

实验指导手册：Bare Metal

Alex Chi

利益相关，本文档中不会出现具体的 Bare Metal 云服务商名称。如果你想在云上裸金属服务器进行实验，可以在互联网上尝试搜寻相关的服务。

云主机提供商大多会有 referral program。同学之间可以互相推荐注册，以获得更多代金券。

服务器选择

尽量选择 NVMe SSD 盘的裸金属服务器。Disk 选 extra disk unformatted。操作系统使用 Ubuntu 20.04 或更高版本。

1.9 TB SSD (2 x 960GB)

\$185/mo

\$0.275/h

Intel E-2286G Processor

6 cores / 12 threads @ 4.0 GHz

32GB Memory

10TB Bandwidth

10Gbps Ethernet

4.0 TB NVMe (2 x 2000GB)

\$350/mo

\$0.521/h

Intel E-2288G Processor

8 cores / 16 threads @ 3.7 GHz

128GB Memory

10TB Bandwidth

10Gbps Ethernet

No RAID

Disks formatted/mounted.

No RAID

Extra disks unformatted.

RAID 1

Software RAID.

大概等十分钟就能使用服务器了。服务器上一共两块盘，一块系统盘，一块没有格式化的数据盘。

准备工作

- 确认 IOMMU 已经启用，确认可以硬件虚拟化。
- 确认数据盘盘型号。
- 关掉数据盘的 I/O 调度器。
- 安装实验所需要的软件。
- 确认系统信息、CPU 拓扑结构等。
- 其他工作
 - Intel 和 AMD CPU 的 IOMMU 启用方法不一样，下面的操作都是以 Intel CPU 为例的，请注意。
 - 如果服务商提供的第二块盘已经挂载为文件系统/组建了 RAID，你可能需要想办法将它恢复成一块能在本实验中使用的盘。
 - 如果没有 NVMe SSD，普通 SSD 也能进行这个实验。下文中 NVMe 相关的操作可能都要换成内核 I/O 栈。比如：
 - SPDK NVMe bdev -> uring bdev
 - SPDK 不受 IOMMU Group 限制
 - 无法做 NVMe 用户态驱动实验
 - `/nvmevnx` -> `/sdv`

安装软件

Bash

```
1 apt update && apt dist-upgrade
2 apt install qemu-kvm nvme-cli smartmontools liburing-dev libaio-dev b
  uild-essential cloud-image-utils hwloc
```

开启 IOMMU 与内核大页

推荐使用 vim 或 nano 命令行编辑器。如果不习惯命令行操作，也可以在裸金属服务器上安装图形化界面和 VNC 远程连接工具。

下面的所有操作都在 root 用户下进行。

确认 CPU 支持 1GB 内核大页。

Shell

```
1 cat /proc/cpuinfo | grep pdpe1gb
```

编辑 `/etc/default/grub`，在下面的选项中加入 Intel IOMMU。

Makefile

```
1 GRUB_CMDLINE_LINUX_DEFAULT="<原来存在的选项> intel_iommu=on  
  default_hugepagesz=1G hugepagesz=1G hugepages=4"
```

更新 grub，重启系统。

Bash

```
1 update-grub  
2 reboot
```

通过 `dmesg` 指令确认 IOMMU 已经启用。

Bash

```
1 → ~ dmesg | grep IOMMU  
2 [ 0.182378] DMAR: IOMMU enabled  
3 [ 0.389901] DMAR-IR: IOAPIC id 2 under DRHD base 0xfed91000 IOMMU  
  0
```

确认 1GB 大页已经成为默认大小。

Shell

```
1 ~ cat /proc/meminfo  
2 ...  
3 Hugepagesize:      1048576 kB  
4 Hugetlb:           4194304 kB  
5 DirectMap4k:       201032 kB  
6 DirectMap2M:       12150784 kB  
7 DirectMap1G:       121634816 kB
```

确认 SSD 的 IOMMU Group 没有其他设备

Shell

```
1 → ~ lshw -businfo -class storage
2 Bus info          Device          Class          Description
3 =====
4 pci@0000:02:00.0          storage          NVMe Datacenter SSD
   [3DNAND, Beta Rock Controller]
5                /dev/nvme0          storage          INTEL SSDPE2KX020T8
6 pci@0000:03:00.0          storage          NVMe Datacenter SSD
   [3DNAND, Beta Rock Controller]
7                /dev/nvme1          storage          INTEL SSDPE2KX020T8
8 pci@0000:00:17.0          storage          Cannon Lake PCH SATA
   AHCI Controller
9 → ~ find /sys/kernel/iommu_groups/ -type l
10 ...
11 /sys/kernel/iommu_groups/1/devices/0000:03:00.0
12 /sys/kernel/iommu_groups/1/devices/0000:00:01.2
13 /sys/kernel/iommu_groups/1/devices/0000:02:00.0
14 /sys/kernel/iommu_groups/1/devices/0000:00:01.0
15 /sys/kernel/iommu_groups/1/devices/0000:01:00.0
16 /sys/kernel/iommu_groups/1/devices/0000:00:01.1
17 /sys/kernel/iommu_groups/1/devices/0000:01:00.1
18 ...
```

