Condor and BOINC
Distributed and Volunteer Computing

Presented by Adam Bazinet
Condor

- Developed at the University of Wisconsin-Madison
- Condor is aimed at High Throughput Computing (HTC) on collections of distributively owned resources
- Mainly used to scavenge idle CPU cycles from workstations
Typical Condor Pool

- Central Manager
  - master
  - startd
  - negotiator
  - schedd

- Regular Node
  - master
  - startd
  - schedd

- Submit-Only
  - master
  - schedd

- Execute-Only
  - master
  - startd

Communication Pathway:

- Dashed line = Process Spawned
- Solid line = ClassAd Communication Pathway
Condor Daemons

- `condor_master` - keeps other daemons running
- `condor_startd` - advertises a given resource
- `condor_starter` - spawns a remote Condor job
- `condor_schedd` - local job scheduler
- `condor_shadow` - coordinates with submitted job
- `condor_collector` - keeps status of Condor pool
- `condor_negotiator` - does all matchmaking
Condor Universes

- Universes are runtime environments for jobs
  - **Standard** universe
    - Provides checkpointing and remote system calls
    - Application must be re-linked with `condor_compile`
  - **Vanilla** universe
    - Instead of with remote system calls, files are accessed with NFS/AFS or explicitly transferred to the executing host
- Other universes: PVM, MPI, Globus, Java, Scheduler
# This is a sample Condor submit description file. It should be modified before using it.
#
# The universe the job will execute in. Other values include "standard" (for programs compiled with Condor checkpointing),
# "java" (for java programs), "MPI", and more.
# Universe = vanilla

# This is the command Condor will attempt to invoke on the remote system. It can be an absolute path or a relative path.
# If the executable doesn't already exist on the remote system, it should be transferred at submit time.
# Executable = /bin/echo

# If you'd like to provide arguments to the program, specify them here.
# Arguments = Hello World

# If your program takes something on standard input, specify the file name here.
# Input = /dev/null

# You should specify where you'd like stdout, stderr, and the Condor log file to be written to, as well as how they should be named.
# If path information is omitted, the file will be written to the directory the job is submitted from.
# Output = /my/home/directory/condor_stdout
Error = condor_stderr
Log = condor.log
Most often, you'll need to transfer files (be they the program executable, various dependencies, or program input files).

Should_transfer_files = YES

This is a list of the input files to be transferred. It is not necessary to specify the executable here.

Transfer_input_files = file1,joe./my/home/dir/file2,~/file3

This directs Condor to return output when the program is finished executing.

WhenToTransferOutput = ON_EXIT

The requirements string identifies the attributes that a host must match in order to execute this job, and it can be very complex. The examples here restrict job execution to a particular architecture and operating system.

Requirements = Arch == "INTEL" && OpSys == "LINUX"
Requirements = Arch == "PPC" && Arch == "OSX"
Requirements = Arch == "SUN4u" && OpSys == "SOLARIS28"

You can be emailed about the results of a job using the directives below.

Notification = Complete
Notification = Error
Notification = Never
Notify_user = pknut777@umiacs.umd.edu

This directive is necessary to "end" the submit file and instruct Condor how many copies of the job to submit.

