5 Spark 分布式内存处理框架

5.1 实验介绍

5.1.1 关于本实验

Spark 是基于内存的分布式批处理引擎，它最大的特点是延迟小，具有很高的容错性和可拓展性。

鲲鹏 BigData Pro 大数据解决方案使用鲲鹏服务器，鲲鹏支持 8 个内存通道和更高的内存速率，总带宽提升 46%，同时优化内存预取和 Cache 访问等过程。内存带宽提升 46%，内存访问性能更优。基于内存的计算引擎，收益明显。

5.1.2 实验目的

本实验手册详细介绍了鲲鹏 BigData Pro 解决方案下 Spark 分布式内存处理组件的安装部署流程，最后以 Spark-SQL 和 WordCount 为例演示 Spark 的基本功能。

5.2 实验任务 1

5.2.1 启动 Hadoop 集群

步骤 1 在 node1 节点执行以下命令：

```
> start-dfs.sh ; start-yarn.sh
```

返回信息中有以下内容，表示 hadoop 集群启动成功：

```
Starting namenodes on [node1]
Starting secondary namenodes [node1]
starting yarn daemons
```

备注：关闭 hadoop 集群的命令为 stop-dfs.sh && stop-yarn.sh

5.2.2 验证 Hadoop 状态

使用jps 命令在 node1-4 中查看 Java 进程，node1 中有 NameNode、SecondaryNameNode 和 ResourceManager 进程：

```
[root@node1 ~]# jps
1538 WrapperSimpleApp
5732 SecondaryNameNode
5508 NameNode
6205 Jps
5918 ResourceManager
```

node2-4 中有 NodeManager 和 Datanode 进程：

```
[root@node2 ~]# jps
3026 Jps
2740 DataNode
1515 WrapperSimpleApp
2862 NodeManager
```

访问 http://node1 弹性公网 IP:50070，可以访问 HDFS 的 WebUI 界面：

图 5-1 HDFS 的 WebUI 界面

5.3 实验任务 2

5.3.1 安装部署

步骤 1 解压 spark 软件包，并重命名文件夹

在 node1 节点上执行如下命令，解压 spark 软件包，并重命名文件夹

```
> cd /home/extend_tools
> wget
https://xunfang.obs.cn-south-1.myhuaweicloud.com/spark-2.3.0-bin-dev
-with-sparkr.tgz
> tar -zxvf spark-2.3.0-bin-dev-with-sparkr.tgz -C /home/modules/
> cd /home/modules/
> mv spark-2.3.0-bin-dev spark-2.3.0
```

步骤 2 替换旧版 jar 包

在 node1 节点，执行如下命令将 hadoop-huaweicloud-2.8.3.36.jar 包拷贝到 jars 目录：

```
> cp /home/extend_tools/hadoop-huaweicloud-2.8.3.33.jar
/home/modules/spark-2.3.0/jars/
```

在 node1 节点，替换 snappy-java-1.0.4.1.jar 文件

```
> cd /home/extend_tools
> wget
https://xunfang.obs.cn-south-1.myhuaweicloud.com/snappy-java-1.0.4.1.
jar
> cp snappy-java-1.0.4.1.jar /home/modules/spark-2.3.0/jars/
> ll /home/modules/spark-2.3.0/jars/snappy-java-1.0.4.1.jar
```

如果提示 snappy-java-1.0.4.1.jar 包已存在，输入“y”覆盖

步骤 3 配置 spark-env.sh

在 node1 节点上，执行命令

```
> cp /home/modules/spark-2.3.0/conf/spark-env.sh.template
/home/modules/spark-2.3.0/conf/spark-env.sh
> vim /home/modules/spark-2.3.0/conf/spark-env.sh
```

并在末尾添加如下配置项：

```
export JAVA_HOME=/usr/lib/jvm/java
export SCALA_HOME=/home/modules/spark-2.3.0/examples/src/main/scala
export HADOOP_HOME=/home/modules/hadoop-2.8.3
export HADOOP_CONF_DIR=/home/modules/hadoop-2.8.3/etc/hadoop
export SPARK_HOME=/home/modules/spark-2.3.0
```

```
export SPARK_DIST_CLASSPATH=$(/home/modules/hadoop-2.8.3/bin/hadoop classpath)
```

