TEG学习与发展园地

已加入

linux环境下c++程序问题诊断与调优V0.2.pdf

v_mxhe 2019-07-09 16:16 大小(3MB) 收藏(0) 下载

附件来源：[文章]【C++后台开发学习专项】第二期复习资料

技术创造未来 科技TEG

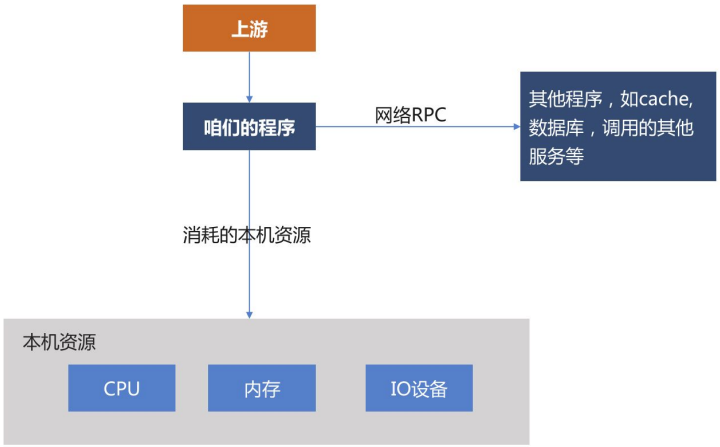
Linux环境下 c++程序问题诊断与调优

Harlylei 2019.3

目录

技术创造未来 科技TEG

程序的运行环境



问题分类

CPU问题

- 机器负载高
- CPU消耗高
- 响应时耗高
- 资源消耗低，但是压力上不去
- 上下文切换频繁，cache miss率高
- 锁冲突严重

内存问题

- 内存消耗多，内存泄漏，内存碎片
- 内存分配回收性能差
- 内存越界，空指针，野指针
- 系统coredump或者数据异常

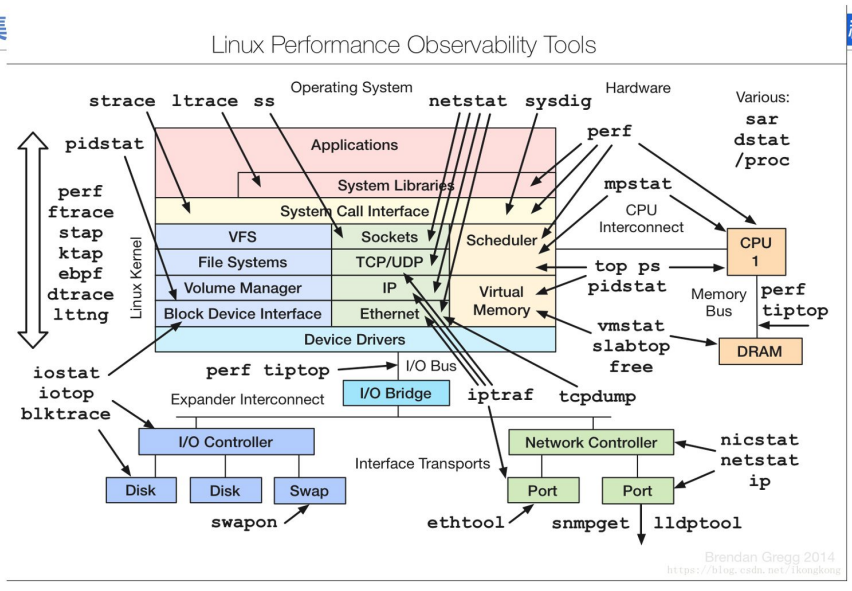
磁盘IO问题

- IO消耗太多，引起系统影响慢
- 大量随机IO
- 句柄泄漏
- swap引起系统响应变慢
- 写入磁盘的数据丢失，磁盘损坏等

网络IO问题

- 网络不通
- 延时大
- 网络超时，丢包
- 连接异常
- 大量无效连接
- 网卡跑满
- CPU处理网卡中断繁忙

工具集



测试程序介绍

为了配合后面各种工具的介绍，所以开发了一个简单的模拟测试程序testperf，用来模拟CPU消耗，IO消耗，内存泄漏等等

Git地址： https://git.code.oa.com/harlylei/perf_tune_solve_problem_demo

程序由以下7个线程组成：

3个执行高CPU消耗的线程，主函数costCpuFun，在循环里执行更新string和大量调用gettimeofday

1个执行中等CPU消耗的线程，主函数costLowCpuFun，执行加法运算，但是会通过usleep睡眠

1个执行IO读的线程（支持direct模式），用来引起较大的IO操作

1个模拟内存泄漏的线程

1个主线程，等待各个子线程退出

目录

CPU负载分析-uptime

[harlylei@node03 ~]$**uptime**

17:44:36 up 40 days, 4:34, 7 users, load average: 1.40, 1.75, 1.90

输出解释：

up 40 days, 4:34 机器已经启动了多久，比如机器做了重启，可以快速发现

7 users： 当前登录了多少个用户

load average：过去1分钟，5分钟，15分钟的负载情况，关于负载的理解：

正在执行的进程数目：当前正在使用CPU（R状态）
+
处于待执行队列的进程数目：只要调度到就能马上执行，比如等待锁成功，时间片耗尽切换到别的进程而进入这里(R状态)
+
不可中断中断的进程数目(D状态)：一般是等待IO完成。

这三个值的总和应该要小于2*cpu逻辑核，否则可能是遇到了严重的性能问题，如CPU资源不够或者IO太慢，阻塞了太多的进程读写。

这个命令非常简单，消耗非常少的资源，执行速度快，可以作为最先执行的命令。但是信息也不算特别充分，只是可以粗略分析是否负载较高。

CPU负载分析-vmstat

接下来从进程，内存，swap,page,系统上下文切换,cpu消耗方面继续加强观察的指标粒度，通vmstat命令

[harlylei@node05 ~]$vmstat 1（1秒钟输出一行）

```
procs -- ----memory---------------------  ---swap-- -----io---   system-- -----    ------cpu---
 r  b       swpd   free        buff        cache        si   so    bi      bo       in       cs          us
sy id wa st
 7  1          0      659604    390112 103467832    0    0    24376  4856     42539    107881    18
8 70  4  0
 8  1          0      646740    390112 103480920    0    0    24508  2692     41230     104447   16
7 73  4  0
```

