PEMANFAATAN HIPHOP FOR PHP PADA WEB SERVER

Gunawan[1], Suwandi Halim[2]

STMIK Mikroskil
Jl. Thamrin No. 112, 124, 140 Medan 20212
gunawan@mikroskil.ac.id[1], wandi.lin13@gmail.com[2]

Abstrak

HipHop for PHP merupakan sebuah teknologi optimasi *website* PHP yang dikembangkan oleh *Facebook* dengan mentransformasi *source code* PHP menjadi C++ yang optimal dan dengan mengorbankan beberapa fungsi PHP yang jarang digunakan. Artikel ini membahas penggunaan *HipHop for* PHP pada *web server* dan pengujiannya dengan cara membandingkannya dengan *web server* yang tidak menggunakan *HipHop for* PHP atau tanpa optimasi apapun. Uji coba terhadap pemakaian *HipHop for* PHP adalah dengan meng-*compile* sebuah *website* PHP atau kumpulan *source code* PHP menjadi *binary files* C++ hingga bisa diakses oleh sebuah *web server*. Juga akan dilakukan *benchmark* terhadap *HipHop for* PHP dengan menggunakan *Apache Benchmark* untuk mengetahui performansi *HipHop for* PHP dalam menangani *request* (permintaan) *user* dalam jumlah yang banyak. Dari hasil pengujian didapatkan bahwa *HipHop for* PHP mampu meningkatkan performansi *web server* untuk *website* PHP hingga lebih dari 50%.

Kata Kunci: *PHP, HipHop, web server, benchmark.*

1. Pendahuluan

PHP *Hypertext Preprocessor* merupakan salah satu bahasa *scripting* bersifat *server-side* yang cukup sering dijumpai pada pembuatan *website*. PHP berjalan pada *web server*, sehingga eksekusi *script* PHP bergantung pada kemampuan *server* yang digunakan. Semakin besar *script* PHP yang dieksekusi, maka penggunaan sumber daya CPU juga semakin besar, begitu pula dengan penggunaan memori. Dengan demikian, ketika semakin banyak *user* yang mengakses *web server* dan banyaknya *script* PHP yang harus dieksekusi, maka semakin besar pula sumber daya CPU dan memori yang digunakan, sehingga respon *web server* terhadap permintaan (*request*) masing-masing *user* akan menjadi semakin lambat. Hal inilah yang sering terjadi pada *website* yang sering diakses oleh banyak *user* setiap harinya, dimana salah satu contohnya adalah *Facebook*.

Sebagai salah satu *website* yang paling banyak diakses pengguna Internet setiap harinya, yaitu tercatat lebih dari 500 juta pengguna aktif (http://www.facebook.com/press/info.php?statistics), *Facebook* berusaha menjaga agar respon dari *web server* mereka terhadap permintaan masing-masing *user* tetap stabil dan cepat. Untuk mengatasi masalah tersebut, biasanya para pemilik *website* akan meningkatkan kemampuan CPU dan memorinya (*hardware*), namun tidak demikian halnya pada *Facebook*. *Developer Facebook* memperkenalkan sebuah teknologi baru yang dapat membuat eksekusi *script* PHP menjadi lebih cepat dan efisien, yaitu *HipHop for* PHP.

[1] Gunawan, S.Kom., M.T.I.

Salah satu *web developer Facebook* yang ikut mengembangkan teknologi baru tersebut, *Haiping Zhao*, menyebutkan bahwa dengan menggunakan teknologi baru tersebut, *Facebook* mampu mengurangi pemakaian sumber daya CPU dan memori hingga mencapai 50% (http://developers.facebook.com/blog/post/358/). Pemakaian sumber daya yang efisien akan menyebabkan respon terhadap permintaan *user* semakin cepat pula. *HipHop for* PHP mentransformasi *source code* PHP menjadi *source code* C++ yang optimal, kemudian mengkompilasinya dengan *compiler* g++. Dengan demikian, *Facebook* tidak perlu menghabiskan biaya besar pada *hardware server* untuk menangani masalah tersebut. *HipHop for* PHP telah diluncurkan sebagai *project open source* pada tanggal 2 Februari 2010, sehingga setiap *web developer* bisa mencoba teknologi milik *Facebook* tersebut untuk mengoptimalkan performansi *website* mereka.

