

# mod\_perlをjob workerとして使う理由

## ～RSSクローラの設計から～



株式会社ミクシィ  
開発部 長野雅広

# 自己紹介

- 所属 株式会社ミクシィ  
開発部 システム運用グループ  
アプリケーション運用チーム
- Blog <http://blog.nomadscafe.jp/>
- CPANID kazeburo

# アジェンダ

- mixiのRSSクローラができるまで
- mod\_perlをjob workerとして使う(兼まとめ)

**RSSクローラができるまで**

# mixiでのRSS取得・表示



## 外部のブログ等



タイトル・URL・更新日付を取得しmixi上に表示

# mixiのRSSクローラ

- User-Agent
  - “mixi-crawler/2.00 (<http://mixi.jp/>)”
- mixiに登録されているBlogのRSSをクローラ

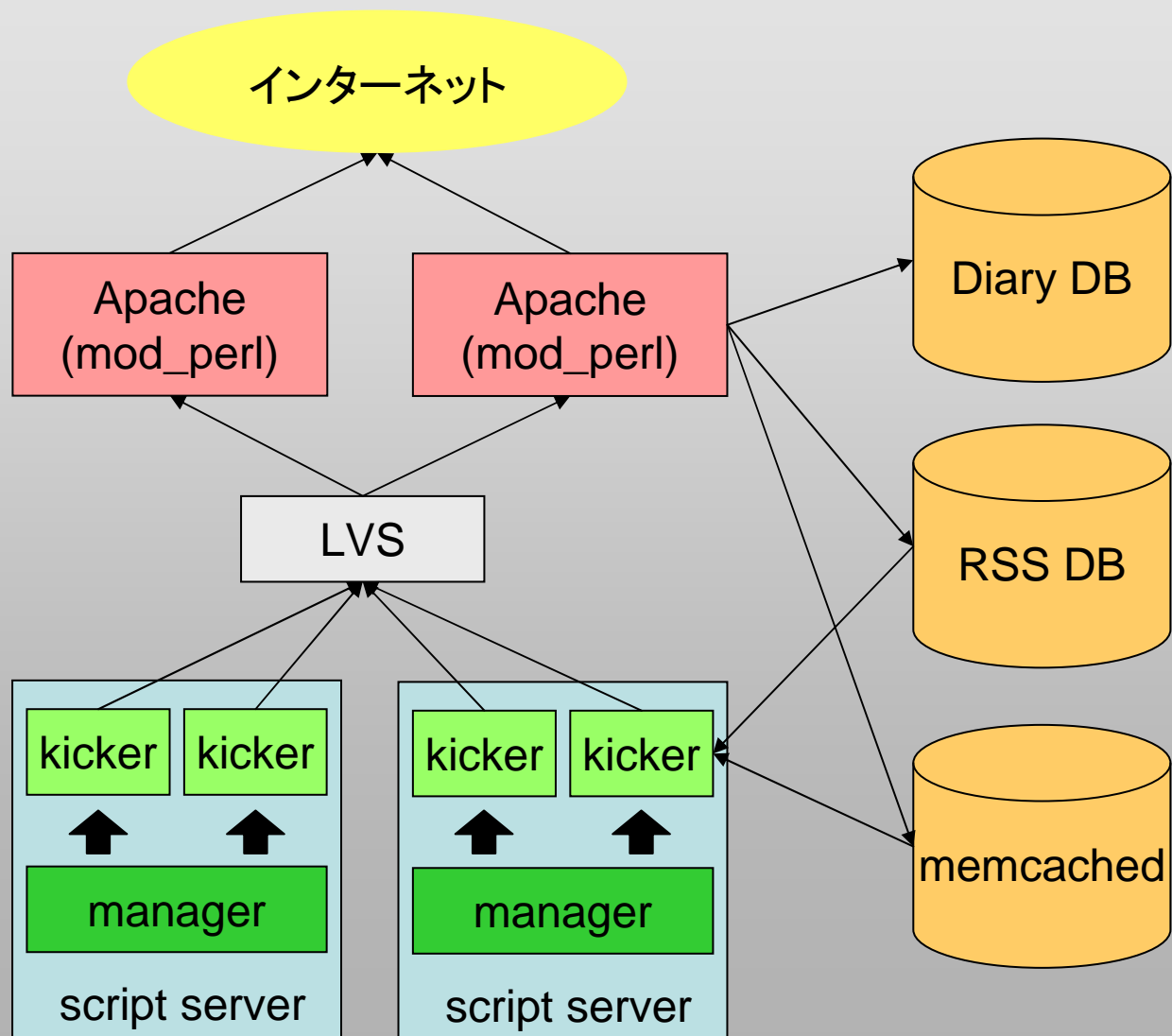
# 取得するRSSの数

**22万件（2008年5月現在）**

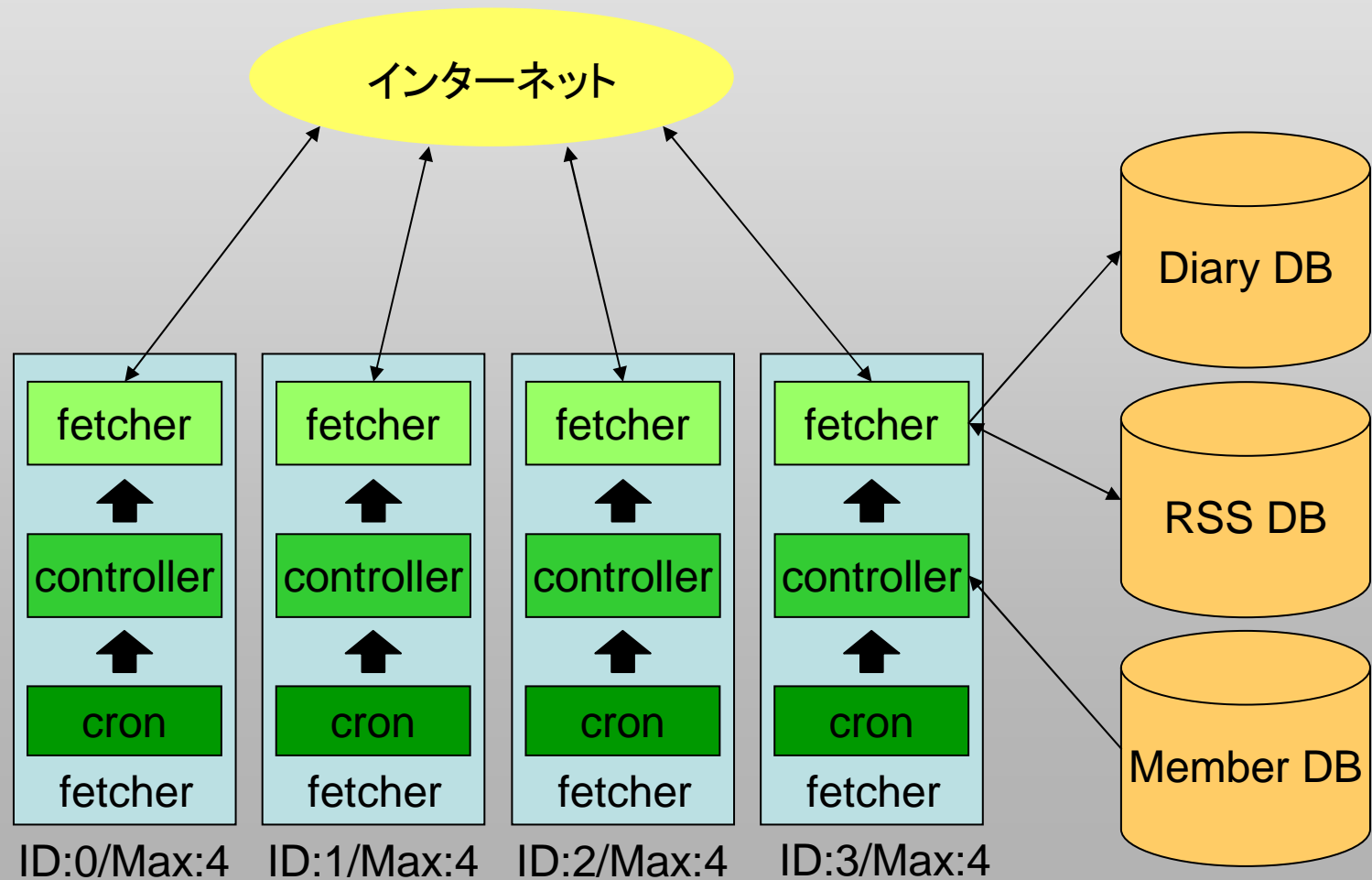
