Democratizing content publication with Coral

Michael J. Freedman, Eric Freudenthal, David Mazi`eres
New York University

presented by Nikola Borisov

Overview

- Introduction
- System Overview
- Application Layer Componenets
- DSHT
- Evaluation
- Related and Future Work

CoralCDN

- Peer-to-Peer
- Content Distribution System (CDN)
- Distributes Load
- Content publisher just uses *.nyud.net:8090
- Uses DHT, DNS, Cache Servers

CoralCDN Usage

- Publishers: http://www.x.com.nyud.net:8090/y.jpg
- Third-parties can Coralize the urls by adding.
- Users can choose to use Coralized Url's.
- The above three are independent

System Overview

1 Browser Request www.x.com.nyud.net:8090/

2 Resolver contacts Coral DNS server

3 Coral DNS probes the client RTT and last few hops

4 Coral DNS asks the DHT for nearby proxies

5 DHT returns list of nearby proxies and NS

6 Passing them to the client

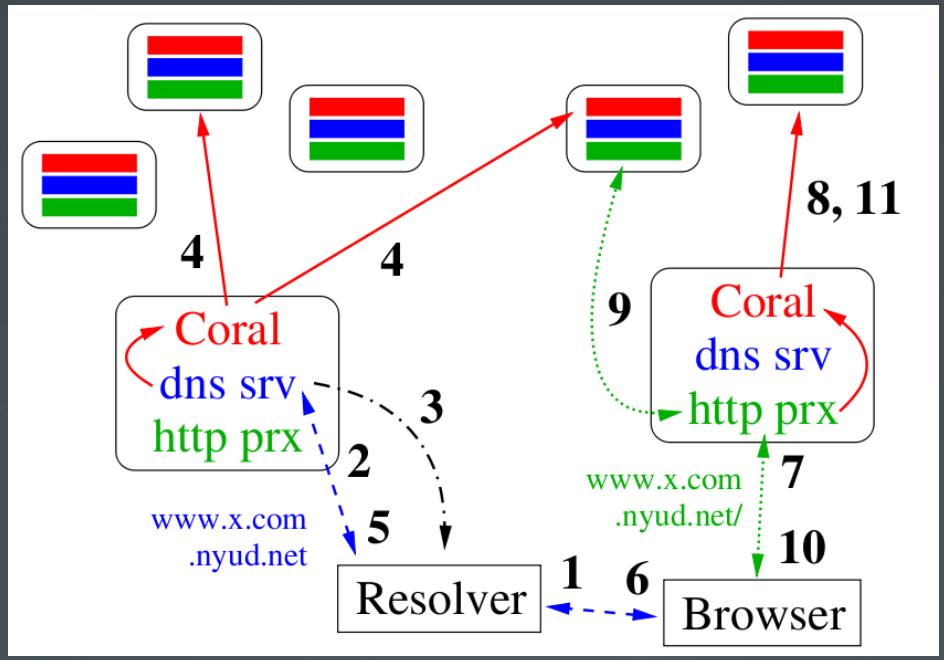
7 Browser sends request to the proxy server. If cached the result is returned right away.

8 If not the proxy asks the DHT about copy in the system.

9 If no copy is found then contact www.x.com

10 Cache and Return

11 The proxy registers itself in the DHT as a holder.



Coral Indexing Abstraction

- Distributed Sloppy Hash Table
  - store multiple values under the same key
  - hot spot preventions
  - each node belongs to several DSHT
  - weaker consistency then usuall DHT
- Interface:
  - put(key, val, ttl, [levels])
  - get(key, [levels]
  - nodes(level, count, [target], [service])
  - levels()

Application-Layer Components

- Coral DNS Server
- Coral HTTP Proxy Server

Coral DNS Server

- Local Resolver gets DNS server in his own cluster
- Assuming Resovers are near Clients
- Measure RTT to Resolver and last 5 hops to find a cluster for
- Return CoralProxy servers near the client; short TTL
- Return ND near the client; long TTL
- L2.L1.L0.nyucdn.net

Coral HTTP Proxy Server

- Goals: low latency, high throughput, min load on original servers
- Fetch from other CoralProxy when possible
- While fetching register in DSHT as replica; ttl 20sec
- If registered in DSHT as replica don't evict.
- Adaptive TTL (not implemented)

