便携式高性能 WNMP 架构的研制

曾 棕 根
(宁波职业技术学院电子信息工程学院 浙江 宁波 315800)

摘 要 由发现每个 PHP Fast-CGI 工作进程最多执行 500 个 Requests 的特征,提出使用 xxfpm 对 PHP Fast-CGI 进行进程维持,使 WNMP 架构能提供稳定的 Web 服务;对 WNMP 架构整体进行了优化。压力测试表明,PHP 运行时间较之前平均下降 19.5%,WNMP 架构并发平均提升 512.5%。研制成功了一款便携、稳定、高性能的 WNMP 架构。

关键词 WNMP PHP 高性能 便携式 服务器

中图分类号 TP393.09 **文献标识码** A DOI:10.3969/j.issn.1000-386x.2018.04.030

DEVELOPMENT OF PORTABLE HIGH PERFORMANCE WNMP ARCHITECTURE

Zeng Zonggen
(*School of Electronic and Information Engineering, Ningbo Polytechnic, Ningbo 315800, Zhejiang, China*)

Abstract This paper found the features that each PHP Fast-CGI work process executed up to 500 Requests. Maintaining PHP Fast-CGI with xxfpm enabled the WNMP architecture to provide a robust Web service. Finally, the overall structure of WNMP was optimized. Stress tests showed that PHP running time dropped an average of 19.5% over the previous run. WNMP architecture concurrently increased by 512.5%. We developed a portable, stable, high-performance WNMP architecture.

Keywords WNMP PHP High-performance Portable Server

0 引 言

过去,在 Windows 操作系统上运行 PHP 网站通常采用 WAMP 架构,即 Windows + Apache + MySQL + PHP 黄金组合[1]。但该架构存在两个缺点,一个是性能较低,只能用于开发环境,无法成为 Windows 操作系统上的生产环境;另一个就是迁移困难,无法做到便捷地从开发机上部署到生产服务器上。

为了解决上述两个难题,本文研制了一款全新的 WNMP 架构,实现了 Windows 操作系统下 PHP 运行环境的便携式、高并发与高性能。

1 WNMP 架构简介

WNMP 架构是指由 Windows 操作系统、Nginx 服务器、MariaDB 服务器和 PHP 模块组成的一套完全免费开源的 PHP 网站运行架构。

Nginx 是一款新型的、高性能的 HTTP 和反向代理服务器,其特点是占有内存少、并发能力强[2]。

MariaDB 是 MySQL 的一个分支,主要由开源社区在维护,采用 GPL 授权许可,它完全兼容 MySQL。

PHP 是一种通用开源脚本语言,具有面向对象特征,拥有丰富的开发框架,执行效率高,是当前开发动态网站的首选语言之一。在 2014 年的 Qcon 分享中有一个数据,全球排名前 100 万的网站中,81.3% 使用的 Web 服务端脚本语言是 PHP[3]。

WNMP 架构与前端用户交互方式:用户从客户端浏览器发起 PHP 页面请求;再由 Nginx 转发给 PHP 去编译、运行;PHP 到 MariaDB 数据库中存取数据,最后以 HTML 标记生成结果页面并转交给 Nginx 服务器;Nginx 服务器再将结果页面返回给客户端浏览器;客户端浏览器则将结果页面解释出可视化结果给用户浏览。WNMP 服务器间交互如图 1 所示。

收稿日期:2017 - 07 - 21。宁波职业技术学院 2016 年科研课题(NZ16013)。**曾棕根**,副教授,主研领域:软件工程。

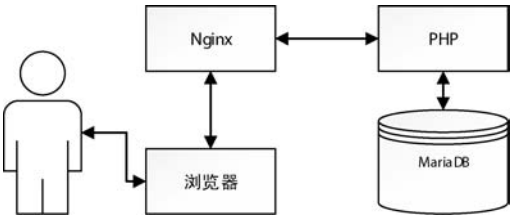

图 1　WNMP 架构 B/S 交互示意图

2　WNMP 架构的组织

为了使 WNMP 架构具备免安装和便携性的特征,把 Nginx、PHP、MariaDB 都放置在 C 盘的 Win32_nmp/Win64_nmp 文件夹下,里面包含这几个文件夹:nginx、php、mysql、tmp、xxfpm 和 startwnmp. bat、stopwnmp. bat 两个批处理文件。

