Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications

Article *in* ACM SIGCOMM Computer Communication Review · December 2001

DOI: 10.1145/964723.383071 · Source: CiteSeer

Ion Stoica
University of California, Berkeley

David Ron Karger
Massachusetts Institute of Technology

Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications

Ion Stoica
Robert Morris
David Liben-Nowell
David R. Karger
M. Frans Kaashoek
Frank Dabek
Hari Balakrishnan

CS856

Nabeel Ahmed

Outline

- P2Ps as Lookup Services
- Related Work
- Chord System Model
- Chord Protocol Description
- Simulation Results
- Current Status and Issues
- Extensions of Chord
- References
- Discussion

A P2P Lookup Service?

- P2P system:
  - Data items spread over a large number of nodes
  - Which node stores which data item?
  - A lookup mechanism needed
    - Centralized directory -> bottleneck/single point of failure
    - Query Flooding -> scalability concerns
    - Need more structure!
- Solution: *Chord* (a distributed lookup protocol)
- Chord supports only one operation: *given key, maps key on to a node*

Related Work

- Unstructured Peer-to-Peer Systems
  - Freenet
  - KaZaa/Napster
  - Gnutella
- Structured Peer-to-Peer Systems
  - CAN
  - OceanStore (Tapestry)
  - Pastry
  - Kademlia, Viceroy etc..
- To many routing structures? How to compare?

Related Work (Contd..)

- Routing Geometry: *"Manner in which neighbors and routes are chosen" Gummadi et al.[6]*
- Classify Routing Geometries:
  - Tree → *PRR, Tapestry, Globe system, TOPLUS*
  - Hypercube → *CAN,*
  - Butterfly → *Viceroy*
  - Ring → *Chord*
  - XOR → *Kademlia*
  - Hybrid → *Pastry (Tree/Ring)*
  - *Maybe more….*
- *Compare degree of flexibility in routing geometries*
  - Neighbor Selection
  - Route Selection
- *Comparative discussion later…..*

Chord System Model

- Design Objectives:
  - Load Balance: Distributed hash function spreads keys evenly over the nodes
  - Decentralization: Fully distributed
  - 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.
- Example Applications:
  - Co-operative Mirroring
  - Time-shared storage
  - Distributed indexes
  - Large-Scale combinatorial search

Chord Protocol

- Assumption: Communication in underlying network is both symmetric and transitive.
- Assigns keys to nodes using *consistent hashing*
- Uses *logical ring* geometry to manage identifier space (identifier circle)
- Utilizes *(sequential) successor/predecessor pointers* to connect nodes on ring
- Distributes routing table among nodes (*Finger pointers*)

Consistent Hashing

- Properties:
  - Minimal Disruption: require minimal key movement on node joins/leaves
  - Load Balancing: distribute keys equally across over nodes

Theorem: For any set of N nodes and K keys, with *high probability*:

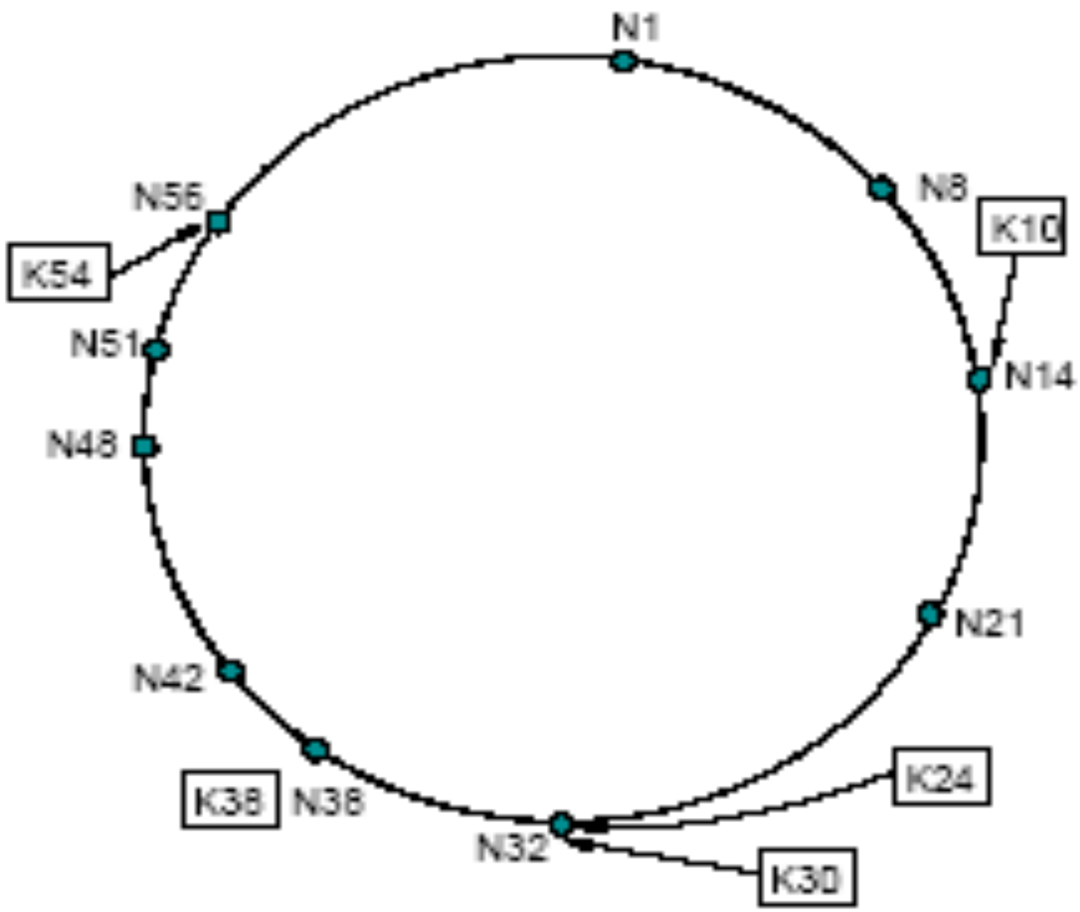
1) Each node is responsible for at most (1+e)K/N keys.
2) When an (N+1)st node joins or leaves the network, responsibility for O(K/N) keys changes hands.

