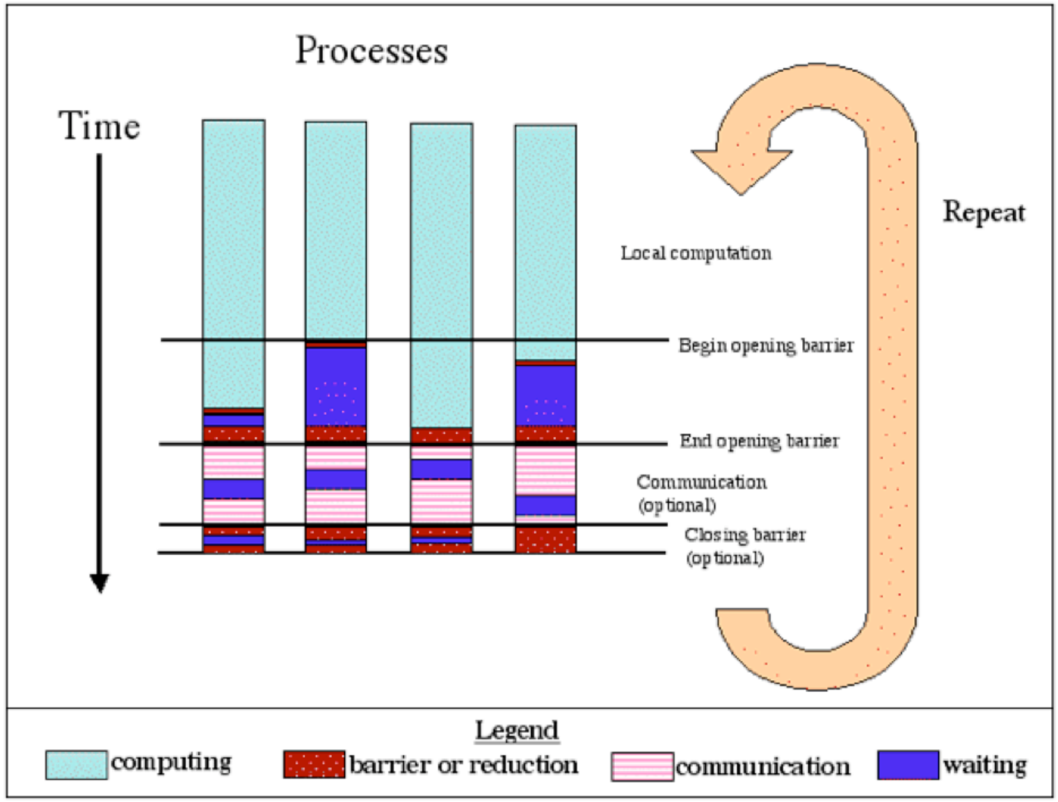
The Need For Coordinated Scheduling

ROSS 2011

Bulk

Synchronous

Programs



The Need For Coordinated Scheduling

ROSS 2011

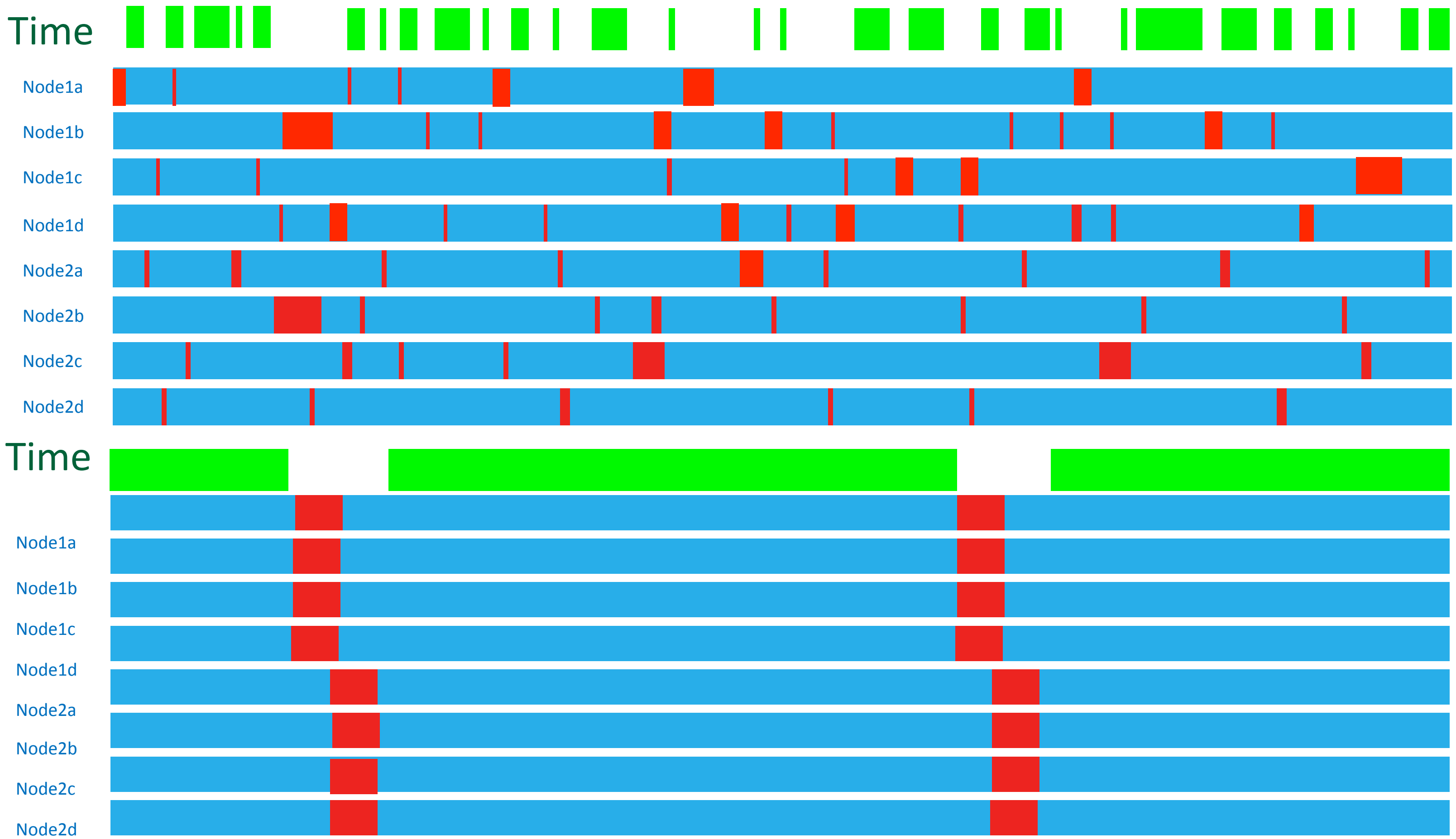
- Permit Full Linux Functionality
- Eliminate Problematic OS Noise
- Metaphor: Cars and Coordinated Traffic Lights

What About …

ROSS 2011

- Core Specialization
- Minimalist OS
- Will Apps Always Be Bulk Synchronous?
- Yeah, but it's *Linux*

