



دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر

عنوان:

پروژه درس سیستم عامل

نگارش

کیارش جولایی

ایمان احمدی

سبحان اسدی

متین عارف نیا

استاد

دکتر جلیلی ، دکتر اسدی

۱۴۰۳ بهمن

۱ چکیده

این تحقیق به بررسی و مقایسه عملکرد ابزارهای مختلف دسترسی به ذخیره‌سازی در سیستم‌های مدرن پرداخته است. در این پژوهه، چهار فناوری عمدۀ شامل *SPDK*, *io_uring*, *libaio*, و *NVMe* به همراه یک شبیه‌ساز (*NVMeVirt*) برای آزمایش عملکرد در پردازش‌های ورودی/خروجی (*I/O*) ارزیابی شده‌اند. آزمایش‌ها با استفاده از ابزار بنچمارک *FIO* انجام شده و عملکرد این فناوری‌ها در شاخص‌های مختلفی مانند IOPS (تعداد عملیات ورودی/خروجی در ثانیه)، تأخیر (*Latency*), تأخیر انتهایی (*Tail Latency*) و پهنای باند (*Bandwidth*) بررسی گردیده است. نتایج نشان می‌دهند که هر کدام از این ابزارها بسته به نوع بار کاری و پارامترهای مختلف عملکرد متفاوتی دارند. در حالی که *SPDK* به دلیل استفاده از درایورهای کاربر-محور و دور زدن کرنل، عملکرد بسیار بهتری در پردازش‌های با بار بالا دارد، *libaio* در بارهای کاری با درخواست‌های تصادفی کوچک عملکرد بهتری را نشان می‌دهد. در نهایت، پیشنهاداتی برای بهینه‌سازی عملکرد و ترکیب این ابزارها برای بهبود کارایی سیستم‌های ذخیره‌سازی ارائه شده است.

اهداف و اهمیت تحقیق

هدف این تحقیق، بررسی و تحلیل عملکرد سیستم‌های ذخیره‌سازی مختلف از طریق مقایسه سه تکنولوژی مهم در دسترسی به ذخیره‌سازی یعنی *SPDK*، *io_uring*، *libaio* می‌باشد. استفاده از این تکنولوژی‌ها در مقایسه با روش‌های سنتی مانند فراخوان‌های سیستمی *POSIX* و دیگر ابزارهای مدرن می‌تواند به درک بهتری از محدودیت‌ها و پتانسیل‌های هر کدام کمک کند. این تحقیق می‌تواند راه حل‌های جدیدی برای بهینه‌سازی عملکرد سیستم‌های ذخیره‌سازی در مراکز داده و برنامه‌های با نیاز به کارایی بالا پیشنهاد دهد.

در دنیای امروز، کارایی ذخیره‌سازی در بسیاری از برنامه‌ها و سیستم‌های مدرن از اهمیت ویژه‌ای برخوردار است. به ویژه در سیستم‌های پردازش داده‌های کلان و زیرساخت‌های ابری که نیاز به ذخیره‌سازی سریع و کارآمد دارند، انتخاب تکنولوژی‌های دسترسی به ذخیره‌سازی به شدت تأثیرگذار است. این تحقیق به بررسی عملکرد این تکنولوژی‌ها در موقعیت‌های مختلف پرداخته و مقایسه‌ای دقیق ارائه می‌دهد که می‌تواند راهنمایی برای انتخاب بهترین ابزار در سناریوهای مختلف باشد.

سیستم‌های ذخیره‌سازی داده‌ها به عنوان یکی از بخش‌های حیاتی عملکرد کلی سیستم‌های رایانه‌ای محسوب می‌شوند. بهبود بهره‌وری این سیستم‌ها در سرعت خواندن و نوشتن اطلاعات از اهمیت ویژه‌ای برخوردار است. در این پژوهه، ابزار *NVMeVirt* مورد بررسی قرار گرفته و عملکرد آن از طریق ابزارهای *FIO* و *SPDK* سنجیده شده است.

مطالعه روش‌های مختلف دسترسی به ذخیره‌سازی

در سیستم‌عامل‌های مدرن، چندین روش برای دسترسی به دیسک‌های ذخیره‌سازی وجود دارد:

POSIX I/O

این روش کلاسیک، از فراخوان‌های سیستمی مانند *read()* و *write()* استفاده می‌کند که موجب ایجاد سربار فراخوانی سیستم و مدیریت صفحه‌ای ورودی/خروجی در کرنل می‌شود.

libaio

کتابخانه *libaio* به منظور اجرای عملیات *I/O* به صورت همزمان طراحی شده و باعث کاهش تأخیر ناشی از انتظار در صفحه‌ای پردازشی کرنل می‌شود. این کتابخانه امکان مدیریت درخواست‌های ورودی/خروجی را از طریق مکانیزم *event-driven* فراهم می‌کند.

io_uring

io_uring یکی از جدیدترین فناوری‌های بهینه‌سازی دسترسی به دیسک در سیستم‌عامل لینوکس است که با بهره‌گیری از صفحه‌ای حلقوی *ring buffer*، تعاملات ورودی/خروجی را با کمترین سربار پردازشی انجام می‌دهد.

SPDK

یک مجموعه ابزار توسعه است که به منظور حذف سربار فراخوان‌های سیستمی طراحی شده و امکان برقراری ارتباط مستقیم با سخت‌افزار ذخیره‌سازی را بدون دخالت کرنل فراهم می‌کند.

توضیحات

شبیه‌ساز *NVMeVirt* یک ماژول کرنل است که یک دستگاه ذخیره‌سازی مجازی را شبیه‌سازی کرده و به عنوان یک دستگاه *NVMe* برای سیستم ظاهر می‌شود. این شبیه‌ساز در سطح *PCIe* پیاده‌سازی شده و امکان ارزیابی عملکرد و توسعه روش‌های بهینه‌سازی سیستم‌های ذخیره‌سازی را فراهم می‌کند.

نحوه تعریف *PCIeRoot*

شبیه‌ساز *NVMeVirt* در لایه *PCIe* عمل می‌کند و برای تعریف آن، از مدل‌های *PCIeRoot* موجود در کرنل لینوکس استفاده می‌شود. این شبیه‌سازی در لایه *bus* انجام شده و به عنوان یک دستگاه مجازی به سیستم عامل معرفی می‌شود.

مطالعه کدهای مربوطه و تهیه مستندات

کد منبع *NVMeVirt* شامل چندین فایل کلیدی است که نحوه ارتباط آن با سیستم را مشخص می‌کنند:

- *pci.c*: مدیریت ارتباط *PCIe* و پیاده‌سازی شبیه‌سازی دستگاه *NVMe*.

- *nvmev.h*: شامل تعاریف کلی و پارامترهای کنترلی ماژول.

- *main.c*: شامل بخش اصلی پیاده‌سازی پردازش ورودی/خروجی و مدیریت حافظه.

نحوه کارکرد شبیه‌ساز

شبیه‌ساز *NVMeVirt* یک ماژول کرنل است که از مکانیزم‌های مدیریت حافظه برای اختصاص فضای فیزیکی به عنوان فضای ذخیره‌سازی شبیه‌سازی شده استفاده می‌کند. این ماژول درخواست‌های ورودی/خروجی را پردازش کرده و آن‌ها را در حافظه اصلی نگهداری می‌کند.

معماری و بخش‌های اساسی

معماری *NVMeVirt* از بخش‌های زیر تشکیل شده است:

- **مدیریت حافظه:** اختصاص و مدیریت فضای حافظه برای شبیه‌سازی دستگاه ذخیره‌سازی.

- مدیریت *PCIe*: ارتباط با زیرسیستم *PCIe* و شبیه‌سازی دستورات *NVMe*
- مدیریت صفحه‌های ورودی/خروجی: پردازش درخواست‌های *I/O* و اجرای آن‌ها به صورت شبیه‌سازی شده.

سطح شبیه‌سازی

شبیه‌ساز *NVMeVirt* در سطح *PCIe* عمل می‌کند و به عنوان یک دیسک مجازی *NVMe* در سیستم شناخته می‌شود. این شبیه‌ساز از حافظه فیزیکی سیستم برای ذخیره داده‌ها استفاده کرده و امکان تعامل مستقیم برنامه‌های کاربر و کرنل را فراهم می‌کند.

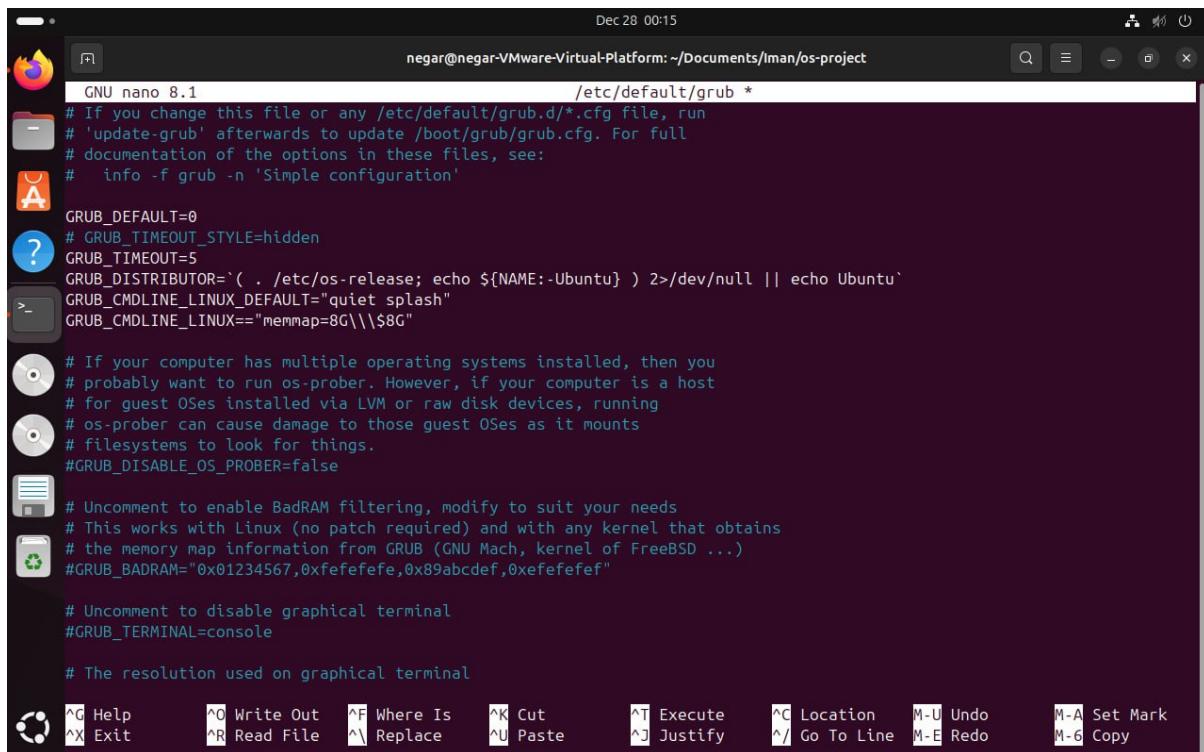
مکان ذخیره‌سازی داده‌ها

داده‌های ذخیره‌شده توسط *NVMeVirt* در حافظه اصلی قرار می‌گیرند و از طریق پارامترهای `memmap_start` و `memmap_size` قابل تنظیم هستند. این روش باعث افزایش سرعت دسترسی به داده‌ها می‌شود اما ممکن است میزان حافظه موجود در سیستم را کاهش دهد.

کنترل سرعت دسترسی و شبیه‌سازی دیسک سرعت بالا

برای شبیه‌سازی یک دستگاه ذخیره‌سازی پسرعت بر روی سیستمی با دیسک کنترل، می‌توان از تنظیمات نرخ ورودی/خروجی استفاده کرد. این امر با تغییر اندازه *queue depth* و استفاده از برنامه‌های آزمون مانند *FIO* امکان‌پذیر است.

مراحل نصب و شبیه سازی



Dec 28 00:15
negar@negar-VMware-Virtual-Platform: ~/Documents/lman/os-project /etc/default/grub *

```
GNU nano 8.1
# If you change this file or any /etc/default/grub.d/*.cfg file, run
# 'update-grub' afterwards to update /boot/grub/grub.cfg. For full
# documentation of the options in these files, see:
#   info -f grub -n 'Simple configuration'

GRUB_DEFAULT=0
# GRUB_TIMEOUT_STYLE=hidden
GRUB_TIMEOUT=5
GRUB_DISTRIBUTOR='( . /etc/os-release; echo ${NAME:-Ubuntu} ) 2>/dev/null || echo Ubuntu'
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"
GRUB_CMDLINE_LINUX=="memmap=8G\\\$8G"

# If your computer has multiple operating systems installed, then you
# probably want to run os-prober. However, if your computer is a host
# for guest OSes installed via LVM or raw disk devices, running
# os-prober can cause damage to those guest OSes as it mounts
# filesystems to look for things.
#GRUB_DISABLE_OS_PROBER=false

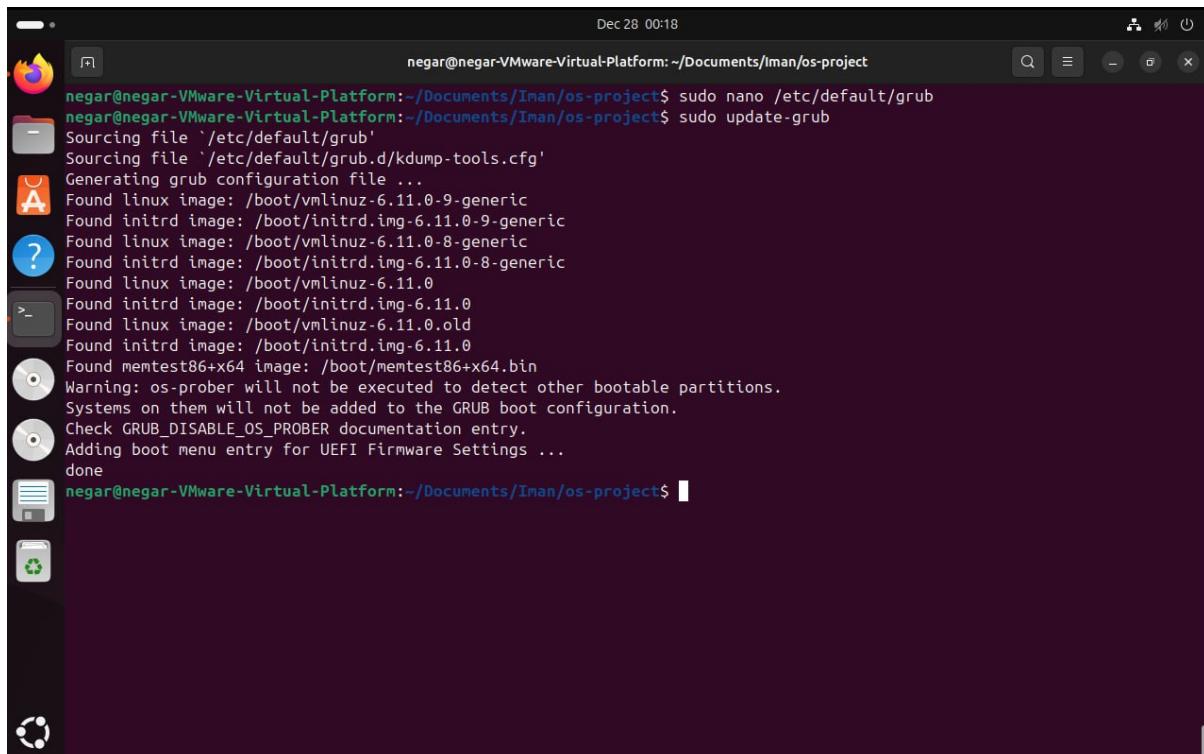
# Uncomment to enable BadRAM filtering, modify to suit your needs
# This works with Linux (no patch required) and with any kernel that obtains
# the memory map information from GRUB (GNU Mach, kernel of FreeBSD ...)
#GRUB_BADRAM='0x01234567,0xfefefefe,0x89abcdef,0xefefefef'

# Uncomment to disable graphical terminal
#GRUB_TERMINAL=console

# The resolution used on graphical terminal

^O Help      ^O Write Out    ^F Where Is     ^K Cut        ^T Execute     ^C Location    M-U Undo     M-A Set Mark
^X Exit      ^R Read File    ^\ Replace     ^U Paste      ^J Justify     ^/ Go To Line  M-E Redo     M-G Copy
```

