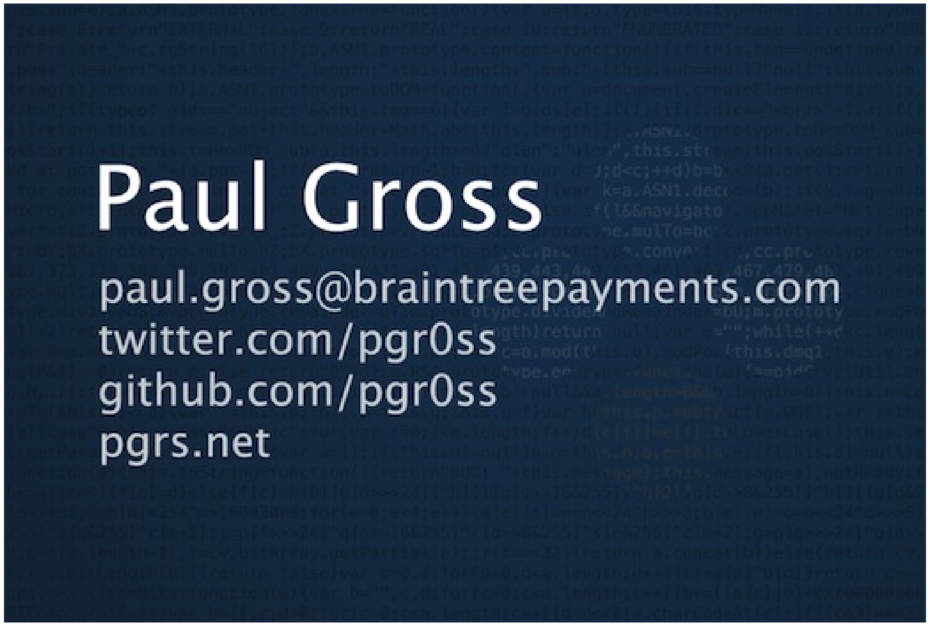
High Availability at Braintree



Paul Gross

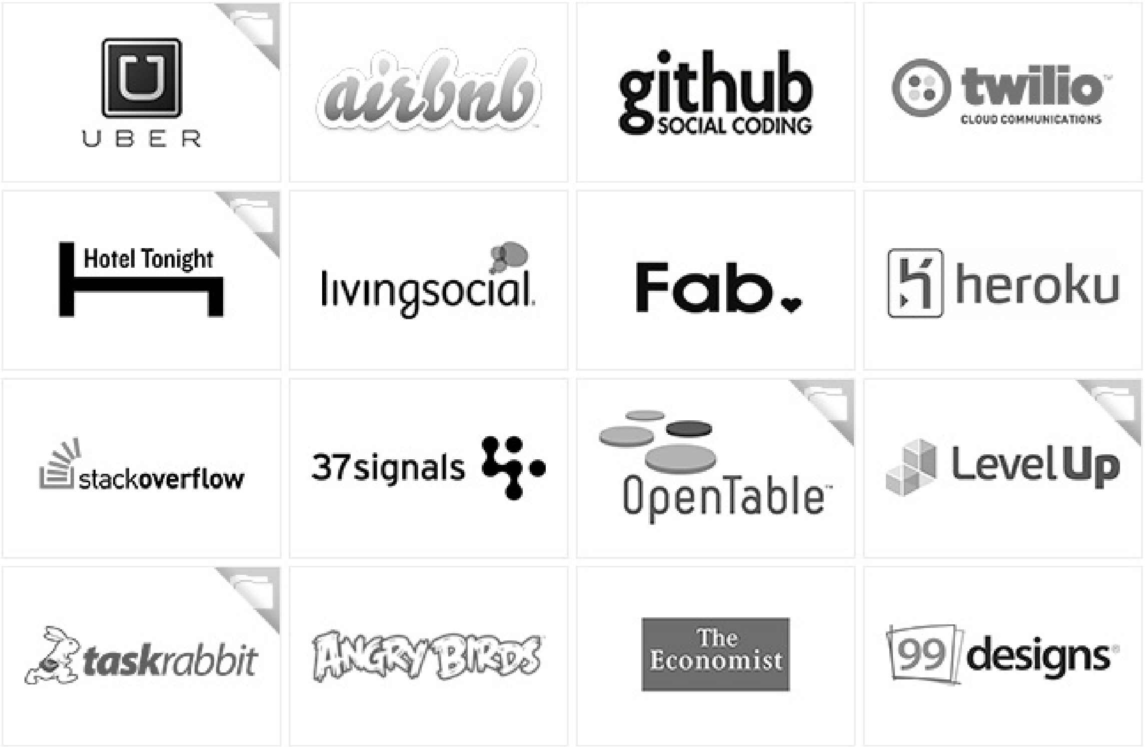
paul.gross@braintreepayments.com
twitter.com/pgr0ss
github.com/pgr0ss
pgrs.net

Braintree

Braintree is a payment gateway

A payment gateway is software that allows merchants to process credit card payments from your website and/or application

