به نام خدا

پروژه‌ی سوم

سیستم‌های عامل - دکتر اسدی و دکتر جلیلی

دانشکده مهندسی کامپیوتر

دانشگاه صنعتی شریف

نیم‌سال اول ۰۳-۰۴

گروه ۲۲ :

آریان افضل‌زاده - 401105572

میترا قلی‌پور - 401106363

الینا هژبری - 401170661

ملیکا علی‌زاده - 401106255

مقایسه‌ی تاثیر غیر همزمان‌سازی فراخوان‌های سیستمی و حذف سربار فراخوانی سیستمی در عملکرد ذخیره‌سازی

مقدمه

با پدید آمدن حافظه‌های سریع‌تر، سربار زمانی فراخوان‌های سیستمی برای درخواست‌های ورودی/خروجی حائز اهمیت شده است. پلتفرم SPDK توسعه یافته است که با حذف برخی از سربارها در فراخوانی‌های سیستمی در لینوکس، سرعت دسترسی به دیسک‌های NVMe را افزایش دهد. همچنین، RocksDB پایگاه داده‌ی پرسرعت key-value برای محیط‌های ذخیره‌سازی سریع و پردازنده‌های چندهسته‌ای فراهم می‌کند.

اهداف

۱. آشنایی با پشته ذخیره سازی در سیستم‌عامل لینوکس

۲. شبیه‌سازی دیسک پرسرعت با پروتکل ارتباطی NVMe

۳. اجرای آزمون‌های عملکردی با RocksDB و ابزار FIO

۴. آشنایی و استفاده از پلتفرم SPDK

صورت پروژه

در این پروژه قصد داریم با استفاده از دیسک شبیه‌سازی شده NVMe در شبیه‌ساز NVMeVirt، تاثیر فراخوان‌های سیستمی io_uring و libaio با SPDK را بر اساس معیارهای IOPS، Latency و TailLatency مقایسه کنیم. همچنین، برنامه‌ای محک روی RocksDB با استفاده از SPDK اجرا و نتایج عملکرد را تحلیل خواهیم کرد.

گام‌های پروژه

۱. بررسی شبیه‌ساز NVMeVirt و تهیه‌ی مستند از ساختار و مراحل نصب آن به همراه تفاوت روش‌های مختلف

۲. ایجاد دیسک NVMeVirt_NVM و مشاهده‌ی عملکرد آن

۳. اجرای ابزار FIO روی دیسک شبیه‌سازی شده با مؤلفه‌های زیر

readwrite	BLOCK_SIZE	SIZE	NUM_JOB	QUEUE_DEPTH
randrw	4kb	4G	1	1
randrw	4kb	1G	4	1
randwrite	4kb	4G	1	1
randwrite	4kb	1G	4	1
randread	4kb	4G	1	1
randread	4kb	1G	4	1

۴. نصب پلتفرم SPDK و استفاده از پلاگین FIO برای آزمایش

۵. اجرای برنامه‌ای محک مانند db_bench روی RocksDB با استفاده از SPDK

۶. تحلیل و مقایسه‌ی نتایج و تهیه‌ی نمودار نمودار

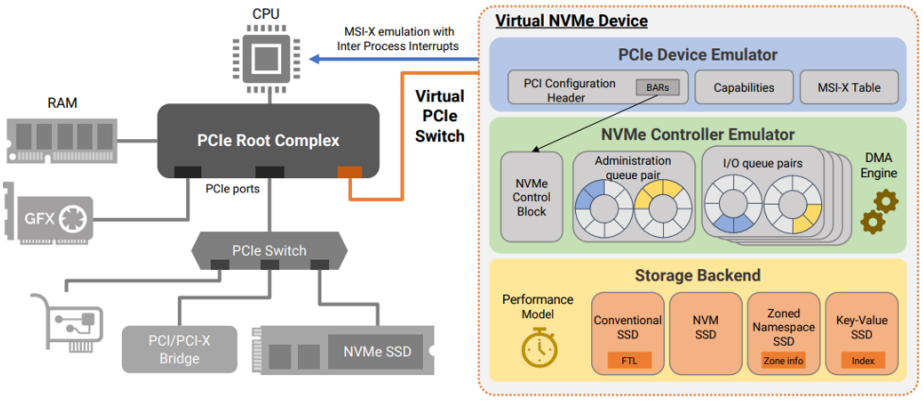
گام اول پروژه

NVMeVrit

NVMeVrit برای شبیه‌سازی دستگاه‌های NVMe در سطح PCIe طراحی شده است تا از دید سیستم‌عامل میزبان، مانند یک دستگاه فیزیکی واقعی عمل کند. در معماری PCIe، ریشه مجتمع PCIe Root Complex ارتباط بین پردازنده و دستگاه‌های متصل به PCIe را مدیریت می‌کند و اطلاعات پیکربندی آن‌ها را از طریق هدر پیکربندی PCIe شناسایی می‌کند.

NVMeVrit برای معرفی یک دستگاه NVMe مجازی، یک باس PCIe مجازی ایجاد کرده و یک هدر پیکربندی PCIe جعلی در حافظه رزروشده ثبت می‌کند. این هدر، اطلاعات مورد نیاز برای شناسایی دستگاه، مانند Device ID و Vendor ID را ارائه می‌دهد. هنگام اسکن باس PCIe، سیستم‌عامل میزبان این دستگاه را مانند یک NVMe واقعی شناسایی کرده و درایور NVMe را برای آن بارگذاری می‌کند، بدون اینکه نیازی به تغییرات سخت‌افزاری باشد.

در دستگاه‌های واقعی، عملیات NVMe از طریق تراکنش‌های PCIe به سخت‌افزار NVMe ارسال می‌شود. اما در NVMeVrit، کنترلر NVMe در حافظه رزروشده سیستم شبیه‌سازی شده است. برای نظارت بر درخواست‌های I/O، NVMeVrit از یک رشته کرنل به نام Dispatcher استفاده می‌کند که تغییرات در بلاک کنترل NVMe را بررسی کرده و درخواست‌های I/O را پردازش می‌کند. به جای استفاده از روش‌های مبتنی بر رویداد (event-driven) که باعث ایجاد تأخیر می‌شوند، از پویش مداوم (busy-waiting) برای کاهش تأخیر پردازش استفاده شده است.



