阿里中间件性能挑战赛java跑进5s“作弊”版

荐

豫川 (/users/78628) 浏览 1017 2014-10-13 07:20:26 发表于：阿里中间件性能挑战赛 (/groups/1265)

Java (/search?q=Java&type=INSIDE_ARTICLE_TAG) 无锁 (/search?q=无锁&type=INSIDE_ARTICLE_TAG)

阿里中间件性能挑战赛 (/search?q=阿里中间件性能挑战赛&type=INSIDE_ARTICLE_TAG) 修改标签 标签历史 (/articles/23737/tags/history)

写在前面的话

- 源码链接：https://github.com/oldmanpushcart/laser (https://github.com/oldmanpushcart/laser)
- 最好成绩：4.3S-5S

  最初采用java bio 客户端发起单个请求阻塞的方式，1G砖头搬完需要30S，网卡只有30M，比赛当天看到大家的分享，用C写的普遍用到协程，才跑到3S，所以私下用 java实现了一个“作弊”版，虽然是“作弊”版，底层原理和协程相似。

问题描述

- 要搬的砖头代表存在于Server上的测试数据, 来自一个大文件（排位赛为1g，决赛为16g）,文件的每一行代表一个砖头(换行符'\r\n')。文件中每一行的长度范围为0-200个字节(ascii32-127)
- client端单个线程中的调用形式为同步调用请求（当前请求未得到响应前，即未收到完整的一行，则不得发起下轮请求）,单次请求只能获取一块砖
- Server端要保证对砖头的处理是顺序的
- 需要对砖头进行处理，去掉行中间的三分之一字符（从size/3字符开始去掉size/3个字符，除法向下取整)后将剩余部分以倒序的方式传输 例如 123456789 => 123789 => 987321 每块砖头需要标记上序号（最终结果，每行前加上准确的序号，序号从零开始自增,为字符串）例如第0行aaa 则结果为 0aaa
- 以上对砖头的处理(截取三分之一,倒序,加序号)必须是在砖头被client请求之后再处理,不得在预处理阶段处理，但不限于在Server端或者Client端做处理(请求哪块砖头处理哪块)
- 保证砖头数量并保证写入结果文件的顺序与服务器文件的顺序一致。

比赛机器配置

- 24核+96G内存+千兆网卡 物理机 * 2
- Linux version 2.6.32-220.23.2.ali878.el6.x86_64 (ads@kbuild) (gcc version 4.4.4 20100726 (Red Hat 4.4.4-13) (GCC))
- Intel Corporation 82580 Gigabit Network Connection
- Intel(R)Xeon(R)E5-24300 @ 2.20GHz
- JDK8

比赛关注点

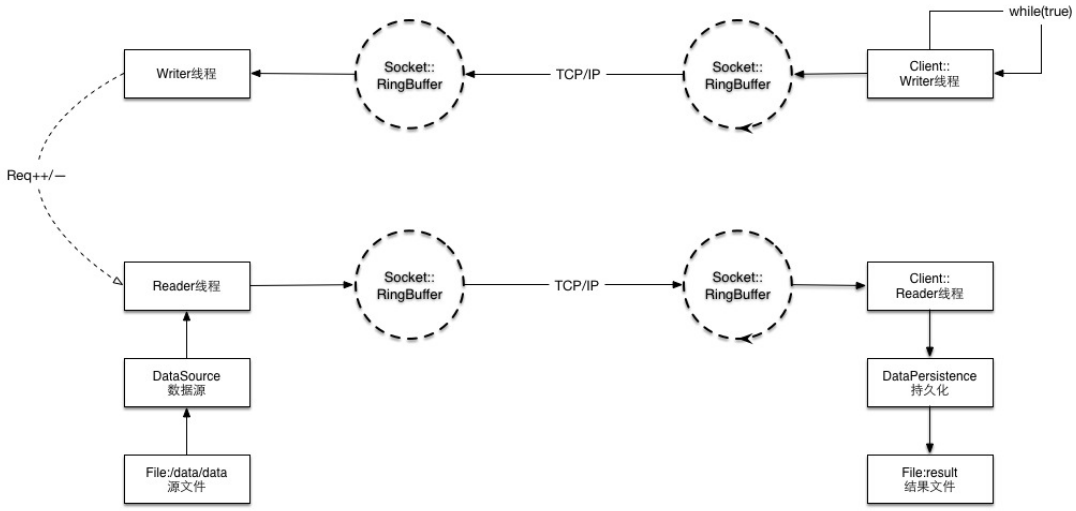
- 如何塞满MTU(数据链路层最大传输单元)，用满整个带宽
- 要保证有序，就需要在client和server端设计合理的数据结构存储每块砖头，同时为了压满网卡，客户端需要制造足够的压力，同时避免线程切换，避免锁竞争，如何减少GC的时间和次数