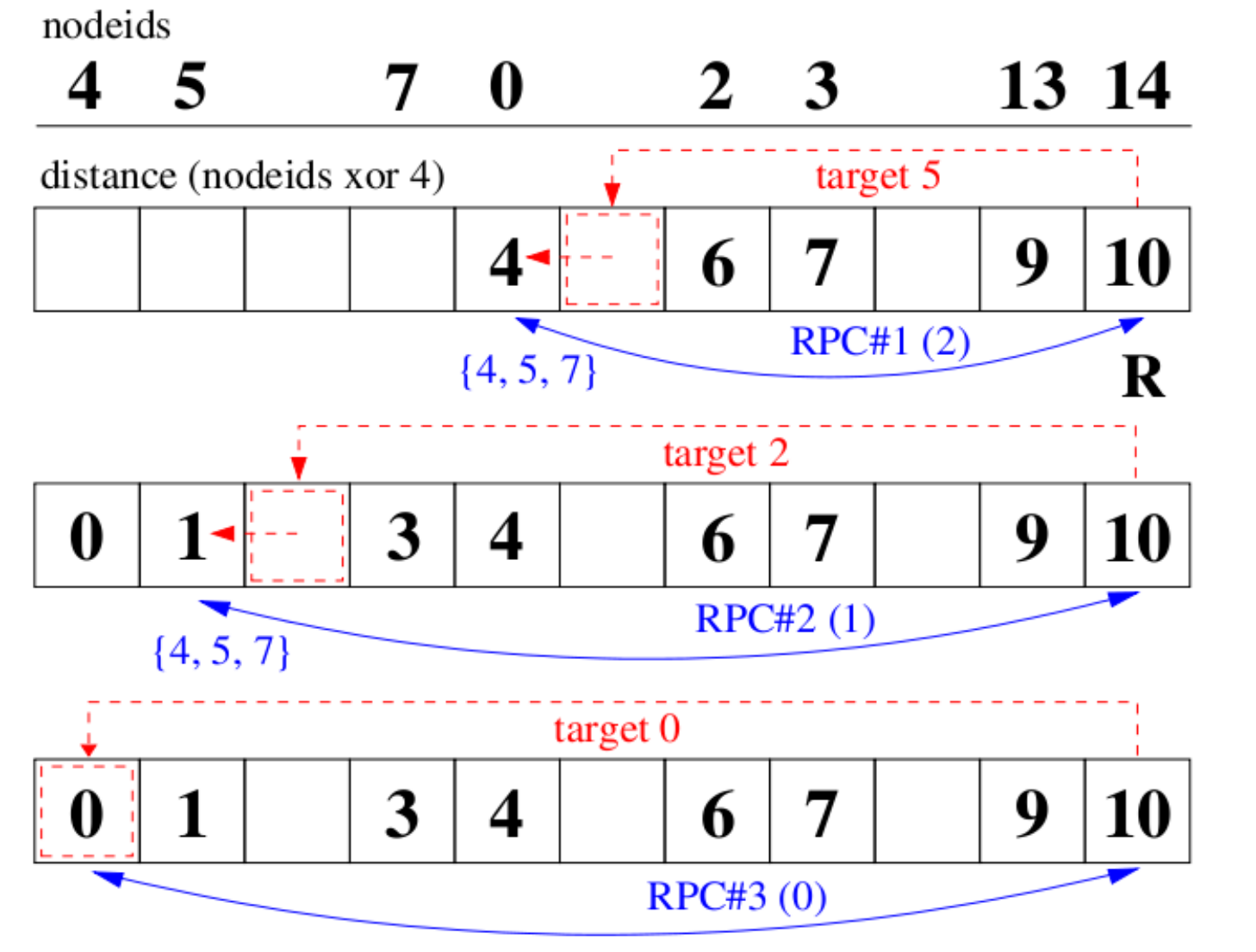
Coral: A Hierarchical Indexing System

- Key-Based Routing Layer
- Sloppy Storage
- Hierarchical Operation
- Joining and Managing Clusters

Key-Based Routing

- Nodes ID: 160 bit SHA-1 hash of IP
- keys of objects close to Node ID's
- Routing table based on Kademlia, using XOR as int
- Logarithmic size routing table.
- On each step get twice as close.
- *tree saturation* problem