Our Merchants

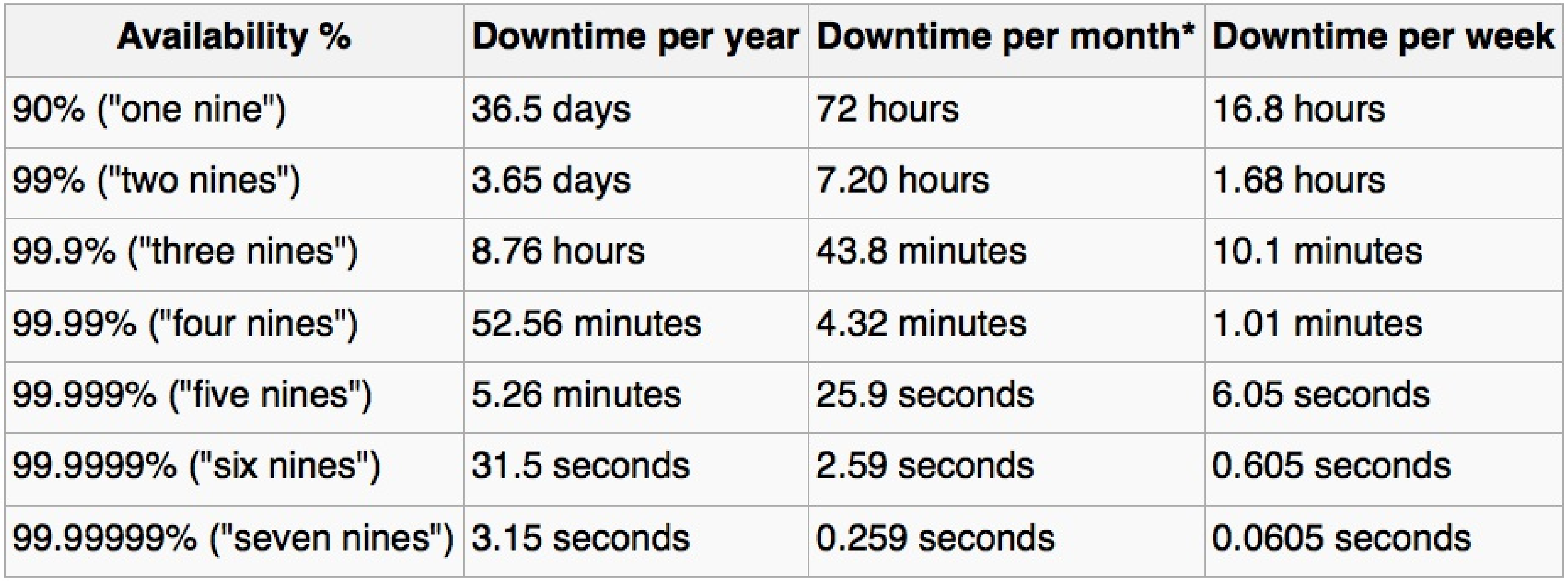


Why is uptime important?

5 billion dollars in annual processing

$9,500 per minute for our merchants

Uptime Percentages



Availability %	Downtime per year	Downtime per month*	Downtime per week
90% ("one nine")	36.5 days	72 hours	16.8 hours
99% ("two nines")	3.65 days	7.20 hours	1.68 hours
99.9% ("three nines")	8.76 hours	43.8 minutes	10.1 minutes
99.99% ("four nines")	52.56 minutes	4.32 minutes	1.01 minutes
99.999% ("five nines")	5.26 minutes	25.9 seconds	6.05 seconds
99.9999% ("six nines")	31.5 seconds	2.59 seconds	0.605 seconds
99.99999% ("seven nines")	3.15 seconds	0.259 seconds	0.0605 seconds

(https://en.wikipedia.org/wiki/High_availability)

2 kinds of downtime

Planned

Unplanned

Reduce our maintenance windows

Switched from MySQL to PostgreSQL

DDL migrations are extremely fast

Add indexes without locking tables

Transactional DDL

http://www.pgrs.net/2011/03/25/migrating-from-mysql-to-postgresql-slides/

Deploy process

Add new tables and columns

Roll out new code (server by server)

Add indexes

```ruby
namespace :db do
  task :migrate_pre => :environment do
    ActiveRecord::Migrator.migrate "db/migrate_pre"
  end

  task :migrate_post => :environment do
    ActiveRecord::Migrator.migrate "db/migrate_post"
  end
end
```

Rails caches columns

Can't drop columns in a post migration

Need to tell Rails to forget the column

```
class User < ActiveRecord::Base
  deleted_columns :old_column
end
```

```
ActiveRecord::Base.class_eval do
  def self.deleted_columns(*column_names)
    @deleted_columns = column_names.map(&:to_s)
  end
end
```

```ruby
ActiveRecord::Base.class_eval do
  def self.deleted_columns(*column_names)
    @deleted_columns = column_names.map(&:to_s)
  end

  def self.columns_with_removing_deleted
    columns_without_removing_deleted.reject do |c|
      @deleted_columns.include?(c.name)
    end
  end
  alias_method_chain :columns, :removing_deleted
end
```

