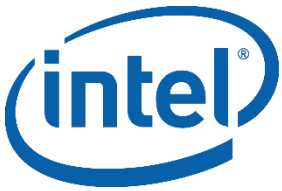


Spark-PMoF Enabling and Testing Guide

March. 2020

Revision 1.2

Contact: jian.zhang@intel.com

Revision History

Version History

Date	Version	Author	Updates	Comments
2019/8/12	1.0	Haodong Tang	Initial draft	
2020/3/3	1.1	Ma Eugene Jian Zhang	Updated with HW and test guide	
2020/3/20	1.2	Ma Eugene	Updated with RDMA configuration	

1. Spark-PMoF introduction.

Intel Optane™ DC persistent memory is the next-generation storage at memory speed. It fills the large performance gap between DRAM memory technology and the highest performance block device in the form of solid-state drives. Remote Persistent Memory extends PM usage to new scenario, lots of new usage cases & value proposition can be developed.

Spark-PMoF (https://github.com/Intel-bigdata/Spark-PMoF) is the Persistent Memory over Fabrics (PMoF) plugin for Spark shuffle, which leverages the RDMA network and remote persistent memory (for read) to provide extremely high performance and low latency shuffle solutions for Spark. The Figure 1 shows how data flows between Spark and shuffle devices in Spark-PMoF and Vanilla Spark. In this guide, we will introduce how to deploy and use Spark-PMoF plugin.

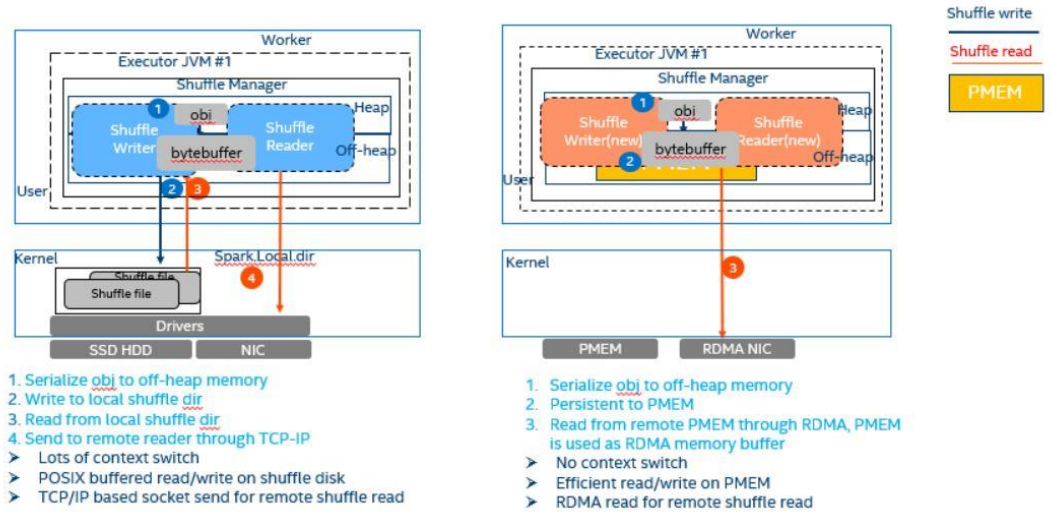


Figure 1: Spark-PMoF design

2. Recommended HW environment

2.1. System Configuration

2.1.1 HW and SW Configuration

A 4 Node cluster is recommended for a POC tests.

Hardware:

- Intel® Xeon™ processor Gold 6240 CPU @ 2.60GHz, 384GB Memory (12x 32GB 2666 MT/s) or 192GB Memory (12x 16GB 2666MT/s)
- An RDMA capable NIC, 40Gb+ is preferred. e.g., 1x Intel X722 NIC or Mellanox ConnectX-4 40Gb NIC
- Shuffle Devices：
  - 1x 1TB HDD for shuffle (baseline)
  - 4x 128GB Persistent Memory for shuffle
- 4x 1T NVMe for HDFS

Software:

- Hadoop 2.7
- Spark 2.3
- Fedora 29 with **ww08.2019** BKC

2.1.2 System Diagram

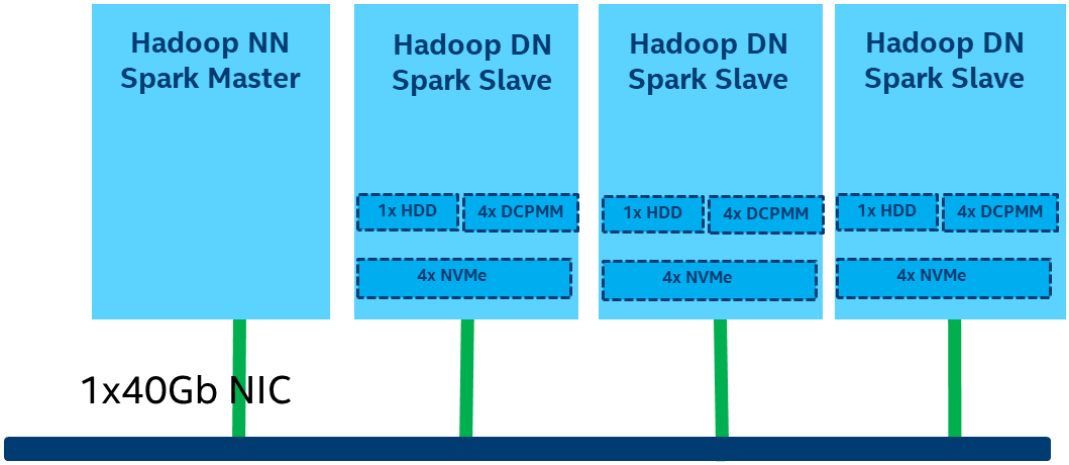
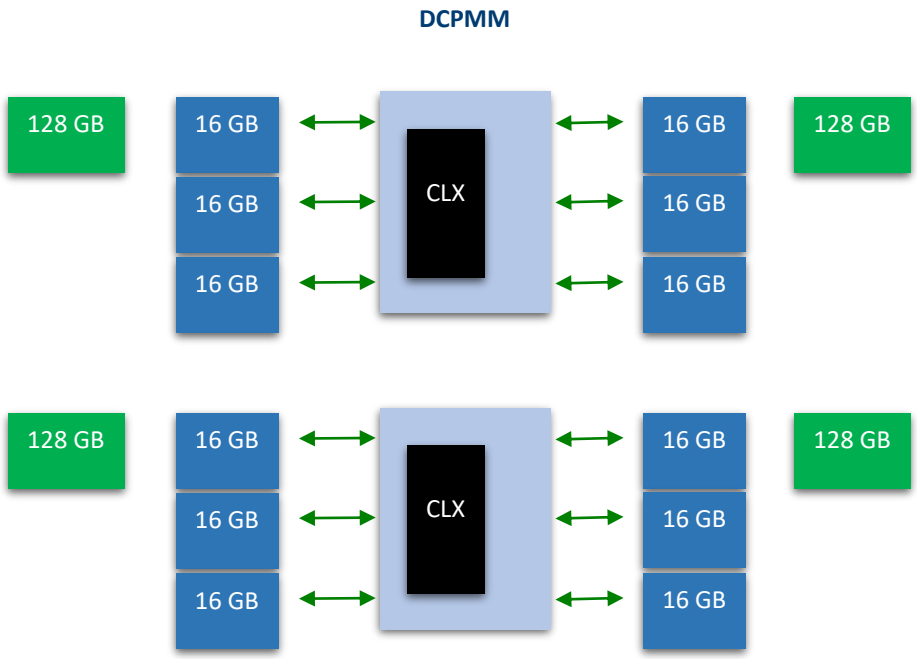


Figure 2: System Diagram

2.2. Recommended RDMA NIC

Spark PMoF is using HPNL (https://cloud.google.com/solutions/big-data/) for network communication, which leverages libfabric for efficient network communication, so a RDMA capable NIC is recommended. Libfabric supports RoCE, iWrap, IB protocol, so various RNICs with different protocol can be used.

2.3 recommended DCPMM configuration



It is recommended to install 4+ DCPMM DIMMs on the SUT, but you can adjust the numbers accordingly. In this enabling guide, 4x 128GB DCPMM was installed on the SUT.

2.4 recommended DCPM BKC

The preferred version of BKC (best known configuration) is **ww08.2019**. Please refer to DCPMM snapshot for more details.

Please refer to backup if you do not have BKC access. BKC installation/enabling without BKC is out of the scope of this guide.

