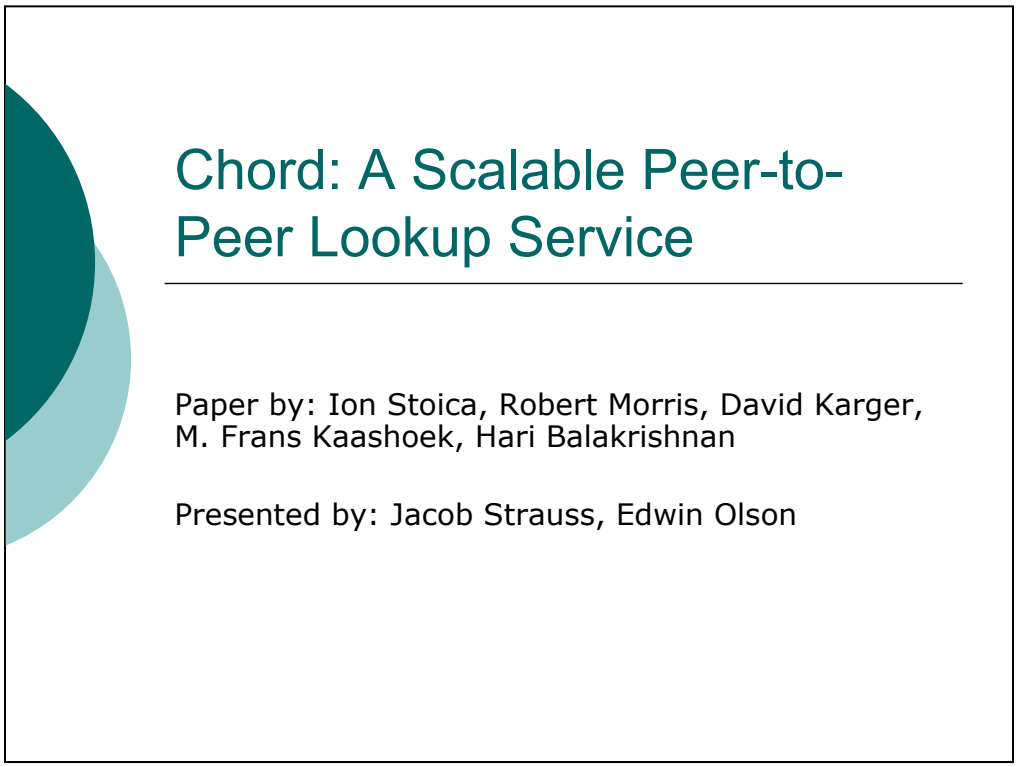
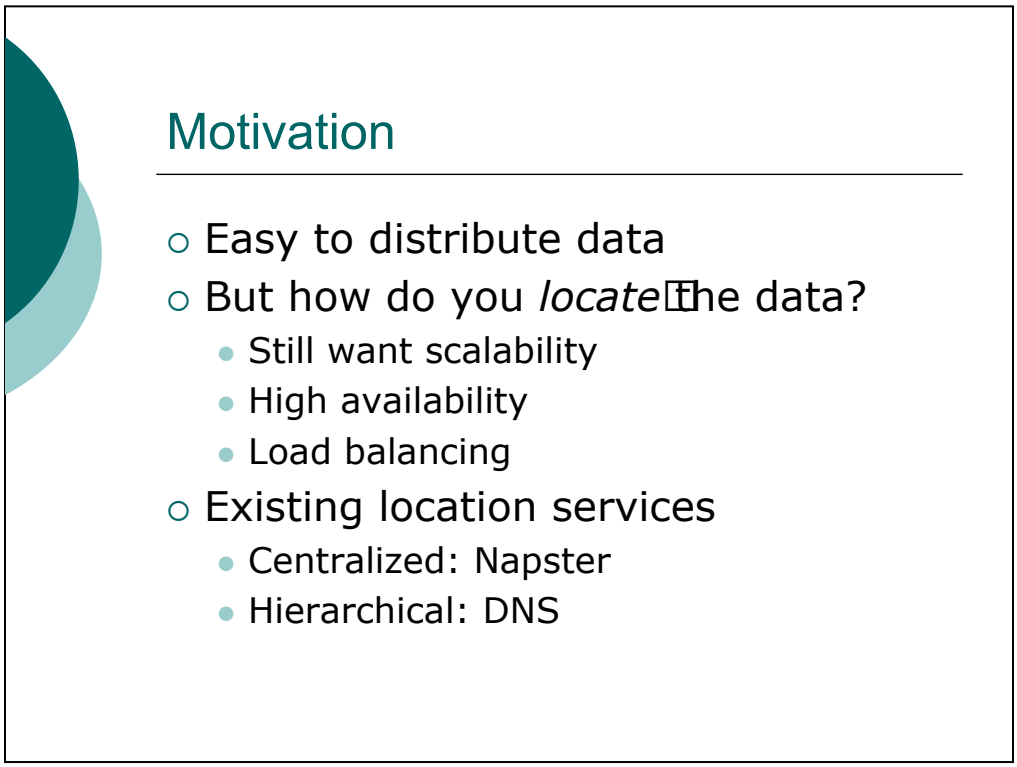
