User Guide For DCPMM Cache Support in OAP

Version: 0.0.1

Last Update: Dec 18th, 2019

Overview

Introduction to Intel Optane DC Persistent Memory

Intel® Optane™ DC persistent memory(DPCM) is an innovative memory technology that delivers a unique combination of affordable large capacity and support for data persistence.

This technology introduces a flexible new tier to the traditional data center memory and storage hierarchy, architected specifically for data center usage. By deploying systems enabled by this new class of memory, customers can now optimize their workloads more effectively by moving and maintaining larger amounts of data closer to the processor, thus minimizing the higher latency that can occur when fetching data from the system storage.

DCPM has two operating modes: App Direct and Memory Mode. In App Direct mode, applications can use persistent memory from user space without costly context switches to the OS by memory mapping files residing in a filesystem that supports Direct Access (DAX). DAX allows applications to address bytes in the storage media and access them directly from the CPU through load/store instructions at cache line granularity, instead of buffering large blocks in DRAM first, thus will benefit a lot to many IO-bound and IO-intensive workloads.

Cache Strategies

To take advantage of large capacity provided by DCPM, data source level cache is provided in OAP to accelerated Spark SQL workloads, which will cache uncompressed and decoded data into DCPM after read from data file for the first time. Thus afterward computation will retrieve data from cache directly instead of read from data file again.

In OAP, we provide additional cache strategies based on different libraries developed for DCPMM cache usage.
Vmemcache cache strategy is implemented based on libvmemcache (buffer based LRU cache), which provides data store API using <key, value> as input. Differently, Non-evictable cache strategy is based on memkind library which is built on top of jemalloc and provides memory characteristics.

Table 1 shows Spark versions supported by these 2 cache strategies.

	Vmemcache Cache Strategy	Non-evictable Cache Strategy
DataProc(Debian image v1.4)	2.4.4	N/A
OpenSource Spark	2.3.2, 2.4.4	2.3.2, 2.4.4
CDH	2.4.0	2.4.0

Table 1 Supported Spark Versions

Table 2 shows data format supported by these 2 cache strategies:

	Vmemcache Cache Strategy	Non-evictable Cache Strategy
Data format	Parquet, ORC	Parquet, ORC
Compression codec	Uncompress, snappy	Uncompress, snappy

Table 2 Supported data formats

Table 3 shows kernel versions that affect DCPM cache support:

Cache Strategy	Supported
Non Evictable Cache	4.18, 4.19
VMEMCache	4.18, 4.19, 5.0.11

Table 3 Supported/unsupported kernel version

Cache Properties:

Property Name	Default	Meaning
spark.sql.oap.parquet.enable	true	Whether enable oap file format when encountering parquet files
spark.sql.oap.orc.enable	true	Whether enable oap file format when encountering orc files
spark.yarn.numa.enabled	false	Whether enable numa-binding feature in cache support
spark.yarn.numa.num	2	Numa nodes provided in each slave, need spark.yarn.numa.enabled set true.
spark.sql.orc.copyBatchToSpark	false	Whether or not to copy the ORC columnar batch to Spark columnar batch in the vectorized ORC reader
spark.sql.oap.orc.data.cache.enable	false	To indicate if enable orc data cache. If want to use this feature, need spark.sql.oap.orc.enable to be true.
spark.sql.oap.parquet.data.cach	false	To indicate if enable parquet data cache. If want to

