A SCALABLE CONTENT-ADDRESSABLE NETWORK

Mema Roussopoulou

*Slides based in part on Sylvia Ratnasamy's talk slides.

What kind of paper is this?

- A New big idea?
- A Measurement paper?
- An Experiences/Lessons Learnt paper?
- A System Description?
- A Performance Study?
- A Refute-Conventional-Wisdom paper?
- A Survey paper?

Back to Basics – CS 101 ☺

- What is a hash table?
- What is it good for?
  - Wise systems folk say: *"A hash table and a level of indirection"* is all you need it to solve a problem in operating systems!!
  - Helps keep track of state in the system
    - Process tables
    - Page tables
    - Etc.

New Big Idea!

- (Remember – this is Sigcomm 2001)
- Create a big distributed, **Internet-scale** Hash Table
  - Could prove useful for distributed systems
- Distributed apps that might use this?

- So how DO we build a LARGE distributed indexing system?

Ideas

- Do **not** impose a rigid, hierarchical naming structure
  - Use uniform hash function
- D-dimensional Cartesian coordinate space on d-torus
- Coordinate space partitioned dynamically across nodes
- Each node maintains its own "zone" within the space

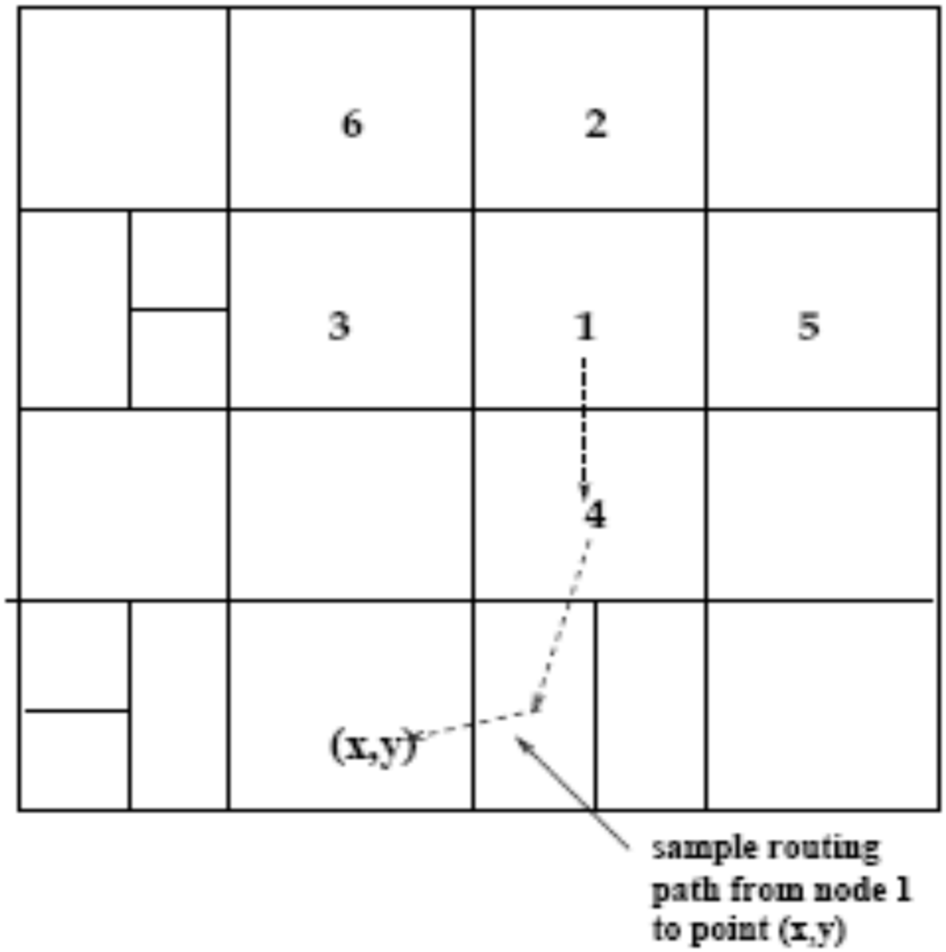
Hash Table Operations

- Lookup (key) → (key, value) pair
- Insert (key, value) pair
- Delete (key, value) pair

Lookup = Routing in a CAN

- Follow straight line path through the Cartesian space from source to destination coordinates.
- To find destination coordinates, hash key to a point in the space
- In d-D space, average routing path length is $(d/4)(n^{1/d})$ hops and each node has 2d neighbors.