شکل ۱: GRUB



Dec 28 00:18
negar@negar-VMware-Virtual-Platform: ~/Documents/lman/os-project

```
negar@negar-VMware-Virtual-Platform: ~/Documents/lman/os-project$ sudo nano /etc/default/grub
negar@negar-VMware-Virtual-Platform: ~/Documents/lman/os-project$ sudo update-grub
Sourcing file '/etc/default/grub'
Sourcing file '/etc/default/grub.d/kdump-tools.cfg'
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-6.11.0-9-generic
Found initrd image: /boot/initrd.img-6.11.0-9-generic
Found linux image: /boot/vmlinuz-6.11.0-8-generic
Found initrd image: /boot/initrd.img-6.11.0-8-generic
Found linux image: /boot/vmlinuz-6.11.0
Found initrd image: /boot/initrd.img-6.11.0
Found linux image: /boot/vmlinuz-6.11.0.old
Found initrd image: /boot/initrd.img-6.11.0
Found memtest86+x64 image: /boot/memtest86+x64.bin
Warning: os-prober will not be executed to detect other bootable partitions.
Systems on them will not be added to the GRUB boot configuration.
Check GRUB_DISABLE_OS_PROBER documentation entry.
Adding boot menu entry for UEFI Firmware Settings ...
done
```

شکل ۲: آپدیت GRUB و در ادامه، ریبوت

```
negar@negar-VMware-Virtual-Platform:~/Documents/Iman/os-project$ sudo dmesg | grep "memmap"
[ 0.00000] Command line: BOOT_IMAGE=/boot/vmlinuz-6.11.0-9-generic root=UUID=01e039df-46a6-45f1-8f9a-9bcfb08eeb4 ro
[ 0.00000] =memmap=8G$8G quiet splash crashkernel=2G-4G:320M,4G-32G:512M,32G-64G:1024M,64G-128G:2048M,128G-:4096M
[ 0.767397] Kernel command line: BOOT_IMAGE=/boot/vmlinuz-6.11.0-9-generic root=UUID=01e039df-46a6-45f1-8f9a-9bcfb08eeb4 ro
[ 0.767397] =memmap=8G$8G quiet splash crashkernel=2G-4G:320M,4G-32G:512M,32G-64G:1024M,64G-128G:2048M,128G-:4096M
[ 0.767574] Unknown kernel command line parameters "splash BOOT_IMAGE=/boot/vmlinuz-6.11.0-9-generic =memmap=8G$8G",
will be passed to user space.
[ 1.305683] HugeTLB: 16380 KiB memmap can be freed for a 1.00 GiB page
[ 1.305683] HugeTLB: 28 KiB memmap can be freed for a 2.00 MiB page
[ 3.415891] =memmap=8G$8G
negar@negar-VMware-Virtual-Platform:~/Documents/Iman/os-project$
```

شکل ۳: بررسی تغییرات انجام شده

```
negar@negar-VMware-Virtual-Platform:~/Documents/Iman/os-project/nvmevirt$ sudo insmod ./nvmev.ko memmap_start=8G memmap_size=8G cpus=7,8
negar@negar-VMware-Virtual-Platform:~/Documents/Iman/os-project/nvmevirt$ sudo insmod ./nvmev.ko memmap_start=8G memmap_size=8G cpus=7,8
insmod: ERROR: could not insert module ./nvmev.ko: File exists
negar@negar-VMware-Virtual-Platform:~/Documents/Iman/os-project/nvmevirt$ lsmod | grep nvmev
nvmev      53248  0
negar@negar-VMware-Virtual-Platform:~/Documents/Iman/os-project/nvmevirt$ sudo dmesg
[ 0.00000] Linux version 6.11.0-9-generic (buildd@lcy02-amd64-093) (x86_64-linux-gnu-gcc-14 (Ubuntu 14.2.0-4ubuntu2) 14.2.0, GNU ld (GNU Binutils for Ubuntu) 2.43.1) #9-Ubuntu SMP PREEMPT_DYNAMIC Mon Oct 14 13:19:59 UTC 2024 (Ubuntu 6.1.0-9.9-generic 6.11.0)
[ 0.00000] Command Line: BOOT_IMAGE=/boot/vmlinuz-6.11.0-9-generic root=UUID=01e039df-46a6-45f1-8f9a-9bcfb08eeb4 ro
memmap=8G$8G quiet splash crashkernel=2G-4G:320M,4G-32G:512M,32G-64G:1024M,64G-128G:2048M,128G-:4096M
[ 0.00000] KERNEL supported cpus:
[ 0.00000]   Intel_GenuineIntel
[ 0.00000]   AMD_AuthenticAMD
[ 0.00000]   Hygon_HygonGenuine
[ 0.00000]   Centaur_CentaurHauls
[ 0.00000]   zhaoxin_Shanghai
[ 0.00000] BIOS-provided physical RAM map:
[ 0.00000] BIOS-e820: [mem 0x0000000000000000-0x000000000000e7ff] usable
[ 0.00000] BIOS-e820: [mem 0x000000000009e800-0x000000000009ffff] reserved
[ 0.00000] BIOS-e820: [mem 0x00000000000dc000-0x00000000000ffff] reserved
[ 0.00000] BIOS-e820: [mem 0x000000000010000-0x000000000bfecffff] usable
[ 0.00000] BIOS-e820: [mem 0x000000000bfed0000-0x000000000bfefeffff] ACPI data
[ 0.00000] BIOS-e820: [mem 0x000000000bf00000-0x000000000bfeffffff] ACPI NVS
[ 0.00000] BIOS-e820: [mem 0x000000000bf00000-0x000000000bfeffffff] usable
[ 0.00000] BIOS-e820: [mem 0x000000000f000000-0x000000000f7fffffff] reserved
[ 0.00000] BIOS-e820: [mem 0x000000000fec0000-0x000000000fec0ffff] reserved
[ 0.00000] BIOS-e820: [mem 0x000000000fee0000-0x000000000fee0ffff] reserved
```

شکل ۴: بارگذاری ماژول

```

negar@negar-VMware-Virtual-Platform:~/Documents/iman/os-project/nvmevirt
peer_label="unconfined"
    exe="/usr/bin/dbus-daemon" sauid=101 hostname=? addr=? terminal=?
[ 251.546368] audit: type=1107 audit(1738187937.567:180): pid=791 uid=101 aid=4294967295 ses=4294967295 subj=unconfine
d msg='apparmor="DENIED" operation="dbus_method_call" bus="system" path="/org/freedesktop/timedate1" interface="org.fre
edesktop.DBus.Properties" member="GetAll" mask="send" name=:1.122' pid=3787 label="snap.firefox.firefox" peer_pid=4069
peer_label="unconfined"
    exe="/usr/bin/dbus-daemon" sauid=101 hostname=? addr=? terminal=?
[ 305.801429] workqueue: pcpu_balance_workfn hogged CPU for >10000us 19 times, consider switching to WQ_UNBOUND
[ 306.615754] workqueue: psi_avgs_work hogged CPU for >10000us 11 times, consider switching to WQ_UNBOUND
[ 448.278485] workqueue: blk_mq_run_work_fn hogged CPU for >10000us 11 times, consider switching to WQ_UNBOUND
[ 486.893300] workqueue: e1000_watchdog [e1000] hogged CPU for >10000us 4 times, consider switching to WQ_UNBOUND
[ 497.343916] workqueue: e1000_watchdog [e1000] hogged CPU for >10000us 5 times, consider switching to WQ_UNBOUND
[ 508.868595] nvmev: loading out-of-tree module taints kernel.
[ 508.868697] nvmev: module verification failed: signature and/or required key missing - tainting kernel
[ 508.870911] NVMeVirt: Version 1.10 for >> NVM SSD <<
[ 508.870919] NVMeVirt: Storage: 0x200100000-0x400000000 (8191 MiB)
[ 509.008749] NVMeVirt: ns 0/1: size 8191 MiB
[ 509.041128] PCI host bridge to bus 0001:00
[ 509.041546] pci_bus 0001:00: root bus resource [io 0x0000-0xffff]
[ 509.041588] pci_bus 0001:00: root bus resource [mem 0x00000000-0xffffffff]
[ 509.041588] pci_bus 0001:00: root bus resource [bus 00-ff]
[ 509.041830] pci 0001:00:00.0: [0c51:0110] type 00 class 0x010802 PCIe Endpoint
[ 509.041858] pci 0001:00:00.0: BAR 0 [mem 0x200000000-0x200003fff 64bit]
[ 509.041875] pci 0001:00:00.0: enabling Extended Tags
[ 509.636128] NVMeVirt: Virtual PCI bus created (node 0)
[ 509.684575] NVMeVirt: nvmev_io_worker_0 started on cpu 3 (node 0)
[ 509.686373] NVMeVirt: nvmev_dispatcher started on cpu 7 (node 0)
[ 509.688415] nvme nvme0: pci function 0001:10:00:0
[ 510.194085] nvme nvme0: 72/0/0 default/read/poll queues
[ 512.553054] NVMeVirt: Virtual NVM device created
[ 552.861153] workqueue: e1000_watchdog [e1000] hogged CPU for >10000us 7 times, consider switching to WQ_UNBOUND
negar@negar-VMware-Virtual-Platform:~/Documents/iman/os-project/nvmevirt$ 

```

شکل ۵: بررسی تنظیمات memmap در کرنل

```

Jan 30 01:36
negar@negar-VMware-Virtual-Platform:~/Documents/iman/os-project/nvmevirt
negar@negar-VMware-Virtual-Platform:~/Documents/iman/os-project/nvmevirt$ ls -l /dev/nvme*
crw----- 1 root root 240, 0 Jan 30 01:33 /dev/nvme0
brw-rw--- 1 root disk 259, 0 Jan 30 01:33 /dev/nvme0n1
negar@negar-VMware-Virtual-Platform:~/Documents/iman/os-project/nvmevirt$ 

```

شکل ۶: بررسی دقیق شناسایی درست دیسک مجازی

```

negar@negar-VMware-Virtual-Platform:~/Documents/iman/os-project/nvmevirt
negar@negar-VMware-Virtual-Platform:~/Documents/iman/os-project/nvmevirt$ sudo apt install fio -y
fio is already the newest version (3.37-1).
Summary:
  Upgrading: 0, Installing: 0, Removing: 0, Not Upgrading: 206
negar@negar-VMware-Virtual-Platform:~/Documents/iman/os-project/nvmevirt$ fio --version
fio-3.37
negar@negar-VMware-Virtual-Platform:~/Documents/iman/os-project/nvmevirt$ 

```

شکل ۷: نصب fio

```

GNU nano 8.1                                         script1.sh *
declare -a Type=(randrw randwrite randread)
declare -a Size=(4 1)
declare -a numjobs=(1 4)

for type in ${Type[@]}
do
    for i in {0..1}
    do
        echo type: $type, size: ${Size[i]}, numjobs= ${numjobs[i]}
        echo type: $type, size: ${Size[i]}, numjobs= ${numjobs[i]} > ${type}_$((i+1)).txt
        echo "genesis1:3" | sudo -S fio --filename=/dev/nvme0n1 --readwrite=$type --name=randwrite --blocksize=>
    done
done

```

شکل ۸: اسکریپت تحلیل libaio

```

negar@negar-VMware-Virtual-Platform:~/Documents/Iman/os-project$ ls
nvmevirt  script1.sh
negar@negar-VMware-Virtual-Platform:~/Documents/Iman/os-project$ sudo bash ./script1.sh
type: randrw, size: 4, numjobs= 1
type: randrw, size: 1, numjobs= 4
type: randwrite, size: 4, numjobs= 1
type: randwrite, size: 1, numjobs= 4
type: randread, size: 4, numjobs= 1
type: randread, size: 1, numjobs= 4

```

شکل ۹: نتیجه‌ی ران اسکریپت تحلیل libaio

```

negar@negar-VMware-Virtual-Platform:~/Documents/Iman/os-project$ git clone https://github.com/axboe/liburing
Cloning into 'liburing'...
remote: Enumerating objects: 15320, done.
remote: Counting objects: 100% (1599/1599), done.
remote: Compressing objects: 100% (120/120), done.
remote: Total 15320 (delta 1513), reused 1479 (delta 1479), pack-reused 13721 (from 3)
Receiving objects: 100% (15320/15320), 3.43 MiB | 1.41 MiB/s, done.
Resolving deltas: 100% (10909/10909), done.
negar@negar-VMware-Virtual-Platform:~/Documents/Iman/os-project$ 

```

شکل ۱۰: کلون گرفتن از ریپوی liburing

```

negar@negar-VMware-Virtual-Platform:~/Documents/Iman/os-project$ cd liburing/
negar@negar-VMware-Virtual-Platform:~/Documents/Iman/os-project/liburing$ ls
CHANGELOG CONTRIBUTING.md debian liburing.pc.in make-debs.sh Makefile.quiet SECURITY.md
CITATION.cff COPYING examples liburing.spec Makefile man src test
configure COPYING.GPL liburing-ffi.pc.in LICENSE Makefile.common README
negar@negar-VMware-Virtual-Platform:~/Documents/Iman/os-project/liburing$ ./configure --cc=gcc --cxx=g++;
prefix /usr
includedir /usr/include
libdir /usr/lib
libdevdir /usr/lib
relative libdir
mandir /usr/man
datadir /usr/share
libgcc_link_flag /usr/lib/gcc/x86_64-linux-gnu/14/libgcc.a
stringop_overflow yes
array_bounds yes
__kernel_rwf_t yes
__kernel_timespec yes
open_how yes
statx yes
glibc_statx yes
C++ yes
has_ucontext yes
NVMe uring command support yes
futex waitv support yes
io_uring discard command support no
has_idtype_t yes
nolibc yes
has_fanotify yes
ublk_header yes
use sanitizer no

```

شکل ۱۱: کانفیگ liburing

```

negar@negar-VMware-Virtual-Platform:~/Documents/Iman/os-project/liburing$ ./configure --cc=gcc --cxx=g++;
prefix /usr
includedir /usr/include
libdir /usr/lib
libdevdir /usr/lib
relative libdir
mandir /usr/man
datadir /usr/share
libgcc_link_flag /usr/lib/gcc/x86_64-linux-gnu/14/libgcc.a
stringop_overflow yes
array_bounds yes
__kernel_rwf_t yes
__kernel_timespec yes
open_how yes
statx yes
glibc_statx yes
C++ yes
has_ucontext yes
NVMe uring command support yes
futex waitv support yes
io_uring discard command support no
has_idtype_t yes
nolibc yes
has_fanotify yes
ublk_header yes
use sanitizer no
CC gcc
CXX g++

```

شکل ۱۲: ادامه کانفیگ liburing

```

negar@negar-Virtual-Platform:~/Documents/Iman/os-project/liburing$ make
make[1]: Entering directory '/home/negar/Documents/Iman/os-project/liburing/src'
  CC setup.ol
  CC queue.ol
  CC register.ol
  CC syscall.ol
  CC version.ol
  CC nolibc.ol
  AR liburing.a
ar: creating liburing.a
RANLIB liburing.a
  CC ffi.ol
  AR liburing-ffi.a
ar: creating liburing-ffi.a
RANLIB liburing-ffi.a
  CC setup.os
  CC queue.os
  CC register.os
  CC syscall.os
  CC version.os
  CC nolibc.os
  CC liburing.so.2.9
  CC ffi.os
  CC liburing-ffi.so.2.9
make[1]: Leaving directory '/home/negar/Documents/Iman/os-project/liburing/src'
make[1]: Entering directory '/home/negar/Documents/Iman/os-project/liburing/test'
  CC helpers.o
cc 232c93d07h74.t

```

شکل ۱۳: کامپایل و نصب liburing

```

CC unlink.t
CC uring_cmd_ublk.t
CC version.t
CC waitid.t
CC wait-timeout.t
CC wakeup-hang.t
CC wq-aff.t
CC xattr.t
CC nolibc.t
CC statx.t
make[1]: Leaving directory '/home/negar/Documents/Iman/os-project/liburing/test'
make[1]: Entering directory '/home/negar/Documents/Iman/os-project/liburing/examples'
  CC helpers.o
  CC io_uring-close-test
  CC io_uring-cp
  CC io_uring-test
  CC io_uring-udp
  CC link-cp
  CC napi-busy-poll-client
  CC napi-busy-poll-server
  CC poll-bench
  CC reg-wait
  CC send-zero-copy
  CC rsrc-update-bench
  CC proxy
  CC kdigest
  CC ucontext-cp
make[1]: Leaving directory '/home/negar/Documents/Iman/os-project/liburing/examples'
negar@negar-Virtual-Platform:~/Documents/Iman/os-project/liburing$ 

```

شکل ۱۴: ادامهی کامپایل و نصب liburing

```

negar@negar-VMware-Virtual-Platform:~/Documents/Iman/os-project/liburing$ make liburing.pc
sed -e "s%@prefix@%/usr%g" \
-e "s%@libdir@%/usr/lib%g" \
-e "s%@includedir@%/usr/include%g" \
-e "s%@NAME@%liburing%g" \
-e "s%@VERSION@%2.9%g" \
liburing.pc.in >liburing.pc