e.enable		use this feature, need spark.sql.oap.parquet.enable to be true.
spark.oap.cache.strategy	vmem	Defined cache strategy used to cache data source in DCPMM, there are two options: vmem and noevict.
spark.sql.oap.fiberCache.memory.manager	self	Sets the implement of memory manager, it currently supports self and hybrid (A combination of offheap and pm). For vmem cache strategy, this memory manager should be vmemcache. For noevict, it should be hybrid.
spark.memory.offHeap.size	0	The absolute amount of memory in bytes which can be used for off-heap allocation.This setting has no impact on heap memory usage, so if your executors' total memory consumption must fit within some hard limit then be sure to shrink your JVM heap size accordingly. This must be set properly and enough to enable OAP cache.
spark.sql.oap.fiberCache.persistent.memory.config.file	persistent-memory.xml	A config file used to config the Intel Optane DC persistent memory initial path and mapping with NUMA node. Default is the persistent-memory.xml under Spark Conf folder.
spark.sql.oap.fiberCache.persistent.memory.initial.size	0	Initial cache size provided by DCPM.
spark.sql.oap.fiberCache.persistent.memory.reserved.size	0	Used to set the reserved size of Intel Optane DC persistent memory. Because the heap management of Intel Optane DC persistent memory are based on jemalloc, so we can't make full use of the total initial size memory. The reserved size should smaller than initial size. Too small reserved size could result in OOM, too big size could reduce the memory utilization rate.sed to set the reserved size of Intel Optane DC persistent memory. Because the heap management of Intel Optane DC persistent memory are based on jemalloc, so we can't make full use of the total initial size memory. The reserved size should smaller than initial size. Too small reserved size could result in OOM, too big size could reduce the memory utilization rate. This property is dedicated for Non-evictable strategy.

Vmemcache Cache Strategy

Introduction

libvmemcache is an embeddable and lightweight in-memory caching solution. It's designed to fully take advantage of large capacity memory, such as Persistent Memory with DAX, through memory mapping in an efficient and scalable way. It provides eviction service using LRU algorithm when memory space is run out of.

In OAP(Optimized Analytics Package for Spark Platform), Vmemcache data source cache is supported based on libvememcache's key-value pattern. Refer following steps to apply vmemcache cache strategy in your workload.

Installation

1. Install libvmemcache on each node.
   Refer https://github.com/pmem/vmemcache.

2. Configure DCPMM properly in AD mode
   a. Change DCPMM into AD mode
      ipmctl create -goal PersistentMemoryType=AppDirect
   b. Reboot server
   c. List regions and create namespaces
      - ndctl list -R ##list all region
      - ndctl create-namespace -m fsdax -r region0 ##create namespace
      - ndctl create-namespace -m fsdax -r region1
   d. Format DCPMM devices and mount to file system
      - mkfs.ext4 /dev/pmem0 ##format DCPMM device
      - mkfs.ext4 /dev/pmem1 ##format DCPMM device
      - mkdir /mnt/pmem0
      - mkdir /mnt/pmem1
      - mount -o dax /dev/pmem0 /mnt/pmem0 ##mount DCPMM
      - mount -o dax /dev/pmem1 /mnt/pmem1
      Note: make sure spark user has access to these directories.

3. Configure properties to enable vmemcache support in OAP
   a. Enable OAP extension in spark-defaults.conf.
      spark.sql.extensions=org.apache.spark.sql.OapExtensions
      Also need to add OAP jar file in spark.executor.extraClassPath and spark.driver.extraClassPath

   b. For Parquet data format, provides following conf options:
      --conf spark.sql.oap.parquet.enable=true \
      --conf spark.sql.oap.parquet.data.cache.enable=true \
      --conf spark.sql.oap.fiberCache.memory.manager=self \
      --conf spark.oap.cache.strategy=vmem \

```
--conf spark.sql.oap.fiberCache.persistent.memory.initial.size=*g \
--conf spark.memory.offHeap.size=*g  \
--conf spark.sql.extensions=org.apache.spark.sql.OapExtensions \
--conf spark.sql.oap.fiberCache.persistent.memory.config.file=path/to/xml
```

c. For Orc data format, provides following conf options

```
--conf spark.sql.oap.orc.enable=true \
--conf spark.sql.orc.copyBatchToSpark=true \
--conf spark.sql.oap.orc.data.cache.enable=true \
--conf spark.sql.oap.fiberCache.memory.manager=self \
--conf spark.oap.cache.strategy=vmem \
--conf spark.sql.oap.fiberCache.persistent.memory.initial.size=*g \
--conf spark.memory.offHeap.size=*g  \
--conf spark.sql.extensions=org.apache.spark.sql.OapExtensions \
--conf spark.sql.oap.fiberCache.persistent.memory.config.file=path/to/xml
```

4. How to use Vmemcache in DataProc cluster

Vmemcache cache strategy is also supported in Google Cloud Dataproc, refer https://github.com/GoogleCloudPlatform/dataproc-initialization-actions/tree/master/intel-ap for detail usage.