e = O(log N)

Consistent Hashing (Contd..)

- Consistent hashing function assigns each node and key an m-bit identifier using SHA-1 base hash function *(160-bits truncated to m)*.
- Node's IP address is hashed.
- Identifiers are ordered on a identifier circle modulo 2^m called a chord ring.
- *succesor(k)* = first node whose identifier is >= identifier of k in identifier space

Example Chord Ring

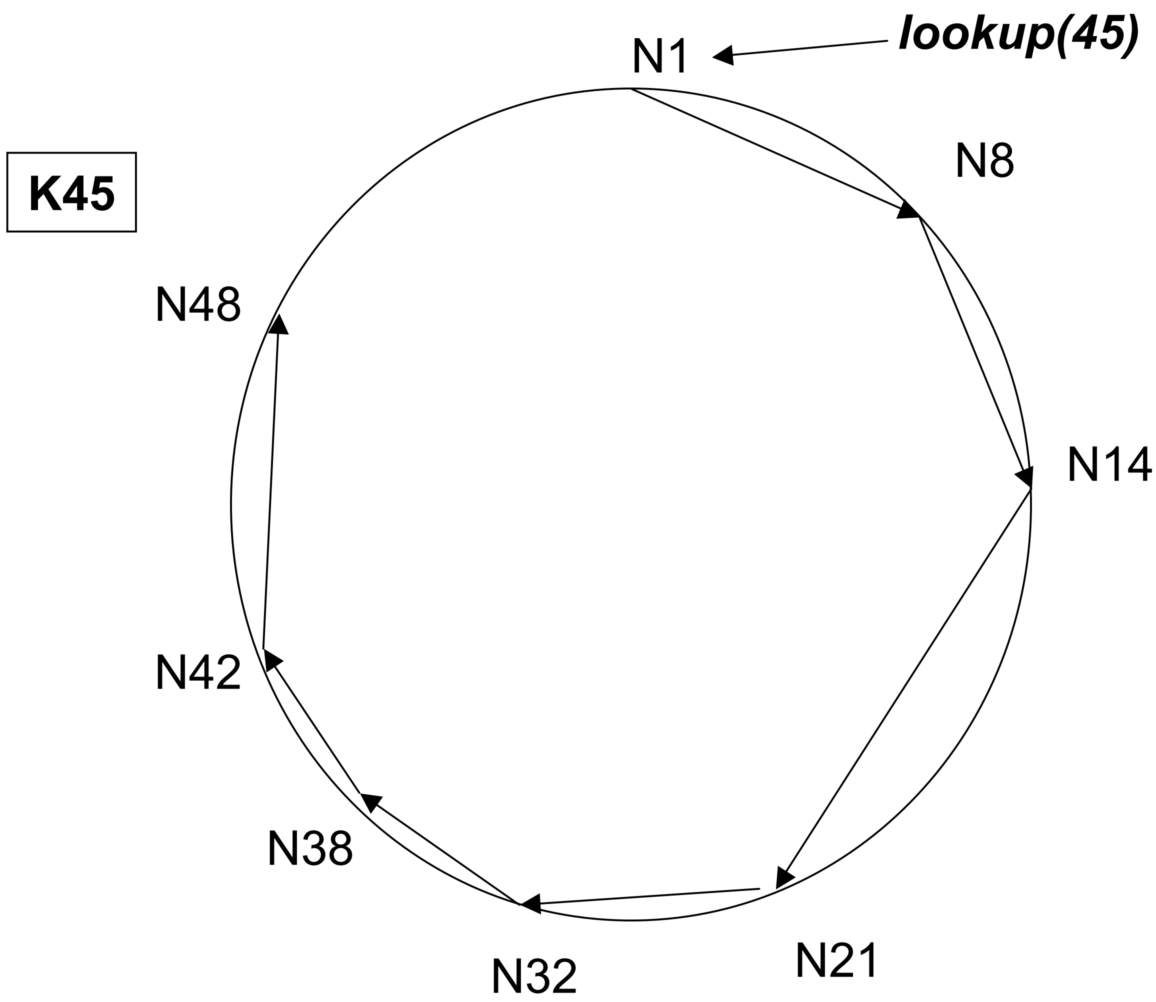


m = 6
10 nodes

Lookups in Chord

- Two techniques:
  - Simple-Key Location scheme:
    - State-maintenance O(1) [no finger table]
    - Lookup-time O(N) [follow successor pointers]
  - Scalable-Key Location scheme:
    - State-maintenance O(log N) [finger table]
    - Lookup-time O(log N) [follow finger pointers]

