Introduction

- Core operation in peer-to-peer systems is to efficiently locate the node that stores a particular data item.
- Chord is a scalable distributed protocol for lookup in a dynamic peer-to-peer system with frequent node arrivals and departures.
- Only one operation: given a key, it maps the key onto a node.
- Simplicity, provable correctness, and provable performance.
The lookup problem

Centralized lookup (Napster)

Simple, but $O(N)$ states and a single point of failure
Flooded queries (Gnutella)

Robust, but worst case $O(N)$ messages per lookup

Routed queries (Freenet, Chord, etc.)
Related Work

• Freenet (Clarke, Sandberg, Wiley, Hong)
• CAN (Ratnasamy, Francis, Handley, Karp, Shenker)
• Pastry (Rowstron, Druschel)
• Tapestry (Zhao, Kubiatowicz, Joseph)
  ... ...
• Chord emphasizes simplicity

Design Objectives

• Load Balance: Distributed hash function spreads keys evenly over the nodes (Consistent hashing).
• Decentralization: Fully distributed (Robustness).
• Scalability: Lookup grows as a log of number of nodes.
• Availability: Automatically adjusts internal tables to reflect changes.
• Flexible Naming: No constraints on key structure.
Applications

- Lookup(key) algorithm that yields the IP address of the node responsible for the key.
- Notify the application of changes in the set of keys that the node is responsible for.
- Example applications:
  - Cooperative Mirroring
  - Time-shared storage
  - Distributed indexes
  - Large-Scale combinatorial search

Routing challenges

- Define a useful key nearness metric.
- Keep the hop count small.
- Keep the routing tables small.
- Stay robust despite rapid changes.
Chord properties

- Efficient: $O(\log(N))$ messages per lookup.
- Scalable: $O(\log(N))$ states per node.
- Robust: survives massive failures, join or leave. $O(\log^2(N))$ messages.
- An $N^{th}$ node joins (or leaves), only an $O(1/N)$ keys are moved to a different location.
  Proofs are in paper / tech report.
  (Assuming no malicious participants)

Chord overview

- Provides peer-to-peer hash lookup:
  - Lookup(key) → IP address.
  - Chord does not store the data.
- How does Chord route lookups?
- How does Chord maintain routing tables?
- How does Chord cope with changes in membership?
Chord IDs

- m-bit identifier space for both keys and nodes.
- Key identifier = SHA-1(key).
- Node identifier = SHA-1(IP address).
- Both are uniformly distributed.

- How to map key IDs to node IDs?

Consistent hashing [Karger 97]

A key is stored at its successor: node with next higher ID
Basic lookup

“Where is key 80?”

“N90 has K80”

“Finger table” allows log(N)-time lookups

Every node knows $m$ other nodes in the ring
Finger $i$ points to successor of $n+2^{i-1}$

Each node knows more about portion of circle close to it

Lookups take $O(\log(N))$ hops
Chord Summary

- Chord provides peer-to-peer hash lookup.
- Efficient: $O(\log(n))$ messages per lookup.
- Robust as nodes fail and join.
- Good primitive for peer-to-peer systems.

http://www.pdos.lcs.mit.edu/chord
OceanStore Context: Ubiquitous Computing

• Computing everywhere:
  – Desktop, Laptop, Palmtop
  – Cars, Cellphones
  – Shoes? Clothing? Walls?
• Connectivity everywhere:
  – Rapid growth of bandwidth in the interior of the net
  – Broadband to the home and office
  – Wireless technologies such as CMDA, Satellite, laser

Questions about information:

• Where is persistent information stored?
  – Want: Geographic independence for availability, durability, and freedom to adapt to circumstances
• How is it protected?
  – Want: Encryption for privacy, signatures for authenticity, and Byzantine commitment for integrity
• Can we make it indestructible?
  – Want: Redundancy with continuous repair and redistribution for long-term durability
• Is it hard to manage?
  – Want: Automatic optimization, diagnosis and repair
• Who owns the aggregate resources?
  – Want: Utility Infrastructure!
• Transparent data service provided by federation of companies:
  – Monthly fee paid to one service provider
  – Companies buy and sell capacity from each other

OceanStore: Everyone’s Data, One Big Utility
“The data is just out there”

• How many files in the OceanStore?
  – Assume $10^{10}$ people in world
  – Say 10,000 files/person (very conservative?)
  – So $10^{14}$ files in OceanStore!

