

# Scalable Parallel Application Launch on Cplant<sup>TM</sup>

**Ron Brightwell**

**Lee Ann Fisk**

**Sandia National Laboratories**

**Scalable Computing Systems Department**

**[bright@cs.sandia.gov](mailto:bright@cs.sandia.gov)**

# Outline

---

- **Motivation**
- **Runtime components**
- **Design issues**
- **Launch phases**
- **Overview of experiment**
- **Results**
- **Analysis**
- **Future work**
- **Summary**



# Why is scalable parallel application launch important?

---

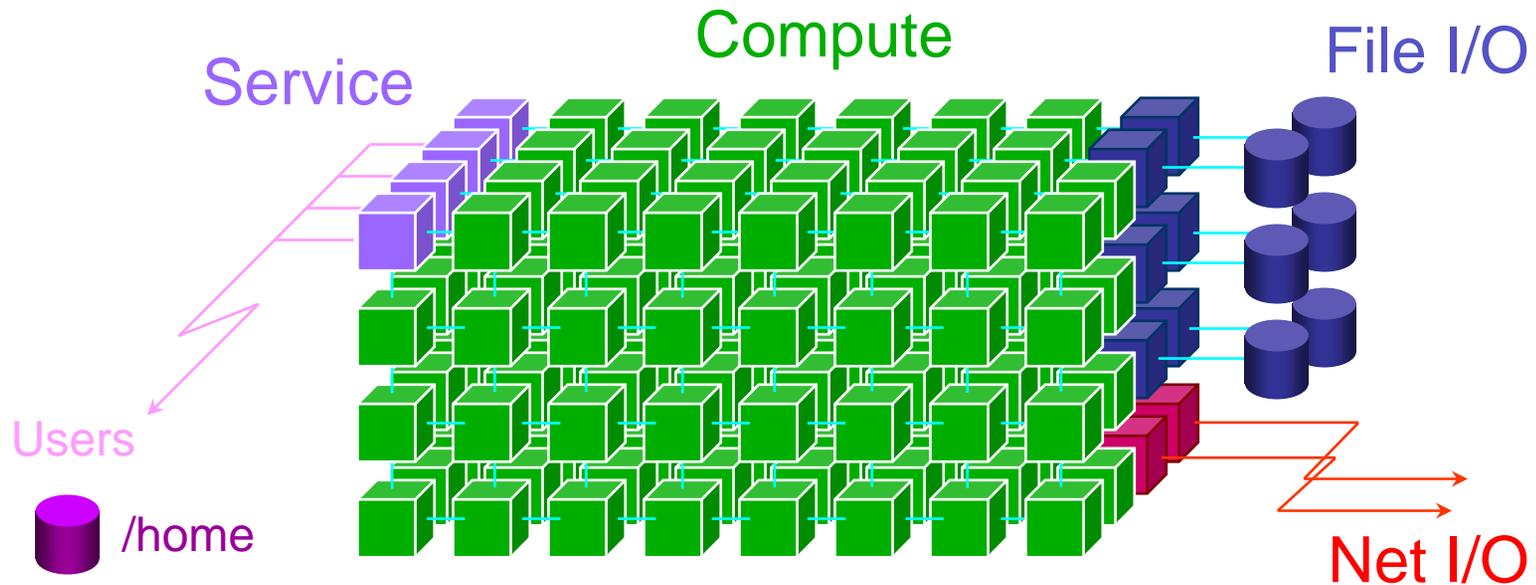
- Reduce time to solution
- Increase effective utilization
- Increase usability
- Most systems address performance once an application is running
- Few address the time spent getting a parallel job started
- Using rsh/ssh has inherent scalability and performance problems

