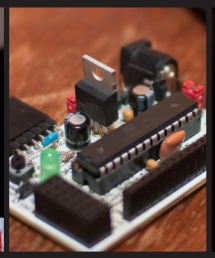
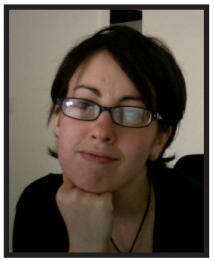
the

Golden Gate Ruby Wrap




Content from Golden Gate RubyConf, the first Ruby show in the birthplace of the web revolution.

About the Organizers



LEAH SILBER

If you've been to a Ruby conference in the last year or two, you've likely met Leah. When she's not traveling the world seeking out the perfect conference, she's here in San Francisco running Community Relations and Marketing at Engine Yard.

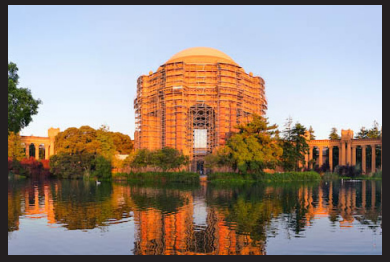
JOSH SUSSER

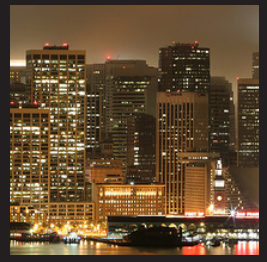
Josh likes to say that Ruby is the love child of Lisp and Smalltalk, raised by Perl the eccentric nanny. At Pivotal Labs he spends most of his time developing Rails web applications for clients, and is a frequent contributor to Rails. You can find his blog at blog.hasmanythrough.com.

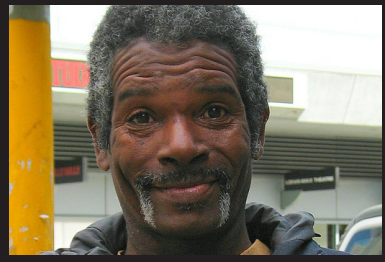
About The City

San Francisco is the home of, of course, landmarks like the Golden Gate Bridge. More importantly, it's a hub of technological advancement and experimentation.

The city of San Francisco is filled to the brim with well known and hush hush Ruby and Rails shops, making a dent in the technological ecosystem in Silicon Valley. User Groups, Meetups, Hackfests -- we've got it all. If you haven't been, do make a point to visit. And say hello while you're here!







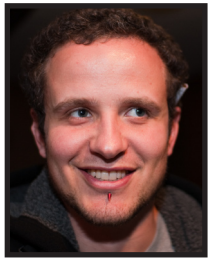
About the Wrap

The *Golden Gate Ruby Wrap* intends to capture some of the content from the conference, for those of you who missed it, or those of you who need a reminder.

Thanks to the speakers, attendees and writers for their contributions, and to the sponsors who helped fund everything.

Event photographs are courtesy of our event photographer, Andy Delcambre.

For questions, comments or corrections email wrap@gogaruco.com.



About the People

While the speakers contribute the most to a conference publicly, the attendees make a show what it is. That said, they don't generally get enough attention!

That's why we've decided to focus some attention on *them*. The *Golden Gate Ruby Wrap* features several contributions and opinions from attendees.

It also features some results from our attendee survey. Check them out on page 18.

A Message from the Organizers

Organizing Golden Gate RubyConf has been challenging, time-consuming, and best of all, rewarding.

We set out to make a contribution to the Ruby conference circuit, and to the community in general, and have surpassed even our own ideas of how incredible things would be. The speakers were great, the volunteers and sponsors were great, and of course, the attendees.

We couldn't have done this without every one of you, and hope you'll join us again next year for a bigger and better Version 2.0.

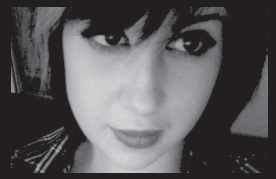
Leah and Josh

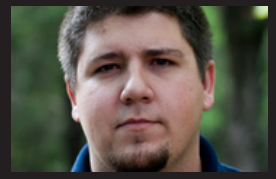
A Shoutout to Some Friends

Thanks to the volunteers who were with us from the beginning!

- Melissa Sheehan
- Bruce Williams
- Geoffrey Grosenbach
- Meghann Millard

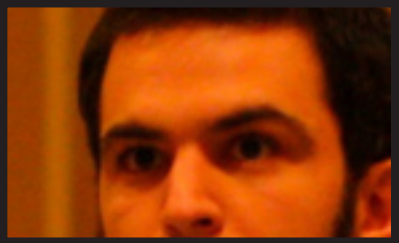
And thanks, of course, to all those volunteers who helped on the day-of - see page 9 for the full list.





Meet the People that Made Golden Gate RubyConf the Quality Show It Was

A conference isn't a conference without the speakers, and our collection of Ruby luminaries was spectacular. Half the speakers on the first half of the Golden Gate RubyConf roster were invited by the organizers. These are people we'd seen speak before, and knew would deliver quality content. The second half of the roster was selected by attendees voting on submissions solicited from the public. They all delivered, and we owe them our thanks.



GREG BORENSTEIN

Greg Borenstein is a programmer and musician in Portland, Oregon. He is the author of a number of open source projects including RAD, a library for programming the Arduino open source hardware platform in Ruby.
In his spare time, Greg organizes PDX Pop Now!, a free all-ages local music festival and non-profit organization, and blogs about Ruby, art, and hardware hacking on Urban Honking. Ruby was the first programming language he ever learned.

SESSIONS
Ruby Application Frameworks Discussion Panel
Arduino is Rails for Hardware Hacking

TIM ELLIOTT

Tim immediately became enamored with Hackety Hack and its cousin Shoes when they were released, and he started contributing to Shoes in late 2008. He has been a programmer for Travidia Inc. since 2005, and shares a 40-acre farm with 10 roommates, 80 chickens, and two sheep in Chico, California.

SESSIONS
Ruby Application Frameworks Discussion Panel
Using Shoes to Create Better iPhone Apps

YEHUDA KATZ

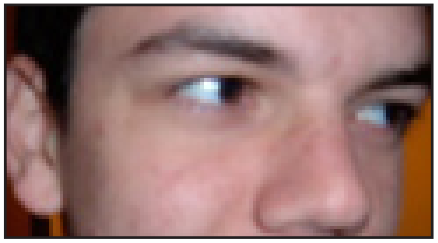
Yehuda is currently employed by Engine Yard, and works full time as a Core Team Member on the Rails and Merb projects. He is the co-author of jQuery in Action and the upcoming Rails 3 in Action, and is a contributor to Ruby in Practice. He spends most of his time hacking on Rails and Merb, but is also active on other Ruby community projects like Rubinius and DataMapper. And when the solution doesn't yet exist, he'll try his hand at creating one — as such, he's also created projects like Thor and DO.rb.

SESSIONS
Ruby Application Frameworks Discussion Panel

BLAKE MIZERANY

Blake has been into Ruby since way back in 2001 and is the creator of Sinatra, the popular Ruby microframework. Blake spends his days at Heroku, where he makes mind-blowing features out of Ruby and Erlang, and often says "you're doing it all wrong". He regularly speaks at Ruby events and in conjunction maintains a completely inexplicable beard-shaving schedule.

SESSIONS
Ruby Application Frameworks Discussion Panel



JOSH PEEK

Josh Peek is a 20 year old Rubyist from Chicago, IL. Currently attending college at DePaul University, he codes part time at 37signals and still makes time to hack on other side projects. Josh joined the Rails core after his work on Rails thread-safety and is now working on full Rack support for Rails.

SESSIONS
Ruby Application Frameworks Discussion Panel

JAY PHILLIPS

Jay Phillips is an innovator in the spaces where sophisticated VoIP development falls apart and where Ruby rocks. As the creator of Adhearsion and its parent consulting company, Jay brings new possibilities to these two technologies through his work on the open-source Adhearsion framework.

SESSIONS
Ruby Application Frameworks Discussion Panel

Starting With Our Framework Panelists

MATT AIMONETTI

You might recognize Matt Aimonetti from movies like "slim & sexy Merb," "DataMapper and its multiple repositories", "Rails Activism, keeping the community involved", and more recently: "MacRuby: when Matz meets Steve Job" as well as "CouchRest: Ruby all the way."

SESSION
CouchDB + Ruby: Perform Like a Pr0n Star

RUSTY BURCHFIELD

Rusty Burchfield is a Software Engineer at Zvents. Rusty is a problem solver who works all over the software stack using technologies like Rails, Cascading (framework over Hadoop), and Hypertable.

SESSION
Hypertable and Rails: DB
Scaling Solutions with HyperRecord

HAMPTON CATLIN

Hampton Catlin is Mobile Development Lead at the Wikimedia Foundation, which runs Wikipedia. He is also the inventor of Haml and Sass. Most recently he has been spending his time introducing Ruby to everyone's favorite online Encyclopedia by launching the new mobile platform in 100% pure, delicious Ruby. Hampton lives in Florida.

SESSION
There Will Be Ruby!

JON CROSBY

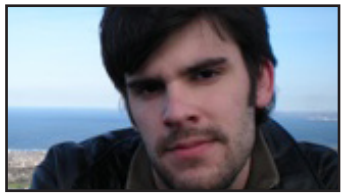
Jon Crosby is a San Francisco Bay Area developer specializing in Ruby, JavaScript, Objective-C, and Open Web technologies. Jon is the author of CloudKit and a committer on several open source projects including rack-contrib and the OAuth ruby gem. He is an active member of the Open Web community and is currently employed by Engine Yard.

SESSION
CloudKit: Hacking the Open Stack with Ruby and Rack

ILYA GRIGORIK

Ilya Grigorik is the founder and CTO of AideRSS. He has been active in the Ruby and cloud computing community for the last three years, documenting and sharing hands on knowledge and experience with the latest architecture, design patterns, and open-source projects (blog: www.igvita.com, twitter: @igrigorik).

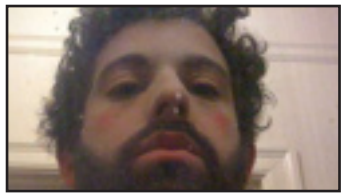
SESSION
Ruby Proxies for Scale, Performance, and Monitoring



BRYAN HELMKAMP

Bryan Helmkamp is the lead maintainer of Webrat, a library to implement acceptance testing of a Ruby web application in a more expressive and maintainable way. He's a co-author of The RSpec Book, which is now available as a beta PDF. Following three years of Ruby on Rails consulting, Bryan is now a software engineer at Weplay, a New York City-based startup.

SESSION
Webrat: Rails Acceptance Testing Evolved



NICK KALLEN

Nick Kallen is a Systems Architect at Twitter, where he focuses on scaling, fault-tolerance, and agile development. He is the author of popular open-source projects such as NamedScope, Screw.Unit, and Cache-Money. Follow Nick on Twitter: @nk

SESSION
Magic Scaling Sprinkles

CARL LERCHE

Carl started building web applications at 13 with PHP. He is a software engineer at Engine Yard, a member of the Merb team (currently working on Rails 3), and a contributor to many OSS projects. He plans to build a web server in haskell, ocaml, or ____ in his spare time for no tangible reason.

SESSION
Writing Fast Ruby: Learning from Merb and Rails 3

AARON QUINT

Aaron Quint is a web developer based in Brooklyn, NY. Working with Ruby and Ruby on Rails he has launched over 25 different web applications in the last two years. Recently he's been releasing open source projects at code.quirkey.com, including the Sinatra skeleton generator sinatra-gen.

SESSION
Magic Scaling Sprinkles

RICH KILMER

Richard Kilmer is the founder of InfoEther, Inc and is a board member of Ruby Central. Rich's background includes peer-to-peer software, wireless web, workflow, and pen computing. Rich's current Ruby efforts are focused on simplifying OS X development with HotCocoa and is a contributor to the MacRuby project.

SESSION
MacRuby and HotCocoa

JACQUI MAHER

Jacqui is a programmer at Hashrocket, and has been a professional programmer since 1998, specializing in Ruby, JavaScript, Perl, Java, and PHP web development with a solid foundation in design, including CSS and HTML. She contributes to several open source projects with a special emphasis on public health.

