mysql的"双1验证"指的是**innodb\_flush\_log\_at\_trx\_commit**和**sync\_binlog**两个参数设置，这两个是是控制MySQL 磁盘写入策略以及数据安全性的关键参数。下面从参数含义，性能，安全角度阐述两个参数为不同的值时对db 性能,数据的影响。

**一、参数意义**

innodb\_flush\_log\_at\_trx\_commit  
如果innodb\_flush\_log\_at\_trx\_commit设置为0：log buffer将每秒一次地写入log file中，并且log file的flush(刷到磁盘)操作同时进行.该模式下，在事务提交的时候，不会主动触发写入磁盘的操作;  
如果innodb\_flush\_log\_at\_trx\_commit设置为1：每次事务提交时MySQL都会把log buffer的数据写入log file，并且flush(刷到磁盘)中去;  
如果innodb\_flush\_log\_at\_trx\_commit设置为2：每次事务提交时MySQL都会把log buffer的数据写入log file，但是flush(刷到磁盘)操作并不会同时进行。该模式下,MySQL会每秒执行一次 flush(刷到磁盘)操作。

**注意：**由于进程调度策略问题,这个"每秒执行一次 flush(刷到磁盘)操作"并不是保证100%的"每秒"。

sync\_binlog  
sync\_binlog 的默认值是0，像操作系统刷其他文件的机制一样，MySQL不会同步到磁盘中去而是依赖操作系统来刷新binary log。  
当sync\_binlog =N (N>0) ，MySQL 在每写 N次 二进制日志binary log时，会使用fdatasync()函数将它的写二进制日志binary log同步到磁盘中去。

**注意：**如果启用了autocommit，那么每一个语句statement就会有一次写操作；否则每个事务对应一个写操作。

**二、性能**

两个参数在不同值时对db的纯写入的影响表现如下：  
测试场景1  
innodb\_flush\_log\_at\_trx\_commit=2  
sync\_binlog=1000

测试场景2  
innodb\_flush\_log\_at\_trx\_commit=1  
sync\_binlog=1000

测试场景3  
innodb\_flush\_log\_at\_trx\_commit=1  
sync\_binlog=1

测试场景4  
innodb\_flush\_log\_at\_trx\_commit=1  
sync\_binlog=1000

测试场景5  
innodb\_flush\_log\_at\_trx\_commit=2  
sync\_binlog=1000

在以上5个场景下的TPS分别为:  
场景1            41000  
场景2           33000  
场景3           26000  
场景4           33000

由此可见，当两个参数设置为双1的时候，写入性能最差，sync\_binlog=N (N>1 ) innodb\_flush\_log\_at\_trx\_commit=2 时，(在当前模式下)MySQL的写操作才能达到最高性能。

**三、安全**

当innodb\_flush\_log\_at\_trx\_commit和sync\_binlog 都为 1 时是最安全的，在mysqld 服务崩溃或者服务器主机crash的情况下，binary log 只有可能丢失最多一个语句或者一个事务。但是鱼与熊掌不可兼得，双11 会导致频繁的io操作，因此该模式也是最慢的一种方式。  
当innodb\_flush\_log\_at\_trx\_commit设置为0，mysqld进程的崩溃会导致上一秒钟所有事务数据的丢失。  
当innodb\_flush\_log\_at\_trx\_commit设置为2，只有在操作系统崩溃或者系统掉电的情况下，上一秒钟所有事务数据才可能丢失。

