PASTRY: Scalable, decentralized object location and routing for large-scale peer-to-peer systems

Antony Rowstron and Peter Druschel, 2001
Outline

- Introduction
- Design of Pastry
  - Node state & routing
  - Pastry API
  - Self-organization and adaptation
  - Locality
- Experimental Results
Pastry

- It’s a scalable, distributed, decentralized object location and routing substrate

- Serves as a general substrate for building P2P applications: SCRIBE, PAST,...etc.

- Seeks to minimize distance messages travel
Pastry Node

- Represented by 128-bit randomly chosen nodeId (Hash of IP or public key)

- NodeId is in base $2^b$ ($b$ is a configuration parameter; $b$ typical value 2 or 4)

- Evenly distributed nodeIds along the circular namespace ($0-2^{128} - 1$ space).

- Routes a message in $O(\log N)$ steps to destination
  - $N$: size of network
Design of Pastry: Node state

- A leaf set
  - $|L|$ nodes with closest nodeIDs
    - $|L|/2$ larger ones and $|L|/2$ smaller ones
  - Useful in message routing

- Routing Table (Prefix-based)

- A neighborhood set
  - $|M|$ nearest neighbors
  - Useful in maintaining locality properties
Leaf Set and Neighborhood Set

In this example

\[ b = 2, \ l = 8 \quad l: \text{row} \]

\[ |L| = 2 \times 2^b = 8 \]

\[ |M| = 2 \times 2^b = 8 \]
Routing Table

- $l$ rows and $2^b$ columns
  - $i^{th}$ row: $i$-prefix
  - $j^{th}$ column: next digit after the prefix is $j$
- $b=2$, $l=8$
- $-\log_2 b \cdot N$ rows
- $2^b - 1$ entries in each row
  - 8 rows and 4 columns

matched digits-column number–rest of ID

<table>
<thead>
<tr>
<th>NodeID 10233102</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leaf set</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>10233033  10233021</td>
</tr>
<tr>
<td>10233001  10233000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Routing table</th>
</tr>
</thead>
<tbody>
<tr>
<td>$j=0$</td>
</tr>
<tr>
<td>$j=1$</td>
</tr>
<tr>
<td>$j=3$</td>
</tr>
<tr>
<td>0 1 2 3</td>
</tr>
<tr>
<td>102-0-0230 102-1-301233 102-2-230203 102-3-021022</td>
</tr>
<tr>
<td>10-0-31203 10-1-32102 10-2-230203 10-3-23302</td>
</tr>
<tr>
<td>1023-0-322 1023-1-000 1023-2-120 1023-3-22</td>
</tr>
<tr>
<td>10233-0-01 10233-1-000 10233-2-32 10233-3-00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Neighborhood set</th>
</tr>
</thead>
<tbody>
<tr>
<td>13021022  1020230  11301233  31301233</td>
</tr>
<tr>
<td>02212102  22301203  31203203  33213321</td>
</tr>
</tbody>
</table>
Routing Example:

Route(d46a1c)
Routing

- Step1: If $k$ falls within the range of nodeIDs covered by $A$’s leaf set, forwarded it to a node in the leaf set whose nodeID is closest to $k$.

- Eg. $k = 10233022$ falls in the range $(10233000, 10233232)$ Forword it to node10233021

- If $k$ is not covered by the leaf set, go to step2

<table>
<thead>
<tr>
<th>NodeID 10233102</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Leaf set</strong></td>
</tr>
<tr>
<td><strong>SMALLER</strong></td>
</tr>
<tr>
<td>10233033</td>
</tr>
<tr>
<td>10233001</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Routing table</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>10-0-31203</td>
</tr>
<tr>
<td>102-0-0230</td>
</tr>
<tr>
<td>1023-0-322</td>
</tr>
<tr>
<td>10233-0-01</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Neighborhood set</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>13021022</td>
</tr>
<tr>
<td>02212102</td>
</tr>
<tr>
<td>11301233</td>
</tr>
<tr>
<td>31301233</td>
</tr>
</tbody>
</table>

| 0233203120          | 10233230         |
| 1023323230          | 10233232         |
Routing

- Step 2: The routing table is used and the message is forwarded to a node whose ID shares a longer prefix with the $k$ than A's nodeID does.

