LINUX CLUSTER MANAGEMENT TOOLS

Parallel Computing and Applications, Jan 7-14, 2005, IMSc, Chennai

N. Sakthivel

Institute for Plasma Research

Gandhinagar - 382 428

svel@ipr.res.in

STRUCTURE OF DISCUSSION

- Definition
- Cluster Components
- Cluster Categorization
- Cluster Management Tools
- Closure

Cluster Management Includes What?

User Requirement Analysis

Applications, No. of users, Criticality

Cluster Requirement Analysis

Hardware

Software tools

- Suitable File system
- Monitoring tools & Load Balancing
- Providing High Level of Security
- Remote Operation and Management
- Fail over configuration policies

HPC Areas

Research & Development

Educational Institutes

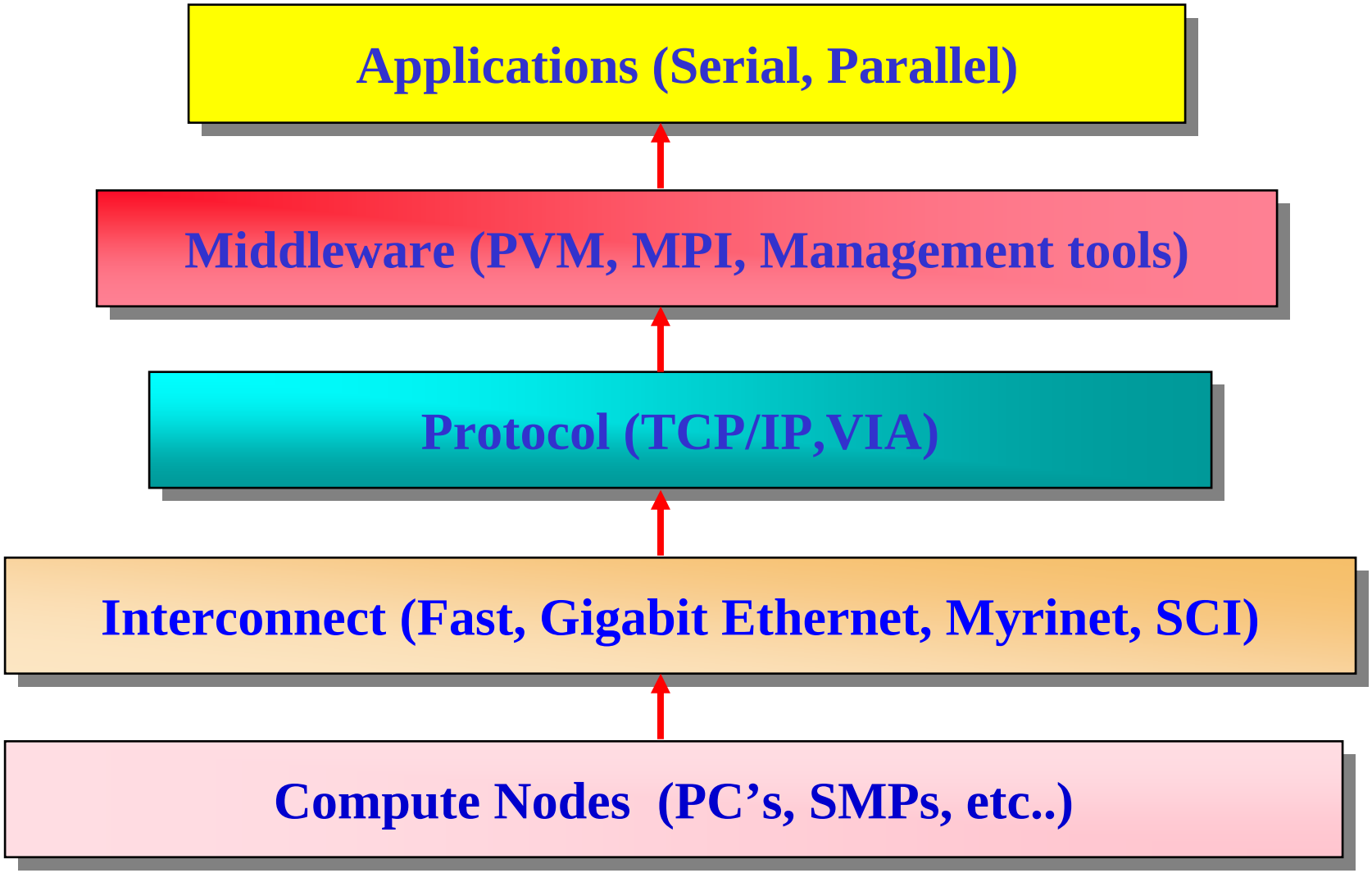
Mission Critical applications

Commercial applications

Efficiency is sacrificed for learning / cost

Efficiency will have greater impact

ARCHITECTURAL STACK OF A TYPICAL CLUSTER



CLUSTER CATEGORIZATION

Smaller Installation **< 20 nodes**

Medium Sized Installation **20–100 nodes**

Large Cluster Installation **> 100 nodes**

Smaller Installation

< 20 nodes

Limited users / applications

Usual management tools + shell/perl scripts

Medium / Large Installations

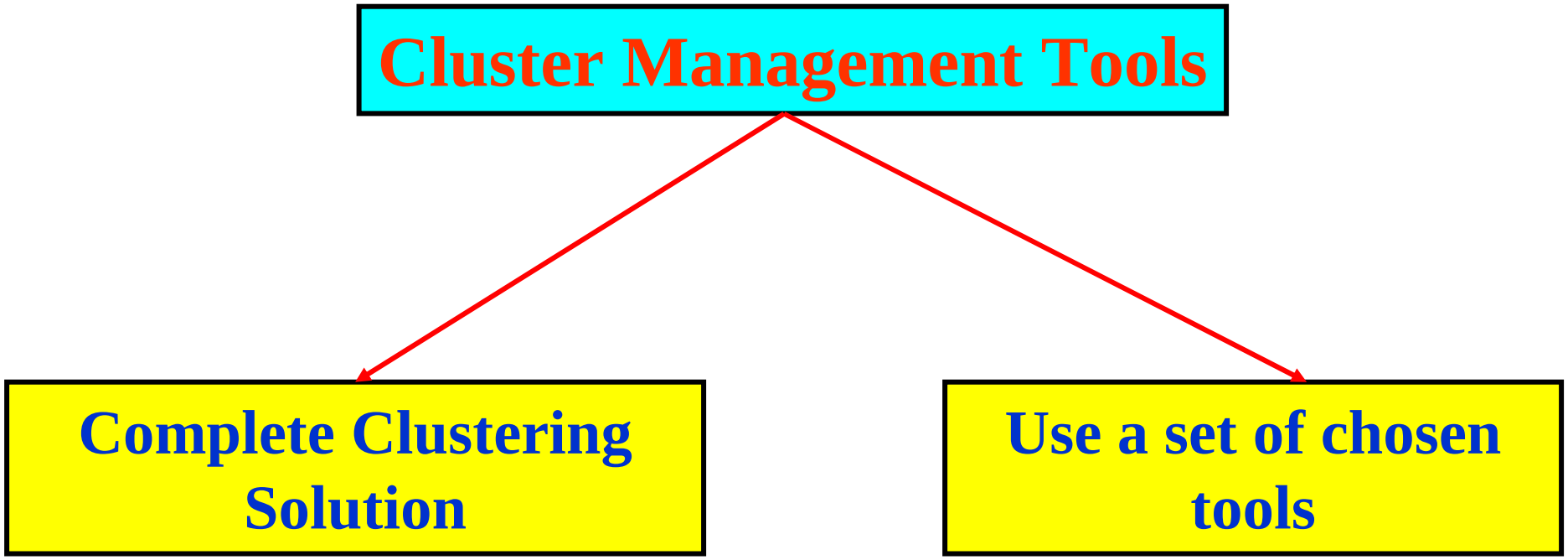
Different Groups

Different Organizations

=> Large number of applications & Users

Proper allocation of CPU / Memory / HDD / Architechture

Strict control access


