Scalable Resource Management in High-Performance Computers

18 November, 2002

Eitan Frachtenberg, Fabrizio Petrini, Juan Fernandez, and Salvador Coll

CCS-3 Modeling, Algorithms, and Informatics Group

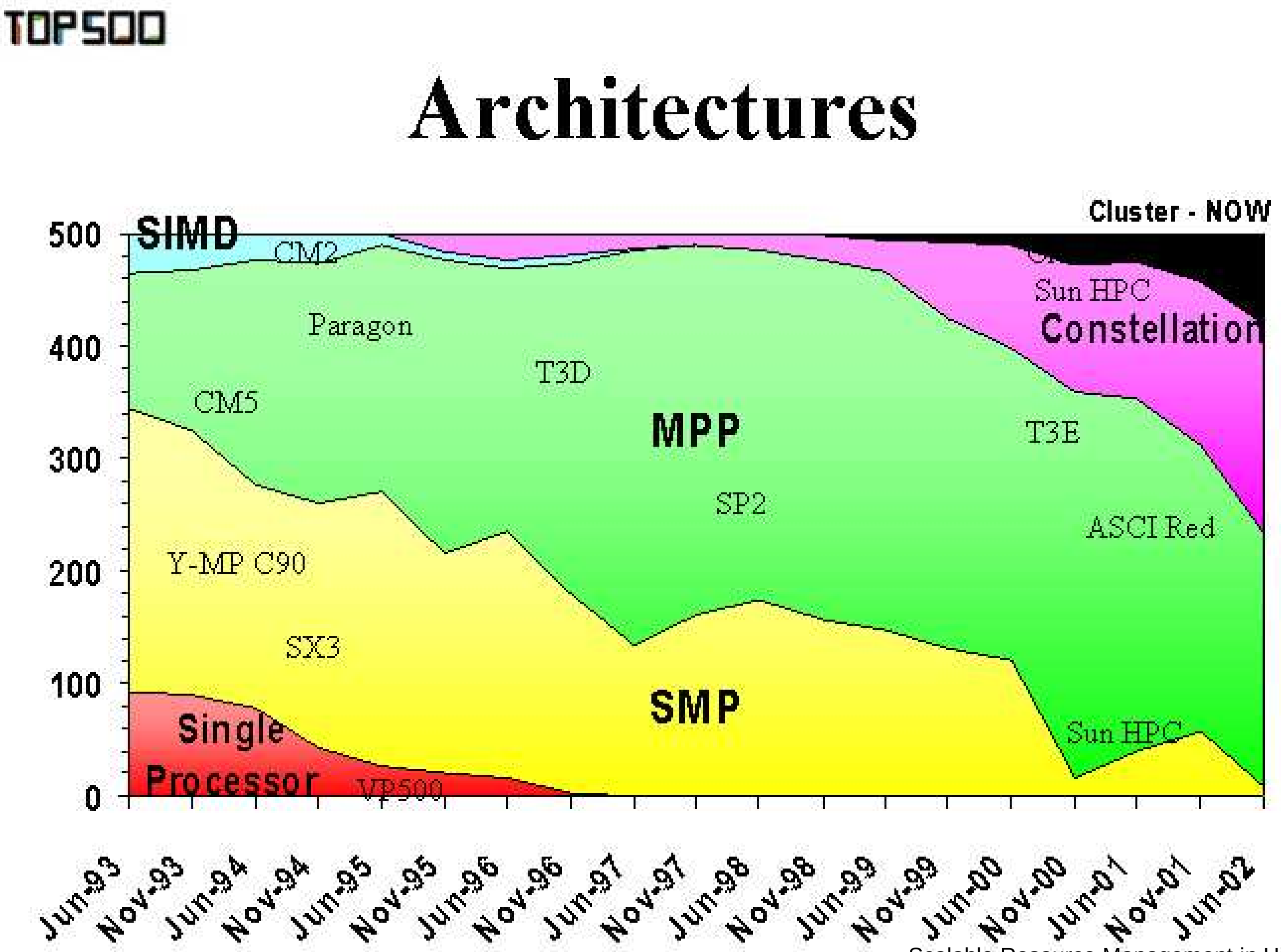
Computer and Computational Sciences (CCS) Division

Los Alamos National Laboratory

`{eitanf,fabrizio,juanf,scoll}@lanl.gov`

Cluster Resource Management

Clusters and other loosely-coupled systems are becoming ubiquitous and larger



Cluster Resource Management

In the desktop/workstation world:

- Job-launching time is typically very short (< second)
- Timeshared machine enables multitasking and interactivity
- Easy to use and quite reliable

Cluster Resource Management

In the desktop/workstation world:

- Job-launching time is typically very short (< second)
- Timeshared machine enables multitasking and interactivity
- Easy to use and quite reliable

In the cluster world:

- Jobs run one a time or gang-scheduled with large quanta
- Job-launching time is arbitrarily long (batch) or many seconds (gang-scheduling)
- Reliability and ease-of-use do not scale
- State-of-the-art RMs are typically implemented using Ethernet / TCP-IP, using non-scalable algorithms for control messages

The STORM Approach

Design goals:

1. Scalable, high-performance mechanisms for RM, leveraging modern interconnect capabilities
2. Support most current and future scheduling algorithms (FCFS, GS, SB, BCS, FCS, ...)
3. Platform for studying system-level fault tolerance