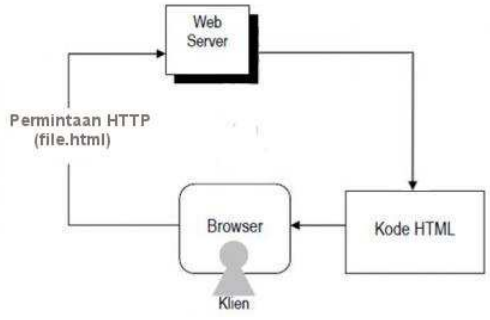
2. Kajian Pustaka

2.1. PHP *Hypertext Preprocessor*

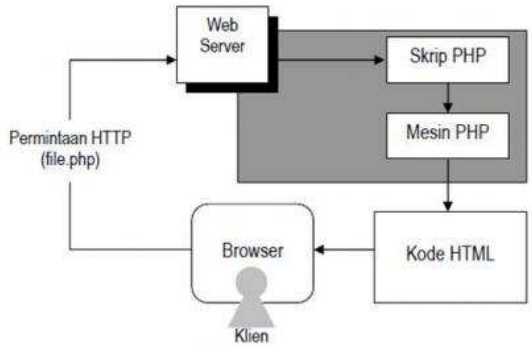
Menurut dokumen resmi PHP, PHP merupakan singkatan dari PHP *Hypertext Preprocessor*. PHP merupakan bahasa berbentuk skrip yang ditempatkan dalam *server* dan diproses di *server*. Hasil dari pemrosesan tersebut yang akan dikirimkan ke klien tempat pemakai menggunakan *browser*. Secara khusus, PHP dirancang untuk membentuk aplikasi *web* dinamis. Artinya, PHP dapat membentuk suatu tampilan berdasarkan permintaan terkini. Pada awalnya PHP dirancang untuk diintegrasikan dengan *web server Apache*. Namun, belakangan *PHP* juga dapat bekerja dengan *web server* seperti *Personal Web Server* (PWS), *Internet Information Server* (IIS), dan Xitami [1].

Model kerja HTML diawali dengan permintaan suatu halaman *web* oleh *browser* (Gambar 1). Berdasarkan *Uniform Resource Locator* (URL) atau dikenal dengan sebutan alamat Internet, *browser* mendapatkan alamat dari *web server*, mengidentifikasi halaman yang dikehendaki, dan menyampaikan segala informasi yang dibutuhkan oleh *web server*. Selanjutnya, *web server* akan mencarikan *file* sesuai permintaan dan memberikan isinya ke *web browser* (atau yang biasa disebut *browser* saja). *Browser* yang mendapatkan isinya segera melakukan proses penerjemahan kode HTML dan menampilkannya ke layar pemakai [1].

Model kerja PHP prinsipnya sama dengan kode HTML (Gambar 2). Hanya saja, ketika berkas PHP diminta dan didapatkan oleh *web server*, isinya segera dikirimkan ke mesin PHP dan mesin inilah yang memproses dan memberikan hasilnya (berupa kode HTML) ke *web server*. Selanjutnya, *web server* menyampaikan ke klien [1].



Gambar 1 Model Kerja HTML



Gambar 2 Model Kerja PHP

2.2. *Hip Hop for* PHP

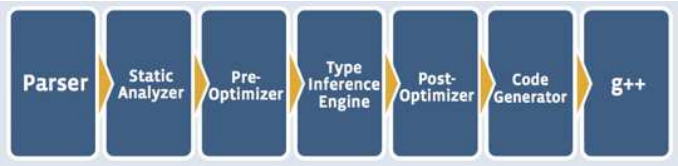
HipHop for PHP mengubah kode sumber PHP menjadi C++ yang sangat optimal. *HipHop for* PHP dikembangkan oleh *Facebook* dan dirilis sebagai *open source* di awal tahun 2010. *HipHop* mengubah kode sumber PHP menjadi C++ yang sangat optimal dan kemudian mengkompilasinya dengan g++ untuk membangun *binary files*. Kode program tetap dituliskan dalam PHP yang lebih sederhana, kemudian *HipHop* mengeksekusi kode sumber dengan cara yang secara semantik sama dan mengorbankan beberapa fitur yang jarang digunakan, sebagai ganti untuk peningkatan kinerja. *Facebook* melihat sekitar 50% pengurangan dalam penggunaan CPU ketika melayani jumlah *traffic* yang sama yang dibandingkan dengan *Apache* dan PHP. API dari *Facebook* mampu melayani *traffic* 2 kali lebih besar dengan CPU 30% lebih sedikit [2].

