上海交通大學

SHANGHAI JIAOTONG UNIVERSITY

工程实践与科技创新 3-D 作业 4

虚拟化性能损耗对比测试及修改源码以降低损耗

姓名：薛春宇

学号：518021910698

完成时间：2020/11/5

虚拟化性能对比测试及修改源码以降低损耗

薛春宇 518021910698

1. 实验目的

(1) 与原生性能对比，通过测试发现虚拟化的性能损耗；

(2) 通过阅读文献等途径，修改开源软件的相关代码，降低虚拟化的性能损耗。

2. 实验内容

2.1 QEMU（5.1.0 版本）虚拟化性能损耗测试及对比

在本节中，我们利用 sysbench 这个 Linux 环境下的 benchmark 性能测试工具，对虚拟机的 CPU、线程、磁盘 IO、内存和 Mutex 五个性能指标进行压力测试。Sysbench 是一个模块化、跨平台、多线程基准测试工具，用于评估测试各种不同系统参数下的数据库负载情况。

在此基础上，我们选择三个对象进行上述测试，分别是原生虚拟机 (Ubuntu18.04 版本)，启动 KVM 模块的 QEMU 虚拟机（Ubuntu20.04 版本）和未启动 KVM 模块的 QEMU 虚拟机（Ubuntu20.04 版本），并对三个对象下的几个性能指标进行横向比较，同时得出两个对比结论：

(1) 原生虚拟机（在 VMware 上直接安装的一级虚拟机）和 QEMU 虚拟机（开启 KVM 模块的二级虚拟机）之间的性能对比；

(2) QEMU 安装二级虚拟机时是否开启 KVM 模块对虚拟机性能的影响。

2.1.1 Ubuntu 环境下 sysbench 工具的安装

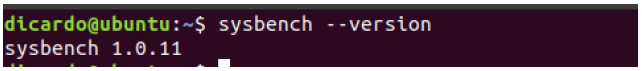
首先，我们需要安装 sysbench 的依赖包：

```
sudo apt-get -y install make automake libtool pkg-config libaio-dev vim-common
```

Sysbench 工具可以直接通过以下命令从 Linux 的 apt 库内下载安装：

```
sudo apt-get install sysbench
```

可以通过命令行查看 sysbench 版本来验证工具是否安装成功：



2.1.2 利用 sysbench 工具进行性能测试所需使用的指令

（1）CPU 测试：

```
sysbench --test=cpu --cpu-max-prime=2000 run
```

（2）线程测试：

```
sysbench  --test=threads --num-threads=500 --thread-yields=100 --thread-locks=4 run
```

（3）磁盘 IO 测试：

a. 准备阶段：生成需要的测试文件，完成后会在当前目录下生成很多小文件。

```
sysbench --test=fileio --num-threads=16 --file-total-size=2G --file-test-mode=rndrw prepare
```

b. 运行阶段：

```
sysbench --test=fileio --num-threads=20 --file-total-size=2G --file-test-mode=rndrw run
```

c. 清理测试时生成的文件：

```
sysbench --test=fileio --num-threads=20 --file-total-size=2G --file-test-mode=rndrw cleanup
```

（4）内存测试：

```
sysbench --test=memory --memory-block-size=8k --memory-total-size=1G run
```

（5）Mutex 测试：

```
sysbench --test=mutex --num-threads=4 --mutex-num=2000 --mutex-locks=10000 --mutex-loops=5000 run
```

需要注意的是，我们分别对上述三个测试对象进行性能测试时，为了控制变量，所使用的指令及参数均相同。接下来，我们会分点展示并分析每个测试对象的相应性能表现，及各虚拟机之间的横向比对。

2.1.3 CPU 性能测试

(1) 原生虚拟机：

```
Running the test with following options:
Number of threads: 1
Initializing random number generator from current time


Prime numbers limit: 2000

Initializing worker threads...

Threads started!

CPU speed:
    events per second: 13401.17

General statistics:
    total time:                          10.0001s
    total number of events:              134043

Latency (ms):
         min:                                    0.07
         avg:                                    0.07
         max:                                    0.66
         95th percentile:                        0.08
         sum:                                 9976.50

Threads fairness:
    events (avg/stddev):           134043.0000/0.00
    execution time (avg/stddev):   9.9765/0.00
```

(2) 开启 KVM 的 QEMU 虚拟机：

```
CPU speed:
    events per second: 10497.41

General statistics:
    total time:                          10.0082s
    total number of events:              104996

Latency (ms):
         min:                                    0.07
         avg:                                    0.09
         max:                                   13.92
         95th percentile:                        0.11
         sum:                                 9966.91

Threads fairness:
    events (avg/stddev):           104996.0000/0.00
    execution time (avg/stddev):   9.9669/0.00
```

(3) 未开启 KVM 的 QEMU 虚拟机：

```
CPU speed:
    events per second:   400.44

General statistics:
    total time:                          10.0082s
    total number of events:              4019

Latency (ms):
         min:                                    1.05
         avg:                                    2.45
         max:                                   19.02
         95th percentile:                        5.77
         sum:                                 9850.71

Threads fairness:
    events (avg/stddev):           4019.0000/0.00
    execution time (avg/stddev):   9.8507/0.00
```

我们首先分析原生虚拟机（S1）与开启 KVM 模块的 QEMU 虚拟机（S2）之间的性能差异。从测试结果可以看到，S1 的 CPU 处理速度（13401.14 event per second）要快于 S2（10497.41 event per second）；二者在 latency 方面的总时长大致相同，但 S1 的最大 latency（0.66ms）要远优于 S2（13.92ms）；最后，二者在 thread fairness 方面的事件数目上差别与处理速度的差别相同（S1 为 134043.0000 / 0.00，S2 为 104996.0000 / 0.00），而执行时间大致相同（S1 为 9.9765 / 0.00，S2 为 9.9669 / 0.00）。

总结：S1 的整体性能表现要略优于 S2，但开启了 KVM 后的 QEMU 虚拟机在各方面已经基本接近了在 VMware 上安装的虚拟机。在相同的测试时间内，母虚拟机 S1 的 CPU 速度（通过处理的事件数衡量）要快于开启 KVM 的 QEMU 虚拟机 S2，但相差在 30%以内；Latency 方面，二者平均延迟接近，但 S1 的稳定性要优于 S2；Thread fairness 方面 S1 也要略优于 S2.