- Eg. $k = \text{10223320}$ forward it to node 10222302

- If the appropriate entry in the routing table is empty, go to step 3.
Step 3: The message is forwarded to a node in the leaf set, whose ID has the same shared prefix as $A$ but is numerically closer to $k$ than $A$

- Eg. $k = 10233320$
- Forward it to node $10233232$
- If such a node does not exist, $A$ is the destination node
Routing

- The routing procedure always converges, since each step chooses a node that
  - Shares a longer prefix
  - Shares the same long prefix, but is numerically closer

- Routing performance
  - The expected number of routing steps is $\log_2 bN$
  - Assumption: accurate routing tables and no recent node failures
Pastry API:

- `nodeId = pastryInit(Credentials, Application)`
- `route(msg, key)`
- `deliver(msg, key)`
- `forward(msg, key, nextId)`
- `newLeafs(leafSet):`
Self-organization: Node Arrival

- Arriving Node X knows “nearby” node A
- X asks A to route a “join” message with key = NodeId(X)
- Message targets Z, whose NodeId is numerically closest to NodeId(X)
- All nodes along the path A, B, ..., Z send state tables to X
- X initializes its state using this information
- X sends its state to “concerned” nodes
State Initialization

- X’s neighborhood set (NS) = A’s NS
- X’s Leaf Set = Z’s leaf set
- X’s routing table is filled as follows:
  - X’s Row 0 = A’s row 0 ($X_0 = A_0$)
  - X’s Row 1 = B’s row 1 ($X_1 = B_1$)
  - ...etc.
- X sends its state to every node in its state tables (Leaf set, neighborhood set, and routing table)
Pastry node join

ID Space: [0, 2^{128} - 1]
Node departure

- **Leaf-Set**
  - Ask for the rest of the nodes in the leaf-set for their leaf-set nodes.

- **Routing Table**
  - Lazy – When a node doesn’t respond
  - Complete the table slot from a node in the same row, if not found - try from lower rows

- **Neighboring-Set**
  - Permanently monitored and maintained
Node failure in routing table: example

If node in red fails

Try asking some neighbor in the same row for its 655x entry

If it doesn’t have one, try asking some neighbor in the row below, etc.
Locality

- Based on proximity metric (i.e., No. of IP hops, geographic distance)
- Proximity space is assumed to be Euclidean
- The route chosen for a message is likely to be “good” with respect to the proximity metric
- We will discuss locality regarding:
  - Routing table locality
  - Route locality
  - Locating the nearest among $k$ nodes
Locality in Routing tables

- Invariant: “all routing table entries refer to a node that is near the present node, according to the proximity metric, among all live nodes with a prefix appropriate for the entry.”

- We wish to maintain the invariant when adding new nodes.

- \( X \) joins; \( A \) is close to \( X \); \( X_0 = A_0 \), so locality holds in \( X \)’s routing table

- \( X_1 = B_1 \). Entries in \( B_1 \) (row 1 of \( X \)) are close to \( B \), but are they necessarily close to \( X \)?
Locality in routing table

- Entries of $B_i$ are reasonable close to $X$ Why?
  - $A$ is much closer to $B$ than entry in $B_i$ to $B$ because every time we choose from an exponentially decreasing set of nodes.

- To improve proximity approximation:
  - $X$ Queries nodes in routing table and neighborhood set for their state.
  - Compares distances (from routing table entries) and update route entries with closer nodes if found.
Route locality

- For larger row numbers the number of possible choices decreases exponentially.
- For larger row numbers the expected distance to the nearest neighbor increases exponentially.
- Though shortest path is not guaranteed, we still get a good route.
Locality among $k$ nodes

- In some Pastry-based applications, object is replicated on $k$ nodes on its route (during insertion)
- In prefix-base routing: goal is to reach any of $k$ numerically closest nodes that has a copy of object
- May miss nearby nodes with different prefix
- Use heuristic to determine when close to $k$ nearest nodes
  - Based on density of nodeIds that store object; using local info
  - Switch to numerically closest address
Arbitrary node failure

- Node continues to be responsive, but behaves incorrectly or maliciously.

- Repeated queries fail each time because they normally take the same route.

- Use randomized routing
  - The choice among multiple nodes that satisfy the routing criteria should be made randomly
Routing Performance

\[ |L| = 16 \times b = 4 \times |M| = 32 \times 200,000 \text{ lookups} \]
Probability versus number of routing hops, $b = 4$, $|L| = 16$, $|M| = 32$, $N = 100,000$ and 200,000 lookups.

$$\left\lceil \log_2 N \right\rceil \cdot \left\lceil \log_2 100,000 \right\rceil = 5$$ as expected.
Routing with failures

Quality of routing tables before and after 500 node failures, $b = 4$, $|L| = 16$, $|M| = 32$ and 5,000 starting nodes.
Routing with failures (2)

**Fig. 10.** Number of routing hops versus node failures, $b = 4$, $|L| = 16$, $|M| = 32$, 200,000 lookups and 5,000 nodes with 500 failing.
Pastry locality

\[ |L|=16 \times b=4 \times |M|=32 \times 200,000 \text{ lookups} \]

Pastry routes are only approximately 30% to 40% longer.
Summary

- Pastry is a generic P2P object location and routing substrate
- Distributed, and scales well
- Used in developing applications like file storage, global file sharing,...etc.
- Considers locality when routing messages