1-client-amd-nvme-tcp-dpu

System description

MangoBoost

August 21, 2024

Contents

1 Hardware

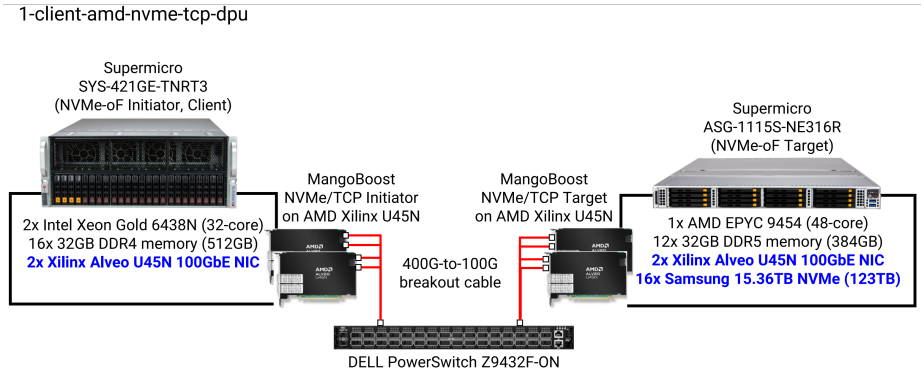

Figure 1: System under test.

Figure 1 shows the system under test. One client and one storage server are connected to the same switch. The client and server has 400G aggregate bandwidth with four 100G ports. On both, client and server sides, MangoBoost DPU is deployed to offload NVMe/TCP processing from the host CPU to the DPU.

1.1 Client

The client is a commercial Supermicro GPU SuperServer SYS-421GE-TNRT3. The MangoBoost DPUs in the server provides NVMe/TCP Initiator Offloading.

1.2 Server

The server is a commercial Supermicro Storage A+ Server ASG-1115S-NE316R. The MangoBoost DPUs in the server provides NVMe/TCP Target Offloading.

1.3 Switch

The switch is a commercial DELL PowerSwitch Z9432F-ON.

2 Software

2.1 Client

The OS is Ubuntu 22.04.4 LTS (GNU/Linux 5.15.0-101-generic x86_64). The NVMe/TCP connections are made in the DPU and exposed as local NVMe devices to the host.

2.2 Server

The OS is Ubuntu 22.04.4 LTS (GNU/Linux 5.15.0-101-generic x86_64). We use a modified Storage Performance Development Kit v23.9 (SPDK) to provide MangoBoost NVMe/TCP target offloading driver and NVMe/TCP target to the client.

3 Settings

3.1 Client

MangoBoost NVMe/TCP Initiator makes NVMe/TCP connections in the on-board ARM processor. The connected NVMe devices are exposed as local NVMe devices to the host at `/dev/nvme*`.

We configured RAID0 to fully exploit the aggregate performance of NVMe SSDs.

```
sudo pvcreate /dev/nvme*n1
sudo vgcreate lvm_raid /dev/nvme*n1
sudo lvcreate --extents 100%FREE --name lvm_raid lvm_raid -i16 -I64k

sudo mkfs.ext4 /dev/lvm_raid/lvm_raid
sudo mount /dev/lvm_raid/lvm_raid /mnt
```

3.1.1 Drop pagecache in the background

When pagecache is full, OS needs to reclaim pagecache pages to read the data from the storage. The training workload is not cache friendly since the dataset size is at least five times bigger than the host memory size. In this case, the page reclamation overhead can significantly slow down the read performance. To avoid page reclamation overhead in the critical path of read I/O, we ran a background process to drop pagecache periodically. This optimization can significantly improve the number of supported accelerators.

```
#!/usr/bin/env bash

PID=$$
PROC_LIMIT=15

rm /tmp/*drop_cache.pid 2&>/dev/null

drop () {
  echo $1 | sudo tee /proc/sys/vm/drop_caches > /dev/null &
  CHILD_PID=$!
  touch /tmp/$CHILD_PID.drop_cache.pid
  echo "$(date) START $CHILD_PID ($1, $(ls /tmp/*drop_cache.pid | wc -l) processes)"
  wait
  rm /tmp/$CHILD_PID.drop_cache.pid
  echo "$(date) FIN $CHILD_PID ($(ls /tmp/*drop_cache.pid 2>/dev/null | wc -l) processes)"
}

while true; do
  REMAINED=$(ls /tmp/*drop_cache.pid 2>/dev/null | wc -l)
  if [[ $REMAINED -le $PROC_LIMIT ]];
  then
    drop 1 &
    drop 2 &
  fi
  sleep 7
done
```

3.2 Server

The server exposes NVMe devices using MangoBoost SPDK software. MangoBoost SPDK binds NVMe devices to DPU and then exposes them to clients

through NVMe/TCP. In the SUT, 4 NVMe devices are assigned to each of the four ports in the server.