

MPICH on Clusters: Future Directions

Rajeev Thakur

Mathematics and Computer Science Division
Argonne National Laboratory

thakur@mcs.anl.gov

<http://www.mcs.anl.gov/~thakur>

Introduction

- Linux clusters are becoming popular all over the world
 - numerous small (~ 8-node) clusters
 - many medium-size (~ 48-node) clusters, which are growing
 - a few large clusters of several hundred nodes, such as, CPlant, LosLobos, Chiba City, FSL NOAA
- Terascale Linux clusters have been proposed (but not yet funded)

Software Challenges in Making Linux Clusters Truly Usable

- Parallel file system
- Scheduling
- Process startup and management
- Scalable communication software
- System administration tools
- Fault detection and recovery
- Programming models
- Interoperability of tools developed by independent parties

What We are Doing at Argonne

- Chiba City cluster
 - open testbed for scalability research
- MPICH Group
 - new MPI implementation -- starting from scratch
 - fast process startup and management (BNR/MPD)
 - MPI-2 functionality
 - efficient support for multithreading
 - new ADI design for better support of Myrinet, LAPI, VIA, etc.; as well as for existing TCP and SHMEM
- System Administration Tools
- Other Projects (e.g., Visualization, Grid)

Chiba City

**An Open Source
Computer Science
Testbed**



<http://www.mcs.anl.gov/chiba/>

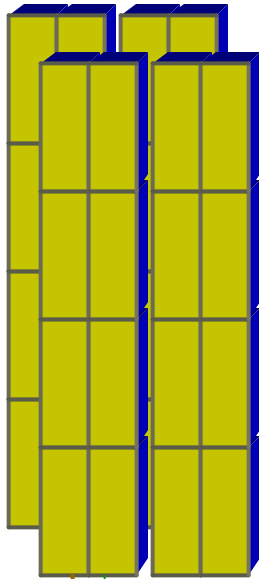
The Chiba City Project

- Chiba City is a 314-node Linux cluster at Argonne
- It is intended as a *scalability testbed*, built from *open source* components, for the high-performance computing and computer science communities
- It is intended as a first step towards a many-thousand node system

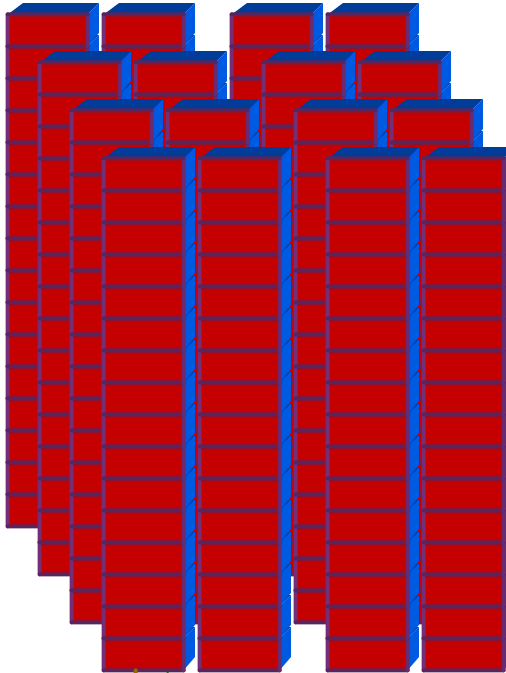
Chiba City

The Argonne Scalable Cluster

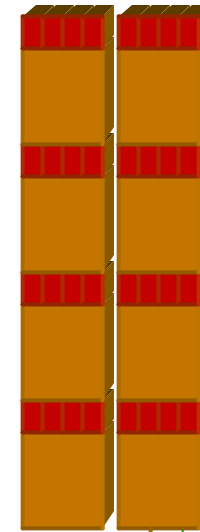
1 Visualization Town
32 Pentium III systems
with Matrox G400 cards



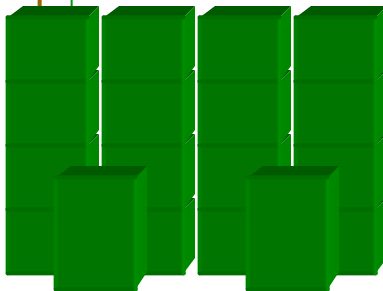
8 Computing Towns
256 Dual Pentium III systems



1 Storage Town
8 Xeon systems
with 300G disk each



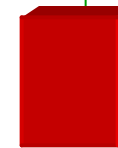
Cluster Management
12 PIII Mayor Systems
4 PIII Front End Systems
2 Xeon File Servers
3.4 TB disk