We run multiple versions of the code at once

Fine for most features

Feature switches to turn on new features at once

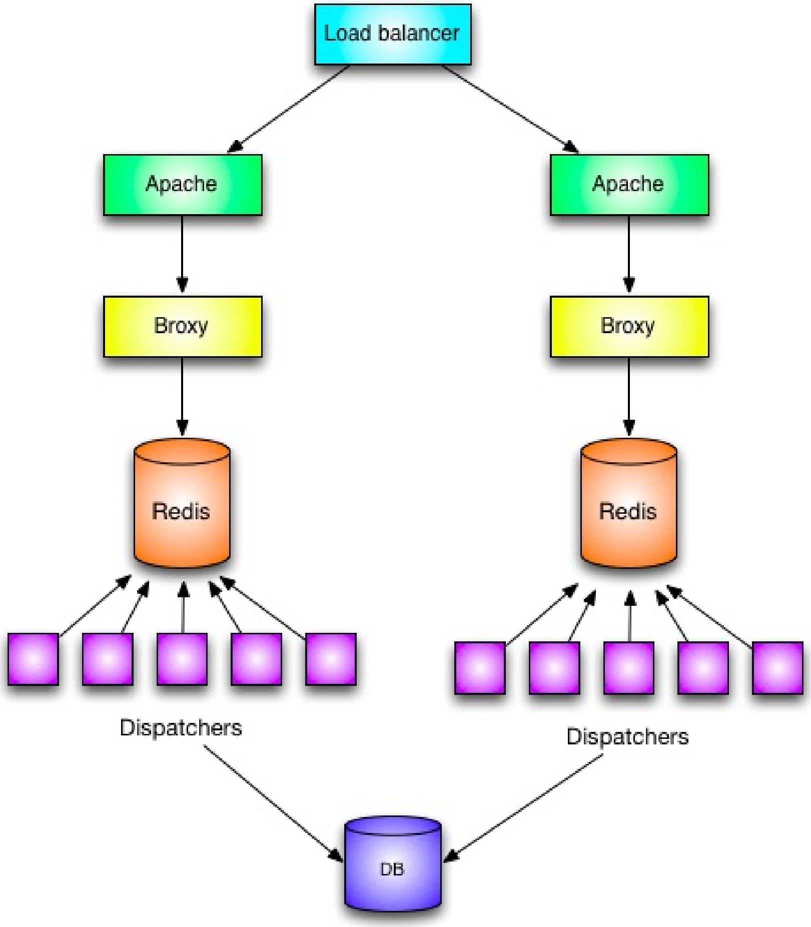
Limitations

Column renames

Database failover

Infrastructure changes

Want a way to pause traffic


Load balancer
Apache
Apache
Broxy
Broxy
Redis
Redis
Dispatchers
Dispatchers
DB

Broxy = Braintree Proxy

Python/Tornado (evented)