SESSION
Using Ruby to Fight AIDS

NATHAN SOBO

Nathan Sobo is the author of Treetop, a packrat parsing framework for Ruby. After two years practicing extreme programming at Pivotal Labs, he now directs engineering at Grockit, where the challenge of supporting rich synchronous interaction on the web is his primary source of technical inspiration.

SESSION
Unison: A Relational Modeling Framework

GREGORY MILLER

Gregory Miller is Chief Development Officer of the Open Source Digital Voting Foundation. Greg is also a (non-practicing) IP lawyer involved in technology public policy. He is dedicated to restoring trust in elections through open source, open data, open process, and open standards voting technology.

SESSION

Trust the Vote: An Open Source Digital Public Works Project

DAVID STEVENSON

David Stevenson has been working in Ruby for over three years. He's contributed many Ruby gems and Rails plugins/patches, most notably the MySQL QueryReviewer plugin. He loves building complex applications that perform and scale well, and also loves to tinker with weird side projects. These days he can be found playing ping-pong at Pivotal Labs.

SESSION

Playing With Fire: Running Uploaded Ruby Code in a Sandbox

JOSH TYLER

Josh Tyler manages the Front-end Engineering team at Zvents, where he focuses on user experience design and building clean software. Josh formerly worked at Xerox PARC and HP Labs, and has been programming in Ruby for several years.

SESSION

Hypertable and Rails: DB Scaling Solutions with HyperRecord

TABLE OF CONTENTS

VOLUNTEERS

DENNIS COLLINSON

SAMER ABUKHEIT

TIM RAND

NOT SHOWN:

ANDY DELCAMBRE

WILL EMERSON

ROB FULLEN

MEGHANN MILLARD

MELISSA SHEEHAN

SCOTT THORPE

BRUCE WILLIAMS

Business-Ready Rails »

Rails is **the framework of choice** for smart developers,
now Rails is also **the intelligent choice for businesses**

Engine Yard™ makes it even easier to leverage the power, speed and scalability of the Ruby on Rails framework. Our suite of Rails development, deployment and management tools enables businesses to run their Rails applications with ease.

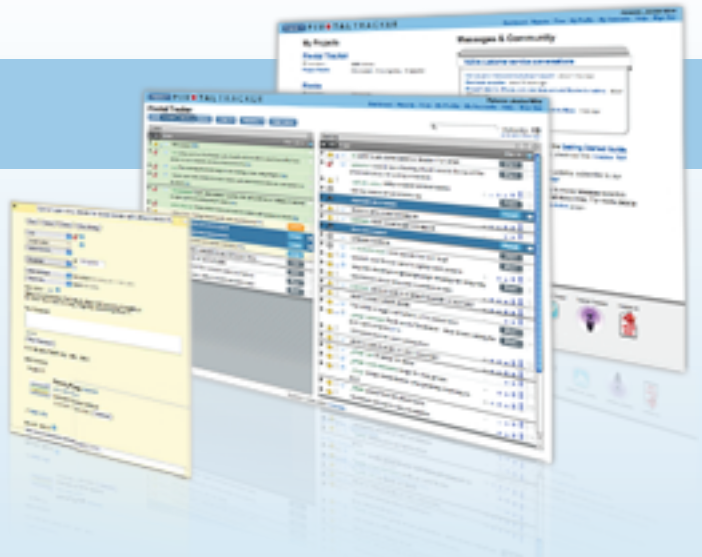
Whether you are an experienced Rails development shop, a startup or an enterprise adopting Rails for your next project, there is no better time to get your application on the Engine Yard platform.

Call us today to get started » **(866) 518-YARD (9273)**
Or visit us online » **www.engineyard.com**

Introducing Engine Yard Solo™

The preferred platform for on-demand management of your Ruby on Rails application in the cloud. See what developers are saying about Solo at www.engineyard.com/solo.

PIVOTAL TRACKER



Tracker is a story-based project planning tool that allows teams to collaborate in real-time.

Velocity tracking and emergent iterations

Make planning decisions using accurate projections based on past performance.

Story-based iterative planning

Base your software project management on proven agile methods.

Real-time collaboration

See what your team is doing and react to change instantly.

"I've waited 15 years for a computer program that out-performs index cards. This product had to come from a team that understands how I work."

Ward Cunningham, CTO, AboutUs.org

"We are using Pivotal Tracker to manage all of our new web apps under development. This thing rocks!"

Ezra Zygmuntowicz, Founder and Senior Fellow, Engine Yard

"Use Pivotal Tracker. Forget the bug tracker or issue tool you're using now. This is an iPod. You're using a Zune. You can run your whole company on it. And you probably should."

Nivi, Venture Hacks

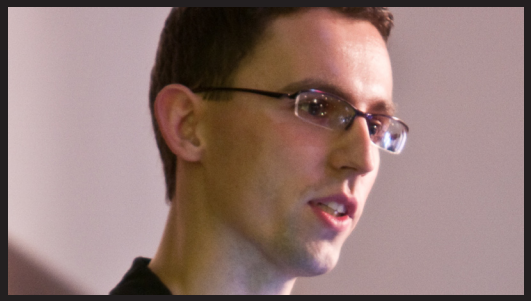
PIVOTAL LABS

Pivotal Labs can take a vision from the back of a napkin to an industrial strength application in as little as a few months. We impart sustainable development practices that free our clients from dependence on our services and enable them to scale their business to meet demand.

Sign up at: **https://www.pivotaltracker.com/signup/new/gogaruco**

2009 Jolt Award Winner
Project Management

Meet Ilya Grigorik



STATS

PLACE OF BIRTH: Minsk, Belarus
CURRENT LOCATION: Waterloo, ON, Canada
YEARS PROGRAMMING: 11
YEARS PROGRAMMING RUBY: 4

ONLINE HEADQUARTERS

WEBSITE: www.igvita.com
BLOG: www.igvita.com
TWITTER: @igrigorik

TOP OPEN SOURCE PROJECTS

NAME: EM-HTTP-Request
REPO: github.com/igrigorik/em-http-request/tree/master

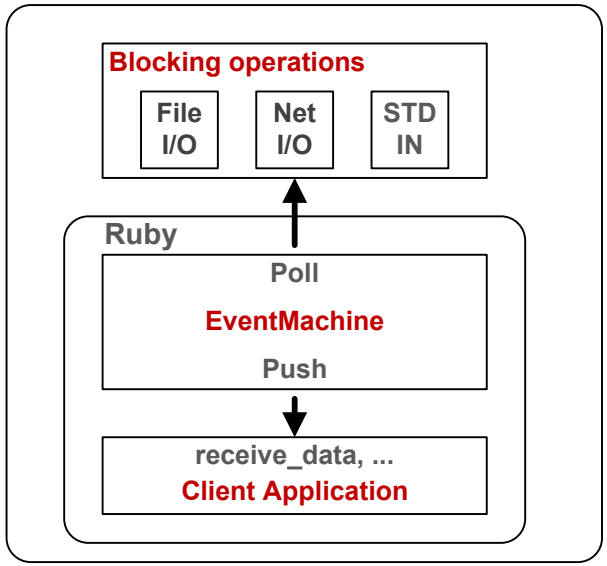
NAME: Bloomfilter
REPO: github.com/igrigorik/bloomfilter/tree/master

Building Custom Web Proxies in Ruby:
Using the EventMachine Library

By Ilya Grigorik

The definition of a high-performance application is highly context dependent: a responsive UI, ability to support multiple users, working with large datasets and the list go on. However, when we think about network based applications, such as a web-server, there are two metrics you should be paying attention to: latency, or time to respond, and the throughput, or number of concurrent clients you can support. The latter is usually limited to the application logic; time to respond is dependent on how fast your code executes and can have a noticeable impact on the user experience (you should aim for sub 250ms response time, as Flickr and Google UX teams have found).

Throughput on the other hand is often harder to quantify, measure, and scale as it is often the realm of the underlying frameworks, database bottlenecks, and other blocking operations. If you have ever had to write a non-blocking web server, you will undoubtedly have a good selection of war stories. Thankfully, the EventMachine library provides us with a framework to simplify this task.



Built on the "Reactor pattern", EventMachine provides an easy to use interface for working with event-driven I/O. Instead of using threads to parallelize the application (for example, support two concurrent clients), it takes care of the polling and listening, and allows our client to handle simple events such as: connection established, data recieved and connection terminated.

Due to this pattern, even though the application is single threaded, we can still service many concurrent requests – aka, optimize throughput. Most of the time any given connection is 'blocked' or sleeping / waiting for data, and instead of suspending our server while we wait, we are able to process other incoming requests.

Like any other framework, EventMachine exposes an array of API's for building on the reactor pattern. However, for the purposes of building a simple proxy server in Ruby, all we need is three methods: #receive_data (new data ready for processing), #unbind (connection terminated), and for clients, #connection_completed (TCP handshake completed, connection ready for use). For a simple ping-pong client/server example, see exhibit 2.

Extending our ping-pong example, it is not hard to arrive at an implementation of a simple proxy server written in Ruby – we just have to combine the server and client code into one application. By doing so, we gain full control over the data, the routing and filtering process (Exhibit 3).

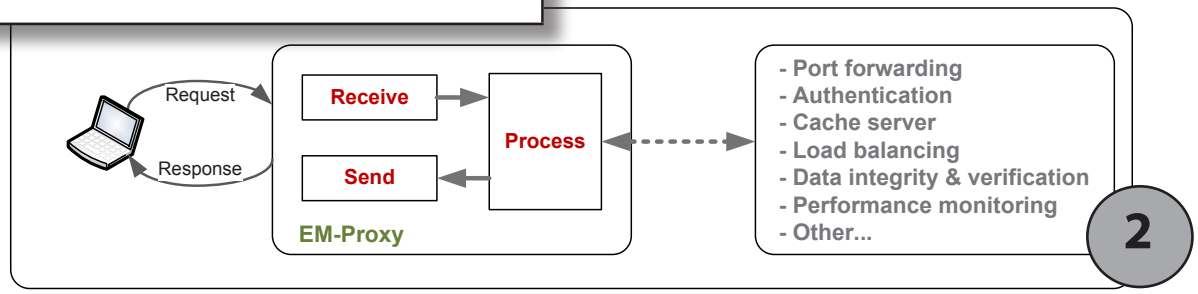
Most web developers are intimately familiar with the concept of a 'transparent proxy' as we use them extensively for load balancing and request routing between multiple application servers (ex: nginx, haproxy, perball, etc.) In these scenarios the client is not aware of the extra hop through the proxy, nor can it detect it , even if it wanted

EventMachine Client (Ping)

```
require 'eventmachine'
class Client < EM::Connection
  def connection_completed
    send_data("Ping!")
  end
  def receive_data(data)
    p [:received, data]
    EM.stop
  end
end
EM.run do
  EM.start_server "0.0.0.0", 3000, Server
end
```

EventMachine Server (Pong)

```
require 'eventmachine'
class Server < EM::Connection
  def receive_data(data)
    send_data("Pong: #{data}")
  end
  def unbind
    p [:connection_terminated]
  end
end
EM.run do
  EM.start_server "0.0.0.0", 3000, Server
end
```



to. Coupled with an intelligent load-balancing, load-monitoring and health check system, this is an essential tool for any non-trivial web application.

However, an 'intercepting proxy' is also an incredibly powerful tool when used in the right context. Instead of just routing the request, we can alter the server's response, server as a cache, or even extend the protocol on the fly! Conceptually, this is not unlike injecting a Rack middleware into our processing stack, except that the uses cases are not limited to Ruby-based web-services.

Intrigued? Navigate to em-proxy repo on GitHub to get started or to take a look at the sample applications: port forwarding server, A/B performance monitoring server and an example of extending a simple text-based Beanstalkd protocol.

What the People Thought

MICHAEL HARTL

"The combination of clear presentation style and technical depth was great. It made deeply technical material accessible, even to normal people like me.

"I loved how he started with simpler examples and then built up. He took something that seemed like it had limited use, and then showed how it could be applied to a whole bunch of situations. It was quite exciting.

"I also liked that it was based on real examples, and on the needs of a real website."