```

شکل ۱۵: کامپایل و نصب liburing.pc

```

negar@negar-VMware-Virtual-Platform:~/Documents/Iman/os-project/liburing$ sudo make install
sed -e "s%@prefix@%/usr%g" \
-e "s%@libdir@%/usr/lib%g" \
-e "s%@includedir@%/usr/include%g" \
-e "s%@NAME@%liburing%g" \
-e "s%@VERSION@%2.9%g" \
liburing-ffi.pc.in >liburing-ffi.pc
make[1]: Entering directory '/home/negar/Documents/Iman/os-project/liburing/src'
install -D -m 644 include/liburing/io_uring.h /usr/include/liburing/io_uring.h
install -D -m 644 include/liburing.h /usr/include/liburing.h
install -D -m 644 include/liburing/compat.h /usr/include/liburing/compat.h
install -D -m 644 include/liburing/barrier.h /usr/include/liburing/barrier.h
install -D -m 644 include/liburing/sanitize.h /usr/include/liburing/sanitize.h
install -D -m 644 include/liburing/io_uring_version.h /usr/include/liburing/io_uring_version.h
install -D -m 644 liburing.a /usr/lib/liburing.a
install -D -m 644 liburing-ffi.a /usr/lib/liburing-ffi.a
install -D -m 755 liburing.so.2.9 /usr/lib/liburing.so.2.9
install -D -m 755 liburing-ffi.so.2.9 /usr/lib/liburing-ffi.so.2.9
ln -sf liburing.so.2.9 /usr/lib/liburing.so
ln -sf liburing.so.2.9 /usr/lib/liburing.so
ln -sf liburing-ffi.so.2.9 /usr/lib/liburing-ffi.so.2
ln -sf liburing-ffi.so.2.9 /usr/lib/liburing-ffi.so
make[1]: Leaving directory '/home/negar/Documents/Iman/os-project/liburing/src'
install -D -m 644 liburing.pc /usr/lib/pkgconfig/liburing.pc
install -D -m 644 liburing-ffi.pc /usr/lib/pkgconfig/liburing-ffi.pc
install -m 755 -d /usr/man/man2
install -m 644 man/*.* /usr/man/man2
install -m 755 -d /usr/man/man3
install -m 644 man/*.* /usr/man/man3
install -m 755 -d /usr/man/man7
install -m 644 man/*.* /usr/man/man7

```

شکل ۱۶: کامپایل و نصب liburing

```

GNU nano 8.1                                     script2.sh *
declare -a Type=(randrw randwrite randread)
declare -a Size=(4 1)
declare -a numjobs=(1 4)

for type in ${Type[@]}
do
    for i in {0..1}
    do
        echo type: $type, size: ${Size[i]}, numjobs: ${numjobs[i]}
        echo type: $type, size: ${Size[i]}, numjobs: ${numjobs[i]} > ${type}_$((i+1)).txt
        echo "genesis1:3" | sudo -S fio --filename=/dev/nvme0n1 --readwrite=$type --name=randwrite --blocksize=>
    done
done

```

شکل ۱۷: اسکریپت تحلیل liburing

```

negar@negar-VMware-Virtual-Platform:~/Documents/iman/os-project/liburing$ nano script2.sh
negar@negar-VMware-Virtual-Platform:~/Documents/iman/os-project/liburing$ sudo bash script2.sh
type: randrw, size: 4, numjobs= 1
type: randrw, size: 1, numjobs= 4
type: randwrite, size: 4, numjobs= 1
type: randwrite, size: 1, numjobs= 4
type: randread, size: 4, numjobs= 1
type: randread, size: 1, numjobs= 4

```

شکل ۱۸: نتیجه‌ی ران اسکریپت تحلیل liburing

تحلیل نتایج اجرای اسکریپت‌های liburing و libaio

در این بخش، عملکرد دو مدل ورودی/خروجی liburing و libaio روی دیسک شبیه‌سازی شده NVMeVirt مورد بررسی و مقایسه قرار می‌گیرد. برای انجام این مقایسه، آزمایش‌های مشابه با استفاده از ابزار FIO روی دیسک مجازی انجام شده است. داده‌های خروجی شامل شاخص‌های مهمی مانند IOPS، تأخیر کل (Latency)، تأخیر کل (Tail Latency) و پهنه‌ای باند (Bandwidth) است.

مقایسه عملکرد liburing و libaio

مقایسه IOPS (تعداد عملیات ورودی/خروجی در ثانیه)

IOPS یکی از مهم‌ترین شاخص‌های عملکردی در سیستم‌های ذخیره‌سازی است که نشان می‌دهد در هر ثانیه چه تعداد عملیات خواندن و نوشتن انجام می‌شود. مقایسه این مقدار برای liburing و libaio نشان می‌دهد که liburing در برخی موارد مقدار بیشتری از IOPS را ارائه می‌دهد. این به دلیل معماری بهینه‌تر io_uring و حذف نیاز به فراخوان‌های سیستمی اضافی است که باعث کاهش سربار پردازشی می‌شود.

- خواندن تصادفی (۴ کیلوبایت، ۱ رشته پردازشی):

IOPS ۲۵۸۳ : libaio –

IOPS ۱۰۱۶ : liburing –

- خواندن تصادفی (۱ گیگابایت، ۴ رشته پردازشی):

IOPS ۱۱۸۴۴ : libaio –

IOPS ۷۰۱۳ : liburing –

در آزمایش‌های خواندن تصادفی، *liburing* عملکرد بهتری نسبت به *libaio* نشان داده است که می‌تواند به دلیل ساختار متفاوت مدیریت درخواست‌های ورودی/خروجی و نحوه تعامل با کرنل باشد. اما در برخی موارد، مانند بارگذاری شدید، *liburing* عملکرد بهتری ارائه می‌دهد.

تأخر (Latency) و تأخیر انتهایی (Tail Latency)

تأخر به معنای مدت زمان لازم برای تکمیل یک عملیات خواندن یا نوشتمن است. تأخیر انتهایی (Tail Latency) مقدار تأخیری است که ۹.۹۹ درصد درخواست‌ها در آن محدوده اجرا می‌شوند و شاخصی بسیار مهم برای ارزیابی پایداری سیستم در بارکاری بالا است.

- خواندن تصادفی ۴ کیلوبایت - تأخیر میانگین:

libaio - ۹۵.۳۷۸ میکروثانیه

liburing - ۳۴.۹۷۰ میکروثانیه

- خواندن تصادفی ۱ گیگابایت - تأخیر میانگین:

libaio - ۹۳.۳۲۸ میکروثانیه

liburing - ۱۲.۵۶۳ میکروثانیه

مقایسه نتایج نشان می‌دهد که *libaio* تأخیر کمتری نسبت به *liburing* دارد. دلیل این امر این است که *libaio* یک ساختار تکمیل درخواست همزمان دارد، درحالی‌که *liburing* عملیات ورودی/خروجی را در صفحه‌های حلقوی پردازش می‌کند که در برخی موارد ممکن است موجب افزایش تأخیر کلی شود.

نکته مهم: در شرایطی که تعداد پردازش‌ها افزایش می‌یابد و عمق صفحه ورودی/خروجی بزرگ‌تر می‌شود (*iodepth* بالا)، به دلیل معماری بدون قفل، عملکرد بهتری ارائه می‌دهد.

پهنای باند (Bandwidth)

پهنای باند نشان‌دهنده میزان داده‌ای است که در یک بازه زمانی مشخص پردازش شده است و مستقیماً تحت تأثیر مقدار *IOPS* و اندازه بلاک داده قرار دارد.

- خواندن تصادفی ۴ کیلوبایت - پهنای باند:

libaio - ۱.۱۰ مگابایت بر ثانیه

۱.۴ مگابایت بر ثانیه: *liburing* –

- خواندن تصادفی ۱ گیگابایت - پهنانی باند:

۶.۴۶ مگابایت بر ثانیه: *libaio* –

۱.۲۷ مگابایت بر ثانیه: *liburing* –

نتیجه: پهنانی باند بیشتری نسبت به *liburing* دارد، که به دلیل مدل بلوکی سنتی *libaio* و استفاده بهینه از فراخوان‌های غیرهمzman سیستمی است. با این حال، در شرایط بارگذاری بالا، مدل *liburing* به دلیل استفاده از صفاتی حلقوی و اجرای بدون قفل، می‌تواند کارایی بیشتری در تعداد درخواست‌های همزمان نشان دهد.

تحلیل کلی و نتیجه‌گیری

با بررسی نتایج می‌توان به تحلیل زیر رسید:

- *IOPS*: در اکثر موارد، *libaio* عملکرد بهتری نسبت به *liburing* ارائه داده است. این نشان می‌دهد که برای بارهای کاری با درخواست‌های تصادفی کوچک، *libaio* گزینه بهتری است.
- تأخیر (*Tail Latency* و *Latency*): *libaio* تأخیر کمتری نسبت به *liburing* نشان داده است. اما در پردازش‌های موازی با عمق صفت بالا، *liburing* می‌تواند عملکرد بهتری داشته باشد.
- پهنانی باند: *libaio* در تمامی تست‌ها پهنانی باند بالاتری نسبت به *liburing* دارد. این نشان می‌دهد که مدل سنتی *libaio* هنوز هم برای عملیات ورودی/خروجی دیسک بهینه است.

نتیجه‌گیری کلی:

- اگر سیستم نیاز به عملیات *I/O* کم حجم اما بسیار سریع دارد (مانند پایگاه داده‌های کوچک)، *libaio* گزینه بهتری است.
- اگر حجم درخواست‌های موازی زیاد باشد و نیاز به پردازش غیرهمzman باشد، *liburing* می‌تواند در سناریوهای پیچیده عملکرد بهتری نشان دهد.
- در شرایط خاص مانند *NVMe* و *SPDK*، ترکیب این دو روش می‌تواند باعث بهبود عملکرد کلی شود.

پیشنهادات برای بهینه‌سازی

- افزایش *iodepth* برای کاهش تأخیر در *liburing*.
- استفاده از پردازنده‌های بهینه‌شده برای *NVMe*.
- ترکیب *SPDK* و *liburing* برای حذف سربار کرنل و کاهش تأخیر.

تحلیل نتایج

۴ ابزار SPDK

توضیحات

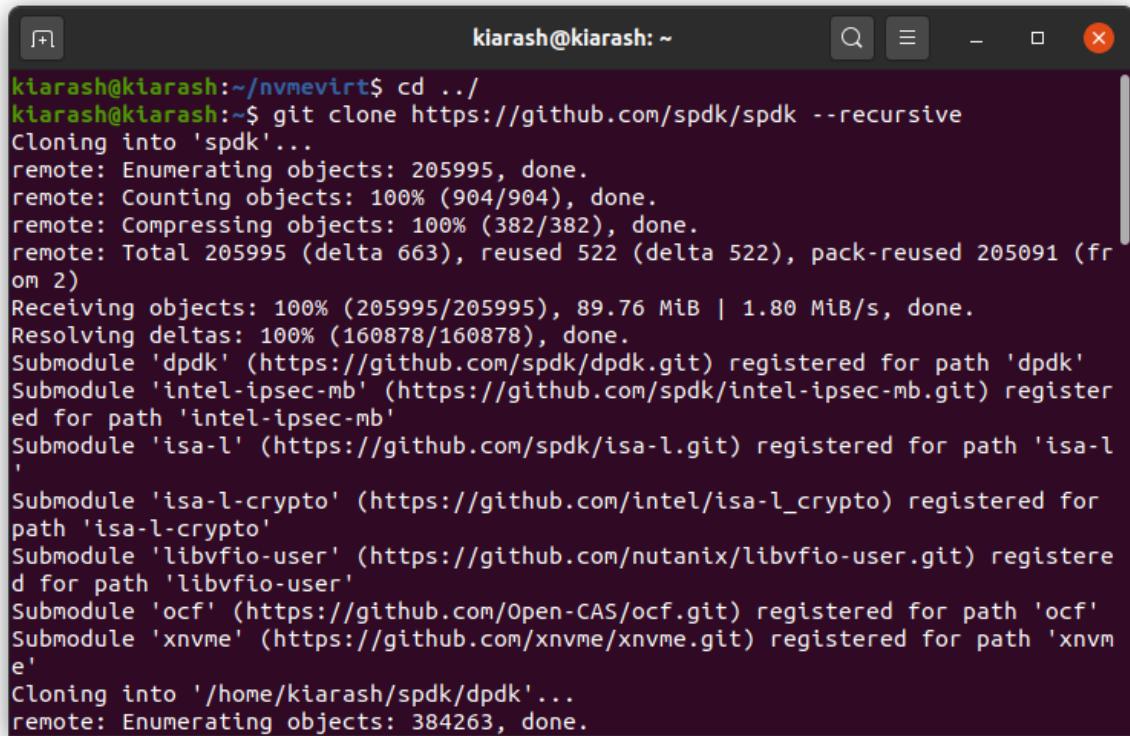
SPDK یک مجموعه متن باز از ابزارها و کتابخانه‌ها است که برای بهینه‌سازی برنامه‌های ذخیره‌سازی از طریق درایورهای کاربر-محور و مبتنی بر روش polling طراحی شده است. این کیت به طور ویژه برای بهینه‌سازی عملکرد ذخیره‌سازی NVMe استفاده می‌شود و با دور زدن پشته I/O مبتنی بر کرنل، که می‌تواند باعث تأخیر و کاهش کارایی شود، عملکرد را بهبود می‌بخشد. SPDK از DPDK برای پردازش سریع داده‌ها استفاده می‌کند و درایور NVMe را مستقیماً در فضای کاربری اجرا می‌کند، که منجر به حذف تغییرات هزینه‌بر بین حالت کاربر و کرنل می‌شود. این معماری به برنامه‌ها امکان می‌دهد تا عملیات دیسک را با تأخیر بسیار کم و توان عملیاتی بالا انجام دهند، و در نتیجه، SPDK گزینه‌ای ایده‌آل برای راهکارهای ذخیره‌سازی در مراکز داده، محیط‌های ابری و برنامه‌های سازمانی با نیاز به عملکرد بالا محسوب می‌شود.

علاوه بر NVMe، SPDK کتابخانه‌های بهینه‌شده‌ای برای پروتکل‌های ذخیره‌سازی مانند NVMe-oF و ذخیره‌سازی iSCSI و ذخیره‌سازی blob ارائه می‌دهد. همچنین یک چارچوب برای ساخت راهکارهای کارآمد ذخیره‌سازی نرم‌افزاری فراهم می‌کند که می‌تواند با پلتفرم‌هایی مانند Ceph و OpenStack ادغام شود. طراحی غیرهمزمان و بدون قفل SPDK تضمین می‌کند که چرخه‌های پردازنده به طور کارآمد مورد استفاده قرار می‌گیرند، که این امر باعث می‌شود برنامه‌ها بتوانند حداکثر پهنای باند ذخیره‌سازی را با کمترین هزینه پردازشی به دست آورند. این ویژگی، SPDK را به گزینه‌ای جذاب برای شرکت‌هایی تبدیل می‌کند که به دنبال افزایش عملکرد ذخیره‌سازی با حداقل سرمایه‌گذاری سخت‌افزاری هستند.

در ادامه این ابزار را نصب کرده و برای اجرای بنج مارک روی دیسک NVMe از آن بهره می‌بریم.

