StratoVirt 构建及性能对比

学号： N/A

姓名： N/A

专业： 计算机科学与技术

1 调研部分

1.1 Linux 启动协议及 E820 表的设计

在物理机 Intel x86_64 平台上，Linux 的启动依赖于 BIOS 的引导，需要启动协议规定内核如何从硬件状态逐步进入操作系统的运行环境。

当计算机通电后，基于 8086 结构的 CPU 会自动进入实模式，只能访问 1MB 的内存空间。BIOS 被加载到内存的最高区域 0xF0000 到 0xFFFFF，CPU 从 0xFFFF0（BIOS 入口）处开始执行 BIOS 代码。BIOS 负责完成系统硬件的自检和初始化，并在实模式内存布局中设置 Linux 内核启动必要的信息，例如中断向量表、BIOS 数据区域、内核启动信息、启动分区信息和显示适配区等。

BIOS 完成相关配置后，会设置相关寄存器。包括将中断向量表地址填入 IDTR 寄存器、将全局描述符表填入 GDTR 寄存器，通过设置 CR0 寄存器的 PE 位使 CPU 进入保护模式。保护模式支持 4GB 的内存寻址，并允许通过 GDTR 和 IDTR 动态管理内存和中断。此时将控制权转交给 BootLoader，加载内核映像到内存中，跳转到内核入口 `startup_32` 开始执行。

E820 表是 BIOS 提供的一个标准化的内存布局表，用于描述系统中的物理内存布局。它的目的是告知操作系统哪些内存区域是可用的，哪些是不可用的。例如 BIOS 和中断向量表所在的区域是预留的，操作系统不应该使用，有一些内存地址被用作 PCI 设备的内存映射区域等。Linux 内核在启动时会读取 E820 表，根据表中的信息初始化内存管理系统，将可用的内存区域映射到虚拟地址空间中。

1.2 Epoll 实现原理

Epoll 是一种 I/O 机制，它可以提供高效的并发 I/O 操作。相比传统的 select 和 poll，Epoll 只会在有事件发生时才会通知应用程序，是异步非阻塞的 I/O 模型。程序使用传统阻塞 I/O 时，程序可能处于自旋等待状态，浪费性能。传统阻塞 I/O 会阻塞一个端口，如果正在等待输入，那么就无法输出。即使可以使用超时设定（如 poll），程序仍然需要频繁请求内核检查是否有事件发生，浪费 CPU 资源。而 Epoll 则可以在有事件发生时通知应用程序，避免了频繁的轮询操作。

Epoll 主要有三个步骤：注册事件、等待事件和处理事件。注册事件时，应用程序向 Epoll 内核模块注册需要监听的文件描述符、事件类型和回调函数。Epoll 内部维护一个事件表，用于存储注册的事件。等待事件时，Epoll 会使用内部的线程轮询注册的文件描述符，当有注册的事件发生时，Epoll 会调用注册的回调函数。处理事件时，应用程序可以在回调函数中处理事件，例如读写数据。

Epoll 的本质仍然是轮询模型，但是通过使用事件通知机制，它可以只运行线程来服务多个监听任务，避免了这这些监听任务自行轮询的开销。

1.3 StratoVirt 技术重点

根据交互式脚本，StratoVirt 由如下技术组成：

- 在虚拟化环境中，输入输出的模拟通常通过端口 I/O 进行。在串行输出时，虚拟机将数据通过 Port I/O 方式写入特定端口（如 0x3f8），从而触发陷入，再将数据传递给 Hypervisor 进行处理。Hypervisor 通过判断虚拟机的退出原因，如串行输出事件，将数据转发至宿主机的控制台标准输出。此外，虚拟机的输入输出还可以通过 epoll 机制进行事件监听，优化虚拟机大量并发 I/O 操作的性能。
- 虚拟机的内存管理涉及多个方面，包括内存分配和内存映射。由于栈内存分配的容量有限且堆内存不能保证连续性，使用 `mmap` 系统调用来映射一块连续的物理内存到虚拟地址空间，从而模拟虚拟机的物理内存。mmap 为虚拟机提供了高效的内存映射方式，并允许将多个分开的内存区域进行配置，例如通过 `kvm_userspace_memory_region` 结构体设置内存区域的起始地址、大小、权限等信息，来支持虚拟机内存的可用区划分。
- 在虚拟化环境中，CPU 控制主要是通过对 vCPU（虚拟 CPU）的寄存器进行读写操作来实现的。Hypervisor 通过读取 kvm 的 CPU 结构体来访问和操作 vCPU 的通用寄存器和特殊寄存器，在 VM-Exit 时保存虚拟机状态，或者进行中断虚拟化。
- MMIO（内存映射输入输出）操作同时涉及到输入输出和内存管理。在虚拟化环境中，MMIO 通过内存映射表来实现，Hypervisor 需要根据 MMIO 地址和内存映射表查找宿主机中对应的内存区域，读取或写入相应的硬件寄存器地址。