其中, startwnmp. bat 负责启动 nginx、mysql 和 xxfpm 三项服务,而 stopwnmp. bat 则负责一次性停止这三项服务。tmp 文件夹则用来保存 WNMP 架构运行过程中的全部临时数据。这种文件组织方式使 WNMP 架构运行过程中产生的所有数据都保留在 Win32_nmp/Win64_nmp 文件夹中,实现了免安装和随时将此架构通过冷拷贝 Win32_nmp 或 Win64_nmp 文件夹的方式将整个架构及网站数据随时进行完整迁移。

Nginx、PHP、MariaDB 程序是从官方网站上下载编译好的二进制压缩包,直接解压缩后,放在 Win32_nmp/Win64_nmp 文件夹相应位置即可,下载位置及版本:

Nginx:从 http://nginx. org/下载 nginx - 1. 12. 0. zip 包;

PHP:从 http://php. net/下载 32 位包 php-7. 0. 21-nts-Win32-VC14-x86. zip 或 64 位包 php-7. 0. 21-nts-Win32-VC14-x64. zip;

需要注意的是,nginx 二进制码不区分 Windows 32/64 位;PHP 组件则需下载非线程安全版本。

3　WNMP 架构的驱动

WNMP 架构由 Nginx 服务器、xxfpm 服务器和 MariaDB 服务器组成,通过各自固定的端口进行通信,具体过程如下:

默认在 80 号端口工作的 Nginx 监控进程在接到来自客户端浏览器的 php 页面请求后,将此请求转发给一个 Nginx 工作进程。再由此工作进程转发给默认工作在 9000 号端口的 xxfpm 这个 PHP Fast-CGI 进程管理器。再由它交给一个 PHP Fast-CGI 工作进程,PHP Fast-CGI 工作进程将此 php 页面编译、运行,并通过 mysqli 或 PDO-mysql 应用层 AIP 与 libmysql 或 mysqlnd 底层 API 去与默认工作在 3306 端口的 MariaDB 数据库服务器进行数据存取。最后 PHP 将程序运行结果以 HTML 格式交给 Nginx 服务器,再返回给客户端浏览器进行可视化解释,使终端用户看到 php 网页的运行效果。WNMP 架构驱动模型如图 2 所示。

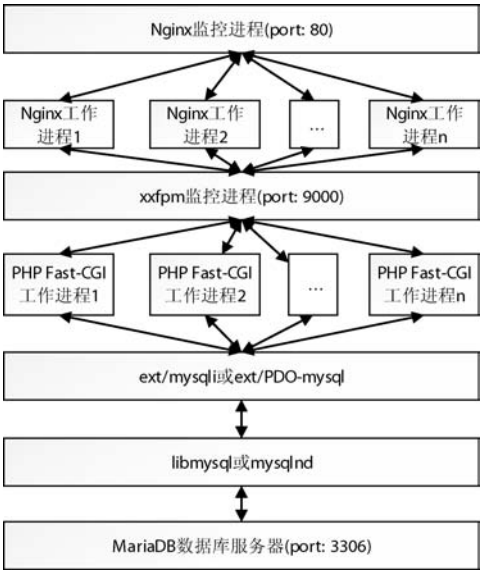

图 2　WNMP 架构驱动示意图

而 xxfpm 监控进程则通过下面这条启动参数来一次性启动 a 个 PHP Fast-CGI 进程,并运行在 9000 端口上:【xxfpm. exe " php - cgi. exe - c php. ini" - n a - i 127. 0. 0. 1 - p 9000】