  – If 1 gig files (ok, a stretch), get 1 mole of bytes!

Truly impressive number of elements…
… but small relative to physical constants
Aside: new results: 1.5 Exabytes/year ($1.5 \times 10^{18}$)
Outline

- Motivation
- Assumptions of the OceanStore
- Specific Technologies and approaches:
  - Naming
  - Routing and Data Location
  - Conflict resolution on encrypted data
  - Replication and Deep archival storage
  - Introspection for optimization and repair
- Conclusion

OceanStore Assumptions

- Untrusted Infrastructure:
  - The OceanStore is comprised of untrusted components
  - Only ciphertext within the infrastructure
- Responsible Party:
  - Some organization (i.e. service provider) guarantees that your data is consistent and durable
  - Not trusted with content of data, merely its integrity
- Mostly Well-Connected:
  - Data producers and consumers are connected to a high-bandwidth network most of the time
  - Exploit multicast for quicker consistency when possible
- Promiscuous Caching:
  - Data may be cached anywhere, anytime
- Optimistic Concurrency via Conflict Resolution:
  - Avoid locking in the wide area
  - Applications use object-based interface for updates
Use of Moore’s law gains

- Question: Can we use Moore’s law gains for something other than just raw performance?
  - Growth in computational *performance*
  - Growth in network *bandwidth*
  - Growth in *storage* capacity

- Examples:
  - Stability through Statistics
    - Use of redundancy of servers, network packets, *etc.* in order to gain more predictable behavior
  - Extreme Durability (1000-year time scale?)
    - Use of erasure coding and continuous repair
  - Security and Authentication
    - Signatures and secure hashes in many places
  - Continuous dynamic optimization

Basic Structure: Irregular Mesh of “Pools”
Secure Naming

- Unique, location independent identifiers:
  - Every *version* of every unique entity has a permanent, *Globally Unique ID (GUID)*
  - All OceanStore operations operate on GUIDs

- Naming hierarchy:
  - Users map from names to GUIDs via hierarchy of OceanStore objects (*ala SDSI*)
  - Requires set of “root keys” to be acquired by user

![Diagram of Secure Naming]

Unique Identifiers

- Secure Hashing is key!
  - Use of 160-bit SHA-1 hashes over information provides uniqueness, unforgeability, and verifiability:
    - *Read-only data*: GUID is hash over actual information
      - Uniqueness and Unforgeability: the data is what it is!
      - Verification: check hash over data
    - *Changeable data*: GUID is combined hash over a human-readable name + public key
      - Uniqueness: GUID space selected by public key
      - Unforgeability: public key is indelibly bound to GUID
      - Verification: check signatures with public key

- Is 160 bits enough?
  - Birthday paradox requires over $2^{80}$ unique objects before collisions worrisome
  - Good enough for now
Routing and Data Location

- **Requirements:**
  - Find data quickly, wherever it might reside
    - Locate nearby data without global communication
    - Permit rapid data migration
  - Insensitive to faults and denial of service attacks
    - Provide multiple routes to each piece of data
    - Route around bad servers and ignore bad data
  - Repairable infrastructure
    - Easy to reconstruct routing and location information

- **Technique: Combined Routing and Data Location**
  - Packets are addressed to GUIDs, not locations
  - Infrastructure gets the packets to their destinations and verifies that servers are behaving

Two-levels of Routing

- **Fast, probabilistic search for “routing cache”:**
  - Built from *attenuated* bloom filters
  - Approximation to gradient search
  - *Not going to say more about this today*

- **Redundant *Plaxton Mesh* used for underlying routing infrastructure:**
  - Randomized data structure with locality properties
  - Redundant, insensitive to faults, and repairable
  - Amenable to continuous adaptation to adjust for:
    - Changing network behavior
    - Faulty servers
    - Denial of service attacks
Attenuated Bloom Filter

• An attenuated Bloom filter of depth $d$ is an array of $d$ normal bloom filters
• For each neighbor link, an attenuated Bloom filter is kept
• The $k$th bloom filter in the array is the merger of all Bloom filters for all of the nodes $k$ hops away through any path starting with that neighbor link

How is a Query Handled?