Non-evictable Cache Strategy

Introduction

Another cache strategy named “Non-evictable” is also supported in OAP but based on a different library “memkind” for DCPMM. The memkind library is a user extensible heap manager built on top of jemalloc which enables control of memory characteristics and a partitioning of the heap between kinds of memory.

To apply Non-evictable cache strategy in your workload, refer following steps.

5. Install memkind on each node
   Refer https://github.com/memkind/memkind.

6. Configure DCPMM properly in AD mode, refer step 2.

7. Configure properties to enable Non-evictable cache support in OAP

   a. Enable OAP extension in spark-defaults.conf.
   spark.sql.extensions=org.apache.spark.sql.OapExtensions
   Also need to add OAP jar file in spark.executor.extraClassPath and spark.driver.extraClassPath

   b. For Parquet data format, provides following conf options:

```
--conf spark.sql.oap.parquet.data.cache.enable=true \
--conf spark.sql.oap.fiberCache.memory.manager=hybrid \
--conf spark.oap.cache.strategy=noevict \
--conf spark.sql.oap.fiberCache.persistent.memory.initial.size=*g \
```

```
--conf spark.sql.oap.fiberCache.persistent.memory.reserved.size=*g \
--conf spark.memory.offHeap.size=*g  \
--conf spark.sql.extensions=org.apache.spark.sql.OapExtensions \
--conf spark.sql.oap.fiberCache.persistent.memory.config.file=path/to/xml
```

c. For Orc data format, provides following conf options

```
--conf spark.sql.orc.copyBatchToSpark=true \
--conf spark.sql.oap.orc.data.cache.enable=true \
--conf spark.sql.oap.orc.enable=true \
--conf spark.sql.oap.fiberCache.memory.manager=hybrid \
--conf spark.oap.cache.strategy=noevict \
--conf spark.sql.oap.fiberCache.persistent.memory.initial.size=*g \
--conf spark.sql.oap.fiberCache.persistent.memory.reserved.size=*g \
--conf spark.memory.offHeap.size=*g  \
--conf spark.sql.extensions=org.apache.spark.sql.OapExtensions \
--conf spark.sql.oap.fiberCache.persistent.memory.config.file=path/to/xml
```

NUMA Support

NUMA is critical to the DCPMM performance. However, Spark doesn't officially support NUMA feature. In OAP, we provide three options for end users to enable NUMA when using DCPMM cache.
Supported Spark version

Option 1: For users who don't want to change Spark binary and there is no conflict in the relate Spark files, there is building option for user to build. How-To-Use steps as follows:

Compile:

```
cd $OAP_HOME/script
./apply_patch_to_spark.sh -v $SPAKR_VERSION
mvn clean package -Pvmemcache,numa-binding -DskipTests
```

Configuration Example:

Provide following conf options in spark-default.conf

```
spark.yarn.numa.enable = true
spark.yarn.numa.num = 2
```

Basically, each persistent memory device is associated with a numa id. Use ndctl -list -v to check numa info as below:

```
{
    "dev":"namespace1.0",
```

```
        "mode":"fsdax",
        "map":"dev",
        "size":532708065280,
        "uuid":"50548386-ba5a-422a-8289-3370158b7e38",
        "raw_uuid":"985e3e2f-de22-4221-b71b-622fa8bee661",
        "sector_size":512,
        "align":2097152,
        "blockdev":"pmem1",
        "numa_node":1
 },
 {
        "dev":"namespace0.0",
        "mode":"fsdax",
        "map":"dev",
        "size":532708065280,
        "uuid":"9626caf5-4a65-46f1-8526-66417e6e9d77",
        "raw_uuid":"00fd8fb9-6029-4363-a1ec-445b37c6e142",
        "sector_size":512,
        "align":2097152,
        "blockdev":"pmem0",
        "numa_node":0
 }
```


