Inside Tabelog's Backend

2008年6月22日

株式会社カカクコム 京和 崇行

自己紹介

- 名前 => 京和崇行(きょうわたかゆき)
- 所属 => (株)カカクコム 事業開発本部
  - 2007年2月入社
  - 食べログ一筋
  - Rails化を提案した張本人
- 担当 => 開発とインフラを兼任
  - コードバリバリ書いてます
  - Apache・mongrel・MySQLなどのミドルウェアの運用、チューニングなど

アジェンダ

1. 食べログについて
2. mongrelについて
3. スケールアウトについて
4. パフォーマンスについて
5. まとめ

1.食べログについて

- 食べログとは
- スペック
- Railsサイトとしての規模
- ソフトウェア
- システム構成

1-1.食べログとは


レストラン総合ランキング
もっと見る→
1
フォリオリーナ・デッラ・ポルタ・フォルトゥーナ (31)
(東京都 / イタリアン)
4.54
2
アロマフレスカ (56)
(東京都 / イタリアン)
4.50
3
よし住 (39)
(埼玉県 / 寿司)
4.44
3
ル・バエレンタル (30)
(北海道 / フレンチ)
4.44
5
すし蓑 (13)
(宮城県 / 寿司)
4.43
6
千松しま (7)
(宮城県 / 割烹・小料理)
4.42
7
ガストロノミー ジョエル・ロブション (88)
(東京都 / フレンチ)
4.40
8
暦匠 壱 (23)
(東京都 / 鳥料理)
4.34
9
横浜 うかい亭 (88)
(神奈川県 / 鉄板焼き)
4.33
10
ビストロ コムシコムサ (34)
(京都府 / フレンチ)
4.32
11
美かさ (33)
(神奈川県 / 天ぷら)
4.28
11
京味 (21)
(東京都 / 京料理)
4.28
13
コート・ドール (61)
(東京都 / フレンチ)
4.27
14
乙女寿司 (27)
(石川県 / 寿司)
4.26
14
鬼怒川 竹やぶ (13)
(茨城県 / そば)
4.26

