**TPD-DC运行规范**

本Spark集群基于Docker Desktop，包含一个节点。

Windows cmd 运行以下命令以进入容器：

docker exec -it master bash –login

**TPC-DS 运行步骤**

启动服务：

**hdfs namenode &**

**hdfs datanode &**

**hdfs secondarynamenode &**

HDFS 已经启动 NameNode 和 DataNode，可以开始任务。

安装sbt：

echo "deb https://repo.scala-sbt.org/scalasbt/debian all main" | tee /etc/apt/sources.list.d/sbt.list

echo "deb https://repo.scala-sbt.org/scalasbt/debian /" | tee /etc/apt/sources.list.d/sbt\_old.list

curl -sL "https://keyserver.ubuntu.com/pks/lookup?op=get&search=0x2EE0EA64E40A89B84B2DF73499E82A75642AC823" |apt-key add

apt-get update

apt-get install sbt

安装spark-sql-perf：

git clone https://github.com/databricks/spark-sql-perf.git

cd spark-sql-perf

sbt package

export SPARK\_HOME=/opt/spark-3.4.0-force

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

spark-shell \

--conf spark.sql.catalogImplementation=hive \

--jars /tpcds/spark-sql-perf/target/scala-2.12/spark-sql-perf\_2.12-0.5.1-SNAPSHOT.jar

或者：spark-submit \

--class TPCDSPlanCollector \

--jars /tpcds/spark-sql-perf/target/scala-2.12/spark-sql-perf\_2.12-0.5.1-SNAPSHOT.jar

TPCDSPlanCollector.scala q64-v2.4 /train/q64.csv

生成数据

import com.databricks.spark.sql.perf.tpcds.TPCDSTables

val sqlContext = spark.sqlContext

val scaleFactor = "10" // 100GB

val dsdgenDir = "/tpcds/tpcds-kit/tools"

val dataLocation = "/tpcds/data"

val dbName = "tpcds"

val format = "ORC"

val tables = new TPCDSTables(

sqlContext,

dsdgenDir = dsdgenDir,

scaleFactor = scaleFactor,

useDoubleForDecimal = false,

useStringForDate = false

)

tables.genData(

location = dataLocation,

format = format,

overwrite = true,

partitionTables = true,

clusterByPartitionColumns = false,

filterOutNullPartitionValues = false,

tableFilter = "",

numPartitions = 10

)

**建立外部表并导入数据**

import com.databricks.spark.sql.perf.tpcds.TPCDSTables

val sqlContext = spark.sqlContext

val tables = new TPCDSTables( sqlContext, dsdgenDir = "/tpcds/tpcds-kit/tools", scaleFactor = "10", useDoubleForDecimal = false, useStringForDate = false)

val dataLocation = "/tpcds/data"

val format = "ORC"

val dbName = "tpcds\_new"

sql(s"CREATE DATABASE IF NOT EXISTS $dbName")

tables.createExternalTables( location = dataLocation, format = format, databaseName = dbName, overwrite = true, discoverPartitions = true)

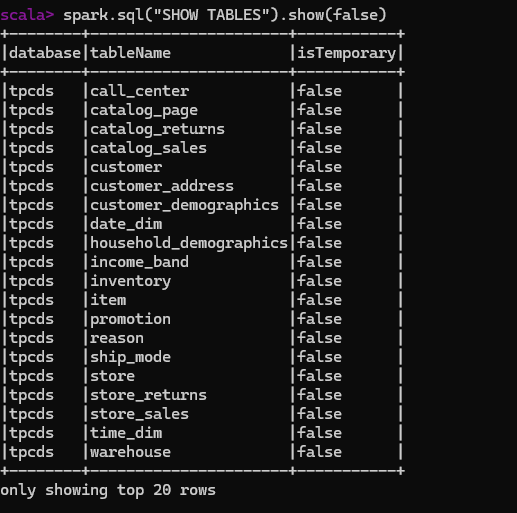
tables.analyzeTables(dbName, analyzeColumns = true)