ATTENDEE INFO

Using Ruby Since: Winter 2005
Employed By: Self-employed
Blog: http://blog.mhartl.com
Twitter: @mhartl

Gregory Miller

The TrustTheVote™ Project:
Towards Trustworthy Critical Democracy Infrastructure

By Gregory Miller

A VERY BRIEF HISTORY OF THE VOTE

There was a time when it all just seemed to work. Ballots were cast and counted. And although we might not like the outcome of an election, there was a presumption of validity and respect for the process. While history is full of incidents of election irregularities, there were few concerns of widespread misbehavior or outright fraud. The 2000 Presidential election brought an end to that innocence. A nation, which until that point had held itself out to be the global pillar of democracy, was faced with doubt in the integrity of its own electoral system and the sanctity of the vote. Elections would never again be presumed certain in outcome.

Volumes have been prepared about the American election experience, but a few points are worth noting. First, the "right to vote" is not per-se guaranteed by the U.S. constitution. Our right to vote – if any – is granted by the State of our residence. Arguably, there is an implied right to vote given legislative initiatives over the years such as the 26th Amendment, the 1965 Voting Rights Act, and the 1993 National Voter Registration Act. However, our system of elections is, by design, highly distributed across the States and territories. Because the Constitution gives States the job of running elections, voting in America has developed into a patchwork of manual, mechanical, and electronic balloting. What's more, conducting U.S. elections relies mostly on a volunteer effort. And thanks in large part to the debacle of the 2000 Presidential election, information technology is now a centerpiece of American voting systems.

A DIGITAL SHIFT

Pundits from politics, sociology, and other corners of study theorize that one of the reasons the 2000 election, and nearly every election since, was so close is a combination of increasingly ubiquitous access to information and a steady increase in citizen interest in politics. Indeed, the rise of the Internet has played a critical role in making information about elected officials' records, politics, and campaigns more available than ever. This translates into a better informed electorate, and some suggest, a higher turn-out in voter participation. Regardless, the shift to online access and sharing of information is driving a new "digital democracy."

Meanwhile, there has been another digital shift: increasing use of computers in voting – from casting to counting, and all aspects of elections. This digital shift – from information availability to conducting elections – has been catalyzed in part by efforts of the Bush Administration to resolve the problems of the 2000 election through the Help America Vote Act of 2002 or "HAVA." Among other provisions, HAVA made matching dollars available to States that provide for a digital means to capture and count ballots. This became the "full employment act" for any vendor who made a computerized election device. The troubles began when vendors did their best to produce reliable voting machines, but due to costs, chose to base solutions on commodity PC technology.

The results are well documented. The very solution that was supposed to lead us away from the hanging chad has actually dissolved trust in computerized systems which cast and count votes. Irregularities abound, and can even invite opportunity for fraud far easier than in the days of punched paper ballots. And efforts to audit this machinery have been met with the commercial realities of proprietary protectionism.

AN IMPERATIVE CAUSE

The cornerstone of our democracy is the vote. In a digital age that cornerstone is made of hardware and

software. The free market enterprise experiment brought about by HAVA has essentially failed. No longer can proprietary technology deliver the most vital process of our democracy. Its time to shift away from black box voting, and move toward glass box voting. And we must pull together the nation's wealth of innovative capability to do so.

The time is now. It is a huge mistake to assume the status quo will carry us forward. Although the 2008 elections appeared to have been successfully conducted, experts unanimously agree that we merely "dodged a bullet" from experiencing a repeat of the 2000 debacle. To the point: 160 days after the general election, the State of Minnesota still had an undecided senate seat. The new President, as part of his platform for change, wants to rethink government information systems. A critical part of that challenge is re-inventing voting systems. And in this digital age, voting systems have become "critical democracy infrastructure."

Voting systems are so vital to the preservation of our democracy that they must be considered critical infrastructure. Their trustworthiness is essential; their reliability must be fault tolerant, and their accountability must be completely transparent. Accordingly, it must also be recognized that our elections systems infrastructure can no longer remain as a privatized service. We know the marketplace for commercial voting systems is deplorable: four major vendors, huge barriers to new entrants, no incentives to innovate, and yet accountability, trustworthiness, and transparency are mandates, not features. The only way to ensure this infrastructure is properly built and maintained to preserve our democracy is to place the underlying technology into a public digital works project trust. This is the imperative cause.

A TRANSPARENCY MANDATE

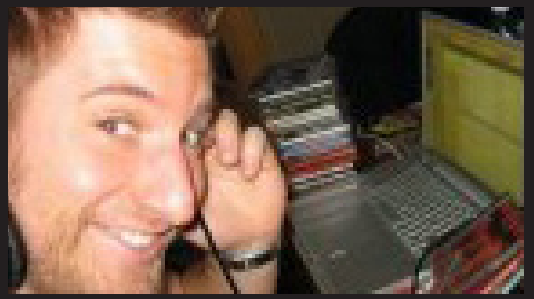
A voting technology repository held in the public trust can ensure transparency. Today's agile development processes and open source methods offer a way forward. This mandate requires accountability throughout technology lifecycle: design, development, implementation, certification, deployment, maintenance, and enhancement. Such is not commercially feasible for the private sector. And we cannot expect the government to do this either, for we run the risk of such a project twisting in the political winds of budget appropriations. In order for this critical democracy infrastructure to be accountable, reliable, and trustworthy it must become public open source.

A SOLUTION

Restoring trust in how voting technology sustains our democracy starts with you. To make this possible, the Open Source Digital Voting Foundation – backed by some renowned philanthropists – is shepherding the TrustTheVote Project and creating a public voting technology trust.

This is a digital public works project calling upon all of us who are passionate about our democracy and have the desire to be the change necessary to ensure that we remain a pillar of democracy with sanctity in the vote. This project needs the creativity and capabilities of many. So if you believe that together we can leverage our know-how to apply open source methods and agile development processes to a democracy project of imperative importance – how America votes in a digital age, then I encourage you to join the movement. Start today by visiting: www.trustthevote.org and www.osdv.org.

What the People Thought



PIOTR BIEGAJ

"You hear a lot about how electronic voting is a fundamentally flawed notion, and Greg made it suddenly seem like a really viable idea. Just because software is free or open, doesn't make it flawed or insecure, and that's an important thing to realize when thinking about the options for electronic voting.

"He gave a really solid talk that was well delivered. I liked that he used some video, which helped draw the audience in."

ATTENDEE INFO

Using Ruby Since: Winter 2008
Employed By: Engine Yard
Twitter: @pbiegaj

Meet Rusty Burchfield

STATS

PLACE OF BIRTH: Buffalo, NY
CURRENT LOCATION: San Francisco, CA
YEARS PROGRAMMING: 10
YEARS PROGRAMMING RUBY: 3

ONLINE HEADQUARTERS

WEBSITE: http://www.linkedin.com/in/gicode

TOP OPEN SOURCE PROJECTS

NAME: HyperRecord
REPO: github.com/tylerkovacs/hyper_record

NAME: Hypertable
REPO: github.com/nuggetwheat/hypertable

NAME: Rusty's Raytracer
REPO: github.com/GICodeWarrior/raytracer

Hypertable at Zvents:
Practical Experiences in a Live Product

By Rusty Burchfield and Josh Tyler

Hypertable is rapidly approaching its official beta release. The platform is stable and as the first and primary sponsor, our site (http://zvents.com) relies on it. Here we introduce Hypertable and describe our efforts to make Hypertable a part of our product, what worked and what did not, where we are today, and where we foresee heading in the future.

WHAT IS HYPERTABLE

Hypertable, an implementation of BigTable, is a sparse, distributed, persistent, multi-dimensional, sorted map. Due to its distributed capabilities, applications built on top of it are able to scale where applications built on top of a traditional RDBMS falter.

In order to support distribution, fast reads and fast writes, Hypertable's data model is different from that of an RDBMS. Columns are defined as column "families", meaning each column may have many member columns, often referred to as qualified columns.

Rows are sparse: if no data is specified for a given column, no data is stored. Each row is actually a collection of cells. Each cell is referenced internally by a 5-part key consisting of the row key, column family, column qualifier, timestamp, and revision. Given this key structure, Hypertable can be used to model up to 5 dimensions of information while functioning as a sorted map.

Sorted ranges of rows are distributed across the cluster to individual range servers. Range servers are coordinated by the master. When retrieving data, clients automatically contact the correct range server based on information from the master.

All of these features come together to form a powerful tool for building scalable applications. In the following sections we will describe our progress in doing just that.

ANALYTICS

Our first application for Hypertable was log analysis. We defined a tab separated log format including latencies, request data, response data, and internal state of the application. One such line is written per Rails request. The data is collected hourly and batch loaded into Hypertable.

In order to use this data in our Rails application, we needed to connect to Hypertable from Ruby. We developed a native binding using Rice, a C++ interface to Ruby's C API. With this Ruby binding in place, we were able to layer on an interface called HyperRecord, an extension of ActiveRecord. Using HyperRecord, we wrote off-line Rails scripts for a number of processing tasks to aggregate data and write it back into Hypertable. HyperRecord also enabled us to display the resulting data on our site.

MAP-REDUCE AND CASCADING

After computing a few page view and click metrics we wanted to compute additional metrics. However, we were reaching a performance bottleneck on the Rails side. The scripts we were running began to overload our hardware, and computation time was stretching into hours or even days. At this point, we started exploring how to use the map-reduce capabilities of the Hadoop processing framework to perform these computations.

There are a number of different processing frameworks built on top of Hadoop, such as Cascading, Hive, Pig, Pipes, and Streaming. After investigating, we settled on Cascading. Cascading provides a Java API that allows us to easily parse, filter, group, join, aggregate, and format data in Hadoop without having to make a distinction of what parts are operated in which map or reduce. With Cascading, we were able to quickly replace many of our existing Rails scripts and reduce their collective running time to a few minutes per hour of data.

LOGGING CHANGES

With our log analytics in place, we turned our focus to a more fundamental and intensive application for Hypertable: Tracking every change made to every listing in our database. Listings (events, venues, etc.) in our system can be modified by a number of different users throughout their lifetime. To maintain the overall quality of our site, we often need to track down which changes had been made, by whom, and when. Previously this was quite challenging for us.

To improve this situation, we created a detailed log of changes in Hypertable. Any time a modification is made to a published listing, an entry is written via HyperRecord. This is a great situation to use Hypertable because as the volume of data grows we do not have to worry about burdening our traditional RDBMS. This tool has proven useful in tracking down mistakes, inappropriate changes, and spammer activity in our listings.

FUTURE IMPROVEMENTS

The community has embraced Hypertable. One positive development is that there is now an interface for Hypertable in Thrift, a cross-language service framework. Through Thrift, Hypertable is now accessible naively in many different languages. We took this opportunity to port HyperRecord over to the new Thrift interface and remove our reliance on the existing native extension. It was also at this point that HyperRecord was released on GitHub.

Given the importance of real data for driving development, our integration with Hypertable continues to be very valuable. It is a core part of the value of our new premium listing and sponsored advertising product, providing detailed metrics to our customers and enabling precise, analytics-based targeting of sponsored results.

Currently we use Cascading to process log data from files on HDFS and load the results in HDFS into Hypertable after processing. A Hypertable connector for Cascading (based on the Thrift interface) is in the works. This will allow us to log data, process data, and store the results all directly in Hypertable.

Since only the single rowkey is indexed, it is often necessary to store extra copies of data to speed look-up. One way to do this is to create secondary tables that act solely as indices, storing the desired value in the rowkey and the rowkey of the primary table in another column. This provides for very fast look-up but places an extra burden on the engineer to keep both places up-to-date. We are currently exploring various methods for abstracting this secondary indexing. Although it is possible to implement this in HyperRecord, we do not want to couple the solution to Rails. We have considered implementing this inside Hypertable but have not fully fleshed out the feasibility and appropriateness of such a solution.

We have also been in touch with the SCADS research team at Berkeley. They are working to solve an even larger problem of abstracting something like Hypertable to provide an interface that will allow data needs to scale independent of code changes. In order to accomplish this, they too will need to use secondary indices.

In conclusion, we have made significant progress in leveraging Hypertable but have many opportunities to go even further. Please checkout HyperRecord and Hypertable on Google Code and GitHub. Patches are always welcome!

