Living in a Polyglot World

Ruby on the Client, Erlang on the Server

Christopher Brown
Chief Technology Officer
Twitter: @skeptomai, Email: cb@opscode.com

Early guy, Amazon EC2

Director of Engineering

Microsoft Edge Computing Network

OPSCODE

Chef

http://dlutzy.wordpress.com/2011/06/16/velocity-2011-day-1/

Agenda

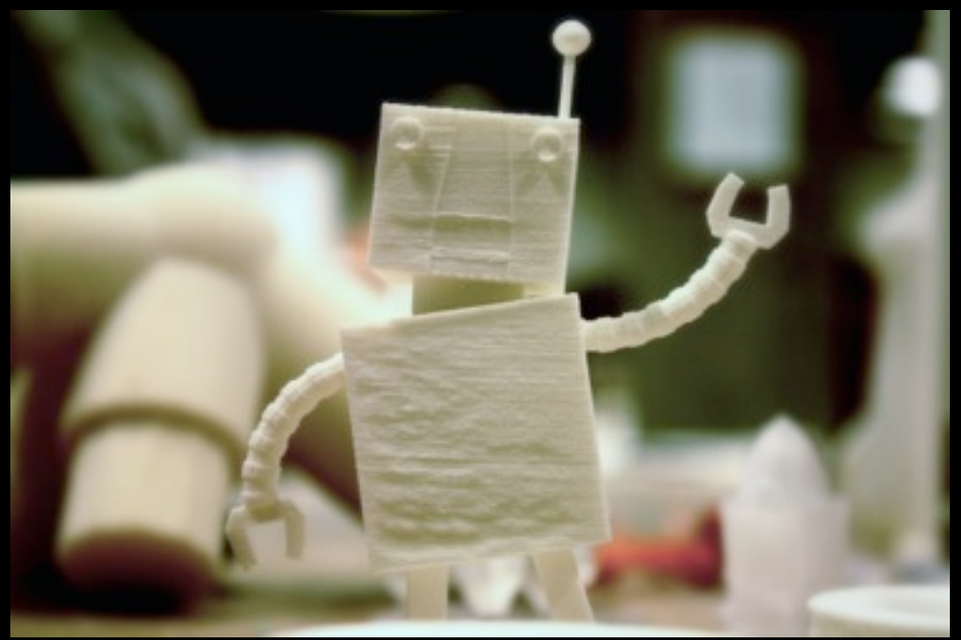
- What is Chef
  - from an implementor's perspective?
  - Why Ruby and Erlang?
  - Porting the server from Ruby to Erlang

What is Chef?

"Chef is like a little system admin robot… you tell it how you want your system configured and it will do all the dirty work."

- Probably Interactive

Architecture



- Code Repository
- Chef Server
- Chef Clients
- Data Bags
- Recipes and Cookbooks
- Roles and Run Lists

Recipe

- Recipes contain lists of resources evaluated in order

```
package "haproxy" do
  action :install
end

template "/etc/default/haproxy" do
  source "haproxy-default.erb"
  owner "root"
  group "root"
  mode 0644
  notifies :restart,
"service[haproxy]"
end

service "haproxy" do
  action [:enable, :start]
end
```

Resource

- Have a type
- Have a name
- Have parameters
- Take action to put the resource in the declared state

```
package "apache2" do
  version "2.2.11-2ubuntu2.6"
  action :install
end

template "/etc/apache2/
apache2.conf" do
  source "apache2.conf.erb"
  owner "root"
  group "root"
  mode 0644
  action :create
end
```

What is Chef?

...from an implementor's perspective

- Client Side Tools
  - command line
  - convergence engine
- Server API Implementation
  - publishing platform
  - scalable web service
  - fault tolerant

Client Design Goals

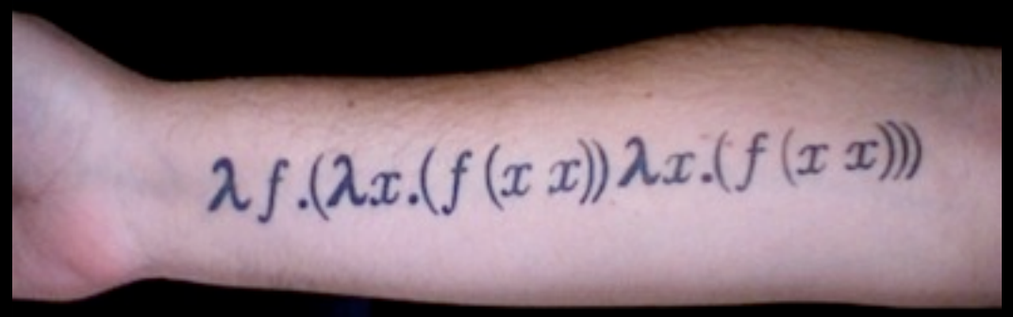
- easy to create command line tools
- easy to hack on (we've got a big open source community which is very important to us)
- startup time / resource consumption secondary
- nearly ubiquitous platform support (wide and deep)
- metaprogramming / internal / external DSL

Languages to consider

...on the client

- JVM
  - Java
  - Clojure
  - Scala
- Perl
- OCaml / ML
- Haskell
- Scheme / Lisp

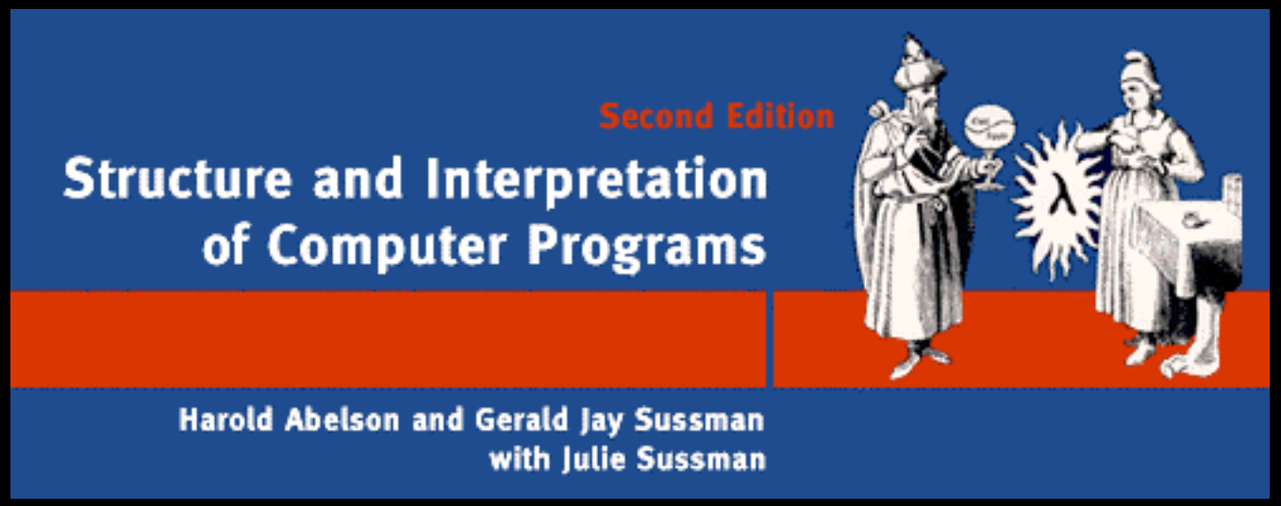
Goals of a DSL



- Translate Domain expertise to language of implementation
- Reduce syntactic / cognitive load

External vs Internal DSL

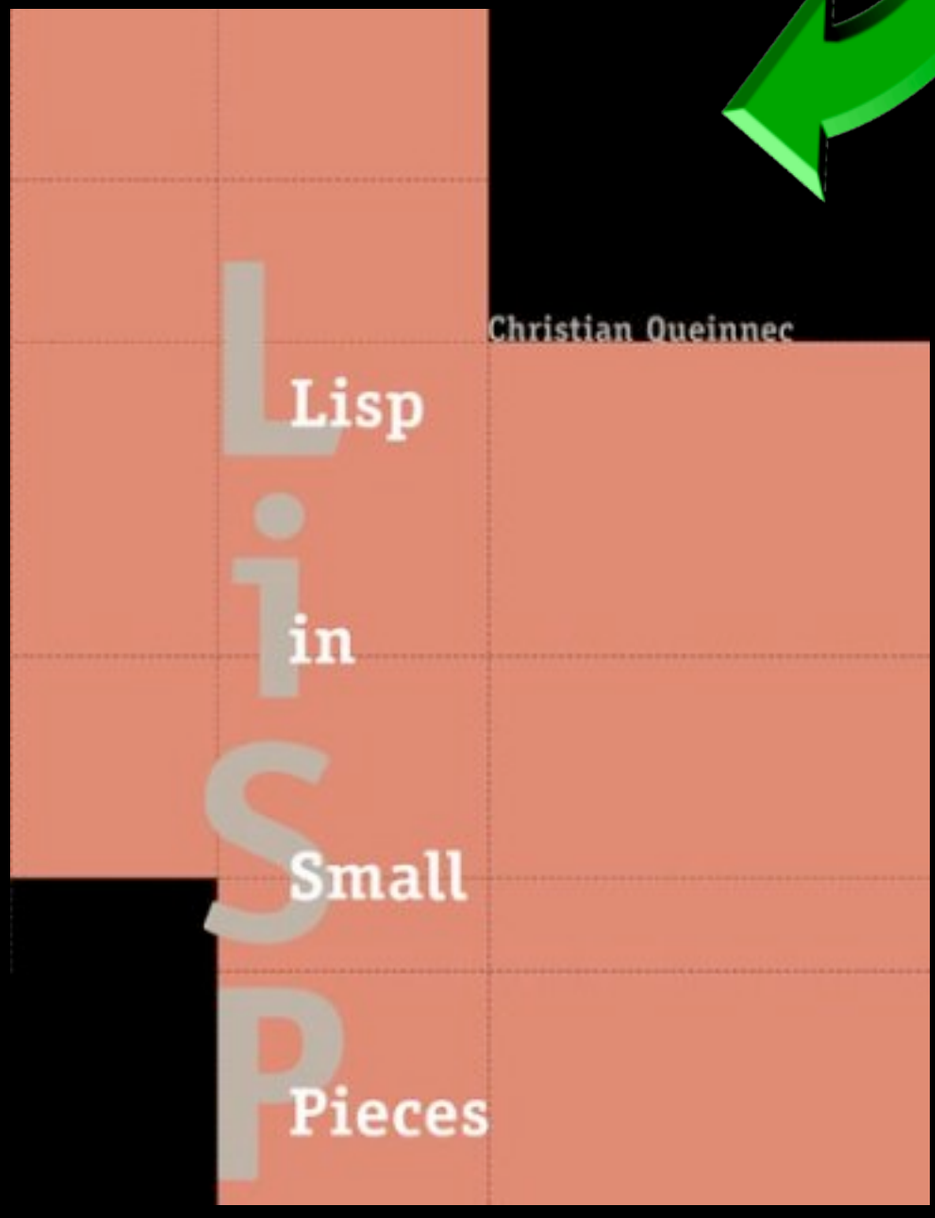
- Problem: How much of the original language do you carry along?
- What standard language constructs are necessary?
- Are they easy to understand / teach your DSL users?
- Cost of creating & maintaining grammar parser, etc.