The STORM Approach

Design goals:

1. Scalable, high-performance mechanisms for RM, leveraging modern interconnect capabilities
2. Support most current and future scheduling algorithms (FCFS, GS, SB, BCS, FCS, ...)
3. Platform for studying system-level fault tolerance

Main differences from standard RMs:

1. Important parts of the RM run on the NIC
2. STORM uses scalable HW multicast mechanism
3. STORM uses pipelined IO-bypass protocol

STORM Layers

STORM functions	Heartbeat, file txfr, termination detection
Helper functions	Flow control, queue management

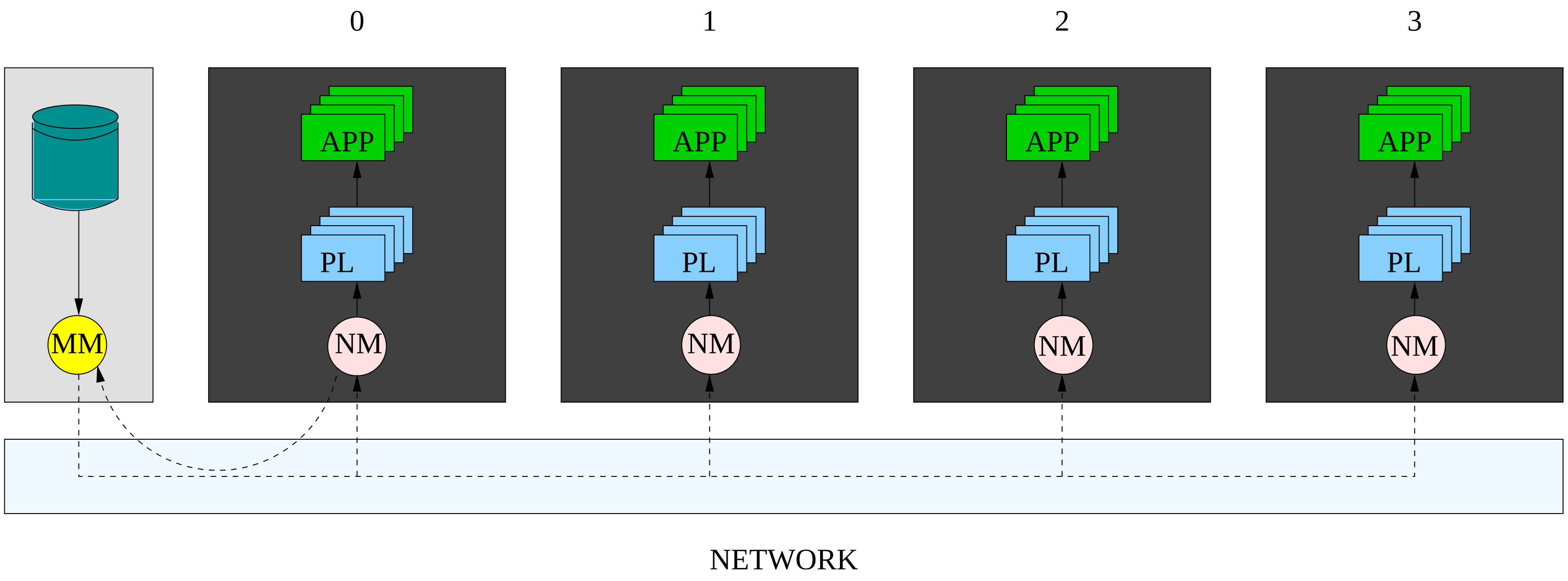
STORM mechanisms	`XFER-AND-SIGNAL`
	`TEST-EVENT`
	`COMPARE-AND-WRITE`

Network primitives	Remote DMA, signaling, event testing

STORM Mechanisms

- XFER-AND-SIGNAL & COMPARE-AND-WRITE are atomic and sequentially consistent.
- Both are collective operations that can (but don’t have to) be implemented on the NIC
- COMPARE-AND-WRITE blocks until comparison completes
- XFER-AND-SIGNAL is asynchronous: the only way to check for completion is with TEST-EVENT

STORM Architecture



- Set of layered, modular dæmons (per node and per machine)
- Lightweight and Loosely-coupled, using the communication primitives
- “Pluggable” scheduling algorithms: FCFS, GS, SB, Local, FCS...

Performance Testing

The ’Wolverine’ cluster at LANL (listed 134th at top500):

- 64-node AlphaServer ES40, running RH Linux 7.1
- 4 Alpha EV68 CPUs ($833MHz$), $8GB$ RAM per node
- Two-rail Quadrics interconnect
- Files are placed in local RAM disks to isolate RM performance

