Condor: Features and Technologies

Presented by

Rajesh Raman

(raman@cs.wisc.edu)

Condor Team, University of Wisconsin–Madison

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

February 20, 1998

Outline

- Introduction to Condor
- Condor Features
- Failure Recovery
- Monitoring and Control
- Condor and PVM
- Summary of Available technologies
- Summary of Future Work

Introduction

Condor: High Throughput Computing with Distributively Owned Resources

- HPC vs HTC: Consider $Power = Work/Time$
  - HPC: Latency oriented
    - Fixed amount of work; how much time?
  - HTC: Throughput oriented
    - Unbounded amount of work, fixed amount of time; how much work?
- Distributed Ownership
  - Resource owners dictate usage policy
- Matchmaking model for resource management
  - (Advertise, Match, Claim) instead of centralized allocation
  - End-to-end resource management

Introduction: Condor Architecture

- Resource Agents (RA's): Represent resource owners
  - Implement owner's policy
  - Advertise state and availability of resource
- Customer Agents (CA's): Represent customers of the system
  - Maintain a fault tolerant queue of requests
  - Advertise resource requirements
- Matchmaker: Brings suppliers and customers together
  - Matches offer and request advertisements
  - Match is a hint, not an allocation
  - Allocation occurs when CA and RA agree to cooperate (i.e., after claiming)

Introduction: Classified Advertisements

Classified Advertisements (classads): a language for expressing attributes, and a semantics for evaluating them
Attributes are named expressions composed of

- Constants: integer, real, string, undefined, error, list and classad
- Calls to inbuilt utility functions
- Attribute references: May refer to attributes outside "hosting" classad through explicit scope resolution
- Operators: Unary, binary and ternary
  - Mostly strict with respect to "undefined" and "error"
  - Non strict operators: Logical AND `&&`, logical OR `||`, if-expressions `?:`, identical `is`, not identical `isnt`

Introduction: Classad Examples

```
[
  Type        = "Machine";
  Activity    = "Idle";
  DayTime     = 36107     // current time (secs since midnight)
  KbdIdle     = 1432;     // seconds
  Disk        = 323496;   // kbytes
  Memory      = 64;       // megabytes
  State       = "Unclaimed";
  LoadAvg     = 0.042969;
  Arch        = "INTEL";
  OpSys       = "SOLARIS251";
  Name        = "leonardo.cs.wisc.edu";
  RsrchGrp    = { "raman", "miron", "solomon", "jbasney" };
  Friends     = { "tannenba", "wright" };
  Untrusted   = { "rival", "riffraff" };
  Rank        = member(other.Owner, RsrchGroup) * 10
                 + member(other.Owner, Friends);
  Constraint = !member(other.Owner, Untrusted)
                 && (Rank == 10 ? true
                  : Rank > 0 ? (LoadAvg < 0.3 && KbdIdle > 15*60)
                  : (DayTime < 8*60*60 || DayTime > 18*60*60)))
]
```

Condor Features

- Resource owner features
  - Customizable usage policy
- Customer features
  - Sophisticated constraints on required resources
  - Transparent access to submission environment
    * Uniform file system not required
    * Uniform UID space not required
  - Application Checkpointing
    * Guarantees forward progress
    * Provides fault tolerance
  - Support for multiple application universes
    * Vanilla, Standard, PVM, Scheduler

Failure Recovery

Recovery in Condor

- Recovery of Condor System
  - Robust protocols
  - Stateless matchmaker
  - Persistent request queues (i.e., persistent classad store)
- Recovery of Condor Applications
  - Rollback to most recent checkpoint

Available Application Level Technologies

- Application triggered checkpointing
- Write-Ahead Logging interface (C++)
- Persistent classad store
  - Includes transaction semantics

Monitoring and Control

- Monitoring status of resources and requests
  - API's for querying resource and request classads
  - Can specify arbitrary constraints in queries
- Monitoring application progress
  - Condor can write an event log describing application status
  - API for reading application event logs
- Tools to control application progress
  - Submit, remove, vacate, checkpoint, throttle
  - Work in progress: Converting the above tools into API's

Aside: Condor Universes

- Every request submitted to Condor belongs to a universe
- Universe defines request's resource management environment
- Current available universes
  1. Vanilla: Prelinked (third party) applications; no checkpointing, no remote system calls
  2. Standard: Relinked applications; checkpointing and remote system calls
  3. PVM: For PVM applications (to be discussed shortly)
  4. Scheduler: For application level scheduling
     - Submitted like ordinary Condor requests
     - Application runs locally
     - Guaranteed to be restarted on failure

Monitoring and Control Example: DagMan

An example of external monitoring and control in Condor: DagMan

- DagMan schedules applications which have dependencies
- Dependencies expressed as a dag
- Each node in the dag is an application instance

DagMan uses

- Submit tool for submitting requests to Condor
- Event log API for monitoring progress of applications
- Write-Ahead Logging interface for persistence

Basically, a master-worker paradigm

Condor and PVM

- PVM Master-Worker applications work best in Condor
- Master runs locally
  - Requests hosts
  - Spawns workers on claimed hosts
  - Monitors workers
- Workers are dynamic
  - Exit due to task completion
  - May be evicted if resource becomes unavailable
- Master is responsible for detecting incomplete work items, and reallocating workers as needed

Differences between PVM and Condor/PVM

- Condor registers itself as the resource manager for the application
  - Resource management requests routed through Condor
- The `add_host()` call is asynchronous
  - Cannot be synchronous (blocking) since time to allocation is unbounded
  - Notification obtained when host is added
- Additional notifications sent
  - `host_suspend`, `host_resume`
- Contact Bin Song (bsong@cs.wisc.edu) for more information

Summary: Available Technologies

- Write-Ahead Logging Interface
- Persistent Classad Store (with transaction semantics)
- API's for
  - Querying resources and requests
  - Application triggered checkpointing
  - Reading event logs
- Tools for externally triggered checkpointing, vacating resource, submitting requests, removing requests, etc.
- PVM

Summary: Future Work

- API's
  - Conversion of existing tools into API's
  - CARMI: Condor Application Resource Management Interface
- PVM
  - Integration with classads
  - Checkpointing workers
- MPI-2