搬砖头整体设计思路

- 整体设计图

  server和client建立12个socket连接,每个连接的两边各维护一个读写线程用于发送REQ和接收RESP，为了压满mss(max segment size),客户端和服务端分别存在24个线程，与CPU核数相对应；客户端每个请求长度为4个字节，客户端写线程每次累计N个请求的消息之后，才会发出请求(这个地方作弊，后面会解释)，服务端每条消息长度为256

字节(保证内存对齐)，服务端写线程在累积到256*N时会向客户端发起响应。

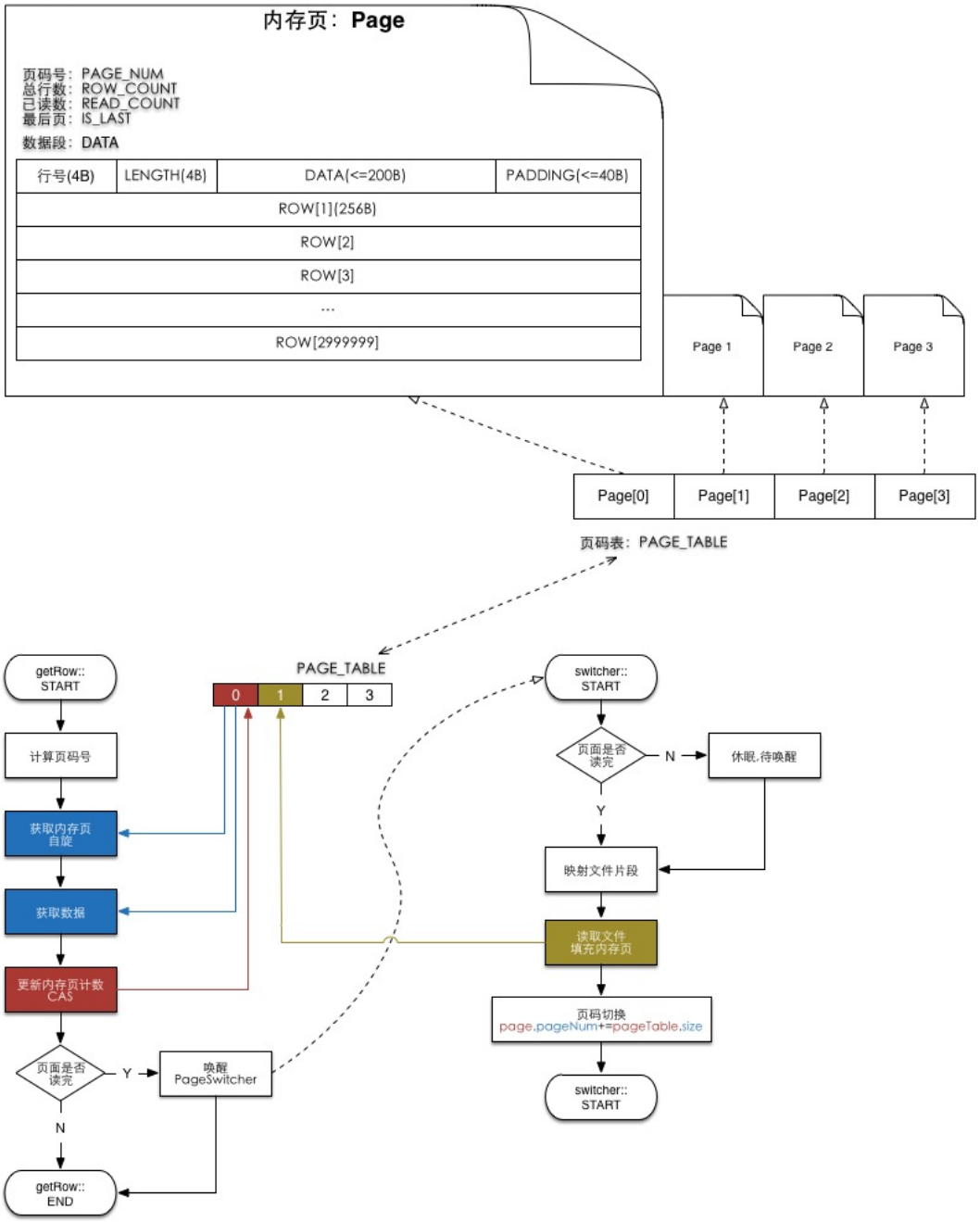


(http://img2.tbcdn.cn/L1/461/1/03b3a9ddba981256f4df295c240854ca9b954736)

