Optimized Analytics Package for Spark* Platform (OAP)

오현주

숙명여자대학교
SOOKMYUNG WOMEN'S UNIVERSITY

pmem 경로의 개수에 따른 패키지 설치 버전

- 1개 이상의 pmem 경로를 가지고 있을 경우
  - oap-tools tag 버전: **v1.1.1-spark-3.1.1**
    - https://oap-project.github.io/pmem-spill/1.1.1/

※ 설치하는 tag 버전이 달라도 설치하는 방법은 거의 비슷함
※ 설치는 **Ubuntu 20.04**에서 진행함

- 2개 이상의 pmem 경로를 가지고 있을 경우
  - OAP tag 버전: **v0.8.4-spark-2.4.4**
    - https://github.com/Intel-bigdata/OAP/tree/v0.8.4-spark-2.4.4
  - v0.9.0-spark-3.0.0 을 사용해도 되지만 OAP 패키지의 RDD Cache PMem Extension 기능을 사용하고자 하므로 설치할 dependency의 개수가 적은 0.8.4 버전을 설치함

현재 ppt의 목적

OAP 설치 전 미리 설치해야하는 것

1. ipmctl 설치

```
$ sudo apt update
$ sudo apt search ipmctl
$ sudo apt info ipmctl
$ sudo apt install ipmctl
```

2. ndctl 설치

```
$ sudo apt search ndctl
$ sudo apt list --installed ndctl
$ sudo apt-get install ndctl
```

OAP 설치 전 미리 설치해야하는 것

3. 현재 app direct mode 인지 확인하기

```
$ sudo ipmctl show -memoryresources
```

```
user@user-Super-Server:~$ sudo ipmctl show -memoryresources
[sudo] password for user:
 MemoryType   | DDR         | DCPMM          | Total
==========================================================
 Volatile     | 64.000 GiB  | 0.000 GiB      | 64.000 GiB
 AppDirect    | -           | 252.000 GiB    | 252.000 GiB
 Cache        | 0.000 GiB   | -              | 0.000 GiB
 Inaccessible | -           | 0.844 GiB      | 0.844 GiB
 Physical     | 64.000 GiB  | 252.844 GiB    | 316.844 GiB
```

OAP 설치 전 미리 설치해야하는 것

3. App direct mode 가 아닌 경우 app direct mode로 바꾸기

```
$ sudo ipmctl show -topology
$ sudo ipmctl show -dimm

$ sudo ipmctl create -goal PersistentMemoryType=AppDirect
```

위의 명령 실행 후 시스템 **reboot**

- command로 reboot하는 방법 ⓘ
  - root account로 진입한 후 reboot 명령 실행하기

```
$ sudo -i
$ sudo reboot
```

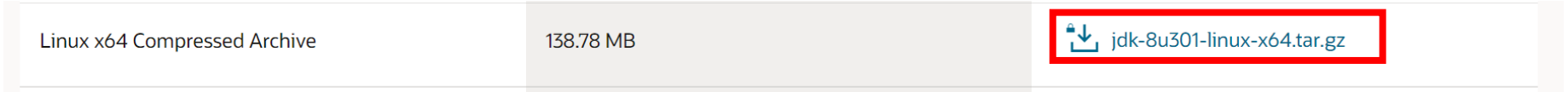
OAP 설치 전 미리 설치해야하는 것

4. Oracle Java 8 설치

- Java 8 를 command로 다운받는 방법
  - 방법 1: 다운받아지지 않는 경우 구글링해서 oracle java 8을 command line에서 다운 받는 방법 찾아보기

```
$ wget -c --header "Cookie: oraclelicense=accept-securebackup-cookie" http://download.oracle.com/otn-pub/java/jdk/8u131-b11/d54c1d3a095b4ff2b6607d096fa80163/jdk-8u131-linux-x64.tar.gz
```

  - 방법 2: oracle 웹사이트에서 직접 oracle JDK 8 다운받기
    - 링크: https://www.oracle.com/kr/java/technologies/javase/javase-jdk8-downloads.html



OAP 설치 전 미리 설치해야하는 것

4. Oracle Java 8 설치

- Oracle Java 8 설치 순서
  - 다음 웹사이트 참고: https://docs.datastax.com/en/jdk-install/doc/jdk-install/installOracleJdkDeb.html
  - 빨간 글씨 유의해서 하기(version은 자기가 다운로드 받은 버전을 넣기)

```
$ sudo mkdir -p /usr/lib/jvm
$ sudo tar zxvf jdk-version-linux-x64.tar.gz -C /usr/lib/jvm
$ sudo update-alternatives --install "/usr/bin/java" "java" "/usr/lib/jvm/jdk1.8.0_version/bin/java" 1
$ sudo update-alternatives --set java /usr/lib/jvm/jdk1.8.0_version/bin/java
$ java -version
```

```
user@user-Super-Server:~$ java -version
java version "1.8.0_301"
Java(TM) SE Runtime Environment (build 1.8.0_301-b09)
Java HotSpot(TM) 64-Bit Server VM (build 25.301-b09, mixed mode)
```