接下来，我们分析开启 KVM 模块的 QEMU 虚拟机（S2）与未开启 KVM 模块的 QEMU 虚拟机（S3）之间的性能差异，进而得出 KVM 模块对 QEMU 虚拟机性能提升的重要意义。

同上类似，我们也分成 CPU 处理速度、latency 和 thread fairness 三个方面对二者进行评估。相较于 S2 的 10497.41 event per second，S3 的 CPU 处理速度会大幅度降低（只有 400.44 event per second）；Latency 方面，S2 的最大、最小、平均延迟都要显著优于 S3；Thread fairness 上，S2 在事件数目上（104996.0000 / 0.00）要明显优于 S3（4019.0000 / 0.00），但二者在执行时间上并没有明显差异。

总结：KVM 对 QEMU 虚拟机性能的提升有着至关重要的作用。在相同的测试时间内，开启 KVM 模块的 QEMU 虚拟机无论是在 CPU 处理速度（超出 95%以上），latency（整体超出 90%以上）还是 thread fairness（整体超出 60%左右），都明显优于未开启 KVM 模块的 QEMU 虚拟机。KVM 模块的重要性可见一斑。

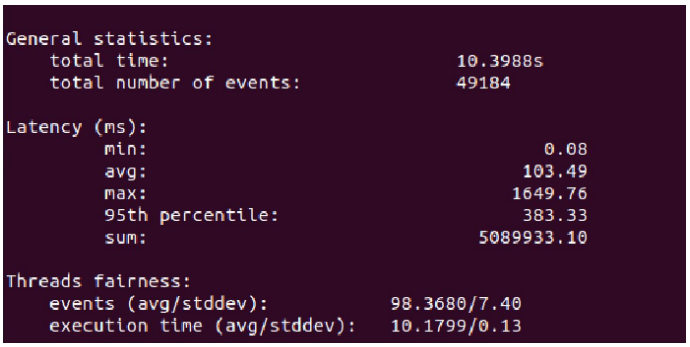
2.1.4 线程测试

测试截图如下所示：

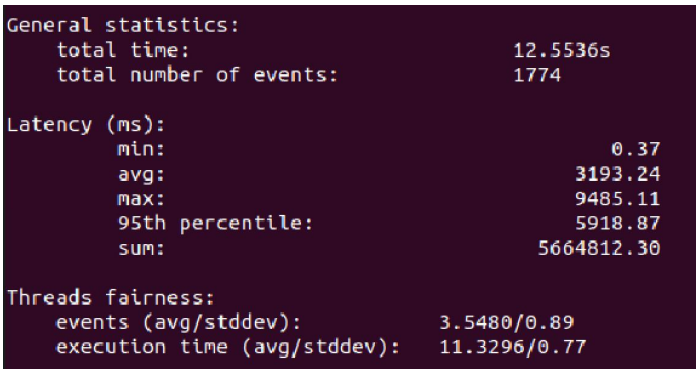
（1）原生虚拟机：

```
General statistics:
    total time:                          10.0278s
    total number of events:              188229

Latency (ms):
         min:                                    0.05
         avg:                                   26.60
         max:                                  519.94
         95th percentile:                      112.67
         sum:                                5007196.80

Threads fairness:
    events (avg/stddev):           376.4580/38.13
    execution time (avg/stddev):   10.0144/0.01
```

(2) 开启 KVM 的 QEMU 虚拟机:



(3) 未开启 KVM 的 QEMU 虚拟机:



首先分析原生虚拟机 (S1) 与开启 KVM 模块的 QEMU 虚拟机 (S2) 在线程性能方面的差异。可以看到，在总时间基本相同的前提下，S1 处理的事件数 (188229) 是 S2 (49184) 的三倍多；Latency 方面，在最大、最小和平均延迟的整体表现上，S1 也比 S2 要短了三倍左右；Thread fairness 上，S1 的 events (avg / stddev) 是 376.4580 / 38.13，execution time (avg / stddev)是 10.0144 / 0.01，而 S2 的 events (avg / stddev) 是 98.3680 / 7.40，execution time (avg / stddev)是 10.1799 / 0.13，显然 S1 的线程 fairness 在整体上要明显优于 S2。

总结：在线程性能上，原生虚拟机 S1 不论是在处理速度，latency 还是 thread fairness 上的表现，都明显优于开启 KVM 模块的 QEMU 虚拟机 S2。S1 各项性能表现得分基本都是 S2 的三倍以上，这个差距远大于在 CPU 性能上二者的差距，说明 VMware 上安装的原生虚拟机和开启 KVM 模块的 QEMU 虚拟机在线程性能上的差距还是非常明显的。

之后分析是否开启 KVM 模块对虚拟机线程性能的影响，令开启 KVM 模块的为 S2，未开启的为 S3。可以看到，即便 S3 的执行时间要略长于 S2，其执行事件数 (1774) 还是远少于 S2 (49184)；Latency 方面，S2 的整体表现要远远优于 S3，平均延迟只有 S3 的十几分之一；Thread fairness 方面的差距也非常明显。

总结：在线程性能上，开启 KVM 模块的 QEMU 虚拟机 S2 要远远优于未开启 KVM 的 S3，且整体性能已经差出了十倍以上。这说明是否开启 KVM 模块对 QEMU 虚拟机的线程性能有着非常重要的影响。

2.1.5 磁盘 IO 测试

测试截图如下所示：

(1) 原生虚拟机：

```
2147483648 bytes written in 18.04 seconds (113.53 MiB/sec).
```

```
File operations:
    reads/s:                      32091.67
    writes/s:                     21395.28
    fsyncs/s:                     68461.00

Throughput:
    read, MiB/s:                  501.43
    written, MiB/s:               334.30

General statistics:
    total time:                          10.0002s
    total number of events:              1220005

Latency (ms):
         min:                                    0.00
         avg:                                    0.16
         max:                                   21.10
         95th percentile:                        1.04
         sum:                               199320.94

Threads fairness:
    events (avg/stddev):           61000.2500/1525.51
    execution time (avg/stddev):   9.9660/0.00
```