Salah satu tujuan desain eksplisit yang membawa menuju *HipHop* adalah kemampuan untuk melanjutkan menulis logika kompleks langsung di dalam PHP. Perusahaan-perusahaan dengan kode basis PHP yang besar biasanya akan menulis ulang fungsionalitas kompleks mereka langsung sebagai ekstensi C atau C++. Melakukan hal tersebut hanya mengurangi jumlah orang untuk menangani seluruh kode basis perusahaan. Dengan tetap menggunakan logika ini di dalam PHP, *Facebook* mampu bergerak cepat dan menjaga jumlah *engineer* yang banyak untuk menangani seluruh kode basis. *HipHop* bukanlah solusi yang tepat untuk setiap orang yang memanfaatkan PHP. *HipHop* akan berguna bagi perusahaan dengan infrastruktur PHP yang sangat besar yang tidak ingin menulis ulang logika yang kompleks ke dalam C atau C++ [2].

Proses transformasi (Gambar 3) meliputi tiga tahap utama [2], yaitu:

1. *Static Analysis*, dimana dilakukan pengumpulan informasi mengenai apa yang dideklarasikan dan setiap ketergantungan (*dependencies*).
2. *Type Inference*, dimana dilakukan pemilihan tipe data yang sesuai atau mendekati dengan nilai yang diberikan pada suatu variabel, misalnya *String*, *Array*, *Object*, dan lainnya.
3. *Code Generation*, dimana pada bagian ini dihasilkan perintah dan ekspresi C++ yang sesuai dengan perintah dan ekspresi pada PHP.

Adapun proses transformasi secara lengkap dapat dilihat pada gambar 3 berikut ini.



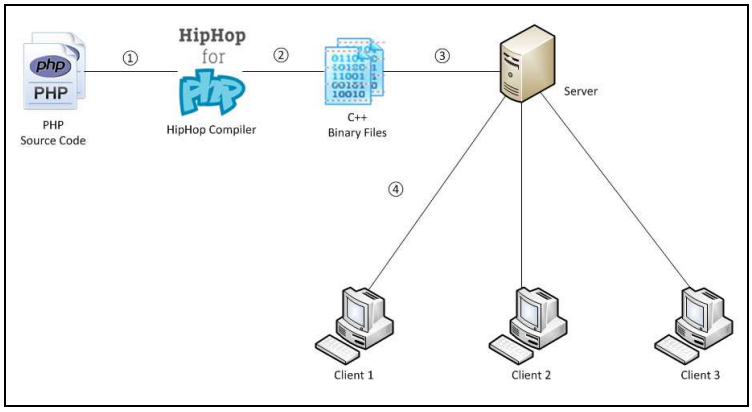
Gambar 3 Proses Transformasi PHP ke C++

Source code HipHop for PHP bisa di-*download* melalui *website* github.com. Github merupakan sebuah layanan berbasis *web* untuk pengembangan proyek yang digunakan sebagai *version control source code* sebuah proyek. Dalam hal ini, *Facebook* memanfaatkan github untuk menyebarkan *source code HipHop for* PHP (https://github.com/facebook/hiphop-php/). Selain *source code*, dari *website* tersebut juga bisa didapatkan informasi-informasi mengenai *HipHop for* PHP, seperti cara instalasi, *bugs*, sampai *update* terbaru pada *source code HipHop for* PHP.

Saat ini *HipHop for* PHP hanya mendukung sistem operasi Linux Ubuntu 64 bit. Sebelum melakukan pengujian, ada beberapa paket perangkat lunak (*software packages*) yang

harus di-*install* terlebih dahulu. *Software Packages* terdiri dari *tools*, *compiler*, dan *library* yang nantinya dibutuhkan untuk men-*download*, membangun (*build*), dan menjalankan (*run*) *HipHop for* PHP.

Adapun tahapan-tahapan yang dilakukan untuk menggunakan *HipHop for* PHP dapat dilihat pada Gambar 4.



