Peer-to-peer resource discovery for self-organizing virtual private clusters

Renato Figueiredo
Associate Professor
Center for Cloud and Autonomic Computing
ACIS Lab
University of Florida
Outlook

- Peer-to-peer overlays in virtual private clusters
  - Self-configuration, self-healing, self-optimization
  - Framework for communication among nodes
    - Virtual machines + virtual networks
    - Isolation, security, encapsulation
  - Also framework for resource discovery

- Brunet/IPOP peer-to-peer overlay design
- Virtual networking and virtual clusters
- Resource discovery
Peer-to-peer virtual networks

Focus:
- Software overlays that provide virtual network infrastructure over existing Internet infrastructure

Why virtual?
- Support unmodified TCP/IP applications and existing Internet physical infrastructure
- Hide heterogeneity of physical network (firewalls, NATs), avoid IPv4 address space constraints

Why self-organizing?
- Autonomous behavior: low management cost compared to typical VPNs
- Decentralized architecture for scalability and fault tolerance
IP-over-P2P (IPOP) overlays

- **Isolation**
  - Virtual address space decoupled from Internet address space

- **Self-organization**
  - Automatic configuration of routes and topologies
    - Decentralized – structured peer-to-peer (P2P)
      - No global state
      - No central points of failure
    - Mobility – keep same virtual IP address

- **Decentralized NAT traversal**
  - No changes to NAT configuration
  - No need for STUN server infrastructure

[IPDPS 2006, Ganguly et al]; http://ipop-project.org
Use case scenarios

- Collaborative environments based on virtual machines and virtual networks
  - VM provides isolation
  - Virtual appliances provide software encapsulation

- WOWs: Virtual appliance clusters
  - Homogeneous software environment on top of heterogeneous infrastructure
  - Homogeneous virtual network on top of wide-area, NATed environments
Based on the Brunet library
Bi-directional ring ordered by 160-bit node IDs
Structured connections:
- “Near”: with neighbours
- “Far”: across the ring

Multi-hop path between n1 and n7

n1 < n2 < n3 < ……… < n13 < n14
Overlay edges and Routing

- **Overlay edges**
  - Multiple transports: UDP, TCP, tunnel
  - UDP: decentralized NAT hole-punching

- **Greedy routing**
  - Send over edge leading closest to destination

- **Constant number of edges**
  - $O((1/k)\log^2(n))$ overlay hops
    - K-connections per node
    - Kleinberg small-world network

- **Adaptive shortcut edges**
  - 1-hop between frequently communicating nodes
  - Autonomous proxy selection if NAT traversal fails
Managing virtual IP spaces

- One P2P overlay, multiple IP namespaces
  - IP assignment, routing occurs within a namespace
- Each IPOP namespace: a unique string
  - Distributed Hash Table (DHT) stores mapping
    - Key=namespace
    - Value=DHCP configuration (IP range, lease, ...)
- IPOP node configured with a namespace
  - Query namespace for DHCP configuration
  - Guess an IP address at random within range
  - Attempt to store in DHT
    - Key=namespace+IP
    - Value=IPOPid (160-bit)
- IP->P2P Address resolution:
  - Given namespace+IP, lookup IPOPid
Autonomic features

- **Self-configuration [IPDPS’06, HPDC’06, PCgrid’07]**
  - Routing tables using structured P2P links
  - NAT traversal, DHCP over DHT

- **Self-optimization [HPDC’06]**
  - Direct shortcut connections created/trimmed based upon IP traffic inspection for fast end-to-end IP tunnels
  - Proximity neighbor selection based on network coordinate estimates for improved structured routing

- **Self-healing [HPDC’08]**
  - “Tunnel” edges created to maintain overlay structure to deal with routing outages and NATs/firewalls that are not traversable

- **VLAN routers, overlay bypass [VTDC09, SC09]**
WOW: Virtual Private Clusters

- Plug-and-play deployment of Condor grids
  - High-throughput computing; LAN and WAN
  - Data sharing

- Wide-area virtual clusters, where:
  - Nodes are virtual machines (VMs: e.g. VMware, Xen)
  - Network is virtual: IPOP

- VMs provide:
  - Sandboxing; software packaging; decoupling

- Virtual network provides:
  - Virtual private LAN over WAN

- Condor provides:
  - Match-making, reliable scheduling, … unmodified
Example: The Archer System

Archer seed resources

Local resource pools:
servers, clusters,
desktop labs

User
desktops

Self-configuring
Virtual appliances

Deployment, support, configuration, troubleshooting

Archer software and management

Voluntary
resources

Community-contributed content: applications, datasets

Web portal,
documentation,
tutorials

Community-contributed content: applications, datasets

Local resource pools: servers, clusters, desktop labs

http://archer-project.org
Example: Archer

PlanetLab overlay: ~450 nodes, 24/7 on a shared infrastructure

Archer cluster ramp-up: UFL, NEU, UMN, UTA, FSU
Resource discovery

- Scale up resources in a virtual cluster, self-configuring middleware, fault-tolerance
- How to leverage the P2P virtual network for flexible, scalable resource discovery?