شکل ۱: شکل ۱- معماری کلی NVMeVrit. NVMeVrit یک دستگاه NVMe مجازی را از طریق گذرگاه و سوئیچ PCIe مجازی می کند.

بخش‌های کلیدی معماری NVMeVrit

۱. **شبیه‌سازی دستگاه PCIe**: شامل ایجاد و مدیریت هدر پیکربندی برای دستگاه‌های مجازی.

۲. **کنترل بلاک NVMe**: اطلاعات کنترلی در حافظه رزرو شده نگهداری می‌شود.

۳. **Dispatcher**: درخواست‌های I/O را بررسی کرده و به صف مناسب منتقل می‌کند.

۴. **I/O Workers**: پردازش درخواست‌های I/O بر اساس نوع دستگاه و پشتیبانی از DMA.

۵. **Performance Model**: شبیه‌سازی زمان تأخیر و پهنای باند دستگاه مطابق با مشخصات واقعی.

۶. **Backend Storage**: ذخیره داده‌ها در حافظه اصلی.

ویژگی‌های منحصربه‌فرد NVMeVrit

- از دید سیستم‌عامل و سایر دستگاه‌های PCIe، NVMeVrit کاملاً مانند یک NVMe واقعی عمل می‌کند.
- برخلاف سایر شبیه‌سازها، NVMeVrit به برنامه‌های کاربری اجازه می‌دهد مستقیماً از فضای کاربری (User Space) به دستگاه دسترسی داشته باشند.
- پشتیبانی از انتقال داده همتا به همتا (Peer-to-Peer DMA) بین دستگاه‌های PCIe.
- بررسی دقیق اطلاعات صف‌ها، تأخیر و وضعیت پردازش درخواست‌ها.
- امکان استفاده به‌عنوان NVMe-oF Target برای آزمایش و پیاده‌سازی سیستم‌های ذخیره‌سازی توزیع‌شده.

مدیریت داده در NVMeVrit

NVMeVrit مانند بسیاری از شبیه‌سازهای ذخیره‌سازی دیگر، داده‌های درخواست‌شده را در حافظه اصلی (RAM) ذخیره می‌کند. از آنجا که این شبیه‌ساز به عنوان یک ماژول کرنل اجرا می‌شود، نمی‌تواند از قابلیت‌های فضای کاربری (User Space) مانند حافظه مجازی استفاده کند. با این حال، باید سربار مدیریت حافظه پایین و عملکرد آن ثابت باشد تا بتواند دستگاه‌های حافظه نسل آینده مانند PRAM و MRAM را شبیه‌سازی کند.
NVMeVrit به مقدار زیادی حافظه برای ذخیره داده‌ها نیاز دارد. این حافظه از طریق رزرو بخشی از فضای آدرس فیزیکی در زمان بوت سیستم تأمین می‌شود. برای جلوگیری از تأثیرگذاری بر عملکرد کلی سیستم، یک حافظه اختصاصی به نام NUMA تعبیه شده که در یک بخش از حافظه سیستم و در یک گره خاص NUMA رزرو می‌شود و تمام پردازش‌های مرتبط با NVMeVrit روی پردازنده‌های این گره انجام می‌شود. برای اینکار ابتدا بخش‌های مربوط به کنترل بلاک NVMe، منابع PCI مانند جدول MSI-X و قابلیت‌های PCI در ابتدای حافظه قرار داده شده‌اند. در ادامه، بخش عمده‌ای از حافظه برای ذخیره‌سازی داده‌ها اختصاص داده می‌شود.

مثال‌های مدیریت حافظه برای SSD، ZNS SSD و KVSSD

- **SSD**: ابتدا داده‌ها در حافظه رزرو شده سیستم نگهداری می‌شوند. سپس هر بلوک/صفحه فیزیکی به‌طور مستقیم در یک موقعیت خاص با استفاده از یک لایه ترجمه فلش (FTL) از حافظه نگاشت شده و مدیریت می‌شود.
- **ZNS SSD**: این درایو مانند SSD عمل می‌کند. همچنین اطلاعات وضعیت مناطق (Zone Status) مانند فهرست مناطق باز و اشاره‌گرهای نوشتن (Write Pointers) را در یک جدول مدیریت متاداده نگهداری می‌کند.
- **KVSSD**: در این درایو، داده‌ها به صورت Key-Value Pairs ذخیره شده و سپس به دو بخش نیمه اول و نیمه دوم تقسیم می‌شوند. نیمه اول شامل بخش‌های ۱ کیلوبایتی می‌شود و برای داده‌های کوچک به کار می‌رود. نیمه دوم که برای داده‌های بزرگ استفاده می‌شود، شامل بخش‌های ۴ کیلوبایتی است. در آخر از یک bitmap برای پیگیری استفاده از هر بلوک حافظه و از یک hash table برای مدیریت موقعیت داده‌های key-value در حافظه استفاده می‌شود.

مدل‌های عملکردی

۱. **مدل ساده (Simple Model)**: این مدل، درخواست‌های I/O را به چندین بخش کوچک‌تر تقسیم کرده و به‌صورت موازی پردازش می‌کند.

- مدت‌زمان پردازش هر درخواست برابر با مدت‌زمان پردازش آخرین بخش آن است.
- زمان تأخیر و پهنای باند دستگاه واقعی از طریق پارامترهای قابل تنظیم شبیه‌سازی می‌شود. برای مثال، SSD Intel Optane با تنظیم ۱۲ میکروثانیه تأخیر خواندن، ۱۴ میکروثانیه تأخیر نوشتن، و ۴.۲ گیگابایت بر ثانیه پهنای باند خواندن مدل‌سازی می‌شود.
- این مدل برای NVM SSD ها و KVSSDها مناسب است، زیرا نیازی به مدیریت Garbage Collection ندارند.

