PAST: P2P Storage Utility

References:
3. P. Druschel, Presentation on Pastry and PAST. http://www.cs.rice.edu/~druschel/comp413/lectures/Pastry.ppt

Presentation by Nathaniel Crowell
Outline

- Review Structured P2P
- Pastry
- Pastry Routing
- PAST
Structured P2P Systems (recall Chord)

- Requirements of Basic Protocol:
  - Route to a node corresponding to a provided key-id
- Qualities:
  - Distributed
  - Decentralized Control
  - Self-organizing
  - Symmetry
- Potential Problems:
  - Maintenance of overlay network
  - Load Balancing
  - Routing efficiency measured based on overlay
How is Pastry Different?

Chord
- Skip list
  - Routing: $O(\log N)$
  - Memory: $O(\log N)$

Pastry
- Routing table
  - Routing: $O(\log_c N)$
- Memory: $O(c \cdot \log_c N)$
  - $c$ is $2^b$
- Leaf Set
  - $L$ closest NodeIds
  - Efficient Routing
  - Robustness
- Neighborhood Set
  - Proximity metric
  - Efficient routing

How do they differ when $b=1$?
Pastry: Routing Table

- Leaf Set
  - Nearest by NodeId
- Routing Table
- Neighborhood Set
  - Nearest by proximity

- Maintenance more costly than Chord but not overwhelming
Pastry: Routing

b is 4 in this example
Pastry: Routing Procedure

if (destination is within range of our leaf set) forward to numerically closest member 
else
let l = length of shared prefix
let d = value of l-th digit in D’s address
if (Rld exists)
forward to Rld
else
forward to a known node that
(a) shares at least as long a prefix
(b) is numerically closer than this node
Pastry: Additional Details

- Routing Table
  - Preference given to nodes with better proximity.
- Performance
  - Success provable unless L/2 adjacent failures.
  - Routing usually $O(\log N)$, worst case $O(N)$
- Like Chord has additional functionality for
  - Node addition
  - Node departure
- Experiments show expected results
  - Protocol works under stated assumptions
  - Efficiency deteriorates without repair from failures
PAST: P2P Storage and Distribution

- **Goals:**
  - Strong persistence
  - High availability
  - Scalability
  - Security

- **Simple API:**
  - Insert
  - Lookup
  - Reclaim (delete)
PAST: File Storage

Storage Invariant:
File "replicas" are stored on k nodes with nodeIds closest to fileId

(k is bounded by the leaf set size)
PAST: File Retrieval

file located in $O(\log N)$ steps (expected)

usually locates replica nearest client C
PAST: Exploiting Pastry

- Random, uniformly distributed nodeIds
  - replicas stored on diverse nodes
- Uniformly distributed fileIds
  - e.g. SHA-1(filename, public key, salt)
  - approximate load balance
- Pastry routes to closest live nodeId
  - availability, fault-tolerance
PAST: Diversion

- Diversion occurs as available space decreases
- Replica Diversion
  - Load-balancing over a leaf set
- File Diversion
  - Load-balancing over entire network
- Experiment results: Diversion necessary
  - No diversion ($t_{\text{pri}} = 1, t_{\text{div}} = 0$):
    - max utilization 60.8%
    - 51.1% inserts failed
  - Replica/file diversion ($t_{\text{pri}} = .1, t_{\text{div}} = .05$):
    - max utilization > 98%
    - < 1% inserts failed
PAST: Caching

- Nodes cache files in the unused portion of their allocated disk space
- Files caches on nodes along the route of lookup and insert messages

Goals:
- Maximize query throughput for popular documents
- Balance query load
- Improve client latency
Questions?