Open Source Frameworks for Rapid Application Development

Marek Krętowski
Krzysztof Bandurski, Tomasz Łukaszuk, Tomasz Rybak

Software Departament
Faculty of Computer Science
Bialystok University of Technology

m.kretowski@pb.edu.pl
k.bandurski@pb.edu.pl, t.lukaszuk@pb.edu.pl, t.rybak@pb.edu.pl

Lecture topic

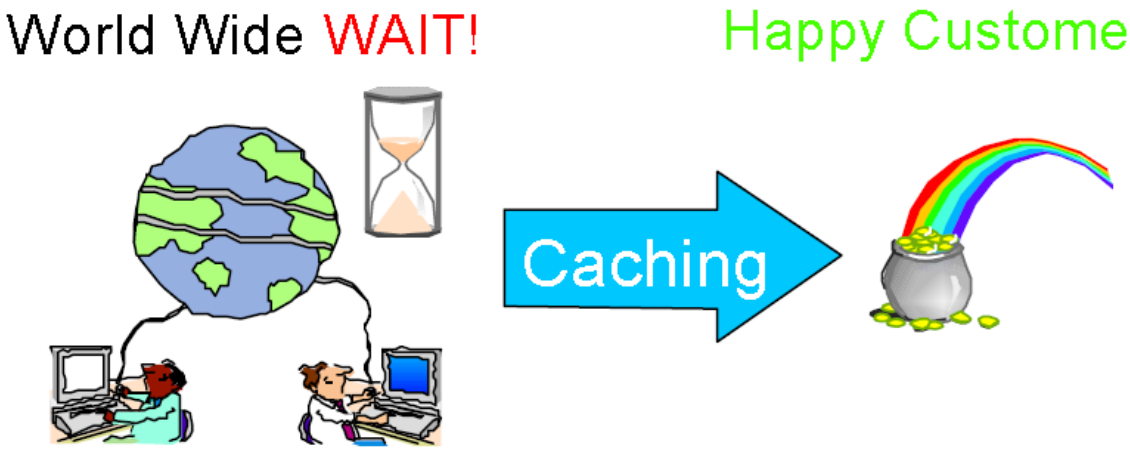
Caching and deploying

Caching and deploying: Table of content

Cache

- cache in computer science - component that improves performance by transparently storing data such that future requests for that data can be served faster, CPU cache, Disk cache
- web cache
  - web browsers cache, web proxy servers cache - store previous responses from web servers, such as web pages, reduce the amount of information that needs to be transmitted across the network
  - web applications cache - stores copies of documents, subsequent requests may be satisfied from the cache if certain conditions are met

Cache



source: www.almaden.ibm.com/u/mohan/Caching_VLDB2001.pdf

Caching in action

```
1  if (the page is in the cache) then
2      return the cached page
3  else {
4      generate the page
5      save the generated page in the cache (for next time)
6      return the generated page
7  }
```

Caching in Ruby on Rails

- Page, action, and fragment caching
- Sweepers
- Alternative cache stores
- Conditional GET support
- Advanced caching (additional plugins)

Caching setup

```
1  config.cache_classes = true | false
2
3  config.action_controller.perform_caching = true | false
4
5  config.cache_store =
6       :memory_store | :file_store | :drb_store
7       :mem_cache_store | :synchronized_memory_store |
8          :compressed_mem_cache_store
9       (custom store)
10
11 config.action_controller.page_cache_directory
12 config.action_controller.page_cache_extension
```

Page caching

- creating static pages as the result of action performing
- is super-fast
- it can't be applied to every situation (authentication)
- cache expiration is an issue that needs to be dealt with
- ignores all parameters (/products?page=1 will be products.html)

```
1  class ProductsController < ActionController
2
3     caches_page :index
4
5     def index
6        @products = Products.all
7     end
8
9     def create
10       expire_page :action => :index
11    end
12
13 end
```

Action caching

- allows authentication and other restriction to be run
- before filters can be run on before the cache is served
- clearing the cache works in the same way as with page caching

```
1  class ProductsController < ActionController
2
3     before_filter :authenticate
4     caches_action :index
5
6     def index
7        @products = Products.all
8     end
9
10    def create
11       expire_page :action => :index
12    end
13
14 end
```

