Peer-to-Peer Systems
Nicole Hatt
Seminar of Distributed Computing
WS 03/04

Papers
• Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems
  Antony Rowstron and Peter Druschel
• Viceroy: A Scalable and Dynamic Emulation of the Butterfly
  Dahlia Malki, Moni Naor and David Ratajczak

Short reminder: Chord
• A distributed lookup protocol
• Key associated with each data item using consistent hashing
• Maps keys onto nodes
• Each of the N nodes needs routing information of O(log N) nodes
• Joins/Leaves cost O(log^2 N) messages
• Each node maintains a finger table

Example:
net with nodes 0,1,3 and keys 1, 2, 6

Pastry

Design: nodeID and key
• Unique 128 bit nodeID for each node assigned at random:
  - e.g. hash of node’s IP address
  - indicates the position in a circular nodeID space from 0 to 2^128 – 1
  - nodes with adjacent nodeID’s are not physically close to each other
• Every message is assigned a key
• nodeIDs and keys are sequences of digits with base 2^b
• Pastry routes a message with key k to the node having the nodeID numerically closest to k

Design: Proximity metric
• When routing messages, Pastry minimizes the distance the messages travel
• Each node has the possibility to determine its distance to any other node
• Distance is measured according to a scalar proximity metric:
  – Geographic distance
Design: Routing (1)

- Messages are routed to node with numerically closest nodeId to the given key

- In each routing step:
  - the message is routed to a node whose nodeId shares one digit more with the key than the nodeId of the present node
  - or to a node whose nodeId shares a prefix with the key as long as the current one but is numerically closer

  Repeat until numerically closest node is found

- Each node maintains a routing state to support the routing procedure

Design: Pastry node state

- Routing table
  - \( \log_2 N \) rows with \( 2^{n-1} \) entries
  - nodeIds of row n’s entries share first n digits
  - Routing table entry is empty if no suitable node is known

- Neighborhood set
  Contains nodeIds of the closest nodes according to the proximity metric

- Leaf set
  A set of nodes with \( \frac{1}{2} \) numerically closest larger and smaller nodeIds

Design: Routing (2)

- Message with key D arrives at node with nodeId A

- Leaf set
  - Check whether D is in the range of the leaf set
  - In this case, directly forward message to corresponding node

- Routing table
  - if entry is empty or not reachable, forward message to a node with nodeId that shares a prefix with the key as long as the present one and is numerically closer
  - Repeat this step until searched node is found

- Routing procedure always converges

- Routing Performance
  - The expected number of routing steps is \( \log_2 N \)

Node Join

Phase 1:
- A routes a join message with key equal to X
- X receives state tables from A to Z
- X initializes its state tables with:
  - A’s neighborhood set
  - Z’s leaf set
  - Row 0 = row 0 of A
  - Row 1 = row 1 of B
  - Row 2 = row 2 of C

- X send a copy of its state to all nodes

- Total cost: \( 3 \log_2 N \)

N = number of nodes

Phase 2:
- Proximity metric: Each node is able to determine the physical distance to any other node
- Improvement of X’s routing table quality: X
- requests state from each node in the routing and the neighborhood set
- compares physical distances of the nodes in those tables
- updates its state when closer nodes are found
- informs other nodes about its state

Node departure

- Failed/departed node:
  - Immediate neighbors can no longer communicate with it

- Node replacement:
  - To replace a node in its leaf set, a node n asks the next alive node m with largest index for its leaf table
  - M’s leaf set partly overlaps with n’s leaf set
  - A non common node among these leaf sets is selected to be the failed node’s replacement
  - It is important to keep the neighborhood set up to date because it is important for testing if nearby nodes are still alive
Node failures: experimental results
- 5000 node Pastry network
- Quality of the network before and after 500 node failures (b=4)

Applications: PAST
- A storage utility on top of Pastry
- PAST replicates a file on its k numerically closest nodes
- PAST profits from the proximity metric:
  - When routing a message from a client to the numerically closest node, the message first reaches a physically close node among the numerically closest nodes
  - Minimize network load and client latency

Applications: SCRIBE
- Publish/subscribe system
- A node (rendez-vous point) with a nodeId numerically closest to a topicId of a given topic stores a list of subscribers
- Subscribers send messages using the topicId as key
- Each node along the path registers the message
- Publishers send data to the rendez-vous point using topicId as key
- Rendez-vous point forwards the data to all subscribers

Viceroy

Viceroy: properties
- Completely distributed and scalable lookup service
- Key-value pairs are distributed across a changing set of servers
- Keys and servers have identifiers chosen in the same metric
- A key-value pair is on the server with the closest identifier to the key
- Viceroy is a combination of a ring and a butterfly network
- Each server in the network is entirely determined by:
  - Its identifier
  - Its level

The Viceroy network
The network consists of three sets of links:
- a general ring each node k is connected to
  - Its successor (k.successor)
  - Its predecessor (k.predecessor)
- level rings all nodes of the same level are connected in a ring with these links:
  - k.nextonlevel
  - k.prevonlevel
- butterfly each node k points to two down nodes and one up level node:
  - k.right
  - k.left
  - k.up