Cluster Management Tools
Complete Clustering Solution
Use a set of chosen tools

Complete Clustering Solution

Free and Open

- OSCAR (OpenSource Cluster Application Resources)
- NPACI Rocks
- OpenSSI (Open Single System Image)

Commercial

- IBM CSM (Cluster Systems Management)
- Scyld Beowulf
- SSI (Single System Image)

OSCAR

(OpenSource Cluster Application Resources)

Features

- Linux Utility for cluster installation (LUI)
- Eetherboot for node boots
- Three installation levels (from “simple” to “expert”)
- C3 for cluster wide commands, file distribution, remote shutdown
- MPI-LAM
- OpenPBS-MAUI - Batch Queue System
- Precompiled packages

URL: http://www.oscar.openclustergroup.org

NPACI Rocks

Features:

- Based on kickstart and RPM (RedHat)
- Use MySQL for DB
- Heterogeneous nodes easily allowed
- NIS for accounts
- NFS for HOME
- MPICH, PVM
- Ganglia – Scalable Distributed Monitoring System
- PBS + MAUI – Batch Queue System

http://www.rocksclusters.org/Rocks

OpenSSI (Single System Image)

Features:

- One system (config files) to manage
- NFS/NIS file system
- Single point of Access and Control system
- All tasks (environmental setup, compilation, program execution) on the same system
- MPICH, MPI-LAM
- Ganglia
- OpenPBS

http://openssi.org/

Use a set of chosen tools

- **OS with suitable File System**
- **Message Passing Libraries**
- **Monitoring tools**
- **Batch queue systems**

