今天上午，运营同事报告一个严重故障，现象是网站页面速度非常慢，基本不可用。工程师开始追查问题。  
首先介绍一下系统架构：前端 Apache，中间PHP，后端MySQL，经典的LAMP架构。

猜测数据库出现性能问题

第一反应，怀疑数据库数据量太大。我们一直定期清理数据库，保证单表数据量在一定范围内。而这段时间一直没有清理，数据量可能过大。立刻执行delete语句，单表减少100W条记录。但是，问题依旧。  
后来，DBA同学发现慢查询，存在filesort。果断增加索引，慢查询消失。但是，问题依旧。  
无奈之下，怀疑机房网络问题。但是其他产品线都没有问题，网络组也没有故障通报。因此，否决这种可能。

线下环境复现故障  
时间已经到了下午5点，决定重新梳理思路，反复观察故障现象。总结几点：  
页面响应速度不稳定，或慢或快，或稍慢，或奇慢，缺少一种规律性  
甚至，在一位同事的电脑上，刷新几次都很快。  
由于没有权限看线上日志（其实代码中也没有打任何日志），无法从日志中获取有效信息。为了方便调试，我们决定在线下环境部署代码并尝试复现故障。  
结果，线下环境中，故障基本必现。可以肯定，与网络环境没有关系，与线上的memcahced/mysql的运行状况也没有关系。  
我们在PHP代码的不同位置插入调试代码 echo ‘xxx’;exit; ， 试图确定何处代码在堵塞。  
我们惊奇的发现，当一个页面处于等待响应的状态时，后续页面必然也在等待。而且，后续页面没有执行任何PHP代码。我们推断，Aapache进程block住了！

Strace Apache进程  
为了搞清楚Apache进程block在什么地方，我们使用strace工具观察Apache进程的系统调用。  
先找出Apache的进程号。

[root@vm11030032 ~]# ps aux|grep httpd

root 3553 0.0 2.6 312672 13476 ? Ss Aug03 0:03 /usr/local/apache2/sbin/httpd -k restart

www 4759 0.0 4.4 320664 22584 ? S 20:31 0:00 /usr/local/apache2/sbin/httpd -k restart

www 4760 0.0 3.2 316548 16672 ? S 20:31 0:00 /usr/local/apache2/sbin/httpd -k restart

www 4761 0.0 3.2 316548 16672 ? S 20:31 0:00 /usr/local/apache2/sbin/httpd -k restart

www 4762 0.0 3.2 316548 16672 ? S 20:31 0:00 /usr/local/apache2/sbin/httpd -k restart

www 4763 0.1 4.9 319552 25036 ? S 20:31 0:00 /usr/local/apache2/sbin/httpd -k restart

www 4766 0.0 3.2 316548 16672 ? S 20:32 0:00 /usr/local/apache2/sbin/httpd -k restart

root 4890 0.0 0.1 61188 732 pts/3 R+ 20:35 0:00 grep httpd

www 30809 0.0 6.0 327304 30772 ? T 16:43 0:01 /usr/local/apache2/sbin/httpd -k restart

然后，随便挑一个进程号，strace上去：strace -p 4759  
接着，反复刷新页面，总有一个请求会落到PID为4759的进程上。  
终于，一个请求过来，快速刷屏，戛然而止，输出内容定格在：

connect(107, {sa\_family=AF\_INET, sin\_port=htons(7634), sin\_addr=inet\_addr("10.73.19.246")}, 16) = -1 EINPROGRESS (Operation now in progress)

poll([{fd=107, events=POLLOUT}], 1, 1000) = 1 ([{fd=107, revents=POLLOUT}])

connect(107, {sa\_family=AF\_INET, sin\_port=htons(7634), sin\_addr=inet\_addr("10.73.19.246")}, 16) = 0

write(107, "get APPS\_SCREEN\_API\_CURR\_TOKEN \r"..., 33) = 33

read(107, 0x14130e48, 8196) = -1 EAGAIN (Resource temporarily unavailable)

poll([{fd=107, events=POLLIN}], 1, 200) = 1 ([{fd=107, revents=POLLIN}])

read(107, "VALUE APPS\_SCREEN\_API\_CURR\_TOKEN"..., 8196) = 107

write(107, "quit\r\n", 6) = 6

read(107, 0x14130e48, 8196) = -1 EAGAIN (Resource temporarily unavailable)

poll([{fd=107, events=POLLIN}], 1, 200) = 1 ([{fd=107, revents=POLLIN}])

read(107, "", 8196) = 0

shutdown(107, 2 /\* send and receive \*/) = 0

close(107) = 0

open("/dev/random", O\_RDONLY) = 107

read(107,

注意，最后一行输出并不完整，说明Apache进程堵塞在read系统调用上。read的对象是 /dev/random，看起来与随机数有关。但是，哪里的代码会用到随机数呢？

