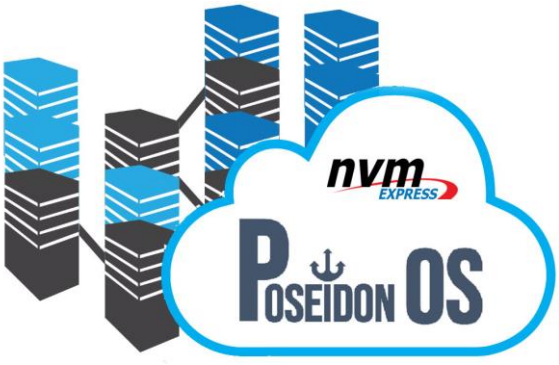
p *+61 1300 113 112*

e *info@hyperscalers.com*

Solving Information Technology's **Complexity**

TECHNICAL WHITEPAPER

PoseidonOS Appliance



ARCHITECTURE AND DEPLOYMENT GUIDE

The PoseidonOS (POS) appliance by Hyperscalers is a distributed storage solution which uses the NVMe-OF protocol. PoSeiDon target server (PSD) delivers a super-fast reliable NVMe RAID5 over the TCP block storage to the system operators and service providers. System administrators can manage and monitor the storage which is facilitated by an interactive Graphical User Interface (GUI).

Created By the Hyperscalers Team
16th November 2021

1 CONTENTS

2 INTRODUCTION

The Poseidon OS (POS) provides an enterprise appliance for super-fast NVMe over TCP. Hyperscalers adopted the open-source light weight storage OS to qualify the state-of-the-art high-speed interface on high density NVMe server S5B (D52B-1U). This appliance is suitable for a dis-aggregated infrastructure and tiered storage datacentre architecture involving databases.

3 ARCHITECTURE

The architecture of POS system consists of a high density NVMe server S5B capable of 12x 2.5" hot-pluggable NVMe SSD with a Debian based Linux distribution which acts as a target system for the application servers on the network to utilise. The NVMe network protocol extended to Ethernet and Fibre channel called as NVMe over Fabrics (NVMe-oF) delivers a highly available and efficient connectivity for the Initiators or clients that require a high-speed storage solution. The below block diagram shows the setup done at Hyperscalers Labs.

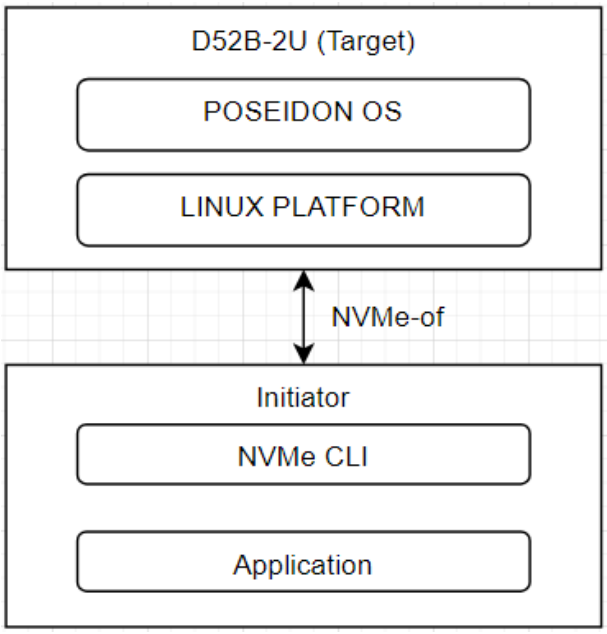


Figure 1 Target-client architecture of PoseidonOS

The key characteristics of POS are as follows [1]:

Highly optimized for low-latency & high-throughput NVMe devices

NVM Express devices allows a host to completely exploit the levels of parallelism possible on a modern SSD and it is attached via dedicated PCI Express bus. They breakthrough the bottlenecks of traditional storage devices (SATA, SAS, HDD, etc) in terms of increased command queue depths, better interrupt processing and uncacheable register access.

The NVMe devices attached to a host eliminates the overhead of context switch for CPU threads that can be assigned for useful work. Thus, the system becomes efficient to provide

a disk-based storage for the applications. POS implements a highly optimized I/O path that minimises the software overhead and maximises the parallelism.

Supporting NVMe-over-Fabrics interface over RDMA and TCP

Remote Direct Memory Access (RDMA) and Transmission Control Protocol (TCP) are the technology and protocol that serves the purpose of sharing the super-fast NVMe with clients on the network. The RDMA technology allows the clients on network to exchange data in the main memory without involving the processor, cache, or operating system. TCP acts as the communication standard that enables the target and client to exchange data over the network. POS utilises these to expose the target's block devices over the network.

By performing dis-aggregated storage, the cloud or storage vendors can isolate their applications and storage devices on a physical level but not compromising on the data access speed. POS provides a flexibility to attach the high-speed tiered storage to any of the application based on requirements. The API interface to the POS management can be useful for the billing process.

Running as user-level application (vs. kernel-level)

POS is tested on UBUTNU 18.04, and it is designed to run as a user-level application. This enables POS to fully control the I/O path and thus avoid the CPU cycles been wasted for general purpose OS activities. This improves the fault tolerance of the POS by leveraging hardware-level isolation and therefore, a driver crash will not bring the whole system down.

Virtualizing storage resources

POS offers virtualized block devices that provide valuable features as well as block read/write functionality: capacity elasticity, performance throttling, data protection (RAID), and more. Customers could benefit from the features for free transparently.

4 NVMe-over-Fabrics (NVMe-oF) Interface

4.1 Why New Interface?

NVMe-oF has emerged to break through the scaling limitations of PCIe-attached NVMe. NVMe-oF allows storage applications to talk to remote NVMe devices over network and consequently unblocks those applications from the physical limitation on the number of PCIe lanes a single host can have. NVMe-oF performs encapsulation/decapsulation at each layer of the software stack to maintain end-to-end NVMe semantics across a range of topology. The following illustrates how an NVMe command travels from client (hosts) to server SSDs through NVMe-oF protocol.