Accepts web requests

Feeds redis queue

Reads responses from redis

Dispatchers

Lightweight rack adapter

Takes requests from redis

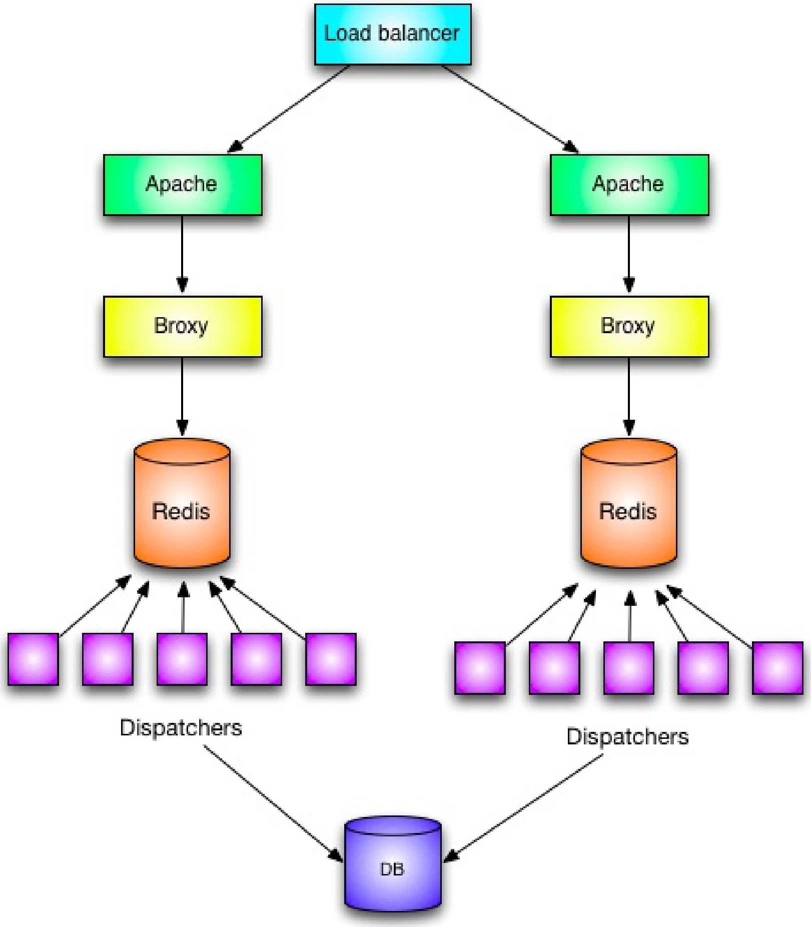
Processes through rails

Puts response back in redis

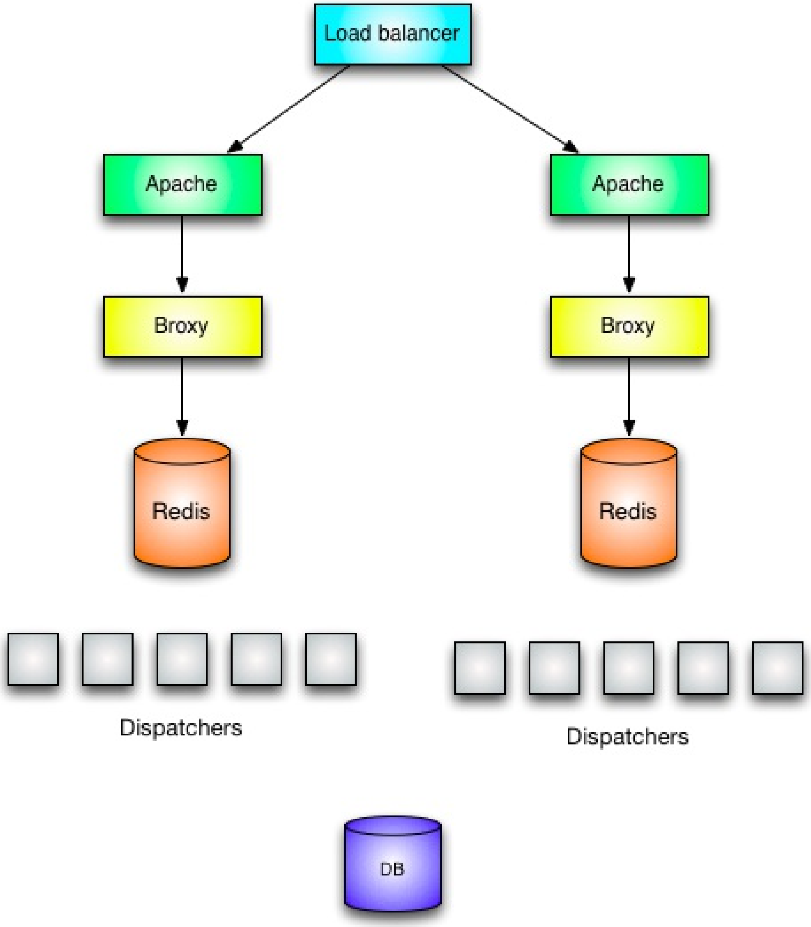
```ruby
require "#{ENV['RAILS_ROOT']}/config/environment"
app = Rack::Builder.new do
  run ActionController::Dispatcher.new
end
```

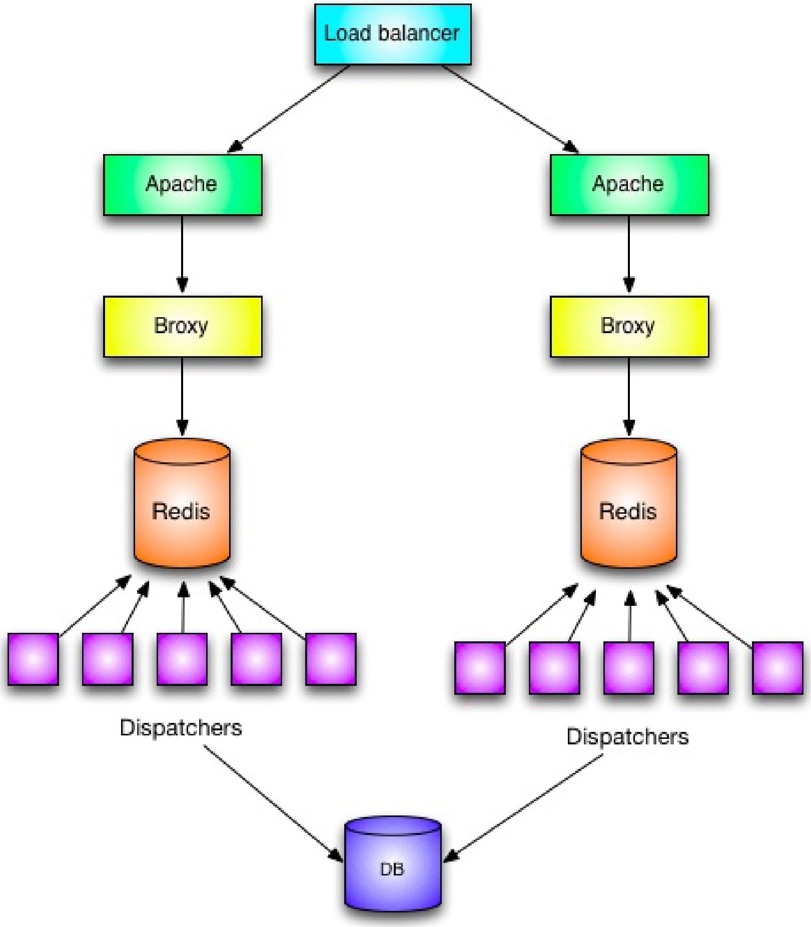
```ruby
require "#{ENV['RAILS_ROOT']}/config/environment"
app = Rack::Builder.new do
  run ActionController::Dispatcher.new
end

loop do
  if request_data = redis.pull_request
    rack_request = {
      "PATH_INFO" => request_data["request"]["uri"]
      "rack.input" => StringIO.new(request_data["body"])
    }

    rack_response = app.call(rack_request)
  end
end
```

```ruby
require "#{ENV['RAILS_ROOT']}/config/environment"
app = Rack::Builder.new do
  run ActionController::Dispatcher.new
end

loop do
  if request_data = redis.pull_request
    rack_request = {
      "PATH_INFO" => request_data["request"]["uri"]
      "rack.input" => StringIO.new(request_data["body"])
    }

    rack_response = app.call(rack_request)

    body = ""; rack_response[2].each { |part| body << part }

    @redis.push_response(
      "status" => rack_response[0].to_i,
      "body" => body
    )
  end
end
```