High Performance Net
64-bit Myrinet



Management Net
Gigabit and Fast Ethernet
Gigabit External Link



MPICH

- Portable MPI implementation
 - implements all of MPI-1 and I/O from MPI-2
- Formed the basis for many vendor and research MPI implementations
- First released in 1994; 15 releases since
- Current release: 1.2.1, September 2000

Limitations of Current MPICH

- Uses rsh to startup processes on a cluster
- Uses P4 for TCP communication
 - P4 is old, not scalable, and uses blocking sockets
- Does not support multimethod communication
- ADI was developed before the existence of GM, LAPI, VIA
- Implementation is not thread safe

Next Generation MPICH: MPICH2

- Scalable process startup (BNR/MPD)
- New ADI called ADI-3
- Support for newer networking technologies such as GM, LAPI, VIA
- Support for multimethod communication
- Full MPI-2 functionality
- Improved collective communication
- Thread safe
- High performance!

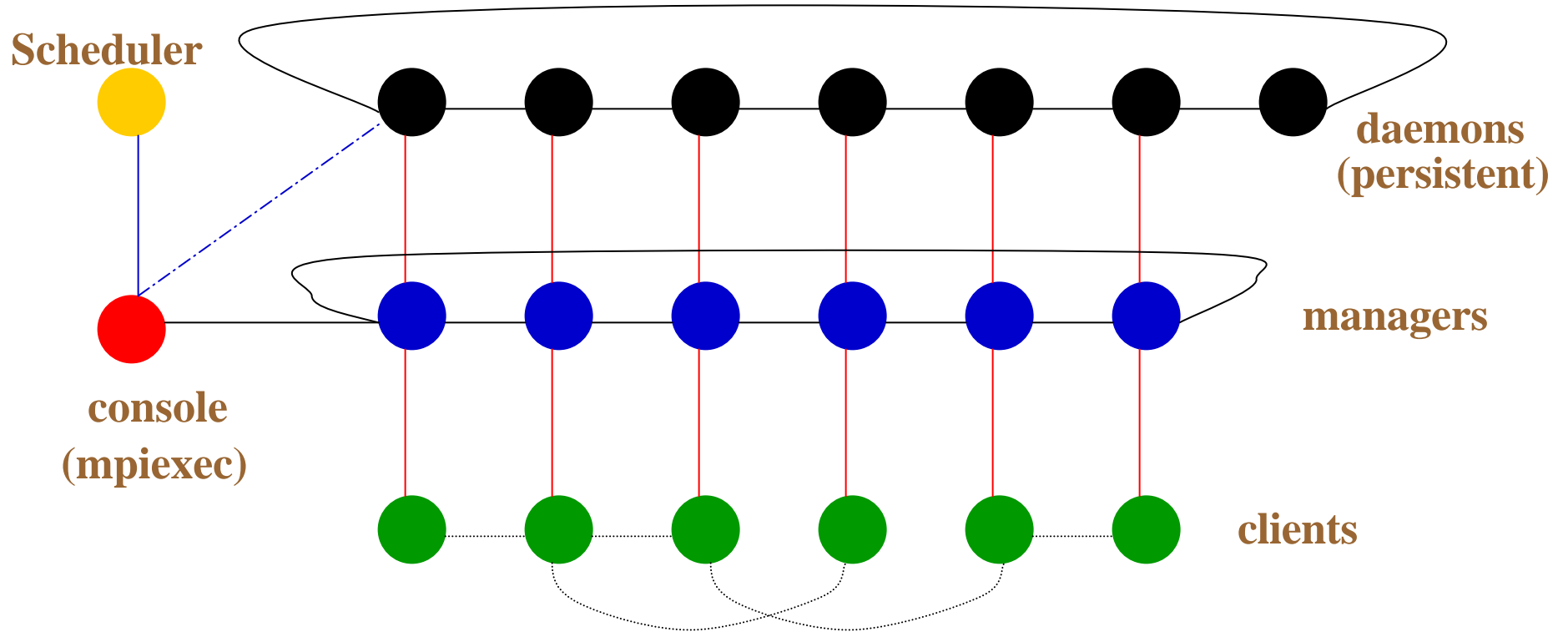
MPICH Team

- Bill Gropp
- Rusty Lusk
- Rajeev Thakur
- David Ashton
- Debbie Swider
- Anthony Chan
- Rob Ross

Multi-Purpose Daemon (MPD)

- Authors: Ralph Butler and Rusty Lusk
- A process management system for parallel programs
- Provides quick startup for parallel jobs of 1,000 processes, delivers signals, and handles `stdin`, `stdout`, and `stderr`
- Implements BNR interface
- Provides various services needed by parallel libraries
- Primary target is clusters of SMPs