Gambar 4 Cara Kerja *HipHop for* PHP

Penjelasan dari Gambar 4:

1. Kumpulan *source code* PHP akan ditransformasi dan di-*compile* menggunakan *compiler HipHop for* PHP.
2. Hasil transformasi dan *compile* adalah berupa kumpulan *executable binary files* C++ yang telah dioptimasi.
3. *Executable binary files* yang dihasilkan kemudian akan dijalankan sebagai *server.* Dalam hal ini, *server* yang dimaksud adalah *server* bawaan yang dimiliki oleh *HipHop for* PHP.
4. *Client* bisa mengakses *website* PHP yang telah di-*compile* melalui *web browser* pada *server* yang telah dijalankan tersebut. [3]

3. Pembahasan

3.1. Spesifikasi Sistem Pengujian

Benchmark pada artikel ini menggunakan *virtual machine* sebagai *sandbox* pengujian. Adapun spesifikasi dari *virtual machine* yang digunakan adalah:

a. Sistem operasi Linux Ubuntu 10.10 – 64 bit
b. *Processor* Intel(R) Core(TM) i7 CPU Q740 @ 1.73 GHz (8 CPUs)
c. *Memory* 2048 MB
d. Tidak ada *swap memory.*

Tool yang digunakan untuk *benchmark HipHop for* PHP adalah *Apache Benchmark* (http://httpd.apache.org/docs/2.0/programs/ab.html). *Apache Benchmark* merupakan sebuah *tool* yang melakukan *benchmark* dengan mengirimkan *request* ke *server* HTTP setiap detik dengan tujuan untuk mengetahui kemampuan *server* dalam menangani *request,* terutama dalam jumlah yang banyak dan konkuren. *Apache Benchmark* merupakan *tool* yang digunakan melalui perintah terminal. Adapun perintah yang digunakan untuk melakukan *benchmark* sebagai berikut:

```
ab –n 100 –c 5 [URL]
```

Parameter *-n 100* memberitahukan *Apache Benchmark* untuk mengirimkan 100 *request* per detik, dan *-n 5* memberitahukan *Apache Benchmark* untuk melakukan 5 *request* secara konkuren. Jadi, total *request* yang dilakukan adalah 100*5 = 500 *request*.

Apache Benchmark akan memberikan informasi mengenai kecepatan *server* dalam bentuk statistik beserta ikhtisarnya dalam menangani *request*. Adapun *script* yang digunakan untuk menguji *server* dengan *Apache Benchmark* adalah bench.php yang bisa di-*download* dari *Official* PHP *Development Repository* (http://svn.php.net/viewvc/php/php-src/trunk/Zend/benc.php?view=markup).

Script bench.php merupakan sebuah *script* yang digunakan untuk melakukan *benchmark* PHP terhadap *web server*. Di dalam *script* tersebut terdapat beberapa fungsi yang akan dijalankan secara berkelanjutan untuk menguji kemampuan *web server* dalam menangani *request*. Secara tidak langsung *script* tersebut juga sudah mencakup beberapa sintaks yang umum digunakan pada PHP, seperti *echo*, *print*, *global*, *foreach*, *strlen*, *number_format*, dan lainnya. Adapun operasi-operasi yang dilakukan oleh *script* tersebut adalah sebagai berikut:

1. Perulangan
2. *Nested Loop*
3. Operasi matematika
4. Pemanggilan fungsi
5. Penggunaan *array*
6. Operasi-operasi pada matriks
7. Fungsi *ackermann*
8. Mencari bilangan *fibonanci* ke-n
9. *Hashing*
10. Menghasilkan bilangan acak (random)
11. *Heap Sort*
12. Fungsi *sieve*
13. Fungsi *strcat* (menggabungkan *string*)
14. Menghitung waktu eksekusi