Load balancer
Apache
Apache
Broxy
Broxy
Redis
Redis
Dispatchers
Dispatchers
DB

Stop dispatchers to suspend traffic


Load balancer
Apache
Apache
Broxy
Broxy
Redis
Redis
Dispatchers
Dispatchers
DB


Load balancer
Apache
Apache
Broxy
Broxy
Redis
Redis
Dispatchers
Dispatchers
DB

Summary - reducing maintenance windows

Pre and post migrations

Rolling deploys

PostgreSQL for fast DDL

Broxy to pause traffic

Unplanned failures

Servers will fail

Networks will go down

The unexpected will happen

We do our best to be resilient

Server failure

Load balancers

Build our own

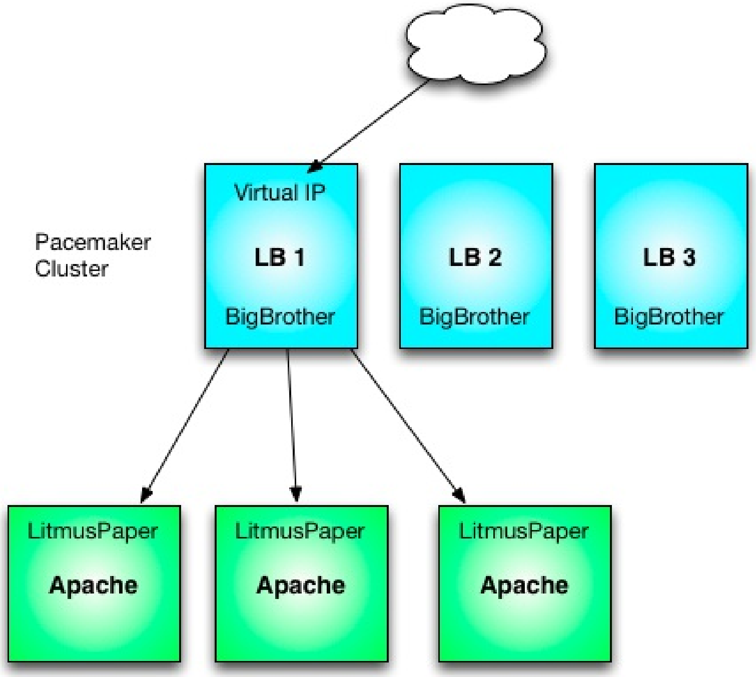
LVS/IPVS

Pacemaker

BigBrother

LitmusPaper

Load balancing



BigBrother

Ruby app

Runs on load balancers

Checks status of servers

Update IPVS rules

https://github.com/braintree/big_brother