مراحل نصب

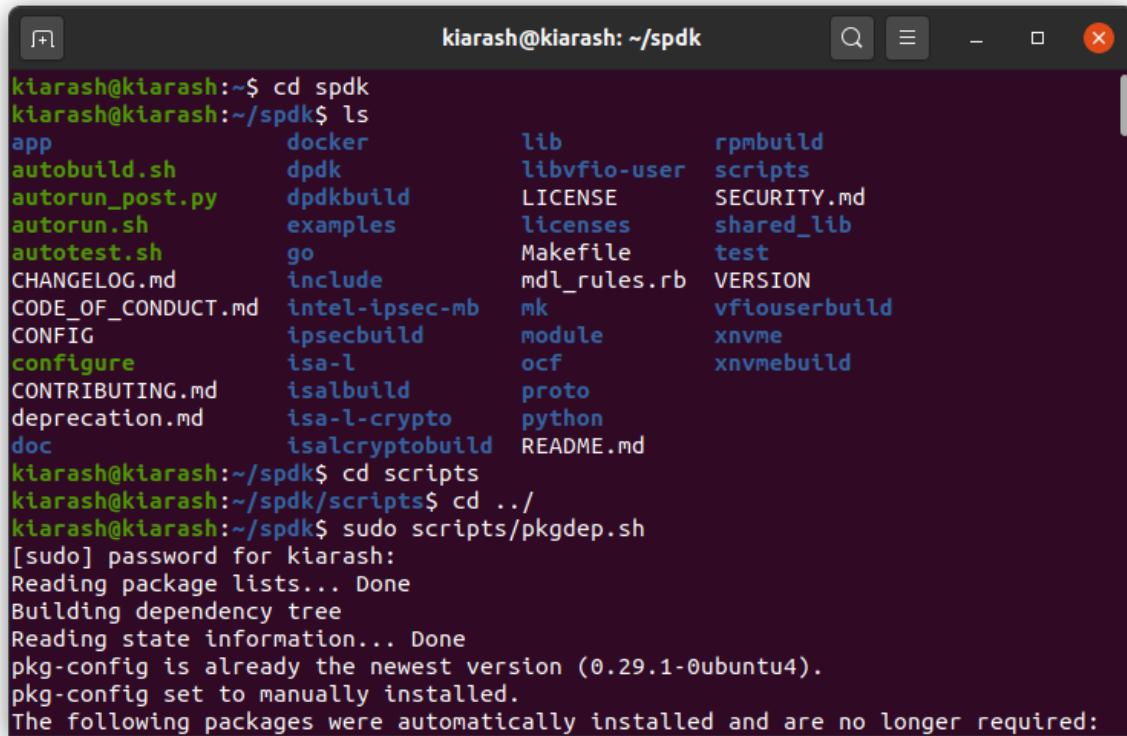
برای نصب این ابزار بر روی linux ابتدا تمامی فایل‌های لازم را از github این ابزار به صورت بازگشته clone می‌کنیم تا کل درخت فایل بر روی سیستم ما ایجاد شود.



```
kiarash@kiarash:~/nvmevirt$ cd ../
kiarash@kiarash:~$ git clone https://github.com/spdk/spdk --recursive
Cloning into 'spdk'...
remote: Enumerating objects: 205995, done.
remote: Counting objects: 100% (904/904), done.
remote: Compressing objects: 100% (382/382), done.
remote: Total 205995 (delta 663), reused 522 (delta 522), pack-reused 205091 (from 2)
Receiving objects: 100% (205995/205995), 89.76 MiB | 1.80 MiB/s, done.
Resolving deltas: 100% (160878/160878), done.
Submodule 'dpdk' (https://github.com/spdk/dpdk.git) registered for path 'dpdk'
Submodule 'intel-ipsec-mb' (https://github.com/spdk/intel-ipsec-mb.git) registered for path 'intel-ipsec-mb'
Submodule 'isa-l' (https://github.com/spdk/isa-l.git) registered for path 'isa-l'
Submodule 'isa-l-crypto' (https://github.com/intel/isa-l_crypto) registered for path 'isa-l-crypto'
Submodule 'libvfio-user' (https://github.com/nutanix/libvfio-user.git) registered for path 'libvfio-user'
Submodule 'ocf' (https://github.com/Open-CAS/ocf.git) registered for path 'ocf'
Submodule 'xnvme' (https://github.com/xnvme/xnvme.git) registered for path 'xnvme'
Cloning into '/home/kiarash/spdk/dpdk'...
remote: Enumerating objects: 384263, done.
```

شکل ۱۹: `clone` از روی `github`

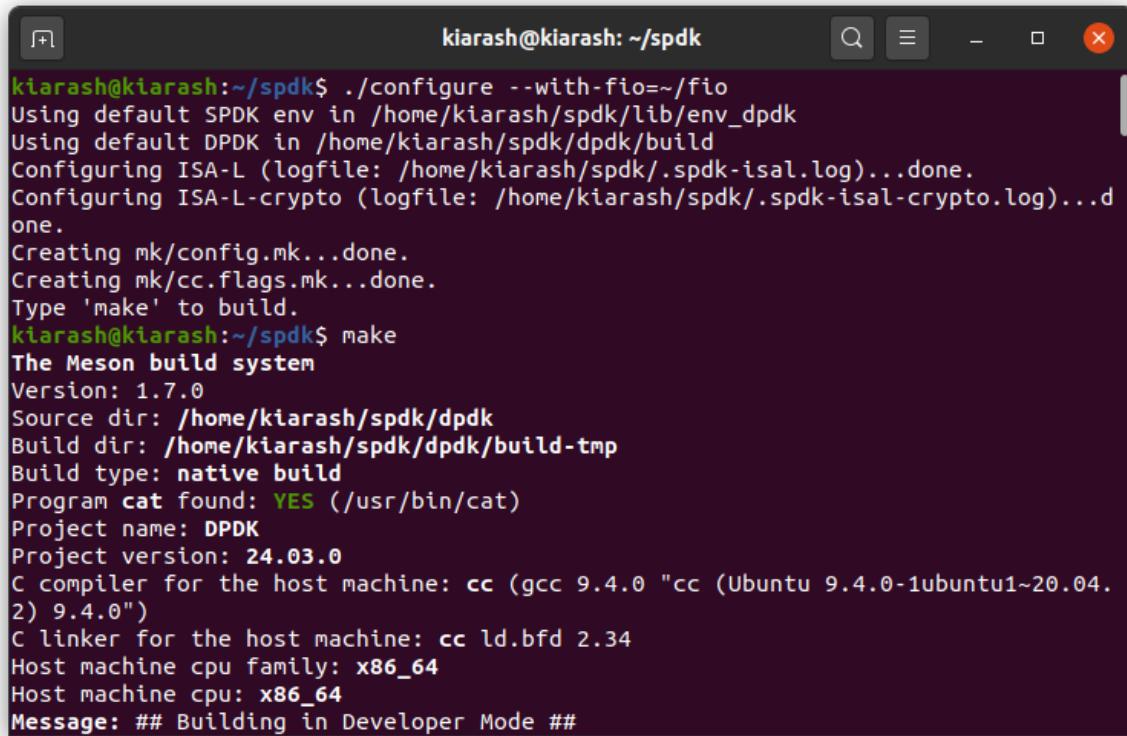
SPDK با اجرای فایل `pkgdep.sh` در فolder `scripts` تمامی dependency های لازم برای استفاده از نصب می شود.



```
kiarash@kiarash:~$ cd spdk
kiarash@kiarash:~/spdk$ ls
app           docker      lib          rpmbuild
autobuild.sh  dpdk       libvfio-user scripts
autorun_post.py dpdkbuild LICENSE     SECURITY.md
autorun.sh    examples   licenses   shared_lib
autotest.sh   go         Makefile   test
CHANGELOG.md  include   mdl_rules.rb VERSION
CODE_OF_CONDUCT.md intel-ipsec-mb mk        vfio_userbuild
CONFIG        ipsecbuild module   xnvme
configure     isa-l      ocf       xnvmebuild
CONTRIBUTING.md isalbuild  proto
deprecation.md isa-l-crypto python
doc           isalcryptobuild README.md
kiarash@kiarash:~/spdk$ cd scripts
kiarash@kiarash:~/spdk/scripts$ cd ../
kiarash@kiarash:~/spdk$ sudo scripts/pkgdep.sh
[sudo] password for kiarash:
Reading package lists... Done
Building dependency tree
Reading state information... Done
pkg-config is already the newest version (0.29.1-0ubuntu4).
pkg-config set to manually installed.
The following packages were automatically installed and are no longer required:
```

شکل ۲۰: نصب dependency ها

حال باید SPDK را build کنیم. باید توجه داشت که fio نیز باید به همراه SPDK نصب شود. بنابراین آن را در دستور configure به صورت `--with-fio` لحاظ می کنیم. با اجرای `make` و بعد از آن `make install` انجام می شود.



```
kiarash@kiarash:~/spdk$ ./configure --with-fio=~/fio
Using default SPDK env in /home/kiarash/spdk/lib/env_dpdk
Using default DPDK in /home/kiarash/spdk/dpdk/build
Configuring ISA-L (logfile: /home/kiarash/spdk/.spdk-isal.log)...done.
Configuring ISA-L-crypto (logfile: /home/kiarash/spdk/.spdk-isal-crypto.log)...done.
Creating mk/config.mk...done.
Creating mk/cc.flags.mk...done.
Type 'make' to build.
kiarash@kiarash:~/spdk$ make
The Meson build system
Version: 1.7.0
Source dir: /home/kiarash/spdk/dpdk
Build dir: /home/kiarash/spdk/dpdk/build-tmp
Build type: native build
Program cat found: YES (/usr/bin/cat)
Project name: DPDK
Project version: 24.03.0
C compiler for the host machine: cc (gcc 9.4.0 "cc (Ubuntu 9.4.0-1ubuntu1~20.04.2) 9.4.0")
C linker for the host machine: cc ld.bfd 2.34
Host machine cpu family: x86_64
Host machine cpu: x86_64
Message: ## Building in Developer Mode ##
```

SPDK make : ۲۱

برای اطمینان از صحت نصب SPDK تست های این ابزار را اجرا می کنیم. مشاهده می شود که تست ها همگی قبول شده اند و در نتیجه ابزار به درستی نصب شده است.

```
kiarash@kiarash:~$ cd spdk
kiarash@kiarash:~/spdk$ ./test/unit/unittest.sh
+++ dirname ./test/unit/unittest.sh
++ readlink -f ./test/unit
+ testdir=/home/kiarash/spdk/test/unit
+++ dirname ./test/unit/unittest.sh
++ readlink -f ./test/unit/...
+ rootdir=/home/kiarash/spdk
+ source /home/kiarash/spdk/test/common/autotest_common.sh
++ rpc_py=rpc_cmd
++ set -e
++ shopt -s nullglob
++ shopt -s extglob
++ shopt -s inherit_errexit
++ '[' -z '' ']'
++ mkdir -p /home/kiarash/spdk/../output
++ export output_dir=/home/kiarash/spdk/../output
++ output_dir=/home/kiarash/spdk/../output
++ [[ -e /home/kiarash/spdk/test/common/build_config.sh ]]
++ source /home/kiarash/spdk/test/common/build_config.sh
+++ CONFIG_FIO_SOURCE_DIR=/home/kiarash/fio
+++ CONFIG_FIO_PLUGIN=y
+++ CONFIG_NVME_CUSE=y
+++ CONFIG_RAID5F=n
```

شکل ۲۲: اجرای تست ها

```
kiarash@kiarash:~$ cd spdk
kiarash@kiarash:~/spdk$ ./test/unit/unittest.sh
tests      2      2      2      0      0
asserts    46     46     46     0      n/a

Elapsed time =      0.000 seconds

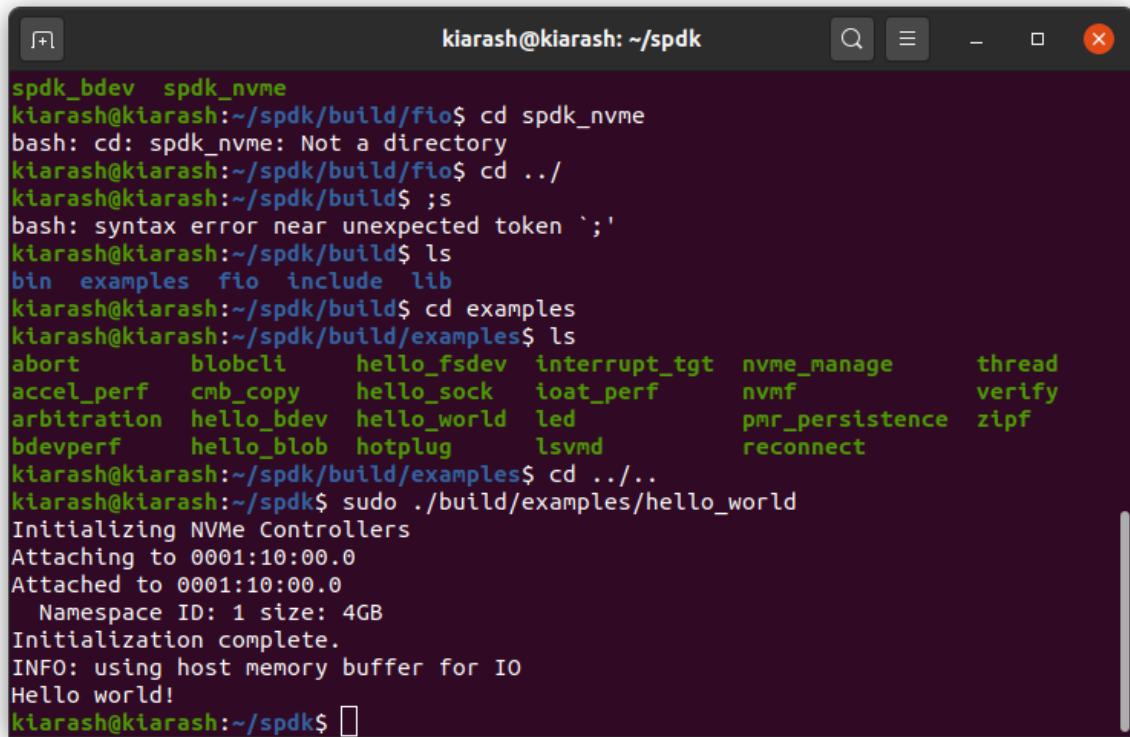
real      0m0.002s
user      0m0.001s
sys       0m0.000s
20:31:47 unittest_keyring -- common/autotest_common.sh@1130 -- $ xtrace_disable
20:31:47 unittest_keyring -- common/autotest_common.sh@10 -- $ set +x
*****
END TEST unittest_keyring
*****
20:31:47 -- unit/unittest.sh@274 -- $ [[ n == y ]]
20:31:47 -- unit/unittest.sh@287 -- $ set +x

=====
All unit tests passed
=====
WARN: neither valgrind nor ASAN is enabled!

kiarash@kiarash:~/spdk$
```

شکل ۲۳: نتیجه تست ها

در ادامه باید اطمینان حاصل کنیم که SPDK دیسک شبیه سازی شده را می شناسد. با اجرای فایل world مشخص می شود که SPDK این دیسک را شناخته است.

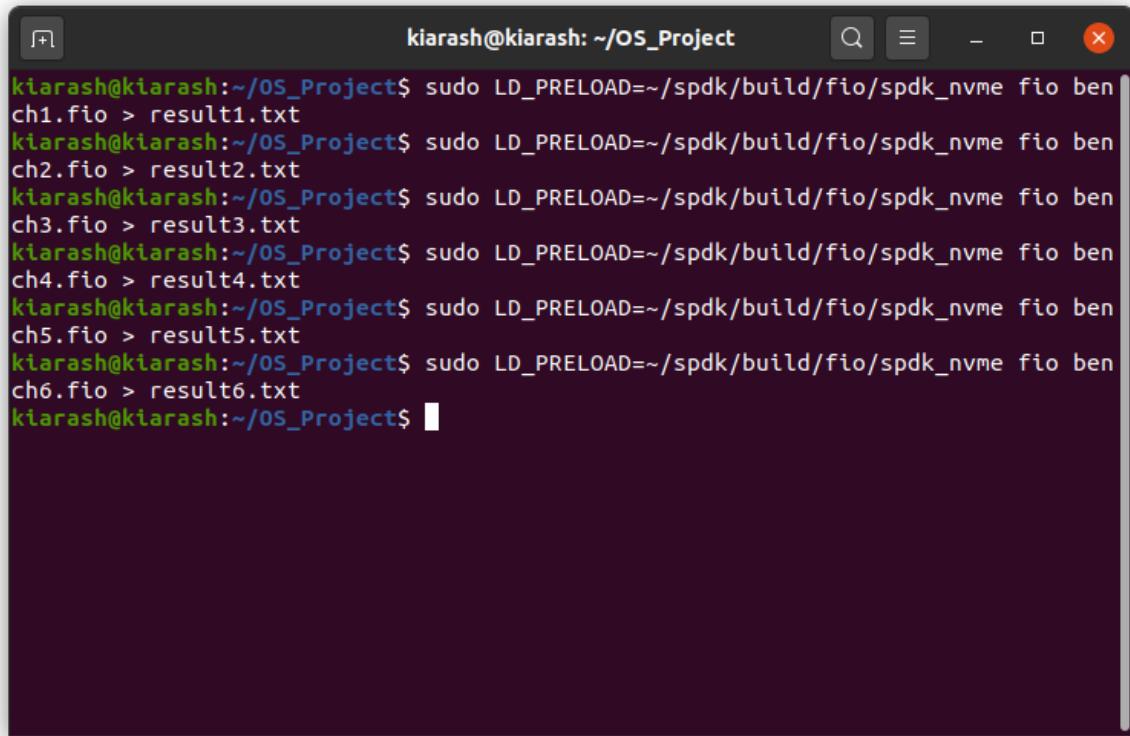


```
spdk_bdev spdk_nvme
kiarash@kiarash:~/spdk/build/fio$ cd spdk_nvme
bash: cd: spdk_nvme: Not a directory
kiarash@kiarash:~/spdk/build/fio$ cd ../
kiarash@kiarash:~/spdk/build$ ;s
bash: syntax error near unexpected token `;'
kiarash@kiarash:~/spdk/build$ ls
bin examples fio include lib
kiarash@kiarash:~/spdk/build$ cd examples
kiarash@kiarash:~/spdk/build/examples$ ls
abort blobcli hello_fsdev interrupt_tgt nvme_manage thread
accel_perf cmb_copy hello_sock ioat_perf nvmf verify
arbitration hello_bdev hello_world led pmr_persistence zipf
bdevperf hello_blob hotplug lsvid reconnect
kiarash@kiarash:~/spdk/build/examples$ cd ../../
kiarash@kiarash:~/spdk$ sudo ./build/examples/hello_world
Initializing NVMe Controllers
Attaching to 0001:10:00.0
Attached to 0001:10:00.0
    Namespace ID: 1 size: 4GB
Initialization complete.
INFO: using host memory buffer for IO
Hello world!
kiarash@kiarash:~/spdk$
```