- server端内存页数据结构及流程图

  无锁文件读写：客户端最终文件结果需要保证和服务端一致的顺序，由于网络的原因，服务端需要对每行记录添加行号，服务端加载本地文件到内存并解析到队列或数组，线程访问共享的队列或数组，需要做到无锁读写。

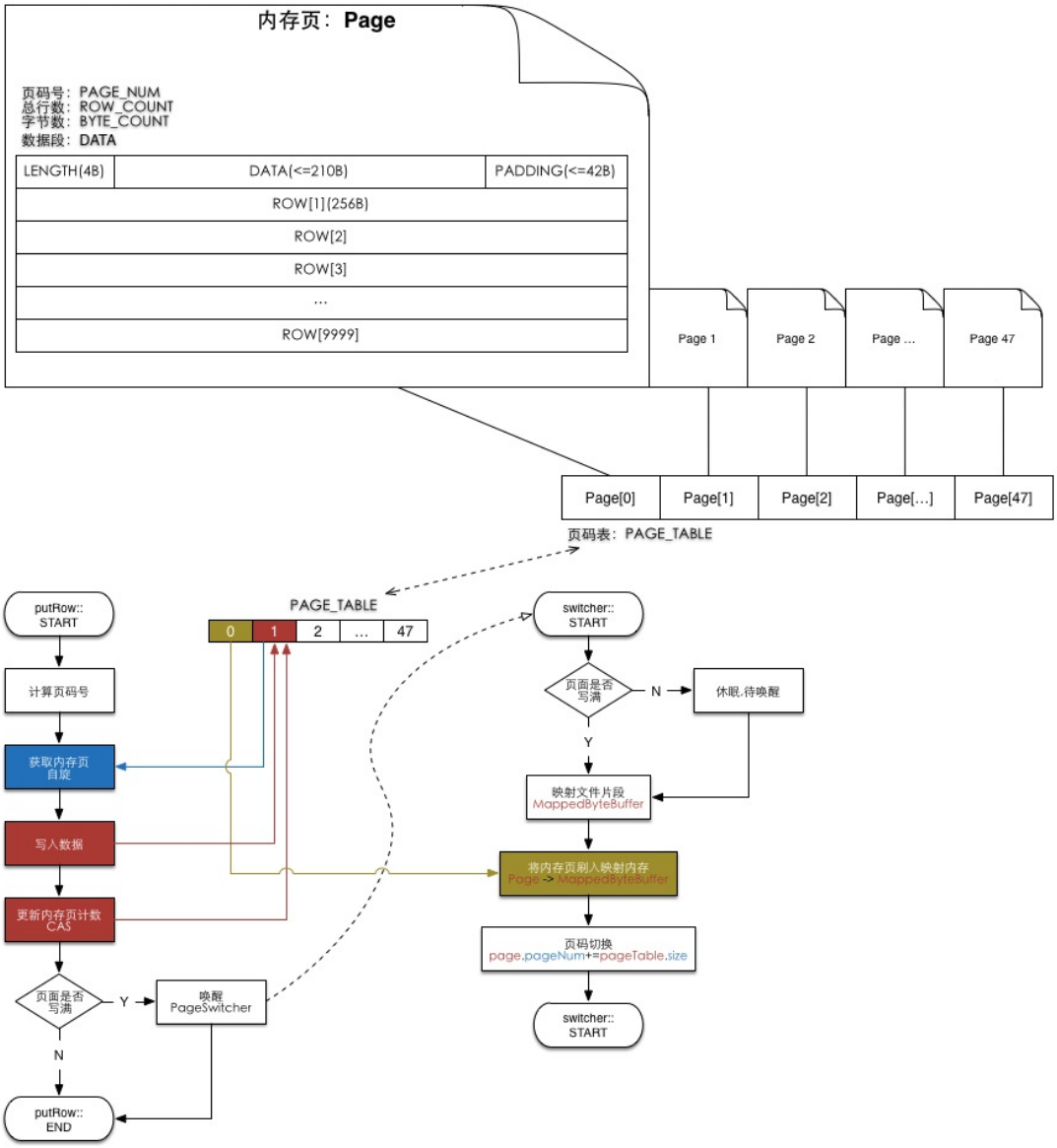
这里采用的我们采用了无锁内存页切换的方案。
维护一个页码表，每个内存页中包含一个定长字节数组，reader线程通过CAS的方式获取每行记录