CPU负载分析-vmstat

输出解释：
第一行是系统启动以来的平均值，一般忽略

r: 可执行的线程数量，正在执行和处于待执行队列都算，跟load的区别是不包含处于D状态的线程，真正的CPU执行需求

b: D状态处于IO阻塞状态的线程数目，如过大则意味着较多的线程在等待IO

memory:为内存信息，见后面free命令

swap:多少个页面换入，换出，如不为0意味着内存不够，性能杀手，实时业务很容易看到毛刺，建议关闭

IO：每秒多少个磁盘块的读写

System: in代表中断的数目，cs代表进程上下文切换的次数，进程上下文切换代价是比较大的，实时应用要尽量降低

Cpu:用户态，系统态，空闲态，等待IO的时间 分配比例。其实wa等待IO也可以算作cpu空闲，但是过大意味着IO忙，所以跟id区分展示

CPU负载分析-top

[harlylei@node03 ~]$top

```
top - 18:30:16 up 140 days,  3:16, 14 users,  load average: 6.14, 4.14, 2.87
Tasks: 669 total,   2 running, 623 sleeping,   0 stopped,  44 zombie
%Cpu(s): 19.4 us,  7.5 sy,  0.0 ni, 69.3 id,  3.8 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 12351808+total, 31779712 free, 38257664 used, 53480712 buff/cache
KiB Swap:        0 total,        0 free,        0 used. 80590560 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
57364 harlylei  20   0  263732   1132    928 S 377.7  0.0  12:39.67 testperf
 7993 root      20   0   37528   3212   2156 S  13.6  0.0  60:08.62 sap1006
```

输出解释：
1. 第一行显示与uptime结果一致
2.Tasks:代表当前总共有多少进程，各种状态的进程数目（不计算进程里的线程）
3. 第三行将cpu消耗进行了按比例分类（下面标红的比例需要重点了解下），
   18.6%us — 用户空间占用CPU的百分比。
   7.5% sy — 内核空间占用CPU的百分比。
   0.0% ni — 改变过优先级的进程占用CPU的百分比
   70.1% id — 空闲CPU百分比
   3.8% wa — IO等待占用CPU的百分比
   0.0% hi — 硬中断（Hardware IRQ）占用CPU的百分比
   0.0% si — 软中断（Software Interrupts）占用CPU的百分比
4.第4行，5行代表内存信息，后面通过free命令再讲

CPU负载分析-top

top的常见用法

1.只针对指定进程进行top观察

比如只想看testperf的结果，先找到进程id，如上页所示为57364,则执行top -p 57364

```
top - 18:31:53 up 140 days,  3:18, 14 users,  load average: 5.83, 4.55, 3.14
Tasks:   1 total,   0 running,   1 sleeping,   0 stopped,   0 zombie
%Cpu(s): 19.0 us,  7.8 sy,  0.0 ni, 69.3 id,  3.7 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 12351808+total, 31766724 free, 38258056 used, 53493308 buff/cache
KiB Swap:        0 total,        0 free,        0 used. 80590104 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
57364 harlylei  20   0  263732   1392    928 S 379.0  0.0  18:45.06 testperf
```

2.进入top之后，默认是按CPU消耗对程序排序，如果想按程序消耗的物理内存排序，以定位出哪些程序最消耗内存，则在top的结果输出里输入M

```
top - 09:49:51 up 31 days, 22:33, 123 users,  load average: 6.16, 6.41, 6.43
Tasks: 1325 total,   1 running, 1304 sleeping,  12 stopped,   8 zombie
%Cpu(s):  8.8 us,  1.3 sy,  0.0 ni, 89.1 id,  0.6 wa,  0.0 hi,  0.1 si,  0.0 st
KiB Mem : 19592467+total,   683412 free, 56644236 used, 13859702+buff/cache
KiB Swap:        0 total,        0 free,        0 used. 13331316+avail Mem
                                虚拟内存  物理内存
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 1933 root      20   0   52.4g  10.8g  34852 S   0.7  5.8 173:56.82 meta-cluster-se
19772 harlylei  20   0   29.2g   5.3g      0 S   0.3  2.8   2192:20 mysqld
10789 tdsql     20   0   12.6g   4.9g   8592 S   0.3  2.6 698:15.68 mysqld
76353 dongzhi   20   0   22.3g   4.7g   9616 S   0.3  2.5  15:31.30 java
```

CPU负载分析-top

技术创造未来 科技TEG

top的常见用法

3.top执行之后，默认情况下每隔3秒输出一次最新的统计信息，这个时间间隔是可以调整的，比如要定位毛刺，可以输入d,然后再输入0.1，表示每隔0.1秒刷新一次统计信息

4.如上面所示，发现testperf消耗了380%的cpu，意味着大概消耗了3.8个逻辑cpu，但是想具体看到分别是哪些线程在消耗CPU，则在top里输入H，则按各个子线程显示详细信息，如下，发现有3个跑满CPU的线程，一个跑了78.7%的线程，一个跑了4.3%的线程（状态为D，等待IO），然后针对具体的线程可以再做分析，见后面其他工具的分析

```
top - 11:21:08 up 140 days, 20:07, 14 users,  load average: 6.73, 6.30, 6.21
Threads:   7 total,   4 running,   3 sleeping,   0 stopped,   0 zombie
%Cpu(s): 18.6 us,  7.6 sy,  0.0 ni, 70.0 id,  3.8 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 12351808+total, 29116556 free, 38286048 used, 56115484 buff/cache
KiB Swap:        0 total,        0 free,        0 used. 80471984 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
73197 harlylei  20   0  198196   1136    932 R 98.7  0.0   1:17.29 testperf
73198 harlylei  20   0  198196   1136    932 R 98.3  0.0   1:17.26 testperf
73199 harlylei  20   0  198196   1136    932 R 98.3  0.0   1:17.19 testperf
73200 harlylei  20   0  198196   1136    932 R 78.7  0.0   1:01.59 testperf
73201 harlylei  20   0  198196   1136    932 D  4.3  0.0   0:03.42 testperf
73196 harlylei  20   0  198196   1136    932 S  0.0  0.0   0:00.00 testperf
73536 harlylei  20   0  198196   1136    932 S  0.0  0.0   0:00.00 testperf
```

CPU负载分析-top

技术创造未来 科技TEG

top的常见用法

5.输入1则按显示每个cpu的统计信息

```
top - 11:23:56 up 140 days, 20:10, 14 users,  load average: 5.75, 6.09, 6.15
Threads:   7 total,   4 running,   3 sleeping,   0 stopped,   0 zombie
%Cpu0  : 52.0 us,  5.7 sy,  0.0 ni, 40.5 id,  1.7 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  : 54.0 us,  4.4 sy,  0.0 ni, 41.6 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  : 27.0 us,  3.7 sy,  0.0 ni, 69.0 id,  0.3 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  : 11.2 us,  7.1 sy,  0.0 ni, 81.4 id,  0.3 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  : 11.6 us,  6.3 sy,  0.0 ni, 80.1 id,  1.7 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu5  : 19.3 us,  7.1 sy,  0.0 ni, 73.6 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  :  3.3 us,  6.7 sy,  0.0 ni, 90.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu7  :  4.4 us, 12.3 sy,  0.0 ni, 82.9 id,  0.3 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu8  : 12.1 us,  2.7 sy,  0.0 ni, 83.9 id,  1.3 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu9  : 14.8 us,  7.4 sy,  0.0 ni, 77.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu10 : 23.4 us,  4.3 sy,  0.0 ni, 72.2 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu11 : 14.7 us,  3.0 sy,  0.0 ni, 82.0 id,  0.3 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu12 : 13.4 us,  4.7 sy,  0.0 ni, 81.9 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu13 :  5.7 us,  5.7 sy,  0.0 ni, 88.3 id,  0.3 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu14 : 14.4 us, 10.1 sy,  0.0 ni, 75.5 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu15 : 17.4 us, 10.1 sy,  0.0 ni, 72.1 id,  0.3 wa,  0.0 hi,  0.0 si,  0.0 st
```