Lookup = Routing in a CAN



What state does a node maintain?

What state does a node maintain?

- Its zone boundaries
- Zone boundaries of its neighbors
- IP address of its neighbors
- Possible zone boundaries of neighbors' neighbors

- What determines how much state a node maintains?

Inserting an index entry

- Insert **(K1, V1)** pair by hashing **K1** onto point in coordinate space
- Route **"Store (K1, V1)"** request to that point
- Store at node that owns the zone where point lies

Deleting an Index Entry

- Same as insertion

CAN Construction

- What happens at a high level when a node joins the CAN?

Node Joins

- 1) Node picks a random point P in coordinate space
- 2) Finds IP address of a node already in CAN → sends it **JOIN(P)** request
- 3) Request routed to node **O** with zone containing P
- 4) Node **O** splits. New node takes half with P
- 5) **O's** old neighbors notified/updated

Node Departures

- Gracefully: zone handover to neighbor with smallest zone.
- Ungracefully: all neighbors of the failed node execute a takeover algorithm so that the zone merges with the smallest neighboring zone.
  - How do we detect a node has failed? (next slide)
- Departures → imbalance in zone loads
  - Background zone reassignment algorithm to make more uniform

Soft State

- A very well-known mechanism in distributed systems – what is it?
- When is it used in CAN?
  - Periodic keepalive messages
    - my zone coordinates
    - my neighbors' zone coordinates
    - my neighbors' IP addresses

