

memcached in mixi



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/>

PAUSE ID: kazeburo

アジェンダ

- memcachedとは
- mixiでの利用事例
- 分散アルゴリズムについて
- memcached互換アプリケーション

memcachedとは

memcachedって何？

- **分散メモリキャッシュシステム**
a distributed memory object caching system
 - keyとvalueのペアをメモリー上で管理するサーバ
 - オンメモリなので再起動等でデータが失われる
 - データが指定した量を超えるとLRU (Least Recently Used) の順で古いデータが消えていく
- **典型的な使い方は、DBのレスポンスのキャッシュ**
- **非常に高速・安定している**

memcachedを利用しているサービス

mixi はてな livedoor

LiveJournal Vox

Facebook YouTube

Digg Twitter

wikipedia

さまざまな言語のクライアント

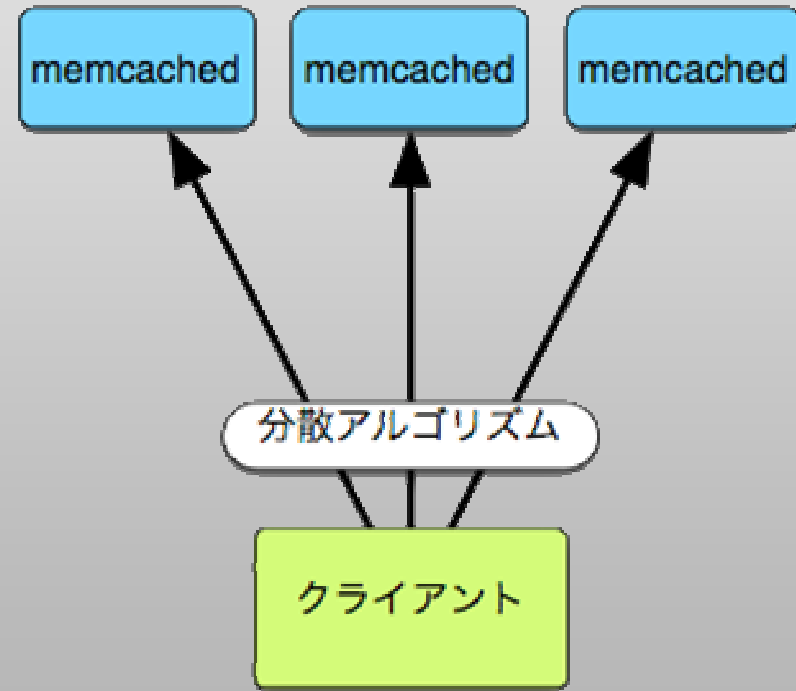
- Perl
- PHP
- Python
- Ruby
- Java
- Lua
- C
- C#

アプリケーションへの組み込み

- MySQL
- PostgreSQL
- Apache
- nginx

memcachedの「分散」について

- サーバ側ではなく、クライアントで分散を行う
- クライアントに実装された分散アルゴリズムにより、key毎に異なるサーバへ保存される



memcachedの使い方

memcachedのインストール

- **libeventが必要**

```
$ wget http://www.danga.com/memcached/dist/memcached-1.2.5.tar.gz
$ tar xzf memcached-1.2.5
$ cd memcached-1.2.5
$ ./configure --enable-thread --prefix=/path/to
$ make
$ sudo make install
```

memcachedの起動

```
$ /path/to/memcached -p 11211 -m 64 -d
```

- **起動オプション**
 - p ポート (11211がデフォルト)
 - m 容量 (MB)
 - d バックグラウンド起動
 - vv デバッグメッセージ

基本的な使い方

Perlのクライアントで

Cache::Memcached

- memcachedと同じBrad Fitzpatrick氏によるもの
- 最新版は1.24
- <http://search.cpan.org/dist/Cache-Memcached/>

Cache::Memcachedの使い方

```
my $memcached = Cache::Memcached->new({  
    servers => [qw/localhost:11211/  
}]);  
$memcached->add( "foo" , "bar" , 120);
```