CPU负载分析-strace

从top观察到内核态用占用比较大的CPU比例，或者想分析系统调用的时耗，则可以借助strace工具来分析

strace -tttT -v -s10 -f -p pid

```
[pid 73201] 1555555668.411438 pread(3,  <unfinished ...>
[pid 73200] 1555555668.411507 nanosleep({0, 10000},  <unfinished ...>
[pid 73201] 1555555668.411598 <... pread resumed> "\0\0\0\0\0\0\0\0\0\0"..., 4096, 1795727360) = 4096 <0.000156>
[pid 73200] 1555555668.411603 <... nanosleep resumed> NULL) = 0 <0.000093>
[pid 73201] 1555555668.411615 pread(3, "\0\0\0\0\0\0\0\0\0\0"..., 4096, 1021042688) = 4096 <0.000154>
[pid 73201] 1555555668.411784 pread(3,  <unfinished ...>
```

进程可能有很多子线程，这里能展示各个子线程的系统调用，每个系统调用的执行时间，参数等非常详细。

如果想分析具体的某个子线程，比如调用pread的线程，则去掉-f参数，指定要分析的进程ID，如：
strace -tttT -v -s10 -p 25327

也可以指定只监控某些系统调用，如监控内存分配,则执行：
strace -e brk,mmap -f -p 25327

还有很多其他的参数，详细参考手册

对性能有非常大的影响，尽量不要在现网执行，现网执行需要慎重考虑

分析CPU瓶颈-perf

1. 类似于top分析热点函数

perf top -g -p `pidof testop`

```
Samples: 28K of event 'cpu-clock', Event count (approx.): 6231777341
+  87.80%  [vdso]                [.] 0x0000000000000f5b
-  11.06%  testop                [.] costLowCpuFun()
     costLowCpuFun()
+   0.45%  testop                [.] costCpuFun()
+   0.44%  testop                [.] gettimeofday@plt
+   0.10%  [kernel]              [k] _raw_spin_unlock_irqrestore
+   0.03%  [kernel]              [k] retint_careful
+   0.01%  libc-2.17.so          [.] usleep
+   0.01%  [kernel]              [k] hrtimer_init
+   0.01%  libpthread-2.17.so    [.] pthread_mutex_lock
```

分析CPU瓶颈-perf

2. 性能分析结果保存起来再分析

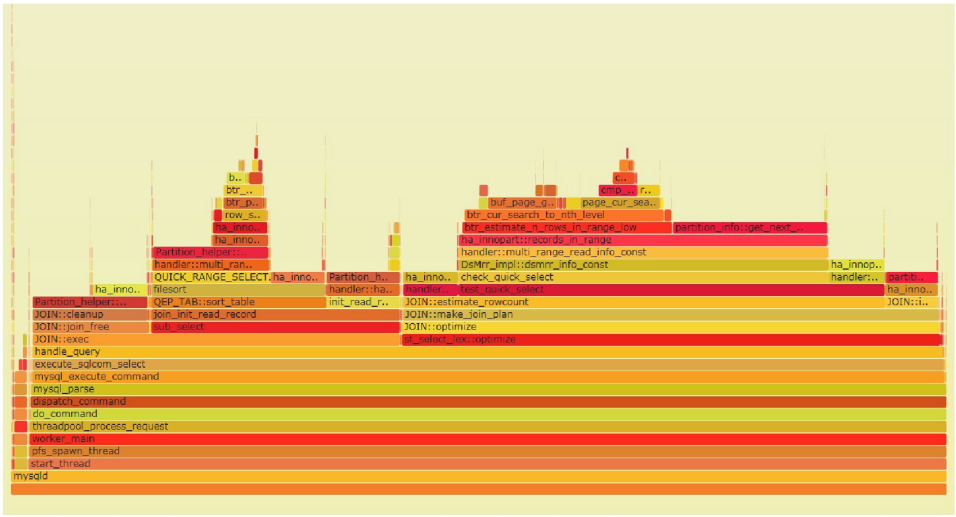
1.1) perf record –F 99 -g -p `pidof testperf` -- sleep 60，执行一段时间后退出

[perf record: Woken up 211 times to write data]
[perf record: Captured and wrote 53.495 MB perf.data (~2337219 samples)]

1.2) perf report输出热点函数信息

```
Samples: 799K of event 'cpu-clock', Event count (approx.): 199756250000
+  87.68%  testop  [vdso]                [.] 0x0000000000000e98
+  11.10%  testop  testop                [.] costLowCpuFun()
+   0.49%  testop  testop                [.] costCpuFun()
+   0.48%  testop  testop                [.] gettimeofday@plt
+   0.08%  testop  [kernel.kallsyms]     [k] _raw_spin_unlock_irqrestore
+   0.03%  testop  [kernel.kallsyms]     [k] retint_careful
+   0.01%  testop  [kernel.kallsyms]     [k] system_call_after_swapgs
+   0.01%  testop  libc-2.17.so          [.] usleep
+   0.01%  testop  [kernel.kallsyms]     [k] hrtimer_init
```

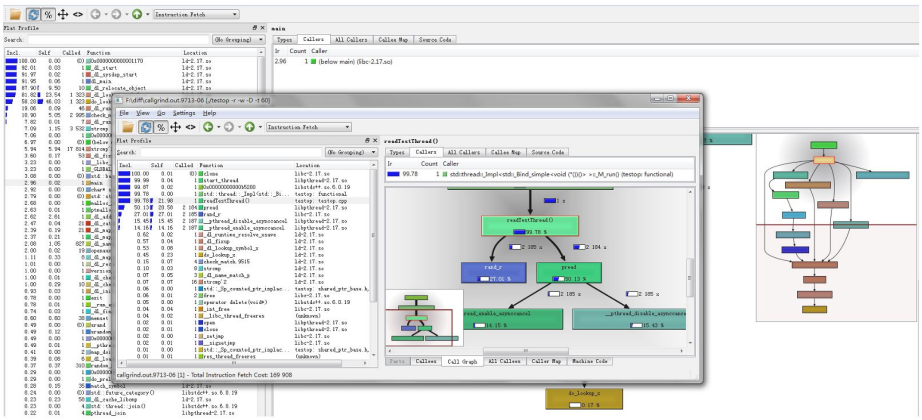
精确到函数的性能分析-火焰图



通过valgriand分析程序性能

valgrind --tool=callgrind --separate-threads=yes ./testop -r -w -D -t 60

让程序执行60秒后退出，将执行的统计信息保存到了callgrind.out.****文件中，然后用KcacheGrind分析



对程序堆栈进行快速分析

分析程序执行过程中各个线程的堆栈情况

1.pstack打印程序的堆栈到文件

pstack `pidof testperf` >p

2.通过pt-pmp对堆栈信息进行汇总观察

pt-pmp p

```
[root@node05 /data1/home/harlylei]# pt-pmp p
      3 gettimeofday,costCpuFun,libstdc++::??(libstdc++.so.6),start_thread(libpthread.so.0),clone(libc.so.6)
      1 pthread_join(libpthread.so.0),std::thread::join(libstdc++.so.6),main
      1 pread64(libpthread.so.0),readTestThread,libstdc++::??(libstdc++.so.6),start_thread(libpthread.so.0),clone(libc.so.6)
      1 nanosleep(libc.so.6),usleep(libc.so.6),costLowCpuFun,libstdc++::??(libstdc++.so.6),start_thread(libpthread.so.0),clone(libc.so.6)
```

