# PHP升级导致系统负载过高问题分析

*2014-10-15* 基础架构快报[1]

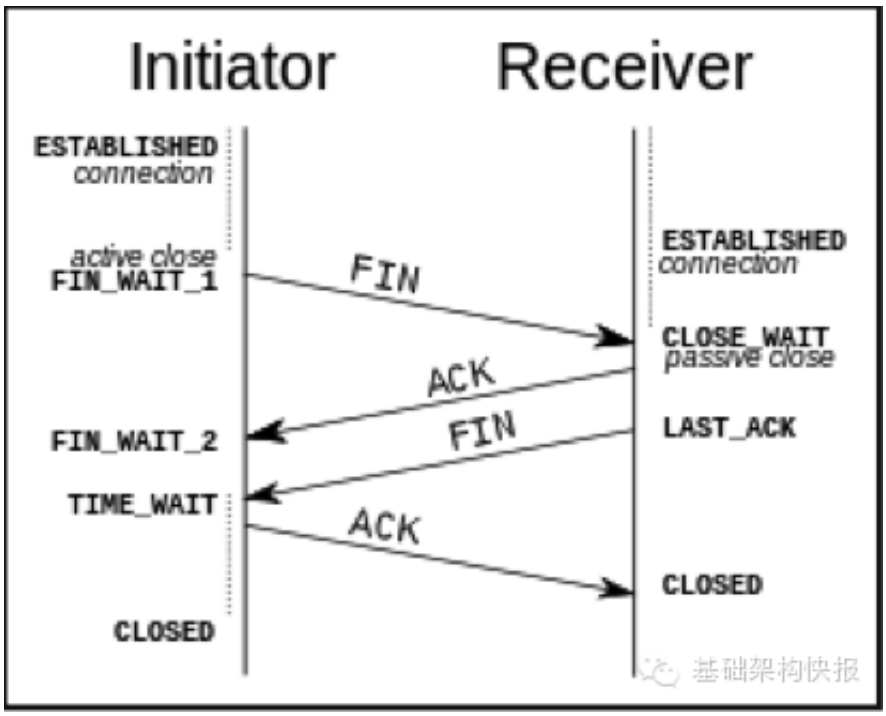
大家在上次的技术分享会中（PPT内容请回复 "4"）提出了若干疑问，真的很赞！做技术嘛就是要刨根问底，由于一些是延伸出的扩展问题，所以鉴于时间限制当时并未细聊，本篇将展开讨论各个疑点，是文章 "4" 中的案例完整版

---

据XX部门兄弟反应, PHP5.3.8 5.5.13 , 开始运行正常, 运行一段时间后, 系统负载变高,200%, netstat看到大量连接处在CLOSE_WAIT 最终导致系统不可用, PHP 5.3. 8 , 一切正常。

php-fpm 配置文件除了由版本引起的差异外， 没做任何改变。

TCP关闭连接的流程图：



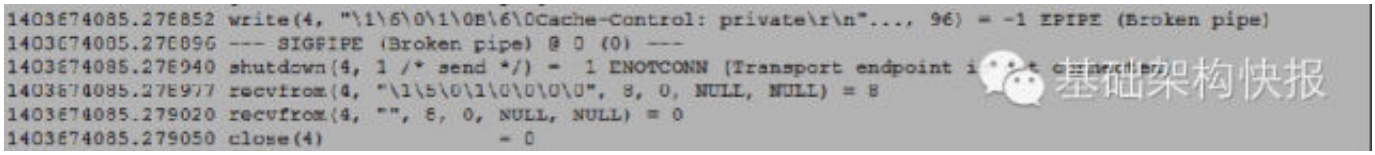

可以看到， 一个处于连通状态（ESTABLISHED）的连接， FIN数据包（对方调用closeTCP 连接的状态由ESTABLISHEDCLOSE_WAIT，等待应用调用closeFIN给对方）。

因此我怀疑PHP由于某种原因被堵塞住（比如数据库等资源太慢），导致请求超时，nginx closePHP 程序由于一直堵塞，导致其无法调用close，造成大量TCPCLOSE_WAIT

由于当时没有故障现场， 因此我们挑选一台机器，将PHP5.5重新上线，等待故障现象重现，我计划问题重现时， strace 看下进程的系统调用，找出PHP进程到底堵塞在哪里。