Example


nodeids
4 5 7 0 2 3 13 14
distance (nodeids xor 4)
target 5
4 6 7 9 10
RPC#1 (2)
{4, 5, 7}
R
target 2
0 1 3 4 6 7 9 10
RPC#2 (1)
{4, 5, 7}
target 0
0 1 3 4 6 7 9 10
RPC#3 (0)

Sloppy Storage – Insertion

- Forward phase: Route through nodes that are closer to the key. Terminate when reaching *full* and *loaded* node.
- A node is *full* with respect to some key k when it stores l values whose TTL are at least one-half of the new value.
- A node is *loaded* with respect to k when it has received more than the maximum *leakage rate* β requests withing the past minute.
- l = 4, β = 12, (one store attempt every 5 sec)
- Backward phase: try inserting the value on the way back.

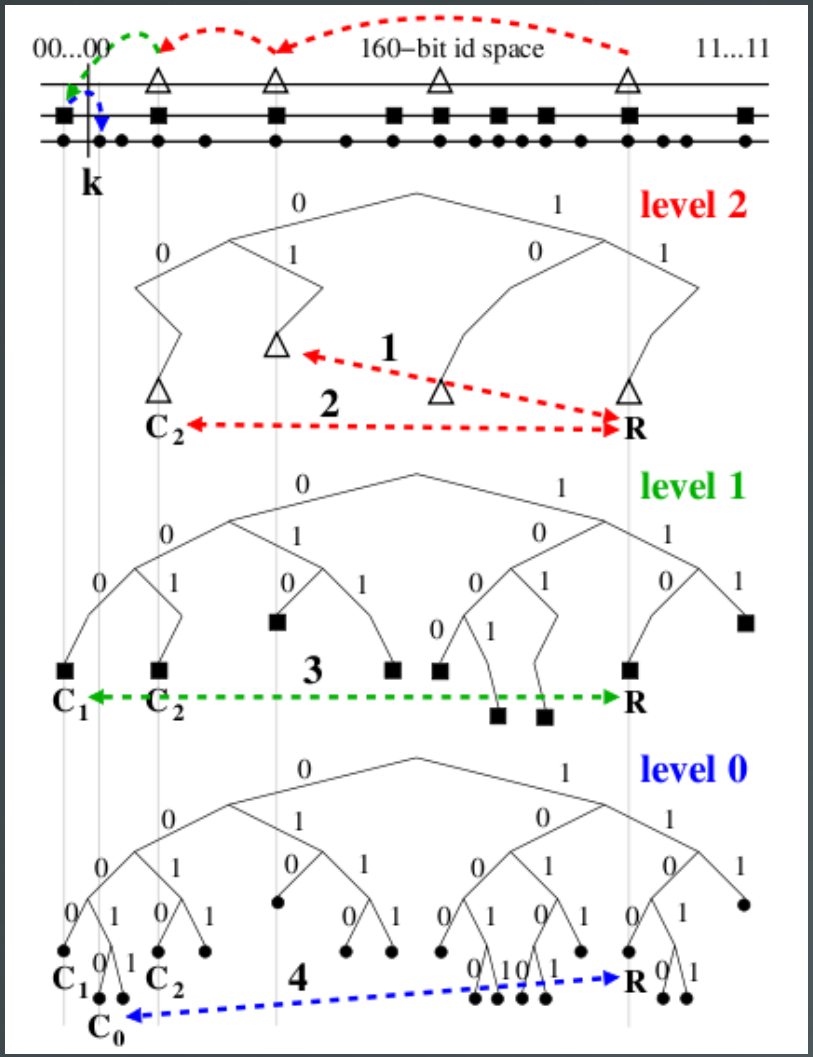
Sloppy Storage – Retrieval

- Simply traverses the ID space getting closer to key ID
- Take the result of the first node storying values for that key.

Hierarchical Operations

- Several levels of DSHT's called clusters
- Arbitrary deep DSHT hierarchy (consider 3-level)
- At each level for nodes to be in cluster their RTT has to be in less then some threshold
- Cluster indentifier: 160 bit random

Hierarchical Retrieval and Insertion



Joining and Managing Clusters

- Goals
  - Join suitable cluster
  - Quickly
- Join only if 80% of nodes in the cluster meet the RTT threshold
- Nodes store in the DSHT maping between subnets and their ip and port
- New nodes looks in the DSHT for its subnet.
- 5 hops out also stored in the DSHT as hints.