在上面的输出结果中，数据 SSD 所在的 IOMMU Group 有其他设备。因此，需要打 kernel patch，或联系云裸金属服务器提供商换一个 PCIe 接口插盘。

- 可以参照 <https://liquorix.net> 在 Ubuntu 上直接安装 Zen Patched Kernel。（推荐）
- 可以用 <https://queuecumber.gitlab.io/linux-acs-override/> 安装 ACS Override Kernel。
- 也可以自己编译带 patch 的内核。

打好 patch 或安装带 patch 的内核后，将 grub 选项修改为：

Bash

```
1 GRUB_CMDLINE_LINUX_DEFAULT=""<原来存在的选项> intel_iommu=on pcie_acs_override=downstream default_hugepagesz=1G hugepagesz=1G hugepages=4""
```

然后执行

Bash

- 1 update-grub
- 2 reboot

在重启后检测 dmesg 中是否成功开启 ACS overrides。

Shell

- 1 → ~ dmesg | grep IOMMU
- 2 [0.000000] Warning: PCIe ACS overrides enabled; This may allow non-IOMMU protected peer-to-peer DMA

确认相关配置

确认 KVM 硬件虚拟化可用。

Bash

- 1 → ~ kvm-ok
- 2 INFO: /dev/kvm exists
- 3 KVM acceleration can be used

查看系统中所有的盘。如果 nvme1n1 存在且没有被挂载到任何位置，则说明配置正确。

Bash

```
1 → ~ lsblk -o NAME,FSTYPE,LABEL,MOUNTPOINT,SIZE,MODEL
2
3 NAME          FSTYPE  LABEL MOUNTPOINT          SIZE MODEL
4 loop0         squashfs          /snap/core18/1997 55.4M
5 loop1         squashfs          /snap/core18/2128 55.4M
6 loop2         squashfs          /snap/core20/1081 61.8M
7 loop3         squashfs          /snap/lxd/20037   68.8M
8 loop4         squashfs          /snap/lxd/21260   68.3M
9 loop5         squashfs          /snap/snapd/11588 32.3M
10 loop6        squashfs          /snap/snapd/12704 32.3M
11 nvme0n1              1.8T INTEL
   SSDPE2KX020T8
12 └─nvme0n1p1 ext4          /          1.8T
13 nvme1n1              1.8T INTEL
   SSDPE2KX020T8
```