Queue 1
```
<table>
<thead>
<tr>
<th>ID</th>
<th>OWNER</th>
<th>SUBMITTED</th>
<th>RUN_TIME</th>
<th>ST</th>
<th>PRI</th>
<th>SIZE</th>
<th>CMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>nedwards</td>
<td>10/24 21:38</td>
<td>0+00:00:00</td>
<td>I</td>
<td>1</td>
<td>9.8</td>
<td>condor_exec_tandem</td>
</tr>
<tr>
<td>2.0</td>
<td>nedwards</td>
<td>10/24 21:38</td>
<td>0+00:00:00</td>
<td>I</td>
<td>1</td>
<td>9.8</td>
<td>condor_exec_tandem</td>
</tr>
<tr>
<td>3.0</td>
<td>nedwards</td>
<td>10/24 21:38</td>
<td>0+00:00:00</td>
<td>I</td>
<td>1</td>
<td>9.8</td>
<td>condor_exec_tandem</td>
</tr>
<tr>
<td>4.0</td>
<td>nedwards</td>
<td>10/24 21:38</td>
<td>0+00:00:00</td>
<td>I</td>
<td>1</td>
<td>9.8</td>
<td>condor_exec_tandem</td>
</tr>
<tr>
<td>5.0</td>
<td>nedwards</td>
<td>10/24 21:38</td>
<td>0+00:00:00</td>
<td>I</td>
<td>1</td>
<td>9.8</td>
<td>condor_exec_tandem</td>
</tr>
<tr>
<td>6.0</td>
<td>nedwards</td>
<td>10/24 21:38</td>
<td>0+00:00:00</td>
<td>I</td>
<td>1</td>
<td>9.8</td>
<td>condor_exec_tandem</td>
</tr>
<tr>
<td>7.0</td>
<td>nedwards</td>
<td>10/24 21:38</td>
<td>0+00:00:00</td>
<td>I</td>
<td>1</td>
<td>9.8</td>
<td>condor_exec_tandem</td>
</tr>
<tr>
<td>8.0</td>
<td>nedwards</td>
<td>10/24 21:38</td>
<td>0+00:00:00</td>
<td>I</td>
<td>1</td>
<td>9.8</td>
<td>condor_exec_tandem</td>
</tr>
<tr>
<td>9.0</td>
<td>nedwards</td>
<td>10/24 21:38</td>
<td>0+00:00:00</td>
<td>I</td>
<td>1</td>
<td>9.8</td>
<td>condor_exec_tandem</td>
</tr>
<tr>
<td>10.0</td>
<td>nedwards</td>
<td>10/24 21:38</td>
<td>0+00:00:00</td>
<td>I</td>
<td>1</td>
<td>9.8</td>
<td>condor_exec_tandem</td>
</tr>
<tr>
<td>11.0</td>
<td>nedwards</td>
<td>10/24 21:38</td>
<td>0+00:00:00</td>
<td>I</td>
<td>1</td>
<td>9.8</td>
<td>condor_exec_tandem</td>
</tr>
<tr>
<td>12.0</td>
<td>nedwards</td>
<td>10/24 21:38</td>
<td>0+00:00:00</td>
<td>I</td>
<td>1</td>
<td>9.8</td>
<td>condor_exec_tandem</td>
</tr>
<tr>
<td>32.0</td>
<td>nedwards</td>
<td>10/24 21:38</td>
<td>0+00:00:00</td>
<td>I</td>
<td>1</td>
<td>9.8</td>
<td>condor_exec_tandem</td>
</tr>
<tr>
<td>33.0</td>
<td>nedwards</td>
<td>10/24 21:38</td>
<td>0+00:00:00</td>
<td>I</td>
<td>1</td>
<td>9.8</td>
<td>condor_exec_tandem</td>
</tr>
<tr>
<td>34.0</td>
<td>nedwards</td>
<td>10/24 21:38</td>
<td>0+00:00:00</td>
<td>I</td>
<td>1</td>
<td>9.8</td>
<td>condor_exec_tandem</td>
</tr>
<tr>
<td>35.0</td>
<td>nedwards</td>
<td>10/24 21:38</td>
<td>0+00:00:00</td>
<td>I</td>
<td>1</td>
<td>9.8</td>
<td>condor_exec_tandem</td>
</tr>
<tr>
<td>36.0</td>
<td>nedwards</td>
<td>10/24 21:38</td>
<td>0+00:00:00</td>
<td>I</td>
<td>1</td>
<td>9.8</td>
<td>condor_exec_tandem</td>
</tr>
<tr>
<td>37.0</td>
<td>nedwards</td>
<td>10/24 21:38</td>
<td>0+00:00:00</td>
<td>I</td>
<td>1</td>
<td>9.8</td>
<td>condor_exec_tandem</td>
</tr>
<tr>
<td>38.0</td>
<td>nedwards</td>
<td>10/24 21:38</td>
<td>0+00:00:00</td>
<td>I</td>
<td>1</td>
<td>9.8</td>
<td>condor_exec_tandem</td>
</tr>
<tr>
<td>39.0</td>
<td>nedwards</td>
<td>10/24 21:38</td>
<td>0+00:00:00</td>
<td>I</td>
<td>1</td>
<td>9.8</td>
<td>condor_exec_tandem</td>
</tr>
<tr>
<td>40.0</td>
<td>nedwards</td>
<td>10/24 21:38</td>
<td>0+00:00:00</td>
<td>I</td>
<td>1</td>
<td>9.8</td>
<td>condor_exec_tandem</td>
</tr>
<tr>
<td>41.0</td>
<td>nedwards</td>
<td>10/24 21:38</td>
<td>0+00:00:00</td>
<td>I</td>
<td>1</td>
<td>9.8</td>
<td>condor_exec_tandem</td>
</tr>
</tbody>
</table>

41 jobs; 41 idle, 0 running, 0 held
```
condor_q -long