(http://img4.tbcdn.cn/L1/461/1/4c18ec3301767e7cd52fb541736e3a387b3f3998)

- client端内存页及流程图

  client端putRow时计算对应的pageNum，以及在对应内存页中的偏移位置，client端维护一个pageSwitcher线程,从页码表下表为0的位置开始。当内存页写满，writer线程唤醒pageSwitcher线程，将当前内存页mmap到文件中，pageSwitcher线程指向下一内存页；如果该页未满，wait



(http://img2.tbcdn.cn/L1/461/1/83c452f9086c5eab0e5b2d88decb246dc054268b)

关键设计代码

- 客户端 writer代码

  根据row获取行号，通过页码表的size计算对应页码，获取pageNum在页码表中的位置tableIdx，当发现页码表中tableIdx处的页码没有被替换，说明pageNum页还未在页码表中创建，此时writer线程自旋，直到该页被创建。

```
    final int lineNum = row.getLineNum();

    // 计算页码
    final int pageNum = lineNum / PAGE_ROWS_NUM;

    // 计算页码表位置
    final int tableIdx = pageNum % PAGE_TABLE_SIZE;

    while (pageTable[tableIdx].pageNum != pageNum) {
        // 如果页码表中当前位置所存放的页面编码对应不上
        // 则认为页切换不及时，这里采用自旋等待策略，其实相当危险
//          log.info("debug for spin, page.pageNum={},pageNum={},lineNum={}",
//                  new Object[]{pageTable[tableIdx].pageNum, pageNum, lineNum});
    }
```

- 客户端reader线程

  多个reader线程可能同时从同一页中读取一行记录，采用 CAS的方式保证每次只有一个线程成功将page.readCount加1，并向下执行。

```
    while (true) {

        final Page page = currentPage == null ?
                 pageTable[0] : currentPage;
        final int readCount = page.readCount.get();
        final int rowCount = page.rowCount;

        if (page.isLast
                && readCount == rowCount) {
            if (null == row) {
                return EMPTY_ROW;
            } else {
                row.setLineNum(EMPTY_ROW.getLineNum());
                row.setData(EMPTY_ROW.getData());
                return row;
            }
        }

        if (readCount == rowCount) {
            continue;
        }

        if (!page.readCount.compareAndSet(readCount, readCount + 1)) {
            continue;
        }

        ....
    }
```

> 页码表设计原理
> 减少GC的开销：每个内存页是一个字节数组，连续的内存空间，直接进老年代，同时在整个过程中被复用，对象复用可以减少内存中对象的个数，减小GC遍历时需要标记的对象的路径长度
>
> 无锁：如果对页码表加锁，那么势必会有线程被挂起，CPU线程切换，性能下降，采用自旋的方式，避免了对锁的争用，保证每个线程都处于活跃状态

- 客户端writer线程发起请求

  客户端每个请求占用四个字节，只有当writer线程将buffer填满是才会发起请求。
  由于采用NIO的方式writer线程每当buffer填满就会向server发起请求，每个请求相当于包含了多条记录(模拟协程)，目的在于给server制造足够的压力

```
            final ByteBuffer buffer =
                ByteBuffer.allocateDirect(options.getClientSendBufferSize());
            while (isRunning) {
//                  final GetDataReq req = new GetDataReq();
                if (buffer.remaining() >= Integer.BYTES) {
//                        buffer.putInt(req.getType());
                    buffer.putInt(LaserConstant.PRO_REQ_GETDATA);
                    continue;
                } else {
                    socketChannel.register(selector, OP_WRITE);
                    buffer.flip();
                }

                selector.select();
                final Iterator<SelectionKey> iter = selector.selectedKeys().iterator();
                while (iter.hasNext()) {
                    final SelectionKey key = iter.next();
                    iter.remove();

                    if (key.isWritable()) {
//                          final int count =
                        socketChannel.write(buffer);
//                          log.info("debug for write, count="+count);
                        buffer.compact();
                        key.interestOps(key.interestOps() & ~OP_WRITE);
                    }
                }
            }
```

- client端reader线程

  reader线程将buffer填满

```
final ByteBuffer buffer =
ByteBuffer.allocateDirect(options.getClientReceiverBufferSize());
final ReadableByteChannel readableByteChannel =
options.isEnableCompress()
        ? new CompressReadableByteChannel(socketChannel,
        options.getCompressSize())
        : socketChannel;
```