找到关键代码  
根据前面输出的IP和PORT，包括调用参数，我们确定这是在访问memcached。于是，顺藤摸瓜，找到访问memcached之后的一段代码：

$size = mcrypt\_get\_iv\_size (MCRYPT\_BLOWFISH,MCRYPT\_MODE\_ECB);

$iv = mcrypt\_create\_iv($size, MCRYPT\_DEV\_RANDOM);

$m = mcrypt\_ecb (MCRYPT\_BLOWFISH,$key,$dmcryptText,MCRYPT\_DECRYPT,$iv);

其中，第二行代码，出现了RANDOM，查了一下php手册，当第二个参数为MCRYPT\_DEV\_RANDOM时，mcrypt\_create\_iv存在堵塞的可能性。MCRYPT\_DEV\_URANDOM则不会阻塞。

线上故障得以解决  
虽然还不知道具体原因，但是本着快速解决问题的原则，决定替换参数立刻上线。  
果然，问题得以解决，刷新页面时，从未如此酣畅淋漓！

分析故障现象  
线上的Apache进程很大的概率会走到上述代码，因此很可能被block一段时间。于是，当前页面就会很慢。  
当所有Apache进程都被block时，后续请求必须等待空闲的Apache进程，因此后续页面都将变得很慢。  
由于Apache进程unblock的时间不可确定，因此后续页面的等待时间也时长时短。

深挖原因  
虽然问题解决，但是本质原因还没搞清楚：为什么MCRYPT\_DEV\_RANDOM会堵塞，而MCRYPT\_DEV\_URANDOM从不会堵塞。  
google了一下/dev/random，维基百科一如既往的靠谱：  
写道

在这个实现中，发生器保存了来自熵池中噪声的数据位数的估计值，而随机数是从该熵池中创建的。

在读取时，/dev/random设备只会返回熵池中噪声数据中的随机字节。

/dev/random应当可以适用于要求非常高质量随机性的应用，例如产生公钥或一次性密码本。

若熵池空了，对/dev/random的读操作将会被阻塞，直到收集到了足够的环境噪声为止[3]。

这样的设计使得/dev/random是真正的随机数发生器，提供了最大可能的随机数据熵，建议用于产生保护高价值或长保护周期的密钥。

/dev/random的一个副本是/dev/urandom （"unlocked"，非阻塞的随机数发生器[4]），它会重用内部池中的数据以产生伪随机数据。

这表示对/dev/urandom的读取操作不会产生阻塞，但其输出的熵可能小于/dev/random的。

该设备文件是设计用于密码学安全的伪随机数发生器的，可以用于安全性较低的应用。

大概的意思就是，/dev/random生成随机数时，依赖熵池。如果熵池空了或不够用，对/dev/random的读取就会堵塞，直到熵池够用为止。/dev/urandom则不会堵塞。有得必有失，urandom的随机性弱于random。

详解熵池  
熵池本质上是若干字节。/proc/sys/kernel/random/entropy\_avail中存储了熵池现在的大小，/proc/sys/kernel/random/poolsize是熵池的最大容量，单位都是bit。如果entropy\_avail的值小于要产生的随机数bit数，那么/dev/random就会堵塞。  
那么，为什么熵池不够用呢？  
google了一下资料，熵池实际上是从各种noice source中获取数据，noice source可能是 键盘事件、鼠标事件、设备时钟中等。linux内核从2.4升级到2.6时，处于安全性的考虑，废弃了一些source。source减少了，熵池补给的速度当然也变慢，进而不够用。  
其实，通过消耗熵池，可以构造DOS攻击。原理很简单，熵池空了，依赖随机数的业务（SSL，加密等）就不能正常进行。

补充熵池  
Linux服务器在运行时，既没有键盘事件，也没有鼠标事件，如何快速积累熵池呢？  
google了一下资料，发现有一些程序可以自动补充熵池，例如rngd或rng-tools。  
我在Linode VPS上尝试了一下rngd，效果非常明显。  
先观察rngd启动前的熵池大小：

watch cat /proc/sys/kernel/random/entropy\_avail

在100~200之间。

然后启动rngd：

sudo rngd -r /dev/urandom -o /dev/random -f -t 1

熵池立刻飙升到3712，接近4096的上限。

总结  
先吐槽：没有日志的系统太扯淡了，追查问题只能靠推测或猜测，耽误大量时间。  
后总结：如果业务依赖随机数，那么最好使用工具主动补充熵池。  
再吐槽：熵池一直够用，今天才出现不够用的情况。究竟是神马原因，还搞不清楚。码农真苦！