**19万3千件が正常に取得完了**

**2時間以内に全URLの巡回**

# 現在の構成



# 以前の設計

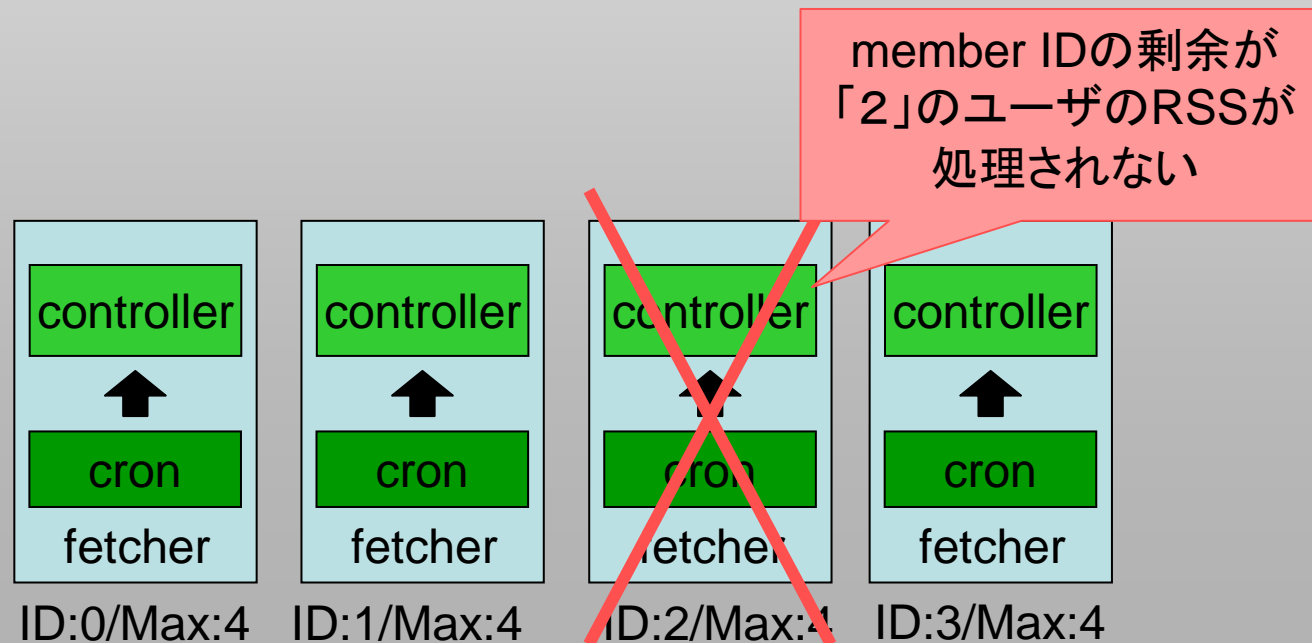


## 以前の設計の問題点

- **Fetcherサーバの耐障害性、Scalability**
  - Fetcherサーバの障害時
  - Fetcherサーバの増設
- **処理速度**
  - 都度のプロセス起動
  - 古いライブラリ
- **DBへの長い接続時間**
  - 運用上好ましくない