查看 I/O 调度器，应当已经被设置为 none。

Bash

```
1 → ~ cat /sys/block/nvme1n1/queue/scheduler
2 [none] mq-deadline
```

Pre-condition SSD

把 NVMe 盘写两遍，保证 SSD 性能达到稳定状态 [\[来源\]](#)。提示 out-of-space 是正常现象。可以用 iotop 工具看写入速度。

Bash

```
1 → ~ smartctl -A /dev/nvme1
2 smartctl 7.2 2020-12-30 r5155 [x86_64-linux-5.11.0-17-generic] (local
   build)
3 Copyright (C) 2002-20, Bruce Allen, Christian Franke, www.smartmontools.org
4
5 === START OF SMART DATA SECTION ===
6 SMART/Health Information (NVMe Log 0x02)
7 Critical Warning:                                0x00
8 Temperature:                                     39 Celsius
9 Available Spare:                                  100%
```

```
10 Available Spare Threshold:          10%
11 Percentage Used:                    0%
12 Data Units Read:                    1,536,083 [786 GB]
13 Data Units Written:                 2,062,839 [1.05 TB]
14 Host Read Commands:                 182,570,667
15 Host Write Commands:                184,638,059
16 Controller Busy Time:               13
17 Power Cycles:                       71
18 Power On Hours:                     2,114
19 Unsafe Shutdowns:                   52
20 Media and Data Integrity Errors:    0
21 Error Information Log Entries:       0
22 Warning Comp. Temperature Time:     0
23 Critical Comp. Temperature Time:    0
24
25 → ~ dd if=/dev/zero of=/dev/nvme1n1 bs=128k oflag=nonblock
26 → ~ dd if=/dev/zero of=/dev/nvme1n1 bs=128k oflag=nonblock
27
28 → ~ smartctl -A /dev/nvme1
29 smartctl 7.2 2020-12-30 r5155 [x86_64-linux-5.11.0-17-generic] (local
    build)
30 Copyright (C) 2002-20, Bruce Allen, Christian Franke, www.smartmontools.org
31
32 === START OF SMART DATA SECTION ===
33 SMART/Health Information (NVMe Log 0x02)
34 Critical Warning:                    0x00
35 Temperature:                        38 Celsius
36 Available Spare:                     100%
37 Available Spare Threshold:           10%
38 Percentage Used:                     0%
39 Data Units Read:                     1,536,090 [786 GB]
40 Data Units Written:                  9,876,897 [5.05 TB]
41 Host Read Commands:                  182,570,864
42 Host Write Commands:                 215,602,792
43 Controller Busy Time:                53
44 Power Cycles:                        71
45 Power On Hours:                      2,118
46 Unsafe Shutdowns:                    52
47 Media and Data Integrity Errors:     0
48 Error Information Log Entries:        0
49 Warning Comp. Temperature Time:      0
50 Critical Comp. Temperature Time:     0
```

确认系统配置

Shell

```
1  → ~ lstopo --taskset
2  Machine (126GB total) cpuset=0xffff
3    Package L#0 cpuset=0xffff
4      NUMANode L#0 (P#0 126GB) cpuset=0xffff
5        L3 L#0 (16MB) cpuset=0xffff
6          L2 L#0 (256KB) cpuset=0x101
7            L1d L#0 (32KB) cpuset=0x101
8              L1i L#0 (32KB) cpuset=0x101
9                Core L#0 cpuset=0x101
10                  PU L#0 (P#0) cpuset=0x1
11                    PU L#1 (P#8) cpuset=0x100
12          L2 L#1 (256KB) cpuset=0x202
13            L1d L#1 (32KB) cpuset=0x202
14              L1i L#1 (32KB) cpuset=0x202
15                Core L#1 cpuset=0x202
16                  PU L#2 (P#1) cpuset=0x2
17                    PU L#3 (P#9) cpuset=0x200
18          L2 L#2 (256KB) cpuset=0x404
19            L1d L#2 (32KB) cpuset=0x404
20              L1i L#2 (32KB) cpuset=0x404
21                Core L#2 cpuset=0x404
22                  PU L#4 (P#2) cpuset=0x4
23                    PU L#5 (P#10) cpuset=0x400
24          L2 L#3 (256KB) cpuset=0x808
25            L1d L#3 (32KB) cpuset=0x808
26              L1i L#3 (32KB) cpuset=0x808
27                Core L#3 cpuset=0x808
28                  PU L#6 (P#3) cpuset=0x8
29                    PU L#7 (P#11) cpuset=0x800
30          L2 L#4 (256KB) cpuset=0x1010
31            L1d L#4 (32KB) cpuset=0x1010
32              L1i L#4 (32KB) cpuset=0x1010
33                Core L#4 cpuset=0x1010
34                  PU L#8 (P#4) cpuset=0x10
35                    PU L#9 (P#12) cpuset=0x1000
36          L2 L#5 (256KB) cpuset=0x2020
37            L1d L#5 (32KB) cpuset=0x2020
38              L1i L#5 (32KB) cpuset=0x2020
39                Core L#5 cpuset=0x2020
40                  PU L#10 (P#5) cpuset=0x20
```



```
41          PU L#11 (P#13) cpuset=0x2000
42      L2 L#6 (256KB) cpuset=0x4040
43          L1d L#6 (32KB) cpuset=0x4040
44          L1i L#6 (32KB) cpuset=0x4040
45          Core L#6 cpuset=0x4040
46          PU L#12 (P#6) cpuset=0x40
47          PU L#13 (P#14) cpuset=0x4000
48      L2 L#7 (256KB) cpuset=0x8080
49          L1d L#7 (32KB) cpuset=0x8080
50          L1i L#7 (32KB) cpuset=0x8080
51          Core L#7 cpuset=0x8080
52          PU L#14 (P#7) cpuset=0x80
53          PU L#15 (P#15) cpuset=0x8000
54  HostBridge
55      PCIBridge
56          PCI 01:00.0 (Ethernet)
57          Net "enp1s0f0"
58          PCI 01:00.1 (Ethernet)
59          Net "enp1s0f1"
60      PCIBridge
61          PCI 02:00.0 (NVMEExp)
62          Block(Disk) "nvme0n1"
63      PCIBridge
64          PCI 03:00.0 (NVMEExp)
65          Block(Disk) "nvme1n1"
66      PCI 00:17.0 (SATA)
67      PCIBridge
68          PCIBridge
69          PCI 06:00.0 (VGA)
```