Joining and Managing Clusters 2

- Continuous measurements
- Every 5 minute conside changing cluster
- If more then 50% of nodes don't meet RTT you create your own cluster
- Consolidate clusters (some problems)

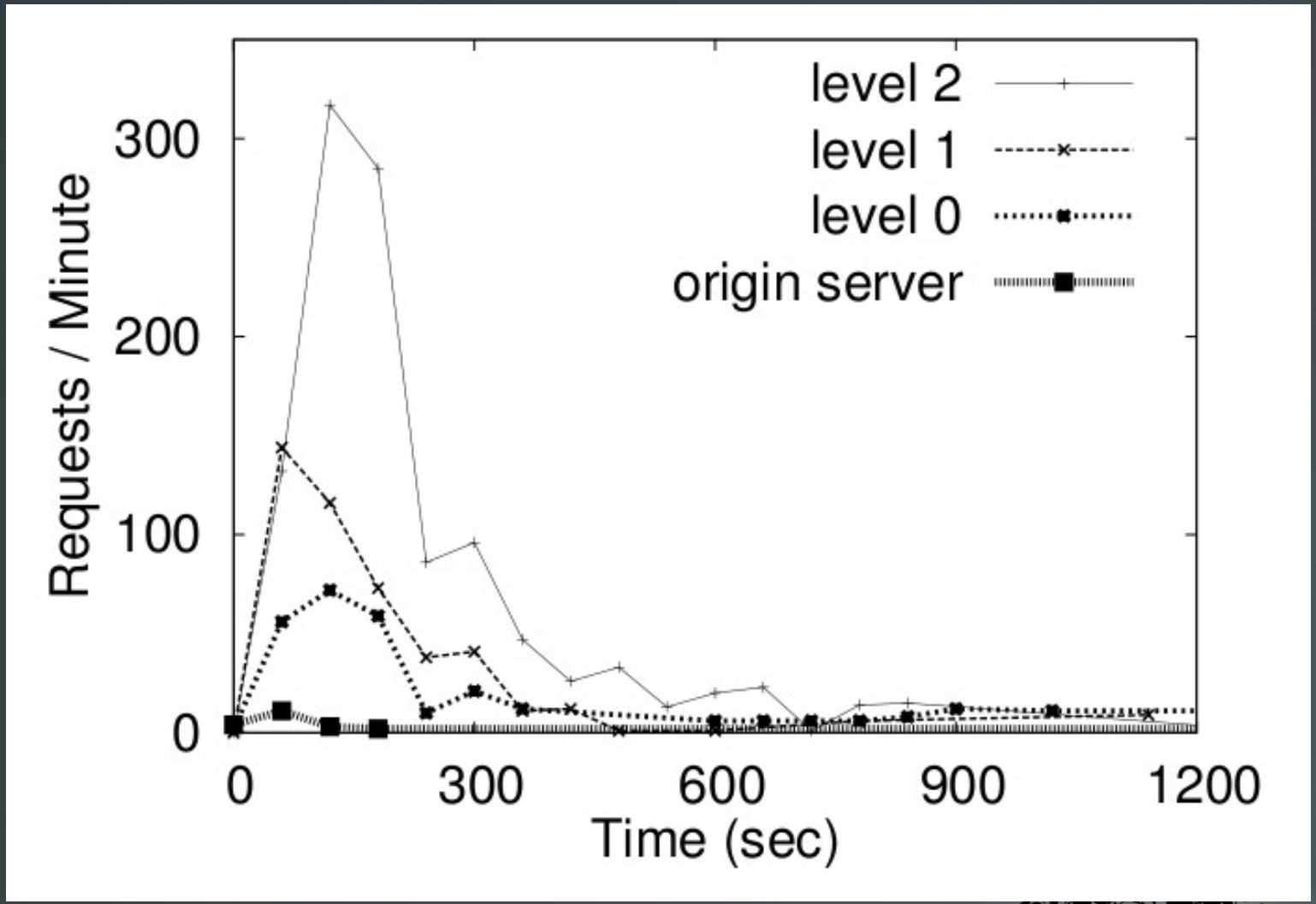
Implementation

- Coral: 14000 lines C++
- CoralDNS: 2000 lines C++
- Proxy: 4000 lines C++

