CAN & pSearch

CAN Structure

- DHT mapping keys in to a *d*-dimensional cartesian space
- Nodes represent a contiguous partition of the space
- Nodes know their neighbors

CAN Routing

- Route by forwarding message to neighbor node closest to the key
  - Heuristics to modify this, including routing to neighbor with max progress to RTT time
- Message is routed along a line from the entry point to the key
  - Not exactly

CAN Bootstrap

- DNS entry with one or more existing CAN nodes
  - Basic idea: can.cs.umd.edu would resolve to a set of CAN nodes
  - When a new node joins, it looks up the CAN address and joins from that entry point

CAN Join

- Randomly choose a point *P*
- Look up *P* in CAN
- Split the node where *P* lies
- Old node gives the new node neighbor information (O(*d*) neighbors)
- new node contacts neighbors

CAN Node Departure

- Start timer when neighbor node failure is detected
- Send TAKEOVER message with zone volume
- When TAKEOVER received:
  - If message has lower reported zone volume, cancel own TAKEOVER
  - Otherwise, respond with own TAKEOVER

CAN Node Departure Questions

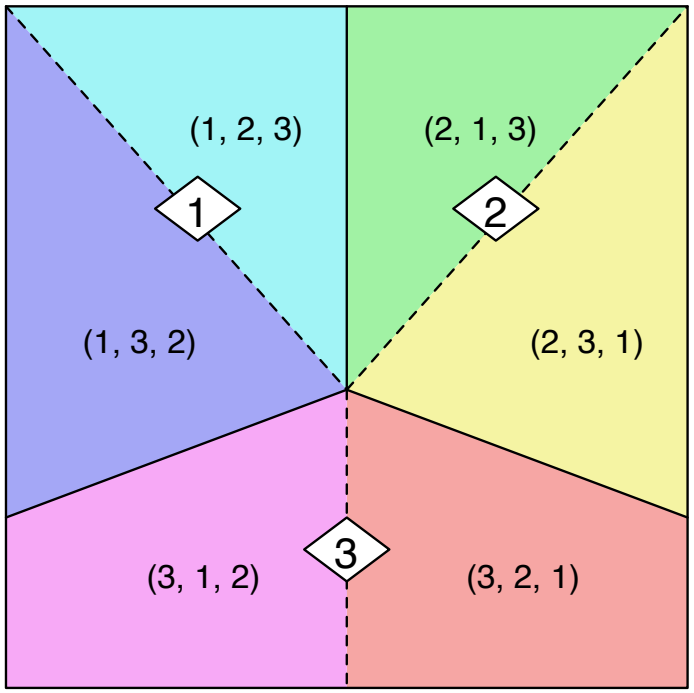
- What if zone volume is the same?
  - Can perturb, to reduce likelihood
  - But all conflicts need to be resolved
- How long does a given node wait before it stops listening for more TAKEOVER messages?

CAN Design Improvements

- Multiple dimensions vs. Multiple realities
  - Dimensions give better performance
  - Realities give better reliability
- Overloading Coordinate Zones
  - Multiple nodes per zone
  - Increases performance at cost of complicated implementation

CAN Design Improvements

- Topologically-sensitive construction
  - Make CAN topology reflect network topology
- More Uniform Partitioning
  - Hack to split busy nodes



pSearch

- Centralized information retrieval systems are
  - not scalable (Google covers a fraction of the content)
  - vulnerable: kill the head and the body dies

pSearch

- Idea: build information retrieval on P2P
- But
  - DHTs reliably index keys
  - keys have nothing to do with content
- They have a couple of solutions ...

pVSM

- For each document
  - Store ($h(t_i)$, `index`) for each of m highest-weighted terms from VSM result
- When searching
  - Look up index for $h(t)$ for each term of search
- Could probably work for any DHT

pLSI

- Basic idea:
  - SVD on matrix from VSM vectors maps term vectors on to semantic vectors
  - Map semantic vectors in to CAN space

pLSI

- Problem
  - Semantic vectors are all vectors on the unit sphere, not fully utilizing the space
- Solution
  - Transform points on sphere in l dimensions in to a (*l*-1) dimension space
  - Sort of like a map projection from the Earth's surface??

pLSI

- Problem
  - Uneven distribution of semantic vectors over the sphere
- Solution
  - New nodes join in zones based on things present in the index
  - Idea: new nodes join where things are probabilistically crowded

pLSI

- Problem
  - Semantic vectors have 100-300 dimensions
  - Way too much for CAN
- Solution
  - Shorten the vector, since important things are closer to the top of the vector
  - Multi-plane scheme: multiple sections of the vector space mapped in to one CAN space?

pLSI

- Problem
  - Global information needed to work
- Solution
  - Slightly old information is OK
  - Statistics updates distributed infrequently
  - SVD-updating, or an approximation

Questions

- CAN Bootstrap

  They look up other CAN nodes through a DNS hack

- Do hash functions guarantee even distribution

  Hopefully, but this is not guaranteed

Questions

- What about this "Every node in the system sends an immediate update message..."

  ??? -- I hope that's an error. Maybe they meant the new and old nodes immediately send an update.

- Are (key, value) pairs lost in other P2P systems if a node departs unexpectedly?

  Yes. At their core, these systems merely map keyspace to nodes.

Questions

- What happens in a multiple-reality search when the key is found?

  Probably nothing -- just ignore further results. Otherwise things get complicated.

- What do they mean by join in "Advanced searches" in pSpace

  Database join. Example: get the set of all papers cited by a given list of papers

Questions

- What happens in CAN when the space is partitioned?

  Basically, it's broken. Maybe the node could re-bootstrap? Some things help prevent this:

  - Multi-dimensions reduce the probability of partition
  - Multi-realities let partitioned nodes in reality *a* fix based on info from reality *b*