HPC Colony Technology – *Coordinated Scheduling*



Goals

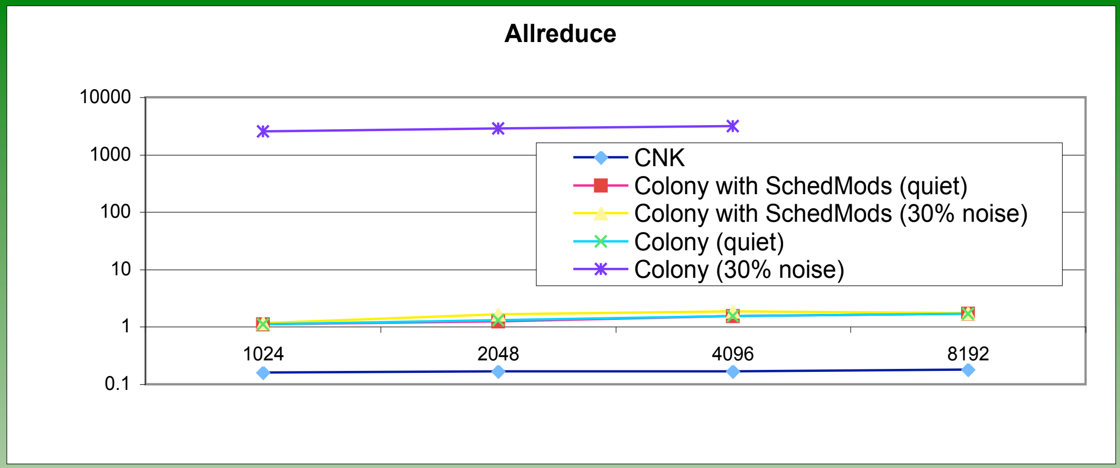
ROSS 2011

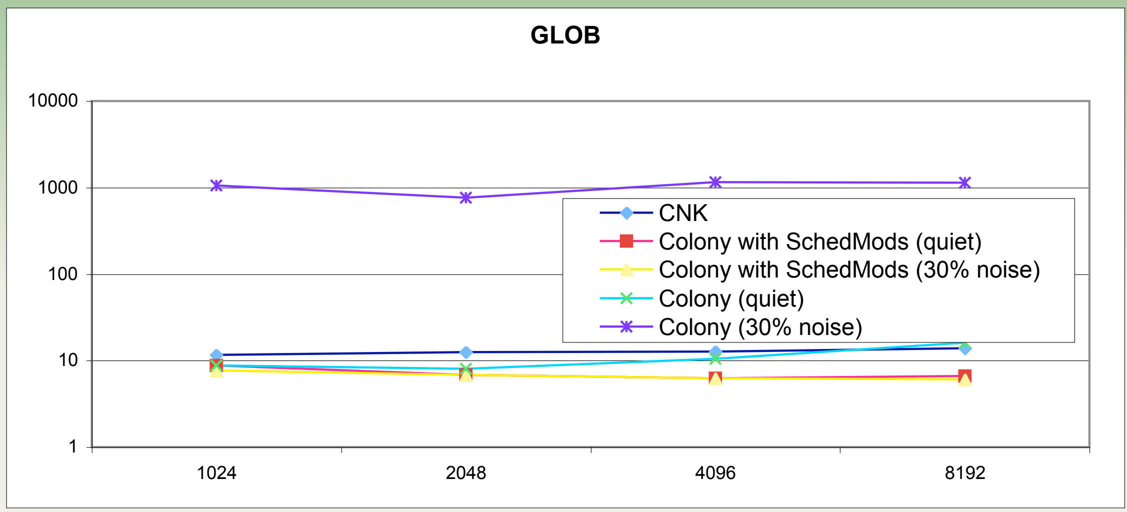
- Portable Performance
- Make OS Noise a non-issue for bulk-synchronous codes
- Permit sysadmin best practices

Proof of Concept – Blue Gene / L

ROSS 2011

Core Counts (cont.) Scaling with Noise (Noise level @ serial task takes 30% longer)