4.2 Network Transport (TCP vs. RDMA)

The network fabric used for NVMe-oF can be several types (i.e., Fibre Channel, RDMA, and TCP) according to the current NVMe-oF 1.1 Specification. POS currently supports TCP and RDMA (RoCE v2). We should be aware of pros and cons of each transport type, so that we could make the best decision under various constraints.

TCP transport relies on commodity network hardware including switches, Ethernet NICs, and cables. Datacentre can utilize their existing TCP network with NVMe/TCP. Operating TCP network infrastructure is relatively cheaper and has wider community support. However, we can't avoid going through kernel stack that uses TCP sockets and handling network interrupts, which could affect the latency of an I/O request, significantly. TCP transport also generally requires more CPU resources than RDMA does.

RDMA transport offloads the entire network processing stack to its custom hardware and saves host CPU cycles to process packets. It can also achieve extremely low latency by bypassing the kernel stack, resulting in better performance. On the other hand, the cost of RDMA-specific switches, NICs, and cables is known to be more expensive than TCP-based commodity hardware because RDMA relies on a lossless network.

The major difference is whether Kernel Space is involved in I/O or not. As described before, a TCP packet goes through Kernel Space with multiple context switches, while RDMA packet bypasses the intermediate layers and directly gets to the User Space.

4.3 POS as an NVMe-oF Target

PSD (Poseidon) server is a (Samsung & Inspur)-engineered storage server that runs POS. POS enables initiator to access PSD server (specifically, NVM subsystem) as if it is a single or multiple local block device(s), while in reality the device is remotely mapped to multiple physical NVMe SSDs on PSD server. POS internally manages a pool of NVMe SSDs called POS array to construct volume(s) and expose the devices as a namespace to the initiator side in accordance with NVMe-oF specification.

PSD server instantiates two NVM subsystems, where the first one (called "0") is assigned with a single controller (called "A") and the second one (called "1") has two controllers (called "A" and "B"). As in the example, an NVMe-oF block device on the Initiator side is always mapped to a single POS volume on the target side, while a single POS volume can be connected by multiple controllers and initiators. The types of available network transport currently are TCP and RDMA (RoCE v2). In addition to the user, I/O path, there is a separate network path that allows a system administrator to perform storage provisioning, monitoring, troubleshooting, testing, etc. This management network is only accessible locally through Unix Domain Socket.

4.4 Device and Array

POS internally manages various types of storage resources to support virtualization. Device and Array are essential building blocks to implement the feature. In this section, we will

explore how they are abstracted out to support various storage use cases and what kinds of administrative knowledge is there to understand the internals of the POS system.

4.4.1 What Is POS Array?

POS array is a user-defined collection of physical storage devices and a basic unit to which storage administrator could issue commands such as status-check, create-volume, delete-volume, mount, or unmount. The same storage device cannot be owned by two different arrays. In this sense, you could think of POS array to group a set of physical storage resources exclusively.

POS array can contain 1 or more POS volumes, each of which can be exposed to initiator(s) as a block device. POS array enforces the same data protection policy across its volumes.

4.4.2 Creating POS Array

Storage administrator may want to manage multiple POS arrays to meet various Service-Level Agreement (SLA) requirements.

- POS can have up to 8 arrays in total.
- Applications having strict performance SLA should run on a dedicated, isolated set of SSD devices, because they should not be interfered by the other applications.
- Certain application has strict durability SLA and needs to have higher redundancy than RAID 5.
- The failure domain of one application needs to be isolated from that of other application. Even if certain POS array gets data loss due to bugs, multiple SSD failures, corruptions, and etc, the issue shouldn't propagate to other POS arrays.
- The lifespans of SSDs should be spread out sufficiently such that we shouldn't have to replace all of them at the same time. Having separate arrays could isolate I/O workloads and effectively spread the lifespans compared to the case that a single array serves all workloads in a mixed fashion.

To create a new array, the following information needs to be provided for POS:

- Array name: POS needs to be able to uniquely identify the target POS array.
- Buffer device: POS accumulates writes in a sequential fashion in each "buffer" device (normally, NVRAM) to improve write performance and latency.
- Data devices: POS stores both user data and metadata in "data" devices. POS internally maintains block mapping to make the best use of SSDs, e.g., by striping across data devices and chunking within a data device.
- Spare device(s): When POS detects a faulty device, it rebuilds missing data onto a spare device by calculating RAID parity. Spare devices can be dynamically attached/detached even when POS array is mounted.
- RAID type: POS protects user data with RAID. Currently only RAID5 is supported.

POS validates if user-supplied inputs are well-formed and complies with system constraints:

- The length of an array name can be up to 63 characters. The possible character is [a-zA-Z0-9_-].
- POS array must have exactly one buffer device by design.
- The minimum number of data devices is 3. This is required by RAID 5.
- The name of a buffer/data/spare must be picked up from SPDK runtime. Getting started describes what those names look like and how to retrieve them by using POS CLI.
- The capacity of a buffer device must be equal to or larger than "(128 MB * # of data devices) + 512 MB".
- The maximum number of data and spare devices in total is 32.
- The capacity of a single NVMe SSD must be between 20 GB and 32 TB.

4.4.3 Partition

POS introduces a logical storage resource called POS partition that internally provides multi-devices abstraction for other components. Every "data" device within an POS array is divided into three areas for different purposes. The collection of the same area from all "data" devices is defined as POS partition. Hence, POS array has three partitions, each of which spanning across all data devices. POS partition enables other software components to access multiple data devices in parallel without detailed knowledge about their physical layout.

The types of POS partition are as follows:

- Master Boot Record (MBR) partition stores array configuration. If POS or underlying host gets restarted for any reason, POS would retrieve the previous array information from MBR partition and reconstruct its internal data structures in memory.
- Meta-data partition manages logical-to-physical mappings and write operations in the partition to support virtualization and consistency.
- User-data partition stores actual user content transferred from initiator(s).

The layout of three areas within a data device looks the same. We name them as MBR, metadata, and user data "partition area" respectively. Here is how POS calculates the size of each partition area and the capacity of POS array:

- MBR partition area = 256 KB
- Metadata partition area = 2% * the raw size of a data device.
- User data partition area = the raw size of a data device - MBR partition area - metadata partition area
- User data partition area reserves 10% of its size for Over Provisioning area.
- Effective user data partition area = user data partition area * 0.9
- The capacity of POS array = effective user data partition area * (# of data devices - # of parity devices)

4.4.4 RAID

POS implements data redundancy at POS partition level. Parity device in POS is defined as a device that contains device recovery information.