cpu问题分析总结

粗看系统负载情况： uptime， vmstat	开发测试环境+线上环境
机器实时的CPU资源消耗详细情况 top	开发测试环境+线上环境
分析系统调用: strace	开发测试环境
分析函数的热点：perf，可以输出成火焰图便于直观观察	开发测试环境+线上环境
分析各个线程的资源消耗情况，线程内的函数消耗情况：valgriand	开发测试环境
分析程序各个线程的堆栈执行情况：pstack & pt-pmp	开发测试环境

目录

内存问题的分析与优化

1.常见内存问题的分类

内存重复释放

释放不是动态分配的内存，如栈上的内存

读写已经释放的内存

读写没有被分配的内存

内存耗尽，分配失败

读写内存超越边界，尤其是对于栈上的局部变量数组

内存泄漏

malloc,free与new,delete混用

new []分配,delete释放

变量未初始化

频繁的内存分配引起碎片和性能问题

2.开发要求

尽量避免手工管理内存，用shared_ptr，unique_ptr等智能指针，这个是对于C++开发人员必须掌握

3.内存问题诊断工具-valgriand的介绍

valgrind --tool=memcheck --leak-check=full --show-reachable=yes ./testperf -t120 >memory.txt 2>&1

最后打开memory.txt文件进行分析

```
==84423==
==84423== HEAP SUMMARY:
==84423==     in use at exit: 120,832 bytes in 118 blocks
==84423==   total heap usage: 143 allocs, 25 frees, 127,030 bytes allocated
==84423==
==84423== 120,832 bytes in 118 blocks are definitely lost in loss record 1 of 1
==84423==    at 0x4C29E83: malloc (vg_replace_malloc.c:299)
==84423==    by 0x40203A: memoryLeak() (testperf.cpp:168)
==84423==    by 0x40556C: void std::_Bind_simple<void (*())()>::_M_invoke<>(std::_Index_tuple<>) (functional:1732)
==84423==    by 0x40547C: std::_Bind_simple<void (*())()>::operator()() (functional:1720)
==84423==    by 0x4053CB: std::thread::_Impl<std::_Bind_simple<void (*())()> >::_M_run() (thread:115)
==84423==    by 0x510821F: ??? (in /usr/lib64/libstdc++.so.6.0.19)
==84423==    by 0x4E3EE24: start_thread (in /usr/lib64/libpthread-2.17.so)
==84423==    by 0x596C35C: clone (in /usr/lib64/libc-2.17.so)
==84423==
==84423== LEAK SUMMARY:
==84423==    definitely lost: 120,832 bytes in 118 blocks
==84423==    indirectly lost: 0 bytes in 0 blocks
==84423==      possibly lost: 0 bytes in 0 blocks
==84423==    still reachable: 0 bytes in 0 blocks
==84423==         suppressed: 0 bytes in 0 blocks
==84423==
==84423== For counts of detected and suppressed errors, rerun with: -v
==84423== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```


4.为多线程优化的通用内存库-jemalloc/tcmalloc

a.多线程

b.程序内部有大量内存分配，如大量使用STL,string等

c.需要做性能和内存走profile等场景

5.free命令观察系统内存使用情况free -hw

```
[harlylei@node05 ~]$free -hw
              total        used        free      shared     buffers       cache   available
Mem:           117G         18G        1.3G        4.1G        362M         97G         95G
Swap:            0B          0B          0B
```

Mem:代表应用能够使用的总物理内存，Swap代表机器swap分区的内存（时耗敏感引用不建议使用，或者交换分区放入SSD）
total：总的物理内存
used :已使用的内存，如应用程序和内核自身，不包括文件系统cache（total - free - buffers - cache)
free ：没有使用的内存， 不包括buffers/cache,所以必要的时候OS还能释放出更多的可用内存
shared: 共享内存，主要被用于shmget和tmpfs文件系统
buffers:文件系统是以块的方式进行读写操作，磁盘数据的缓存（直接读写块设备则缓存磁盘数据，读写文件则量很少）
cache:文件的页面缓存，提升读写效率
available:应用程序能真正使用的内存，一般为free+cache中能被回收的内存

5.主动清理page cache

刷脏页 sync

清理page cache: echo 1 >/proc/sys/vm/drop_caches ;

```
[root@node05 ~]# free -hw
              total        used        free      shared     buffers       cache   available
Mem:           117G         18G        2.3G        4.1G        368M         96G         95G
Swap:            0B          0B          0B

[root@node05 ~]# echo 1 >/proc/sys/vm/drop_caches ;