MPD Architecture



mpigdb

A Poor Man's Parallel Debugger

- `mpigdb -np 4 cpi`
runs `gdb cpi` on each process
- redirects `stdin` to `gdb` and forwards `stdout` from `gdb` on each process
- adds line labels to output from `gdb` to indicate process rank
- User can send `gdb` commands to either a specific process or to all processes
- Result: A useful parallel debugger

Parallel Debugging with `mpigdb`

```
donner% mpigdb -np 3 cpi
(mpigdb) b 33
0: Breakpoint 1 at 0x8049eac: file cpi.c, line 33.
1: Breakpoint 1 at 0x8049eac: file cpi.c, line 33.
2: Breakpoint 1 at 0x8049eac: file cpi.c, line 33.
(mpigdb) r
2: Breakpoint 1, main (argc=1, argv=0xbffffab4) at cpi.c:33
1: Breakpoint 1, main (argc=1, argv=0xbffffac4) at cpi.c:33
0: Breakpoint 1, main (argc=1, argv=0xbffffad4) at cpi.c:33
(mpigdb) n
2: 43          MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
0: 39          if (n==0) n=100; else n=0;
1: 43          MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
(mpigdb) z 0
(mpigdb) n
0: 41          startwtime = MPI_Wtime();
(mpigdb) n
0: 43          MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
(mpigdb)
```

Continuing...

```
(mpigdb) z
(mpigdb) n
      . . . .
(mpigdb) n
0: 52      x = h * ((double)i - 0.5);
1: 52      x = h * ((double)i - 0.5);
2: 52      x = h * ((double)i - 0.5);
(mpigdb) p x
0: $2 = 0.0050000000000000000001
2: $2 = 0.0250000000000000000001
1: $2 = 0.014999999999999999999
(mpigdb) c
0: pi is approximately 3.1416009869231249,
0: Error is 0.000008333333333318
0: Program exited normally.
1: Program exited normally.
2: Program exited normally.
(mpigdb) q
donner%
```

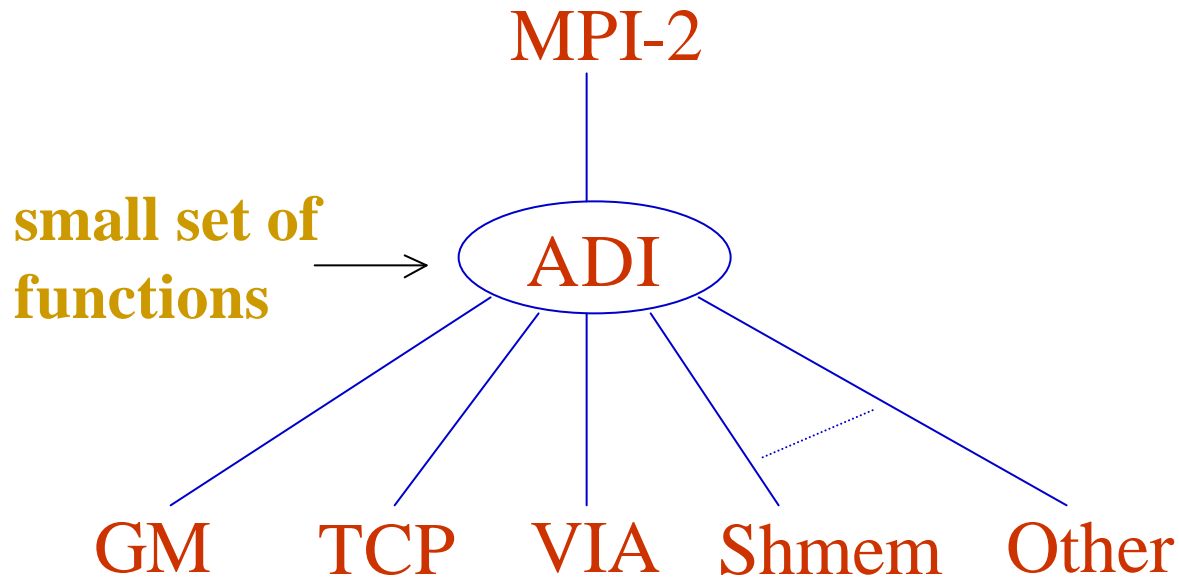
BNR: An Interface to Process Managers

- MPD has a client library that application processes can call to interact with MPD
- BNR is the API for this library
- Intended as a small, simple interface that anyone can implement
- Any MPI implementation can run on any process manager if both use the BNR API
- In this way, MPICH2 can run on IBM's POE, for example

BNR

- BNR provides a repository of (key, value) pairs that a process can insert (`BNR_Put`) for another to retrieve (`BNR_Get`)
- Using this, a process can publish the TCP port on which it is listening, for example
- `BNR_Spawn` is used to spawn new process
- `BNR_Merge` is used to merge groups of processes
- BNR is used both by `mpiexec` to launch new jobs, as well as by the MPI library on each user process

Abstract Device Interface (ADI-3)



Question: How should the ADI be designed so that MPI can be implemented efficiently on any of the underlying methods?

