RPMem extension for Spark Shuffle Enabling and Testing Guide

June 2020

Revision 1.4

Revision History

Version History

Date	Version	Updates
2019/8/12	1.0	Initial draft
2020/3/3	1.1	Updated with HW and test guide
2020/3/20	1.2	Updated with RDMA configuration
2020/5/27	1.3	Rename Spark PMoF to RPMem extension for spark shuffle
2020/6/13	1.4	Updated Enabling guide

Notes

Please be noted the this project will be migrated to OAP https://github.com/intel-bigdata/oap, the enabling guide might be not up to date.

RPMem extension for Spark Shuffle (AKA. Spark PMoF) depends on multiple native libraries like libfabrics, libcuckoo, PMDK. This enabling guide covers the installing process for the time being, but it might change as the install commands and related dependency packages for the 3rd party libraries might vary depends on the OS version and distribution you are using.

The benchmark is for reference only.

1. RPMem extension for spark shuffle introduction.

Intel Optane^{TM} 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.

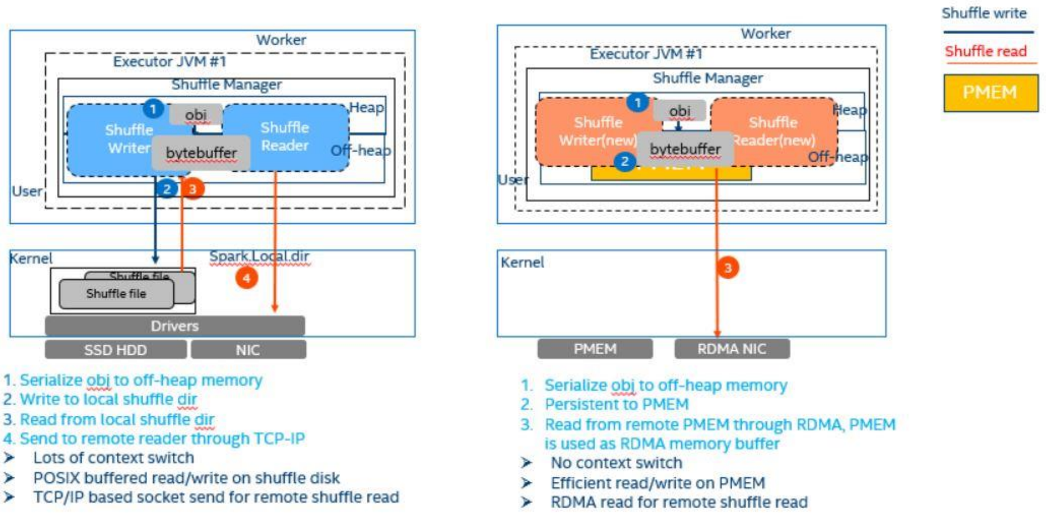
RPMem extension for spark shuffle (previously Spark PMoF) https://github.com/Intel-bigdata/Spark-PMoF) is the Persistent Memory over Fabrics (PMoF) plugin for Spark shuffle, which leverages the RDMA network and remote persistent memory (for read) to provide extremely high performance and low latency shuffle solutions for Spark.

Spark shuffle is a high cost operation as it issues a great number of small random disk IO, serialization, network data transmission, and thus contributes a lot to job latency and could be the bottleneck for workloads performance.

Spark PMoF Extension brings follow benefits:

- Leverage high performance persistent memory as shuffle media as well as spill media, increased shuffle performance and reduced memory footprint
- Using PMDK libs to avoid inefficient context switches and memory copies with zero-copy remote access to persistent memory.
- Leveraging RDMA for network offloading

The Figure 1 shows how data flows between Spark and shuffle devices in RPMem extension for spark shuffle and Vanilla Spark. In this guide, we will introduce how to deploy and use RPMem extension for spark shuffle plugin.