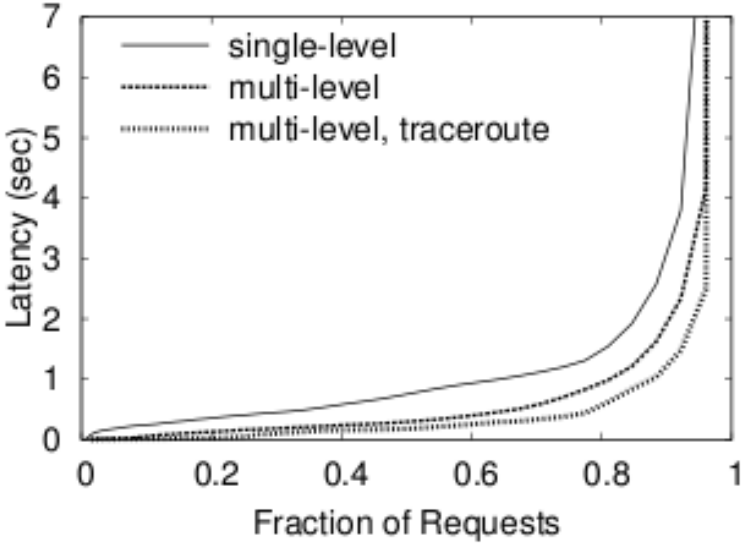
Evaluation

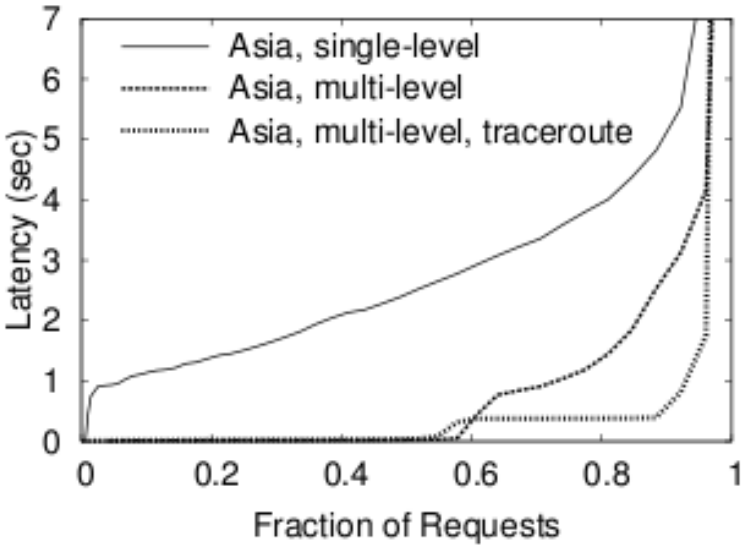
- 166 PL nodes with dnssrv and CoralProxy
- 3-level; 20ms, 60ms, rest
- 30 min stabilization
- 384Kbit/S Server with 12 pictures of 40KB
- Client on each machine making requests starting with random delay
- 99.6 requests/sec, 32800Kb/sec trafic