Meet Josh Tyler

STATS

PLACE OF BIRTH: Pittsburgh, PA
CURRENT LOCATION: San Mateo, CA
YEARS PROGRAMMING: 27
YEARS PROGRAMMING RUBY: 4

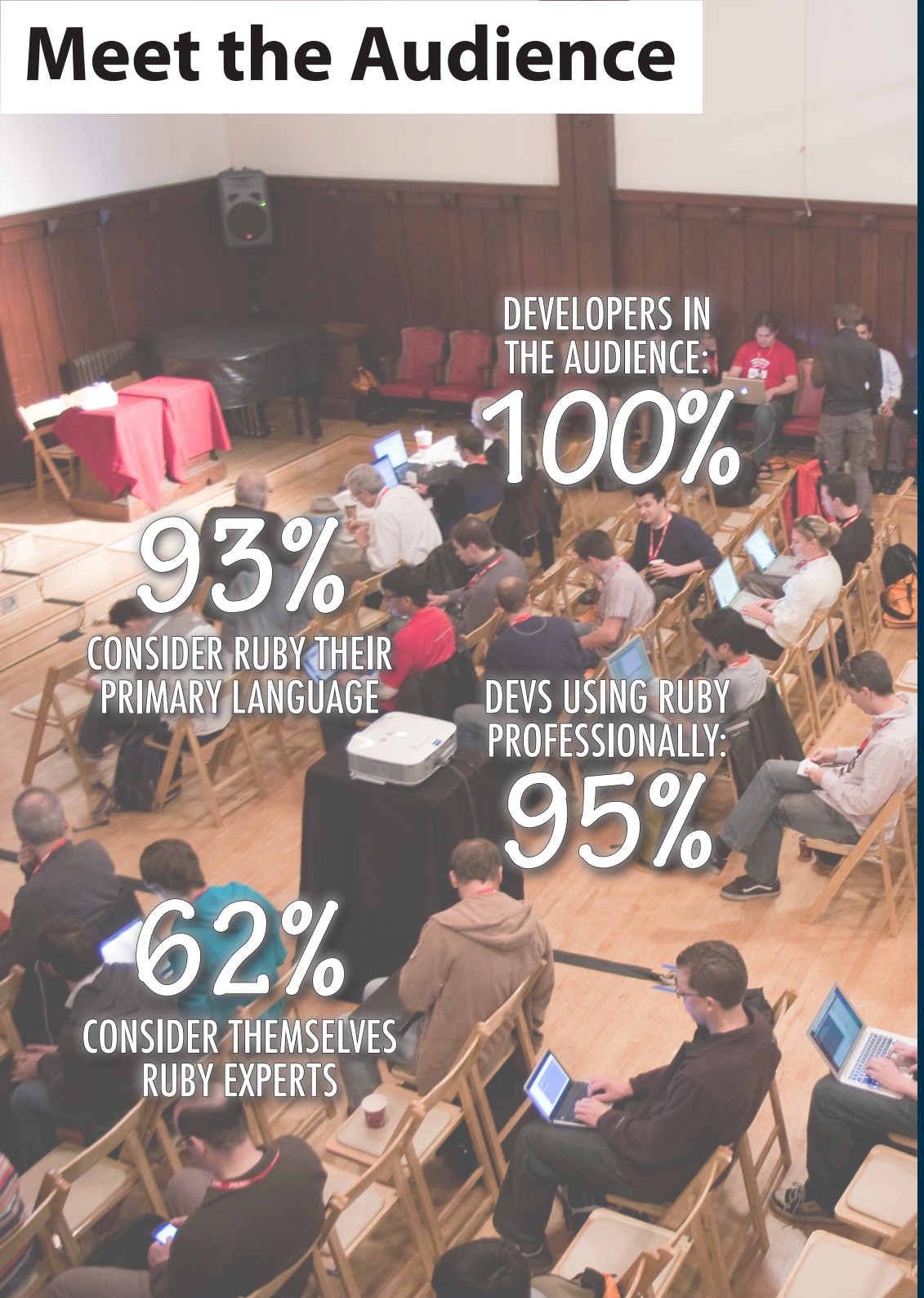
ONLINE HEADQUARTERS

WEBSITE: http://www.joshtyler.com/
BLOG: http://developers.zvents.com/

TOP OPEN SOURCE PROJECTS

NAME: HyperRecord
REPO: github.com/tylerkovacs/hyper_record

Meet the Audience



AT&T Interactive is devoted to creating compelling experiences that connect people, businesses and places so everyone can more easily engage with their world, wherever they work, live or play.

We have a passion for the Internet, mobile applications and emerging media platforms, and are proud to nourish a collaborative, tech-driven culture that promotes innovation, the entrepreneurial spirit and a desire to enhance everyone's daily lives.

As a tech-driven company, AT&T Interactive is always looking for passionate talented individuals who can solve difficult problems and develop new and existing products to exceed consumer expectations and drive value for our advertisers - whether it's online, mobile, interactive TV, video or other media platforms.

Tim Elliott

Exploring the Real World and Goofing Off:
Practical and Fun Applications of the Shoes Toolkit

By Tim Elliott

This article explores three scenarios in which Shoes is a useful everyday tool for Ruby programmers. You can wrap productivity scripts in a GUI and share them with non-developers, it lends itself well for prototyping mobile applications, and it lets you create robots that eat each other -- until they explode.

Ruby shines as a language for productivity scripts. I have created many scripts to automate tasks on my development workstation, and if you use Ruby on a day to day basis I'm sure you have written a few of these as well. These scripts can interact directly with files on your computer, and they can fetch & parse third party web pages at will. This is incredibly easy with Ruby. For the most part, these scripts never run outside of a developer's machine.

When I want to do something that will interact with a client, I instinctively reach for the nearest web application framework. However, with a web application you have to give up a lot of the things that make Ruby so powerful. If you want to interact with a user's files, they have to be painstakingly and individually uploaded to your server. Browser security policies will severely limit the networking that you can do on the client side. You are crippled by browser security, and you have to give up a lot of the conveniences that you take for granted when creating productivity scripts.

Shoes fits nicely into situations where a web application won't work because of browser security limitations. I recently dealt with a client who needed to upload a file via FTP, but was not savvy enough to install & run a regular FTP client. Using Shoes I was able to quickly create an FTP client that requires the user only to select a file on their file system and submit it to a pre-configured FTP server.

Shoes has opened up new possibilities for interacting with my clients' machines. It makes it easy to share productivity scripts with my users, complete with a friendly GUI. The resulting installer is small and will run on any Mac, Windows, or Linux desktop. Since it runs Ruby directly on the client machine, I am free to parse content from third party web sites and I have full access to the users' files.

This niche isn't the only application of Shoes (and it isn't even the use it was intended for) but it has proven to be very useful.

Aside from client-side productivity scripts, Shoes is good at prototyping GUI applications. It has a flexible layout engine and includes common GUI controls. The API that takes advantage of Ruby metaprogramming tricks to minimize the amount of code required to create interfaces. The controls are easily manipulated and animated. This lends itself well for prototyping mobile applications.

Shoes relies on two simple layout elements: stacks and flows. In short, elements that are placed in a stack are arranged top to bottom, and elements that are placed in a flow are displayed left to right, and then top to bottom. By combining these elements, you can create complex layouts with ease and design complex and simple GUI applications.

iPhone and Android applications are written in C-based languages. These languages are wordy and a lot less familiar for Ruby developers, and exploring data structures and architecture in these languages can be tedious. With Shoes you can create visual and programmatic mocks of your application. You can explore usability ideas and the architecture of your mobile application, and even create a fully functional prototype before delving into the native toolkit.

One trick that Shoes uses to simplify its API is self indirection. This technique places the 'self' method on a stack as you open blocks. This allows common methods, such

as 'background' to be directed to the first element on the stack without requiring block variables to be used.

Finally, Shoes is designed to make it fun to create graphical and animated applications. One of my first experiments with Shoes was in creating a playing field of robots that eat each other. This kind of playful programming allowed me to share my passion for programming with my non-programming friends at a level that they understand. I also rediscovered vector math, and picked up some new programming ideas in the process. I pick up this project any time I run into 'programmer's block', or I start feeling burned out by work projects.

By now my robots grow in size as they eat other robots, until they get too fat and explode -- into smaller robots. It reminds me why I started programming in the first place -- to convert my ideas into something real, to be creative, and to have fun.

What the People Thought

JOHN BAYLOR

"The presentation style was good. He spent enough time on the code slides that we could actually read them, rather than speeding through them. Shoes is a pretty cool application. And, Tim rode his bike down here, which is not something a geek is supposed to do. He's making the rest of us look bad ;)

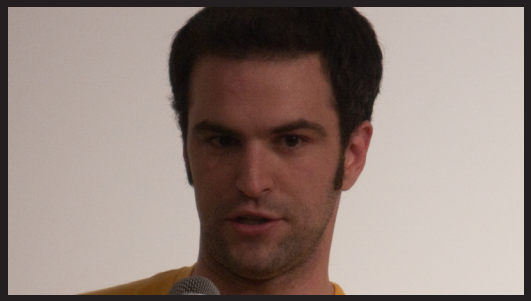
"I want to know why I can't use Shoes on a website yet. It would make it more ubiquitous.

"It's been great. I go to zero conferences in a year, so this is the best one I've been to."

ATTENDEE INFO

Using Ruby Since: 2002
Employed By: WildPackets
Blog: http://johnbaylor.org
Twitter: @JohnB

Meet Greg Borenstein



STATS

PLACE OF BIRTH: Los Angeles, CA
CURRENT LOCATION: Portland, OR
YEARS PROGRAMMING: 4
YEARS PROGRAMMING RUBY: 4

ONLIN HEADQUARTERS

WEBSITE: http://ideasfordozens.com
BLOG: http://ideasfordozens.com
TWITTER: @atduskgreg

TOP OPEN SOURCE PROJECTS

NAME: RAD
REPO: github.com/atduskgreg/rad

NAME: jmacs
REPO: github.com/atduskgreg/jmacs

Ruby Arduino Development:
Rails for Hardware Hacking

By Greg Borenstein

In January 1975, Popular Electronics ran a cover story about a new computer for hobbyists. The Altair 8800 came as a kit and cost $439 (the equivalent of $1,778.58 in today's dollars). It came with no on-board memory.

You programmed it by entering Intel 8080 opcodes by hand via a series of switches on the front panel. Buying 4k of memory, the ability to read in programs from paper tape, and a teletype interface would increase the price 6 fold. You had to solder the thing together by hand. By comparison with the big university and corporate mainframes it was basically useless.

But Popular Electronics was the Make Magazine of its day and engineering schools had begun to require their graduates to learn some programming, so Forest Mims and Ed Roberts, the two guys in Albuquerque who'd put the Altair together, figured they could probably sell a few hundred in the first year to this emerging group of hackers avant la lettre.

THE GIT BELL: DEMOED LIVE ON STAGE, THE GIT BELL RINGS WHEN YOU COMMIT CODE

They took 1,000 orders in the first month. Six months after the release they'd sold 5,000. By October of that year their company had 90 employees.

Why was the Altair such a runaway success? After all, by comparison to the cutting edge computers of its day, it was underpowered and extremely difficult to work with.

The answer is ownership. The Altair offered bourgeoning hackers their first chance at a computer that would be fully and completely theirs. They could take it home, throw it up on their work bench, take it apart, put it back together and try to get it to do totally new unimagined things. They could program their Altairs to play Fool on the Hill through a transistor radio. They could build a board to let the Altair drive a color TV in Times Square. They could start a small company in Redmond, Washington to sell programming languages for it. They could get together with their fellow Altair owners to share programs and cool hacks.

This last point can't be emphasized enough. The Altair crystallized growing local groups of DIY computer enthusiasts like the Homebrew Computer Club in Silicon Valley. It gave them an outlet for their energies and an example of what could be done. It made them believe that their incredible fantasy of having their own computers might really come true.

And after that, there was no stopping them. This new generation of hackers dedicated itself with an almost religious zeal to spreading the idea of computer ownership

across the world. They invented the personal computer ‚Äì a whole industry dedicated to the notion that computers could make life better and work easier for everyone, not just huge institutions. The Homebrew Computer club alone included the founders of Apple, Ozborne (builders of the first portable), and a raft of other industry pioneers.

Today, the world of physical computing closely resembles the personal computer industry circa 1975. We've been around for a few years struggling around the edges with tools and products that were designed, priced, and packaged for serious industry, but we haven't made any money and we haven't moved the world. That's about to change.