...

```
my $memcached = Cache::Memcached->new({  
    servers => [qw/localhost:11211/  
});  
say $memcached->get( "foo" ) ## bar
```

get / get_multi

```
my $memcached = Cache::Memcached->new({  
    servers => [qw/localhost:11211 192.168.0.2:11211/  
});  
$memcached->add( "foo", "bar", 120);  
$memcached->add( "hoge", "fuga", 120);  
  
my $value = $memcached->get( "foo" ); ## bar  
  
my $values = $memcached->get_multi( "foo", "hoge" );  
## { foo => "bar", hoge => "fuga" }
```

- **get_multiは複数のサーバに対して非同期にリクエストを行うので高速**

add / replace / set

```
my $memcached = Cache::Memcached->new({  
    servers => [qw/localhost:11211 192.168.0.2:11211/]  
});  
  
$memcached->add( "foo" , "bar" , 120);  
  
$memcached->replace( "foo" , "bar" , 120);  
  
$memcached->set( "foo" , "bar" , 120);
```

- **引数はすべて、(key、値、保存期間(秒))**

delete

```
my $memcached = Cache::Memcached->new({  
    servers => [qw/localhost:11211 192.168.0.2:11211/]  
});  
  
$memcached->delete( "foo" , 120);
```

- deleteの2つ目の引数は、**上書き禁止の秒数**
- setを使うと上書きできる

memcachedの使い方・その他

- perldoc Cache::Memcached

- **プロトコルの詳細**

<http://code.sixapart.com/svn/memcached/trunk/server/doc/protocol.txt>

mixiでのmemcached利用事例

マシン / OS

- **memcached専用のマシンを用意**
 - Pentium4 / Pentium D
 - memory 4GB
 - Linux Kernel 2.6.x x86_64
- **バージョン**
 - memcached 1.2.5
- **台数**
 - 100台以上

容量 / 使用量 / Hit率

- **容量**
 - サーバー台につきmemcachedをメモリ3GBで1つ起動

```
memcached -d -p 11211 -u nobody -m 3000 -c 30720 -P /path/to/memcached.pid
```

- 最も大きいmemcachedのpoolは、3GB*76台
- **メモリ使用量**
 - 88%
- **cache Hit率**
 - 94%

memcachedのレポート (抜粋)

ip	ver	uptime	connect	max	used	ratio	hit	
1	1.2.5	16 days	11301	3000.00 MB	2654.74 MB	(88.49%)	93.40%	
2	1.2.5	15 days	11288	3000.00 MB	2652.96 MB	(88.43%)	91.02%	
3	1.2.5	16 days	11254	3000.00 MB	2654.09 MB	(88.47%)	93.61%	
4	1.2.5	15 days	11283	3000.00 MB	2652.82 MB	(88.43%)	93.18%	
~~~~~								
*	1.2.5	16 days	11300	3000.00 MB	2654.23 MB	(88.47%)	94.48%	
*	1.2.5	16 days	11300	3000.00 MB	2654.80 MB	(88.49%)	93.68%	
*	1.2.5	16 days	11285	3000.00 MB	2653.39 MB	(88.45%)	93.82%	
~~~~~								
					222.66 GB	195.24 GB	(87.69%)	93.94%
~~~~~								

# 監視

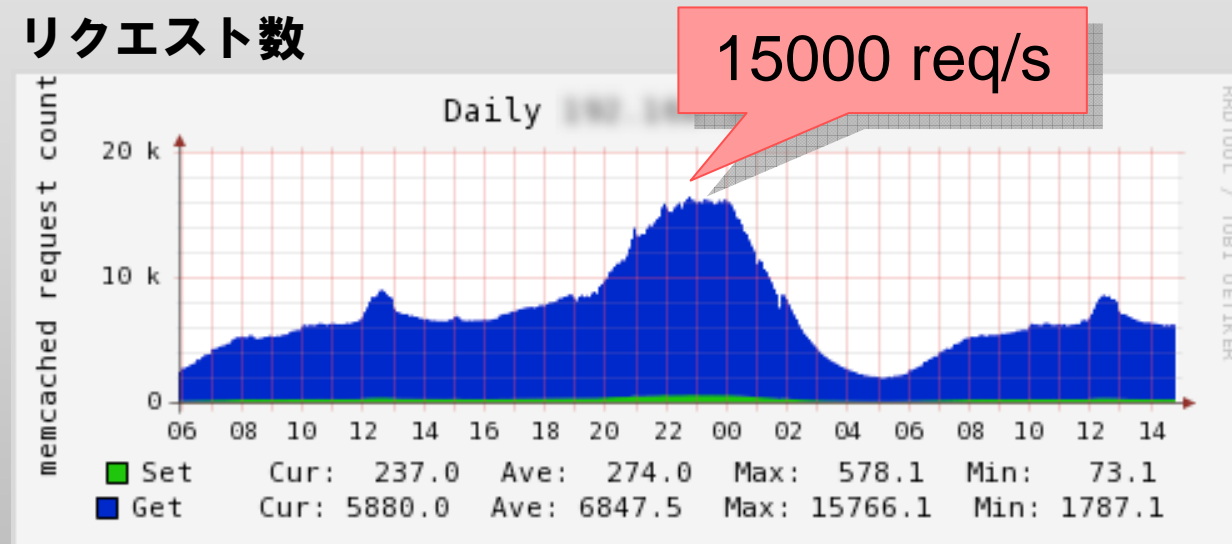
- **nagiosを利用**

