

CS290f - Lecture 12 Caching

Scalable Internet Services, Fall 2006

Thorsten von Eicken
Department of Computer Science
University of California at Santa Barbara

Some database material from J. Hellerstein @UCB

Caching pages and fragments

Let's first repeat the mantra:

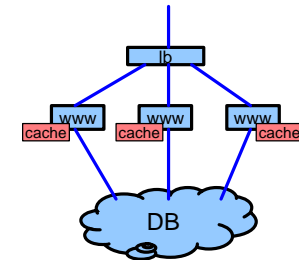
- Make your application **scalable on day one**
- On subsequent days, **don't do anything that compromises scalability**

And also:

- Premature optimization is the root of all evil

Did I forget?

- 1 programmer year == 100 Amazon machine years



Caching in Rails

Caching granularity

- Page caching
- Action caching

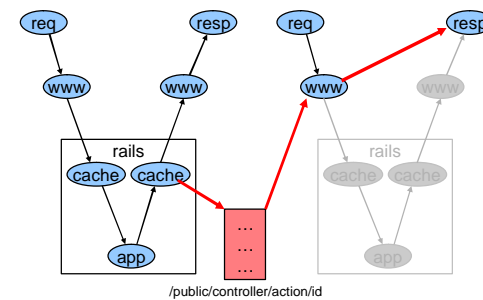
Invalidations

- Explicit invalidations
- Cache sweeper
- Timed invalidations

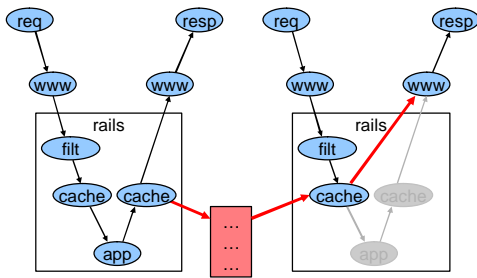
Reference

- Agile web development book, section 21.5 & 22.10

Page caching



Action caching



Page & action caching example

```
class ContentController < ApplicationController
  before_filter :verify_premium_user, :except=>:public_content

  caches_page :public_content
  caches_action :premium_content

  def public_content
    @articles = Article.list_public
  end

  def premium_content
    @articles = Article.list_premium
  end

private
  def verify_premium_user
    user = session[:user_id]
    user = User.find(user) if user
    unless user && user.active?
      redirect_to :controller=>"login", :action=>"signup_new"
    end
  end
end
```

Cache limitations

■ Content based on session information

- In particular if `current_user` determines content!

■ 3rd party database changes

- Other app changes database content
- Our app can't invalidate cache

■ Time-based content

- "this record changed 12 minutes ago"
- "this auction closes in 1 hour 10 minutes"

■ Note: caching enabled in `production.rb`

- `config.action_controller.perform_caching = true`

Explicit cache invalidation

```
def create_article
  article = Article.new(params[:article])
  if article.save
    expire_page :action=>"public_content"
  else
    # ...
  end
end

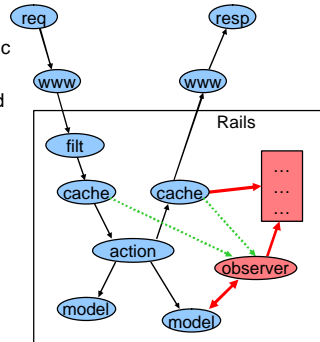
def update_article
  article = Article.new(params[:article])
  if article.save
    expire_action :action=>"premium_content", :id=>article
  else
    # ...
  end
end

def delete_article
  Article.destroy(params[:id])
  expire_page :action=>"public_content"
  expire_action :action=>"premium_content", :id=>params[:id]
end
```

Cache sweepers

■ Motivation

- Controller contains business logic
 - ◆ “no connection” to HTML
- Cache invalidation intimately tied to HTML
- Don't intertwine the two!



Cache sweeper example

```
class ArticleSweeper < ActionController::Caching::Sweeper
  observe Article
  def after_create(article)
    expire_public_page
  end
  def after_update(article)
    expire_article_page(article.id)
  end
  def after_destroy(article)
    expire_public_page
    expire_article_page(article.id)
  end
private
  def expire_public_page
    expire_page :controller=>"content", :action=>'public_content'
  end
  def expire_article_page(article_id)
    expire_action :controller=>"content",
                 :action=>"premium_content", :id => article_id)
  end
end
```

Cache sweeper cont.

```
class ContentController < ApplicationController
  before_filter :verify_premium_user, :except=>:public_content

  caches_page :public_content
  caches_action :premium_content
  cache_sweeper :article_sweeper,
    :only => [:create_article, :update_article, :delete_article]

  # ...
end
```

Dynamic content & caching

■ Suppose:

```
class BlogController < ApplicationController
  def list
    @dynamic_content = Time.now.to_s
  end
end
```

■ How to cache a view that contains @dynamic_content?

