如何优化PHP程序

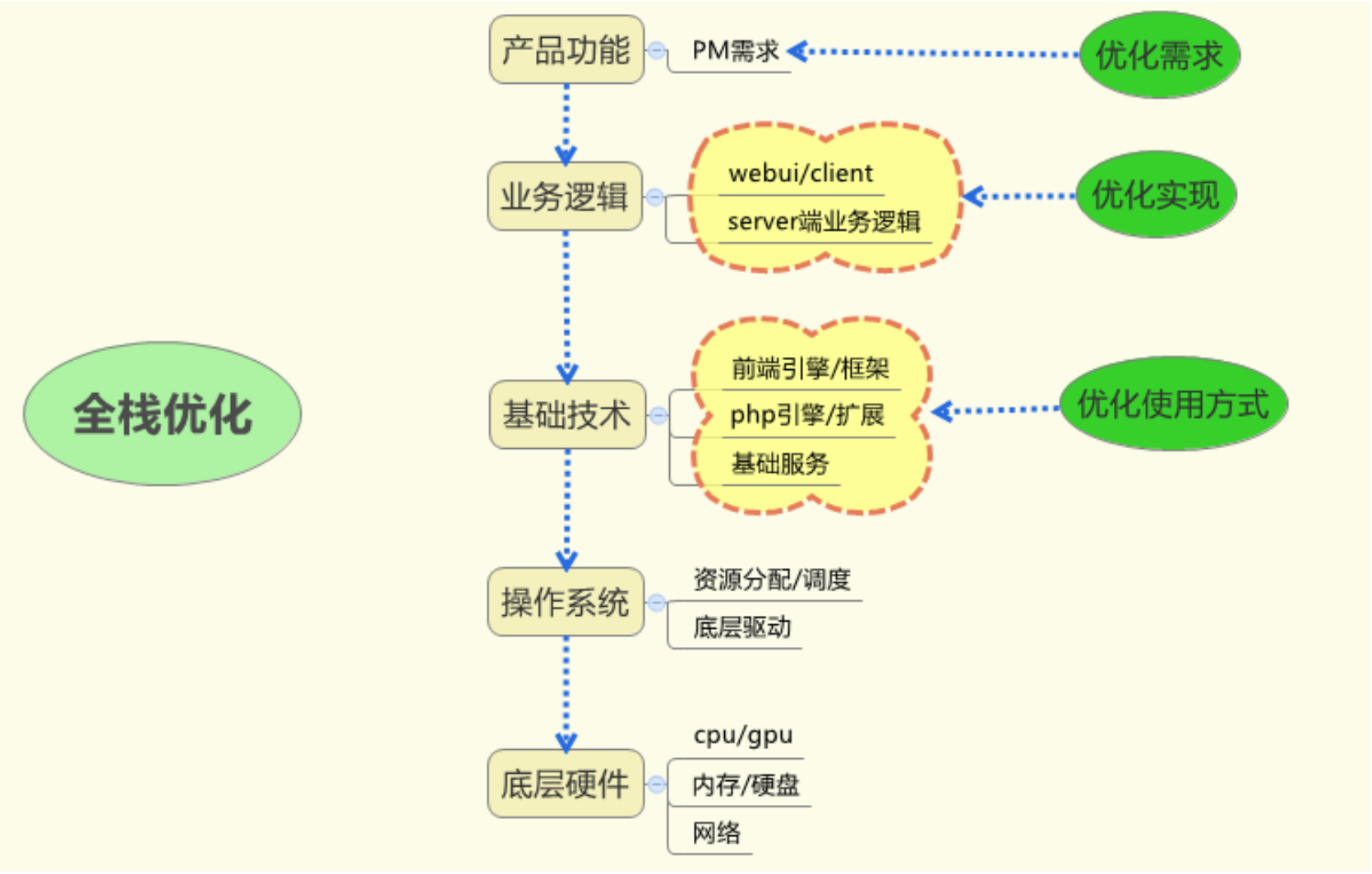
Weibing Wang

2014-12

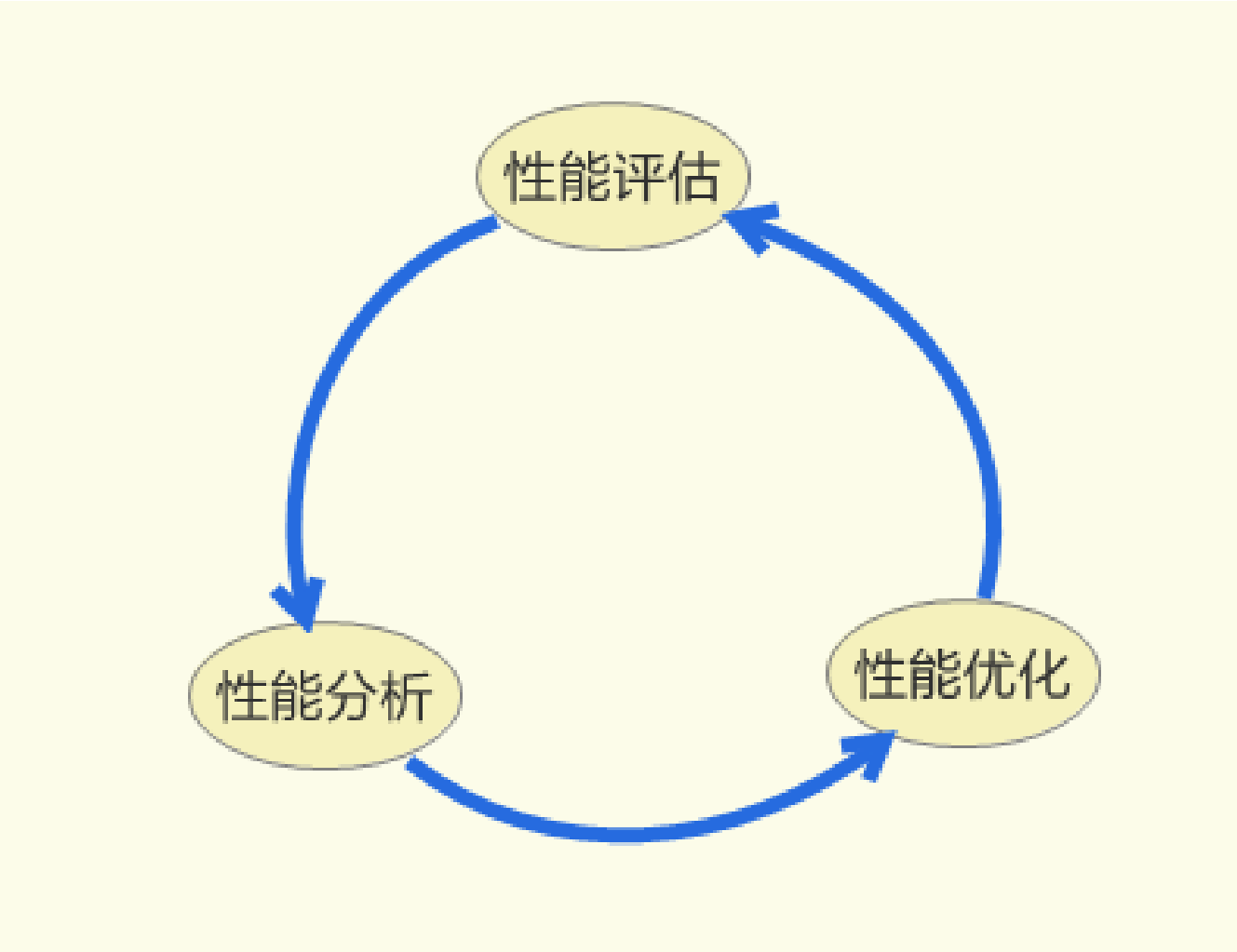
提纲

- 概述
- 性能评估与分析
- 性能优化方法

性能优化是全栈的


全栈优化
产品功能
PM需求
优化需求
业务逻辑
webui/client
server端业务逻辑
优化实现
基础技术
前端引擎/框架
php引擎/扩展
基础服务
优化使用方式
操作系统
资源分配/调度
底层驱动
底层硬件
cpu/gpu
内存/硬盘
网络