- User data partition is protected by RAID 5. The user data partition area is divided into multiple chunks, each of which containing either data or parity bits. POS stores parity chunk in a round-robin fashion, so keeps one parity device effectively.

When POS detects a data device failure, it automatically enters "degraded" mode, which is a fallback mode that continues general array operations but with potential performance degradation caused by 1) decreased read parallelism and 2) resource contention due to RAID rebuild operation.

POS conditionally initiates data construction procedure, called as "RAID rebuild", to replace a failed device with a new one. In degraded mode, POS checks periodically whether there is available spare device and proceeds to rebuild if there is any. Otherwise, POS array will remain in degraded mode until a spare device is newly added. The RAID rebuild doesn't block user I/Os from being processed, although it may impact on the performance of user I/Os. Please note that RAID rebuild generates intense internal I/Os to copy blocks between spare and data devices. POS offers a CLI command called "perf_impact" for a storage administrator to be able to adjust the level of the intensity and find a balance between fast recovery and user's perf SLA.

4.4.5 State Transition Diagram

Upon internal/external events, POS may switch from one state to another and adjust its behaviour. For example, if POS is in PAUSE state to recover from journals, it will block any user I/Os during the period. In this section, we will explain the states that POS could enter and what their implications are.

4.4.5.1 Types of Array State

Array is always in one of POS states in the following table.

State	Description
OFFLINE	POS array has been created or loaded, but not mounted yet.
STOP	POS array is running into an unusual situation that it cannot be corrected by itself. POS array becomes unavailable to avoid further data loss/corruption. For example, if the number of failed devices exceeds the level of fault tolerance (e.g., 2 failures in RAID 5-protected POS partition), a lost device cannot be restored.
NORMAL	POS array is operating normally and able to serve user I/Os without performance impact.
BUSY	POS array is handling both internal I/Os and user I/Os. The user I/O performance could be potentially degraded due to resource contention.

State	Description
PAUSE	POS array is blocking user I/Os to change online - offline state.

4.4.5.2 Types of Situations

Situation indicates the detailed explanation of POS array state. Situation is always mapped to a single state, while State can be mapped to multiple Situations.

Situation	Description	State
DEFAULT	Initial state of POS array	OFFLINE
NORMAL	POS array is operating normally and able to serve user I/Os without performance degradation or blockers	NORMAL
TRY_MOUNT	POS array is being mounted. It is retrieving configuration information and checking integrity	PAUSE
DEGRADED	One storage device has failed, resulting in RAID rebuild	BUSY
TRY_UNMOUNT	POS array is performing "unmount" by blocking both user and internal I/Os	PAUSE
JOURNAL_RECOVERY	During "mount" operation, POS array has detected unapplied journal logs from meta-data partition and started recovery operation	PAUSE
REBUILDING	RAID rebuild operation is in progress	BUSY
FAULT	POS array is not able to proceed with RAID rebuild because the number of failed devices has exceeded the level the array can tolerate	STOP

5 KEY DELIVERABLES

Storage server that can expose the NVMe data pool as a block device to the clients. Clients can read and write any data type into the exposed storage pool.

6 TOOLS USED

- IP Appliance Design Process Document Click here
- Appliance Optimizer Utility Click here
- Anydesk – Remote desktop with recording tool and player
- SSH
- Regression Testing Tool
- Shell script tool
- FIO
- IOMeter

7 HARDWARE SPECIFICATION

- Networking – 1U 100G Switch (BMS T7032-IX1)
- Server – D52B-1U (S5B)
    - 2x Platinum 8276M Processor 28c 2.20 - 4.00 GHz 38.5 MB 165W DDR4 2933
    - 24x DDR4_LR 23400(2933MHz) 64GB Register Samsung M386A8K40CM2-CVF
    - 2x SSD - 2.5'' SATA 6Gb\s 480GB Samsung PM883 HSMZ7LH480HAHQ (480GB DWPD = 1.3)
    - 5x 2.5'' U.2 NVMe SSD HGST ULTRASTAR SN200 7.6TB HUSMR7676BDP3Y1 0TS1357 (7.6TB DWPD = 1)
    - 1x Mellanox ConnectX-5 EN Dual-Port 100GbE DA/SFP NIC

Figure 2 High density fast storage NVMe server

- System management - IPMI v2.0 Compliant, on board "KVM over IP" support
- Operating environment –
    - Operating temperature: 5°C to 35°C (41°F to 95°F)
    - Non-operating temperature: -40°C to 65°C (-40°F to 149°F)
    - Operating relative humidity: 50% to 85%RH.
    - Non-operating relative humidity: 20% to 90%RH
- Video - Integrated Aspeed AST2400 with 8MB DDR3 video memory
- Operating system – UBUNTU 18.04

8 NETWORKING

In this PoC, we have used the NVMe over TCP for network connection and the same can be performed on a RDMA network switch as required.

The BMS T7032-IX1 is installed with QNOS5 Network Operating System (NOS) and the switchports are configured to 100GbE ethernet throughput. The QNOS CLI commands to setup the default gateway for the switch is as below

serviceport protocol none

serviceport ip *ipaddress netmask* [*gateway*]

The configuration of each port to behave as a 100GbE speed interface is shown below:

(Switch) #configure

(Switch) (Config)#interface 0/1

(Switch) (Interface 0/1)#speed-duplex 100000

(Switch) (Interface 0/1)#exit

9 OPERATING SYSTEM AND HARDWARE

The LINUX operating system installed on the storage server is UBUNTU 18.04 LTS (BIONIC BEAVER) and the kernel version is 5.4.0-91-generic. The below are the list of certified Operating System that can be used on the S5B (D52B-1U) storage server that can support PoseidonOS NVME over TCP