1. Serialize obj to off-heap memory
2. Write to local shuffle dir
3. Read from local shuffle dir
4. Send to remote reader through TCP-IP

➢ Lots of context switch
➢ POSIX buffered read/write on shuffle disk
➢ TCP/IP based socket send for remote shuffle read

1. Serialize obj to off-heap memory
2. Persistent to PMEM
3. Read from remote PMEM through RDMA, PMEM is used as RDMA memory buffer

➢ No context switch
➢ Efficient read/write on PMEM
➢ RDMA read for remote shuffle read

Figure 1: RPMem extension for spark shuffle design

2. Recommended HW environment

2.1. System Configuration

2.1.1 HW and SW Configuration

A 4x or 3x Node cluster is recommended for a POC tests, depending your system configurations, if using 3 nodes cluster, the Name node and Spark Master node can be co-located with one of the Hadoop data nodes.

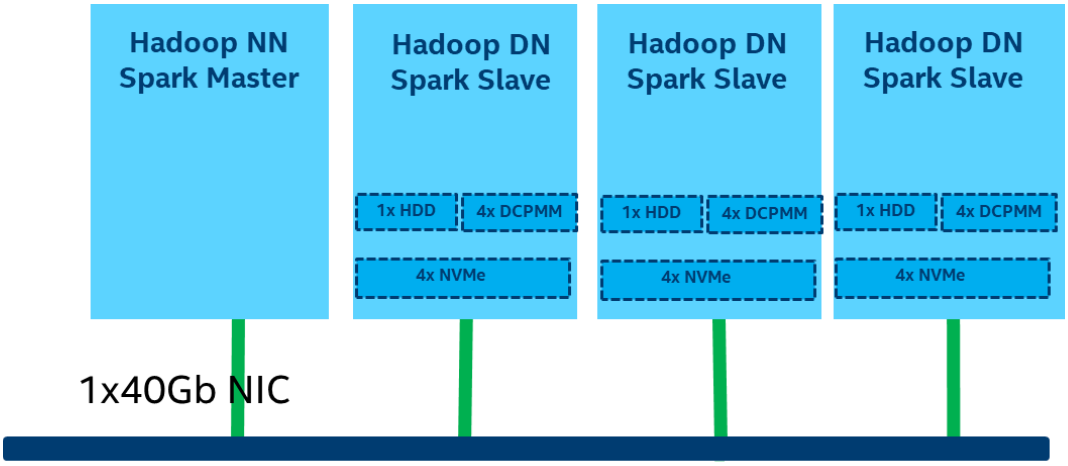
Hardware:

- 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
    - RDMA cables:
    - Mellanox MCP1600-C003 100GbE 3m 28AWG
- Shuffle Devices：
    - 1x 1TB HDD for shuffle (baseline)
    - 4x 128GB Persistent Memory for shuffle
- 4x 1T NVMe for HDFS
- **Switch**:
    - Arista 7060 CX2 (7060CX2-32S-F) 100Gb switches was used, but the port was configured to 40Gb for the Mellanox NICs
- Please refer to section 4.2 for configurations

Software:

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

2.1.2 System Diagram



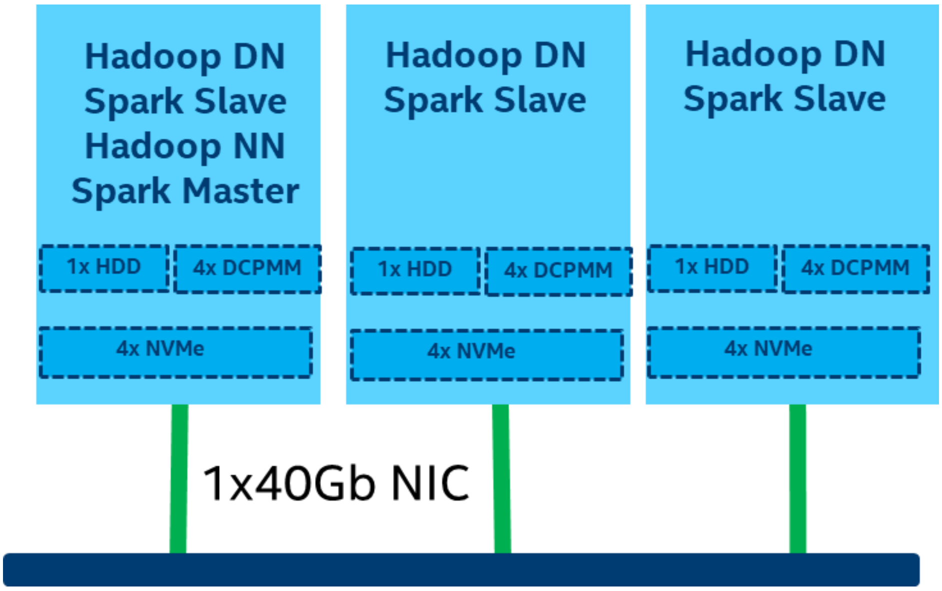
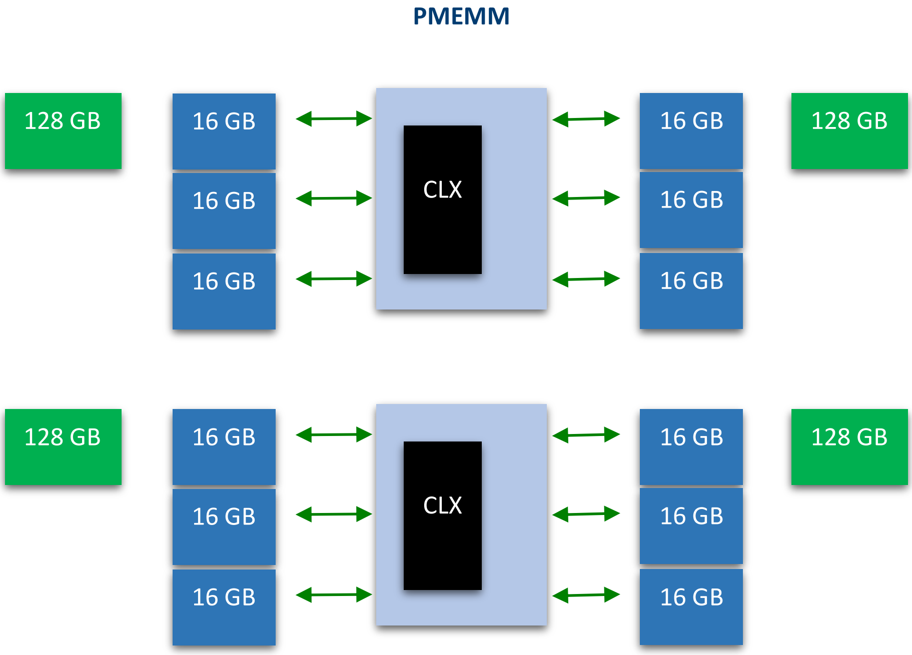


Figure 2: 4-node and 3-node System Diagram

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 PMEMM configuration



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

2.4 recommended PMEM BKC

The preferred version of BKC (best known configuration) is ww08.2019. Please refer to PMEMM 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 PMEM

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 PMEM 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:

```
$    git clone https://github.com/linux-rdma/rdma-core.git
$    dnf install cmake gcc libnl3-devel libudev-devel pkgconfig valgrind-devel ninja-build python3-devel python3-Cython python3-docutils pandoc
$    //change to yum on centos
$    bash build.sh
$    on centos 7
$    yum install cmake gcc libnl3-devel libudev-devel make pkgconfig valgrind-devel
$    $ yum install epel-release
$    $ yum install cmake3 ninja-build pandoc
$
```

4.1.2 Switch Configuration

This part is vendor specific, please check your switch menu accordingly.

Below example is Arista 7060 CX2 100Gb Switch, it is to configure the 100Gb port to work at 40Gb.

```
# Connect the console port to PC. Username is admin. No password. Enter global configuration mode.
# Config Switch Speed to 40Gb/s
switch# enable
switch# config
switch(config)# show interface status
Configure corresponding port to 40 Gb/s to match the NIC speed.
switch(config)# interface Et(num_of_port)/1
switch(config)# speed forced 40gfull
```

RoCE might have performance issues, so PFC configuration is strongly suggested. You will need to check the RDMA NIC driver manual and switch manual to configure PFC. Below is the example for ConnectX-4 and Arista 7060-CX2 switches.