LitmusPaper

Ruby app

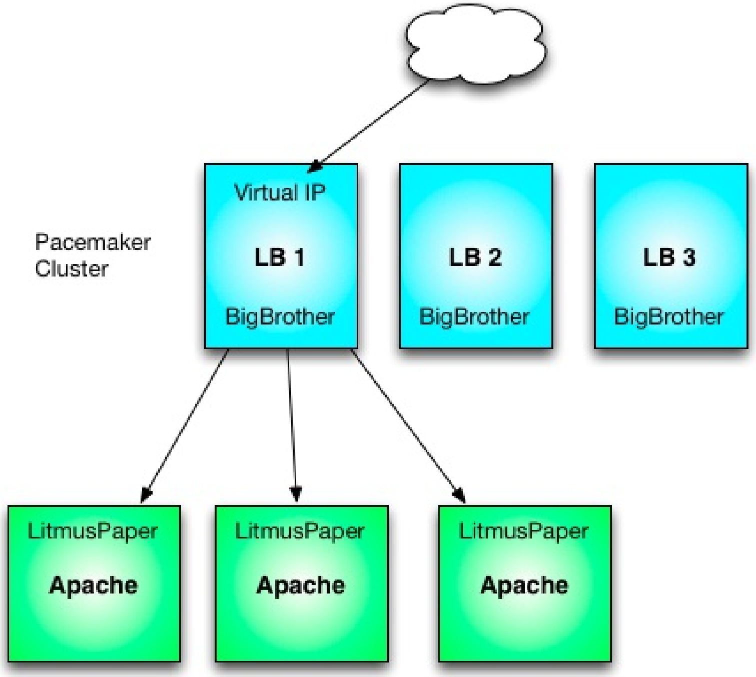
Runs on backend servers

Queried by BigBrother via HTTP

Returns a health level

https://github.com/braintree/litmus_paper

Load balancing



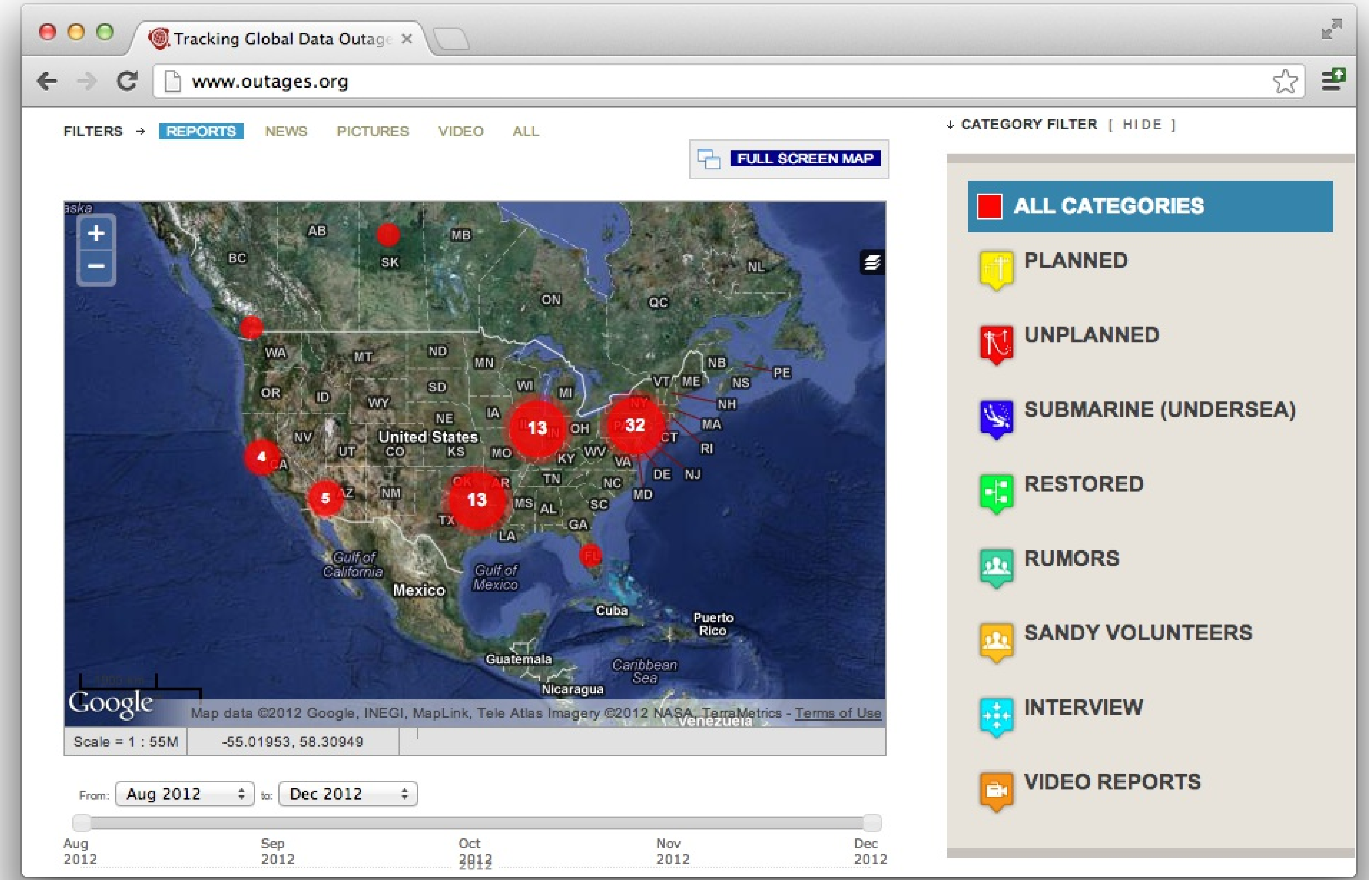
Stateful services

Load balancers and PostgreSQL clusters

Pacemaker manages failover

Virtual IP follows the new primary

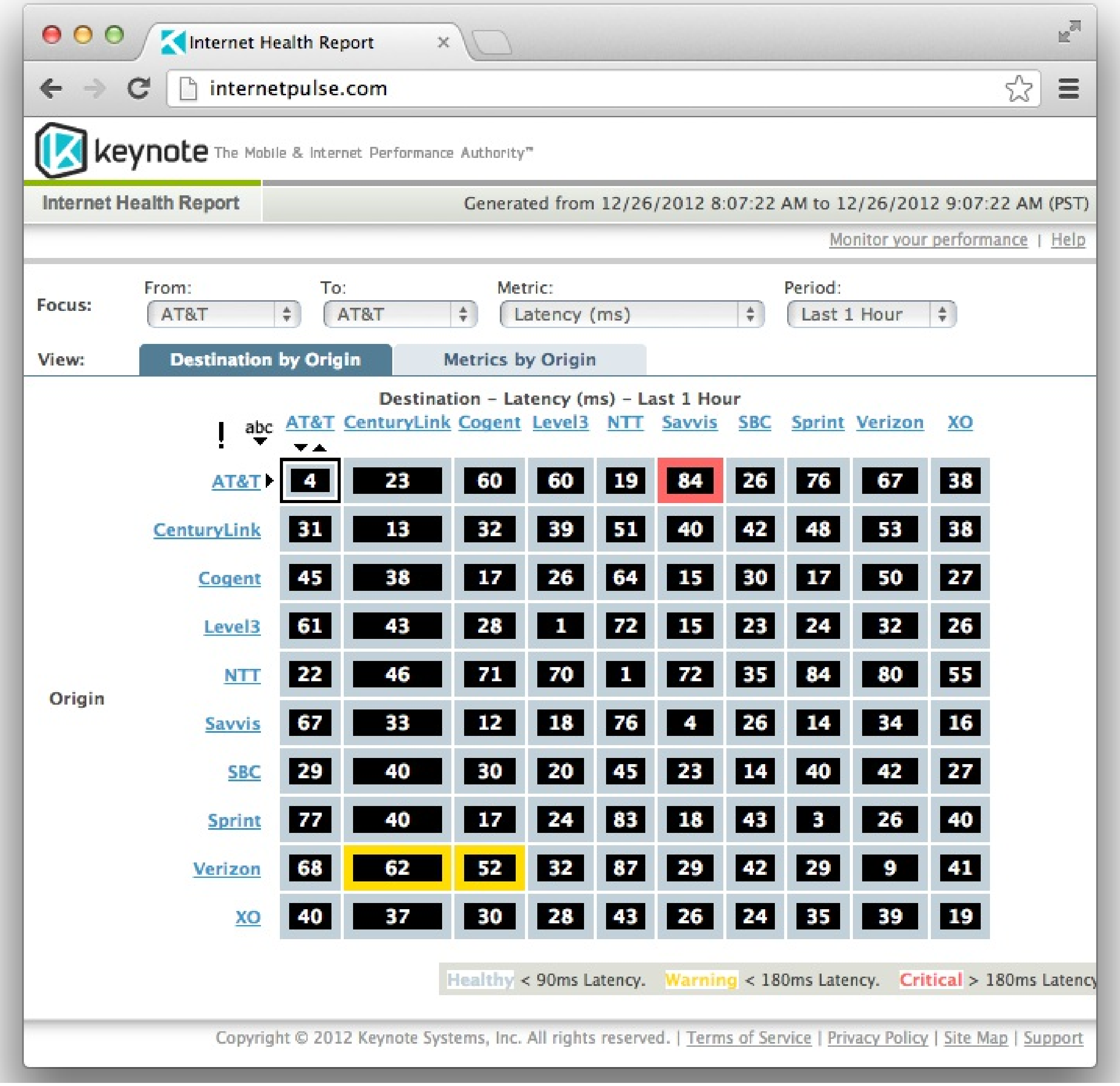
Network failures



Map data ©2012 Google, INEGI, MapLink, Tele Atlas Imagery ©2012 NASA, TerraMetrics - Terms of Use

Scale = 1 : 55M | -55.01953, 58.30949

From: Aug 2012 to: Dec 2012


Internet Health Report
internetpulse.com
keynote The Mobile & Internet Performance Authority™

Internet Health Report

Generated from 12/26/2012 8:07:22 AM to 12/26/2012 9:07:22 AM (PST)

Monitor your performance | Help

Focus: From: AT&T To: AT&T Metric: Latency (ms) Period: Last 1 Hour

View: Destination by Origin | Metrics by Origin

Destination – Latency (ms) – Last 1 Hour

Origin	AT&T	CenturyLink	Cogent	Level3	NTT	Savvis	SBC	Sprint	Verizon	XO
AT&T	4	23	60	60	19	84	26	76	67	38
CenturyLink	31	13	32	39	51	40	42	48	53	38
Cogent	45	38	17	26	64	15	30	17	50	27
Level3	61	43	28	1	72	15	23	24	32	26
NTT	22	46	71	70	1	72	35	84	80	55
Savvis	67	33	12	18	76	4	26	14	34	16
SBC	29	40	30	20	45	23	14	40	42	27
Sprint	77	40	17	24	83	18	43	3	26	40
Verizon	68	62	52	32	87	29	42	29	9	41
XO	40	37	30	28	43	26	24	35	39	19

Healthy < 90ms Latency. Warning < 180ms Latency. Critical > 180ms Latency

Networking - inbound

BGP routes traffic through multiple ISPs and data centers

We use Pingdom and a handful of globally distributed servers to test connectivity