OAP 설치 전 미리 설치해야하는 것

4. Oracle Java 8 설치

- 환경변수 설정 - root 계정에서 설정하기

```
$ which javac      # java가 설치된 경로 확인
$ readlink -f /usr/bin/javac        # java가 실제로 설치된 경로
$ sudo vi ~/.bashrc          # bashrc 파일 하단에 아래의 내용 추가하기
```

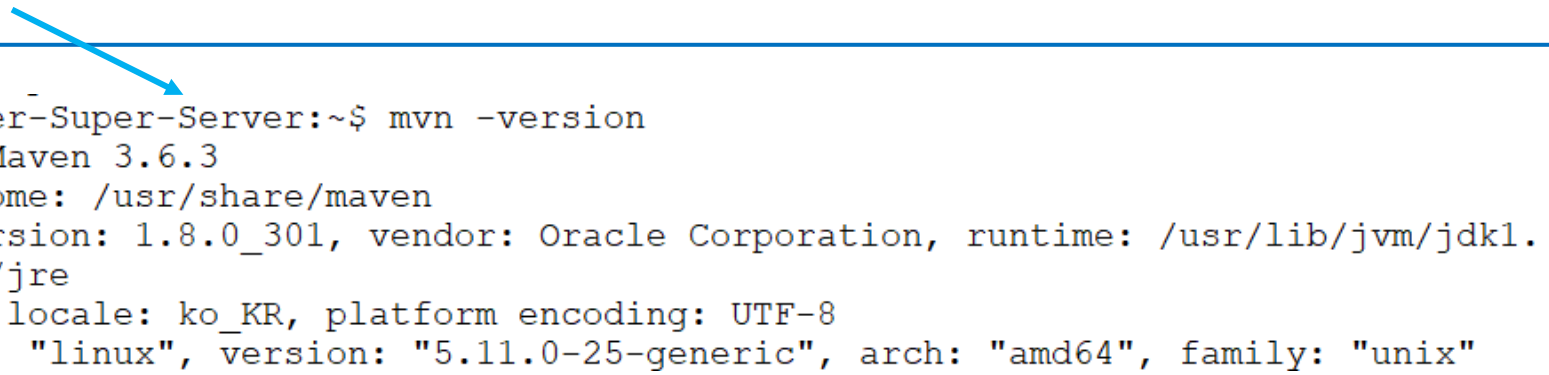
```
export JAVA_HOME=(java가 실제로 설치된 경로에서 bin 이전의 경로까지)
export PATH=$PATH:$HOME:$JAVA_HOME/bin
```

```
$ source ~/.bashrc
$ echo $JAVA_HOME    #자바 설치 경로 확인 가능(확인이 되지 않을 경우 reboot하기)
```

OAP 설치 전 미리 설치해야하는 것

5. maven(자바용 프로젝트 관리도구) 설치

```
$ sudo apt install maven
$ mvn -version
```



```
user@user-Super-Server:~$ mvn -version
Apache Maven 3.6.3
Maven home: /usr/share/maven
Java version: 1.8.0_301, vendor: Oracle Corporation, runtime: /usr/lib/jvm/jdk1.
8.0_301/jre
Default locale: ko_KR, platform encoding: UTF-8
OS name: "linux", version: "5.11.0-25-generic", arch: "amd64", family: "unix"
```

6. git 설치

```
$ sudo apt install git
```

oap-tools 설치 방법

참고 웹사이트:

https://oap-project.github.io/pmem-spill/1.1.1/OAP-Developer-Guide/

[Step 0] root 계정으로 전환

- OAP, hadoop, spark 설치 및 실행 시 주의 사항
  - root user로 진행해야 함(현 ppt부터 **계속 root 계정으로 모든 것을 실행하기**)
  - 우분투에서 root 계정으로 바꾸는 command

```
$ sudo -i
```

