Using HTTP Link: Header for Gateway Cache Invalidation

Mike Kelly <mike@mykanjo.co.uk>
Michael Hausenblas <michael.hausenblas@deri.org>
What is a gateway cache?

"reverse proxy cache"
A layer between all clients and destination server

Objective:
Minimize demand on destination server
Not so concerned with reducing bandwidth
How do they work?

They can leverage the 3 principal caching mechanisms:

- Expiration
- Validation
- Invalidation

HTTP has mechanisms for each of these
Expiration-based caching

< 200 OK
< Content-Type: text/html
< Cache-Control: public, s-maxage=600
< ....

Pros:
+ Simple
+ No contact with server until expiration

Cons:
- Inefficient
- Difficult to manage
Validation-based caching

< 200 OK
< ETag: "686897696a7c876b7e"
> GET /example
> If-None-Match: "686897696a7c876b7e"
< 304 Not Modified

Pros:
+ Reduces bandwidth
+ Ensures freshness

Cons:
- Server handling every request
- Generating 304 still costs processing and I/O
Expiration+Validation caching

< 200 OK
< ETag: "686897696a7c876b7e"
< Cache-Control: public, s-maxage=600

Pros:
+ Expiration reduces contact with server
+ Validation reduces bandwidth

Cons:
- "Worst case" inefficiency
- Still managing caching rules
Invalidation-based caching

- Responses fresh until invalidated
  (by non-safe requests)

In HTTP:

- PUT
- POST
- PATCH
- DELETE
  (PURGE?)
How is this possible?

Product of adhering to constraints of REST, particularly:

**Uniform Interface**
+ Self-descriptive messages

Intermediaries can make *assertions* about client-server interactions.
Invalidation-based caching

Pros:
+ Caches have self-control
+ "Best case" efficiency
+ Ensured freshness*

Cons:
- Only reliable for gateway caches
- Impractical*

* (sort of)
Cache invalidation in practice

Two main problems for cache invalidation arise from pragmatism and trade-offs in resource granularity and identification:

- The "Composite Problem"
- The "Split-resource Problem"
Composite Problem

Perfect World:
<collection>
  <item rel="item" href="/items/123"/>
  <item rel="item" href="/items/asdf"/>
  <item rel="item" href="/items/foobar"/>
</collection>

Real World:
<collection>
  <item rel="item" href="/items/123">
    <title>Item 123</title>
    <content>Content for item 123 - an example of embedded state</content>
  </item>
  <item rel="item" href="/items/asdf">
    <title>Item asdf</title>
    <content>This state is also embedded</content>
  </item>
  <item rel="item" href="/items/foobar">
    <title>FooBar</title>
    <content>Yet more embedded state!! :(</content>
  </item>
</collection>
Composite Problem

What effect would the following interaction have on the composite collection it belongs to?

> PUT /composite-collection/item123
< 200 OK
The Split-resource Problem

Given `/document` resource with representations:

`/document.html`
`/document.xml`
`/document.json`

When a client does this:

PUT `/document`

Then invalidation of each representation is invisible to intermediaries
What's the Problem?

Visibility
.. The Solution

Beef up the uniform interface:

Express these common types of resource dependency as control data using Link header and standard link relations

This increases:

- Self-descriptiveness of messages
- Visibility

"Link Header-based Invalidation of Caches" (LHIC)
Express dependency in response to an invalidating request

> PUT /composite-collection/item123

< 200 OK
< Link: </composite-collection>;
< rel="http://example.org/rels/dependant"
Express dependencies in initial cacheable responses

> GET /document.html
< 200 OK
< Link: </document>; rel="http://example.org/rels/dependsOn"

> GET /document.xml
< 200 OK
< Link: </document>; rel="http://example.org/rels/dependsOn"

> GET /document.json
< 200 OK
< Link: </document>; rel="http://example.org/rels/dependsOn"

> PUT /document
< 200 OK
Comparison

LHIC-I
+ More dynamic control of invalidation
- DoS risk
- Invalidation does not cascade

LHIC-II
+ No DoS risk
+ Cascading invalidation
- Complexity
Conclusion

LHIC injects lost visibility. Resulting mechanism:

+ Very efficient
+ Ensures freshness
+ Easily managed
+ Leverages existing specs

- Only for gateway caching
  + Combine Invalidation (gateway) & Validation (client)
Considerations

Resource state altered outside of uniform interface
- Don't do that
- Reintroduce expiration and validation

Peering
- Further research

Size limits for HTTP headers