3. Install and configure DCPM

1) Please install *ipmctl* and *ndctl* according to your OS version
2) Run ipmctl show -dimm to check whether dimms can be recognized
3) Run ipmctl create -goal PersistentMemoryType=AppDirect to create AD mode
4) Run ndctl list -R , you will see **region0** and **region1** in screen
5) Suppose we have 4x DCPM on two sockets.
   a) Run ndctl create-namespace –m devdax -r region0 -s 120g
   b) Run ndctl create-namespace –m devdax –r region0 –s 120g
   c) Run ndctl create-namespace –m devdax –r region1 –s 120g
   d) Run ndctl create-namespace –m devdax –r region1 –s 120g
   e) Then we will see /dev/dax0.0, /dev/dax0.1, /dev/dax1.0, /dev/dax1.1

4. Configure and Validate RDMA

4.1 Configure and test iWARP RDMA

4.1.1 Download rdma-core and install dependencies

The rdma-core provides the necessary userspace libraries to test rdma connectivity with tests such as rping. Refer to latest rdma-core documentation for updated installation guidelines (https://github.com/linux-rdma/rdma-core.git). You might refer to vendor specific

instructions or guide to enable your RDMA NICs. Take Mellanox as an example, perform below steps to enable it:

a, Download and install drivers per guide here

b, Enable PFC (Priority Flow Control) to guarantee stable performance.

/etc/init.d/openibd restart

mlnx_qos -i -ens803f1 -pfc 0,0,0,1,0,0,0,0

modprobe 8021q

vconfig add ens803f1 100

ifconfig ens803f1.100 172.168.0.40/16 up

ifconfig ens803f1 172.167.0.40/16 up

for i in {0..7}; do vconfig set_egress_map ens803f1.100 $i 3 ; done

tc_wrap.py -i ens803f1 -u 3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3

Modify highlight part based on your environment and execute the script.

4.1.2 Load RDMA module

Make sure the following modules are loaded: modprobe ib_core i40iw iw_cm rdma_cm rdma_ucm ib_cm ib_uverbs

4.1.3 Check RDMA

Check that you see your RDMA interfaces listed on each server when you run the following command: ibv_devices

Check with rping for RDMA connectivity between target interface and client interface.

1) Assign IPs to the RDMA interfaces on Target and Client.

2) On Target run:rping -sdVa <Target IP>

3) On Client run: rping -cdVa <Target IP>

4.2 Configure and test Mellanox RoCE V2 RDMA

Please refer to the document from Mellanox. Here is the detailed configuration. (https://community.mellanox.com/s/article/howto-enable--verify-and-troubleshoot-rdma).

5. Install Spark-PMoF

5.1 Install HPNL (https://github.com/Intel-bigdata/HPNL)

HPNL is a fast, CPU-Efficient network library designed for modern network technology. HPNL depends on Libfabric, which is protocol independent, it supports TCP/IP, RoCE, IB, iWRAP etc. Please make sure the Libfabric is installed in your setup. Based on this issue, make sure NOT to install Libfabric 1.9.0.

5.1.1 Install the dependency

Spark PMoF leverages HPNL for network communication. It leverages libpmemobj to access the persistent media. PMDK, a library to manage and access persistent memory devices is required to be installed.

1) sudo apt-get install cmake libboost-dev libboost-system-dev

5.1.2 Build and install HPNL

1) git clone https://github.com/Intel-bigdata/HPNL.git

2) cd HPNL

3) git checkout spark-pmof-test

4) git submodule update --init --recursive

5) mkdir build; cd build;

6) cmake -DWITH_VERBS=ON -DWITH_JAVA=ON ..

7) make && make install

8) cd ${project_root_path}/java/hpnl; mvn package

5.2 Install Spark-PMoF

1) git clone https://github.com/Intel-bigdata/Spark-PMoF.git

2) cd Spark-PMoF; mvn install

5.3 Configure Spark-PMoF in Spark

Spark-PMoF is designed as a plugin to Spark. Currently the plugin supports Spark 2.3 and works well on various Network fabrics, including Socket, RDMA and Omni-Path. There are several configurations files needs to be modified in order to run Spark PMoF.

modify spark-defaults.conf

Before running Spark workload, add following contents in spark-defaults.conf.

spark.driver.extraClassPath Spark-PMoF-PATH/target/Spark-PMoF-1.0-jar-with-dependencies.jar

spark.executor.extraClassPath Spark-PMoF-PATH/target/Spark-PMoF-1.0-jar-with-dependencies.jar

spark.shuffle.manager org.apache.spark.shuffle.pmof.RdmaShuffleManager

spark.shuffle.pmof.enable_rdma true

spark.shuffle.pmof.enable_pmem true

spark.shuffle.pmof.max_stage_num 1

spark.shuffle.pmof.max_task_num 50000

spark.shuffle.spill.pmof.MemoryThreshold 16777216

spark.shuffle.pmof.pmem_capacity 507340914688

spark.shuffle.pmof.pmem_list /dev/dax0.0,/dev/dax1.0

spark.shuffle.pmof.dev_core_set dax0:0-71,dax0:0-71,dax1:0-71,dax1:0-71,dax0:0-71,dax0:0-71