From the above ouput, we can see that pmem0 is associated with numa node 0 and pmem1 is associated with numa node 1. In previous Installation chapter, /dev/pmem0 mount at /mnt/pmem0 and /dev/pmem1 mount at /mnt/pmem1. So the configuration in persistent-memory.xml will provided as below:

```
        <!-- The numa number should be consistent with spark.yarn.numa.num -->
        <persistentMemoryPool>
                <!--The numa id-->
                <numanode id="0">
                        <!--The initial path for Intel Optane DC persistent memory-->
                        <initialPath>/mnt/pmem0/spark</initialPath>
                 </numanode>
                <numanode id="1">
                         <initialPath>/mnt/pmem1/spark</initialPath>
                </numanode>
        </persistentMemoryPool>
```

Option 2: For users who want to apply NUMA changes on their own, a NUMA patch is provided for end users to apply manually. How-To-Use steps as follows:

Compile:

```
cd $CUSTOMER_SPARK_HOME
git apply $OAP_HOME/patches/*
cp ${ALL_YOUR_PATCHED_FILE} $OAP_HOME/patched_file
mvn clean package -Pvmemcache,numa-binding -DskipTests
```

Configuration Example:

Same with Option1

Option 3: NUMA binding is critical for the performance. If option 1 and option 2 do not work, a NUMA binding option is provided as well. How-To-Use steps as follows:

Memory Usage

Figure 1 shows memory usage in OAP. As we can see, memory contains DRAM and DCPM. Spark manages executor memory and OAP manages two parts, one is total DCPM as cache memory and the other one is offheap memory, which is used as a buffer between spark and DCPM . User could set these configuration accordingly to increase or decrease different memory usage.

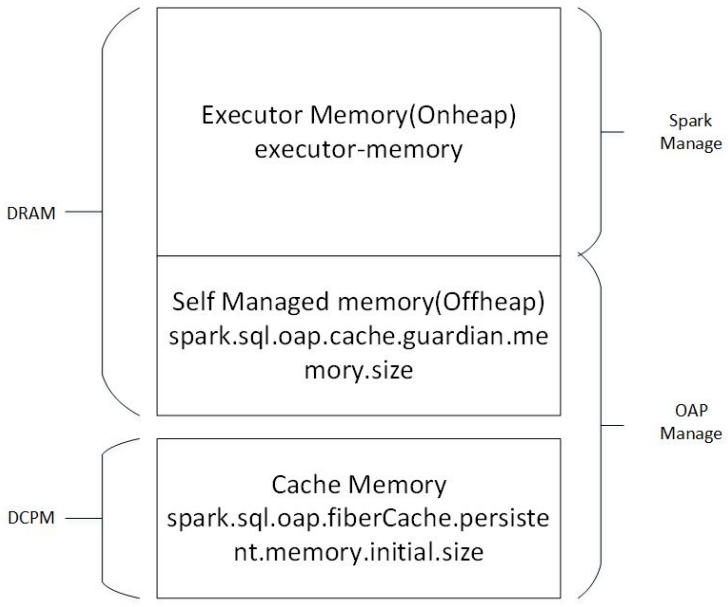

Figure1 Memory Usage in OAP

Troubleshooting

Issue 1: Failed task due to ‘cache guardian use too much memory’

If ‘PendingFiber Size’ (on spark web-UI OAP page) is large, or some tasks failed due to ‘*cache guardian use too much memory*’, user could set ‘spark.sql.oap.cache.guardian.memory.size’ config to a larger number, which default size is 10GB. Besides, user could increase ‘spark.sql.oap.cache.guardian.free.thread.nums’ or decrease ‘spark.sql.oap.cache.dispose.timeout.ms’ to accelerate memory free.

Issue 2: GC tuning

When GC time takes over 10% of task time, the column on Web UI will turn red, which means your workload may have GC problem.It’s especially for some memory intensive queries(like broadcast join query). We advice follow spark tuning guide, which you can refer at garbage-collection-tuning

Issue 3: Killed by node manager since virtual memory usage

When enable yarn.nodemanager.vmem-check-enabled settings to true, it’s strongly suggested to increase the yarn.nodemanager.vmem-pmem-ratio settings as DCPMM is accounted as virtual memory.

Issue 4: Spark executor exit due to Java Heap OutOfMemory

For some memory intensive queries, it’s suggested to adjust spark executor memory.

Issue 5: Failed to restart Spark executor due to Numa binding

Restarted executor may bind to a wrong numa node, and executor will failed to initial DCPM since another executor is occupying it. Please double check numa binding configuration.