Fragment caching

- allows a fragment of view logic to be wrapped in a cache block and served out of the cache store when the next request comes in

```
1  <% Order.find_recent.each do  |o| %>
2    <%= o.buyer.name %> bought <% o.product.name %>
3  <% end %>
4
5  <% cache do %>
6    All available products:
7    <% Product.all.each do |p| %>
8      <%= link_to p.name, product_url(p) %>
9    <% end %>
10 <% end %>
```

Fragment caching II

- managing fragment caching

```
1  #view
2  <% cache(:action => 'recent', :action_suffix => 'all_products') do %>
3     ...
4  <% end %>
5
6  #controller
7  expire_fragment(:controller => 'products', :action => 'recent',
8                  :action_suffix => 'all_products')
```

```
1  #view
2  <% cache('all_available_products') do %>
3    ...
4  <% end %>
5
6  #controller
7  expire_fragment('all_available_products')
```

Sweepers

- moving all the work required to expire cached content into an ActionController::Caching::Sweeper subclass (observer)

```
1  class ProductSweeper < ActionController::Caching::Sweeper
2     observe Product
3
4     def after_create(product)
5        expire_cache_for(product)
6     end
7     def after_update(product)
8        expire_cache_for(product)
9     end
10    def after_destroy(product)
11       expire_cache_for(product)
12    end
13
14    private
15    def expire_cache_for(product)
16       expire_page(:controller => 'products', :action => 'index')
17       expire_fragment('all_available_products')
18    end
19 end
```

Cache stores I

- page caches are always stored on disk
- different stores for the cached data created by action and fragment caches

1. ActiveSupport::Cache::MemoryStore
   - stores everything into memory in the same process
   - for application never performs manual cache item expiry
   - is able to store strings and arbitrary Ruby objects
   - is not thread-safe
2. ActiveSupport::Cache::FileStore
   - data is stored on the disk
   - default store, default path is tmp/cache
   - works well for all types of environments

Cache stores II

3. ActiveSupport::Cache::DRbStore
   - data is stored in a separate shared DRb process that all servers communicate with
   - keeps one cache around for all processes
   - requires that you run and manage a separate DRb process
4. ActiveSupport::Cache::MemCacheStore
   - works like DRbStore, but uses Danga's memcached instead
   - currently the most popular cache store for production websites
5. ActiveSupport::Cache::SynchronizedMemoryStore
6. ActiveSupport::Cache::CompressedMemCacheStore

Conditional GET support

- conditional GETs are a feature of the HTTP specification
- web servers can tell browsers that the response to a GET request hasn't changed since the last request and can be safely pulled from the browser cache
- HTTP_IF_NONE_MATCH, HTTP_IF_MODIFIED_SINCE

```
1  class ProductsController < ApplicationController
2    def show
3      @product = Product.find(params[:id])
4      if stale?(:last_modified => @product.updated_at.utc, :etag => @product)
5        # ... normal response processing
6      end
7    end
8  end
```

```
1  class ProductsController < ApplicationController
2    def show
3      @product = Product.find(params[:id])
4      fresh_when :last_modified=>@product.published_at.utc, :etag=>@product
5    end
6  end
```

Low level cache access

You can access cache stores for storing queries and other objects.

```
1  Rails.cache.read("city")    # => nil
2  Rails.cache.write("city", "Duckburgh")
3  Rails.cache.read("city")    # => "Duckburgh"
```

Caching with Django

- Cache stores
- The Per-Site Cache, The Per-View Cache, Template fragment caching
- The Low-Level Cache API

Setting up the cache - cache stores I

- settings.py: CACHES
- default store

```
1  CACHES = {
2      'default': {
3          'BACKEND': 'django.core.cache.backends.locmem.LocMemCache',
4      }
5  }
```

- some CACHES arguments: LOCATION, TIMEOUT, OPTIONS (MAX_ENTRIES, CULL_FREQUENCY)