性能优化三步曲



性能评估与分析

- 性能评估
  - 评估指标
  - 评估方法
- 性能分析
  - 整体分析
  - 函数级分析

性能评估指标

- 主要指标
  - QPS
  - 响应时间
- Why?
  - 提升QPS -> 节省机器 -> 节省成本
  - 减少响应时间 -> 提升用户体验 -> 增加收入

评估方法

- QPS
  - 压力测试工具，如ab
- 响应时间
  - 计时，如gettimeofday
- 结果不稳定，怎么办？
  - 多次测量取平均值，算标准差
  - 正式测试前先发一些请求预热
  - 排除不稳定因素，如调用后端服务改成mock数据

整体分析-QPS

- QPS瓶颈来源：

qps = min(

min(client并发数,server并发数) * 1s/平均响应时间,

cpu核数*1s/单位请求cpu时间

)

- 因此，提升QPS方法：
  - 如果是响应时间瓶颈，则应提升并发数或减少平均响应时间
  - 如果是cpu瓶颈，则应减少cpu时间

整体分析-耗时

- cpu时间构成
  - 系统调用开销
  - 用户态代码执行
- 响应时间构成
  - cpu时间
  - 文件io
  - 网络io
  - 线程同步、sleep
  - 其它会block线程的操作

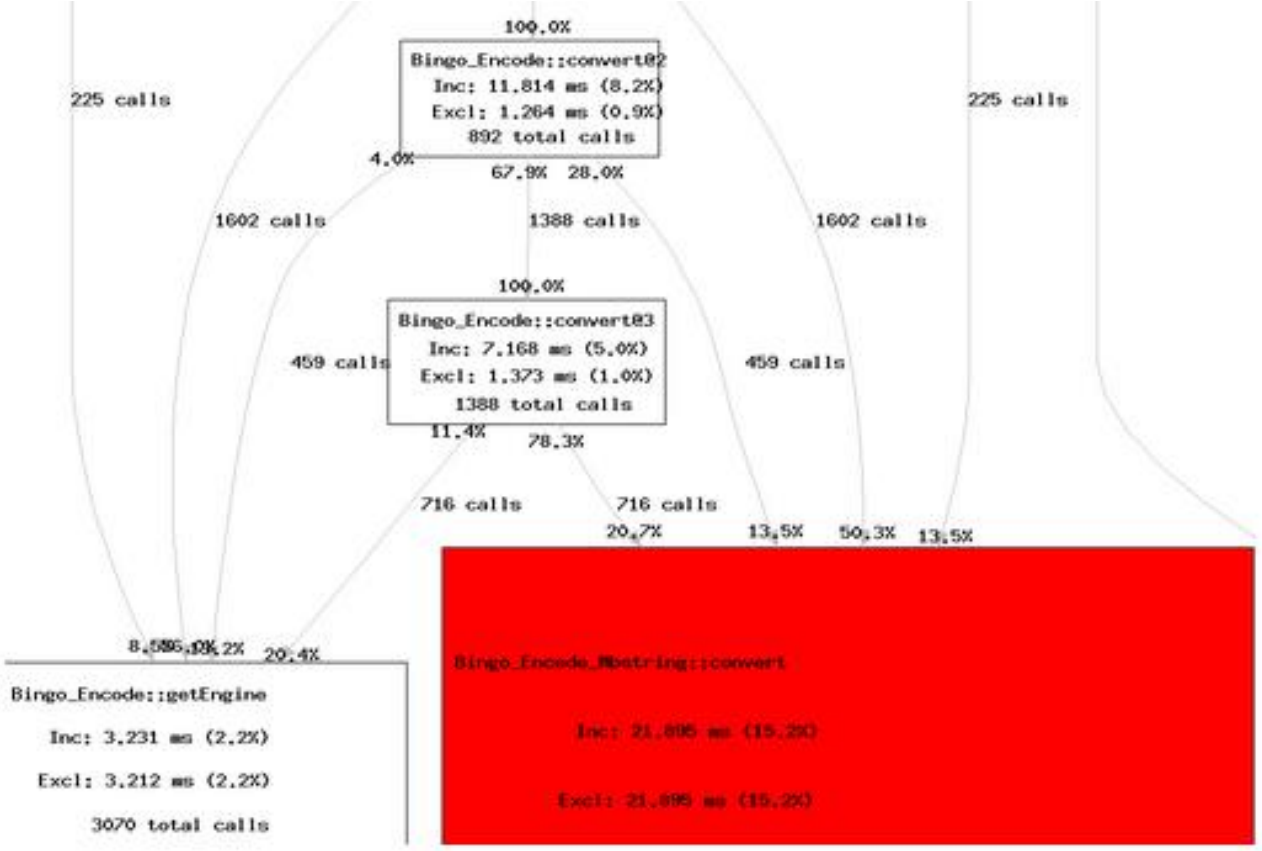
函数级分析

- Xhprof-函数耗时top列表

Function Name	Calls	Calls%	Incl. Wall Time (microsec)	IWall%	Excl. Wall Time (microsec)	EWall%
Bingo_Encode_Mbstring::convert	3,070	6.6%	21,540	14.8%	21,540	14.8%
TbView_ConfiguratorLoader::load	371	0.8%	10,791	7.4%	8,573	5.9%