Our Progress So Far

- We considered and partially implemented three ADI designs so far:
 - a very low level “Channel Device”
 - an intermediate level “RMQ Device”
 - a higher level “MPID Device”
- Eventually discarded the first two for performance reasons
- Channel device may be resurrected in some form to enable quick ports to new platforms

Channel Device

- A very small set of basic communication functions
- Short, eager, rendezvous protocols handled above device
- Advantages
 - can be quickly and easily implemented
- Disadvantages
 - difficult to define one that is good for all communication methods
 - ill-suited for communication using shared memory
 - hard to exploit idiosyncrasies of specific methods
 - performance limitations

RMQ Device

- Higher level than channel device
- Provides a mechanism for queuing messages on send and receive sides
- Fundamental operation is “match on remote queue and transfer data”
- Advantages
 - allows implementation to optimize communication for underlying method
- Disadvantages
 - always requires queuing requests on both sides.
 - expensive for short messages

MPID Device

- Higher level than RMQ
- Does not assume queuing
- Supports noncontiguous transfers
- Short, eager, rendezvous implemented underneath, as and if required by the method
- Provides a mechanism to deal with “any source” receives in multimethod case
- Provides utility functions that methods may use for queueing, handling datatypes, etc.
- Prototype implementation for SHMEM and TCP; VIA implementation underway

