Frequently asked questions from the previous class survey

- Napster
  - Clients are the only ones used for resources?
  - Any backup servers that host resources, so that you have retrieval guarantees?
  - How can you locate a specific node with partial information about other nodes?

Topics covered in this lecture

- Pastry
  - Simplified algorithm
  - The complete routing algorithm
  - Assimilation of new nodes
  - Host failures and departures

Implementing DHTs

- Mapping keys to nodes
- Forwarding a lookup for a key to the appropriate node
- Building routing tables

Implementing DHTs:

- Mapping keys to nodes
  - Must be load balanced
  - Done using one-way hash functions
    - MD5 (128-bit) or SHA-1 (160-bit)
  - Ensures that content is distributed uniformly
Implementing DHTs:

Forwarding lookups
- Any node that receives query for key
  - Must forward it to a node whose ID is closer to the key
- Above rule guarantees that query eventually arrives at the closest node
- For e.g.:
  - Node has ID 346, and key has ID 542
  - Forwarding to node 495 gets it numerically closer

Distributed hash tables:
Identifiers
- Data items are assigned an identifier from a large random space
  - 128-bit UUIDs or 160-bit SHA1 digests
- Nodes are also assigned a number from the same identifier space

Crux of the DHT problem
- Implement an efficient, deterministic scheme to
  - Map data items to node
- When you look up a data item
  - Network address of node holding the data is returned

Pastry
- All nodes and objects are assigned 128-bit UUIDs
- Applies secure hash function to:
  - The public-key assigned to each node ➔ Node GUID
  - The object’s name or some part of the object’s stored state
Resulting GUIDs have usual properties of secure hash values

- They are randomly distributed in the range $0 - (2^{128} - 1)$
- Provide no clue about the values from which they were computed
- Collisions in the GUID space (for nodes and objects) are extremely unlikely

The Pastry routing

- The number of nodes in the network, $N$
- The algorithm will correctly route messages addressed to any GUID in $O(\log N)$ steps
  - Delivered to an active node whose GUID is numerically closest to it
- Active nodes take responsibility for processing requests addressed to all objects in their numerical neighborhood

Pastry routing

- Routing transfers message to a node that is closer to its destination
- Closeness is in an artificial space
  - The space of GUIDs

Minimizing unnecessarily extended transport paths

- Pastry uses a locality metric based on network distance
  - Hop-counts, round-trip delay measurements
- Uses locality metric to select appropriate neighbors when setting up the routing tables

Managing churn: Nodes joining and leaving the system

- Fully self-organizing
  - When new nodes join the overlay?
    - Obtain data needed to construct routing table and other required state from existing members
      - In $O(\log N)$ messages; $N$ is the number of hosts in overlay
  - When a node fails or departs?
    - Remaining nodes detect its absence
    - Nodes cooperatively reconfigure to reflect required changes in routing structure
      - In $O(\log N)$ messages

The Pastry Routing Algorithm
We will look at the routing algorithm in two parts:

- **STAGE I**: A simplified form
  - Routes messages correctly but inefficiently without a routing table
  - Features:
    - Full routing algorithm
    - Routes requests to any node in $O(\log N)$ messages

- **STAGE II**: A modified approach that uses a routing table
  - Full routing algorithm
  - Routes requests to any node in $O(\log N)$ messages

### Stage I: Simplified Pastry Algorithm

- Each active node stores a **leaf set**
  - A vector $L$ of size $2^{l}$
  - Contains GUIDs and IP addresses of nodes
    - With GUIDs that are numerically closer on either side of its own
    - $l$ above and $l$ below
  - Leaf sets are maintained as nodes join and leave

### Pastry GUID space

- Is treated as a **circular space**
  - Similar to Chord
- GUID 0’s lower neighbor is $2^{128} - 1$

### Invariant of the Pastry system

- Leaf sets reflect a recent state of the system, and that they converge on the current state
- In the face of failures, up to some maximum failure rate

### Stage 1:

- Leaf set for a node contains the GUIDs and IP addresses of the node’s immediate neighbors
- With correct leaf sets of size at least 2:
  - Message routing to any node is possible
  - Node $A$ that receives a message $M$ with destination address $D$
    - Compares $D$ with its own GUID $A$ and with each of the GUIDs in the leaf-set
    - Forwards $M$ to nodes in leaf-set that are numerically closest to $D$
STAGE 1: Pastry routing example with leaf sets of size 8 (l=4)

Routing of message D46A1C from node 65A1FC

Stage 1: Routing analysis
- It will require about N/2 hops to deliver a message in a network with N nodes
- Number of hops is very inefficient

Stage 2: Pastry Routing
- Each node maintains a tree-structured routing table
- Table contains GUIDs and IP addresses for nodes spread throughout the 2128 possible GUID values
- Increased density of coverage for GUIDs numerically closer to its own

Structure of the routing table
- GUIDs are viewed as hexadecimal values
- Table classifies GUIDs based on their hexadecimal prefixes
- Table has as many rows as there are hexadecimal digits in a GUID
  - For a 128-bit GUID 128/4 = 32 rows
- Any row n contains 15 entries
  - 1 for each possible value of the n-th hexadecimal digit
  - Excludes values in the local node’s GUID

Structure of the routing table at node 65A1

Each entry points to one of the potentially many nodes whose GUIDs have a relevant prefix.
Pastry’s Routing Algorithm

```c
if (L1 < D < L2) {
    /* Destination is within leaf set or is the current node */
    forward M to element Li of the leafset with GUID closest to D or the current node A
} else {
    /* Use the routing table to dispatch M to a node with a closer GUID */
}
```