TbView_Common_PageConfigurator::_checkContainValue	559	1.2%	7,925	5.5%	7,925	5.5%
Bingo_Log_Net::connect	2	0.0%	5,800	4.0%	5,721	3.9%
array_merge	792	1.7%	5,873	4.0%	5,592	3.8%
Bingo_String::xssSafe	5,240	11.2%	8,287	5.7%	5,437	3.7%
PhizView::includeOnce	372	0.8%	7,789	5.4%	4,035	2.8%
Bingo_Encode::getEngine	3,070	6.6%	3,262	2.2%	3,243	2.2%
Bingo_String::escapeHtml	2,765	5.9%	3,230	2.2%	3,230	2.2%
Bingo_View::assign	20	0.0%	11,005	7.6%	3,210	2.2%
Bingo_Encode::convert@1	2,376	5.1%	27,061	18.6%	2,745	1.9%
FrsThread::setIcons	51	0.1%	2,646	1.8%	2,298	1.6%

函数级分析

- Xhprof-函数调用图



性能优化方法

- 架构级优化——数量级提升
- 代码级优化——常数级提升

架构级优化

- 架构级优化
  - 引擎选型
  - 存储选型/索引处理
  - 加cache
  - 并行化
  - ……

PHP引擎选型

- 典型实现
  - PHP 5.x，PHPNG (PHP 7)，HHVM，HippyVM，……

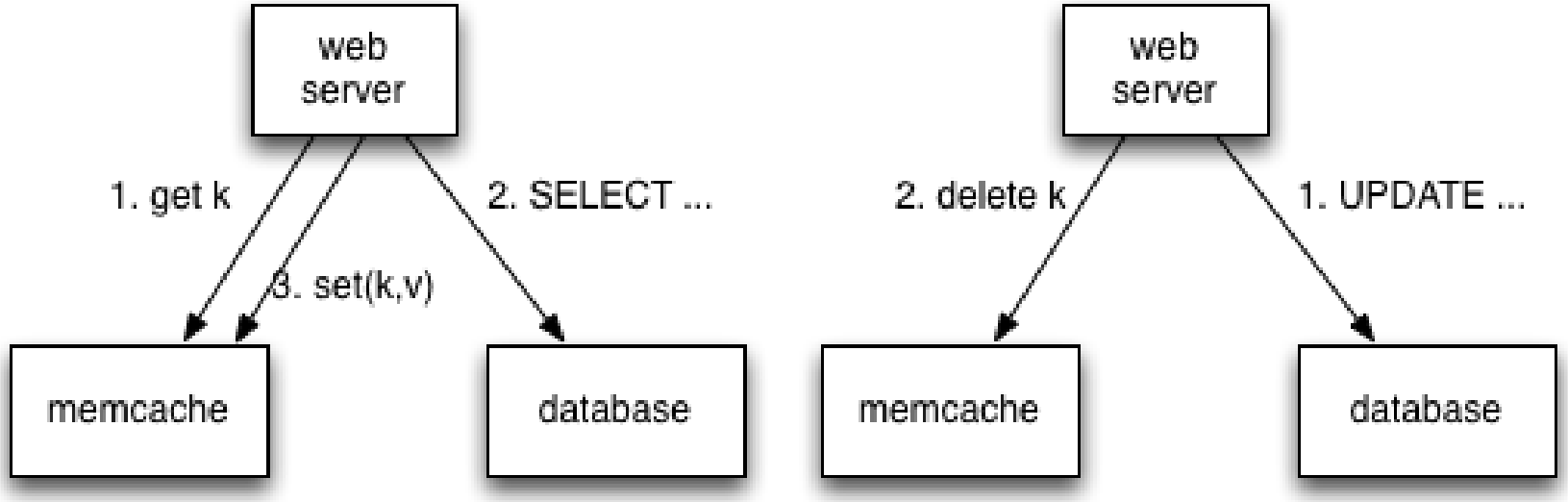
引擎	bench.php耗时	micro_bench.php耗时
php5.2	6.696s	41.914s
php5.5	3.408s	14.887s
phpng	2.361s	12.292s
hhvm2.2	0.465s	4.305s
hhvm3.0	0.569s	4.140s
hippyvm	1.412s	0.860s