Recently, our Altair arrived. It's called the Arduino. This is 2009 so instead of being built by two engineers in Albuquerque, it was built by an open source international cabal of programmers and professors.

A lot of people complain that it's underpowered and overpriced (even though it only costs $8.64 in 1975 dollars). But you don't need special hardware to program it. It lets you do all the basic tasks with just a line or two of perfectly comprehensible code. And there's a thriving community of people busily using it to do all the useless, fun, creative things they'd always dreamed of if only they could get their hands on a computer that could sense and control the world around it. They're using it to teach houseplants to call for help if they need watering. And they're using it to play music on glasses of water.

If the Arduino is the Altair of physical computing then what will be its VisiCalc? What will be the killer application that makes the physical computer of the future a necessity for business. If the Arduino is the Altair, what will physical computing's Mac look like? I don't think anyone today knows the answers to these questions.

But the answers are coming. In the next few years, physical computing has as much of a shot at changing the world as technologies ever get. And this is the time to get involved. Unlike the web, personal computer, and green energy industries, physical computing is a space where two guys in a garage can come along and invent something that will touch billions of people around the world without anyone else's permission. That's because what's needed in physical computing is not advanced research, massive infrastructure investment, or huge production facilities. What's needed is close attention to applying the existing technology to solving human-scale problems. Microcontrollers and sensors, CNC milling machines and laser cutters, GPS receivers and accelerometers need to be transformed into tools that regular people can use to improve their daily lives: to make themselves more connected to the people and places around them, to entertain, educate, and distract them.

In 30 years, when people tell the story of the Physical Computing Revolution and how it changed the world, what will they say about you?

Where to Buy Arduino Hardware

www.sparkfun.com: sparkfun is the most popular source in the US for buying Arduinos and they stock lots of the basic accessories for beginner and intermediate projects like servos and gps modules

www.adafruit.com: adafruit has some great shields and kits for more advanced projects like motor control and ethernet integration.

www.wulfden.org/TheShoppe: home of the really bare bones arduino kits which let you build your own arduino-compatible microcontroller for a fraction of the price of a fully assembled board

What the People Thought

AMY WOODWARD

"It was really interesting. I wasn't sure what to expect and it was a lot more interesting and hands on than I had thought. I basically want to go and get myself a board now.

"Greg was entertaining. He primed the audience to ooh and ahh over each stage of the presentation, just in case, whether they worked or not. I particularly liked the little Git Bell.

"I've been enjoying the show. Lots of interesting people to meet and talk to, and a lot of really interesting, informative well-presented talks."

ATTENDEE INFO

Using Ruby Since: March 2006
Employed By: Grockit

Hampton Catlin

The iPhone App That Opened Doors:
Bringing Ruby to the Wikimedia Foundation

By Hampton Catlin

I finally got the chance of a lifetime. There I was standing in front of the *super secret* Wikimedia offices in San Francisco. I was reflecting on how I got there and pumping myself up so that I didn't look like a total idiot. Three months prior, my friend Melissa was complaining to me that there wasn't an easy way to read Wikipedia on the iPhone. We were a bit drunk on wine and as she left that evening, I promised her I'd have it done in the next few days. It seemed like something that was so obvious that someone had to do it and it may as well be me. I stayed home from work the next day (a Friday) and worked through the weekend. I taught myself Objective-C and Cocoa over that weekend and had a working demo that Monday. I showed it to my friends at Unspace and they were ecstatic.

A week later it got up in the Apple store and I was selling it for $0.99. The title of the application was "iWik: Wikipedia for the iPhone." It was a huge success. Not as huge as some of the apps have been, but I made enough money over the next two months that I really didn't have to do any client work. In September, I got contacted by Mike Godwin, Wikimedia's General Counsel, about using their name in my title. I was infringing on their trademark with the name of my application, and they couldn't have been nicer about it.

As we were talking, Mike was mentioning that they were interested in having someone do mobile work. At the time, Wikipanion was already out. Which, is a much more full-featured (and free) app in the store. My sales were falling, but I was ok with it. I figured that they were talking to those developers also. A few weeks later I had a call with Brion and we talked about Wikipedia and what I had in mind for them.

At this point, I was nearly shitting myself. I couldn't believe that I was talking to the CTO of Wikimedia. Wikipedia has been a passion of mine for a long time. Its the closest thing I have to a religion. That is, the religion of information. Believing in information's power to transform and improve the human race. This wasn't just a cool job to me, this was a mission.

I flew to San Francisco and I got the job. How did I get the job? My open source work. What they needed more than anything was someone who could run an open source project. And, that's something I have a lot of experience doing with Haml and etc. Now, before me was the greatest task in my life. I have to make Wikipedia, a site I hold in holy reverence, awesome for mobile devices. It was up to me to make it fly or fail.

Part of hiring me, I told them that I must use Ruby. That I thought it was up to the task and if they wanted me to do the work, it would be using my preferred tools. Brion is absolutely awesome in this respect. Whatever it takes to get your job done and do it well is good with him. He does his job by keeping the end-goal in mind and I do my job by making it happen.

The key to my idea was having a formatting-service. As I was building it, I wasn't just focused on doing iPhone or even just mobile. To me, this was an abstraction-worthy service. Why build it to just work with iPhone? What about Blackberry? What about things we haven't invented yet? What about an API for reading where our servers do the heavy lifting? That's what "Wikimedia Mobile" is all about. Its really a formatting service for Wikipedia articles focusing on the READ task. I really encourage people to go look at the code. Its all in Merb and is pure ruby and I am pretty damn proud of it. It is extremely easy to extend languages, formats, and formatters. Nearly everything is overrideable with some really simple commands and Yaml files.

At first, I got pushback from the server guys. They were

all predicting total failure to scale or handle any decent amount of traffic. However, I am proud to say that handling 100,000 unique page views a day, with no caching turned on at all, is producing a load of just 0.01 on our single formatter server.

Next, the task to me was to build a native application. I knew I could repeat my Objective-C work in a week or retrofit my old application. But, that seemed too small for me. I wanted an Android version. I wanted a Symbian version. I wanted native apps on any phone that would have us. In came Rhodes Mobile! One code base, 5 phones, all in Ruby! Plus, they were excited about the project, so they were willing to help as much as they could. I am very proud to announce that the official *native* application of Wikipedia on the iPhone is running *ruby* also! Next up comes the small task of porting to Android and the other phones.

Where do I go from here? Is my job done now? Not at all. I am now pushing Wikimedia to have an even more open stance when it comes to data usage. Instead of rewriting parsers and crawlers for the main site, I want to support development of applications that want to use the read capabilities of Wikipedia. Integration is the future on the web. I can't tell you how many different services my applicaitons use now. Why not make Wikipedia just another data source? Well, that's what I'm aiming for now. But, I am always looking for the support of the Ruby community. The code is open source and accepting patches. What do YOU want to see Wikimedia Mobile do? Well, make it happen.

Meet Jon Crosby

STATS

PLACE OF BIRTH: Kansas City, KS
CURRENT LOCATION: Walnut Creek, CA
YEARS PROGRAMMING: 13
YEARS PROGRAMMING RUBY: 4

ONLINE HEADQUARTERS

WEBSITE: http://joncrosby.me
BLOG: http://blog.joncrosby.me
TWITTER: @jcrosby

TOP OPEN SOURCE PROJECTS

NAME: CloudKit
REPO: github.com/jcrosby/cloudkit

NAME: CloudKit jQuery Plugin
REPO: github.com/jcrosby/jquery-cloudkit

Toward Composable REST Services:
Building Stronger APIs

By Jon Crosby

During the CloudKit presentation at the first Golden Gate Ruby Conference, a rapid-fire look into architecture, Rack, HTTP, REST, discovery, Ruby and JavaScript was packed into the short time available. One of the trade-offs for building such a presentation is one of technique vs. philosophy. Given the conference name, I chose technique and detail over a more general philosophical discussion about the Web. In this article, I will give a few pointers to big-picture ideas that drive CloudKit development and those I believe will move the state of the Web in a positive direction.

BEYOND THE BOX

A powerful shift in web-perspective occurs when considering APIs, cache-ability and generic user agent utility during the initial development cycle. Nearly a decade after the publication of RFC 2616, the specification for HTTP 1.1, many of its benefits are overlooked by focusing only on "the box" in the browser. Clearly, many successful web applications have been built in this style. Using the ideas presented here will allow developers to continue focusing on user interactions while extending web service capabilities to other non-human agents (such as smart in-browser data stores) and non-browser user agents (such as other web services, desktop and mobile applications, etc.).

Breaking out of the box isn't as simple as inventing new one-off techniques. There are a number of standard HTTP techniques, many of which were mentioned in the CloudKit presentation, that are already available. Essential reading for any web developer should include RFC 2616 and Fielding's dissertation where the REST architectural style was derived through the application of useful constraints to the Web.

DISCOVERY, CONNECTEDNESS AND THE IETF

There are several notable drafts under IETF review that have strong benefits for web service and API developers. If the techniques provided by HTTP and RESTful design have been applied and there are still areas of your API that cannot be adequately addressed, consider the following drafts before building your own abstractions.

WEB LINKING

Just as HTML and Atom provide methods of indicating relationship types using the common "rel" attribute, Mark Nottingham's Web Linking IETF draft describes a method for defining these relationships without being tied to serialized document formats. CloudKit, for example, uses the "Link" header to link JSON resource representations to their version history, resources to their indexes, and more.

HOST METADATA FOR THE WEB

CloudKit hosts its own "bootstrap" information under a single URI -- /cloudkit-meta. This allows clients to discover what resource collections are being hosted, and then begin a series of OPTIONS requests to those collections to further explore the service.

As web services grow and single sites combine many of them under the same root, a need arises for a common, root-level technique to discover all of the capabilities of a given domain. Host Metadata for the Web attempts to specify a single "well-known location" that acts as a directory for site-wide metadata. Examples of common items that might be listed as host metadata include a flexible location for robots.txt, privacy policies, etc. Using a single well-known location reduces the possibility of collisions with present and future URI spaces.

Mapping this back to CloudKit, if all of its techniques were to become standardized, a host metadata entry indicating a type of "json-rest-service" (an arbitrary name used only for this example) could link to "/cloudkit-meta"

in one domain, while another might link to a similar point of discovery for CouchDB or Persevere. More to the point of Host Metadata, all three could be linked under the same domain without collisions.

LINK-BASED RESOURCE DESCRIPTOR DISCOVERY

Building on Web Linking and Host Metadata, Link-based Resource Descriptor Discovery (also known as LRDD or "lard") describes a method of using a "describedby" relation type to provide a machine-readable description of a given resource that is separate from the resource's representation. The draft does not cover specifics of the descriptor document format, covering only the mechanics of obtaining the document. Standardizing the method in which agents can obtain information about a resource sets the stage for a more powerful -- more open -- web, described below.

TYING IT TOGETHER ON THE OPEN WEB

A solid description of the Open Web was published on Brad Neuberg's blog in July of 2008: "The Open Web is an interoperable, ubiquitous, and searchable network where everyone can share information, integrate, and innovate without having to ask for permission, accessible through powerful and universal clients." This description was followed by a litmus test including the properties of Composability, Interoperability, Ubiquity, and the idea of the Universal Client.

Given what has already been built with OpenID and OAuth, followed by its rapid adoption across large providers such as Google, Yahoo!, MySpace, Plaxo, and others, we have a method of universal authentication and authorization that provide the foundation for future Open Web building blocks. OpenID and OAuth were created prior to the IETF drafts described previously, requiring them to invent techniques that are now being solidified and abstracted by various standards efforts. This is where CloudKit and other RESTful bits of machinery come into play.

By answering the questions of "who am I?" and "what can I do?" with OpenID and OAuth, we have the remaining questions of "with what?" and "how?" to answer. While REST already provides "what" (resources and URIs) and "how" (HTTP verbs and more), new possibilities emerge when those questions are refined for specific problem domains.

As an example, imagine a JSON Schema describing the format for a photo and its metadata within a collection of related photos. With this small bit of information, a standard for an open, portable photo collection could be created consisting of existing REST techniques plus relevant details about discovery, formats and methods. This would enable users to point a photo printing web service to a previously unknown photo collection service in order to import recent photos for printing -- all by simply entering a URL. Browsers or other generic user agents (even desktop applications) could also provide optimized methods of viewing and interacting with these services. All of this could be accomplished without the present chore of finding each provider's one-off API documentation and writing specific clients for each service.

