Rubyを使った分散全文検索ミドルウェア

須藤功平

株式会社クリアコード

RubyWorld Conference 2014

2014/11/13

趣意書

Rubyを
普通の人々に
浸透させたい！

この発表の内容

浸透促進案の提案

浸透対象の普通の人々

- よくプログラムを書く人
  - ただしRubyとは縁遠い
  - →よくRubyを書くようになる
- たまにプログラムを書く人
  - インフラの人とか
  - →たまにRubyを書くようになる

浸透方法

- 技術的な攻め方
  - 多機能・高機能・高性能
  - 他よりいいですよ
- 心理的・政治的な攻め方
  - 流行り
  - 「みんな」使っていますよ

提案方針

- ✓ 技術的な攻め方
  - ✓ 多機能・高機能・高性能
  - ✓ 他よりいいですよ
- ✓ 心理的・政治的な攻め方
  - ✓ **流行り**
  - ✓ 「みんな」使っていますよ

流行っている状態

シェアを独占している

- キラーアプリがある
  - 例：Ruby on Rails
- 有用アプリの多くがRuby製
  - 例：Chef, Puppet, Vagrant, Serverspec

キラーアプリは難しい

シェアを独占している

- ~~キラーアプリがある~~
  - 例：Ruby on Rails
- 有用アプリの多くがRuby製
  - 例：Chef, Puppet, Vagrant, Serverspec

多有用アプリ状況を目指す

- Ruby製アプリ使う
  - Ruby製アプリを優先して選ぶ
  - 使っていることを広くアピール
- Ruby製アプリを作る
  - そこそこ有用で十分
  - たくさん作る

Ruby製アプリ作りを促す

- 成功事例（きっかけ）
  - 後続が真似したくなればよい
  - そんなに流行らなくてもよい
- 開発ノウハウ（助け）
  - 後続が開発しやすくなる
  - ライブラリー化されていると尚よし

どんなアプリがよいか

ミドルウェア

↓

自作のアプリは
好きな言語で書ける

（言語による採用障壁が低め）

（例：RabbitMQはErlang製だがアプリはErlang以外が多い）

ミドルウェア

- データストア（RDBMSやKVS）
- 検索システム
- メールシステム
- メッセージキュー
- ログ活用（分析や監視）

Rubyでミドルウェア

- 成功事例
  - Fluentd, ROMA
  - milter manager
  - 分散全文検索エンジンを開発中
    (Droonga)
- 開発ノウハウ
  - →これから紹介

開発ノウハウ：方針

- トレードオフと向き合う
  - どこを強みにするか
  - どこは競合と戦わないか
- 全方位で勝つことはできない
  - Rubyだって速さじゃCに勝てない
  - でも楽しさならCに勝てる

強みの選び方

- ✓ 使いやすさで勝負する
- ✓ 最高速で勝負しない
- ✓ 多機能で勝負しない

ミドルウェアの使いやすさ

- 導入・設定・運用の簡単さ
  - 多くのミドルウェアは大変←ヒント
  - 例：設定なしで動くと簡単
- 止めないことが前提
  - 無停止で設定再読み込み
  - 無停止でアップグレード

