Beowulf Clusters for Evolutionary Computation

Arun Khosla
Department of Electronics and Communication Engineering
National Institute of Technology
Jalandhar – 144 011. INDIA
khoslaak@nitj.ac.in

Pramod Kumar Singh
Department of Computer Science and Engineering
Indian Institute of Technology
Kharagpur – 721 302. INDIA
pksingh@cse.iitkgp.ernet.in

Diptanu Gon Choudhury
Department of Electronics and Communication Engineering
National Institute of Technology
Jalandhar – 144 011. INDIA
admin.nitjece@gmail.com

Acknowledgements

- Gurpreet Singh Dhami
- Gaurav Dixit
- Jagmohan Singh

(ST Microelectronics, Noida. India)

Tutorial Objective

To focus on the details of assembling, configuring, using and managing a cluster especially for those who have never done it before

Expected Background of Participants

There are no pre-requisites for this tutorial, but some familiarity with the Linux operating system shall be useful.

GECCO'07, July 7–11, 2007, London, England, United Kingdom.
ACM 978-1-59593-698-1/07/0007.

Expected Enrollment

Useful and relevant for the researchers working in the domain of evolutionary computing

Tutorial Detailed Outline

- Setting up Beowulf Cluster
- Cluster Building Blocks
  - Off the shelf components
  - Interconnects
  - Why Linux for Clusters?
  - Cluster Deployment
  - Cluster Packages

Tutorial Detailed Outline (Contd.)

- Cluster Benchmarking
- Parallel Programming for Clusters
  - Writing Parallel Programs for Clusters
- Parallel Programming with MPI (Message Passing Interface)
- Cluster Management
- Live Demonstration

What is a Beowulf Cluster?

- Introduction
- What is Cluster Computing?
- Why use a Cluster?
  - High Availability
  - High Performance Computing (HPC)
- Why use a Cluster for Evolutionary Computation?

Some Architectures for Parallel and Distributed Evolutionary Computations

- Master-Slave
- Island-Model
- Hybrid Approach

ROCKS CLUSTER

This software developed by the Rocks Cluster Group at the San Diego Supercomputer Center at the University of California, San Diego and its contributors.

Rocks Identity

- System to build and manage Linux Clusters
  - o General Linux maintenance system for N nodes
  - o Happens to be good for clusters
- Free
- Mature
- High Performance
  - o Designed for scientific workloads

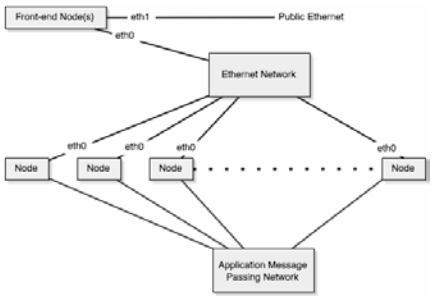
Rocks Mission

- Make Clusters Easy
- Most cluster projects assume a system admin will help build the cluster.
- Build a cluster without assuming CS knowledge
  - o Simple idea, complex ramifications
  - o Clusters for Scientists
- Results in a very robust system that is insulated from human mistakes

Rocks Cluster Distribution

- Fully-automated cluster-aware distribution
- Software Packages
  - Full Red Hat Linux distribution
  - De-facto standard cluster packages
  - Rocks packages
  - Rocks community package
- System Configuration

Rocks Hardware Architecture



Source: www.rocksclusters.org

Processors Supported

- x86 (Pentium/Athlon)
- Opteron
- Itanium

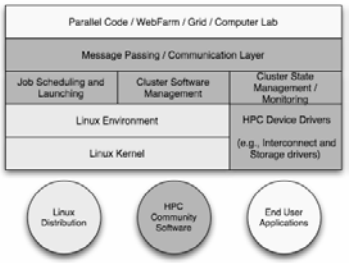
Interconnects Supported

- Ethernet
- Myrinet

Minimum Hardware Requirements

- Frontend:
  - 2 Ethernet connections
  - 18 GB disk drive
  - 512 MB memory
- Compute:
  - 1 Ethernet connection
  - 18 GB disk
  - 512 MB Memory
- Ethernet switches

Cluster Software Stack



Source: www.rocksclusters.org

Rocks 'Rolls'

- Rolls are containers for software packages and the configuration scripts for the packages
- Rolls dissect a monolithic distribution