(2) 开启 KVM 的 QEMU 虚拟机：

```
2147483648 bytes written in 41.39 seconds (49.49 MiB/sec).
```

```
File operations:
    reads/s:                      898.81
    writes/s:                     599.21
    fsyncs/s:                     2155.07

Throughput:
    read, MiB/s:                  14.04
    written, MiB/s:               9.36

General statistics:
    total time:                          10.6130s
    total number of events:              36214

Latency (ms):
         min:                                    0.00
         avg:                                    5.52
         max:                                   47.39
         95th percentile:                       17.01
         sum:                               199760.41

Threads fairness:
    events (avg/stddev):           1810.7000/165.09
    execution time (avg/stddev):   9.9880/0.01
```

(3) 未开启 KVM 的 QEMU 虚拟机：

```
2147483648 bytes written in 127.67 seconds (16.04 MiB/sec).
```

```
    reads/s:                      68.85
    writes/s:                     46.91
    fsyncs/s:                     299.70

Throughput:
    read, MiB/s:                  1.08
    written, MiB/s:               0.73

General statistics:
    total time:                          16.3725s
    total number of events:              4259

Latency (ms):
         min:                                    0.02
         avg:                                   46.84
         max:                                  692.99
         95th percentile:                      139.85
         sum:                                199504.04

Threads fairness:
    events (avg/stddev):           212.9500/18.50
    execution time (avg/stddev):   9.9752/0.04
```

首先分析原生虚拟机（S1）与开启 KVM 模块的 QEMU 虚拟机（S2）在磁盘 IO 性能方面的差异。从创建相同数目测试文件的速度上来看，S1 为 113.53 MiB / sec，S2 为 49.49 MiB / sec。在读写速度，Throughput 和总事件数的整体表现上，S1 达到了惊人的 S2 的四十倍左右；Latency 方面，S1 在平均表现上是 S2 的几十分之一，而最小和最大延迟接近；Thread fairness 上的表现也要远好于 S2。

总结：在磁盘 IO 性能上，VMware 上安装的原生虚拟机要远远好于开启 KVM 模块的 QEMU 虚拟机。其中原因一方面是后者属于二次虚拟化，性能上本身会受较大影响；另一方面是虚拟化之后对磁盘 IO 的变成了间接操作，因此速度和性能上受到了明显的影响。

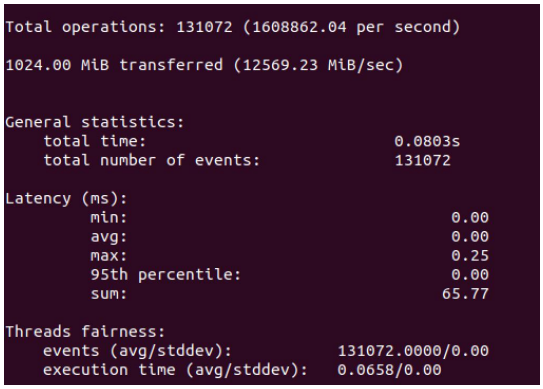
之后分析是否开启 KVM 模块对虚拟机磁盘 IO 性能的影响，令开启 KVM 模块的为 S2，未开启的为 S3。可以看到，无论是读写速度、Throughput、总事件数、thread fairness 还是 latency，S2 的表现都要远远好于 S3。

总结：在磁盘 IO 性能上，开启 KVM 模块对 QEMU 虚拟机的性能提升十分明显。

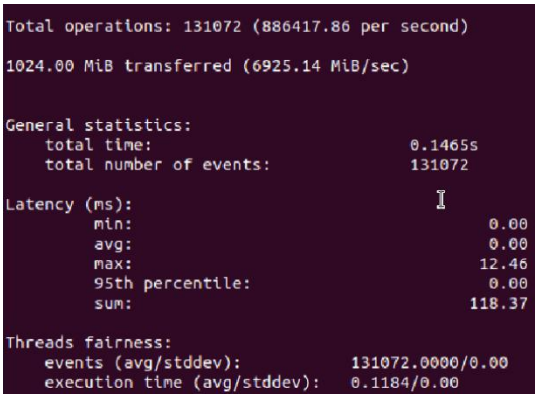
2.1.6 内存测试

测试截图如下所示：

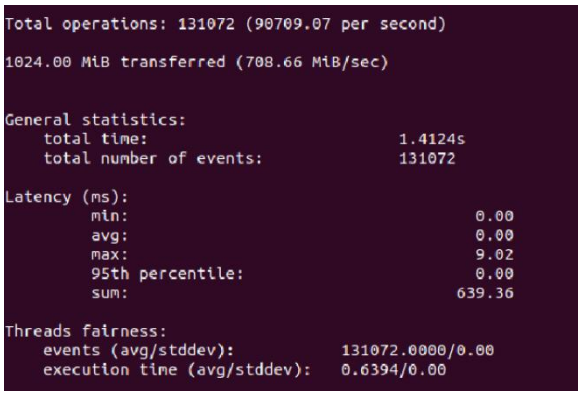
（1）原生虚拟机：



（2）开启 KVM 的 QEMU 虚拟机：



（3）未开启 KVM 的 QEMU 虚拟机：



首先分析原生虚拟机（S1）与开启 KVM 模块的 QEMU 虚拟机（S2）在内存性能方面的差异。可以看到，S1 的传输速率（12569.23 MiB / sec）大约为 S2（6925.14 MiB / sec）的两

倍；二者的最小、平均延迟都为 0，但 S1 的最大延迟（0.25ms）仅为 S2（12.46ms）的几十分之一；Thread fairness 上二者也较为接近。

总结：在内存性能上，VMware 上安装的原生虚拟机要优于开启 KVM 模块的 QEMU 虚拟机，但二者的差距并没有那么大。

之后分析是否开启 KVM 模块对虚拟机磁盘 IO 性能的影响，令开启 KVM 模块的为 S2，未开启的为 S3。可以看到 S2 的传输速率（6925.14 MiB / sec）大约为 S3（708.66 MiB / sec）的十倍；二者的最小、平均延迟都为 0，但 S2 的最大延迟（12.46ms）要略大于 S3（9.02ms）；Thread fairness 上二者仅在 execution time (avg / stddev)上有不大的差异。