File system Strategy

Each node has individual local file system (identical)

Pros:

Redundancy, Better performance

Cons

Waste space

Difficulty in administration

Global shared file system

Pros

Minimal changes for administration

Cons:

Performance

Reliability

Monitoring Tools

In General Consists

- Are made of scripts (shell, perl, Tcl/Tk etc.)
- Use DB text files
- Checks can be customized simply writing a plugin script or program
- Generally work by polling nodes at fixed intervals using “plugins”
- Many have web interface

Ouputs

- No. of Users
- No. of Jobs
- Memory
- CPU utilization
- Swap space

Monitoring Tools

- **A node hangs**
- **A FS is full**
- **Provides cluster level, node level information**
- **Email if a node hangs**

Available tools

bWatch - Beowulf Watch

SCMS – Scalable Cluster Management System (SCMSWeb)

NetSaint – New name Negios (Network Monitor)

Big Brother – System and Network monitor

Ganglia – Scalable distributed monitoring system

Available as stand-alone tool: take the preferred one

Program Execution

Option 1:

Interactive : **User's programs are directly executed by the login shell, and run immediately**

Option: 2

Batch: **User's submit jobs to system program which will be executed according to the available resources and site policy**

Option 2 has control over jobs running in the Cluster

User's Working Model

What we want users to do?

- **Prepare programs**
- **Understand resource requirements of their programs**
- **Submit a request to run a program (job), specifying resource requirements to the central program**
- **Which will execute them in the future, according to the resource requirements and site policy**

What User has to do?

- **Create a resource description file**
- **ASCII text file (Use either “vi” or with GUI editor)**
- **Contains commands & Keywords to be interpreted by the Batch Queue system**

  Job name

  Maximum run time

  Desired Platform & resources

Keywords depends on the Batch Queue system

What Batch Queue Systems can do?

- Knows what are all the resources available in the HPC environment
  - Architechture
  - Memory
  - Hard Disk space
  - Load
- Who will be allowed to run applications on which nodes
- How long one will be allowed to run his/her job
- **Checkpoint** jobs at regular intervals for restarting
- Job **migration** if needed - Shifting job from one node to other
- Start job on specified date/time

What Batch Queue Systems can do?

- **100 nodes with 20% of CPU usage**

  = 20 nodes's with 100% of CPU usage

Available Batch Queue Systems / Schedulers for Linux

- **Condor**
- **OpenPBS & PBS-Pro**
- **DQS**
- **Job Manager**
- **GNU QUEUE**
- **LSF**

CONDOR

(Cluster Scheduler)

Outline

- **Condor Features**
- **Working Configuration (Daemons)**
- **Availability**
- **Installation & Configuration**
- **Job submission**