[root@node05 ~]# free -hw
              total        used        free      shared     buffers       cache   available
Mem:           117G         18G         91G        4.1G         23M        8.1G         95G
Swap:            0B          0B          0B
```

6.关闭swap

swapoff -a

内存问题分析总结

1.C++开发的时候必须掌握智能指针

2.熟练使用valgriand分析内存泄漏

3.了解多线程内存库jemalloc/tcmalloc，可以熟悉下jemalloc源代码

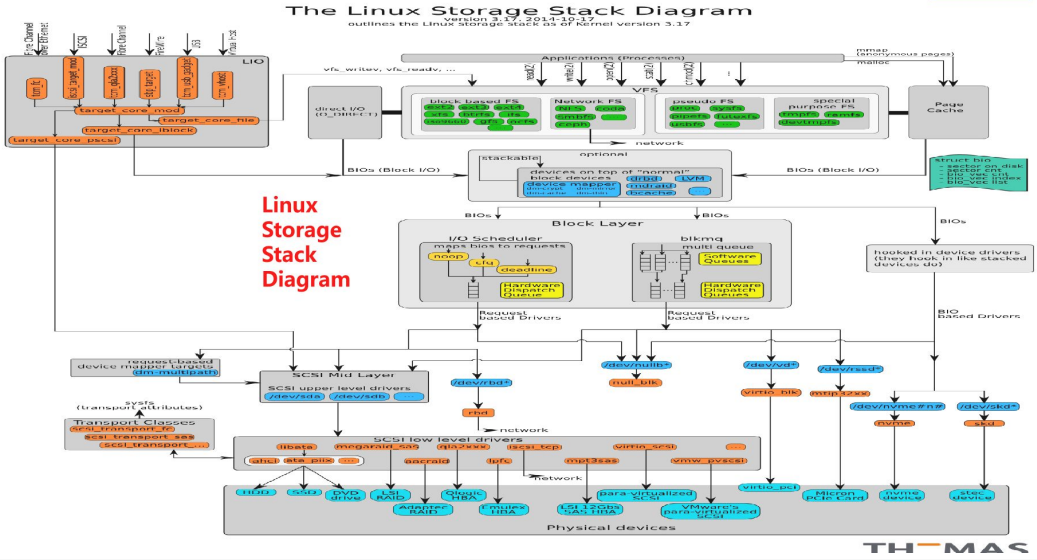
4.free的输出结果会解读

5.建议关闭swap

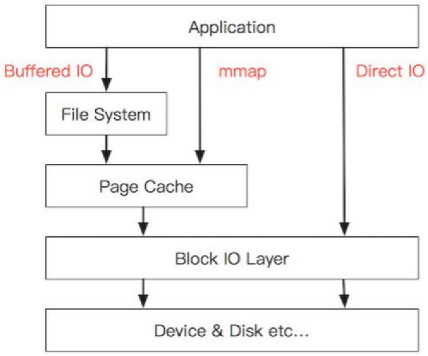
6.熟悉/proc/sys/vm/下的参数，比如控制刷脏页的频率，脏页占有量，swappinness等

目录

IO问题的分析



IO问题的分析



IO问题的分析

几个重要的概念

顺序IO

顺序读写文件，能很好地利用预读，对于机械磁盘也不用做磁头寻道，所以是性能最好的读写模式，但是一般适用于日志类场景

随机IO

随机读写，机械磁盘性能会非常差，如果是这种场景建议用SSD。当然应用层也会做各种努力尽量会将随机IO改成顺序IO

fsync刷盘

为了数据安全，每次写入完成通过fsync做强制刷盘操作，避免机器突然掉电能情况丢失数据，但是这个操作对吞吐和响应延时有较大的影响，一般会采用一些批量合并fsync的模式做优化

direct io

绕过page cache，直接对设备进行读写，一般性能不如带page cache，适用于在业务层做了cache,比如数据库

aio:异步IO

读写操作异步化，一般也是direct io模式，编程会比较复杂，对于大部分应用不要采用，一般也是适用于数据库场景

IO问题的分析-测试磁盘IO的性能

dd命令简单测试

测试写，direct&sync，顺序写模式：rm testfile; dd if=/dev/zero of=testfile bs=4k count=30000 oflag=direct,sync
122880000 bytes (123 MB) copied, 7.85962 s, 15.6 MB/s

测试写，direct，顺序写模式：rm testfile; dd if=/dev/zero of=testfile bs=4k count=30000 oflag=direct
122880000 bytes (123 MB) copied, 2.05195 s, 59.9 MB/s

测试写，page cache顺序写模式：rm testfile; dd if=/dev/zero of=testfile bs=4k count=30000
122880000 bytes (123 MB) copied, 0.167806 s, 732 MB/s

测试读，page cache顺序读模式：dd if=testfile of=/dev/null bs=4k count=30000
122880000 bytes (123 MB) copied, 0.123824 s, 992 MB/s

测试读，direct顺序读模式：dd if=testfile of=/dev/null bs=4k count=30000 iflag=direct
122880000 bytes (123 MB) copied, 1.77522 s, 69.2 MB/s

多线程随机读写，aio等方式如何测试？

IO问题的分析-测试磁盘IO的性能

fio命令测试随机读性能

1. 16线程采用异步io，队列深度为1的随机读
   fio --thread --directory=. --size=1000M --iodepth 1 --iodepth_batch_submit=1 -rw=randread -bs=4k -ioengine libaio -direct=1 -numjobs=16 --name testfio

```
fio-3.13
Starting 16 threads

Jobs: 16 (f=16): [r(16)][35.3%][r=321MiB/s][r=82.2k IOPS][eta 00m:33s]
```

2. 16线程采用异步io，队列深度为16,batch为8的随机读
   fio --thread --directory=. --size=1000M --iodepth 16 --iodepth_batch_submit=8 -rw=randread -bs=4k -ioengine libaio -direct=1 -numjobs=16 --name testfio

```
fio-3.13
Starting 16 threads

Jobs: 16 (f=16): [r(16)][29.0%][r=530MiB/s][r=136k IOPS][eta 00m:22s]
```

3. 16线程采用pread普通IO随机读
   fio --thread --directory=. --size=1000M -rw=randread -bs=4k -ioengine psync -direct=1 -numjobs=16 --name testfio

```
fio-3.13

Starting 16 threads
Jobs: 16 (f=16): [r(16)][19.2%][r=315MiB/s][r=80.7k IOPS][eta 00m:42s]
```

IO问题的分析-测试磁盘IO的性能

fio命令测试随机写性能

1. 16线程采用异步io，队列深度为1的随机写
   fio --thread --directory=. --size=1000M --iodepth 1 --iodepth_batch_submit=1 -rw=randwrite -bs=4k -ioengine libaio -direct=1 -numjobs=16 --name testfio

```
fio-3.13
Starting 16 threads
Jobs: 16 (f=16): [w(16)][2.2%][w=50.0MiB/s][w=12.8k IOPS][eta 04m:27s]
```

2. 16线程采用异步io，队列深度为16,batch为8的随机读
   fio --thread --directory=. --size=1000M --iodepth 16 --iodepth_batch_submit=8 -rw=randwrite -bs=4k -ioengine libaio -direct=1 -numjobs=16 --name testfio

```
fio-3.13
Starting 16 threads
Jobs: 16 (f=16): [w(16)][3.6%][w=49.8MiB/s][w=12.7k IOPS][eta 05m:21s]
```

3. 16线程采用pwrite普通IO随机写
   fio --thread --directory=. --size=1000M -rw=randwrite -bs=4k -ioengine psync -direct=1 -numjobs=16 --name testfio

```
fio-3.13
Starting 16 threads
Jobs: 16 (f=16): [w(16)][2.2%][w=45.4MiB/s][w=11.6k IOPS][eta 04m:30s]
```

IO问题的分析-观测磁盘的IO状况

iostat工具

```
fio --thread --directory=. --size=1000M --iodepth 16 --iodepth_batch_submit=8 -rw=randrw -bs=4k -ioengine libaio -direct=1 -numjobs=16 --name testfio
#构造随机读写的IO
Jobs: 16 (f=16): [m(16)][8.9%][r=39.1MiB/s,w=39.0MiB/s][r=10.0k,w=9988 IOPS]
```

通过iostat命令进行观察： iostat -x -m -t 1

```
04/11/19 17:40:36
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           2.61    0.00    6.75   22.13    0.04   68.47