总结：在内存性能上，开启 KVM 的 QEMU 虚拟机在传输速率上大幅优于未开启 KVM 的，但在 latency 方面却略有落后；在 thread fairness 方面略有领先。

2.1.7 Mutex 测试

测试截图如下所示：

（1）原生虚拟机：

```
General statistics:
    total time:                          0.0255s
    total number of events:              4

Latency (ms):
         min:                                    18.45
         avg:                                    21.77
         max:                                    23.26
         95th percentile:                        23.10
         sum:                                    87.09

Threads fairness:
    events (avg/stddev):           1.0000/0.00
    execution time (avg/stddev):   0.0218/0.00
```

（2）开启 KVM 的 QEMU 虚拟机：

```
General statistics:
    total time:                          0.1245s
    total number of events:              4

Latency (ms):
         min:                                   113.82
         avg:                                   116.05
         max:                                   120.38
         95th percentile:                       121.08
         sum:                                   464.19

Threads fairness:
    events (avg/stddev):           1.0000/0.00
    execution time (avg/stddev):   0.1160/0.00
```

(3) 未开启 KVM 的 QEMU 虚拟机：

```
General statistics:
    total time:                          0.9866s
    total number of events:              4

Latency (ms):
         min:                                  914.73
         avg:                                  934.73
         max:                                  954.37
         95th percentile:                      960.30
         sum:                                 3738.93

Threads fairness:
    events (avg/stddev):           1.0000/0.00
    execution time (avg/stddev):   0.9347/0.02
```

首先分析原生虚拟机（S1）与开启 KVM 模块的 QEMU 虚拟机（S2）在 Mutex 性能方面的差异。可以看到，在总事件数都为 4 的前提下，S1 的总时间（0.0255s）是 S2（0.1245s）的六分之一左右；Latency 上的整体表现 S1 略好于 S2.

总结：在 Mutex 性能上，VMware 上安装的原生虚拟机要优于开启 KVM 模块的 QEMU 虚拟机，且在执行速率、延迟方面都要明显优于后者。

之后分析是否开启 KVM 模块对虚拟机 mutex 性能的影响，令开启 KVM 模块的为 S2，未开启的为 S3。可以看到，S2 的总时间（0.1245s）约为 S3（0.9866s）的八分之一；整体延迟也只为后者的八分之一；Thread fairness 上无明显差异。

总结：在 Mutex 性能上，开启 KVM 模块对处理速率及整体延迟的提升很大，但对 thread fairness 的提升有限。

2.2 QEMU 源码修改以降低虚拟化性能损耗

在完成了 QEMU 环境下虚拟化性能损耗的测试和对比之后，我们开始在能够通过编译的前提下，尝试修改 QEMU 的部分源码，以此来尽可能降低虚拟化的性能损耗。

在本节中，我们使用的是 QEMU-3.0.1 版本作为母版进行修改，原因是更高版本的 QEMU 代码优化程度更高，更不容易从源码进行改进以降低虚拟化性能损耗。

2.2.1 QEMU 源码结构分析

在阅读 QEMU 源码之前，我们需要先对源码文件有一个宏观的认知，以掌握 QEMU 源码众多文件的一个整体代码结构。

从宏观上来看，源码结构主要包含以下几个部分：

(1) **vl.c**：最主要的模拟循环，虚拟机环境初始化，CPU 的执行；

(2) **translate.c**：将 guest 代码翻译成不同框架结构的 TCG 操作码；

（3）**tcg.c**：主要的 TCG 代码；

（4）**tcg-target.c**：将 TCG 代码转化成主机代码；

（5）**cpu-exec.c**：主要寻找下一个二进制翻译代码块，如果没有找到就请求得到下一个代码块，并操作生成的代码块；

涉及的几个主要函数如下：

（1）**main_loop ()**：很多条件的判断，如电源是否断等；

（2）**qemu_main_loop_start ()**：分时运行 CPU 核；

（3）**struct CPUState**：CPU 状态结构体；

（4）**cpu_exec ()**：主要的执行循环；

（5）**struct TransitionBlock ()**：TB（二进制翻译代码块）结构体；

（6）**cpu_gen_code ()**：初始化真正代码生成；

（7）**tcg_gen_code ()**：tcg 代码翻译成 host 代码。

2.2.2 QEMU-3.0.1 部分源码改进

我们主要从以下三个方面进行 QEMU-3.0.1 源码的部分改进。

2.2.2.1 CPU 分时系统的改进

本节中，我们讨论 **cpu-exec.c** 文件 **cpu_exec_step_atomic ()**函数中 **start_exclusive ()**函数的调用位置给 CPU 分时系统带来的性能损耗，以及改进方法。

首先阐释 **start_exclusive ()**函数（**cpu-common.c** 文件中实现）的用途：

通过 **atomic_set ()**、**atomic_read ()**等原子操作函数来读取当前 cpu 总的运行状态，若有其他 cpu 正在运行，则将其暂停并保存在一个等待队列中。当且仅当 **end_exclusive ()**函数被某个已经完成某个执行单元的 cpu 执行时，该 cpu 才可能被执行。同时，该函数设置了一个锁 mutex，来保证每个时刻最多只有一个 cpu 能够访问 pending_cpus 变量和全部其他 cpu 的运行状态信息。

下面，我们来分析 QEMU-3.0.1 在 **cpu_exec_step_atomic ()**函数中调用 **start_exclusive ()**函数的缺陷。

首先来观察原始的调用情况：

```
if (sigsetjmp(cpu->jmp_env, 0) == 0) {
    tb = tb_lookup__cpu_state(cpu, &pc, &cs_base, &flags, cf_mask);
    if (tb == NULL) {
        mmap_lock();
        tb = tb_gen_code(cpu, pc, cs_base, flags, cflags);
        mmap_unlock();
    }

    start_exclusive();
```

可以看到，**start_exclusive ()**函数在上面的 if 语句结束之后才被调用，而这是不合理的。if 语句中 **sigsetjmp ()**函数是系统提供的一个函数，其作用是保存当前的堆栈环境，然后将目前的地址作一个记号；在程序其他地方调用 **sinlongjmp ()**时会直接跳到这个记号位置，然后还原堆栈，继续程序的执行。若返回值为 0，说明已经做好标记，否则未成功标记。