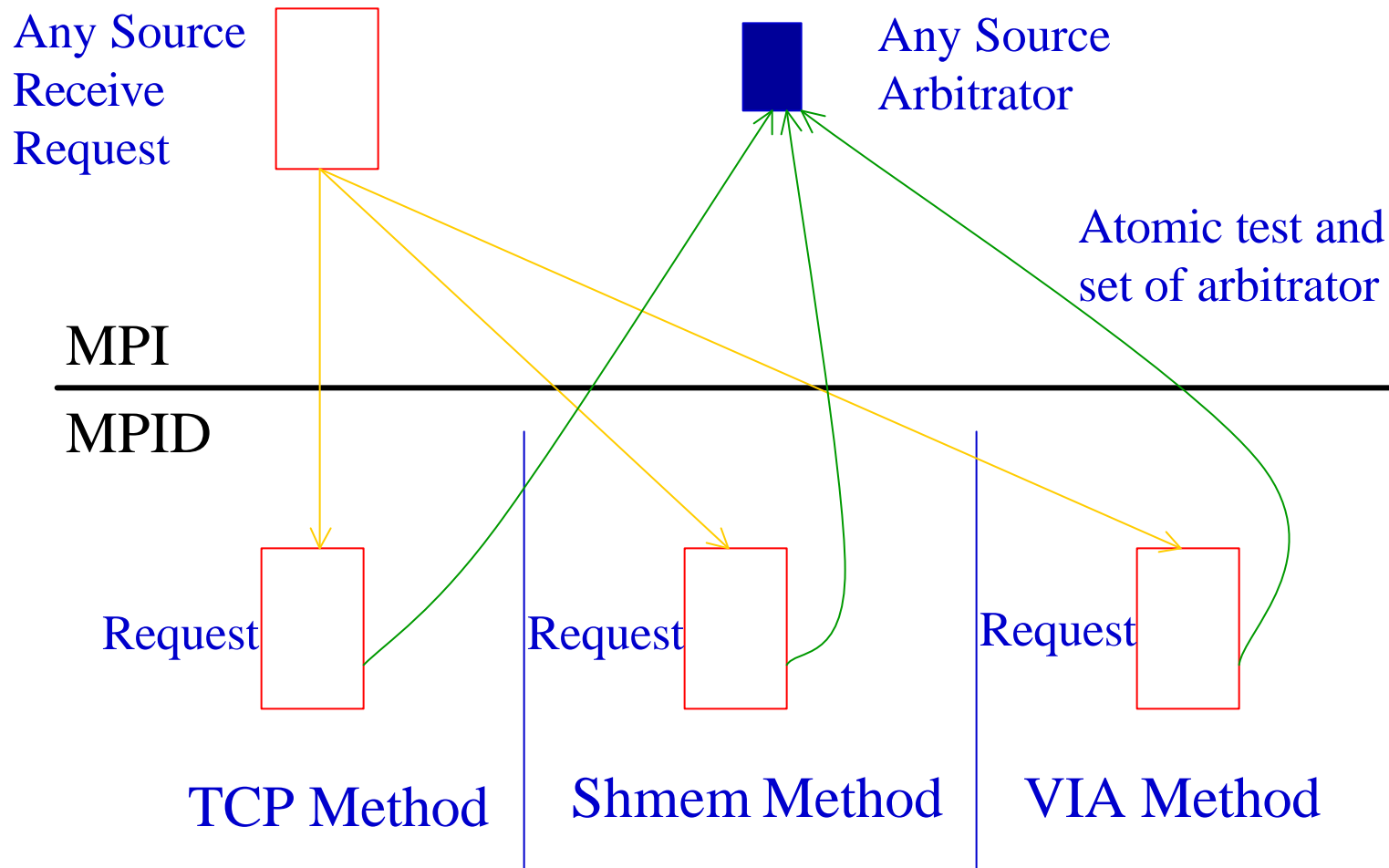
MPID Device

- Advantages
 - method implementation has complete flexibility
 - higher performance
- Disadvantages
 - harder to implement
- The sample implementations we provide as part of MPICH2 and the utility functions should help others implement MPID on new platforms

“Any Source” Receives

- In MPI, the user can receive data from any source by specifying `MPI_ANY_SOURCE`
- This can be a complication in multimethod case, because a message from any, and only one, method must be allowed to match
- MPID provides an arbitration scheme (utility functions) for supporting any source receives