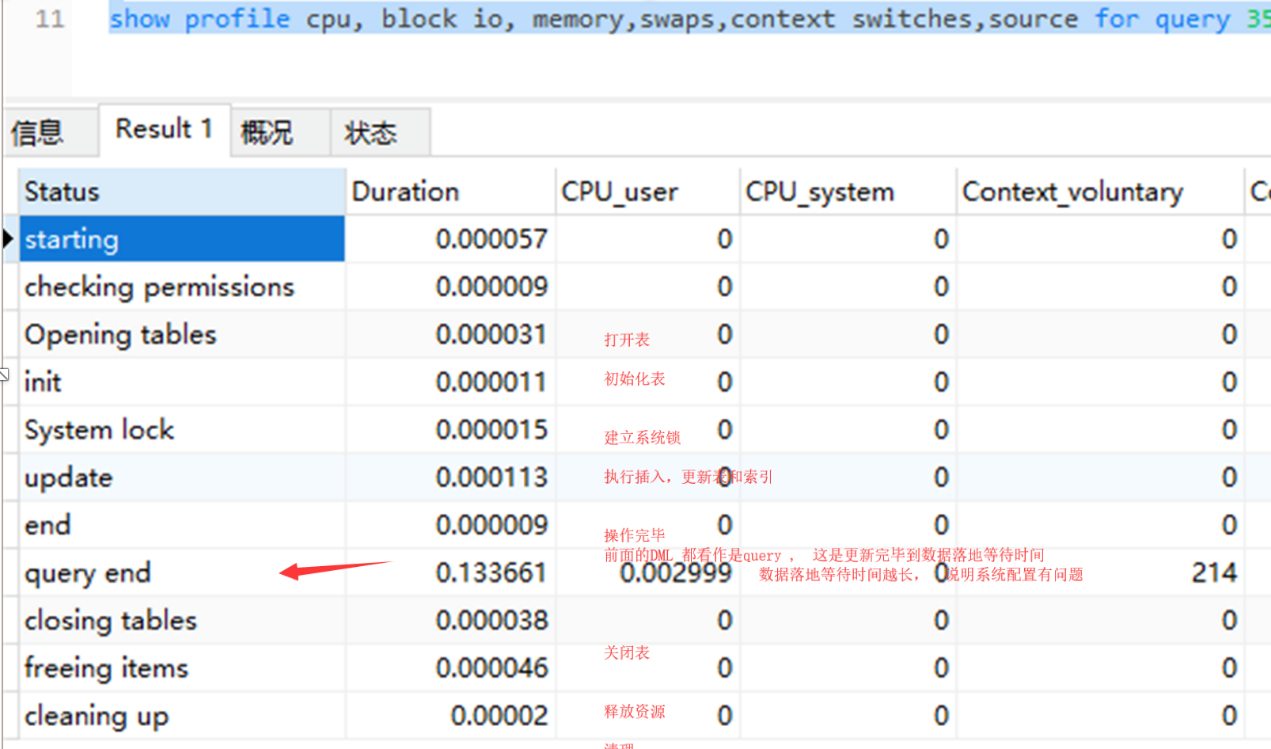
"双1设置"适合数据安全性要求非常高，而且磁盘IO写能力足够支持业务，比如订单,交易,充值,支付消费系统。双1模式下，当磁盘IO无法满足业务需求时 比如11.11 活动的压力。推荐的做法是 innodb\_flush\_log\_at\_trx\_commit=2 ，sync\_binlog=N (N为500 或1000) 且使用带蓄电池后备电源的缓存cache，防止系统断电异常。

**四、小结**

系统性能和数据安全是业务系统高可用稳定的必要因素。我们在对系统的优化需要寻找一个平衡点，合适的才是最好的，根据不同的业务场景需求，可以将两个参数做组合调整，以便是db系统的性能达到最优化。