- root 계정으로 바꾸지 않을 경우 다음과 같은 오류들이 계속 발생함

```
21/08/12 19:29:03 ERROR Executor: Exception in task 3.0 in stage 1.0 (TID 23)
java.lang.NoClassDefFoundError: com/intel/oap/common/unsafe/PersistentMemoryPlatform
        at org.apache.spark.storage.memory.SerializedValuesHolder.$anonfun$allocator$3(MemoryStore.scala:707)
        at org.apache.spark.storage.memory.SerializedValuesHolder.$anonfun$allocator$3$adapted(MemoryStore.scala:707)
        at org.apache.spark.util.io.ChunkedByteBufferOutputStream.allocateNewChunkIfNeeded(ChunkedByteBufferOutputStream.scala:87)
        at org.apache.spark.util.io.ChunkedByteBufferOutputStream.write(ChunkedByteBufferOutputStream.scala:75)
        at org.apache.spark.storage.memory.RedirectableOutputStream.write(MemoryStore.scala:800)
        at java.io.ObjectOutputStream$BlockDataOutputStream.drain(ObjectOutputStream.java:1877)
        at java.io.ObjectOutputStream$BlockDataOutputStream.setBlockDataMode(ObjectOutputStream.java:1786)
        at java.io.ObjectOutputStream.<init>(ObjectOutputStream.java:
        at org.apache.spark.serializer.JavaSerializationStream.<init
        at org.apache.spark.serializer.JavaSerializerInstance.serial
        at org.apache.spark.storage.memory.SerializedValuesHolder.<t
        at org.apache.spark.storage.memory.MemoryStore.putIteratorAs
        at org.apache.spark.storage.BlockManager.$anonfun$doPutItera
        at org.apache.spark.storage.BlockManager.org$apache$spark$st
        at org.apache.spark.storage.BlockManager.doPutIterator(Block
        at org.apache.spark.storage.BlockManager.getOrElseUpdate(Blo
        at org.apache.spark.rdd.RDD.getOrCompute(RDD.scala:384)
        at org.apache.spark.rdd.RDD.iterator(RDD.scala:335)
        at org.apache.spark.rdd.MapPartitionsRDD.compute(MapPartitio
        at org.apache.spark.rdd.RDD.computeOrReadCheckpoint(RDD.scal
        at org.apache.spark.rdd.RDD.iterator(RDD.scala:337)
        at org.apache.spark.scheduler.ResultTask.runTask(ResultTask.scala:90)
        at org.apache.spark.scheduler.Task.run(Task.scala:131)
        at org.apache.spark.executor.Executor$TaskRunner.$anonfun$run$3(Executor.scala:497)
        at org.apache.spark.util.Utils$.tryWithSafeFinally(Utils.scala:1439)
        at org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:500)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
        at java.lang.Thread.run(Thread.java:748)
```

```
#
# SIGSEGV (0xb) at pc=0x00007fe652c474e5, pid=473848, tid=0x00007fe65aff5700
#
# JRE version: Java(TM) SE Runtime Environment (8.0_301-b09) (build 1.8.0_301-b09)
# Java VM: Java HotSpot(TM) 64-Bit Server VM (25.301-b09 mixed mode linux-amd64 compressed oops)
# Problematic frame:
# C  [libmemkind.so.0+0xd4e5]  memkind_malloc+0x15
#
# Failed to write core dump. Core dumps have been disabled. To enable core dumping, try "ulimit -c unlimited" before starting Java again
```

[Step 1] PMem 포맷 및 마운트하기

- 참고 사이트: https://oap-project.github.io/pmem-spill/1.1.1/User-Guide/
- region 정보 확인하기

```
$ ipmctl show -region
```

- namespace 생성하기
  - 커맨드의 맨 마지막 region[숫자] 부분을 다르게 해서 region의 개수만큼 아래의 명령 실행

```
$ ndctl create-namespace -m fsdax -r region0
```

  - 생성된 namespace 확인하기

```
$ fdisk -l
```

[Step 1] PMem 포맷 및 마운트하기

- 참고 사이트: https://oap-project.github.io/pmem-spill/1.1.1/User-Guide/
- 파일 시스템 생성 및 마운트하기
  - pmem의 개수만큼 생성하면 됨(아래의 예시는 1개일 경우)

```
$ echo y | mkfs.ext4 /dev/pmem0
$ mkdir -p /mnt/pmem0
$ mount -o dax /dev/pmem0 /mnt/pmem0
```

[Step 2] OAP 설치

- OAP, Spark, Hadoop 설치를 위한 디렉토리 생성

```
$ mkdir -p /opt/user
$ cd /opt/user
```

- git에서 oap-tools 가져오기

```
$ git clone https://github.com/oap-project/oap-tools.git
```

[Step 2] OAP 설치

1. oap-tools 설치를 위한 dependency들 설치
   - 방법 1: 제공된 script를 이용하여 설치 진행하기

```
$ cd oap-tools
```