pknut777@leucine:~
> condor_q -submitter nedwards -long 39

-- Submitter: nedwards@umiacs.umd.edu : <128.8.119.159:46276> : precursor.umiacs.umd.edu
MyType = "Job"
TargetType = "Machine"
ClusterId = 39
QDate = 1193276330
CompletionDate = 0
Owner = "nedwards"
RemoteWallClockTime = 0.000000
LocalUserCpu = 0.000000
LocalSysCpu = 0.000000
RemoteUserCpu = 0.000000
RemoteSysCpu = 0.000000
ExitStatus = 0
NumCkpts = 0
NumRestarts = 0
NumSystemHolds = 0
CommittedTime = 0
TotalSuspensions = 0
LastSuspensionTime = 0
CumulativeSuspensionTime = 0
ExitBySignal = FALSE
CondorVersion = "$CondorVersion: 6.8.0 Jul 19 2006 $"
CondorPlatform = "$CondorPlatform: I386-LINUX_RHEL3 $"
RootDir = "/
Iwd =="/"
JobUniverse = 5
Cmd = "/fs/nehomes/nedwards/projects/GridPepIDE/bin/condor_exec_tandem.sh"
MinHosts = 1
MaxHosts = 1
CurrentHosts = 0
```
conda_status

pknut777@leucine:~
> condor_status

<table>
<thead>
<tr>
<th>Name</th>
<th>OpSys</th>
<th>Arch</th>
<th>State</th>
<th>Activity</th>
<th>LoadAv</th>
<th>Mem</th>
<th>ActvtyTime</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:vm1@blue02.um">vm1@blue02.um</a> LINUX</td>
<td>LINUX</td>
<td>INTEL</td>
<td>Unclaimed</td>
<td>Idle</td>
<td>0.000</td>
<td>500</td>
<td>0:00:11:49</td>
</tr>
<tr>
<td><a href="mailto:vm2@blue02.um">vm2@blue02.um</a> LINUX</td>
<td>LINUX</td>
<td>INTEL</td>
<td>Unclaimed</td>
<td>Idle</td>
<td>0.000</td>
<td>500</td>
<td>0:00:09:43</td>
</tr>
<tr>
<td><a href="mailto:vm1@blue03.um">vm1@blue03.um</a> LINUX</td>
<td>LINUX</td>
<td>INTEL</td>
<td>Unclaimed</td>
<td>Idle</td>
<td>0.010</td>
<td>500</td>
<td>0:00:05:21</td>
</tr>
<tr>
<td><a href="mailto:vm2@blue03.um">vm2@blue03.um</a> LINUX</td>
<td>LINUX</td>
<td>INTEL</td>
<td>Unclaimed</td>
<td>Idle</td>
<td>0.000</td>
<td>500</td>
<td>0:00:04:51</td>
</tr>
<tr>
<td><a href="mailto:vm1@blue06.um">vm1@blue06.um</a> LINUX</td>
<td>LINUX</td>
<td>INTEL</td>
<td>Unclaimed</td>
<td>Idle</td>
<td>0.000</td>
<td>374</td>
<td>0:00:06:47</td>
</tr>
<tr>
<td><a href="mailto:vm2@blue06.um">vm2@blue06.um</a> LINUX</td>
<td>LINUX</td>
<td>INTEL</td>
<td>Unclaimed</td>
<td>Idle</td>
<td>0.000</td>
<td>374</td>
<td>0:00:05:32</td>
</tr>
<tr>
<td><a href="mailto:vm1@blue07.um">vm1@blue07.um</a> LINUX</td>
<td>LINUX</td>
<td>INTEL</td>
<td>Unclaimed</td>
<td>Idle</td>
<td>0.000</td>
<td>500</td>
<td>0:00:08:56</td>
</tr>
<tr>
<td><a href="mailto:vm2@blue07.um">vm2@blue07.um</a> LINUX</td>
<td>LINUX</td>
<td>INTEL</td>
<td>Unclaimed</td>
<td>Idle</td>
<td>0.000</td>
<td>500</td>
<td>0:00:08:54</td>
</tr>
<tr>
<td><a href="mailto:vm1@blue08.um">vm1@blue08.um</a> LINUX</td>
<td>LINUX</td>
<td>INTEL</td>
<td>Unclaimed</td>
<td>Idle</td>
<td>0.000</td>
<td>500</td>
<td>0:00:11:02</td>
</tr>
<tr>
<td><a href="mailto:vm2@blue08.um">vm2@blue08.um</a> LINUX</td>
<td>LINUX</td>
<td>INTEL</td>
<td>Unclaimed</td>
<td>Idle</td>
<td>0.000</td>
<td>500</td>
<td>0:00:10:17</td>
</tr>
<tr>
<td><a href="mailto:vm1@blue09.um">vm1@blue09.um</a> LINUX</td>
<td>LINUX</td>
<td>INTEL</td>
<td>Unclaimed</td>
<td>Idle</td>
<td>0.000</td>
<td>500</td>
<td>0:00:04:36</td>
</tr>
<tr>
<td><a href="mailto:vm2@blue09.um">vm2@blue09.um</a> LINUX</td>
<td>LINUX</td>
<td>INTEL</td>
<td>Unclaimed</td>
<td>Idle</td>
<td>0.020</td>
<td>500</td>
<td>0:00:04:45</td>
</tr>
</tbody>
</table>