Operating System	Support
Ubuntu Server 16.04 LTS	Yes
Red Hat Enterprise Linux 7.3 x86_64 including KVM	Yes
Red Hat Enterprise Linux 6.8 x86_64 including KVM	Yes
Red Hat Enterprise Linux 7.6 x86_64 including KVM	Yes
Ubuntu Server 18.04.2 LTS	Yes

10 INSTALLING POSEIDONOS

To install the PoseidonOS NVME-of solution, open a terminal in UBUNTU 18.04 and make a directory named "ibofos" in "/root/workspace/". Pull the Git-hub repository for the PoseidonOS using the command below.

git clone https://github.com/poseidonos/poseidonos.git

Note: There must be at least three drives to proceed with the PoseidonOS installation as the system provides redundancy on the storage using RAID 5. Additional drives can be added to the storage pool or as spare drives in case of a RAID member drive failure.
There are several dependency packages for the PoseidonOS like SPDK which can be installed by running the build script below

```
cd script
sudo ./pkgdep.sh
```

The libraries and directories for the PoseidnOS can be installed with the script below. The build script has a cmake version incompatibility that can be cleared by reinstalling the 3.20.2 version of cmake using the command below

```
cd lib
sudo ./build_lib.sh
sudo apt remove --purge cmake
wget https://github.com/Kitware/CMake/releases/download/v3.20.2/cmake-3.20.2.tar.gz
tar -zxvf cmake-3.20.2.tar.gz
cd cmake-3.20.2
./bootstrap
Make
sudo make install
```

After building the library, the source code for PoseidonOS is built using the command below and this will create the bin folder to run the PoseidonOS application.

```
cd script/
sudo ./build_ibofos.sh
cd script/
sudo ./start_poseidonos.sh
ps -ef | grep poseidonos | grep -v grep
```

This will start the PoseidonOS and the system will need a URAM location to store the storage volume memory. The below command will create a new uram location of 8GB size and add it to the devices available for clustering.

```
./bin/poseidonos-cli device create --device-name uram0 --device-type uram --num-blocks
16777216 --block-size 512
./bin/poseidonos-cli device scan
./bin/poseidonos-cli device list –unit
```

The below command creates a RAID 5 array of the five drives along with the uram for exposing it via TCP NQN subsystem.

./bin/poseidonos-cli array create --array-name POSArray --buffer uram0 --data-devs unvme\
-ns-0,unvme-ns-1,unvme-ns-2,unvme-ns-3,unvme-ns-4,unvme-ns-5 --raid RAID5

./bin/poseidonos-cli array list --array-name POSArray --unit

./bin/poseidonos-cli array mount --array-name POSArray

These five drives can be exposed via subsystem NQN by creating a new subsystem and transport layer via TCP. The command below can be used to "nqn.2019-04.ibof:subsystem1" with buffer size of 4K.

./bin/poseidonos-cli subsystem create --subnqn nqn.2019-04.ibof:subsystem1 --serial-number\
IBOF00000000000001 --model-number IBOF_VOLUME_EXTENSION --max-namespaces 256 -o

./bin/poseidonos-cli subsystem create-transport --trtype tcp -c 64 --num-shared-buf 4096

Now we create a new volume of the required size that is available from the array and this volume will be automatically exposed via TCP network to all the clients in the network after we mount it to the PoseidonOS.

./bin/poseidonos-cli volume create --volume-name vol1 --array-name POSArray --size 50TB --maxiops 0 --maxbw 0

./bin/poseidonos-cli volume mount --volume-name vol1 --array-name POSArray

./bin/poseidonos-cli subsystem add-listener -q nqn.2019-04.ibof:subsystem1 -t tcp -i <target ip address> -p 1158

The clients can use the below command to connect from anywhere to this storage pool if they are in the same network.

sudo systemctl stop ufw.service

sudo apt-get install nvme-cli

sudo modprobe nvme

sudo modprobe nvme-tcp

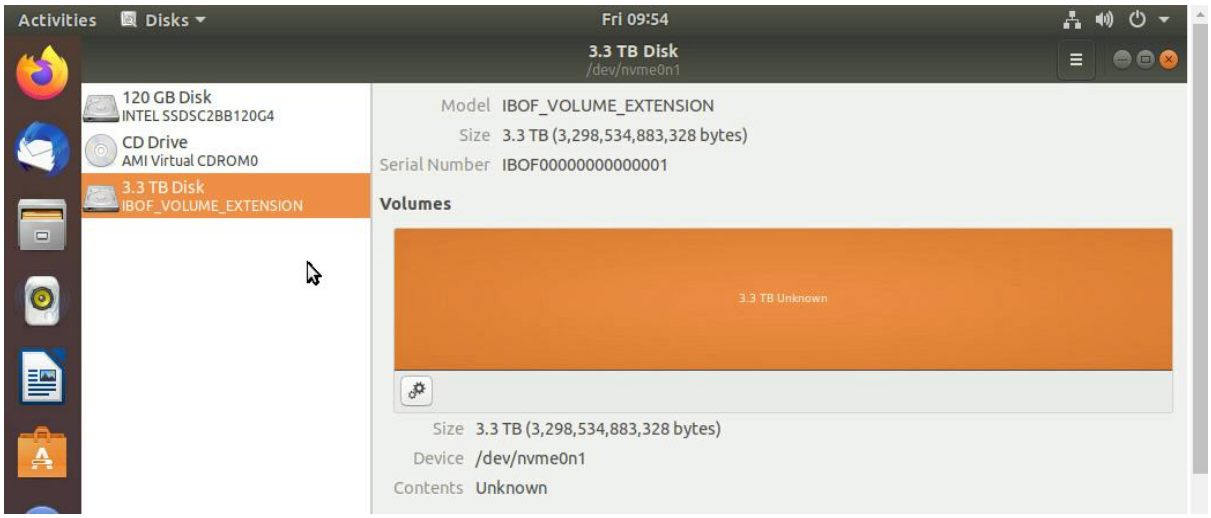
sudo nvme discover -t tcp -a <target IP> -s 1158

sudo nvme connect -t tcp -n nqn.2019-04.ibof:subsystem1 -a <target ip> -s 1158

```
ubuntu@ubuntu-desktop:~/Downloads$ sudo nvme discover -t tcp -a 192.168.18.220 -
s 1158

Discovery Log Number of Records 1, Generation counter 5
=====Discovery Log Entry 0======
trtype:  unrecognized
adrfam:  ipv4
subtype: nvme subsystem
treq:    not required
portid:  0
trsvcid: 1158
subnqn:  nqn.2019-04.ibof:subsystem1
traddr:  192.168.18.220
ubuntu@ubuntu-desktop:~/Downloads$ sudo nvme connect -t tcp -n nvme-test -a 192.
168.18.220 -s 1158
```

The Client will be seeing a block NVMe device via its operating system as shown below.