# What is Cplant™?

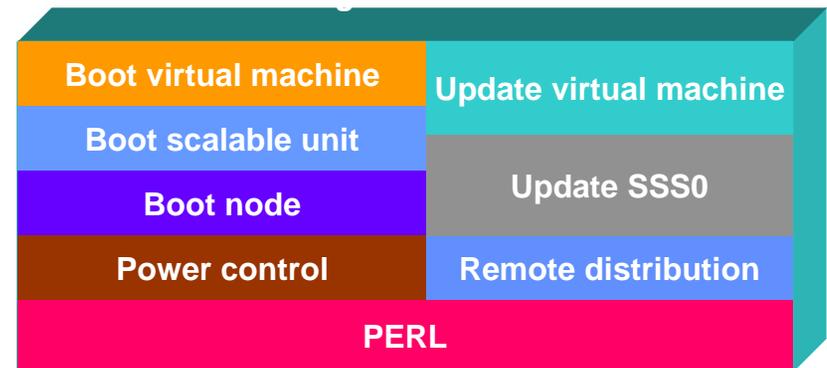
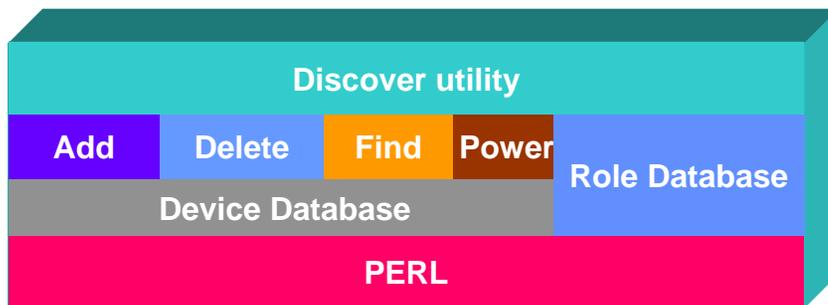
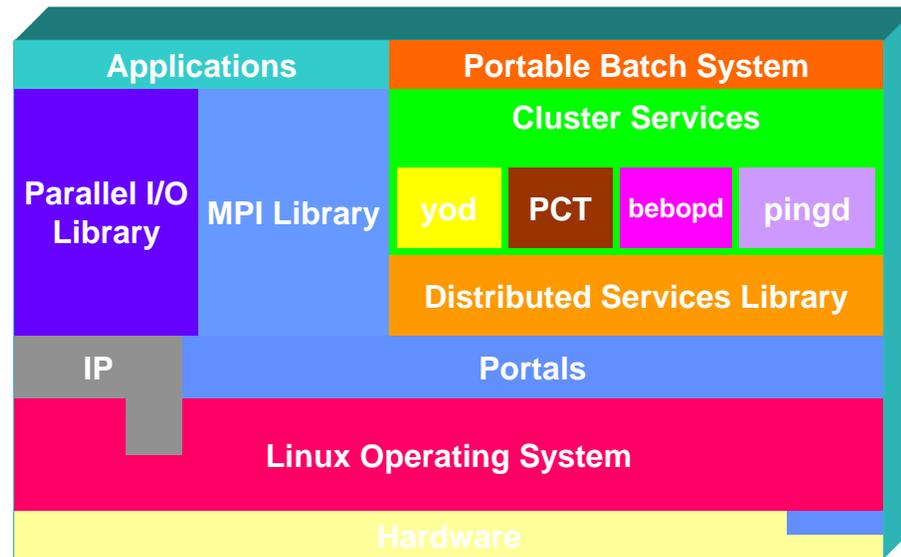
---

- **Cplant™ is a concept**
  - Provide computational capacity at low cost
  - Build MPPs from commodity components
  - Follow ASCI/Red model and architecture
- **Cplant™ is an overall effort:**
  - Multiple computing systems in NM & CA
  - Multiple projects
    - Portals 3.0 message passing (with UNM)
    - Cluster infrastructure tools (with HPTi)
    - System integration & test
    - Operations & management
- **Cplant™ is a software package**
  - Recently released as open source (GPL)
  - Commercial license to Unlimited Scale, Inc. (see Compaq booth)

# Conceptual Partition Model



# Cplant™ Software



# Runtime Environment Components

---

- **Yod**
  - xnc++
  - Service node parallel job launcher
- **Bebopd**
  - Better Engineered Bag Of PCs Daemon
  - Compute node allocator
- **PCT**
  - Process Control Thread
  - Compute node daemon
- **pingd/showmesh**
  - Compute node status tools
- **PBS**
  - Batch scheduler