Number of requests



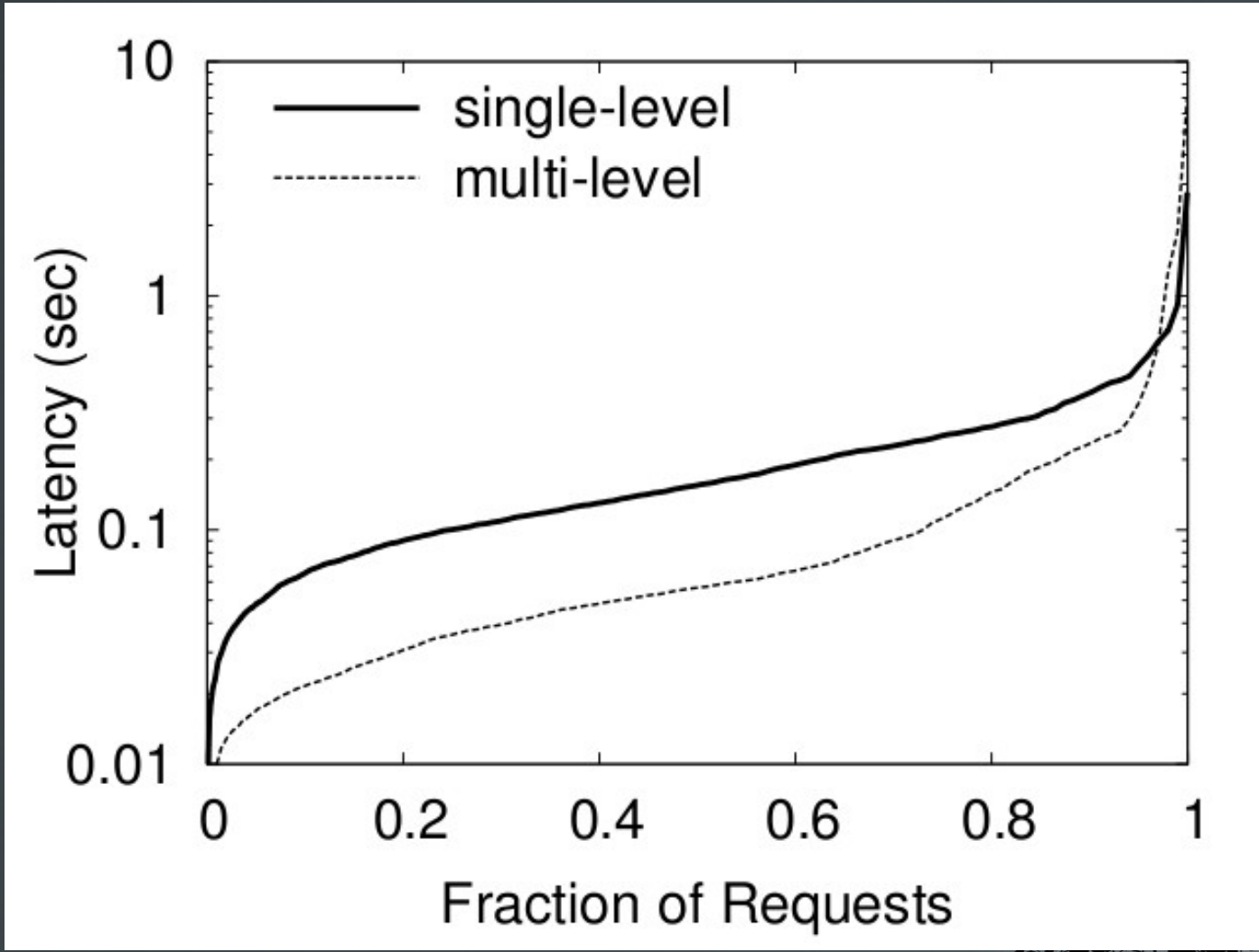
Latency



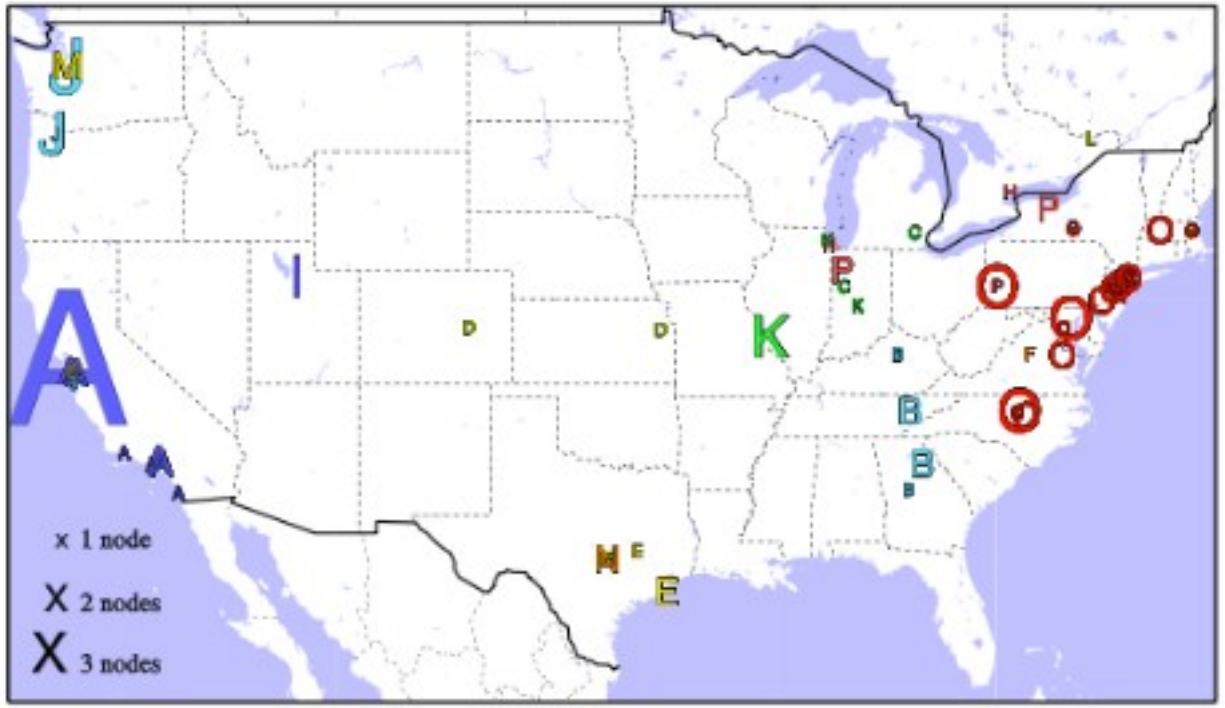


Request latency (sec)	All nodes		Asian nodes	
	50%	96%	50%	96%
single-level	0.79	9.54	2.52	8.01
multi-level	0.31	4.17	0.04	4.16