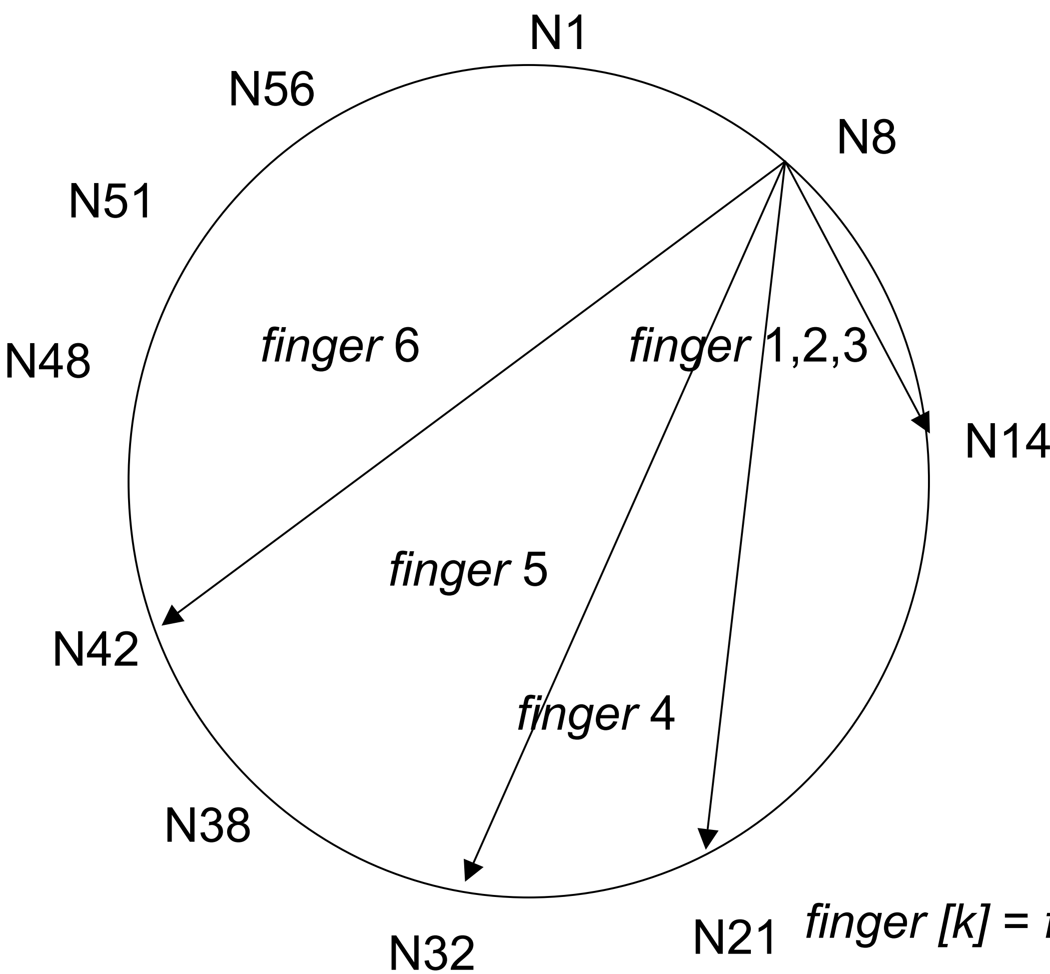
Simple Key Location Scheme



Scalable Key Lookup Scheme

- Finger Pointers
  - n.finger[i] = successor $(n + 2^{i-1})$
  - Each node knows more about portion of circle close to it!
- Query the finger-node that is nearest predecessor of key (closest preceding finger)
- Recursive querying till immediate predecessor p of key found
- Return p.successor