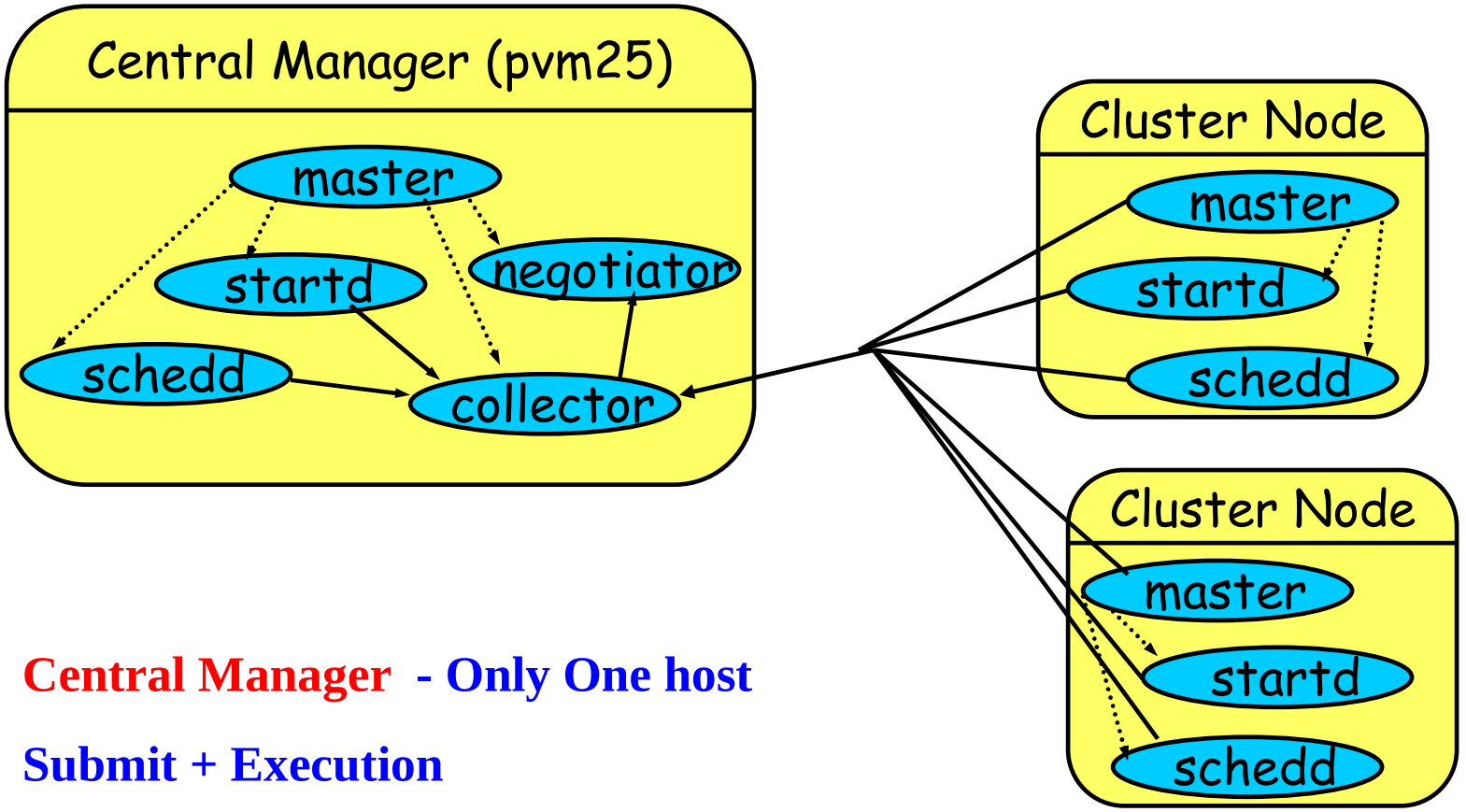
Condor Features

- **ClassAds** - Resource matching
- **Cycle Stealing** - Checks for free nodes & run job
- **Distributed Job Subm.**– Submit from any node
- **Job Priorities** – Priority for imprt. jobs
- Job **Checkpoint** and **Migration**
- **PVM** and **MPI** jobs
- Job Dependency (Job A→ Job B → Job C)

Working Configuration of Condor

- Set of nodes and set of Daemons – Condor Pool
- Central Manager, Execution Host, Submit host, Checkpoint server
- Users submit their jobs in the Submit host with required resources
- Collector responsible for collecting all the status of a condor pool
- Negotiator does the match making and places the job