StratoVirt 与传统的 QEMU 相比，更加注重轻量化设计，通过简化架构和优化资源管理，提高了虚拟机的性能和效率。具体体现在如下几个方面：

- 抛弃传统的 BIOS+GRUB 启动模式。无需依次加载 BIOS、启动 GRUB、进入 CPU 实模式、配置信息、切换到保护模式、加载内核等步骤。StratoVirt 采用了统一的启动协议，在 bootloader 模块中完成 BIOS 和 GRUB 的工作，跳过实模式流程，直接进入保护模式加载内核，启动时间更短。
- 内存布局采用树状结构和平坦视图结合的方案。通过树状结构可以快速了解到各个内存区域之间的拓扑结构关系；通过平坦视图可以更快速地查找内存区域。
- 大量采用 virtio 半虚拟化技术，增加磁盘、网卡的访问速度。

2 实验目的

对比 StratoVirt 和 QEMU 的如下性能指标，并分析两种平台的轻量化程度。

- Linux 启动时间
- 内存占用
- CPU 性能
- 内存性能
- I/O 速度

3 实验步骤

3.1 实验环境

- 操作系统：Ubuntu 22.04.3 LTS
- 内核版本：5.15.0-86-generic x86_64
- CPU 型号：Intel(R) Xeon(R) Gold 6230 CPU @ 2.10GHz
- CPU 逻辑处理器数：1
- CPU 缓存：L1d 32K, L1i 32K, L2 4M, L3 16M

3.2 实验过程

3.2.1 启动时间对比

分别使用以下命令，在 StratoVirt 和 QEMU 中启动 Linux 镜像。

- StratoVirt：

```
sudo stratovirt
  -machine microvm
  -smp 1
  -m 1024
  -kernel ./vmlinux.bin
  -append "console=ttyS0 root=/dev/vda reboot=k panic=1"
  -drive file=./openEuler-21.03-stratovirt-
x86_64.img,id=rootfs,readonly=off
  -device virtio-blk-device,drive=rootfs,id=rootfs
  -qmp unix:stratovirt.sock,server,nowait
  -serial stdio
```

- QEMU:

```
sudo qemu-system-x86_64
  -machine q35
  -smp 1
  -m 1024
  -drive file=openEuler-21.03-x86_64.qcow2,if=virtio
  -qmp unix:stratovirt.sock,server,nowait
  --enable-kvm
  -serial stdio
```

启动 L2 虚拟机后，可以使用 `systemd-analyze time` 命令查看内核的启动时间，重复测试 10 次取平均值。

3.2.2 内存占用对比

在启动 L2 虚拟机时，`-m 1024` 参数指定了虚拟机的内存大小为 1024MB。可以使用 `pmap` 命令在 L1 虚拟机中查看 L2 虚拟机的内存占用情况。对于 StratoVirt 和 QEMU 分别执行以下命令：

- StratoVirt: `sudo pmap -x $(pgrep stratovirt)`
- QEMU: `sudo pmap -x $(pgrep qemu)`

重复测试 10 次取平均值。

3.2.3 CPU 性能对比

为了进行性能测试，需要将 `sysbench` 工具安装到 L2 虚拟机中，对于 StraoVirt 还需要额外扩展硬盘空间以便测试。分别在 StratoVirt 和 QEMU 文件夹中执行以下命令：

