Orientation Agenda – Part 1

Scyld ClusterWare foundations
  » Origins of the Bproc architecture
  » Differences compared with traditional Linux clusters
  » Benefits of the Bproc architecture

Scyld Cluster Conventions and Commands
  » Hostname conventions
  » Cluster monitoring and status
  » Remote access limitations
  » Local disc and shared data access
  » Common ClusterWare commands

Break
Orientation Agenda – Part 2

- **Interactive and Serial jobs**
  - Basic command spawning
  - Resource management
  - Process monitoring and job control

- **Parallel jobs and Queuing**
  - Commands to spawn parallel jobs
  - Job queuing

- **Additional resources and documentation**

- **Questions and Answers**
Orientation Agenda – Part 1

- **Scyld ClusterWare foundations**
  - Origins of the Bproc architecture
  - Differences compared with traditional Linux clusters
  - Benefits of the Bproc architecture

- **Scyld Cluster Conventions and Commands**
  - Hostname conventions
  - Cluster monitoring and status
  - Remote access limitations
  - Local disc and shared data access
  - Common ClusterWare commands

- **Break**
Orientation Agenda – Part 1

- Scyld ClusterWare foundations
  - Origins of the Bproc architecture
  - Differences compared with traditional Linux clusters
  - Benefits of the Bproc architecture

- Scyld Cluster Conventions and Commands
  - Hostname conventions
  - Cluster monitoring and status
  - Remote access limitations
  - Local disc and shared data access
  - Common ClusterWare commands

- Break
Orientation Agenda – Part 2

- Interactive and Serial jobs
  - Basic command spawning
  - Resource management
  - Process monitoring and job control

- Parallel jobs and Queuing
  - Commands to spawn parallel jobs
  - Job queuing

- Additional resources and documentation

- Questions and Answers
Questions??
Beowulf Clustering: The Early Years

- Conceived by Becker and Sterling in '93 and initiated at NASA in '94
- Objective: show that commodity clusters could solve problems usually handled by $million supercomputers but at a fraction of the cost
- Build a system that scaled in all dimensions
  - Networking, bandwidth, disks, main memory, processing power
- Initial prototype
  - 16 processors, Channel-bonded Ethernet, under $50k
  - Matched performance of contemporary $1M machine
Traditional “Ad-Hoc” Linux Cluster

- Full Linux install to disk; Load into memory
  - Manual & Slow; 5-30 minutes
- Full set of disparate daemons, services, user/password, & host access setup
- Basic parallel shell with complex glue scripts run jobs
- Monitoring & management added as isolated tools
What can be improved on this approach?

- Complexity
- Clusters required extensive training to install, configure and use
- Long-term administration and updates were difficult
- Only “static scalability”

Good cluster systems virtualize

- Create a unified view of independent machines
- Single System Image (Illusion)
Cluster Virtualization Architecture Realized

- **Minimal in-memory OS with single daemon rapidly deployed in seconds - no disk required**
  - Less than 20 seconds

- **Virtual, unified process space enables intuitive single sign-on, job submission**
  - Effortless job migration to nodes

- **Monitor & manage efficiently from the Master**
  - Single System Install
  - Single Process Space
  - Shared cache of the cluster state
  - Single point of provisioning
  - Better performance due to lightweight nodes
  - No version skew is inherently more reliable
Some important elements of a cluster system

» Booting and Provisioning
» Process creation, monitoring and control
» Update and consistency model
» Name services
» File Systems
» Physical management
» Workload virtualization
Booting and Provisioning
Booting requirements

- Reliable booting
- Automatic scaling
- Centralized control and updates
- Reporting and diagnostics designed into the system
- No dependency on machine-local storage or configuration
The Scyld solution

- Integrated, automatic network boot
- Basic hardware reporting and diagnostics in the Pre-OS stage
- Only CPU, memory and NIC needed
- Kernel and minimal environment from master
- Just enough to say “what do I do now?”
- Remaining configuration driven by master
Booting Implementation

- **Integrated boot server, supporting PXE and other protocols**
  - Avoids network booting reliability issues
  - Understands PXE versions and bugs
  - Avoids TFTP "capture effect"
  - Built-in dynamic node addition
  - DHCP service for non-cluster hosts

- **Hardware reporting and diagnostics in pre-boot environment**
  - Used to provide specific boot images
Provisioning Design:

- Traditional full standard installation on Masters
- Dynamically built "distribution" on other nodes

Core system is "stateless" non-persistent

- Provided by the boot server as ramdisks
- No local storage or network file system used
- Additional elements provided under centralized master control
### Installation, initial provisioning, and updates

<table>
<thead>
<tr>
<th>FROM (traditional)</th>
<th>TO (Stateless Auto-Provisioning)</th>
</tr>
</thead>
<tbody>
<tr>
<td>§ Disk-Based Install of entire cluster</td>
<td>§ Single Install on the Master</td>
</tr>
<tr>
<td>- Script or tool maintenance (kickstart)</td>
<td>- Auto-provisions in-memory OS to nodes</td>
</tr>
<tr>
<td>- 15-30 minutes/node, days/cluster</td>
<td>- &lt; 20 seconds/node; min. to hours/cluster</td>
</tr>
<tr>
<td>§ Complex, Scripted Node Updates</td>
<td>§ Single install of the update on Master</td>
</tr>
<tr>
<td>- Login/apply foreach node</td>
<td>- Master auto-updates nodes</td>
</tr>
<tr>
<td>- Manually correct failed updates</td>
<td>- Guarantees consistency</td>
</tr>
<tr>
<td>§ Replacing a failed node is painful</td>
<td>§ Replacing a failed node is effortless</td>
</tr>
<tr>
<td>- Replace hardware, power on</td>
<td>- Replace hardware, power on</td>
</tr>
<tr>
<td>- Manual Install and Configuration</td>
<td>- Node is up in 20 seconds</td>
</tr>
<tr>
<td>- 15-30 minutes</td>
<td></td>
</tr>
<tr>
<td>§ Disk installs get stale, inconsistent</td>
<td>§ Stateless provisioning always consistent</td>
</tr>
<tr>
<td>- Tedious manual verification</td>
<td>- Refresh in 20 seconds if need be</td>
</tr>
<tr>
<td>- Or 30 minute reprovision</td>
<td></td>
</tr>
</tbody>
</table>

#### Turning Provisioning from Painful to Effortless

Focus on application challenges - Not node administration
Process creation, monitoring, and control
Process Initiation, Monitoring and Control

**Problems:**

» Starting jobs on a dynamic cluster

» Monitoring and controlling running processes

» Allowing interactive and scheduler-based usage

**Opportunity:**

» Clusters jobs are issued from designated masters

» That master has exactly the required environment

» We already have a POSIX job control model

» Tight communication
§ **Semantics: Execution consistency**

» Remote execution produces same results as local execution

§ **Implications:**

» same executable (including version!)