测试结果

- 测试结果在4.3S－5S之间
    - **client_vmstat**

client端启动时，与server端的连接和页码表的创建均采用异步任务的方式，所以启动时线程切换频繁。当Client完成启动，开始发送请求后，线程切换的次数将保持相对低的值。

```
 0  0   7224 72071696 300628 19586576    0    0     0     0  351   415  0  0 100  0  0
 9  0   7224 71896752 300628 19586576    0    0     0     0 11379 14794  4  1 96  0  0
14  0   7224 71627080 300628 19586608    0    0     0     0 204881 121580 62 12 25  0  0
 4  0   7224 71040688 300628 19586608    0    0     0     0 128707 154827 24  9 67  0  0
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
11  0   7224 70471776 300628 19586608    0    0     0     0 140024 164611 21 10 69  0  0
 6  0   7224 70061824 300628 19586608    0    0     0     0 145904 174304 14 10 76  0  0
 1  1   7224 69788288 300628 19586608    0    0     0 623580 83946 107134  6  7 85  2  0
 0  4   7224 69776880 300632 19586608    0    0     0 156724 2485 1470  0  1 92  7  0
 0  3   7224 69799200 300632 19586608    0    0     0     4 2401 1587  0  0 92  8  0
 0  5   7224 69818672 300632 19586608    0    0     0    64 2377 1770  0  0 90 10  0
 0  5   7224 69840496 300632 19586608    0    0     0     0 2352 1443  0  0 83 16  0
 0  5   7224 69864192 300632 19586608    0    0     0     0 2398 1819  0  0 84 16  0
 0  3   7224 69885184 300632 19586608    0    0     0     4 2406 1612  0  0 89 11  0
 0  0   7224 72070064 300632 19586608    0    0     0    84 10234 5976  0  1 99  0  0
 0  0   7224 72070160 300632 19586608    0    0     0     0  363   451  0  0 100  0  0
```

(http://img4.tbcdn.cn/L1/461/1/3dc93f4534cbb363a35cd996043a16816f182a7b)

- **client_gc**

client端，没有full gc 只发生一次young gc，时间16ms

```
$jstat -gccause `jps|grep laser|awk '{print $1}'` 1000 1000
  S0     S1     E      O      M     CCS    YGC     YGCT    FGC    FGCT     GCT    LGCC                 GCC
  0.00   0.00  74.00   1.31  15.13  17.34      0    0.000     0    0.000    0.000 No GC                No GC
  0.00   0.00  84.00   1.31  15.13  17.34      0    0.000     0    0.000    0.000 No GC                No GC
  0.00   0.00  94.00   1.31  15.13  17.34      0    0.000     0    0.000    0.000 No GC                No GC
  0.00  16.45  26.28   1.31  94.31  83.88      1    0.016     0    0.000    0.016 Allocation Failure   No GC
  0.00  16.45  47.76   1.31  94.31  83.88      1    0.016     0    0.000    0.016 Allocation Failure   No GC
  0.00  16.45  50.82   1.31  94.31  83.88      1    0.016     0    0.000    0.016 Allocation Failure   No GC
  0.00  16.45  50.82   1.31  94.31  83.88      1    0.016     0    0.000    0.016 Allocation Failure   No GC
  0.00  16.45  50.82   1.31  94.31  83.88      1    0.016     0    0.000    0.016 Allocation Failure   No GC
  0.00  16.45  50.82   1.31  94.31  83.88      1    0.016     0    0.000    0.016 Allocation Failure   No GC
  0.00  16.45  50.82   1.31  94.31  83.88      1    0.016     0    0.000    0.016 Allocation Failure   No GC
  0.00  16.45  50.82   1.31  94.31  83.88      1    0.016     0    0.000    0.016 Allocation Failure   No GC
  0.00  16.45  50.82   1.31  94.31  83.88      1    0.016     0    0.000    0.016 Allocation Failure   No GC
```

(http://img4.tbcdn.cn/L1/461/1/0e5fc1ad5fa51deb09c3c48f370b4242e1409977)