- StratoVirt:

```
sudo su
mkdir mnt
mount openEuler-21.03-stratovirt-x86_64.img  mnt/
cd ./mnt/root
cp -r ../../../../utils/sysbench_src/ .
cd ../../
fuser -m -v mnt
umount ./mnt
rm -rf ./mnt
```

- QEMU:

```
sudo su
mkdir mnt
guestmount -a openEuler-21.03-x86_64.qcow2 -m /dev/sda1 mnt
cd ./mnt/root
cp -r ../../../../utils/sysbench_src/ .
cd ../../
guestunmount ./mnt
ls mnt
rm -rf mnt
```

随后在 L2 虚拟机中使用 `rpm -ivh *.rpm --nodeps` 命令安装 sysbench 工具。

安装完成后使用如下命令进行 CPU 性能测试，其中[threads]分别取 1、4、16、32、64:

```
sysbench --cpu-max-prime=10000 --threads=[threads] cpu run
```

对每种线程数重复测试 2 次取平均值。

3.2.4 内存性能对比

同样使用 sysbench 工具进行内存性能测试。可以使用如下命令进行测试，其中[size]分别取 1k、2k、4k、8k，[mode]分别取 sequential、random。

```
sysbench
  --threads=4
  --memory-block-size=1k
  --memory-access-mode=seq
  memory run
```

对每种配置重复测试 2 次取平均值，每次测试时间为 60s。

3.2.5 I/O 速度对比

测试前需要进行相应配置。由于 StratoVirt 的默认磁盘空间不足，需要在 StratoVirt 文件夹中使用以下命令扩展磁盘空间：

```
qemu-img resize openEuler-21.03-stratovirt-x86_64.img +3G
e2fsck -f openEuler-21.03-stratovirt-x86_64.img
resize2fs openEuler-21.03-stratovirt-x86_64.img
```

QEMU 需要设置成针对物理设备的 I/O，使用 raw 格式磁盘而非 qcow2 格式，并且需要禁用缓存。启动该 QEMU 的命令也需要相应修改：

```
qemu-img create -f raw ./disk.raw 4G

sudo qemu-system-x86_64
  -machine q35
  -smp 1
  -m 1024
  -drive file=openEuler-21.03-x86_64.qcow2,id=vd0,if=virtio
  -drive file=disk.raw,id=vd1,cache=none,format=raw,if=virtio
  -qmp unix:stratovirt.sock,server,nowait
  --enable-kvm
  -serial stdio
  --curses
```

测试 I/O 速度时，仍然使用 sysbench 工具，执行以下命令，其中第一组为 size=4k、threads=1，第二组为 size=128k、threads=32，第三组为 size=4k、threads=32。每组测试都包含随机读、随机写、顺序读、顺序写四种操作，分别对应 mode=rndrd, rndwr, seqrd, seqwr。

```
sysbench
  --file-block-size=[size]
  --threads=[threads]
  --file-test-mode=[mode]
  fileio prepare
sysbench
  --file-block-size=[size]
  --threads=[threads]
  --file-test-mode=[mode]
  fileio run
```