» same libraries, including library linkage order

» same parameters and environment
Run-time provisioning architecture

We "copy" running processes

» Correct: We know the current versions, library linkage order, environment, etc.

» Fast: libraries and executables are cached

» Transfer and cache as versioned whole files

» With dynamic caching we can start with "nothing"!

» Dynamically builds a minimal custom installation

Feasible and efficient because "slave nodes" run only applications,

» not full OS install
Cluster-wide process space

- Single virtual process space over cluster
  - Standard process monitoring tools work unchanged
  - Negligible performance impact
  - Careful design avoids cascading failures
  - Major operational and performance benefits

- Consider cluster-wide “killall”
  - Over 7 minutes on University of Arizona cluster with 'ssh'
  - Real-time, interactive response with Scyld approach
Unified process space implementation

§ Correct semantics and efficient monitoring/control

» Implemented by extending master's process table

» 2.2/2.4 Linux kernel implementation with custom hooks

» Redesigned 2.6 implementation minimizes/eliminates hooks
Update and Consistency Model
## Maintaining Efficiency, Scalability, Security

<table>
<thead>
<tr>
<th>FROM (Traditional)</th>
<th>TO (Lightweight Compute Nodes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuring large numbers of servers is painful</td>
<td>Compute nodes are slaves to run jobs directed by Master</td>
</tr>
<tr>
<td></td>
<td>Users, Services, Access, Security etc.</td>
</tr>
<tr>
<td>Multiple Services, Agents, Daemons consume resources &amp; compromise</td>
<td>Right Sized Compute Node OS consumes less resources</td>
</tr>
<tr>
<td>Performance and Scalability</td>
<td>(8 MB typical RAM)</td>
</tr>
<tr>
<td>Memory Bandwidth (400Mb Full Install)</td>
<td>Can always auto provision extra libraries</td>
</tr>
<tr>
<td>Scheduling latencies of daemons, services</td>
<td>5-15% Performance Improvement typical</td>
</tr>
<tr>
<td>Multiple agents, extra traffic won’t scale</td>
<td>Optimized management agent design scales well</td>
</tr>
<tr>
<td>Full User and Access requires configuration and opens security holes</td>
<td>Virtual Fortress</td>
</tr>
<tr>
<td></td>
<td>Lightweight Slave with no attack points is inherently more secure and reliable</td>
</tr>
</tbody>
</table>

Dynamically Provisioned Compute Nodes Dramatically Improves Efficiency, Scalability, Security
### Simplifying the complexity of managing server pools

<table>
<thead>
<tr>
<th>FROM (Traditional)</th>
<th>TO (Scyld)</th>
</tr>
</thead>
<tbody>
<tr>
<td>§ Manual management of multiple, individual servers</td>
<td>§ Present the cluster as a single, powerful Linux system</td>
</tr>
<tr>
<td>1. Tedious, time-consuming</td>
<td>1. As easy as managing a single workstation</td>
</tr>
<tr>
<td>§ Job control requires interaction with multiple servers</td>
<td>§ Single point of job submission and job control</td>
</tr>
<tr>
<td>1. Single unified process space</td>
<td>1. Single unified process space</td>
</tr>
<tr>
<td>2. Signals and Std I/O Forwarding</td>
<td>2. Signals and Std I/O Forwarding</td>
</tr>
<tr>
<td>§ Multiple, disparate methods and tools for monitoring</td>
<td>§ Single point of monitoring</td>
</tr>
<tr>
<td>1. Complex to setup and manage</td>
<td>1. Logs, statistics, status automatically forward to the Master node</td>
</tr>
<tr>
<td>2. Does not scale</td>
<td>2. Exceptional ease of use and scalability</td>
</tr>
<tr>
<td>§ Setup users and security on all machines</td>
<td>§ Single point of user/security administration</td>
</tr>
</tbody>
</table>

**Manage Large Pools of Servers as a Single, Consistent, Virtual System**
Some important elements of a cluster system

» Booting and Provisioning
» Process creation, monitoring and control
» Update and consistency model
» Name services
» File Systems
» Physical management
» Workload virtualization
Name services
"Name Service" AKA "Directory Service" is the general concept of a

» mapping a name to a value, or
» providing a list of names.

Concrete examples are

» User names and passwords
» Host names
» Network groups
» IP addresses and Ethernet MAC addresses
Cluster Name Services

Why are cluster nameservices important?

» Simplicity
» Eliminates per-node configuration files
» Automates scaling and updates

Performance

» Avoid the serialization of network name lookups.
» Avoid communicating with a busy server
» Avoid failures and retry cost of server overload
» Avoid the latency of consulting large databases
Opportunities: Why we can do better

- Cluster nodes have predictable names and addresses
- We always issue jobs from Masters
  - One point in space and time where we know the answer
- Clusters have a single set of users per master
  - "Single Sign On" means similar credentials over all machines
- Many table lookups are to never-changing results
- Uncommon queries are rarely performance problems
  - We don't need to get it right, we just can't get it wrong
  - Failback to standard mechanisms if needed
Kickback Daemon

- Head node runs a kickback daemon
- Compute nodes run a kickback proxy service
- `/etc/beowulf/nsswitch.conf` configures the order of the name services used on the compute nodes

**Example**

- Configure head node for LDAP or AD integration as a standard Linux workstation
- Compute nodes can use the head node configuration to resolve user authentication
$ Opportunity: We control IP address assignment

» Assign node IP addresses in node order
» Changes name lookup to addition
» Master: 10.54.0.1
  GigE Switch: 10.54.10.0
  IB Switch: 10.54.11.0
  NFS/Storage: 10.54.30.0
  Nodes: 10.54.50.$node

$ Name format

» Cluster hostnames have the base form n<N>
» Options for admin-defined names and networks

$ Special names for "self" and "master"

» Current machine is ".-2" or "self".
» Master is known as ".-1", “master”, “master0”
Names are reported as password table entry 'pwent'

- BeoNSS reports only the current user
- And a few other e.g. root
- By default, nodes do not run remote access daemons (e.g. sshd, rshd) so users cannot “login” to nodes directly

Cluster jobs do not need to know other users

- Much faster than scanning large lists

Use BProc credentials (full passwd entry) if available

- Otherwise compute from USER, HOME, and SHELL
BeoNSS Netgroups

- Netgroups are used primarily for file server permissions
  - Netgroups are used in /etc/exports for NFS
  - Other file systems have similar security
  - Other uses, such as rexec and rsh, are supported

- The archetype netgroup is "cluster"
  - Alias for "cluster0"
  - Group members are all compute nodes
  - Hostnames are reported as “n0”, “n1” ... “n31"
  - Maximum cluster node count is important

- Use ‘getnetgrent()’ to access this name service
File Systems
Just like in managing processes and administering the cluster, the optimal file system would have a single system image to all nodes of the cluster.

Such file systems exist but have various drawbacks, one in-particular being degraded performance.

