Condor and the Grid

Condor and the Grid
Authors: D. Thain, T. Tannenbaum, and M. Livny

Presenter: Ibrahim H Suslu

CSC 7700
Data Intensive Distributed Computing
Fall 2006

What is Condor?
- Specialized job and resource management system (RMS) for compute intensive jobs
  1. User submit their jobs to Condor
  2. Condor chooses when and where to run them based upon a policy
  3. Condor monitors their progress
  4. Condor informs the user upon completion

Condor Provide
- A job management mechanism
- Scheduling policy
- Priority schema
- Resource monitoring
- Resource management

(like other full-featured systems)

Why Condor?
- High-throughput computing
  - Provide large amounts of fault-tolerant computational power
  - Effective utilization of resource
- Opportunistic computing
  - Use resource whenever available
- ClassAds
  - Resource allocation Language that describe resources and jobs
- Job checkpoint and migration
  - Record a checkpoint and resume the application from it.
  - A checkpoint permit a job to migrate from one machine to other
- Remote system calls
  - Preserve local execution environment

The Philosophy of Flexibility
- Let communities grow naturally
  - Relationships and obligations will develop according to user necessity
- Plan without being picky
  - Be prepared to retry or reassign work when failures come
- Leave the owner in control
  - Happy owners \(\rightarrow\) more resources \(\rightarrow\) higher throughput
- Land and borrow
  - Collaborate with related fields
- Understand previous research

Condor Kernel
Planning and Scheduling

- Planning
  - Acquisition of resources by users
  - Concerned with ‘what’ and ‘where’
- Scheduling
  - Management of a resource by its owner
  - Concerned with ‘who’ and ‘when’

Matchmaker

- Bridge between planning and scheduling
- Agents and resources advertise characteristics and requirements as ClassAds
- Pairs satisfying each other’s constraints are created
- Both parties are informed
- Claiming- independent authorization and authentication
### ClassAds

- **Resource allocation Language**
  - Attribute name-value pairs
  - No specific schema
- **Requirements**
  - Constraints, for a match these should evaluate to true
- **Rank**
  - Desirability of a match

#### Job ClassAd

- MyType = "Job"
- TargetType = "Machine"
- Requirements = Machine="tnt.isi.edu"
- Rank = dept==self.dept

#### Machine ClassAd

- MyType = "Machine"
- TargetType = "Job"
- Requirements = Machine="tnt.isi.edu"
- Rank = (Memory ∗ 10000) + KFlops
- Arch="Intel"
- Cmd = "/home-exe"
- OpSys="Linux"
- Department = "CompSci"
- Disk=60000
- Owner = "tannenba"
- DiskUsage = 6000

### Problem Solvers

- High level structure built on top of the Condor agent
- Manage large number of jobs
  - Concern with the application-specific details of ordering and task selection
- Relies on a Condor agent in two ways
  - Uses agent as service for reliably executing jobs
  - Making the problem solver itself reliable
- Two are provided with Condor
  - Master-worker (MW)
    - System for solving a problem of indeterminate size on a large and unreliable workforce
    - Directed acyclic graph manager (DAGMAN)
  - Split Execution
    - Service for executing multiple jobs with dependencies in a declarative form

### Split Execution

- Facilitates successful remote execution of jobs
- Shadow represents the user to the system
  - Has information that specifies the job at run time
    - Sandbox, arguments, input files...
- Sandbox is responsible for giving the job a safe place to play
  - Creates an environment for job execution
- A Matched Sandbox and Shadow form the universe

### Condor Universes

- Create a specific job environment
- Defined by a matched sandbox and shadow
- Different Universes provide different functionality for your job:
  - **Standard** /barb2right
    - Support for transparent process checkpoint and restart
  - **Vanilla** /barb2right
    - Run any Serial Job
  - **Java** /barb2right
    - Provide a complete Java environment
  - **Globus** /barb2right
    - Manage your Grid jobs

### Standard Universe

- Requires re-linking your program with special library provided by condor
- Allows checkpointing and remote System Calls
  - **Checkpointing**
    - Condor’s Process Checkpointing mechanism saves all the state of a process into a checkpoint file
    - Memory, CPU, I/O, job details, etc.
    - The process can then be restarted from right where it left off
  - **Remote System Calls**
    - Provides an I/O service over secure RPC channel
    - Provides remote access to the user’s home storage device
    - Multi-process jobs are not allowed
    - Interprocess communication is not allowed

### Vanilla Universe

- You can run any program
  - C/C++/Perl/Python/Fortran/Java/Lisp...
  - No checkpointing: if your job is interrupted or the machine crashes, Condor has to restart it from the beginning.
  - No remote system calls
    - Input and output files
Java Universe

- Works better for Java programs
- Checks for valid Java environment
- Distinguishes Java environment exceptions from program exceptions
- No checkpointing
- Remote I/O

Globus Universe

- Advantages of using Condor-G to manage your Grid jobs
  - Full-featured queuing service
  - Credential Management
  - Fault-tolerance
- Disadvantages
  - No matchmaking or dynamic scheduling of jobs
  - No job checkpoint or migration
  - No remote system calls

Condor-G

- Computation management agent for Grid Computing
  - Merges Globus and Condor technologies

- Application, problem solver...
- Job submission
  - Condor-G
  - Globus Toolkit
  - Resource discovery, authentication

- Job execution
  - Condor
  - Processing, storage...

Which Universe?

- Standard:
  - Good for mixed Condor pools, flocked pools, and the Grid at large.
- Vanilla:
  - Good for a Condor pool of identical machines
- Java:
  - Good for Java application
- Globus:
  - Good for Globus jobs

Access to Data in Condor

- Use shared filesystem if available
- No shared filesystem?
  - Condor can transfer files
    - Can automatically send back changed files
    - Atomic transfer of multiple files
    - Can be encrypted over the wire
  - Remote I/O Socket
  - Standard Universe can use remote system calls
Example: Nug30

- nug30 (a Quadratic Assignment Problem instance of size 30) had been the “holy grail” of computational QAP research for > 30 years
- In 2000, Anstreicher, Brixius, Goux, & Linderoth set out to solve this problem
- Using a mathematically sophisticated and well-engineered algorithm, they still estimated that we would require **11 CPU years** to solve the problem.

Nug30 solved

<table>
<thead>
<tr>
<th>Wall Clock Time</th>
<th>6 days 22:04:31 hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg # Machines</td>
<td>653</td>
</tr>
<tr>
<td>CPU Time</td>
<td>11 years</td>
</tr>
<tr>
<td>Parallel Efficiency</td>
<td>93%</td>
</tr>
</tbody>
</table>

Questions