Shell

```
1  → ~ fdisk -l
2
3  ...
4
5  Disk /dev/nvme0n1: 1.82 TiB, 2000398934016 bytes, 3907029168 sectors
6  Disk model: INTEL SSDPE2KX020T8
7  Units: sectors of 1 * 512 = 512 bytes
8  Sector size (logical/physical): 512 bytes / 512 bytes
9  I/O size (minimum/optimal): 512 bytes / 512 bytes
10 Disklabel type: dos
11 Disk identifier: 0xbab35b63
12
13 Device          Boot Start          End      Sectors  Size Id Type
14 /dev/nvme0n1p1 *      2048 3907029134 3907027087   1.8T 83 Linux
15
16
17 Disk /dev/nvme1n1: 1.82 TiB, 2000398934016 bytes, 3907029168 sectors
18 Disk model: INTEL SSDPE2KX020T8
19 Units: sectors of 1 * 512 = 512 bytes
20 Sector size (logical/physical): 512 bytes / 512 bytes
21 I/O size (minimum/optimal): 512 bytes / 512 bytes
```

安装相关软件

从 GitHub 源码编译安装 fio。

Shell

```
1  $ git clone https://github.com/axboe/fio
2  $ cd fio && git checkout fio-3.27
3  $ make
4  $ make install
```

从 GitHub 源码编译 SPDK。

Shell

```
1 $ git clone https://github.com/spdk/spdk
2 $ cd spdk && git checkout v21.07
3 $ git submodule update --init
4 $ scripts/pkgdep.sh --all
5 $ ./configure --with-fio=$HOME/fio --with-uring
6 $ make
7 $ make install
```

下载 Ubuntu VM 镜像并启动

可以参考这篇文章：<https://powersj.io/posts/ubuntu-qemu-cli/>

建议使用 tmux 或 screen 工具在同一个 SSH shell 中启动多个程序。

启动过程主要分成这么几步：

- 下载 Ubuntu Cloud Image 镜像。如果你想体验自己从头安装系统的快乐，也可以下载 Desktop Image 或 Server Image。国内服务器可以使用 SJTUG 镜像站加速下载。
- 配置 Cloud Image 的启动参数。
- 由于之后需要通过 SSH 协议连接 VM，需要生成一个 SSH key 并导入 Cloud Image 配置。
- 而后即可正常通过 QEMU 启动 VM。

Shell

```
1 $ wget https://cloud-images.ubuntu.com/releases/hirsute/release/ubuntu-21.04-server-cloudimg-amd64.img
2 $ # check SHA256SUM and GPG key
3 $ cat > metadata.yaml <<EOF
4 instance-id: iid-local01
5 local-hostname: cloudimg
6 EOF
7 $ ssh-keygen -t ed25519
8 Generating public/private ed25519 key pair.
9 Enter file in which to save the key (/root/.ssh/id_ed25519):
10 Enter passphrase (empty for no passphrase):
11 Enter same passphrase again:
12 Your identification has been saved in /root/.ssh/id_ed25519
13 Your public key has been saved in /root/.ssh/id_ed25519.pub
14 The key fingerprint is:
15 SHA256:L1Exa9UZM4bGFtG34ARlVy6yzJ5R4tb+2ahdYTQ0lx4 root@guest
16 The key's randomart image is:
```

```

17 +--[ED25519 256]--+
18 |          oo*0=*|=|
19 |          =*+*E+|
20 |          +o= =++|
21 |          o + B.+.|
22 |          S   B .o |
23 |          o o +. .|
24 |          . . o . .|
25 |          .   . +o|
26 |          ..o.o|
27 +-----[SHA256]-----+
28 $ cat ~/.ssh/id_ed25519.pub
29 $ cat > user-data.yaml <<EOF
30 #cloud-config
31 ssh_authorized_keys:
32   - <输出的 pub key 内容>
33 EOF
34 $ cloud-localds seed.img user-data.yaml metadata.yaml
35 $ qemu-system-x86_64 \
36   -machine accel=kvm,type=q35 \
37   -cpu host \
38   -m 2G \
39   -nographic \
40   -device virtio-net-pci,netdev=net0 \
41   -netdev user,id=net0,hostfwd=tcp::2222-:22 \
42   -drive if=virtio,format=qcow2,file=ubuntu-21.04-server-cloudimg-amd
    64.img \
43   -drive if=virtio,format=raw,file=seed.img
44 # Note: Use Ctrl-A X or run poweroff in SSH shell to exit qemu

