Using Intel DCPMM to accelerate Redis in Kubernetes environment

# Introduction

This documentation is about how to enable pmem in a all-in-one kubernetes cluster and run benchmark on redis application to show performance improvement of leveraging Intel DCPMM.

**Intel DCPMM**: Intel Optane DC Persistent Memory Module, is an innovative memory technology featured with large capacity and data persistence. It has 2 operation modes. **Two Level Memory Mode** requires no software changes, DCPMM is seen as a bigger memory, and use DRAM as its cache layer. **AppDirect Mode** expose the device as persistent memory, with software stack supported, it can be used to accelerate different applications. We’ll use the AppDirect Mode in this article. For more information about DCPMM, see

<https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory.html>

**Kubernetes**: Kubernets is an open-source system for automating deployment, scaling, and management of containerized applications. It groups containers that make up an application into logical units for easy management and discovery. In the example below, we will run redis applications in Kubernetes and show performance improvement of redis by adopting Intel DCPMM.

**pmem-redis**: Redis is an open source, in-memory data structure store, used as a database, cache and message broker. Pmem-redis is one redis version that support Intel DCPMM(Data Center Persistent Memory) based on open source redis-4.0.0. It benefits the redis's performance by taking advantage of competitive performance and persistency of DCPMM.

Redis provides a mechanism of persistence called AOF. The AOF persistence logs every write operation received by the server, that will be played again at server startup, reconstructing the original dataset. Commands are logged using the same format as the Redis protocol itself, in an append-only fashion. Redis is able to rewrite the log on background when it gets too big.

Pmem-redis provides an optimization on AOF called PBA(Pointer based AOF) by utilizing pmem. Instead of writing keys and values in AOF file, pmem-redis stores values in pmem and only stores pointers in AOF. In the benchmark section, we will compare performances of volatile Redis, AOF enabled mode and PBA enabled mode.

# Cloud deployment

## Physical environment

Intel(R) Xeon(R) Gold 6240 CPU @ 2.60GHz

16G DRAM \* 12

256G Intel DCPMM \* 12

## System software preparation

Centos 7.6

Kubernetes 1.15

pmem-redis

memtier\_benchmark

pmem-CSI

## Install and customize k8s

We are using Kubespray to deploy Kubernetes, for more information about Kubespray, please check out <https://github.com/kubernetes-sigs/kubespray>.

Before deploying Kubernetes, some configuration changes about resource reservations are needed to ensure cpu pinning in Kubernetes.

Edit group\_vars/k8s-cluster/k8s-cluster.yml file, uncomment the following lines to set system reserved resources:

*## Optionally reserve resources for OS system daemons.*

*system\_reserved: true*

*## Uncomment to override default values*

*system\_memory\_reserved: 512M*

*system\_cpu\_reserved: 500m*

And add the following lines into file:

*kubelet\_node\_config\_extra\_args:*

*cpuManagerPolicy: static*

After that, you can continue to run Kubespray to deploy your Kubernetes cluster.

To make sure changes have been applied into your Kubernetes cluster, start a proxy for Kubernetes api and check node config:

*kubectl proxy*

*http* [*http://localhost:8001/api/v1/nodes/*](http://localhost:8001/api/v1/nodes/)*{{ node\_name }}/proxy/configz*

## Set up a local Docker registry

# docker run -d -p 5000:5000 --restart=always --name registry registry:2

You can download pmem csi image and push it to local registry:

# sudo docker pull intel/pmem-csi-driver:v0.7.16

# sudo docker tag intel/pmem-csi-driver:v0.7.16 localhost:5000/pmem-csi-driver:latest

# sudo docker push localhost:5000/pmem-csi-driver:latest

You should see the image stored in the registry

# curl --insecure localhost:5000/v2/\_catalog

{"repositories":["pmem-csi-driver"]}

## Configure DCPMM in app direct mode(both bare metal server and virtual machine)

For a kubernetes cluster deployed on a bare metal server:

First install ipmctl library:

cd /etc/yum.repos.d/

wget https://copr.fedorainfracloud.org/coprs/jhli/ipmctl/repo/epel-7/jhli-ipmctl-epel-7.repo

wget https://copr.fedorainfracloud.org/coprs/jhli/safeclib/repo/epel-7/jhli-safeclib-epel-7.repo

yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm

subscription-manager repos --enable "rhel-\*-optional-rpms" --enable "rhel-\*-extras-rpms" --enable "rhel-ha-for-rhel-\*-server-rpms"

yum install ndctl ndctl-libs ndctl-devel libsafec rubygem-asciidoctor

yum install ipmctl

Enable DCPMM in app direct mode:

# sudo ipmctl delete -goal

# sudo ipmctl create -goal PersistentMemoryType=AppDirect

A reboot is required to process new memory allocation goals:

# sudo reboot

If you are using a kubernetes cluster inside OpenStack vm, you have to manually initialize persistent memory:

# yum install ndctl

# ndctl disable-region region0

# ndctl init-labels nmem0

# ndctl enable-region region0

# ndctl list -RN

[

{

"dev":"region0",

"size":34357641216,

"available\_size":34357641216,

"max\_available\_extent":34357641216,

"type":"pmem",

"iset\_id":10248187106440278,

"persistence\_domain":"unknown"

}

]

For information about pre-provision on persistent memory, please see <https://github.com/intel/pmem-csi#persistent-memory-pre-provisioning>.

## Deploy pmem-CSI plugin in k8s

First please make sure that the system has been configured to use Intel DCPMM in App Direct mode. Second, it is necessary to make sure that the volume groups are clean. It can be checked by using the following command:

# sudo vgs

Please confirm that it should return nothing before moving forward.