multi-level, traceroute	0.19	2.50	0.03	1.75

DSHT Latency



Clustering



Load Balancing

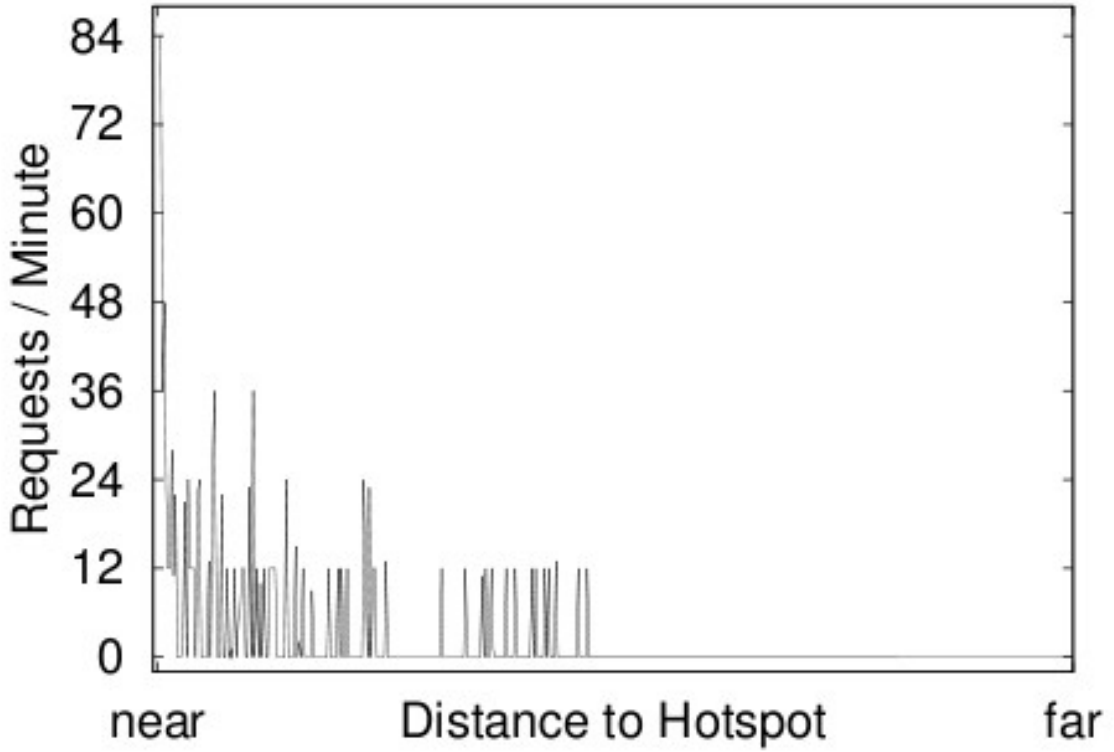


Figure 8: The total number of *put* RPCs hitting each Coral node per minute, sorted by distance from node ID to target key.