```
check_tcp -H $HOSTADDRESS$ -p 11211 -t 5 ¥  
-E -s 'stats¥r¥nquit¥r¥n' -e 'uptime' -M crit
```

- **自社製監視ツールとrrdtoolでグラフ化**
  - 通信量
  - クエリー数
  - 使用量

# memcachedの性能グラフ(1)

リクエスト数

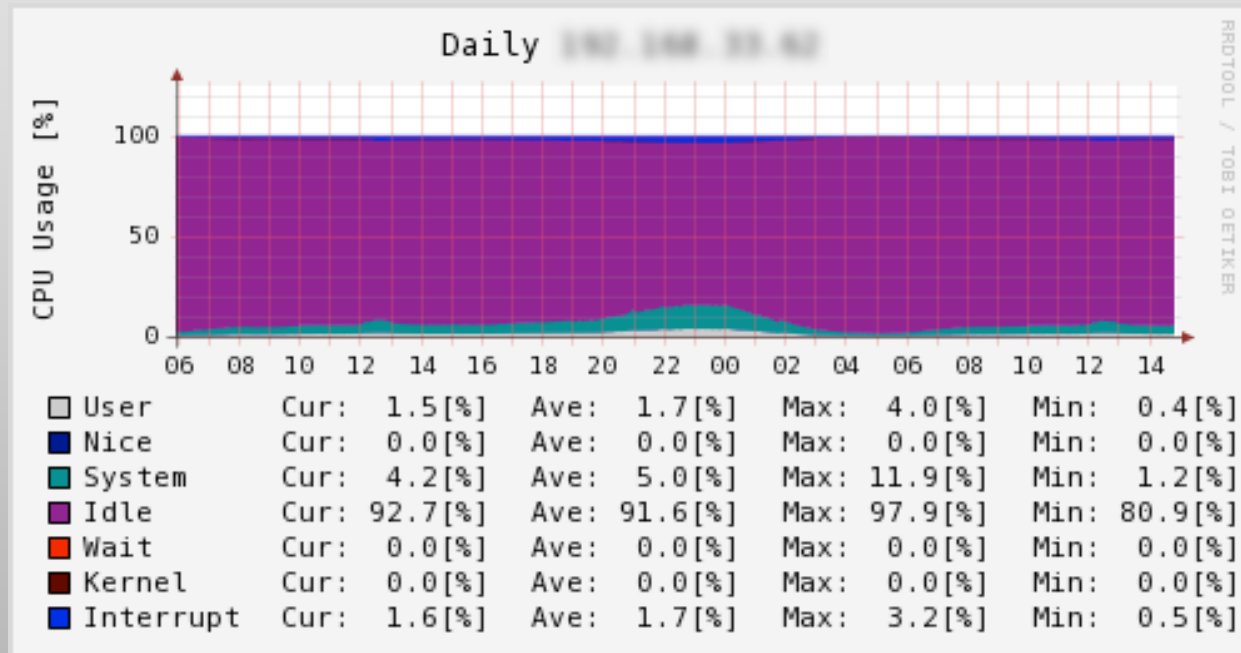


最も極端にアクセスが集中しているmemcached



# memcachedの性能グラフ (3)

## CPU



ロードアベレージは常に1以下

## 特定のmemcachedへのアクセス集中

- homeに表示されるニュースや設定等のkeyはアクセスの多いページで集中的に読まれる
- 1台のサーバに数百Mbpsの通信が起きてしまう
  - アプリケーションでkeyを書き換えて複数のmemcachedへ分散を行う

# assoc_findでbusy loop問題

- memcached内の、assoc.cのassoc_findで無限loopが起きる問題
  - 一切応答ができない状態
- MLにも報告があるが、再現させることができないので、修正に時間がかかっている

# assoc_findでbusy loop問題への対応

- linked listが壊れるのが原因
- memcachedのhash tableを途中で大きくする  
assoc_expandが起こらないように調整