**案例分享1：一条insert语句的执行，耗时40ms原因剖析**

背景：一个简单的带有主键的insert语句，执行起来居然要耗时40ms ，实在是难以忍受！排查分析过程如下：



因此需要关注的是数据从插入落地的IO中间都干了什么？

**一、MySQL的文件**  
首先简单介绍一下MySQL的数据文件，MySQL 数据库包含如下几种文件类型：  
1）数据文件 (datafile)  
存放表中的具体数据的文件。  
2）数据字典  
记录数据库中所有innodb表的信息。  
3）重做日志 (redolog)  
记录数据库变更记录的文件，用于系统异常crash(掉电)后的恢复操作，可以配置多个(配置这个参数inodb\_log\_files\_in\_group)比如 ib\_logfile0、 ib\_logfile1。  
4）回滚日志 (undolog)  
也存在于mysql 的ibdata文件，用户记录事务的回滚操作。注在mysql5.6以上版本可以拆开出来，单独文件夹存在。  
5）归档日志 (binlog)  
事务提交之后，记录到归档日志中。  
6）中继日志 (relaylog)  
从master获取到slave的中转日志文件，sql\_thread则会应用relay log并重放于从机器。  
7）其他日志slowlolg, errorlog, querylog  
这里慢日志也经常用。可以结合pt-query-digest工具和anemometer一起展示出来。  
对于以上文件的IO访问顺序可以分为顺序访问 比如binlog ，redolog ，relay log是顺序读写，datafile，ibdata file是随机读写，这些IO访问的特点决定了在os 配置磁盘信息时候，如何考虑分区 ，比如顺序写可以的log可以放到SAS盘 ，随机读写的数据文件可以放到ssd或者fio高性能的存储。

**二、写操作**  
为了保证数据写入操作的安全性，数据库系统设置了 undo，redo 保护机制，避免因为os或者数据库系统异常导致的数据丢失或者不一致的异常情况发生。

1）先写undo log。  
2）在内存更新数据，这步操作就在内存中形成了脏页，如果脏页过多，checkpoint机制进行刷新，innodb\_max\_dirty\_pages\_pct决定了刷新脏页比例。innodb\_io\_capacity参数可以动态调整刷新脏页的数量，innodb\_lru\_scan\_depth这个参数决定了刷新每个innodb\_buffer\_pool的脏页数量。  
3）记录变更到redo log，prepare这里会写事务id。innodb\_flush\_log\_at\_trx\_commit决定了事务的刷盘方式。为0时，log buffer将每秒一次地写入log file中，并且log file的flush(刷到磁盘)操作同时进行。该模式下，在事务提交的时候，不会主动触发写入磁盘的操作。为1，每次事务提交时MySQL都会把log buffer的数据写入log file，并且flush(刷到磁盘)中去.为2，每次事务提交时MySQL都会把log buffer的数据写入log file.但是flush(刷到磁盘)操作并不会同时进行。该模式下，MySQL会每秒执行一次 flush(刷到磁盘)操作。  
4）写入binlog这里会写入一个事务id这里有个sync\_binlog参数决定多个事务进行一次性提交。  
5）redo log第二阶段，这里会进行判断前2步是否成功，成功则默认commit，否则rollback。刷入磁盘操作。这里是先从脏页数据刷入到内存2M大小的doublewrite buffer，然后是一次性从内存的doublewrite buffer刷新到共享表空间的doublewrite buffer，这里产生了一次IO。然后从内存的内存的doublewrite buffer刷新2m数据到磁盘的ibd文件中，这里需要发生128次io。然后校验，如果不一致，就由共享表空间的副本进行修复。这里有个参数innodb\_flush\_method决定了数据刷新直接刷新到磁盘，绕过os cache。  
6）返回给client。  
如果有slave，第4步之后经过slave服务线程io\_thread写到从库的relay log ，再由sql thread应用relay log到从库中。

**三、关于性能**  
写undo redo log ，binlog的过程中都是顺序写，都会很快的完成，随机写操作，inset\_buffer功能。  
对于非聚集类索引的插入和更新操作(5.5 版本及以上支持Update/Delete/Purge等操作的buffer功能)，不是每一次都直接插入到索引页中，而是先插入到内存中。具体做法是：如果该索引页在缓冲池中，直接插入；否则，先将其放入插入缓冲区中，再以一定的频率和索引页合并，就可以将同一个索引页中的多个插入合并到一个IO操作中，改随机写为顺序写，大大提高写性能。