对每种配置重复测试 2 次取平均值，每次测试时间为 60s。

3.3 实验结果

3.3.1 启动时间对比

实验结果示例如图：

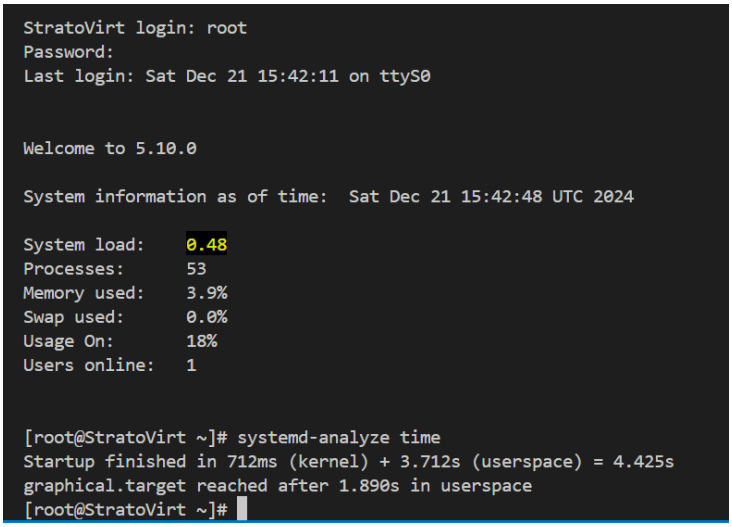

Figure 1: StratoVirt 启动时间

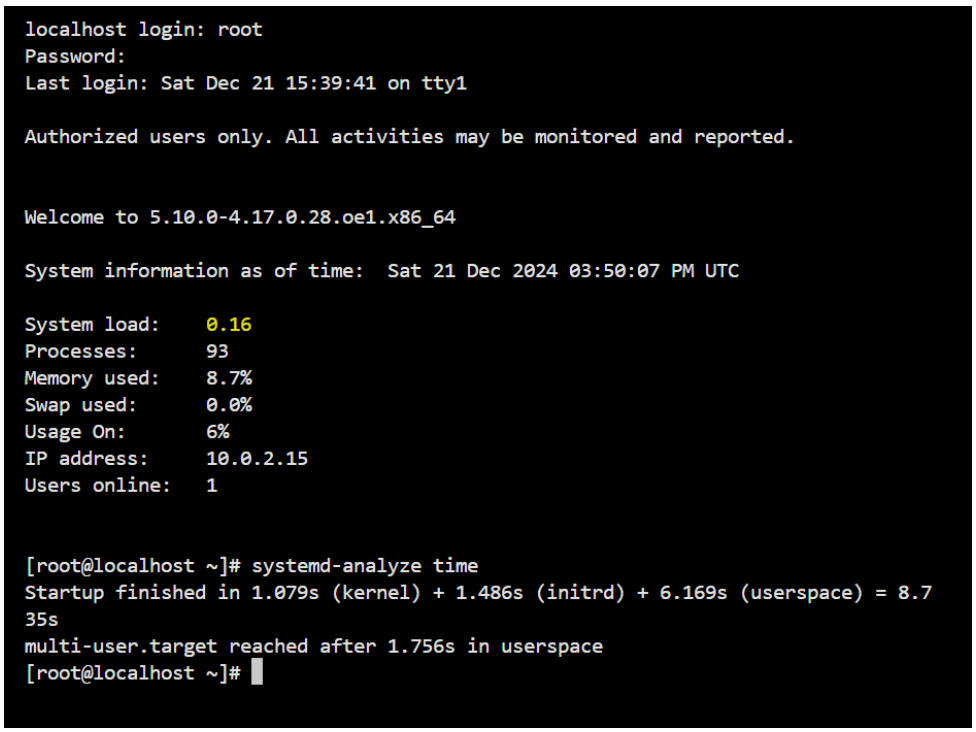

Figure 2: QEMU 启动时间

虚拟化平台	StratoVirt(ms)	QEMU(ms)
1	712	1079

虚拟化平台	StratoVirt(ms)	QEMU(ms)
2	738	1031
3	689	1036
4	702	1016
5	682	1032
6	700	1029
7	696	1107
8	685	1063
9	695	1092
10	700	1079
平均值	700	1056

3.3.2 内存占用对比

实验结果示例如图：

```
00007f94d0271000       4       4       4 rw-s- zero (deleted)
00007f94d0272000       8       8       8 r---- ld-linux-x86-64.so.2
00007f94d0274000       8       8       8 rw--- ld-linux-x86-64.so.2
00007ffd0d10a000     132      20      20 rw---   [ stack ]
00007ffd0d16e000      16       0       0 r----   [ anon ]
00007ffd0d172000       8       4       0 r-x--   [ anon ]
ffffffffff600000       4       0       0 --x--   [ anon ]
---------------- ------- ------- -------
total kB         1128048  127984  122192
root@virtlab:/home/virtlab/labs/compare#
```