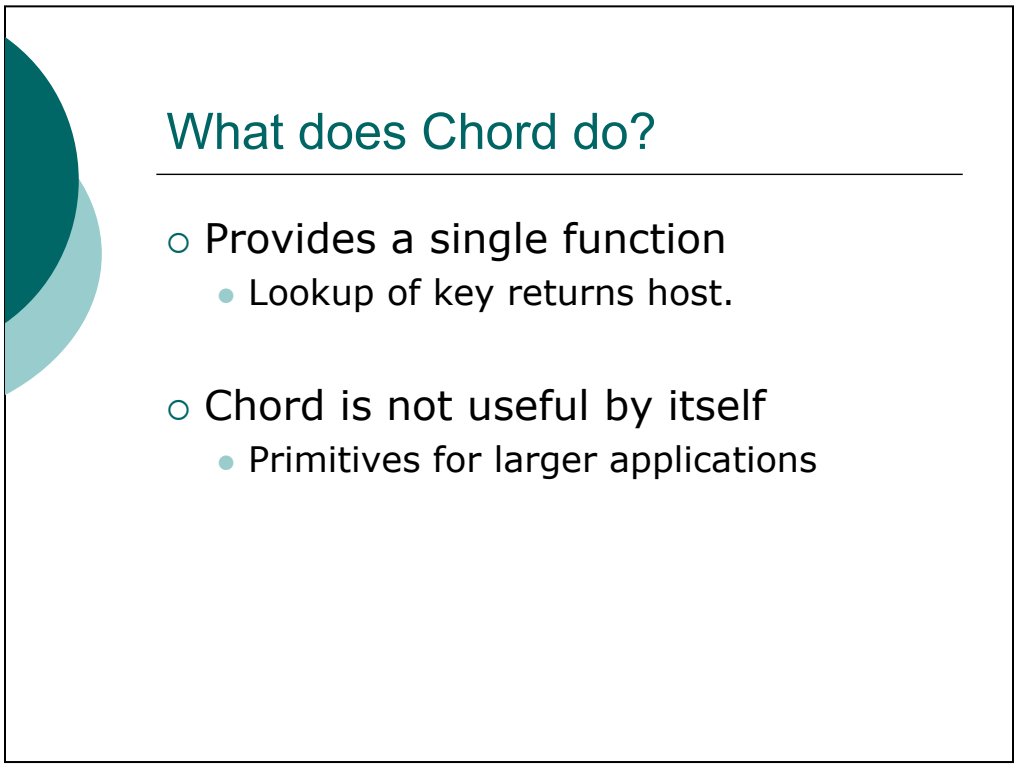