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
vda               0.00   579.00 9491.00 10165.00    37.07    48.21     8.89   149.80    7.67    7.63    7.71   0.05 100.00
```

rrqm/s: 设备队列里每秒合并读请求的次数，读完全没法合并，非常随机
wrqm/s:设备队列里每秒合并写请求的次数，写也基本没被合并，非常随机
r/s: 合并之后设备每秒实际提供的读次数，读的IOPS为9491
w/s:合并之后设备每秒实际提供的写次数,写的IOPS为10165，其他进程有写，所以比读稍微多了点
rMB/s: 设备提供的每秒读的数据大小, 37.07M/s
wMB/s: 设备提供的每秒写的数据大小,48.21M/s
avgrq-sz: 设备平均每次操作的扇区数目，每次IO操作8.89个扇区
avgqu-sz:设备队列里保存的IO数目的平均值，表示IO队列里平均积攒了149.80，值越大表示阻塞越严重，超过1一般就表示IO请求有堆积，设备处于饱和状态

IO问题的分析-观测磁盘的IO状况

await: IO进入队列到处理完成总时间，单位为毫秒,7.76表示每次IO平均需要7.76毫秒才被处理好
r_await:读IO进入队列到处理完成总时间，单位为毫秒,7.63表示每次读IO平均需要7.63毫秒才被处理好
w_await:写IO进入队列到处理完成总时间，单位为毫秒,7.71表示每次写IO平均需要7.71毫秒才被处理好
svctm:每次IO的平均处理时间,单位为毫秒,0.05表示每次IO在0.05毫秒处理完成，但是不要忽略前面的等待时间哦
%util: 设备的IO利用率，当接近100%的时候就已经饱和了，超过60%即表示已经比较繁忙了。

顺序读写产生的IO统计：
fio --thread --directory=. --size=1000M --iodepth 16 --iodepth_batch_submit=8 -rw=readwrite -bs=4k -ioengine libaio -direct=1 -numjobs=16 --name testfio

```
fio-3.13
Starting 16 threads
Jobs: 16 (f=16): [M(16)][28.0%][r=330MiB/s,w=329MiB/s][r=84.6k,w=84.2k IOPS][eta 00m:18s]
```

```
04/11/19 17:54:01
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           3.96    0.00   19.36    2.60    0.04   74.04

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
vda           32201.00 33442.00 51219.00 51709.00   325.88   333.29    13.12   132.75    1.28    1.12    1.45   0.01 100.40
```

IO问题的分析-观测进程IO和句柄

1.iotop观察各个进程的IO消耗情况

```
Total DISK READ :      39.09 M/s | Total DISK WRITE :      42.08 M/s
Actual DISK READ:      39.02 M/s | Actual DISK WRITE:      43.76 M/s
  TID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN     IO>    COMMAND
31433 be/3 root        0.00 B/s   27.85 K/s  0.00 % 93.59 % [jbd2/vda2-8]
89620 be/4 harlylei    2.26 M/s    2.21 M/s  0.00 % 60.15 % fio --thread --directory=. --size=1000M --i~ libaio -direct=1 -numjobs=16 --name testfio
89633 be/4 harlylei    2.50 M/s    2.41 M/s  0.00 % 60.07 % fio --thread --directory=. --size=1000M --i~ libaio -direct=1 -numjobs=16 --name testfio
89625 be/4 harlylei    2.48 M/s    2.33 M/s  0.00 % 58.49 % fio --thread --directory=. --size=1000M --i~ libaio -direct=1 -numjobs=16 --name testfio
89624 be/4 harlylei    2.52 M/s    2.32 M/s  0.00 % 56.07 % fio --thread --directory=. --size=1000M --i~ libaio -direct=1 -numjobs=16 --name testfio
89618 be/4 harlylei    2.51 M/s    2.28 M/s  0.00 % 55.98 % fio --thread --directory=. --size=1000M --i~ libaio -direct=1 -numjobs=16 --name testfio
```

2.通过proc文件系统查看进程句柄的情况
ls /proc/87091/fd -l

```
[root@node05 ~]# ls /proc/87091/fd -l
total 0
lr-x------ 1 tdsql users 64 Apr  1 12:04 0 -> /dev/null
lrwx------ 1 tdsql users 64 Apr  1 12:04 1 -> /data/tdsql_run/15023/gateway/log/update_instance_15023
l-wx------ 1 tdsql users 64 Apr  1 12:04 10 -> /data/tdsql_run/15023/gateway/log/slow_sql_instance_15023.2019-04-12.0
l-wx------ 1 tdsql users 64 Apr  1 12:04 100 -> pipe:[3347861033]
lrwx------ 1 tdsql users 64 Apr  1 12:04 101 -> anon_inode:[eventpoll]
lrwx------ 1 tdsql users 64 Apr  1 12:04 102 -> socket:[3347861034]
lrwx------ 1 tdsql users 64 Apr  1 12:04 103 -> socket:[3347861035]
lr-x------ 1 tdsql users 64 Apr  1 12:04 104 -> pipe:[3347861036]
```

IO问题的分析-观测进程IO和句柄

3.通过lsof命令查看进程句柄的情况
lsof -p 87091

```
mysql-pro 87091 tdsql  135u     unix 0xffff88060cf0ec80      0t0 3347861053 socket
mysql-pro 87091 tdsql  136r     FIFO                0,8      0t0 3347861054 pipe
mysql-pro 87091 tdsql  137w     FIFO                0,8      0t0 3347861054 pipe
mysql-pro 87091 tdsql  138u  a_inode                0,9        0       6991 [eventpoll]
mysql-pro 87091 tdsql  139u     unix 0xffff88060cf0d780      0t0 3347861056 socket
mysql-pro 87091 tdsql  140r     FIFO                0,8      0t0 3347861057 pipe
mysql-pro 87091 tdsql  141w     FIFO                0,8      0t0 3347861057 pipe
mysql-pro 87091 tdsql  142w     FIFO                0,8      0t0  714133898 pipe
mysql-pro 87091 tdsql  143r     FIFO                0,8      0t0  713890638 pipe
mysql-pro 87091 tdsql  144w     FIFO                0,8      0t0  713890638 pipe
mysql-pro 87091 tdsql  145u     IPv4         1369795799      0t0        TCP node05:58329->node03:eforward (ESTABLISHED)
mysql-pro 87091 tdsql  146u     IPv4          713679433      0t0        TCP node05:53826->node03:eforward (ESTABLISHED)
```

IO问题分析总结

1.IO执行栈

2.IO相关的重要概念，如顺序IO，随机IO，同步，异步等

3.使用dd和fio测试IO的性能

4.iostat输出结果会解读

5.iotop观察具体进程IO的消耗情况

6.通过proc文件系统和lsof分析句柄的使用，定位是否有句柄泄漏，某个句柄具体对应哪个文件

目录

网络问题的分析

1.查看机器的网络连接信息netstat
netstat -apn | grep 4100

