Building Scalable Servers
With EventMachine and Rails
EventMachine

DRY
DRAG

(Don’t Repeat Aman Gupta)
require 'aman_gupta'
EventMachine
scalable non-blocking i/o in ruby
About Me

- Aman Gupta
- San Francisco, CA
- Been maintaining EventMachine for 18 months (last 4 releases)
- Ruby Hero 2009
- amqp, REE patches, perftools.rb, gdb.rb, memprof
- github.com/tmm1
- @tmm1
What is EventMachine?

- Implementation of the Reactor pattern
  - similar to Python’s Twisted also node.js, libev
- Ruby VM Support
  - Ruby 1.8 (MRI)
  - Ruby 1.9 (YARV)
  - Rubinius
  - JRuby
- + simple Pure Ruby version

java reactor

c++ reactor
What is I/O?

- Generally Network I/O
  - mysql query responses
  - http responses
  - memcache results

- Most web applications are I/O bound, not CPU bound

- Basic idea behind EM: Instead of waiting on a response from the network, use that time to process other requests
What can I use EM for?

- Scaling I/O heavy applications
  - handle 5-10k concurrent connections with a single ruby process
- Any type of network I/O
  - http requests
  - sending emails
  - custom tcp proxies
  - data access
    - redis
    - couchdb
    - mysql
    - postgres
    - cassandra
    - memcached
Simple TCP Server

- TCPServer#accept to accept connections from new clients
- TCPSocket#read* blocks, so you can only handle one client at a time
- Common Solution: use a thread per client

```ruby
require 'socket'
server = TCPServer.new(2202)
while client = server.accept
  msg = client.readline
  client.write "You said: #{msg}"
  client.close
end
```

```ruby
require 'socket'
server = TCPServer.new(2202)
while true
  Thread.new(server.accept){ |client|
    msg = client.readline
    client.write "You said: #{msg}"
    client.close
  }
end
```
require 'socket'
s
server = TCPServer.new('2202')
clients = []
buffers = {}

while true
  sockets = [server] + clients
  readable, writable = IO.select(sockets)

  readable.each do |sock|
    begin
      if sock == server
        clients << server.accept_nonblock
      else
        client, buf = sock, buffers[sock] || ''
        buf << client.read_nonblock(1024)
      end
      if buf =~ /\r?\n/ 
        client.write "You said: #{buf}"
        client.close
      end
    end
  end
  rescue Errno::EAGAIN, Errno::EWOULDBLOCK
    # socket would block, try again later
  end
end
EM does Non-Blocking I/O

- handles low level sockets for you
- inbound/outbound buffers for maximal throughput
- efficient i/o with writev/readv
- epoll & kqueue support

```ruby
module EchoServer
  def post_init
    @buf = ''
  end
  def receive_data(data)
    @buf << data
    if @buf =~ /^.*??
      send_data "You said: #{@buf}"
      close_connection_after_writing
    end
  end
end

require 'eventmachine'
EM.run do
  EM.start_server '0.0.0.0', 2202, EchoServer
end
```
So, what’s a Reactor?

- reactor is simply a **single threaded while loop**, called the “reactor loop”
- your code “reacts” to incoming events
- if your event handler takes too long, other events cannot fire
- lesson: never block the reactor
  - no sleep(1)
  - no long loops (100_000.times)
  - no blocking I/O (mysql queries)
  - no polling (while !condition)
Writing Asynchronous Code

- synchronous ruby code uses return values
  
  ```ruby
  ret = operation()
  do_something_with(ret)
  ```

- evented async code uses blocks instead
  
  ```ruby
  operation{ |ret| do_something_with(ret) }
  ```

- different from how you usually use ruby blocks. the block is stored and invoked later (it’s asynchronous)
  
  ```ruby
  puts(1)
  1.times{ puts(2) }
  puts(3)
  operation{ puts(3) }
  puts(2)
  ```
Receiving Email Using ActionMailer

- port 25 (smtp)
- postfix (spool)
- fork process
- ActionMailer
- ActiveRecord
- postgresql (db)
Receiving Email — Scalably via LMTP

postfix (spool)

postfix (spool)

message systems

EM (w/Rails)  EM (w/Rails)  EM (w/Rails)  EM (w/Rails)
LMTP Daemon

POP3 Daemon

IMAP Daemon

Geo Firehose
(one connection – Port 80)

twittervision.com/stream

RewriteRule ^/stream(.*)$ http://127.0.0.1:8192/%{REQUEST_URI} [P,QSA,L]
replyz.com
Does Anybody Know...?

mailstrom.410labs.com
Power Tools for Your Inbox

shortmail.com
Email, simplified.
Some Other Handy Stuff...

- **start_tls**: secure any TCP connection
- **Timers**: Periodic, Once, Scheduled
- **epoll, kqueue, threadpool_size**
- **set_effective_user** (listen on protected ports)
So, Some Stuff To Remember...

- EventMachine: awesome for I/O, events
- Code is Data, Data is Code
- Often Working with State Machines
- Don’t peg the CPU. No really, don’t.
- TEBWTDI
- Testing can be interesting
Tomorrow @ BohConf!
http://emrubyconf.com/