Condor and the Grid

Douglas Thain
Todd Tannenbaum
Miron Livny
University of Wisconsin-Madison

Slides based on those by Koh Kwangwon
Supercomputing Lab, Yonsei Univ, Korea

Grid Computing – Making the Global Infrastructure a Reality
Fran Berman, Geoffrey Fox, and Tony Hey
2002 John Wiley & Sons, Ltd.
Chapter 11

Introduction

- Distributed computing would be difficult
  - When messages may be lost, corrupted, or delayed, precise algorithms must be used in order to build an understandable system
- “Can we satisfy the needs of users who need extra capacity without lowering the QoS experiences by the owners of under utilized workstations?”
  - Condor??

The Philosophy of Flexibility

- Flexibility is the key to surviving in a hostile environment
  - Large scale, Heterogeneous
- Let communities grow naturally
  - They aim to build structures that permit but do not require cooperation
- Plan without being picky
  - An over dependence on the correct operation of any remote device is a recipe for disaster
  - Spend more time contemplating the consequences of failure than the potential benefits of success
- Leave the owner in control
- Lend and borrow
- Understand previous research
The Condor Project Today

- Research
  - Harnessing the power of opportunistic and dedicated resources (Condor)
  - Job management services for grid applications (Condor-G)
  - Fabric management services for grid resources (Condor, GlideIn)
  - Resource discovery, monitoring, and management (ClassAds)
  - Problem solving environments (MW, DAGMan)
  - Distributed I/O technology (Bypass, PFS, Kangaroo, Nest)
- Participation in Community
  - GriPhyN, iVDGL, PPDG, NSF NMI, TeraGrid
- Engineering of Complex Software
  - Mission critical software
  - Each release subject to 200+ test automatic regression suite
- Maintenance of production environments
  - 1000 CPUs at CS dept in U. Wisconsin-Madison

The Condor Software: Condor and Condor-G

- Condor: A System for High Throughput Computing
  - A special job and resources management system (RMS) for compute-intensive jobs
    - Job queueing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management
  - Condor’s novel architecture and unique mechanisms allow it to perform well in environments where a traditional RMS is weak
  - High-throughput computing: large amounts of fault tolerant computational power over prolonged periods of time by effectively utilizing all resources available to the network
  - Opportunistic computing: the ability to utilize resources whenever they are available, without requiring one hundred percent availability

The Condor Software: Condor and Condor-G

- Some of the enabling mechanisms of Condor
  - ClassAds – to match resource requests with resource offers
  - Job Checkpoint and Migration
  - Remote System Calls
    - Part of the mobile sandbox environment for directing I/O to originating workstation
- Condor-G: A Computation Management Agent for Grid Computing
  - Represents the marriage of technologies from the Condor and Globus projects
  - From Condor: job submission, job allocation, error recovery, and creation of a friendly execution environment
  - From Globus: the use of protocols for secure inter-domain communications and standardized access to a variety of remote batch systems

The Layers of Condor

- Application
- Application Agent
- Customer Agent
- Matchmaker
- Owner Agent
- Remote Execution Agent
- Local Resource Manager
- Resource
- Submit (client)
- Execute (service)
The Condor Software: Condor and Condor-G

- Condor-G: the reliable submission and job management service for one or more sites
- Condor: the fabric management service (a grid “generator”)
- Globus Toolkit: the bridge between Condor-G and Condor

A History of Computing Communities

- Condor ca. 1987
  - Agents and resources independently report information about themselves to a well-known matchmaker, which then makes the same information available to the community
    - Both an agent and a resource daemon are running on the same machine
    - Agents and resources are logically distinct
    - Did not answer need to share across organizational boundaries
A History of Computing Communities

- Gateway Flocking ca. 1994
  - Gateways advertise idle resources to peers
  - Advantage: completely transparent to participants
  - Significant limitations
    - The accounting of use by individual remote users is essentially impossible because of represent through a single gateway machine
    - Only allow sharing at the organizational level

- Direct Flocking ca. 1998
  - To overcome the limitation that the gateway flocking only allows sharing at the organizational level
  - Comparison
    - Gateway Flocking
      - Require agreement at the organizational level
      - Provides immediate and transparent benefit to all users
      - Gateway participates in every interaction in Condor kernel
    - Direct Flocking
      - Only requires agreement between one individual and another organization
      - Only benefits the user who takes the initiative

A History of Computing Communities

- Problem of Flocking
  - Gateway Flocking: complex
  - Direct Flocking: less powerful, but simple design

- Condor G ca. 2000
  - To take advantage of GRAM - uniform interface to batch systems
  - Needed to add durability and two-phase commit to GRAM to prevent loss/repetition of jobs
  - If job fails, the system must analyze the failure and re-submit the job if necessary
  - Need queuing, prioritization, logging, and accounting

- Some disadvantage
  - Condor-G couples resource allocation and job execution
    - Jobs have to be submitted without knowledge of availability/load on resources
  - Condor-G does not support all of the varied features of each batch system underlying GRAM
A History of Computing Communities

Gliding In

Step One:
User submits Condor jobs as batch jobs on foreign systems.