存储选型/索引处理

- 典型实现
  - Mysql，Redis，Mongo，……
- 不作深入讨论

加Cache

- 分布式缓存
  - Memcached，Redis，……
- 本地缓存
  - HHVM Apc，PHP Yac/Eacc/Apc



并行化

- 多线程
  - HHVM Xbox/Pagelet Server，PHP Pthread
- 协程
  - PHP Swoole
- 异步回调
  - PHP Yar，PHP Eio/Ev/Libevent，HHVM async/await，mysqli_poll

代码级优化

- 代码级优化
  - 减少重复计算
  - 合理使用扩展
  - 后端调用优化
  - 日志及异常处理
  - 引擎特定优化
  - ……

减少重复计算

```
foreach ($arr as $v) {
    func1(func2($x), $v);
}
```

改成

```
$y = func2($x);
foreach ($arr as $v) {
    func1($y, $v);
}
```

合理使用扩展

- 扩展函数>自己定义的函数
- str系列函数>正则系列>自己遍历字符串
- array系列函数>自己遍历数组
- isset>in_array

后端调用优化

- 批量请求>逐个请求
- 并行请求>串行请求
- 本地调用>http rpc

日志及异常处理

- 不要在线上打印DEBUG和TRACE级别的日志
- 关闭E_NOTICE级别的日志
- 消除正常情况下所有的PHP Warning和Exception
- 尽量不要使用set_error_handler自定义错误处理
- 尽量不要使用@来抑制错误

HHVM特定优化

- 全局代码放入函数里
- 避免使用PHP动态特性
- 用类代替关联数组

全局代码放入函数里

```php
$s = 0;
for ($i = 0; $i < 100000; $i ++) {
    $s += $i;
}
var_dump($s);
```

改成

```php
function f() {
    $s = 0;
    for ($i = 0; $i < 100000; $i ++) {
        $s += $i;
    }
    var_dump($s);
}

f();
```

避免使用PHP动态特性

```
function f($arr, $file, $code, $name) {
    $a = include($file);
    $b = eval($code);
    $c = get_defined_vars();
    $d = $$name;
    $e = compact($name);
    $f = extract($arr);
    $g = create_function($code);
}
```

用类代替关联数组

```
function f($arr) {
    $arr['key1'] = g();
    h($arr['key2']);
}
```

改成

```
class A {
    public $key1;
    public $key2;
}

function f(A $a) {
    $a->key1 = g();
    h($a->key2);
}
```

参考资料

- 性能工具
  - http://httpd.apache.org/docs/2.2/programs/ab.html
  - http://php.net/xhprof
- PHP引擎
  - http://hhvm.com/
  - https://wiki.php.net/phpng
  - https://github.com/hippyvm/hippyvm
- 缓存
  - http://memcached.org/
  - http://redis.io/
  - http://docs.hhvm.com/manual/en/book.apc.php
  - https://github.com/laruence/yac
  - http://eaccelerator.net/
- 并行化
  - https://github.com/facebook/hhvm/blob/master/hphp/doc/threading
  - http://cn2.php.net/manual/zh/refs.fileprocess.process.php
  - http://www.swoole.com/
  - https://github.com/laruence/yar
  - http://cn2.php.net/mysqli_poll