Condor

Smruti R. Sarangi

Department of Computer Science
Indian Institute of Technology
New Delhi, India

Outline

1. Overview
2. Design of Condor
   - Main Modules
   - Condor Pools
   - Match Making
   - Problem Solver
3. Detailed Operation
   - Universes
   - Data Intensive Computing
   - Security

History of Condor

- Towards the mid 80s, the power of distributed computing was realized
- Clusters of machines could outperform supercomputers
- There was a need for a middleware to integrate third party computers
    - Integrate computers with different types of hardware and software
    - Provide consistency and reliability guarantees
    - Provide security, and trust
    - Ensure fairness among users
    - Be able to efficiently run large scale distributed jobs.

Condor was thus born in the University of Wisconsin

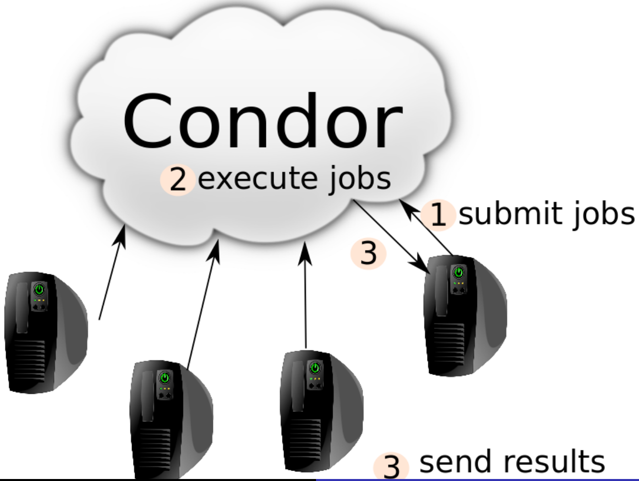
Philosophy of Condor

1. Flexibility
2. Let communities grow naturally – Build software that permits co-operation among users.
3. Leave the owner of the computing resource in control.
4. Make the system fault tolerant
5. Lend and borrow from other disciplines.

Condor High Throughput Computing System

- Condor provides a method for a set of users to submit their jobs in batch mode.
- Condor provides:
    - Job Management Mechanisms
    - Scheduling Policies
    - Resource Monitoring
    - Resource Management

View of Condor



Outline

1 Overview

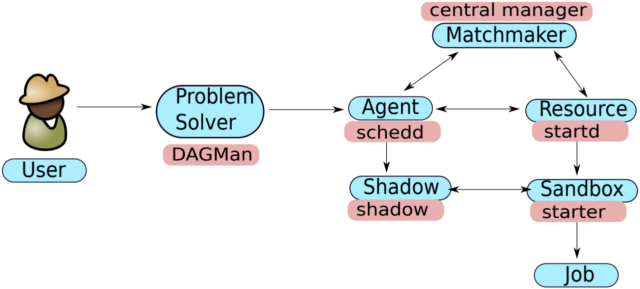
2 Design of Condor
- Main Modules
- Condor Pools
- Match Making
- Problem Solver

3 Detailed Operation
- Universes
- Data Intensive Computing
- Security

Main Modules in Condor

- ClassAds System : This is a language that lets users specify the type of the job, the type of the resource offered to the cloud, and the matching policies.
- Execution Engine : Executes user jobs (respects DAG based constraints) on a large grid.
- Job Checkpoint and Migration Can transparently checkpoint jobs, and can migrated them among machines. For example, if a user on an idle desktop presses a key, then any Condor job running on it seamlessly migrates to another machine.
- Remote Sandbox : All I/O related system calls are redirected to the machine that submitted the job.

View of the Condor Kernel



Flow of Actions in Condor

- User submits a job to the DAGMan manager. It parses the DAG structure of jobs, and sends it to an Agent .
- Agent : It stores the jobs in persistent storage, and finds resources to run them.
- Agents and resources periodically send messages to a dedicated MatchMaker . It pairs agents with resources.
    - Once the matchmaker reports a match, the agent checks with the resource if it is still available.
    - The agent spawns a process called a shadow to handle the execution of the job.
    - The resource creates a sandbox to run the job.

Outline

1. Overview
2. Design of Condor
   - Main Modules
   - Condor Pools
   - Match Making
   - Problem Solver
3. Detailed Operation
   - Universes
   - Data Intensive Computing
   - Security

Condor Pools

Condor Pools

- Pools of machines (agents/resources) can get together and form a Condor pool.
- Every pool has one matchmaker.
- A resource can enforce some policies regarding the type of resource offered, and the type of agents it will accept.
- The matchmaker can enforce additional policies.
- Users in the mid nineties expressed the desire to access machines from remote pools also.

Gateway Flocking

Gateway Flocking

- Every pool will have a gateway that can interact with gateways of other remote pools.
- If a pool has an idle machine, then its gateway can send its advertisement to other gateways.
- They can forward this information in their local pools.

Direct Flocking

- An agent reports itself to multiple matchmakers, and effectively joins multiple pools.

Interaction with Globus

- Direct and gateway flocking are complicated.
- In the late nineties, the Globus toolkit emerged:
  - It was a standard architecture to interconnect clusters and grids.
  - Provided trust, security, and secure file transfer services.
  - GRAM Protocol: Grid Resource Access and Management
  - Condor interacts with GRAM using a dedicated module called Condor-G.