- **server_gc**

server端，没有full gc，两次young gc用时17ms

```
$jstat -gccause `jps|grep laser|awk '{print $1}'` 1000 1000
  S0     S1     E      O      M     CCS    YGC     YGCT    FGC    FGCT     GCT    LGCC                 GCC
  0.00   0.00  64.00  47.68  15.13  17.34      0    0.000     0    0.000    0.000 No GC                No GC
  0.00   0.00  80.00  47.68  15.13  17.34      0    0.000     0    0.000    0.000 No GC                No GC
  0.00   0.00  98.00  47.68  15.13  17.34      0    0.000     0    0.000    0.000 No GC                No GC
  0.00   1.10  17.00  47.68  96.13  92.67      1    0.009     0    0.000    0.009 Allocation Failure   No GC
  0.00   1.10  37.00  47.68  96.13  92.67      1    0.009     0    0.000    0.009 Allocation Failure   No GC
  0.00   1.10  43.00  47.68  96.13  92.67      1    0.009     0    0.000    0.009 Allocation Failure   No GC
  0.00   1.10  43.00  47.68  96.13  92.67      1    0.009     0    0.000    0.009 Allocation Failure   No GC
  0.00   1.10  43.00  47.68  96.13  92.67      1    0.009     0    0.000    0.009 Allocation Failure   No GC
  0.00   1.10  43.00  47.68  96.13  92.67      1    0.009     0    0.000    0.009 Allocation Failure   No GC
  0.00   1.10  43.00  47.68  96.13  92.67      1    0.009     0    0.000    0.009 Allocation Failure   No GC
  0.00   1.10  43.00  47.68  96.13  92.67      1    0.009     0    0.000    0.009 Allocation Failure   No GC
  0.00   1.10  59.60  47.68  96.13  92.67      1    0.009     0    0.000    0.009 Allocation Failure   No GC
  0.00   1.10  70.40  47.68  96.13  92.67      1    0.009     0    0.000    0.009 Allocation Failure   No GC
  0.00   1.10  78.40  47.68  96.13  92.67      1    0.009     0    0.000    0.009 Allocation Failure   No GC
  0.00   1.10  86.40  47.68  96.13  92.67      1    0.009     0    0.000    0.009 Allocation Failure   No GC
  1.79   0.00   5.05  47.68  95.99  95.08      2    0.017     0    0.000    0.017 Allocation Failure   No GC
  1.79   0.00   9.72  47.68  95.99  95.08      2    0.017     0    0.000    0.017 Allocation Failure   No GC
  1.79   0.00   9.72  47.68  95.99  95.08      2    0.017     0    0.000    0.017 Allocation Failure   No GC
  1.79   0.00   9.72  47.68  95.99  95.08      2    0.017     0    0.000    0.017 Allocation Failure   No GC
  1.79   0.00   9.72  47.68  95.99  95.08      2    0.017     0    0.000    0.017 Allocation Failure   No GC
  1.79   0.00   9.72  47.68  95.99  95.08      2    0.017     0    0.000    0.017 Allocation Failure   No GC
  1.79   0.00   9.72  47.68  95.99  95.08      2    0.017     0    0.000    0.017 Allocation Failure   No GC
  1.79   0.00   9.72  47.68  95.99  95.08      2    0.017     0    0.000    0.017 Allocation Failure   No GC
  1.79   0.00   9.72  47.68  95.99  95.08      2    0.017     0    0.000    0.017 Allocation Failure   No GC
  1.79   0.00   9.72  47.68  95.99  95.08      2    0.017     0    0.000    0.017 Allocation Failure   No GC
  1.79   0.00   9.72  47.68  95.99  95.08      2    0.017     0    0.000    0.017 Allocation Failure   No GC
  1.79   0.00  12.03  47.68  95.99  95.08      2    0.017     0    0.000    0.017 Allocation Failure   No GC
```

(http://img1.tbcdn.cn/L1/461/1/18f12ef3a56ac004ea7f14445b333cba3953c7a7)

- **server_vmstat**

server端，采用无锁队列，线程切换次数低，但cpu时间片还存在空闲，说明客户端制造的压力还不够

```
 2  0 2371712 87427736 192572 4033560    0    0     0     0 1407  490  4  0 96  0  0
 0  0 2371712 87428232 192572 4033560    0    0     0     0  867  533  2  0 98  0  0
 0  0 2371712 87428368 192572 4033560    0    0     0     0  420  503  0  0 100  0  0
 2  0 2371712 87398848 192572 4033560    0    0     0     0 102835 98729 13  2 85  0  0
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 6  0 2371712 87398728 192572 4033560    0    0     0     0 167649 190936  5  3 92  0  0
11  0 2371712 87399688 192572 4033560    0    0     0     0 173144 173057 15  5 80  0  0
14  0 2371712 87399688 192572 4033560    0    0     0     0 214989 139478 45  9 46  0  0
 0  0 2371712 87395224 192572 4033560    0    0     0     0 224682 138978 44  9 46  0  0
 0  0 2371712 87394232 192572 4033560    0    0     0     0 1599 1020  0  0 100  0  0
 0  0 2371712 87394112 192572 4033560    0    0     0     0 1532 1009  0  0 100  0  0
```