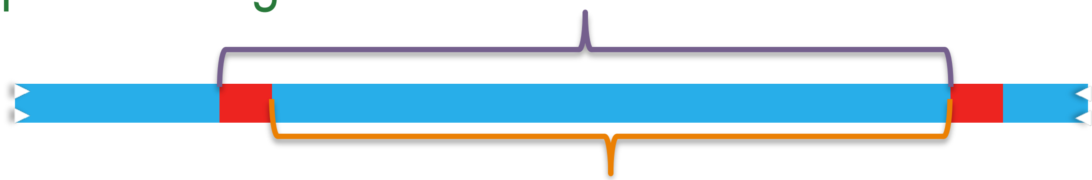




ROSS 2011

Approach

- Introduces two new process flags & two new tunables
  - total time of epoch

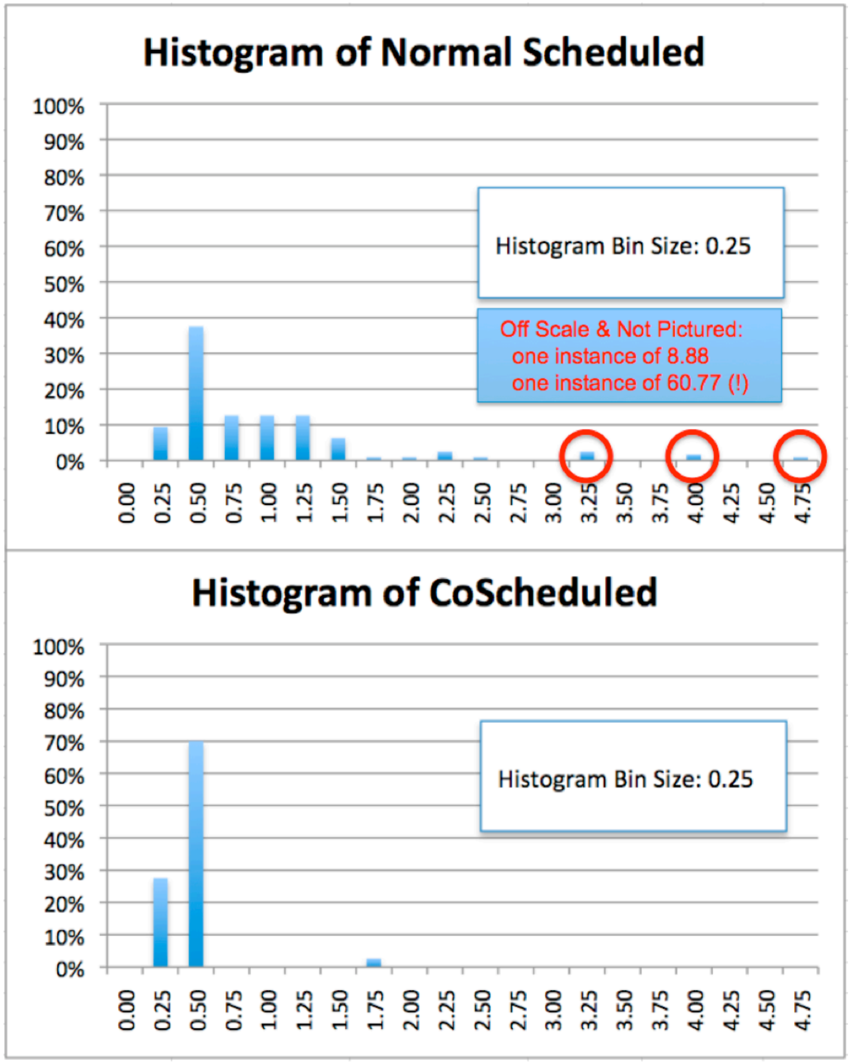


  - percentage to parallel app (percentage of blue from co-schedule figure)
- Dynamically turned on or off with new system call
- Tunables are adjusted through use of a second new system call

Salient Features

- Utilizes a new clock synchronization scheme
- Uses existing fair round-robin scheduler for both epochs
- Permits needed flexibility for time-out based and/or latency sensitive apps