Chord is a set of algorithms developed at MIT and Berkely that serve as a location service for peer-to-peer systems targeted at Internet applications.



E.g., Napster. Napster distributes data on client machines, but relies on a centralized location service. This resulted in a single point of failure, and a need for a massively capable uber-server (or serverfarm).



The key is a number, usually very large, such as 160b.

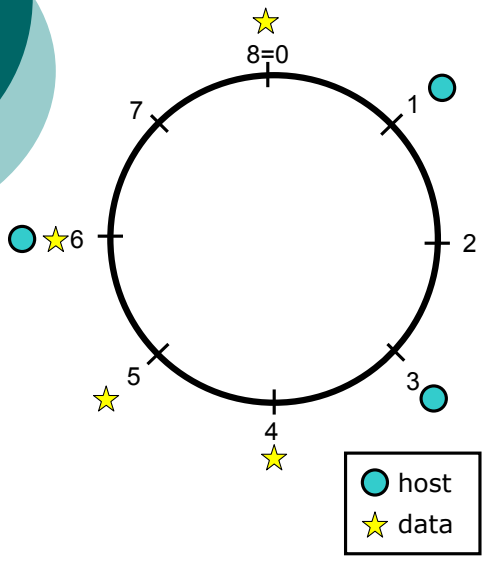
Chord does not specify how keys are generated or what the host does once identified.

Example of Napster-esque application: A user searches for “oopsIdiditAgain.mp3”. Application generates key by SHA1(“oopsIdiditAgain.mp3”) which is fed into Chord. The host returned by chord could provide the mp3 directly or provide a URL to it. Note that Chord only performs a key to host mapping, and is completely indifferent to the semantic meaning of the key.

Related Work

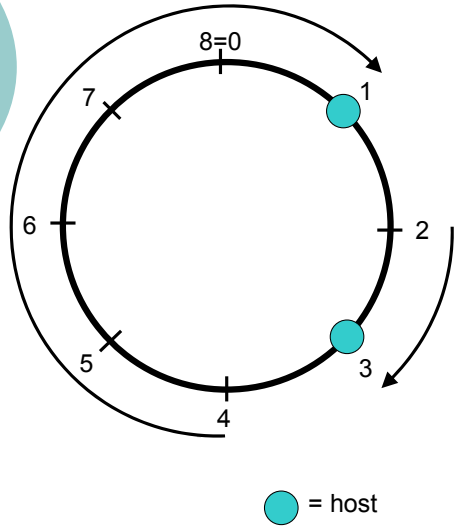
- Distributed file systems (that use lookup services)
  - CFS
  - FreeNet
  - Ocean Store
- Other lookup services
  - Domain Name System (DNS)
  - Pastry (another circular space scheme)
  - CAN (mesh-based)

Details of Chord



- Keyspace
  - A continuous ring of integer values with operations performed mod m
- Both requests and hosts mapped into same space
- Can do arithmetic on key values

Chord Hosts

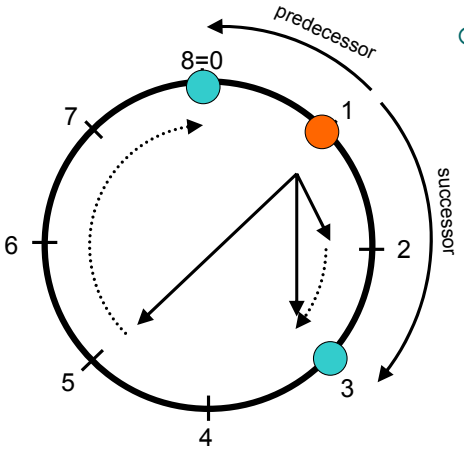


- Host mapped into keyspace: SHA(IP address)
- Each host responsible for some interval of keyspace
- Each key is assigned to host whose key is next clockwise.
- Ex: successor(5)=1

= host

For this example, successor(2..3)=3 and successor(4..1)=1

Host State



- Each host maintains:
  - Host key of next clockwise host, "successor"
  - Host key of previous host, "predecessor"
  - Table of successors of exponentially distant keys, "fingers"