   - 필요한 스크립트: dev/install-compile-time-dependencies.sh 와 dev/scripts/prepare_oap_env.sh
   - 위의 2개의 스크립트가 실행되지 않을 경우 - 권한 추가하기

```
$ cd dev
$ chmod +x install-compile-time-dependencies.sh
$ cd scripts
$ chmod +x prepare_oap_env.sh
$ cd /opt/user/oap-tools
```

[Step 2] OAP 설치

1. oap-tools 설치를 위한 dependency들 설치
   - 방법 1: 제공된 script를 이용하여 설치 진행하기
     - dev/install-compile-time-dependencies.sh 수정 - sh 커맨드를 인식하지 못함
       - 현재 설치는 v1.2.0-rc1 을 기준으로 설치를 진행하는 것이기 때문에 이 수정의 경우 코드를 읽어보고 수정할 필요성이 있으면 수정하시기 바랍니다.

```
$ nano ./dev/install-compile-time-dependencies.sh
```

```
sh $OAP_HOME/dev/scripts/prepare_oap_env.sh --prepare_all
```

⬇ 수정 후(스크립트 안에 2줄 수정하기)

```
/opt/user/oap-tools/dev/scripts/prepare_oap_env.sh --prepare_all
```

[Step 2] OAP 설치

1. oap-tools 설치를 위한 dependency들 설치
   - 방법 1: 제공된 script를 이용하여 설치 진행하기
     - dev/scripts/prepare_oap_env.sh - DEV_PATH 검색해서 수정하기(27번째 줄 근처에 있음)
       - 현재 설치는 v1.2.0-rc1 을 기준으로 설치를 진행하는 것이기 때문에 이 수정의 경우 코드를 읽어보고 수정할 필요성이 있으면 수정하시기 바랍니다.

```
$ nano ./dev/scripts/prepare_oap_env.sh
```

```
DEV_PATH=$OAP_HOME/dev/
```


```
DEV_PATH=$OAP_HOME/dev
```

[Step 2] OAP 설치

1. oap-tools 설치를 위한 dependency들 설치
   - 방법 1: 제공된 script를 이용하여 설치 진행하기

```
$ cd /opt/user/oap-tools
$ ./dev/install-compile-time-dependencies.sh
```

     - prepare_oap_env.sh 를 통해 어떤 dependency들이 설치되었는지 확인할 수 있음
       - 현재 OAP 버전에서는 스크립트로 cmake, gcc(7버전 이상), memkind, vmemcache, HPNL, PMDK, OneAPI, Arrow, LLVM 가 설치됨
     - RDD Cache PMem Extension 기능을 사용하기 위해서는 memkind와 vmemcache 의 설치가 중요
       - 특히, memkind의 설치 디렉토리가 어디인지 확인하기 → /usr/local/lib에 설치되는 것으로 보임

[Step 2] OAP 설치

1. oap-tools 설치를 위한 dependency들 설치
   - 방법 2: 만약 제공된 스크립트로 dependency들이 제대로 설치되지 않을 경우, OAP Developer Guide 에서 해당 OAP 버전에 필요한 dependency들이 링크로 연결되어 있으므로 수동으로 설치하기
     - 참고 링크: https://oap-project.github.io/pmem-spill/1.1.1/OAP-Developer-Guide/
     - dev/scripts/prepare_oap_env.sh을 바탕으로 필요한 dependency들 및 해당 dependency들을 설치하기 위한 command들을 확인할 수 있음 → 이를 바탕으로 설치하기

[Step 2] OAP 설치

2. oap-tools 패키지 빌드하기

```
$ cd /opt/user/oap-tools/dev
$ chmod +x compile-oap.sh        # 실행 권한이 없을 경우
$ cd ..
$ ./dev/compile-oap.sh
```