(http://img1.tbcdn.cn/L1/461/1/2779bdbc9fc6bb77c2ef64481cdea76c5d5a2c12)

- **server_network**

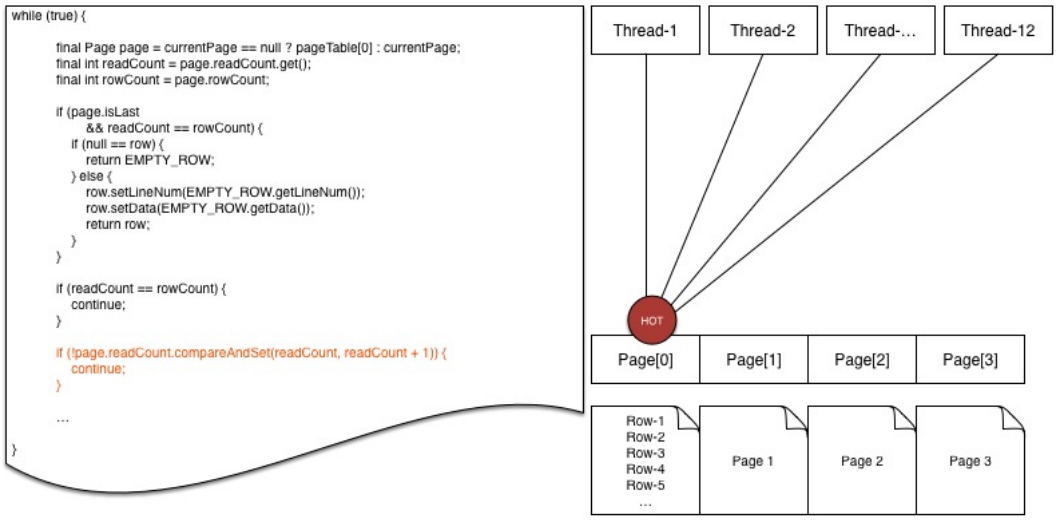
server端网卡，写网卡，平均在200M左右，没压满双网卡

```
Time                ------------traffic------------
Time                 bytin   bytout   pktin   pktout
13/10/14-05:01:15 540.00   744.00    7.00    4.00
13/10/14-05:01:17   1.3K     1.6K   15.00    7.00
13/10/14-05:01:19 249.00   341.00    3.00    2.00
13/10/14-05:01:21 950.00   916.00   11.00    6.00
13/10/14-05:01:23 351.00   477.00    5.00    2.00
13/10/14-05:01:25 395.00   420.00    4.00    2.00
13/10/14-05:01:27 162.00   420.00    2.00    2.00
13/10/14-05:01:29 314.00   420.00    4.00    2.00
13/10/14-05:01:31 285.00   605.00    4.00    2.00
13/10/14-05:01:33  11.6M    67.1M   93.2K   70.6K
13/10/14-05:01:35  17.7M   221.7M  148.7K  191.8K
13/10/14-05:01:37  13.3M   132.7M  108.7K  123.2K
13/10/14-05:01:39  19.5K   225.9K  302.00    3.2K
13/10/14-05:01:41  19.5K   225.9K  302.00    3.2K
13/10/14-05:01:43  19.5K   225.9K  302.00    3.2K
```

(http://img1.tbcdn.cn/L1/461/1/5d6510a1436437e2a152b16a5c89c4095a43b458)