Outline

1. Overview
2. Design of Condor
   - Main Modules
   - Condor Pools
   - Match Making
   - Problem Solver
3. Detailed Operation
   - Universes
   - Data Intensive Computing
   - Security

Match Making

Overview of Match Making

1. Agents and resources advertise their details using small snippets of text called ClassAds .
2. The matchmaker pairs agents and resources.
3. The agent then goes and **claims** the resource.

Examples of ClassAds

```
Job ClassAd
[
MyType = "Job"
TargetType = "Machine"
Requirements = ((other.Arch =="INTEL" && other.OPSy
Rank = (Memory * 10000) + KFlops
Cmd = "abc/abc.exe"
Owner = "myself"
]
```

- “Requirements” indicates the constraints
- “Rank” is the objective function of the match
- Among the available resources, the matchmaker chooses the highest rank

Enhancements to Matchmaking

- Support for writing custom Java and C modules
- Gang matching – coallocation of more than one resource (machine and license)
- Collections provide database support for saving ClassAds
- Set matching involves selecting a large number of classads
- Named references permit one classAd to refer to another one.

Outline

1. Overview
2. Design of Condor
   - Main Modules
   - Condor Pools
   - Match Making
   - Problem Solver
3. Detailed Operation
   - Universes
   - Data Intensive Computing
   - Security

Problem Solver – Master-Worker Mode

- Master-Worker Mode has one master process the directs of the work of many worker processes
- The master contains
  - work-list : Record of all the outstanding work that needs to be performed
  - tracking-module : Keeps track of remote processes, and allots them work items.
  - steering-module : Examines the results of workers, modifies the work lists, and co-ordinates with Condor
- Workers can die at any time. The tracking module then returns them to the work list.
- The tracking module can replicate work items (work item should not have side effects)

Problem Solver – DAGMan

- Jobs are specified as a DAG (directed acyclic graph)
- Pre and post processing supported
- If a given job fails (because of the system or because of a bug)
  - DAGMan prints a rescue DAG
- It is possible to have a RETRY command

Outline

1. Overview
2. Design of Condor
   - Main Modules
   - Condor Pools
   - Match Making
   - Problem Solver
3. Detailed Operation
   - Universes
   - Data Intensive Computing
   - Security

Shadow and Sandbox

- The Shadow is responsible for communicating the requirements of the job to the resource
  - Input files
  - Network connections
  - Database connections
  - Executable, arguments, environment
- A resource creates a sandbox
  - It needs to create the appropriate environment for the job.
  - Needs to ensure that the job cannot harm the host
  - Needs to ensure that the host cannot harm the job
  - In some cases, it needs to marshal I/O data

Shadow and Sandbox

- The Shadow is responsible for communicating the requirements of the job to the resource
  - Input files
  - Network connections
  - Database connections
  - Executable, arguments, environment
- A resource creates a sandbox
  - It needs to create the appropriate environment for the job.
  - Needs to ensure that the job cannot harm the host
  - Needs to ensure that the host cannot harm the job
  - In some cases, it needs to marshal I/O data

Universe: Matching sandbox and shadow pair

Standard Universe

- Emulates a standard Unix envirnoment
- Provides support for I/O marshalling
  - The shadow runs an I/O server. It takes requests from the running job, satisfies the request at the home file system, and returns the data.
  - At compile time, user code needs to be linked with Condor libraries. They wrap the I/O system calls, and convert them to RPCs.
  - It is possible to define a virtual file system using this mechanism (how???)
- Provides support for checkpointing

Java Universe

- The sandbox creates an environment with a Java virtual machines.
- It places all necessary class and archive files in the job's classpath.
- The job is linked against a Java I/O library
  - Uses a proxy I/O interface
    - Can authenticate and pass through firewalls
    - Compatible java.io.InputStream and OutputStream

Outline

1. Overview
2. Design of Condor
   - Main Modules
   - Condor Pools
   - Match Making
   - Problem Solver
3. Detailed Operation
   - Universes
   - Data Intensive Computing
   - Security

Data Intensive Computing

- Massive amounts of data processing can be done on Condor – biological, simulation, scientific
- Create new resource manager called Nest
- Condor implemented a new file transfer agent called Stork that can synchronize large file transfers
- Using a variety of protocols – http, ftp, and Nest, Stork communicates with Nest
- To smooth out very large data transfers, Condor adds a series of Disk Routers
- A new module called Parrot helps Condor communicate with all kinds of unusual storage devices.

Outline

1. Overview
2. Design of Condor
   - Main Modules
   - Condor Pools
   - Match Making
   - Problem Solver
3. Detailed Operation
   - Universes
   - Data Intensive Computing
   - Security

Security

- Secure Communication
  - Condor uses a secure communication library called Cedar
  - Cedar is a wrapper for SASL, Kerberos, and other authentication protocols
- Secure Execution
  - Users are given a restricted login at the resource (no chroot feature)
  - Condor can either use the Unix *nobody* account
  - Even better, Condor dynamically assigns a user id to a job
  - Possible to set a domain of users, such that users have same permissions in all machines in a workgroup
- Condor has a cleanup feature that kills all processes.

- Distributed Computing in Practice: The Condor Experience, Douglas Thain, Todd Tanenbaum, Miron Livny, Concurrency and Computation: Practice and Experience – Grid Performance, Volume 1, Issue 2-4, February, 2005