۲. **مدل موازی (Parallel Model)**: این مدل برای SSDهای فلش طراحی شده که از لایه ترجمه فلش (FTL) و پردازش موازی استفاده می‌کنند. در سه مرحله انجام می‌شود:

(آ) **مدیریت داده‌ها با FTL**:

- FTL به‌طور پویا داده‌ها را مدیریت کرده و Garbage Collection را هنگام کمبود فضای آزاد انجام می‌دهد.
- یک بافر نوشتن (Write Buffer) درخواست‌های نوشتن را موقتاً ذخیره کرده و سپس در حافظه اصلی ثبت می‌کند.
- ZNS SSDها به FTL نیاز ندارند، زیرا میزبان مستقیماً داده‌ها را مدیریت می‌کند.

(ب) **پردازش موازی در SSDها**: SSDهای مدرن دارای چندین بخش پردازشی (Partitions) هستند که هرکدام دارای یک نمونه مستقل از FTL هستند.

- ارتباط با میزبان از طریق لینک PCIe مشترک انجام می‌شود.
- داده‌ها از طریق چندین کانال NAND و Dieهای حافظه پردازش می‌شوند.

(ج) **مدل‌سازی معماری NAND**

- هر بخش SSD دارای ۸ کانال NAND است که هر کانال به چندین Die حافظه متصل است.
- پردازش روی Dieها موازی انجام می‌شود، اما انتقال داده از طریق کانال‌های NAND سریالی است.

کدهای مربوط به بخش PCIe

این کد در فایل pci.c در سایت گیت‌هاب NVMeVirt می‌توانید به طور کامل مشاهده کنید. این کد مسئول پیاده‌سازی و مدیریت دستگاه NVMe مجازی در لایه PCIe است. این فایل شامل توابع و ساختارهایی است که به سیستم‌عامل امکان می‌دهد دستگاه NVMe مجازی را شناسایی و با آن تعامل کند.
حال به توضیح کوتاهی از کد می‌پردازیم:

- در ابتدای کد هدرهای مورد نیاز و ساختارهای داده‌ای مرتبط با PCIe و NVMe تعریف شده‌اند.
- **خط ۱۳ تا ۴۹**: این بخش مربوط به ماژول کرنل لینوکس است که وظیفه مدیریت وقفه‌های سخت‌افزاری (IRQ) در معماری x86 را دارد. اگر `CONFIG_NVMEV_FAST_X86_IRQ_HANDLING` فعال باشد، آرایه `apicid_to_cpuid[256]` مقداردهی می‌شود و یک وقفه به پردازنده مقصد ارسال می‌کند. اگر فعال نباشد، با استفاده از قابلیت‌های پیش‌فرض مدیریت وقفه، وقفه را دوباره می‌فرستد.
- **خط ۵۱ تا ۷۳**: این کد وظیفه پردازش وقفه‌های MSI (Message Signaled Interrupts) را دارد و تغییرات مربوط به پردازش وقفه‌ها را برای سازگاری با تغییرات API کرنل مدیریت می‌کند و از روش بهینه‌تر برای نسخه‌های جدیدتر استفاده می‌کند.
- **تابع `nvmev_proc_bars`**: این تابع درایور یا بخش نرم‌افزاری مرتبط با دستگاه NVMe را مدیریت می‌کند و با بررسی تغییرات در رجیسترهای BAR (Base Address Registers) (مانند `aqa`، `asq`، `acq` و `cc`)، اقدامات مناسبی انجام می‌دهد. اگر مقدار `aqa` (اندازه صف‌های ارسال و دریافت) تغییر کند، صف مدیریتی جدید ایجاد می‌شود و مقادیر مربوط به وضعیت صف‌ها تنظیم می‌گردد. اگر مقادیر صف ارسال (`asq`) و صف تکمیل (`acq`) تغییر کند، حافظه مربوط به صف‌های ارسال و تکمیل تخصیص داده شده و تنظیمات لازم اعمال می‌شود. اگر رجیستر `cc` تغییر کند و فعال‌سازی (`en`) دستگاه مشخص شود، وضعیت آماده (`rdy`) تنظیم می‌گردد. همچنین در حالت خاموشی (`shn`) صف‌های مدیریت پاک‌سازی می‌شوند. در آخر نیز از دستورات مانند `smp_mb()` برای اطمینان از همگام‌سازی و جلوگیری از مشکلات در دسترسی‌های هم‌زمان استفاده می‌شود.
- **تابع `nvmev_pci_read`**: این تابع برای خواندن از حافظه استفاده می‌شود.
- **تابع `nvmev_pci_write`**: این تابع برای نوشتن داده در رجیسترهای یک دستگاه PCI طراحی شده است و به مدیریت قابلیت‌های INTx و MSI-X در دستگاه‌های PCI کمک می‌کند.
- **تابع `init_nvme_ctrl_regs`**: این تابع وظیفه مقداردهی اولیه و آماده‌سازی رجیسترهای کنترل‌کننده دستگاه را به‌عهده دارد.
- **تابع `create_pci_bus`**: این تابع bus مجازی PCI را ایجاد می‌کند.
- بقیه توابع برای مدیریت تنظیمات بخش‌های مختلف از PCI است.

گام دوم پروژه

ایجاد دیسک NVMeVirt_NVM و مشاهده‌ی عملکرد آن

برای اینکه بتوانیم یک دیسک را با استفاده از ابزار NVMeVirt شبیه‌سازی کنیم، طبق دستورعمل آورده شده در مخزن https://github.com/snu-csl/nvmevirt عمل می‌کنیم. پس در ابتدا لازم است که اطلاعات فایل grub را به این شکل تغییر دهیم:

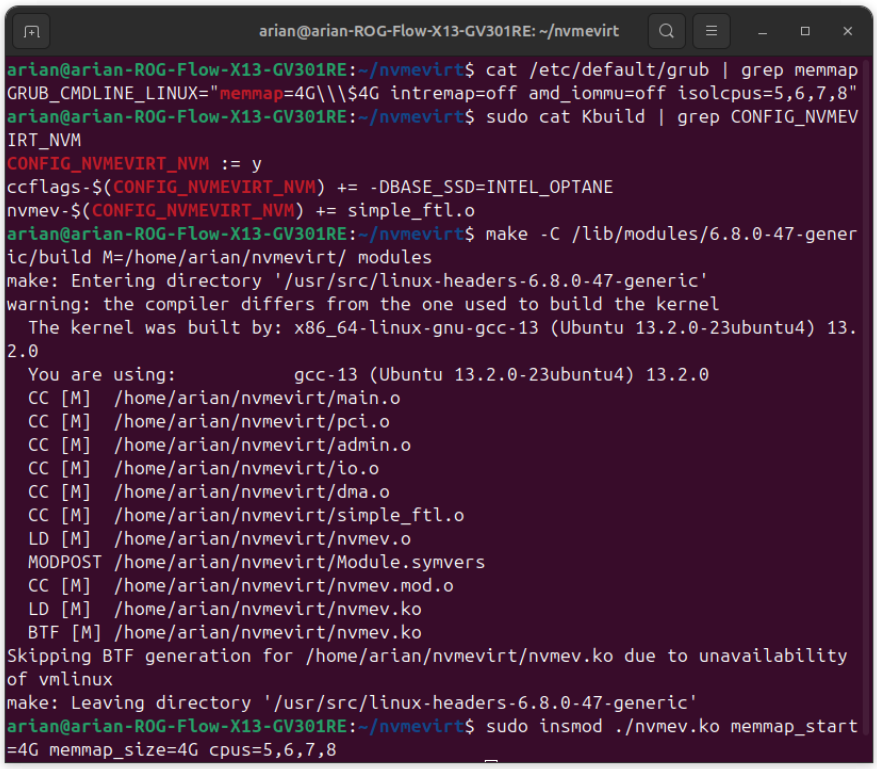
GRUB_CMDLINE_LINUX=”memmap=4G$4G intremap=off amd_iommu=off isolcpus=5,6,7,8”

با این دستور ۴گیگ حافظه را اختصاص داده و از آفست ۴گیگ شروع می‌کند. همچنین، IOMMU غیر فعال می‌شود که در کار شبیه‌ساز اختلال ایجاد نشود. پردازنده‌هایی که به شبیه‌ساز اختصاص داده می‌شود هم از دسترس برنامه‌ریز سیستم‌عالم دور می‌مانند که عملکرد شبیه‌ساز کاهش نیابد. و بعد آن را آپدیت کنیم:

```
sudo update-grub
sudo reboot
```

حال با دریافت فایل این شبیه‌ساز از مخزن و کامپایل آن می‌توانیم یک دیسک را شبیه‌سازی کنیم.

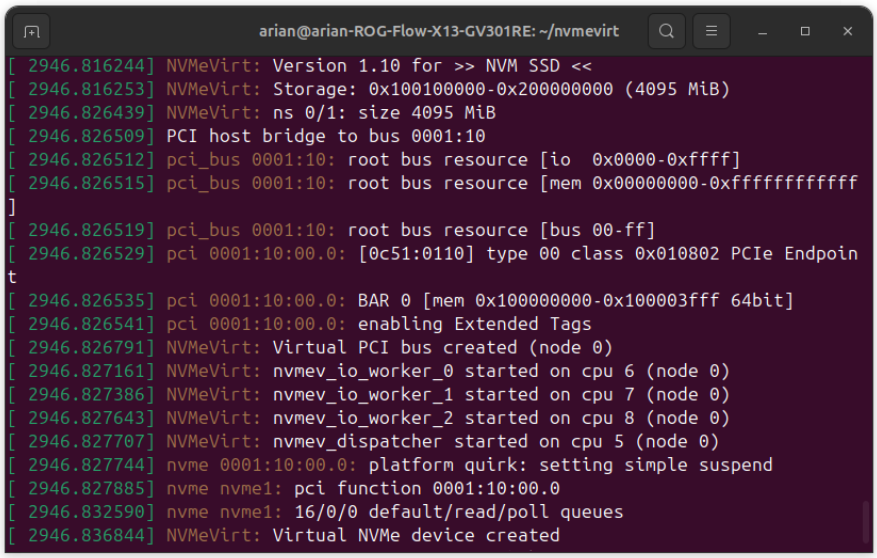
```
git clone https://github.com/snu-csl/nvmevirt
cd nvmevirt
make -C /lib/modules/6.8.0-47-generic/build M=/home/arian/nvmevirt/ modules
```



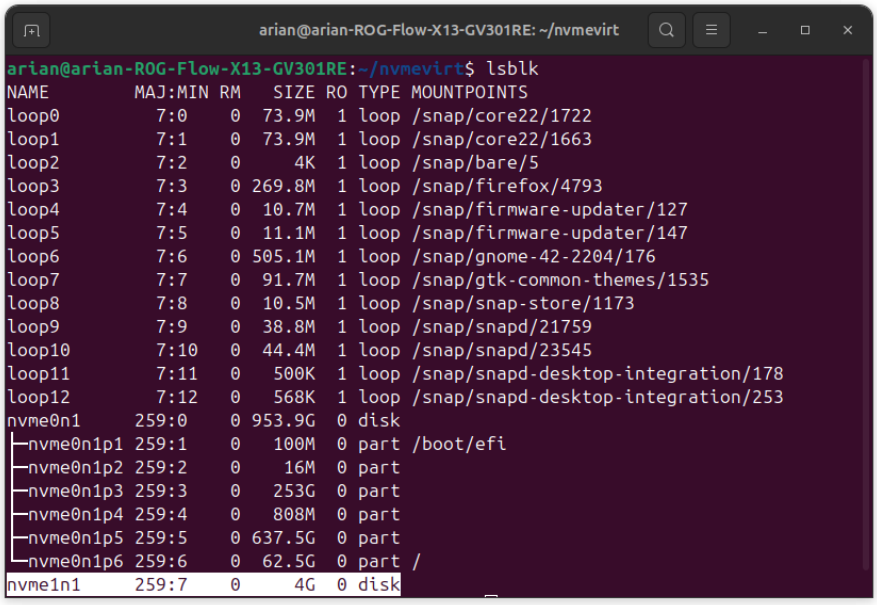
شکل ۲: تغییر فایل grub و ساخت دیسک

حال با اجرای دستور زیر دیسک شبیه‌سازی شده را ایجاد می‌کنیم:

```
sudo insmod ./nvmev.ko memmap_start=4G memmap_size=4G cpus=5,6,7,8
```



