Chord on Demand

Alberto Montresor
Mark Jelasity
Ozalp Babaoglu

Department of Computer Science
University of Bologna
Italy

Summary
1) Motivation
2) Chord
3) T-Man
4) T-Chord
5) Results
6) Conclusions
Chord on Demand

Alberto Montresor
Mark Jelasity
Ozalp Babaoglu

Department of Computer Science
University of Trento
Italy
Motivation

- **Topology jump-starting**
  - Informal definition: *building a topology from the ground up as quickly and efficiently as possible*

- **Scenarios:**
  - Enable the rapid deployment of overlay networks
  - Protocol switch
  - Future P2P grid systems

- **Why Chord?**
  - It's just an example
  - It is the more “natural” given the approach of this paper
  - We are now able to generate any “routing” topology
Motivation

- **Do not confuse with node bootstrap**
  - Placing a single node in the right place in the topology

- **Do not confuse with topology maintenance**
  - Stabilization (routing table cleanup)
  - Can be used later to optimize the topology

- **Current DHT protocols do not support jump-starting:**
  - They assume an already formed network
  - Even supposing the network exists, they work “unless a tremendous number of nodes joins the system” [Chord]
  - We could envision a “join orchestration approach”, but it would require a linear time
Chord: Basic structure

Every node knows its successor(s) in the ring

Where is “LetItBe”?

Hash(“LetItBe”) = K60

“N90 has K60”
Chord: Fingers

• Finger table with $m$ entries at each node

• Increase distance exponentially: 
  \[ \text{Finger } i \text{ points to successor of } n+2^i \text{ mod } 2^m \]

• Lookup is $O(\log n)$

• On average, only $O(\log N)$ entries in the table are non-null

Note: Chord figures from a presentation of Robert Morris
Joining the ring

- **Three step process:**
  - Initialize all fingers of new node
  - Update fingers of existing nodes
  - Transfer keys from successor to new node
  - $O(\log^2 N)$

- **Less aggressive mechanism (lazy finger update):**
  - Initialize only the successor(s) node
  - Periodically verify immediate successor, predecessor
  - Periodically refresh finger table entries
Overlay computing through gossip protocols

- Top. bootstrap
  - T-Pastry

- Top. bootstrap
  - T-Chord

- Topology man., ranking
  - T-Man

- Load balancing

- Ranking

- Aggregation

- Semantic topologies
  - Proximity selection
    - Quickpeer

- Random substrate
  - Newscast, Cyclon
The T-Man algorithm

// view is a collection of neighbors
Init: view = rnd.view ∪ { (myaddress, mydescriptor) }

// active thread
// executed by p
do once every δ time units
    q = selectNeighbor(view)
    msg_p = extract(view, q)
    send msg_p to q
    receive msg_q from q
    view = merge(view, msg_q)

// passive thread
// executed by p
do forever
    receive msg_q from *
    msg_p = extract(view, q)
    send msg_p to q
    view = merge(view, msg_q)

A "round" of length δ
T-Man for T-Chord

- **selectPeer():**
  - randomly select a peer \( q \) from the \( r \) nodes in my view that are *nearest to p in terms of ID distance*

- **extract():**
  - send to \( q \) the \( r \) nodes in local view that are *nearest to q*
  - \( q \) responds with the \( r \) nodes in its view that are *nearest to p*

- **merge():**
  - both \( p \) and \( q \) *merge* the received nodes to their view
T-Man for T-Chord

- **selectPeer()**: randomly select a peer $q$ from the $r$ nodes in my view that are *nearest to p in terms of ID distance*

- **extract()**:  
  - send to $q$ the $r$ nodes in local view that are *nearest to q*  
  - $q$ responds with the $r$ nodes in its view that are *nearest to p*

- **merge()**:  
  - both $p$ and $q$ *merge* the received nodes to their view

© 2005 Alberto Montresor
Convergence of the ring
The T-Man algorithm

- **T-man is a generic protocol for topology formation**
  - Topologies are expressed through *distance functions*
  - It can be used to generate rings, tori, trees, DHTs, etc.
  - It can be used for distributed sorting and ranking
  - Merging, selecting, etc. can be optimized toward the goal

- **Examples**
  - Ids in a virtual ID space
  - Semantic proximity for file-sharing
  - Latency for proximity selection
  - Etc.
T-Chord

- **ID distance:**
  - final output is the Chord ring
  - As by-product, many other nodes are discovered

- **Example of execution**
  - Size = $2^{14}$
  - Message size=20 (larger than required, but nicer)
T-Chord

• Run the “T-man with history” until the ring is formed
  • How to stop: size estimation, stability tests

• Use the ring to select successors

• Use the history to fill the finger table
  • Let \( id_p \) be the identifier of \( p \)
  • For each \( i = 0 \ldots m-1 \) (\( m \) is the number of bits)
  • search in the view the node with identifier \( id' \) s.t.
    • \( id' \in [(id_p + 2^i) \mod 2^m, (id_p + 2^{i+1}-1) \mod 2^m] \)
    • \( id' \) is the nearest value in that range
T-Chord-Prox

- Run the “T-man with history” until the ring is formed
  - How to stop: size estimation, stability tests
- Use the ring to select successors
- Use the history to fill the finger table
  - Let $id_p$ be my identifier
  - For each $i = 0 \ldots m-1$ ($m$ is the number of bits)
  - search in the history the node with identifier $id'$ s.t.
    - $id' \in [(id_p + 2^i) \mod 2^m, (id_p + 2^{i+1}-1) \mod 2^m]$
    - the node is the nearest (in terms of latency) among “probed” nodes (randomly selected)
Convergence: loss rate & hops – N=2^{16}, r=10
Scalability: convergence – r=10

1-regular lattice
5-regular lattice

© 2005 Alberto Montresor
Scalability: hops – r=10
Scalability: latency – $r=10$
Message size - hops – $N=2^{16}$

- Hop count
- Message Delay (ms)

Graphs showing the relationship between message size and hop count, as well as message delay, for different protocols: Chord, T-Chord, and T-Chord-Prox.
Measurements studies have estimated that 1% of the nodes fail in 100s (20 cycles x 5 seconds)
Robustness - hops – $N=2^{16}$, $r=10$
Robustness – failed hops – $N=2^{16}$, $r=10$
Conclusions

- **T-MAN can be used to bootstrap Chord**
  - Efficiently: $N \log N$ messages, $\log N$ time
  - In a robust way: 1 order of magnitude more churn than “usual”

- **Related work**
  - [Aberer05, Aspnes05] have considered similar problems

- **Current & Future Work**
  - Theory!
  - The approach can be used with other topologies (Pastry, etc.)
  - Can be used to maintain a DHT topology?
    - Definitely yes, but with some tricks (restarting)
  - How the maintenance version of T-Chord compares with original protocols?