步骤 4 配置 slaves

在 node1 节点上，执行命令：

```
> vim /home/modules/spark-2.3.0/conf/slaves
```

并在末尾添加如下节点：

```
node2
node3
node4
```

步骤 5 分发组件

在 node1 执行如下命令，将 spark-2.3.0 目录拷贝到其他各个节点

```
> for i in {2..4};do scp -r /home/modules/spark-2.3.0 root@node${i}:/home/modules/;done
```

拷贝完成后，在 node2-4 节点中执行命令一下命令发现都有了安装包：

```
> ls /home/modules  | grep spark
spark-2.3.0
```

步骤 6 配置环境变量

在 node1~node4 节点上，执行命令

```
> vim /etc/profile
```

然后在文件末尾添加如下环境变量：

```
export SPARK_HOME=/home/modules/spark-2.3.0
export PATH=$SPARK_HOME/bin:$SPARK_HOME/sbin:$PATH
```

添加完成后，继续如下命令使环境变量生效

```
> source /etc/profile
```

5.3.2 验证 Spark-SQL 基本功能

步骤 1 登录 spark-sql 客户端

在 node1 上执行 spark-sql 命令，登录 spark-sql 客户端

```
> spark-sql


# 显示以下内容时，表示已登录 spark-sql 客户端
spark-sql>
```

步骤 2 创建数据库

```
spark-sql> show databases;
# 返回以下信息
default
Time taken: 2.057 seconds, Fetched 1 row(s)
20/10/15 23:10:28 INFO thriftserver.SparkSQLCLIDriver: Time taken:
2.057 seconds, Fetched 1 row(s)
```

表示目前只有默认的 default 数据库。

创建数据库并再次查看，根据输出结果显示 testdb 数据库创建成功，最后

```
spark-sql> create database testdb;
Time taken: 0.175 seconds

spark-sql> show databases;
default
testdb
Time taken: 0.016 seconds, Fetched 2 row(s)

spark-sql> use testdb;
Time taken: 0.015 seconds
```

步骤 3 创建表

创建 testtable 表，并查看表结构

```
spark-sql> create table testtable(value INT);
Time taken: 0.399 seconds

spark-sql> desc testtable;
......
value   int     NULL
Time taken: 0.346 seconds, Fetched 1 row(s)
20/10/15 23:11:30 INFO thriftserver.SparkSQLCLIDriver: Time taken:
0.404 seconds, Fetched 1 row(s)
```

步骤 4 插入数据

```
spark-sql> insert into testtable values (1000);
Time taken: 1.024 seconds

spark-sql> select * from testtable;
......
```

```
1000
Time taken: 0.241 seconds, Fetched 1 row(s)
20/10/15 23:12:11 INFO thriftserver.SparkSQLCLIDriver: Time taken:
0.323 seconds, Fetched 1 row(s)
```

