Democratizing Content Publication with Coral

Mike Freedman
Eric Freudenthal
David Mazières
New York University

www.scs.cs.nyu.edu/coral

A problem...

- Feb 3: Google linked banner to “julia fractals”
- Users clicking directed to Australian University web site
- …University’s network link overloaded, web server taken down temporarily…
The problem strikes again!

Feb 4: Slashdot ran the story about Google

...Site taken down temporarily...again

The response from down under...

Feb 4, later...Paul Bourke asks:

“They have hundreds (thousands?) of servers worldwide that distribute their traffic load. If even a small percentage of that traffic is directed to a single server ... what chance does it have?”

→ Help the little guy ←
Existing approaches

- **Client-side proxying**
  - Squid, Summary Cache, hierarchical cache, CoDeeN, Squirrel, Backslash, PROOFS, ...
  - Problem: Not 100% coverage

- **Throw money at the problem**
  - Load-balanced servers, fast network connections
  - Problem: Can’t afford or don’t anticipate need

- **Content Distribution Networks (CDNs)**
  - Akamai, Digital Island, Mirror Image
  - Centrally managed, needs to recoup costs

Coral’s solution...

Pool resources to dissipate flash crowds

- Implement an open CDN
- Allow anybody to contribute
- Works with unmodified clients
- CDN only fetches once from origin server
Coral’s solution...

Pool resources to dissipate flash crowds
- Strong locality without a priori knowledge
- No hotspots in CDN
- Should all work automatically with nobody in charge

Contributions
- Self-organizing clusters of nodes
  - NYU and Columbia prefer one another to Germany
- Rate-limiting mechanism
  - Everybody caching and fetching same URL does not overload any node in system
- Decentralized DNS Redirection
  - Works with unmodified clients

No centralized management or a priori knowledge of proxies’ locations or network configurations
Using CoralCDN

- Rewrite URLs into “Coralized” URLs
  
  \[ \text{www.x.com} \rightarrow \text{www.x.com.nyud.net:8090} \]

- Directs clients to Coral, which absorbs load

- Who might “Coralize” URLs?
  - Web server operators Coralize URLs
  - Coralized URLs posted to portals, mailing lists
  - Users explicitly Coralize URLs

CoralCDN components

- DNS Redirection: Return proxy, preferably one near client
- Cooperative Web Caching
- Fetch data from nearby
Functionality needed

- **DNS**: Given network location of resolver, return a proxy near the client
  
  \[
  \text{put (network info, self)} \\
  \text{get (resolver info)} \rightarrow \{\text{proxies}\}
  \]

- **HTTP**: Given URL, find proxy caching object, preferably one nearby
  
  \[
  \text{put (URL, self)} \\
  \text{get (URL)} \rightarrow \{\text{proxies}\}
  \]

Use a DHT?

- Supports put/get interface using key-based routing
- Problems with using DHTs as given

- Lookup latency
- Transfer latency
- Hotspots
Coral distributed index

- Insight: Don’t need hash table semantics
  - Just need one well-located proxy

- put (key, value, ttl)
  - Avoid hotspots

- get (key)
  - Retrieves some subset of values put under key
  - Prefer values put by nodes near requestor

- Hierarchical clustering groups nearby nodes
  - Expose hierarchy to applications
  - Rate-limiting mechanism distributes puts

CoralCDN components

- DNS Redirection
  - Return proxy, preferably one near client

- Cooperative Web Caching
  - Fetch data from nearby

- Resolver

- Browser

www.x.com.nyud.net
Coral CDN components

DNS Redirection
Return proxy, preferably one near client

Coral httpprx dnssrv

Coral httpprx dnssrv

get

get

Coral httpprx dnssrv

Coral httpprx dnssrv

Fetch data from nearby

Cooperative Web Caching

Resolver

Browser

www.x.com.nyud.net

Key-based XOR routing

000... Distance to key 111...

Thresholds
None
< 60 ms
< 20 ms

- Minimizes lookup latency
- Prefer values stored by nodes within faster clusters
Prevent insertion hotspots

- Store value once in each level cluster
  - Always storing at closest node causes hotspot
- Halt put routing at full and loaded node
  - Full → M vals/key with TTL > ½ insertion TTL
  - Loaded → β puts traverse node in past minute
- Store at furthest, non-full node seen

Challenges for DNS Redirection

- Coral lacks…
  - Central management
  - *A priori* knowledge of network topology
    - Anybody can join system
  - Any special tools (e.g., BGP feeds)

- Coral has…
  - Large # of vantage points to probe topology
  - Distributed index in which to store network hints
  - Each Coral node maps nearby networks to self
Coral’s DNS Redirection

- Coral DNS server probes resolver

- Once local, stay local
  
  **When serving requests from nearby DNS resolver**
  
  - Respond with nearby Coral proxies
  - Respond with nearby Coral DNS servers
    → Ensures future requests remain local

- Else, help resolver find local Coral DNS server

---

DNS measurement mechanism

- Server probes client (2 RTTs)

  - Return servers within appropriate cluster
    - e.g., for resolver RTT = 19 ms, return from cluster < 20 ms
  
  - Use network hints to find nearby servers
    - i.e., client and server on same subnet
  
  - Otherwise, take random walk within cluster
Experimental results

Consider requests to Australian web site:
- Does Coral absorb flash crowds?
- Does clustering help latency?
- Does Coral form sensible clusters?
- Does Coral prevent hotspots?

Experimental setup
- 166 PlanetLab hosts; Coral node and client on each
- Twelve 41-KB files on 384 Kb/sec (DSL) web server
- (0.6 reqs / sec) / client → 32,800 Kb/sec aggregate

Solves flash-crowd problem

Coral hits in 20 ms cluster

Local caches begin to handle most requests

Hits to origin web server
Benefits end-to-end client latency

![Graph showing benefits of end-to-end client latency](image)

- multi-level, hints
- multi-level
- single-level

![Graph showing benefits of end-to-end client latency](image)

- Asia, multi-level, hints
- Asia, multi-level
- Asia, single-level
Finds natural clusters

Nodes share letter → in same < 60 ms cluster
Size of letter → number of collocated nodes in same cluster

Prevents put hotspots

Nodes aggregate put/get rate: ~12 million / min
Rate-limit per node (β): 12 / min
RPCs at closest leaked through 7 others: 83 / min
Conclusions

- Coral indexing infrastructure
  - Provides non-standard P2P storage abstraction
  - Stores network hints and forms clusters
    - Exposes hierarchy and hints to applications
    - Prevents hotspots

- Use Coral to build fully decentralized CDN
  - Solves Slashdot effect
  - Popular data $\rightarrow$ widely replicated $\rightarrow$ highly available
    - Democratizes content publication

For more information…

www.scs.cs.nyu.edu/coral

www.scs.cs.nyu.edu.nyud.net:8090/coral