Working Configuration of CONDOR



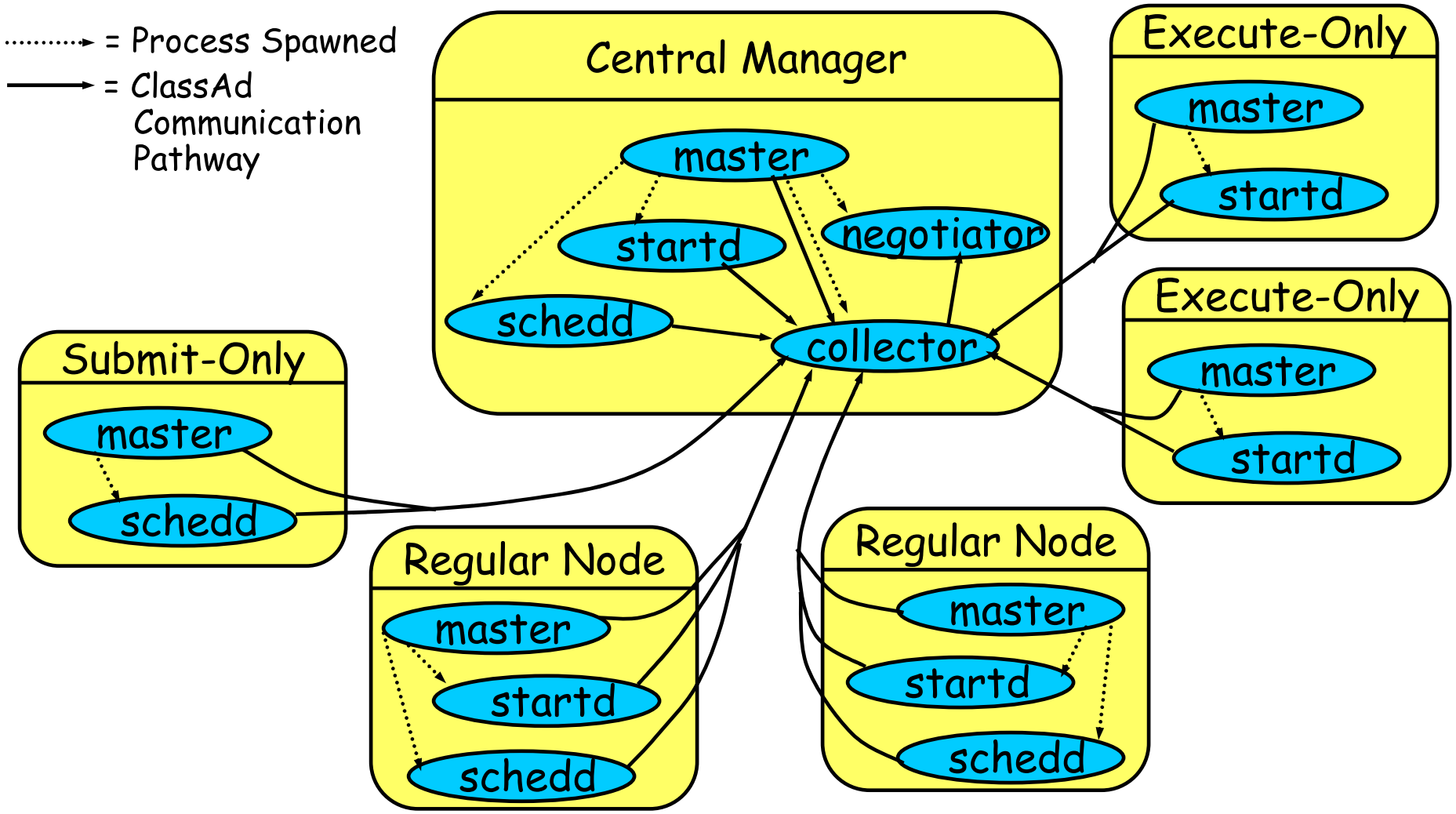
Central Manager - Only One host

Submit + Execution

Submit only

Execution only

Typical Condor Pool



Condor master

- Starts up all other Condor daemons
- If a daemon exits unexpectedly, restarts deamon and emails administrator
- If a daemon binary is updated (timestamp changed), restarts the daemon
- Provides access to many remote administration commands:
  - `condor_reconfig`, `condor_restart`, `condor_off`, `condor_on`, etc.

Condor_startd

- Represents a machine to the Condor pool
- Should be run on any machine you want to run jobs
- Enforces the wishes of the machine owner (the owner's "policy")
- Starts, stops, suspends jobs
- Spawns the appropriate **`condor_starter`**, depending on the type of job

Condor_schedd

- Represents jobs to the Condor pool
- Maintains persistent queue of jobs
  - Queue is not strictly FIFO (priority based)
  - Each machine running **`condor_schedd`** maintains its own queue
- Should be run on any machine you want to submit jobs from
- Responsible for contacting available machines and spawning waiting jobs
  - When told to by **`condor_negotiator`**
- Services most user commands:
  - **`condor_submit, condor_rm, condor_q`**

Condor_collector

- Collects information from all other Condor daemons in the pool
- Each daemon sends a periodic update called a ClassAd to the collector
- Services queries for information:
  - Queries from other Condor daemons
  - Queries from users (**`condor_status`**)

Central Manager

- The Central Manager is the machine running the master, collector and negotiator

```
DAEMON_LIST = MASTER, COLLECTOR,
NEGOTIATOR
```

```
CONDOR_HOST = pvm23.plasma.ernet.in
```

Defined in condor_config file

Condor Availability

- **Developed by University of Wisconsin, Madison**

  http://www.cs.wisc.edu/condor

- **Stable Version 6.6.7 – Oct. 2004**

  Development Version 6.7.2 – Oct. 2004

  Free and Open Source with agreement

- **Fill a Registration form and download**
- **Mailing Lists**

condor-world@cs.wisc.edu	Announce new versions
condor-users-request@cs.wisc.edu	Forum to users to learn

Condor Version Series

Two Versions

Stable Version

Development Series

Stable Version

Well tested

2nd number of version string is even (6.4.7)

Development Series

Latest features, but not recommended

2nd number of version string is odd (6.5.2)

Considerations for Installing Condor

- Decide the version
- Choose your central manager?
- Shared file system? Individual file system?
- Where to install Condor binaries and configuration files?
- Where should you put each machine's local directories?
- If the central manager crashes, jobs that are currently matched will continue to run, but new jobs will not be matched

