クラウドコンピューティングを生かすプログラミング言語

川中 真耶 (KAWANAKA Shinya)

私は誰？

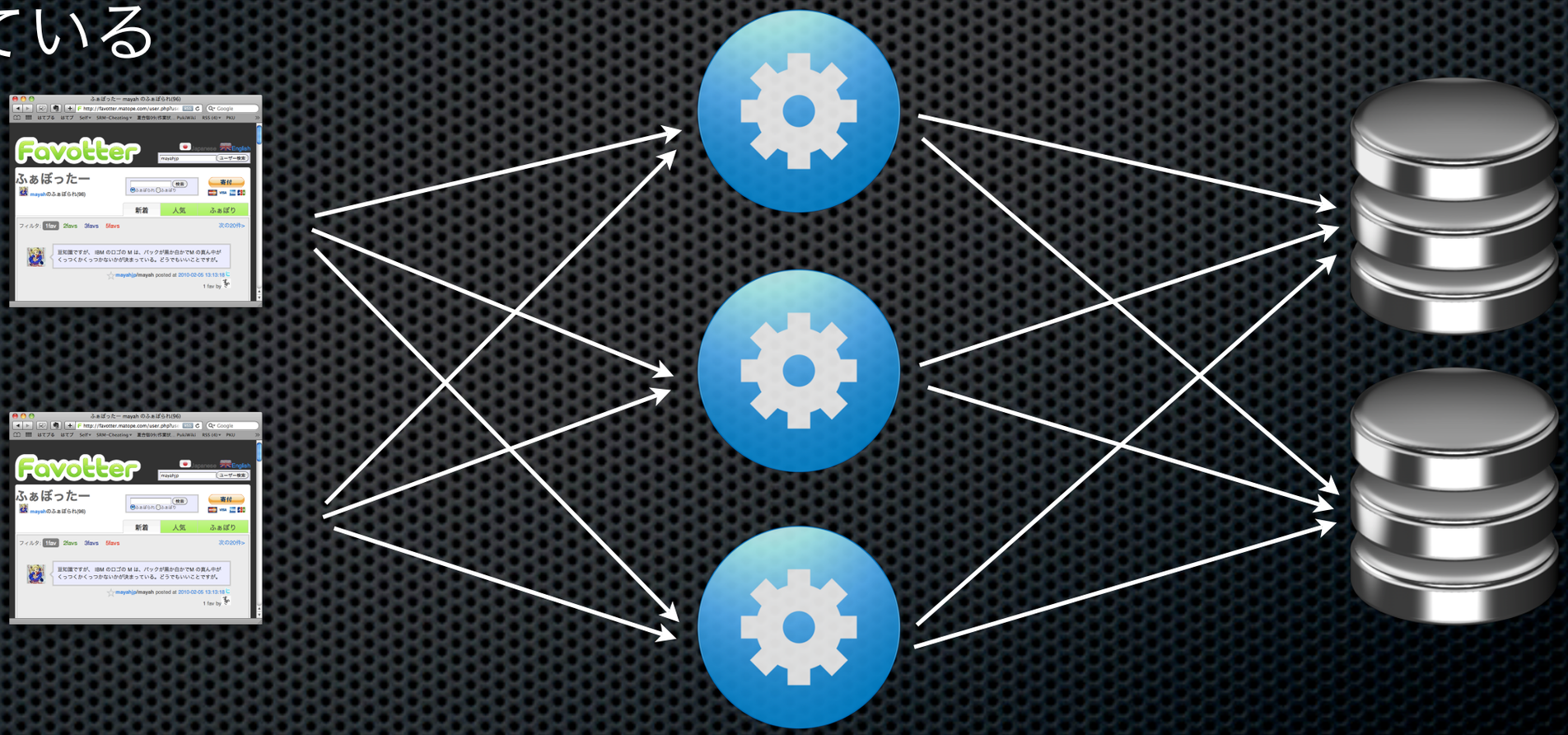
- 今は In-memory DB 等を作って製品の性能改善をやってます
- ネット上での活動
  - ハンドル: mayah, twitter: @mayahjp URL: http://mayah.jp/
- 好きな言語: Objective Caml
- ４月から東京大学大学院コンピュータ科学専攻博士課程 (予)
  - 平木研究室

今日のお話

- クラウド環境はスケールさせることが命
- スケールさせるための言語的サポート
- 今起こりつつある潮流

クラウドコンピューティングでよくあるアプリケーション (要するに Web App)

- みんなこの種のアプリケーションをスケールさせたいと思っている



MODEL 部 (DB) をスケールさせることがこれまで大変で重要だった

- MODEL 部は必ずアクセスされるところなので、並列性が確保できないとすぐにボトルネックになってくる
- RDB ではなかなかスケールしない (R が邪魔！)
- KVS (Key Value Store) などの、RDB よりも簡単な構造を使って (これまで保証されてきた何かを保証しないことで)、スケールさせるのが最近の流行

VIEW は大体勝手にスケールする

- VIEW の処理は、CONTROLLER に投げて帰ってきた結果を表示するだけ
- 並列性が高いので、台数を増やして回線の帯域を確保すれば、勝手にスケールする

CONTROLLER 部もスケールさせたい！

- 大体のアプリでは、MODEL から VIEW にデータを渡すだけなので、分散並列計算出来てスケールすることが多い
- しかし、処理やアプリケーションによっては、１台のマシンで計算を処理するには重すぎる場合がある
- 重い計算のときにクラウドらしく必要なリソースを借りて処理をする為にはスケールできなければならない
  - 例：　検索、インデックス生成、バッチ処理系