Login to Frontend

- Create ssh public/private key
  - Ask for 'passphrase'
  - These keys are used to securely login into compute nodes without having to enter a password each time you login to a compute node
- Execute 'insert-ethers'
  - This utility listens for new compute nodes

Insert-ethers

- Used to integrate “appliances” into the cluster
  - We’ll choose “Compute”

Source: www.rocksclusters.org

Boot a Compute Node in Installation Mode

- Instruct the node to network boot
  - Network boot forces the compute node to run the PXE protocol (Pre-eXecution Environment)
- Also can use the Rocks Base CD
  - If no CD and no PXE-enabled NIC, can use a boot floppy built from ‘Etherboot’ (http://www.rom-o-matic.net)

Insert-ethers Discovers the Node

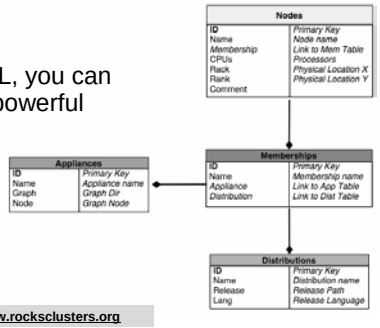
Source: www.rocksclusters.org

eKV
Ethernet Keyboard and Video

- Monitor your compute node installation over the ethernet network
  - No KVM required!
- Execute: ‘ssh compute-0-0’

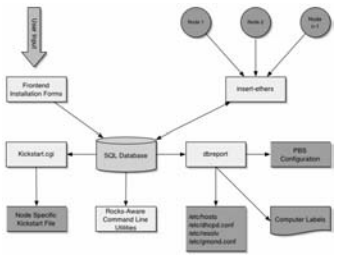
Node Info Stored In A MySQL Database

- If you know SQL, you can execute some powerful commands



Source: www.rocksclusters.org

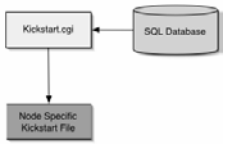
Cluster Database



Source: www.rocksclusters.org

Kickstart

- Red Hat's Kickstart
  - Monolithic flat ASCII file
  - No macro language
  - Requires forking based on site information and node type.
- Rocks XML Kickstart
  - Decompose a kickstart file into nodes and a graph
    - Graph specifies OO framework
    - Each node specifies a service and its configuration
  - Macros and SQL for site configuration
  - Driven from web cgi script



Source: www.rocksclusters.org

Extra insert-ethers Usage

- If you have more than one cabinet:

```
insert-ethers --cabinet=1
```

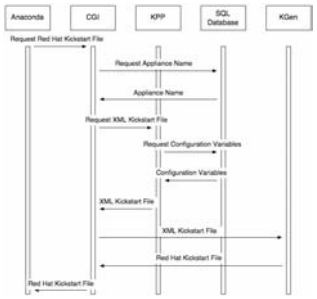
- To replace a dead node:

```
insert-ethers --replace=compute-0-0
```

- To rebuild and restart relevant services:

```
insert-ethers --update
```

Installation Timeline



MPI

The Message Passing Model

- Parallel programs consist of cooperating processes, each with its own memory
- Processes send data to one another as messages
- Messages may have *tags* that may be used to sort messages
- Messages may be received in any order

What is MPI?

- A *message-passing library specification*
  - extended message-passing model
  - not a language or compiler specification
  - not a specific implementation or product
- For parallel computers, clusters, and heterogeneous networks
- Full-featured
- Designed to provide access to advanced parallel hardware for
  - end users
  - library writers
  - tool developers

Why Use MPI?

- MPI provides a powerful, efficient, and *portable* way to express parallel programs
- MPI was explicitly designed to enable libraries…
- … which may eliminate the need for many users to learn (much of) MPI

Why was MPI needed?

- Software crisis in parallel computing
  - Each vendor provided their own, different interface
  - No "critical mass" of users
- Enabling libraries (code sharing)
- Turnkey parallel applications
  - CFD, pharmaceutical design, etc.