強みの選び方 - 使いやすさ

- 使いやすさで勝負する
  - ユーザーの手間を減らす
  - かゆいところに手が届く
- 最高速で勝負しない
- 多機能で勝負しない

性能

- 最高速は目指さない
  - C/C++とかJavaに負ける
- 十分な速度は目指す
  - ミドルウェアが
    ボトルネックにならない程度

ミドルウェアと性能

- ミドルウェア=サーバー
  - 並行処理をいかにがんばるか
- 評価基準
  - レイテンシー（1リクエストに注目）
  - スループット（単位時間に注目）

ボトルネックの解消方法

- なりやすい箇所
  - CPU
  - ネットワーク
- なるかもしれない箇所
  - I/Oとメモリー

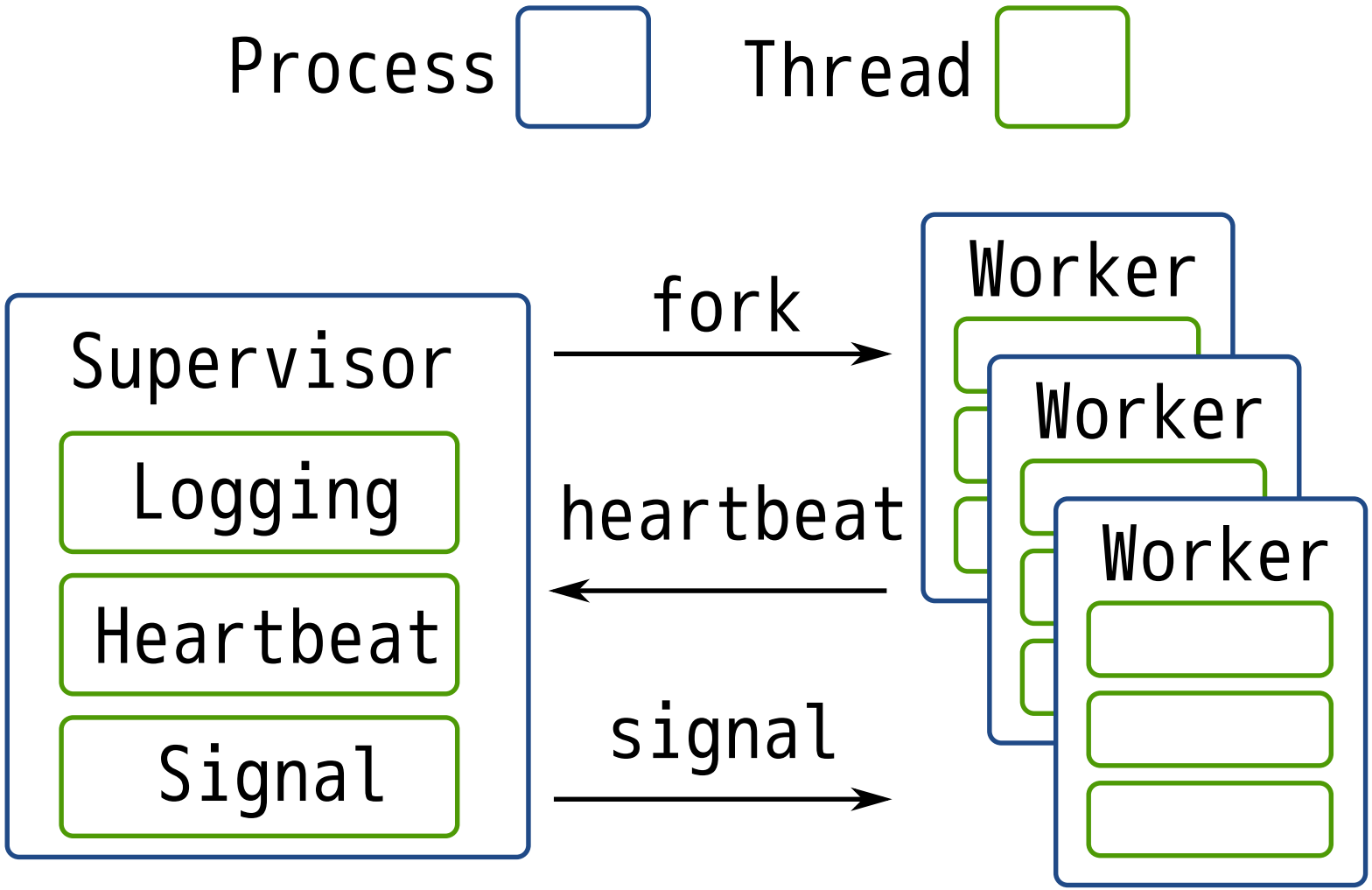
CPUネック

- 処理を減らす
- マルチプロセス
  - 後述
- Cで拡張ライブラリーを書く
  - Fluentd: MessagePack
  - Droonga: MessagePack, Groonga

マルチプロセス実装

- ServerEngine（ライブラリー）
  - nスレッド+シグナル+fork(spawn)
- Droonga
  - 1スレッド+パイプ+spawn+イベントループ
  - ↑の方がオススメ

ServerEngine



ServerEngineモデル1

- ✓ 基本はSupervisor→Worker
- ✓ gracefulな再起動をしにくい
  （無停止で設定再読み込みできれば必要ない）
  - ✓ 新Workerの準備完了を知れない
- ✓ 無停止アップグレード×