شکل ۳: بررسی لاگ‌های NVMeVirt



شکل ۴: مشاهده دیسک شبیه‌سازی شده با دستور lsblk

برای اینکه از صحت عملکرد این دیسک اطمینان حاصل کنیم، می‌توانیم با دستورات زیر فایل سیستم را روی آن ایجاد

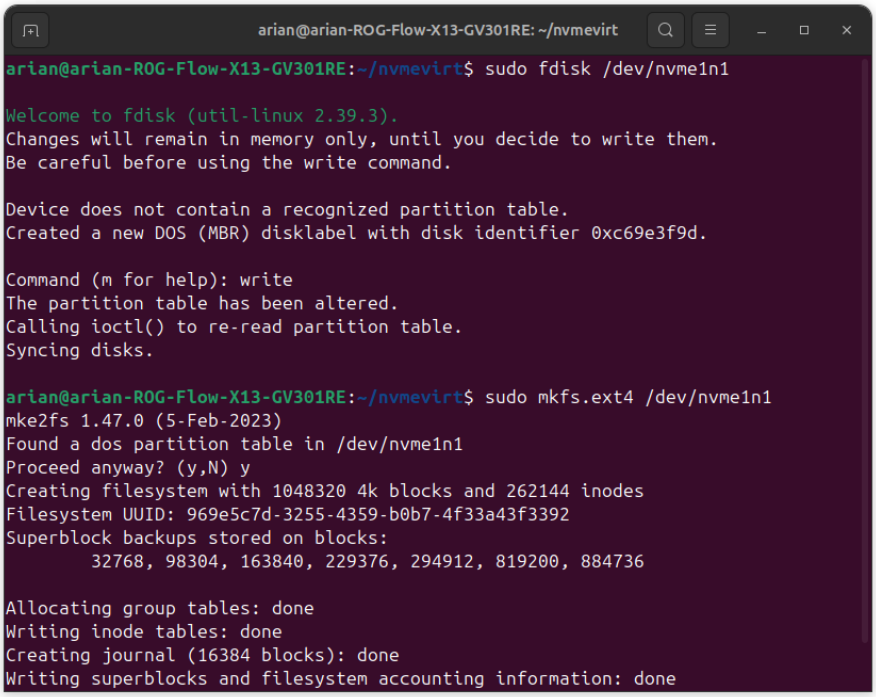
کنیم:

sudo fdisk /dev/nvme1n1

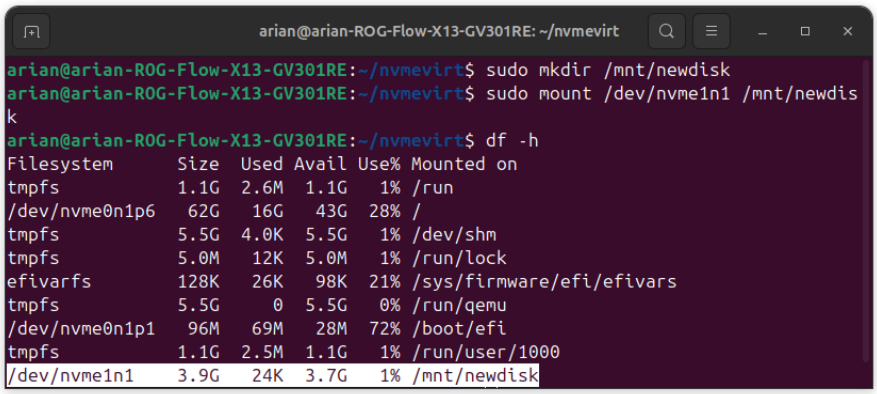
sudo mkfs.ext4 /dev/nvme1n1

sudo mkdir /mnt/newdisk

sudo mount /dev/nvme1n1 /mnt/newdisk



شکل ۵: فایل سیستم



شکل ۶: ایجاد فایل در دیسک شبیه‌سازی شده

گام سوم پروژه

برای خودکارسازی اجرای دستورات fio و ذخیره‌ی معیارهای مورد توجه، از اسکریپت پایتون زیر استفاده می‌کنیم. این اسکریپت، دستورهای مد نظر را با رجوع به جدول پارامترها برای یک ioengine خاص تشکیل می‌دهد. سپس معیارهای عملکردی را از خروجی json از fio دریافت و به فایل json نهایی اضافه می‌کند. برای spdk، باید علاوه بر تغییر ioengine، در ابتدای دستور پلاگین fio از spdk را preload کنیم که دستور به درستی کار کند. همچنین، آدرس درایو را باید از SPDK Transport Identifier بگیریم و مقدار PCI function را برای درایو داشته باشیم.

```
import subprocess
import json

def run_fio(iodepth, numjobs, test_type, size):
    cmd = [
        "sudo", "fio", "--name=test", "--filename=/dev/nvme1n1", "--dire
        "--rw={}".format(test_type), "--bs=4k", "--iodepth={}".format(io
        "--numjobs={}".format(numjobs), "--size={}".format(size),
        "--ioengine=libaio", "--group_reporting", "--norandommap=1", "--
    ]

    try:
        result = subprocess.run(cmd, capture_output=True, text=True, che
        return json.loads(result.stdout)
    except subprocess.CalledProcessError as e:
        print("Error running fio:", e)
        return None

def extract_metrics(fio_output):
    if not fio_output:
        return None

    jobs = fio_output.get("jobs", [])
    if not jobs:
        return None

    read_stats = jobs[0].get("read", {})
    write_stats = jobs[0].get("write", {})

    return {
        "read": {
            "iops": read_stats.get("iops", 0),
            "mean_latency_ns": read_stats.get("lat_ns", {}).get("mean",
```

```
            "p99_9_latency_ns": read_stats.get("clat_ns", {}).get("perc
        },
        "write": {
            "iops": write_stats.get("iops", 0),
            "mean_latency_ns": write_stats.get("lat_ns", {}).get("mean",
            "p99_9_latency_ns": write_stats.get("clat_ns", {}).get("perc
        }
    }

def main():
    parameters = [
        {"iodepth": 1, "numjobs": 1, "Type": "randrw", "size": "4G"},
        {"iodepth": 1, "numjobs": 4, "Type": "randrw", "size": "1G"},
        {"iodepth": 1, "numjobs": 1, "Type": "randwrite", "size": "4G"},
        {"iodepth": 1, "numjobs": 4, "Type": "randwrite", "size": "1G"},
        {"iodepth": 1, "numjobs": 1, "Type": "randread", "size": "4G"},
        {"iodepth": 1, "numjobs": 4, "Type": "randread", "size": "1G"},
    ]

    results = []

    for param in parameters:
        print(f"Running test: {param}")
        fio_output = run_fio(param["iodepth"], param["numjobs"], param["
        metrics = extract_metrics(fio_output)
        if metrics:
            results.append({"parameters": param, "metrics": metrics})
        else:
            results.append({"parameters": param, "metrics": "Failed to e

    with open("libaio_results.json", "w") as f:
        json.dump(results, f, indent=4)

if __name__ == "__main__":
    main()
```