0 0.5 1
Level 1
Level 2
System model: General ring

- Distribution of key-value pairs among servers:
  - each server is referred by an unique identifier
  - keys and server ID’s are treated as dots in the same metric
  - keys and servers are mapped to the unit ring [0..1)
  - a key-value pair is on the server with the closest ID to the key
  - a server manages key-value pairs with keys between its counter-clockwise neighbor’s ID and its own ID

System model: Level ring

- Goal:
  - select levels that require as few level changes as possible when joins and leaves occur
  - Select levels so that a good dispersal of levels among servers is achieved

Distributed SELECT-LEVEL algorithm:
1. A server s estimates \( n_0 \), the total number of servers in the configuration
   Let \( n_0 = \frac{1}{\text{distance}(s, \text{SUCC}(s))} \)
2. Based on this estimate \( n_0 \), select a level \( l \) between \( \left\lfloor \log_{1/n_0} \right\rfloor \) uniformly at random and return \( l \)

System model: butterfly

- Each server \( s \) at level \( l \) points to:
  - A right down link to level \( l+1 \) clockwise closest node to \( s + \frac{1}{2l} \) \( \text{NLEVEL}_{l+1}(s + \frac{1}{2l}) \)
  - A left down link to level \( l+1 \) clockwise-closest node to \( s \) \( \text{NLEVEL}_{l+1}(s) \)
  - An up link (if \( l>1 \)) to level \( l-1 \) clockwise closest node to \( s \) \( \text{NLEVEL}_{l-1}(s) \)

Simple \text{LOOKUP} subroutine

- Only global ring and butterfly links are used
- \text{LOOKUP}(x,y) finds the clockwise closest to the value \( x \) starting at server \( y \)
- It consists of 3 routing phases:
  - Proceed to root
    find root server by following level up links
  - Traverse tree
    lookup down from the root, if down link does not exist, go directly to traverse ring
    if distance \( d(\text{current}, x) < \frac{1}{2^{\text{current..level}}} \) then
    current = current.left
    else current = current.right
  - Traverse ring
    select closer server to \( x \) between current.successor and current.predecessor
    repeat until closest server is found and return result

Viceroy construction: Join

A joining server \( s \) performs the steps:
1. Select an identifier
2. Find its successor using the \text{LOOKUP} function, insert \( s \) in the ring and update the pointers
3. Transfer all key-value pairs from successor with key between \( s\text{.predecessor} \) and \( s \)
4. Select a level \( l \), update level ring pointers
5. Find \( s\text{.left} \), \( s\text{.right} \) and \( s\text{.up} \)

Viceroy construction: Leave

- Outbound connections have to be removed
- Inbound connections must find a replacement
  - Using the \text{LOOKUP} subroutine
  - By pointing to the successor
- Transfer resources to its successor
Improved **LOOKUP** subroutine

- **Problem of the simple LOOKUP:**
  In the third phase the current and target node might be at a distance of \(O(\log^n)\) when the ring is traversed.

- **Improvement to achieve an \(O(\log n)\) dilation:**
  Third phase is a combination of level and global rings:
  
  ```
  if current.nextonlevel ≤ stretch(current, x)
  then current = current.nextonlevel
  elsif current.prevonlevel ≤ stretch(current, x)
  then current = current.prevonlevel
  else current = current.successor or predecessor
  repeat until clockwise closest node to x found
  ```

  **Stretch** \((x, y)\) = clockwise region between server \(x\) and \(y\)

---

**Simple Viceroy Analysis**

If \(n\) servers are present:

- The first two phases take \(O(\log n)\) steps
- The last phase takes \(O(\log^n)\) steps in expectation and \(O(\log^2 n)\) at worst w.h.p. with the simple lookup and \(O(\log n)\) with the improved
- For any server the expected load is \(O((\log n)/n)\) and w.h.p. the maximum load on all servers is \(O((\log^2 n)/n)\)
- The outdegree of each node is 7 (in simple version only 5), the expected indegree is \(O(1)\) and the largest indegree is \(O(\log n)\) w.h.p.

---

**The Bucket solution**

- Largest indegree in the number can be as large as the log of the number of servers
- „Buckets“ are added:
  - Sets of \(O(\log n)\) servers
    - In case of a size drop, two buckets are merged
    - If the size exceeds \(\log n\), the bucket is split in two
  - One set does not overlap with any other set
  - Inside a bucket a ring is maintained
  - In one bucket is at least one server of each level and no more than \(c\)

---

**Comparison: Pastry/Viceroy**

- **Implementation**
  - Pastry:
    - Implemented in java
    - Report on experimental results
    - Applications running on top of it
- **Assumptions**
  - Viceroy: multiple join/leave operations can fail

---

**Comparison: Pastry/Viceroy (2)**

- **Routing table**
  Each Pastry node provides routing information in a state table

- **Locality**
  Pastry has the additional ability to root messages along the shortest distance according to the proximity metric

- **Network**
  - Viceroy: butterfly/ ring combination
  - Pastry: ring