```

```
condor_status -long

pknut777@leucine:~
> condor_status -long alanine.pc.umiacs.umd.edu
MyType = "Machine"
TargetType = "Job"
Name = "alanine.pc.umiacs.umd.edu"
Machine = "alanine.pc.umiacs.umd.edu"
Rank = 0.000000
CpuBusy = ((LoadAvg - CondorLoadAvg) >= 0.500000)
COLLECTOR_HOST_STRING = "condorsrv.umiacs.umd.edu"
CondorVersion = "$CondorVersion: 6.8.3 Jan 5 2007 $"
CondorPlatform = "$CondorPlatform: INTEL-WINNT50 $"
VirtualMachineID = 1
VirtualMemory = 1264292
Disk = 27062816
CondorLoadAvg = 0.000000
LoadAvg = 0.340000
KeyboardIdle = 707684
ConsoleIdle = 707684
Memory = 511
Cpus = 1
StartdIpAddr = "<128.8.141.73:1476>"
Arch = "INTEL"
OpSys = "WINNT51"
UidDomain = "alanine.pc.umiacs.umd.edu"
FileSystemDomain = "alanine.pc.umiacs.umd.edu"
Subnet = "128.8.141"
HasIOProxy = TRUE
CheckpointPlatform = "WINNT51 INTEL Unknown normal"
TotalVirtualMemory = 1264292
TotalDisk = 27062816
TotalCpus = 1
TotalMemory = 511
Matchmaking

- Matchmaking is Condor’s scheduling mechanism
- Jobs specify their requirements as a list of attributes and values
- Resources advertise their capabilities as a list of attributes and values (ClassAds)
- The `condor_negotiator` matches jobs to resources using these criteria
Matchmaking (cont’d)

Job Requirements

# The requirements string identifies the attributes that a host must match in order to execute this job, and it can be very complex.
# The examples here restrict job execution to a particular architecture and operating system.

Requirements = Arch == "INTEL" && OpSys == "LINUX"
Requirements = Arch == "PPC" && Arch == "OSX"
Requirements = Arch == "SUN4u" && OpSys == "SOLARIS28"

Resource Requirements and Capabilities

HasFileTransfer = TRUE
HasPerFileEncryption = TRUE
HasReconnect = TRUE
HasMPI = TRUE
HasTDP = TRUE
HasJobDeferral = TRUE
HasJICLocalConfig = TRUE
HasJICLocalStdin = TRUE
HasWindowsRunAsOwner = TRUE
StarterAbilityList = "HasFileTransfer,HasPerFileEncryption,HasReconnect,HasMPI,HasTDP,HasJobDeferral,HasJICLocalConfig,HasJICLocalStdin,HasWindowsRunAsOwner"
CpuBusyTime = 0
CpuIsBusy = FALSE
TimeToLive = 2147483647
State = "Unclaimed"
EnteredCurrentState = 1194625606
Activity = "Idle"
EnteredCurrentActivity = 1195324890
Start = KeyboardIdle > 15 * 60
Requirements = (START) && (IsValidCheckpointPlatform)
IsValidCheckpointPlatform = (((TARGET.JobUniverse == 1) == FALSE) || ((MY.CheckpointPlatform != UNDEFINED) && ((TARGET.LastCheckpointPlatform == MY.CheckpointPlatform) || (TARGET.NumCkpts == 0))))
Previous Work

- In three key areas:
  - The analysis of workstation usage patterns
  - The design of remote capacity allocation algorithms
  - The development of remote execution facilities
Design Goals

• Condor is designed to serve users executing long running background jobs on idle workstations
  • Job placement should be transparent
  • Job migration should be supported
  • Fair access to cycles is expected
  • The system should be low overhead
The Scheduling Spectrum

- At one end: a centralized, static coordinator would handle scheduling
- At the other end: workstations cooperate to conduct a scheduling policy
- In the middle: Condor!

Figure 1: The Condor Scheduling Structure.
Remote Unix (RU) Facility

- Turns idle workstations into cycle servers
- When invoked, a shadow process runs locally as the surrogate of the remotely executing process
- System calls go over the network back to the shadow (an RPC of sorts)
- Used in the standard universe, nowadays
Checkpointing

• When a job is interrupted, RU checkpoints it - the state of the program is sent back to submitting machine, and the job may be rescheduled

• Checkpoints consist of the text, data, bss, and stack program segments, registers, status of open files, outstanding messages to the shadow, and so on...
Checkpointing (cont’d)