- Shuffle Remote PMem Extension 같은 경우에는 추가적으로 필요한 dependency들이 있기 때문에 위의 명령으로 모든 module이 다 **빌드되지 않을 수 있음**
  - **pmem-spill과 pmem-common이 제대로 빌드되어 있는지만 확인하면 됨**
    - pmem-common의 경우 /opt/user/oap-tools/pmem-common/target 에서 pmem-common-1.2.0-snapshot-with-spark-3.1.1.jar 파일이 생성되었는지 확인하기(설치하는 패키지 버전마다 이름이 달라질 수 있음 - jar 파일 생성 여부를 확인하면 됨)
    - pmem-spill의 경우 /opt/user/oap-tools/pmem-spill/RDD-Cache/target 에 pmem-common처럼 jar 파일이 생성되었는지 확인하기

```
Please check the result in  /opt/hyunju/oap-tools/dev/release-package!
root@sdp:/opt/hyunju/oap-tools# nano /opt/hyunju/oap-tools/dev/release-package!
root@sdp:/opt/hyunju/oap-tools# nano /opt/hyunju/oap-tools/dev/release-package
root@sdp:/opt/hyunju/oap-tools# cd /opt/hyunju/oap-tools/dev/release-package
root@sdp:/opt/hyunju/oap-tools/dev/release-package# ls
oap-1.2.0-bin-spark-3.1.1  oap-1.2.0-bin-spark-3.1.1.tar.gz
root@sdp:/opt/hyunju/oap-tools/dev/release-package# cd oap-1.2.0-bin-spark-3.1.1
/
root@sdp:/opt/hyunju/oap-tools/dev/release-package/oap-1.2.0-bin-spark-3.1.1# ls
jars
root@sdp:/opt/hyunju/oap-tools/dev/release-package/oap-1.2.0-bin-spark-3.1.1# cd
 jars/
root@sdp:/opt/hyunju/oap-tools/dev/release-package/oap-1.2.0-bin-spark-3.1.1/jar
s# ls
arrow-plasma-4.0.0.jar
hcfs-sql-ds-cache-1.2.0-snapshot.jar
oap-mllib-1.2.0.jar
plasma-sql-ds-cache-1.2.0-snapshot-with-spark-3.1.1.jar
pmem-common-1.2.0-snapshot-with-spark-3.1.1.jar
pmem-rdd-cache-1.2.0-snapshot-with-spark-3.1.1.jar
```

[Step 2] OAP 설치

3. 추가적으로 필요한 라이브러리

- /opt/user/oap-tools/pmem-common/src/native 의 모든 디렉토리 내의 compile.sh 실행시켜주기
  - 패키지 버전에 따라 디렉토리의 개수가 달라짐 - 현재 버전에서는 4개가 존재

```
user@user-Super-Server:/opt/hyunju/oap-tools/pmem-common/src/native$ ls
libpmem  libpmemblk  memkind  vmemcache
```

  - 각 디렉토리 내의 compile.sh

```
user@user-Super-Server:/opt/hyunju/oap-tools/pmem-common/src/native/libpmemblk$ ls
CMakeLists.txt  com_intel_oap_common_unsafe_PMemBlockPlatform.cpp  com_intel_oap_common_unsafe_PMemBlockPlatform.h  compile.sh
```

  - 현재 버전은 디렉토리가 4개이므로 compile.sh를 4번 실행시켜주기

```
$ ./compile.sh                #각 디렉토리 내에서
```

- compile.sh를 실행시켜주게 되면 /opt/user/oap-tools/pmem-common/src 에 resources 디렉토리가 생김
  - resources/linux/64/lib 내에 생성된 라이브러리들을 /usr/local/lib으로 복사해주기

```
user@user-Super-Server:/opt/hyunju/oap-tools/pmem-common/src/resources/linux/64/lib$ ls
libpmblkplatform.so  libpmemmemorymapper.so  libpmplatform.so  libvmemcachejni.so
```

[Step 3] Hadoop 다운로드

- Hadoop 3.2.0 다운로드하기
  - 다운로드 사이트: https://hadoop.apache.org/release/3.2.0.html → tar.gz 확장자 파일 다운로드 받기
  - 혹은 다음 command로 다운받기

```
$ cd /opt/user
$ wget https://archive.apache.org/dist/hadoop/common/hadoop-3.2.0/hadoop-3.2.0.tar.gz
```

  - 기본적인 configuration들 다 설정한 후, 추가적으로 설정이 필요한 파일: yarn-site.xml
    - 다음 피피티의 내용을 더 추가하기

```
<property>
    <name>yarn.nodemanager.pmem-check-enabled</name>
    <value>false</value>
</property>
<property>
    <name>yarn.log-aggregation-enable</name>
    <value>true</value>
</property>
<property>
    <name>yarn.nodemanager.resource.memory-mb</name>
    <value>188416</value>
</property>
<property>
    <name>yarn.scheduler.maximum-allocation-mb</name>
    <value>98304</value>
</property>
<property>
    <name>yarn.scheduler.minimum-allocation-mb</name>
    <value>100</value>
</property>
```