关于数据安全，这是数据库写入的重点？  
1，2，3过程失败就是事务失败，因为此时还未写入磁盘，对磁盘中的数据无影响，返回事务失败给client，从库也不会受到影响。 4，5过程失败的时候或者已经将写成功返回给客户，可以根据redo log的记录来进行恢复，如果出现部分写失效请参考《double write》。  
MySQL的写redo log的第一个阶段会把所有需要做的操作做完，记录数据变更，第二阶段的工作比较简单 ，只做事务提交确认。如果写入binlog成功，而第二阶段失败，MySQL启动时也会将事务进行重做，最终更新到磁盘中。MySQL 5.5+的smei sync可以更好的保障主从的事务一致性。

**四、文件访问方式**  
IO 访问的方式分为两种顺序读写和随机读写， 在MySQL的io过程中可以以此来将数据库文件分类。  
顺序读写：重做日志ib\_logfile\*，binlog file。  
随机读写：innodb表数据文件，ibdata文件。  
根据系统的访问类型，对硬件做如下分类：读多（SSD+RAID）、写多FIO(flashcache)、容量密集（fio + flashcache）。  
由于随机io会严重降低系统的性能，在当前的硬件水平下，可以考虑选择奖数据库服务器配置ssd/fusionio。

**五、影响IO的参数和策略**  
影响mysql io的参数有很多个，这里罗列几个重要的参数。  
innodb\_buffer\_pool\_size  
该参数控制innodb缓存大小，用于缓存应用访问的数据，推荐配置为系统可用内存的80%。  
binlog\_cache\_size  
该参数控制二进制日志缓冲大小，当事务还没有提交时，事务日志存放于cache，当遇到大事务cache不够用的时，mysql会把uncommitted的部分写入临时文件,等到committed的时候才会写入正式的持久化日志文件。  
innodb\_max\_dirty\_pages\_pct  
该参数可以直接控制Dirty Page在BP中所占的比率，当dirty page达到了该参数的阈值，就会触发MySQL系统刷新数据到磁盘。  
innodb\_flush\_log\_at\_trx\_commit  
该参数确定日志文件何时write、flush。  
为0，log buffer将每秒一次地写入log file中，并且log file的flush(刷到磁盘)操作同时进行.该模式下，在事务提交的时候，不会主动触发写入磁盘的操作。  
为1，每次事务提交时MySQL都会把log buffer的数据写入log file，并且flush(刷到磁盘)中去.  
为2，每次事务提交时MySQL都会把log buffer的数据写入log file.但是flush(刷到磁盘)操作并不会同时进行。该模式下,MySQL会每秒执行一次 flush(刷到磁盘)操作。  
注意：由于进程调度策略问题，这个“每秒执行一次flush(刷到磁盘)操作”并不是保证100%的“每秒”。  
sync\_binlog  
sync\_binlog的默认值是0，像操作系统刷其他文件的机制一样，MySQL不会同步到磁盘中去而是依赖操作系统来刷新binary log。  
当sync\_binlog =N (N>0) ，MySQL 在每写 N次 二进制日志binary log时，会使用fdatasync()函数将它的写二进制日志binary log同步到磁盘中去。  
innodb\_flush\_method  
该参数控制日志或数据文件如何write、flush。可选的值为fsync，o\_dsync，o\_direct，littlesync，nosync。  
数据库的I/O是一个很复杂和细致的知识层面，涉及数据库层和OS层面的IO写入策略，也和硬件的配置有关。