然而，按照上面 QEMU-3.0.1 的代码逻辑，无论 **sigsetjmp ()**函数是否成功保存了堆栈环境，都会调用 **start_exclusive ()**函数进行 cpu 的上下文切换，而如果未成功保存，这样的上下文切换将是无效甚至危险的，这会降低 cpu 的整体使用效率。

因此，我们对其作出如下改进：

```
if (sigsetjmp(cpu->jmp_env, 0) == 0) {
    start_exclusive();

    tb = tb_lookup__cpu_state(cpu, &pc, &cs_base, &flags, cf_mask);
    if (tb == NULL) {
        mmap_lock();
        tb = tb_gen_code(cpu, pc, cs_base, flags, cflags);
        mmap_unlock();
    }
```

这样，当且仅当 **sigsetjmp ()**函数成功保存当前堆栈环境之后，**start_exclusive ()**函数才会被执行，cpu 的利用效率得到了极大的提升，虚拟化带来的性能损耗也能够得到一定程度上的降低。

2.2.2.2 CPU 线程池空闲判断的改进

本节中，我们讨论 **cpus.c** 文件中 **cpu_thread_is_idle ()**函数（布尔函数类型）判断依据的缺陷，及如何通过增添判断函数来提高该函数的判断能力。

```
static bool cpu_thread_is_idle(CPUState *cpu)
{
    if (cpu->stop || cpu->queued_work_first) {
        return false;
    }
```

在 QEMU-3.0.1 判断 cpu thread 非空闲的实现中（见上图），仅仅使用了 **cpu->stop** 和一个简单的标志 **cpu->queued_work_first** 来判断该函数返回值为 false 的条件。我们需要注意的是，**cpu->queued_work_first** 所标志的是是否允许当前队列中的 cpu 优先执行，这与我们

在该处使用的目的并不完全契合。而且，单独一个 flag 可能会在其他函数的执行中被错误地修改，导致“死锁”现象的发生。

因此，我们单独创建一个静态内联函数 **cpu_work_list_empty ()**，来检测当前 cpu 工作队列是否为空，并利用 **mutex_lock** 锁来保证查询的互斥性。这样，我们避免了错误判断 cpu 工作队列非空而导致的“死锁”现象的发生，提高了 cpu 的并行效率，进而降低了虚拟化性能损耗。改进如下：

```
static inline bool cpu_work_list_empty(CPUState *cpu)
{
    bool ret;

    qemu_mutex_lock(&cpu->work_mutex);
    ret = QSIMPLEQ_EMPTY(&cpu->work_list);
    qemu_mutex_unlock(&cpu->work_mutex);
    return ret;
}

static bool cpu_thread_is_idle(CPUState *cpu)
{
    if (cpu->stop || !cpu_work_list_empty(cpu)) {
        return false;
    }
```

2.2.2.3 CPU 调试器的改进

本节中，我们讨论 **cpu-exec.c** 文件中静态内联函数 **cpu_handle_interrupt ()**函数在 **GDB（GNU Debugger）** 可能存在的指令丢失问题，以及相应的解决方案。

在 QEMU-3.0.1 的 **cpu-exec.c** 文件中，**cpu_handle_interrupt ()**函数的设计使得当 cpu 开启 **single step** 模式时，GDB（GNU Debugger）在执行时可能会 miss 下一条指令（因为在该 if 语句中 **cpu->exception_index** 无论是否处于 **single step** 模式都被赋为-1）。

```
/* The target hook has 3 exit conditions:
   False when the interrupt isn't processed,
   True when it is, and we should restart on a new TB,
   and via longjmp via cpu_loop_exit.  */
else {
    if (cc->cpu_exec_interrupt(cpu, interrupt_request)) {
        replay_interrupt();
        cpu->exception_index = -1;
        *last_tb = NULL;
    }
    /* The target hook may have updated the 'cpu->interrupt_request';
     * reload the 'interrupt_request' value */
    interrupt_request = cpu->interrupt_request;
}
```

因此我们需要增添一个标志 **cpu->singlestep_enabled**，来标志执行该语句时，cpu 是否开启了 **single step**，以及一个宏 **EXCP_DEBUG** 来给处于 **single step** 模式下的 **cpu->exception_index** 赋值。通过这样的改进，我们较好地避免了 GNB 工作时产生的指令丢失，降低了调试损耗，提高了系统的工作效率。

改进的代码如下：

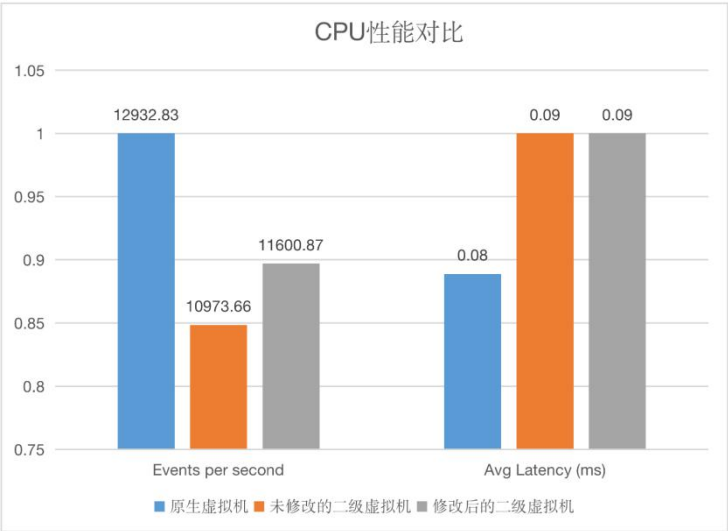
```
/* The target hook has 3 exit conditions:
   False when the interrupt isn't processed,
   True when it is, and we should restart on a new TB,
   and via longjmp via cpu_loop_exit.  */
else {
    if (cc->cpu_exec_interrupt(cpu, interrupt_request)) {
        replay_interrupt();
        /*
         * After processing the interrupt, ensure an EXCP_DEBUG is
         * raised when single-stepping so that GDB doesn't miss the
         * next instruction.
         */
        cpu->exception_index =
            (cpu->singlestep_enabled ? EXCP_DEBUG : -1);
        *last_tb = NULL;
    }
    /* The target hook may have updated the 'cpu->interrupt_request';
     * reload the 'interrupt_request' value */
    interrupt_request = cpu->interrupt_request;
}
```