```
diff -ur memcached-1.2.5.orig/assoc.c memcached-1.2.5/assoc.c
--- memcached-1.2.5.orig/assoc.c      2008-03-04 04:13:45.000000000 +0900
+++ memcached-1.2.5/assoc.c          2008-04-07 18:29:52.000000000 +0900
@@ -449,7 +449,7 @@
 typedef unsigned char ub1; /* unsigned 1-byte quantities */

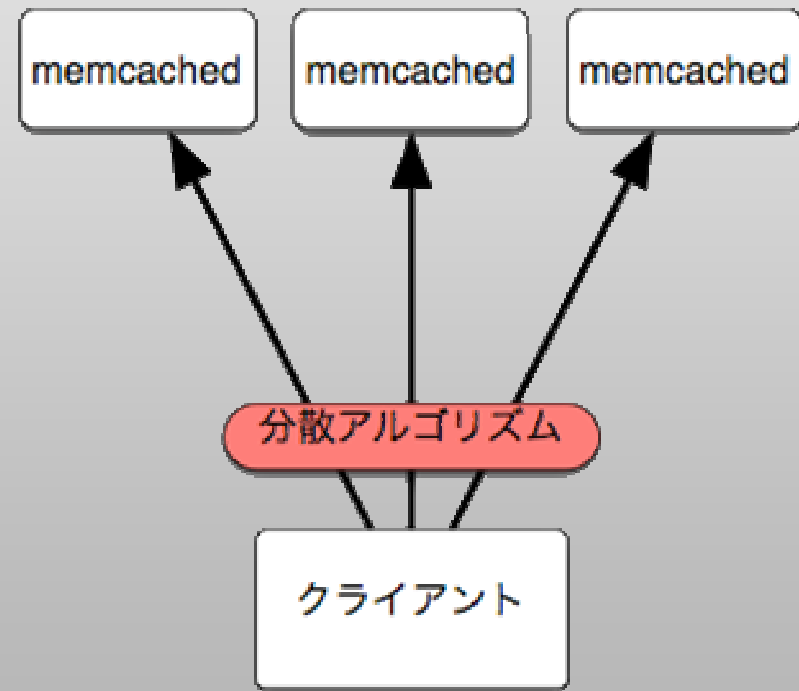
 /* how many powers of 2's worth of buckets we use */
-static unsigned int hashpower = 16;
+static unsigned int hashpower = 25;
```

<http://lists.danga.com/pipermail/memcached/2008-May/006816.html>

# 分散アルゴリズム

# memcachedの分散

- 分散の実装はクライアント側で行う
- key毎に任意のサーバを選択する



# Cache::Memcachedの分散方法

- keyのCRC値と、サーバ台数の剰余