Isi *script* bench.php untuk *benchmark* PHP terhadap *web server* adalah sebagai berikut.

```
<?php
if
(function_exists("date_default_timezone_
set")) {
  date_default_timezone_set("UTC");
}

function simple() {
  $a = 0;
  for ($i = 0; $i < 1000000; $i++)
    $a++;

  $thisisanotherlongname = 0;
  for ($thisisalongname = 0;
$thisisalongname < 1000000;
$thisisalongname++)
    $thisisanotherlongname++;
}

/****/
function simplecall() {
  for ($i = 0; $i < 1000000; $i++)
    strlen("hallo");
}

/****/
function hallo($a) {
}

function simpleucall() {
  for ($i = 0; $i < 1000000; $i++)
    hallo("hallo");
}

/****/
function simpleudcall() {
  for ($i = 0; $i < 1000000; $i++)
    hallo2("hallo");
}

function hallo2($a) {
}

/****/
function mandel() {
  $w1=50;
  $h1=150;
  $recen=-.45;
  $imcen=0.0;
  $r=0.7;
  $s=0; $rec=0; $imc=0; $re=0; $im=0;
  $re2=0; $im2=0; $x=0; $y=0; $w2=0;
  $h2=0; $color=0;
  $s=2*$r/$w1;
  $w2=40;
  $h2=12;
  for ($y=0 ; $y<=$w1; $y=$y+1) {
    $imc=$s*($y-$h2)+$imcen;
      for ($x=0 ; $x<=$h1; $x=$x+1) {
        $rec=$s*($x-$w2)+$recen;
        $re=$rec;
        $im=$imc;
        $color=1000;
        $re2=$re*$re;
        $im2=$im*$im;
        while( ((($re2+$im2)<1000000) &&
$color>0)) {
          $im=$re*$im*2+$imc;
          $re=$re2-$im2+$rec;
          $re2=$re*$re;
```

```
          $im2=$im*$im;
          $color=$color-1;
        }
      if ( $color==0 ) {
        print "_";
      } else {
        print "#";
      }
    }
    print "<br>";
    flush();
  }
}

/****/
function mandel2() {
  $b = " .:,;!/>)|&IH%*#";
  //float r, i, z, Z, t, c, C;
  for ($y=30; printf("\n"), $C = $y*0.1
- 1.5, $y--;){
    for ($x=0; $c = $x*0.04 - 2, $z=0,
$Z=0, $x++ < 75;){


  for ($r=$c, $i=$C, $k=0; $t = $z*$z -
$Z*$Z + $r, $Z = 2*$z*$Z + $i, $z=$t,
$k<5000; $k++)
if ($z*$z + $Z*$Z > 500000) break;
echo $b[$k%16];
    }
  }
}

/****/
function Ack($m, $n) {
  if($m == 0) return $n+1;
  if($n == 0) return Ack($m-1, 1);
  return Ack($m - 1, Ack($m, ($n - 1)));
}

function ackermann($n) {
  $r = Ack(3,$n);
  print "Ack(3,$n): $r\n";
}

/****/
function ary($n) {
  for ($i=0; $i<$n; $i++) {
    $X[$i] = $i;
  }
  for ($i=$n-1; $i>=0; $i--) {
    $Y[$i] = $X[$i];
  }
  $last = $n-1;
  print "$Y[$last]\n";
}

/****/
function ary2($n) {
  for ($i=0; $i<$n;) {
    $X[$i] = $i; ++$i;
    $X[$i] = $i; ++$i;
    $X[$i] = $i; ++$i;
    $X[$i] = $i; ++$i;
    $X[$i] = $i; ++$i;

    $X[$i] = $i; ++$i;
    $X[$i] = $i; ++$i;
    $X[$i] = $i; ++$i;
  }
  for ($i=$n-1; $i>=0;) {
    $Y[$i] = $X[$i]; --$i;
    $Y[$i] = $X[$i]; --$i;
    $Y[$i] = $X[$i]; --$i;
    $Y[$i] = $X[$i]; --$i;
    $Y[$i] = $X[$i]; --$i;

    $Y[$i] = $X[$i]; --$i;
    $Y[$i] = $X[$i]; --$i;
    $Y[$i] = $X[$i]; --$i;
    $Y[$i] = $X[$i]; --$i;
    $Y[$i] = $X[$i]; --$i;
  }
  $last = $n-1;
print "$Y[$last]\n";
}

/****/
function ary3($n) {
  for ($i=0; $i<$n; $i++) {
    $X[$i] = $i + 1;
    $Y[$i] = 0;
  }
  for ($k=0; $k<1000; $k++) {
    for ($i=$n-1; $i>=0; $i--) {
      $Y[$i] += $X[$i];
    }
  }
  $last = $n-1;
  print "$Y[0] $Y[$last]\n";
}

/****/
function fibo_r($n) {
  return(($n < 2) ? 1 : fibo_r($n - 2) +
fibo_r($n - 1));
}

function fibo($n) {
  $r = fibo_r($n);
  print "$r\n";
}

/****/
function hash1($n) {
  for ($i = 1; $i <= $n; $i++) {
    $X[dechex($i)] = $i;
  }
  $c = 0;
  for ($i = $n; $i > 0; $i--) {
    if ($X[dechex($i)]) { $c++; }
  }
  print "$c\n";
}

/****/
function hash2($n) {
  for ($i = 0; $i < $n; $i++) {
    $hash1["foo_$i"] = $i;
    $hash2["foo_$i"] = 0;
  }
  for ($i = $n; $i > 0; $i--) {
    foreach($hash1 as $key => $value)
$hash2[$key] += $value;
```

```
  }
  $first = "foo_0";
  $last  = "foo_".($n-1);

  print "$hash1[$first] $hash1[$last]
$hash2[$first] $hash2[$last]\n";
}

/****/
function gen_random ($n) {
  global $LAST;