スレッドでの並列計算は問題が噴出

- みなさんお気づきの通り、スレッドで並列計算させるのは結構大変
  - 何度もデッドロックに悩まされる
  - 全状態をテストするのは大変でテスト漏れが出やすい
- 開発工数が膨らみやすい
- そもそも複数台で並列計算することが出来ない

昔からある並列計算のライブラリ

- MPI (= Message Passing Interface)
  - グリッドコンピューティングで使われてきた技術
  - 非常にローレベルな API で書きづらい
- OpenMP
  - C 中に、ここを並列化して欲しいと #pragma を書いておくと自動的に並列化してくれる(そんなに頭良くない)

ライブラリを使った最近の分散並列計算

- 最近の流行は MapReduce (Hadoop)
  - 複数のマシンに計算を投げて (map)、その結果を集めて新しい結果を作る (reduce)
- Google が論文発表
- オープンソースの Hadoop

しかしどちらかと言えば並列計算を言語に組み込んだものを使いたい

- 言語のサポートが受けられれば、シンタックスやセマンティクス上恩恵がある
- 並列計算のめんどくさいところを言語に組み込むことにより、開発者はよりロジックに集中できる
- 開発がシンプルになりやすい

Actor + Channel が並列計算を言語に組み込んだものが最近の流行

- Actor model (考え出されたのは 1973 年)
  - 非同期メッセージで動き、全てのものは actor 内で閉じている (つまり、自分以外の情報はいらない)
- Channel
  - メッセージを別の actor に送ることが出来る
  - 複数メッセージはキューに積まれる
- Erlang, Scala, Google Go 等がこの種の機能を持っている

Actor Model + Channel の利点

- わかりやすい
  - ロックが Channel に隠蔽され、ロックに関する煩わしさから解放される
  - 各 actor はもらったメッセージに対してどのように動くかという点(つまりロジック)だけに集中すればよい
- スケールする
  - 複数台でも通信レイテンシを除いて何の問題もなく動く

Actor Model の実際 -- Erlang を例として

- 例として Erlang 言語を取り上げる
  - 動的型付け (変数に型がない) の関数型言語
  - Actor (Erlang 用語ではプロセスと呼ばれる) を数万以上作ることが出来る (thread を数万作るのは無理！)
  - チャネルを使った非同期メッセージ通信で動く

Erlang の文法

- factorial の例
- 変数は大文字から始まる
- パターンマッチとガード
  条件が記述できる

```
% モジュール定義
-module(fact).

% factorial 関数
fac(0) -> 1;
fac(N) when N > 0 ->
    N * fac(N-1).
```

spawn で actor (プロセス) を作ることが出来る

- fac(10) を計算するプロセスを作った例
- プロセスを作ると、プロセス id が返ってくる
- spawn は、外部ノードを指定してプロセスを作ることも可能

```
Pid = spawn(fact,fac, [10])
```

プロセス間はメッセージ通信で情報をやりとりできる

- pid ! message で、プロセス pid にメッセージ送信
- 受け取り側は、receive でパターンマッチ
- 値ならなんでも送れる

```
% message 通信
Pid ! 10

% message receive
receive
  x -> io:format(“~w~n”, x)
end.
```

fac を別プロセスに計算させて値を受け取るには？

```
% calcfac に自分の pid (self())
と引数 10 を渡す
Pid = spawn(fact, calcfac,
                  [self(), 10]).

receive
  x -> io:format(“~w~n”, x)
end.
```

```
-module(fact).
 % calcfac 定義
calcfac(Pid, N) ->
    Pid ! fac(N).

% fac の定義は先ほどと同じ
fac(0) -> 1;
fac(N) when N > 0 ->
    N * fac(N-1).
```

メッセージでは関数も送れるので任意の計算を別プロセスにさせることができる

```
% 何か関数定義
f() -> ....

Pid = spawn(fact, apply,
                  [self(), f]).
receive
  x -> ... % do something
end.
```

```
% apply 定義。実行して値を返す
apply(Pid, F) ->
   Pid ! F().
```

ということは MapReduce も簡単

- map
  - たくさんプロセスを作って、関数を送って実行させればよい
- reduce
  - receive で値を受け取って値をまとめればよい

他の言語での Actor + Channel

```
func main() {
    ch := make(chan int);

    go func() {
        ch <- 10;
    }();

    <-ch; // 読み出し
}
```

Google Go

```
object ActorObj {
    def main(args : Array[String]) = {
        val a = actor {
            receive {
                case msg =>
                    System.out.println(msg)
            }
        }
        a ! "Hello, World!"
    }
}
```

Scala

今、言語は並列化が書きやすくなる方向にすすんでいる

- Multicore CPU もスケーラビリティを確保する言語のサポートが欲しいはず
  - Apple は Grand Central Dispatch で C/C++, Objective-C にタスク分割機能をサポート

```
// タスクを n 個作って分割実行
dispatch_apply(n, queue,
  ^(size_t i) {
    result[i] = do_something(i);
  }
);
```

他の並列化

- GPU による汎用演算 GPGPU (General Purpose GPU)
  - GPU は並列性が非常に高い処理が得意
  - GPU 用の C 言語 CUDA や OpenCL
  - そのうち Cloud 環境でも普通に使えるようになるかも知れない
    - NVIDIA RealityServer: GPU による3Dクラウドコンピューティング (2009年10月) というものも発表されている

まとめ

- クラウド環境を生かすには、
- 分散並列計算出来る必要がある
- Actor+Channel はスケールしやすく書きやすくてオススメ