سپس با اسکریپت جداگانه‌ای، از خروجی‌های سه روش به‌دست آمده، نمودار رسم می‌کنیم. این اسکریپت، برای سه مولفه‌ی ذکر شده در حالات خواندن و نوشتن، در مجموع شش نمودار رسم می‌کند که در هر کدام، یک مولفه میان سه روش ioengine مقایسه می‌شود. عکس نمودارها، اسکریپت‌های اجرای fio، اسکریپت رسم نمودار و خروجی‌های خام آزمون‌ها همگی به پیوست ارسال شده‌اند.

```
import json
import matplotlib.pyplot as plt
import numpy as np

def load_results(file_name):
    with open(file_name, "r") as f:
        return json.load(f)

def extract_metrics(data):
    metrics = {"read": {"iops": [], "mean_latency_ns": [], "p99_9_latenc
               "write": {"iops": [], "mean_latency_ns": [], "p99_9_laten
    labels = []

    for entry in data:
        param_label = f"iodepth={entry['parameters']['iodepth']}, numjob
        labels.append(param_label)

        for op in ["read", "write"]:
            metrics[op]["iops"].append(entry["metrics"][op]["iops"])
            metrics[op]["mean_latency_ns"].append(entry["metrics"][op]["
            metrics[op]["p99_9_latency_ns"].append(entry["metrics"][op][

    return metrics, labels

def plot_metrics(metrics_dict, labels):
    categories = ["iops", "mean_latency_ns", "p99_9_latency_ns"]
    operations = ["read", "write"]

    x = np.arange(len(labels))
    width = 0.2  # Bar width

    fig, axes = plt.subplots(2, 3, figsize=(18, 10))
    axes = axes.flatten()

    for idx, (op, category) in enumerate([(op, cat) for op in operations
        ax = axes[idx]
        for i, (engine, metrics) in enumerate(metrics_dict.items()):
            ax.bar(x + i * width, metrics[op][category], width, label=en

        ax.set_title(f"{op.capitalize()} {category.replace('_', ' ')}")
```

```
        ax.set_xticks(x + width)
        ax.set_xticklabels(labels, rotation=45, ha="right")
        ax.set_ylabel(category)
        ax.legend()

    plt.tight_layout()
    plt.show()

def main():
    file_names = {"libaio": "libaio_results.json", "io_uring": "io_uring
    metrics_dict = {}
    labels = None

    for engine, file in file_names.items():
        data = load_results(file)
        metrics, labels = extract_metrics(data)
        metrics_dict[engine] = metrics

    plot_metrics(metrics_dict, labels)

if __name__ == "__main__":
    main()
```

گام چهارم پروژه

نصب پلتفرم SPDK و استفاده از پلاگین FIO برای آزمایش

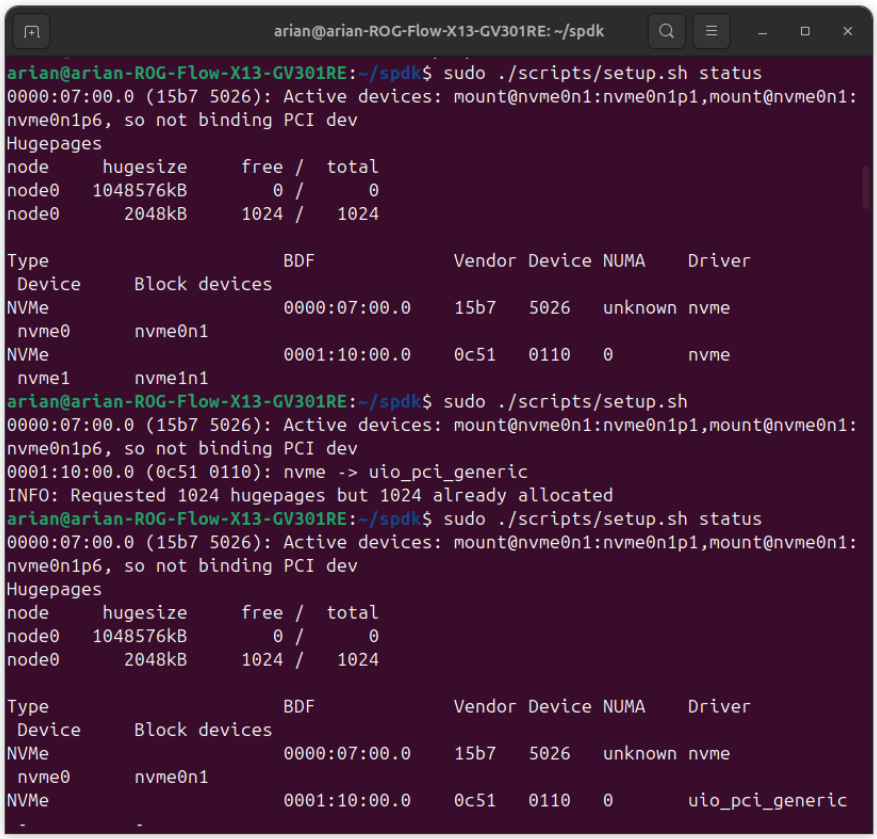
SPDK یک مجموعه ابزار و کتابخانه برای بهینه‌سازی عملکرد ذخیره‌سازی در سیستم‌های NVMe و دیگر سخت‌افزارهای با کارایی بالا است. حالا مراحل نصب و راه‌اندازی SPDK را توضیح می‌دهیم: باید ابتدا با دستور git clone https://github.com/spdk/spdk کدهای منبع را از گیت‌هاب دریافت کرده و با دستور cd spdk وارد دایرکتوری آن شده و با دستورات زیر پکیج‌های مورد نیاز آن را نصب می‌کنیم:

```
git submodule update –init
sudo ./scripts/pkgdep.sh –all
sudo apt-get install libfdt-dev libpcap-dev python3-sphinx meson python3-pyelftools libbsd-dev libarchive-dev libjansson-dev
```