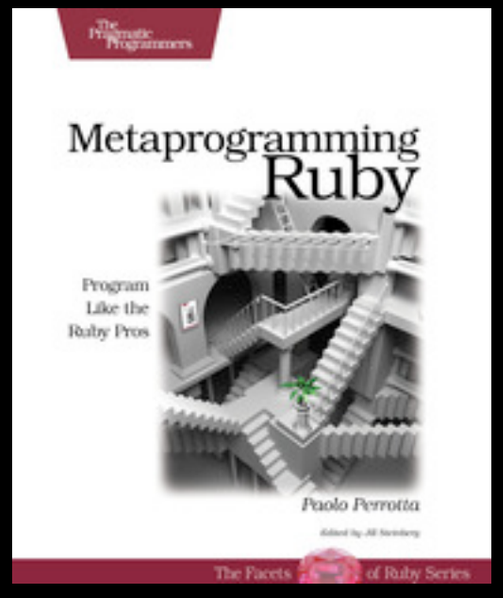


"SICP"

http://mitpress.mit.edu/sicp/

Lisp in Small Pieces





Metaprogramming Ruby

All I see are blondes, brunettes and redheads...

Internal DSL

- Exploit Metaprogramming
  - Heritage of Lisp, Smalltalk, Ruby
  - Requires Flexible syntax
- May inherit types, data structures

External DSL

- You are responsible for grammar
  - Lex / Yacc / ANTLR
- You are responsible for types, data structures and execution

Java?

"I think I realized a long time ago that Java is not a good enough language to implement applications."

http://ola-bini.blogspot.com/2008/01/language-explorations.html

http://www.javaworld.com/javaworld/jw-06-2008/jw-06-dsls-in-java-1.html

XML?

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- generated by N1 SPS -->
<component label='1.0.0' xmlns='http://www.sun.com/schema/SPS' name='tuxedo81.cont' version='5.1' description='For deploying
tuxedo81 domain for ocsccm 1.0.0' xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance' path='/ocs/ccm/1.0'
xsi:schemaLocation='http://www.sun.com/schema/SPS component.xsd'>
    <extends>
        <type name='system#container'></type>
    </extends>
    <varList>
        <var name='execUser' default=':[target:execUser]'></var>
        <var name='execGroup' default=':[target:execGroup]'></var>
        <var name='installRoot' default=':[target:installRoot]'></var>
        <var name='installPath' default=':[target:installRoot]'></var>
        <var name='platform' default=':[target:platform]'></var>
        <var name='platformPath' default=':[installRoot]/:[platform]'></var>
        <var name='environment' default=':[target:environment]'></var>
        <var name='environmentPath' default=':[platformPath]/:[environment]'></var>
    </varList>
    <componentRefList>
        <componentRef name='deploy_tuxedo81_dir'>
            <argList installPath=':[installRoot]' installGroup=':[execGroup]' installDiffDeploy='FALSE' installPermissions='754'
installName='tuxedo81' installUser=':[execUser]' installDeployMode='REPLACE'></argList>
            <component name='tuxedo81' path='/ocs/ccm/1.0/components' version='1.2'></component>
        </componentRef>
    </componentRefList>
    <installList>
        <installSteps name='default'>
            <install blockName='default'>
                <allNestedRefs></allNestedRefs>
            </install>
            <execNative userToRunAs='root'>
                <inputText><![CDATA[
#cd :[installRoot]
#find :[platform] -type d -print | xargs chown -R :[execUser]::[execGroup]
        ]]></inputText>
                <exec cmd='sh'></exec>
            </execNative>
        </installSteps>
    </installList>
</component>
```

Erlang?

```
#!/usr/bin/env escript
%% -*- erlang -*-
%%! -smp enable -sname factorial -mnesia debug verbose
main([String]) ->
    try
        N = list_to_integer(String),
        F = fac(N),
        io:format("factorial ~w = ~w\n", [N,F])
    catch
        _:_ ->
            usage()
    end;
main(_) ->
    usage().

usage() ->
    io:format("usage: factorial integer\n"),
    halt(1).
```

Perl?

```
use Chef;

resource file => '/tmp/foo', sub {
   my $r = shift;
   $r->owner('adam');
   $r->action('create');
};
```

Common Lisp?

```
(require 'chef)

(resource :file "/tmp/foo"
          :owner "cb"
          :action :create)

(resource :file (concatenate 'string
                             "/tmp/"
                             (node-attributes :hostname)
                             "-made-with-lisp")
          :action :create)

;; or

(file :path "/tmp/foo"
      :owner "cb"
      :action :create)