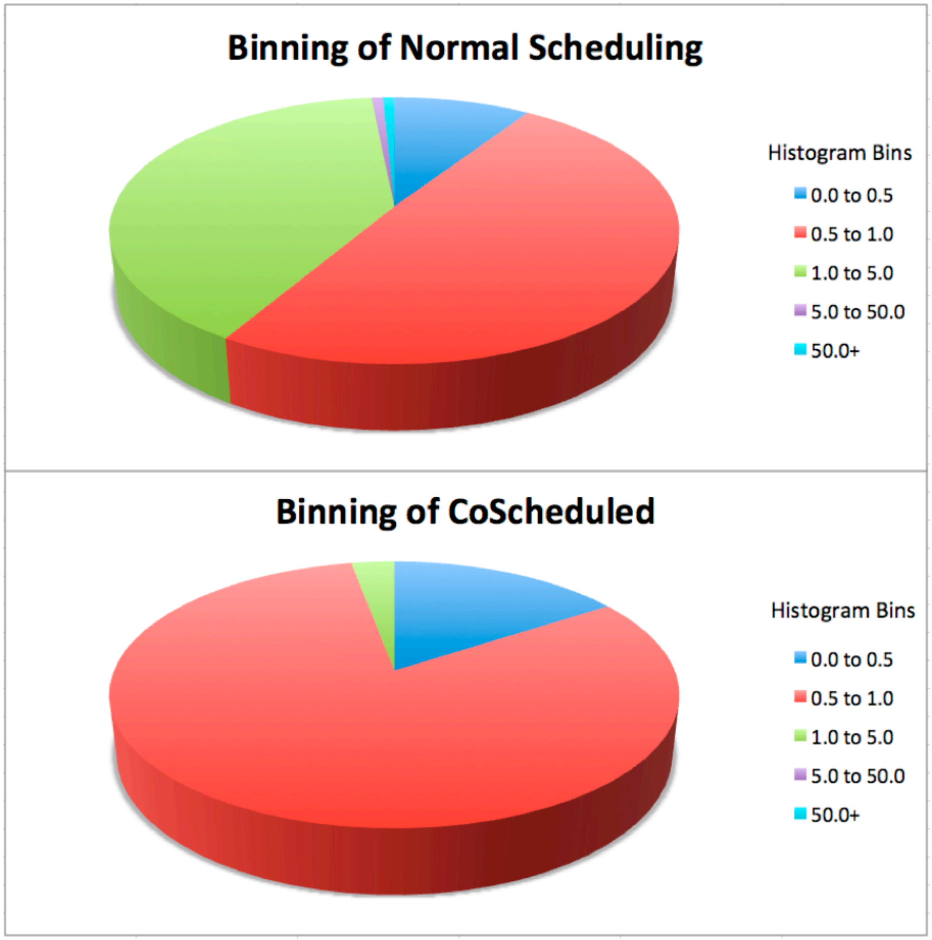
Results



Histogram of Normal Scheduled
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
Histogram Bin Size: 0.25
Off Scale & Not Pictured:
one instance of 8.88
one instance of 60.77 (!)
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
3.25
3.50
3.75
4.00
4.25
4.50
4.75
Histogram of CoScheduled
Histogram Bin Size: 0.25

Results

ROSS 2011



…and in conclusion…

- For Further Info
  - contact: Terry Jones trj@ornl.gov
  - http://www.hpc-colony.org
  - http://charm.cs.uiuc.edu
- Partnerships and Acknowledgements
  - Synchronized Clock work done by Terry Jones and Gregory Koenig
  - DOE Office of Science – major funding provided by FastOS 2
  - Colony Team

Thank You

Any Questions?