Scalable Lookup Scheme: Finger Table

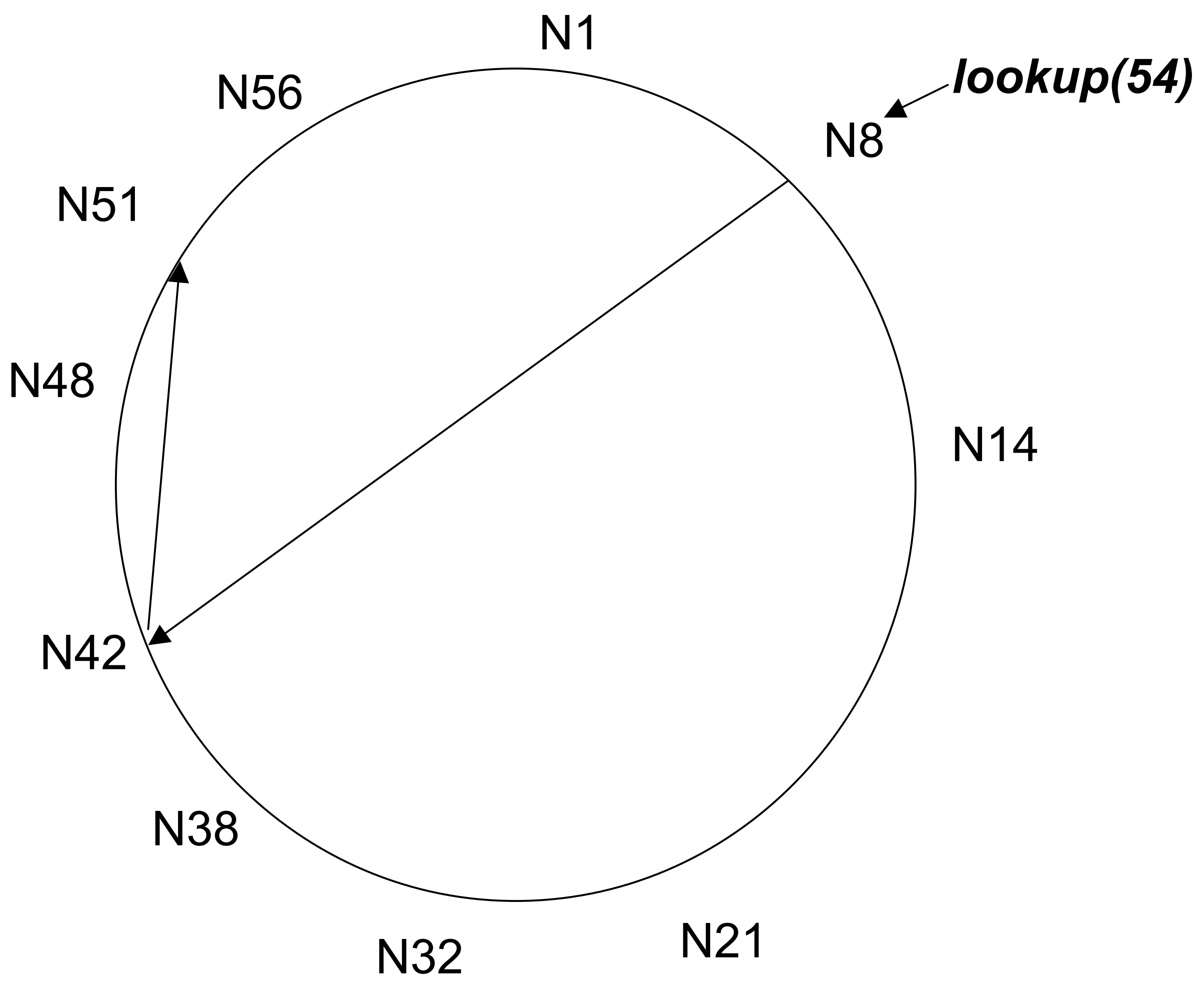


Finger Table for N8

N8+1	N14
N8+2	N14
N8+4	N14
N8+8	N21
N8+16	N32
N8+32	N42

finger [k] = first node that succeeds $(n+2^{k-1}) \bmod 2^m$

Scalable Lookup Scheme



What about Churn?

- Churn: Term used for dynamic membership changes
- Problems related to Churn:
  - Re-delegation of key-storage responsibility
  - Updation of finger tables for routing
- Need to support:
  - Concurrent Node Joins/Leaves (Stabilization)
  - Fault-tolerance and Replication (Robustness)

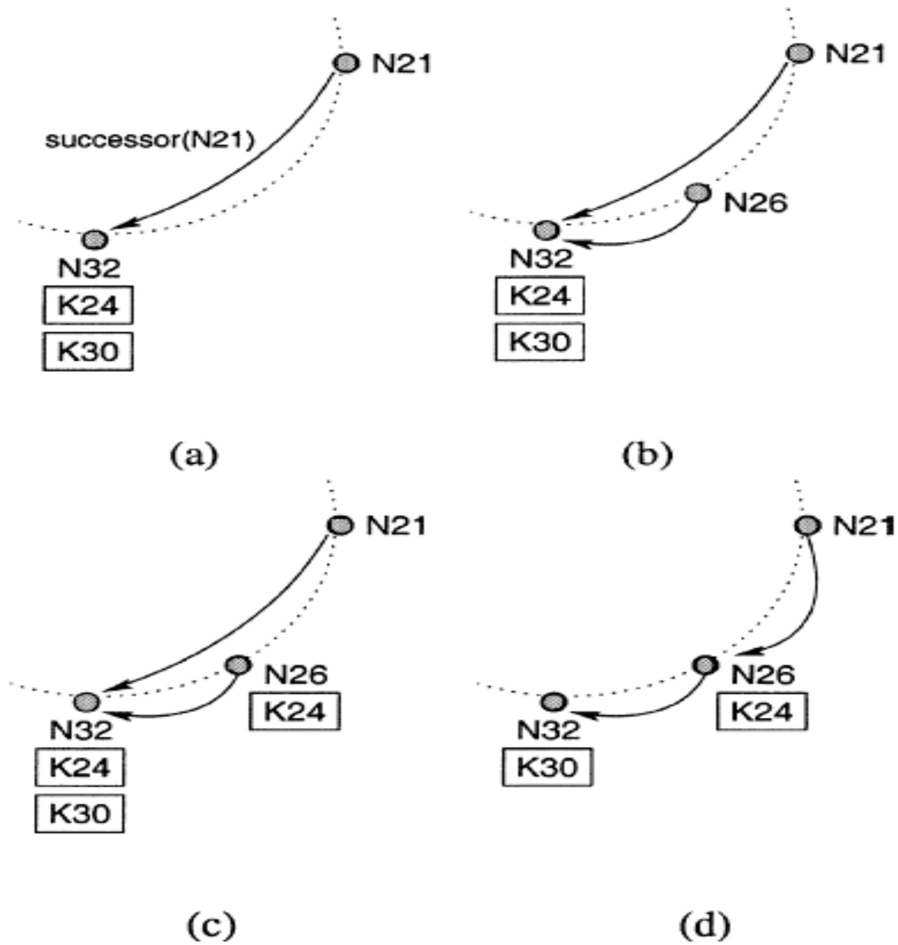
Node Joins

- New node B learns of at least one existing node A via external means
- B asks A to lookup its finger-table information
  - Given B's hash-id b, A does lookup for B.finger[i] = successor (b + 2^{i-1}) if interval not already included in finger[i-1]
  - B stores all finger information and sets up pred/succ pointers
- Updation of finger table required at certain existing nodes
- Key movement is done from successor(b) to b