- Adding checkpointing requires re-linking an application with `condor_compile`, which fattens up the binary a good deal.

- Programs now use much more RAM than they did in the past, so checkpointing in the Condor fashion may be problematic in some cases...
Fair Access to Remote Cycles

- By means of the Up-Down algorithm
- In essence, the fewer cycles you burn, the greater your priority over other users of the system... (a dynamic equilibrium)
Performance Study

- 23 workstations executing Condor jobs were monitored for 1 month
- Study simulated a “heavy” user, and several light users
- Jobs ranged from 30 minutes to 6 hours
- Queue length as high as 40 jobs, for the heavy user
Results

• On average, light users didn’t have to wait long for their jobs to run - that’s good

• Utilization of remote resources was substantially increased - an additional 200 machine days of capacity were consumed by the Condor system

• Coordinator predicted to be able to manage at least 100 workstations with low overhead
Results (cont’d)

• Average cost of job placement and checkpointing was 2.5 seconds (again, would be higher nowadays)

• On average, all jobs experienced less than one checkpoint per hour

• Remote Unix calls are 20x more expensive than a comparable local call

• A metric called leverage is defined as the ratio of remote capacity consumed to local capacity consumed
Results: Leverage

- All jobs show very high leverage values - that’s good

Figure 9: Remote Execution Leverage.
Conclusions

- The major design goals were achieved!
  - Job placement is transparent
  - Job migration is supported
  - Fair access to cycles is granted
  - The system is low overhead
Condor Today

• Condor has been extremely successful

• It is used by a variety of organizations: large corporations, small businesses, and of course, academic institutions

• At least one company formed to provide Condor support: www.cyclecomputing.com

• Requests for source code are evaluated on a case-by-case basis
<table>
<thead>
<tr>
<th>Condor Pool</th>
<th>CPU Years</th>
<th>Lattice Jobs</th>
<th>Arch. &amp; OS</th>
<th>Idle</th>
<th>Running</th>
<th>Free CPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>UMIACS</td>
<td>24.29</td>
<td></td>
<td>Intel Linux</td>
<td>0</td>
<td>0</td>
<td>154</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Intel Win</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Sun4u Solaris</td>
<td>0</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>Gridiron</td>
<td>25.69</td>
<td></td>
<td>Intel Linux</td>
<td>0</td>
<td>0</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Intel Win</td>
<td>0</td>
<td>0</td>
<td>87</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Ppc Osx</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>CLFS</td>
<td>8.62</td>
<td></td>
<td>Intel Linux</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Intel Win</td>
<td>0</td>
<td>0</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Sun4u Solaris</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Terpcondor</td>
<td>105.40</td>
<td></td>
<td>Intel Win</td>
<td>0</td>
<td>0</td>
<td>197</td>
</tr>
<tr>
<td>Lattice on BOINC</td>
<td>820.32</td>
<td></td>
<td>Intel Linux</td>
<td>900</td>
<td>2500</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Ppc Osx</td>
<td>0</td>
<td>0</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Intel Win</td>
<td>0</td>
<td>0</td>
<td>503</td>
</tr>
<tr>
<td>USM</td>
<td>0.14</td>
<td></td>
<td>Intel Linux</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Sun4u Solaris</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Deepthought</td>
<td>3.15</td>
<td></td>
<td>Intel Linux</td>
<td>0</td>
<td>0</td>
<td>116</td>
</tr>
<tr>
<td>Lattice on BOINC Web Site</td>
<td></td>
<td></td>
<td>Ppc Osx</td>
<td>0</td>
