# 一．集群配置

1.修改虚拟机的主机名

sudo vi /etc/hostname

sudo reboot

sudo hostnamectl set-hostname <新主机名>

2.配置静态IP

sudo vi /etc/netplan/[ip配置文件名]

配置文件:

network:

version: 2

renderer: networkd

ethernets:

ens33: # 替换为你的实际网卡名称

dhcp4: no

addresses:

- 192.168.80.154/24 # 你的静态 IP 和子网掩码

gateway4: 192.168.80.2 # 默认网关

nameservers:

addresses:

- 8.8.8.8 # Google 的 DNS 服务器

- 114.114.114.114 # Google 的备用 DNS 服务器

sudo netplan apply

3.配置主机映射

sudo vi /etc/hosts

4.配置SSH服务

sudo apt update

sudo apt install openssh-server

sudo systemctl status ssh

生成SSH密钥对

ssh-keygen -t rsa -P ""

将生成的公钥添加到主节点自身的 ~/.ssh/authorized\_keys 文件中，以便主节点可以通过 SSH 无密码登录自己

cat ~/.ssh/id\_rsa.pub >> ~/.ssh/authorized\_keys

确保 authorized\_keys 文件的权限是安全的：

chmod 600 ~/.ssh/authorized\_keys

主机登录自身

ssh localhost

将公钥传送到从节点

ssh-copy-id kyle@hadoop1

5.虚拟机新建文件夹

mkdir -p /export/data

mkdir -p /export/servers

mkdir -p /export/software

6.虚拟机安装jdk

sudo apt install openjdk-8-jdk

可以运行 java 和 javac 等命令，而无需显式配置 JAVA\_HOME

export JAVA\_HOME= /usr/lib/jvm/java-1.8.0-openjdk-amd64

export PATH=$PATH:$JAVA\_HOME/bin

7.安装hadoop(.tar.gz)

tar -zxvf

8.配置hadoop环境变量

export JAVA\_HOME= /usr/lib/jvm/java-1.8.0-openjdk-amd64

export PATH=$PATH:$JAVA\_HOME/bin

export HADOOP\_HOME=/home/kyle/export/servers/hadoop-3.0.0

export PATH=$PATH:$HADOOP\_HOME/bin:$HADOOP\_HOME/sbin

9.hadoop集群配置

主节点配置目录

cd ~/export/servers/hadoop-3.0.0/etc/hadoop/

1).修改hadoop-env.sh

vi hadoop-env.sh

export JAVA\_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64

2).修改core-site.xml

vi core-site.xml

<configuration>

<!-- 指定 NameNode 的地址 -->

<property>

<name>fs.defaultFS</name>

<value>hdfs://hadoop1:9000</value>

</property>

<!-- 指定 hadoop 数据的存储目录 -->

<property>

<name>hadoop.tmp.dir</name>

<value>/home/kyle/hadoop\_tmp/hdfs/data</value>

</property>

</configuration>

3).修改hdfs-site.xml

vi hdfs-site.xml

<configuration>

<!--指定HDFS的数量-->

<property>

<name>dfs.replication</name>

<value>3</value>

</property>

<!-- nn web端访问地址-->

<property>

<name>dfs.namenode.http-address</name>

<value>hadoop1:9870</value>

</property>

<!-- 2nn web端访问地址-->

<property>

<name>dfs.namenode.secondary.http-address</name>

<value>hadoop2:9868</value>

</property>

</configuration>

4).修改mapred-site.xml

vi mapred-site.xml

<configuration>

<!--指定MapReduce运行时的框架，这里指定在YARN上，默认在local-->

<property>

<name>mapreduce.framework.name</name>

<value>yarn</value>

</property>

</configuration>

5).修改yarn-site.xml

vi yarn-site.xml

<configuration>

<!--指定YARN集群的管理者（ResourceManager）的地址-->

<property>

<name>yarn.resourcemanager.hostname</name>

<value>hadoop1</value>

</property>

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

</property>

</configuration>

6).修改workers文件

vi works

hadoop1

hadoop2

hadoop3

7).将主节点hadoop目录copy给子节点

scp /etc/profile hadoop2:/etc/profile

scp /etc/profile hadoop3:/etc/profile

scp -r ~/export/ hadoop2:~

scp -r ~/export/ hadoop3:~

8).在主节点格式化文件系统

hdfs namenode -format

10.hadoop集群测试

1).主节点启动dfs、yarn

start-dfs.sh start-yarn.sh

dfs:分布式文件系统,存储大规模数据集，确保数据在集群中的分布式节点上冗余存储和高可用性;

yarn:资源管理和作业调度框架，负责管理 Hadoop 集群中的计算资源，并调度用户提交的任务(如 MapReduce、Spark 任务等);

2).关闭防火墙

sudo ufw status #查看防火墙状态

sudo ufw disable #关闭防火墙

sudo systemctl disable ufw #关闭防火墙开机启动

3).主节点关闭dfs、yarn

stop-dfs.sh stop-yarn.sh

# win客户端操作集群

1.下载win-utils，并设置环境变量

<https://github.com/steveloughran/winutils>

1. 命令操作

hdfs dfs -ls [文件名]

hdfs dfs -mkdir [文件夹名]

hdfs dfs -rm [文件名] 删除HDFS文件 -r 递归删除目录

hdfs dfs -mv [源路径] [目标路径]

hdfs dfs -put /[本地文件] /[hdfs文件]

hdfs dfs -get /[hdfs文件] /[本地文件]

hdfs dfs -chmod 777 [hdfs文件]

hdfs dfs -chmod -R 777 [hdfs文件]

