Distributed Hash Tables
What is a DHT?

- **Hash Table**
  - data structure that maps “keys” to “values”
  - essential building block in software systems

- **Distributed Hash Table (DHT)**
  - similar, but spread across many hosts

- **Interface**
  - insert(key, value)
  - lookup(key)
How do DHTs work?

Every DHT node supports a single operation:

- Given *key* as input; route messages to node holding *key*
- DHTs are *content-addressable*
DHT: basic idea
DHT: basic idea

Neighboring nodes are “connected” at the application-level
DHT: basic idea

Operation: take key as input; route messages to node holding key
DHT: basic idea

Operation: take \textit{key} as input; route messages to node holding \textit{key}
DHT: basic idea

Operation: take *key* as input; route messages to node holding *key*
DHT: basic idea

Operation: take *key* as input; route messages to node holding *key*
DHT: basic idea

Operation: take *key* as input; route messages to node holding *key*
• For what settings do DHTs make sense?
• Why would you want DHTs?
Fundamental Design Idea I

- Consistent Hashing
  - Map keys \textit{and} nodes to an \textit{identifier} space; implicit assignment of responsibility

Mapping performed using hash functions (e.g., SHA-1)

- What is the advantage of consistent hashing?
Fundamental Design Idea II

• Prefix / Hypercube routing
Nodes are randomly chosen points on a clock-wise ring of values.

Each node stores the id space (values) between itself and its predecessor.
Chord Topology and Route Selection

• Neighbor selection: \( i^{th} \) neighbor at \( 2^i \) distance

• Route selection: pick neighbor closest to destination
How to design a DHT?

- **State Assignment:**
  - what “(key, value) tables” does a node store?

- **Network Topology:**
  - how does a node select its neighbors?

- **Routing Algorithm:**
  - which neighbor to pick while routing to a destination?

- **Various DHT algorithms make different choices**
  - CAN, Chord, Pastry, Tapestry, Plaxton, Viceroy, Kademlia, Skipnet, Symphony, Koorde, Apocrypha, Land, ORDI ...
How do you characterize the performance of DHTs?
Issues

- How do you improve the performance of DHTs?
Issues

- What are the fault tolerance/correctness issues?
  - how do you improve fault-tolerance or reliability?
Issues

- What are the security issues?
- how do you improve security?
Issues

- What are the load balance issues?
- how do you improve load balance?
Dynamo

- Real DHT (1-hop) used inside datacenters
- E.g., shopping cart at Amazon
- More available than Spanner etc.
- Less consistent than Spanner
- Influential — inspired Cassandra
Context

• SLA: 99.9th delay latency < 300ms
• constant failures
• always writeable
Quorums

- Sloppy quorum: first N reachable nodes after the home node on a DHT
- Quorum rule: \( R + W > N \)
  - allows you to optimize for the common case
Eventual Consistency

- accept writes at any replica
- allow divergent replicas
- allow reds to see stale or conflicting data
- resolve multiple versions when failures go away
  - latest version if no conflicting updates
  - if conflicts, reader must merge and then write
More Details

- Coordinator: successor of key on a ring
- Coordinator forwards ops to N other nodes on the ring
- Each operation is tagged with the coordinator timestamp
- Values have an associated “vector clock” of coordinator timestamps
- Gets return multiple values along with the vector clocks of values
- Client resolves conflicts and stores the resolved value