<td>0</td>
<td>44</td>
</tr>
<tr>
<td>SEIL</td>
<td>23.33</td>
<td></td>
<td>Intel Linux</td>
<td>0</td>
<td>0</td>
<td>308</td>
</tr>
<tr>
<td>Bluegrit</td>
<td>1.56</td>
<td></td>
<td>Ppc Linux</td>
<td>0</td>
<td>0</td>
<td>44</td>
</tr>
</tbody>
</table>

**Total Lattice Jobs** Idle 900  Running 2500

**Total Free CPUs**
- Linux 681
- Windows 828
- Mac OS X 112
- Solaris 9

**Grand Total** 1630
Top Five Myths About Condor

- **Myth**: Condor requires users to recompile their applications.
  - **Reality**: Condor runs ordinary, unmodified applications.

- **Myth**: Condor has a single point of failure.
  - **Reality**: Condor has excellent failure isolation.

- **Myth**: Condor is only good at "cycle stealing."
  - **Reality**: Condor can effectively manage many kinds of distributed systems.

- **Myth**: Condor only runs sequential jobs.
  - **Reality**: Condor has extensive support for parallel programming environments.

- **Myth**: Condor doesn't do "Grid" computing.
  - **Reality**: Condor is involved in many forms of distributed computing, including the "Grid."
Designing a Runtime System for Volunteer Computing

David P. Anderson, Carl Christensen, Bruce Allen
BOINC

- BOINC - Berkeley Open Infrastructure for Network Computing
- A platform for volunteer computing
- Popular in the scientific community
- Well established projects include SETI@home, Folding@home, and others
Design Goals

- To attract and retain volunteers
- To handle widely varying applications
- Support for application debugging
- Support for all popular platforms
BOINC Runtime System

- Consists of an application, the core client, the BOINC manager, and an optional BOINC screensaver
BOINC Manager

The BOINC Manager is a software application used to manage distributed computing projects. In the image, the BOINC Manager interface is displayed, showing a list of projects and their status. Each project row includes columns for Project, Application, Name, CPU time, Progress, To completion, and Status. The projects listed are associated with the 'The Lattice Project' and are in various stages of completion, from Ready to start to Running. The application version is HMMPfam 5.04.
BOINC Core Client (CC)

- Can be run as a standalone command line program, or as a service
- Responsible for scheduling applications
- Also checks resource consumption of the running application
- BOINC runtime library allows application to interact with core client
For each application, the CC creates a shared memory segment containing a number of unidirectional message channels.
Applications are threaded (pthreads on UNIX, native threads on Windows)
Compound Applications

- Consists of several programs - typically a coordinator that executes one or more worker programs
CC can perform various operations on running tasks: suspend, resume, quit, abort.

These operations are implemented by sending messages to the process control channel.
Status Reporting

• CC needs to know the CPU time and memory usage of each application every second (or so)

• The BOINC runtime library makes the measurements and reports them through the status channel
Credit Reporting

- By default, credit is computed by multiplying a benchmark score by the application’s total CPU time.
- However, for a number of reasons, this estimate can be erroneous.
- Hence, there is support in the BOINC API for allowing the application to directly compute floating point operations.
• BOINC must run tasks in separate directories, but we want to avoid making unnecessary copies of data

  • boinc_resolve_filename("infile", physical_name);
  • f = boinc_fopen(physical_name, "r");
Checkpointing

- Not absolutely necessary, but extremely helpful when trying to get long-running results back, or when a reliable turnaround time is desired.