Figure 3: StratoVirt 内存占用

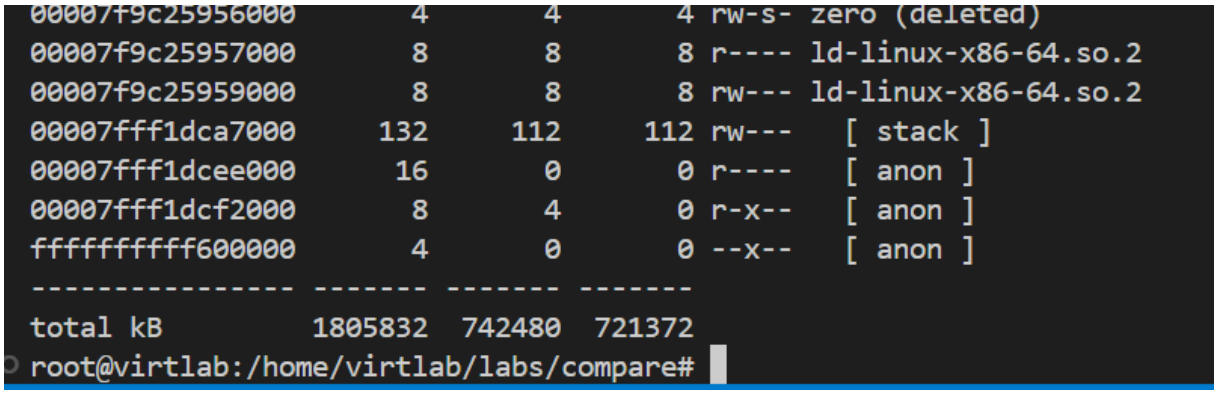

Figure 4: QEMU 内存占用

虚拟化平台	StratoVirt(KB)	QEMU(KB)
1	1128048	1805832
2	1128048	1803748
3	1128048	1795552
4	1128048	1805832
5	1128048	1795552
6	1128048	1805832
7	1128048	1807888
8	1128048	1807888
9	1128048	1807888
10	1128048	1656284
平均值	1128048	1789229

3.3.3 CPU 性能对比

实验结果示例如图：

```
sysbench 1.0.20 (using bundled LuaJIT 2.1.0-beta2)

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


Prime numbers limit: 10000

Initializing worker threads...

Threads started!

CPU speed:
    events per second:  1185.02

General statistics:
    total time:                          10.0001s
    total number of events:              11852

Latency (ms):
         min:                                    0.76
         avg:                                    0.84
         max:                                    4.74
         95th percentile:                        0.99
         sum:                                 9991.14

Threads fairness:
    events (avg/stddev):           11852.0000/0.00
    execution time (avg/stddev):   9.9911/0.00

[root@StratoVirt sysbench_src]#
```

Figure 5: StratoVirt CPU 性能

虚拟化平台	StratoVirt(events/s)	QEMU(events/s)
thread=1 (test 0)	1185.02	1216.73
thread=1 (test 1)	1192.44	1210.47
thread=4 (test 0)	1189.61	1220.39
thread=4 (test 1)	1199.98	1218.63
thread=16 (test 0)	1190.68	1206.37
thread=16 (test 1)	1194.03	1207.03
thread=32 (test 0)	1200.03	1216.67
thread=32 (test 1)	1237.08	1216.58

虚拟化平台	StratoVirt(events/s)	QEMU(events/s)
thread=64 (test 0)	1188.74	1224.77
thread=64 (test 1)	1189.16	1223.59

3.3.4 内存性能对比

实验结果示例如图：

```
sysbench 1.0.20 (using bundled LuaJIT 2.1.0-beta2)

Running the test with following options:
Number of threads: 4
Initializing random number generator from current time


Running memory speed test with the following options:
  block size: 1KiB
  total size: 102400MiB
  operation: write
  scope: global

Initializing worker threads...

Threads started!

Total operations: 21827156 (2182448.92 per second)

21315.58 MiB transferred (2131.30 MiB/sec)


General statistics:
    total time:                          10.0001s
    total number of events:              21827156

Latency (ms):
         min:                                    0.00
         avg:                                    0.00
         max:                                   22.60
         95th percentile:                        0.00
         sum:                                15049.59

Threads fairness:
    events (avg/stddev):           5456789.0000/30591.00
    execution time (avg/stddev):   3.7624/0.06
```