xxfpm 有两个功能,一个是将 Nginx 传递过来的 php 网页请求转发给 PHP Fast-CGI 工作进程去执行;另一个是监控 PHP Fast-CGI 工作进程的数量,一旦发现 Windows 系统中 PHP Fast-CGI 工作进程的数量少于最初启动的 a 个,它会立即补充一个新的 PHP Fast-CGI 工作进程,实际上是对 PHP Fast-CGI 进行进程维持。因为 PHP Fast-CGI 的设计原理是,每个工作进程默认的生命周期是执行完 500 个 requests 网页请求后便自杀,并回收内存。如果 xxfpm 不对 PHP Fast-CGI 进行进程维持,那么 WNMP 架构在执行完【a × 500】个 requests 网页请求后,所有的 PHP Fast-CGI 都将自杀殆尽。PHP 这样的设计目的是为了避免某个 PHP Fast-CGI 工作进程开销内存过高而使操作系统内存耗尽。通过利用 Sublime 编辑器进行全文扫描,发现了 PHP 源码 php - 7. 0. 21 - src. zip 包的\sapi\cgi\cgi_main. c 的 1727 行和\sapi\fpm\fpm\fpm_main. c 的 1585 行都定义了最大请求执行量为 500:

1727: int max_requests = 500;

1585: int max_requests = 500;

从而确定了 WNMP 架构运行一段时间就必需重启才能让网站正常访问的原因。虽然 PHP 自身设计了 php-fpm 服务来进行维持 PHP Fast-CGI 进程,但它是为 Linux 操作系统设计的,在 Windows 上并不适用,所以,WNMP 架构包必须要采用自己编写的 xxfpm 服务进行维持,它调用了 pthreadGC2. dll,并使用 CreateProcess 创建进程,使用 Wait For Single Object 等待进程结束[4]。只有 xxfpm 停止时,它所创建的所有的 PHP Fast-CGI 子进程才能被全部关闭。但当它所创建的某个 PHP Fast-CGI 进程关闭后,xxfpm 会立即再创建一个,从而实现进程维持,保证了 WNMP 架构的稳定性。

4 WNMP 架构性能优化

对于单服务器上的 WNMP 架构的性能提升,主要从六个层面着手:提高计算能力、减少 I/O 延时、提高算法效率、充分发挥 WNMP 并行计算特征、缩短 PHP 编译时间和提高 PHP 与 MariaDB 的存取性能。

(1) 提高计算能力。CPU 越多计算能力越强,WNMP 架构也越快。另外,内存至少在 16 GB 以上,才更好发挥 WNMP 架构的并行计算性能。

(2) 减少 I/O 延时。应当尽可以利用缓存(内存)代替直接磁盘访问。在 MariaDB 配置文件 my. ini 中,提高 MariaDB 数据库 InnoDB 存储引擎的缓存,将缓存设置为总内存大小的 50% ~80 % 能起到很好的效果[5],下面是 32 GB 内存下的优化实例:

innodb_buffer_pool_size = 16G

(3) 提高算法效率。优化 SQL 慢查询、优化 PHP 代码算法,有效减少 CPU 计算次数,能大大提高 WNMP 架构响应性能,从而缩短用户等待时间。

(4) 充分发挥 WNMP 并行计算特征。要合理设置 Nginx 和 PHP Fast-CGI 并行工作进程的数量,才能充分发挥服务器的计算性能。Nginx 工作进程的数量"一般等于 CPU 的总核数或总核数的两倍,例如两个四核 CPU,则总核数为 8"[6]。

而 Nginx 的配置文件 nginx. conf 中【worker_processes n;】的参数 n 表示要启动多少个 Nginx 工作进程,n 设置应当设置为服务器的 CPU 总核数或其两倍即可。例如,该服务器 CPU 总核心是 8 核,那么,可设置【worker_processers 8;】或【worker_processers 16;】。

假定每个静态 PHP Fast-CGI 最多开销 30 MB 内存,那么,一次性启动的静态 PHP Fast-CGI 的工作进程数量一般设置为【内存总 MB ÷ 30 MB】,一般取 2^n 个,如 16、32、64、128,但最多不要超过 128 个。这里的内存总量应当要先减去 Windows 操作系统开销的 4 GB内存,再减去数据库开销的缓存。WNMP 架构在 startwnmp. bat 中指定了启动 PHP Fast-CGI 的数量:

【xxfpm. exe " php - cgi. exe - c php. ini" - n a - i 127. 0. 0. 1 - p 9000】

【- n a】中的 a 就是指 PHP Fast-CGI 的启动数量。

(5) 缩短 PHP 编译时间。应当开启 OPcache 缓存功能,使 PHP 运行效率得到大幅提高,其工作原理是:OPcache 通过将 PHP 脚本预编译的字节码存储到共享内存中来提升 PHP 的性能,存储预编译字节码的好处就是省去了每次加载和解析 PHP 脚本的开销[7]。直接在 php. ini 中开启 opcache:

zend_extension = "C:\win64_nmp\
php\ext\php_opcache. dll"
opcache. enable = 1

(6) 提高 PHP 与 MariaDB 的存取性能。应当使用 mysqlnd 底层 API 代替 libmysql。PHP 要与 MariaDB 数据库服务器通信,以前采用 Oracle 公司开发的 libmysql 这个底层 API 库,该库有版权, PHP 默认不启用这个库,另一方面 libmysql 与 Zend 引擎完全独立,无法利用 Zend 引擎的很多特性。为了解决这个问题,Zend 公司开发了 mysqlnd 这个本地驱动库来替代 libmysql 库。一方面解决了版权问题;另一方面,mysqlnd 与 Zend 引擎高度集成,因此提供更多高级特性,如内存管理受 Zend 引擎管理,使得内存消耗更少;有效利用 Zend 进行加速,使得 PHP 执行速度更快。最后,使用 mysqlnd 驱动后,无需在服务器上必须先编译安装 MariaDB 然后才能编译安装 PHP 了,有利于分布式 PHP 服务器的安装和部署。mysqlnd 与 libmysql 工作原理[8]如图 3 所示。

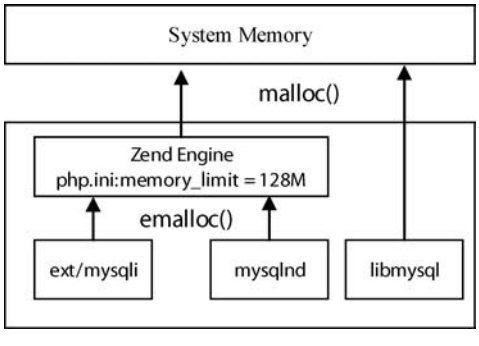

图 3 mysqlnd 与 libmysql 工作原理

使用 libmysql 驱动, php 的编译参数:

--with-mysqli = /usr/local/mysql/bin/mysql_config
--with-pdo-mysql = /usr/local/mysql

而使用 mysqlnd 驱动, php 的编译参数改为:

--with-mysqli = mysqlnd
--with-pdo-mysql = mysqlnd

5　WNMP 架构稳定性与性能测试

研制成功 win32_nmp 和 win64_nmp 两个版本的 WNMP 架构后,在同一台服务器上对传统的 WAMP 和按本文优化后的 WNMP 架构进行了 PHP 计算性能评估和 WNMP 架构并发性能评估。

测试服务器硬件配置:CPU 为 Intel® Xeon® CPU E3-1230 v3 @3.3 GHz,1 块 CPU 共 4 核心;内存为 32 GB,DDR 3,1 600 MHz,ECC。

WAMP 架构测试服务器软件配置:OS 为 Windows 7 64 位旗舰版;Apache 为 1 个工作进程;PHP 为 1 个工作进程;MariaDB 为 innodb 缓存 16 GB。

WNMP 架构测试服务器软件配置:OS 为 Windows 7 64 位旗舰版;Nginx 为 8 个工作进程;PHP 为 128 个工作进程;MariaDB 为 innodb 缓存 16 GB。

