Experiences with CoralCDN: A Five-Year Operational View

Michael J. Freedman

Presented by John Rula
Outline

• Background
• CoralCDN
• Usage Analysis
• Lessons
  • Web
  • CDNs
  • Platforms
Background

• DHTs (Chord, Kalemdia)

• PlanetLab

• CDNs (CoralCDN in particular)
CoralCDN

- Open CDN
- Service for Coral indexing layer
  - Sloppy hashing and self-organizing
- Launched in 2004
- Runs on PlanetLab
- Designed to prevent “slashdot effect”
Key components

- DNS-based redirection (*.nyud.net)
- Coral (Distributed Sloppy Hash Table)
- Latency based clusters
- Hierarchical overlays
How it works

1. Client looks up http://x.nyud.net address through local resolver. IP of close proxy is returned from Coral authoritative DNS server.
2. Client sends request to proxy.
3. CoralCDN node queries the Coral index layer for content.
4. CoralCDN node retrieves and caches the requested content from other CoralCDN nodes.
5. This is returned to the client.
6. The content is announced to other CoralCDN nodes through the Coral index.
# CoralCDN usage

<table>
<thead>
<tr>
<th>Year</th>
<th>Unique Domains</th>
<th>Unique URLs</th>
<th>% URLs with 1 req</th>
<th>Reqs to most popular URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>2005</td>
<td>7881</td>
<td>577K</td>
<td>54%</td>
<td>697K</td>
</tr>
<tr>
<td>2007</td>
<td>21555</td>
<td>588K</td>
<td>59%</td>
<td>410K</td>
</tr>
<tr>
<td>2009</td>
<td>20680</td>
<td>1787K</td>
<td>77%</td>
<td>1578K</td>
</tr>
</tbody>
</table>

## Summary

- 15-25M requests / day
- # of requests vs. URL follows zipf distribution
  - Long tail
- >70% hit local cache
Usage Scenarios

• Resurrecting old content
  • Request coral-ized URL to check for archive cache
  • Caches are evicted over time (24 hour max w/ expiration time)

• Accessing unpopular content
  • CDN is unnecessary for unpopular content
  • Long tail?

• Serving long-term popular content
  • Cooperation is not needed

• Surviving flash crowds to content
  • Designed purpose
Lessons for Web

• Open API – elastic resource for content distribution
  • Transparency
  • Deep caching
  • Server control
  • Ad-friendly
  • Forward compatible

RewriteEngine on
RewriteCond % {HTTP_USER_AGENT} !^CoralWebPrx
RewriteCond % {QUERY_STRING} !(^|&)coral-no-serv$
RewriteRule ^(.*)$ http://%{HTTP_HOST}.nyud.net % {REQUEST_URI} [R,L]
Lessons for Web

• Security and resource protection
  • Functionality limited to GET and HEAD
  • Prevents spammers and anonymous pass-throughs (CoDeeN)
  • Bandwidth monitoring and limiting
  • Domain blacklist
  • Access-control restrictions (Academic journals and PlanetLab)
Lessons for Web

• Security and naming conflation
  • Same Origin Policy (SOP) restrictions
Lessons for CDNs

• How and when to cache

• Faulty origins
  • “Proxies should accept content conservatively and serve results liberally”
  1. Cache negative service results
  2. Serve stale content if origin faulty
  3. Prevent truncations through whole-file overwrites
Lessons for CDNs

• Managing oversubscribed bandwidth
  • Fairness vs. utilization

• Performance jitter
  • Cluster hysteresis
  • Stability in the face of performance variations
Lessons for Platforms

• Exposing information and expressing preferences across layers
  • Hosting platforms should provide better visibility and control (esp. to higher levels)
  • Also more control over base services

• Providing support for fault-tolerance
Future Directions

• Fully P2P CDN
• Hybrid CDNs
  • Akamai Netsession
Questions?