각 환경에 맞게 설정하기

[Step 4] Spark 다운로드

- Spark v3.1.1 **소스코드** 다운로드 받기
  - 앞서 pmem-common/target과 pmem-spill/RDD-Cache/target 에서 확인한 **jar 파일에 언급되어 있는 스파크 버전**을 다운로드 받아야함(아닐 경우, 서로 호환되지 않음)

```
$ cd /opt/user
$ git clone -b v3.1.1 https://github.com/apache/spark.git
```

  - 다운받은 spark-3.1.1 소스코드에 더 좋은 성능을 위해 패치 적용

```
$ cd spark
$ cp /opt/user/oap-tools/pmem-spill/docs/numa-binding-spark-3.1.1.patch .
$ git apply numa-binding-spark-3.1.1.patch
```

아무 메시지가 뜨지 않으면 정상적으로 패치가 적용된 것

[Step 4] Spark 다운로드

- Spark 소스코드 빌드하기
  - 참고 사이트: https://spark.apache.org/docs/latest/building-spark.html
  - YARN을 활성화하고 특정 하둡 버전에 대한 스파크를 빌드하고 싶을 경우

```
$ cd /opt/user/spark
$ ./build/mvn -Pyarn -Dhadoop.version=3.2.0 -DskipTests clean package
```

원하는 하둡 버전 넣기

[Step 4] Spark 다운로드

- spark-env.sh 설정

```
$ cp /opt/user/spark/conf/spark-env.sh.template /opt/user/spark/conf/spark-env.sh
$ nano /opt/user/spark/conf/spark-env.sh
```

  - 하단에 다음 내용을 추가하기 - hdfs에 write하고 YARN 리소스매니저를 사용하기 위함

```
export HADOOP_CONF_DIR=/opt/user/hadoop/etc/hadoop
export YARN_CONF_DIR=/opt/user/hadoop/etc/hadoop
```

- spark-defaults.conf 설정

```
$ cp /opt/user/spark/conf/spark-defaults.conf.template /opt/user/spark/conf/spark-defaults.conf
$ nano /opt/user/spark/conf/spark-defaults.conf
```

  - 다음 피피티 내용을 spark-defaults.conf 하단에 추가해주기

spark.memory.pmem.extension.enabled true
spark.memory.pmem.initial.path /mnt/pmem0
spark.memory.pmem.initial.size 256g
spark.memory.pmem.usable.ratio 0.85
spark.yarn.numa.enabled true
spark.yarn.numa.num 0
spark.memory.pmem.mode AppDirect

각 환경에 맞게 설정하기

spark.files file:///opt/user/oap-tools/pmem-spill/RDD-Cache/target/pmem-rdd-cache-1.2.0-snapshot-with-spark-3.1.1.jar,file:///opt/user/oap-tools/pmem-common/target/pmem-common-1.2.0-snapshot-with-spark-3.1.1.jar
spark.executor.extraClassPath ./pmem-rdd-cache-1.2.0-snapshot-with-spark-3.1.1.jar:./pmem-common-1.2.0-snapshot-with-spark-3.1.1.jar
spark.driver.extraClassPath file:///opt/user/oap-tools/pmem-spill/RDD-Cache/target/pmem-rdd-cache-1.2.0-snapshot-with-spark-3.1.1.jar:file:///opt/user/oap-tools/pmem-spill/RDD-Cache/target/pmem-rdd-cache-1.2.0-snapshot-with-spark-3.1.1.jar

spark.executorEnv.LD_LIBRARY_PATH /usr/local/lib
spark.executor.extraLibraryPath /usr/local/lib
spark.driver.extraLibraryPath /usr/local/lib

memkind 및 추가적인 library 들이 존재하는 디렉토리

최종적인 bashrc

```
export JAVA_HOME=/usr/lib/jvm/jdk1.8.0_131
export PATH=$PATH:$HOME:$JAVA_HOME/bin

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib
export LD_RUN_PATH=$LD_RUN_PATH:/usr/local/lib

export HADOOP_HOME="/opt/user/hadoop"
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$HADOOP_HOME/sbin
export HADOOP_MAPRED_HOME=${HADOOP_HOME}
export HADOOP_COMMON_HOME=${HADOOP_HOME}
export HADOOP_HDFS_HOME=${HADOOP_HOME}
export YARN_HOME=${HADOOP_HOME}
export HADOOP_CLASSPATH=$(hadoop classpath)

export SPARK_HOME="/opt/user/spark"
export PATH=$PATH:/opt/user/spark:/opt/user/spark/bin
```

k-means 벤치마크 실행

참고 사이트:

https://github.com/Intel-bigdata/HiBench

[Step 1] HiBench 빌드하기

- HiBench 다운로드

```
$ cd /opt/user
$ git clone https://github.com/Intel-bigdata/HiBench.git
```