Since each node in a Beowulf can have its own disk, making the same files available on each node can be problematic.
File System Architecture

"Diskless Administrative" model
- Reliability of no file system dependencies
- Yet with no constraints on application root or performance penalty

File system agnostic
- Underlying system needs no local or remote file system
- File systems are mounted to support the application

Clean consistency and administrative model
Mounting application file systems is a per-installation configuration

» ClusterWare needs “no” NFS file systems

Administration is done on the master

» File system configuration tables (fstab) are on the master
» Cluster-wide default, with per-node specialization
» Standard format, but in /etc/beowulf/fstab

» Filesystem options:
  • Local
  • Remote (e.g. NFS)

Mounting is a final step of initial provisioning

» Flow-of-control on master: failures are diagnosable, non-fatal
» Handles kernel extension (module) loading
Local - Use storage on each node's disk

- Relatively high performance
- Each node has a potentially different filesystem
- Shared data files must be copied to each node
- No synchronization
- Most useful for temporary/scratch files accessed only by copy of program running on single node
Remote Filesystems

Remote - Share a single disk among all nodes

- Every node sees same filesystem
- Synchronization mechanisms manage changes
- Locking has either high overhead or causes serial blocking
- "Traditional" UNIX approach
- Relatively low performance
- Doesn't scale well; server becomes bottleneck in large systems
- Simplest solution for small clusters, reading/writing small files
Remote Parallel Filesystems

Parallel - Stripe files across multiple disks on multiple nodes

- PVFS - the Parallel Virtual File System
  - Developed at Clemson University
- Relatively high performance
- Each node sees same filesystem
- Works best for I/O intensive applications
- Not a good solution for small files
- Certain slave nodes are designated I/O nodes, local disks used to store pieces of filesystem
Remote Parallel Filesystems

- **Global File System (GFS)**
  - Commercially available cluster file system software
  - Provides global consistency and locking
  - Works best with Fibre Channel

- **Panasas**
  - Hardware file server with software on compute nodes
  - Switched Gigabit Ethernet connects cluster nodes to multiple Panasas blades
  - Single metadata blade controls file layout and access
  - Direct file I/O from cluster node to I/O blade
Physical Management
### Physical Management

§ **ipmitool**
- Intelligent Platform Management Interface (IPMI) is integrated into the base management console (BMC)
- Serial-over-LAN (SOL) can be implemented
- Allows access to hardware such as sensor data or power states
- E.g. `ipmitool -H n$NODE-ipmi -U admin -P admin power {status, on, off}`

§ **bpctl**
- Controls the operational state and ownership of compute nodes
- Examples might be to reboot or power off a node
  - Reboot: `bpctl -S all -R`
  - Power off: `bpctl -S all -P`
- Limit user and group access to run on a particular node or set of nodes
Physical Management

$ beostat
  » Displays raw data from the Beostat system
    • Basic hardware data (CPU's, RAM, network)
    • Load and utilization

$ beosetup
  » GUI to administer a Scyld cluster
  » Shows the dynamic node addition when a new compute node is booted
  » Edit other values which will be correctly entered into the global /etc/beowulf/config file

$ service beowulf {start,stop,restart,reload}
  » O/S level control of beowulf service. Restart will cause all nodes to reboot

$ beochkconfig
  » Controls the node startup scripts in /etc/beowulf/init.d
    • Scripts that are run on the headnode when a compute node boots
    • Modifications to the compute node configuration are done via bpsh commands
  » Sets the execute bit on or off
Physical Management – User level

§ bpstat
  » Unified state, status and statistics used for
    • Scheduling
    • Monitoring
    • Diagnostics
  » Report status of compute nodes and which processes are associated with each
    • ps aux | bpstat –P

§ beostatus
  » Display status information about the cluster
  » X-windows and curses options are available
    • ‘beostatus’ versus ‘beostatus –c’

§ beoconfig
  » Returns keyword values from the Scyld cluster /etc/beowulf/config file for use in scripts if needed
  » e.g. beoconfig interface
Interactive and Serial Jobs
Lessons Learned

What can be improved on this approach?

- Complexity
- Clusters required extensive training to install, configure and use
- Long-term administration and updates were difficult
- Only “static scalability”

Good cluster systems virtualize

- Create a unified view of independent machines
- Single System Image (Illusion)
Cluster Virtualization Architecture Realized

- Minimal in-memory OS with single daemon rapidly deployed in seconds - no disk required
  - Less than 20 seconds

- Virtual, unified process space enables intuitive single sign-on, job submission
  - Effortless job migration to nodes

- Monitor & manage efficiently from the Master
  - Single System Install
  - Single Process Space
  - Shared cache of the cluster state
  - Single point of provisioning
  - Better performance due to lightweight nodes
  - No version skew is inherently more reliable
Run-time provisioning architecture

We "copy" running processes

- Correct: We know the current versions, library linkage order, environment, etc.
- Fast: libraries and executables are cached
- Transfer and cache as versioned whole files
- With dynamic caching we can start with "nothing"
- Dynamically builds a minimal custom installation

Feasible and efficient because "slave nodes" run only applications,
- not full OS install
User directed process migration

- Basic mechanism for running programs on nodes
  - patterned after the rsh and ssh commands

- By default, input and output are redirected from the remote process
  - -N: no IO forwarding
  - -n: /dev/null is stdin
  - -I, -O, -E: redirect from/to a file

- User environment is also forwarded to the compute node

- bpsh [options] nodenumber command [command-args]
  - Compare with ssh –l user hostname uname –a
  - nodenumber can be a single node, a comma separated list of nodes, or –a for all nodes that are up
    - bpsh 1,3,2 hostname
  - No guarantee for order unless –s is specified
One of the key advantages of the Sclyd cluster systems is a unified process space.

Users can submit multiple jobs using bpsh and use standard POSIX job control (i.e. &, bg, fg, kill, etc.)

`ps aux | bpstat -P` will show which processes are associated with which nodes.
bpsh requires a nodenumber to be provided, but how does a user choose which node?

- Assign a given node to a particular user
- Randomly choose a node
- Etc.