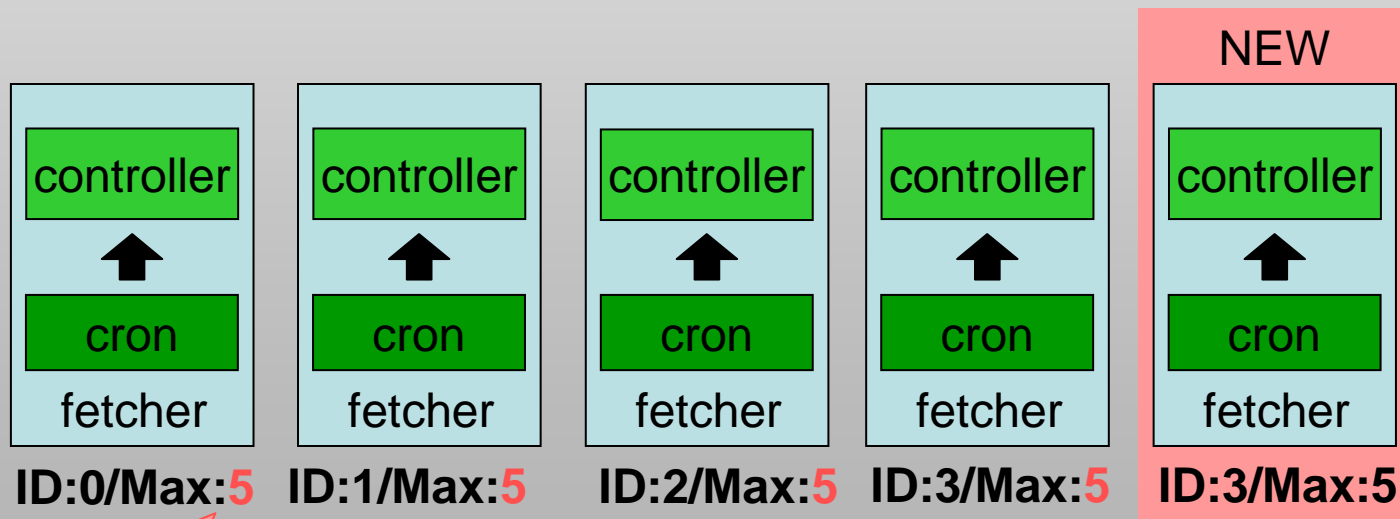
## Fetcherサーバの障害時

- IDが固定なので、手作業でそのIDを別のサーバや新規サーバに振り替えないとRSSの取得が停止する



# Fetcherサーバの増設

- すべてのサーバで「全体の台数」の設定の変更が必要



全体の台数を  
全台で設定する

# 処理速度

- **都度のプロセス起動**
  - DBI、XML系のライブラリを毎回読み込み直し
  - Perlのプロセス起動自体重い
- **古いライブラリ**
  - XML::Parserベースのライブラリは処理速度が遅い

# 新規に作成するにあたり

- **クローल時間の短縮**
  - member idを全部なめることをやめる
  - 2時間以内に1周する
- **XML::LibXMLベース**
- **対応するフォーマット**
  - Atom 1.0への対応
- **耐障害性とスケーラビリティ**

# それPla

RSS ParserはPlaggerのそれを利用できるYo!

# 「それPla」構想

- SubscriptionとPublish
- 非同期アクセス可能なAggregator

# 非同期のAggregator

- XangoやGunghoよりも軽い実装として  
**HTTP::Async**を使う
- Aggregator::Asyncの誕生
  - 本家にcommit

# HTTP::Asyncとは

- **Net::HTTP::NB (LWP同梱) を利用して非同期にHTTPアクセスする**

```
use HTTP::Async;
my $async = HTTP::Async->new;

$async->add( HTTP::Request->new( GET => 'http://mixi.jp/'      ) );
$async->add( HTTP::Request->new( GET => 'http://yapcasia.org/' ) );

while ( my $response = $async->wait_for_next_response ) {
    # Do some processing with $response
    ref $response ## HTTP::Response
}
}
```

# HTTP::Asyncの問題点

- **オンメモリ処理**
  - RSSのURLが数GBのファイルだったら？
- **connectは同期処理**
  - つながらないサーバがあるとブロックされてしまう
  - 不特定多数のURLへのリクエストには使いにくい