Job Launching

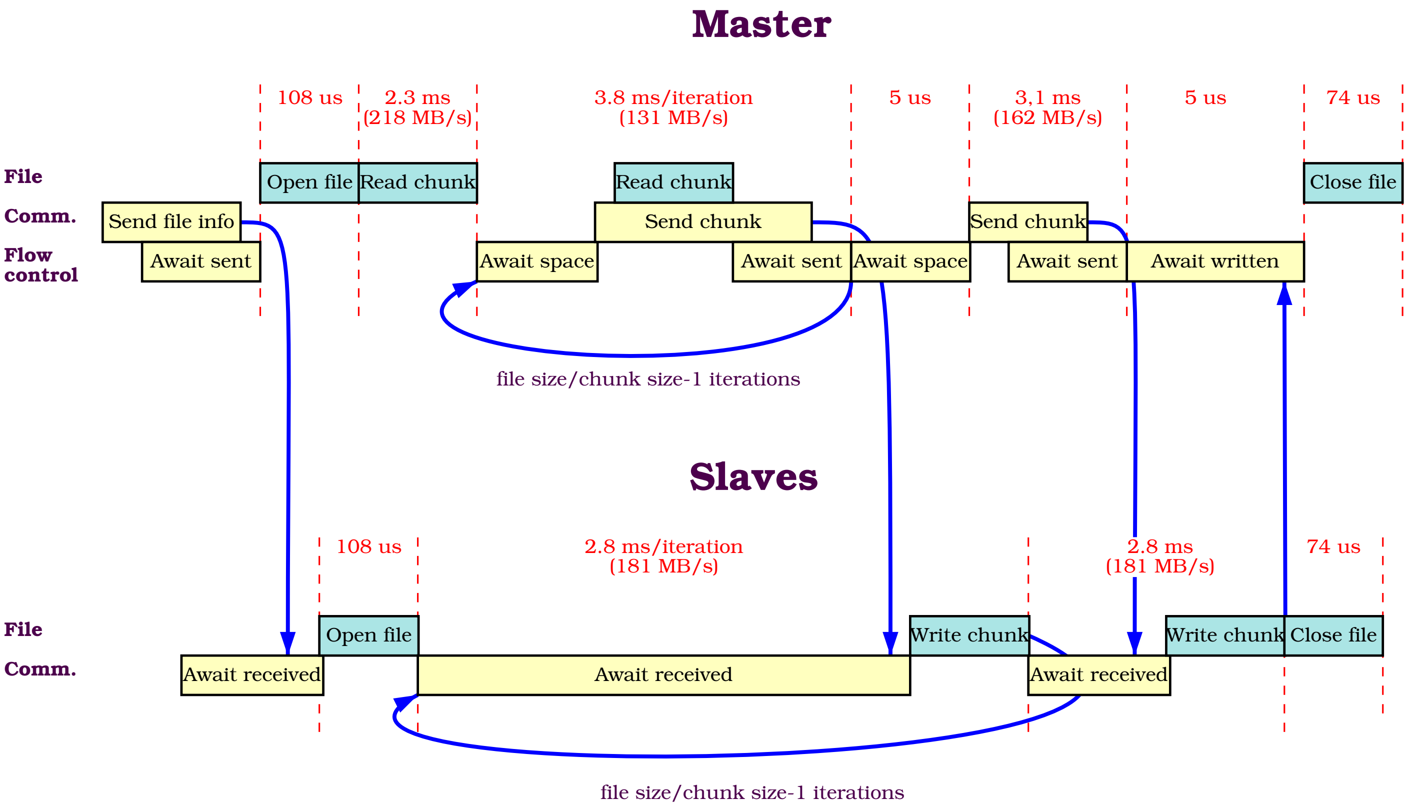
Job launching time becomes an issue when:

- Machine size grows (usual methods scale poorly)
- Debugging or running short/interactive jobs