در نهایت آن را پیکربندی و کامپایل می‌کنیم:

```
./configure –with-fio=/home/arian/fio
make -j13
```

و برای تخصیص منابع حافظه مثل دیسک شبیه‌سازی شده از دستور sudo ./scripts/setup.sh استفاده می‌کنیم.



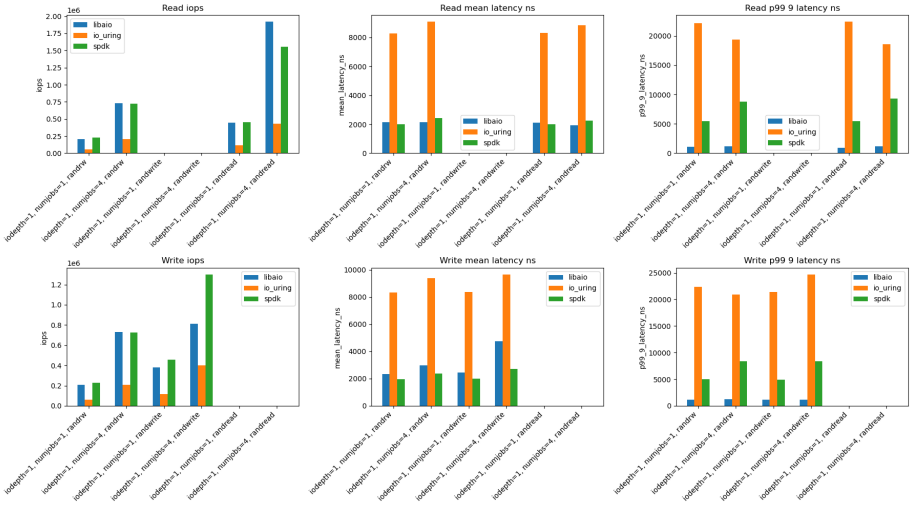
شکل ۷: تخصیص دیسک شبیه‌سازی شده با ابزار SPDK

گام پنجم پروژه

تحلیل و مقایسه‌ی نتایج و تهیه‌ی نمودار نمودار

در این پروژه، هدف شبیه‌سازی عملکرد ذخیره‌سازی NVMe با استفاده از ابزارهای مختلف و ارزیابی کارایی آن‌ها در شرایط مختلف است. مراحل این پروژه به تفصیل توضیح داده شد نتایج به‌دست آمده از آزمون‌های عملکردی باید تجزیه و تحلیل شوند. با توجه به نتایج آزمون‌ها، می‌توانیم بررسی کنیم که چگونه پارامترهای مختلف تأثیر گذاشته‌اند و کدام تنظیمات برای دستیابی به بهترین عملکرد مناسب‌تر هستند. این نمودارها می‌توانند نشان دهند که کدام ترکیب از پارامترها بیشترین کارایی را در شرایط خاص به دست می‌آورد.

با انجام این گام‌ها و تحلیل نتایج به‌دست آمده، می‌توان به درک بهتری از نحوه عملکرد شبیه‌سازی ذخیره‌سازی NVMe و بهینه‌سازی آن در محیط‌های مجازی رسید. همچنین، می‌توان عملکرد ذخیره‌سازی را در شرایط مختلف تست کرده و به بهینه‌سازی سیستم‌های ذخیره‌سازی با کارایی بالا کمک کرد. در نهایت، با استفاده از نمودارهای مقایسه‌ای می‌توان عملکرد مختلف تنظیمات و پیکربندی‌ها را به وضوح نمایش داد.



شکل ۸: نمودارهای مقایسه نتایج بدست آمده از آزمون‌های عملکردی

بخش امتیازی

اجرای برنامه‌ای محک مانند db_bench روی RocksDB با استفاده از SPDK

برای این بخش نیاز به نصب دو ابزار RocksDB و SPDK داریم. ابتدا با استفاده از دو دستور زیر پکیج‌های مورد نیاز را نصب می‌کنیم:

sudo pacman -S –needed git cmake make gcc ninja clang llvm boost
sudo pacman -S –needed gflags snappy zlib bzip2 lz4

حال با اجرای به ترتیب دستورات زیر باید RocksDB را ابتدا از مخزن آن دانلود کرده و سپس با دستورات زیر آن را کامپایل می‌کنیم:

git clone https://github.com/facebook/rocksdb.git
cd rocksdb
mkdir -p build cd build
cmake -DCMAKE$_BUILD_TYPE = Release - DWITH_SPDK = ON$..
make -j4
sudo make install

حال با استفاده از دستور زیر چک می‌کنیم که به درستی نصب انجام شده باشد:

./dbbench –version

حال باید ابزار SPDK را از مخزن آن دریافت و با دستورات زیر آن را کامپایل کنیم:

```
~/OS-project 13:22 git clone https://github.com/facebook/rocksdb.git
Cloning into 'rocksdb'...
remote: Enumerating objects: 136668, done.
remote: Counting objects: 100% (231/231), done.
remote: Compressing objects: 100% (154/154), done.
remote: Total 136668 (delta 137), reused 83 (delta 77), pack-reused 136437 (from 5)
Receiving objects: 100% (136668/136668), 220.27 MiB | 627.00 KiB/s, done.
Resolving deltas: 100% (104714/104714), done.
```