Step Two:
Submitful redirections form an ad-hoc personal Condor pool.

A History of Computing Communities

A History of Computing Communities

Condor-G and Gliding In ca. 2001
- To solve these problems
- I/O communities
  - Resources may group themselves together to express that they are “nearby” in measurable properties such as network latency or system throughput

Planning, Scheduling and Matchmaking

- Scheduling
  - Somebody must decide how to allocate resources to jobs because of more requests than available resources

- Planning: the acquisition of resources by users
  - Increasing personal metrics such as response time, turnaround time, and throughput of their own job within reasonable costs

- Scheduling: the management of a resource by its owner
  - Increasing system metrics such as efficiency, utilization, and throughput without losing the customers they intend to serve

- Feedback between planning and scheduling, is important!!
  - Matchmaker!!!

Planning, Scheduling and Matchmaking

- Condor-G and Gliding In ca. 2001
- I/O communities
- Resources may group themselves together to express that they are “nearby” in measurable properties such as network latency or system throughput

Planning, Scheduling and Matchmaking

- Scheduling
  - Somebody must decide how to allocate resources to jobs because of more requests than available resources

- Planning: the acquisition of resources by users
  - Increasing personal metrics such as response time, turnaround time, and throughput of their own job within reasonable costs

- Scheduling: the management of a resource by its owner
  - Increasing system metrics such as efficiency, utilization, and throughput without losing the customers they intend to serve

- Feedback between planning and scheduling, is important!!
  - Matchmaker!!!
Planning, Scheduling and Matchmaking

• No specific schema
• Attributes may not be defined
• Class-Ads use 3-valued logic
  – True, false, undefined
• Special attributes
  – Requirements: indicate constraint and must evaluate to true
  – Rank: evaluates to fp number indicating rank among compatible choices

Planning, Scheduling and Matchmaking

• Matchmaking in Practice
  – More information on availability times and policies allow better planning of Condor job submissions
  – First implementation: control expressions (1992)
  – ClassAds is available in both Java and C++
  – Being replaced by new implementation (2000)
  – Other concepts
    • Gang matching: coallocation of resources
    • Collections: persistent storage for ClassAds with db features
    • Set Matching: claim large number of resources concisely
    • Indirect reference: to Class Ads

Problem Solvers

• Program Solver
  – A higher-level structure built on top of the Condor agent.
  – master-worker, directed acyclic graph manager
  – A program solver uses the agent as a service for reliably executing jobs
  – The program solver is presented as a normal Condor job which simply executes at the submission site

Problem Solvers

• Master-Worker (MW)
  – Work List: a record of all outstanding work the master wishes to be done
  – Tracking: accounts for remote worker processes and assigns them uncompleted work
  – Steering: directs the computation by examining results, modifying the work list, and communicating with Condor to obtain a sufficient number of worker processes
Problem Solvers

- Directed Acyclic Graph Manager (DAGMan)
  - A service for executing multiple jobs with dependencies in a declarative form
  - PRE and POST jobs not Condor jobs but run on submitting machine
  - A distributed, fault-tolerant version of the traditional make(?)
    - But it does not depend on the file-system
    - Read about multilevel error handling

Split Execution

- The techniques of getting a job to an appropriate execution site
- Shadow provides everything needed to specify the run-time job
  - Executable, environment, input files etc
- Sandbox responsible for creating a safe place to play
  - Asks shadow for details and creates environment

- Standard Universe – POSIX environment
  - Process creation & IPC not supported
  - Provides checkpointing
  - Sandbox creates temp directory and fetches job details there
  - Job must be relinked with Condor libraries – same API as C library
    - SecureRPC, HTTP, GridFTP, Nest, and Kangaroo can be used
    - Shadow remains in control of the entire operation
      - Neither the sandbox or the Condor library is permitted to simply open a file by name
      - Maximizes the flexibility of the user to make run-time decisions about exactly what runs where and when
Split Execution

The Java Universe

Case Study - Read
C.O.R.E. Digital Pictures Inc.

Conclusion

• They believe the key to lasting system design is to outline structure first in terms of responsibility rather than expected functionality

• Look forward to the challenges ahead!