Experiences with CoralCDN

A Five-Year Operational View

Michael J. Freedman
Princeton University

www.coralcdn.org
A Cooperative, Self-Organizing CDN

**Goal:** To make desired content widely available regardless of publisher’s own resources, by organizing and utilizing any cooperative resources.
http://example.com/path

http://example.com.nyud.net/path
Adopted by:

Clients

Servers

Third-parties
Many of you have used CoralCDN.
Many of you have used CoralCDN
Many of you have used CoralCDN
Many of you have used CoralCDN
Many of you have used CoralCDN
Many of you have used CoralCDN
Daily Request Volume

2M clients – 2 TB content – 20K origin domains

From 300-400 PlanetLab servers
Based on peer-to-peer DHT

1. Weakened consistency + algorithms that prevent tree saturation during lookup
2. Decentralized clustering for locality and hierarchical lookup
3. Cooperative HTTP / DNS that leverages locality
Based on peer-to-peer DHT

1. Weakened consistency + algorithms that prevent tree saturation during lookup
2. Decentralized clustering for locality and hierarchical lookup
3. Cooperative HTTP / DNS that leverages locality
Interactions with the External Environment

Clients

Origin Domains

Virtualization Layer
1. Experiences
   - Naming
   - Fault Tolerance
   - Resource management

2. Revisit CoralCDN’s design
Naming

- Flexible, open API
- Mismatch with domain-based access control policies
CoralCDN’s Platform-as-a-Service API

Rewrite rules in origin webservers

RewriteEngine on

RewriteCond %{HTTP_USER_AGENT} !^CoralWebPrx
RewriteCond %{QUERY_STRING} !(^|&)coral-no-serve$
RewriteRule ^(.*)$ http://%{HTTP_HOST}.nyud.net%{REQUEST_URI} [R,L]
CoralCDN’s Platform-as-a-Service API

Rewrite rules in origin webservers

```
RewriteEngine on
RewriteCond %{HTTP_USER_AGENT} !^CoralWebPrx
RewriteCond %{QUERY_STRING} !(^|&)coral-no-serve$
RewriteCond %{HTTP_REFERER} slashdot\.[org $[NC]
RewriteCond %{HTTP_REFERER} digg\.[com $[NC,OR]
RewriteCond %{HTTP_REFERER} blogspot\.[com $[NC,OR]
RewriteRule ^(.*)$ http://%{HTTP_HOST}.nyud.net%{REQUEST_URI}$ [R,L]
```

Sites integrate with load/bandwidth monitoring

Elastic Provisioning
Naming Conflation

http://domain.service1.service2/path

1. Location to retrieve content ✅
2. Human-readable name for administrative entity ❌
3. Security policies to govern objects’ interactions ❌
Domain-based Security Policies

Web Page

Cookies

evil.com

target.com

Document Object Model
Domain-based Security Policies

Web Page

evil.com.nyud.net

target.com.nyud.net

Cookies

Document Object Model

Defaults violate least privilege
Fault Tolerance: Failure Decoupling

Internal failures:
- DHT nodes
- DNS servers, HTTP proxies
- Management service

External failures:
- Decouple IPs from hosts
- Interactions with origin sites
happens!

<table>
<thead>
<tr>
<th>Origin Status</th>
<th>CoralCDN Reaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Unresponsive</td>
<td>• Cache negative results</td>
</tr>
<tr>
<td>2. Returns error code</td>
<td>• Serve stale content</td>
</tr>
<tr>
<td>3. Reply truncated</td>
<td>• Use whole-file overwrites</td>
</tr>
</tbody>
</table>
happens!

**Origin Status**

1. Unresponsive
2. Returns error code
3. Reply truncated

**CoralCDN Reaction**

- Cache negative results
- Serve stale content
- Use whole-file overwrites

Maintain status quo unless improvements are possible
What is “failure”?  

Return values should have fail-safe defaults
Resource Management

- Control over bandwidth consumption
- Control and visibility into environment’s resources
Some timeline...

Mar 2004
CoralCDN released on PlanetLab
Some timeline...

Mar 2004
CoralCDN released on PlanetLab

Aug 2004
Slashdotted

---

Coral CDN entered public beta in August 2004 through PlanetLab. It was slashdotted in March 2004.

---

**IT: Coral P2P Cache Enters Public Beta**

Posted by smoothy on Saturday August 28, from the will-it-scale dept.

Elloquence writes: "infoAnarchy reports that Coral, a large scale distributed research network of 400 servers. You can use Coral right now by appending "nyox.net:8090" to a hostname. View Slashdot through Coral. Is this the end of the Slashdot effect?"
1. PlanetLab traffic jumps
2. Site threatens to yank PL
3. PL admin kills slice
4. Slice restored next day
5. Initiates discussion of resource limits for slices
Demand >> Supply:
Enter Fair-Sharing Algorithms

$$\sum_i d_i \leq S$$

Avg MB per hour ($d_i$)

Domains with heaviest consumption
Demand >> Supply:
Enter Fair-Sharing Algorithms

find max $\lambda$, s.t.

$$\sum_i \min (\lambda, d_i) \leq S$$

Domains with heaviest consumption
Demand >> Supply:
Enter Fair-Sharing Algorithms

find max $\lambda$, s.t.

$$\sum_i \min (\lambda, d_i) \leq S$$

Domains with heaviest consumption
Admission Control under Fair-Sharing

~10 kB imgs 3.3% rejected
~5 MB videos 89% rejected

Requests per Domain

Unique Domains Ordered by Decreasing Popularity

All Responses

Forbidden Responses

Demand > 10 TB

Supply ≤ 2 TB
Some timeline...