- Cache fragments that do not have dynamic content

Caching with dynamic content

Cache page fragments

```
<%= @dynamic_content %>      <!-- Here's dynamic content. -->
<% cache do %>                <!-- Here's the content we cache -->
  <ul>
    <% for article in Article.find_recent -%>
      <li><p><%= h(article.body) %></p></li>
    <% end -%>
  </ul>
<% end %>
<%= @dynamic_content %>      <!-- End of cached content -->
<%= @dynamic_content %>      <!-- More dynamic content. -->
```

Can you spot the ugliness?

13

Controller "fix"

Check for cached content

```
class Blog1Controller < ApplicationController
  def list
    @dynamic_content = Time.now.to_s
    unless read_fragment(:action => 'list' )
      logger.info("Creating fragment" )
      @articles = Article.find_recent
    end
  end
end
```

Expiring cache content

- `expire_fragment(:controller=>'blog', :action=>'list')`

14

Multiple fragments

Name the fragments

```
<% cache(:action=>'list', :part=>'articles') do %>
  <ul>
    <% for article in @articles -%>
      <li><p><%= h(article.body) %></p></li>
    <% end -%>
  </ul>
<% end %>
<% cache(:action=>'list', :part=>'counts') do %>
  <p>
    There are a total of <%= @article_count %> articles.
  </p>
<% end %>
```

Expiration with regexp

```
expire_fragment(%r{/blog2/list.*})
```

Per-user fragments?

15

Cache location

Fragment cache location

- Defined in `development.rb / production.rb`:
 - ◆ `ActionController::Base.fragment_cache_store = ...`

Available caching storage mechanisms:

- In memory, not particularly scalable
 - ◆ `ActionController::Caching::Fragments::MemoryStore.new`
- In the directory path, not good for many fragments
 - ◆ `ActionController::Caching::Fragments::FileStore.new(path)`
- In an external DRb server
 - ◆ `ActionController::Caching::Fragments::DRbStore.new(url)`
- In a memcached server.
 - ◆ `ActionController::Caching::Fragments::MemCachedStore.new(host)`

16

Memcached

- <http://www.danga.com/memcached/>
 - high-performance, distributed memory object caching system
 - generic in nature, intended for dynamic web applications
 - developed by Danga Interactive as part of LiveJournal.com
- **Running memcached**
 - Command line args for memory size, port, etc...
 - May need multiple memcached on 32-bit OS w/>2GB memory
- **Hash table interface**
 - put(key, value)
 - get(key, value)
 - Hash keys across machines (memcached instances)
 - "If a host goes down, the API re-maps that dead host's requests onto the servers that are available."
 - ◆ In a very rudimentary manner

17

Bare bones usage

- **Connecting to memcached**

```
require 'memcache'
cache = MemCache::new('10.0.0.15:11211', '10.0.0.17:11211:3',
                    :debug => true, :namespace => 'foo')
```
- **Adding servers, changing parameters**
 - cache.servers += ["10.0.0.15:11211:5"]
 - cache.c_threshold = 10_000
 - cache.compression = true
- **Putting values into the cache**
 - cache["my_key"] = "Some value"
 - cache[:other_key] = "Another value"
 - cache["object_key"] = { 'complex' => ["object", 2, 4] }
 - cache[Time::now.to_a[1..7]] ||= 0
- **Retrieving values**
 - val = cache["my_key"] # => "Some value"
 - val = cache["object_key"] # => {"complex" => ["object",2,4]}

18

Why not use the database instead?

- **ACID overhead**
 - Databases fundamentally commit to disk
 - Locks geared towards modification
 - ◆ Memcached never blocks
- **SQL overhead**
 - Parsing of query statements
 - Construction of query execution plan
- **Database query cache**
 - May be flushed often (mysql on every table update ?)
 - Caches "raw data", not generated view

19

Consistency

- **Race conditions**
 - get_foo() function adds a stale version of the Foo object
 - ◆ Query cache – miss – query DB – add to cache
 - update_foo() writes new version
 - ◆ Update DB – update cache
- **3 ways to put data into the cache**
 1. set -- unconditionally sets a given key with a given value
 - update_foo() should use this
 2. add -- adds to the cache, only if it doesn't already exist
 - get_foo() should use this
 3. replace -- sets in the cache only if the key already exists
 - not as useful, only for completeness
 - All three support an expiration time

20

Memcached design

- **Non-blocking**
- **libevent**
 - scale to any number of open connections
 - using epoll on Linux, if available
- **non-blocking network I/O**
- **refcounts internal objects**
 - objects can be in multiple states to multiple clients
- **uses its own slab allocator and hash table**
 - virtual memory never gets externally fragmented
 - allocations are guaranteed O(1)

21

Summary

- **Did you perhaps forget?**
 - Make your application **scalable on day one**
 - On subsequent days, **don't do anything that compromises scalability**
- **And also:**
 - Premature optimization is the root of all evil
- **Seriously**
 - Notice the difference between SQL's SELECT and SELECT FOR UPDATE
 - SELECT goes together with:
 - ◆ get info -> cache -> display
 - SELECT FOR UPDATE goes together with:
 - ◆ get info -> make decision -> update database -> possibly invalidate cache

22