- Checkpointing scheme is application specific! Unlike the Condor mechanism...

- BOINC users care about checkpointing immensely (and will harass you indefinitely until you implement it)
Graphics

- Applications supplied graphics are viewable either as a screensaver or in a window
- BOINC runtime library limits the fraction of CPU time used by the graphics thread
Remote Diagnostics

- Application’s standard error is directed to a file and returned to the server for all tasks
- If an application crashes or is aborted, a stack trace is written to standard error
- Problems may occur only with specific OSes, architectures, library versions, etc.
Long-running Applications

- Some projects run tasks that take an extremely long time to complete.
- Besides checkpointing, other mechanisms are necessary to support these tasks - for example, periodically granting users credit, or communicating intermediate results to the server for processing.
- These mechanisms use the trickle messages channel.
Non-CPU Intensive Applications

- Examples include apps that study network structure, dynamics of computer usage, or apps that provide a network service.
- Such tasks are treated specially.
Conclusions

• BOINC is very flexible - it satisfies those who want it to stay out of the way completely, as well as those who really want to be involved in the science

• BOINC supports a wide range of applications and runs on every major platform

• Future plans include making better use of GPUs and multi-core machines
The Lattice BOINC Project

http://boinc.umiacs.umd.edu/
The Lattice Project

http://lattice.umiacs.umd.edu/

The Lattice Project

A community of researchers, scientists, and staff at the University of Maryland are working to integrate and deploy computing resources, Grid middleware, specialized scientific application software and semantic web services in a comprehensive Grid system for scientific analysis. We call this effort The Lattice Project. The lead researchers are:

- Michael Cummings
- Adam Bazinet

A broad spectrum of life science research will benefit greatly from increased access to high performance computing resources. Several properties of modern life sciences research drive the need for high performance computing:

- Trend toward quantitative analysis in experimental biology. The nature of much life sciences research involves inference and hypothesis testing that require computation.
- The rapidly increasing size and number of analysis inputs including databases, samples, and experiments.
- Increasing complexity of analytical models and concomitant increase in possible solution space. Maximum likelihood calculations, Monte Carlo and other stochastic simulation methods increasingly involve more parameters. While such models can increase our understanding of the underlying natural processes under investigation, in many cases they lead to an exponential increase in the number of possible solutions.

Despite promising developments that individually address these issues, the computational environment in which many life scientists work remains substantively deficient, materially impeding progress in a number of fields. It is our goal to make sufficient computational and data resources available to an active community of researchers.

The Lattice Project is developing a community-based Grid system that integrates Grid middleware technologies and widely used life science applications. This system is based on a novel Grid architecture that encompasses resources from high-end clusters and multiprocessors to individual desktop computers. We are strongly committed to the principles of open source software development, and we intend to release all software as freely-available source code except in those very few cases where commercial software is used.