```

执行完 QEMU 指令后，控制台进入 VM console。使用 Ctrl-A X 按键组合即可强制关机。

`-netdev user,id=net0,hostfwd=tcp::2222-:22` 用于将 VM 的 22 端口转发到本地 2222 端口。第一个 drive 挂载系统盘，第二个 drive 用于 cloud-init 初始化 VM 的网络。

接下来，就可以在服务器上通过 SSH 连接本地已经启动的 VM。

Shell

```
1 → ~ ssh -o "StrictHostKeyChecking no" -p 2222 ubuntu@127.0.0.1
```

开启防火墙

QEMU 暴露的 2222 端口通过公网 IP 也可以访问。为了防止黑客扫描攻击，我们需要开启防火墙，只允许宿主机的 SSH 从公网 IP 连入。

Shell

```
1 ufw allow ssh
2 ufw enable
```

开始实验

建议的实验顺序：

- 宿主机测试
 - fio 直接操作块设备
 - 解绑 NVMe 设备
 - SPDK
- 开虚拟机的测试
 - 绑定 NVMe 设备回系统
 - QEMU 直接操作块设备
 - 解绑 NVMe 设备
 - IOMMU 和用户态 NVMe 驱动测试
 - 启动 SPDK
 - SPDK 相关测试

实验前请务必确认 **filename** 等参数是否正确，以防写坏系统盘。

宿主机块设备性能

Shell

```
1 fio bench.fio
```

参考 `bench.fio`

Plain Text

```
1  [global]
2  name=NVMe-benchmark-uring
3  filename=/dev/nvme1n1
4  ioengine=io_uring
5  sqthread_poll=1
6  direct=1
7  numjobs=1
8  runtime=300
9  ramp_time=60
10 thread=1
11 time_based=1
12 randrepeat=0
13 norandommap=1
14 group_reporting=1
15 size=2000398934016
16 stonewall
17
18 [4K-RR-QD1J1]
19 bs=4k
20 rw=randread
21 iodepth=1
22
23 [4K-RR-QD32J1]
24 bs=4k
25 rw=randread
26 iodepth=32
```

宿主机 SPDK 性能

解绑 NVMe 设备。

Shell

```
1 → spdk git:(adeb04968) ./scripts/setup.sh
2 0000:02:00.0 (8086 0a54): Active mountpoints on nvme0n1:nvme0n1p1, so
   not binding PCI dev
3 0000:03:00.0 (8086 0a54): nvme -> vfio-pci
4 "root" user memlock limit: 16075 MB
5
6 This is the maximum amount of memory you will be
7 able to use with DPDK and VFIO if run as user "root".
8 To change this, please adjust limits.conf memlock limit for user
   "root".
```

具体操作步骤可以参考：

- https://github.com/spdk/spdk/tree/master/examples/nvme/fio_plugin
- <https://software.intel.com/content/www/cn/zh/develop/articles/evaluate-performance-f-or-storage-performance-development-kit-spdk-based-nvme-ssd.html>

参考 `bench.fio`

Plain Text

```
1  [global]
2  name=NVMe-benchmark-spdk-nvme
3  filename=trtype=PCIE traddr=0000.03.00.0 ns=1
4  ioengine=${HOME}/spdk/build/fio/spdk_nvme
5  direct=1
6  numjobs=1
7  runtime=300
8  ramp_time=60
9  thread=1
10 time_based=1
11 randrepeat=0
12 norandommap=1
13 group_reporting=1
14 size=2000398934016
15 stonewall
16
17 [4K-RR-QD1J1]
18 bs=4k
19 rw=randread
20 iodepth=1
21
22 [4K-RR-QD32J1]
23 bs=4k
24 rw=randread
25 iodepth=32
```

VM 中 Passthrough 性能、块设备性能、SPDK 性能

QEMU 启动参数可以参考实验手册。

在 VM 中安装下面的包，即可运行实验。

Shell

```
1  sudo apt install build-essential liburing-dev libaio-dev
2  # 编译并安装 fio
```

fio 配置可以沿用宿主机的块设备配置。注意 filename 需要对应做更改。