2.2.3 虚拟化损耗前后对比

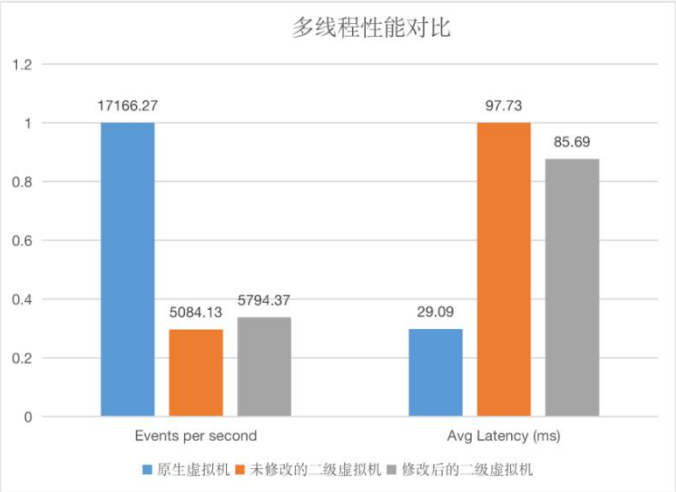
本节中，我们讨论按照 2.2 节所示方案进行代码修改之后对虚拟化损耗降低的效果。我们所使用的工具是 Linux 环境下的 benchmark 测试工具 sysbench，其具体安装办法见 2.1.1。与 2.1 节相同，我们从 cpu、线程、文件 IO、内存和 Mutex 五个方面分别对原生虚拟机、未修改的二级虚拟机和修改后的二级虚拟机的性能进行评测。

我们将针对五个评测指标分别给出一张图表，进行更加直观的比对。具体的测试结果截图见附录一。为了便于观察虚拟化损耗的情况，所有未特殊指明的指标数据默认都已进行标准化处理（将最高数据设置为 1）。

