JRuby at ThoughtWorks

Ola Bini
JRuby Core Developer
obini@thoughtworks.com
About me

- Ola Bini
- From Gothenburg, Sweden
- Works for ThoughtWorks Studios in London
- Programming language geek (LISP, Io, Erlang, ML, Smalltalk, etc)
ThoughtWorks

- Global consulting firm
- About 900 people worldwide (UK, US, Canada, Oz, India, China)
- Known for Agile practices
- Martin Fowler is our Chief Scientist
- 40% projected revenue in the US from Ruby/Rails
ThoughtWorks Studios

- Product development
- CruiseControl Enterprise
- Mingle
- RubyWorks
  - CruiseControl.rb
  - Production stack
  - JRuby
Agenda

• Problems with Ruby
• JRuby
• Mingle
• Other uses
• Current problems
• Q&A
What's wrong with MRI

- Ruby 1.8: Green threading
  - Can’t scale across processors/cores
  - C libraries won’t/can’t yield to Ruby code (DNS)
  - One-size-fits-all scheduler - doesn’t really fit all
- Ruby 1.9: Native, non-parallel threads
  - Core classes and extensions not ready for parallel execution
  - Lots of work to ensure thread safety
What’s wrong with MRI

- Ruby 1.8: Partial Unicode
  - Internet-connected apps MUST have solid Unicode
  - Ruby provides partial, inconsistent support
  - App developers must roll their own: Rails Multibyte
- Ruby 1.9: Full Unicode but drastic changes
  - String interface changes to per-char, not per-byte
  - Each String can have it’s own encoding
What’s wrong with MRI

- Ruby 1.8: Slower than most languages
  - 1.8 is usually called “fast enough”
  - ... but routinely finishes last in benchmarks
  - ... and no plans to improve the situation in 1.8
- Ruby 1.9: Improvement, but still more to do
  - New engine averages 3-4x improvement
  - Only AOT - No JIT
  - More to do: GC and threading still slow
What's wrong with MRI

- Ruby 1.8: Memory management
  - Simple design
  - Good for many apps, but doesn’t scale
  - Stop-the-world GC
- Ruby 1.9: No change
  - Improved performance => more garbage
  - GC problems could well multiply
What’s wrong with MRI

- Ruby 1.8: C language extensions
  - C is difficult to write well
  - No encapsulation in core
  - Threading, GC issues
  - Portable, but often needs recompilation
  - No security restrictions
- Ruby 1.9: No change
What's wrong with MRI

- Politics
  - You want me to switch to what?
  - ... and it needs servers/software/training?
  - This will probably improve with time
- Legacy
  - Lots of Java apps in the world
  - Extensive library of Java frameworks/libraries
What is JRuby

- Java implementation of the Ruby language
- Current version 1.0.1, released in August
- Based on Ruby 1.8.5
- Started in 2001 by Jan-Arne Petersen
- Currently 6 Core developers
- Open Source - about 30-40 contributors
What can it do?

- All “pure Ruby” code works (with some caveats)
- Rake and RubyGems run well
- Rails works near nigh perfectly
- Many projects using JRuby+RSpec
- New combinations of JRuby+X popping up
What can’t it do?

- Deterministic threading
- Continuations
- Some file system operations
- fork, and other POSIX-ilk
JRuby solutions to MRI problems

- Native threading
- Scale across all processors/cores
- Concurrent execution, even in extensions
- Allow systems to schedule threads
- Ensure reasonable safety in core classes
JRuby solutions to MRI problems

- World-class, native Unicode support
- Provide Ruby’s byte[]-based String
- ... but also provide native Rails Multibyte backend
- ... and you can use the Java UTF-16 Strings directly
- ... and we’re working at implementing 1.9 Strings
- Java has complete, reliable Unicode
- ... and all libraries are Unicode-ready
JRuby solutions to MRI problems

- Scalable performance
- Make interpreter as fast as possible
  - Should be as fast as Ruby 1.8
- Support Ruby 1.9/2.0 bytecode engine
  - Same resulting performance boost
  - Future-proof
- Each new version of Java improves JRuby performance substantially
JRuby solutions to MRI problems

- Compile to Java bytecodes
- AOT and JIT compilation
- Let HotSpot take over
  - ... by simplifying
    - HotSpot JIT
    - Code inlining
    - Dynamic optimizations
JRuby solutions to MRI problems

- Let Java manage memory
- Best memory management and GC in the world
- Wide variety of GC options
  - Concurrent
  - Generational
  - Real-time
- Scales up to enormous apps and systems
JRuby solutions to MRI problems

- Java-based extensions
- Easier to write than C
- Truly portable: WORA
- Clean separation between core and extensions
- No GC, threading or security issues
- Easier to expose Java libraries
JRuby solutions to MRI problems

- Politics don’t get in the way
  - JRuby is “just another Java library”
  - Minimal impact dev, admin processes
  - Over ten years of mainstream Java
- Legacy integrates just fine
  - Use existing services and libraries
Why ThoughtWorks likes JRuby

- JRuby gives access to the “enterprise” features of Java
- Conservative environments will not use MRI
- Quick integration with legacy systems
- Cost: Java+Ruby is more cost effective than MRI
Why JRuby on Rails for TW?