September 12, 2019

Using the Routing Table: Core concept

- Compare the hexadecimal digits of D with those of A (this is the GUID of the current node where the message is being processed)
- Comparison proceeds from left-to-right to discover the length, p, of their longest common prefix
  - Used as row offset
  - The first non-matching digit of D is used as the column offset
  - This gets us to the required element in the routing table
- Construction of the routing table ensures that this element (if not empty) contains the IP address of a node whose GUID has (p + 1) prefix digits in common with D

September 12, 2019

Using the routing table to dispatch M to a node with a closer GUID [1/3]

- R[p, i]: Element at row p and column i of the routing table
- Find
  - p: the length of the longest common prefix of D and A
  - i: the (p + 1)th hexadecimal digit of D

September 12, 2019

Using the routing table to dispatch M to a node with a closer GUID [2/3]

- If (R[p, i] ≠ null) forward M to R[p, i]
- Route M to a node with a longer common prefix
- This step comes into play when:
  - D does not fall within the numeric range of current node’s leaf set
  - Relevant routing table entries are available

September 12, 2019

Using the routing table to dispatch M to a node with a closer GUID [3/3]

- If (R[p, i]) is null?
  - Forward M to any node in L or R with a common prefix of length p but a numerically closer GUID
- D falls outside the numeric range of leaf set and there isn’t a relevant routing table entry
  - Rare!
- If it is in R?
  - Then it must be closer to D than any node in L.
  - We are improving on Stage 1

September 12, 2019

INTEGRATING NEW NODES INTO PASTRY

September 12, 2019
Adding new nodes

- New nodes use a joining protocol
- Join protocol allows:
  - The new node to acquire their routing table and leaf set contents
  - Notifying other nodes of changes that they must make to their tables

Let's look at the join protocol involving a new node

- New node's GUID is X
- Nearby node that this new node contacts is A
- Node X sends a special join request message to A
  - Giving X as its destination
  - Node A dispatches the join message via Pastry
  - Pastry will route message to an existing node with GUID numerically closest to X
  - Let's call this the destination node Z

Routing and transmissions relating to the join message

- The join message is routed through the network:
  - A, Z and intermediate nodes (B, C, ...)
- This results in the transmission of relevant parts of their routing tables and leaf sets to X
- X examines and constructs its own routing table and leaf set from them

How X builds its own routing table

1/2

- First row of X depends on the value of X's GUID
  - To minimize routing distances, table should be constructed to route messages via neighboring nodes
  - A is a neighbor of X, so first row of A's table A_0 is a good initial choice for the first row of X's table X_0

- A's table is not relevant for the second row
  - GUIDs for X and A may not share the first hexadecimal digit
- But the routing algorithm ensures that:
  - X and B's GUID do share the first hexadecimal digit
  - Second row of B's routing table B_1 is a suitable initial value for X_1
  - Similarly, C_2 is suitable for X_2 and so on.

Leaf sets for X

- Since Z's GUID is numerically closest to X's
  - X's ideal leaf set will differ from Z's by just one member
- Z's leaf set is an adequate approximation
  - Eventually optimized through interaction with the neighbors

How X builds its own routing table

2/2
Once \( X \) has constructed the its leaf set and routing table ...

- \( X \) sends their contents to all nodes identified in the leaf set and the routing table.
- The nodes that receive these updates, adjust their own tables to incorporate the node.

## Detection and coping with node failures

- When a node's immediate neighbors (in GUID space) cannot communicate with it:
  - The node is considered failed.
- Necessary to repair leaf sets and routing tables that contain the failed GUID.
  - Leaf sets are repaired proactively.
  - Routing tables at the other nodes are updated on a “when discovered basis.”

## Repairing leaf sets

- Node that discovers the failure:
  - Looks for a live node close to the failed node, and requests copy of that node's leaf set, \( L' \).
  - This should contain GUIDs that partly overlap those in the node that discovered failure.
  - Include one that should replace the failed node.
- Other neighboring nodes are informed:
  - They perform a similar procedure.

## Locality

- Pastry routing structure is redundant.
  - Multiple routes between pairs of nodes.
- Construction of routing tables tries to take advantage of this redundancy.
  - Reduce message transmission times by exploiting locality properties of underlying network.

## Routing table: Exploiting locality. [1/2]

- In the routing table, each row contains 16 entries.
  - Entries in the \( i \)th row give addresses of 16 nodes with GUIDs with \( i-1 \) initial hexadecimal digits.
  - \( i \)th digit takes each of the possible hexadecimal values.
- Well-populated Pastry system contains more nodes than can be contained in an individual routing table.
Routing table: Exploiting locality. [2/2]

- When routing table is constructed choice is made for each position
  - Between multiple candidates
  - Based on proximity neighbor selection
- Locality metric
  - IP hops or measured latency

Performance of exploiting locality

- Since the information in the routing table is not comprehensive
  - Mechanism does not produce globally optimal routing
- Simulations show that
  - On average, the routing is 30-50% longer than the optimum

Coping with malicious nodes

- Small degree of randomness is introduced into route selection
- Randomized to yield a common prefix that is less than the maximum length
  - With a certain probability
- Routes are taken from an earlier row
  - Less optimal, but different than standard version
  - Client transmission succeed in the presence of small numbers of malicious nodes

The contents of this slide-set are based on the following references