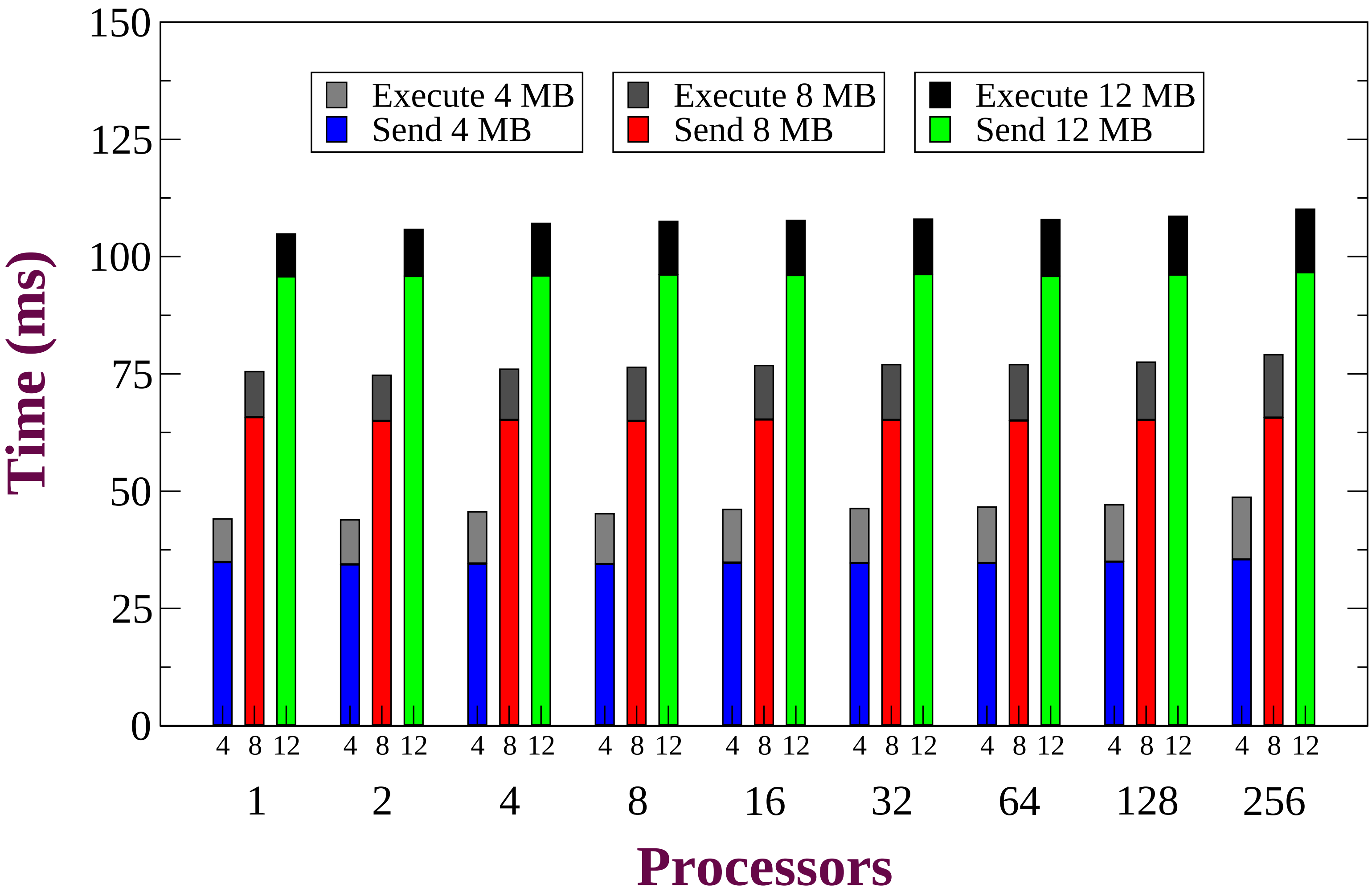
Job Launching Breakdown

- Reading binary and data files
- disseminating to compute nodes (NFS, tree, ...)
- Executing program
- Notifying job control of termination