Successor(1)=3

Predecessor(1)=0

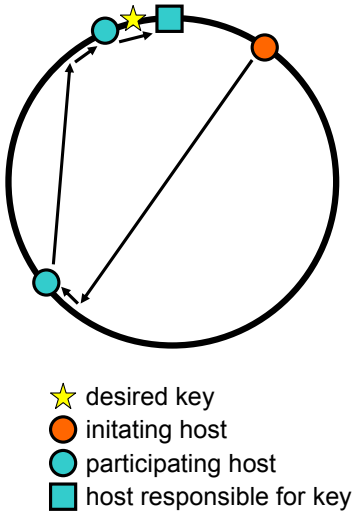
Finger(1,1)=3

Finger(1,2)=3

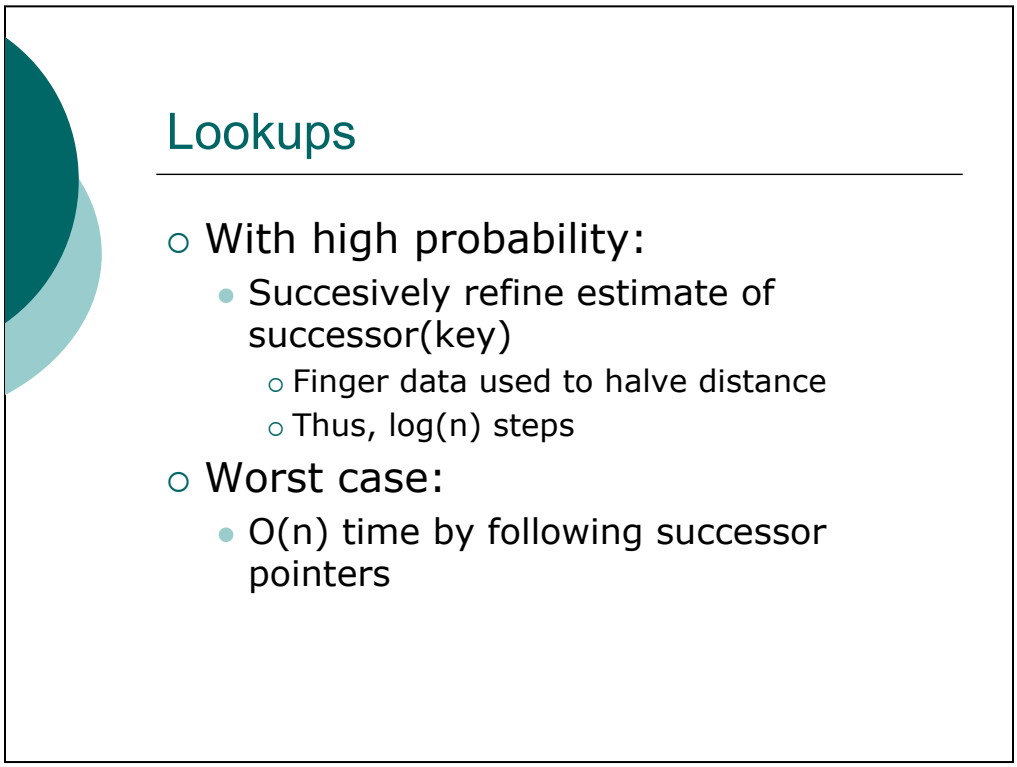
Finger(1,3)=0

For correctness, only successor/predecessor pointers need to be up to date; finger pointers are shortcuts

Lookup Example



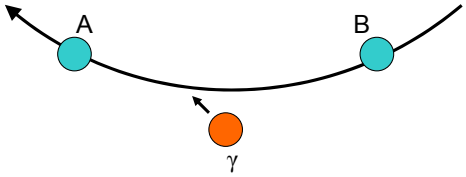
- When searching for a key, each node uses its finger data to forward the request to the closest known host.
- The distance to the key is reduced by at least half at every step.



Assuming successor pointers are correct. Stopping failures considered later.

Time bounds are subject to assumptions discussed later.

Host Joins



- Node γ wants to join the ring
  - Oracle gives it a node p already in the ring
- γ asks p to do a lookup(γ) which yields A.
- γ contacts A. A and γ adjust successor and predecessor pointers
- γ populates its finger table, using A's as hints.
- B eventually fixes successor pointer

Host Join Timebound

- Update finger data for new node:
  - New node has M finger entries. Can be shown that only log N will be different. Each different one takes log N to lookup
    - Thus, total time is $O(\log^2 n)$
- Update other nodes' finger data
  - On average, log n nodes will need to be updated. Each one takes a lookup to update, thus $O(\log^2 n)$
- Transfer keys
  - #keys/N will be relocated
- Total time complexity is $O(\log^2 n)$

Claim: A typical node is likely to have log N unique entries in its finger table.

Proof sketch: Suppose a node has s unique entries. This implies a density of nodes in the keyspace, namely that there is one node every 2^{M-s} positions around the keyspace. Thus there are $2^M/2^{M-s}$ nodes in the system, which shows that $N=2^s$ and therefore that s=log N.

2^m=size of keyspace

Reduces to $O(\log^2 n)$ because of the m entries, only log N of them are likely to have different values. If the node for finger[i] is so far away that it is also the successor of finger[i+1], then no need to lookup finger[i+1].