Determine which node has the lowest utilization and run there

- Manually using beostat –l to display load averages
- Use beomap to do so automatically
Map concept

Mapping is the assignment of processes to nodes based on current CPU load

» parse data from beostat automatically

» Colon delimited list of nodes

» default mapping policy consists of the following steps:
  • run on nodes that are idle
  • run on CPUs that are idle
  • minimize the load per CPU

bpsh `beomap --nolocal` command

» Benefit that std IO is forwarded and redirected
Caveats

§ What about user data?
  » NFS-mounted home directory can be made available
  » bpcp can be used to stage data on compute nodes locally to improve performance
    • Default directory is the current directory
    • bpcp host1:file1 host2:file2
    • Global copy can be done when combined with bpsh
      – bpsh –a bpcp master:file /tmp

§ Shell scripts
  » Binaries need to be available when running on compute nodes
    • Use the full path, i.e. /bin/grep vs. grep
    • Pre-stage the binary using bpcp
    • NFS mount the directory (e.g. bin)
Distributed Serial Applications

- mpprun and beorun provide you with true "dynamic execution" capabilities, whereas bpsh provides "directed execution" only
  - Specify the number of processors on which to start copies of the program
  - Start one copy on each node in the cluster
  - Start one copy on each CPU in the cluster
  - Force all jobs to run on the master node
  - Prevent any jobs from running on the master node

- Key difference between mpprun and beorun:
  - beorun runs the job on the selected nodes concurrently
  - mpprun runs the job sequentially on one node at a time
beorun vs. mpprun

beorun takes ~1 second to run all 8 threads

```bash
[user@cluster username]$ date;beorun -np 8 sleep 1;date
Mon Aug 18 11:48:30 PDT 2008
```

mpprun takes 8 seconds to run all 8 threads

```bash
[user@cluster username]$ date;mpprun -np 8 sleep 1;date
Mon Aug 18 11:48:46 PDT 2008
Mon Aug 18 11:48:54 PDT 2008
```
Combining with beomap

beorun and mpprun can be used to dynamically select nodes when combined with beomap

» mpprun –map `beomap -np 4 -nolocal` hostname

Can be used to specify a mapping specifically:

» mpprun –map 0:0:0:0 hostname
» mpprun –map 0:1:2:3 hostname
Parallel Jobs and Queuing
Explicitly Parallel Programs

- **Different paradigms exist for parallelizing programs**
  - Shared memory
  - OpenMP
  - Sockets
  - PVM
  - Linda
  - MPI

- **Most distributed parallel programs are now written using MPI**
  - Different options for MPI stacks: MPICH, OpenMPI, HP, Intel, Scali, Verari
  - ClusterWare comes integrated with customized versions of MPICH and OpenMPI
mpicc, mpiCC, mpif77, mpif90 are used to automatically compile code and link in the correct MPI libraries from /usr/lib64/MPICH

- Environment variables can be used to set the compiler:
  - CC, CPP, FC, F90
- Command line options to set the compiler:
  - -cc=, -cxx=, -fc=, -f90=
- GNU, PGI, Intel, and Pathscale compilers are supported
Running MPICH programs

- mpirun is used to launch MPICH programs
- Dynamic allocation can be done when using the –np flag
- Mapping is also supported when using the –map flags
- If Infiniband is installed, the interconnect fabric can be chosen using the machine flag:
  » -machine p4
  » -machine vapi
Additional environment variable control:

- **NP** — The number of processes requested, but not the number of processors. As in the example earlier in this section, `NP=4 ./a.out` will run the MPI program `a.out` with 4 processes.

- **ALL_CPUS** — Set the number of processes to the number of CPUs available to the current user. Similar to the example above, `--all-cpus=1 ./a.out` would run the MPI program `a.out` on all available CPUs.

- **ALL_NODES** — Set the number of processes to the number of nodes available to the current user. Similar to the **ALL_CPUS** variable, but you get a maximum of one CPU per node. This is useful for running a job per node instead of per CPU.

- **ALL_LOCAL** — Run every process on the master node; used for debugging purposes.

- **NO_LOCAL** — Don’t run any processes on the master node.

- **EXCLUDE** — A colon-delimited list of nodes to be avoided during node assignment.

- **BEOWULF_JOB_MAP** — A colon-delimited list of nodes. The first node listed will be the first process (MPI Rank 0) and so on.
Compiling and Running OpenMPI

Set user environment prior to compiling
  » export PATH=/usr/openmpi/bin:${PATH}

mpicc, mpiCC, mpif77, mpif90 are used to automatically compile code and link in the correct MPI libraries from /usr/lib64/OMPI
  » Environment variables can used to set the compiler:
    • OPMI_CC, OPMI_CXX, OPMI_F77, OPMI_FC

Prior to running PATH and LD_LIBRARY_PATH should be set
  » export PATH=/usr/openmpi/bin:${PATH}
  » export LD_LIBRARY_PATH=/usr/lib64/OMPI/{gnu,intel,pathscale,pgi}:${LD_LIBRARY_PATH}
  » /usr/openmpi/bin/mpirun –np 16 a.out

Interconnect can be selected at runtime
  » -mca btl openib,tcp,sm,self
  » -mca btl udapl,tcp,sm,self
Only a brief introduction is provided here for MPI. Many other in-depth tutorials are available on the web and in published sources.

- http://www.llnl.gov/computing/tutorials/mpi/

Paradigms for writing parallel programs depend upon the application

- SIMD (single-instruction multiple-data)
- MIMD (multiple-instruction multiple-data)
- MISD (multiple-instruction single-data)

SIMD will be presented here as it is a commonly used template

- A single application source is compiled to perform operations on different sets of data
- The data is read by the different threads or passed between threads via messages (hence MPI = message passing interface)
  - Contrast this with shared memory or OpenMP where data is locally via memory
  - Optimizations in the MPI implementation can perform localhost optimization; however, the program is still written using a message passing construct

MPI specification has many functions; however most MPI programs can be written with only a small subset
§  cpi.c calculates $\pi$ using MPI in C

compute $\pi$ by integrating $f(x) = \frac{4}{1 + x^2}$

```c
#include "mpi.h"
#include <stdio.h>
#include <math.h>

double f( double );
double f( double a ) {
    return (4.0 / (1.0 + a*a));
}

int main( int argc, char *argv[] )
{
    int done = 0, n, myid, numprocs, i;
    double PI25DT = 3.141592653589793238462643;
    double mypi, pi, h, sum, x;
    double startwtime = 0.0, endwtime;
    int namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    MPI_Get_processor_name(processor_name,&namelen);

    printf(stderr,"Process %d on %s\n", myid, processor_name);

    n = 0;

    while (!done)
    {
        if (myid == 0)
        {
            /*
             * printf("Enter the number of intervals: (0
             * quits) ");
             * scanf("%d",&n);
             */
            if (n==0) n=100; else n=0;
            startwtime = MPI_Wtime();
            MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
            if (n == 0)
            {
                done = 1;
            }
            else
            {
                h = 1.0 / (double) n;
                sum = 0.0;
                for (i = myid + 1; i <= n; i += numprocs)
                {
                    x = h * ((double)i - 0.5);
                    sum += f(x);
                }
                mypi = h * sum;
                MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE,
                        MPI_SUM, 0, MPI_COMM_WORLD);
            }
            if (myid == 0)
            {
                printf("pi is approximately %.16f, Error
                is %.16f\n", pi, fabs(pi - PI25DT));
                printf("wall clock time = %.16f", endwtime-startwtime);
            }
        }
    }
    MPI_Finalize();
    return 0;
}
```