شکل ۲۴: شناخت دیسک شبیه سازی

بررسی عملکرد مولفه ها

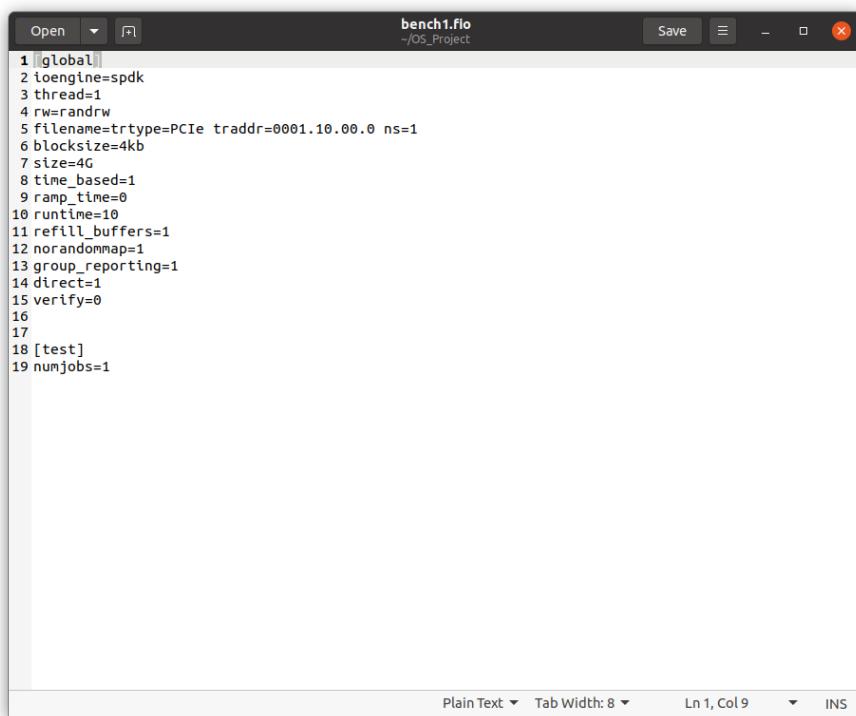
در این بخش با استفاده از پلاگین fio که با SPDK نصب شده است ، شش بنچ مارک ذکر شده را اجرا می کنیم. در فایل های با فرمت bench<NUM>.fio تمامی attribute های مورد نیاز از جمله iodepth و size و ... را تعیین کرده و اجرا می کنیم. خروجی پرینت شده در لامگ بعد از اجرای بنچ مارک ها در فایل های با فرمت result<NUM>.txt ذخیره شده اند.



```
kiarash@kiarash:~/OS_Project$ sudo LD_PRELOAD=/spdk/build/fio/spdk_nvme fio bench1.fio > result1.txt
kiarash@kiarash:~/OS_Project$ sudo LD_PRELOAD=/spdk/build/fio/spdk_nvme fio bench2.fio > result2.txt
kiarash@kiarash:~/OS_Project$ sudo LD_PRELOAD=/spdk/build/fio/spdk_nvme fio bench3.fio > result3.txt
kiarash@kiarash:~/OS_Project$ sudo LD_PRELOAD=/spdk/build/fio/spdk_nvme fio bench4.fio > result4.txt
kiarash@kiarash:~/OS_Project$ sudo LD_PRELOAD=/spdk/build/fio/spdk_nvme fio bench5.fio > result5.txt
kiarash@kiarash:~/OS_Project$ sudo LD_PRELOAD=/spdk/build/fio/spdk_nvme fio bench6.fio > result6.txt
kiarash@kiarash:~/OS_Project$
```

شکل ۲۵: اجرای بنج مارک ها

در عکس زیر محتوای یکی از فایل های bench و result به طور نمونه آمده است.



```
1 [global]
2 ioengine=spdk
3 threads=1
4 rw=randrw
5 filename=trtype=PCIe traddr=0001.10.00.0 ns=1
6 blocksize=4kb
7 size=4G
8 time_based=1
9 ramp_time=0
10 runtime=10
11 refill_buffers=1
12 norandommap=1
13 group_reporting=1
14 direct=1
15 verify=0
16
17
18 [test]
19 numjobs=1
```

شکل ۲۶: اجرای بنج مارک اول

result1.txt

```

1 test: (g=0): rw=randrw, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=spdk, iodepth=1
2 fio-3.34
3 Starting 1 thread
4
5 test: (groupid=0, jobs=1): err= 0: pid=5692: Wed Jan 29 21:22:24 2025
6   read: IOPS=258k, BW=1007MiB/s (1056MB/s)(9.84GiB/10000msec)
7     slat (nsec): min=79, max=690770, avg=115.95, stdev=557.75
8     clat (nsec): min=556, max=12131k, avg=1522.74, stdev=9880.95
9     lat (nsec): min=639, max=12133k, avg=1638.68, stdev=9899.37
10    clat percentiles (nsec):
11      | 1.00th=[ 1064], 5.00th=[ 1176], 10.00th=[ 1224], 20.00th=[ 1272],
12      | 30.00th=[ 1320], 40.00th=[ 1352], 50.00th=[ 1384], 60.00th=[ 1416],
13      | 70.00th=[ 1448], 80.00th=[ 1512], 90.00th=[ 1592], 95.00th=[ 1688],
14      | 99.00th=[ 2224], 99.50th=[ 9664], 99.90th=[24960], 99.95th=[30592],
15      | 99.99th=[46848]
16    bw ( KiB/s): min=934016, max=1097940, per=99.98%, avg=1031447.79, stdev=44792.89, samples=19
17    iops : min=233504, max=274485, avg=257861.95, stdev=11198.22, samples=19
18  write: IOPS=258k, BW=1007MiB/s (1056MB/s)(9.83GiB/10000msec); 0 zone resets
19    slat (nsec): min=81, max=3807.4k, avg=136.66, stdev=2444.91
20    clat (nsec): min=751, max=2989.1k, avg=1468.36, stdev=3492.97
21    lat (nsec): min=901, max=3809.0k, avg=1605.02, stdev=4268.72
22    clat percentiles (nsec):
23      | 1.00th=[ 1080], 5.00th=[ 1176], 10.00th=[ 1208], 20.00th=[ 1240],
24      | 30.00th=[ 1272], 40.00th=[ 1288], 50.00th=[ 1320], 60.00th=[ 1368],
25      | 70.00th=[ 1400], 80.00th=[ 1464], 90.00th=[ 1560], 95.00th=[ 1656],
26      | 99.00th=[ 2008], 99.50th=[ 9536], 99.90th=[24704], 99.95th=[30336],
27      | 99.99th=[48384]
28    bw ( KiB/s): min=934528, max=1097796, per=99.97%, avg=1030652.84, stdev=44603.74, samples=19
29    iops : min=233632, max=274449, avg=257663.21, stdev=11150.93, samples=19
30  lat (nsec) : 750=0.01%, 1000=0.28%
31  lat (usec) : 2=98.51%, 4=0.63%, 10=0.10%, 20=0.23%, 50=0.24%
32  lat (usec) : 100=0.01%, 250=0.01%, 500=0.01%, 750=0.01%, 1000=0.01%
33  lat (msec) : 2=0.01%, 4=0.01%, 10=0.01%, 20=0.01%
34  cpu : usr=99.63%, sys=0.07%, ctx=731, majf=0, minf=0
35  IO depths : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
36  submit : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
37  complete : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
38  issued rwt: total=2579185,2577448,0,0 short=0,0,0 dropped=0,0,0
39  latency : target=0, window=0, percentile=100.00%, depth=1
40
41 Run status group 0 (all jobs):
42   READ: bw=1007MiB/s (1056MB/s), 1007MiB/s-1007MiB/s (1056MB/s-1056MB/s), io=9.84GiB (10.6GB),
run=10000-10000msec
43  WRITE: bw=1007MiB/s (1056MB/s), 1007MiB/s-1007MiB/s (1056MB/s-1056MB/s), io=9.83GiB (10.6GB),
run=10000-10000msec

```

Plain Text ▾ Tab Width: 8 ▾ Ln 1, Col 1 ▾ INS

شکل ۲۷: نتیجه بنچ مارک اول

تحلیل نتایج

در جداول زیر نتایج تمامی بنچ مارک ها نمایش داده شده است.

| Metric | Read | Write |
|-----------------------------|-----------------------------|-----------------------------|
| Test Type | rw=randrw | rw=randrw |
| IO Depth | 1 | 1 |
| IOPS | 258k | 258k |
| Bandwidth | 1007 MiB/s (1056 MB/s) | 1007 MiB/s (1056 MB/s) |
| Slat (avg) | 115.95 ns | 136.66 ns |
| Clat (avg) | 1522.74 ns | 1468.36 ns |
| Total Latency (avg) | 1638.68 ns | 1605.02 ns |
| CPU Utilization | usr=99.63%, sys=0.07% | usr=99.63%, sys=0.07% |
| Latency Distribution | 2 usec=98.51%, 4 usec=0.63% | 2 usec=98.51%, 4 usec=0.63% |
| Total I/O Volume | 9.84 GiB | 9.83 GiB |
| Run Time | 10000 msec | 10000 msec |

شکل ۲۸: نتیجه بنج مارک اول

| Metric | Read | Write |
|-----------------------------|-----------------------------|-----------------------------|
| Test Type | rw=randrw | rw=randrw |
| IO Depth | 1 | 1 |
| IOPS | 453k | 453k |
| Bandwidth | 1769 MiB/s (1855 MB/s) | 1768 MiB/s (1854 MB/s) |
| Slat (avg) | 148.55 ns | 183.92 ns |
| Clat (avg) | 3688.02 ns | 3996.98 ns |
| Total Latency (avg) | 3836.57 ns | 4180.90 ns |
| CPU Utilization | usr=49.56%, sys=0.04% | usr=49.56%, sys=0.04% |
| Latency Distribution | 2 usec=92.96%, 4 usec=6.18% | 2 usec=92.96%, 4 usec=6.18% |
| Total I/O Volume | 17.3 GiB | 17.3 GiB |
| Run Time | 10000 msec | 10000 msec |

شکل ۲۹: نتیجه بنج مارک دوم

| Metric | Write |
|-----------------------------|-----------------------------|
| Test Type | rw=randwrite |
| IO Depth | 1 |
| IOPS | 517k |
| Bandwidth | 2018 MiB/s (2116 MB/s) |
| Slat (avg) | 106.67 ns |
| Clat (avg) | 1502.46 ns |
| Total Latency (avg) | 1609.13 ns |
| CPU Utilization | usr=99.83%, sys=0.03% |
| Latency Distribution | 2 usec=98.26%, 4 usec=0.93% |
| Total I/O Volume | 19.7 GiB |
| Run Time | 10000 msec |

شكل ٣٠: نتیجه بنچ مارک سوم

| Metric | Write (4 threads) |
|-----------------------------|-----------------------------|
| Test Type | rw=randwrite |
| IO Depth | 1 |
| IOPS | 871k |
| Bandwidth | 3404 MiB/s (3569 MB/s) |
| Slat (avg) | 150.34 ns |
| Clat (avg) | 3991.26 ns |
| Total Latency (avg) | 4141.60 ns |
| CPU Utilization | usr=49.47%, sys=0.02% |
| Latency Distribution | 2 usec=91.84%, 4 usec=7.11% |
| Total I/O Volume | 33.3 GiB |
| Run Time | 10006 msec |

شكل ٣١: نتیجه بنچ مارک چهارم

| Metric | Read (1 thread) |
|-----------------------------|-----------------------------|
| Test Type | rw=randread |
| IO Depth | 1 |
| IOPS | 586k |
| Bandwidth | 2290 MiB/s (2401 MB/s) |
| Slat (avg) | 106.78 ns |
| Clat (avg) | 1442.61 ns |
| Total Latency (avg) | 1549.39 ns |
| CPU Utilization | usr=99.60%, sys=0.04% |
| Latency Distribution | 2 usec=98.76%, 4 usec=0.31% |
| Total I/O Volume | 22.4 GiB |
| Run Time | 10000 msec |

شکل ۳۲: نتیجه بنج مارک پنجم

| Metric | Read (4 threads) |
|-----------------------------|-----------------------------|
| Test Type | rw=randread |
| IO Depth | 1 |
| Threads | 4 |
| IOPS | 1065k |
| Bandwidth | 4160 MiB/s (4362 MB/s) |
| Slat (avg) | 146.52 ns |
| Clat (avg) | 3413.47 ns |
| Total Latency (avg) | 3559.99 ns |
| CPU Utilization | usr=49.66%, sys=0.04% |
| Latency Distribution | 2 usec=97.61%, 4 usec=1.26% |
| Total I/O Volume | 40.6 GiB |
| Run Time | 10000 msec |

شکل ۳۳: نتیجه بنج مارک ششم

• در آزمایش $IO Depth$ با $rw=randrw$ برابر ۱ :

- در اولین آزمون، میزان $IOPS$ برابر ۲۵۸ هزار بود، در حالی که در آزمون دوم این مقدار به ۴۵۳ هزار افزایش یافت.

- پهنهای باند در آزمون اول MB/s ۱۰۵۶ و در آزمون دوم MB/s ۱۸۵۵ بود.
- زمان تاخیر کل در آزمون اول ۱۶۳۸ نانوثانیه و در آزمون دوم ۵۷.۳۸۳۶ نانوثانیه برای خواندن و ۹۰.۴۱۸ نانوثانیه برای نوشتن بود.
- استفاده از پردازنده در آزمون اول بسیار بالا بود (٪۶۳.۹۹) اما در آزمون دوم به ٪۵۶.۴۹ کاهش یافت.
- در آزمایش *:rw=randwrite*
 - با یک نخ، *IOPS* برابر ۵۱۷ هزار بود و پهنهای باند MB/s ۲۱۱۶ به دست آمد.
 - با چهار نخ، *IOPS* به ۸۷۱ هزار و پهنهای باند به MB/s ۳۵۶۹ افزایش یافت.
 - زمان تاخیر کل در حالت یک نخ ۱۳.۱۶۰۹ نانوثانیه و در حالت چهار نخ ۶۰.۴۱۴۱ نانوثانیه بود.
 - استفاده از پردازنده در حالت یک نخ بسیار بالا (٪۸۳.۹۹) ولی در حالت چهار نخ به ٪۴۷.۴۹ کاهش یافت.
- در آزمایش *:rw=randread*
 - با یک نخ، *IOPS* برابر ۵۸۶ هزار و پهنهای باند MB/s ۲۴۰۱ بود.
 - با چهار نخ، *IOPS* به ۱۰۶۵ هزار و پهنهای باند به MB/s ۴۳۶۲ رسید.
 - زمان تاخیر کل در حالت یک نخ ۳۹.۱۵۴۹ نانوثانیه و در حالت چهار نخ ۹۹.۳۵۵۹ نانوثانیه بود.
 - استفاده از پردازنده در حالت یک نخ بسیار بالا (٪۶۶.۹۹) اما در حالت چهار نخ به ٪۶۶.۴۹ کاهش یافت.

۵ ابزار RocksDB

توضیحات

RocksDB یک پایگاه داده کلید-مقدار یا Key-Value بسیار سریع است که توسط Facebook توسعه داده شده است. این پایگاه داده به طور خاص برای ذخیره‌سازی و پردازش داده‌های حجم بالا در سیستم‌های دیسک محور بهینه‌سازی شده است. RocksDB بر اساس ساختار Tree LSM یا Log-Structured Merge Tree ساخته شده است که اجازه می‌دهد عملیات خواندن و نوشتمن به طور مؤثر و با کارایی بالا انجام شوند.