- Deployment, deployment, deployment
- JDBC for database access
- Other libraries that provide needed, cross-platform functionality (Java2D instead of RMagick?)
- Management (JMX and others)
- Common to do JRuby on Rails applications that work with legacy data
Mingle

- Team Collaboration Tool
- First commercial JRuby on Rails application
- Originally choose JRuby for SVN plugin
- Originally developed in MRI - still MRI compatible
- Very well tested
- Mingle test suite is slower in JRuby than MRI
- ... but in production the JRuby version is quicker and scales better
## Mingle stats

<table>
<thead>
<tr>
<th>Name</th>
<th>Lines</th>
<th>LOC</th>
<th>Classes</th>
<th>Methods</th>
<th>M/C</th>
<th>LOC/M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controllers</td>
<td>2809</td>
<td>2377</td>
<td>32</td>
<td>276</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>Helpers</td>
<td>1255</td>
<td>1038</td>
<td>8</td>
<td>138</td>
<td>17</td>
<td>5</td>
</tr>
<tr>
<td>Models</td>
<td>11203</td>
<td>9079</td>
<td>176</td>
<td>1564</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>Libraries</td>
<td>4784</td>
<td>3919</td>
<td>88</td>
<td>383</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>Integration tests</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Functional tests</td>
<td>3494</td>
<td>2881</td>
<td>27</td>
<td>337</td>
<td>12</td>
<td>6</td>
</tr>
<tr>
<td>Unit tests</td>
<td>16109</td>
<td>13272</td>
<td>105</td>
<td>1449</td>
<td>13</td>
<td>7</td>
</tr>
<tr>
<td>Acceptance tests</td>
<td>13318</td>
<td>10689</td>
<td>87</td>
<td>1100</td>
<td>12</td>
<td>7</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>52972</td>
<td>43255</td>
<td>523</td>
<td>5247</td>
<td>10</td>
<td>6</td>
</tr>
</tbody>
</table>

Code LOC: 16413  Test LOC: 26842  Code to Test Ratio: 1:1.6
Mingle license decryption

- Licenses uses strong cryptography
- Using a Java RSA library
- Would have been very hard in MRI
- JRuby Java integration made it dead simple to use the RSA library
Obfuscation

• Override JRuby's LoadService
• This allows us to encrypt/decrypt all Ruby files in app/*
• Will probably move to using AOT compilation when that is finished
• This is obfuscation - there is no real, safe protection in it
• ... but it seems to work well enough. =)
Mingle + ChartDirector

- ChartDirector is a proprietary library for making charts
- Have both a Java library and a C extension library for Ruby
- A thin wrapper over the Java library makes it possible to use the same chart code in MRI and JRuby
Mingle code example: chartdirector4jr.rb
Mingle + SVNkit

- No MRI subversion library worked on all platforms
- Initial reason for going with JRuby
- SVNkit is a Java library that provides uniform subversion access on Linux, MacOS X, Windows, Solaris, and all other Java platforms
- SVNkit supports file system, DAV and SVN
Mingle deployment

- install4j - installation and bundling of JVM
- Runs locally using Jetty
- BYO database - (used to be Derby)
- 1.1 or 1.2 will support WAR deployment
- Uses custom built Jetty launching
- Uses custom built AntBuilder scripts to generate WAR
- Uses CC.rb for MRI and Java CC for JRuby
Finance client 1

- Conservative, large company
- IS department standardized on Java
- JRuby on Rails application
- JRuby improves integration
- Uses the UnitRecord plugin to speed-up test suites
- Lack of good RMagick replacement have been hard
- Tomcat+Lucene seems to be causing bugs
Finance client 2

- Large, conservative company
- Separate business units: most information is not shared
- Project aims to consolidate all accounts and customer information into one place
- It’s written in mainly Ruby with small amounts of Java
- 2 persons, probably 4 months from start to finish
- Interacting with 5 disparate data sources
Finance client 2 - why JRuby?

- Infrastructure barrier was primary reason
  - “Those infrastructure guys are just seeing another Java app with some static textual content (happens to be Ruby source files)”

- Higher business value by directly lowering integration estimates

- Using Java APIs (especially JDBC) allowed quicker development, since no need to write new functionality for Ruby
Waffle

- Java web framework
- No XML - except minimal web.xml
- Easy to learn
- No base classes/interfaces needed
- Allows most functionality to work with Ruby
- ... ERb templates can be used as views
- Ruby classes as controllers
Forthcoming Studios projects

- Other tools for development/team collaboration
- Will use JRuby in different ways
- ... including allow Ruby to be used as extension language for Java based applications
- ... and improving the deployment and management story for JRuby
- Next product will probably be GA in Jan/Feb ‘08
Challenges with JRuby

- Performance of unit tests
  - Solution: running MRI precommit and JRuby in CI
  - Solution: using UnitRecord instead
- It’s not free to run on both MRI and JRuby
- Start-up time (especially of Rails)
  - Solution: staged start-up in background
  - Not always enough for day-to-day development use
Challenges with JRuby

- JRuby regular expressions have different performance characteristics and big-O running time in certain cases
- JRuby YAML isn’t completely stable yet (but it’s getting there)
- High memory consumption (but still lower than Mongrel)
- Good replacement for RMagick needed
- Lack of documentation
The future of JRuby (at TW)

- Continue looking at products around JRuby
- JRubyWorks
  - Alleviate current problems with GoldSpike
- ActiveHibernate?
- Performance
- Other neat things
Resources

- www.jruby.org
- studios.thoughtworks.com
- waffle.codehaus.org
- jruby-extras.rubyforge.org
- ola-bini.blogspot.com
- JRuby mailing lists at Codehaus
Shameless plug

24 September ‘07

Practical JRuby on Rails
Web 2.0 Projects

Bringing Ruby on Rails to Java

Ola Bini

Apress