System include file which defines the MPI functions

```
double f( double );
double f( double a ) {
    return (4.0 / (1.0 + a*a));
}
```
cpi.c calculates π using MPI in C

calculate π by integrating f(x) = 4/(1 + x^2)

```
#include "mpi.h"
#include <stdio.h>
#include <math.h>
double f(double a) {
    return (4.0 / (1.0 + a*a));
}

int main(int argc, char *argv[]) {
    int done = 0, n, myid, numprocs, i;
double PI25DT = 3.141592653589793238462643;
    double mypi, pi, h, sum, x;
    double starttime = 0.0, endtime;
    int namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    MPI_Get_processor_name(processor_name,&namelen);
    fprintf(stderr,"Process %d on %s\n", 
        myid, processor_name);
    n = 0;
    while (!done)
    {
        if (myid == 0) {
            printf("Enter the number of intervals: (0 quits) ");
            scanf("%d",&n);
            if (n==0) n=100; else n=0;
            starttime = MPI_Wtime();
        }
        MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
        if (n == 0)
            done = 1;
        else {
            h = 1.0 / (double) n;
            sum = 0.0;
            for (i = myid + 1; i <= n; i += numprocs)
            {
                x = h * ((double)i - 0.5);
                sum += f(x);
            }
            mypi = h * sum;
            MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE,
                MPI_SUM, 0, MPI_COMM_WORLD);
            if (myid == 0)
            {
                printf("pi is approximately %.16f, Error 
                    is %.16f\n", 
                    pi, fabs(pi - PI25DT));
                endtime = MPI_Wtime();
                printf("wall clock time = %f\n", 
                    endtime-starttime);
            }
        }
    }
    MPI_Finalize();
    return 0;
}
```
cpi.c calculates π using MPI in C

compute pi by integrating
f(x) = 4/(1 + x**2)

#include "mpi.h"
#include <stdio.h>
#include <math.h>
double f( double a )
{  return (4.0 / (1.0 + a*a)); }

int main( int argc, char *argv[])
{
  int done = 0, n, myid, numprocs, i;
  double PI25DT = 3.141592653589793238462643;
  double mypi, pi, h, sum, x;
  double startwtime = 0.0, endwtime;
  int nameilen;
  char processor_name[MPI_MAX_PROCESSOR_NAME];

  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
  MPI_Comm_rank(MPI_COMM_WORLD,&myid);
  MPI_Get_processor_name(processor_name,&nameilen);

  fprintf(stderr,"Process %d on %s\n",
          myid, processor_name);

  n = 0;

while (!done)
{
  if (myid == 0)
  {
    /*
    printf("Enter the number of intervals: (0
    quits) ");
    scanf("%d",&n);
    */
    if (n==0) n=100; else n=0;
    startwtime = MPI_Wtime();
    MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
    if (n == 0)
      done = 1;
    else
    {
      h   = 1.0 / (double) n;
      sum = 0.0;
      for (i = myid + 1; i <= n; i += numprocs)
      {  x = h * ((double)i - 0.5);
          sum += f(x);
      }
      mypi   = h * sum;
      MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE,
                  MPI_SUM, 0, MPI_COMM_WORLD);
      if (myid == 0)
      {
        printf("pi is approximately %.16f, Error
        is %.16f\n", pi, fabs(pi - PI25DT));
        endwtime = MPI_Wtime();
        printf("wall clock time = %f\n",
                endwtime-startwtime);
      }
    }
  }
  MPI_Finalize();
  return 0;
}
Example MPI Source

§ cpi.c calculates π using MPI in C

compute π by integrating f(x) = 4/(1 + x^2)

```c
#include "mpi.h"
#include <stdio.h>
#include <math.h>

double f( double );

double f( double a ) {
    return (4.0 / (1.0 + a*a));
}

int main( int argc, char *argv[] ) {
    int done = 0, n, myid, numprocs, i;
    double PI25DT = 3.141592653589793238462643;
    double mypi, pi, h, sum, x;
    double startwtime = 0.0, endwtime;
    int namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    MPI_Get_processor_name(processor_name,&namelen);

    fprintf(stderr,"Process %d on %s
", myid, processor_name);

    n = 0;

    while (!done) {
        if (myid == 0) {
            /*
             * printf("Enter the number of intervals: (0 quits) ");
             * scanf("%d",&n);
             */
            if (n==0) n=100; else n=0;
            startwtime = MPI_Wtime();
            MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
            if (n == 0) done = 1;
            else {
                h = 1.0 / (double) n;
                sum = 0.0;
                for (i = myid + 1; i <= n; i += numprocs) {
                    x = h * ((double)i - 0.5);
                    sum += f(x);
                }
                mypi = h * sum;
                MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
                if (myid == 0) {
                    printf("pi is approximately %.16f, Error is %.16f\n", pi, fabs(pi - PI25DT));
                    endwtime = MPI_Wtime();
                    printf("wall clock time = %.16f\n",
                            endwtime-startwtime);
                }
            }
        }
        MPI_Finalize();
        return 0;
    }
}
```

Determines the rank of the calling process in the communicator
cpi.c calculates \( \pi \) using MPI in C

compute \( \pi \) by integrating \( f(x) = \frac{4}{1 + x^2} \)

```c
#include "mpi.h"
#include <stdio.h>
#include <math.h>

double f( double a )
{
    return (4.0 / (1.0 + a*a));
}

int main( int argc, char *argv[] )
{
    int done = 0, n, myid, numprocs, i;
    double PI25DT = 3.141592653589793238462643;
    double mypi, pi, h, sum, x;
    double startwtime = 0.0, endwtime;
    int namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    MPI_Get_processor_name(processor_name,&namelen);

    fprintf(stderr,"Process %d on %s\n", myid, processor_name);

    n = 0;
    while (!done)
    {
        if (myid == 0)
        {
            /*
            *    printf("Enter the number of intervals: (0 quits) ");
            *    scanf("%d",&n);
            */
            if (n==0) n=100; else n=0;
            startwtime = MPI_Wtime();
            MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
            if (n == 0)
                done = 1;
            else
            {
                h   = 1.0 / (double) n;
                sum = 0.0;
                for (i = myid + 1; i <= n; i += numprocs)
                {
                    x = h * ((double)i - 0.5);
                    sum += f(x);
                }
                mypi = h * sum;
                MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

                if (myid == 0)
                {
                    printf("pi is approximately %.16f, Error is %.16f\n", pi, fabs(pi - PI25DT));
                    endwtime = MPI_Wtime();
                    printf("wall clock time = %f\n", endwtime-startwtime);
                }
            }
        }
        MPI_Finalize();
    return 0;
}
```