To monitor the status of the system, device array and the subsystem NQN, we can use the below set of commands.

./poseidonos-cli system info

./poseidonos-cli array list

./poseidonos-cli subsystem list --subnqn nqn.2019-04.ibof:subsystem1

As an uninstallation procedure, below are the commands that had to be run in the specified sequence to unmount the storage volume, delete the volume, unmount the array of the drives and delete the array.

./poseidonos-cli volume unmount --volume-name vol1 --array-name POSArray

./poseidonos-cli volume delete --volume-name vol1 --array-name POSArray

./poseidonos-cli array unmount --array-name POSArray

./poseidonos-cli array delete --array-name POSArray

./poseidonos-cli system stop

To stop the entire PoseidonOS application, use the poseidonos CLI system stop command.

11 INSTALLING POSEIDONOS GRAPHICAL USER INTERFACE (GUI)

The pre-requisites for the Poseidon OS are listed below. For deploying the PoseidonOS GUI, recommended operating system is UBUNTU version 18.04. The NIC interface must be set with a static IP address to ensure the target is highly available in the network.

Prerequisite-

1. python3
2. go v1.14+
3. nodejs 14.x
4. InfluxDB (1.8.x)
5. The POS setup on the machine should run from this directory - /root/workspace/ibofos/ (e.g the POS binaries should be present at /root/workspace/ibofos/bin directory)

Setup-

The source code is pulled from the GitHub repository and "install_all" shell script ensures the dependencies are met for the PoseidonOS GUI. Run the "build_all" script to build the source code and finally "run_all" script will host the PoseidonOS GUI in localhost for access and management.

git clone https://github.com/poseidonos/poseidonos-gui.git

cd poseidonos-gui

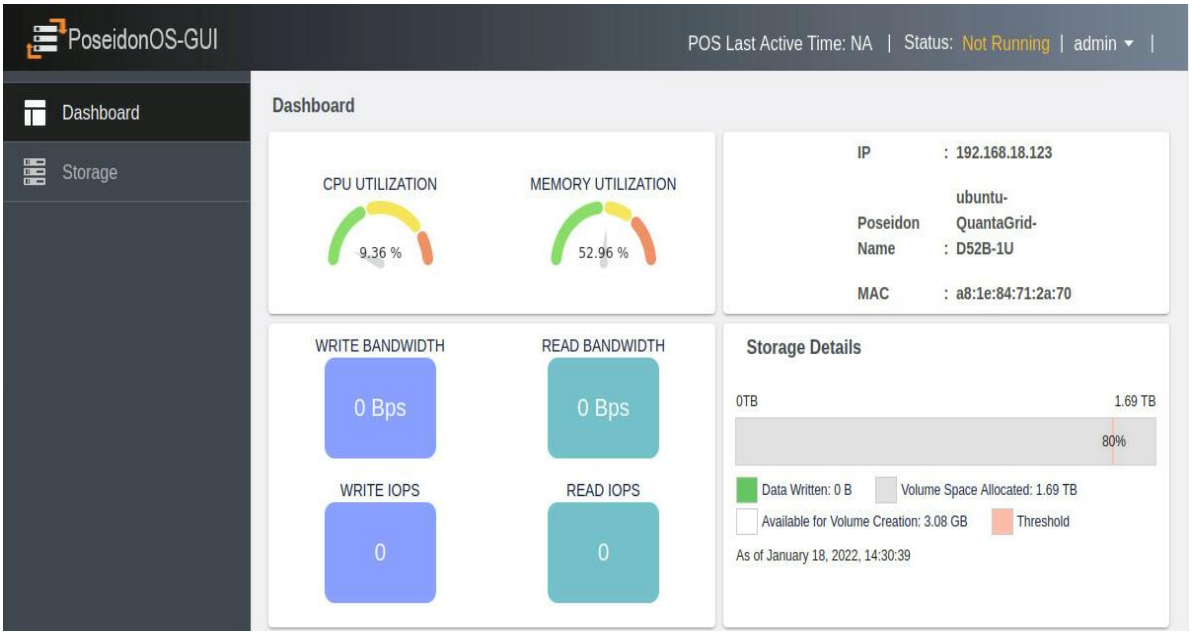
./script/install_all.sh

./script/build_all.sh

./script/run_all.sh

Access the application from the browser by hitting http://localhost

The dashboard of the PoseidonOS GUI looks like below. The Storage details like available space, used space and volume information of the NVMe over TCP can be monitored from the Dashboard. Also, it gives critical information about the CPU and Memory utilization along with the IP of the target system.



The storage section of the PoseidonOS consists of create and manage options that enables the datacentre operations to create a new array of disks, mount or delete the array. This will make the user interface easier for managing the NVMe array and storage volume.