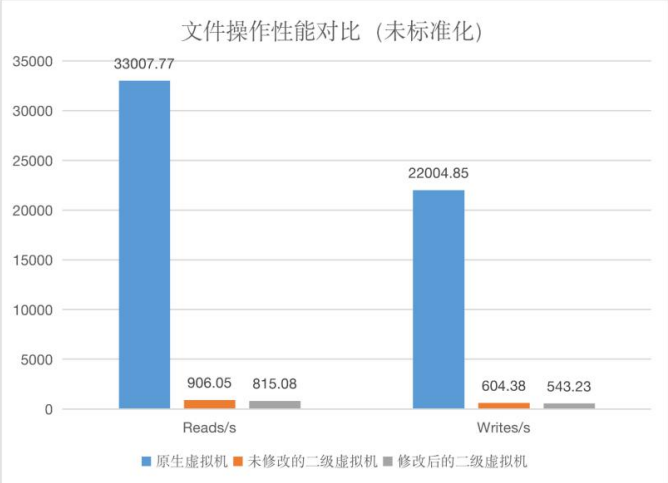
（1）CPU 性能对比：

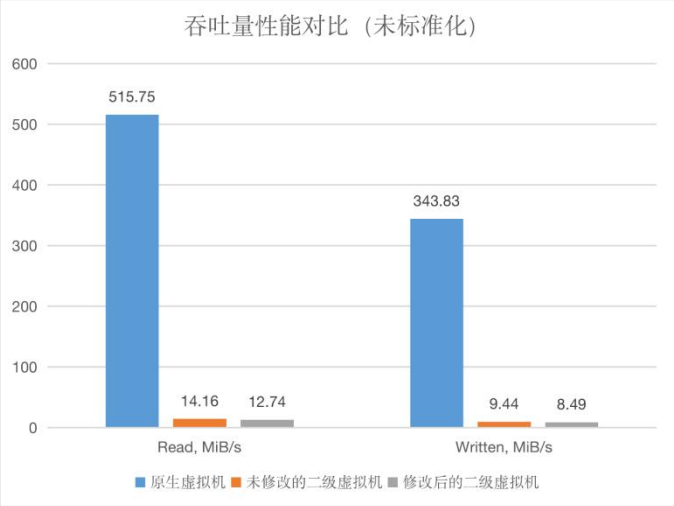


(2) 多线程性能对比：

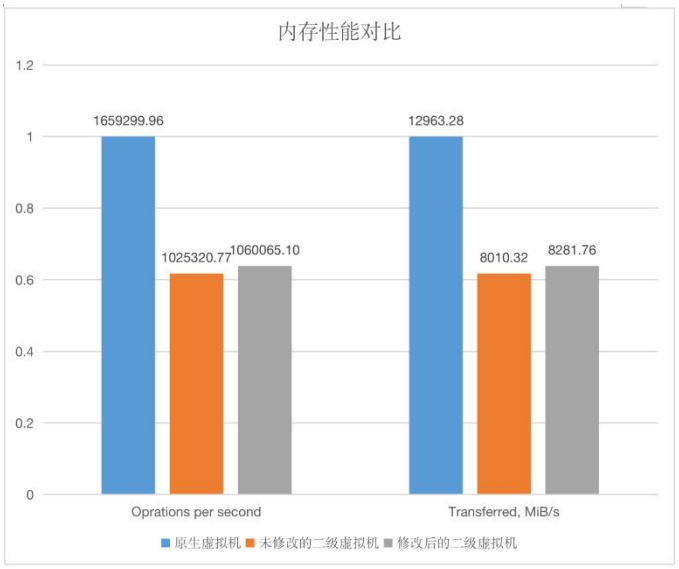


(3) 磁盘 IO 性能对比：

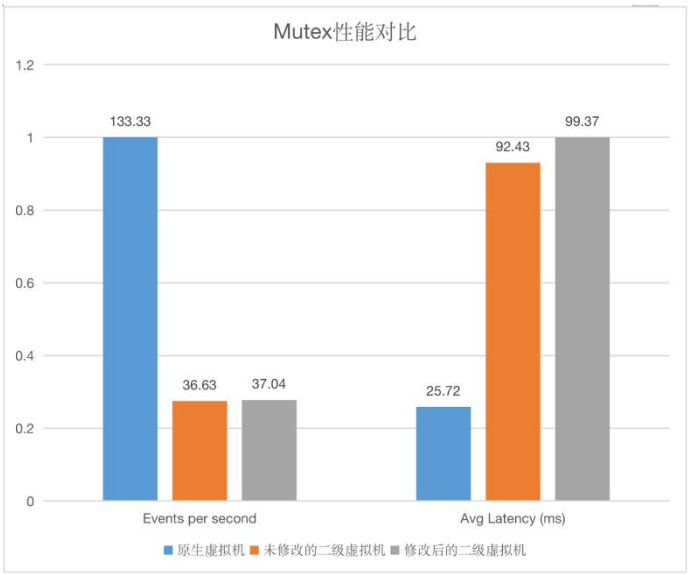




(4) 内存性能对比：



(5) Mutex 性能对比：



可以看到，修改后的二级虚拟机在 cpu、多线程和内存上都要优于修改前，cpu 和多线程上的性能提升尤为明显。分析原因，我们在 2.2.2 节中针对 CPU 分时系统和线程池进行了源码上的改进，提高了分时系统及线程池的运行效率。尽管平均延迟有所增加，修改后的二级虚拟机在事件处理速度上的 Mutex 性能表现也要略优于修改前。

然而，在磁盘 IO 的处理速度及吞吐量上，修改后的二级虚拟机要劣于修改前，分析原因可能是因为我们在 CPU 线程池及调试器的判据上进行了丰富，在进行文件 IO 时花费在这方面的资源有所增多。

无论是否对 QEMU 源码进行改进，各评测指标上的虚拟化性能损耗都是不可避免的。

3. 实验结论

本次实验中，我们成功完成了以下工作：

(1) 与原生性能对比，通过测试发现虚拟化的性能损耗；

(2) 修改 QEMU 部分源码，降低虚拟化的性能损耗。

第一部分中，我们实现了通过 sysbench 从 cpu、多线程、磁盘 IO、内存和 Mutex 五个指标对原生虚拟机、未开启 KVM 的二级虚拟机和开启 KVM 的二级虚拟机进行性能测试及横向对比，较为直观地得出了虚拟化带来的性能损耗，并了解了 KVM 模块对 QEMU 虚拟机的重要性。

第二部分中，我们通过对 CPU 分时系统、CPU 线程池空闲判据和 CPU 调试器三个方面对 QEMU-3.0.1 的源码进行了部分改进，显著降低了 cpu、多线程、内存及 Mutex 事件处理速度上的虚拟化性能损耗，但在磁盘 IO 上的损耗出现了略微上升。总体来看，本次源码修改对虚拟化性能损耗的降低是成功的。

4. 实验心得

在开始进行本次实验的选题时，我原本的打算是做 GPU 应用热迁移。但在经过一系列的调研之后，我发现目前关于 GPU 应用热迁移的相关文章是少之又少，上手难度很大。因此，我不得不转向工作量更大的修改源码降低虚拟化性能损耗。

在完成从官网上下载 QEMU 源码之后，我发现整个软件源码的体量是极为庞大的，令人生畏。因此，我决定首先对 QEMU 的源码结构有一个宏观了解，再对症下药，对重要的源码文件进行仔细分析（QEMU 源码分析见 2.2.1 节）。

经过分析，我选择了针对 cpu-exec.c 和 cpus.c 这两个主管 CPU 调度使用的源码文件进行细读，继而决定了从 CPU 分时系统、线程池及调试器几个方面进行源码上的改进。

修改源码的过程是较为艰难的，为了实现某一个改进的目标，我们往往需要修改包含函数调用、头文件定义和函数实现等数个源码文件，才能保证修改后的源码能够通过编译并正确安装。

在设计两个部分的性能测试时，我们参考了作业 2 中的性能测试方案。第一部分中，虽然 sysbench 能够完全提供几项重要的性能测试，但考虑到如果只对开启了 KVM 模块的虚拟机进行测试会缺少一个对比组，难以通过数据（如 CPU 速度、延迟等）给出一个较为直观的结果，所以我设计了三组实验对象相互对照。这样，既能够比较得出开启 KVM 模块的 QEMU 虚拟机与原生虚拟机之间的性能差距，也能够很直观地展示出 KVM 模块对 QEMU 虚拟机性能的重要性。第二部分修改源码之后的性能测试也参考了这样的思想，设置了原生虚

拟机、未修改的二级虚拟机和修改后的二级虚拟机三个实验组进行对比测试，进而得出更为直观的结论。

本次实验总体来说是较为成功的，我们较好地完成了所有实验目标，也增长了很多关于 QEMU 虚拟机源码编译、安装的知识，为时两天的 QEMU 源码阅读和改进也让我的知识和能力得到了极大的提升。希望在下一次的实验中能够同样收获满满!

附录一：性能测试结果截图

CPU 性能对比：

(1) 原生虚拟机：

```
CPU speed:
    events per second: 12932.83

General statistics:
    total time:                          10.0002s
    total number of events:              129346

Latency (ms):
         min:                                    0.07
         avg:                                    0.08
         max:                                    4.19
         95th percentile:                        0.09
         sum:                                 9974.95

Threads fairness:
    events (avg/stddev):           129346.0000/0.00
    execution time (avg/stddev):   9.9749/0.00
```

(2) 未修改的二级虚拟机：

```
CPU speed:
    events per second: 10973.66

General statistics:
    total time:                          10.0001s
    total number of events:              109753

Latency (ms):
         min:                                    0.08
         avg:                                    0.09
         max:                                   17.66
         95th percentile:                        0.12
         sum:                                 9971.22

Threads fairness:
    events (avg/stddev):           109753.0000/0.00
    execution time (avg/stddev):   9.9712/0.00
```