  return( ($n * ($LAST = ($LAST * IA +
IC) % IM)) / IM );
}

function heapsort_r($n, &$ra) {
  $l = ($n >> 1) + 1;
  $ir = $n;

  while (1) {
    if ($l > 1) {
      $rra = $ra[--$l];
    } else {
      $rra = $ra[$ir];
      $ra[$ir] = $ra[1];
      if (--$ir == 1) {
        $ra[1] = $rra;
         return;
      }
    }
    $i = $l;
    $j = $l << 1;
    while ($j <= $ir) {
      if (($j < $ir) && ($ra[$j] <
$ra[$j+1])) {
        $j++;
      }
      if ($rra < $ra[$j]) {
        $ra[$i] = $ra[$j];
        $j += ($i = $j);
      } else {
        $j = $ir + 1;
      }
    }
    $ra[$i] = $rra;
  }
}

function heapsort($N) {
  global $LAST;

  define("IM", 139968);
  define("IA", 3877);
  define("IC", 29573);

  $LAST = 42;
  for ($i=1; $i<=$N; $i++) {
    $ary[$i] = gen_random(1);
  }
  heapsort_r($N, $ary);
  printf("%.10f\n", $ary[$N]);
}

/****/
function mkmatrix ($rows, $cols) {
  $count = 1;
  $mx = array();
    for ($i=0; $i<$rows; $i++) {
      for ($j=0; $j<$cols; $j++) {
        $mx[$i][$j] = $count++;
      }
    }
  return($mx);
}

function mmult ($rows, $cols, $m1, $m2)
{
  $m3 = array();
  for ($i=0; $i<$rows; $i++) {
    for ($j=0; $j<$cols; $j++) {
      $x = 0;
      for ($k=0; $k<$cols; $k++) {
        $x += $m1[$i][$k] * $m2[$k][$j];
      }
      $m3[$i][$j] = $x;
    }
  }
  return($m3);
}

function matrix($n) {
  $SIZE = 30;
  $m1 = mkmatrix($SIZE, $SIZE);
  $m2 = mkmatrix($SIZE, $SIZE);
  while ($n--) {
    $mm = mmult($SIZE, $SIZE, $m1, $m2);
  }
  print "{$mm[0][0]} {$mm[2][3]}
{$mm[3][2]} {$mm[4][4]}\n";
}

/****/
function nestedloop($n) {
  $x = 0;
  for ($a=0; $a<$n; $a++)
  for ($b=0; $b<$n; $b++)
  for ($c=0; $c<$n; $c++)
  for ($d=0; $d<$n; $d++)
  for ($e=0; $e<$n; $e++)
  for ($f=0; $f<$n; $f++)
  $x++;
  print "$x\n";
}

/****/
function sieve($n) {
  $count = 0;
  while ($n-- > 0) {
    $count = 0;
    $flags = range (0,8192);
    for ($i=2; $i<8193; $i++) {
      if ($flags[$i] > 0) {
        for ($k=$i+$i; $k <= 8192;
$k+=$i) {
          $flags[$k] = 0;
        }
        $count++;
      }
    }
  }
  print "Count: $count\n";
}

/****/
function strcat($n) {
  $str = "";
```

```
  while ($n-- > 0) {
    $str .= "hello\n";
  }
  $len = strlen($str);
  print "$len\n";
}

/*****/
function getmicrotime() {
  $t = gettimeofday();
  return ($t['sec'] + $t['usec'] /
100000);
}
function start_test() {
  ob_start();
  return getmicrotime();
}

function end_test($start, $name) {
  global $total;
  $end = getmicrotime();
  ob_end_clean();
  $total += $end-$start;
  $num = number_format($end-$start,3);
  $pad = str_repeat(" ", 24-
strlen($name)-strlen($num));

  echo $name.$pad.$num."\n";
  ob_start();
  return getmicrotime();
}

function total() {
  global $total;
  $pad = str_repeat("-", 24);
  echo $pad."\n";
  $num = number_format($total,3);
  $pad = str_repeat(" ", 24-
strlen("Total")-strlen($num));
  echo "Total".$pad.$num."\n";
}
```

```
$t0 = $t = start_test();
simple();
$t = end_test($t, "simple");
simplecall();
$t = end_test($t, "simplecall");
simpleucall();
$t = end_test($t, "simpleucall");
simpleudcall();
$t = end_test($t, "simpleudcall");
mandel();
$t = end_test($t, "mandel");
mandel2();
$t = end_test($t, "mandel2");
ackermann(7);
$t = end_test($t, "ackermann(7)");
ary(50000);
$t = end_test($t, "ary(50000)");
ary2(50000);
$t = end_test($t, "ary2(50000)");
ary3(2000);
$t = end_test($t, "ary3(2000)");
fibo(30);
$t = end_test($t, "fibo(30)");
hash1(50000);
$t = end_test($t, "hash1(50000)");
hash2(500);
$t = end_test($t, "hash2(500)");
heapsort(20000);
$t = end_test($t, "heapsort(20000)");
matrix(20);
$t = end_test($t, "matrix(20)");
nestedloop(12);
$t = end_test($t, "nestedloop(12)");
sieve(30);
$t = end_test($t, "sieve(30)");
strcat(200000);
$t = end_test($t, "strcat(200000)");
total($t0, "Total");
?>
```