**案例分享2: 同一条sql语句，有时插入块，有时插入慢原因剖析**

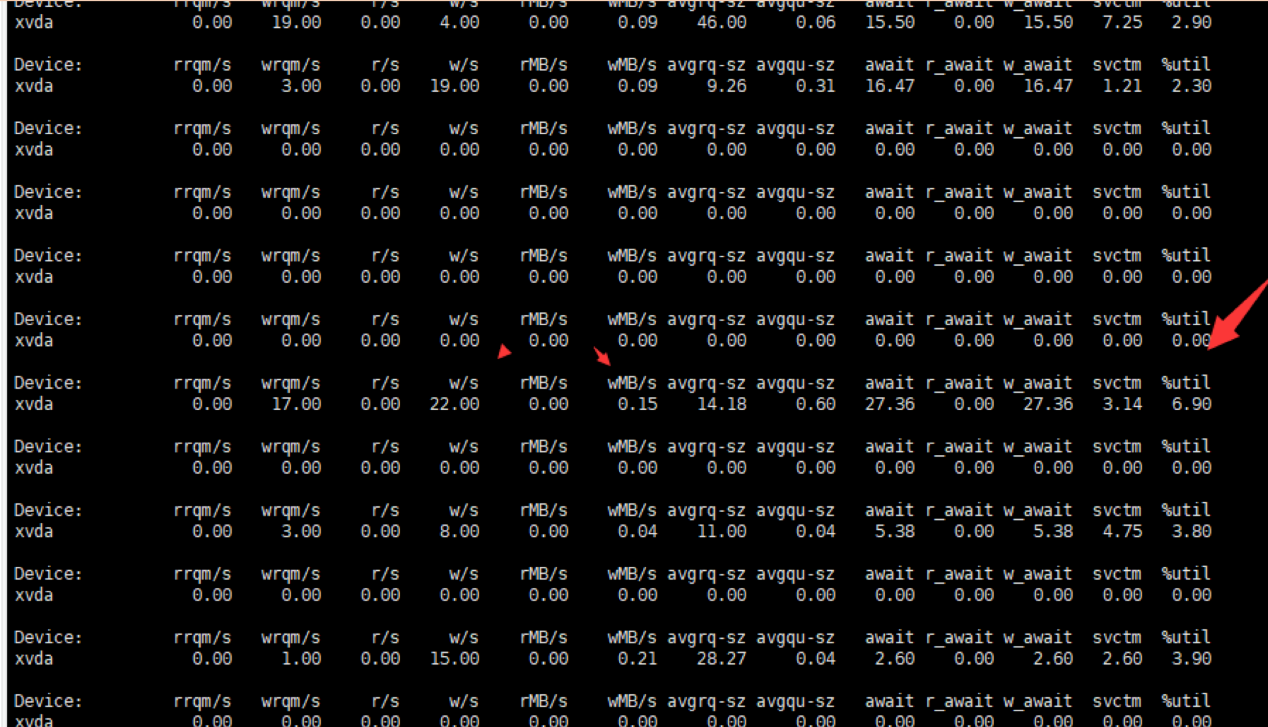
背景：同一条sql ，有时插入时间几毫秒，有时插入时间却是几十毫秒！为什么呢？ 分析过程如下：