1. Strace

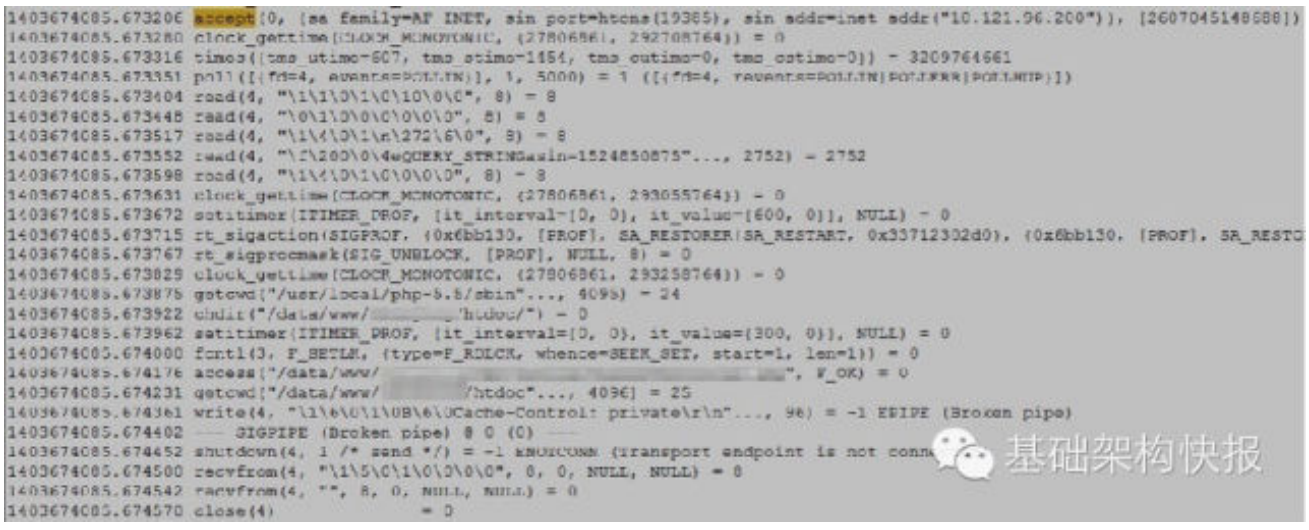
PHP5.5.13 周三大约上午930左右，系统负载飙升至200% strace 查看，结果没有发现任何堵塞的情况，却发现另一个异常现象：write函数调用失败




PHPwrite返回响应时，结果报错：BrokenpipeTCP连接其实已经关闭。PHP并没有堵塞住，这跟我猜想的完全不同，一时想不明白为什么PHPwrite 一个关闭了的连接，于是我先用sar 将当前的系统状态保持起来，包括磁盘，内存，网卡，CPU，中断，上下文切换等，留待以后慢慢分析。

sar 发现内存，CPU基本没有什么大的变化， 网卡流量明显降低，上下文切换（cswch/s）明显升高。网卡流量降低可以理解，因为当前系统已不能正常返回响应，但上下文切换（cswch/s）升高却不知道什么原因。