Gets the name of the processor

`cpi.c` calculates \( \pi \) using MPI in C. It integrates \( f(x) = \frac{4}{1 + x^2} \) to compute \( \pi \). The program uses MPI to distribute the integration over multiple processors, with each processor computing a portion of the integral. The result is then reduced to a single value using MPI. The program prompts the user for the number of intervals and prints the computed value of \( \pi \) along with the error compared to the known value of \( \pi \).
§ cpi.c calculates $\pi$ using MPI in C

compute $\pi$ by integrating $f(x) = \frac{4}{1 + x^2}$

```c
#include "mpi.h"
#include <stdio.h>
#include <math.h>

double f( double );
double f( double a ) {
    return (4.0 / (1.0 + a*a));
}

int main( int argc, char *argv[] ) {
    int done = 0, n, myid, numprocs, i;
    double PI25DT = 3.141592653589793238462643;
    double mypi, pi, h, sum, x;
    double startwtime = 0.0, endwtime;
    int namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    MPI_Get_processor_name(processor_name,&namelen);

    printf(stderr,"Process %d on %s\n", myid, processor_name);

    n = 0;
    while (!done)
    {
        if (myid == 0)
        {
            /*
                printf("Enter the number of intervals: (0
                quits) ");
                scanf("%d",&n);
            */
            if (n==0) n=100; else n=0;
            startwtime = MPI_Wtime();
            MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
            if (n == 0)
                done = 1;
            else
            {
                h = 1.0 / (double) n;
                sum = 0.0;
                for (i = myid + 1; i <= n; i += numprocs)
                {
                    x = h * ((double)i - 0.5);
                    sum += f(x);
                }
                mypi = h * sum;
                MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE,
                                MPI_SUM, 0, MPI_COMM_WORLD);
                if (myid == 0)
                {
                    printf("pi is approximately %.16f, Error
                    is %.16f\n", pi, fabs(pi - PI25DT));
                    endwtime = MPI_Wtime();
                    printf("wall clock time = %.16f\n",
                            endwtime-startwtime);
                }
            }
        }
        MPI_Finalize();
        return 0;
    }
```

Differentiate actions based on rank. Only "master" performs this action.
Example MPI Source

§ cpi.c calculates $\pi$ using MPI in C
compute $\pi$ by integrating $f(x) = \frac{4}{1 + x^2}$

```
#include "mpi.h"
#include <stdio.h>
#include <math.h>

double f( double a )
{
    return (4.0 / (1.0 + a*a));
}

int main( int argc, char *argv[])
{
    int done = 0, n, myid, numprocs, i;
    double PI25DT = 3.141592653589793238462643;
    double mypi, pi, h, sum, x;
    double startwtime = 0.0, endwtime;
    int namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    MPI_Get_processor_name(processor_name,&namelen);

    fprintf(stderr,"Process %d on %s
", myid, processor_name);
    n = 0;

    while (!done)
    {
        if (myid == 0)
        {
            /*
             * printf("Enter the number of intervals: (0 quits) ");
             */
            scanf("%d",&n);
            if (n==0) n=100; else n=0;
            startwtime = MPI_Wtime();
            MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
            if (n == 0)
            done = 1;
            else
            {
                h = 1.0 / (double) n;
                sum = 0.0;
                for (i = myid + 1; i <= n; i += numprocs)
                {
                    x = h * ((double)i - 0.5);
                    sum += f(x);
                }
                mypi = h * sum;
                MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
                if (myid == 0)
                {
                    printf("pi is approximately %.16f, Error is %.16f\n", pi, fabs(pi - PI25DT));
                    endwtime = MPI_Wtime();
                    printf("wall clock time = %f\n", endwtime-startwtime);
                }
            }
            MPI_Finalize();
        return 0;
    }
```
cpi.c calculates π using MPI in C

compute π by integrating
f(x) = 4/(1 + x^2)

#include "mpi.h"
#include <stdio.h>
#include <math.h>

double f( double );
double f( double a ) {
    return (4.0 / (1.0 + a*a));
}

int main( int argc, char *argv[] )
{
    int done = 0, n, myid, numprocs, i;
    double PI25DT = 3.141592653589793238462643;
    double mypi, pi, h, sum, x;
    double startwtime = 0.0, endwtime;
    int namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    MPI_Get_processor_name(processor_name,&namelen);

    fprintf(stderr,"Process %d on %s\n", myid, processor_name);

    n = 0;

    while (!done)
    {
        if (myid == 0)
        {
            /*
            * printf("Enter the number of intervals: (0
            * quits) ");
            * scanf("%d",&n);
            */
            if (n==0) n=100; else n=0;
            startwtime = MPI_Wtime();
            MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
            if (n == 0)
                done = 1;
            else
            {
                h = 1.0 / (double) n;
                sum = 0.0;
                for (i = myid + 1; i <= n; i += numprocs)
                {
                    x = h * ((double)i - 0.5);
                    sum += f(x);
                }
                mypi = h * sum;
                MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
                if (myid == 0)
                {
                    printf("pi is approximately %.16f, Error is %.16f\n", pi, fabs(pi - PI25DT));
                    endwtime = MPI_Wtime();
                    printf("wall clock time = %f\n", endwtime-startwtime);
                }
            }
        }
        MPI_Finalize();
        return 0;
    }
Example MPI Source

§ cpi.c calculates π using MPI in C
compute π by integrating f(x) = 4/(1 + x^2)

#include "mpi.h"
#include <stdio.h>
#include <math.h>

doUBLE f( double );
doUBLE f( double a ) {
   return (4.0 / (1.0 + a*a));
}

int main( int argc, char *argv[] )
{
   int done = 0, n, myid, numprocs, i;
   double PI25DT = 3.141592653589793238462643;
   double mypi, pi, h, sum, x;
   double startwtime = 0.0, endwtime;
   int namelen;
   char processor_name[MPI_MAX_PROCESSOR_NAME];

   MPI_Init(&argc,&argv);
   MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
   MPI_Comm_rank(MPI_COMM_WORLD,&myid);
   MPI_Get_processor_name(processor_name,&namelen);

   fprintf(stderr,"Process %d on %s\n", myid, processor_name);

   n = 0;

   while (!done)
   {
      if (myid == 0)
      {
         /*
         printf("Enter the number of intervals: (0 quits) ");
         scanf("%d",&n);
         */
         if (n==0) n=100; else n=0;
         startwtime = MPI_Wtime();
      }
      MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
      if (n == 0)
      {
         done = 1;
      }
      else
      {
         h   = 1.0 / (double) n;
         sum = 0.0;
         for (i = myid + 1; i <= n; i += numprocs)
         {
            x = h * ((double)i - 0.5);
            sum += f(x);
         }
         mypi = h * sum;
         MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
         if (myid == 0)
         {
            printf("pi is approximately %.16f, Error is %.16f\n", pi, fabs(pi - PI25DT));
            endwtime = MPI_Wtime();
            printf("wall clock time = %f\n", endwtime-startwtime);
         }
      }
   }
   MPI_Finalize();
   return 0;

Each worker does this loop and increments the counter by the number of processors (versus dividing the range -> possible off-by-one error)
Example MPI Source