**Sql角度**：简单insert

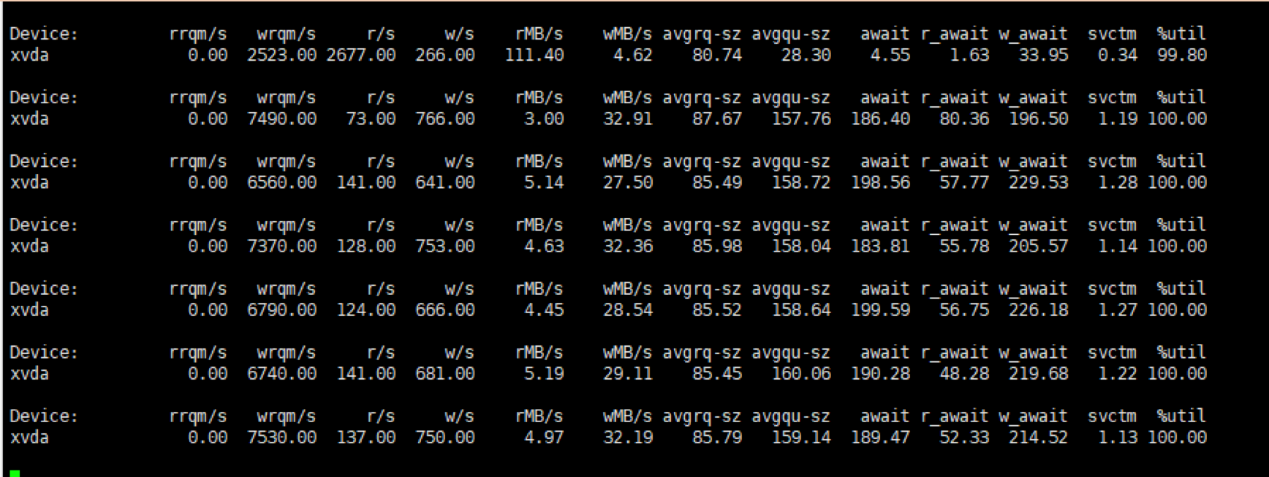
**表角度**: 一个主键

**系统参数角度：**  
开启了双1 策略。  
也就意味着每次事物就会有刷新磁盘  
关闭双1 ，设置为 0 100 ，或者 2 100 ，会极大提升性能。这是因为不刷硬盘了，但不能解决为什么时快时慢问题。

**操作系统角度**  
使用"**iostat -xmd 1**"命令看磁盘使用情况

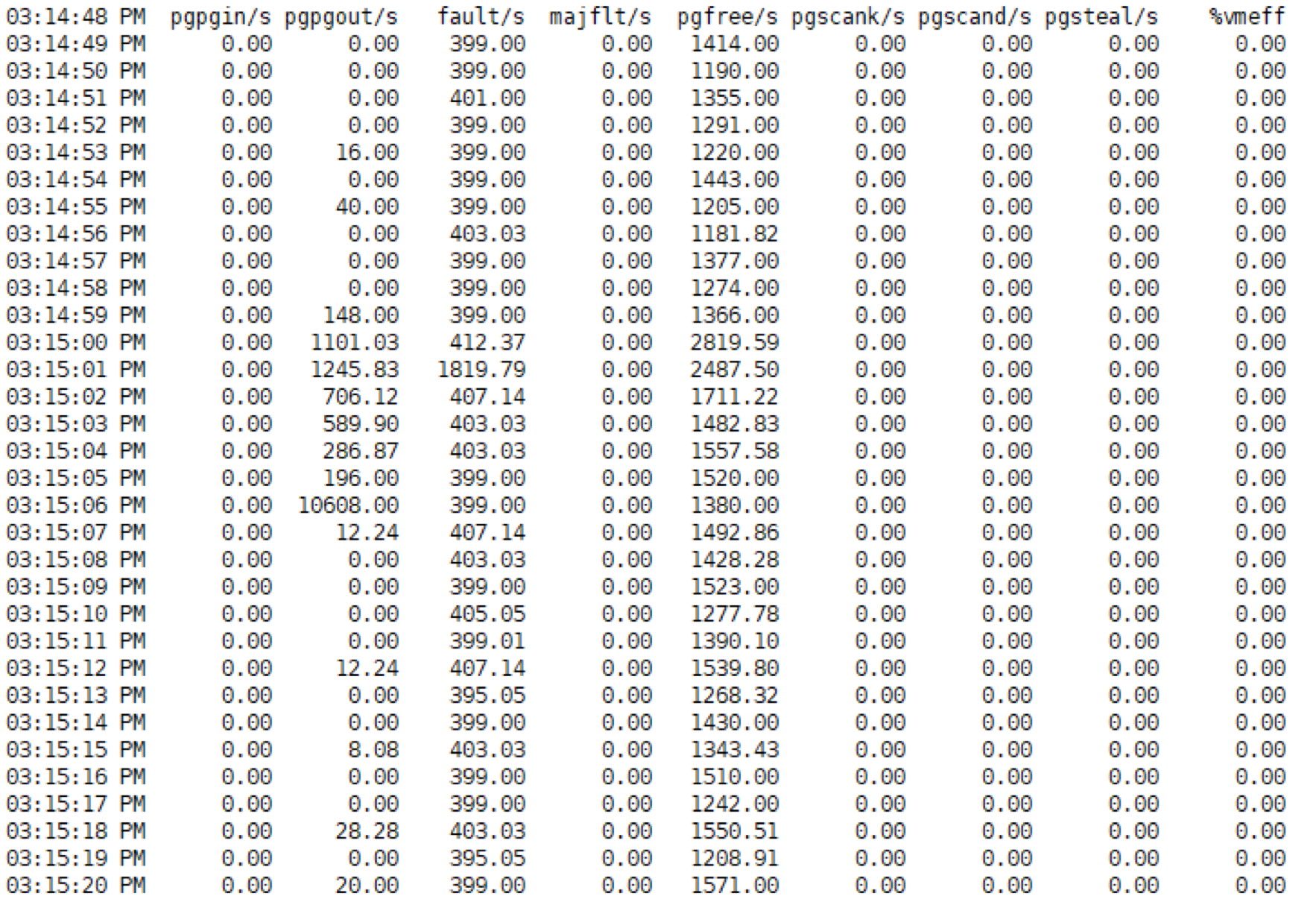


以上看出磁盘明显不够快，读写0.15M就使用了7%



上面可以看出，来个顺序文件拷贝操作，30M使用就使用了100%，离散读写更慢了！

使用"**sar -B 1**"命令可以查看页面交换

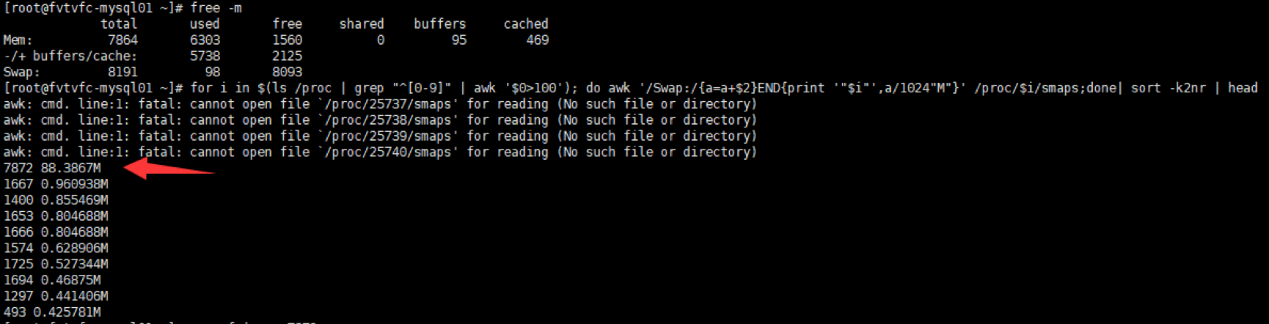


pgpgin/s:   表示每秒从磁盘或SWAP置换到内存的字节数(KB)  
pgpgout/s:  表示每秒从内存置换到磁盘或SWAP的字节数(KB)  
fault/s:  每秒钟系统产生的缺页数,即主缺页与次缺页之和(major + minor)  
majflt/s:  每秒钟产生的主缺页数.  
pgfree/s:  每秒被放入空闲队列中的页个数  
pgscank/s:  每秒被kswapd扫描的页个数  
pgscand/s:  每秒直接被扫描的页个数  
pgsteal/s:  每秒钟从cache中被清除来满足内存需要的页个数  
%vmeff:  每秒清除的页(pgsteal)占总扫描页(pgscank+pgscand)的百分比

以上表示内存和swap进行了频繁的数据交换！

**那个进程在使用swap呢？**

下面截图中命令是for i in $(ls /proc | grep "^[0-9]" | awk '$0>100'); do awk '/Swap:/{a=a+$2}END{print '"$i"',a/1024"M"}' /proc/$i/smaps;done| sort -k2nr | head



