

Sharing The Load: managing tasks in Ruby app

Or: how to "outsource" long-running
(CPU-intensive) or precise tasks
from any Ruby

- including Rails, Erb, Wheels etc. -
application.

(and not shoot ourselves in the foot)

Tomek Stachewicz
Euruko 2008, Praha

About the author

- First steps in Ruby in late 2006, Rails developer since march 2007
- (OMG total n00b!)
- RoR programmer at Aenima, a Ruby-exclusive startup company from Poland
- First talk at a conference. Ever.

The sad truth

As Ruby/Rails developers we have been, are, or are going to be developing a social (or generally Web2.0*) site.

* Web2.0: "you make the content, we keep the revenue"

The sad truth pt. 2

Community-based sites pretty quickly (are going to) require:

- ability to add browser-viewable videos, or
- ability to have tasks that end at that one, precisely defined moment (auctions, contests, polls)

The sad truth pt. 3

Or anything else emerges that shouldn't block our application nor depend on its lifecycle.

The Problem

Find a way to handle such tasks by our Ruby application.

The Requirements

- **Long-running tasks** cannot block "main" application
- **Time-precise tasks** cannot be tied with "main" application lifecycle (e.g. restarting Mongrel)
- => has to be a separate, "stand-alone" application
- ...that has some means of interaction with "main" application
- It'd be great to have it framework-agnostic

BackgroundDrb

- A separate Drb server, controlled independently and outside Mongrels
- Runs every job in separate process, able to crunch many tasks concurrently
- so, why not use it?...

BackgroundDrb drawbacks

- not easy to install and setup
- Rails plugin (not framework agnostic)
- immediately runs given tasks;
workaround with using `sleep()`

Different (yet similar) Approach

Our own Drb server:

- easy to install and setup (there is none)
- elastic, customizable (100% own code)
- framework-agnostic - pure Ruby
- easier to implement than You may think

Yeah right...

but...

How do we implement concurrency and
running at declared time?

The Nightmare

...assuming we don't want to write our
own scheduler?

OpenWFeru (Scheduler)

- Ruby port of Java's OpenWFE (WorkFlow Engine), both by **John Mettreaux**
- Available as gem
- Runs every job in separate thread
- Keeps reference to pending jobs
- Can run job at: given time, time-from-now or cron-style
- Easy to use and intuitive

Installation and setup

- Might install full OpenWFeru gem or **just OpenWFeru Scheduler**
- 1: `gem install openwferu-scheduler`
- 2: `require 'openwferu'`
- 3: no step 3!

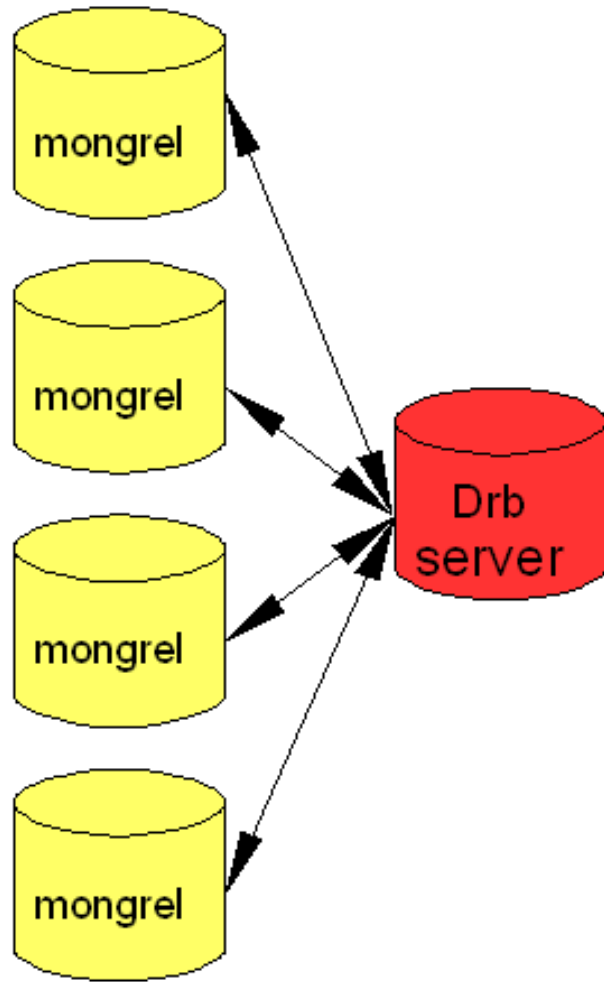
Conclusion

By writing our own Drb server using OpenWFERu Scheduler we are able to deal with both:

- CPU-intensive tasks
- tasks to be run at given moment

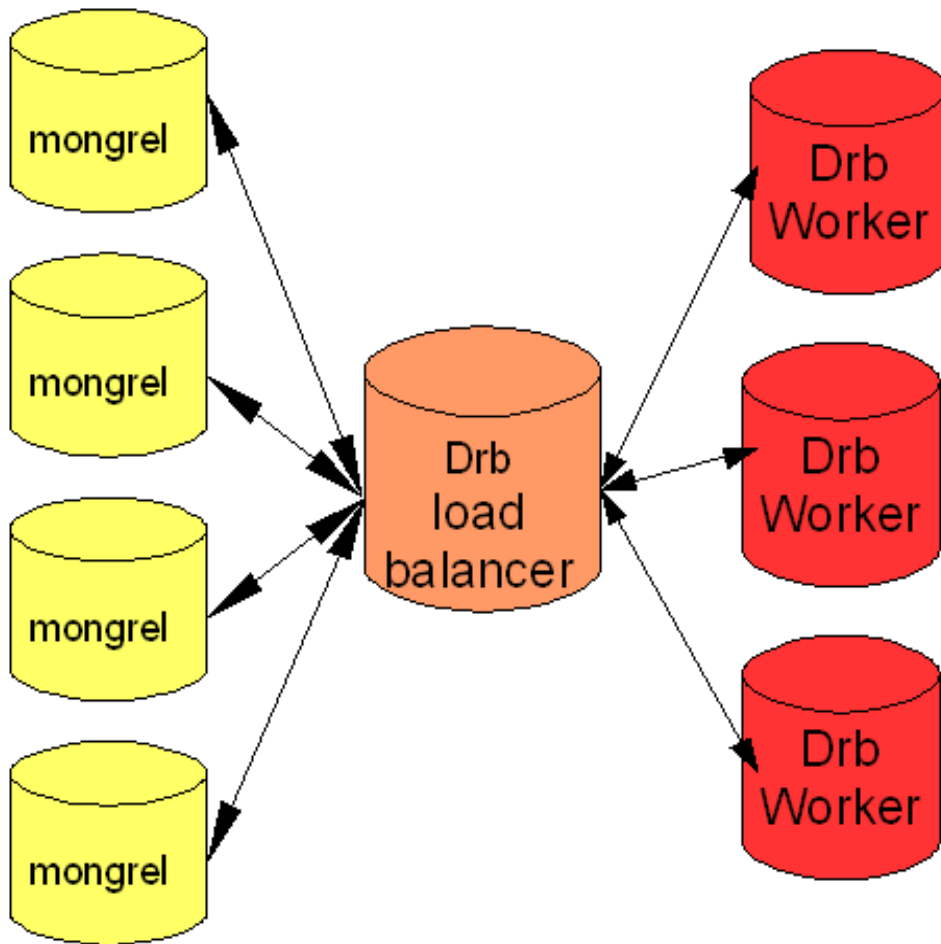
And still having a pure Ruby solution, usable from any application (regardless of framework, if any)

Chart for J2EE refugees



- Rejoice: our tasks are independent from (for example) mongrel instances
- And what if one Drb processing server ain't enough?

Scaling it up



- Multiple Drb servers for processing
- A middleman server-client, One Drb Script To Rule Them All

Let's see some

"Talk is cheap. Show me the code"
-- Linus Torvalds

Right about time

Thank You for your attention.

Discussion



Discussion conclusions

- Drb implementation in Ruby 1.8 is unreliable and buggy
- Even BackgroundDrb threw Drb away
- We should use different protocol
- (or wait for good Drb implementation in Ruby 2.0)