ROSS 2011

Extra Viewgraphs

Improved Clock Synchronization Algorithms

Sponsor: DOE ASCR
FWP ERKJT17

ROSS 2011

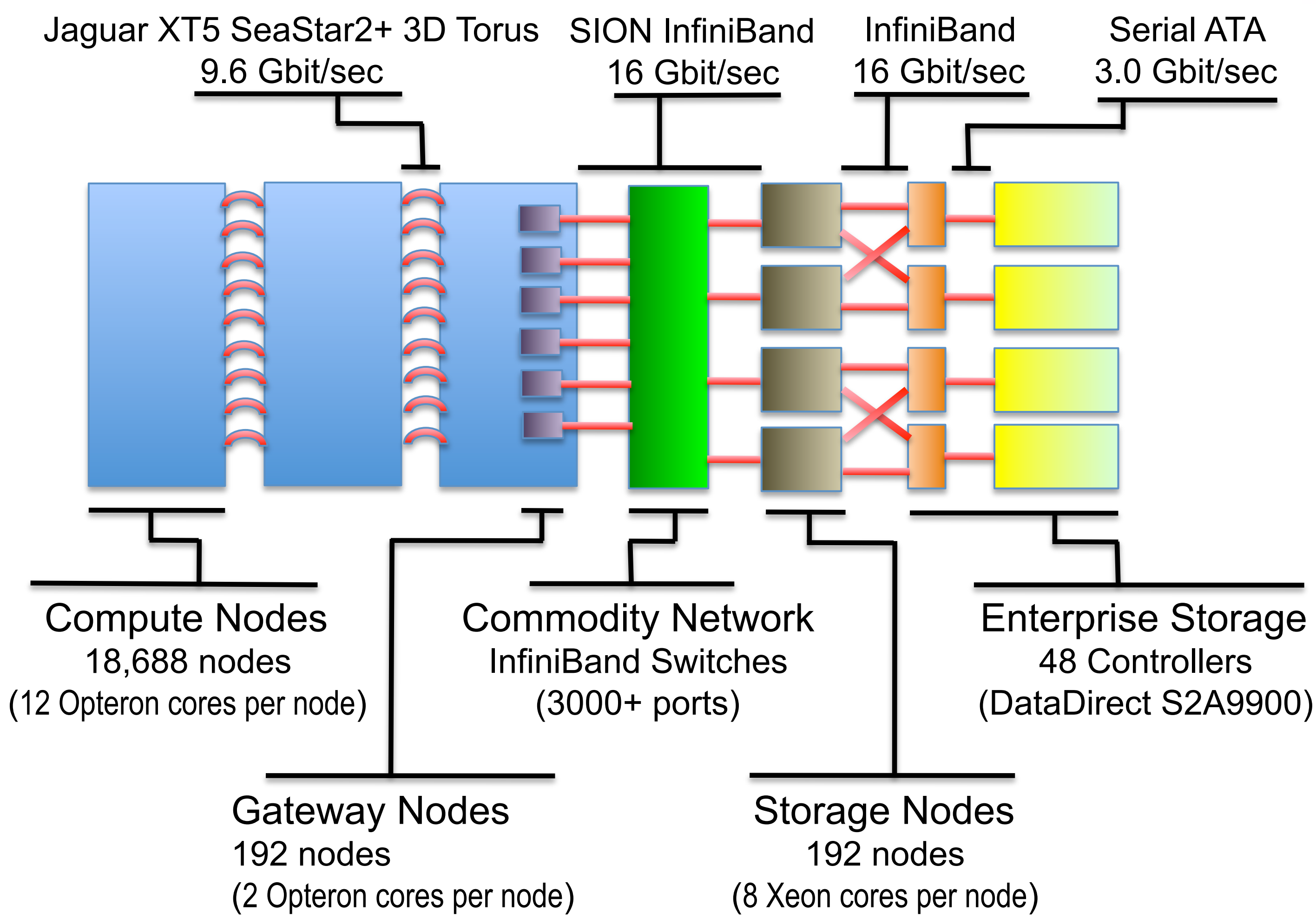
- Achievement

  Developed a new clock synchronization algorithm. The new algorithm is a high precision design suitable for large leadership-class machines like Jaguar. Unlike most high-precision algorithms which reach their precision in a post-mortem analysis after the application has completed, the new ORNL developed algorithm rapidly provides precise results during runtime.

- Relevance
  - To the Sponsor;
    - Makes more effective use of OLCF and ALCF systems possible.
  - To the Laboratory, Directorate, and Division Missions; and
    - Demonstrates capabilities in critical system software for leadership-class machines.
  - To the Computer Science Research Community.
    - High precision global synchronized clock of growing interest to system software needs including parallel analysis tools, file systems, and coordination strategies.
    - Demonstrates techniques for high-precision coupled with guaranteed answer at runtime.

Test Setup

ROSS 2011



Test Setup (continued)

ROSS 2011

Jaguar is a Cray XT system consisting of XT4 and XT5 partitions

Jaguar	XT4	XT5	Total
Nodes per blade	4		
CPUs per node[1]	1	2	
Cores per node	4	12	
Compute nodes[2]	7,832	18,688	
AMD Opteron cores	31,328	224,256	255,584
Memory per CPU	8 GB/CPU		
System Memory	~61.2 TB	~292 TB	~353.2 TB
Disk Bandwidth	~44 GB/s	~240 GB/s	~284 GB/s
Disk Space	~750 TB	~10,000 TB	~10,750 TB
Interconnect Bandwidth	~157 TB/s	~374 TB/s	~532 TB/s
Floor Space	1400 $feet^2$	4400 $feet^2$	5800 $feet^2$
Ideal Performance per core[3] (4 FLOPs/cycle times $2.1*10^9$ cycles/sec)	8.4 GFLOPS	10.4 GFLOPS	
Overall Ideal Performance	~263.16 TFLOPS	~2.33 PFLOPS	~2.60 PFLOPS

ping pong latency ~5.0 μsecs