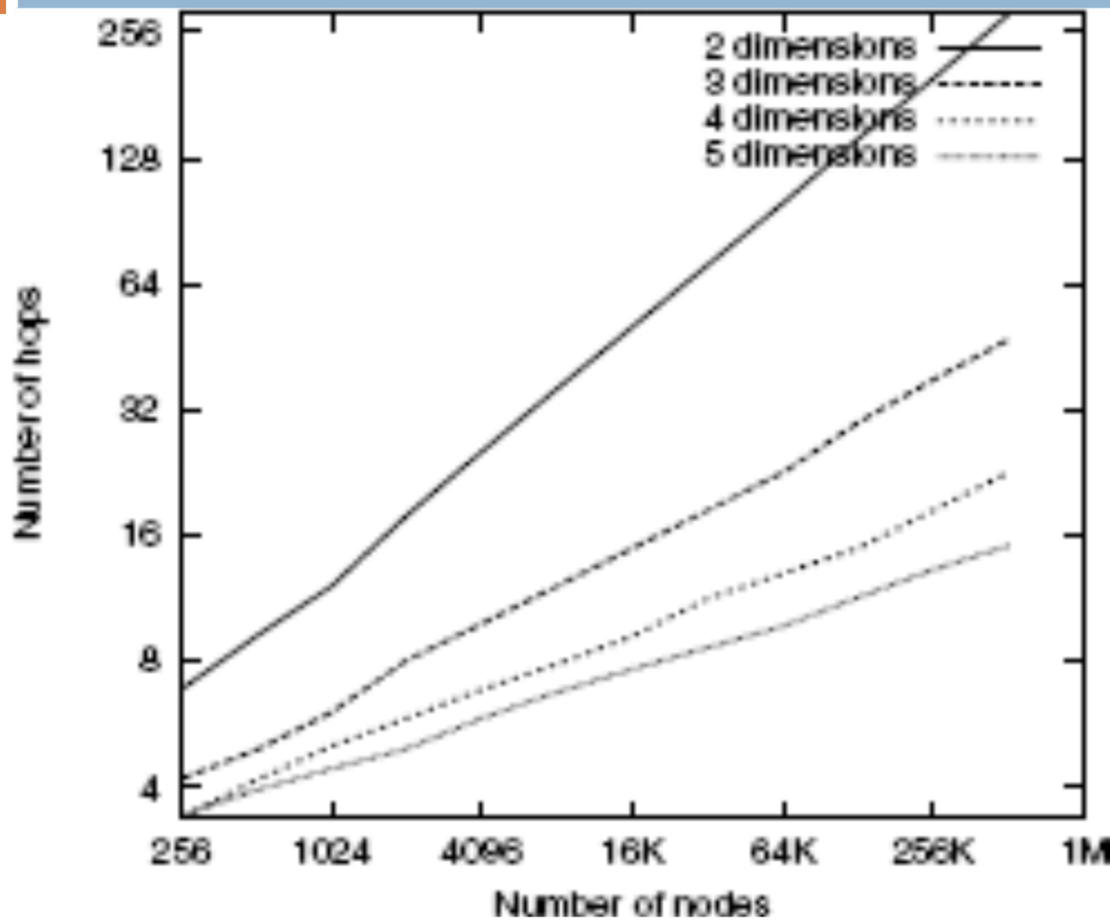
Theoretical performance

- O(d) state maintained per node
- $O(d(n^{1/d}))$ path length between any two nodes
  - Avg lookup latency = (avg CAN path length) * (avg IP latency of a CAN hop)
- Can we do better?
- Yes, lots of design improvements!

It's all about the Tradeoffs

- Systems design is all about tradeoffs
  - Cannot win everywhere
- What do the proposed design improvements trade off?
- For each improvement, ask
  - What do we gain?
  - What do we lose?

Multi-dimensioned coordinate spaces

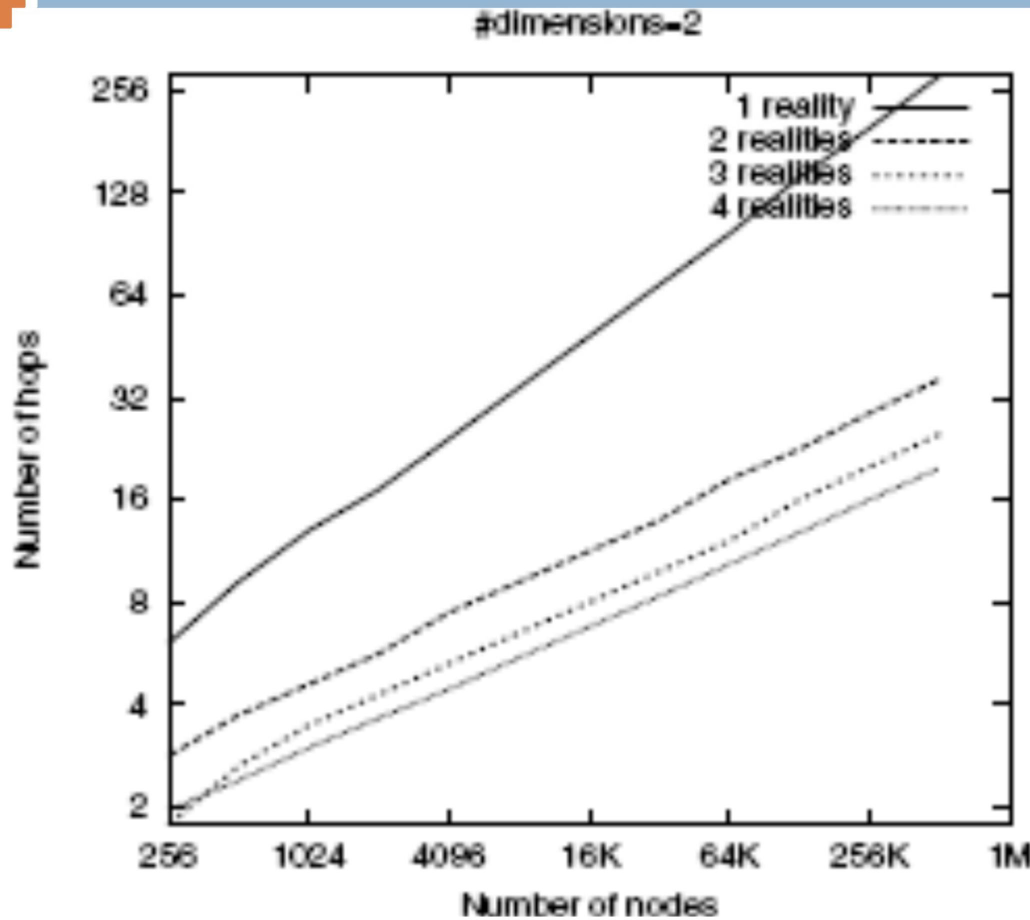


- Path length scales $O(d(n^{1/d}))$
- Per node state increases
- More fault-tolerance

Multiple Realities

- Maintain multiple, independent coordinate spaces (**realities**)
- Every node has a different zone in every reality and a different set of neighbors.
- Node routes to neighbor who is (across all realities) closest to the destination.