For example, finger[1] might be equal to finger[2], and n_{new} can check for this, and skip performing a lookup for finger[1].

Log n nodes need to be updated: There are nlog m finger entries. Again, this can be reduced to nlog n because some finger indexes will contain the same node. But there are only n distinct node values, thus, on average, there are log n copies of each.

Node Stopping Failures

- Other nodes' successor/finger pointers are now incorrect
  - Stabilization protocol
  - Multiple successor pointers
- Any data at stopped node becomes unavailable
  - It was the only host responsible for a set of keys
  - Up to application to cope

Stabilization Protocol

- For node i:
  - Contact last known successor and ask who its predecessor p is.
    - Common case: p IS i. All done.
    - If p is a better successor then i.successor=p.
  - Tell p that we've chosen them as our successor.
    - They can then conditionally update their predecessor.
  - Occasionally refresh finger data by performing lookups.

Multiple successor pointers

- Chord especially vulnerable when successor chain is broken
  - Keep pointers to next r successors.
- Theorem: If $r=\log_2 n$ and every node fails with probability ½, then with high probability:
  - Lookup returns the proper value.
  - Lookup still takes O(log n) time.

When the closest successor fails, move immediately to the next successor.

Proof Sketch 1: With each node maintaining r successors, probability that all fail is $2^{-r}=1/n$.

Proof Sketch 2: Half of the time, you'll be able to halve the distance, so time is still O(log n)

Time Bounds

- Chord relies on keys (and nodes) being uniformly distributed around the keyspace.
- But locations in keyspace are deterministic… but they're good enough!
  - Based on "standard hardness" assumptions
  - The pseudo-random distribution of a cryptographic hash is believed to work

Worst Case

- Paper talks about "high probability" cases only
- Worst case scenarios for Chord are very bad
  - Broken successor pointers can make lookups fail… No safety property! Has liveness property.
  - Possibility of ring splitting into multiple, independent rings

Safety property- any result that is returned is correct

Liveness- a result will be returned in a bounded amount of time.

Experimental Results

- Lookups usually only fail when the node who last had the key failed
- Lookup failures scale linearly with node failures
- Average number of keys per node close to theoretical keys/N, but O(keys) variance.
- Failures due to inconsistencies do occur, getting worse with higher rates of joins/fails.

Analysis of key distribution is totally dependent on the way keys are generated, a matter which Chord does not address. For example, if a key was the string prefix of a user-provided search string, then obviously keys would NOT be uniformly distributed.

Load Balancing Reconsidered

- Chord assumes both keys and requests are uniformly distributed.
- Keys might not be uniformly distributed
  - key could be plaintext ASCII
- Requests might not be uniformly distributed
  - key=SHA("Britney Spears")) then some nodes will be disproportionately loaded.
- The higher level application can (and should) deal with this (e.g., caching).

In other words, there are a lot of implicit assumptions behind Chord's performance claims. If these assumptions are satisfied, then Chord can be an excellent tool. However, one must be careful to consider these potential pitfalls.

Applications

- CFS
  - Read only file system
  - Tamper resistant: servers cannot alter data
- CFS uses Chord
  - Datablock = “Dear Mom, …”
  - Key = SHA(DataBlock)
  - Chord is used to specify which server stores datablock for a given key.

Atomic Object

- Can Chord be used as a basis for an Atomic object?
  - Chord, by itself, is read only; there's no update!
    - Imposing the orderings on reads and writes is up to the application.
  - Even if the application does it "right", Chord's lack of a safety property makes implementation of an atomic object difficult.

Further Reading

- Stoica, Morris, et. al. Chord: A scalable peer-to-peer lookup service for Internet Applications.
- Stoica, Morris, et. al. Chord: A scalable peer-to-peer lookup service for Internet Applications. Tech Report. TR-819.
- Dabek, Kaashoek, et. al. Wide area cooperative storage with CFS. 18th ACM SOSP.
- Su, Postel. Domain Name Serivce RFC 819
- Clarke, Sandberg, et. al. Freenet: A distributed anonymous information storage and retrieval system. ICSI WODIAU.

Chord doesn't address updates, so ensuring read/write ordering is up to the application. Replicas, of course, complicate this. Chord further complicates due to the possibility of failures, so the application must be robust to them as well… An atomic object build with Chord would satisfy termination with high probability.