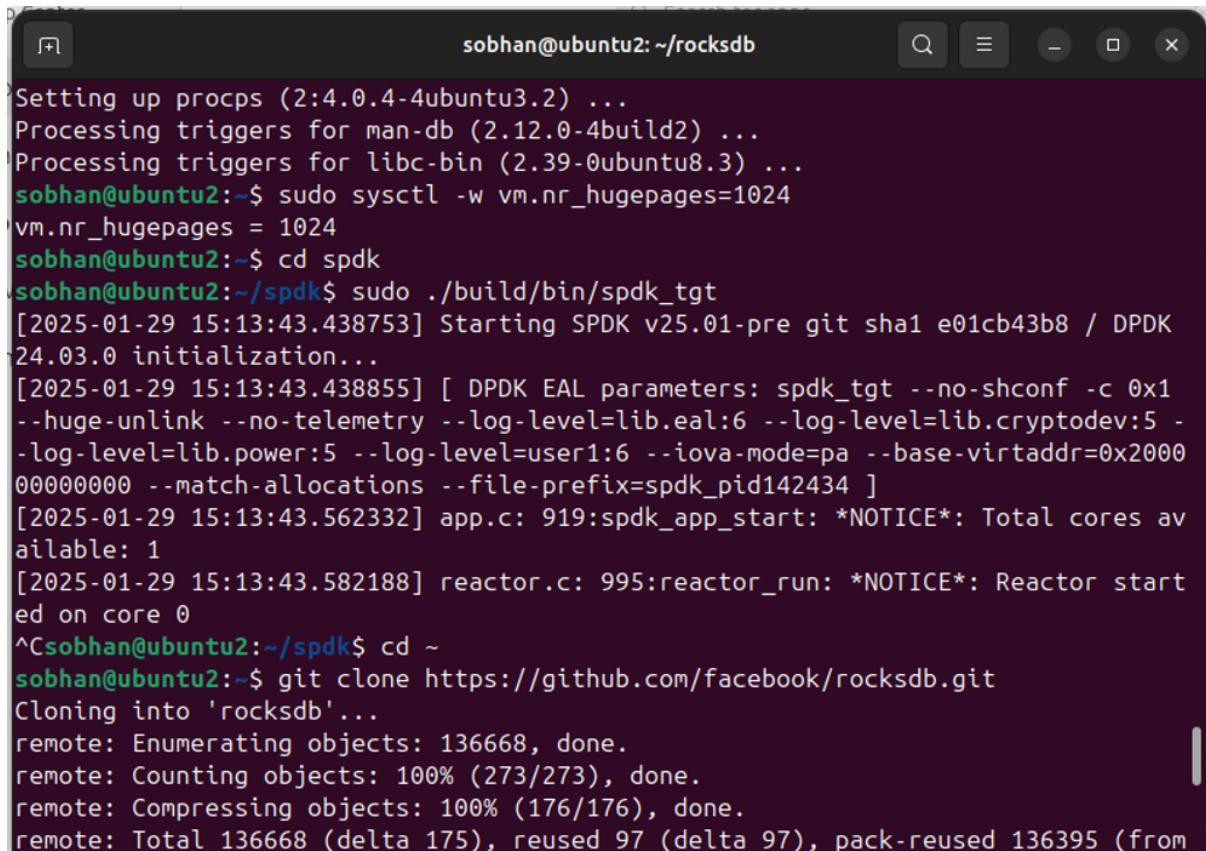
ویژگی‌ها و مزایا اصلی RocksDB:

- **عملکرد بالا:** RocksDB با استفاده از تکنیک‌های بهینه‌سازی حافظه و دیسک، عملکرد بسیار بالایی را برای حجم‌های عظیم داده‌ها ارائه می‌دهد.
- **پشتیبانی از فشرده‌سازی:** این پایگاه داده از انواع مختلف فشرده‌سازی برای ذخیره داده‌ها پشتیبانی می‌کند که باعث کاهش استفاده از فضای ذخیره‌سازی و افزایش سرعت عملیات می‌شود.
- **پشتیبانی از ویژگی‌های پیچیده:** RocksDB می‌تواند برای پردازش داده‌های مقیاس‌پذیر، پشتیبانی از چندین لایه کش (Caching) و تراکنش‌های پیچیده، استفاده شود.
- **پشتیبانی از پردازش موازی:** این پایگاه داده می‌تواند برای خواندن و نوشتمن داده‌ها در چندین نخ یا هسته پردازشی به طور همزمان پردازش کند و به همین دلیل برای سیستم‌های چند هسته‌ای بسیار مناسب است.
- **پشتیبانی از عملیات‌های خواندن و نوشتمن تصادفی و ترتیبی:** با توجه به طراحی Tree، LSM می‌تواند به طور مؤثر هر دو نوع عملیات را پردازش کند.

RocksDB برای برنامه‌های کاربردی‌ای که نیاز به پردازش داده‌های سریع و مقیاس‌پذیر دارند، مانند سیستم‌های ذخیره‌سازی بلادرنگ، تجزیه و تحلیل داده‌ها، و برنامه‌های مبتنی بر تحلیل داده‌های بزرگ یا Big Data بسیار مناسب است. این پایگاه داده معمولاً در سیستم‌های NoSQL، برنامه‌های کشینگ، و سیستم‌های ذخیره‌سازی مقیاس‌پذیر استفاده می‌شود.

مراحل نصب

ابتدا آن را کلون می‌کنیم:



```
sobhan@ubuntu2:~/rocksdb
Setting up procps (2:4.0.4-4ubuntu3.2) ...
Processing triggers for man-db (2.12.0-4build2) ...
Processing triggers for libc-bin (2.39-0ubuntu8.3) ...
sobhan@ubuntu2:~$ sudo sysctl -w vm.nr_hugepages=1024
vm.nr_hugepages = 1024
sobhan@ubuntu2:~$ cd spdk
sobhan@ubuntu2:~/spdk$ sudo ./build/bin/spdk_tgt
[2025-01-29 15:13:43.438753] Starting SPDK v25.01-pre git sha1 e01cb43b8 / DPDK
24.03.0 initialization...
[2025-01-29 15:13:43.438855] [ DPDK EAL parameters: spdk_tgt --no-shconf -c 0x1
--huge-unlink --no-telemetry --log-level=lib.eal:6 --log-level=lib.cryptodev:5 -
--log-level=lib.power:5 --log-level=user1:6 --iova-mode=pa --base-virtaddr=0x2000
00000000 --match-allocations --file-prefix=spdk_pid142434 ]
[2025-01-29 15:13:43.562332] app.c: 919:spdk_app_start: *NOTICE*: Total cores available: 1
[2025-01-29 15:13:43.582188] reactor.c: 995:reactor_run: *NOTICE*: Reactor started on core 0
^Csobhan@ubuntu2:~/spdk$ cd ~
sobhan@ubuntu2:~$ git clone https://github.com/facebook/rocksdb.git
Cloning into 'rocksdb'...
remote: Enumerating objects: 136668, done.
remote: Counting objects: 100% (273/273), done.
remote: Compressing objects: 100% (176/176), done.
remote: Total 136668 (delta 175), reused 97 (delta 97), pack-reused 136395 (from
```

شکل ۳۴: کلون RocksDB

سپس به پوشه مربوطه رفته و make می‌کنیم و (برای مدت طولانی) منتظر می‌مانیم تا روند make تمام شود.

```
sobhan@ubuntu2:~/rocksdb
```

```
2)
Receiving objects: 100% (136668/136668), 219.92 MiB | 5.05 MiB/s, done.
Resolving deltas: 100% (104753/104753), done.
sobhan@ubuntu2:~$ cd rocksdb
sobhan@ubuntu2:~/rocksdb$ make
$DEBUG_LEVEL is 1, $LIB_MODE is shared
Makefile:186: Warning: Compiling in debug mode. Don't use the resulting binary in production
CC      cache/cache.o
CC      cache/cache_entry_roles.o
CC      cache/cache_key.o
CC      cache/cache_helpers.o
CC      cache/cache_reservation_manager.o
CC      cache/charged_cache.o
CC      cache/clock_cache.o
CC      cache/lru_cache.o
CC      cache/compressed_secondary_cache.o
CC      cache/secondary_cache.o
CC      cache/secondary_cache_adapter.o
CC      cache/sharded_cache.o
CC      cache/tiered_secondary_cache.o
CC      db/arena_wrapped_db_iter.o
CC      db/attribute_group_iterator_impl.o
CC      db/blob/blob_contents.o
```

شکل ۳۵ RocksDB make :

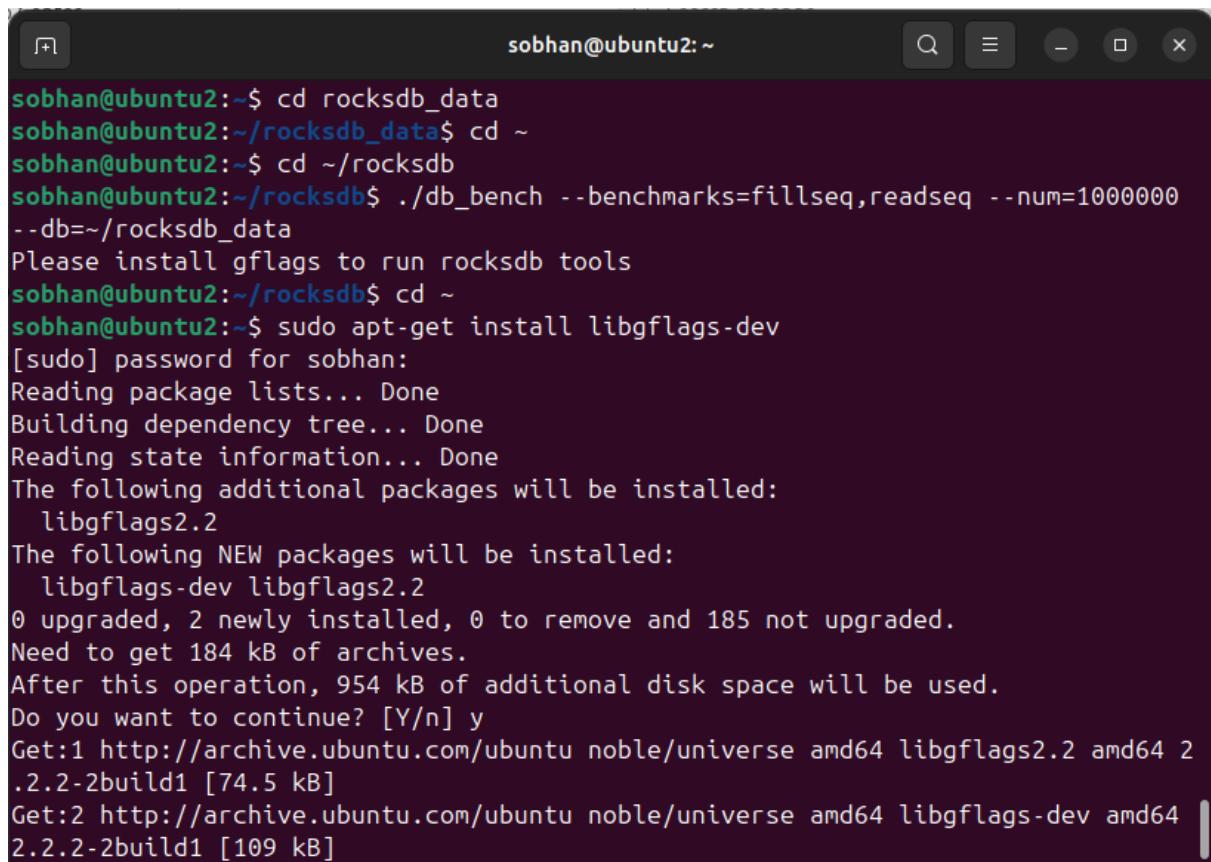
```
sobhan@ubuntu2: ~/rocksdb
CCLD    range_locking_test
CC      utilities/transactions/transaction_test.o
CCLD    transaction_test
CC      utilities/transactions/lock/point/point_lock_manager_test.o
CCLD    point_lock_manager_test
CC      utilities/transactions/write_prepared_transaction_test.o
CCLD    write_prepared_transaction_test
CC      utilities/transactions/write_unprepared_transaction_test.o
CCLD    write_unprepared_transaction_test
CC      utilities/transactions/write_committed_transaction_ts_test.o
CCLD    write_committed_transaction_ts_test
CC      utilities/transactions/timestamped_snapshot_test.o
CCLD    timestamped_snapshot_test
CC      utilities/ttl/ttl_test.o
CCLD    ttl_test
CC      utilities/types_util_test.o
CCLD    types_util_test
CC      utilities/util_merge_operators_test.o
CCLD    util_merge_operators_test
CC      utilities/write_batch_with_index/write_batch_with_index_test.o
CCLD    write_batch_with_index_test
CC      db/c_test.o
CCLD    c_test
sobhan@ubuntu2:~/rocksdb$
```

شکل ۳۶: اتمام make

پس از اتمام make، دستور مورد نظر برای اجرای تست ها را اجرا می کنیم.

با این دستور، اطلاعاتی مانند عملیات در ثانیه (Bandwidth)، تاخیرها (Latencies) و پهنای باند (IOPS) را دریافت می کنیم.

متوجه شدیم که برای اجرای ابزارهای RocksDB به بسته gflags نیاز داریم. لذا ابتدا باید آن را نصب کنیم:

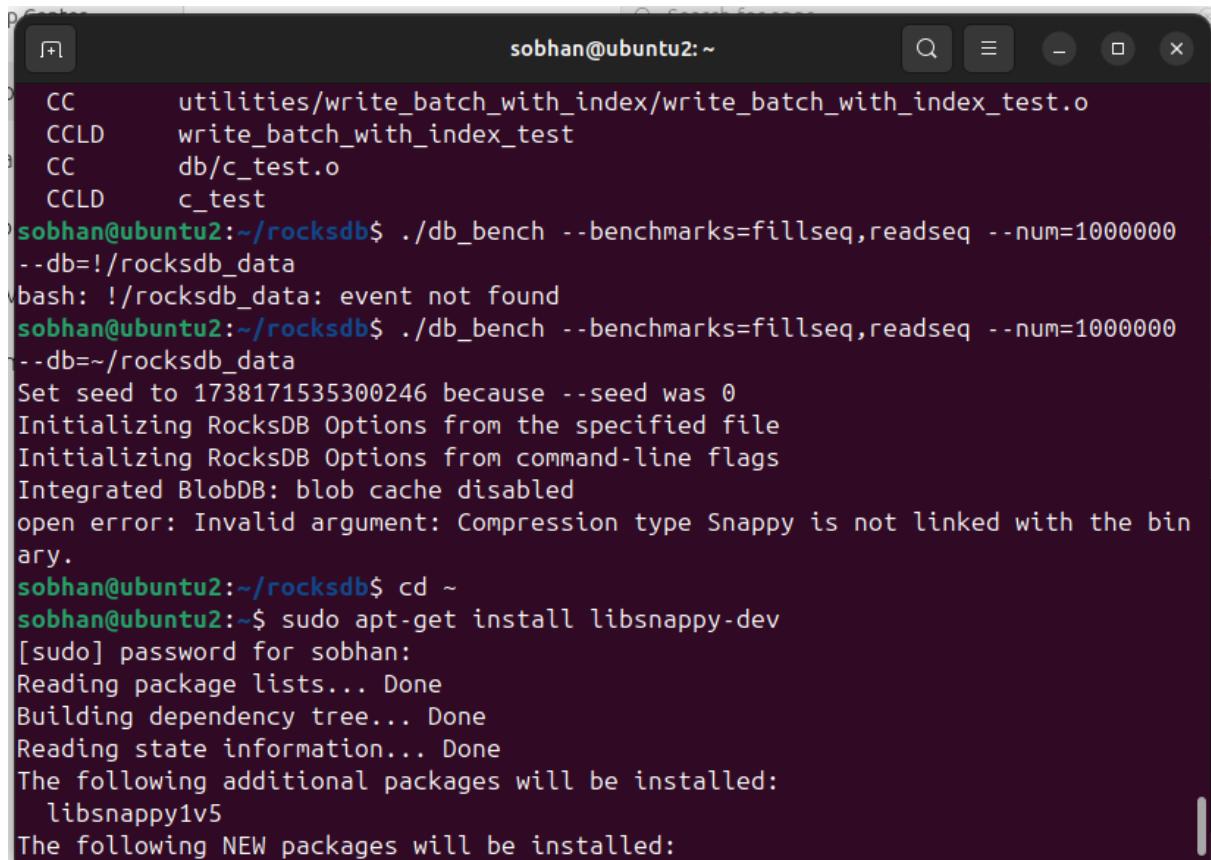


```
sobhan@ubuntu2:~$ cd rocksdb_data
sobhan@ubuntu2:~/rocksdb_data$ cd ~
sobhan@ubuntu2:~$ cd ~/rocksdb
sobhan@ubuntu2:~/rocksdb$ ./db_bench --benchmarks=fillseq,readseq --num=1000000
--db=~/rocksdb_data
Please install gflags to run rocksdb tools
sobhan@ubuntu2:~/rocksdb$ cd ~
sobhan@ubuntu2:~$ sudo apt-get install libgflags-dev
[sudo] password for sobhan:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libgflags2.2
The following NEW packages will be installed:
  libgflags-dev libgflags2.2
0 upgraded, 2 newly installed, 0 to remove and 185 not upgraded.
Need to get 184 kB of archives.
After this operation, 954 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://archive.ubuntu.com/ubuntu noble/universe amd64 libgflags2.2 amd64 2
.2.2-2build1 [74.5 kB]
Get:2 http://archive.ubuntu.com/ubuntu noble/universe amd64 libgflags-dev amd64
2.2.2-2build1 [109 kB]
```

شکل ۳۷: نصب بسته gflags

پس از اتمام نصب، نیاز است دوباره روند make را انجام دهیم. لذا وارد پوشه rocksdb می‌شویم و با make و پس از آن clean، این کار را انجام می‌دهیم.