## Runtime Environment (cont'd)

---

- **Yod**
  - **Contacts compute node allocator**
  - **Launches the application into the compute partition**
  - **Redirects all application I/O (stdio, file I/O)**
  - **Makes any filesystem visible in the service partition visible to the application**
  - **Redirects any UNIX signals to compute node processes**
  - **Allows user to choose specific compute nodes**
  - **Can launch multiple different binaries**
  - **Displays launch timing information**
  - **Same basic interface as SUNMOS and Cougar**

## Runtime Environment (cont'd)

---

- PCT

- Contacts bebopd to join compute partition
- Forms a spanning tree with other PCT's to fan out the executable, shell environment, signals, etc.
- Puts executable in a 16 MB RAM disk
- *fork()*'s, *exec()*'s, and monitors status of child process
- Cleans up after parallel job

## Runtime Environment (cont'd)

---

- **Bebopd**
  - Accepts requests from PCT's to join the compute partition
  - Accepts requests from yod for compute nodes
  - Accepts requests from pingd for status of compute nodes
  - Coordinates scheduling with PBS server
  - Allows for multiple compute partitions

## Runtime Environment (cont'd)

---

- **Pingd**
  - Displays list of available compute nodes
  - Displays list of compute nodes in use
  - Displays owner, elapsed time of jobs
  - Allows users to kill their jobs
  - Allows administrators to kill jobs and free up specific nodes
  - Allows administrators to remove nodes from the compute partition
- **Showmesh**
  - Massages pingd output into TFLOPS-like showmesh

# Runtime Environment (concl'd)

---

- **PBS**
  - Enhanced version of OpenPBS
  - Added non-blocking I/O for fault tolerance
  - PBS Moms and Server only run in the service partition
  - Added new attribute – “nodes”
  - Contacts bebopd to get a list of nodes to give to yod

# Timed Phases of Launch

---

- **Allocate nodes – yod asks bebopd for nodes**
- **Initial load message – yod sends a message to all PCTs**
- **Form group- the PCTs form a spanning tree and report to yod**
- **Pull arguments – lead PCT pulls arguments from yod and fans them out**
- **Pull environment – lead PCT pulls environment from yod and fans it out**
- **Read file – yod reads executable**
- **Fan out executable – lead PCT reads executable and fans it out**
- **Pull map – yod gathers nid/pid information and fans it out**
- **Log time – yod sends log data to bebopd**
- **Total time – total time from invocation to job start**

# Design Issues

---

- **Two ways to move executable to compute nodes**
  - **Pull executable to compute nodes**
    - Requires some intelligence in the filesystem
    - Filesystems can't handle N-to-1 reads
  - **Push executable to compute nodes**
    - No filesystem dependency
    - Easier to implement
- **Need to start processes in parallel**
- **Support for other programming models**
  - Job launch should not be specific to the programming model
- **Fault detection**

## Design Issues (cont'd)

---

- **Bebopd is a single point of failure**
  - No new jobs runs if bebopd goes away
  - Distributed bebopd
    - Failure only affects part of the cluster
    - Haven't needed to do it yet
  - **Bebopd checkpoints the state of the machine and can be restarted**

# Experiment

---

- **Built a minimal MPI executable**
- **Padded it with zeroes to build 2 MB – 12 MB files**
- **Launched jobs at least 10 times and took average**
- **Gathered data on**
  - **Dedicated system**
  - **System under high load**
  - **Jobs launched under PBS**

