Canon in G Major: Designing DHTs with Hierarchical Structure

Prasanna Ganesan (Stanford University)
Krishna Gummadi (U. of Washington)
Hector Garcia-Molina (Stanford University)
Distributed Hash Table (DHT)

- Hash table over dynamic set of nodes
  - \text{Insert(key)}, \text{Lookup(key)}
  - \text{Join(node)}, \text{Leave(node)}
- Partition hash space by node ID
- “Flat” overlay network
  - \( O(\log n) \) operations
  - Homogeneous components
  - No central point of failure
Why Hierarchies?

• Hierarchies exist!

• Scalability, fault isolation, autonomy, etc.

• Goal: Inherit best of both worlds with a hierarchical DHT
What is a Hierarchical DHT?

Path Locality, Path Convergence, Efficiency, Caching, Security, Bandwidth, Fault Isolation, Optimization, Access Control
Problem Statement

• Convert flat DHTs to hierarchical (Canon-
  ization)
  – Chord ⇒ Crescendo
  – CAN ⇒ Can-Can
  – Symphony ⇒ Cacophony
  – Kademlia ⇒ Kandy

• Caveat: Preserve homogeneity and state vs.
  routing trade-offs of flat DHTs
Roadmap

- Chord and Crescendo
- Routing in Crescendo
- Storage and Caching
- Experimental Evaluation
Chord

- Circular N-bit ID space
- Node $x$ links to $\text{succ}(x + 2^i)$
Crescendo

- Key idea: Recursive structure
  - Construct bottom-up; merge smaller DHTs
  - Lowest level: Chord

Stanford
Merging two Chord rings

Black node $x$: Connect to $y$ iff
- $y$ closer than any other black node
- $y = \text{succ}(x + 2^i)$
Crescendo Details

• Generalize two-ring merge
  – Merging multiple rings
  – Multi-level hierarchies

• Making it incremental
  – New node joins bottom-up

• How many links per node?
  – Roughly log $n$ ; independent of hierarchy

\[
\text{Total} = \log m + \log (n/m + 1)
\]
Routing in Crescendo

- Greedy clockwise routing!

- Path locality by greedy routing!
- Path convergence at closest node to destination of same color
- Local DHTs by construction!
Extensions

• Can apply to other DHTs

• Mix-and-match DHTs
  – Stanford runs Chord, Washington runs Kademlia

• Additional support for proximity
Storage and Caching

• Hierarchical Storage
  – Specify storage domain subtree
• Access control provided by routing itself
• Caching: Cache at convergent nodes at each level
  – Nodes form a distributed cache
Experiments

• *c-level* hierarchy
  – Uniform/Zipf distribution at each level

• Basic metrics
  – #links / node
  – #hops / query
Number of Links vs. Number of Levels

![Graph showing the relationship between Number of Links and Number of Nodes for different Levels.

Legend:
- Chord
- Levels=2
- Levels=3
- Levels=4
- Levels=5]
Levels vs. Routing Hops

![Graph showing the relationship between the number of nodes and the number of hops for different levels. The graph has a logarithmic scale on the x-axis for the number of nodes and a linear scale on the y-axis for the number of hops. The levels tested are 2, 3, and 4, with Chord as a reference.](image-url)
Path Locality

- GT-ITM topology as hierarchy
- Compare Crescendo with Chord
  - With proximity adaptation
- Path locality
  - Latency to different level destinations
Path Locality

Graph showing latency (ms) vs query locality for Chord (Prox.), Crescendo, and Crescendo (Prox.) across different query locations: Top, First, Second, Third, and Fourth.
Conclusions

• Generic mechanism for hierarchical DHTs
  – *Locality*: Fault isolation, Security, Efficiency
  – *Convergence*: Caching, Bandwidth savings
  – *Local DHTs*: Hierarchical Access Control
  – Preserves degree vs. routing trade-offs

• Potentially useful in large-scale deployment