پس از اجرای دوباره دستور مذکور، متوجه می‌شویم که برای استفاده از فشرده‌سازی با Snappy که نوعی الگوریتم فشرده‌سازی است، اتصال صحیح به RocksDB ایجاد نشده است. لذا باید بسته Snappy نصب شود:



```
sobhan@ubuntu2:~
```

```
CC      utilities/write_batch_with_index/write_batch_with_index_test.o
CCLD    write_batch_with_index_test
CC      db/c_test.o
CCLD    c_test
sobhan@ubuntu2:~/rocksdb$ ./db_bench --benchmarks=fillseq,readseq --num=1000000
--db=/rocksdb_data
bash: !/rocksdb_data: event not found
sobhan@ubuntu2:~/rocksdb$ ./db_bench --benchmarks=fillseq,readseq --num=1000000
--db=~/rocksdb_data
Set seed to 1738171535300246 because --seed was 0
Initializing RocksDB Options from the specified file
Initializing RocksDB Options from command-line flags
Integrated BlobDB: blob cache disabled
open error: Invalid argument: Compression type Snappy is not linked with the binary.
sobhan@ubuntu2:~/rocksdb$ cd ~
sobhan@ubuntu2:~$ sudo apt-get install libsnappy-dev
[sudo] password for sobhan:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libsnappy1v5
The following NEW packages will be installed:
```

شکل ۳۸: نصب بسته Snappy

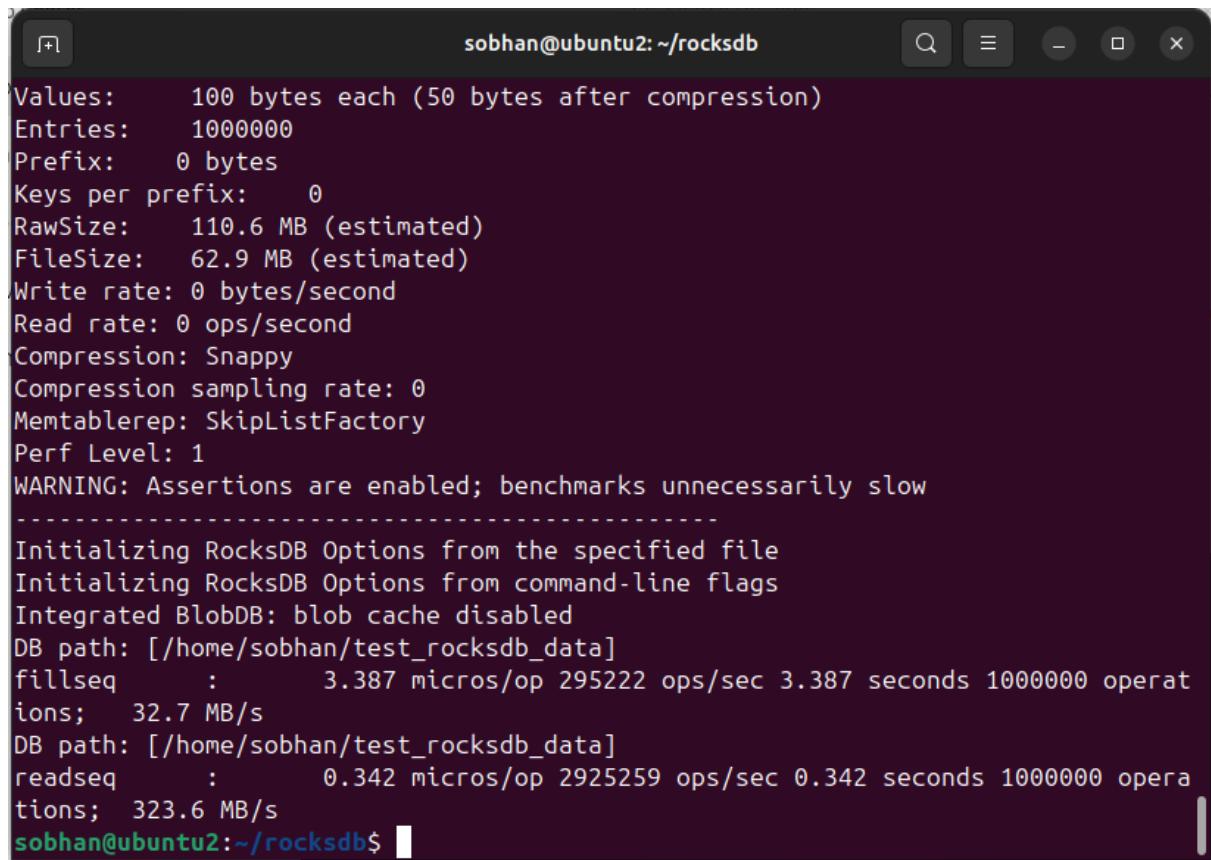
نهایتاً پس از نصب بسته Snappy، دستور db_bench را اجرا می‌کنیم و خروجی را مشاهده می‌کنیم تفاوت این دستور با قبلی این است که آدرس مربوطه را در این دستور به طور کامل نوشته‌ایم و از ~ استفاده نشده است.

بررسی کارکرد

حال که RocksDB را به طور کامل نصب کرده ایم بنچ مارک db_bench را به ترتیب مانند عکس‌های زیر تست می‌کنیم.

```
sobhan@ubuntu2:~/rocksdb$ ./db_bench --benchmarks=fillseq,readseq --num=1000000  
--db=/home/sobhan/test_rocksdb_data  
Set seed to 1738176409314081 because --seed was 0  
Initializing RocksDB Options from the specified file  
Initializing RocksDB Options from command-line flags  
Integrated BlobDB: blob cache disabled  
RocksDB: version 10.0.0  
Date: Wed Jan 29 18:46:49 2025  
CPU: 4 * AMD Ryzen 7 5800H with Radeon Graphics  
CPUCache: 512 KB  
Keys: 16 bytes each (+ 0 bytes user-defined timestamp)  
Values: 100 bytes each (50 bytes after compression)  
Entries: 1000000  
Prefix: 0 bytes  
Keys per prefix: 0  
RawSize: 110.6 MB (estimated)  
FileSize: 62.9 MB (estimated)  
Write rate: 0 bytes/second  
Read rate: 0 ops/second  
Compression: Snappy  
Compression sampling rate: 0  
Memtablerep: SkipListFactory  
Perf Level: 1
```

شکل ۳۹: اجرای دستور db_bench برای حالت ترتیبی



```
sobhan@ubuntu2: ~/rocksdb
Values:      100 bytes each (50 bytes after compression)
Entries:     1000000
Prefix:      0 bytes
Keys per prefix:   0
RawSize:    110.6 MB (estimated)
FileSize:   62.9 MB (estimated)
Write rate: 0 bytes/second
Read rate: 0 ops/second
Compression: Snappy
Compression sampling rate: 0
Memtablerep: SkipListFactory
Perf Level: 1
WARNING: Assertions are enabled; benchmarks unnecessarily slow
-----
Initializing RocksDB Options from the specified file
Initializing RocksDB Options from command-line flags
Integrated BlobDB: blob cache disabled
DB path: [/home/sobhan/test_rocksdb_data]
fillseq      :      3.387 micros/op 295222 ops/sec 3.387 seconds 1000000 operations; 32.7 MB/s
DB path: [/home/sobhan/test_rocksdb_data]
readseq      :      0.342 micros/op 2925259 ops/sec 0.342 seconds 1000000 operations; 323.6 MB/s
sobhan@ubuntu2:~/rocksdb$
```

شکل ۴۰: اجرای دستور `db_bench` برای حالت ترتیبی

پس از آن نیز دستور درون تصویر را اجرا می‌کنیم تا عملکرد با داده‌های تصادفی را نیز بررسی کنیم:

```
sobhan@ubuntu2:~/rocksdb$ ./db_bench --benchmarks=fillrandom,readrandom --num=1000000 --db=/home/sobhan/rocksdb_data
Set seed to 1738176764419889 because --seed was 0
Initializing RocksDB Options from the specified file
Initializing RocksDB Options from command-line flags
Integrated BlobDB: blob cache disabled
RocksDB:      version 10.0.0
Date:        Wed Jan 29 18:52:44 2025
CPU:         4 * AMD Ryzen 7 5800H with Radeon Graphics
CPUCache:    512 KB
Keys:        16 bytes each (+ 0 bytes user-defined timestamp)
Values:      100 bytes each (50 bytes after compression)
Entries:     1000000
Prefix:      0 bytes
Keys per prefix:   0
RawSize:    110.6 MB (estimated)
FileSize:   62.9 MB (estimated)
Write rate: 0 bytes/second
Read rate: 0 ops/second
Compression: Snappy
Compression sampling rate: 0
Memtablerep: SkipListFactory
Perf Level: 1
WARNING: Assertions are enabled; benchmarks unnecessarily slow
```

شکل ۴۱: اجرای دستور db_bench برای حالت تصادفی

The screenshot shows a terminal window with the following text output:

```
Entries: 1000000
Prefix: 0 bytes
Keys per prefix: 0
RawSize: 110.6 MB (estimated)
FileSize: 62.9 MB (estimated)
Write rate: 0 bytes/second
Read rate: 0 ops/second
Compression: Snappy
Compression sampling rate: 0
Memtablerep: SkipListFactory
Perf Level: 1
WARNING: Assertions are enabled; benchmarks unnecessarily slow
-----
Initializing RocksDB Options from the specified file
Initializing RocksDB Options from command-line flags
Integrated BlobDB: blob cache disabled
DB path: [/home/sobhan/rocksdb_data]
fillrandom : 5.509 micros/op 181504 ops/sec 5.509 seconds 1000000 operations; 20.1 MB/s
DB path: [/home/sobhan/rocksdb_data]
readrandom : 9.717 micros/op 102904 ops/sec 9.718 seconds 1000000 operations; 7.2 MB/s (632563 of 1000000 found)

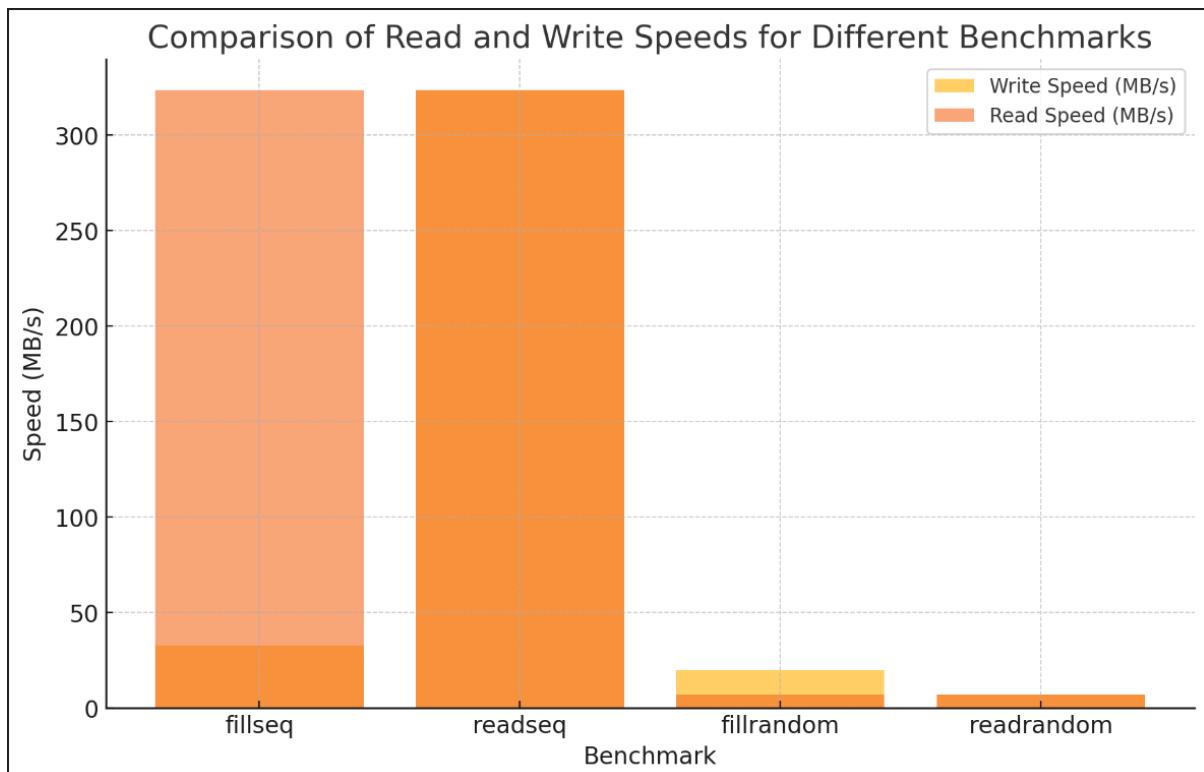
sobhan@ubuntu2:~/rocksdb$
```

شکل ۴۲: اجرای دستور `db_bench` برای حالت تصادفی

بدین شکل خروجی‌های مدنظرمان را دریافت می‌کنیم.

تحلیل نتایج

نمودار مقایسه بر اساس سرعت بین ۴ حالت مختلفمان بین نوشتن و خواندن‌های ترتیبی و تصادفی به شکل زیر می‌باشد:



شکل ۴۳: نمودار مقادیر خروجی برای هر کدام از عملیات

در ادامه به تحلیل این آمار این نمودار و خروجی هایمان می پردازیم:

عملکرد نوشتن (*fillseq, fillrandom*)

نوشتن به صورت ترتیبی (*fillseq*)

- زمان عملیات: ۳۸۷.۳ میکروثانیه

- تعداد عملیات در ثانیه: ۲۹۵۲۲۲

- سرعت نوشتن: 32.7 MB/s

در هنگام نوشتن داده‌ها به صورت ترتیبی، *RocksDB* عملکرد بسیار خوبی دارد. این نوع نوشتن داده‌ها معمولاً به دلیل اینکه داده‌ها به صورت پیوسته در دیسک نوشته می‌شوند، نسبت به نوشتن تصادفی بسیار سریع‌تر است.

نوشتن به صورت تصادفی (*fillrandom*)

- زمان عملیات: ۵۰۹.۵ میکروثانیه

- تعداد عملیات در ثانیه: ۱۸۱۵۰۴

- سرعت نوشتن: 20.1 MB/s

در مقایسه با نوشتن ترتیبی، نوشتن تصادفی زمان بیشتری می‌برد. این تأخیر بیشتر ناشی از تغییر مکان‌های ذخیره داده‌ها در دیسک است. در این روش، *RocksDB* باید جستجوهای بیشتری انجام دهد تا داده‌ها را در مکان‌های مختلف ذخیره کند که باعث افزایش زمان نوشتن می‌شود.

عملکرد خواندن (*readseq, readrandom*)

خواندن به صورت ترتیبی (*readseq*)

- زمان عملیات: 342.0 میکروثانیه

- تعداد عملیات در ثانیه: 2925259

- سرعت خواندن: 323.6 MB/s

همانطور که انتظار می‌رود، خواندن داده‌ها به صورت ترتیبی سریع‌ترین عملکرد را دارد. علت این است که خواندن داده‌ها به صورت ترتیبی به طور معمول به کش و سیستم‌های ذخیره‌سازی پیوسته کمک می‌کند؛ زیرا داده‌ها به طور فشرده و بدون وقفه از دیسک خوانده می‌شوند.