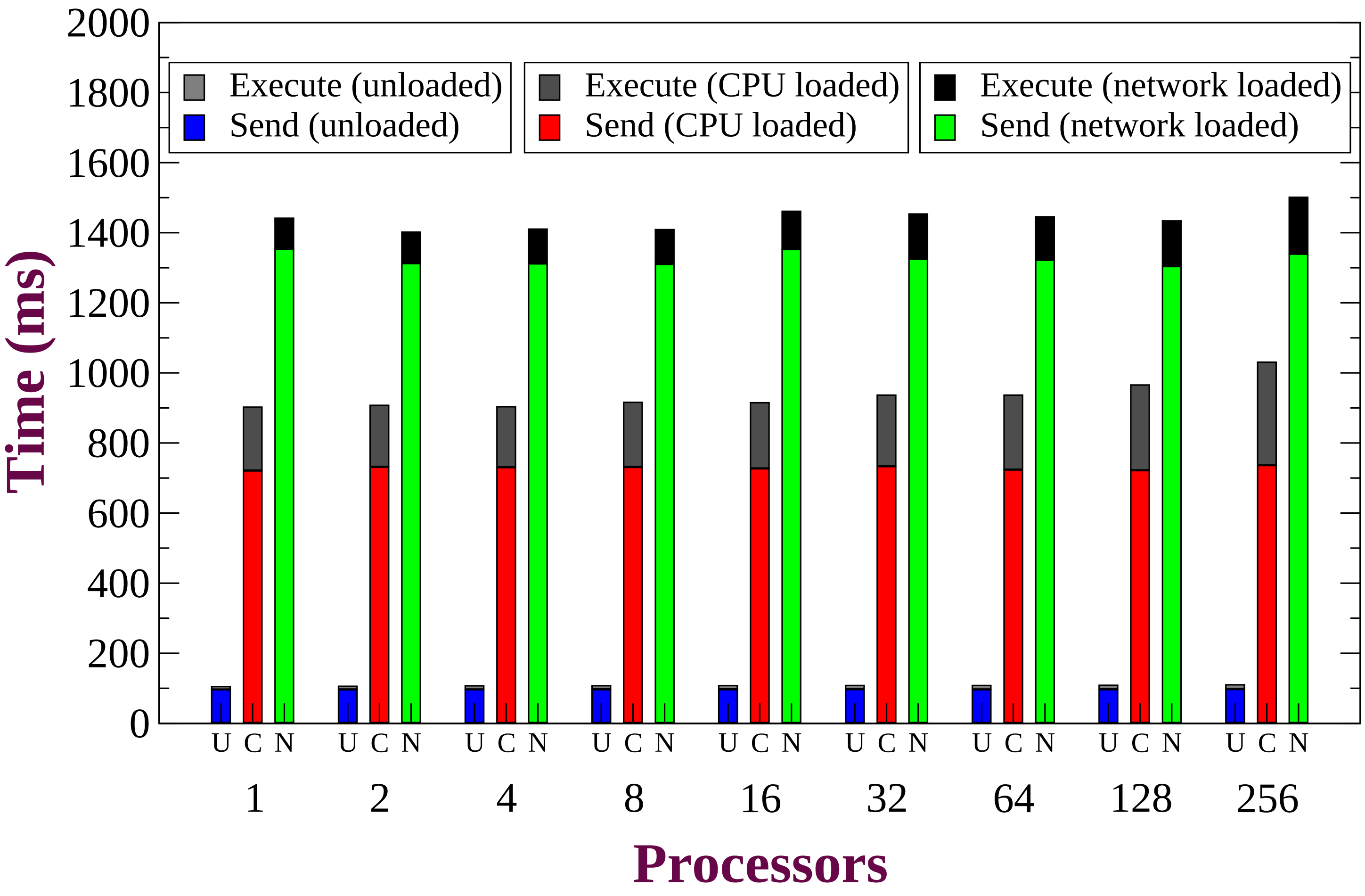
File Send Model



Job Launching Performance



Launch Times on a Loaded System

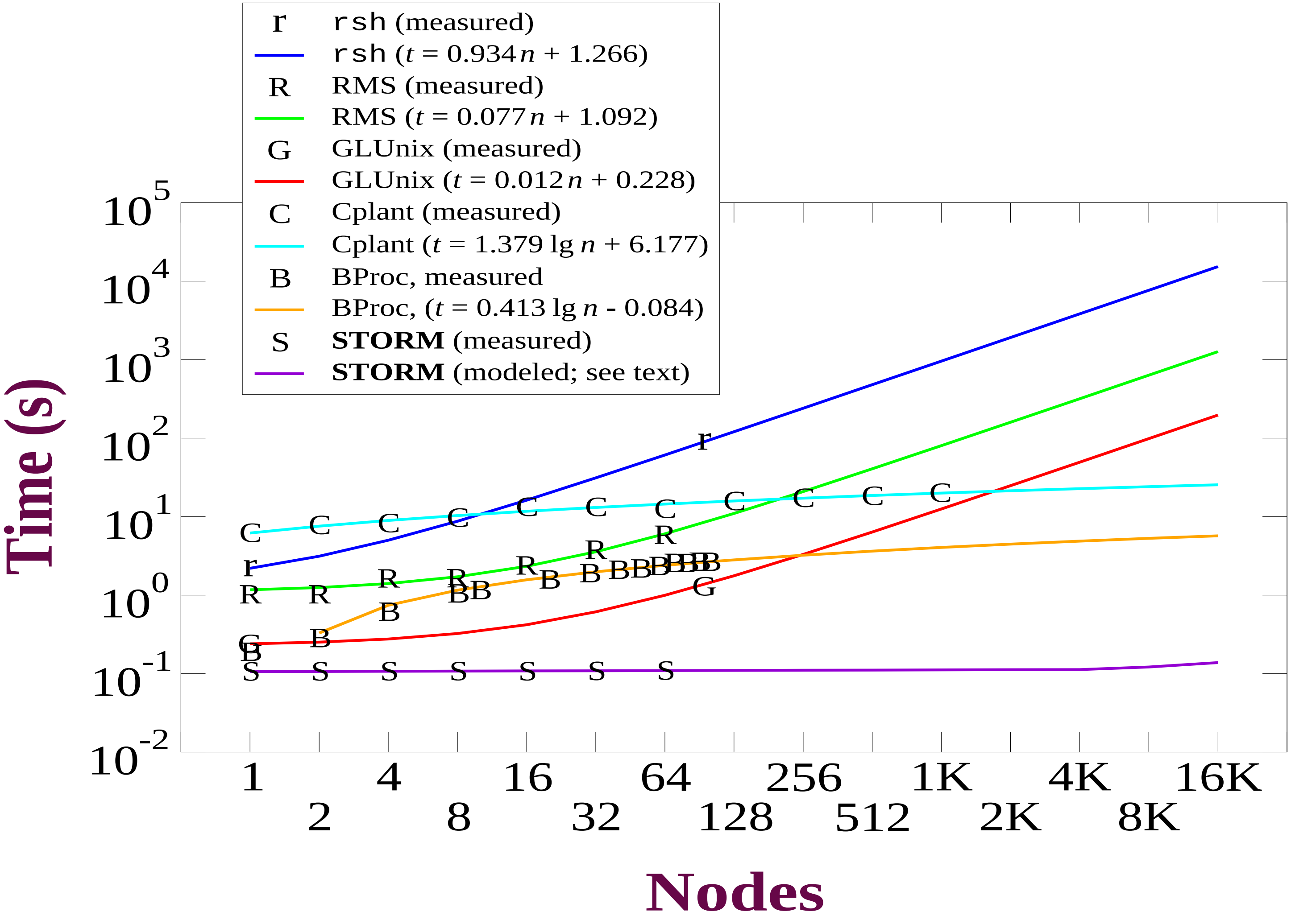


Comparison References

STORM compared to:

- GLUnix [Ghormley 98]
- BProc [Hendriks 02]
- SCore-D [Hori 98]
- Cplant [Brightwell 99]
- RMS [Frachtenberg 01]
- NFS / `rsh` (PBS)

Performance Comparison



Multiprogramming

Many supercomputers and clusters use batch scheduling, where each job receives a dedicated partition. Suspending a parallel job in the partitions and starting another can be useful for:

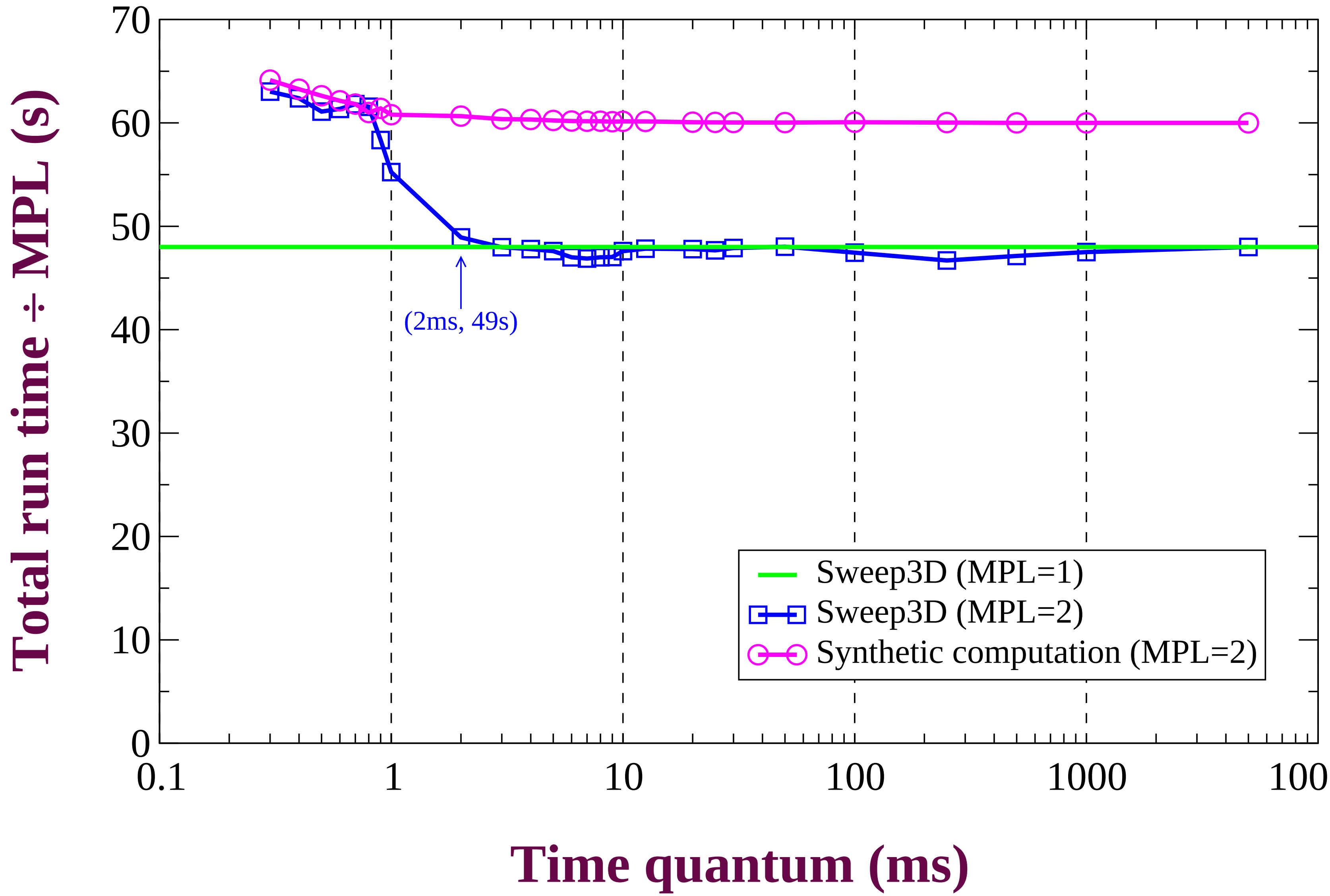
- Preempting a job for a higher-priority job and restarting later
- Running more than one interactive application (e.g. visualization applications)
- Improving system responsiveness and resource utilization through gang-scheduling

