Structured and Unstructured Overlays for Distributed Data

Arvind Krishnamurthy
Fall 2004

Evolution of Overlays in P2P Systems

Ad-hoc Overlays (Freenet, Gnutella)

Structured Overlays (Chord, Pastry, CAN)

Unstructured Overlays (Narada, NICE)

Implicit routing
Probabilistic load-balance
Small routing state
Highly scalable systems

Explicit routing protocol
Medium-scale systems

Complex Queries
Distribute with locality

Performance-sensitivity
Dynamic changes in quality

?
**System Requirements for Structured Overlays**

- System should be able to store complex objects
- Be capable of supporting similarity searches, range queries
- Should minimize:
  - Storage overhead
  - Routing state
  - Routing costs
- Should balance:
  - Data distribution
  - Routing state
  - Query/routing costs

**SkipIndex: Overview**

- Use some hierarchical tree data structure which is used in centralized settings to manage high-dimensional data
- Distribute tree while retaining locality
  - Nearby tree-nodes maintained by nearby processors
  - Processors manage sub-trees
- Do not navigate the tree in a top-down manner
  - Instead jump from one part of the tree to another part of the tree using “long” pointers
- Employ a load-balancing algorithm to ensure that data distribution is balanced
Motivating Reasons for Unstructured Overlays

- **Overlay networks:**
  - Construct a virtual network on top of the physical network
  - Can communicate from lambda.cs.yale.edu to comix.cs.berkeley.edu:
    - Either directly through the internet path
    - Or through an intermediate node at bolle.cs.princeton.edu
      - In which case, it enters Yale’s ISP, traverses Internet, reaches Princeton, is retransmitted through Princeton’s ISP, traverses Internet and reaches Berkeley

- **Three primary reasons:**
  - Inefficient Internet routes
  - Avoid failures in Internet routes
  - Provide functionality not currently available in the Internet

Inefficient Internet Routing

- Internet routing is inefficient:
  - Does not always pick the lowest latency paths
  - Does not always pick paths with low drop rates
- Experimental evidence with 43 nodes: (Detour project at Washington)
  - 50% of routes had a faster alternate one-hop route
  - 80% of routes had an alternate route with lower loss rate
Reasons for path inflation #1

- Using AS-hop-count as routing metric:
  - Using actual latency or router-hop-count would be better

![Diagram showing shortest AS path and shortest router path]

Reasons for path inflation #2

- Policy routing might result in picking a longer path
- No-valley routing policy:
  - An AS does not provide transit between any two of its providers or peers.

![Diagram showing preferred path]

- Prefer Customer routing policy:
  - Prefer the free of charge customer route over the peer or provider route.
- Early exit strategy:
  - AS might try to get rid of a packet as soon as possible and minimize its intra-domain traffic
Avoid Internet Path Outages

- When a path fails try to find an alternate path if possible
- Internet:
  - Path outages are reasonably common
  - Recovery time could be substantial

<table>
<thead>
<tr>
<th>Source</th>
<th>Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paxson 95-97</td>
<td>3.3% of all routes had serious problems</td>
</tr>
<tr>
<td>Labovitz 97-00</td>
<td>10% of routes available &lt; 95% of the time</td>
</tr>
<tr>
<td></td>
<td>65% of routes available &lt; 99.9% of the time</td>
</tr>
<tr>
<td></td>
<td>3-min minimum detection+recovery time; often 15 mins</td>
</tr>
<tr>
<td></td>
<td>40% of outages took 30+ mins to repair</td>
</tr>
<tr>
<td>Chandra 01</td>
<td>5% of faults last more than 2.75 hours</td>
</tr>
</tbody>
</table>

Advanced Routing Mechanisms

- Internet routers rarely support advanced protocols such as IP multicast
- Ideally, routers should have intelligence to form multicast trees, maintain membership information, and split flows
- More advanced applications that could benefit from network-embedded intelligence include wide-area file systems
Solution: Overlay Networks

- Route around faults
  - Use an “overlay path” that comprises of two or more physical connections through the internet
- Route around inefficiencies
  - Intelligence (for multicasting, wide-area file-systems, etc.) is pushed to the edge of the internet

Overlay Multicast

- Multicast performed through multiple unicasts
- Edge-hosts serve as forwarding agents (results in a distribution tree)
Optimizing Unstructured Overlays

- Explicit routing mechanisms are expensive
  - Link-state, distance-vector protocols scale as $O(n^3)$
  - Routing protocols exchange routing tables:
    - Routing table size is $O(n)$ in fully connected mesh
    - Tables exchanged between every pair of neighbors in a fully connected mesh
  - Designers of Resilient Overlay Networks (RON) do not expect such systems to scale beyond 50-100 nodes

Unstructured Overlay Networks

- Overlay networks tend to be complete graphs
- To reduce complexity of communication and routing, you want a pruned graph
**Problem Statement**

- High performance
  - Best links with respect to latency, bandwidth, and/or loss-rate
- Multiple paths
  - Multipath transport
  - Fault tolerance
  - Performance tolerance
- Self organizing
  - Exploit network information
  - Incremental improvement
  - Minimal use of central authority

What should a pruned graph look like?