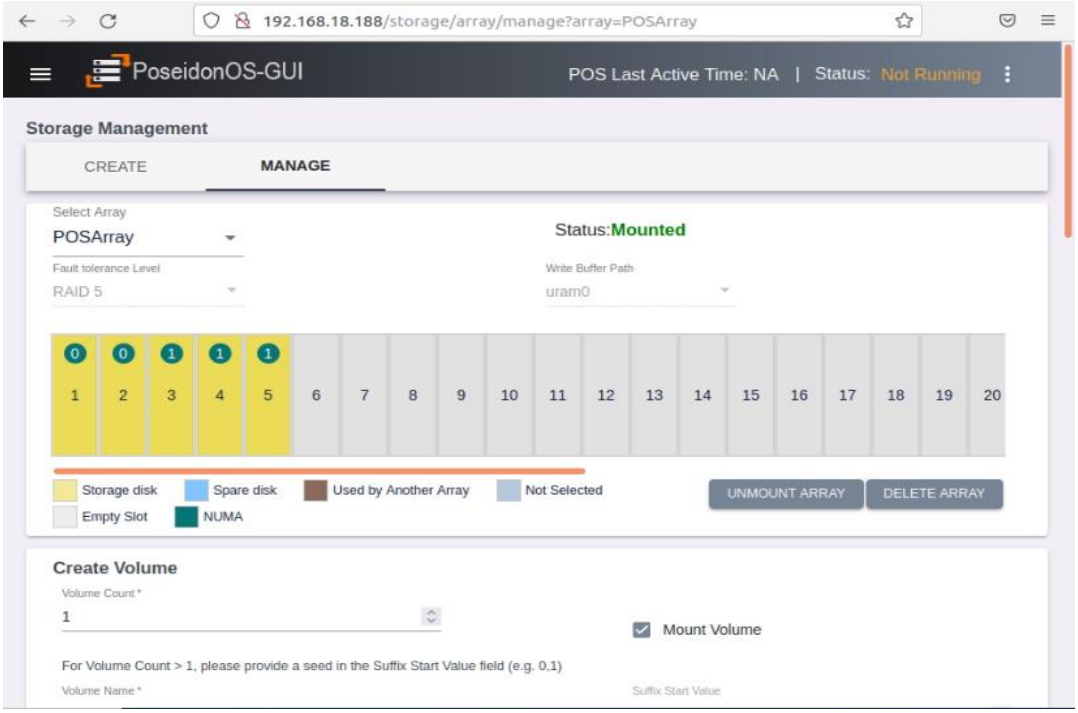


Figure 3 PoseidonOS five drive pool

The redundancy for any drive failure is addressed by the RAID 5 on the disks and it is necessary to at least have three drives for the PoseidonOS NVMe RAID pool. We can have spare drives in case the drive failure needs to be immediately replaced with the spare drive. This can eliminate the need for immediate manual drive replacement and eliminate the risk of additional drive failure causing the RAID pool to fail.

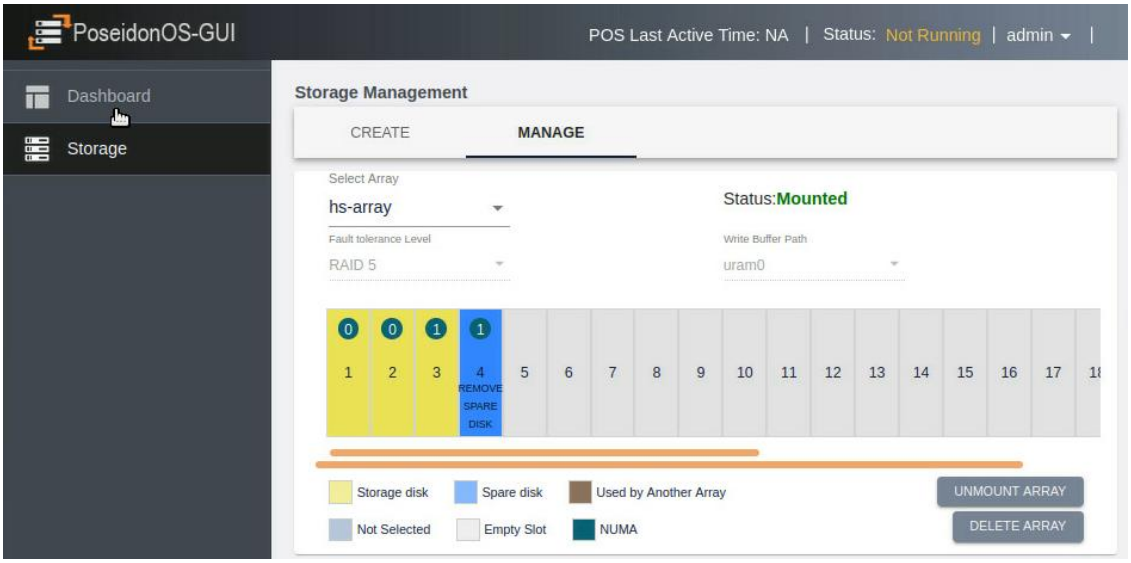


Figure 4 PoseidonOS three drive pool with one drive as spare

12 SPEED TEST

The baseline UBUNTU Disks benchmarking is performed with the client connected to the block storage of block size 2.5TB and the average read rate was about 1.4 GB/s and average write rate was about 865 MB/s through a network of bandwidth 5 GB/s.