Concurrent Joins/Leaves

- Problem: Join operation difficult to run for concurrent joins/leaves in large networks
- Solution: Use a stabilization protocol that runs periodically to guard against inconsistency
- Each node periodically runs stabilization protocol
  - Check consistency of succ. pointer <basic stabilization>
  - Check consistency of finger pointers <fix_fingers>
  - Check consistency of pred. pointer <check_predecessor>
- Note:
  - Stabilization protocol guarantees to add nodes in a fashion to preserve reachability
  - Incorrect finger pointers may only increase latency, but incorrect successor pointers may cause lookup failure!

Modified Node Join



Fault-tolerance and Replication

- Fault-tolerance:
  - Maintain successor invariant
  - Each node keeps track of r successors
  - If r = O(log(N)), then lookups succeed with high probability despite a failure probability of ½
- Replication:
  - Supports replication by storing each item at some k of these r successor nodes

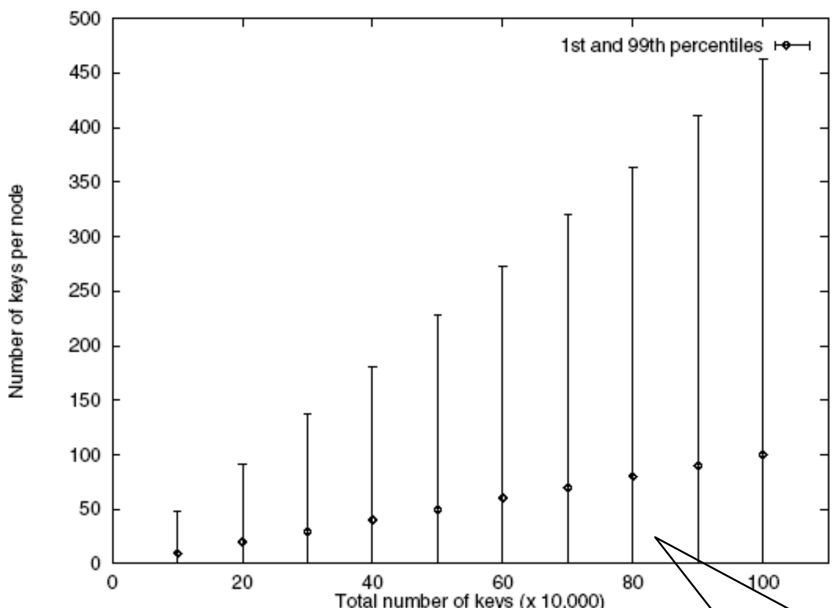
Voluntary Node Departures

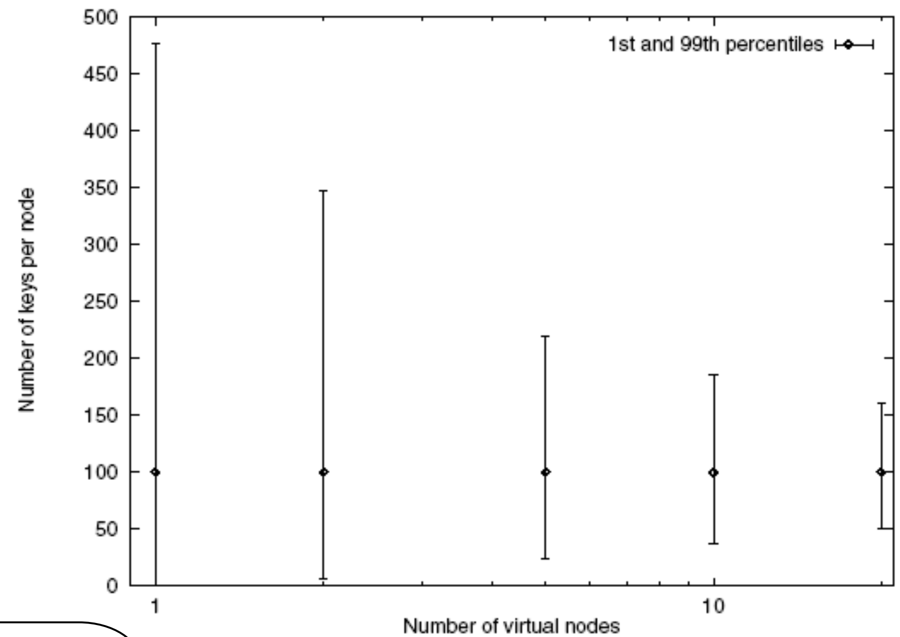
- Can be treated as node failures
- Two possible enhancements
  - Leaving node may transfers all its keys to its successor
  - Leaving node may notify its predecessor and successor about each other so that they can update their links