خواندن به صورت تصادفی (*readrandom*)

- زمان عملیات: 717.9 میکروثانیه

- تعداد عملیات در ثانیه: 102904

- سرعت خواندن: 7.2 MB/s

خواندن تصادفی معمولاً زمان بیشتری می‌برد؛ زیرا سیستم باید به طور تصادفی به مکان‌های مختلف دیسک دسترسی پیدا کند. این امر باعث می‌شود که سرعت خواندن از نظر عملکرد بسیار کندر از خواندن ترتیبی باشد. همچنین، به دلیل اینکه داده‌ها به صورت پراکنده ذخیره شده‌اند، عملکرد به طور مستقیم به وضعیت حافظه کش و مکان‌های ذخیره‌سازی بستگی دارد.

مقایسه سرعت‌های خواندن و نوشتن در دو حالات جداگانه

- سرعت نوشتن ترتیبی (*fillseq*): 32.7 MB/s

- سرعت خواندن ترتیبی (*readseq*) : 323.6 MB/s

سرعت خواندن بسیار بیشتر از سرعت نوشتن است، که این رفتار در بسیاری از سیستم‌های ذخیره‌سازی معمولی مشاهده می‌شود. این به این دلیل است که عملیات خواندن به صورت ترتیبی اغلب به حافظه کش یا سیستم ذخیره‌سازی سریع‌تر (مانند SSD) انجام می‌شود و به این ترتیب سرعت خواندن بسیار بالاتر از نوشتن است.

- سرعت نوشتن تصادفی (*fillrandom*) : 20.1 MB/s

- سرعت خواندن تصادفی (*readrandom*) : 7.2 MB/s

در حالت تصادفی، نوشتن و خواندن هر دو کندر هستند. برای نوشتن تصادفی، سیستم نیاز به پردازش بیشتری برای ذخیره داده‌ها در مکان‌های مختلف دارد، و برای خواندن تصادفی، همانطور که اشاره شد، سیستم باید به طور تصادفی به مکان‌های مختلف داده‌ها دسترسی پیدا کند که سرعت را کاهش می‌دهد.

تأثیر فشرده‌سازی

فسرده‌سازی در آزمایش‌ها از نوع *Snappy* است که معمولاً به عنوان یک الگوریتم فشرده‌سازی سریع شناخته می‌شود. معمولاً تأثیر فشرده‌سازی در سرعت خواندن و نوشتن به صورت ترتیبی بیشتر از حالت تصادفی است.

- تأثیر بر نوشتن: فشرده‌سازی می‌تواند حجم داده‌های ذخیره شده را کاهش دهد و سرعت نوشتن را در حالت‌های خاص افزایش دهد. به ویژه برای نوشتن داده‌های بزرگ، می‌تواند عملکرد بهتری را فراهم کند.

- تأثیر بر خواندن: هنگام خواندن داده‌ها، فشرده‌سازی ممکن است کمی زمان بر باشد؛ زیرا داده‌ها باید از حالت فشرده‌شده خارج شوند؛ اما این تأثیر معمولاً در مقایسه با حجم داده‌های ذخیره‌شده قابل چشم‌پوشی است.

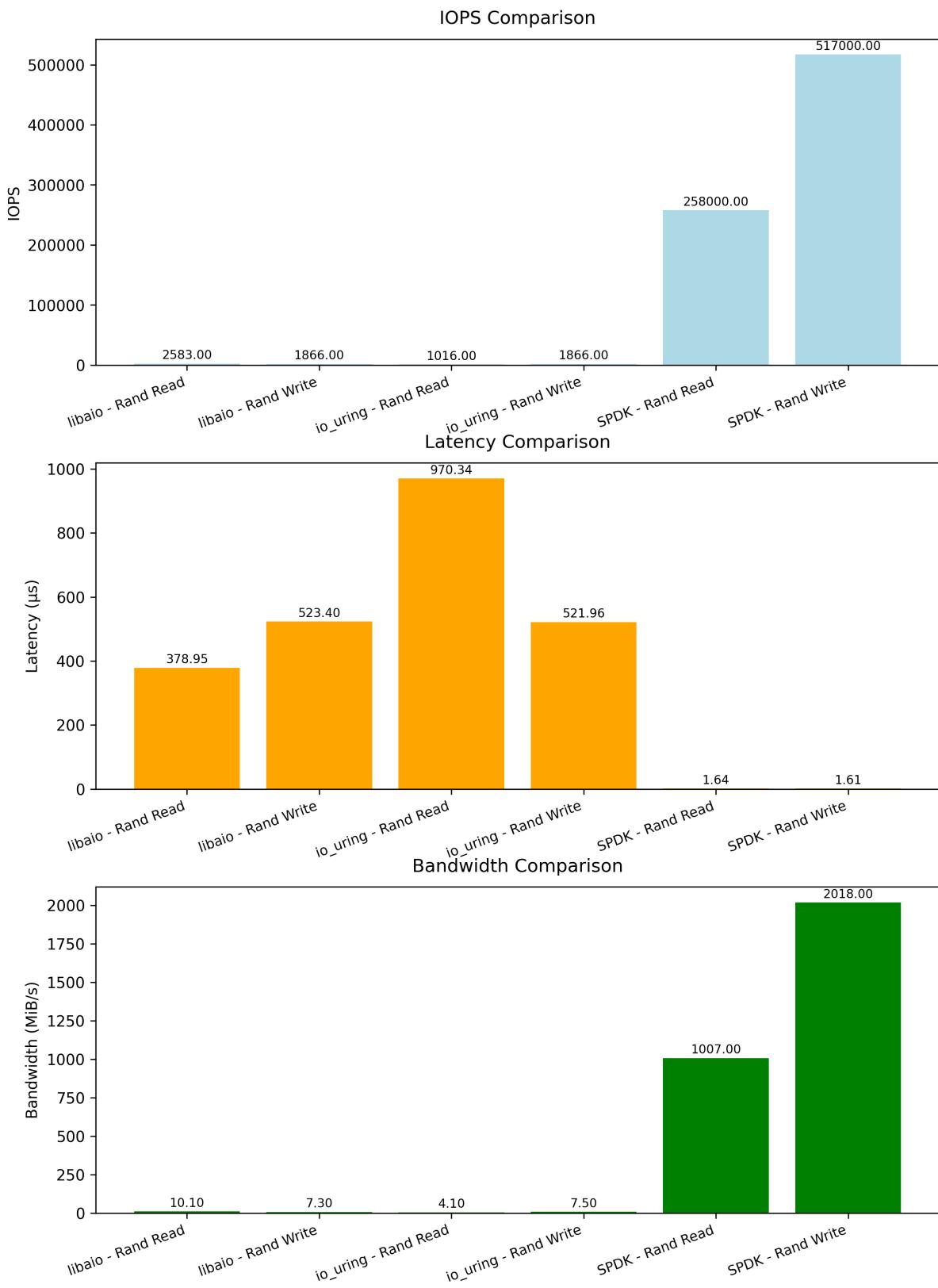
۶ تحلیل و نتیجه‌گیری

تحلیل نتایج حاصل از این تحقیق نشان می‌دهد که در مجموع، عملکرد ابزار *libaio* در سناریوهای با بار کاری پایین و درخواست‌های تصادفی کوچک بهتر از سایر ابزارها است. این ابزار به دلیل ساختار تکمیل درخواست همزمان و تعامل بهتر با کرنل، تأخیر کمتری نسبت به *SPDK* و *io_uring* دارد. در مقایسه، *SPDK* به دلیل استفاده از درایورهای کاربر-محور و اجتناب از تعامل با کرنل، در شرایط با بار پردازشی بالا عملکرد بسیار بهتری از خود نشان داده است.

در آزمایش‌هایی که شامل بارهای کاری سنگین و تعداد پردازش‌های زیاد بودند، *SPDK* توانست با استفاده از صفحه‌ای ورودی/خروجی و پردازش‌های بدون قفل، IOPS بالاتری نسبت به *io_uring* و *libaio* ارائه دهد. این به این معنی است که برای سیستم‌هایی که نیاز به پردازش‌های همزمان و بدون تأخیر بالا دارند، *SPDK* گزینه بهتری است.

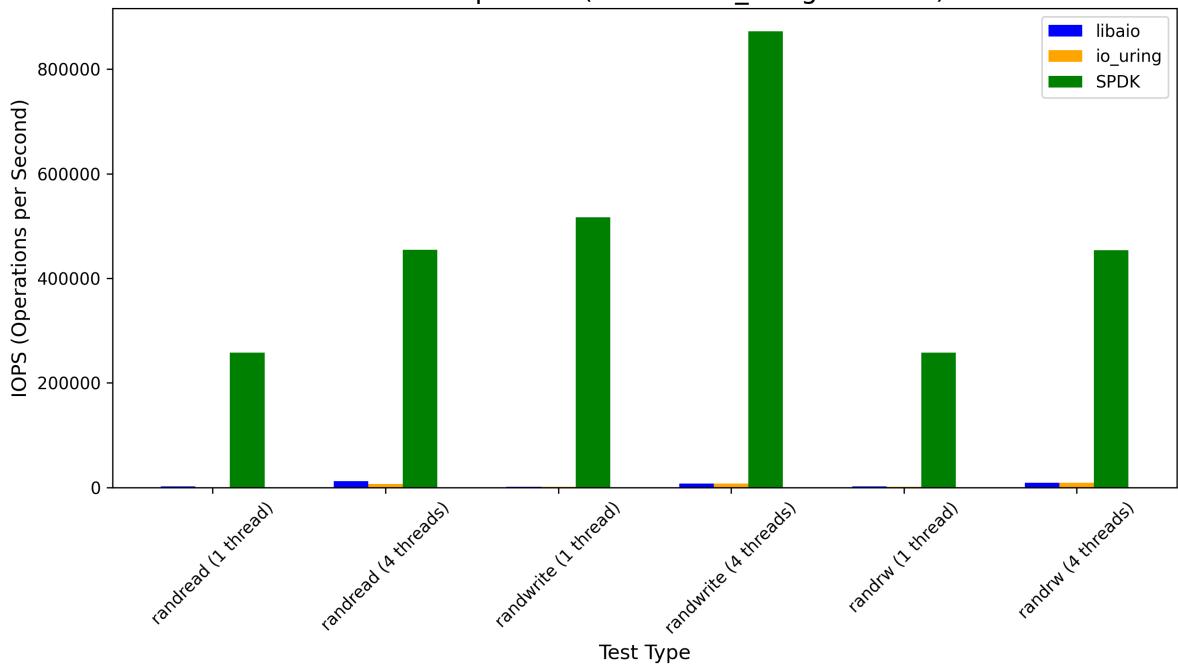
نتایج نشان داد که هر یک از این ابزارها در شرایط خاص خود بهترین عملکرد را دارند. برای مثال، در شرایط با عمق صفحه بالا (*IO depth*)، *io_uring* به دلیل ساختار حلقوی خود عملکرد بهتری از خود نشان می‌دهد. از سوی دیگر، در سیستم‌هایی که نیاز به پردازش‌های سریع دارند، *libaio* انتخاب بهتری است.

در نهایت، برای بهینه‌سازی عملکرد سیستم‌های ذخیره‌سازی واقعی، می‌توان از ترکیب این ابزارها استفاده کرد. به عنوان مثال، در سناریوهایی که نیاز به IOPS بالا دارند، می‌توان از *SPDK* استفاده کرده و در موقعی که تأخیر کم مدنظر است، از *libaio* بهره برد.



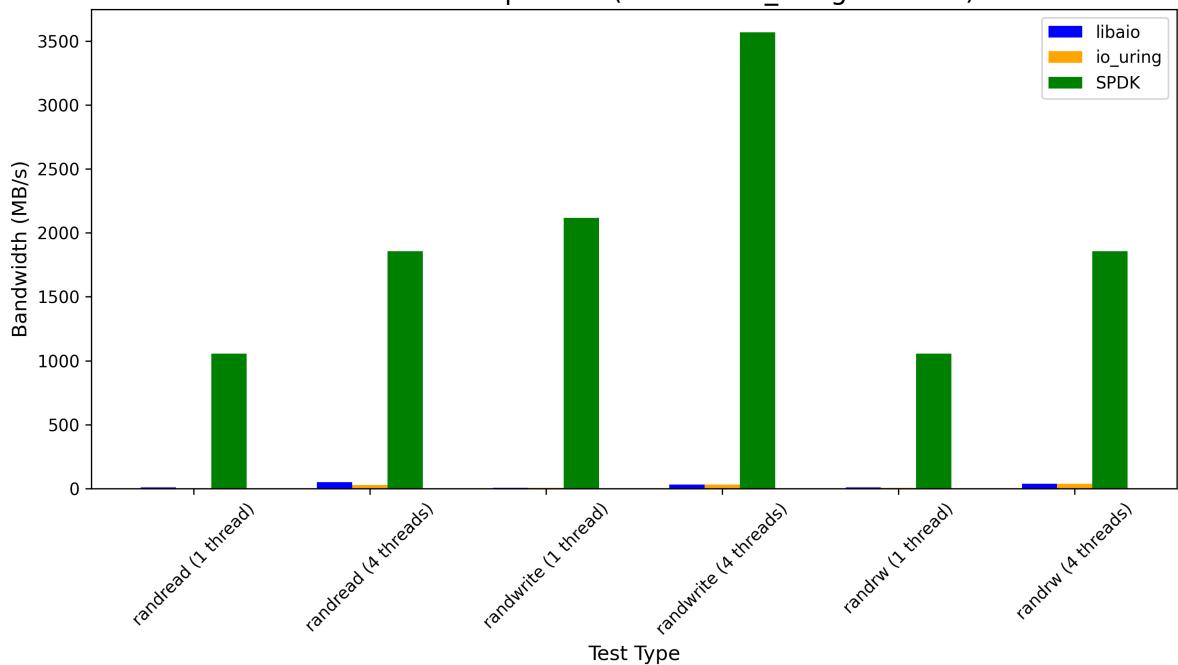
شکل ۴۴: مقایسهی کلی

IOPS Comparison (libaio vs io_uring vs SPDK)

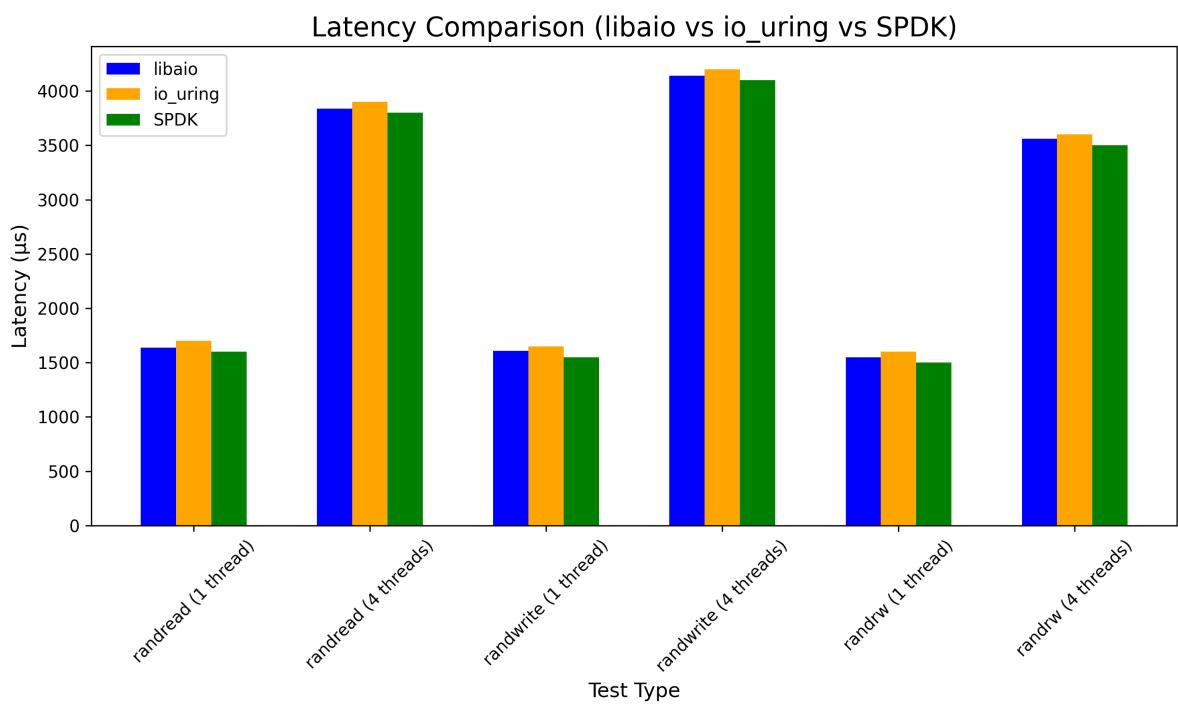


شکل ٤٥: مقایسه IOPS

Bandwidth Comparison (libaio vs io_uring vs SPDK)



شکل ٤٦: مقایسه Bandwidth



شکل ٤٧: مقایسه Latency