SCALLOP
A Scalable and Load-Balanced Peer-to-Peer Lookup Protocol for High-Performance Distributed System

Jerry Chou, Tai-Yi Huang & Kuang-Li Huang
Embedded & Operating System (EOS) Lab
http://eos.cs.nthu.edu.tw/
Computer Science Department
National Tsing Hua University, Taiwan
Outline

- Contributions
- Methodology
- Experimental Results
- Conclusions
- Future work
Motivations and Purposes (1/2)

- Many large-scaled servers are implemented in a peer-to-peer distributed system due to:
  - Low cost of workstations
  - Availability of high-speed network

- Two main concerns of the system
  - Performance ➔ response time
  - Stability ➔ hot spot problem
Motivations and Purposes (2/2)

- **Response time** = *Lookup time + Service time*
  - Shorter lookup forwarding path $\Rightarrow$ smaller lookup time
  - Balanced-load on nodes $\Rightarrow$ reduce service time

- **Hot spot problems**
  - An overloaded node becomes disable or malfunction
  - The network traffic around hot spots increases
    $\Rightarrow$ Resulting in an unstable system
    $\Rightarrow$ Decreasing system performance
Related Work

- **CAN**: A scalable content addressable network. In *ACM SIGCOMM 2001*
- **Pastry**: Scalable, decentralized object location, and routing for large-scale P2P system. In *Lecture Notes in Computer Science 2001*
- **Chord**: A scalable P2P lookup service for Internet applications. *In ACM SIGCOMM 2001*

→ All of them do not address the hot spot problem
Our Contributions (Stability)

- Reduce or eliminate hot spots
  - Balanced-load
    - Both data items and lookup requests are evenly distributed
    - Service time is also reduced
  - The number of children nodes is limited
    - Each node is at most accessed by other $d$ nodes

- Decentralized
  - Without the bottleneck on centralized servers
Our Contributions (Performance)

- Scale with the number of nodes
  - Each node is only aware of other $O(d \log_d N)$ nodes
- Provide an upper bound on lookup paths
  - The lookup path for any request is $O(\log_d N)$
- Allow a tradeoff between space and time
  - If $d$ becomes larger
    - More routing information required
    - Shorter lookup path
Methodology

- Data partition
- Structure of the balanced lookup tree
- Construction of the lookup table
- Self-organized mechanism
Data Partition

- Use a variant of consistent hashing to partition the data set among all nodes
  - key $k$ is stored in node $n$ where $|k-n|$ is minimum
- Two proven properties of consistent hashing
  - Each node stores similar amount of data items
  - Data movement is minimum when system changes
- Our protocol assigns each node a number, called SID, between 0 to $N-1$
Balanced Lookup Tree

- We construct a balanced lookup tree for each node
- The root of a lookup tree of a node is the node itself

- Lookup path is bounded by $O(\log_d N)$
Construction of Lookup Table

- Lookup table is a collection of **lookup pairs**
- Get lookup pairs \((\text{target}, \text{forwarder})\) from lookup trees
  - **forwarder** is the next node in the lookup path to the **target** node
- Group **lookup pairs** to reduce the number of entries in a lookup table
(1) There is a lookup pair (7, 4) for node 0
(2) There is a lookup pair (1, 1) for node 0
Node 0 Lookup Table

- Take the lookup table of node 0 as an example
- Collect and group lookup pairs for node 0
- Reduce the entries from 16 to 7

<table>
<thead>
<tr>
<th>Target</th>
<th>Forwarder</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Target</th>
<th>Forwarder</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>11</td>
<td>8</td>
</tr>
<tr>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td>13</td>
<td>12</td>
</tr>
<tr>
<td>14</td>
<td>12</td>
</tr>
<tr>
<td>15</td>
<td>12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Entry</th>
<th>Target range</th>
<th>Forwarder</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(15,0]</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>(0,1]</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>(1,2]</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>(2,3]</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>(3,8]</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>(8,12]</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>(12,15]</td>
<td>12</td>
</tr>
</tbody>
</table>
Generic Lookup Table (1/3)

- All lookup tables can be constructed by the generic lookup table
  - \textit{without} examining any lookup tree

- Distance tree
  - Used to show the lookup pairs in distance relationship
  - Constructed by replacing each node in balanced lookup tree with a pair \((D_{2T}, D_{2F})\)
    - \(D_{2T}\): the distance to target node (root – node)
    - \(D_{2F}\): the distance to forwarder node (parent – node)
Any pair of (D2T, D2F) is independent of $k$

⇒ there is only one unique distance tree

⇒ there are $N$ different lookup trees
Generic Lookup Tree (3/3)

- Group D2T and D2F to get generic lookup table
- A node adds its SID to generic lookup table to establish its own lookup table

<table>
<thead>
<tr>
<th>level</th>
<th>D2T range</th>
<th>D2F value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[0, 0]</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>[i, 2i]</td>
<td>i</td>
</tr>
<tr>
<td>2</td>
<td>[id + 1, id + d]</td>
<td>id</td>
</tr>
<tr>
<td>3</td>
<td>[id^2 + d + 1, id^2 + d + d^2]</td>
<td>id^2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>l</td>
<td>[Σ_{x=0}^{l-1}(d^x) + (i - 1)d^{l-1}, Σ_{x=0}^{l-1}(d^x) + id^{l-1} - 1]</td>
<td>id^{l-1}</td>
</tr>
</tbody>
</table>

- i = 1, 2, 3, ..., d
- total entries = l * i = O(log_d N) * d = O(d log_d N)
Self-organized Mechanism

- The lookup tables need to be updated each time a node joins, leaves or fails
- All lookup tables are aware of the system change in $O(\log N)$ hops
  - The updating message is forwarded along a balanced lookup tree
- A new lookup table only relies on the old lookup table of its own & its adjacent nodes
  - The lookup tables of adjacent nodes are offset by 1
  - SID to VID mapping is resolved locally
Experimental Results

- Compared SCALLOP with a binomial-tree lookup protocol (e.g., Chord, Pastry, etc)
  - Hot spot avoidance
  - Balanced-load distribution
  - Customized performance
  - Self-organized mechanism
Structure Comparison

- Our lookup tree distributes lookup requests evenly
- Eliminate the hot spot problem on node 15
Balanced-Load Distribution

- Environment
  - 10,000 nodes
  - $10^6$ lookup requests
  - Hot spot scenario: 30% of the requests ask for a data item stored in node 0
Balanced-Load Distribution (Cont.)

Results

- Our protocol reduces about 50% load on node 9999 (a hot-spot node in binomial lookup protocol)
- Flatter distribution with or without hot spots
Customized Performance

- SCALLOP allows customizable performance by changing its degree of the lookup tree
Self-organized Mechanism

- Environment
  - Assume each hop takes one time unit
  - 1,000 nodes
  - 1,000 lookup requests per time unit
  - Nodes fail scenario: 5% nodes fail per 10 time unit from time 50 to 100 in a 200 time unit simulation
Self-organized Mechanism

- Results
  - Average lookup path length grows as fault rate increases
  - No impact on path length once fault rate is below 20%
  - System is stabilized quickly and eventually
Conclusions & Future Work

- We develop a lookup protocol for peer-to-peer distributed system
  - **Scalable**: $O(d \log_d N)$ lookup table
  - **High-performance**: $O(\log_d N)$ lookup path
  - **Balanced-load**: even data and lookup distribution

- Future work
  - More experimental results
  - Implementation on PlanetLab distributed system.
Question?