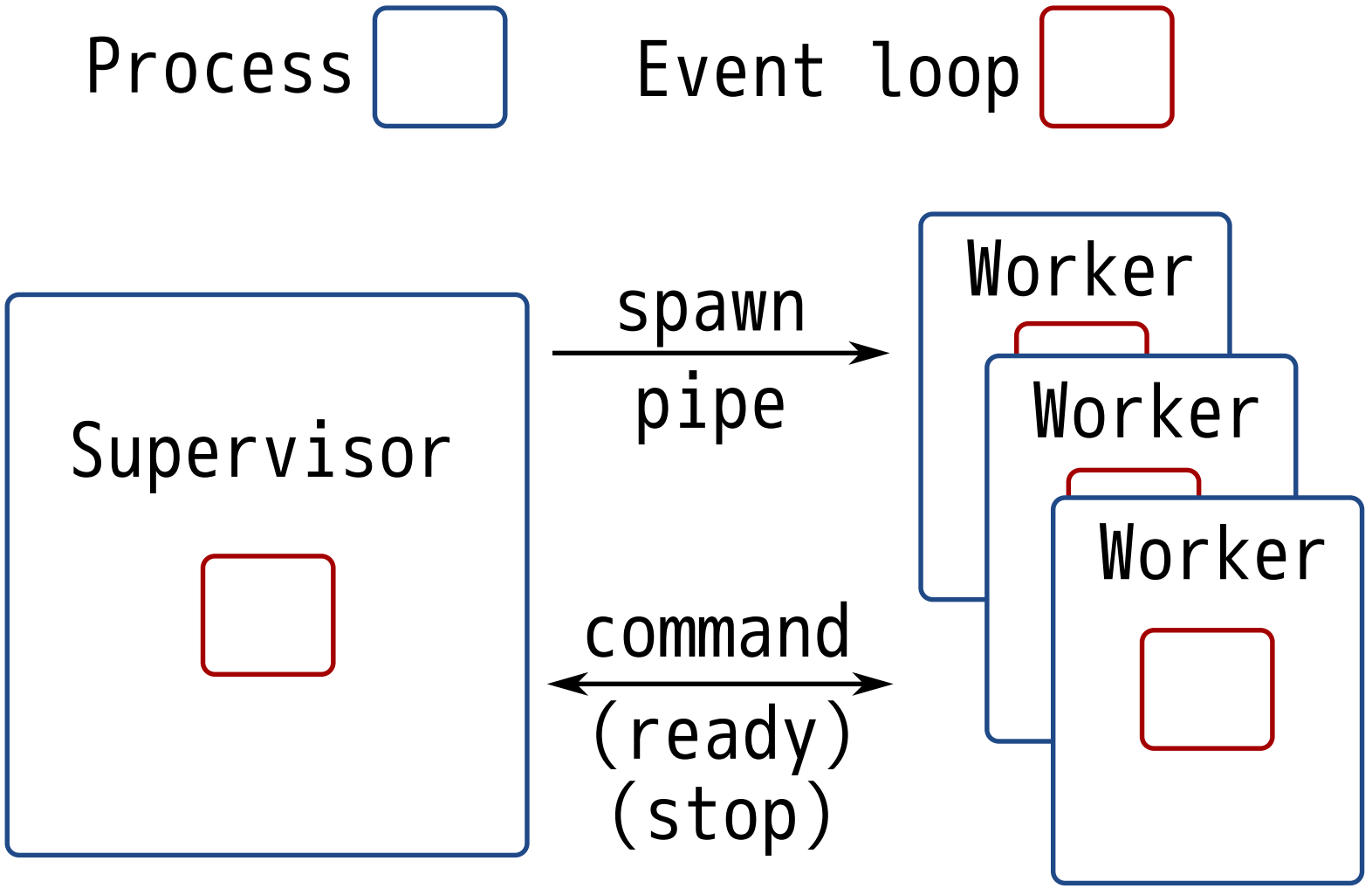
ServerEngineモデル2

- スレッドは難しい
  - エラーをちゃんと処理しないと問題を見逃す
- シグナルは難しい
  - 終了中に何度でもSIGINT
- forkは難しい

ServerEngineモデル3

- ✓ ライブラリー化していてえらい

Droonga


Process
Event loop
Supervisor
spawn
pipe
command
(ready)
(stop)
Worker
Worker
Worker

Droongaモデル1

- 通信はSupervisor↔Worker
- gracefulな再起動をしやすい
  - 新Workerが準備完了
    →Supervisorに通知
    →旧Workerをgracefulに終了
- 無停止更新ができる

Droongaモデル2

- シンプルになる
  - 1スレッド・シグナルなし
  - stop中にn回stopがこない
  - 終了処理中に割り込まれない
- イベント駆動は複雑

ネットワークネック

- 通信量を減らす
  - **データを減らす**（ムリなら圧縮）
    （LZ4で十分ならLZ4、ムリならzlib）
- ノンブロッキングI/Oと多重化
  - 拡張ライブラリー必須
  - →Cool.io, nio4r, EventMachine
  - イベント駆動なコードは複雑

イベント駆動なコード

```
Coolio::TCPServer.new(HOST, PORT) do |client|
  n_reads = 0
  client.on_read do |data|
    p data
    client.write(data)
    n_reads += 1
    if n_reads == 2
      client.on_write_complete {client.close}
    end
  end
end
```