sar的结果暂时没有什么思路， straceBrokenpipe说明连接早已经被对方关闭，nginxaccept连接到关闭连接的整个流程：

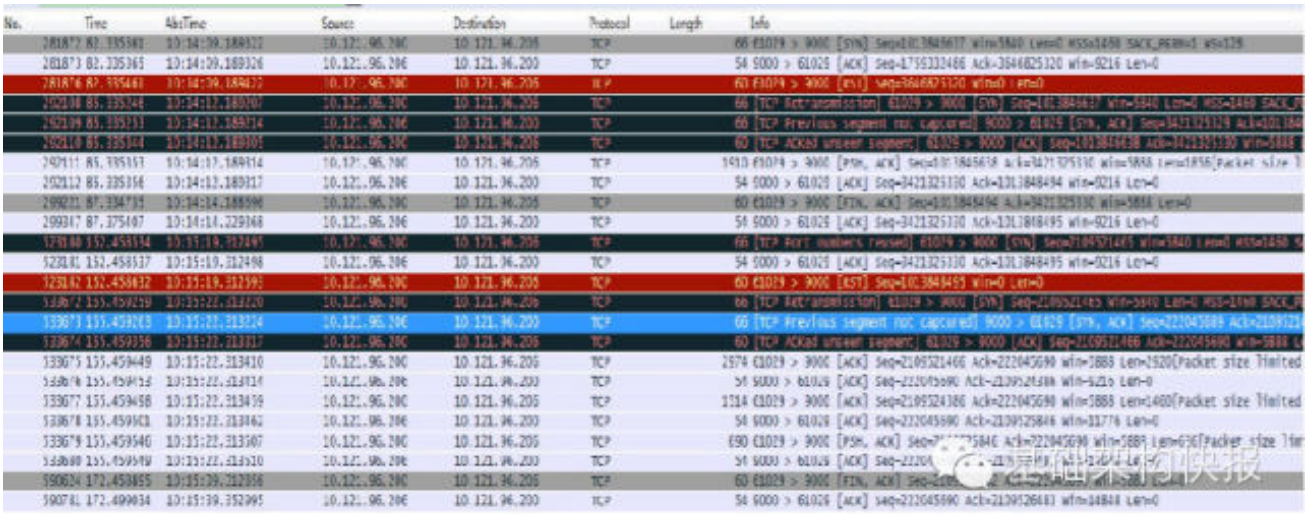



accept socket read 读取数据一直正常， write当返回响应时，却报错

Brokenpipeacceptwrite 一共花费大约1ms 这段时间内nginx肯定是不会超时的！那为什么连接会关闭呢？tcpdump抓包看下网络到底发生了什么。

1. Tcpdump

tcpdump 抓包，由于数据量太大， 我只选择了一台Nginx IP10.121.96.200抓包并传回到本地用wiresharkwireshark便于分析），发现网络已经一团糟：



tcp.port == 61029 的过滤结果，10.121.96.200NGINX 10.121.96.206 后面为了方便直接用NGINX来代称这两台机器。

1 从图上可以看到， NGINXPHPSYN TCP创建连接三次握手的第一次， PHPACK PHPSYN+ACK 才是正确的，acknumber 3646825320 NGINX SYN的序列号对应不起来，因此接下来NGINX RST直接重置了连接。

2 先不管异常1 我们继续往下看， 4个数据包，过了3snginx SYNPHPSYN+ACK 序列号也能对应上，NGINX ACK，三次握手完成，连接成功建立。然后NGINXPHPHTTP请求，到了第9 NGINXFIN 关闭连接，而PHP除了返回对应的ACK外，没有返回任何东西：没有返回响应数据（len=0也没有返回FIN来关闭连接。从AbsTime字段可以看到第8,9包之间相隔2s因此合理推测NINGX2s主动关闭了连接， PHP什么也没做，此时PHP机器上连接一定处于CLOSE_WAIT

3 继续看下面的数据包， 11个数据包，又是一个SYN 这应该是一个新的连接，从时间上看， 距离上一个包1 NGINX reuse）了这个端口（61029 但是从异常2可以知道，PHP上的连接还处于CLOSE_WAIT SYN后，只是返回了ACKacknumber=1013848495，说明这个ACK确实是上一个连接的；同异常1 NGINXRST重置了连接。

从抓包结果可以得出结论：PHP 上大量连接一直处于CLOSE_WAIT NGINX复用端口新建连接， PHP由于还处于CLOSE_WAIT 直接返回了ACK NGINX 期待返回的是SYN+ACK 由于不符合预期NGINXRST重置连接， 连接重置后PHPCLOSE_WAIT就消失

了。3sNGINXSYN 成功建立连接，发送请求，超时后close PHP什么也没做，又变为CLOSE_WAIT 一段时间后NGINX 复用端口新建连接……如此循环往复。

那么问题是， PHP机器是怎么到达这种循环往复的状态呢？ 总有成因吧？到目前为止， 我们已经知道PHP 机器当前的状态， 从抓包来看， PHPCLOSE_WAIT 并且不响应NGINX strace结果来看， PHP没有堵塞，也在处理响应， 但是在返回响应的时候，连接却早已经被NGINX 关闭。这两种结果明显矛盾！