```
use String::CRC32;  
  
my @servers = qw/localhost:11211 192.168.0.2:11211/;  
my $server = @servers[ String::CRC32::crc32($key) % @servers ];
```

# Cache::Memcachedの問題点

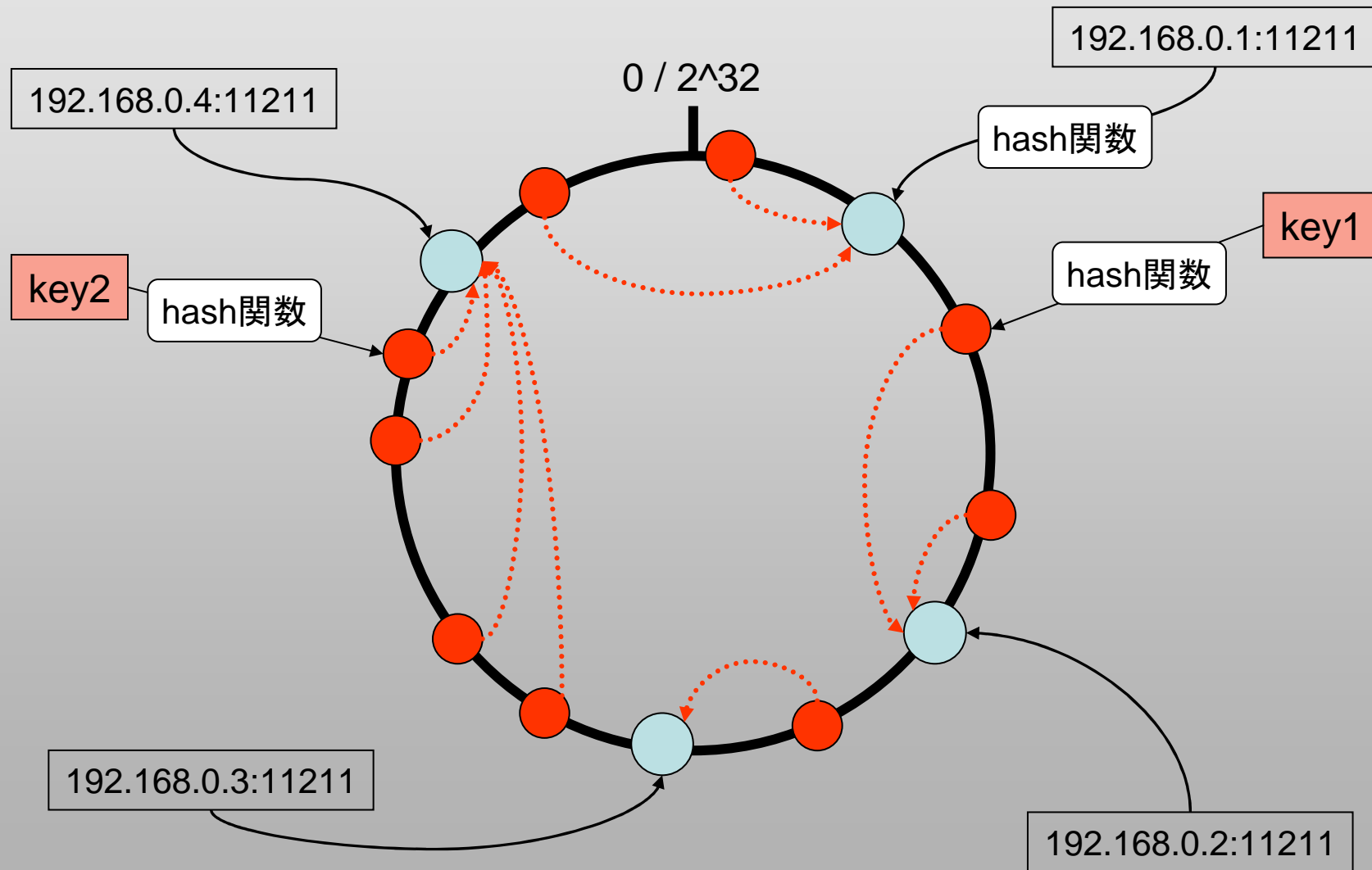
- サーバの追加・削除・変更を行うとHit率に大きな影響がでる