spark.shuffle.pmof.server_buffer_nums 64

spark.shuffle.pmof.client_buffer_nums 64

spark.shuffle.pmof.map_serializer_buffer_size 262144

spark.shuffle.pmof.reduce_serializer_buffer_size 262144

spark.shuffle.pmof.chunk_size 262144

spark.shuffle.pmof.server_pool_size 3

spark.shuffle.pmof.client_pool_size 3

spark.shuffle.pmof.shuffle_block_size 2097152

spark.shuffle.pmof.node sr140-172.168.0.40,sr141-172.168.0.41,sr142-172.168.0.42

spark.driver.rhost 172.168.0.43

spark.driver.rport 61000

6. Spark PMoF Testing

Spark PMoF have been tested with TPC-DS and Terasort.

6.1 TPC-DS

The TPC-DS is a decision support benchmark that models several general applicable aspects of a decision support system, including queries and data maintenance.

6.1.1 Download spark-sql-perf

The link is https://github.com/databricks/spark-sql-perf and follow README to use sbt build the artifact

6.1.2 Download tpcds-kit

As per instruction from spark-sql-perf README, tpcds-kit is required and please download it from https://github.com/databricks/tpcds-kit, follow README to setup the benchmark

6.1.3 Prepare data

As an example, generate parquet format data to HDFS with 1TB data scale. The data stored path, data format and data scale are configurable. Please check script below as a sample.

```
import com.databricks.spark.sql.perf.tpcds.TPCDSTables
import org.apache.spark.sql._

// Set:
val rootDir: String = "hdfs://sr143:9000/tpcds_1T" // root directory of location to create data in.
val databaseName: String = "tpcds_1T" // name of database to create.
val scaleFactor: String = "1024" // scaleFactor defines the size of the dataset to generate (in GB).
```

```
val format: String = "parquet" // valid spark format like parquet "parquet".
val sqlContext = new SQLContext(sc)
// Run:
val tables = new TPCDSTables(sqlContext,
   dsdgenDir = "/mnt/spark-pmof/tool/tpcds-kit/tools", // location of dsdgen
   scaleFactor = scaleFactor,
   useDoubleForDecimal = false, // true to replace DecimalType with DoubleType
   useStringForDate = false) // true to replace DateType with StringType

tables.genData(
   location = rootDir,
   format = format,
   overwrite = true, // overwrite the data that is already there
   partitionTables = true, // create the partitioned fact tables
   clusterByPartitionColumns = true, // shuffle to get partitions coalesced into single files.
   filterOutNullPartitionValues = false, // true to filter out the partition with NULL key value
   tableFilter = "", // "" means generate all tables
   numPartitions = 400) // how many dsdgen partitions to run - number of input tasks.

// Create the specified database
sql(s"create database $databaseName")
// Create metastore tables in a specified database for your data.
// Once tables are created, the current database will be switched to the specified database.
tables.createExternalTables(rootDir, "parquet", databaseName, overwrite = true,
discoverPartitions = true)
```

6.1.4 Run the benchmark

Launch TPC-DS queries on generated data, check *benchmark.scale* below as a sample, it runs query64.

```
import com.databricks.spark.sql.perf.tpcds.TPCDS
import org.apache.spark.sql._

val sqlContext = new SQLContext(sc)
val tpcds = new TPCDS (sqlContext = sqlContext)
// Set:
val databaseName = "tpcds_1T" // name of database with TPCDS data.
val resultLocation = "tpcds_1T_result" // place to write results

val iterations = 1 // how many iterations of queries to run.
val query_filter = Seq("q64-v2.4")
val randomizeQueries = false

def queries = {
  val filtered_queries = query_filter match {
    case Seq() => tpcds.tpcds2_4Queries
    case _ => tpcds.tpcds2_4Queries.filter(q => query_filter.contains(q.name))
  }
  filtered_queries
}
```

```
val timeout = 24*60*60 // timeout, in seconds.
// Run:
sql(s"use $databaseName")
val experiment = tpcds.runExperiment(
  queries,
  iterations = iterations,
  resultLocation = resultLocation,
  forkThread = true)

experiment.waitForFinish(timeout)
```

6.1.5 Check the result

Check the result under *tpcds_1T_result* folder. It can be an option to check the result at spark history server. (Need to start history server by *$SPARK_HOME/sbin/start-history-server.sh*)

6.2 TeraSort

TeraSort is a benchmark that measures the amount of time to sort one terabyte of randomly distributed data on a given computer system.