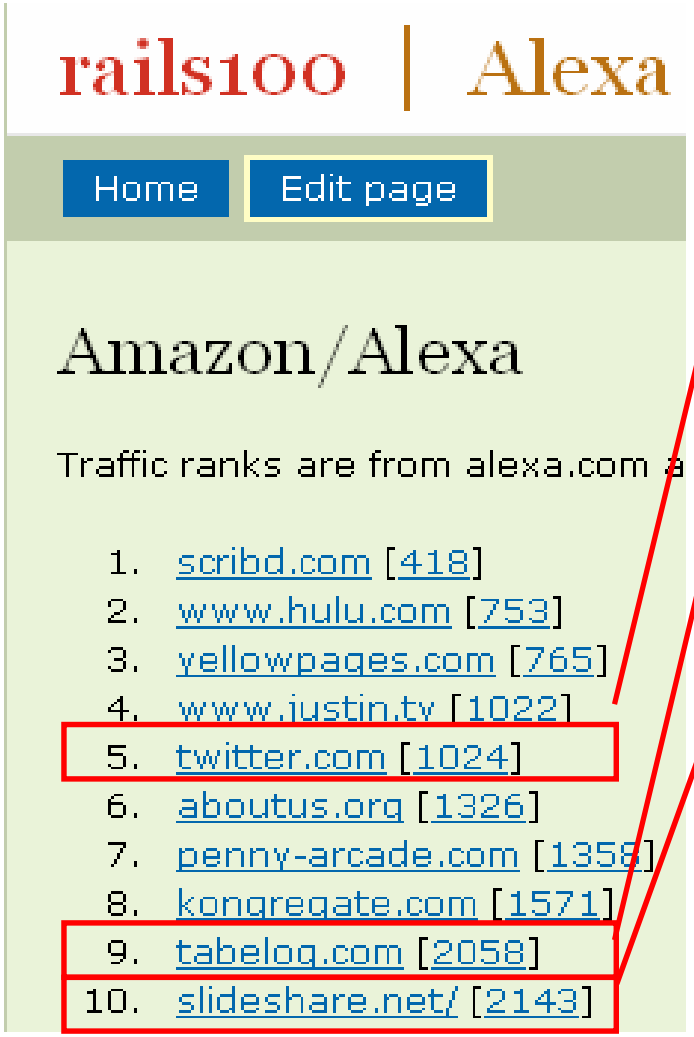
1-1.食べログとは

- CGM型グルメサイト
  - ユーザが口コミを投稿 → ランキング化
- 2007年10月19日 Railsにリニューアル
  - それまではWindows + ASP

1-2.スペック

- PV(月間) => 7000万PV
- UU(月間) => 600万UU
- サーバ => 13台
- コード => 20,000行
- テーブル => 150個

1-3.Railsサイトとしての規模



5位:twitter

9位:食べログ

10位:slideshare

出典:

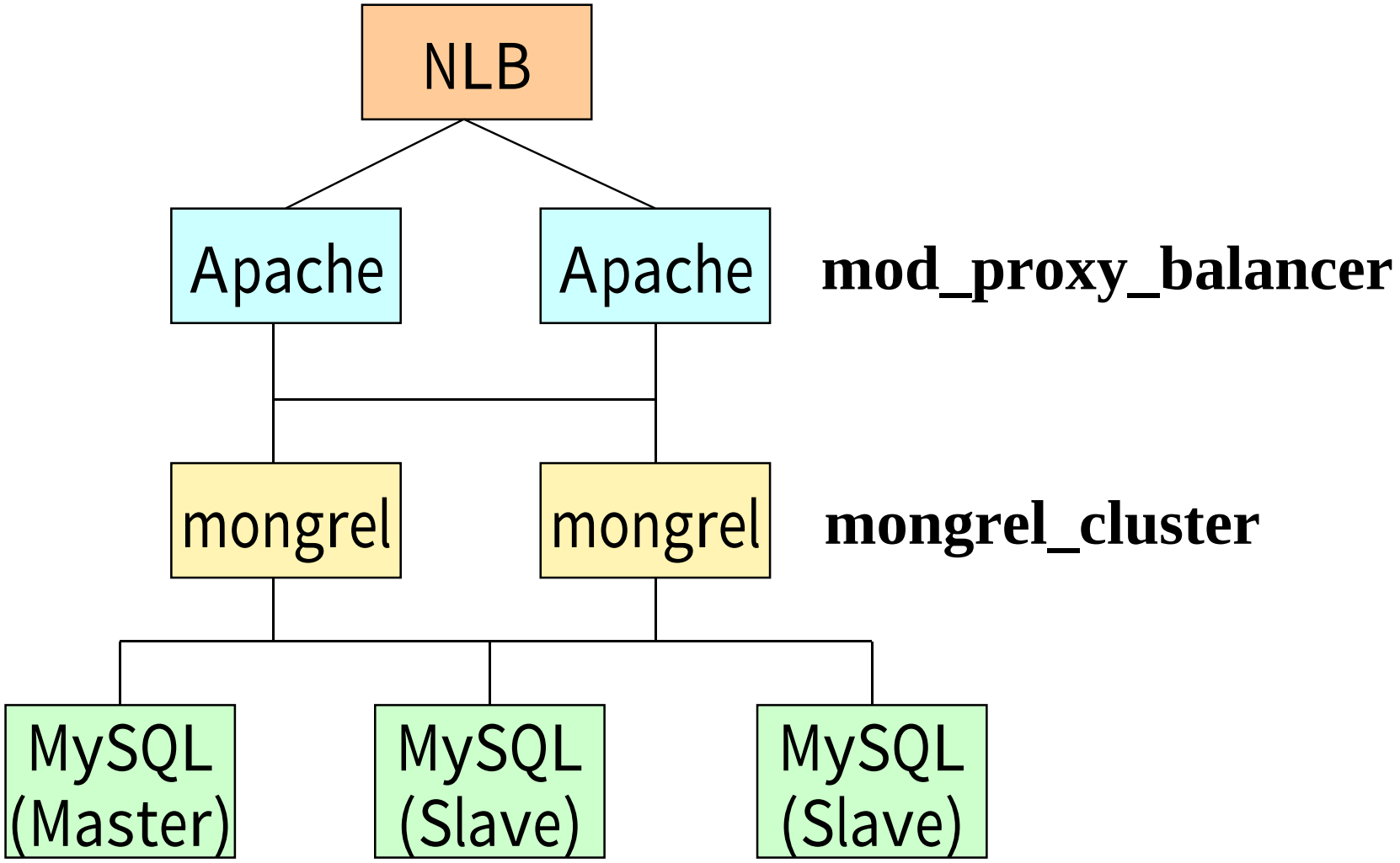
Rails100 wiki(2008/6/7)