§ cpi.c calculates π using MPI in C

calculate pi by integrating
\[ f(x) = \frac{4}{1 + x^2} \]

```c
#include "mpi.h"
#include <stdio.h>
#include <math.h>

double f( double a )
{
    return (4.0 / (1.0 + a*a));
}

int main( int argc, char *argv[] )
{
    double PI25DT = 3.141592653589793238462643;
    double mypi, pi, h, sum, x;
    double startwtime = 0.0, endwtime = 0.0;
    int namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    MPI_Get_processor_name(processor_name,&namelen);

    fprintf(stderr,"Process %d on %s\n", myid, processor_name);
    n = 0;

    while (!done)
    {
        if (myid == 0)
        {
            /*
                printf("Enter the number of intervals: (0 quits) ");
                scanf("%d", &n);
            */
            if (n==0) n=100; else n=0;
            startwtime = MPI_Wtime();
        }
        MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
        if (n == 0)
            done = 1;
        else
        {
            h = 1.0 / (double) n;
            sum = 0.0;
            for (i = myid + 1; i <= n; i += numprocs)
            {
                x = h * ((double)i - 0.5);
                sum += f(x);
            }
            mypi = h * sum;
            MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
        }
    }

    if (myid == 0)
    {
        printf("pi is approximately %.16f, Error is %.16f\n", pi, fabs(pi - PI25DT));
        endwtime = MPI_Wtime();
        printf("wall clock time = %f\n", endwtime-startwtime);
    }

    MPI_Finalize();
    return 0;
}
```

Does MPI_SUM function on “1” MPI_DOUBLE at &mypi on all workers in MPI_COMM_WORLD to a single value at &pi on rank “0”
cpi.c calculates $\pi$ using MPI in C

compute $\pi$ by integrating $f(x) = 4/(1 + x^2)$

```c
#include "mpi.h"
#include <stdio.h>
#include <math.h>

double f( double a )
{
    return (4.0 / (1.0 + a*a));
}

int main( int argc, char *argv[] )
{
    int done = 0, n, myid, numprocs, i;
    double PI25DT = 3.141592653589793238462643;
    double mypi, pi, h, sum, x;
    double startwtime = 0.0, endwtime;
    int namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    MPI_Get_processor_name(processor_name,&namelen);

    fprintf(stderr,"Process %d on %s\n", myid, processor_name);

    n = 0;
    while (!done)
    {
        if (myid == 0) {
            /*
               printf("Enter the number of intervals: (0
               quits) ");
               scanf("%d",&n);
               */
            if (n==0) n=100; else n=0;
            startwtime = MPI_Wtime();
            MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
            if (n == 0) done = 1;
            else {
                h   = 1.0 / (double) n;
                sum = 0.0;
                for (i = myid + 1; i <= n; i += numprocs){
                    x = h * ((double)i - 0.5);
                    sum += f(x);
                }
                mypi = h * sum;
                MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
                if (myid == 0) {
                    printf("pi is approximately %.16f, Error
is %.16f\n", pi, fabs(pi - PI25DT));
                    endwtime = MPI_Wtime();
                    printf("wall clock time = %f\n", endwtime-startwtime);
                }
            }
        }
    }
    MPI_Finalize();
    return 0;
}
```

Only rank "0" outputs the value of $\pi$
cpi.c calculates $\pi$ using MPI in C

compute $\pi$ by integrating $f(x) = 4/(1 + x^2)$

#include "mpi.h"
#include <stdio.h>
#include <math.h>

double f( double a ) {
    return (4.0 / (1.0 + a*a));
}

int main( int argc, char *argv[]) {
    int done = 0, n, myid, numprocs, i;

double PI25DT = 3.141592653589793238462643;
double mypi, pi, h, sum, x;
double startwtime = 0.0, endwtime;
int namelen;
char processor_name[MPI_MAX_PROCESSOR_NAME];

MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
MPI_Comm_rank(MPI_COMM_WORLD,&myid);
MPI_Get_processor_name(processor_name,&namelen);

fprintf(stderr,"Process %d on %s\n",
    myid, processor_name);

n = 0;

while (!done) {
    if (myid == 0) {
        /*
            printf("Enter the number of intervals: (0 quits) ");
            scanf("%d",&n);
        */
        if (n==0) n=100; else n=0;
        startwtime = MPI_Wtime();
    }

    MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
    if (n == 0)
        done = 1;
    else {
        h   = 1.0 / (double) n;
        sum = 0.0;
        for (i = myid + 1; i <= n; i += numprocs) {
            x = h * ((double)i - 0.5);
            sum += f(x);
        }
        mypi = h * sum;
        MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE,
            MPI_SUM, 0, MPI_COMM_WORLD);
        if (myid == 0) {
            printf("pi is approximately %.16f, Error
            is %.16f\n", 
                pi, fabs(pi - PI25DT));
            endwtime = MPI_Wtime();
            printf("wall clock time = %.16f\n",
                endwtime-startwtime);
        }
    }
}

MPI_Finalize();
return 0;

Terminate MPI execution environment
Other Common MPI Functions

- **MPI_Send, MPI_Recv**
  - Blocking send and receive between two specific ranks

- **MPI_Isend, MPI_Irecv**
  - Non-blocking send and receive between two specific ranks

- Man pages exist for the MPI functions

- Poorly written programs can suffer from poor communication efficiency (e.g. stair-step) or lost data if the system buffer fills before a blocking send or receive is initiated to correspond with a non-blocking receive or send

- Care should be used when creating temporary files as multiple threads may be running on the same host overwriting the same temporary file (include rank in file name in a unique temporary directory per simulation)
How are resources allocated among multiple users and/or groups?

- Statically by using bpctl user and group permissions
- ClusterWare supports a variety of queuing packages
  - TaskMaster (advanced MOAB policy based scheduler integrated in ClusterWare)
  - Torque
  - SGE
A number of queuing systems have been developed (e.g. NQS, LSF, OpenPBS, PBSPro, Torque)

» PBSPro is a commercial product

» OpenPBS was an open source component of the product (compare with RedHat versus Fedora Core)

» OpenPBS had many contributions from the community, but the vendor ceased development

» TORQUE (Terascale Open-source Resource and QUEue manager) was forked from the OpenPBS project

All of the PBS-type schedulers consist of three components:

» pbs_server – keeps tracks of jobs in the queue and resources available to run jobs

» pbs_sched – scheduler that analyzes information from the pbs_server and returns with which jobs should be run

» pbs_mom – communicates with the pbs_server about what resources are available and used, ALSO spawns the job submission scripts
Scheduler Improvements

§ Default scheduler in Torque is pbs_sched
  » Essentially a FIFO scheduler. Some capabilities exist for more complex policies such as priority based scheduling
  » Queue based scheduling where multiple queues are defined for different job profiles

§ MAUI was an improvement on the default pbs_sched
  » Rather than use queues, all jobs are put into a stack. Based on administrator defined variables (classes) and priority functions, jobs are sorted in the heap. The next eligible job is at the top of the heap.
  » Utilizes Torque’s pbs_server and pbs_mom components