(1) PHP 计算性能评估　本测试只运行纯 CPU 任务脚本的基准测试(不需要 I/O 操作的任务,例如访问文件,网络或数据库连接)。使用的基准测试脚本 bench.php 和 micro_bench.php 来源于 php-7.0.21.tar.gz 的 Zend 目录中。如图 4 所示。

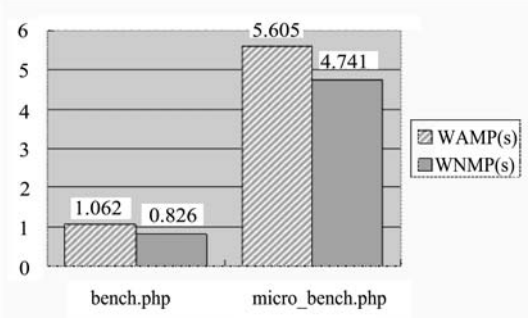

图 4　PHP 性能评估对比图

从测试结果看, bench.php 和 micro_bench.php 脚本在 WNMP 架构上较在 WAMP 架构上耗时平均下降了 19.5%。

(2) WNMP 架构并发性能评估　本测试采用 Apache的 ab 测试,模拟 100 个用户同时向 WAMP 和 WNMP 架构发起 PHP 连接请求,共完成 10 000 个连接请求。

ab 测试指令:ab - c 100 - n 10000 http://服务器 IP/php 页面。

请求的 PHP 页面说明:

insert.php:向 MariaDB 插入一条记录;

select.php:从 MariaDB 数据表中查询一条记录。

图 5 为并发性能评估对比图。

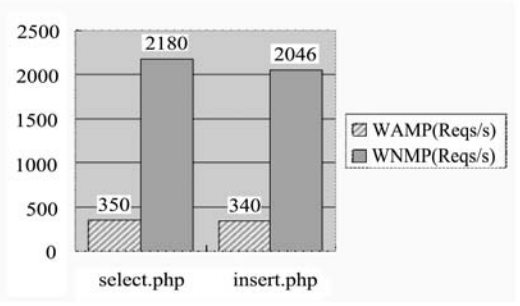

图 5　WAMP/WNMP 架构并发性能评估对比图

上述测试结果表明,在 WNMP 架构上,通过 PHP 存取 MariaDB 数据库性能平均提升了 512.5%。

6　结　语

本文详细介绍了 WNMP 架构的文件组织方式、服务器间通信方式和性能优化方法,并对 PHP 性能和 WNMP 架构采用典型方法进行了压力的测试。结果表明,这款 WNMP 架构较传统的 WAMP 架构具有便携、稳定、高并发的特点,性能接近 LNMP 架构。该 WNMP 架构不仅可作为 PHP 网站的开发环境,也能作为 Windows 服务器操作系统上的 PHP 网站生产环境。下一步将研制出一款更高性能的 LNMP 架构。本文研制的 WNMP 架构可在 QQ 群 263569269 的群共享中下载。

参　考　文　献

[1] 曾棕根. LAMP(PHP)程序设计[M]. 北京:北京大学出版社,2012:1-2.

[2] Clément Nedulcu. 学习 Nginx HTTP Server[M]. 北京:清华大学出版社,2012:218-219.

[3] 徐汉彬. PHP7 和 HHVM 的性能之争[EB/OL]. http://www.aichengxu.com/php/1879060.htm.

[4] ChinaZ 源码报导. Windows 下的 Nginx 和 php 搭配 php-cgi.exe 自动关闭退出的完美解决方法[EB/OL]. http://down.chinaz.com/server/201111/1334_1.htm.

[5] 简朝阳. MySQL 性能调优与架构优化[M]. 北京:电子工业出版社,2009:212-213.

[6] 张宴. 实战 Nginx:取代 Apache 的高性能 Web 服务器[M]. 北京:电子工业出版社,2010:23-24.

[7] php 官网. OPcache 简介[EB/OL]. http://php.net/manual/zh/intro.opcache.php.

[8] Yeho. PHP 5.3 以上版本推荐使用 mysqlnd 驱动[EB/OL]. https://blog.linuxeye.cn/395.html.