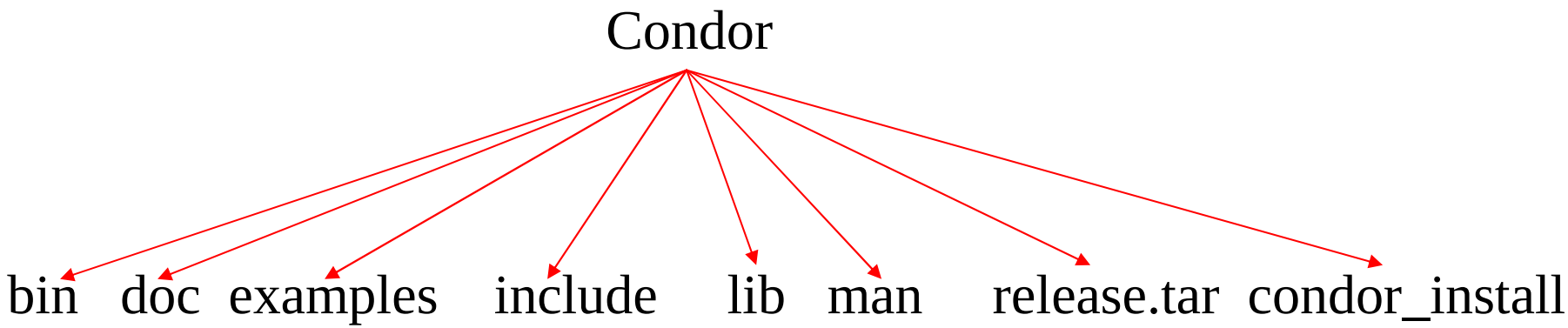
File System?

- Shared location for configuration files can ease administration of a pool
- Binaries on a shared file system makes upgrading easier, but can be less stable if there are network problems
- **`condor_master`** on the local disk is adviced

Machine's local directories?

- You need a fair amount of disk space in the spool directory for each **`condor_schedd`** (holds job queue and binaries for each job submitted)
- The execute directory is used by the **`condor_starter`** to hold the binary for any Condor job running on a machine

Condor directory structure



etc home INSTALL LICENSE.TXT README sbin

```
$ condor_install – Execute for Condor installation
```

Submit-only
Full-install
Central Manager

Hostnames

Machines in a condor pool communicates with machines names, So machine name is a must.

Configuration files

–Global config file

–Local config files

Global Config File

- Found either in file pointed to with the **`CONDOR_CONFIG`** environment variable, **`/etc/condor/condor_config`**, or **`~condor/condor_config`**
- Most settings can be in this file
- Only works as a global file if it is on a shared file system
- In a cluster of independent nodes, this changes has to be done on each machine

Global Config File

Divided into four main parts:

Part 1: Settings you MUST customize

Part 2: Settings you may want to customize

Part 3: Settings that control the policy of when condor will start and stop jobs on your machines

Part 4: Settings you should probably leave alone

Part 1: Settings you MUST customize

CONDOR_HOST = pvm23.plasma.ernet.in → Central Manager

RELEASE_DIR = /packages/condor

LOCAL_DIR = $(RELEASE_DIR)/home

LOCAL_CONFIG_FILE = $(LOCAL_DIR)/condor_config_local OR

LOCAL_CONFIG_FILE= $(LOCAL_DIR)/etc/$(HOSTNAME).local

CONDOR_ADMIN = yes

MAIL = /bin/mail

Part 2: Settings you may want to customize

FLOCK_FROM = Allowing access to machines from different pool

FLOCK_TO = Running jobs on other pool

USE_NFS = yes

USE_CKPT_SERVER = yes

CKPT_SERVER_HOST = pvm23.plasma.ernet.in

DEFAULT_DOMAIN_NAME = plasma.ernet.in

MAX_JOBS_RUNNING = 150 (Max. no. of jobs from a single submit machine)

MAX_COLLECTOR_LOG = 640000

MAX_NEGOTIATOR_LOG = 640000

Part 3: Settings that control the policy of when condor will start and stop jobs on your machines

START = TRUE

SUSPEND = FALSE

CONTINUE = TRUE

PREEMPT = FALSE

KILL = FALSE

Local Configuration File

- LOCAL_CONFIG_FILE - macro
- Can be on local disk of each machine

`/packages/condor/condor_config.local`

- Can be in a shared directory

`/packages/condor/condor_config.$(HOSTNAME)`

`/packages/condor/hosts/$(HOSTNAME)/condor_config.local`

- **Machine specific settings were done here**

Condor Universe

Standard	**- Default (Checkpoint, remote system calls) link with Condor_compile**
Vanilla	**- Can not Checkpoint & migrate jobs**
PVM	**- PVM jobs**
MPI	**- MPICH jobs**
Java	**- Java jobs**

Job Submit Description file

```
##########################
# Example1
# Simple Condor job submission file
##########################
Executable = test.exe
Log = test.log
Queue
```

`$ condor_submit <Example1>`

View the queue with *condor_q*
Job completion may be intimated by
notification=e-mail ID

```
#######################
# Example 2
# Multiple directories
#########################
Executable = test.exe
Universe = vanilla
input = test.input
output = test.out
Error = test.err
Log = test.log
initialdir = run1
Queue
initialdir = run2
Queue
```