CONCLUSION

The above standards, techniques, and ideas are not exhaustive. Many of them are still taking shape. The intent of this brief overview was to point the way to more composable REST services through links to related reading, offering methods of building web applications with these ideas in mind. Doing so will allow developers to continue delivering value on the web while architecting with the future -- and increasingly, the present -- in mind.

What the People Thought

JOSH KNOWLES

"It was a great talk. He packed a lot of great info into a short amount of time and it was just enough to peek my interest and make me go and look at more. The 30 minute format gives you just enough info to decide if it's worth researching on your own.

"There's a great group of people here. I think that the technical level of the talks is all at the right level. If there's a talk you're not interested in it's short enough that you can enjoy the hallway track and come back an hour later.

"I'm really impressed with the way that he leverages the http protocol to handle caching and versioning. The use cases for this for desktop and iPhone apps are endless."

ATTENDEE INFO

Using Ruby Since: 2002
Employed By: WePlay
Blog: http://joshknowles.com/
Twitter: @JoshKnowles

Aaron Quint

Maximize the Minimized:

Unleashing the Power of the Local Web with Sinatra

By Aaron Quint

If Sinatra is *the* Ruby web library, Vegas is it's stage to preform locally. Vegas is a small project I've started who's goal is to make creating web based interfaces for your Ruby libraries even easier. Sinatra takes the first step of making it easy to define routes and ways to interact with your code over HTTP, and Vegas takes the next step by making it easy to package libraries in an executable thats easily opened in a browser.

In our current mindsets the web and the HTTP protocol serve a very specific function. They exist to serve data from a remote server rendered as HTML and displayed in a web browser, a desktop application for viewing this data. There has been a shift happening - traditional desktop based apps are being replaced by web alternatives that provide the same functionality (or at least almost the same). For desktop apps that need to be connected to the web to function, or are really based on data fetched from the internet, a web or browser based client makes complete sense. I'd like to push this one step further. For developers - let's use localhost as more then just a staging ground. Let's use the local web to give our apps or libraries new life.

For certain apps, giving a user a web interface has a large number of benefits. The command line is great for many things, however, it fails with the need to input a large amount of options or view any sort of visual (non-text based) data. As a user interface the command line is utilitarian, it gets the job done, and for power users, it provides access to the more advanced features out of the box. However, to be perfectly honest, its ugly. It's not that CLI are antiquated or 'DEAD', but in an age when even our phones are web browsers, it would make much more sense if our code could present itself in a browser. Allowing our libraries to speak HTTP gives end-users a standard, and hopefully more friendly, way of configuring, interacting with, and using our code.

In my vision of the near future, all of our gems have a simple command that opens a web interface in the browser. We can use these interfaces to validate code, configure output, start and stop processes, inspect databases - all using paradigms and interface elements we're accustomed to. Beyond that, using the power of HTTP these integrated web interfaces can become web services as well communicating amongst each other or through a central service all on our local web.

Making this vision a reality, rests (pun intended) on Sinatra. Sinatra is 'the classy web framework', but my belief is that Sinatra is much less than a framework. This sounds like an insult but its a good thing. Sinatra is a web library. It's lack of features and constraints give it the amazing ability to be included side by side with existing code to instantly add the power of the web. At it's heart Sinatra is just a simple wrapper around rack, but its syntax is clean and easy to use. Making use of Sinatra in your existing library is as easy as requiring, subclassing, and defining some routes.

Once you have your Sinatra class/app you can create a simple executable for it. Vegas provides a number of features and helpers for creating these apps, most notably an easy wrapper for creating these Sinatra executables. Vegas also has a web interface that allows you to view or run these 'web-ified' gems. In the future I'd like Vegas to act as a central brain for all of these apps, allowing them to send messages back to a hub to be distributed to other apps or displayed or handled by Vegas. This could be as simple as displaying a system message using 'growl' or as complex as spawning a long running process to crunch data.

The tools are there to create these interfaces, but its up to you, the maintainers, creators, and developers to give your gems this power. After a little thought and minimal effort it could breathe new life into a project or reveal or clarify features that had been left in the dust.

New Relic RPM lets you see and understand performance metrics in real time so you can fix problems fast. It's intuitive. It's granular. And, it's a 10-second Rails plug-in install.

Take **RPM Lite** for a spin — it's free!
NewRelic.com

New Relic
Rails Performance Management

Don't listen to us, listen to them.

RPM gives your developers **transparency** into the performance details of our applications, **saving significant time** in isolating bottlenecks.
— Ian McFarland, Pivotal Labs

New Relic's RPM has **dramatically improved** the way we monitor the health and performance of our applications.
— Mark Imbriaco, 37signals

If you are running a Rails app of any reasonable size **you *have* to use New Relic**.
There is no way around it.
— Tobias Lüetke, Shopify

With RPM we had **more time to add new features** and fix issues, rather than scanning through piles of log files.
— Rick Olson, Lighthouse

We used the other tools out there, and were **not happy until we switched to New Relic**.
I was amazed how much we found in just a few hours of monitoring. We got the **real-time picture we had been dreaming about**.
— Yaroslav Lazor, railsware

David Stevenson

Playing With Fire:
Running Untrusted Ruby Code in a Sandbox

By David Stevenson

THE ULTIMATE CUSTOMIZABILITY

Users expect applications to be increasingly more customizable and personalizable than ever before. One approach to provide these services is to offer an external API where advanced users can directly access the low level domain objects of an applica-tion. Facebook has made their API very successful by doing exactly this technique. Unfortunately, this places an immense hurdle in front of users who want to build on your application by requiring that they build and run their own application. Because of the overhead involved, only the most advanced users will be able to use an external API.

If you want to make customization easier and not sacrifice any power, you might end up wanting to run user-provided code directly in your application. There are a variety of ways to accomplish this, but all of them tend to have a security mechanism to protect the parent application. Sandboxing is the most common technique, where the user generated code is executed in a context where it cannot cause any dangerous side ef-fects. A less common technique is code signing, where user provided code executed in a unlimited security context, but is first reviewed by the parent application developers and approved as safe.

Allowing users to contribute code can make applications incredibly dynamic. As a gam-ing platform, SecondLife has built an empire on this idea alone. The overhead for a programmer to build an interactive object in the SecondLife universe is fairly low, and it's conceivable that some non-programmers could begin to provide code.

RUBY SANDBOX IMPLEMENTATIONS

Unlike the C-like language that SecondLife uses, ruby is an excellent choice as an ap-plication scripting language. It is close to english, allows for easy creation of DSLs, and does not require compilation or verification in advance.

There are two main sandbox implementations: one for MRI ruby 1.8.6 and another for JRuby. Both provide a "safe" sandbox where functionality is limited, and a "full" sandbox which has the exact same classes as the parent application. The latter is useful when you want to run code that is trusted, but guarantee that it won't interfere with the definition of other trusted code.

The MRI implementation is known as freaky freaky sandbox and was developed mostly by why-the-lucky-stiff in C. The implementation is unquestionably a hack on top of the existing ruby VM, and relies on swapping out sets of defined classes. In the sandbox's object space there is a 2nd smaller set of classes which are much more limited than their normal ruby counterparts. When executing sandboxed code, the class table is swapped to the safer set.

Developers using the sandbox can allow access to their classes either by copying those classes into the sandbox or by referencing them. Methods of copied classes execute inside the sandbox, and cannot do anything that the sandbox cannot already do. Meth-ods of referenced classes actually execute outside of the sandbox temporarily, allowing them to anything. Referenced classes are dangerous only in that great care must be paid when developing them to prevent any unexpected usage of these classes for evil purposes. Code inside the sandbox cannot change the behavior of a referenced class, it can only execute the methods provided by the developer.

The JRuby implementation, written by Ola Bini and known as javasand, provides the same exact interface as the MRI implementation. The mechanism is very similar: creat-ing a new table of defined constants and swapping it in and out. It's much less of a hack, however, because

unlike MRI, JRuby provides hooks where the swapping can be accomplished without hacking the interpreter.

Neither implementation is easy to install. MRI requires a patched version of ruby 1.8.6, although the patch is trivial and related more to error handling and not to sandboxing itself. The sandboxing is accomplished using the C API. Once ruby is patched, how-ever, installation is straightforward. The JRuby implementation should be easier to install, since it's just a binary gem, but I was unable to get it running without downloading the gem source and compiling it against my particular version of JRuby (1.1.5). After being manually compiled, packaged, and installed, it worked identically to the freaky freaky sandbox.

BUILDING APPLICATIONS USING SANDBOX

The simplest use of the sandbox is as an expression parsing engine. User uploaded code consists of short expressions that contain simple ruby operations such as mathe-matical operations and string operations. The sandbox is used to temporarily and safely evaluate the result of the expression, either just-in-time or once before storage.

More complex sandboxed applications usually provide an internal API available to user-uploaded-code. All of the standard ruby data types and operators are available inside the sandbox through copied classes (automatically), and the API classes and methods are referenced classes (so that they run outside of the sandbox and do "real" work). The developer must ensure that the methods of the API classes cannot be manipulated by the untrusted code to cause unexpected side effects.

Creating an internal API often involves exposing domain objects that are already exist. When these objects contain more functionality than can be safely exposed, such as us-ing an ActiveRecord object, the proxy pattern provides a simple solution. By wrapping the object in a proxy class, the proxy can control which methods are safe to call. The unsafe object must be wrapped in the proxy before being passed into the sandbox and exposed as an API. The acts_as_wrapped_class ruby gem, written by David Steven-son, provides an automatic and simple implementation of this approach. If desired, the acts_as_runnable_code gem can work with the wrapped class gem to provide automatic set up of the sandbox and evaluation of uploaded code in the context of a top-level-binding.

RISKS OF USING A SANDBOX

1. The protection provided by the sandbox could be bypassed. Developers using the sandbox and inherently trusting it to provide protection to the rest of their applications. Research into potential vulnerabilities and testing for weaknesses should be performed by any developer seriously considering adding the sandbox to their application. Depending on the consequences of a security breach, the sandbox might simply be too risky.

2. User-generated code could not terminate. The sandbox provides protection from this risk by allowing a timeout, accomplished through throwing an exception. Developers using it should write tests verifying that this method for killing long running executions cannot be bypassed by catching that exception, using an ensure block, or by any other means.

3. The API could be dangerously misused This is probably the biggest risk of using the sandbox, but it applies to any API pro-vided to users of any application. The fact that the method of interaction with the in-terface is through ruby code introduces new potential vulnerabilities. Ruby was de-signed as a powerful language, not for internal security (unlike a web SOAP inter-face, for example). Developers should spent extra time writing tests to ensure only the objects and methods they expect to be usable in the sandbox are actually avail-able.

THE FUTURE

Ruby sandboxing is still in its infancy, as an experimental project. All of the develop-ment was done by a handful of people between 2006 and 2007 for personal use, so it has not yet been proven in a production environment. The JRuby implementation seems more promising for stability and maintainability, but no work has been done on either implementation for more than a year now. I personally think that there are busi-ness applications for running user uploaded code, and I hope that development can eventually continue on the sandbox project.

Meet Carl Lerche

STATS

PLACE OF BIRTH: Würzburg, Germany
CURRENT LOCATION: San Francisco, CA
YEARS PROGRAMMING: 10
YEARS PROGRAMMING RUBY: 5

ONLINE HEADQUARTERS

BLOG: http://splendificent.com
TWITTER: @carllerche

TOP OPEN SOURCE PROJECTS

NAME: Rack::Router
REPO: github.com/carllerche/rack-router/

Write Fast Ruby:
It's All About the Science

By Carl Lerche

It has been said that Ruby is slow. The benchmarks from the Computer Language Benchmarks Game shows the various ruby implementations as ranking close to last. Benchmarks only tell a partial truth. The reality of the situation is that for the vast majority of situations the ruby interpreter is fast enough. Slow code is most likely your fault. In this article, I'll be showing you a process by which you can take your slow ruby code and tune it to become as fast as it can be.

So, how can you write some fast ruby? I guess that this is the overarching question here. The answer might come as a surprise, but the first thing to do is to write slow code. More importantly than slow, however, you will want to write clean and well structured code. You want to write code that is well tested and easy to refactor, because you will be tweaking internals quite a bit and playing with different implementations on your quest to the fastest possible ruby. Also, there is no way of telling if the application that you are writing will be slow before you actually get it written.