Gang Scheduling

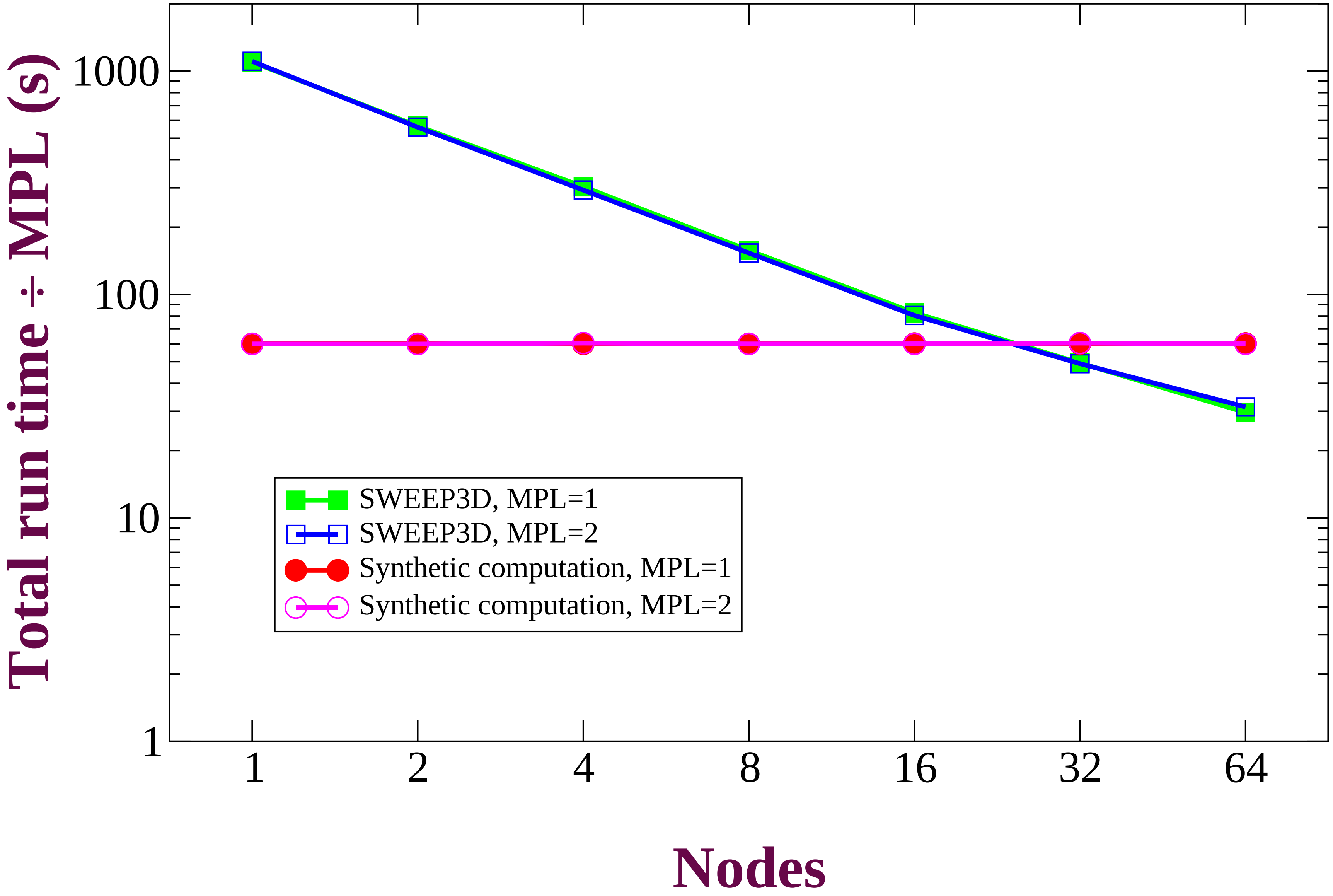
- Extends multiprogramming to parallel jobs
- Short jobs do not have to wait for long jobs to terminate
- Improves utilization by having multiple “virtual machines”
- Global context switches change jobs on the entire machine every time quantum
- Performance penalty of global context switches can be amortized by long time quanta

The combination of STORM’s mechanisms and modern HW can make the performance hit negligible.

Context Switch Overhead



Context-switch scalability



Overhead Comparison

Comparison of minimum feasible scheduling quantum with RMS and SCore-D:

RM	quantum (ms)	observed overhead
RMS	$30,000$ (15 nodes)	1.8% slowdown
SCore-D	100 (64 nodes)	2% slowdown
STORM	2 (64 nodes)	no observable slowdown

Portability Issues

Network	COMPARE-AND-WRITE (μs)	XFER-AND-SIGNAL (MB/s)
Gigabit Ethernet	$46log\,n$	Not available
Myrinet	$20log\,n$	$15n$
Infiniband	$20log\,n$	In Spec
QsNET	< 10*	$> 150n$
BlueGene/L	< 2	$700n$

* - For all sizes upto about 4,096 SMP nodes

Future and Ongoing Work

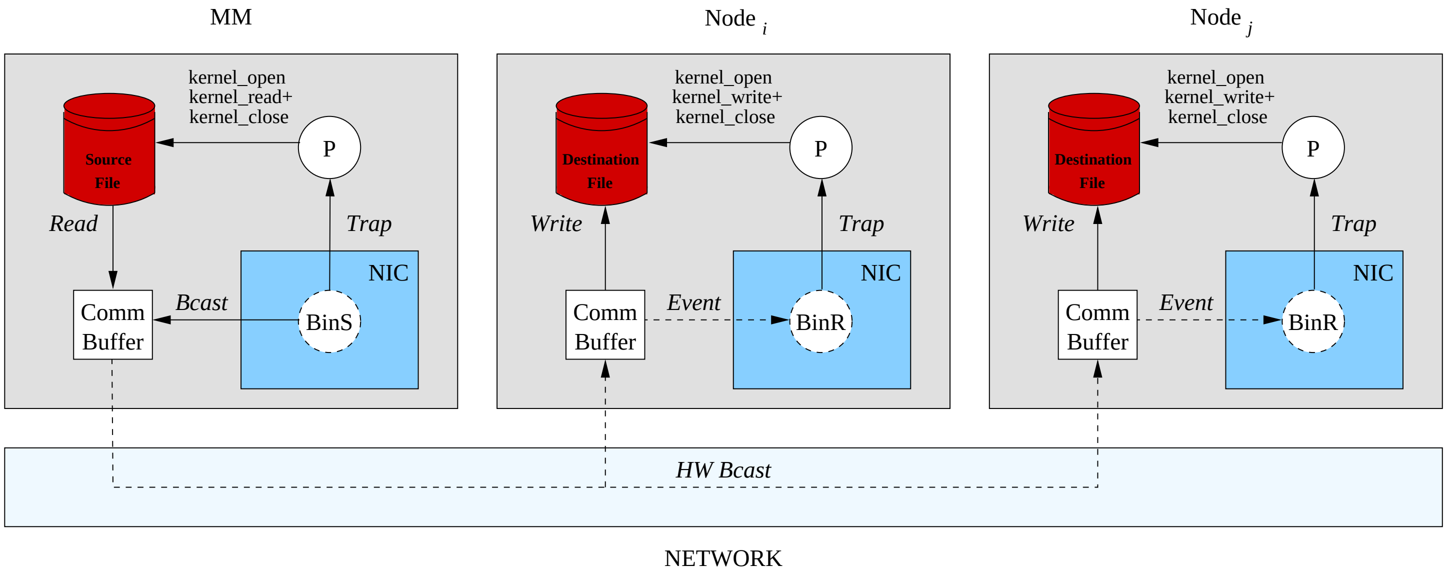
1. Load balancing jobs with different requirements
2. Improve resource utilization
3. Making systems deterministic and debuggable
4. System-level transparent fault tolerance