§ Cluster Resources has improved and commercialized MAUI as the MOAB product
  » More functionality and administration and user interfaces

§ Penguin Computing has licensed MOAB and integrated it with ClusterWare producing TaskMaster
Different integration schemes for TaskMaster

- pbs_mom on compute nodes only (default TaskMaster >= 2.1)
To submit a job:

» qsub script.sh

  • Example script.sh:
    
    ```bash
    #!/bin/sh
    #PBS -j oe
    #PBS -l nodes=4
    cd $PBS_O_WORKDIR
    hostname
    ```

  • qsub does not accept arguments for script.sh. All executable arguments must be included in the script itself

» Administrators can create a ‘qapp’ script that takes user arguments, creates script.sh with the user arguments embedded, and runs ‘qsub script.sh’
Other commands

- qstat – Status of queue server and jobs
- qdel – Remove a job from the queue
- qhold, qrls – Hold and release a job in the queue
- qmgr – Administrator command to configure pbs_server
- /var/spool/torque/server_name: should match hostname of the head node
- /var/spool/torque/mom_priv/config: file to configure pbs_mom
  - ‘$usecp */home */home’ indicates that pbs_mom should use ‘cp’ rather than ‘rcp’ or ‘scp’ to relocate the stdout and stderr files at the end of execution
- pbsnodes – Administrator command to monitor the status of the resources
- qalter – Administrator command to modify the parameters of a particular job (e.g. requested time)
Interacting with TaskMaster

Because TaskMaster uses the MOAB scheduler with Torque pbs_server and pbs_mom components, all of the Torque commands are still valid

- qsub will submit a job to Torque, MOAB then polls pbs_server to detect new jobs
- msub will submit a job to Moab which then pushes the job to pbs_server

Other TaskMaster commands

- qstat -> showq
- qdel, qhold, qrls -> mjobctl
- pbsnodes -> showstate
- qmgr -> mschedctl, mdiag
- Configuration in /opt/moab/moab.cfg
Other options to qsub

Options that can be included in a script (with the #PBS directive) or on the qsub command line

» Join output and error files: #PBS -j oe
» Request resources: #PBS -l nodes=2:ppn=2
» Request walltime: #PBS -l walltime=24:00:00
» Define a job name: #PBS -N jobname
» Send mail at jobs events: #PBS -m be
» Assign job to an account: #PBS -A account

To start an interactive queue job use:
» qsub -I
**qapp script:**

» Be careful about escaping special characters in the redirect section (\$, \', \\)

```bash
#!/bin/bash
# Usage: qapp arg1 arg2
dbg=0
opt1="\$1"
opt2="\$2"

if [[ \"\$opt2\" == "" ]] ; then
    echo "Not enough arguments"
    exit 1
fi

cat > app.sh << EOF
#!/bin/bash
#PBS –j oe
#PBS –l nodes=1
cd \$PBS_O_WORKDIR
app $opt1 $opt2
EOF

if [[ \"\$dbg\" < 1 ]] ; then
    qsub app.sh
fi

if [[ \"\$dbg\" == 0 ]] ; then
    /bin/rm -f app.sh
fi
```
$ Using local scratch:

```bash
#!/bin/bash

#PBS -j oe
#PBS -l nodes=1

cd $PBS_O_WORKDIR

tmpdir="/scratch/$USER/$PBS_JOBID"
/bin/mkdir -p $tmpdir
/rsync -a ./ $tmpdir

cd $tmpdir

$path/to/app $1 $2

cd $PBS_O_WORKDIR

/rsync -a $tmpdir/ .

/bin/rm -fr $tmpdir
```
Using local scratch for MPICH parallel jobs:

```
#!/bin/bash

#PBS -j oe
#PBS -l nodes=2:ppn=8

cd $PBS_O_WORKDIR

tmpdir="/scratch/$USER/$PBS_JOBID"
/usr/bin/pbsdsh -u "/bin/mkdir -p $tmpdir"

/usr/bin/pbsdsh -u bash -c "cd $PBS_O_WORKDIR ; rsync -a . / $tmpdir"

cd $tmpdir

mpirun -machine vapi $path/to/app $1 $2

cd $PBS_O_WORKDIR

/usr/bin/pbsdsh -u "rsync -a $tmpdir/ $PBS_O_WORKDIR"

/usr/bin/pbsdsh -u "/bin/rm -fr $tmpdir"
```
Using local scratch for OpenMPI parallel jobs:

```bash
#!/bin/bash

#PBS -j oe
#PBS -l nodes=2:ppn=8

export PATH=/usr/openmpi/bin:{$PATH}
export
    LD_LIBRARY_PATH=/usr/lib64/OMPI/gnu:{$LD_LIBRARY_PATH}

cd $PBS_O_WORKDIR

tmpdir="/scratch/$USER/$PBS_JOBID" /usr/bin/pbsdsh -u "/bin/mkdir -p $tmpdir"

/usr/bin/pbsdsh -u bash -c "cd $PBS_O_WORKDIR ; rsync -a ./ $tmpdir"

cd $tmpdir

/usr/openmpi/bin/mpirun -np `cat $PBS_NODEFILE | wc -l` -mca btl openib,sm,self $path/to/app $1 $2

cd $PBS_O_WORKDIR

/usr/bin/pbsdsh -u "rsync -a $tmpdir/ $PBS_O_WORKDIR"
/usr/bin/pbsdsh -u "/bin/rm -fr $tmpdir"
```
Other considerations

A queue script need not be a single command

- Multiple steps can be performed from a single script
  - Guaranteed resources
  - Jobs should typically be a minimum of 2 minutes
- Pre-processing and post-processing can be done from the same script using the local scratch space
- If configured, it is possible to submit additional jobs from a running queued job

To remove multiple jobs from the queue:

- qstat | grep " [RQ] " | awk '{print $1}' | xargs qdel
Other Resources
PDF Manuals (/usr/share/doc/PDF)

» Administrator’s Guide

Adobe Acrobat Document

» Programmer’s Guide

Adobe Acrobat Document

» User’s Guide

Adobe Acrobat Document

» Reference Guides

Adobe Acrobat Document
Online Documentation

- **man pages exist for most commands**
  - man command

- **HTML documentation is available in a web browser (/usr/share/doc/HTML)**
  - Need to start httpd service to access remotely

- **Penguin Computing Masterlink**
    - Login ID can be generated using ClusterID

- **TaskMaster/Torque**
  - If httpd is started, [http://cluster/taskmaster-doc](http://cluster/taskmaster-doc)
  - Also available at [http://www.clusterresources.com](http://www.clusterresources.com) at the “Support” link
Support Contacts

§ Penguin Computing Technical Support
  » Can help with ClusterWare configuration and basic system questions
  » Provided as part of the software support contract
    • 1-888-PENGUIN
    • support@penguincomputing.com

§ Penguin Computing Professional Services
  » Higher level application specific support and optimization based on a pre-scoped Statement of Work
  » Other custom consulting