```
my $m = Cache::Memcached->new({servers=>[qw/a b c d/]});  
for(1..1000){ $m->add($_, $_) }
```

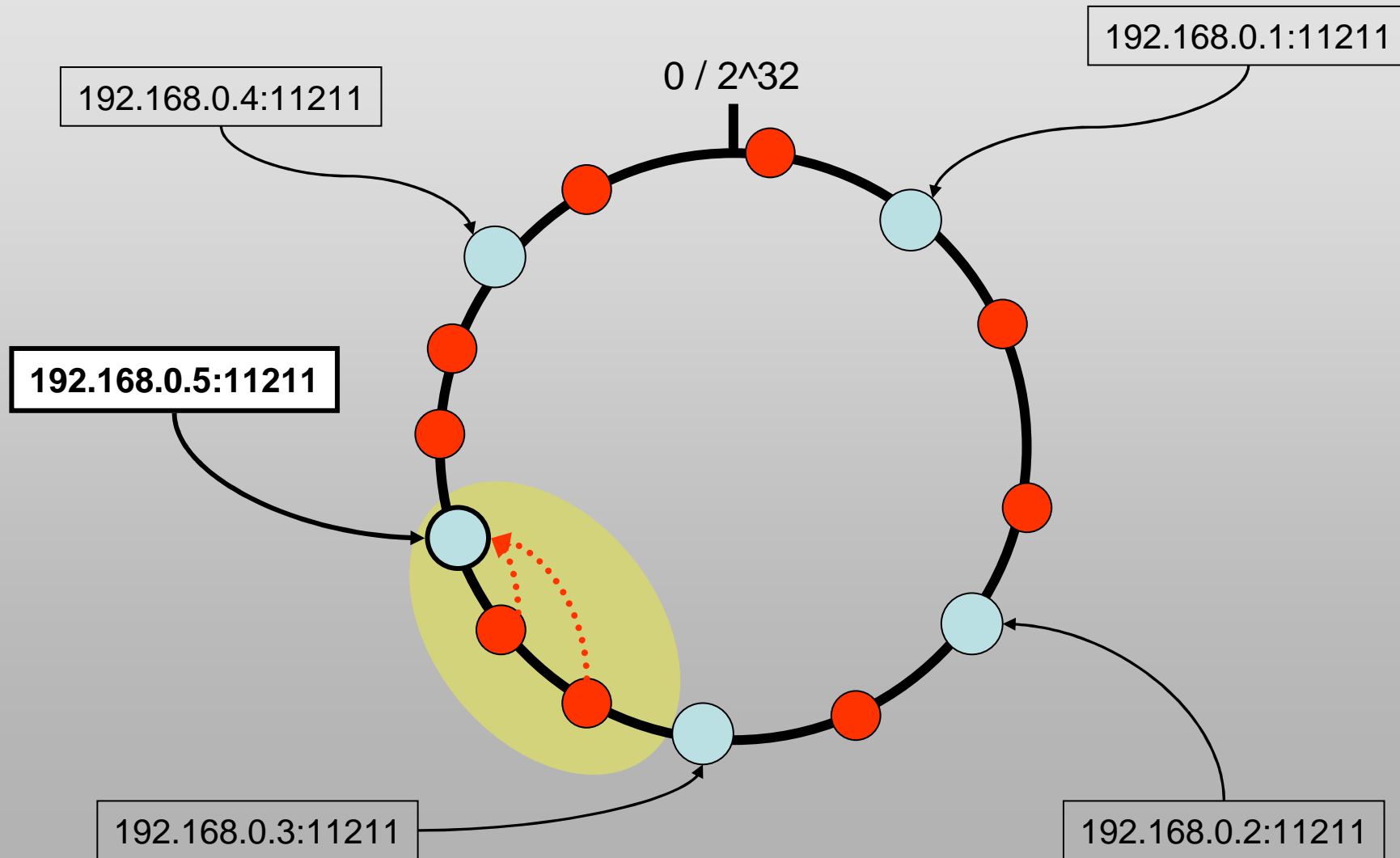
```
my $hit=0;  
map { $hit++ if $m->get($_) == $_ } 1..1000;  
say $hit ## 1000 = 100%
```