(file :path (concatenate 'string
                         "/tmp/"
                         (node-attributes :hostname)
                         "-made-with-lisp")
      :action :create)
```

Server Design Goals

- Horizontally scalable
- Highly concurrent
- Stateless request handling
- Fault tolerant

Languages to consider

...on the server

- JVM
  - Java
  - Clojure
  - Scala
- Perl
- OCaml / ML
- Haskell
- Scheme / Lisp

Chef Server API

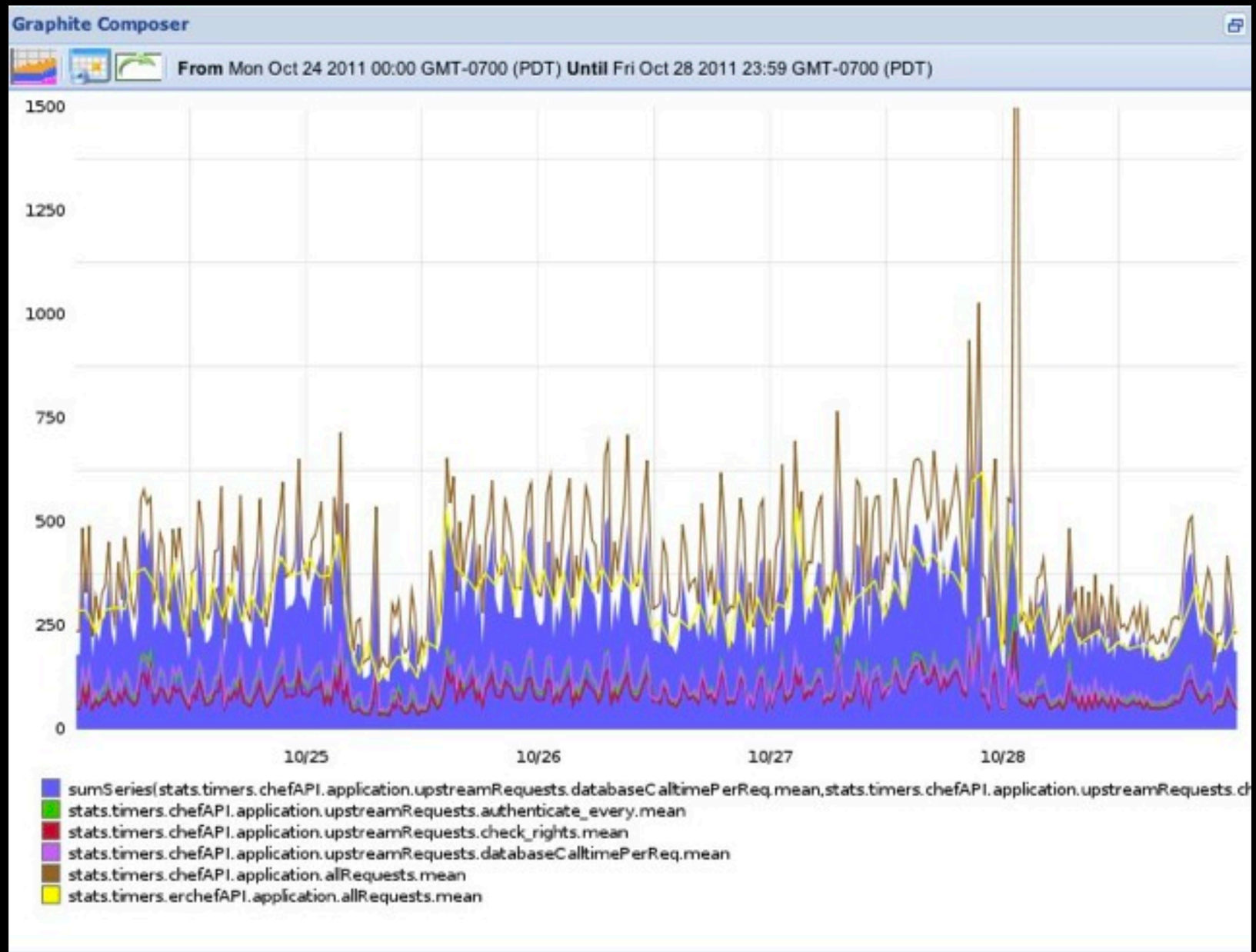
- Merb, Ruby, Unicorn, Nginx
- Stateless, horizontally scalable
- Talks to
- CouchDB,
- authorization service (Erlang),
- Solr

Typical Chef Server API Request

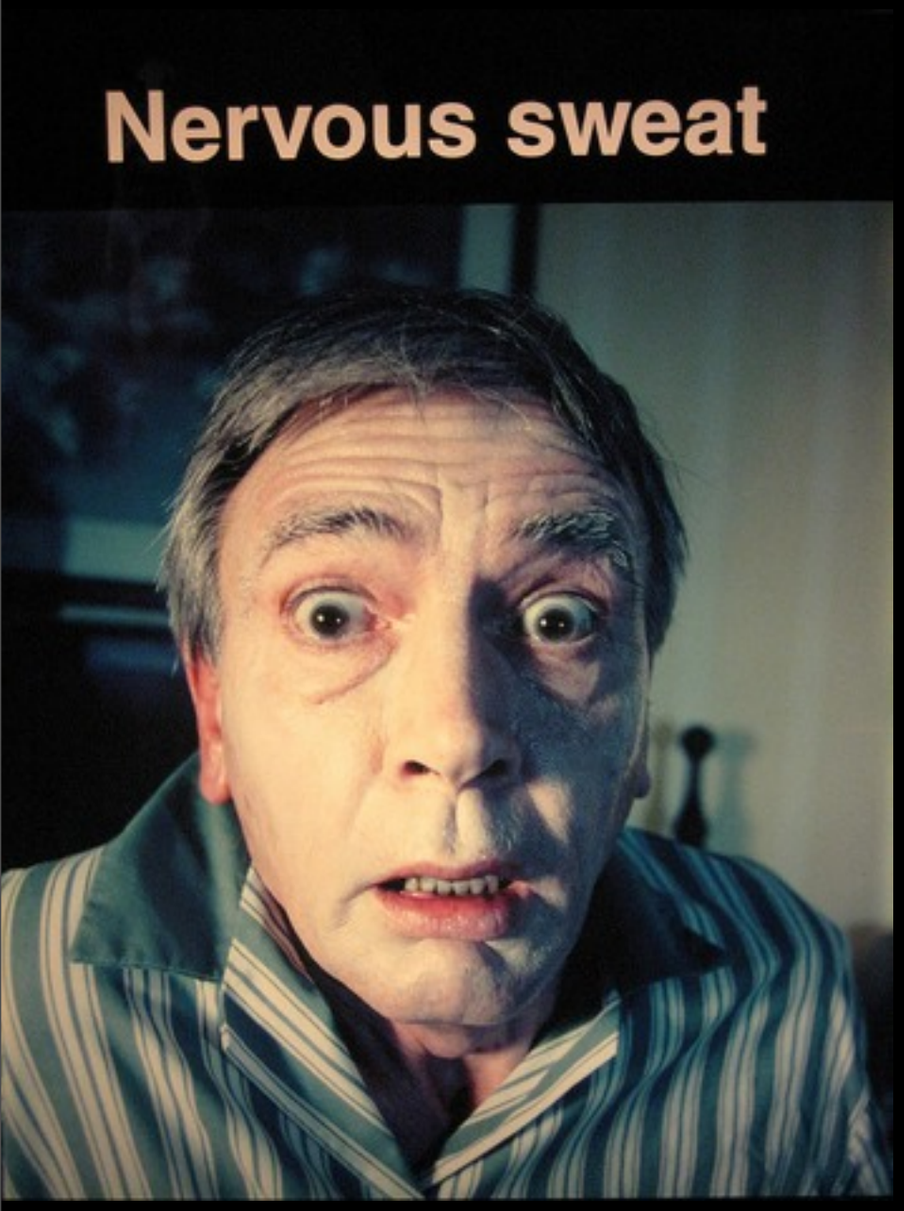
1. User public key for authentication
2. Node data from CouchDB (median 22K, 3rd Qu. 44K)
3. Authorization check
4. POST, GET, PUT, DELETE

Average Chef Server API Response Times

500 ms

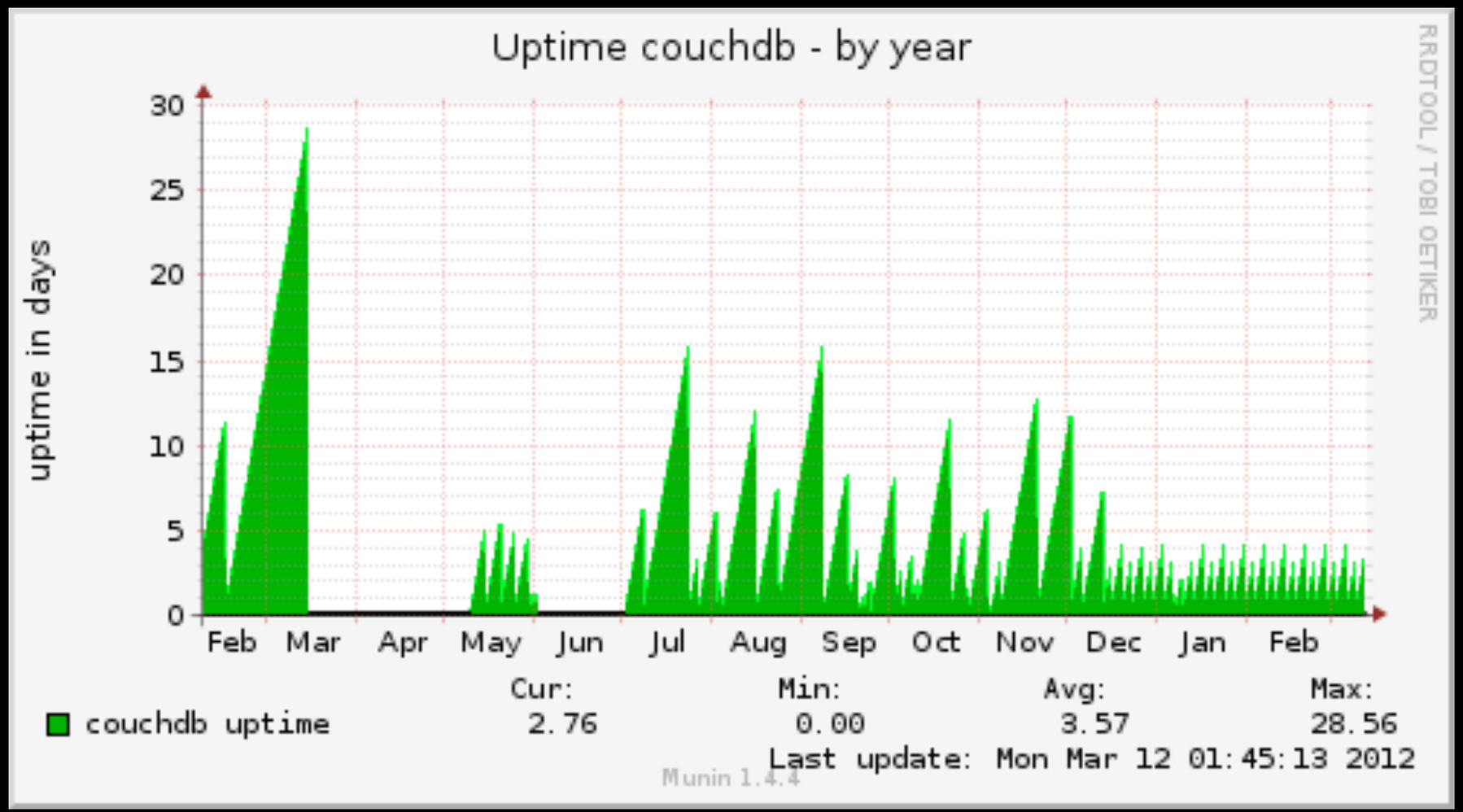


CTO



Slow, Irregular, and Out of Control

CouchDB Uptime



Heavy on system resources

How much RAM should it use?

60 req/sec × 44K =

2.7MB

2.7MB data + code

+ copies...

27MB?

100MB

at rest, after startup

Concurrency?
One request per worker.

204 MB

per unicorn worker

under load

12 workers per server

8 servers

12 × 204 MB = 2.4 GB

8 × 2.4 GB =

19.2 GB

for pulling JSON out of a database and returning it

Unicorns Eat RAM


ERLANG

basho wiki

How It Works

Quickstart

Demo

Dispatching

Request

Resource

Streamed Body

Debugging

Flow Diagram

Referential Transparency

Home > Webmachine

Webmachine

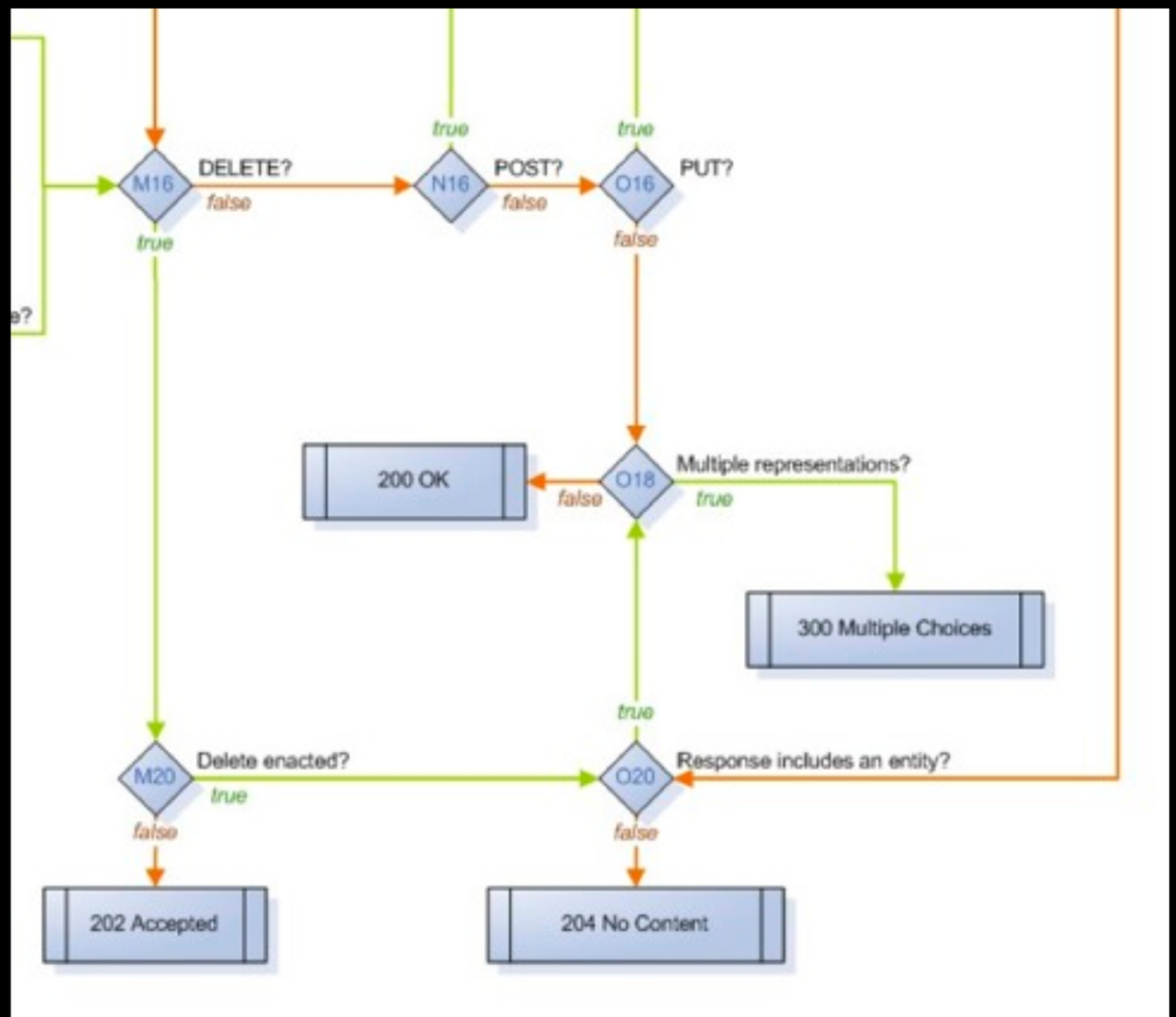
REST Toolkit

Webmachine is not like the web frame
in Erlang, Webmachine is an applicati
management provided by mochiweb, a

Resources

A Webmachine application is a set of r