Below is to set the two connection ports in the same vlan and **co**nfigure it in trunk mode.

```
# Configure interface as trunk mode and add to vlan
switch(config)# vlan 1
switch(config-vlan-1)#
switch(config)# interface ethernet 12-16
switch(config-if-Et12-16)# switchport trunk allowed vlan 1
switch (config-if-et1) # priority-flow-control on
switch (config-if-et1) # priority-flow-control priority 3 no-drop
```

4.1.3 Download and install drivers per guide

A. (Optional) Mellanox Enabling RoCE V2 RDMA

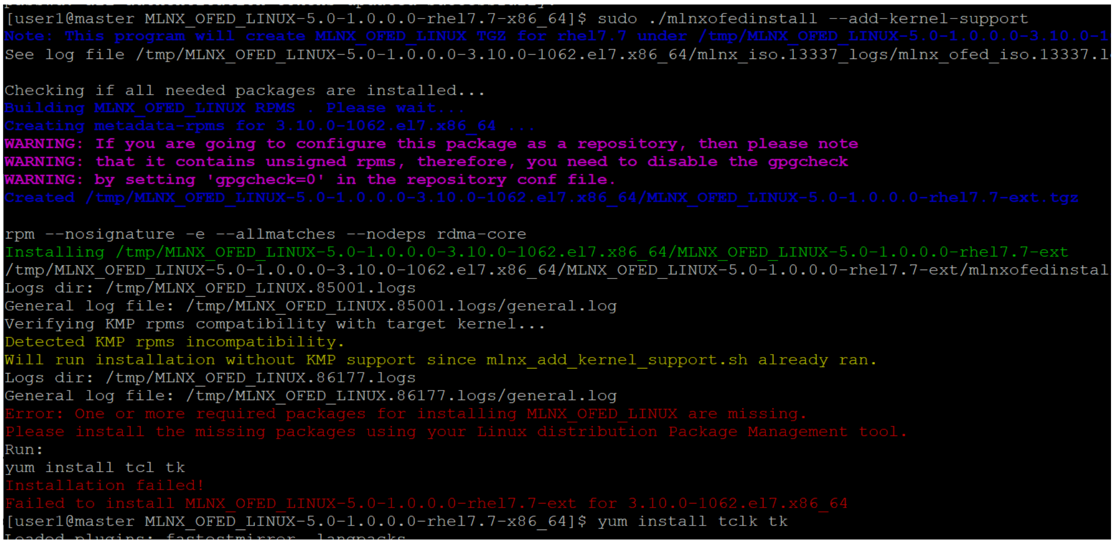
Below Firmware is required when you are using Mellanox RoCE V2 RDMA

There are lots of packages needs to be installed for dependency:

yum install atk gcc-gfortran tcsh gtk2 tcl tk

please install accordingly.

```
# Download MLNX_OFED_LINUX-4.7-3.2.9.0-* from https://community.mellanox.com/s/article/howto-install-mlnx-ofed-driver
# e.g., wget http://www.mellanox.com/downloads/ofed/MLNX_OFED-<version>/MLNX_OFED_LINUX-<version>-<distribution>-<arch>.tgz .
# (inside lab mirror) root@vsr140:/mnt/spark-pmof/tool
$      tar zxf MLNX_OFED_LINUX-4.7-3.2.9.0-*
$      cd MLNX_OFED_LINUX-4.7-3.2.9.0-
$      ./mlnxofedinstall --add-kernel-support.
# The process might interpret and promote you to install dependencie
s and try again
# This process will take some time.
```



```
[user1@master MLNX_OFED_LINUX-5.0-1.0.0.0-rhel7.7-x86_64]$ sudo ./mlnxofedinstall --add-kernel-support
Note: This program will create MLNX_OFED_LINUX TGZ for rhel7.7 under /tmp/MLNX_OFED_LINUX-5.0-1.0.0.0-3.10.0-1
See log file /tmp/MLNX_OFED_LINUX-5.0-1.0.0.0-3.10.0-1062.el7.x86_64/mlnx_iso.13337_logs/mlnx_ofed_iso.13337.l

Checking if all needed packages are installed...
Building MLNX_OFED_LINUX RPMS . Please wait...
Creating metadata-rpms for 3.10.0-1062.el7.x86_64 ...
WARNING: If you are going to configure this package as a repository, then please note
WARNING: that it contains unsigned rpms, therefore, you need to disable the gpgcheck
WARNING: by setting 'gpgcheck=0' in the repository conf file.
Created /tmp/MLNX_OFED_LINUX-5.0-1.0.0.0-3.10.0-1062.el7.x86_64/MLNX_OFED_LINUX-5.0-1.0.0.0-rhel7.7-ext.tgz

rpm --nosignature -e --allmatches --nodeps rdma-core
Installing /tmp/MLNX_OFED_LINUX-5.0-1.0.0.0-3.10.0-1062.el7.x86_64/MLNX_OFED_LINUX-5.0-1.0.0.0-rhel7.7-ext
/tmp/MLNX_OFED_LINUX-5.0-1.0.0.0-3.10.0-1062.el7.x86_64/MLNX_OFED_LINUX-5.0-1.0.0.0-rhel7.7-ext/mlnxofedinstal
Logs dir: /tmp/MLNX_OFED_LINUX.85001.logs
General log file: /tmp/MLNX_OFED_LINUX.85001.logs/general.log
Verifying KMP rpms compatibility with target kernel...
Detected KMP rpms incompatibility.
Will run installation without KMP support since mlnx_add_kernel_support.sh already ran.
Logs dir: /tmp/MLNX_OFED_LINUX.86177.logs
General log file: /tmp/MLNX_OFED_LINUX.86177.logs/general.log
Error: One or more required packages for installing MLNX_OFED_LINUX are missing.
Please install the missing packages using your Linux distribution Package Management tool.
Run:
yum install tcl tk
Installation failed!
Failed to install MLNX_OFED_LINUX-5.0-1.0.0.0-rhel7.7-ext for 3.10.0-1062.el7.x86_64
[user1@master MLNX_OFED_LINUX-5.0-1.0.0.0-rhel7.7-x86_64]$ yum install tclk tk
```

Restart the driver:

```
$      # /etc/init.d/openibd restart
$      Might need to unload the modules if it is in use.
```

Make sure the that the field link_layer is “Ethernet” with below commend:

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

Then you can use following command to gett he device name

If you’re using Mellanox NIC, PFC is a must to guarantee stable performance.

Fetch RDMA info with rdma command:

```
$      $ rdma link
0/1: i40iw0/1: state DOWN physical_state NOP
1/1: i40iw1/1: state ACTIVE physical_state NOP
2/1: mlx5_0/1: state DOWN physical_state DISABLED netdev ens803f0
3/1: mlx5_1/1: state ACTIVE physical_state LINK_UP netdev ens803f1
```

```
$      $ lspci | grep Mellanox
86:00.0 Ethernet controller: Mellanox Technologies MT27700 Family [ConnectX-4]
```

```
86:00.1 Ethernet controller: Mellanox Technologies MT27700 Family [ConnectX-4]
```

```
$   /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 ens803f1.100 172.168.0.209/16 up
$   ifconfig ens803f1 172.167.0.209/16 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 the IP address part based on your environment and execute the script.

4.1.4 Check RDMA module

Make sure the following modules are loaded:

```
$     modprobe ib_core i40iw iw_cm rdma_cm rdma_ucm ib_cm ib_uverbs
```

4.1.5 Validate RDMA works

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>

Example:

On the service side:

```
$     rping -sda 172.168.0.209
created cm_id 0x17766d0
rdma_bind_addr successful
rdma_listen
…
accepting client connection request
cq_thread started.
recv completion
Received rkey 97a4f addr 17ce190 len 64 from peer
cma_event type RDMA_CM_EVENT_ESTABLISHED cma_id 0x7fe9ec000c90 (child)
```

```
ESTABLISHED
…
Received rkey 96b40 addr 17ce1e0 len 64 from peer
server received sink adv
rdma write from lkey 143c0 laddr 1771190 len 64
rdma write completion
…
```

On Client run: rping -cdVa <Target IP>

```
$     # Client side use .100 ip 172.168.0.209
$     rping -c -a 172.168.0.209 -v -C 4
ping data: rdma-ping-0: ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqr
ping data: rdma-ping-1: BCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrs
ping data: rdma-ping-2: CDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrst
ping data: rdma-ping-3: DEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstu
```

Notes:

Detail official guide:

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 RPMem extension for spark shuffle

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.

You might need to install automake/libtool first to resolve dependency issues.

```
$     git clone https://github.com/ofiwg/libfabric.git
$     git checkout v1.6.0
$     ./autogen.sh
$     ./configure --disable-sockets --enable-verbs --disable-mlx
$     make -j && sudo make install
```

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.

5.1.2 Build and install HPNL

Project_root_path is HPNL folder’s path.

1) sudo apt-get install cmake libboost-dev libboost-system-dev
2) ?? remove with_java.

```
Ubuntu
$     sudo apt-get install cmake libboost-dev libboost-system-dev
# Fedora
$     dnf install cmake boost-devel boost-system
$     git clone https://github.com/Intel-bigdata/HPNL.git
$     cd HPNL
$     git checkout origin/spark-pmof-test --track
$     git submodule update --init --recursive
$     mkdir build; cd build
$     cmake -DWITH_VERBS=ON  -DWITH_JAVA=ON ..
$     make -j && make install
$     cd ${project_root_path}/java/hpnl
$     mvn install
```

5.2 Install RPMem extension for spark shuffle

5.2.1 install dependencies

```
$   yum install -y autoconf asciidoctor kmod-devel.x86_64 libudev-devel libuuid-devel json-
    c-devel jemalloc-devel
$   yum groupinstall -y "Development Tools"
```

5.2.2 install ndctl

```
$   git clone https://github.com/pmem/ndctl.git
$   cd ndctl
$   git checkout v63
$   ./autogen.sh
$   ./configure CFLAGS='-g -O2' --prefix=/usr --sysconfdir=/etc --libdir=/usr/lib64
$   make -j
$   make check
$   make install
$   cd ../java/hnpl
$   mvn install
```

5.2.3 install PMDK

```
$     yum install -y pandoc
$    git clone https://github.com/pmem/pmdk.git
$    cd pmdk
$    git checkout tags/1.8
$    make -j && make install
$    export PKG_CONFIG_PATH=/usr/local/lib64/pkgconfig/:$PKG_CONFIG_PATH
$    echo “export PKG_CONFIG_PATH=/usr/local/lib64/pkgconfig/:$PKG_CONFIG_PATH” >
     /etc/profile.d/pmdk.sh
```

5.2.4 Install RPMem extension for spark shuffle

```
$  git clone  https://github.com/efficient/libcuckoo
$  mkdir build
$  cd build
$  cmake -DCMAKE_INSTALL_PREFIX=/usr/local -DBUILD_EXAMPLES=1 -DBUILD_TEST
   S=1 ..
$  make all && make install
$  git clone https://github.com/Intel-bigdata/Spark-PMoF.git
$  cd Spark-PMoF
$  mvn install -DskipTests
```

5.3 Configure RPMem extension for spark shuffle in Spark

RPMem extension for spark shuffle 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.shuffle.compress                      false
$  spark.driver.extraClassPath                 /$path /Spark-PMoF/core/target/Spark-P
   MoF-1.0-jar-with-dependencies.jar
$  spark.executor.extraClassPath               /$path/Spark-PMoF/core/target/Spark-P
   MoF-1.0-jar-with-dependencies.jar
$  spark.shuffle.manager                       org.apache.spark.shuffle.pmof.PmofShuf
   fleManager
$  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                126833655808 // size should be same as pmem size
$   spark.shuffle.pmof.pmem_list                    /dev/dax0.0, /dev/dax0.1, /dev/dax0.2, /dev/dax0.3, /dev/dax1.0,/dev/dax1.1, /dev/dax1.2,/dev/dax1.3
$   spark.shuffle.pmof.dev_core_set                 dax0.0:0-17,dax0.1:0-17,dax0.2:36-53,dax0.3:36-53,dax1.0:18-35, dax1.1:18-35,dax1.2:54-71,dax1.3:54-71
$   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                         sr609-172.168.0.209,sr611-172.168.0.211
$   spark.driver.rhost                              172.168.0.209 //change to your host
$   spark.driver.rport                              61000
```