Conclusions

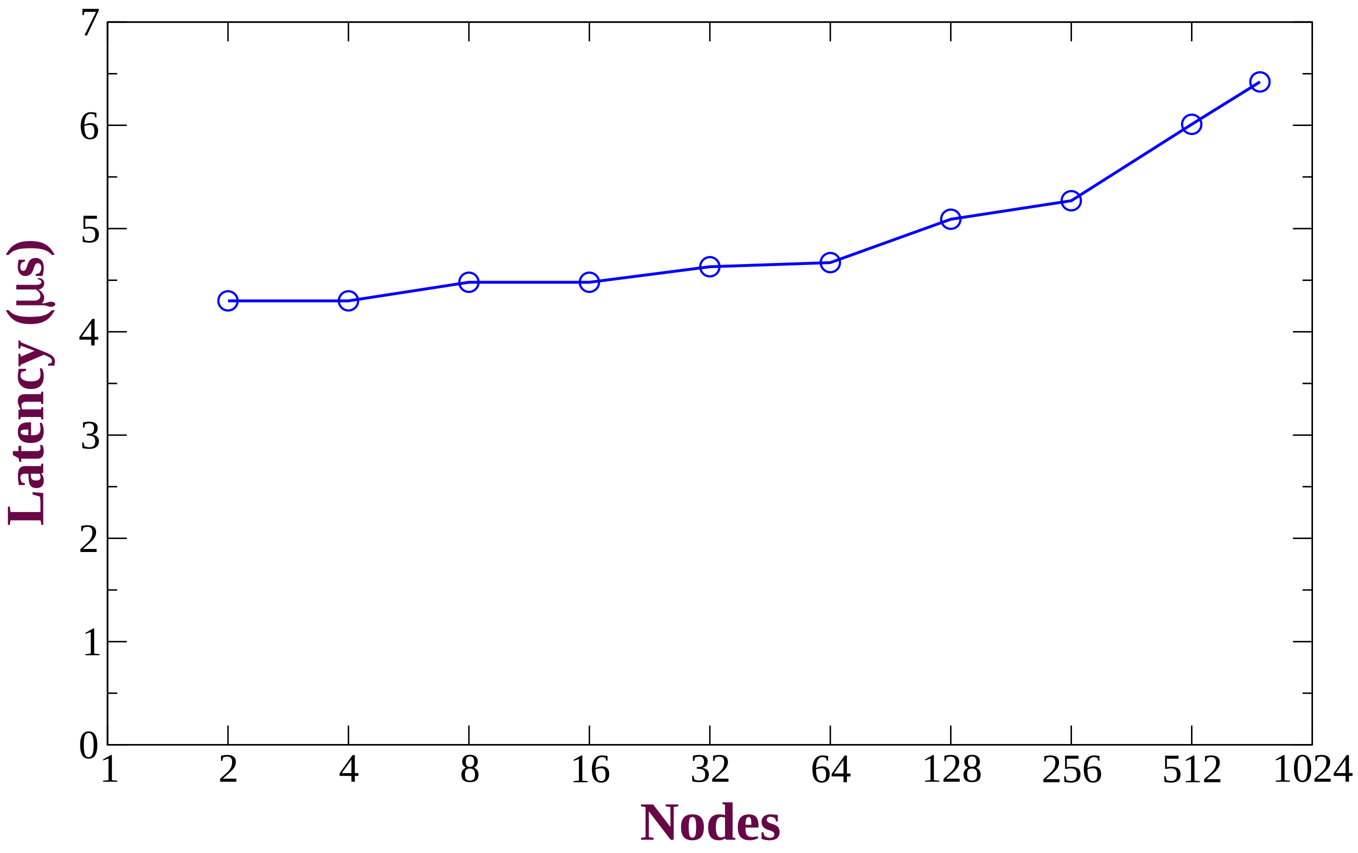
- Efficient combination of SW methods with modern interconnect HW can offer extremely scalable resource management
- Relatively simple to implement (10K-30K lines of C code)
- High-performance job launching and multiprogramming
- Global process coordination is as efficient in a large cluster as in a small cluster or even a desktop machine
- One step ahead in usability for large-scale machines

For more information:
`http://www.cs.huji.ac.il/~etcs`
or e-mail etcs@cs.huji.ac.il

I/O bypass mechanism in STORM



Quadrics interconnect scalability - barrier


Latency (μs)
Nodes