http://rails100.pbwiki.com

1-4. ソフトウェア

- Red Had Enterprise Linux 4(RHEL)
- Ruby 1.8.5
- Rails 1.2
- Apache 2.2
- MySQL 5.0
- mongrel 1.0
- capistrano,memcached, etc..

1-5.システム構成


NLB
Apache
Apache
mod_proxy_balancer
mongrel
mongrel
mongrel_cluster
MySQL
(Master)
MySQL
(Slave)
MySQL
(Slave)

2. mongrelについて

- 選択理由
- メモリの使用量
- 安定性
- パフォーマンス

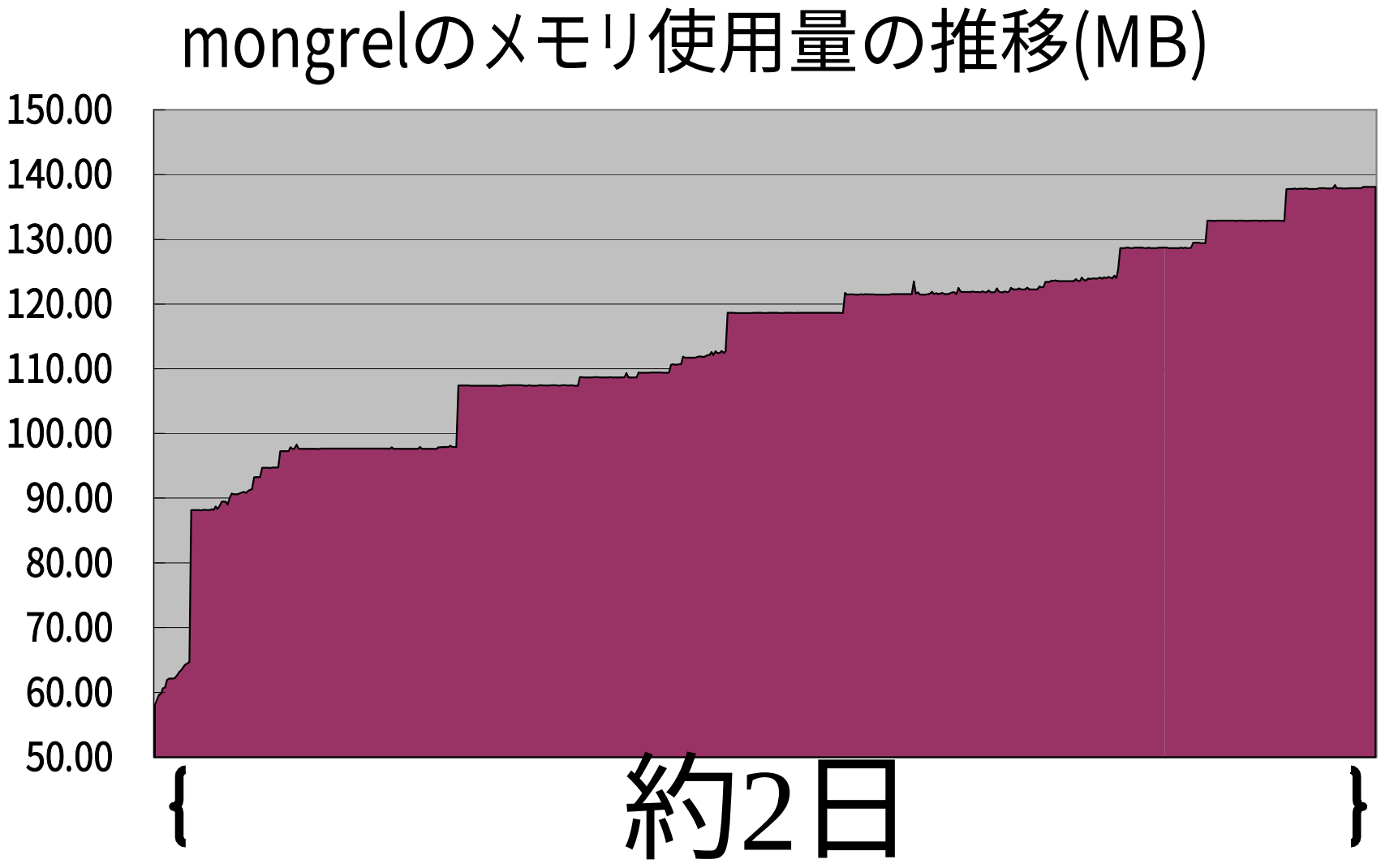
2-1.選択理由

- 簡単
- 大規模事例あり(twitter)
- 推奨(AWDwR 第二版)
- 他に選択肢がない
  - 実質Lighttpd + FCGIとの二択
    - Apache+FCGIは安定しない
    - Lighttpdは高負荷時の安定性に疑問？大規模サイトでの実績が少ない

2-2.メモリの使用量

- 大食い
  - 4〜5日稼動すると**8GB食いつぶす！**
    - 当時は1台辺り15プロセス
- 対策
  - 再起動用のシェルをcronで
    - でもほぼ毎日デプロイしてたり
  - アプリの改善
    - mongrelのメモリ使用量をモニタリング
    - 急激に増加していたら要チェック

2-2.メモリの使用量


mongrelのメモリ使用量の推移(MB)
150.00
140.00
130.00
120.00
110.00
100.00
90.00
80.00
70.00
60.00
50.00
約2日

2-3.安定性

- 「基本的に」安定している
  - 正常稼動中の障害は一度もない
- どんなとき？