没有任何其他思路， 走了很多弯路， 首先怀疑是网卡问题， ifconfig/ethtool看网卡状态， dropped errors 都是正常范围，又怀疑是TCP bugSSH登录却没有任何问题，SSH也很正常，TCP这么容易出BUG也不太可能，因此确定一定不是TCP 及底层的问题，问题仍然出在PHP

我突然想到， strace tcpdump的结果对应起来， strace 中选取一个（IPPORT 查看其对应tcpdump 于是同时运行strace tcpdumpstrace NGINX IPPORT

1403674216.729752 accept(0, {sa_family=AF_INET, sin_port=htons(), sin_addr=inet_addr("10.121.96.200")}, [2688649527312]) = 4

tcpdump的结果过滤host10.121.96.200 and port 6470 :

13:27:57.942284 IP 10.121.96.200.6470 > 10.121.96.206.9000: S3965696332:3965696332(0) win 5840 <mss 1460,nop,nop,sackOK,nop,wscale 7>

1403674216.729752accept 调用时的时间戳， 转化为时间就是：WedJun 25 13:30:16 CST 2014tcpdump发起连接的时间却是13:27:57accept 调用比接收到SYN2分钟多。到这里我立刻明白连接被积压在队列里了，2 PHPaccept 从队列获取一个连接，而此时这个连接早已经因为超时被NGINXPHP 返回响应调用writeBroken pipe

连接队列的大小由backloglisten 系统调用的第二个参数），PHP5.5 65535PHP5.3.8 128我们都是用的默认值，因此这两个值是不同的。由于PHP5.5 65535这个值比较大，造成很多连接积压在队列里， 由于队列处理的比较慢， 导致连接超时， 超时的连接由于还没有accept 因此不会处理请求，也不会被close CLOSE_WAIT状态，这就是我们从tcpdump中看到的现象。 accept获取到一个连接， 实际是获取到一个CLOSE_WAIT状态的连接， write调用向连接写数据时，自然会报错。 这就同时完美解释了tcpdumpstrace看似矛盾的结果。

这里有必要解释下， TCP连接为什么会积压在队列里， 要理解这个问题， 需要先理解linux TCP 三次握手的一些具体实现。

我们知道， server端，监听一个端口， socket,bind listen

int listen(intsockfd, int backlog);

listen的第二个参数叫做backlog 用来设置连接队列的大小。实际Linux 维护两个队列，一个是在接收到SYN后，此时没有完成三次握手， 处于半连接状态，存放到SYNqueue 另一个是三次握手完成后， 成功建立连接，存放到acceptqueue，等待应用调用accept 来消费队列。这里的backlog就是用来设置accept queue（旧版内核用来设置SYN queueman listen）的大小。

TCP 传输跟系统调用实际是一个异步的过程， 系统通过队列来保存最新的TCP状态或数据。也就是说，TCP三次握手由内核来完成， 跟应用层是否调用accept 内核将完成三次握手的socket acceptqueue中，应用调用accept accept queue中获取连接。那么，backlog 而我又不及时调用accept 来消费队列，则连接就被积压到accept queue

在三次握手完成后， 客户端就可以发送数据了， 数据由内核接收， TCP buffer 而此时应用（PHP）可能还没有调用accept

如果积压在accept queue中的连接如果已经被对方close 应用仍然可以accept到这个连接， 也可以调用read buffer中的数据， 但是，当调用write Broken pipe

Backlog过高的猜想，可以解释当前的故障现象， 但仍然有很多疑问需要解决

1. accept queue 是如果慢慢积压的？流量突增还是后端变慢？
2. 连接积压后， PHP获取到的都是close掉的连接，不需要返回响应，应该会快速失败，那么消费accept queue的速度应该大大提高，close掉的连接都快速消费掉，系统应该会恢复正常，但现实情况下负载却持续很高，并且没有恢复，
3. Nginx 做反向代理时， 可以自动摘掉失效的后端， 后端摘掉后，不再有新的请求路由过来，从而accept queue得以消费，慢慢恢复正常，nginx nginx 根本就没摘掉后端，
4. 上下文切换cswch/s 为什么突升？