```
[harlylei@node01 ~]$netstat -apn | grep 4100
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
tcp        0      0 9.30.17.168:4100        0.0.0.0:*               LISTEN      1149/mysqld
tcp        0      0 9.30.17.168:14100       0.0.0.0:*               LISTEN      1149/mysqld
tcp        0      0 9.30.17.168:4100        9.30.17.168:60328       ESTABLISHED 1149/mysqld
tcp        0      0 9.30.17.168:4100        9.30.17.168:60329       ESTABLISHED 1149/mysqld
tcp        0      0 9.30.17.168:4100        9.30.17.168:38147       ESTABLISHED 1149/mysqld
tcp        0     65 9.30.17.168:38148       9.30.17.168:4100        ESTABLISHED 61922/./checkmysqlp
tcp        0      0 9.30.17.168:4100        9.30.17.168:38149       ESTABLISHED 1149/mysqld
tcp        0      0 9.30.17.168:4100        9.30.17.168:38146       ESTABLISHED 1149/mysqld
tcp        0     65 9.30.17.168:38147       9.30.17.168:4100        ESTABLISHED 61922/./checkmysqlp
tcp        0     65 9.30.17.168:38150       9.30.17.168:4100        ESTABLISHED 61922/./checkmysqlp
tcp        0      0 9.30.17.168:4100        9.30.17.168:38150       ESTABLISHED 1149/mysqld
tcp        0     65 9.30.17.168:38146       9.30.17.168:4100        ESTABLISHED 61922/./checkmysqlp
```

2. lsof -i :36000 //查看开了这个端口的进程

```
[harlylei@node01 ~]$lsof -i :36000
COMMAND   PID     USER   FD   TYPE     DEVICE SIZE/OFF NODE NAME
ssh      1554 harlylei    3u  IPv4  330688934      0t0  TCP node01:56187->node01:36000 (ESTABLISHED)
ssh      6704 harlylei    3u  IPv4  334017744      0t0  TCP node01:38789->node01:36000 (ESTABLISHED)
ssh      9865 harlylei    3u  IPv4 2155615865      0t0  TCP node01:36481->node01:36000 (ESTABLISHED)
ssh     18271 harlylei    3u  IPv4 2146228482      0t0  TCP node01:43369->node01:36000 (ESTABLISHED)
```

网络问题的分析

3.查看网卡流量和包量
sar -n DEV -n EDEV 1

```
[harlylei@node01 ~]$sar  -n  DEV -n EDEV 1
Linux 3.10.107-1-tlinux2-0047 (node01)  04/19/19        _x86_64_        (80 CPU)

09:52:40        IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s
09:52:41         eth0      0.00      0.00      0.00      0.00      0.00      0.00      0.00
09:52:41         eth1   1208.00    951.00    196.51    233.35      0.00      0.00      0.00
09:52:41           lo    207.00    207.00     53.04     53.04      0.00      0.00      0.00

09:52:40        IFACE   rxerr/s   txerr/s    coll/s  rxdrop/s  txdrop/s  txcarr/s  rxfram/s  rxfifo/s  txfifo/s
09:52:41         eth0      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
09:52:41         eth1      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
09:52:41           lo      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
```

IFACE:网卡设备名字
rxpck/s:每秒收到包的数量
Txpck/s:每秒发包的数量
rxkB/s:每秒收到了多少kB的包
txkB/s:每秒发送了多少kB的包

网络问题的分析

3.数据传输问题诊断利器tcpdump

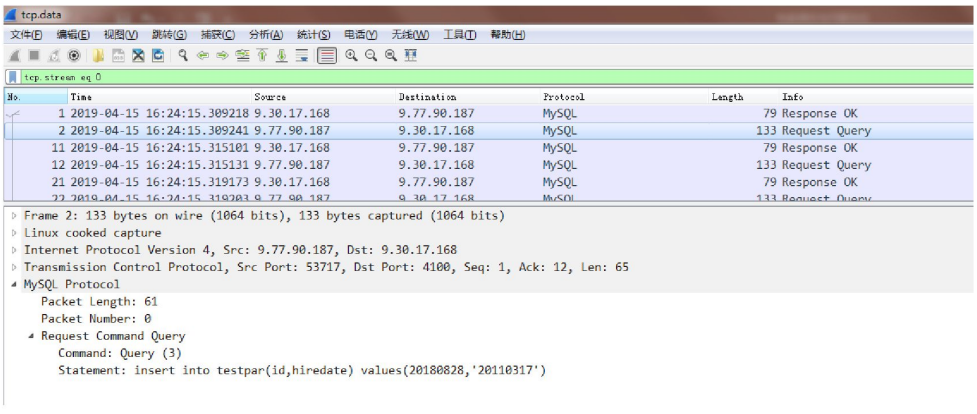
tcpdump -AAXn -s1000 -iany port 4100

```
16:21:31.845022 IP 9.30.17.168.igo-incognito > 9.77.90.187.53809: Flags [P.], seq 1844:1921, ack 8806, win 122, options [nop,nop,TS val 674101661 ec
r 3024944144], length 77
        0x0000:  4508 0081 6924 4000 3f06 537d 091e 11a8  E...i$@.?.S}....
        0x0010:  094d 5abb 1004 d231 80e7 9ebe 6ce2 bc49  .MZ....1....l..I
        0x0020:  8018 007a 68ca 0000 0101 080a 282d f99d  ...zh.......(-..
        0x0030:  b44c fc10 4900 0001 ff26 0423 3233 3030  .L..I....&.#2300
        0x0040:  3044 7570 6c69 6361 7465 2065 6e74 7279  0Duplicate.entry
        0x0050:  2027 3230 3131 3038 3233 2d32 3031 302d  .'20110823-2010-
        0x0060:  3035 2d30 3720 3030 3a30 303a 3030 2720  05-07.00:00:00'.
        0x0070:  666f 7220 6b65 7920 2750 5249 4d41 5259  for.key.'PRIMARY
        0x0080:  2700 0000 0000 0000 0000 0000 0000 0000  '...............
        0x0090:  00                                       .
16:21:31.845122 IP 9.77.90.187.53809 > 9.30.17.168.igo-incognito: Flags [P.], seq 8806:8871, ack 1921, win 14, options [nop,nop,TS val 3024944145 ec
r 674101661], length 65
        0x0000:  4508 0075 4197 4000 4006 7a16 094d 5abb  E..uA.@.@.z..MZ.
        0x0010:  091e 11a8 d231 1004 6ce2 bc49 80e7 9f0b  .....1..l..I....
        0x0020:  8018 000e 7f35 0000 0101 080a b44c fc11  .....5.......L..
        0x0030:  282d f99d 3d00 0000 0369 6e73 6572 7420  (-..=....insert.
        0x0040:  696e 746f 2074 6573 7470 6172 2869 642c  into.testpar(id,
        0x0050:  6869 7265 6461 7465 2920 7661 6c75 6573  hiredate).values
        0x0060:  2832 3031 3531 3031 312c 2732 3031 3030  (20151011,'20100
        0x0070:  3530 3727 2900 0000 0000 0000 0000 0000  507').........
        0x0080:  0000 0000 00                             .....
16:21:31.847215 IP 9.30.17.168.igo-incognito > 9.77.90.187.53809: Flags [P.], seq 1921:1932, ack 8871, win 122, options [nop,nop,TS val 674101661 ec
r 3024944145], length 11
        0x0000:  4508 003f 6925 4000 3f06 53be 091e 11a8  E..?i%@.?.S.....
```

网络问题的分析

4.tcpdump的输出倒出到文件通过wireshark进行分析

tcpdump -AAXn -s1000 -iany port 4100 -w tcp.data



网络问题分析总结

1.通过netstat查看网络连接状况

2.lsof分析具体端口被谁占用

3.通过sar查看网卡流量

4.熟练掌握tcpdump& wireshark进行网络包的分析

目录

CPU,内存，IO，网络综合展示工具

dstat

1. dstat -t -a --proc-count -i -l -m -p --aio --disk-util 10