Install pmem-CSI plugin:

# cd pmem-CSI

Labeling the node that provide the persistent memory devices, please replace {{ node\_name }} with the one in your environment

# kubectl label node {{ node\_name }} pmem-csi.intel.com/node={{ node\_name }}

# kubectl label node {{ node\_name }} storage=pmem

Setting up certificates for securities

# curl -L https://pkg.cfssl.org/R1.2/cfssl\_linux-amd64 -o \_work/bin/cfssl --create-dirs

# curl -L https://pkg.cfssl.org/R1.2/cfssljson\_linux-amd64 -o \_work/bin/cfssljson --create-dirs

# chmod a+x \_work/bin/cfssl \_work/bin/cfssljson

# export PATH=$PATH:$PWD/\_work/bin

# ./test/setup-ca-kubernetes.sh

Deploying the driver to K8s using LVM mode, please choose yaml files corresponding to your kubernetes version

# kubectl create -f deploy/kubernetes-1.14/pmem-csi-lvm.yaml

Applying a storage class

# kubectl apply -f deploy/kubernetes-1.14/pmem-storageclass-ext4.yaml

Let's confirm that everything is configured and running correctly so far:

# kubectl get nodes --show-labels

NAME STATUS ROLES AGE VERSION LABELS

clr1 Ready master 22h v1.14.1 beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=clr1,kubernetes.io/os=linux,node-role.kubernetes.io/master=,**pmem-csi.intel.com/node=clr1,storage=pmem**

# kubectl get po

NAME READY STATUS RESTARTS AGE

pmem-csi-controller-0 2/2 Running 0 22h

pmem-csi-node-rmc6s 2/2 Running 0 22h

# kubectl get sc

NAME PROVISIONER AGE

pmem-csi-sc-ext4 pmem-csi.intel.com 22h

For more information about installing pmem-csi, please refer to <https://github.com/intel/pmem-csi/blob/devel/docs/install.md#install-pmem-csi-driver>

## Run pmem-redis

Pmem-redis is one redis version that support Intel DCPMM(Data Center Persistent Memory) based on open source [redis-4.0.0](https://github.com/antirez/redis/tree/4.0). It benefits the redis's performance by taking advantage of DCPMM competitive performance and persistency. For more information about pmem-redis, check out <https://github.com/pmem/pmem-redis>

First build pmem-redis image and push it to local registry:

# git clone <https://github.com/clayding/opencloud_benchmark.git>

# cd opencloud\_benchmark/k8s/redis/docker

# sudo docker build -t pmem-redis:latest --network host .

# sudo docker tag pmem-redis:latest localhost:5000/pmem-redis:latest

# sudo docker push localhost:5000/pmem-redis:latest

In the same directory also ships with some sample yaml files to launch redis server with different mode:

volatile.yaml for redis using volatile memory, aof.yaml for redis enabling append-only file feature and pba.yaml for redis supporting Pointer Based Aof feature, only pba mode will use persistent memory.

Deploy a redis server with pba enabled with the following command:

# cd opencloud\_benchmark/k8s/redis

# kubectl create -f pba.yaml

You could check status of the pod by typing:

# kubectl get po

Check pods and volumes are created correctly:

# kubectl get po

NAME READY STATUS RESTARTS AGE

pmem-csi-controller-0 2/2 Running 0 10d

pmem-csi-node-zdsv8 2/2 Running 0 10d

redis-with-pba 1/1 Running 0 10s

# lsblk

NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT

sda 8:0 0 1.8T 0 disk

├─sda1 8:1 0 1.8T 0 part /

└─sda15 8:15 0 512M 0 part /boot/efi

pmem0.2 259:1 0 909G 0 disk

└─**ndbus0region0fsdax-7472642f--c583--11e9--bb62--0249f3840349**

**253:0 0 2G 0 lvm /var/lib/kubelet/pods/97d83ba0-a5c4-41b5-bdf7-c1ffc45e6d22/volumes/kubernetes.io~csi/pvc-398c2**

pmem1.1 259:0 0 909G 0 disk

To make sure redis works as expected, you could play around in the pod:

# kubectl exec -it redis-with-pba /bin/bash

# redis-cli

127.0.0.1:6379> set name test

OK

127.0.0.1:6379> get name

"test"

If you execute `kubectl describe` to one of pmem redis pods, you should see QoS class of the pod is set to Guaranteed, this is required because Kubernetes will only pin cpu on guaranteed pods.

## The benchmark part

Install memtier\_benchmark prerequisites :

# yum install autoconf automake make gcc-c++

# yum install pcre-devel zlib-devel libmemcached-devel

Remove system libevent and install new version:

# sudo yum remove libevent

# wget https://github.com/downloads/libevent/libevent/libevent-2.0.21-stable.tar.gz

# tar xfz libevent-2.0.21-stable.tar.gz

# pushd libevent-2.0.21-stable

# ./configure

# make

# sudo make install

# popd

# export PKG\_CONFIG\_PATH=/usr/local/lib/pkgconfig:${PKG\_CONFIG\_PATH}

Build and install memtier:

# git clone <https://github.com/RedisLabs/memtier_benchmark>

# cd memtier\_benchmark

# autoreconf -ivf

# ./configure --disable-tls

# make

# sudo make install

The format of memtier benchmark we use to evaluate redis performance:

memtier\_benchmark -s <IP of redis pod> -p 6379 -R -n 10000 -d 1024 --key-maximum=1000000 --ratio=1:0

```