Multiple Realities



- Data replication => data availability (fault-tolerance)
- Routing to point P translates to routing to P on *every* reality
- Increased per-node-state

Better CAN routing metrics

- Each node measures net-level RTT to each neighbor
- Choose neighbor with max progress/RTT

Number of dimensions	Non-*RTT* weighted routing (ms)	*RTT* weighted routing (ms)
2	116.8	88.3
3	116.7	76.1
4	115.8	71.2
5	115.4	70.9

Overloading coordinate zones

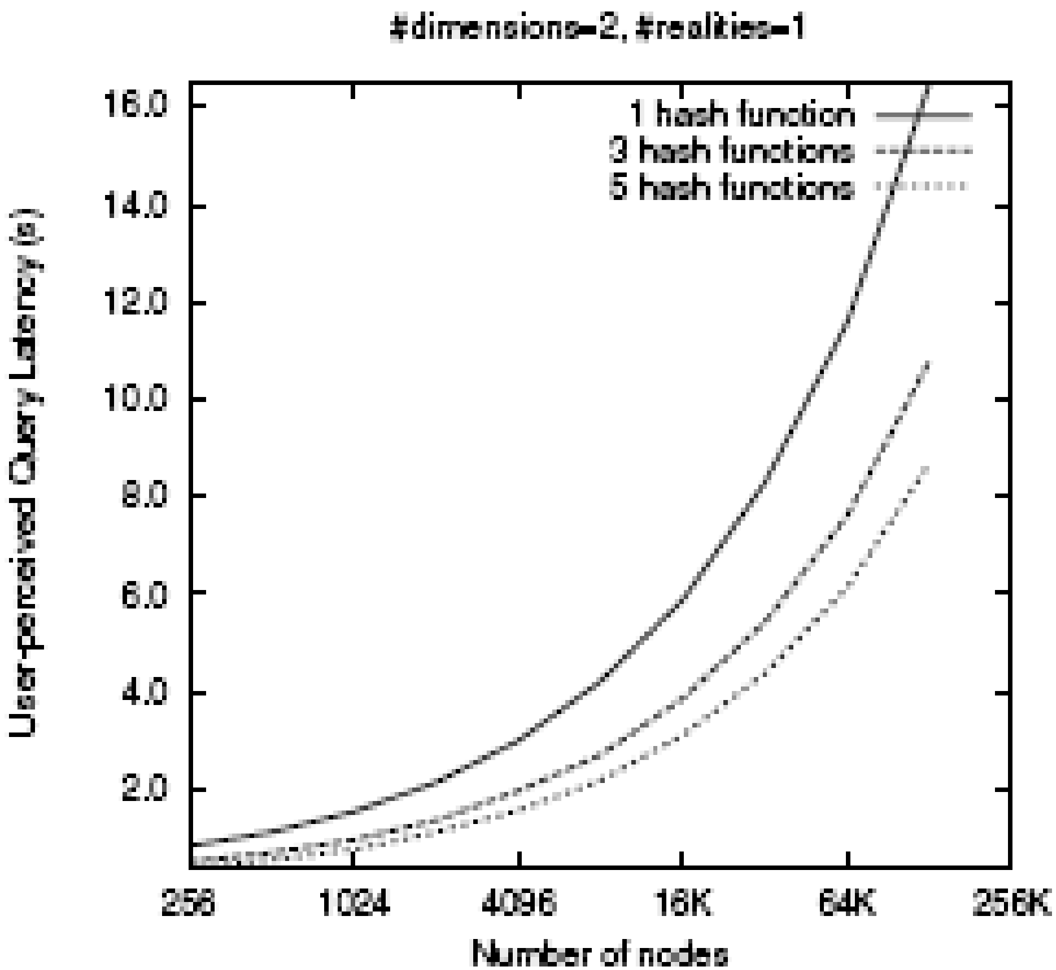
- Multiple peers (up to MAXPEERS) share the same zone.
- Increased state : all peers in same zone but only one peer (the RTT-closest) from each neighbor zone.
- The index entries of a zone may be either partitioned or replicated across the peer nodes.

Overloading coordinate zones

- Reduced path length
  - It's like we have fewer nodes in the system
- Reduced per-hop latency
  - Can choose from a lot of possible neighbor peers
- Improved fault-tolerance
- BUT more complexity

- Note Table 2: what is the number of dimension here?