# Spark

1.简介

spark是基于scala语言开发的,专为大规模数据处理而设计的快速通道的计算引擎

pyspark是python中apache spark的接口

2.配置spark文件

配置环境变量

export SPARK\_HOME=/home/kyle/export/servers/spark

export PATH=$PATH:$SPARK\_HOME/bin:$SPARK\_HOME/sbin

1).spark-env.sh

cp spark-env.sh.template spark-env.sh

vi spark-env.sh

# 指定JAVA\_HOME (java主目录路径)

export JAVA\_HOME = /usr/lib/jvm/java-1.8.0-openjdk-amd64

# 指定Spark Master地址(spark主节点地址)

export SPARK\_MASTER\_HOST = hadoop1

export SPARK\_MASTER\_PORT = 7077

export SPARK\_HISTORY\_OPTS="

-Dspark.history.ui.port=4000

-Dspark.history.retainedApplications=3

-Dspark.history.fs.logDirectory=hdfs://hadoop1:9000/spark\_log"

hdfs dfs -mkdir -p /spark\_log [!!!important]

2).workers

cp workers.template workers

vi workers

hadoop1

hadoop2

hadoop3

3).配置日志服务

cp spark-defaults.conf.template spark-defaults.conf

spark.event.enabled true

spark.event.Log.dir hdfs://hadoop1:9000/spark\_log

spark.event.Log.compress true

4).启动spark master 和 slaves, 以及HistoryServer

cd /export/servers/spark

sbin/start-all.sh

sbin/start-history-server.sh

sbin/start-master.sh

sbin/start-workers.sh spark://hadoop1:7077

5).将主节点hadoop目录copy给子节点

scp -r spark kyle@hadoop2:$PWD

scp -r spark kyle@hadoop3:$PWD

# 四．PySpark

pip install pyspark==3.4.3

pyspark --master spark://hadoop1:7077

conda install -c conda-forge hadoop

代码部分:

1.demo部分

Eg:

import os

from pyspark import SparkConf, SparkContext

os.environ['PYSPARK\_PYTHON'] = '/home/kyle/miniconda3/envs/pyspark/bin/python3.8'

conf = SparkConf().setAppName("classification\_model").setMaster("spark://hadoop1:7077")

sc = SparkContext(conf=conf)

print(sc)

ls = [1, 2, 3, 4]

result = sc.parallelize(ls).map(lambda l: l \*\* 2)

for element in result.collect():

print(element)

sc.stop()

2.数据集加载部分

Eg:

1).导入数据

# 从pyspark.sql模块导入SparkSession类

from pyspark.sql import SparkSession

# 创建一个SparkSession，设置应用程序名称为"load\_dataset"

spark = SparkSession.builder.appName("load\_dataset").getOrCreate()

# 从指定路径读取CSV文件，推断数据类型，并将第一行作为表头

iris\_df = spark.read.csv('../dataset/iris.csv', inferSchema=True, header=True)

inferSchema:自动推断每列的数据类型 header=True:第一行视为列名

iris\_df = iris\_df.drop(iris\_df.columns[0]) # 忽略第0列数据

# 打印DataFrame的第一行

print(iris\_df.take(1))

2).创建特征向量

# 指定了输入列和输出列,即将四个特征列合并为一个名为 features 的向量列

# numeric\_cols = [field.name for field in iris\_df.schema.fields if

# isinstance(field.dataType, (DoubleType, FloatType, IntegerType))]

vectorAssembler = VectorAssembler(inputCols=["sepal\_length", "sepal\_width", "petal\_length", "petal\_width"], outputCol="features")

# transform 方法将特征向量添加到原始的 DataFrame 中

viris\_df = vectorAssembler.transform(iris\_df)

# 返回DataFrame的第一行,包括新生成的 features 列

viris\_df.take(1)

3).标签转化

# 创建StringIndexer,指定输入列为species,输出列为labels

label\_indexer = StringIndexer(inputCol='species', outputCol='labels')

# fit方法会计算出 species 列中的所有类别,并为每个类别分配一个唯一的数字索引,但并不会返回已转换的 DataFrame

# transform 方法将转换后的标签添加到原始的 DataFrame 中

iris\_df = label\_indexer.fit(iris\_df).transform(iris\_df)

print(iris\_df.take(1))

4).多层感知器模型创建和训练

# 4.1 划分数据集和训练集

train\_df, test\_df = iris\_df.randomSplit([0.8, 0.2], seed=123)

print("iris\_df: ", iris\_df.count(), "train\_df: ", train\_df.count(), "test\_df: ", test\_df.count())

# 4.2 创建多层感知器

layers = [4, 5, 6, 3]

mlp\_model=MultilayerPerceptronClassifier(layers=layers,featuresCol='features',labelCol='labels')

# 4.3 训练模型

tarin\_model = mlp\_model.fit(train\_df)

# 4.4 模型预测

predictions = tarin\_model.transform(test\_df)

# 4.5 模型预测结果

predictions.select("labels", "prediction", "features").show()

# 4.6 评估模型

evaluator=MulticlassClassificationEvaluator(labelCol="labels",predictionCol="prediction",metricName="accuracy")

accuracy = evaluator.evaluate(predictions)

print(f"准确率: {accuracy}")

# 五．Pandas

1.DataFrame操作

hdfs dfs -rm -r /outputs/params/iris\_model.pkl

hdfs dfs -rm -r /outputs/test\_result/test\_iris\_datasets.csv

hdfs dfs -rm -r /outputs/test\_result/test\_metrics.txt