Simulation Results

- Iterative implementation
- 10,000 nodes
- No. of keys range from 10^5 to 10^6
- Presented results:
  - Load Balance
  - Path Length
  - Lookups during stabilization
- Comparative discussion on DHTs

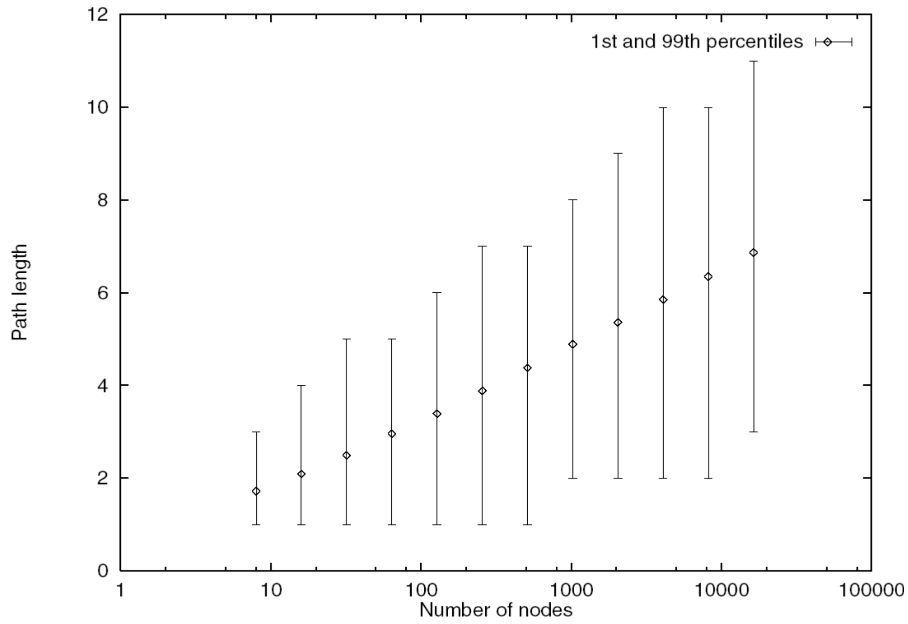
Load Balance



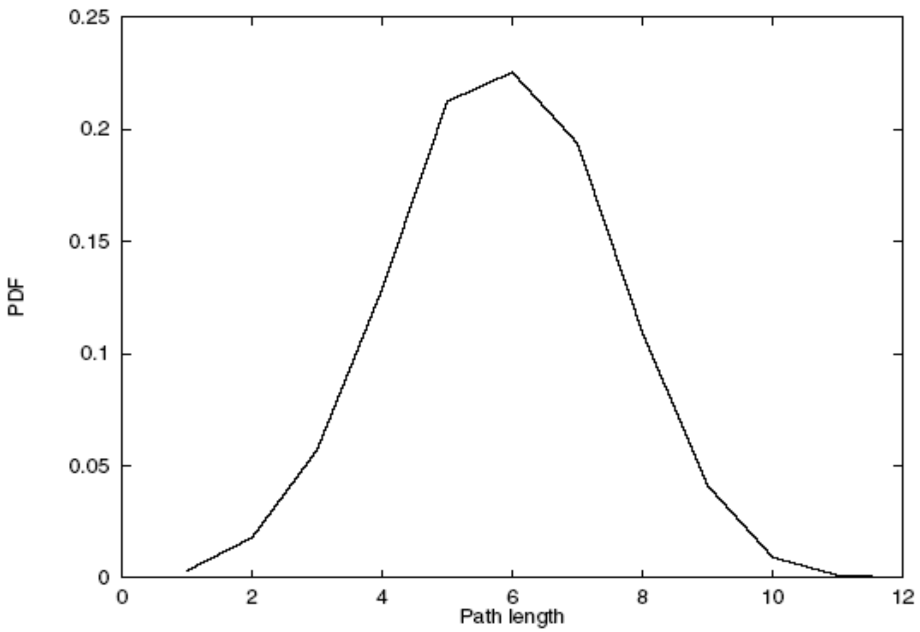


Drastic Variation in Key Allocation: Poor Load Balance

Path Length

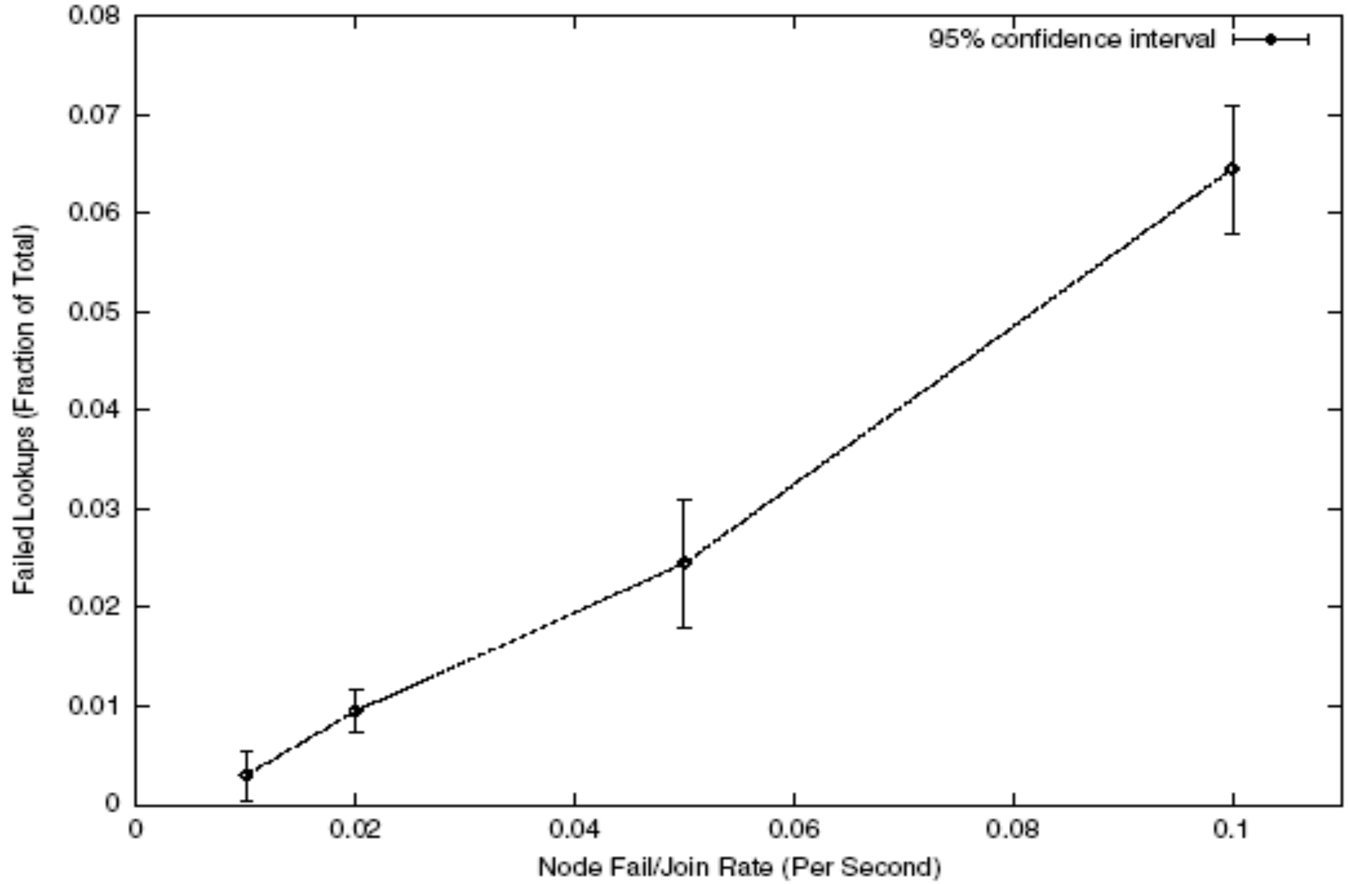


1st and 99th percentiles
Path length
Number of nodes
0
2
4
6
8
10
12
1
10
100
1000
10000
100000



PDF
Path length
0
0.05
0.1
0.15
0.2
0.25
0
2
4
6
8
10
12

Lookups during Stabilization



Comparative Discussion on DHTs

- Comparison metrics: *(degree of flexibility) Gummadi et. al [6]*
  - Static Resilience: Ability to route successfully w/out recovery
  - Path Latency: Average end-to-end latency for a lookup
  - Local Convergence: Property that 2 messages for same location converge at a node near the two sources
- From study, [6] conclude ring-structure performs the best!

property	tree	hypercube	ring	butterfly	xor	hybrid
Neighbor Selection	$n^{\log n/2}$	1	$n^{\log n/2}$	1	$n^{\log n/2}$	$n^{\log n/2}$
Route Selection (optimal paths) Route Selection (non-optimal paths)	1 -	$c_1(\log n)$ -	$c_1(\log n)$ $2c_2(\log n)$	1 -	1 $c_2(\log n)$	1 $c_2(\log n)$
Natural support for sequential neighbors?	no	no	yes	no	no	Default routing: no Fallback routing: yes

Current Status

- Is actively being investigated as project IRIS:
  - Infrastructure for Resilient Internet Systems (http://project-iris.com/)
  - Government funded project active since 2002 ($12M)
  - Goal: "*develop novel decentralized infrastructure based on distributed hash-tables that enable a new generation of large-scale distributed applications*".
- Has been used in:
  - General-purpose DHASH layer for various applications
  - DDNS (Distributed DNS)
  - CFS (Wide-area Co-operative File System for distributed read-only storage)
  - Ivy (peer-to-peer read/write file-system)
  - Internet Indirection Infrastructure (I3)

Still many issues…

- Security considerations: (many possible attacks beyond data integrity)
  - Routing attacks: incorrect lookups/updates/partitions
  - Storage & Retrieval attacks: *denial-of-service/data*
  - Other misc. attacks: inconsistent behavior, overload, etc.

