High Performance Ruby on Rails and MySQL

David Berube
Who am I?

- Freelance software developer
- Currently working mostly with clients in the entertainment industry, notably including the Casting Frontier

Author:
- Practical Ruby Gems
- Practical Reporting with Ruby and Rails
- Practical Rails Plugins (with Nick Plante)
What is Rails?

- Popular web development framework
- MVC paradigm – Model View Controller
- Agile, rapid development
Why are Rails apps Slow?

- Makes assumptions:
  - Selects all fields by default

- Limited knowledge:
  - can't preload without help.

- Ruby Oriented culture
  - ”SQL is bad” attitude

- Test environment different from production environment.

- Many more possible reasons.
How can I tell what's slow?
Rails development log

- Gross metrics: percentage of time spent in DB, view, etc
- Tools:
Ajax popup - displays query count, query time for each query on a page
mysql_slow_log

New Relic

- Awesome solution.
- Easy to install plugin reports performance and error data.
- Provides user response time reports, SQL query delay breakdown
- Has API for custom instrumentation.
- Per-server cost is relatively high, but worth it.
- (I have no financial interest in New Relic.)
Is it a database problem?

- Firebug
- Yslow
- Ping, tracert, and friends
Specific Rails Problems and Solutions
Do Sorting and Limiting in the Database

- Do this:
  - `@items = Item.find(:all, :order=>'name ASC')`
  - `@items.each do |item|`
  - `# do something...`

- ... not this:
  - `@items = Item.find(:all)`
  - `@items.sort_by(&:name).each do |item|`
  - `# do something...`

- EXCEPTION: If you need to sort data multiple ways in the same action.
N+1 query problem

- How many queries does this produce?
  - `<%@posts = Post.all
  - @posts.each do |p|%>
  - `<h1><%=p.category.name%></h1>
  - `<p><%=p.body%></p>
  - `<%end%>
Answer:

- one query plus one query per row in @posts. Not good.
Fix:

@posts = Post.find(:all, :include=>[:category])

Will generate at most two queries no matter how many rows in the posts table.
What if we want to do this?

```ruby
@posts.each do |p|
  <h1><%= p.category.name%></h1>
  <%= image_tag p.author.image.public_filename %>
  <p><%= p.body%>

Problem: how do we eager load nested relationships?
@posts = Post.find(:all, :include=>{
  :category=>[],
  :author=>{
    :image=>[]}})
How many queries does this produce?

```ruby
<% @user = User.find(5) %>
@user.posts.each do |p|%>
  <%= render :partial=>'/posts/summary', :locals=>:post=>p%>
<%end%>
```
<h1><%=post.user.name%></h1>
...snip...
Answer

- Produces an extra query per row in post looking up the users name.
  - (Rails does not, as of yet, associate child records with their parents.)
- Solution?
  - Self referential eager loading like this:
    - `@user = User.find(5, :include=>{:posts=>[:user]})`
Rails provides a series of grouping and aggregate functions for you

- all_ages = Person.find(:all, :group=>[:age])
- oldest_age = Person.calculate(:max, :age)

See pages 20-32 of Practical Reporting with Ruby and Rails

...but if the builtins aren't enough, use custom SQL.
Custom SQL with Rails

- \texttt{sql = "SELECT vehicle\_model, \text{AVG(age) as average\_age, AVG(accident\_count) AS average\_accident\_count \}}
- \texttt{FROM persons}
- \texttt{GROUP BY vehicle\_model"
- \texttt{Person.find\_by\_sql(sql).each do |row|}
- \texttt{puts "#{row.vehicle\_model}, "}
- \texttt{"avg. age: #{row.average\_age}, "}
- \texttt{"avg. accidents: #{row.average\_accident\_count}"
- \texttt{end}
- \texttt{RESULT:}
- Ford Explorer, avg. age: 43.010, avg. accidents: 0.400
- Honda CRX, avg. age: 18.720, avg. accidents: 1.25
Caching

- Important
- Simple out-of-box caching
- Has many relationships and counter caches
- Cache Fu
- MySQL triggers for DB function caching
- Rails triggers for other caching
Roll-your-own Rails Caching API

- Rails 2.1+
- Rails.cache.write('test_key', 'value')
- Rails.cache.read('test_key') #=> 'value'
- Rails.cache.delete('test_key')
Rails caching API Configuration

- Pluggable backends:
  - Memory
  - File
  - Drb
  - Memcache
  - Redis (via plugin)

- Configuration:

  config.cache_store = :mem_cache_store, 'localhost', '192.168.1.1:1001',
  { :namespace => 'test' }

  (from http://thewebfellas.com/blog/2008/6/9/rails-2-1-now-with-better-integrated)
Use Other Tools Where Appropriate

- Sphinx
- Memcached
- Redis
- Mongo
- Cassandra
- Client side javascript
  - Do you need another server roundtrip?