1. Memcached
   - the fastest, most efficient type of cache
   - entirely memory-based
   - runs as a daemon and is allotted a specified amount of RAM
   - all data is stored directly in memory

Setting up the cache - cache stores II

- 'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache', 'LOCATION': 'ip:port'

2. Database Caching
   - create a cache table in your database
   - python manage.py createcachetable [cache_table_name]
   - 'BACKEND': 'django.core.cache.backends.db.DatabaseCache', 'LOCATION': 'my_cache_table'
3. Filesystem Caching
   - store cached items on a filesystem
   - 'BACKEND': 'django.core.cache.backends.filebased.FileBasedCache', 'LOCATION': '/var/tmp/django_cache'
   - each cache value will be stored as a separate file
4. Local-Memory Caching
   - multi-process and thread-safe
   - isn't as efficient as Memcached due to its simplistic locking and memory allocation strategies

Setting up the cache - cache stores III

- 'BACKEND': 'django.core.cache.backends.locmem.LocMemCache', 'LOCATION': 'unique-snowflake'

5. Dummy Caching (for Development)
   - cache that doesn't actually cache — it just implements the cache interface without doing anything
   - 'BACKEND': 'django.core.cache.backends.dummy.DummyCache'

The Per-Site Cache

- the simplest way
- cache entire site
- each page that doesn't have GET or POST parameters will be cached for a specified amount of time the first time it's requested
- to activate add 'django.middleware.cache.CacheMiddleware' to MIDDLEWARE_CLASSES setting
- add the following required settings to settings.py
  CACHE_MIDDLEWARE_SECONDS
  CACHE_MIDDLEWARE_KEY_PREFIX
- additional settings
  CACHE_MIDDLEWARE_ANONYMOUS_ONLY (needs AuthenticationMiddleware)

The per-view cache

- more granular way to use the caching framework
- caching the output of individual views
- the same effects as the per-site cache (including the ommission of caching on requests with GET and POST parameters)
- decorator cache_page

```
1  from django.views.decorators.cache import cache_page
2
3  @cache_page(60 * 15)
4  def my_view(request, param):
5      # ...
```

```
1  from django.views.decorators.cache import cache_page
2
3  urlpatterns = ('',
4          (r'^foo/(\d{1,2})/$', cache_page(my_view, 60 * 15)),
5  )
```

Template fragment caching

- cache template fragments
- `cache` template tag

```
1   {% load cache %}
2   {% cache 500 sidebar %}
3       .. sidebar ..
4   {% endcache %}
```

cache multiple copies of a fragment depending on some dynamic data that appears inside the fragment

```
1   {% load cache %}
2   {% cache 500 sidebar request.user.username %}
3       .. sidebar for logged in user ..
4   {% endcache %}
```

The low-level cache API

- module django.core.cache
- store objects in the cache with any level of granularity (strings, dictionaries, lists of model objects, etc.)
- basic interface is set(key, value, timeout_seconds) and get(key):