Figure 6: StratoVirt 内存性能

虚拟化平台	StratoVirt(MB/s)	QEMU(MB/s)
size=1k, mode=seq (test 0)	2131.30	397.26

虚拟化平台	StratoVirt(MB/s)	QEMU(MB/s)
size=1k, mode=seq (test 1)	2119.10	396.25
size=2k, mode=seq (test 0)	3847.55	762.02
size=2k, mode=seq (test 1)	3806.12	745.68
size=4k, mode=seq (test 0)	6583.85	1508.42
size=4k, mode=seq (test 1)	6514.91	1421.19
size=8k, mode=seq (test 0)	9900.70	2736.18
size=8k, mode=seq (test 1)	9845.70	2759.55
size=1k, mode=ran (test 0)	1198.51	327.19
size=1k, mode=ran (test 1)	1204.98	315.94
size=2k, mode=ran (test 0)	1563.93	539.93
size=2k, mode=ran (test 1)	1620.59	571.03
size=4k, mode=ran (test 0)	1868.01	910.96
size=4k, mode=ran (test 1)	1954.54	897.09
size=8k, mode=ran (test 0)	2107.94	1312.86
size=8k, mode=ran (test 1)	2093.26	1319.72

3.3.5 I/O 速度对比

实验结果示例如图：

```
sysbench 1.0.20 (using bundled LuaJIT 2.1.0-beta2)

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


Extra file open flags: (none)
128 files, 16MiB each
2GiB total file size
Block size 4KiB
Number of IO requests: 0
Read/Write ratio for combined random IO test: 1.50
Periodic FSYNC enabled, calling fsync() each 100 requests.
Calling fsync() at the end of test, Enabled.
Using synchronous I/O mode
Doing random read test
Initializing worker threads...

Threads started!


File operations:
    reads/s:                      3879.60
    writes/s:                     0.00
    fsyncs/s:                     0.00

Throughput:
    read, MiB/s:                  15.15
    written, MiB/s:               0.00

General statistics:
    total time:                          10.0001s
    total number of events:              38802

Latency (ms):
         min:                                    0.00
         avg:                                    0.26
         max:                                    2.41
         95th percentile:                        0.57
         sum:                                 9962.02

Threads fairness:
    events (avg/stddev):           38802.0000/0.00
    execution time (avg/stddev):   9.9620/0.00

[root@StratoVirt sysbench_src]#
```

Figure 7: StratoVirt I/O 速度

虚拟化平台	StratoVirt(MB/s)	QEMU(MB/s)
size=4k, threads=1, mode=rndrd (test 0)	15.15	12.44

虚拟化平台	StratoVirt(MB/s)	QEMU(MB/s)
size=4k, threads=1, mode=rndrd (test 1)	15.06	11.93
size=4k, threads=1, mode=rndwr (test 0)	7.29	8.01
size=4k, threads=1, mode=rndwr (test 1)	7.23	7.82
size=4k, threads=1, mode=seqrd (test 0)	225.46	216.22
size=4k, threads=1, mode=seqrd (test 1)	280.60	249.55
size=4k, threads=1, mode=seqwr (test 0)	15.97	30.35
size=4k, threads=1, mode=seqwr (test 1)	16.33	30.17
size=128k, threads=32, mode=rndrd (test 0)	929.04	411.31
size=128k, threads=32, mode=rndrd (test 1)	903.31	396.01
size=128k, threads=32, mode=rndwr (test 0)	449.00	256.29
size=128k, threads=32, mode=rndwr (test 1)	446.14	269.95
size=128k, threads=32, mode=seqrd (test 0)	690.77	297.04
size=128k, threads=32, mode=seqrd (test 1)	679.28	297.10

虚拟化平台	StratoVirt(MB/s)	QEMU(MB/s)
size=128k, threads=32, mode=seqwr (test 0)	464.93	229.10
size=128k, threads=32, mode=seqwr (test 1)	450.12	227.46
size=4k, threads=32, mode=rndrd (test 0)	55.29	41.45
size=4k, threads=32, mode=rndrd (test 1)	53.72	43.82
size=4k, threads=32, mode=rndwr (test 0)	31.55	18.44
size=4k, threads=32, mode=rndwr (test 1)	31.14	18.94
size=4k, threads=32, mode=seqrd (test 0)	221.11	270.48
size=4k, threads=32, mode=seqrd (test 1)	292.23	288.93
size=4k, threads=32, mode=seqwr (test 0)	36.53	33.41
size=4k, threads=32, mode=seqwr (test 1)	36.88	31.63