**General Strategies**

- K best links
  - Each node find “k” best peers
- K-random links
  - Each node find “k” random peers
- Short-long
  - K/2 best links and K/2 random links
- Connect-improve
  - Select random and carefully improve (for example, Narada)
Find K-Minimum Spanning Trees

Algorithm:
for $j := (1 \ldots k)$ do
    $F_j = MST(G_{j-1})$
    $F^j = F^{j-1} \cup F_j$
    $G_j = G_{j-1} - F_j$
end for
Find K-Minimum Spanning Trees

Algorithm:
\[ \text{for } j := (1 \ldots k) \text{ do} \]
\[ F_j = MST(G_{j-1}) \]
\[ F^j = F^{j-1} \cup F_j \]
\[ G_j = G_{j-1} - F_j \]
\[ \text{end for} \]
**Find K-Minimum Spanning Trees**

**Algorithm:**

```
for j := (1 \ldots k) do
    F_j = MST(G_{j-1})
    F^j = F^{j-1} \cup F_j
    G_j = G_{j-1} - F_j
end for
```

---

**Motivation for K-MST Approach**

- **Theory:**
  - Algorithm by Khuller and Vishkin gives the best approximation for finding a minimum-weight k-connected subgraph
  - Approximation finds K directed trees of minimum cumulative weight
  - Related work by Roskind and Tarjan finds K disjoint trees of minimum weight; might use more graph edges than previous work
  - K-MST is an approximation to the Roskind-Tarjan work

- **Systems:**
  - Many systems use MST for multicasting data-intensive streams
  - Some systems augment a single multicast tree with additional links to find “shortcut” edges
Protocol Components

Mesh Construction:
Based on algorithm by Gallagher Humblet, and Spira (GHS), which computes a Minimum Spanning Tree in a fully distributed fashion.
We use K instances of GHS to compute K trees.

Mesh Repair:
Maintain Minimum Spanning Tree in a dynamic topology with failures.

Mesh Improvement
Improve sub-optimal trees in a dynamic environment that preserves properties of the tree:

MST Algorithm (GHS)

To Build Single MST in Distributed Environment, use GHS:
Algorithm proceeds in levels, where at start each process comprises a subcomponent of level 0.
To Build Single MST in Distributed Environment, use GHS:

- Algorithm proceeds in levels, where at start each process comprises a subcomponent of level 0.

- Each process finds its best or Minimum Weight Outgoing Edge (MWOE) and convergecasts this information to the leader of the subcomponent.

- Leader selects the lowest weight edge, which is the MWOE for the component. When both components agree on MWOE, a new edge is selected.
MST Algorithm (GHS)

To Build Single MST in Distributed Environment, use GHS:

- Algorithm proceeds in levels, where at start each process comprises a subcomponent of level 0.
- Each process finds its Minimum Weight Outgoing Edge (MWOE) and convergecasts this information to the leader of the subcomponent.
- Leader selects the lowest weight edge, which is the MWOE for the component. When both components agree on MWOE, a new edge is selected.
- Interior edges are Rejected to prevent cycles.
MST Algorithm (GHS)

To Build Single MST in Distributed Environment, use GHS:

- Algorithm proceeds in levels, where at start each process comprises a subcomponent of level 0.

- Each process finds its Minimum Weight Outgoing Edge (MWOE) and convergecasts this information to the leader of the subcomponent.

- Leader selects the lowest weight edge, which is the MWOE for the component. When both components agree on MWOE, a new edge is selected.

- Interior edges are Rejected to prevent cycles.

Done!
K-MST Construction

- k Instances of the GHS algorithm
  MST1, MST2, … MSTk

- Each Process Sorts Edges in non-decreasing order, and tries to add it to MST1

Diagram: A set of points connected with lines, representing the MST construction process with different MSTs.
K-MST Construction

- k Instances of the GHS algorithm
  MST1, MST2, … MSTk
- Each Process Sorts Edges in non-decreasing order, and tries to add it to MST1
K-MST Construction

- k Instances of the GHS algorithm
  MST1, MST2, … MSTk

- Each Process Sorts Edges in non-decreasing order, and tries to add it to MST1

- If the edge is already in MST1, it tries to add it to MST2,
K-MST Construction

- k Instances of the GHS algorithm
  MST1, MST2, ... MSTk
- Each Process Sorts Edges in non-decreasing order, and tries to add it to MST1
- If the edge is already in MST1, it tries to add it to MST2,
K-MST Construction

- k Instances of the GHS algorithm
  MST1, MST2, … MSTk
- Each Process Sorts Edges in non-decreasing order, and tries to add it to MST1
- If the edge is already in MST1, it tries to add it to MST2, and so on.
- Parallelism at different levels
Incremental Improvement

- Based on tree property, that adding one link creates a fundamental cycle.
- Processes have complete or partial routing information, which allows it to reconstruct portions of the tree.
- Over time, link weights may change.
- Nodes can minimize global weight if replacing an existing link with a new link decreases the weight of a cycle.

Incremental Improvement

- Problem: nodes may concurrently try to improve along the same cycle

- Cycles must be avoided, requires a two-phase protocol for locking the edges in the cycle in a distributed fashion before performing the updates