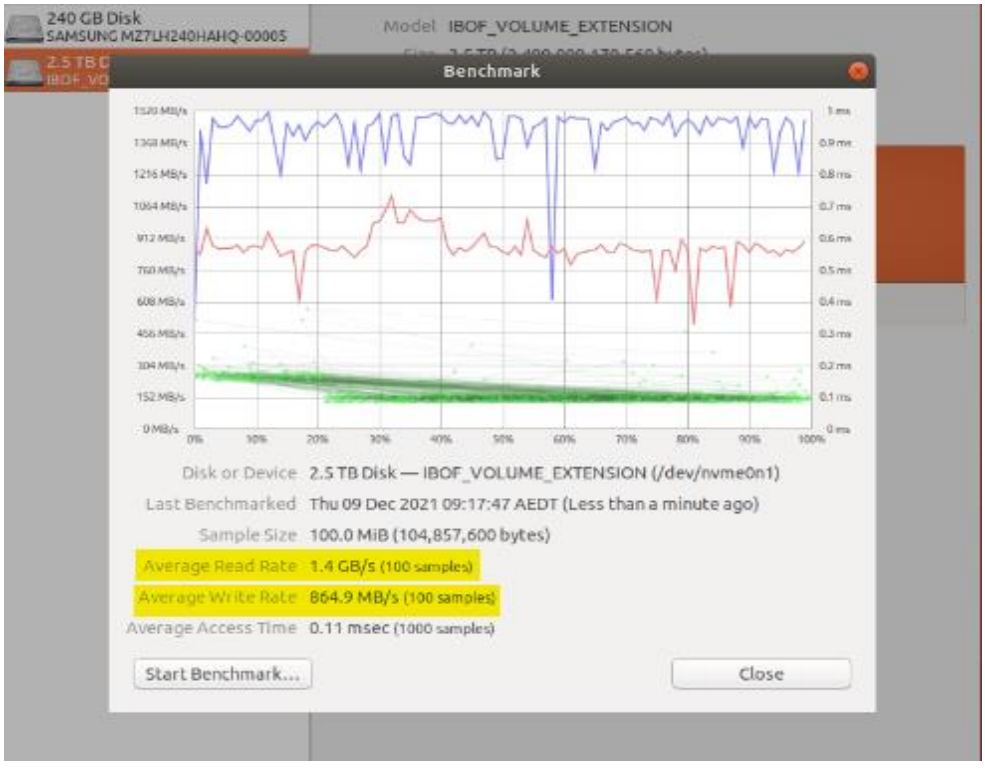


Figure 5 Disks Benchmark

Performing the FIO 3.7 sequential read test on the client that connects to the PoseidonOS target, we achieve a bandwidth of 1.9GB/s with 4k block size. The command used for FIO test is given below:

```
[client@localhost poseidon]$ sudo fio --name=seqread --ioengine=libaio --iodepth=16 --rw=read
 --bs=4k --direct=0 --size=512M --numjobs=40 --runtime=240 --group_reporting
[sudo] password for client:
seqread: (g=0): rw=read, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libai
o, iodepth=16
...
fio-3.7
```

Figure 6 FIO sequential read test

```
seqread: (groupid=0, jobs=40): err= 0: pid=31766: Tue Jan 25 10:02:03 2022
   read: IOPS=486k, BW=1900MiB/s (1992MB/s)(20.0GiB/10779msec)
    slat (nsec): min=1281, max=60770k, avg=78002.12, stdev=1057785.73
    clat (nsec): min=1367, max=60860k, avg=1190442.59, stdev=3955818.76
     lat (usec): min=3, max=60864, avg=1268.56, stdev=4074.91
    clat percentiles (usec):
     |  1.00th=[   45],  5.00th=[   47], 10.00th=[   49], 20.00th=[   51],
     | 30.00th=[   53], 40.00th=[   58], 50.00th=[   62], 60.00th=[   70],
     | 70.00th=[  106], 80.00th=[ 1057], 90.00th=[ 3556], 95.00th=[ 5866],
     | 99.00th=[16909], 99.50th=[33817], 99.90th=[46924], 99.95th=[50070],
     | 99.99th=[54789]
   bw (  KiB/s): min=14848, max=80624, per=2.54%, avg=49434.00, stdev=10431.54, samples=818
   iops        : min= 3712, max=20156, avg=12358.47, stdev=2607.87, samples=818
  lat (usec)   : 2=0.01%, 4=0.01%, 10=0.01%, 20=0.01%, 50=15.43%
  lat (usec)   : 100=53.95%, 250=8.30%, 500=1.34%, 750=0.49%, 1000=0.41%
  lat (msec)   : 2=5.47%, 4=6.22%, 10=6.31%, 20=1.22%, 50=0.81%
  lat (msec)   : 100=0.05%
  cpu          : usr=1.22%, sys=5.73%, ctx=121155, majf=0, minf=1036
  IO depths    : 1=0.1%, 2=0.1%, 4=0.1%, 8=0.1%, 16=100.0%, 32=0.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.1%, 32=0.0%, 64=0.0%, >=64=0.0%
     issued rwts: total=5242880,0,0,0 short=0,0,0,0 dropped=0,0,0,0
     latency   : target=0, window=0, percentile=100.00%, depth=16

Run status group 0 (all jobs):
   READ: bw=1900MiB/s (1992MB/s), 1900MiB/s-1900MiB/s (1992MB/s-1992MB/s), io=20.0GiB (21.5GB
), run=10779-10779msec
```

Figure 7 Sequential read performance results

13 VOLUMES

13.1 WHAT IS POS VOLUME?

POS volume is storage resource visible as a block device to a client host. Internally, POS volume is an NVM namespace and attached to one of the NVM subsystems on a target. When POS volume is mounted, the corresponding NVM namespace allows incoming connection and is accessible to initiator(s). When the connection is established, NVM subsystem creates a new controller, from which point the initiator can send/receive block requests over the connection.

Here are a few constraints on the relationship among the components:

- POS volume cannot be shared by multiple NVM namespaces.
- NVM controller is associated with one host at a time.

The following figure elaborates more on how block devices on an initiator are mapped to NVM namespaces on a target. The target is exposing two NVM namespaces from NVM subsystem 0 and another two from NVM subsystem 1. Each NVM namespace is called as POS volume and abstracted as a block device on the initiator. The naming convention of a standard NVMe-oF block device is, /dev/nvme {subsystem number} n {namespace number}. For example, /dev/nvme1n1 will be mapped to (NVM subsystem 1, NVM namespace 1) on a target.

```
Node          SN                Model                                  Namespace Usage                 Format         FW Rev
----------------- -------------------- ---------------------------------------- --------- -------------------------- ---------------- --------
/dev/nvme0n1    IBOF00000000000001  IBOF_VOLUME_EXTENTION                 1          2.15 GB /   2.15 GB
512   B + 0 B   19.10
/dev/nvme0n2    IBOF00000000000001  IBOF_VOLUME_EXTENTION                 2          2.15 GB /   2.15 GB
512   B + 0 B   19.10
/dev/nvme1n1    IBOF00000000000002  IBOF_VOLUME_EXTENTION                 1          2.15 GB /   2.15 GB
512   B + 0 B   19.10
/dev/nvme1n2    IBOF00000000000002  IBOF_VOLUME_EXTENTION                 2          2.15 GB /   2.15 GB
512   B + 0 B   19.10
```

13.2 Constraints on POS Volume Creation

POS validates user-supplied inputs when creating POS volume. The properties of POS volume must satisfy the following constraints:

- The size must be multiples of 1 MB.
- The minimum size is 1 MB.
- The maximum size can be as large as the free space of POS array.
- The size can be specified in units of Byte, Mega Byte, Giga Byte.
- The length of a volume name must be between 2 and 255 (inclusive).
- Any whitespace(s) in the front or the end of a volume name is trimmed.
- Only the following character set is allowed for a volume name: [a-zA-Z0-9_-]
- The volume name must be unique within POS array.
- The maximum number of POS volumes is limited to 256 for a single POS array.
- POS array must be in "NORMAL" (i.e., successfully mounted) or "BUSY" state to be able to create new POS volume.