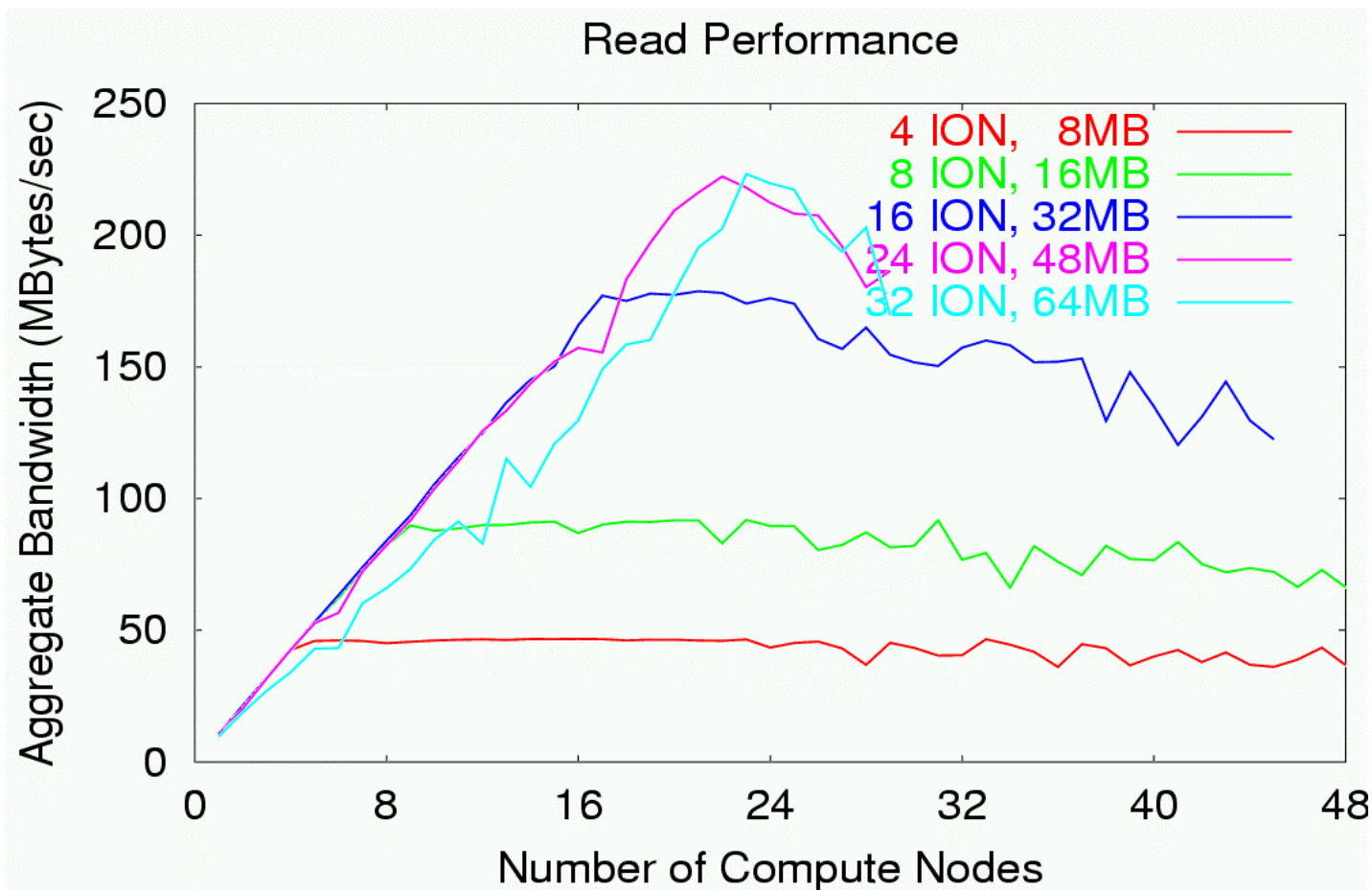
Any Source Arbitration



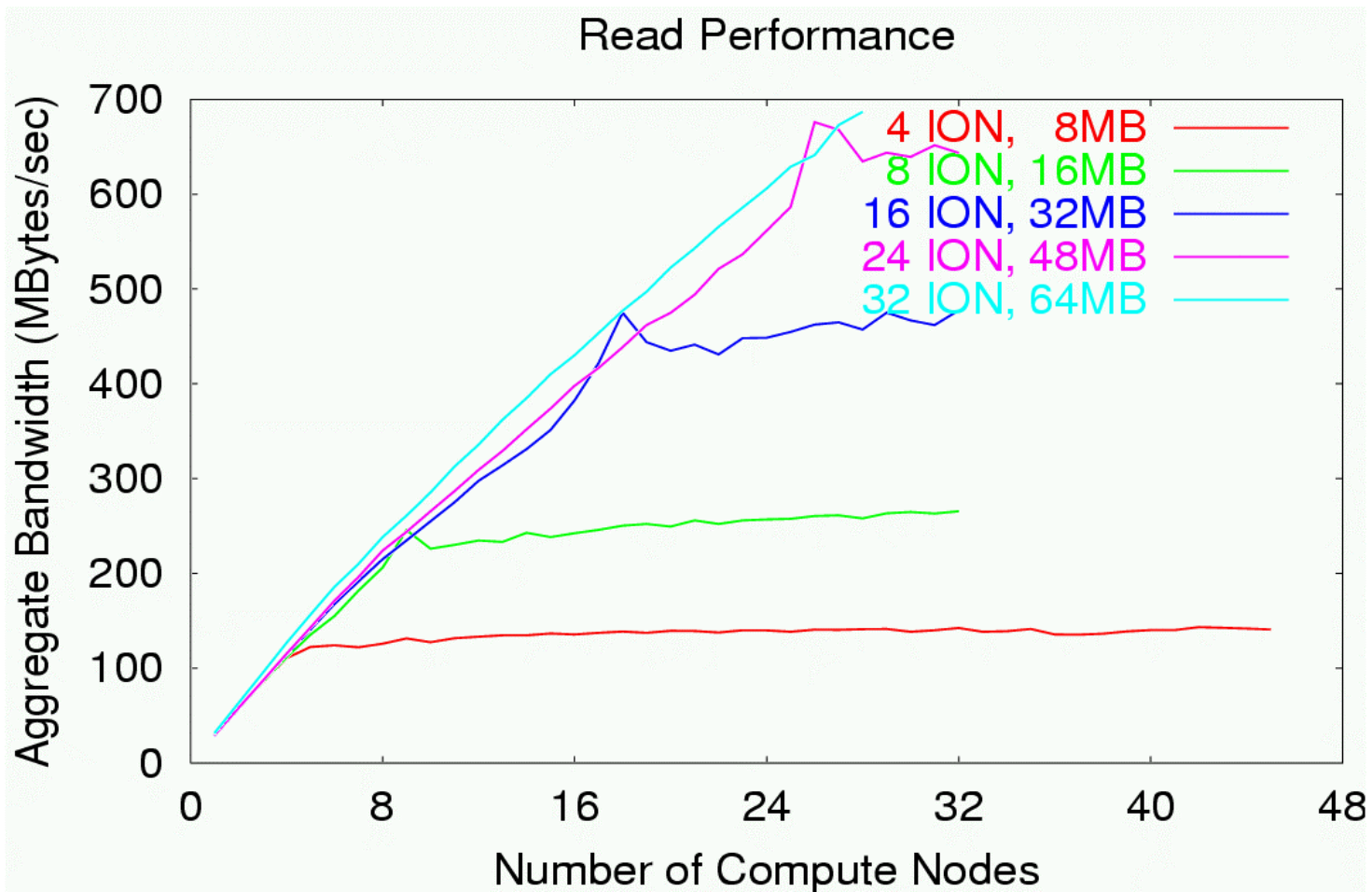
Parallel I/O

- The greatest weakness of Linux clusters
- We are using the PVFS parallel file system developed at Clemson University
- It works, but needs more work to be usable as a reliable, robust file system
- Rob Ross, author of PVFS, has joined MCS as a postdoc
- ROMIO (MPI-IO) has been implemented on PVFS
- See Extreme Linux 2000 paper