1. PlanetLab traffic jumps
2. Site threatens to yank PL
3. PL admin kills slice
4. Slice restored next day
5. Initiates discussion of resource limits for slices
Resource Management: Us vs. Them

**Application Hammer**
- Track HTTP traffic
- If site > fair share rate, reject via HTTP 403
- If total > peak rate, close server socket

**Platform Hammer**
- Track all network traffic
- If total > 80% daily rate, BW shaping in kernel
Resource Management: Us vs. Them

Application Hammer
- Track HTTP traffic
- If site > fair share rate, reject via HTTP 403
- If total > peak rate, close server socket

Platform Hammer
- Track all network traffic
- If total > 80% daily rate, BW shaping in kernel

Result: HTTP traffic is 1/2 - 2/3 of all traffic

Lower layers should expose greater visibility and control over resources
1. Experiences
   — Naming
   — Fault Tolerance
   — Resource management

2. Revisit CoralCDN’s design
Usage Scenarios

1. Resurrecting old content
2. Accessing unpopular content
Usage Scenarios

1. Resurrecting old content
2. Accessing unpopular content
3. Serving long-term popular content
Usage Scenarios

1. Resurrecting old content
2. Accessing unpopular content
3. Serving long-term popular content
Usage Scenarios

1. Resurrecting old content
2. Accessing unpopular content
3. Serving long-term popular content

<table>
<thead>
<tr>
<th>Top URLs</th>
<th>% Reqs</th>
<th>Agg Size (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01 %</td>
<td>49.1 %</td>
<td>14</td>
</tr>
<tr>
<td>0.10 %</td>
<td>71.8 %</td>
<td>157</td>
</tr>
<tr>
<td>1.00 %</td>
<td>84.8 %</td>
<td>3744</td>
</tr>
<tr>
<td>10.00 %</td>
<td>92.2 %</td>
<td>28734</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Result</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local Cache</td>
<td>70.4 %</td>
</tr>
<tr>
<td>Origin Site</td>
<td>9.9 %</td>
</tr>
<tr>
<td>CoralCDN Proxy</td>
<td>7.1 %</td>
</tr>
<tr>
<td>4xx/5xx Error</td>
<td>12.6 %</td>
</tr>
</tbody>
</table>
Usage Scenarios

1. Resurrecting old content
2. Accessing unpopular content
3. Serving long-term popular content
4. Surviving flash crowds to content
Usage Scenarios

1. Resurrecting old content
2. Accessing unpopular content
3. Serving long-term popular content
4. Surviving flash crowds to content
## Usage Scenarios

1. Resurrecting old content
2. Accessing unpopular content
3. Serving long-term popular content
4. Surviving flash crowds to content

<table>
<thead>
<tr>
<th>5 second epochs</th>
<th>10 minute epochs</th>
</tr>
</thead>
<tbody>
<tr>
<td>24% epochs</td>
<td>99.93% epochs</td>
</tr>
<tr>
<td>≥ 1 domain with</td>
<td>≥ 1 domain with</td>
</tr>
<tr>
<td>10x incr</td>
<td>10x incr</td>
</tr>
<tr>
<td>0.006% epochs</td>
<td>28% epochs</td>
</tr>
<tr>
<td>≥ 1 domain with</td>
<td>≥ 1 domain with</td>
</tr>
<tr>
<td>100x incr</td>
<td>100x incr</td>
</tr>
<tr>
<td>0 % epochs</td>
<td>0.21% epochs</td>
</tr>
<tr>
<td>≥ 1 domain with</td>
<td>≥ 1 domain with</td>
</tr>
<tr>
<td>1000x incr</td>
<td>1000x incr</td>
</tr>
</tbody>
</table>
Conclusions?

- Most requested content is long-term popular and already cached locally
- “Flash” crowds occur, but on order of minutes
Conclusions?

- Most requested content is long-term popular and already cached locally
- "Flash" crowds occur, but on order of minutes

Focus on long-term popular
- Little / no HTTP cooperation
- Global discovery (e.g., DNS)

Focus on flash crowds
- Regional coop. as default
- Global coop. as failover
Reconfiguring Coral CDN’s design

• Leverage Coral hierarchy for lookup

Latency 90% ↓  Origin Load 5% ↑  Failover to global 0.5%
Reconfiguring CoralCDN’s design

- Leverage Coral hierarchy for lookup

- During admission control, bias against long-term use

\[ \sum_{i} \min (\lambda, d_i) < S \]

heavily weight history in ewma
Conclusions

1. Experiences
   - Naming
   - Fault Tolerance
   - Resource management

2. Revisit CoralCDN’s design
   - Current design unnecessary for deployment / most use
   - Easy changes to promote flash-crowd mitigation
Can we reach Internet scale?

www.firecoral.net

Initial beta-release
of browser-based P2P web cache