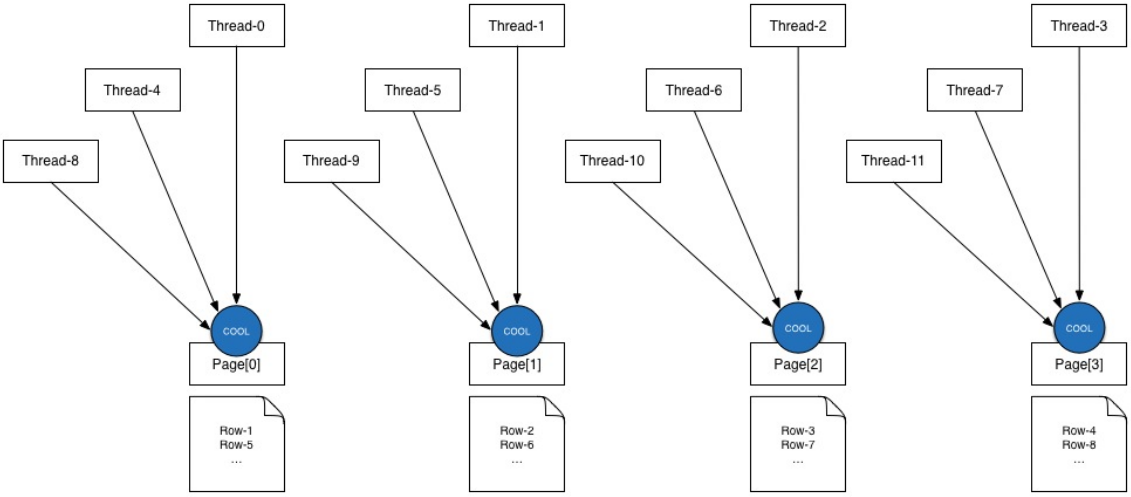
反思

- 如下图所示，对于页码表，同一个内存页在同一时刻CAS竞争会非常激烈，那么CAS重试的次数增多，消耗CPU资源



(http://img2.tbcdn.cn/L1/461/1/2c85e48b537bd3b27c85a91bab50f8184fd4a838)

- 优化思路：如果将线程按照编号hash，将其散列到不同的分组，同一个分组的reader线程，去对应的内存页getRow，可以大大减少CAS的竞争，降低重试次数，同时可以创建两份页码表，在第一份页码表被线程访问时，后台线程，将第二份页码表填充，第一份页码表为空之后，各线程组直接切换到第二份



(http://img4.tbcdn.cn/L1/461/1/298f92944e42cef8af2667d94cb506b0e6ce1740)

- 观察网络情况，发现MSS并未压满，大概在1100B,client制造的压力还不够

评论文章 (7) 30 (/articles/23737/unvoteup) 0 ★ 17 (/articles/23737/mark)

关注 (/articles

他们赞过该文章

(/users/22469) (/users/22961) (/users/92846) (/users/15823) (/users/66288) (/users/10570)
(/users/80912) (/users/72622) (/users/74361) (/users/24680) (/users/81253) (/users/71841)
(/users/54183) (/users/10631) (/users/12016) (/users/82502) (/users/7797) (/users/95908)
(/users/66558) (/users/65534) (/users/70779) (/users/67583) (/users/34292) (/users/93405)
(/users/74872) (/users/12679) (/users/70643) (/users/74470) (/users/71568) (/users/35)

27人关注该文

(/users/7747
(/users/2162
(/users/7097
(/users/1448
(/users/2040
(/users/6689
(/users/2510
(/users/6628
...
(/articles/237

相似文章

- netty4 UDP 通信 (/articles/24090)
- axis1.4bug造成的所有线程阻塞 (/articles/7854)
- 采用threadlocal缓存替代连接池对连接进行管理 (/articles/12751)
- okHttp功能与源码分析 (/articles/17520)
- NTP授时方式及原理简介 (/articles/15828)
- 璧说：从数据库连接池说起 (/articles/31833)

下一篇：delivery运费计算逻辑 (/articles/26459)

山大 (/users/35) 2014-10-13 09:58:29
人工点赞
(/users/35) 1F
0 (/comments/31130/voteup) 0

无朽 (/users/67583) 2014-10-13 19:05:30
人工再次点赞。
(/users/67583) 2F
0 (/comments/31220/voteup) 0

激酶 (/users/68180) 2014-10-13 19:54:20
人工三次点赞
(/users/68180) 3F
0 (/comments/31230/voteup) 0

杜琨 (/users/12679) 2014-10-14 13:14:44
(/users/12679) 4F

对楼上的点赞行为表示无语～说多几个字吧～

我看到vmstat中表露出在系统最繁忙的时候，无论Client还是Server都有比较多的空闲，另外根据tsar反馈的网卡流量计算，MSS只有1183.63，远远没有达到填满（1460）的预期，说明在4.3S的基础上还有很大提升空间。

0 (/comments/31257/voteup) 0 编辑 删除 (/comments/destroy/31257/)

(/users/82502) 5F

墨嘿 (/users/82502) 2014-10-14 15:44:30

人工四次点赞。

0 (/comments/31287/voteup) 0

(/users/78628) 6F

豫川 (/users/78628) 2014-10-14 16:11:21

给点优化的建议、优化的方向或者由此想到的更好的方案

0 (/comments/31289/voteup) 0

(/users/76749) 7F

姬望 (/users/76749) 2014-10-20 09:36:09

呵呵 顶！

0 (/comments/31547/voteup) 0

写下你的评论…

评论

© 2015 阿里巴巴集团 版权所有 Copyright© 15 ATA. All rights reserved. (http://www.cnzz.com/stat/website.php?web_id=5837538) (http://www.cnzz.com/stat/website.php?web_id=1254194462)