```
[harlylei@node05 ~/data]$dstat -t -a --proc-count -i -l -m -p  --aio --disk-util 10
Terminal width too small, trimming output.
----system---- ----total-cpu-usage---- -dsk/total- -net/total- ---paging-- ---system-- proc ----interrupts--- ---load-avg--- ---procs--- async vda->
     time     |usr sys idl wai hiq sig| read  writ| recv  send|  in   out | int   csw |tota|  41   43   44 | 1m   5m  15m |run blk new| #aio|util>
              |  3   3  94   0   0   0|  58k 4388k|   0     0 |  2B   9B|  1k   5k| 662| 304   142     0 |2.30 2.33 2.25|0.0   0 264|  12k|6.36>
              |  4   7  89   0   0   0| 819B 432k| 20k  218k|   0     0 |  3k   8k| 666| 429  1133     0 |2.49 2.37 2.26|3.3   0 381|  12k|2.80>
              |  4   7  89   0   0   0| 410B 577k| 19k  258k|   0     0 |  3k   8k| 670| 436  1172     0 |2.34 2.34 2.25|3.3   0 438|  12k|2.36>
15-04 16:50:53|  5   8  87   0   0   0|4096B 6688k| 26k  296k|   0     0 |  3k   8k| 666| 526  1499     0 |2.34 2.34 2.25|8.0   0 342|  12k|0.80>
```

2. dstat -t --disk-util -s --fs --socket --tcp --udp --top-bio --top-cpu 10

```
[harlylei@node01 ~/txsql/build/mariadb/install]$./dstat2.sh
----system---- nvme-nvme-nvme-nvme-sda- ----swap--- --filesystem- ------sockets------ ----tcp-sockets---- --udp-- ----most-expensive---- -most-expensive-
     time     |util:util:util:util:util| used  free|files  inodes|tot tcp udp raw frg|lis act syn tim clo|lis act|  block i/o process  |  cpu process
15-04 16:59:22|1.02:1.25:1.03:1.22:5.84|    0     0 |29440   103k|  5   2  12   0   0| 56   3   0 331  12| 15   0|mysqld      1072B  406k|mysqld       0.2
15-04 16:59:32|5.68:4.72:6.92:5.68:28.8|    0     0 |30240   102k|  5   2  12   0   0| 56   3   0  --  11| 15   0|mysqld       392k 6043k|tdstore       20
15-04 16:59:42|7.88:6.88:6.76:5.84:31.0|    0     0 |30400   111k|  5   2  12   0   0| 56   3   1 893  11| 15   0|mysqld       324k 6414k|tdstore       21
15-04 16:59:52|5.60:5.60:5.96:4.48:27.6|    0     0 |30560   111k|  5   2  12   0   0| 56   3   0   1   9| 15   0|mysqld       258k 5419k|tdstore       21
15-04 17:00:02|6.96:4.56:5.56:4.56:33.0|    0     0 |30400   111k|  5   2  12   0   0| 56   3   0   1   9| 15   0|mysqld       264k 5496k|tdstore       21
15-04 17:00:12|6.16:5.40:6.72:5.28:31.5|    0     0 |30560   111k|  5   2  12   0   0| 56   3   1   2   7| 15   0|mysqld       334k 6127k|tdstore       20
15-04 17:00:22|6.24:6.00:7.00:6.08:30.3|    0     0 |31040   107k|  4   2  12   0   0| 56   2   0   2   7| 15   0|mysqld       284k 6010k|meta-cluster- 21
15-04 17:00:32|5.08:4.20:5.88:5.68:29.1|    0     0 |29920   108k|  4   2  12   0   0| 56   2   0   2   8| 15   0|mysqld       248k 4789k|meta-cluster- 50
15-04 17:00:42|6.24:5.68:5.00:3.76:29.4|    0     0 |29600   106k|  4   2  12   0   0| 56   2   1   2   8| 15   0|mysqld       310k 5426k|meta-cluster- 48
15-04 17:00:52|4.36:3.88:5.56:4.12:27.2|    0     0 |28800   105k|  4   2  12   0   0| 56   2   0   2   8| 15   0|mysqld       298k 5242k|meta-cluster- 48
```

程序调试工具gdb

gdb：常见的调试入口信息

1. gdb ./testperf 通过调试的方式启动程序
2. gdb -p 81369 对指定的进程进行调试
3. gdb testperf core-testperf-52784-1555319546 程序异常退出产生core，可以用来调试

core文件的tips:
1.执行程序之前先确保开启了core　　ulimit -c unlimited
2.man core查看更多关于控制core文件生成的规则

编译参数方便进行问题定位

-ggdb3

-O0 #0级别调试最方便，不过性能不是最佳，正式环境一般会用-O2或者-O3

Linux系统错误日志

对于系统异常，linux会打印日志，比如进程oom，磁盘，网络故障等等

因此不要忘了分析这个日志

vim /var/log/messages 分析故障时间点看是否有啥异常信息(需要root权限)

参考资源

《性能之巅-洞悉系统，企业与云计算》

http://www.brendangregg.com/

60秒快速分析linux性能

性能调优利器 -- 火焰图

https://github.com/brendangregg/FlameGraph

Valgrind+KcacheGrind实战性能优化

《Linux 多线程服务端编程：使用 muduo C++ 网络库》

The Valgrind Quick Start Guide

知识拓展

https://www.thomas-krenn.com/en/wiki/Linux_Storage_Stack_Diagram

Fio手册

Linux aio使用参考

100个gdb小技巧

课后作业

1.假如运营人员突然反馈你负责的系统响应变慢，并给出了具体的IP机器，如何分析问题

2.线上某个多线程程序的CPU突然有大量CPU消耗，如何定位到哪个线程或者函数最消耗资源

3.什么场景下可以使用火焰图

4.线上程序突然被重启了，如何分析重启的原因

5.开发的程序锁冲突严重，如何定位是哪个地方的锁引起的

6.如何查看机器脏页的数量，如何加快淘汰方式,page cache如何清理

7.内存泄漏用啥工具检测，检测的原理是什么

8.如何查看本机网卡是否有丢包现象

9.怎么查看本机是否time_wait状态的网络连接，如果存在大量这种连接，如何处理，处理方案是否存在风险

10.如何查看机器历史某一天IO的消耗情况，能查看哪些指标，时间精度是咋样的，查看是平均值还是峰值

11.举2-3个你觉得很重要，但是本课程又没有覆盖到的工具或者分析问题的方法