## HTTP::Asyncが適用しやすい場所

1. アクセス先のコンテンツのサイズが確認できる
2. connectが一瞬で完了するローカル接続

# 「それPla」構想

# mod\_perlをworkerとして使う

## mod\_perlを使うアイデア

- **最初のアイデアはメルマガの設計**

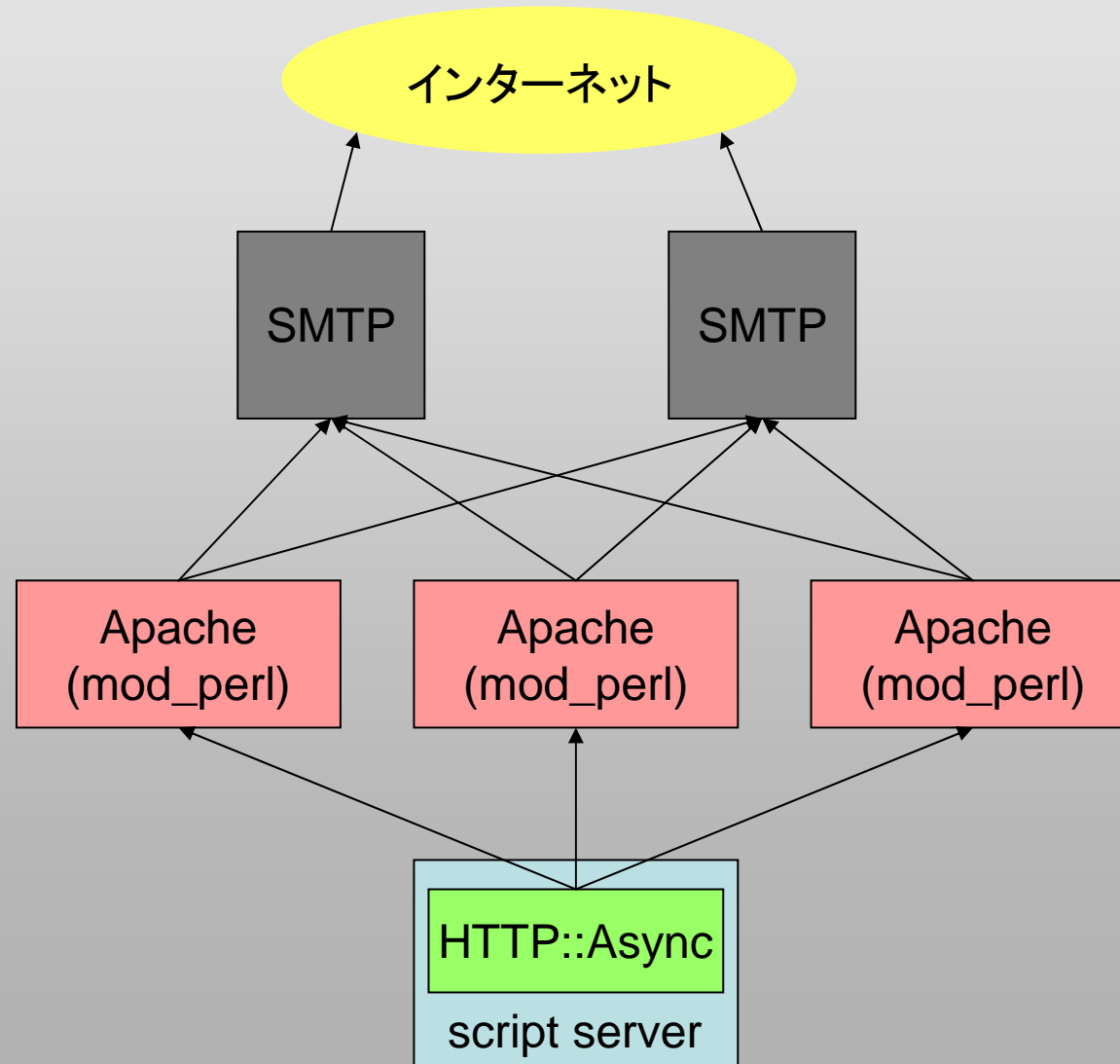
## メルマガ「コミュニティニュース」

- **1通1通ユーザ毎に異なる内容のメルマガ**
  - mixiのホームの内容が主
  - DBやmemcachedを参照しながら本文を作成して送信

## メルマガの効率化

- **mod\_perlで通常のページを作るように作る**
  - HTMLを書き出す代わりにメールを送る
  - そのアプリケーションサーバにRequestをHTTP::Asyncで送る
  - mod\_perl 1台でもMaxClientsの設定まで並行処理が可能
  - Scalabilityについては、アプリケーションサーバ(mod\_perlサーバ)を増やすだけ

# メルマガ「コミュニティニュース」



## RSSクローラへの展開

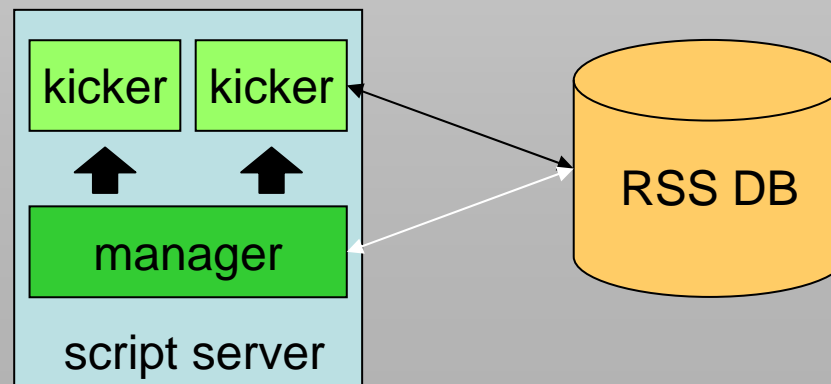
- リクエストを受けると、1つのFeedを取得し、解析、DBへの格納を行う
  - シンプルな1つのjobに
- そのサーバに対して、HTTP::Asyncを使いリクエストを行う
  - リクエストを行うプログラムをkickerと呼んでいる