  - 過負荷な状態が続いた時
  - メモリを食いつぶした状態が続いた時
- どうなる？
  - いつの間にかプロセスが消えてる！！
    - ログにも何も残っていない
    - ゾンビ化よりはマシだけど、原因究明が…

2-4. パフォーマンス

- プロセス数の見積もり
  - ピーク時で秒間50アクセスくらい
  - 100プロセス稼動
    - 2秒に1リクエスト処理できれば詰まらない
- 実際
  - 平均0.2秒 / リクエスト
    - 1秒超えるリクエストは全体の5%
    - DB:レンダリングの時間比は3:2
    - やっぱりDBが重い、でもRailsも重い方？

3.スケールアウトについて

- アプリケーションサーバのスケールアウト
- DBの分散アクセス対応
- Magic Multi-Connections
- ActsAsReadonlyable

3-1.APサーバのスケールアウト

- なんだか難しそうに聞こえるけど...
- ほとんどの場合、問題は二つ

(1)セッションの共有
(2)DBの分散アクセス対応

=> Railsで問題になるのは **(2)**

3-2.DBの分散アクセス対応

- ActiveRecordはDB複数接続未対応
- 独自実装？
  - 時間も技術もないので難しい
- プラグイン？
  - 良さそうなのが幾つか
  - Magic Multi-Connections
  - ActsAsReadonlyable

=> どちらを使うべきか？

3-3.Magic Multi-Connections

- 不採用
  - 既存ロジックの修正箇所が多い
    - ほぼすべてのDB関連処理が対象
    - **Slave::**Restaurant.find(...)
  - 初回アクセス時はマスタに行くバグ
  - :includeを沢山(入れ子)すると例外を吐くバグ
  - ソースが難解
  - コネクション張りすぎ
    - テーブルの数だけコネクションを張る

