Distributed Computing in Practice: The Condor Experience

by Douglas Thain, Todd Tennebaum and Miron Livny
Presented by Shiran Dudy
28/05/14
Outline

• Introduction

• aspects of Condor
Frieda’s Application ...

Run a Parameter Sweep of $F(x, y, z)$ for 20 values of $x$, 10 values of $y$ and 3 values of $z$
- $20 \times 10 \times 3 = 600$ combinations
- $F$ takes on the average 6 hours to compute on a “typical” workstation ($total = 600 \times 6 = 3600$ hours)
- $F$ requires a “moderate” (256MB) amount of memory
- $F$ performs “moderate” I/O - $(x, y, z)$ is 5 MB and $F(x, y, z)$ is 50 MB
Intro

I have 600 simulations to run.
Where can I get help?
The Philosophy of Flexibility

• Let communities grow naturally
• Leave the owner in control, whatever the cost
• Plan without being picky
• Lend and borrow
The Condor Software

- The condor high throughput computing system
  - ClassAds
  - Job checkpoint and Migration
  - Remote system calls
- Condor-G: An agent for grid computing
The Condor Software

- **The condor high throughput computing system**
  - ClassAds
  - Job checkpoint and Migration
  - Remote system calls

- **Condor-G: An agent for grid computing**
The Condor Software

- The condor high throughput computing system
  - **ClassAds**
  - Job checkpoint and Migration
  - Remote system calls
- Condor-G: An agent for grid computing
The Condor Software

- The condor high throughput computing system
  - ClassAds
  - Job checkpoint and Migration
  - Remote system calls
- Condor-G: An agent for grid computing
The Condor Software

• The condor high throughput computing system
  • ClassAds
  • Job checkpoint and Migration
  • Remote system calls

• Condor-G: An agent for grid computing
The Condor Software

• The condor high throughput computing system
  • ClassAds
  • Job checkpoint and Migration
  • Remote system calls
• Condor-G: An agent for grid computing
The Condor Software

- The condor high throughput computing system
  - ClassAds
  - Job checkpoint and Migration
  - Remote system calls

- **Condor-G: An agent for grid computing**
The Condor Software

- Condor-G: An agent for grid computing
An Architectural History of Condor

Figure 2. The Condor Kernel  This figure shows the major processes in a Condor system. The common generic name for each process is given in large print. In parentheses are the technical Condor-specific names used in some publications.
Figure 4. Gateway Flocking ca. 1994 An agent (A) is shown executing a job on a resource (R) via a gateway (G). Step 1: The agent and resource advertise themselves locally. Step 2: The gateway forwards the agent’s unsatisfied request to Condor Pool B. Step 3: The matchmaker informs the two parties that they are potentially compatible. Step 4: The agent contacts the resource and executes a job via the gateway.
An Architectural History of Condor

Figure 5. **Condor World Map ca. 1994** This is a map of the worldwide Condor flock in 1994. Each dot indicates a complete Condor pool. Numbers indicate the size of each Condor pool. Lines indicate flocking via gateways. Arrows indicate the direction that jobs may flow.
Figure 4. Gateway Flocking ca. 1994  An agent (A) is shown executing a job on a resource (R) via a gateway (G). Step 1: The agent and resource advertise themselves locally. Step 2: The gateway forwards the agent’s unsatisfied request to Condor Pool B. Step 3: The matchmaker informs the two parties that they are potentially compatible. Step 4: The agent contacts the resource and executes a job via the gateway.

what’s the problem here?
An Architectural History of Condor

Figure 6. Direct Flocking ca. 1998  An agent (A) is shown executing a job on a resource (R) via direct flocking. Step 1: The agent and the resource advertise themselves locally. Step 2: The agent is unsatisfied, so it also advertises itself to Condor Pool B. Step 3: The matchmaker (M) informs the two parties that they are potentially compatible. Step 4: The agent contacts the resource and executes a job.
An Architectural History of Condor

Figure 10. Condor World Map ca. 2004 Each dot indicates an independent Condor pool. The area covered by each dot is proportional to the number of machines in the pool. The location of each pool is determined by the top-level country domain name of the matchmaker, if present, or otherwise from public WHOIS records. Each dot is scattered by a small random factor, thus some appear to fall in the sea. A small number of pools that could not be mapped are plotted in the South Pacific.
Planning and Scheduling

Figure 11. Matchmaking
Planning and Scheduling

Job ClassAd

```
[  
  MyType = "Job"
  TargetType = "Machine"
  Requirements = 
    ((other.Arch=="INTEL" &&
      other.OpSys=="LINUX")
     && other.Disk > my.DiskUsage)
  Rank = (Memory * 10000) + KFlops
  Cmd = "/home/tannenba/bin/sim-exe"
  Department = "CompSci"
  Owner = "tannenba"
  DiskUsage = 6000
]
```

Machine ClassAd

```
[  
  MyType = "Machine"
  TargetType = "Job"
  Machine = "nostos.cs.wisc.edu"
  Requirements = 
    (LoadAvg <= 0.300000) &&
    (KeyboardIdle > (15 * 60))
  Rank = other.Department==self.Department
  Arch = "INTEL"
  OpSys = "LINUX"
  Disk = 3076076
]
```

Figure 12. Two Sample ClassAds from Condor.
Problem Solver

Figure 13. Structure of a Master-Worker Program
Problem Solver

Figure 14. A Directed Acyclic Graph

JOB A a.condor
JOB B b.condor
JOB C c.condor
JOB D d.condor
JOB E e.condor
PARENT A CHILD B C
PARENT C CHILD D E
SCRIPT PRE C in.pl
SCRIPT POST C out.pl
RETRY C 3
Split Execution

Figure 15. The Standard Universe
Split Execution

Figure 16. Two-Phase Open Using the Shadow
Data intensive Computing Tools
Let's move on to Scalability...
MPI for runners

- Message Passing Interface
  - Most useful on distributed memory machines
  - The *de facto* standard parallel programming interface

Many implementations, interfaces in C/C++, Fortran, Python via MPI4Py

https://www.youtube.com/watch?v=hlCr8AuEuSc
https://www.youtube.com/watch?v=Udn9wmmb9YY
MPI for runners

- A parallel MPI program is launched as separate processes (tasks), each with their own address space.
  - Requires partitioning data across tasks.
- Data is explicitly moved from task to task
  - A task accesses the data of another task through a transaction called “message passing” in which a copy of the data (message) is transferred (passed) from one task to another.
- There are two classes of message passing
  - Point-to-point involve only two tasks
  - Collective messages involve a set of tasks
MPI for runners

- MPI4Py provides an interface very similar to the MPI standard C++ interface
- You can communicate Python objects.
- What you may lose in performance, you gain in shorter development time
MPI for runners

- MPI uses communicator objects to identify a set of processes which communicate only within their set.
- `MPI.COMM_WORLD` is defined as all processes (ranks) of your job.
  - Usually required for most MPI calls
- Rank
  - Unique process ID within a communicator
  - Assigned by the system when the process initializes
  - Used to specify sources and destinations of messages

```python
#!/usr/bin/env python
from mpi4py import MPI
comm = MPI.COMM_WORLD
print "Hello, World! My rank is: " + str(comm.rank)
```
MPI for runners

demos of trapezoidal rule: serial and point to point
MPI for runners