```
my $m = Cache::Memcached->new({servers=>[qw/a b c d e/]});  
my $hit=0;  
map { $hit++ if $m->get($_) == $_ } 1..1000;  
say $hit ## 197 = 19%
```

# Consistent-Hashing



# Consistent-Hashing (node追加時)



# ketama

- last.fmの人が作った汎用Consistent-Hashingライブラリ
- 仮想ノード
  - 1つのnodeに対して複数個の仮想ノードを作る
- PHPのバインディング

<http://www.audioscrobbler.net/development/ketama/>

# 分散アルゴリズムが改善されたクライアント

- `Set::ConsistentHash+Cache::Memcached`
- `Cache::Memcached::libmemcached`
- `Cache::Memcached::Fast`

# Set::ConsistentHash+Cache::Memcached

- IRCやMLで話題に
- Set::ConsistentHash  
Brad FitzpatrickのCPANモジュール  
従来のCache::Memcachedを組み合わせているらしい

# Cache::Memcached::libmemcached

- Daisuke Maki氏作成
- MySQLのChief ArchitectのBrianAker氏が作成したlibmemcachedライブラリを利用したもの
- かなり高速に動作

(検証しているがConsistent-Hashingモードで動かすことができません)

CPAN: <http://search.cpan.org/dist/Cache::Memcached::libmemcached>

MakiさんのBlog: <http://mt.endeworks.jp/d-6/>

# Cache::Memcached::Fast

- Tomash Brechko氏作成
- Cで書かれたPerlのmemcachedクライアント
- ketamaを再実装したconsistent hashingをサポート
- 高速かつ軽量
- mixiで採用予定

CPAN: <http://search.cpan.org/dist/Cache::Memcached::Fast>

# Cache::Memcached::Fast (ベンチマーク)

```
use Cache::Memcached::Fast;
use Benchmark qw/cmpthese/;

my $fast = Cache::Memcached::Fast->new({
    servers => ¥@servers,
    ketama_points => 150,
});
$fast->set( "x" . $_ => rand ) for 1..500;

foreach my $num ( qw/1 10/ ) {
    my @keys = map { "x" . $_ } 0..$num;
    cmpthese( $n => {
        parser_xs => sub { $m->get_multi(@keys) },
        fast => sub { $fast->get_multi(@keys) },
    });
}
```

# Cache::Memcached::Fast (ベンチマーク)

- memcachedサーバ 6台での実験結果

```
$ perl bench.pl
### 1
          Rate parser_xs      fast
parser_xs 5425/s          --    -92%
fast      65217/s      1102%    --
### 10
          Rate parser_xs      fast
parser_xs 2392/s          --    -94%
fast      37500/s      1467%    --
```

# Cache::Memcached::Fast (Hit率テスト)

```
my $m = Cache::Memcached::Fast->new({  
    servers=>[qw/a b c d/], ketama_points=>150});  
for(1..1000){ $m->add($_, $_) }
```

```
my $hit=0;  
map { $hit++ if $m->get($_) == $_ } 1..1000;  
say $hit ## 1000 = 100%
```

```
my $m = Cache::Memcached::Fast->new({  
    servers=>[qw/a b c d e/], ketama_points=>150});  
my $hit=0;  
map { $hit++ if $m->get($_) == $_ } 1..1000;  
say $hit ## 766 = 76.6%
```

サーバ増加後のHIT率は  
n=元の台数 α=増加台数  
 $100 - ( 100 * \alpha / (n + \alpha) )$

# Cache::Memcached::FastのTips

- Cache::Memcachedのようにmemcachedとのconnectionを維持しない
- 自前でインスタンスをキャッシュ