Multiple runs

```
# Example 3: Multiple runs with requirements

Executable = test.exe

Requirements = Memory >=32 && Opsys = = “LINUX” && Arch = = “INTEL”

Rank = Memory >= 64

Error = err.$(Process)

Input = in.$(Process)

Ouput = out.$(Process)

Log = test.log

notify_user = svel@ipr.res.in

Queue 200
```

MPI Jobs

```
# MPI example submit description file

Universe = MPI

Executable = test.exe

Log = test.log

Input = in.$(NODE)
output = out.$(NODE)

Error = err.$(NODE)

machine_count = 10

queue
```

$(NODE) – Rank of a program

MPI jobs has to be run on dedicated resources

File Transfer Mechanisms

Standard Universe – File transfer is done automatically

Vanialla & MPI – Assumes a common file across all nodes, otherwise

Use file transfer machanism

```
transfer_input_file = file1 file2
transfer_files = ONEXIT
transfer_ouput_file = final-results
```

VERIFICATION

Finding what all machine in condor pool

[condor@pvm25]$ condor_status

```
Name               OpSys      Arch     State      Activity   LoadAv  Mem    ActvtyTime

pvm25.plasma.  LINUX      INTEL    Unclaimed  Idle       0.000    998    0+00:46:04
pvm26.plasma.  LINUX      INTEL    Unclaimed  Idle       0.000    998    0+00:42:04
pvm27.plasma.  LINUX      INTEL    Unclaimed  Idle       0.000    998    0+00:46:57

                     Machines  Owner Claimed Unclaimed Matched Preempting

        INTEL/LINUX        3          0       0          3          0          0

               Total       3          0       0          3          0          0
```

Submitting a job

[condor@pvm25 myjobs]$ condor_submit example.cmd
Submitting job(s).
Logging submit event(s).2
1 job(s) submitted to cluster 215

Overview of condor commands

User Commands

- **condor_status** View Pool Status
- **condor_q** View Job Queue
- **condor_submit** Submit new Jobs
- **condor_rm** Remove Jobs
- **condor_prio** Change user priority
- **condor_history** Completed Job Info
- **condor_checkpoint** Force checkpoint
- **condor_compile** Link Condor library

- **condor_master** Starts master daemon
- **condor_on** Start Condor
- **condor_off** Stop Condor
- **condor_reconfig** Reconfig on-the-fly
- **condor_config_val** View/set config
- **condor_userprio** User Priorities
- **condor_stats** View detailed usage stats

PBS
(OpenPBS, PBS-Pro)

(Free and Open) (Commercial)

Outline

- **OpenPBS Features**
- **Working Configuration (Daemons)**
- **Availability**
- **Installation & Configuration**
- **Job submission**

OpenPBS Features

- **Job Priority**
- **Automatic File Staging**
- **Single or Multiple Queue support**
- **Multiple Scheduling Algorithm**
- **Support for Parallel Jobs**

PBS Working Configuration

- **A resource manager (PBS Server)**
- **A Scheduler (PBS scheduler)**
- **Many Executors (PBS moms)**

PBS Server & PBS Scheduler – Front end

MOMS – all nodes

PBS Working Configuration

PBS Server	PBS Scheduler	PBS MOM
(One server process)	(One scheduler process)	(One for each comp. node)
Receives batch jobs	Contains policy	Places jobs into execution
Invokes the scheduler	Communicates with moms to learn about the state of the system	
		Takes instruction from server
Instruct moms to execute jobs	Communicate with server to learn the availability of jobs	Reports back to server

Availability

- Developed by MRJ Technology Solutions for NASA Ames Research Centre,
- From Veridian Corporation, USA
- Fill in the registration form and download the tar file
- RPM packages are available, but sources are better for configuration and customization

http://www.openpbs.org

Installing PBS from Source

- Decide where the PBS source and objects to be
- “Untar” the distribution
- Configuration – [`# configure` --options]
- Compiling PBS modules – [`# make`]
- Installing PBS modules - [`# make install`]
- Create node description file
- Configure the Server
- Configure the Scheduler
- Configure the Moms
- Test out the scheduler with sample jobs

Installing PBS from RPM

- **RPM for front end – Containing Server & Scheduler**
- **RPM for mom/client – Containing MOM server**

Default Installation Directories

/usr/local/bin – User commands

/usr/local/sbin – Daemons & administrative commands

/usr/spool/pbs - $(PBS_HOME)

Configuration files

$(PBS_HOME)/server_priv/nodes – list of hostname:ts – time share

$(PBS_HOME)/serv_priv

$(PBS_HOME)/mom_priv

$(PBS_HOME)/sched_priv

PBS Sever Configuration

Two parts

Configuring the server attributes

Configuring queues and their attributes

Usage:

```
# qmgr
```

default_node

managers

query other jobs

Resource_max, resource_min (specific queue)

PBS Server Configuration

Commands operate on three main entities

- server set/change server parameters
- node set/change properties of individual nodes
- queue set/change properties of individual queues