# kickerの設計

- **kickerの耐障害性をあげる**
  - 冗長性を確保
- **Scalability**
  - サーバを足すだけを目指す
- **DBへの長時間の接続はしない**
  - 運用上の理由
- **動作速度**

# kickerとkicker manager

- **一定数処理したら終了するkicker**
  - 動作速度とDBとの接続時間の短縮の両立
- **DBとの接続は行わずkickerを管理するだけのmanager**



# kicker managerの設計

- POE
  - Wheel::Run
  - Component::Server::TCP
- PoCo::Server::TCPで任意のportをlisten
  - 簡単に状況を確認できる
  - nagiosで外部から監視

# kicker manager(一部ソースコード)

```
sub wheel_init {
    my ($kernel, $heap) = @_ [KERNEL, HEAP];
    $kernel->alias_set('wheel_ctl');
    $kernel->post('wheel_ctl','wheel_settimeout');
}

sub wheel_settimeout {
    my ($kernel, $heap) = @_ [KERNEL, HEAP];
    my $procs = $STASH->{procs};
    my $time = time() - 10*60; #10min
    my @freezed_procs;
    foreach my $id ( keys %$procs ) {
        push @freezed_procs,$id
        if $procs->{$id}->[0] < $time;
    }
    foreach (@freezed_procs) {
        $procs->{$_}->[1]->kill(9);
    }
    if ( keys %$procs < $OPT{MAX_PROCS} ) {
        $kernel->post('wheel_ctl','wheel_start');
    }
    $kernel->delay( 'wheel_settimeout', 1 );
}
```

```
sub wheel_start {
    my ($kernel, $heap) = @_ [KERNEL, HEAP];

    my $wheel = POE::Wheel::Run->new(
        Program => [qw/perl kicker.pl/],
        ErrorEvent => 'wheel_error',
        CloseEvent => 'wheel_close',
        StdoutEvent => 'wheel_stderr',
        StderrEvent => 'wheel_stderr',
    );

    my $pid = $wheel->PID;
    $kernel->sig( 'CHLD', 'wheel_sigchld' );
    $STASH->{procs}->{$wheel->ID} =
        [time(), $wheel];
}
```

# kicker managerの冗長性

- **2台以上のサーバで動く**
  - **処理が2重にならない工夫が必要になる**

# 処理が2重にならない工夫

## DBの設計

```
CREATE TABLE `rss_data` (  
  member_id int(10) unsigned NOT NULL default '0',  
  url varchar(255) NOT NULL default "",  
  last_crawl datetime NOT NULL default '1970-01-01 00:00:00',  
  fetcher_seed tinyint(3) unsigned NOT NULL default '0',  
  PRIMARY KEY (`member_id`),  
  KEY `last_crawl` (`last_crawl`, `fetcher_seed`),  
) TYPE=InnoDB;
```

urlをinsertする際にfetcher\_seedに0～59までの整数を入れておく

# 処理が2重にならない工夫 (2)

## 取得対象URLのSELECT

```
SELECT * FROM rss_data
  WHERE fetcher_seed=? AND
        last_crawl < ?
 ORDER BY last_crawl LIMIT 1000
```