(3) 修改后的二级虚拟机：

```
CPU speed:
    events per second: 11600.87

General statistics:
    total time:                          10.0001s
    total number of events:              116026

Latency (ms):
         min:                                    0.07
         avg:                                    0.09
         max:                                   16.18
         95th percentile:                        0.10
         sum:                                 9979.95

Threads fairness:
    events (avg/stddev):           116026.0000/0.00
    execution time (avg/stddev):   9.9799/0.00
```

多线程性能对比：

(1) 原生虚拟机：

```
General statistics:
    total time:                          10.0292s
    total number of events:              172164

Latency (ms):
         min:                                    0.05
         avg:                                   29.09
         max:                                  597.19
         95th percentile:                      123.28
         sum:                                5007712.72

Threads fairness:
    events (avg/stddev):           344.3280/34.95
    execution time (avg/stddev):   10.0154/0.01
```

(2) 未修改的二级虚拟机：

```
General statistics:
    total time:                          10.1272s
    total number of events:              51488

Latency (ms):
         min:                                    0.08
         avg:                                   97.73
         max:                                 1075.39
         95th percentile:                      350.33
         sum:                                5032081.49

Threads fairness:
    events (avg/stddev):           102.9760/7.64
    execution time (avg/stddev):   10.0642/0.06
```

(3) 修改后的二级虚拟机：

```
General statistics:
    total time:                          10.0917s
    total number of events:              58475

Latency (ms):
         min:                                    0.08
         avg:                                   85.69
         max:                                 1000.21
         95th percentile:                      303.33
         sum:                                5010888.03

Threads fairness:
    events (avg/stddev):           116.9500/8.18
    execution time (avg/stddev):   10.0218/0.12
```

磁盘 IO 性能对比：

(1) 原生虚拟机：

```
File operations:
    reads/s:                      33007.77
    writes/s:                     22004.85
    fsyncs/s:                     70411.66

Throughput:
    read, MiB/s:                  515.75
    written, MiB/s:               343.83

General statistics:
    total time:                          10.0002s
    total number of events:              1254411

Latency (ms):
         min:                                    0.00
         avg:                                    0.16
         max:                                   39.50
         95th percentile:                        0.99
         sum:                               199352.03

Threads fairness:
    events (avg/stddev):           62720.5500/1918.69
    execution time (avg/stddev):   9.9676/0.00
```

(2) 未修改的二级虚拟机：

```
File operations:
    reads/s:                      906.05
    writes/s:                     604.38
    fsyncs/s:                     2165.95

Throughput:
    read, MiB/s:                  14.16
    written, MiB/s:               9.44

General statistics:
    total time:                          10.6363s
    total number of events:              36547

Latency (ms):
         min:                                    0.00
         avg:                                    5.46
         max:                                   44.23
         95th percentile:                       16.71
         sum:                               199430.32

Threads fairness:
    events (avg/stddev):           1827.3500/71.57
    execution time (avg/stddev):   9.9715/0.01
```

(3) 修改后的二级虚拟机：

```
File operations:
    reads/s:                      815.08
    writes/s:                     543.23
    fsyncs/s:                     1979.26

Throughput:
    read, MiB/s:                  12.74
    written, MiB/s:               8.49

General statistics:
    total time:                          10.6003s
    total number of events:              32823

Latency (ms):
         min:                                    0.00
         avg:                                    6.09
         max:                                   51.81
         95th percentile:                       19.65
         sum:                               199941.23

Threads fairness:
    events (avg/stddev):           1641.1500/90.62
    execution time (avg/stddev):   9.9971/0.01
```

内存性能对比：

(1) 原生虚拟机：

```
Total operations: 131072 (1659299.96 per second)

1024.00 MiB transferred (12963.28 MiB/sec)


General statistics:
    total time:                          0.0774s
    total number of events:              131072

Latency (ms):
         min:                                    0.00
         avg:                                    0.00
         max:                                    0.09
         95th percentile:                        0.00
         sum:                                   63.32

Threads fairness:
    events (avg/stddev):           131072.0000/0.00
    execution time (avg/stddev):   0.0633/0.00
```

(2) 未修改的二级虚拟机：

```
Total operations: 131072 (1025320.77 per second)

1024.00 MiB transferred (8010.32 MiB/sec)


General statistics:
    total time:                          0.1264s
    total number of events:              131072

Latency (ms):
         min:                                    0.00
         avg:                                    0.00
         max:                                   12.04
         95th percentile:                        0.00
         sum:                                  109.64

Threads fairness:
    events (avg/stddev):           131072.0000/0.00
    execution time (avg/stddev):   0.1096/0.00
```

(3) 修改后的二级虚拟机：

```
Total operations: 131072 (1060065.10 per second)

1024.00 MiB transferred (8281.76 MiB/sec)


General statistics:
    total time:                          0.1220s
    total number of events:              131072

Latency (ms):
         min:                                    0.00
         avg:                                    0.00
         max:                                    8.03
         95th percentile:                        0.00
         sum:                                  101.63

Threads fairness:
    events (avg/stddev):           131072.0000/0.00
    execution time (avg/stddev):   0.1016/0.00
```

Mutex 性能对比：

(1) 原生虚拟机：

```
General statistics:
    total time:                          0.0300s
    total number of events:              4

Latency (ms):
         min:                                    19.64
         avg:                                    25.72
         max:                                    29.87
         95th percentile:                        29.72
         sum:                                   102.87

Threads fairness:
    events (avg/stddev):           1.0000/0.00
    execution time (avg/stddev):   0.0257/0.00
```

(2) 未修改的二级虚拟机：

```
General statistics:
    total time:                          0.1092s
    total number of events:              4

Latency (ms):
         min:                                    83.61
         avg:                                    92.43
         max:                                   100.41
         95th percentile:                       101.13
         sum:                                   369.71

Threads fairness:
    events (avg/stddev):           1.0000/0.00
    execution time (avg/stddev):   0.0924/0.01
```

(3) 修改后的二级虚拟机：

```
General statistics:
    total time:                          0.1080s
    total number of events:              4

Latency (ms):
         min:                                    88.92
         avg:                                    99.37
         max:                                   105.16
         95th percentile:                       104.84
         sum:                                   397.49

Threads fairness:
    events (avg/stddev):           1.0000/0.00
    execution time (avg/stddev):   0.0994/0.01
```