Quick Tour of MPI

- Point-to-point
- Collective
- Process groups and topology
- Profiling
- Other

Point-to-point

- Send/Receive

MPI_Send(A, 10, MPI_DOUBLE, 1, …)

MPI_Recv(B, 20, MPI_DOUBLE, 0, …)

- Datatype
  - Basic for heterogeneity
  - Derived for non-contiguous
- Contexts
  - Message safety for libraries
- Buffering
  - Robustness and correctness

Collective

- Process groups
  - Collections of cooperating processes
  - Hierarchical algorithms need nested collections
- Categories
  - Communication: Broadcast data
  - Computation: Global sum
  - Synchronization: Barrier

New MPI-2 Features

- Remote Memory
- Parallel I/O
- Dynamic Process
- Threads

MPI Implementations

- MPICH (Argonne National Lab)
- LAM-MPI (Ohio, Notre Dame, Bloomington)
- Cray, IBM, SGI
- MPI-FM (Illinois)
- MPI / Pro (MPI Software Tech.)
- Sca MPI (Scali AS)
- C-MPI (CDAC)
- Others

MPI Services

- Hide details of architecture
- Hide details of message passing, buffering
- Provides message management services
  - packaging
  - send, receive
  - broadcast, reduce, scatter, gather message modes

MPI Program Organization

- MIMD Multiple Instruction, Multiple Data
  - Every processor runs a different program
- SPMD Single Program, Multiple Data
  - Every processor runs the same program
  - Each processor computes with different data
  - Variation of computation on different processors through if or switch statements

MPI starting and finishing

- Statement needed in every program before any other MPI code

```
MPI_Init(&argc, &argv);
```

- Last statement of MPI code must be

```
MPI_Finalize();
```

  - Program will not terminate without this statement

MPI Messages

- Message content, a sequence of bytes
- Message needs wrapper
  - analogous to an envelope for a letter

Letter	Message
Address	Destination
Return Address	Source
Type of Mailing (class)	Message type
Letter Weight	Size (count)
Country	Communicator
Magazine	Broadcast

A Minimal MPI Program (C)

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

int main( int argc, char *argv[] )
{
    MPI_Init( &argc, &argv );
    printf( "Hello, world!\n" );
    MPI_Finalize();
    return 0;
}
```

Notes on C and Fortran

- C and Fortran bindings correspond closely
- In C:
  - mpi.h must be #included
  - MPI functions return error codes or **`MPI_SUCCESS`**
- In Fortran:
  - mpif.h must be included, or use MPI module (MPI-2)
  - All MPI calls are to subroutines, with a place for the return code in the last argument.
- C++ bindings, and Fortran-90 issues, are part of MPI-2.

Error Handling

- By default, an error causes all processes to abort.
- The user can cause routines to return (with an error code) instead.
  - In C++, exceptions are thrown (MPI-2)
- A user can also write and install custom error handlers.
- Libraries might want to handle errors differently from applications.

Running MPI Programs

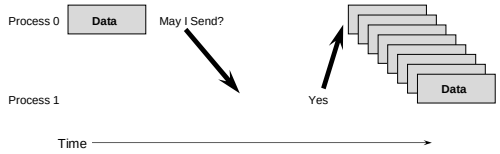
- The MPI-1 Standard does not specify how to run an MPI program, just as the Fortran standard does not specify how to run a Fortran program.
- In general, starting an MPI program is dependent on the implementation of MPI you are using, and might require various scripts, program arguments, and/or environment variables.
- **`mpiexec <args>`** is part of MPI-2, as a recommendation, but not a requirement
  - You can use mpiexec for MPICH and mpirun for SGI's MPI in this class

Finding Out About the Environment

- Two important questions that arise early in a parallel program are:
  - How many processes are participating in this computation?
  - Which one am I?
- MPI provides functions to answer these questions:
  - **`MPI_Comm_size`** reports the number of processes.
  - **`MPI_Comm_rank`** reports the *rank*, a number between 0 and size-1, identifying the calling process

What is message passing?

- Data transfer plus synchronization



- Requires cooperation of sender and receiver
- Cooperation not always apparent in code

Some Basic Concepts

- Processes can be collected into *groups*.
- Each message is sent in a *context*, and must be received in the same context.
- A group and context together form a *communicator*.
- A process is identified by its *rank* in the group associated with a communicator.
- There is a default communicator whose group contains all initial processes, called **`MPI_COMM_WORLD`**.

MPI Datatypes

- The data in a message to sent or received is described by a triple (address, count, datatype), where
- An MPI *datatype* is recursively defined as:
  - predefined, corresponding to a data type from the language (e.g., MPI_INT, MPI_DOUBLE_PRECISION)
  - a contiguous array of MPI datatypes
  - a strided block of datatypes
  - an indexed array of blocks of datatypes
  - an arbitrary structure of datatypes
- There are MPI functions to construct custom datatypes, such an array of (int, float) pairs, or a row of a matrix stored columnwise.

MPI Tags

- Messages are sent with an accompanying user-defined integer *tag*, to assist the receiving process in identifying the message.
- Messages can be screened at the receiving end by specifying a specific tag, or not screened by specifying **`MPI_ANY_TAG`** as the tag in a receive.
- Some non-MPI message-passing systems have called tags “message types”. MPI calls them tags to avoid confusion with datatypes.

MPI Basic (Blocking) Send

MPI_SEND (start, count, datatype, dest, tag, comm)

- The message buffer is described by (**`start, count, datatype`**).
- The target process is specified by **`dest`**, which is the rank of the target process in the communicator specified by **`comm`**.
- When this function returns, the data has been delivered to the system and the buffer can be reused. The message may not have been received by the target process.

MPI Basic (Blocking) Receive

MPI_RECV(start, count, datatype, source, tag, comm, status)

- Waits until a matching (on **`source`** and **`tag`**) message is received from the system, and the buffer can be used.
- **`source`** is rank in communicator specified by **`comm`**, or **`MPI_ANY_SOURCE`**.
- **`status`** contains further information
- Receiving fewer than **`count`** occurrences of **`datatype`** is OK, but receiving more is an error.

Retrieving Further Information

- **`Status`** is a data structure allocated in the user's program.
- In C:

```
int recvd_tag, recvd_from, recvd_count;
MPI_Status status;
MPI_Recv(..., MPI_ANY_SOURCE, MPI_ANY_TAG, ..., &status )
recvd_tag  = status.MPI_TAG;
recvd_from = status.MPI_SOURCE;
MPI_Get_count( &status, datatype, &recvd_count );
```

- In Fortran:

```
integer recvd_tag, recvd_from, recvd_count
integer status(MPI_STATUS_SIZE)
call MPI_RECV(..., MPI_ANY_SOURCE, MPI_ANY_TAG, .. status, ierr)
tag_recvd  = status(MPI_TAG)
recvd_from = status(MPI_SOURCE)
call MPI_GET_COUNT(status, datatype, recvd_count, ierr)
```

When to use MPI

- Portability and Performance
- Irregular Data Structures
- Building Tools for Others
  - Libraries
- Need to Manage memory on a per processor basis

PVM (Parallel Virtual Machine) versus MPI

- PVM
  - o The development of PVM started in summer 1989 at Oak Ridge National Laboratory (**ORNL**).
  - o PVM was effort of a single research group, allowing it great flexibility in design of this system
- MPI
  - o The development of **MPI** started in April 1992. MPI was designed by the **MPI Forum** (a diverse collection of implementers, library writers, and end users) quite independently of any specific implementation

PVM and MPI - GOALS

PVM

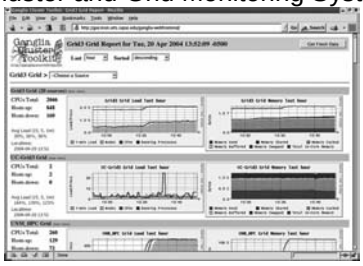
- ✓A distributed operating system
- ✓Portability
- ✓Heterogeneity
- ✓Handling communication failures

MPI

- ✓A library for writing application program, not a distributed operating system
- ✓portability
- ✓High Performance
- ✓Heterogeneity
- ✓Well-defined behavior

Ganglia

Cluster and Grid Monitoring System



Live Demonstration

References

- William Gropp, Ewing Lusk and Thomas Sterling (Editors), Beowulf Cluster Computing with Linux, MIT Press, 2003
- http://www.rocksclusters.org
- http://ganglia.sourceforge.net/
- MPI Forum: http://www.mpi-forum.org
- MPI Tutorials: http://www.mcs.anl.gov/mpi/learning.html
- Marc Dubreuil, Christian Gagne and Marc Parizeau, “Analysis of a Master-Slave Architecture for Distributed Evolutionary Computation”, IEEE Transactions on Systems, Man and Cybernetics-Part B: Cybernetics, Vol. 36, No. 1, 2006, pp. 229-235.

Thanks