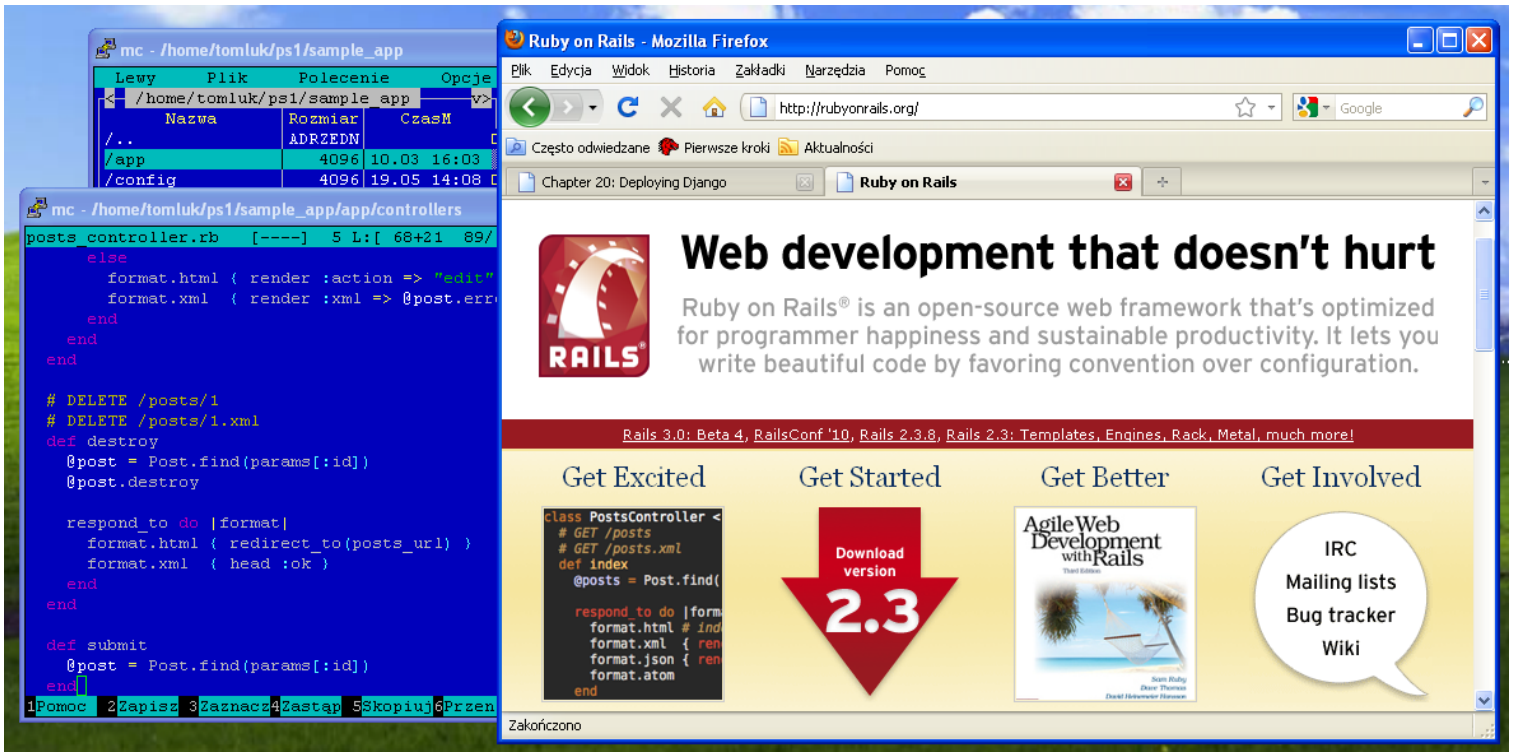
```
>>> cache.set('my_key', 'hello, world!', 30)
>>> cache.get('my_key')
'hello, world!'

>>> cache.set('a', 1)
>>> cache.set('b', 2)
>>> cache.set('c', 3)
>>> cache.get_many(['a', 'b', 'c'])
{'a': 1, 'b': 2, 'c': 3}

>>> cache.delete('a')
```

Deploying

- Software deployment is all of the activities that make a software system available for use.
- Deploying web application - make them available on the Internet



Web servers

1. WEBrick
   - quick and easy webserver to start developing
   - written in Ruby
   - not suitable for any production environment
   - rails server [-p 3000]
2. Mongrel
   - a small library that provides a very fast HTTP 1.1 server
   - supports clustered operation (Loadalancing)
   - gem install mongrel
   - mongrel_rails start -d -e production -a 127.0.0.1 -p 3000
3. Phusion Passenger
   - uses mod_rails or mod_rack to cooperate with Apache or Nginx
   - supports Apache and the fast and lightweight Nginx web servers
   - allows Ruby on Rails applications to use about 33% less memory, when used in combination with Ruby Enterprise Edition

Phusion Passenger with Apache I

- installation

```
1  gem install passenger
2  passenger-install-apache2-module
```

Listing 1: /etc/apache2/mods-enabled/passenger.conf

```
1  LoadModule passenger_module \
2    /usr/lib/ruby/gems/1.8/gems/passenger-2.2.11/ext/apache2/mod_passenger.so
3  PassengerRoot /usr/lib/ruby/gems/1.8/gems/passenger-2.2.11
4  PassengerRuby /usr/bin/ruby1.8
```