Users: A list of users who may submit jobs

```
# qmgr set server acl_user = user1, user2
# qmgr set server acl_user_enable = True
# qmgr create queue <queue_name>
# qmgr create queue cfd
```

True = turn this feature on, False = turn this feature of

PBS Attributes

Max jobs per queue

Controls how many jobs in this queue can run simultaneously

```
set queue cfd max_running = 5
```

Max user run

Controls how many jobs an individual userid can simultaneously across the entire server

Helps prevent a single user from monopolizing system resources

```
set queue cfd max_user_run = 2
```

Priority

Sets the priority of a queue relative to other queues

```
set queue mdynamics priority = 80
```

How PBS Handles a Job

- User determines resource requirements for a job and writes a batch script
- User submits job to PBS with the **`qsub`** command
- PBS places the job into a queue based on its resource requests and runs the job when those resources become available
- The job runs until it either completes or exceeds one of its resource request limits
- PBS copies the jobs output into a directory from which the job was submitted and optionally notifies the user via email that the job has ended

Job Requirements

- For single CPU jobs, PBS needs to know at least two resource requirements

  CPU time

  Memory

- For multiprocessor parallel jobs, PBS also needs to know how many nodes/CPU are required

- Other things to consider

  Job name?

  Where to put standard output and error output?

  Should the system email when the job completes?

PBS Job Scripts

- Just like a regular Shell script which some comments which are meaningful to PBS

  #PBS - Every PBS script line starts with this

- Starts with $HOME. If you need to work another directory, your job script will need to ***cd*** to there

- Characteristics of a typical UNIX session associated with them

  A login procedure
  stdin, stdout, stderr

PBS Job Script

-l mem=N[KMG] (request N[kilo|mega|giga] bytes of memory)

-l cput=hh:mm:ss (max CPU time per job request)

-l walltime=hh:mm:ss (max wall clock time per job request)

-l nodes=N:pp=M (request N nodes with M processors per node)

-I (run as an interactive job)

-N jobname (name the job jobname)

-S shell (use shell instead of your login shell to interpret the batch script)

-q queue (explicitely request a batch destination queue)

-o outfile (redirect standard output to outfile)

-e errfile (redirect standard output to outfile)

-j joe (combine stdout, stderr together)

-m e (mail the user when the job completes)

PBS Script file example

```
#PBS -l cput=10:00:00

#PBS –l mem=256MB

#PBS –l nodes=1:ppn=1

#PBS –N test

#PBS –j oe

#PBS –S /bin/ksh

cd $HOME/project/test

/usr/bin/time ./theory > test.out
```

This job asks for one CPU on one node, 256 MB of memory, and 10 hours of CPU time.

Interactive Batch Setup

`$ qsub –I <script>`

-I indicates the interactive request

No shell script command in the batch script.

Typical PBS script

```
#PBS -l cput=10:00:00
#PBS –l mem=256MB
#PBS –l nodes=1:ppn=1
#PBS –N test
#PBS –j oe
#PBS –S /bin/ksh
cd $HOME/project/test
/usr/bin/time ./theory > test.out
```

PBS script for Interactive jobs

```
#PBS -l cput=10:00:00
#PBS –l mem=256MB
#PBS –l nodes=1:ppn=1
#PBS –N test
#PBS –j oe
#PBS –S /bin/ksh
```

SMP Jobs

The difference between a uniprocessor and an SMP jobs is the resource request limit

-l nodes=1:ppn=N

With N>1, contained in the SMP job script

This tells PBS that the job will run N processes cuncurrently on one node, so PBS allocates the required CPUs for you

If you simply request a number of nodes (eg –l nodes=1), PBS will assumes that you want one processor per node.

Parallel jobs

```
#PBS –j oe
#PBS –l nodes=4:ppn=4
#PBS –l cput=5:00:00
#PBS –l mem=256MB
cd /users/svel/mpi
mpirun ./torch
```

PBS Logs

Daemons logs can be found in:

$PBS_HOME/server_logs

$PBS_HOME/sched_log

$PBS_HOME/mom_logs

Named with the YYYYMMDD naming convention

Accounting logs

$PBS_HOME/server_priv/accounting

Closure

- **Understand user requirements**
- **Choose the suitable hardware**
- **Survey the available management tools and choose**
- **Follow up the updates and corresponding mailing lists**

Submitting a job

```
$ qsub test.cmd
```

qsub [-a date_time] [-A account_string] [-c interval] [-C directive_prefix] [-e path] [-h][-I] [-j join] [-k keep] [-l resource list] [-m mail_options] [-M user_list] [-N name] [-o path] [-p priority] [-q destination] [-r c] [-S path_list] [-u user_list] [-v variable_list] [-V] [-W additional_attributes] [-z] [script]