object-methods, infinite-server-loops, e
are relatively easy to understand and e

These functions give you a place to de
that the first-class things on the Web a
constrained.



How did we do?

Erlang	Ruby
idle 19MB	100MB
loaded 75MB	204MB

Erlang	Ruby
600MB	19.2GB

We win!

But wait! There's more.

Where is Ruby API spending time?

DB calls?

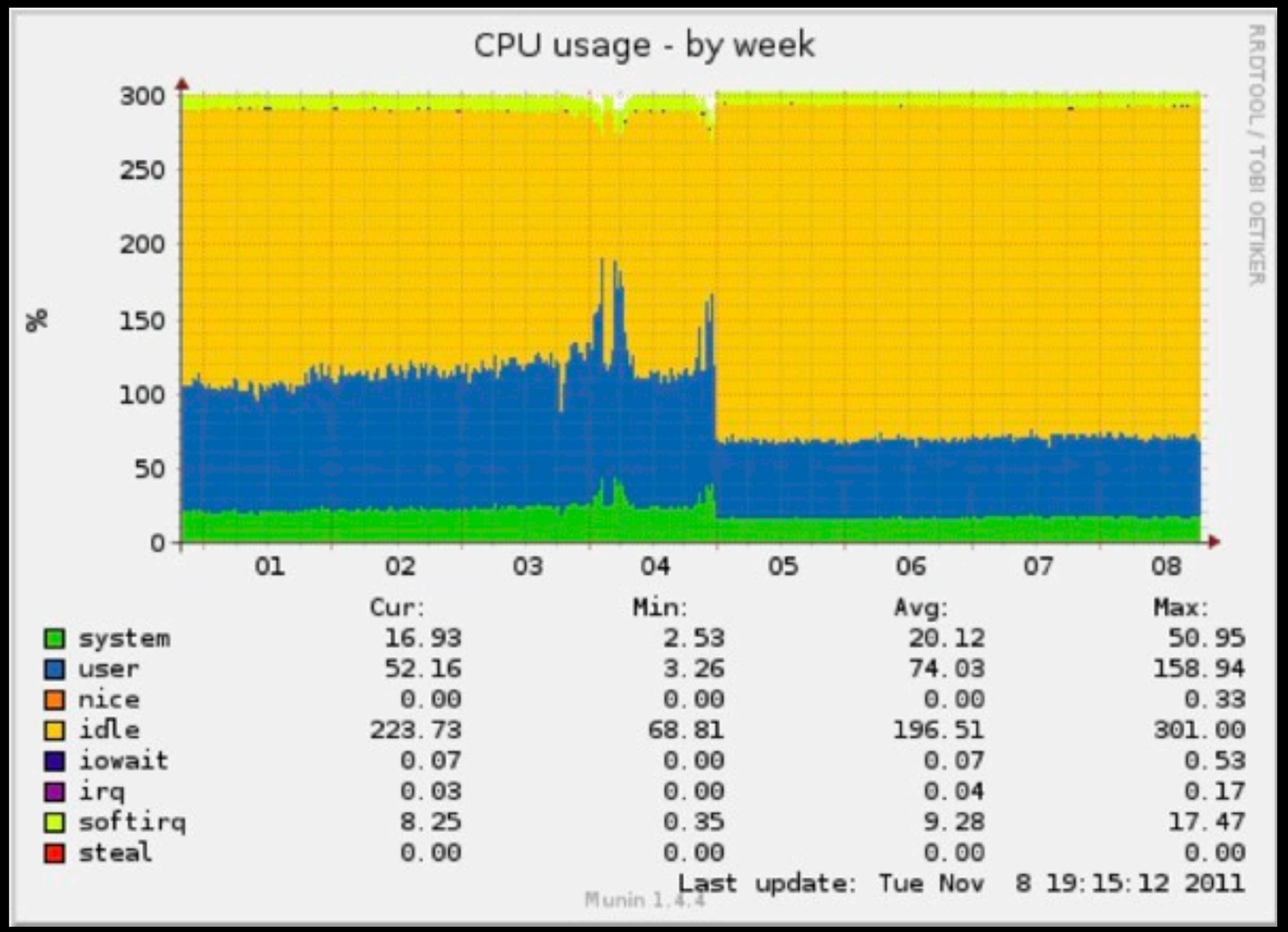
JSON parsing/
rendering?

Crypto?

Garbage Collection?

Garbage Collection!

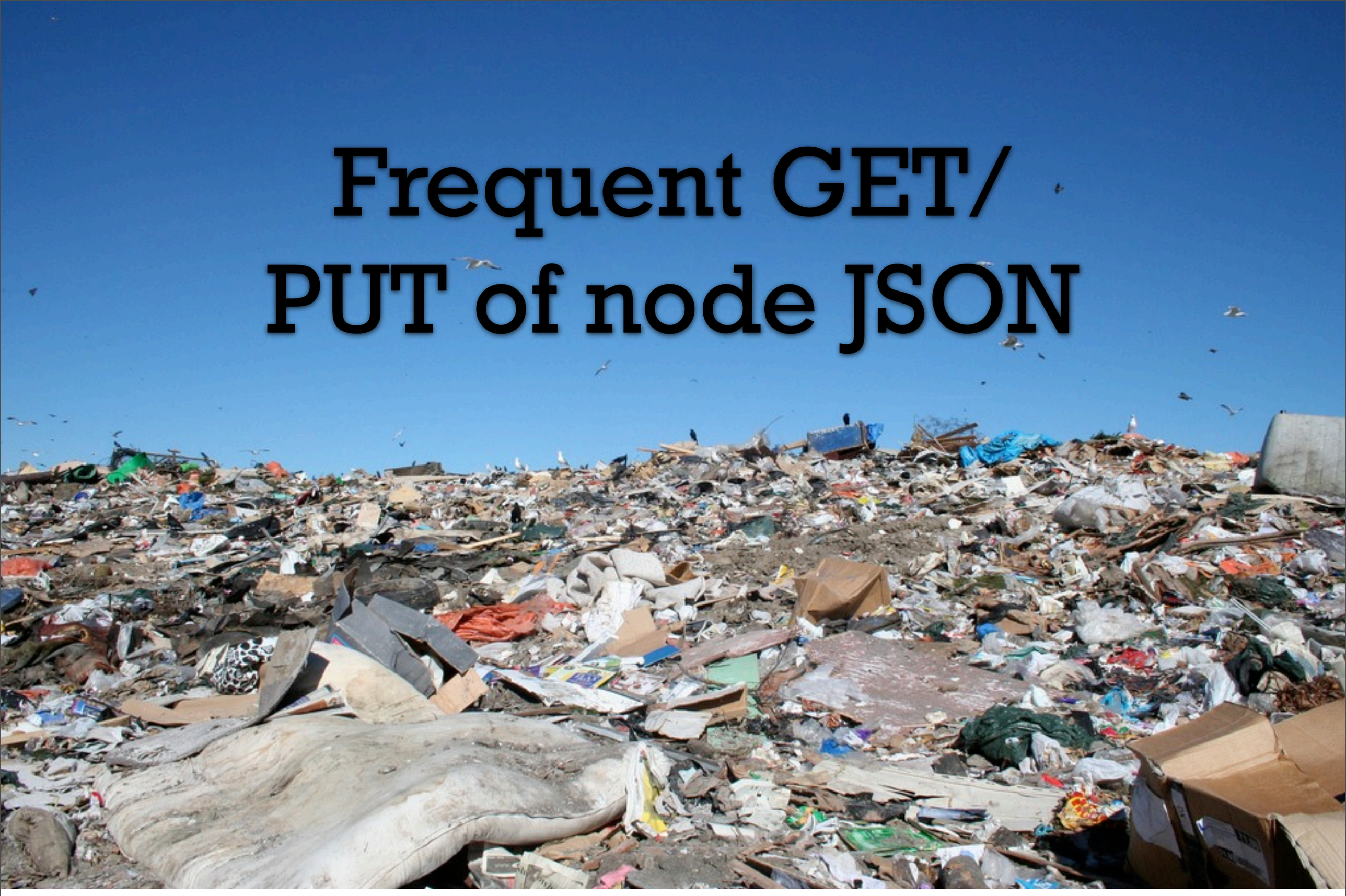
>40% CPU in GC



CPU Usage on Chef Server


CouchDB
relax

Frequent GET/ PUT of node JSON



compaction


No concurrency accessing a single database
(until recently)

Database replication unreliable for 1000s of databases.

Motivation: Why not CouchDB?

File handle and memory resource leaks

Motivation: Why not CouchDB?

It became an operations “thing”

What we need in a data store

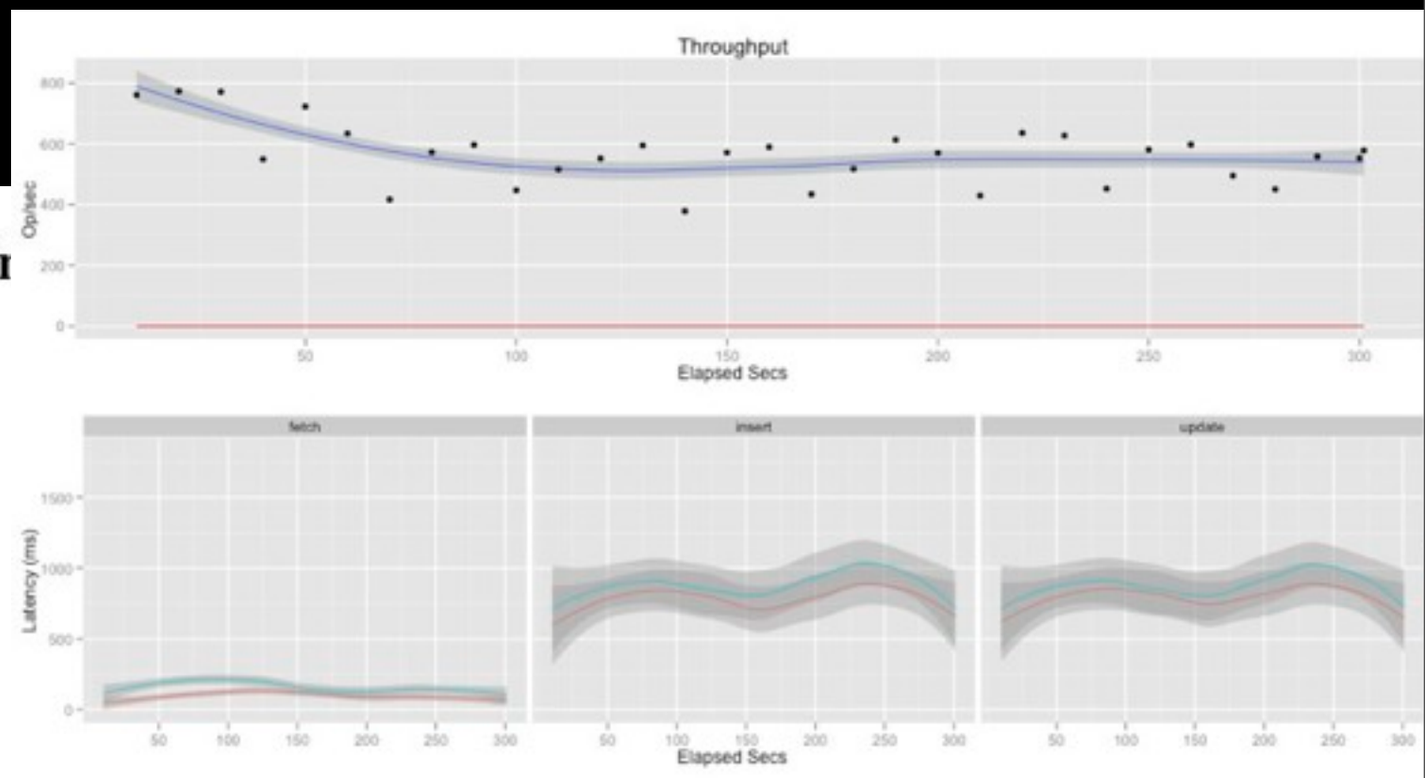
- Happy with write heavy load
- Support for sophisticated queries
- Able to run HA

Did you consider NoSQL database X?

Yes, but we also asked:

Why not SQL?

Measure!



Testing Node Operations i

Table of Contents

Testing Node Operations in MySQL

We tested two scenarios: insertion only and a combined insert, fetch, and update test that was run after the insertion test with a base of twice current production node count).

Tests were performed using `basho_bench` with custom drivers and generators (see Software Used below). The back-end Private C was used to host the database. The load generation was run on the front-end Private Chef PoC box (172.28.5.24).

Test system

Private Chef PoC boxes. Test client on front-end, MySQL on back-end.

MySQL installed from Percona's apt repo:

```
mysql  Ver 14.14 Distrib 5.5.15, for Linux (x86_64) using readline 5.1
```

This was installed from Percona as:

```
percona-server-server-5.5/lucid uptodate 5.5.15-rel21.0-158.lucid
```

Insert 40K nodes test

The insert test was run for five minutes or until 40K nodes were added to the database.

Generated nodes are randomly assigned to one of 500 orgs and one of four environments. Node names and IDs are randomly generate

basho_bench

So we replaced CouchDB with MySQL

...while the system was running

Live Migration:
Starts out easy!

Live Migration in 3 Easy Steps

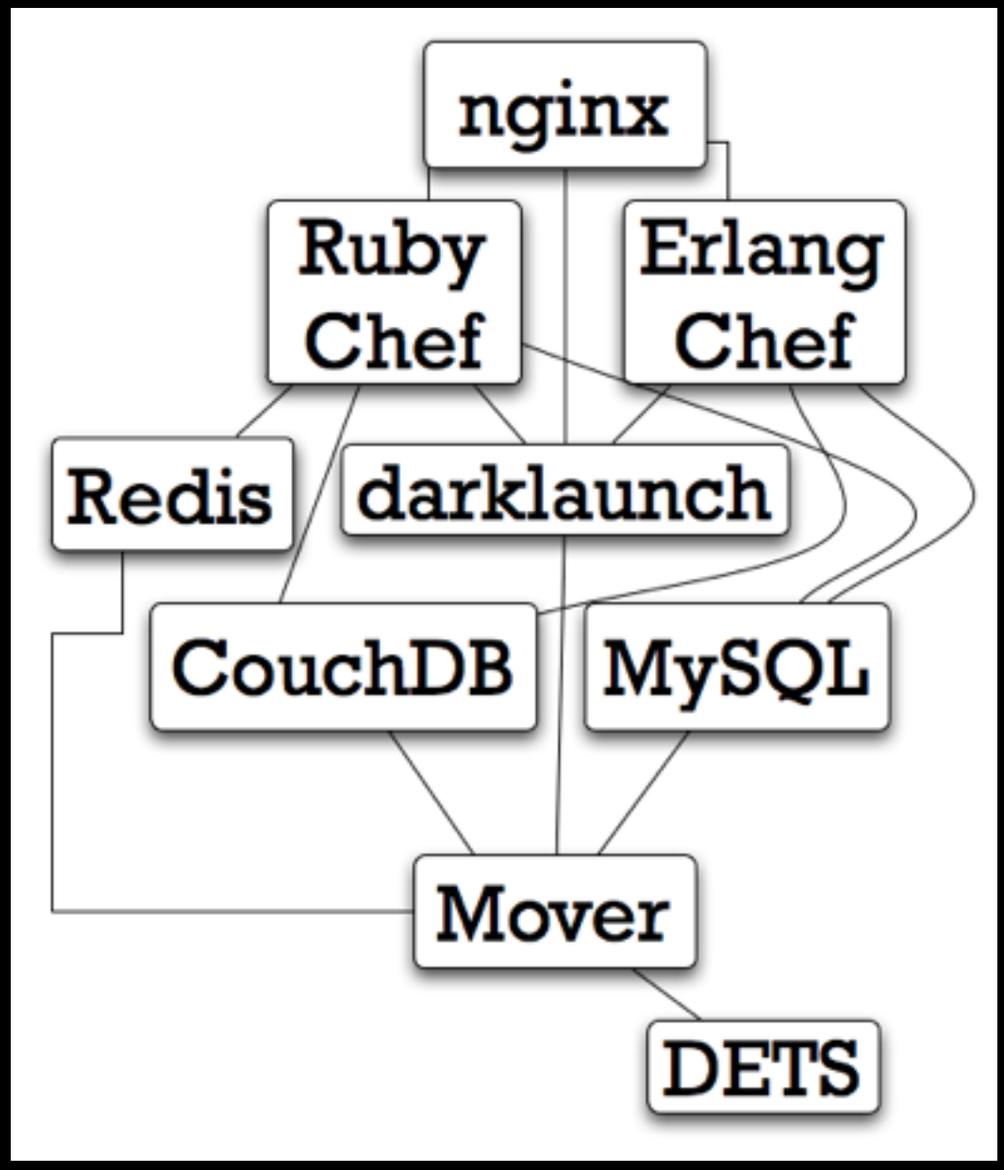
1. Put org into read-only mode
2. Copy from CouchDB to MySQL
3. Route org to Erchef

It Gets Harder

Real World Hard

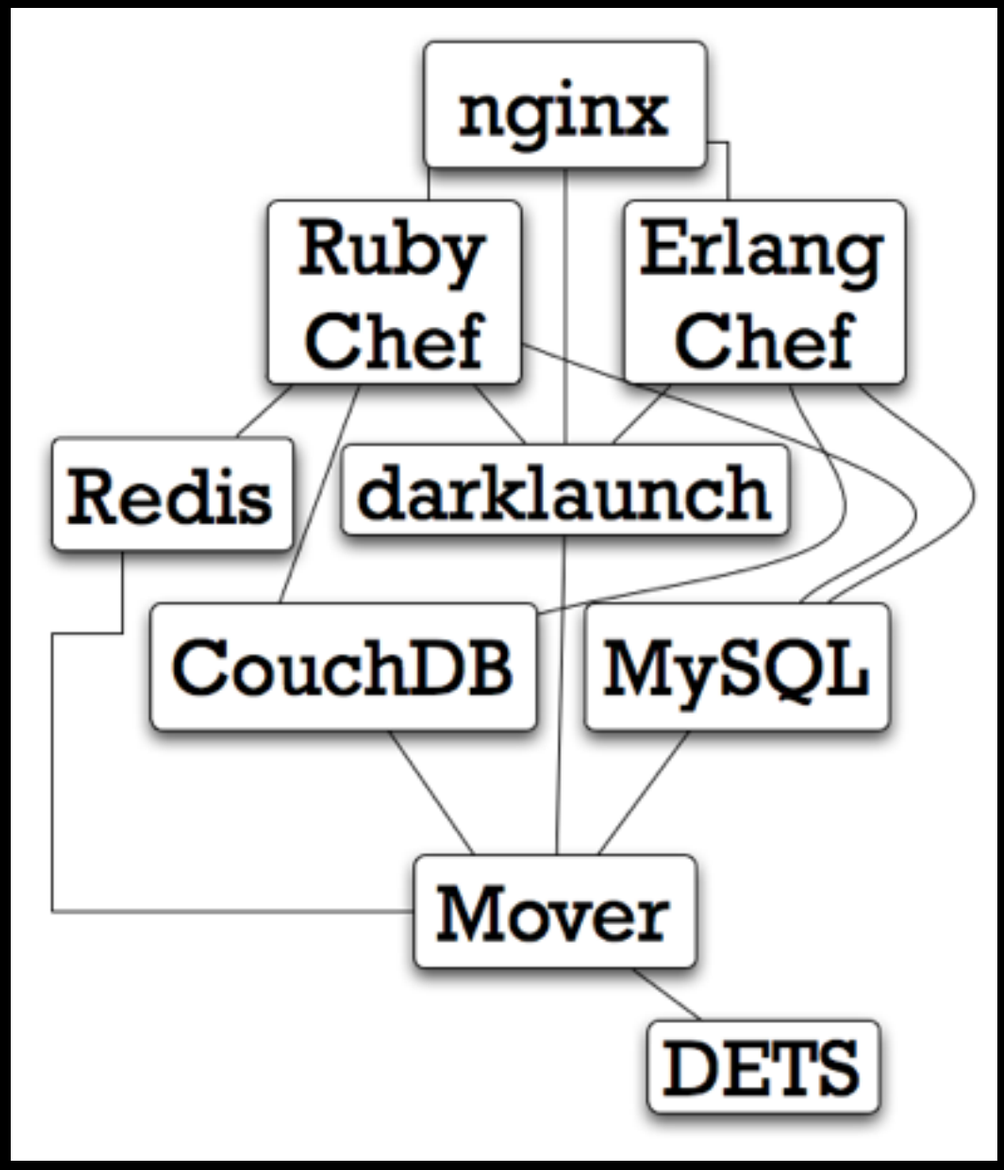
Migration Tool

1. Coordinate feature flippers and load balancer config
2. Move batches of orgs through migration
3. Track status of migration and individual orgs
4. Resume after crash



Migration Tool

1. Track inflight write requests
2. Put org into read-only mode
3. Wait for inflight write requests to complete
4. Migrate org data
5. Reconfig/HUP load balancer
6. Handle errors



Scripting with gen_fsm