同期っぽく書けるAPI

```
Coolio::TCPServer.new(HOST, PORT) do |client|
  Fiber.run do # <- 並行にしたい処理を明示
    client.extend(Synchronizable) # <- 42行
    2.times do
      data = client.read
      p data
      client.write(data)
    end
    client.close
  end
end
```

同期っぽく書けるAPI

- ユーザーがFiberを書くのがカッコ悪い
- 同期っぽい中で並行に処理したくなったら？
  - pubsubっぽいことをしたいとか

Promiseな世界

```
server.accept.then do |client|
  client.read.then do |data|
    p data
    client.write(data)
  end.then do
    client.close
  end
end.catch do |error|
end
```

Promiseな世界

- ✓ 繰り返しを書きにくい
- ✓ メソッドチェーンがカッコ悪い
- ✓ catchがカッコ悪い

API案：基本

```
# 同期っぽいAPI
clinet = server.accept
# 非同期API
server.accept do |request|
  begin
    client = request.socket
  rescue
  end
end
```

API案：組み合わせ

```
server.accept do |request|
  client = request.socket
  2.times do
    data = client.read
    p data
    client.write(data)
  end
  client.close
end
```

API案

- ✓ Fiberが見えない
- ✓ 書き方の組み合わせが自然
  - ✓ ブロックなし→同期っぽいAPI
  - ✓ ブロックあり→非同期API
- ✓ 実装していない😜

I/Oとメモリーネック

- データストアをCで書く
  - Droonga: Groonga
  - ROMA: Tokyo Cabinet, SQLite3
- コアの機能もCで書く
  - データコピーも減らしたいとき

強みの選び方 - 性能

- ✓ 使いやすさで勝負する
- ✓ 最高速で勝負しない
  - ✓ でも、十分な速度は目指す
  - ✓ ボトルネックにならなければよい
- ✓ 多機能で勝負しない

機能

- 多機能をウリにしない
  - 多機能だと遅くなる（ことが多い）
- 組み込みの機能より拡張性
  - →プラグイン機能
  - Fluentd, ROMA, Droonga

プラグイン機能のポイント

- 開発者向け
  - 作りやすい
  - テストしやすい
  - リリースしやすい
- ユーザー向け
  - インストールしやすい
  - 設定しやすい

開発者向け

- 作りやすさ
  - scaffoldいらずのAPI
- テストしやすさ
  - ドライバー・スタブを提供
- リリースしやすさ
  - gem

ユーザー向け

- インストールのしやすさ
  - Rubyをそんなに知らない前提なのに直接gemを使ってもらうのってアリ？
- 設定のしやすさ
  - できるだけ少なく
  - できればno configuration

強みの選び方 - 機能

- ✓ 使いやすさで勝負する
- ✓ 最高速で勝負しない
- ✓ 多機能で勝負しない
  - ✓ 組込機能よりも簡単拡張で勝負
  - ✓ Ruby初心者でも開発できる簡単さ
  - ✓ →プラグイン開発でRubyデビュー
    （tDiaryスタイルのRuby浸透方法）

まとめ1

- ✓ 趣意書
  - ✓ Rubyを浸透させたい！
- ✓ この発表
  - ✓ 浸透促進案の提案
  - ✓ ミドルウェア分野での促進案

まとめ2：促進案

- たくさんの人が使う
  - 他の方がよさそうでも Ruby製を優先して使う
  - 使っていることを広める
- たくさん作る

まとめ3：作るノウハウ

- ✓ 使いやすさで勝負する
  - ✓ ユーザーの手間を減らす
- ✓ 最高速で勝負しない
  - ✓ ボトルネックにならなければよい
- ✓ 多機能で勝負しない
  - ✓ 組込機能より拡張性

おまけ

Droongaの紹介

Droongaとは

- ✓ Ruby製
- ✓ 分散全文検索エンジン
- ✓ SPOFなしの構成

Droongaの特徴

- ✓ 処理をパイプラインとしてつなげられる予定
- ✓ 処理はプラグイン可能
  - ✓ Rubyで簡単に（予定）書ける
- ✓ Groonga互換API提供
  - ✓ Groonga = 既存全文検索エンジン

Droongaの実装：性能

- レイテンシー
  - Groongaより高いけど
    気になるほどではない
- スループット
  - ノード数を増やせばGroongaより速い

Droongaの実装：機能

- ✓ プラグインで拡張可能

Droongaの実装：使いやすさ

- ✓ インストール
  - ✓ インストーラー提供で簡易化
- ✓ これからがんばる
  - ✓ 設定・更新・運用
  - ✓ プラグインの作りやすさ

Droonga

http://droonga.org/

11/29（いい肉の日）
Groongaイベント開催

（東京）
http://groonga.doorkeeper.jp/events/15816