4 实验分析

根据实验获得的数据，可以对 Stratovirt 和 QEMU 的轻量化程度进行分析。

从启动时间的平均值及波动性来看，StratoVirt 显然在轻量化方面优于 QEMU。StraoVirt 的平均启动时间为 700ms，而 QEMU 的平均启动时间为 1056ms。QEMU 启动时间比 StratoVirt 高出 50% 左右，且波动性较大，说明 StratoVirt 在设计上更加专注于轻量化，启动速度更快。

从内存占用来看，StratoVirt 的内存占用平均值为 1128048KB，而 QEMU 的内存占用平均值为 1789229KB，QEMU 的内存占用比 StratoVirt 高出 58% 左右。值得注意的是，StraoVirt 的内存占用在 10 次测试中保持不变，而 QEMU 的内存占用会发生波动，说明 StratoVirt 在内存管理上更加稳定，更加注重减少资源占用。

在 CPU 性能方面，StratoVirt 和 QEMU 的表现相近，两者的事件数在不同线程数下基本保持一致，只有约 3% 的差距。说明 StratoVirt 能够在追求轻量化的同时保持 CPU 性能。不同的线程数对于 CPU 性能的影响不大，这是因为 L1 虚拟机只有一个逻辑处理器，本质上仍然只有一个线程在执行。如果要测试两平台对多核处理器的调度性能，需要在配置更高的 L1 虚拟机上进行测试。

StratoVirt 的内存访问速度明显优于 QEMU。在不同的内存块大小和访问模式下，StratoVirt 的内存访问速度都明显高于 QEMU，平均值相差约 3 倍。这说明 StratoVirt 在内存管理上远比 QEMU 更加高效。可能的原因是 StratoVirt 采用了内存映射 mmap 的方式直接映射物理内存到虚拟地址空间，而 QEMU 则采用了 HPT 等更复杂的内存管理方式，导致内存访问效率较低。

在 I/O 速度方面，StratoVirt 与 QEMU 的表现在不同配置下各有优劣。在单读写队列的情况下，StratoVirt 的读速度略高于 QEMU，而写速度只有 QEMU 的一半左右。在吞吐量较高的情况下，StratoVirt 的读写速度都表现好于 QEMU，平均比 QEMU 高出约 1 倍。这说明 StratoVirt 总体上在 I/O 性能上更优于 QEMU，但差距不大。可见 StratoVirt 对多线程并发、高吞吐量的 I/O 操作有更好的支持。

总体而言，StratoVirt 在轻量化方面明显优于 QEMU，其启动时间更短、内存占用更低且更稳定。在内存访问和 I/O 性能上，StratoVirt 也展示出了较为显著的优势，尤其是在内存管理上，StratoVirt 的效率远超 QEMU。然而，在 CPU 性能方面，两者差距较小，都能够满足常见的虚拟化需求。在高吞吐量的 I/O 操作下，StratoVirt 的表现更为出色，显示其在多线程并发处理和高负载场景下的优势。

StratoVirt 通过简化设计和优化资源管理，能够实现更好的性能表现，尤其适用于对启动速度和资源占用有较高要求的场景。QEMU 尽管在某些方面如写性能和多核处理能力上具有一定的优势，但其较高的资

源消耗和波动性表现使其在轻量化虚拟化场景中的竞争力不如 StratoVirt。

因此，StratoVirt 是一个更适合轻量级环境的“容器原生”的虚拟化技术。如果需要高级特性，需要组装更多组件，可能会带来性能下降。而 QEMU 则是一个功能更加全面的虚拟化环境，使用场景更广，但在轻量化方面表现不如 StratoVirt。

附录

参考文献

- 教材 深入浅出系统虚拟化：原理与实践
- sysbench 文档：https://github.com/akopytov/sysbench