- Helper methods → states
- Server state and supervision tree make crash recovery easier
- Free REPL

OTP + gen_fsm =:= Happy Migration Tool

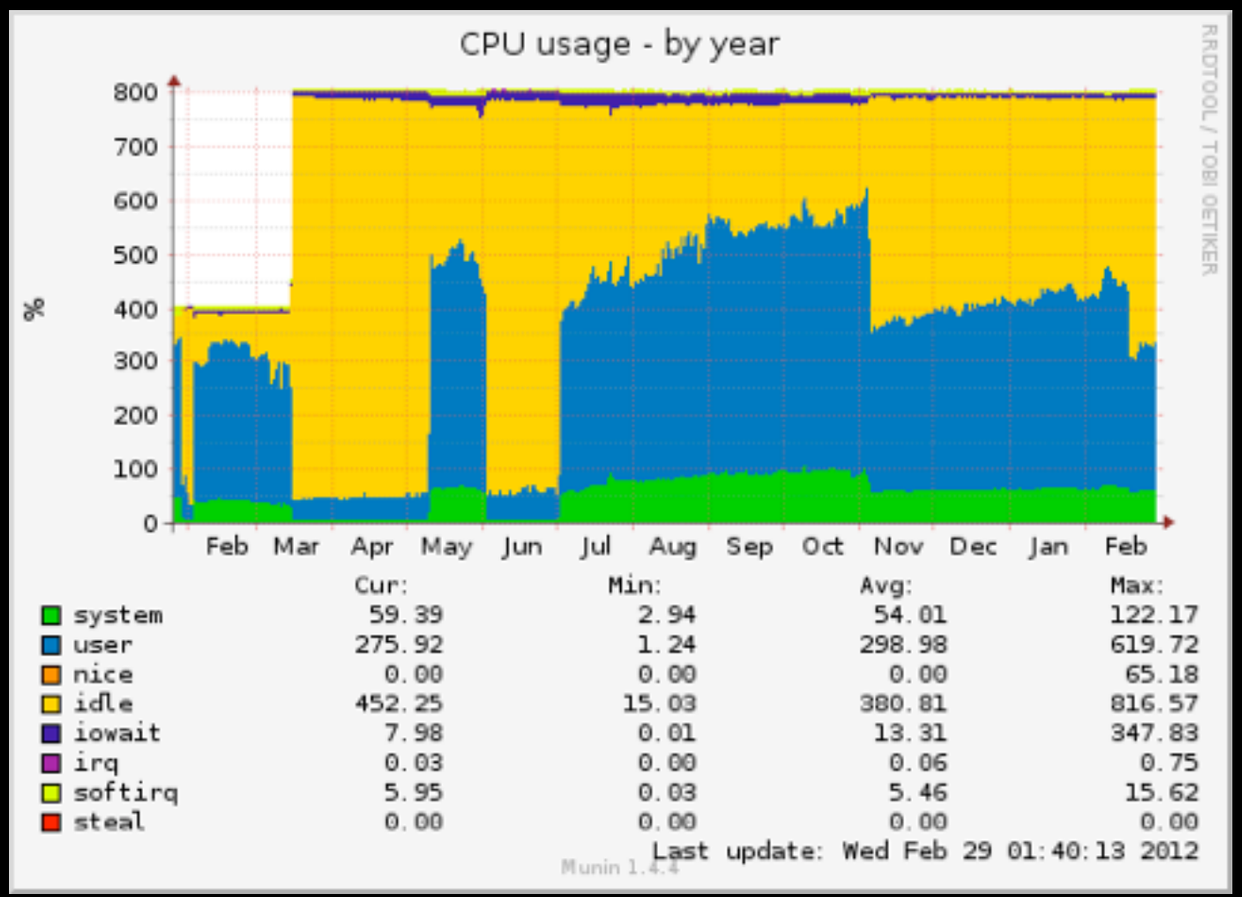
	Organization	Robustness
state functions	✔	
state record	✔	✔
manager/worker processes	✔	✔
supervision tree		✔
DETS local store		✔

No migration plan survives contact with production

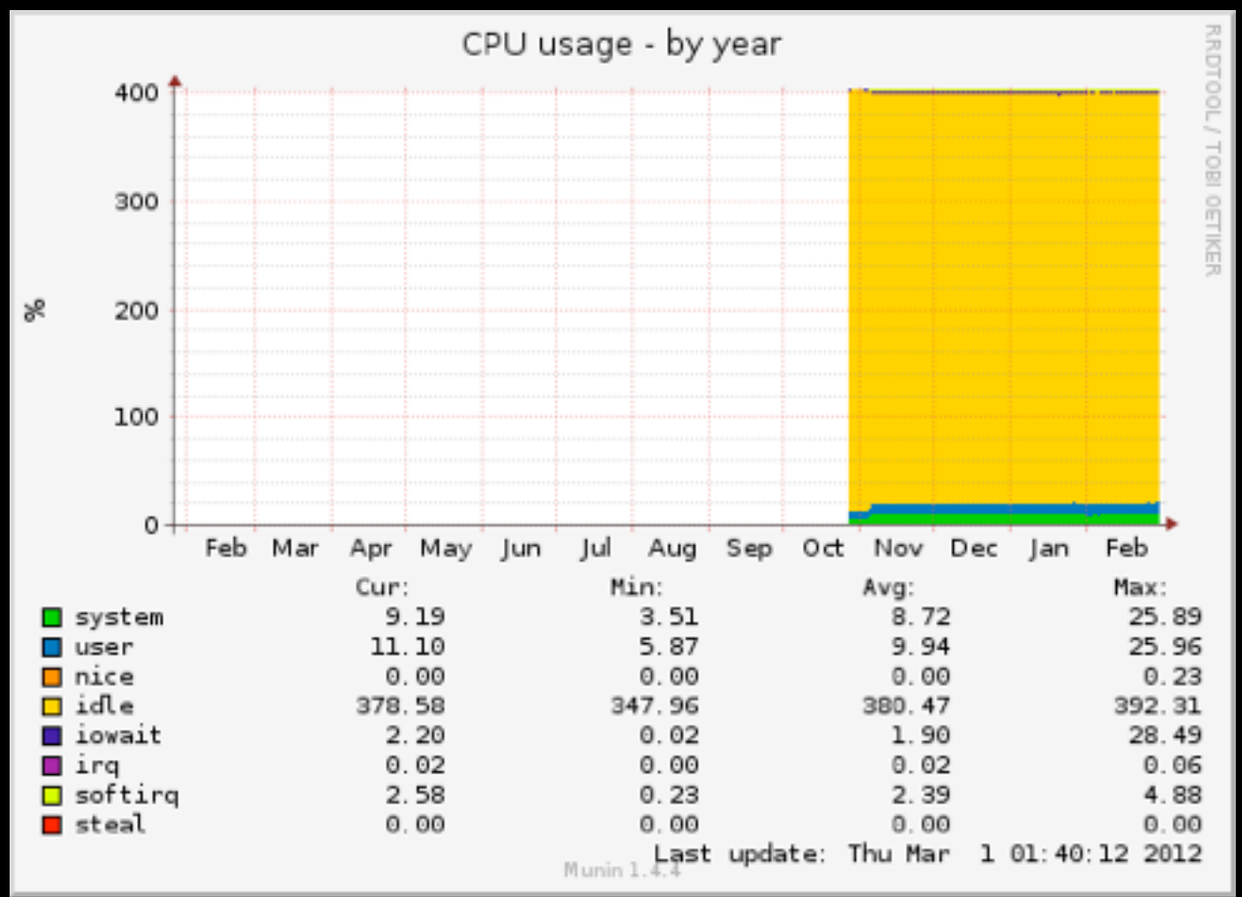
http://en.wikiquote.org/wiki/Helmuth_von_Moltke_the_Elder

Database CPU

CouchDB

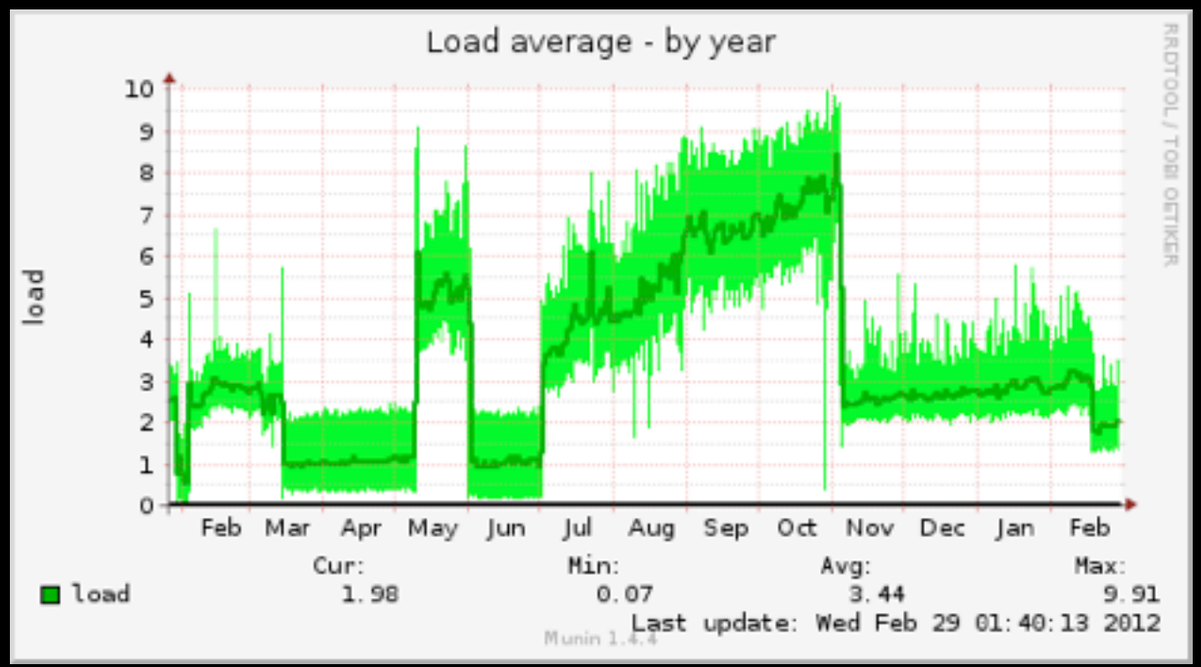


MySQL

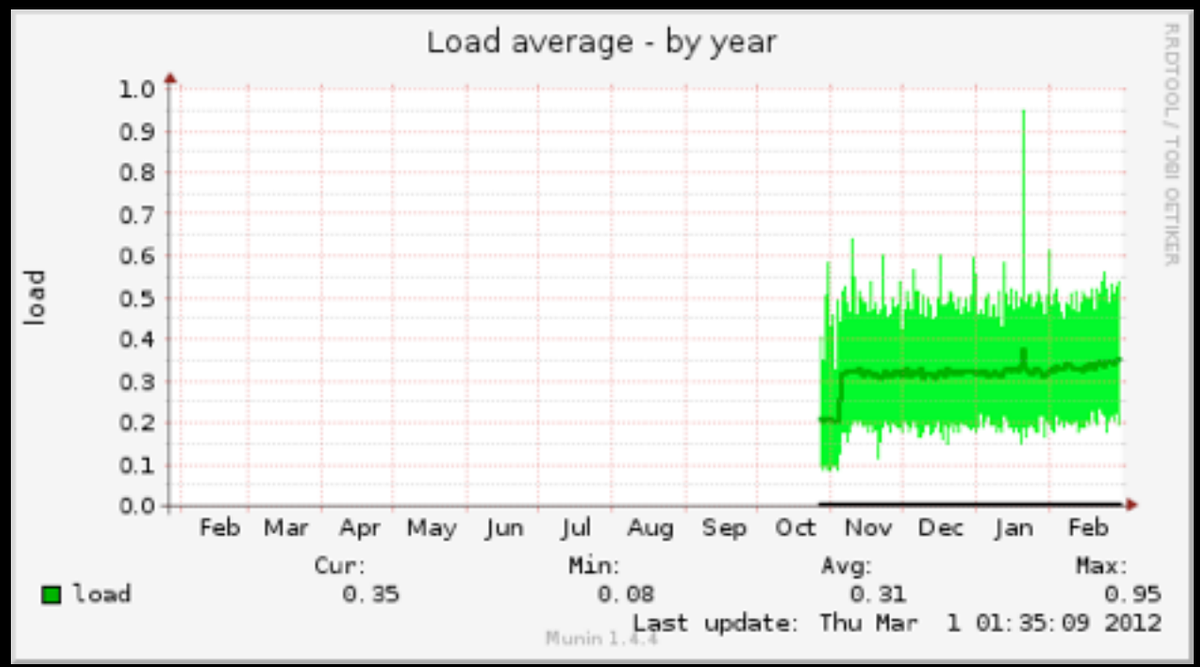


Database Load Average