3.2. Hasil *Benchmark*

Di dalam pengujian dibandingkan hasil *benchmark* antara *script* yang dijalankan pada *server apache2* dengan *script* yang dijalankan pada *HipHop for* PHP. Adapun hasil *benchmark* dengan menggunakan *Apache Benchmark* pada *server apache2* adalah sebagai berikut:

```
kiddo@kiddo-VirtualBox:~/dev/web$ ab -n 100 -c 5 http://localhost/bench.php
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking localhost (be patient).....done
Server Software:        Apache/2.2.16
Server Hostname:        localhost
Server Port:            80

Document Path:          /bench.php
Document Length:        500 bytes

Concurrency Level:      5
Time taken for tests:   447.187 seconds
```

```
Complete requests:      100
Failed requests:        0
Write errors:           0
Total transferred:      69100 bytes
HTML transferred:       50000 bytes
Requests per second:    0.22 [#/sec] (mean)
Time per request:       22359.375 [ms] (mean)
Time per request:       4471.875 [ms] (mean, across all concurrent requests)
Transfer rate:          0.15 [Kbytes/sec] received

Connection Times (ms)
min  mean[+/-sd] median   max
Connect:        0    1   2.6      0      24
Processing: 20504 22333 725.8  22181   24558
Waiting:     4554 5052 332.1   4988    6421
Total:      20504 22333 726.2  22181   24558

Percentage of the requests served within a certain time (ms)
  50%  22181
  66%  22488
  75%  22709
  80%  22868
  90%  23362
  95%  23808
  98%  24473
  99%  24558
 100%  24558 (longest request)
```