```
package Mixi::Memcached;
use strict;
use Cache::Memcached::Fast;

my $mmc;
sub new {
    my $self = bless {}, shift;
    $self->{mmc} = $mmc ||= Cache::Memcached::Fast->new({...});
    $self;
}
```

# Cache::Memcached::FastのTips

- rehashしない
  - サーバに接続できない場合にhashを作り直して別のサーバに接続しない
- *max_failures*と*failure_timeout*のオプションを使用
  - *failure_timeout*の時間内に*max_failures*に達したサーバには接続しない

# Consistent-Hashingへの移行

- mixiではConsistent Hashingへの移行を準備中
- 新旧両方のインスタンスを用意して同時に書き込むようにして移行期間を設ける

```
package Mixi::Memcached;  
  
sub set {  
    my ($self, $key, $val, $expire) = @_;  
    $self->fast->set($key, $val, $expire);  
    $self->mmc->set($key, $val, $expire);  
}
```

# memcached互換 アプリケーション

# プロトコル互換

- memcachedのプロトコルは非常に簡単
- さまざまな機能追加のアイデア
  - replication機能
  - ストレージを永続化できるものに変更
  - 別の言語で実装

# 一部紹介

- TokyoTyrant
  - tokyo cabinetに保存
- Flared
  - qdbmに保存。レプリケーション等
- repcached
  - memcachedにレプリケーションの機能をつけるpatch
- mmcmmod
  - ストレージをモジュラー形式に
- memcachedb
  - BerkeleyDBに保存

DSAS開発者の部屋:最近のmemcached界隈の賑わいについてのメモ  
<http://dsas.blog.klab.org/archives/51181526.html>

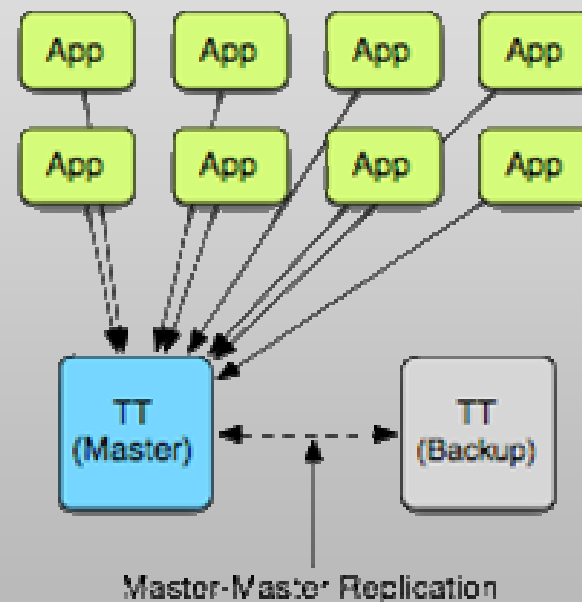
# TokyoTyrant

- **HyperEstraiierの平林氏作成**
  - TokyoCabinet DBMのネットワークインターフェイス
  - memcachedと互換プロトコル
  - ディスクに書き込む形ながら非常に高速に動作
- **mixiで導入実績**
  - ログイン時間の保存

データベースマネージャ TokyoCabinet: <http://tokyocabinet.sourceforge.net/>

# TokyoTyrant (ログイン時間DB)

- Master-Masterの2台構成
- 10000QPS以上
- 常時1万コネクション以上
- ハードウェア
  - CPU: QuadCore Xeon
  - Memory: 8GB
  - HDD: SAS 146GB



Tokyo Tyrantによる耐高負荷DBの構築: <http://alpha.mixi.co.jp/blog/?p=166>

# まとめ

# まとめ

- memcachedは高速なメモリキャッシュサーバ。扱いも非常に簡単
- 分散アルゴリズムはConsistent Hashingを選ぶ
- TokyoTyrantも使える

以上です

## [CM] 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/jobs/>

**質問などありましたらお気軽にどうぞ**