13.3 Constraints on Throttling POS Volume Performance

Storage administrator can choose to set performance limit on per-volume basis. The maximum bandwidth (BW) or IOPS value can be provided during volume creation or configured dynamically afterwards. This configuration affects both read/write performance of a volume.

POS validates the configuration and uses it only when they are within an expected range. Otherwise, POS may cut it down or pick up a smaller one. Here are a few contracts and constraints:

- If the sum of (BW, IOPS) pairs from all POS volumes exceeds the capacity of POS performance, POS chooses a smaller value than what is provided by a user.
- If both IOPS and BW are set, POS chooses whichever is satisfied first for throttling.
- BW must be between 10 ~ 17592186044415(UINT64_MAX / 1024 / 1024) and in the unit of MiB.
- If BW is null (not given) or 0, POS will not impose any BW limit on a volume.
- IOPS must be between 10 ~ 18446744073709551(UINT64_MAX / 1000) and in the unit of KIOPS.

- If IOPS is null (not given) or 0, POS will not impose any IOPS limit on a volume.
- POS array must be in "NORMAL" (i.e., successfully mounted) or "BUSY" state to be able to configure a volume with (BW, IOPS).

13.4 Mounting POS Volume

"Mount" operation establishes NVMe-oF connection between an initiator and a target. If storage administrator does not specify an NVM subsystem to attach POS volume to, POS chooses the next available NVM subsystem in a round-robin fashion.

Restriction: A subsystem can only be associated with a single array, whereas an array can be associated with multiple subsystems. The association between an array and a subsystem is established when a volume of the array is first mounted to the subsystem. Once an association is established, the volumes of the other arrays cannot be mounted to that subsystem.

13.5 Unmounting Volume

"Unmount" operation detaches POS volume from its corresponding NVM subsystem, stopping I/O requests between an initiator and a target.

14 Data Persistence and Consistency

14.1 Overview

In this section, we describe how POS persists user data and guarantees metadata consistency to deal with crash failures. We also discuss the limitation of POS at the time of writing. This will help to set the right expectation about the level of persistence in the current software release and hardware configuration. POS adopts software-only approach for persistence and consistency, making it highly portable and flexible across different types of hardware.

14.2 Data Persistence and Consistency Issues

When user data arrives at POS, it is first accumulated in internal storage resource called "write buffer" and then later flushed into NVMe SSDs; this is one of the key I/O path optimizations. The buffer is located within a buffer device of POS array, hence inheriting the same persistence of the buffer device. POS sends back write acknowledgement to an initiator right after the buffering, without waiting for the result of the flushing. This technique effectively reduces write latency.

Sometimes, however, it is possible that POS crashes due to power failure before flushing the user data into NVMe SSDs, although POS sent back the write acknowledgement to the initiator. To deal with this situation, POS should be able to handle the following issues:

- Persistence issue: If a buffer device is allocated from volatile media (e.g., DRAM) for any reason, the write buffer would be lost after a server reboot. In this case, an initiator would think its data should be persistent on POS since it got write ack already, but in fact has lost its data.
- Consistency issue: POS may have been in the middle of updating its internal metadata, resulting in partial updates. After a server reboot, POS might see inconsistent image of metadata such as dangling pointers or orphans.

POS offers a few workarounds to deal with such expected issues.

14.3 How To Keep Data Persistent and Metadata Consistent

14.3.1 Issue NVM Flush Command

NVMe specification explains what to expect from NVM flush command: "The Flush command shall commit data and metadata associated with the specified namespace(s) to non-volatile media". Provided that POS volume is implemented as an NVM namespace, it is supposed to copy all dirty blocks to NVMe SSDs synchronously upon receiving the command. The feature was enabled/disabled through build option in earlier version, but it has been configurable through config file change (at /etc/pos/pos.conf) since 0.9.2. It is not enabled by default yet. This does not solve the sudden crash problem perfectly but can help to reduce the data loss window if it is run on a reasonably short interval.

14.3.2 Unmount POS Array Gracefully

We offer the best practice to shut down POS array, which is called "graceful unmount". The word "graceful" means that user application prepares for a stop and should not observe any errors during the procedure. The following steps guarantee that all in-transit writes are successfully done to NVMe SSDs consistently:

1. Stop receiving user I/Os from the initiator by stopping user application and unmounting the file system
2. Unmount POS volume from the target
3. Unmount POS array from the target
4. Stop POS at the target

This still does not solve the sudden crash problem, but can help with shutting down POS reliably for any maintenance job, e.g., OS patch, library upgrade, POS configuration change, etc.

14.3.3 Enable Journal

With enabling journal and configuring non-volatile write buffer, both persistence and consistency can be achieved transparently without any explicit user commands such as NVM flush and graceful exit. All write-acknowledged data in the buffer would be eventually flushed to NVMe SSDs even in case of sudden crashes. Also, all metadata updates are recorded and

replayed to implement all-or-nothing semantics, achieving the consistency. This feature is experimental at the time of writing and will be included in a later release.

With volatile write buffer, it is required to issue NVM flush command explicitly or exit Poseidon OS gracefully to store user data and meta data safely.
With non-volatile write buffer, user data consistency can be guaranteed by enabling journal.

With journal feature disabled, it is desirable to issue NVM flush command on a periodic basis and/or perform graceful exit to ensure persistence and consistency.

With journal feature enabled, the metadata consistency is guaranteed but the data can be lost on power or kernel failure. User data can be recovered from POS's internal failures if the write buffer is restored before POS starts up.

14.4 REFERENCES:

[1] "GitHub - poseidonos/poseidonos: Poseidon OS (POS) is a light-weight storage OS", *GitHub*, 2021. [Online]. Available: https://github.com/poseidonos/poseidonos. [Accessed: 21- Nov- 2021].