Traditional approaches

- Centralized approach
  - Multiple central managers (e.g., Condor Flocking)
  - Pull-based job discovery/scheduling (e.g., Boinc)

- Decentralized approach
  - DHT-based
    - MAAN, SWORD, Mercury
  - Multi-dimension based
    - Squid, RD on CAN
Overall Approach

- Decentralized resource discovery framework
  - No single point-of-failure; self-configuring
  - Leveraging self-organizing multicast tree
    - Query distribution/processing ("Map")
      - Resolve matches “in-situ”
      - No periodic resource attribute update needed
    - Result aggregation ("Reduce")

- Rich query processing capacity
  - Regular expression match
  - Appropriate for dynamic attributes
Query propagation

- Self-organizing multicast tree
 Modules

- **Matchmaking**
  - Checking if a requirement matches the resource status
  - Using timely local information
    - Appropriate for dynamic attributes

- **Aggregation**
  - Sort requirement satisfying nodes based on a rank value
  - Hierarchical query results summary at internal nodes

- **Implementation**
  - Condor ClassAds and condor_startd
    - Provides a rich query processing
Query Processing

Requirements = (Memory>1GB)
Rank = Memory, Return top 2 rank value nodes
Dealing with Faults

- Nodes join/leave
- Slow nodes
  - Adaptive timeouts
  - Redundant query topologies
Timeouts on multicast tree

- Dynamic timeout based on completion score
- Node determines timeout based on the latency and fraction of results returned by a child

$WF : \text{User input: } WF > 0$

$l_i : \text{latency until } i-th \text{ result returned } (0 < i < n_c)$

$C_i : \text{Completion score at } i-th \text{ result}$

$\text{Timeout}_i = \frac{l_i \times WF}{C_i}$
Timeouts on multicast tree

- Dynamic timeout based on completion score

Node A: Query begin

Node B returns a result

Node D returns a result

Node G returns a result

Node N returns a result

\[ TO_1 = \frac{WF}{0.1} \left( T_1 - T_0 \right) \]

\[ TO_2 = \frac{WF}{0.3} \left( T_2 - T_0 \right) \]

\[ TO_3 = \frac{WF}{0.4} \left( T_3 - T_0 \right) \]

\[ TO_4 = \frac{WF}{0.8} \left( T_4 - T_0 \right) \]
Timeouts on multicast tree

- Guarantee the bounded response time
- Reflects user’s input
  - WaitFactor
- More results returned, less waiting time
  - Completion score
- Dynamically adjusts to network status
Redundant topology

- Multicast tree generation
  \( P_j \): An arbitrary parent node
  \( C_{i-1}, C_i, C_{i+1} \): Child nodes of \( P_j \), \( C_{i-1} < C_i < C_{i+1} \)

- Clockwise tree generation

  \( C_i \) is responsible for \([C_i, C_{i+1})\)

- Counter-clockwise tree generation

  \( C_i \) is responsible for \((C_{i-1}, C_i]\)
Redundant topology

<table>
<thead>
<tr>
<th>ID</th>
<th>mode</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>clk</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>c-clk</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>clk</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>c-clk</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>clk</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>c-clk</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>clk</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>c-clk</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>clk</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>c-clk</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Redundant topology + timeouts

- Experiments on PlanetLab

A. Query response time

B. Query coverage
Ongoing/future work

- Self-organizing virtual clusters
  - Discover nodes that match job class-ads
  - Self-organize virtual private namespace, self-configure scheduling middleware on demand
    - Condor
    - Hadoop

- Techniques to bound scope of queries for improved latency/reduced bandwidth
  - First-fit
  - Sub-region queries
  - Machine learning
Thank you!

- For more information and downloads
  - http://ipop-project.org
  - http://grid-appliance.org
  - http://archer-project.org
  - http://futuregrid.org

- Acknowledgments
  - ACIS P2P group
    - Resource discovery slides - Kyungyong Lee
  - National Science Foundation
    - CAC center
    - Archer, FutureGrid and DiRT