6.2.1 Download HiBench

The link is https://github.com/Intel-bigdata/HiBench. The HiBench is a big data benchmark suite and contains a set of Hadoop, Spark and streaming workloads including TeraSort.

6.2.2 Build HiBench as per instructions from build-bench.

6.2.3 Configuration

Modify *$HiBench-HOME/conf/spark.conf* to specify the spark home and other spark configurations. It will overwrite the configuration of *$SPARK-HOME/conf/spark-defaults.conf* at run time

6.2.4 Launch the benchmark

Change directory to *$HiBench-HOME/bin/workloads/micro/terasort/spark* and launch the run.sh. You can add some PMEM cleaning work to make sure it starts from empty shuffle device every test iteration. Take *run.sh* below as a sample.

```
ssh sr140 pmempool rm /dev/dax0.0
ssh sr140 pmempool rm /dev/dax0.1
ssh sr140 pmempool rm /dev/dax1.0
ssh sr140 pmempool rm /dev/dax1.1

ssh sr141 pmempool rm /dev/dax0.0
ssh sr141 pmempool rm /dev/dax0.1
ssh sr141 pmempool rm /dev/dax1.0
ssh sr141 pmempool rm /dev/dax1.1

ssh sr142 pmempool rm /dev/dax0.0
ssh sr142 pmempool rm /dev/dax0.1
ssh sr142 pmempool rm /dev/dax1.0
ssh sr142 pmempool rm /dev/dax1.1
```

```
current_dir=`dirname "$0"`
current_dir=`cd "$current_dir"; pwd`
root_dir=${current_dir}/../../../../..
workload_config=${root_dir}/conf/workloads/micro/terasort.conf
. "${root_dir}/bin/functions/load_bench_config.sh"

enter_bench ScalaSparkTerasort ${workload_config} ${current_dir}
show_bannar start

rmr_hdfs $OUTPUT_HDFS || true

SIZE=`dir_size $INPUT_HDFS`
START_TIME=`timestamp`
run_spark_job com.intel.hibench.sparkbench.micro.ScalaTeraSort $INPUT_HDFS
$OUTPUT_HDFS
END_TIME=`timestamp`

gen_report ${START_TIME} ${END_TIME} ${SIZE}
show_bannar finish
leave_bench
```

6.2.4 Check the result

Check the result at spark history server to see the execution time and other spark metrics like spark shuffle spill status. (Need to start history server by *$SPARK_HOME/sbin/start-history-server.sh*)

Backup:

Recommended OS (without BKC access)

If you do not have BKC access, please following below official guide: (this is official DCPMM guide, it is a pre-request for PMoF deployment)

(1): General DCPMM support: DCPMM support

https://www.intel.com/content/www/us/en/support/products/190349/memory-and-storage/data-center-persistent-memory/intel-optane-dc-persistent-memory.html

(2) DCPMM population rule: Module DIMM Population for Intel® Optane™ DC Persistent Memory

https://www.intel.com/content/www/us/en/support/articles/000032932/memory-and-storage/data-center-persistent-memory.html?productId=190349&localeCode=us_en

(3) OS support requirement: Operating System OS for Intel® Optane™ DC Persistent Memory

https://www.intel.com/content/www/us/en/support/articles/000032860/memory-and-storage/data-center-persistent-memory.html?productId=190349&localeCode=us_en

Operating System Support

OS Version	Memory Mode	App Direct Mode	Dual Mode
RHEL* 7.5	Yes		
Ubuntu* 16.04 LTS	Yes		
Windows* Server 2016	Yes		
Oracle* Linux* 7.6 with UEK R5 Update 2	Yes	Yes	
VMware* vSphere 6.7 EP10	Yes	Yes	
CentOS* 7.6 or later	Yes	Yes	Yes
RHEL 7.6 or later	Yes	Yes	Yes
SLES* 12 SP4 or later	Yes	Yes	Yes
SLES 15 or later	Yes	Yes	Yes
Ubuntu 18.04 LTS	Yes	Yes	Yes
Ubuntu 18.10 or later	Yes	Yes	Yes
VMWare* ESXi 6.7 U1 or later	Yes	Yes	Yes
Windows 10 Pro for Workstation Version 1809 or later	Yes	Yes	Yes
Windows Server 2019 or later	Yes	Yes	Yes

(4): Quick Start Guide: Provision Intel® Optane™ DC Persistent Memory

https://software.intel.com/en-us/articles/quick-start-guide-configure-intel-optane-dc-persistent-memory-on-linux