  - HiBench가 spark 2.4.x, 3.0.x 을 지원하는 버전으로 다운받기

- Sparkbench의 K-means 벤치마크를 실행
  - HiBench의 sparkbench를 빌드

```
$ cd HiBench
$ mvn -Psparkbench -Dspark=3.0 -Dscala=2.12 clean package
```

  - 현재(21/08/29)까지 HiBench가 spark 3.0까지 고려해서 만들었으므로 위와 같이 스파크와 스칼라 버전을 명시하여 HiBench를 빌드함

[Step 2] Sparkbench 실행하기 전 사전 준비

- 실행하기 전 Python 2.x를 요구할 수도 있음
  - python2가 없어서 실행에 **오류가 날 경우** 다음으로 python2 설치

```
$ sudo apt install python2
```

- HDFS, Yarn, Spark 를 클러스터에서 실행

```
$ cd /opt/user
$ ./hadoop/sbin/start-all.sh
$ ./spark/sbin/start-all.sh
```

[Step 3] Sparkbench 실행

1. Configuration 설정
   - hadoop.conf

```
$ cd /opt/user/HiBench/conf
$ cp hadoop.conf.template hadoop.conf
```

```
# Hadoop home
hibench.hadoop.home      /opt/user/hadoop

# The path of hadoop executable
hibench.hadoop.executable     ${hibench.hadoop.home}/bin/hadoop

# Hadoop configraution directory
hibench.hadoop.configure.dir  ${hibench.hadoop.home}/etc/hadoop

# The root HDFS path to store HiBench data
hibench.hdfs.master       hdfs://localhost:9000/user/user

# Hadoop release provider. Supported value: apache
hibench.hadoop.release    apache
```

※ 자신의 설정에 맞춰서 다음을 고치기

[Step 3] Sparkbench 실행

1. Configuration
   - spark.conf

```
$ cd /opt/user/HiBench/conf
$ cp spark.conf.template spark.conf
```

```
# Spark home
hibench.spark.home      /opt/user/spark

# Spark master
#   standalone mode: spark://xxx:7077
#   YARN mode: yarn-client
hibench.spark.master    yarn

# executor number and cores when running on Yarn
hibench.yarn.executor.num     4
hibench.yarn.executor.cores   4

# executor and driver memory in standalone & YARN mode
spark.executor.memory  10g
spark.driver.memory    5g

spark.memory.pmem.initial.path /mnt/pmem0,/mnt/pmem1,/mnt/pmem2,/mnt/pmem3
#spark.memory.pmem.initial.size 1539316278886
spark.memory.pmem.initial.size 1450g
spark.memory.pmem.usable.ratio 0.85
spark.yarn.numa.enabled true
spark.yarn.numa.num 4
spark.memory.pmem.mode AppDirect

spark.files                     file:///opt/user/OAP/oap-spark/target/oap-spark-0.8.4-with-spark-2.4.4.jar,file:///opt/
spark.executor.extraClassPath   ./oap-spark-0.8.4-with-spark-2.4.4.jar:./oap-common-0.8.4-with-spark-2.4.4.jar
#spark.executor.extraClassPath  file:///home/sdp/OAP/oap-spark/target/oap-spark-0.8.4-with-spark-2.4.4.jar:file:///home
spark.driver.extraClassPath     file:///opt/user/OAP/oap-spark/target/oap-spark-0.8.4-with-spark-2.4.4.jar:file:///opt/

spark.executorEnv.LD_LIBRARY_PATH       /usr/local/lib
spark.driver.extraLibraryPath           /usr/local/lib
spark.executor.extraLibraryPath         /usr/local/lib
```

※ 자신의 환경과 맞게 작성하기

spark-defaults.conf 에서 작성했던 내용과 같은 내용을 작성해주기

- 문서에는 맨 아래의 3가지 configuration만 추가해주면 된다고 하지만 그럴 경우 PMEM_AND_DISK 스토리지 레벨을 인식할 수 없다는 에러가 뜨게 됨

[Step 3] Sparkbench 실행

1. Configuration
   - kmeans.conf

```
$ cd /opt/user/HiBench/conf
$ nano ./workloads/ml/kmeans.conf
```

```
hibench.kmeans.storage.level                    PMEM_AND_DISK
hibench.kmeans.initializationmode Random
```

기본적으로 MEMORY_ONLY라고 되어 있는데 PMEM_AND_DISK 스토리지 레벨을 실험해보고 싶기 때문에 PMEM_AND_DISK로 바꾸기

