弹窗后台压力测试报告

# 测试环境

## 服务器

* 内存: 16G
* Cpu: 8核, Intel(R) Xeon(R) CPU E3-1270 V2 @ 3.50GHz

## 软件

* Centos linux 6.4
* Nginx 1.2.8.6
* Php 5.4.17
* Redis 2.6.14
* Mysql 5.5.33
* Webbench1.5

# 详情表

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | 服务器设置 | | | 压力参数 | |  |  | 结果 |  |
| 分组 | 标记 | 逻辑 | php-fpm进程数 | nginx进程数 | 并发数 | 持续时间(分钟) | 成功返回数 | 每分钟成功返回数 | cpu使用率 | 内存使用量 |
| A | A1 | 请求一个有效的弹窗数据 | 80 | 8 | 10000 | 1 | 217114 | 217114 |  | 1.5G左右 |
| A2 | 请求一个有效的弹窗数据 | 80 | 16 | 10000 | 1 | 194010 | 194010 |  | 1.5G左右 |
| A3 | 请求一个有效的弹窗数据 | 128 | 16 | 10000 | 1 | 170072 | 170072 |  | 1.5G左右 |
| A4 | 请求一个有效的弹窗数据 | 128 | 16 | 20000 | 1 | 170008 | 170008 |  | 1.5G左右 |
| A5 | 请求一个有效的弹窗数据 | 128 | 8 | 20000 | 1 | 169097 | 169097 |  | 1.5G左右 |
| A6 | 请求一个有效的弹窗数据 | 64 | 8 | 10000 | 1 | 226413 | 226413 |  | 1.5G左右 |
| A7 | 请求一个有效的弹窗数据 | 48 | 8 | 10000 | 1 | 230194 | 230194 |  | 1.5G左右 |
| A8 | 请求一个有效的弹窗数据 | 32 | 8 | 10000 | 1 | 206457 | 206457 |  | 1.5G左右 |
| B | B1 | 请求一个有效的弹窗数据 | 48 | 8 | 10000 | 5 | 1167016 | 233403 | 70% | 1.5G左右 |
| B2 | 请求一个有效的弹窗数据 | 64 | 8 | 10000 | 5 | 1158712 | 231742 | 70% | 1.5G左右 |
| B3 | 请求一个有效的弹窗数据 | 80 | 8 | 10000 | 5 | 1137364 | 227473 | 70% | 1.5G左右 |
| B4 | 请求一个有效的弹窗数据 | 80 | 16 | 10000 | 5 | 1156697 | 231339 | 70% | 1.5G左右 |
| B5 | 请求一个有效的弹窗数据 | 128 | 16 | 10000 | 5 | 850083 | 170017 | 50% | 1.5G左右 |
| B6 | 请求一个有效的弹窗数据 | 56 | 16 | 10000 | 5 | 1158366 | 231673 | 70% | 1.5G左右 |
| B7 | 请求一个有效的弹窗数据 | 56 | 8 | 10000 | 5 | 1153353 | 230671 | 70% | 1.5G左右 |
| C | C1 | 进入thinkphp的action后立即结束 | 128 | 16 | 10000 | 1 | 280000 | 280000 |  | 1.5G左右 |
| C2 | 进入thinkphp的action后立即结束 | 56 | 8 | 10000 | 1 | 397327 | 397327 |  | 1.5G左右 |
| D | D1 | 访问一个html页面 | 56 | 8 | 10000 | 1 | 2141704 | 2141704 | 33% | 1.5G左右 |
| D2 | 进入一个独立的php文档，输出一个字符 | 56 | 8 | 10000 | 1 | 1998165 | 1998165 | 35% | 1.5G左右 |
| D3 | 访问一个不存在的php脚本 | 56 | 8 | 10000 | 1 | 2138920 | 2138920 | 33% | 1.5G左右 |
| E | E1 | 拼接redis并取出少量数据 | 56 | 8 | 10000 | 1 | 1171605 | 1171605 | 16% | 1.5G左右 |
| E2 | 拼接redis并取出少量数据 | 56 | 8 | 10000 | 5 | 5410768 | 1082154 | 16% | 1.5G左右 |
| F | F1 | 拼接redis并取出少量数据 | 256 | 128 | 10000 | 1 | 1192016 | 1192016 | 41% | 1.5G左右 |
| F2 | 进入thinkphp的action后立即结束 | 256 | 128 | 10000 | 1 | 399554 | 399554 |  | 1.5G左右 |
| F3 | 请求一个有效的弹窗数据 | 256 | 128 | 10000 | 1 | 169133 | 169133 |  | 1.5G左右 |

# 结论

## 服务的进程数量

由A组和F组数据可知:

* **php-pfm和nginx的进程数不是越多越好,开太多进程并不能带来性能的提升.**
* **nginx的进程数对性能的整体影响不大,进程数可以每cpu上1个或两个进程就足够了.**

由B组数据可知:

* **针对当前项目php-fpm进程数在64左右性能最好.**

## 性能的极限数值(并发为10000-20000时)

**极限**:是一个参考标准,通过它可以发现程序的瓶颈,不是目标.程序优化做得越好,结果会越接近这个值,但不一定能能达到这个值.

极限1: 由D1,D3可知:

* Nginx的极限为每分钟210W次访问

极限2: 由D2可知

* Nginx+Php-fpm的极限为每分钟近200W次访问

极限3: 由E组数据和F1可知:

* Nginx+php-fpm+redis的极限为每分钟110W次访问

极限4: 由C组数据和F2可知

* Nginx+php-fpm+thinkphp的极限为每分钟近40W次访问.

## 进一步优化方案

由B组数据可知,当前弹窗程序能承受的压力为每分钟访问23W次左右.

当前程序使用的软件包括linux+php-fpm+nginx+redis等.由极限3,和极限4可知,制约性能最大的因素是thinkphp.

如果需应对比较大的压力,首先得从thinkphp入手可,选择的解决方案为:

1. **增加服务器,增强负载能力.**
2. **减少对thinkphp的依赖.替代方案有:**
   1. **yaf框架,它只能提供简单的mvc驱动,编写业务逻辑需要花费更多一些时间.**
   2. **自己写一个简单的框架专门用来应对访问量较大的页面,访问量不大或者业务逻辑复杂的页面仍然可以用thinkphp.**
   3. **把yaf和thinkphp整合一下,保留thinkphp的功能和yaf的性能.**

以上两个解决方案都会带来成本的增加,具体选择哪个需要针对具体项目需求进行评估.

关于memcache和redis : 这两个都是K-V数据库,但它们的结构和api有很大的不同,具体如何选择要看业务逻辑决定.但性能上没有太大的差别.

## Redis使用情况

共压入测试用户2195731个,占用内存488M

这些用户都是新用户,无弹窗历史记录.加上弹窗记录,数量量要大一些,大概是现在的1.3倍.

如果按600W用户计算488M\*3倍\*1.3 = 1903M

现在服务器内存使用率比较低,可以给redis开4G的内存

如果redis内存使用超过4G,则需要增加定时清理机制来节省内存,提高效率.