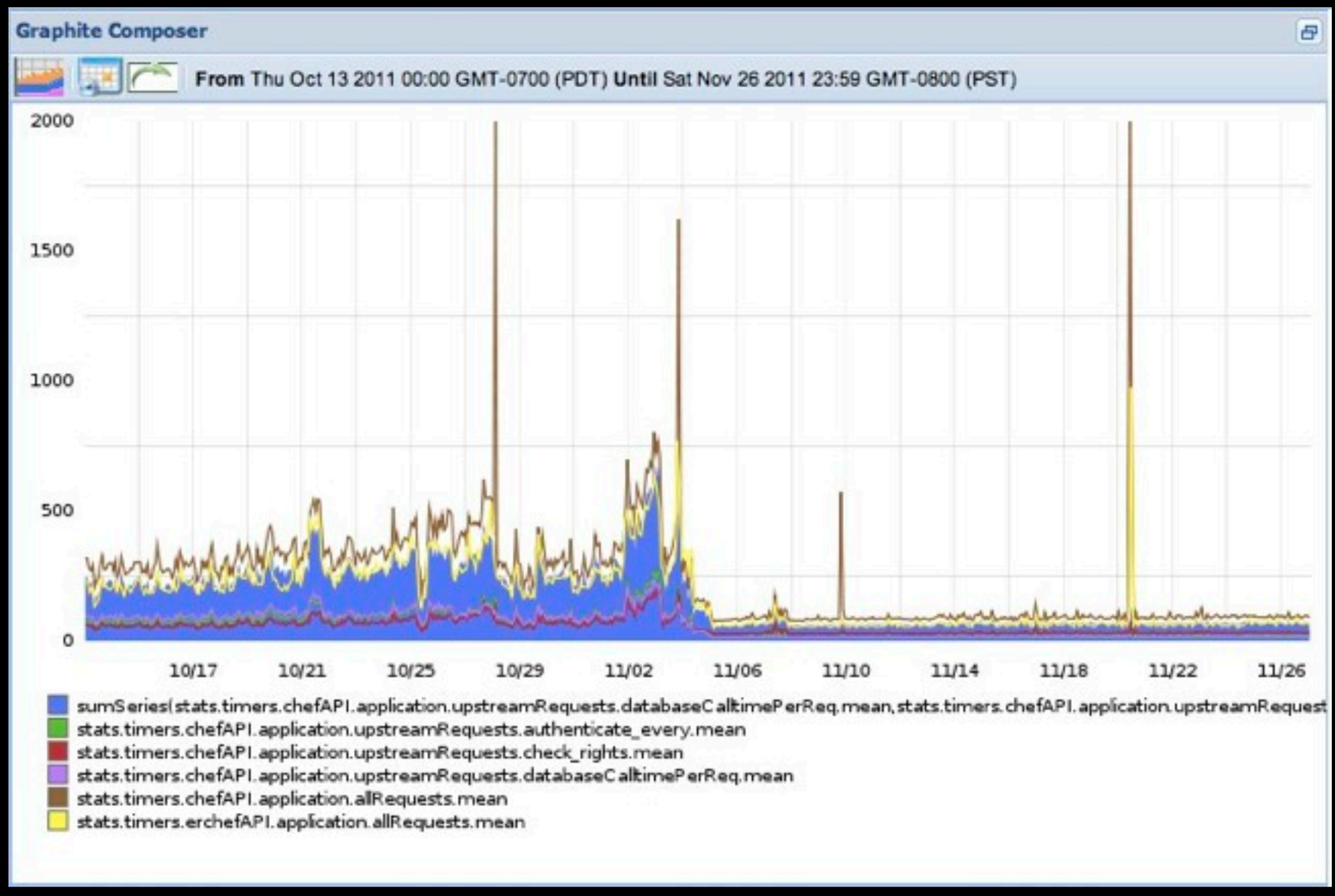
CouchDB



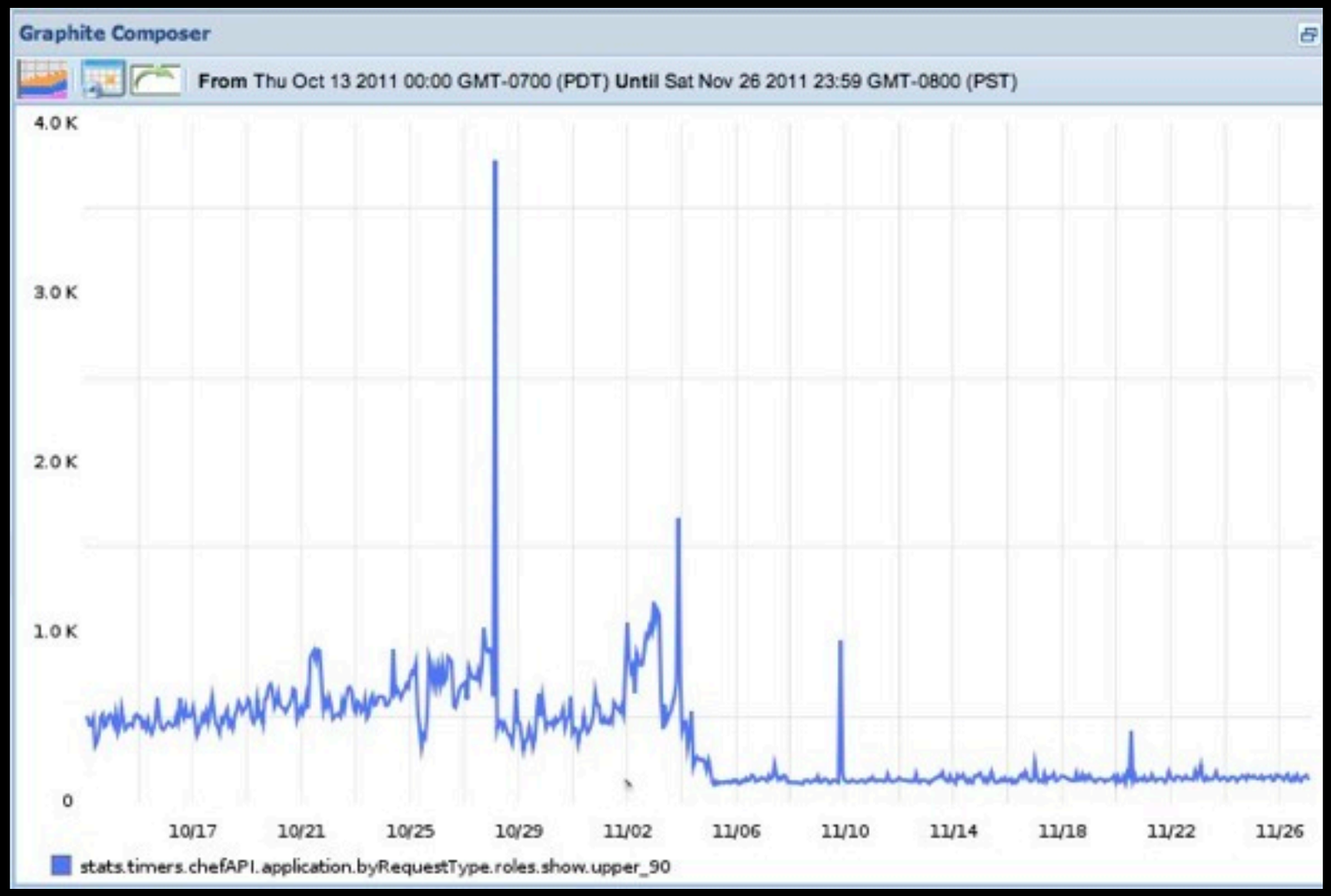
MySQL



API Average Latency

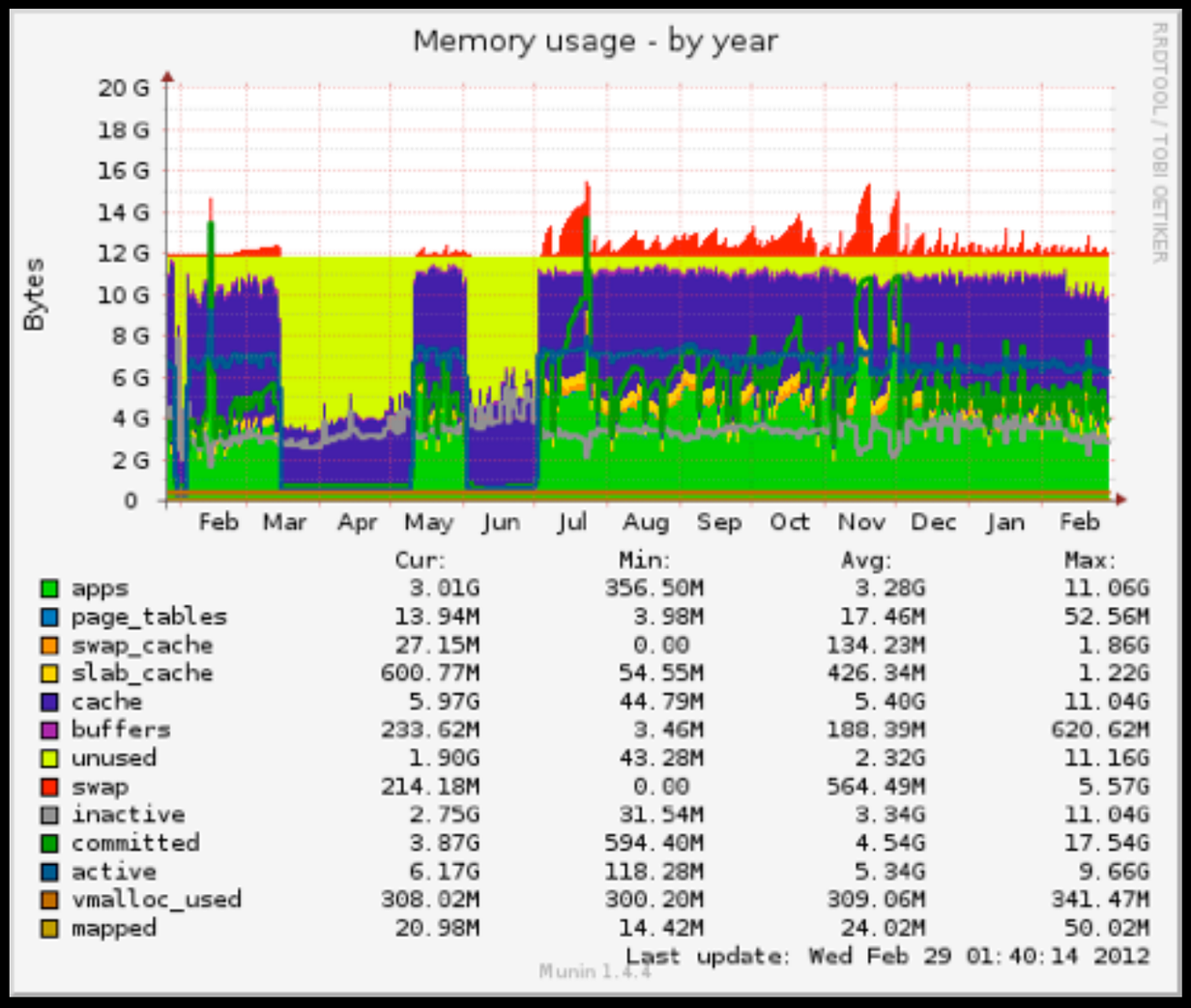


Chef Server Roles Endpoint 90th Latency

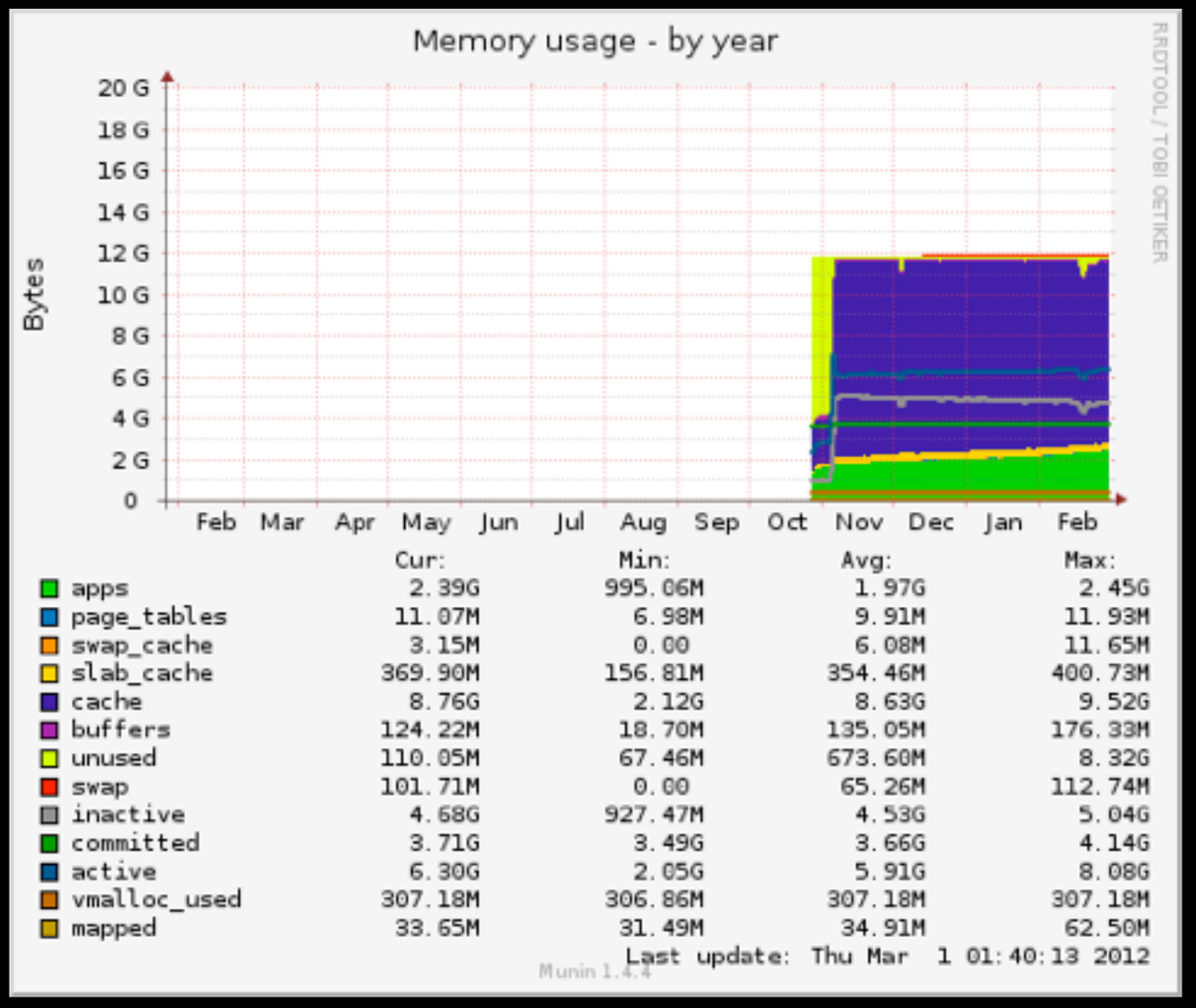


Database Memory

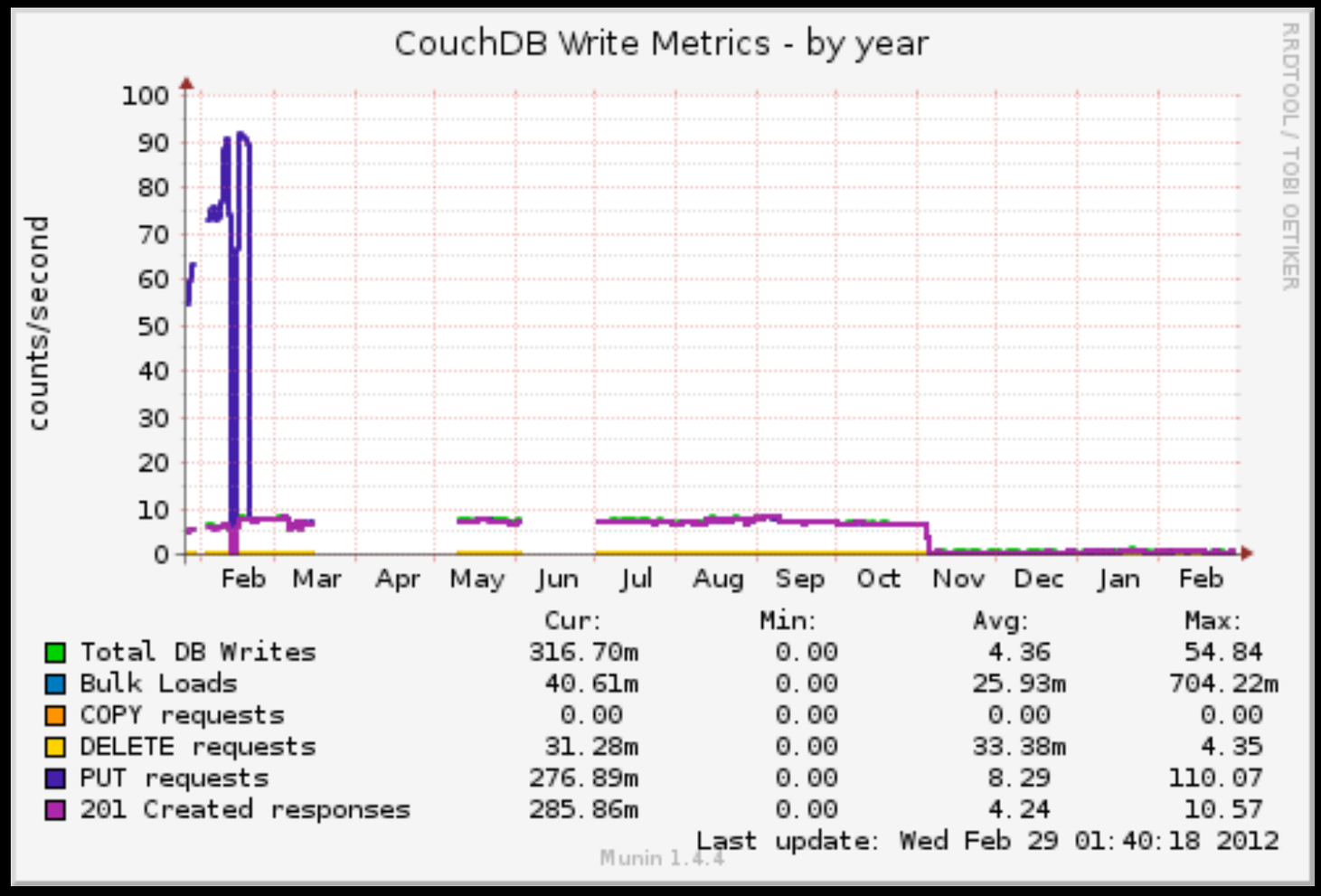
CouchDB



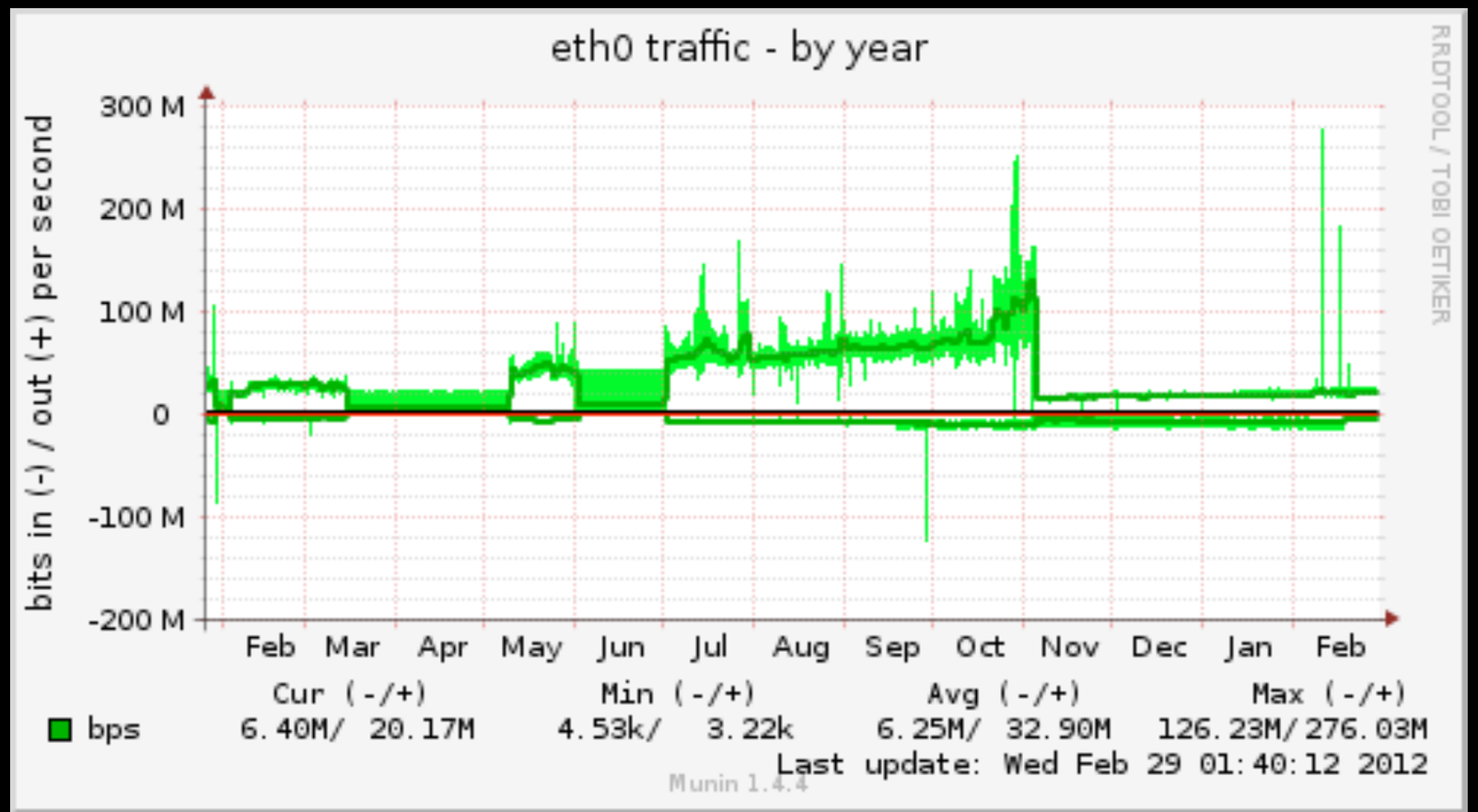
MySQL



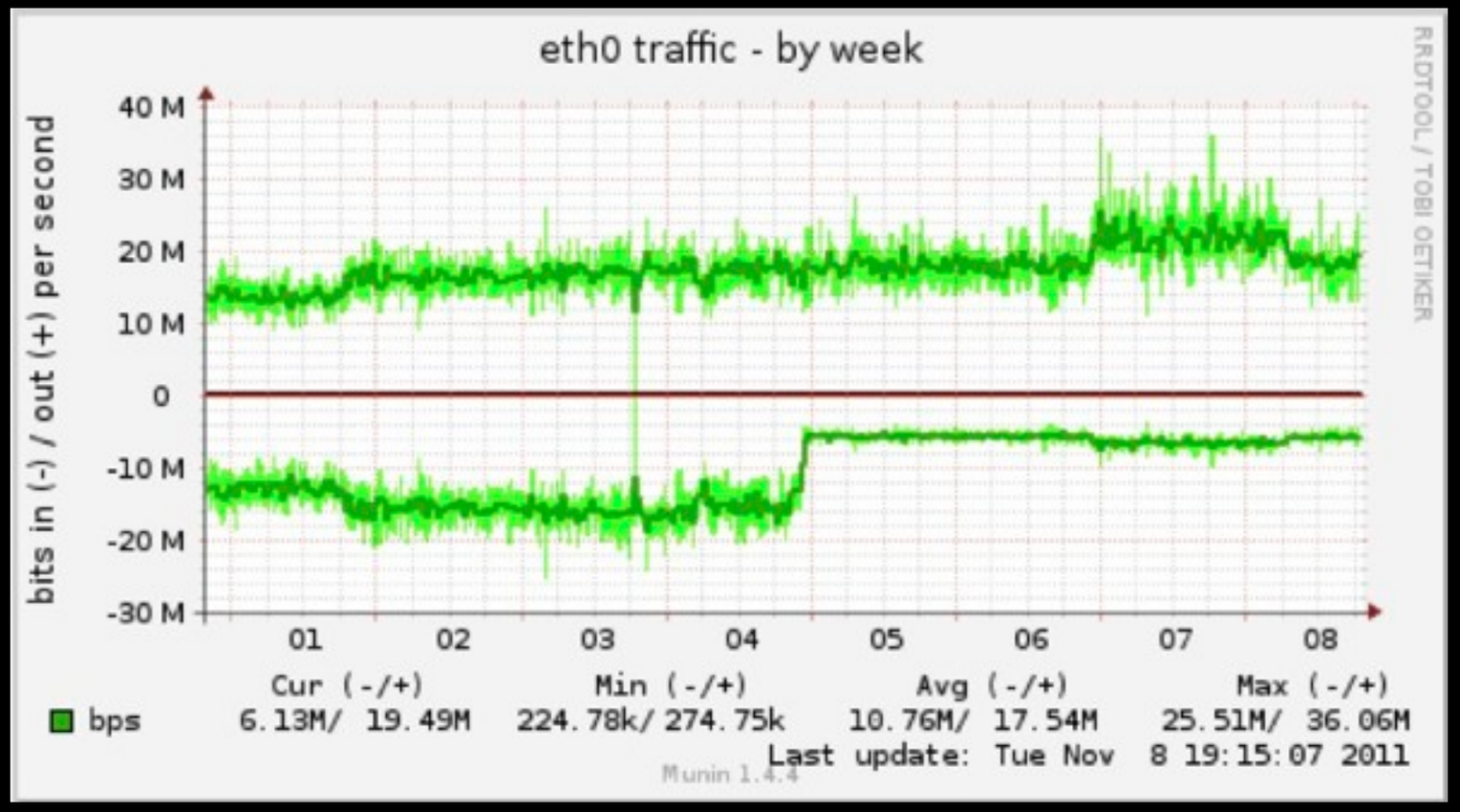
CouchDB Write Requests



CouchDB Network Traffic



Network traffic on Chef Server



The Road Ahead?

OPSCODE

Christopher Brown
Chief Technical Officer
Twitter: @skeptomai, Email: cb@opscode.com