- deploying a Ruby on Rails application
  /webapps/rails/myapp => www.myhost.pl/myrailssite

Phusion Passenger with Apache II

Listing 2: apache.conf current virtual host

```
1  <VirtualHost *:80>
2      ServerName www.myhost.pl
3      DocumentRoot /var/websites
4      <Directory /var/websites>
5          Allow from all
6      </Directory>
7  </VirtualHost>
```

Listing 3: make a symlink

```
1  ln -s /webapps/rails/myapp/public var/websites/myrailssite
```

Phusion Passenger with Apache III

Listing 4: apache.conf

```
1  <VirtualHost *:80>
2      ServerName www.myhost.pl
3      DocumentRoot /var/websites
4      <Directory /var/websites>
5          Allow from all
6      </Directory>
7  </VirtualHost>
8
9  RailsBaseURI /myrailssite
10 <Directory /var/websites/myrailssite>
11     Options -MultiViews
12 </Directory>
```

Rails hosting

- cal.pl (129 PLN/year)
- hostingrails.pl (149 PLN/year)
- newrails.pl (96 PLN/year)
- bluehost.com
- hostmonster.com
- lunarpages.com
- railsplayground.com
- hostingrails.com
- www.alwaysdata.com

Web servers

1. development server
   - python manage.py runserver
   - http://localhost:8000 - by default
   - automatically reloads Python code for each request, as needed. You don't need to restart the server for code changes to take effect.
   - DO NOT USE THIS SERVER IN A PRODUCTION SETTING
2. Apache and mod_python
   - Support for mod_python has been deprecated, and will be removed in Django 1.5.
   - mod_python is an Apache plugin which embeds Python within Apache and loads Python code into memory when the server starts
   - requires Apache 2.x and mod_python 3.x
3. WSGI
   - Django's primary deployment platform
   - the Python standard for web servers and applications
   - default standard in Django 1.4

Django creators preferences to deploying Django

- Linux — Ubuntu specifically — as our operating system
- Apache and WSGI for the web server
- PostgreSQL as a database server

Apache and mod_python

- installation

```
1  LoadModule python_module /usr/lib/apache2/modules/mod_python.so
```

- deploying a Django application
  /webapps/djangoapps/myproj => www.myhost.pl/mydjangosite

Listing 5: apache.conf

```
1  Alias /media/ "/usr/lib/python2.4/site-packages/django/contrib/admin/media/"
2  Alias /mda_static/ "/webapps/djangoapps/myproj/_static/"
3  <Location "/mydjangosite">
4     SetHandler python-program
5     PythonHandler django.core.handlers.modpython
6     SetEnv DJANGO_SETTINGS_MODULE myproj.settings
7     PythonOption django.root /mydjangosite
8     PythonDebug On
9     PythonPath "['/webapps/djangoapps/myproj'] + sys.path"
10    PythonInterpreter mda
11 </Location>
12 <Location "/mda_static">
13    SetHandler none
14 </Location>
```

Apache and WSGI

- installation

```
1  LoadModule wsgi_module /usr/lib/apache2/modules/mod_wsgi.so
```

- deploying a Django application
  /webapps/djangoapps/myproj => www.myhost.pl/mydjangosite

Listing 6: apache.conf

```
1  Alias /media/ "/usr/lib/python2.4/site-packages/django/contrib/admin/media/"
2  Alias /mda_static/ "/webapps/djangoapps/myproj/static/"
3  WSGIScriptAlias /mydjangosite /webapps/djangoapps/myproj/wsgi.py
4  WSGIPythonPath /webapps/djangoapps/myproj
5
6  <Directory  /webapps/djangoapps/myproj>
7    <Files wsgi.py>
8      Order deny,allow
9      Allow from all
10   </Files>
11 </Directory>
```

Django hosting

- linuxpl.com (49 PLN/year)
- www.alwaysdata.com
- djangohosting.ch
- dreamhost.com
- http://code.djangoproject.com/wiki/DjangoFriendlyWebHosts