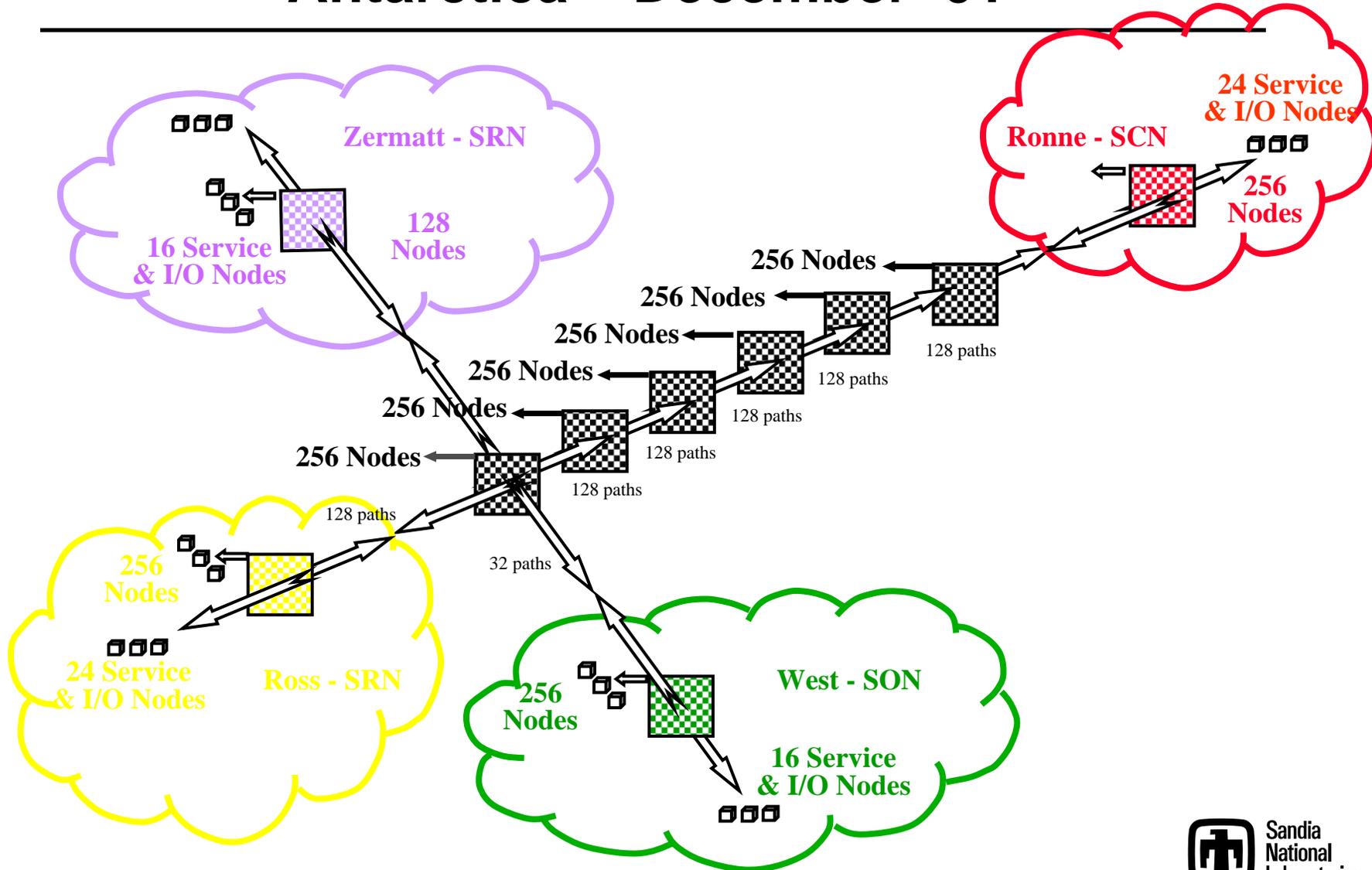
# Antarctica

---

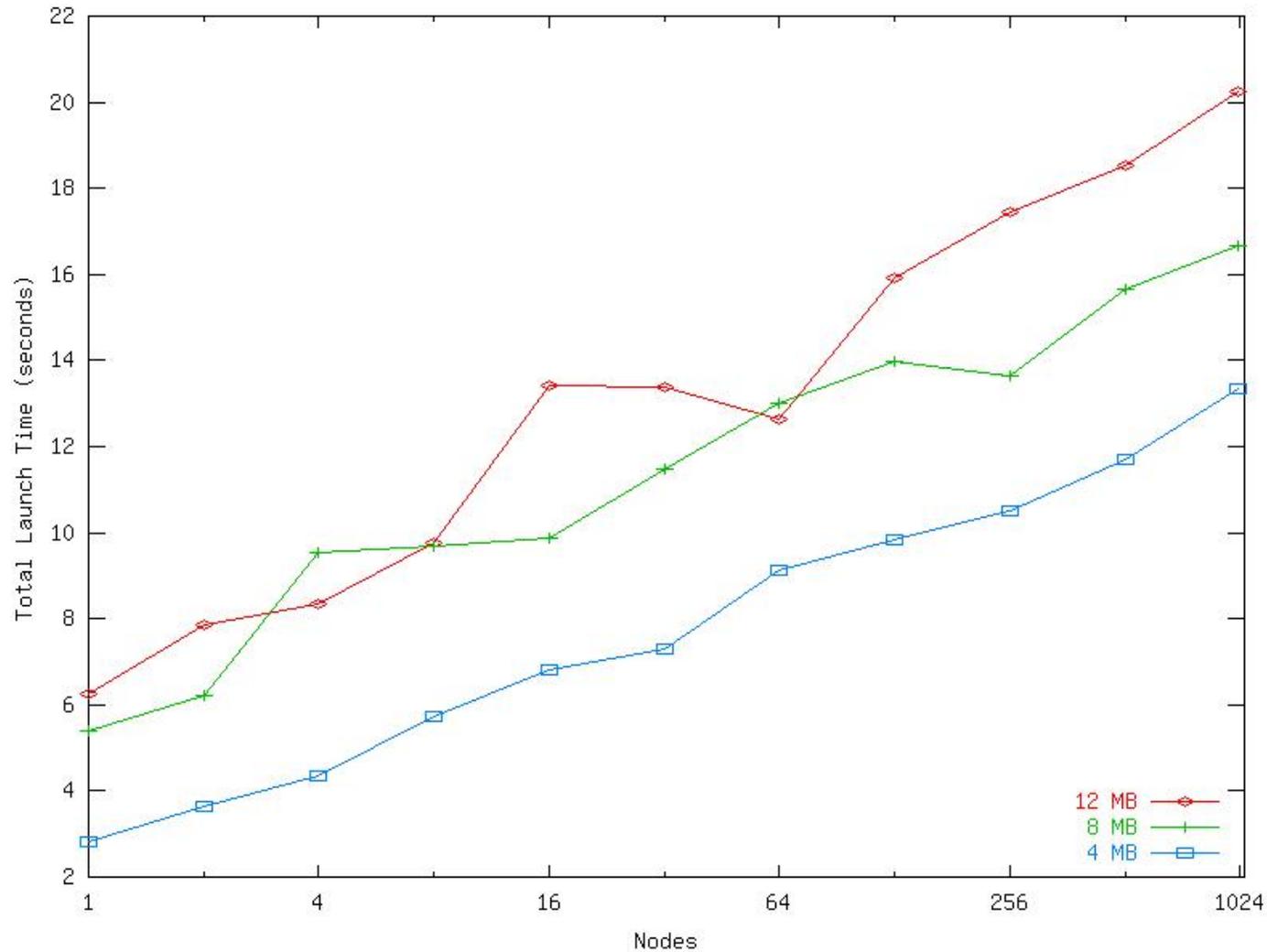
- 1792+ Compaq DS10L Slates
    - 466MHz EV6, 256 MB RAM
  - 590 Compaq XP1000s
    - 500 MHz EV6, 256 MB RAM
  - Myrinet 33MHz 64bit LANai 7.x and 9.x
  - Myrinet Mesh64 switches
  - Classified, unclassified, open, and development network heads
- 
- Fastest Linux cluster
  - 706.7 GFLOPS on 1369 nodes
  - #30 on current Top500
  - Launched in 26 seconds