• The query node examines the first level of each of its neighbors’ filters
  – If matches are found, the query is forwarded to closest neighbor
• If none of the filters match, then the querying node examines the next level of each filter at each step and forwards the query if a matching node is found
Probabilistic Algorithm

Searching for an object with GUID 11010

Basic Plaxton Mesh
Incremental suffix-based routing
Use of Plaxton Mesh
Randomization *and* Locality

- As in original Plaxton scheme:
  - Scheme to directly map GUIDs to root node IDs
  - Replicas publish toward a document root
  - Search walks toward root until pointer located \(\Rightarrow\) *locality*

- OceanStore enhancements for reliability:
  - Documents have multiple roots (Salted hash of GUID)
  - Each node has multiple neighbor links
  - Searches proceed along multiple paths
    - Tradeoff between reliability and bandwidth?
  - Routing-level validation of query results

- Dynamic node insertion and deletion algorithms
  - Continuous repair and incremental optimization of links
OceanStore Consistency via Conflict Resolution

- Consistency is form of optimistic concurrency
  - An update packet contains a series of *predicate-action* pairs which operate on encrypted data
  - Each predicate tried in turn:
    - If none match, the update is *aborted*
    - Otherwise, action of first true predicate is *applied*

- Role of Responsible Party
  - All updates submitted to Responsible Party which chooses a final total order
  - Byzantine agreement with threshold signatures

- This is powerful enough to synthesize:
  - ACID database semantics
  - release consistency (build and use MCS-style locks)
  - Extremely loose (weak) consistency

Oblivious Updates on Encrypted Data?

- Tentative Scheme:
  - Divide data into small blocks
  - Updates on a per-block basis
  - Predicates derived from techniques for searching on encrypted data

- Still exploring other options

<table>
<thead>
<tr>
<th>TimeStamp</th>
<th>Client ID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>{Pred1, Update1}</td>
</tr>
<tr>
<td></td>
<td>{Pred2, Update2}</td>
</tr>
<tr>
<td></td>
<td>{Pred3, Update3}</td>
</tr>
</tbody>
</table>

Unique Update ID is hash over packet
Data Coding Model

- Two distinct forms of data: active and archival
- **Active Data** in Floating Replicas
  - Per object virtual server
  - Logging for updates/conflict resolution
  - Interaction with other replicas to keep data consistent
  - May appear and disappear like bubbles
- **Archival Data** in Erasure Coded Fragments
  - OceanStore equivalent of stable store
  - During commit, previous version coded with erasure-code and spread over 100s or 1000s of nodes
  - Fragments are self-verifying
  - Advantage: *any* 1/2 or 1/4 of fragments regenerates data
Introspective Optimization

- Monitoring and adaptation of routing substrate
  - Optimization of Plaxton Mesh
  - Adaptation of second-tier multicast tree
- Continuous monitoring of access patterns:
  - Clustering algorithms to discover object relationships
    - Clustered prefetching: demand-fetching related objects
    - Proactive-prefetching: get data there before needed
  - Time series-analysis of user and data motion
- Continuous testing and repair of information
  - Slow sweep through all information to make sure there are sufficient erasure-coded fragments
  - Continuously reevaluate risk and redistribute data
  - Diagnosis and repair of routing and location infrastructure
  - *Provide for 1000-year durability of information?*
First Implementation [Java]:

- Event-driven state-machine model
- Included Components
  - Initial floating replica design
    - Conflict resolution and Byzantine agreement
  - Routing facility (Tapestry)
    - Bloom Filter location algorithm
    - Plaxton-based locate and route data structures
  - Introspective gathering of tacit info and adaptation
    - Language for introspective handler construction
    - Clustering, prefetching, adaptation of network routing
  - Initial archival facilities
    - Interleaved Reed-Solomon codes for fragmentation
    - Methods for signing and validating fragments
- Target Applications
  - Unix file-system interface under Linux (“legacy apps”)
  - Email application, proxy for web caches, streaming multimedia applications

OceanStore Conclusions

- OceanStore: everyone’s data, one big utility
  - Global Utility model for persistent data storage
- OceanStore assumptions:
  - Untrusted infrastructure with a responsible party
  - Mostly connected with conflict resolution
  - Continuous on-line optimization
- OceanStore properties:
  - Local storage is a cache on global storage
  - Provides security, privacy, and integrity
  - Provides extreme durability
  - Lower maintenance cost through continuous adaptation, self-diagnosis and repair
  - Large scale system has good statistical properties
- http://oceanstore.cs.berkeley.edu/