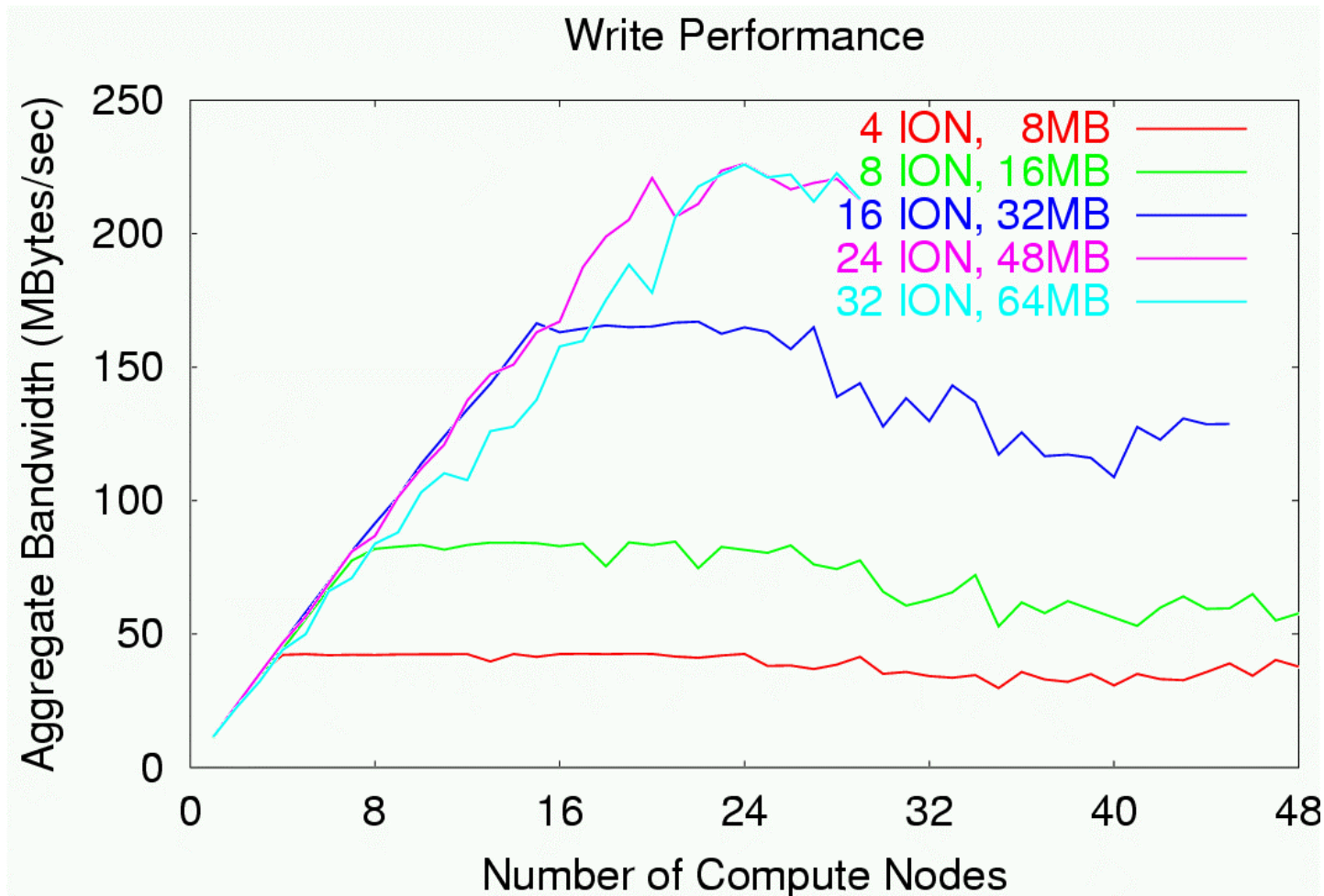
PVFS Read Performance: Fast Ethernet



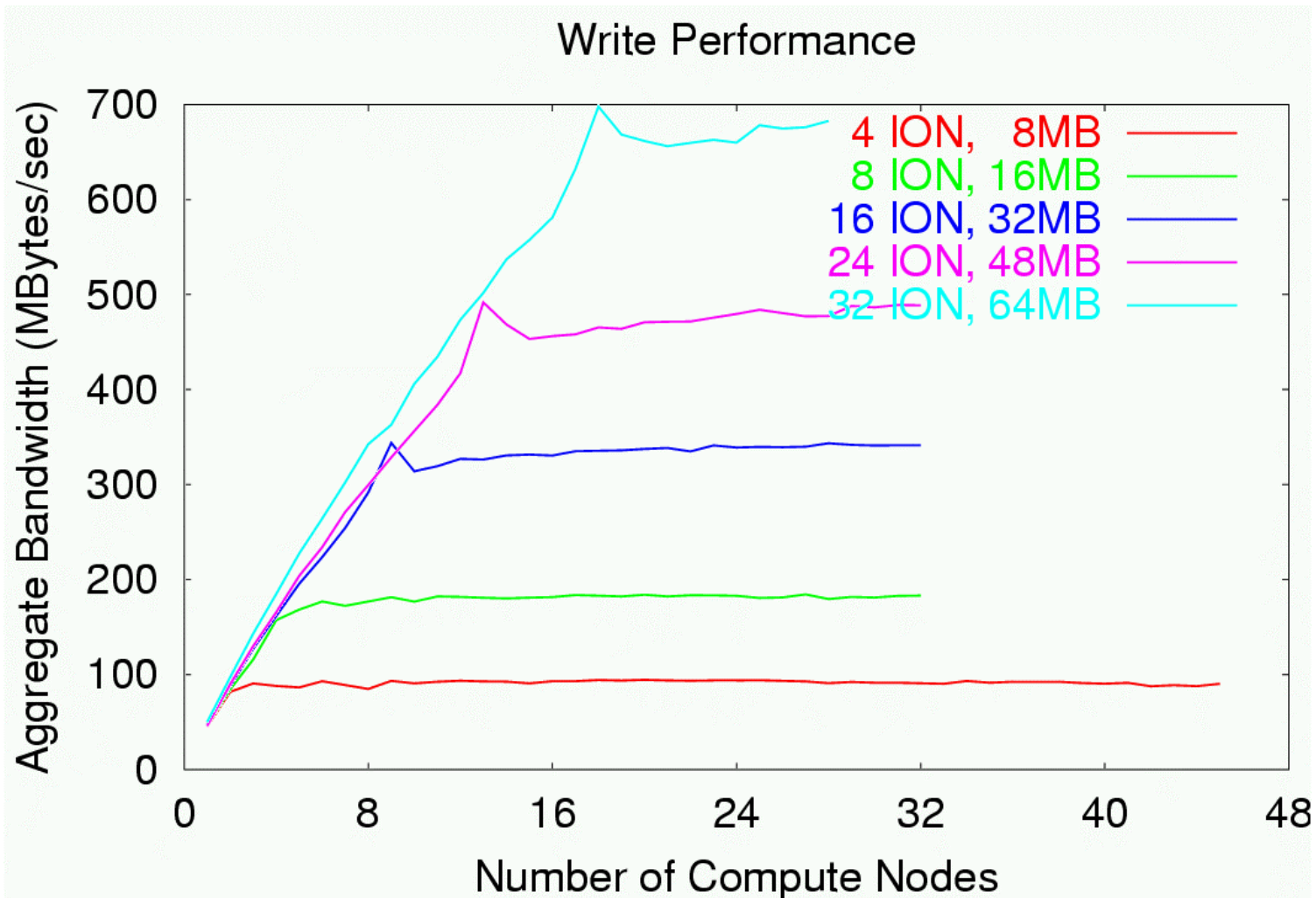
PVFS Read Performance: TCP on Myrinet



PVFS Write Performance: Fast Ethernet



PVFS Write Performance: TCP on Myrinet



Summary

- Linux clusters have captured the imagination of the parallel-computing world
- Much work needs to be done to make them really usable, however
- Argonne's work focuses on scalability and performance in the areas of testbeds, process management, message passing, parallel I/O, system administration, and others

MPICH on Clusters: Future Directions

Rajeev Thakur

Mathematics and Computer Science Division
Argonne National Laboratory

thakur@mcs.anl.gov

<http://www.mcs.anl.gov/~thakur>