Dan hasil *benchmark* dengan *Apache Benchmark* pada *HipHop for* PHP adalah sebagai berikut:

```
kiddo@kiddo-VirtualBox:~/dev/web$ ab -n 100 -c 5 http://localhost:8080/bench.php
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking localhost (be patient).....done
Server Software:
Server Hostname:        localhost
Server Port:            8080

Document Path:          /bench.php
Document Length:        500 bytes

Concurrency Level:      5
Time taken for tests:   103.703 seconds
Complete requests:      100
Failed requests:        0
Write errors:           0
Total transferred:      60000 bytes
HTML transferred:       50000 bytes
Requests per second:    0.96 [#/sec] (mean)
Time per request:       5185.135 [ms] (mean)
Time per request:       1037.027 [ms] (mean, across all concurrent requests)
Transfer rate:          0.57 [Kbytes/sec] received

Connection Times (ms)
min  mean[+/-sd] median   max
Connect:        0    1   4.4      0      33
Processing:  4839 5173 185.3   5132    5763
Waiting:     4839 5157 181.4   5124    5747
Total:       4839 5174 187.5   5132    5767

Percentage of the requests served within a certain time (ms)
  50%   5132
  66%   5219
```

```
  75%   5277
  80%   5289
  90%   5479
  95%   5653
  98%   5708
  99%   5767
 100%   5767 (longest request)
```

Dari hasil pengujian, dapat dilihat beberapa informasi penting yang dapat dijadikan sebagai komponen perbandingan, yaitu lama eksekusi (*Time taken for tests*), lama per *request* (*Time for request*), *request* terlama (*Longest request*), dan *transfer rate.* Dari komponen tersebut didapatkan hasil perbandingan seperti Tabel 1 berikut ini.

Tabel 1 Hasil *Benchmark Server Apache2* dan *HipHop for* PHP

	Lama Eksekusi (s)	**Lama Per *Request* (ms)**	***Request* Terlama (ms)**	***Transfer Rate* (kb/s)**
Apache2	447.187	4471.875	24.558	0.15
HipHop for PHP	103.703	1037.027	5.767	0.57
Perbedaan	343.484	3434.848	18.791	0.42
Perubahan	- 76.81 %	- 76.81 %	- 76.52 %	+ 280 %

Dari tabel hasil *benchmark* di atas dapat dilihat beberapa informasi mengenai optimasi yang dilakukan oleh *HipHop for* PHP dibandingkan dengan *server apache2* tanpa *HipHop for* PHP. Dilihat dari segi lama eksekusi, terjadi pengurangan waktu sebesar 76.81 %, yaitu dari 447.187 detik menjadi 103.703 detik. Artinya dari segi lama eksekusi, *HipHop for* PHP mampu meningkatkan performansi *web server* secara optimal. Begitu pula pada lama per *request* terjadi pengurangan waktu sebesar 76.81%.

Tabel hasil *benchmark* juga menunjukkan *request* terlama yang diperoleh *HipHop for* PHP hanya 5.767 ms, dimana 18.791 ms lebih cepat dibandingkan dengan *web server* tanpa *HipHop for* PHP (peningkatan sebesar 76.52%). Dari segi *transfer rate*, terlihat peningkatan yang sangat signifikan, yaitu sebesar 280%, dari 0.15 kb/s menjadi 0.57 kb/s. Jadi, *HipHop for* PHP mampu meningkatkan *transfer rate* sebesar 0.42 kb/s. Peningkatan tersebut menunjukkan pengaruh *HipHop for* PHP yang sangat besar dalam optimasi performansi *web server*.

4. Penutup

Dari hasil pembahasan di atas, dapat diambil kesimpulan *HipHop for* PHP mampu meningkatkan kemampuan *web server* untuk *website* PHP hingga lebih dari 50%, seperti yang disebutkan oleh *Haiping Zhao* melalui *website developer facebook* [3]. Meskipun disebutkan bahwa *HipHop for* PHP mengorbankan beberapa fungsi PHP yang jarang digunakan, namun *HipHop for* PHP mampu memberikan peningkatan performansi *web server* yang pantas untuk pengorbanan tersebut.

Referensi

[1] Kadir, A., 2008, *Dasar Pemrograman Web Dinamis Menggunakan PHP*, Edisi Revisi, Penerbit Andi, Yogyakarta.

[2] Github, *facebook/hiphop-php*, *https://github.com/facebook/hiphop-php/wiki/*, 8 Februari 2011.

[3] Zhao, H., *HipHop for PHP: Move Fast*, *http://developers.facebook.com/blog/post/358/*, 8 Februari 2011.