Networking - outbound

We connect to many processing networks

ISP outages are usually partial

Sometimes, we can't reach every endpoint on all of our ISPs

Needed a way to choose an ISP per processing network

Processor proxies

Instead of connecting directly, connect through proxies

One proxy per TCP endpoint and uplink ISP

Load balance over these proxies

Allows us to route around ISP connection issues

Mallory

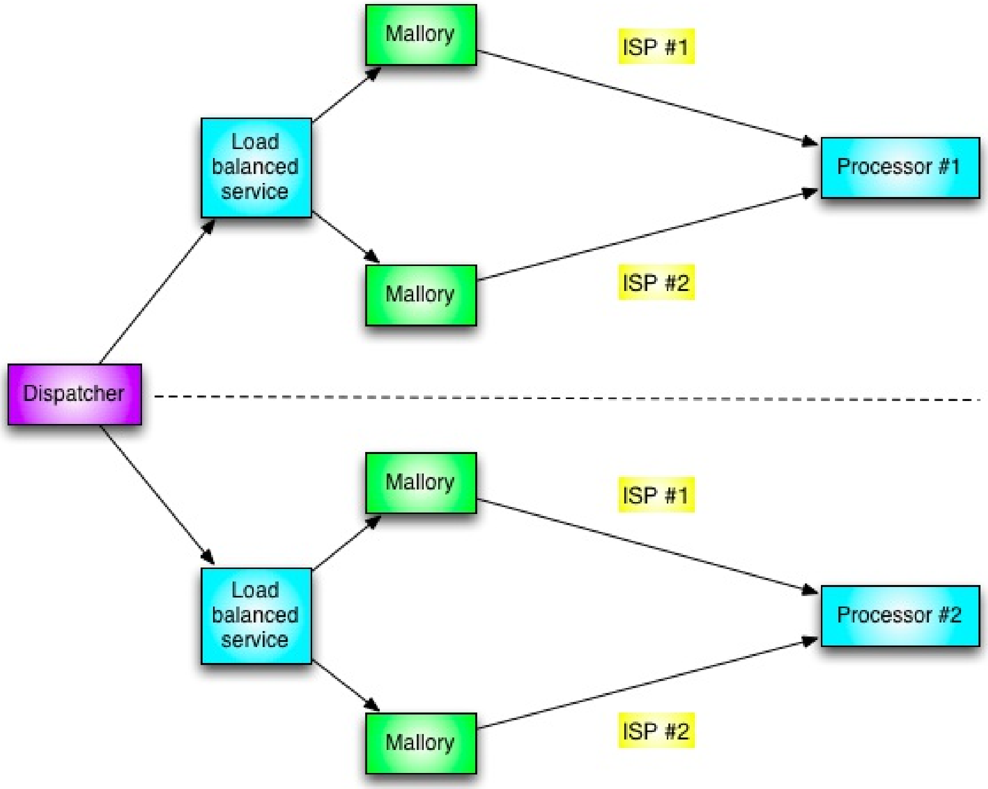
Python/Tornado (evented)

Proxies requests

SSL verification

Acts like LitmusPaper

https://github.com/braintree/mallory


Mallory
ISP #1
Load balanced service
Processor #1
Mallory
ISP #2
Dispatcher
Mallory
ISP #1
Load balanced service
Processor #2
Mallory
ISP #2

Connection failures

Let the service heal (unbalance or pacemaker)

Retry request

Automate everything

Reduces human errors

Gives confidence that task will work

Speeds up processes

Less fiddling around in production

Capistrano

Summary - unplanned failures

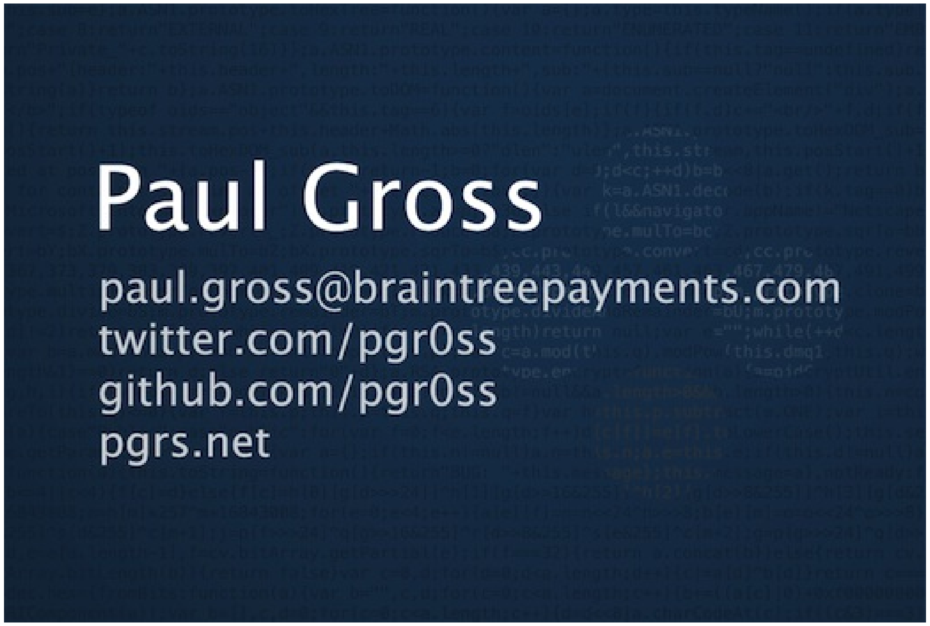
Load balancing

Redundancy across ISPs

Let the system heal and retry

Automation

Questions?



Paul Gross

paul.gross@braintreepayments.com
twitter.com/pgr0ss
github.com/pgr0ss
pgrs.net