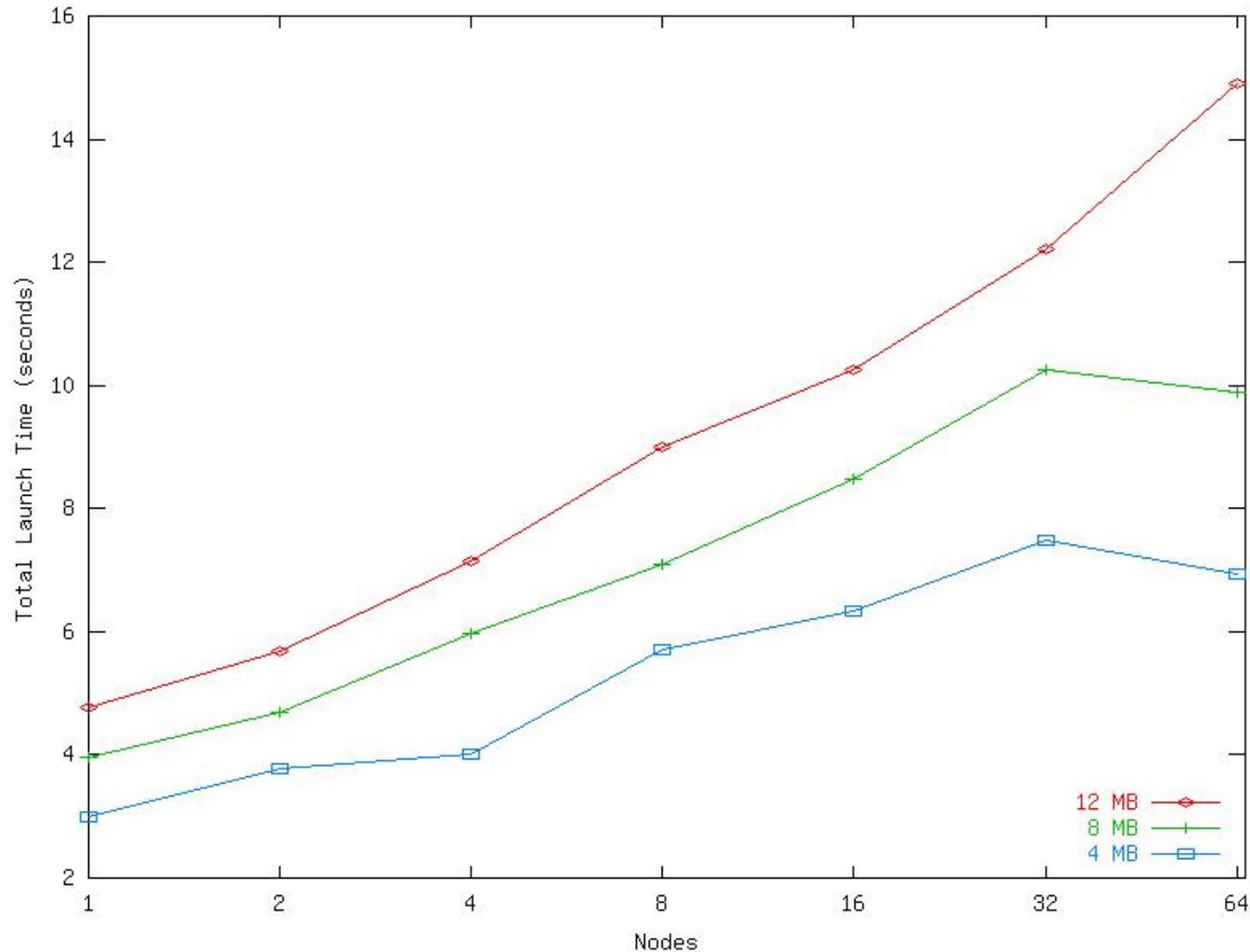
# Antarctica – December '01



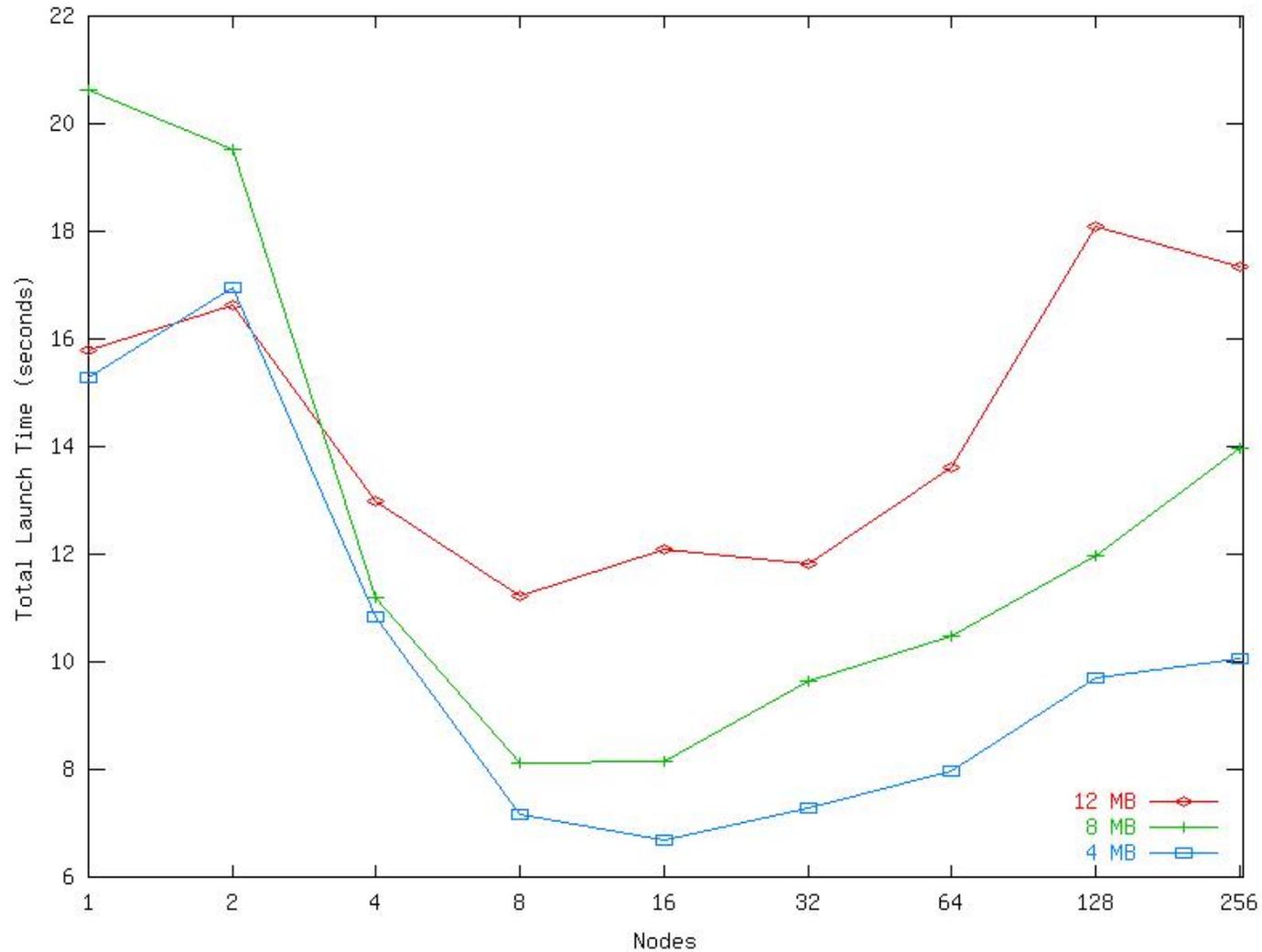
# Launch Time – Dedicated System



# Launch Time – Loaded System

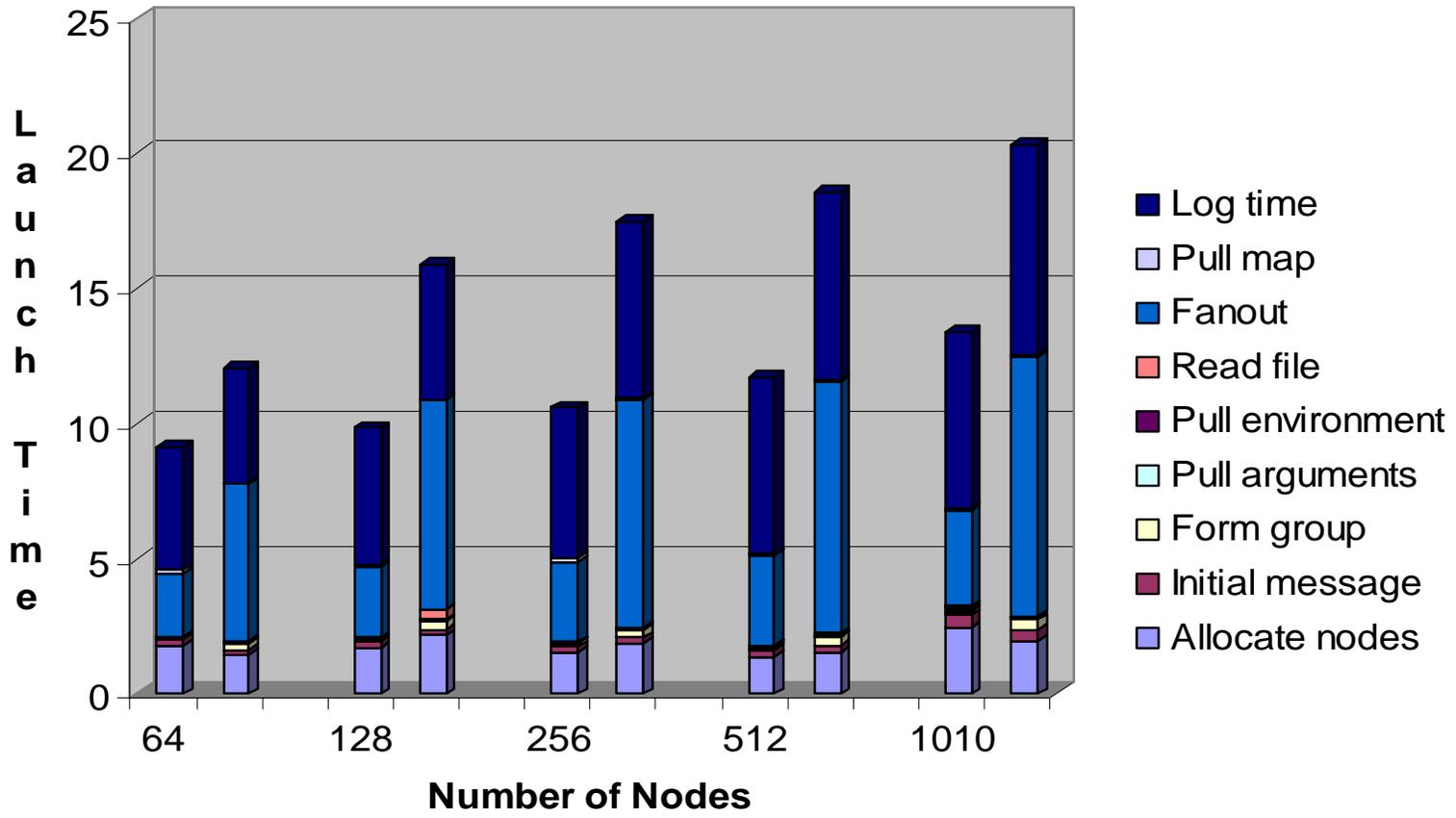


# Launch Time – PBS Job





# Launch Breakdown for 4 MB and 12 MB Executables



# Analysis

---

- **Form group**
  - Involves all nodes sending directly to yod
  - Inherently non-scalable, but simple and easy to detect failures
  - Move to more scalable approach when message passing is more robust
- **Broadcast**
  - Expected result
  - Better network bandwidth will help
  - Higher-degree fan out may be needed
- **Log time**
  - Yod sends a node list to bebopd
  - Bebopd writes log info to a file
  - More investigation needed
- **Responsiveness of the bebopd is important**

# Future Work

---

- **Intelligent allocator**
  - Current allocator does not account for network topology or routes
  - Ideal allocator would allocate contiguous nodes
  - Measure impact on load time
- **Dynamic process creation**
  - Support for MPI-2 dynamic process creation functions
- **Multiprocessor support**
  - Current environment supports one process per node
- **Library API for runtime system interaction**
  - Host library for custom allocator

# Summary

---

- **Ability to launch jobs fast is critical to usability and efficiency of a large parallel machine**
- **Designed and implemented a flexible runtime system for launching a job on more than a thousand nodes in several seconds**
- **Working on improvements and enhancements**
- **Baseline for measuring the impact of these new features**