Multiple hash functions

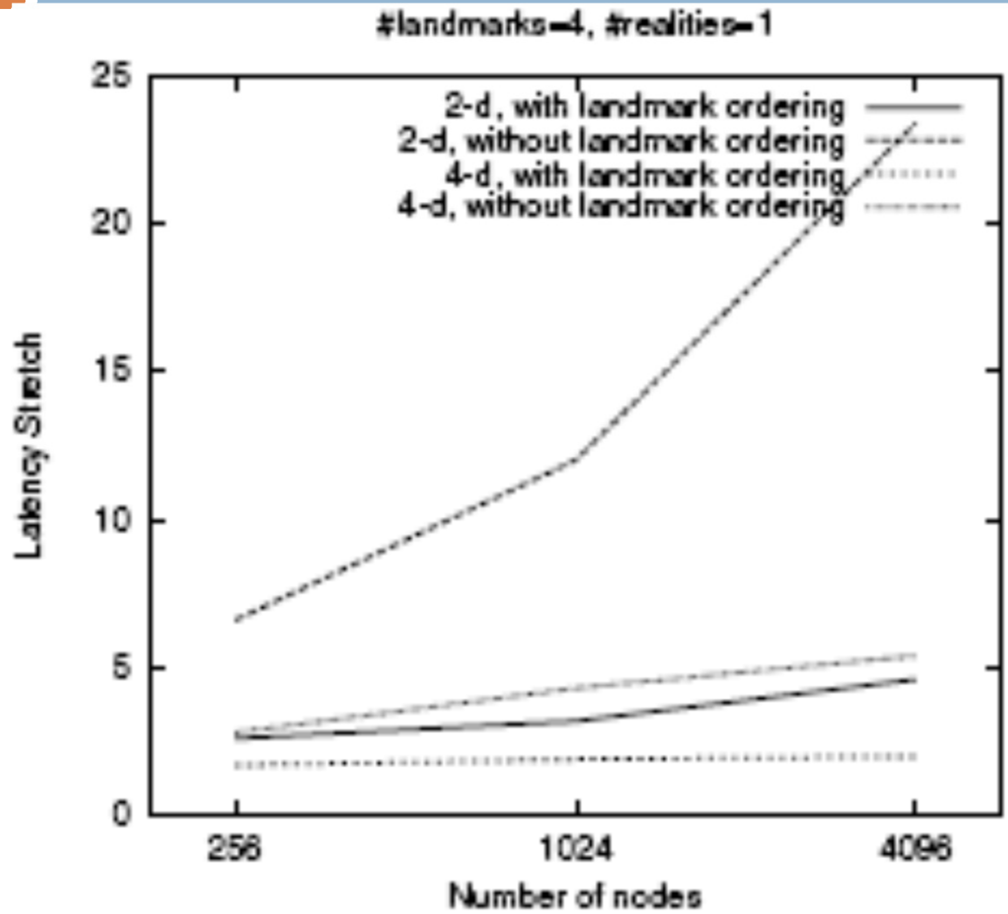


- Assign same key to many points in space with the use of k different hash functions
- A query can be sent towards the closest node or all k directions.

Topologically-sensitive construction

- There are m landmarks (well-known set of machines, e.g. the DNS root name servers).
- Each node orders the landmarks in order of increasing RTT to them.
- Coordinate space is partitioned into m! portions (one for each landmark ordering)
- Nodes now join at a random point IN the corresponding portion of space.

Topologically-sensitive construction



- Improves the path latency.
- Coordinate space is no longer uniformly populated =>Background load balancing techniques.