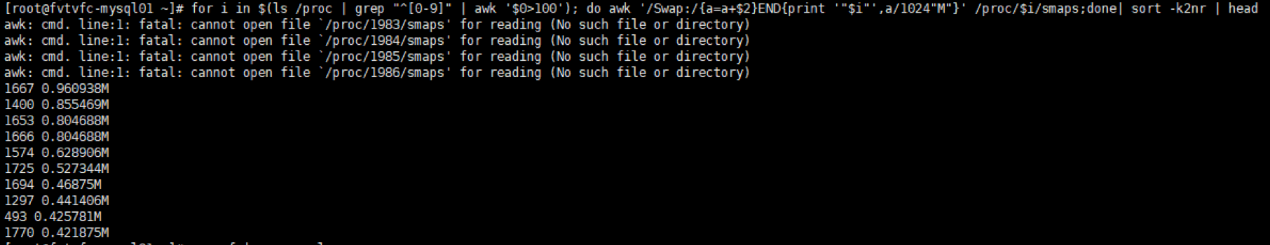
在经过几个小时后 ，mysql 使用swap由88M变成了104M ，这说明一直在使用和增加的。

**问题基本定位**  
1）首先是磁盘性能不高，顺序写才30M ,离散写会降低10倍  
2）其次是mysql又使用了swap 空间，这就使得性能更差  
3） Mysql 开启了双1 验证，就会等待数据刷磁盘，

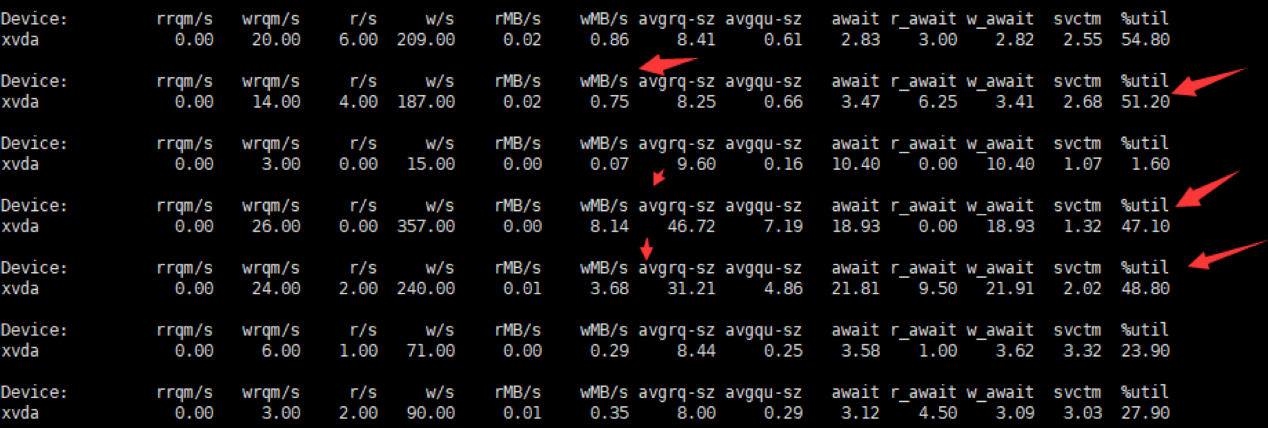
磁盘使用频率不稳定，导致了mysql的插入时间会时快时慢

**如何解决？**  
1）减少mysql使用swap方式。即把swapness设置为1。  
即执行"sysctl vm.swappiness=1",并且在/etc/sysctl.conf文件中也要设置为1；  
2）降低内存，比如设置innodb\_buffer\_pool\_size =4G，原来设置的是6G ,这样可以节约一部分内存空间；  
3）开启innodb\_numa\_interleave = ON 来操作numa；  
4）更换SSD 或者不用开启双1，改成 2 100

只调整操作系统参数，不更换硬件，依然开启双一，重启mysql之后呢？



可以看到mysql已经不再使用swap空间了。但是因为双一参数的使用，每次事物都会刷磁盘，而这个机械磁盘的性能在随机读写的情况下不稳定。会依然存在时快时慢的问题。



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 当你发现自己的才华撑不起野心时，就请安静下来学习吧！\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*