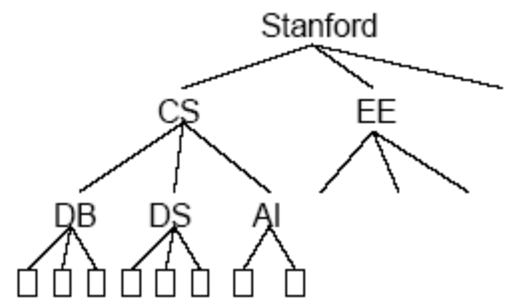
- Performance considerations:
  - No consideration of underlying routing topology (locality properties)
  - No consideration of underlying network traffic/congestion condition
  - Bound on lookups still not good enough for some applications
    - E.g. Failure of DDNS since 8-orders of magnitude worse than conv. DNS

- Application-Specific considerations:
  - Each application requires its own set of access functions in the DHT
  - Lack of sophisticated API for supporting such applications
    - E.g DHASH API is too basic to support sophisticated functionality
  - Support only for DHT as library *vs.* as a service

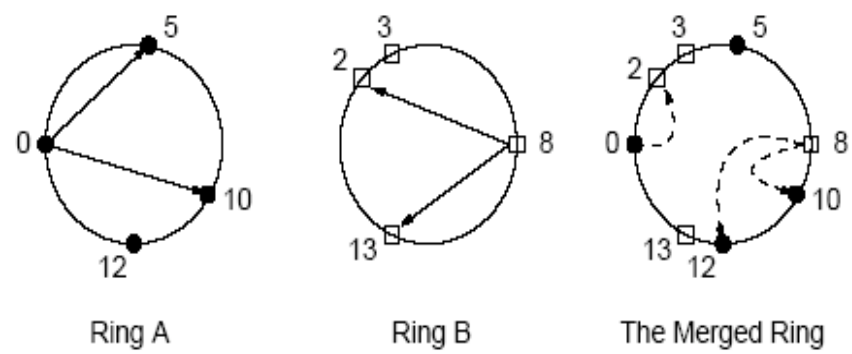
- *And many more…*

Extensions of Chord

- Hierarchical Chord (Crescendo)
  - *"Canon"* generic transformation applied to create hierarchy structure on any flat DHT.
  - Each domain/sub-domain in hierarchy is represented by a ring
  - Larger domains consist of *merged* ring of smaller domains
  - Is this adequate for *locality properties?*



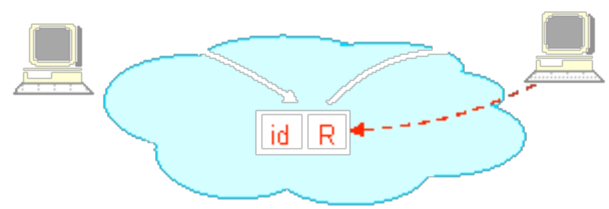
Hierarchy of Domains

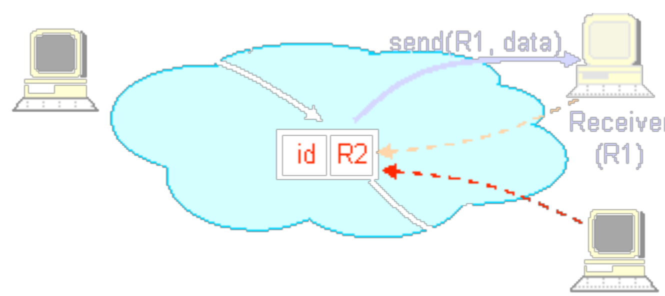


Merging two Chord Rings

Extensions of Chord (Contd..)

- Internet Indirection Infrastructure (*i3*)
  - Combines Chord's lookup with forwarding
  - Receiver inserts trigger (Id, R) into ring
  - Sender sends data to receiver's Id
- Supports:
  - Mobility with location privacy (ROAM)
  - Multicast/ Anycast
  - Service-composition





References

[1] E. Sit and R. Morris, *Security Considerations for Peer-to-Peer Distributed Hash Tables,* In the proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS '02), March, 2002; Cambridge, MA

[2] F. Dabek, E. Brunskill, F. Kaashoek, D. Karger, R. Morris, I. Stoica, and H. Balakrishnan, *Building Peer-to-Peer Systems With Chord, a Distributed Lookup Service*, Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII), May 2001

[3] R. Cox, A. Muthitacharoen, R. Morris, *Serving DNS using a Peer-to-Peer Lookup Service,* In the proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS '02), March, 2002; Cambridge, MA

[4] B. Karp, S. Ratnasamy, S. Rhea, and S. Shenker. Spurring Adoption of DHTs with OpenHash, a Public DHT Service, *In Proceedings of the 3nd International Workshop on Peer-to-Peer Systems (IPTPS '04)*, February 2004

[5] Ganesan, Prasanna; Gummadi, Krishna; Garcia-Molina, Hector. Canon in G Major: Designing DHTs with Hierarchical Structure, Proc. International Conference on Distributed Computing Systems (ICDCS) 2004 .

[6] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, I. Stoica, The Impact of DHT Routing Geometry on Resilience Proximity, *In Proceedings of ACM SIGCOMM 2003*

[7] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, S. Surana, "Internet Indirection Infrastructure," *Proceedings of ACM SIGCOMM*, August, 2002

[8] Host Mobility using an Internet Indirection Infrastructure, *First International Conference on Mobile Systems, Applications, and Services (ACM/USENIX Mobisys)*, May, 2003

Discussion

- Chord could still suffer from potential network partitioning problems
  - How to enforce stricter guarantees on robustness with minimal additional overhead?
- How scalable is the stabilization protocol?
  - Is there a stabilization rate that is suitable for all deployments?
  - How do we balance consistency and network overhead?
- Utilize *caching* on search path for performance?
  - Improve performance for popular DHT lookups *(hay)*
  - Cache coherency problems?
- Performance and Security seem to be at direct odds with each other
  - Can we provide a solution that supports both?
- What is a better approach, DHTs as a library? Or as a service?
- How can we incorporate query models beyond exact-matches?
- What adoption incentives do DHTs need to provide?