شکل ۹: مراحل نصب RocksDB

```
…/spdk spdk master v14.2.1 v3.13.1 02:21 cd rocksdb/

…/spdk/rocksdb spdk master v3.31.5 02:21 mkdir -p build && cd build

…/spdk/rocksdb/build spdk master 02:21 cmake -DCMAKE_BUILD_TYPE=Release -DWITH_SPDK=ON ..
-- The CXX compiler identification is GNU 14.2.1
-- The C compiler identification is GNU 14.2.1
-- The ASM compiler identification is GNU
-- Found assembler: /usr/bin/cc
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Performing Test HAVE_OMIT_LEAF_FRAME_POINTER
-- Performing Test HAVE_OMIT_LEAF_FRAME_POINTER - Success
-- Performing Test BUILTIN_ATOMIC
-- Performing Test BUILTIN_ATOMIC - Success
-- Could NOT find uring (missing: uring_LIBRARIES)
-- Enabling RTTI in Debug builds only (default)
```

شکل ۱۰: مراحل نصب RocksDB

```
git clone https://github.com/spdk/spdk.git
cd spdk
sudo pacman -S –needed libaio numactl
./scripts/pkgdep.sh
./configure –with-rocksdb –enable-debug
make -j4
```

همچنین برای لود کرنل ماژول‌های SPDK از دستورات زیر استفاده می‌کنیم:

```
sudo modprobe uio
sudo insmod build/lib/spdk uio.ko
sudo scripts/setup.sh
```

حال با استفاده از دستور زیر برنامه محک db_bench را با SPDK اجرا می‌کنیم:

```
./db bench –use spdk=1 –benchmarks=fillseq,readrandom –num=1000000 –value size=4096
– threads=4
```

```
…/spdk/rocksdb/build  spdk  master  v3.31.5  02:21  make -j4
[  0%] Building CXX object third-party/gtest-1.8.1/fused-src/gtest/CMakeFiles/gtest.dir/gtest-all.cc.o
[  0%] Building CXX object CMakeFiles/rocksdb.dir/cache/cache.cc.o
[  0%] Building CXX object CMakeFiles/rocksdb-shared.dir/cache/cache.cc.o
[  1%] Building CXX object CMakeFiles/rocksdb.dir/cache/cache_entry_roles.cc.o
[  2%] Building CXX object CMakeFiles/rocksdb-shared.dir/cache/cache_entry_roles.cc.o
[  2%] Building CXX object CMakeFiles/rocksdb-shared.dir/cache/cache_key.cc.o
[  2%] Building CXX object CMakeFiles/rocksdb.dir/cache/cache_key.cc.o
[  2%] Building CXX object CMakeFiles/rocksdb.dir/cache/cache_helpers.cc.o
[  2%] Building CXX object CMakeFiles/rocksdb-shared.dir/cache/cache_helpers.cc.o
[  2%] Building CXX object CMakeFiles/rocksdb.dir/cache/cache_reservation_manager.cc.o
[  2%] Building CXX object CMakeFiles/rocksdb.dir/cache/charged_cache.cc.o
[  2%] Building CXX object CMakeFiles/rocksdb-shared.dir/cache/cache_reservation_manager.cc.o
[  2%] Building CXX object CMakeFiles/rocksdb.dir/cache/clock_cache.cc.o
[  2%] Building CXX object CMakeFiles/rocksdb.dir/cache/compressed_secondary_cache.cc.o
[  2%] Building CXX object CMakeFiles/rocksdb-shared.dir/cache/charged_cache.cc.o
[  2%] Building CXX object CMakeFiles/rocksdb-shared.dir/cache/clock_cache.cc.o
[  2%] Building CXX object CMakeFiles/rocksdb.dir/cache/lru_cache.cc.o
[  3%] Building CXX object CMakeFiles/rocksdb.dir/cache/secondary_cache.cc.o
[  3%] Building CXX object CMakeFiles/rocksdb-shared.dir/cache/compressed_secondary_cache.cc.o
[  3%] Linking CXX static library libgtest.a
[  3%] Built target gtest
[  3%] Building CXX object CMakeFiles/rocksdb.dir/cache/secondary_cache_adapter.cc.o
[  4%] Building CXX object CMakeFiles/rocksdb-shared.dir/cache/lru_cache.cc.o
[  5%] Building CXX object CMakeFiles/testharness.dir/test_util/mock_time_env.cc.o
```

شکل ۱۱: مراحل نصب RocksDB

```
…/spdk/rocksdb/build  spdk  master  v3.31.5  02:33  sudo make install -j4
[sudo] password for melika:
[  0%] Built target gtest
[  1%] Built target testharness
[ 47%] Built target rocksdb
[ 93%] Built target rocksdb-shared
[ 93%] Built target range_del_aggregator_bench
[ 93%] Built target memtablerep_bench
[ 93%] Built target cache_bench
[ 94%] Built target db_bench
[ 94%] Built target hash_table_bench
[ 94%] Built target block_cache_trace_analyzer
[ 94%] Built target filter_bench
[ 94%] Built target table_reader_bench
[ 95%] Built target ldb
[ 95%] Built target sst_dump
[ 95%] Built target trace_analyzer
[ 95%] Built target db_sanity_test
[ 97%] Built target rocksdb_dump
[ 97%] Built target rocksdb_undump
[ 97%] Built target write_stress
[ 97%] Built target db_repl_stress
[100%] Built target db_stress
Install the project...
-- Install configuration: "Release"
```

شکل ۱۲: مراحل نصب RocksDB

نتیجه خروجی در تصویر ۱۴ آمده است که به ترتیب نشان‌دهنده throughput و latancy است در حالت تست نوشتن ترتیبی و تست خواندن تصادفی است.

شکل ۱۳: تست نسخه نصب شده

شکل ۱۴: نتیجه اجرای برنامه محک