On Topologically-sensitive construction

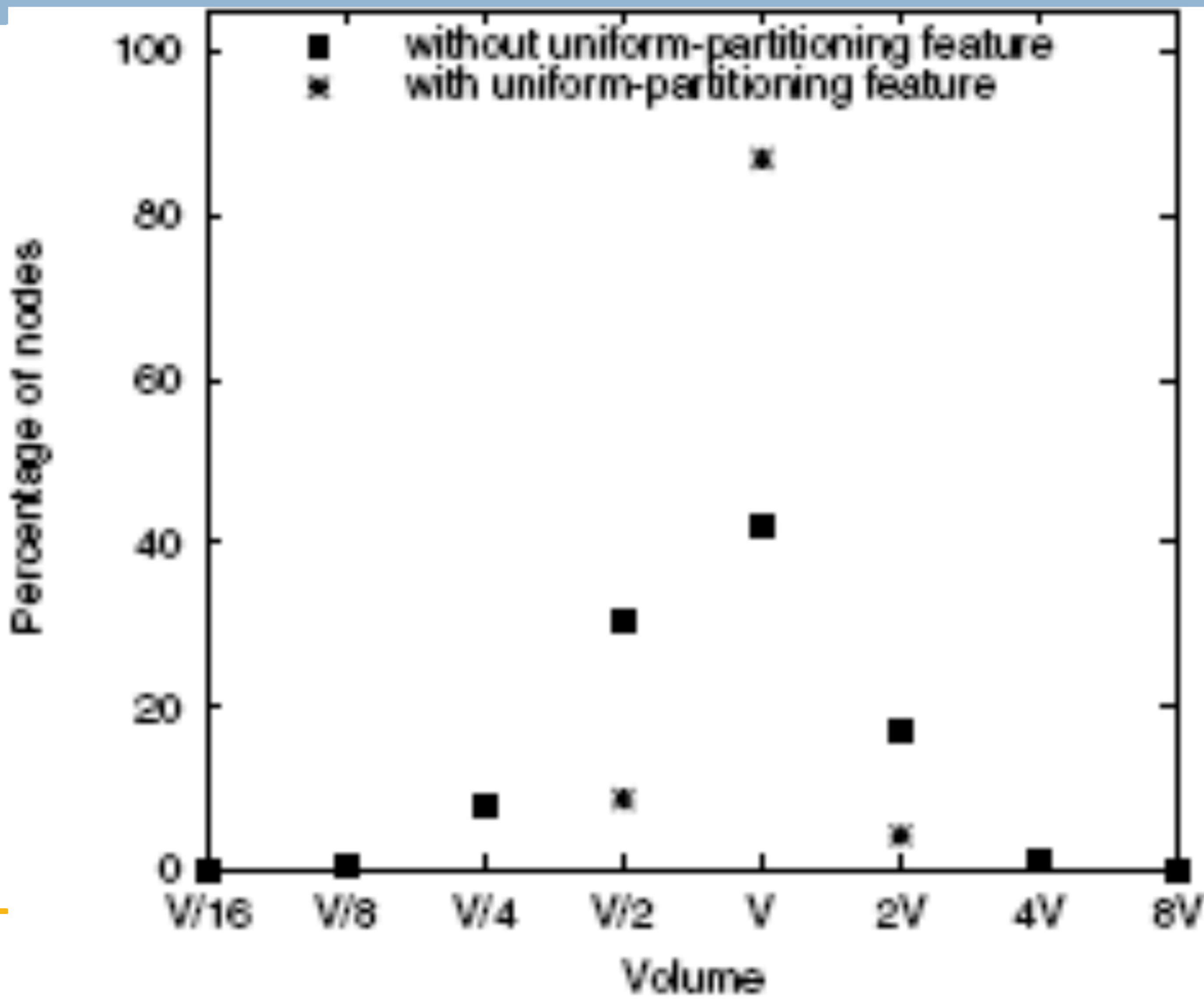
- Landmarks chosen 5 hops away from each other -- Agree?
- Uneven distribution of zones -- what to do?
- How would you continue from here?

More Uniform Partitioning

- On a JOIN request, instead of splitting zone
  - Node checks neighbors' zone sizes
  - Forwards request to neighbor with largest zone

- A uniform hash function guarantees that volume of a node's zone is indicative of the size of the (key,value) database the node will have to store
- So uniform partitioning helps balance the load
  - Is this correct? (what about hot spots?)

More Uniform Partitioning



Caching and Replication

- Caching: huge technique in distributed systems and for the Web
  - Whole careers based on caching!
- Node maintains a cache of the data keys it recently accessed. More requests = higher availability
  - How long do we cache something?
- Replication: node that is overwhelmed by requests for a particular data key replicates key at each of its neighbors

Design Review

Parameter	"bare bones" CAN	"knobs on full" CAN
d	2	10
r	1	1
p	0	4
k	1	1
RTT weighted routing metric	OFF	ON
Uniform partitioning	OFF	ON
Landmark ordering	OFF	OFF

Metric	"bare bones" CAN	"knobs on full CAN"
path length	198.0	5.0
# neighbors	4.57	27.1
# peers	0	2.95
IP latency	115.9ms	82.4ms
CAN path latency	23,008ms	135.29ms

Can you think of more experiments?