The second step is to use science! Once the software is written, don't just stab at things wildly or pick a random section of code that you think might be a bottleneck and try optimizing it ad hoc. Doing this will only end with a lot of wasted time and no real way of knowing if any progress has actually been made. Science, on the other hand, provides a set of techniques that collectively are referred to as the scientific method. These techniques have been good enough to get humanity to where it is now, so it's a safe bet to assume that those techniques would be adequate for speeding up a little code. Here are the steps to follow:

Define the question, gather needed information, form a hypothesis, perform an experiment and collect data, analyze and interpret the data, publish the results and / or retest. I'll go into each of these in a bit more detail.

DEFINE THE QUESTION

"Why is my code slow?" is not an adequate question. There are no metrics associated with this question. What does slow even mean in context of your application? A web application and an XML parser have completely different reference points when it comes to speed. You can't optimize your code unless you really know what you are optimizing. Some examples of valid questions might be "Why is the UsersController#index action of my rails app taking 400 ms on average" or "Why is 60% of the time spent in a Merb dispatch cycle stuck in content negotiation". Both examples include measurable elements.

If you are writing a Rails application, a good way to get a high level overview of what your rails application is by using tools like Rack::Bug or especially New Relic. New Relic's tool collects production performance data and will let you know what your weakest links are and where you should spend time optimizing. If you are writing any kind of ruby based software, RBench is a nice benchmarking tool that lets you setup common use cases and time how long they take. Merb has a directory containing a suite of benchmarks that allows us to keep track of how various commits effect performance.

GATHER NEEDED INFORMATION

Gathering the needed information is the most important step in the processes. Without solid information, you will not be able to form a hypothesis regarding why the code is slow. To get this information, my favorite tools are a combination of ruby-prof with kcachegrind. Ruby-prof is a profiler that can output data in a multitude of formats; however, the only format that I found useful has been the call tree. After you setup ruby-prof with your project (see the documentation for specifics) and setting the output to the call tree, you can load up the file in kcachegrind (avail-

able on macports). Kcachegrind is an amazing tool to visualize exactly what's going on in the bowels of your code. The software takes a bit of time to get used to, so I would suggest reading up on available documentation.

While you are digging through the enormous amount of information that Kcachegrind provides, the main thing that you want to keep in mind is that you are looking for specific granular bits of logic that are taking a suspicious amount of time. In other words, self-contained logic that you can optimize (remember, you only write clean code).

FORM A HYPOTHESIS

You are finally ready to take a guess as to what specifically is the culprit to your slow code. You have all the information that you need to form an educated guess, so take a shot. Was it a lack of eager loading when using ActiveRecord? Perhaps you might have been creating too many ruby objects and abundantly triggering the garbage collector. No matter what the source of the bottleneck, phrase your hypothesis in a way that is verifiable: "#generate_from_segments is taking 80% of the runtime due to an excess of Enumerable operations that could be pushed to initialize time."

PERFORM AN EXPERIMENT AND COLLECT DATA

We now have a hypothesis as to why our code is so slow, so let's test our theory. The obvious way to do this is to make changes to the code base that removes the bottleneck discovered in the previous step. After doing this, setup your benchmarks so that they can compare the code before the optimizations with the code written after. Without comparing the same things before and after, there is no way of knowing if the changes made actually improved performance a significant amount or not.

HALLWAY TRACK: CARL CHATS WITH YEHUDA KATZ AND MATT AIMONETTI BETWEEN SESSIONS

ANALYZE AND INTERPRET THE DATA

Alright, we're almost done. The obvious first thing to check is to compare the various benchmarks. Were the use cases that you were focusing on actually improved in a measurable amount? If the answer is no, then either your original hypothesis or the experiment performed were incorrect. However, there is a lot more than simple benchmarks to keep in mind. Did you slow down any other use cases (it's probably a good idea to keep general performance metrics around)? How did your changes affect memory usage? Were you optimizing for a common use case, if so, how did your changes effect various edge cases? There are also many other factors to keep in mind besides raw performance depending on your application, such as latency, throughput, etc... All these aspects are why you need a comprehensive high level overview of your application.

PUBLISH THE RESULTS AND / OR RETEST

If your results are satisfactory, you are done for now. Show your team the results that you have obtained. Get some feedback. See if they get the same results. You can also push your code out to the world and check to make sure that you are getting the expected results. If you are working on an open source project, you can also publish your results to the community. No matter what you do, you are better off getting more eyes on your work.

In conclusion, don't worry about performance until you actually know that you have a performance problem to focus on. Don't be a code cowboy and follow the outlined steps, even just as a mental exercise. Quite often, you will find that your original expectations are quite off target.

Meet Matt Aimonetti

STATS

PLACE OF BIRTH: France
CURRENT LOCATION: San Diego, CA
YEARS PROGRAMMING: 8
YEARS PROGRAMMING RUBY: 3

ONLINE HEADQUARTERS

BLOG: http://merbist.com/
TWITTER: @merbist

TOP OPEN SOURCE PROJECTS

NAME: Merb
REPO: github.com/mattetti/merb/tree/master

NAME: couchrest
REPO: github.com/mattetti/couchrest/tree/master

NAME: macruby-mirror
REPO: github.com/mattetti/macruby-mirror/tree/master

CouchDB:

Apache's Document Oriented Database

By Matt Aimonetti

If you are developing web applications and you need to store data in a persistent way, you are more than likely using what's often referred to as an RDBMS.

The term "Relational Database Management System" was defined by E. F. Codd in 1970. Most existing web applications use Sqlite3, MySQL, PostgreSQL, Oracle DB or another RDMS flavor. The concept behind an RDBMS can be summed up as: everything can be defined as a relationship. Data and relationships are shoved into tables with columns and rows.

With everything moving to the cloud and applications evolving, we are facing new challenges.

Most fields become optional, most relationships are many-to-many, joins don't scale well, adding more databases causes replication challenges. In other words, scaling an RDBMS is very challenging and usually involves a large amount of ibuprofen.

SO HOW DO YOU DEAL WITH THIS CHALLENGE?

- Flickr doesn't use any DB joins.
- Google uses a column oriented database solution called BigTable. Hypertable, an open source erstaz of BigTable brings most of google's solution goodies on your own servers.
- FriendFeed stores schema-less data in MySQL.
- Facebook greatly relies on memcached, a non-persistent Distributed Hash Table solution (DHT).

DHT solutions are pullulating: project Voldermort, Scalaris, Tokyo Cabinet, Redis, MongoDB etc... Looking at what the pros are for using a DHT, you quickly understand why people are willing to think outside the box. A DHT gives you a distributed, easy to scale, fault tolerant solution.

Enter Apache's CouchDB. Couch is a Document Oriented Database written in Erlang and is designed to handle load and scale. It can be used as any of the other persistent distributed hash tables but with a few twists.

- First, instead of storing strings or serialized objects, Couch stores all data in a schema-less structure using the powerful JSON format. The main advantage with this feature is that you don't need to try to shoehorn your objects into tables. You can dump/load full objects with their relationships to/from a document. You can also define relationships the way you want and optimize retrieval of objects.
- Secondly, Couch uses a built-in map/reduce making your data dynamic and your queries static. (very powerful concept once you wrap your mind around it)
- Thirdly, Couch has a great replication system allowing n-masters and conflict management.
- Conflict management is provided by a built-in, multiversion concurrency control which keeps all the revisions of a document until you compact your database.
- To assure that no transactions are lost, Couch is fully ACID compliant.
- Couch has a full REST HTTP interface making scaling much easier since we already know how to scale HTTP. Your usual caching/load balancing techniques and tools can be used with CouchDB.
- Documents can have 'attachments', in other words, documents can have a link to uploaded files which are available on the file system and via the DB. The attachment model is inspired by the email attach-

ment concept and attachments are replicated with your database.

- Couch comes with a web interface called Futon which helps you manage your DB nodes.
- Finally, Couch is expandable using 3rd party plugins like CouchDB-Lucene, which adds a full text search engine to Couch.

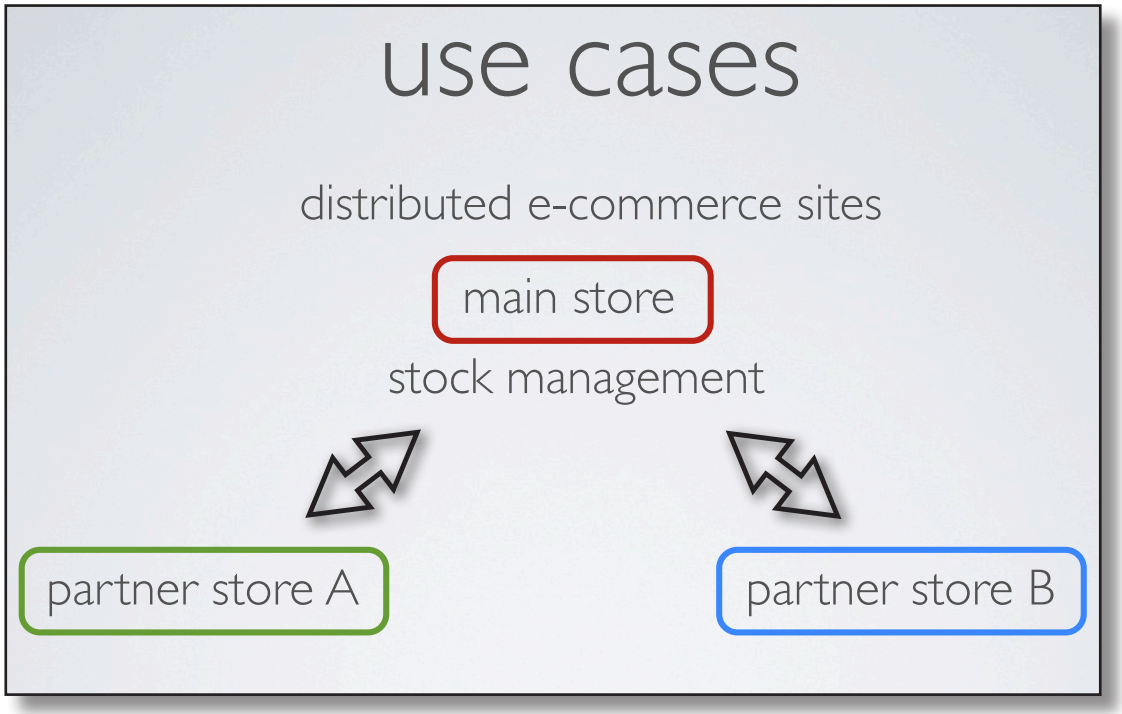
Of course, like most of the databases, CouchDB doesn't care what programming language you use. And because most programming languages already know how to communicate via HTTP, you don't need to write any custom drivers. (Note that there is an Erlang library for Couch and anyone can 'easily' write a low level driver if need be).

When it comes to Ruby, you have the choice of using multiple libraries. Some of them try to reproduce the ORM approach of ActiveRecord while some try to stick to CouchDB's philosophy. My personal preference is CouchRest, a library authored by Chris Anderson who is also one of the CouchDB team members.

CouchRest can be used in two ways: the low level approach, and the more abstracted DSL.

In the low level approach, the DSL is very simple, you define a DB node, and use the REST interface using Ruby's syntax. The full API is mapped for you allowing for very low level optimization and simple interaction with your documents.

The other option is a richer/more abstract DSL. Inheriting from CouchRest's ExtendedDocument class, your models can map full documents, define properties, validation, callbacks and casting. Very often a document might contain multiple models and CouchRest gives you all the tools required to cast these models as expected and put them back together when you save/update one of your objects.



COUCHDB: MATT TALKED ABOUT SOME REASONS TO USE COUCHDB

Unlike ORMs, CouchRest doesn't try to define relationships. The DSL leaves that to you. You can use a map view, use a foreign key and make an extra query or do it your own way. This flexibility allowed me to not have to fight with the data store.

Finally, CouchRest is designed to be extended by 3rd party developers who want to extend CouchRest Core and create their own DSL.

So, when should you use Couch?

(before answering this question, don't forget that you can use Couch and an RDBMS simultaneously.)

