A Self-Organizing Flock of Condors

Ali Raza Butt
Rongmei Zhang
Y. Charlie Hu
{butta, rongmei, ychu}@purdue.edu
The need for sharing compute-cycles

- Scientific applications
  - Complex, large data sets

- Specialized hardware
  - Expensive

- Modern workstation
  - Powerful resource
  - Available in large numbers
  - Underutilized

=> Harness idle-cycles of network of workstations
Condor: High throughput computing

- Cost-effective idle-cycle sharing
- Job management facilities
  - Scheduling, checkpointing, migration
- Resource management
  - Policy specification/enforcement
- Solves real problems world-wide
  - 1200+ machines Condor pools, 100+ researchers @Purdue
Sharing across pools: Flocking

Central manager
• Static flocking requires
  – Pre-configuration
  – Apriori knowledge of all remote pools
    • Does not support dynamic resources
Our contribution: Peer-to-peer based dynamic flocking

• Automated remote Condor pool discovery

• Dynamic resource management
  – Support dynamic membership
  – Support changing local policies
Agenda

- Background: peer-to-peer networks
- Proposed scheme
- Implementation
- Evaluation
- Conclusions
Overlay Networks

P2P networks are self-organizing overlay networks without central control.
Advantages of structured p2p networks

- Scalable
- Self-organization
- Fault-tolerant
- Locality-aware
- Simple to deploy

- Many implementations available
  - E.g. Pastry, Tapestry, Chord, CAN…
Pastry: locality-aware p2p substrate

- 128-bit circular identifier space
  - Unique random nodeIds
  - Message keys

- Routing: A message is routed reliably to a node with nodeId numerically closest to the key

- Routing in overlay < 2 * routing in IP
Agenda

- Background: peer-to-peer networks
- Proposed scheme
- Implementation
- Evaluation
- Conclusions
Step 1:  
*P2p organization of Condor pools*

- Participating central managers join an overlay
  - Just need to know a single remote pool

- P2p provides self-organization
  - Pools can reach each other through the overlay
  - Pools can join/leave at anytime
P2p organized central managers
Step 2: Disseminating resource information

- Announcements to nearby pools
  - Contain pool status information
  - Leverage locality-aware routing table
    - Routing table has $O(\log N)$ entries matching increasingly long prefix of local nodeId
  - Soft state
    - Periodically refreshed
Resource announcements

are physically close to
Step 3: Enable dynamic flocking

- Central managers flock with nearby pools
  - Use knowledge gained from resource announcements
  - Implement local policies
  - Support dynamic reconfiguration
Interactions between central managers

Locality-aware flocking
Matchmaking

- Orthogonal to flocking
- Condor matchmaking within a pool
- P2p approach affects the flocking decisions only
Are we discovering enough pools?

- Only subset of nearby pools reached using the Pastry routing table
- Multi-hop TTL based announcement forwarding
Agenda

• Background: peer-to-peer networks
• Proposed scheme
• Implementation
• Evaluation
• Conclusions
Software

- Implemented as a daemon: \textit{poolD}
  - Leverages FreePastry 1.3 from Rice
  - Runs on central managers
  - Manages self-organized Condor pools

- Condor version 6.4.7

- Interfaced to Condor configuration control
Agenda

- Background: peer-to-peer networks
- Proposed scheme
- Implementation
- Evaluation
- Conclusions
Evaluation

• Measured results
  – Effect of flocking on job throughput
    • Time spent in queue
  – Four pools, three compute machines each
  – Synthetic job trace
Job trace

• Sequence
  – 100 (issue time: T, job length: L) pairs
  – Interval \((T_n - T_{n-1})\), L uniform distribution \([1,17]\)
  – Designed to keep a single machine busy
  – Random overload/idle periods

• Trace
  – One or more job sequences merged together
PlanetLab experimental setup

Dynamic flocking

A  Interxion, Germany
B  U.C. Berkeley
C  Columbia
D  Rice
## Time spent in queue

<table>
<thead>
<tr>
<th>Pool</th>
<th>No. of sequences in trace</th>
<th>Without flocking</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>mean</td>
<td>min</td>
</tr>
<tr>
<td>A</td>
<td>2</td>
<td>1.76</td>
<td>0.03</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>3.30</td>
<td>0.08</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>46.58</td>
<td>0.03</td>
</tr>
<tr>
<td>D</td>
<td>5</td>
<td>284.91</td>
<td>0.25</td>
</tr>
<tr>
<td>overall</td>
<td>12</td>
<td>131.20</td>
<td>0.03</td>
</tr>
</tbody>
</table>
Simulations

- 1000 Condor pools
- GT-ITM transit-stub model
  - 50 transit domains
  - 1000 stub domains
- Size of pool: uniform distribution [25,225]
- Number of sequences in trace: uniform distribution [25,225]
Cumulative distribution of locality

![Cumulative distribution of locality graph](image-url)
Total job completion time: without flocking
Total job completion time: with flocking
Agenda

- Background: peer-to-peer networks
- Proposed scheme
- Implementation
- Evaluation
- Conclusions
Conclusions

• Design and implementation of a self-organizing flock of Condors
  – Scalability
  – Fault-tolerance
  – Locality-awareness which yields flocking with nearby resources
  – Local sharing policy enforced

• P2p mechanisms provide an effective substrate for discovery and management of dynamic resources over the wide-area network
Questions?
What about security?

- Authenticated pools / users
  - Enforced by policy manager
  - Accountability

- Restricted access
  - Limited privileges e.g. UNIX user nobody
  - Condor libraries

- Controlled execution environment
  - Sandboxing
  - Process cleanups on job completion

- Intrusion detection