1. 1 acceptqueue 如何慢慢积压，暂时我还没有确凿的证据，不过可以通过日志和配置做一些推想。首先从PHP slow log PHP进程偶尔会出现fastcgi_finish_request 执行慢的情况， 2s accept queue 必然会有增长， PHP-FPM max_request=102400也就是处理102400 个请求后， fpmworker进程会自动退出， fpm重新派生（respawnworker进程，如果负载比较均衡的话，所有子进程应该几乎同时退出(pmstatic)PHPPHP没有对这个机制做任何优化，在这个过程中 accept queue也必然会积压。

6.2 PHP 进程为什么没有因为快速失败而快速消费队列呢， strace 可以找到答案， 我

们看下， PHPwrite brokenpipe到底又做了什么。通过对比正常php进程跟异常phpstrace我发现异常PHPlogflock耗时比较多：

1403674085.279482 flock(4, LOCK_EX) = 0

1403674085.668528 write(4, "1\t1\t1403674085\xx\t11"..., 76) =76

1403674085.668565flock(4, LOCK_UN) = 0

1403674085.668– 1403674085.279400ms PHP

1403680571.144737flock(4, LOCK_EX) = 0 <0.000014>

1403680571.144784write(4, "1\t1\t1403680571\xx\t11"..., 74) = 74 <0.000014>

1403680571.144833flock(4, LOCK_UN) = 0<0.000017>

几乎没有耗费时间！ 因此我们可以想到， 当大多进程都快速失败时， 他们会同时去写日志， 由于我们的日志程序在写日志前都调用flock PHP进程由于争夺锁而堵塞， 因此丧失了快速消费acceptqueue

flock PHP 其实一般情况下， append方式打开文件写日志时，是不需要加锁的， 因为其本身为原子操作，具体参考：http://cn2.php.net/manual/en/function.fwrite.php

If handle was fopen()ed in appendmode, **fwrite()**s are atomic(unless the size of string exceeds the filesystem's block size, onsome platforms, and as long as the file is on a local filesystem). That is,there is no need to flock() a resourcebefore calling **fwrite()**;all of the data will be written without interruption.

6.3 Nginx 为什么没有摘掉后端?

nginx 的配置文件， 我发现目前有两个vhost 其中一个是 A.cn fastcgifastcgi_read_timeout100ms可见这个时间是很容易造成超时的。vhost 一系列域名，我这里就用B.cn来代替了，其中没有配置fastcgi 相关的超时。会采用默认值60s

upstream 的配置是这样的：

server10.121.96.206:9000 max_fails=10 fail_timeout=5m;

10次连续失败， 则摘掉后端5 5分钟后再尝试恢复后端。从这里看到， nginx 是配置了

摘除后端的功能的。我统计了故障时段以及故障以前的请求分布，发现故障时段请求数并没有减少，nginx根本没有摘除后端。

我确认了相应版本nginx 没有发现摘除功能的任何bug 于是试图分析nginx access_log 发现大量请求的返回状态码为499499 nginx发现客户端主动断开了连接，nginxupstream的连接并记录日志，状态码记为499一般出现在客户端设置有超时的情况下。问题于是出现了，client 主动断开连接， nginx 认为是客户端放弃了请求，并不是后端失败，因此不会计入失败次数，也就不会摘掉后端了。

Vhost A.cn fastcgi_read_timeout 100ms而客户端的超时一般不会是毫秒级别的，vhost 如果超时不可能是客户端超时，fastcgi因此记录状态码应该为504499499 必定大多是由 vhost B.cn B.cn fastcgi_read_timeout 60s因此很可能导致客户端超时。

从请求分布的统计结果来看， 故障前后每分钟请求数基本没有变化， 那么说明不能摘掉后端的vhost B.cn的请求占比例应该很大，从而导致可以摘掉后端的vhost A.cn 就算摘除了后端对请求数影响也不大。为了验证这个推论， accesslog B.cn 85%的流量。因此推论正确。

Nginx 由于客户端超时，不认为是后端失败， 从而没有摘掉后端，丧失了故障恢复的机会。

6.4 上下文切换cswch/s 为什么突升?