- You could totally switch to CouchDB, no reason not to ;)
- If you expect to need more than one database server
- When availability is more important than consistency
- When your data is decentralized
- When you need to compute data

I hope you will have at least as much fun as I do when using CouchDB. And, most importantly, be sure to look at other options and pick what most makes sense for your own project.

Nick Kallen

Attendee Coverage:
Magic Scaling Sprinkles

Write up By Sarah Allen
Talk by Nick Kallen

There is no magic technology to solve scalability problems; however, an understanding of the fundamental computer science concepts and applying some proven techniques will allow you to develop scalable software. Nick's talk explored three fundamental concepts underlying scalability: distribution, balance, and locality.

Before we dive in, it is important to understand two important concepts: throughput and latency. Throughput is how much work can be done in a given amount of time. Latency refers to the elapsed time. For example, you have written some code that we'll call a "worker," which can independently complete a specific task, which we'll call a "job." You could have multiple workers on different jobs that can be distributed across different CPU resources without affecting each other. Suppose that 1 worker is able to complete a job in 1 second, we would say that 1 job/sec is the throughput (number of jobs per unit time). In this example, 1 second is the latency (elapsed duration from start of job to the end of a job). Latency is an efficiency question. Throughput is a scalability question. The focus of this talk is scalability and increasing throughput.

A SIMPLE NETWORK SERVICE

You may not have thought about it this way, but your website is a distributed network service. In order to illustrate the talk, Nick wrote a simple echo service with EventMachine. The output yields pretty boring jokes for a "joke server," but makes for a great simple service to illustrate scalability. The key code that does the work is in the "receive line" method:

```
def receive_line(line)
    result = "KNOCK KNOCK: #{line}\n"
    send_data(result)
end
```

This event service is very simple and the latency could be easily measured and the throughput simply calculated. Nick did a simple load test on it, using a custom TCP benchmarking system based on apache benchmark. The code above showed 8450 requests per second, not surprising since it does almost nothing.

Any service you would write would be more complicated: it would make system calls, allocate memory and typically call another service or do some other I/O which then makes the latency not so deterministic. To simulate this, Nick added some code to the inner loop:

```
  sleep rand *3 # an efective representation
of blocking i/o
  100000.times { Time.now } # memory alloc
+ system call

  # represents an intense loop
```

When he reran the benchmark, the throughput dropped to around 1 request per second. If thousands of users needed to use this service at once, we would need to run multiple instances of the service at the same time in order to process that many requests with reasonable latency.

DISTRIBUTION

In looking at how to scale, we need to understand how to distribute our services across the compute resources that we have (or how many machines we need to hit target performance). Specifically, we want to answer the questions: How many can we run in parallel? How many can we run per machine? How many can we run per core?

To answer these questions, we need to measure what's going on:

```
def receive_line(line)
    $stats.transaction do
```

```
  data,source_transaction_id    =    line.
split(';')
  $stats.set('source_transaction_id',
source_transaction_id)
  $stats.measure('job') do
```

Log everything extensively. Thread transaction ids through your logs.

```
    result = "KNOCK KNOCK: #{line}\n"
     send_data(result)
  send_data(result)
  end
  end
  end
```

The code uses a statistics library, statosaurus, that they use at twitter. (Nick's example, which is posted on gitbuh, contains a version of statosaurus which contains the key parts of the proprietary package). Nick strongly recommends that we log everything extensively. Thread transaction ids throughout your logs. This becomes essential for tracing down failed distributed transactions, for example, in SQL queries, HTTP headers, etc.

This example logs the following:

- a time stamp
- a transaction id
- wall-clock time: amount of elapsed real time
- system time: amount of time the process has spent in the CPU while in kernel mode
- user time: amount of time the process has spend in the CPU while in user mode

Note that system time + user time < wall-clock time. Since there is time spent waiting on I/O (simulated in our example by sleep) or if there are too many processes on the machine at the same time for the number of cores, your process will be waiting in the "run queue." That latter excessive context switching is what we want to investigate.

Generally if we take the wall-clock time and divide by the CPU time (system time + user time) we get the optimal number of processes per core. This leads us to a distribution strategy, which we will need to implement with some kind of load-balancing technique.

BALANCE

Nick talked about different mechanisms to balance load: TCP Proxy, DNS (compelling for a chatty protocol), client (has some serious drawbacks for maintenance/upgrades). In this case, proxy is an optimal solution.

First, he demonstrated the "random" strategy, sending new requests to a random worker. This is better than having no concurrency, but it isn't optimal. Then he tried "round-robin" where requests are sent sequentially across each worker in order. This sounds great, but only based on the incorrect assumption that each job takes exactly the same amount of time. It performed even worse. The best choice is a "least busy" strategy where the proxy sends a request to the worker with the fewest number of open connections.

LOCALITY

For true efficiency, never do the same work twice. You can significantly reduce latency for specific requests by caching. Nick demonstrated the addition of a primitive cache that can only store 2 values (to simulate resource starvation). This allowed him to demonstrate a modest performance improvement. By associating a certain class of request with a specific server, we can do much better. This takes advantage of "locality."

A good way to understand locality is with the analogy to tape drive: if you write close to where you last wrote or read, then access will be significantly faster due to spatial locality. The same applies to hard drives and databases.

A cache is a spatial locality that keeps the data near the code. Put the requests on processes where the data is most likely to be cached. Nick showed how similar requests can be funneled to the same server consistently. In that case, throughput jumped to several hundred requests per second as the cache hit ratio grew close to 100%.

CONCLUSION

Building a massively scaled, fault tolerant web service isn't easy, but it's definitely easier if you know which architectures work and which don't. The techniques we learned in this talk were to logging everything in your code, look at the resulting statistics and apply appropriate strategies based on a clear understanding of distribution, balance and locality.

END NOTES

Complete code from this talk: http://github.com/nkallen/gogaruco/

Originally published with less detail at: http://www.ultrasaurus.com/sarahblog/2009/04/magic-scaling-sprinkles

Special thanks to David Stevenson from Pivotal for his notes on the talk: http://pivotallabs.com/users/stevend/blog/articles/793-gogaruco-09-magic-scaling-sprinkles-nick-kalen

Thanks To All The Generous Sponsors Who Helped Us Make It All Happen

AT&T Interactive is devoted to creating compelling experiences that connect people, businesses and places so everyone can more easily engage with their world, wherever they work, live or play.

We have a passion for the Internet, mobile applications and emerging media platforms, and are proud to nourish a collaborative, tech-driven culture that promotes innovation, the entrepreneurial spirit and a desire to enhance everyone's daily lives.

As a tech-driven company, AT&T Interactive is always looking for passionate talented individuals who can solve difficult problems and develop new and existing products to exceed consumer expectations and drive value for our advertisers - whether it's online, mobile, interactive TV, video or other media platforms.

Rely on Engine Yard: 70 people, two data centers, and easy access to the cloud.

Engine Yard came about in early 2006 because we saw a genuine need: customers were developing business-critical Rails apps, but they didn't want to worry about deployment issues, nor did they want to hire IT staff to manage servers. Instead, customers wanted Rails-focused 24/7 operations support, great infrastructure, and a smooth path from one to one million users.

We handle deployment and maintenance, so you can focus on building your application and running your business. Our deep roots in the Ruby and Rails communities, combined with our commitment to driving the next generation of Ruby technologies, offer our customers the peace of mind of knowing their applications are managed by a team of true technical experts.

Engine Yard is invested in the future of the Ruby and Rails communities, and is helping foster and lead development by supporting open source projects like Rails, Merb and Rubinius.

New Relic is the leading provider of application performance management tools for Rails.

Time spent looking for Rails performance problems is time not spent writing code. We want you to fix those problems fast—so you can get back to what you love.

More than 1200 customers use RPM to monitor more than 12,000 Rails application instances, detecting performance problems and drilling down to their root cause.

For a free, fully-supported version of RPM go to www.newrelic.com.

Pivotal Labs: Discipline, collaboration, results.

At Pivotal Labs, we believe software development should enable your innovative vision, not constrain it.

Our sustainable development practice combines our diverse and brilliant team of Pivots with the very best in agile software management techniques.

Pivotal Labs partners with you in a collaborative and iterative process that not only produces the highest quality software, it also spreads our wealth of experience to your development team. Let our focus, expertise, and perspective inform and improve your development process throughout and beyond our partnership.

Pivotal Labs: come for the best software, leave with the best practices.

Scribd is the world's largest social publishing network, changing the way tens of millions of people discover and share original writings and documents.

Our innovative document reader technology, iPaper, turns written works into web documents, which can be shared on Scribd.com or embedded on any other website.

Many of today's leading companies and organizations are using Scribd — including the Barack Obama Presidential Campaign Team, The New York Times, Simon & Schuster, Random House, the World Economic Forum, FOXBusiness.com, Harvard University, Ford, Duke Medicine, and The Atlantic.

Launched in 2007, Scribd is backed by Charles River Ventures, Redpoint Ventures, Paul Graham's Y Combinator, the Kinsey Hills Group and several prominent angel investors.

Integrum emphasizes business value, speed to market and open communication to build remarkable Ruby on Rails web applications.

Our business model is simple, do things that matter. Whether it is for our clients or for ourselves, we craft dreams and visions into reality. We specialize in taking the impossible and through iterative design and development growing your ideas into reality. We trust you will find out what our clients and our team have already learned; that through Iterative Designs anything is possible.

Rails Kits provides developers with ready-made and well-tested code for getting projects off to a quick start.

Developers can save time, skipping the boring stuff and diving straight in to the fun stuff. Get your project done in less time with less work.

Scout is a dead-simple server monitoring solution powered by open-source plugins. Built by (and for) Rails developers, Scout shows what you need to care about and not the stuff you don't. Start monitoring your first server for free at scoutapp.com.

Marakana is a San Francisco-based open source training company. We were first to run Ruby on Rails training classes on the west coast.

Marakana provides hands-on, instructor-led training to software teams of large corporations and government organizations. Unlike other training companies, Marakana focuses exclusively on open source technology.

Our courses can also be customized to your specific needs, and delivered on-site, at your location for groups of three or more participants. For more info, visit marakana.com.

RubyRunCE is a free, GEM installed performance diagnostic tool for Ruby and Ruby-on-Rails applications.

It generates RSS/HTML reports/charts; traces code by method and SQL automagically including code you don't know; diagnose hanging, slow, and leaking code.

Comes with pdf/html documentation, and you OWN everything IN HOUSE.

Founded in 1991, ELC Technologies is powering the hottest Ruby on Rails web applications.

From Cisco to ESPN to Live Nation, ELC Technologies is bringing Ruby on Rails to business-critical applications. Our work and speed of implementation has received praise from the Rails core team.

To learn more, visit elctech.com.

Addison-Wesley Professional is a leading publisher of high-quality and timely information for programmers, developers, engineers, and system administrators. Our mission is to provide educational materials about new technologies and new approaches to current technologies written by the creators and leading authorities. Addison-Wesley is a part of InformIT, the online presence of the family of information technology publishers and brands of Pearson.

Apress is a technical publisher with more than 700 books in print and a continually expanding portfolio of publications. Apress provides high-quality, no-fluff content in print and electronic formats that help serious technology professionals build a comprehensive pathway to career success. Apress is part of Springer Science+Business Media.

Manning Publications publishes computer books for professionals—programmers, system administrators, designers, architects, managers and others. Our focus is on computing titles at professional levels. We care about the quality of our books. We consult with technical experts on book proposals and manuscripts, and we may use as many as two dozen reviewers in various stages of preparing a manuscript.

PeepCode Screencasts are a high-intensity way to learn Ruby on Rails website development.

Experienced developers have said that PeepCode is "fantastic and the price is a steal."

Designer/developers have said they are "exactly the type of instruction I need to pick up new things and understand how they work."

Pragmatic Screencasts help you quickly get up to speed on timely topics including Ruby, Rails, Sinatra, Objective-C, iPhone, and Erlang. In these video tutorials, you'll learn directly from the experts. You can download and watch DRM-free episodes when and where it's convenient for you.

With over 200 years experience in the publishing business, Wiley has been a leading source of information and understanding for generations of customers. From computers to cooking, travel to technology, and everything in between, Wiley has a book for you. If you can dream it, we'll help you do it.

Golden Gate Ruby Conference LLC
www.gogaruco.com
April 2009