步骤 5 退出

```
exit;
```

5.3.3 验证 WordCount 实例

步骤 1 在本地编辑 WordCount.txt

新打开一个终端，在/root 路径下创建一个 wordcount.txt 文件：

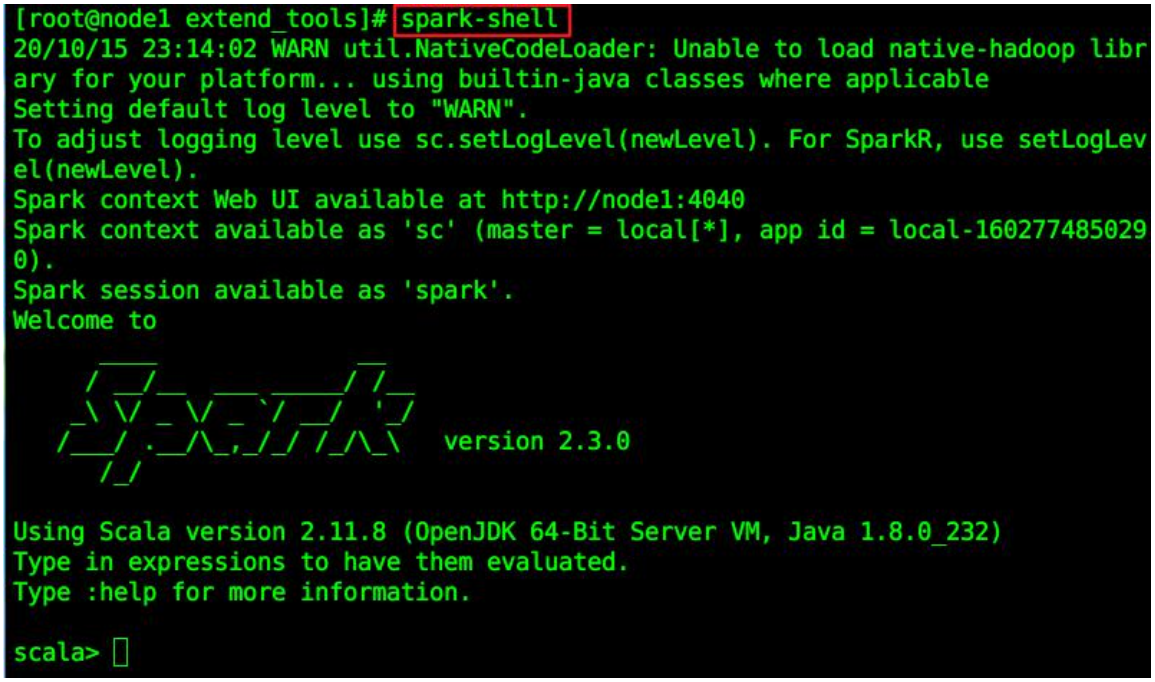
```
> vim wordcount.txt


# 添加以下内容：
ni hao wa
hello spark java
hdfs hadoop spark java
hello spark java
```

步骤 2 上传到 HDFS

```
> hdfs dfs -put wordcount.txt /tmp/
```

步骤 3 启动 Spark-Shell

```
> spark-shell
```



步骤 4 在 spark-shell 中执行程序

```
# 1、获取 hdfs 中的 wordcount.txt 文本
scala> val fileRDD=sc.textFile("hdfs://node1:8020/tmp/wordcount.txt")
fileRDD: org.apache.spark.rdd.RDD[String] =
hdfs://node1:8020/tmp/wordcount.txt MapPartitionsRDD[3] at textFile at
<console>:24


# 2、使用空格进行分词，并计数
scala> val count=fileRDD.flatMap(line => line.split(" ")).map(word =>
(word,1)).reduceByKey(_+_)
count: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[10] at
reduceByKey at <console>:25


# 3、输出计数结果
scala> count.collect()
res1: Array[(String, Int)] = Array((hello,2), (java,3), (wa,1), (hao,1),
(spark,3), (hadoop,1), (ni,1), (hdfs,1))
```

```
scala> val file=sc.textFile("hdfs://node1:8020/tmp/wordcount.txt")
file: org.apache.spark.rdd.RDD[String] = hdfs://node1:8020/tmp/wordcount.txt Ma
pPartitionsRDD[1] at textFile at <console>:24

scala> val fileRDD=sc.textFile("hdfs://node1:8020/tmp/wordcount.txt")
fileRDD: org.apache.spark.rdd.RDD[String] = hdfs://node1:8020/tmp/wordcount.txt
 MapPartitionsRDD[3] at textFile at <console>:24

scala> val count=fileRDD.flatMap(line => line.split(" ")).map(word => (word,1))
.reduceByKey(_+_)
count: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[6] at reduceByKey
at <console>:25

scala> count.collect()
res0: Array[(String, Int)] = Array((hello,2), (java,3), (wa,1), (hao,1), (spark
,3), (hadoop,1), (ni,1), (hdfs,1))

scala>
```

根据 count.collect()输出结果，可以查看到分词成功。

---结束!

5.4 实验小结

本章实验讲述了鲲鹏 BigData Pro 解决方案下 Spark 组件的安装部署，体验了 Spark-sql 的基本操作和 wordcount 案例。完成本章实验后，学员可以掌握 Spark 的安装部署方法，和 Spark SQL 的类 SQL 操作。