3-3.Magic Multi-Connections

- テーブルの数だけコネクションの実例

```
>3.times do Slave::Restaurant.find(:first) end
>3.times do Slave::User.find(:first) end
>ActiveRecord::Base.active_connections.each{|k,v| p v}
=>"ActiveRecord::Base"
   "Slave1::User"
   "Slave1::Restaurant"
   "Slave2::User"
   "Slave2::Restaurant"
```

100プロセス＊150テーブル ＝15000コネクション！

3-4.ActsAsReadonlyable

- 採用！
  - 導入が簡単(30分)
  - 既存ロジックの修正箇所が少ない
    - マスタでReadしたい時だけ修正
    - Restaurant.find(... **:readonly => false**)
  - 実装方法がシンプル
    - 自分達でメンテ出来る

3-4.ActsAsReadonlyable

- 導入方法
  - config/enviroment.rbに以下を追加

```
class << ActiveRecord::Base
  def inherited_with_readonlyable(child)
    child.acts_as_readonlyable :slave1, :slave2
    inherited_without_readonlyable(child)
  end
  alias_method_chain :inherited, :readonlyable
end
```

3-4.ActsAsReadonlyable

- 課題
- フェイルオーバー
  - 障害時に自動で切離したりしてくれない
    - 一度の参照でアウトなので大体エラーに
- 同時接続数
  - すべてのスレーブへ接続する
    - プロセス数 = コネクション数
  - 2~3DBくらい接続すれば十分

4.パフォーマンスについて

- routes
- cache
- query
- session

4-1.routes

- routesはほとんど使っていない！
- mod_rewrite + create_url(自作)
  - IIS時代の資産を継承
  - マルチドメイン対応
    - r.tabelog.com, o.tabelog.com,u.tabelog.com..
  - routesのパフォーマンス問題
    - とにかく遅いらしい

4-1.routes

- create_urlについて
  - url_forと引数互換
  - controller+actionのメソッドでハッシュ → パスの変換マップを記述
- 欠点
  - テストが書きづらい
  - メンテナンスコストが高い

=> routesとパフォーマンスを比較したい

4-2.cache

- memcached
  - 各アプリケーションサーバ上で稼動
  - キャッシュ対象
    - DBへのクエリ
    - バッチ生成のhtml
    - テンプレートはキャッシュしていない
  - 統計
    - アクセス数 => 約100リクエスト/秒
    - Hit率 => 約80% (512Mでも1Gでも同じ)
    - CPU使用率 => 約1%

4-2.cache

- QueryCache(MySQL)
  - 意外と優秀
  - Hit率 => 約50%

4-3.query

- 1:Nのincludeはなるべく避ける
  - 重ねると危険！
  - 100 * 100 = 10000
    - メモリ大食い
- クエリのチューニング
  - 一覧系画面では必須！
  - paginate, will_paginateは遅い

4-3.query

- 一覧系画面のチューニング方法

3.検索結果のレコード
→ 画面表示用のテーブル

2. 検索結果のID一覧
→ order句のテーブル

1. 検索結果の件数
→ where句のテーブル

AoR2007のライトニングトークで解説しています
http://cookpad.typepad.jp/lt/2007/09/97award_on_rail_8f37.html#12

4-4.session

- DBに格納(Rails標準)
- 1日に100万レコードくらい増える
  - 削除に時間がかかる
    - MyISAMだとテーブルロックなので致命的
  - レプリケーションしない
    - セッションはマスタのみを参照
    - replicate-ignore-table = database.table
    - MySQL5.0だとバイナリログの制御がDB単位でしかできないので、ネットワークの負荷はかかる

=> memcachedに変更予定

5.まとめ

- そんなに特別な事はしていない
  - routesくらい
- でもRailsでぜんぜん運用出来てます
  - **Rubyのおかげ！**
- やっぱり重いのはDB
  - あと地味に広告が……
- 食べログくらいのサイト規模でも普通に運用できる時代が来ているのかも

=> サービス開発に注力できるように！

ご清聴ありがとうございました