[Step 3] Sparkbench 실행

2. K-means 벤치마크 실행

```
$ cd /opt/user/HiBench
$ bin/workloads/ml/kmeans/prepare/prepare.sh
$ bin/workloads/ml/kmeans/spark/run.sh
```

```
root@sdp:/opt/user/HiBench# bin/workloads/ml/kmeans/spark/run.sh
patching args=
Parsing conf: /opt/user/HiBench/conf/hadoop.conf
Parsing conf: /opt/user/HiBench/conf/hibench.conf
Parsing conf: /opt/user/HiBench/conf/spark.conf
Parsing conf: /opt/user/HiBench/conf/workloads/ml/kmeans.conf
probe sleep jar: /opt/user/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-client
-jobclient-2.7.0-tests.jar
start ScalaSparkKmeans bench
hdfs rm -r: /opt/user/hadoop/bin/hadoop --config /opt/user/hadoop/etc/hadoop fs
-rm -r -skipTrash hdfs://localhost:9000/user/user/HiBench/Kmeans/Output
rm: `hdfs://localhost:9000/user/user/HiBench/Kmeans/Output': No such file o...
hdfs du -s: /opt/user/hadoop/bin/hadoop --config /opt/user/hadoop/etc/hadoop fs
-du -s hdfs://localhost:9000/user/user/HiBench/Kmeans/Input
Export env: SPARKBENCH_PROPERTIES_FILES=/opt/user/HiBench/report/kmeans/spark/co
nf/sparkbench/sparkbench.conf
Export env: HADOOP_CONF_DIR=/opt/user/hadoop/etc/hadoop
Submit Spark job: /opt/user/spark/bin/spark-submit  --properties-file /opt/user/
HiBench/report/kmeans/spark/conf/sparkbench/spark.conf --class com.intel.hibench
.sparkbench.ml.DenseKMeans --master yarn --num-executors 4 --executor-cores 4 --
executor-memory 10g /opt/user/HiBench/sparkbench/assembly/target/sparkbench-asse
mbly-8.0-SNAPSHOT-dist.jar -k 10 --numIterations 5 --storageLevel PMEM_AND_DISK
--initMode Random hdfs://localhost:9000/user/user/HiBench/Kmeans/Input/samples
21/08/28 17:46:18 INFO util.ShutdownHookManager: Deleting directory /tmp/sp...
finish ScalaSparkKmeans bench
```

실행이 성공적이라는 표시

[Step 3] Sparkbench 실행 도중 오류 해결

```
root@sdp:/opt/hyunju/HiBench/bin/workloads/ml/kmeans/prepare# ./prepare.sh
patching args=
Parsing conf: /opt/hyunju/HiBench/conf/hadoop.conf
Parsing conf: /opt/hyunju/HiBench/conf/hibench.conf
Parsing conf: /opt/hyunju/HiBench/conf/spark.conf
Parsing conf: /opt/hyunju/HiBench/conf/workloads/ml/kmeans.conf
probe sleep jar: /opt/hyunju/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-client-jobclient-3.2.0-tests.jar
Traceback (most recent call last):
  File "/opt/hyunju/HiBench/bin/functions/load_config.py", line 685, in <module>
    load_config(conf_root, workload_configFile, workload_folder, patching_config)
  File "/opt/hyunju/HiBench/bin/functions/load_config.py", line 217, in load_config
    generate_optional_value()
  File "/opt/hyunju/HiBench/bin/functions/load_config.py", line 613, in generate_optional_value
    probe_masters_slaves_hostnames()
  File "/opt/hyunju/HiBench/bin/functions/load_config.py", line 549, in probe_masters_slaves_hostnames
    probe_masters_slaves_by_Yarn()
  File "/opt/hyunju/HiBench/bin/functions/load_config.py", line 500, in probe_masters_slaves_by_Yarn
    assert 0, "Get workers from yarn-site.xml page failed, reason:%s\nplease set `hibench.masters.hostnames` and `hibench.slaves.hostnames` manually" % e
AssertionError: Get workers from yarn-site.xml page failed, reason:Unknown resourcemanager, please check `hibench.hadoop.configure.dir` and "yarn-site.xml" file
please set `hibench.masters.hostnames` and `hibench.slaves.hostnames` manually
/opt/hyunju/HiBench/bin/functions/workload_functions.sh: line 38: .: filename argument required
.: usage: . filename [arguments]
start HadoopPrepareKmeans bench
```

※ hibench.conf에 있는
hibench.masters.hostnames와
hibench.slaves.hostnames를 localhost
라고 manual하게 해주기

```
hibench.masters.hostnames       localhost
hibench.slaves.hostnames        localhost
```