这个问题， 我没有确切答案， 不过可以合理推论， 几百进程同事flock 当锁可用时，几百进程会同时唤醒，但只有一个进程能够得到锁，其他进程在分配到时间片后，由于无法获取锁，只能放弃时间片再次wait，从而导致上下文切换飙升，但CPU使用率却升高的不明显。当然这只是推论，大家如果有什么想法，欢迎一起讨论。(Update: 2014-07-04, 经试验，频繁flockcswch/s)

从以上分析看出， 问题虽然是由于backlog 128 65535 但问题实际涉及从客户端，nginxphp等多个方面，而这多个方面都有问题，综合导致了这次故障。

因此我建议多方面同时调整，彻底解决问题:

1. nginx 设置超时， 至少小于客户端的超时时间， 否则摘除机制形同虚设。
2. backlog 设置为合理值， 可以参考排队论， qps，以及处理时间实际可以确定队列大小， qps1000100ms 则队列大小设置为1000*0.1=100比较合适， 否则我明明知道等我处理到这个连接，其早就超时了， 还要将连接放到队列，肯定是不合理的。
3. 写日志时，去掉flockappend 方式打开文件， fwrite是原子操作（除非要写的字符串大于文件系统的blockblock4k应该不会超过吧？）。

PHP5.5.13listen.backlog12812周，观察故障是否会再次出现。预期：不出现

PHP5.3.8 listen.backlog 6553512 观察故障能否出现。预期：出现

PHP5.3.28listen.backlog12812周，观察故障能否出现。预期：不出现

之所以验证PHP 5.3.28是因为据业务反映，以前测试时，这个版本也有问题，但5.3.28 backlog128，如果存在问题，则上面的结论就完全错了。因此我要求重新验证一下。

**(2014-07-31)**

经过一个月的线上验证，PHP 5.5.13listen.backlog 128后，表现正常。PHP5.3.28也没有问题。这两个都符合预期，但listen.backlog 65535PHP5.3.8出乎意料，没有出现异常。

backlog积压的真正原因，我这边已经确定，是凌晨0点切割日志导致。切割日志脚本在mv,

SIGUSR1php-fpmmasterphp-fpmreopen日志文件，并通知所有worker进程退出。这一瞬间会造成一定的连接队列的积压，不过这不是主要原因，主要原因是随后脚本查找并删除2天前的日志文件，由于文件较多，估计应该几百Giowait偏高。导致连接队列的积压。最高积压可达到上千。

Tue Jul 29 00:00:50CST 2014 3: CE60790A:2328 00000000:0000 0A 00000000:000004FC00:00000000 00000000 0 0 4205691288 1 ffff8101b2c7e6c0 3000 0 0 2 -1

统计数据从/proc/net/tcp 并每秒打印一次， 00:00:50 连接队列的积压达到0x000004FC 10进制，也就是1276

另外，我采集了凌晨日志切割时负载的相关数据，listen.backlog 65535 PHP5.3.8 负载最高达100以上。连接积压最多达1000listen.backlog 128 PHP5.5.13, 由于限制了连接队列，其最大积压为129128+1），最高负载在70左右。负载之所以仍然很高，这是由删除日志导致。

系统负载居高不下，无法恢复，与多个因素有关，首先，删除日志，导致iowait过高，堵塞了PHP进程，从而无法及时消费连接队列，由于listen.backlog 65535时，连接几乎一直积压在连接队列，由于nginx100ms导致大量连接超时关闭。大量PHP进程由于无法返回响应，时间都集中在写日志方面，引起flock惊群。此时，就算iowait恢复正常，由于flocknginx 无法及时摘除后端，系统也无法恢复正常了。

从以上信息可以，各影响因素有一个发生改变，可能都不会导致系统负载居高不下的问题：

1. Nginx A.cn 100ms，如果时间够长，则连接队列的socketclose掉，就不会导致大量PHP进程集中写日志。
2. listen.backlogbacklog足够小，新的连接会直接拒掉，连接队列就不会积压太多连接。就算都被close掉，也不会有太多影响。
3. 写日志去掉flock，这样避免flockiowait恢复时，系统负载也就恢复了。
4. Nginx B.cn 配置合理的超时时间，这样在后端超时时，可以自动摘除有问题的后端，使后端有机会恢复负载。
5. iowait过高，这样就不会拖慢PHP，不会积压连接，也就不存在负载过高的问题了。

1. javascript:void(0);