6. Spark PMoF Testing

Spark PMoF have been tested with DECISION SUPPORT WORKLOADS and Terasort.

6.1 Decision support workloads

The DECISION SUPPORT WORKLOADS 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 the 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 DECISION SUPPORT WORKLOADS 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

Need to prepare the data with

$HiBench-HOME/bin/workloads/micro/terasort/prepare/prepare.sh

Kick off the evaluation by $HiBench-HOME/bin/workloads/micro/terasort/spark/.run.sh

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.

```
$    Change below command accordingly
$    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 PMEMM guide, it is a pre-request for PMoF deployment)

(1): General PMEMM support: PMEMM support

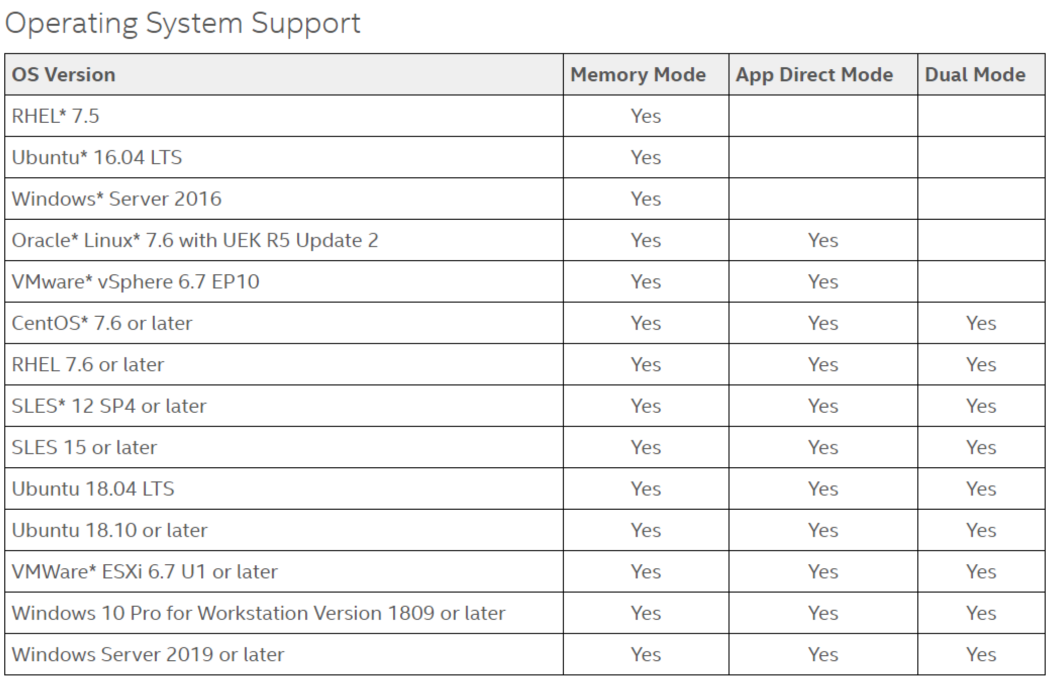
https://www.intel.com/content/www/us/en/support/products/190349/memory-and-storage/data-center-persistent-memory/intel-optane-dc-persistent-memory.html

(2) PMEMM 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