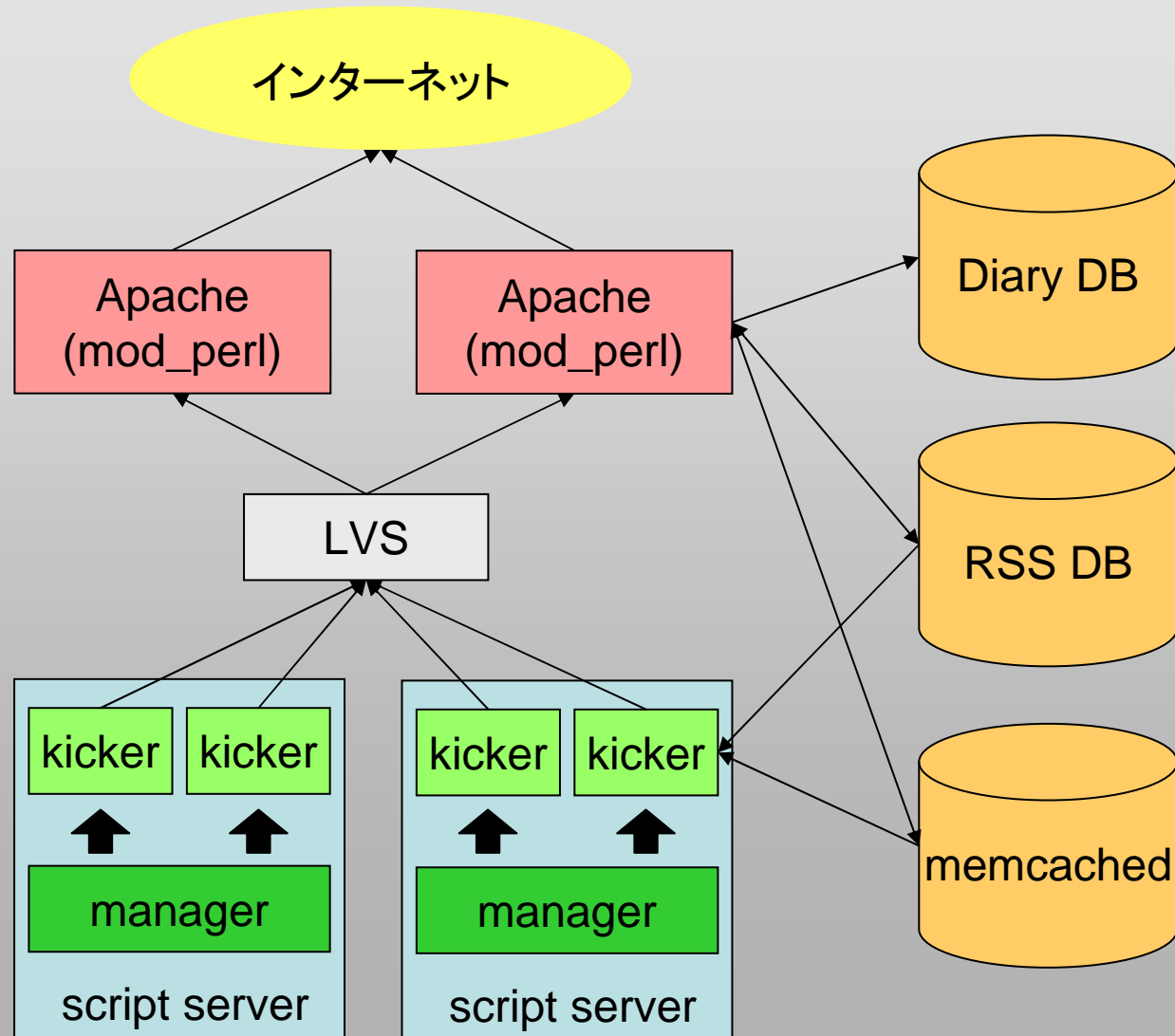
## fetcher\_seedはmemcachedのincrementで生成

```
my $incr = $memcached->incr( 'fetcher_seed' );
my $seed = $incr % 60;
$sth->execute(
    $seed,
    DateTime->now->subtract(hour=>2)->format('MySQL'),
)
```

# kickerのScalability

- kicker managerで複数個のkickerを同時起動
- kickerおよびkicker managerが動作するマシンを増やすことでrequestの回数を増やし続けていくことができる

# 完成



# mod\_perlをjob workerとして使う (兼まとめ)

## mod\_perlを使った利点

- Apacheで並列処理をみてる
- Apacheでメモリの管理もできる
  - MaxRequestPerChild
  - startup.plでのメモリ効率化
- Web Application Frameworkをそのまま活用できる
- デプロイツールがそのまま使える
- 監視も行いやすい
- サーバを足せばスケールする
- APIとしてもつかえる

## mod\_perlを使うTips

- **jobを細かく分割する**
  - 並行処理度を高める
  - 障害時の範囲を狭くすることができる
- **Requestを非同期に行うクライアントを用意**
  - HTTP::Asyncはお勧め

以上

## エンジニア募集してます

- mixiでは一緒に働くウェブアプリケーション・エンジニア、システム・エンジニアを募集しています

採用サイト: <http://mixi.co.jp/job/>

**質問などありましたらお気軽にどうぞ**