**查看数据库和数据库中的表格**

spark.sql("SHOW DATABASES").show(false)

spark.sql("USE tpcds")

spark.sql("SHOW TABLES").show(false)



val count = spark.table("tpcds. catalog\_sales ").count()

println(s"Row count: $count")

**收集执行时间（在原版spark中）：**

import com.databricks.spark.sql.perf.tpcds.TPCDS

val tpcds = new TPCDS(sqlContext = spark.sqlContext)

val q64 = tpcds.tpcds2\_4Queries.filter(\_.name == "q64-v2.4").head

val sqlText = q64.sqlText.get

val start = System.nanoTime()

val df = spark.sql(sqlText)

df.collect()

val end = System.nanoTime()

println(s"Query 64 execution time: ${(end - start) / 1e9} Seconds")

**对于spark候选物理计划每次全部执行并收集指标的源码修改：**

修改文件：/spark/sql/core/src/main/scala/org/apache/spark/sql/execution/QueryExecution.scala

修改效果：每次通过Catalyst得到候选物理计划列表后，执行全部在列表中的物理计划，记录每次执行时的指标

存储到/train文件夹

Spark版本3.0.0 Hadoop2.7 Scala

编译命令：./build/mvn -DskipTests clean package -Phadoop-2.7 -Phive -Phive-thriftserver

编译后的包的位置：/spark-3.0.0/sql/core/target/spark-sql\_2.12-3.0.0.jar

复制到本机默认spark位置：cp spark-sql\_2.12-3.0.0.jar /opt/spark-3.0.0-bin-hadoop2.7/jars

原版编译后文件：/Originalsql/ spark-sql\_2.12-3.0.0.jar

修改后的编译后文件：/Newsql/spark-sql\_2.12-3.0.0.jar

spark.sql("USE tpcds\_new")

import com.databricks.spark.sql.perf.tpcds.TPCDS

val tpcds = new TPCDS(sqlContext = spark.sqlContext)

tpcds.tpcds2\_4Queries.foreach { q =>

println(s"===== Executing query: ${q.name} =====")

val sqlText = q.sqlText.get

try {

val df = spark.sql(sqlText)

df.show(20, false)

} catch {

case e: Exception =>

println(s"[Error] Query ${q.name} failed: ${e.getMessage}")

}

}

**笛卡尔积版Spark3.0.0 (收集训练数据)**

./build/mvn -DskipTests clean package -Phadoop-2.7 -Phive -Phive-thriftserver

**启动Spark**

spark-shell \

--conf spark.sql.catalogImplementation=hive \

--jars /tpcds/spark-sql-perf/target/scala-2.12/spark-sql-perf\_2.12-0.5.1-SNAPSHOT.jar

spark.conf.set("spark.sql.adaptive.enabled", "false")

spark.sql("USE tpcds\_new")

import com.databricks.spark.sql.perf.tpcds.TPCDS

val tpcds = new TPCDS(sqlContext = spark.sqlContext)

val q = tpcds.tpcds2\_4Queries.filter(\_.name == "q98-v2.4").head

val sqlText = q.sqlText.get

val df = spark.sql(sqlText)

df.show(20, false)

**查看所有查询名称**

tpcds.tpcds2\_4Queries.map(\_.name).sorted.foreach(println)

q1-v2.4

q10-v2.4

q11-v2.4

q12-v2.4

q13-v2.4

q14a-v2.4

q14b-v2.4

q15-v2.4

q16-v2.4

q17-v2.4

q18-v2.4

q19-v2.4

q2-v2.4

q20-v2.4

q21-v2.4

q22-v2.4

q23a-v2.4

q23b-v2.4

q24a-v2.4

q24b-v2.4

q25-v2.4

q26-v2.4

q27-v2.4

q28-v2.4

q29-v2.4

q30-v2.4

q31-v2.4

q32-v2.4

q33-v2.4

q34-v2.4

q35-v2.4 x

q36-v2.4

q37-v2.4

q38-v2.4

q39a-v2.4

q39b-v2.4

q40-v2.4

q41-v2.4

q42-v2.4

q43-v2.4

q44-v2.4

q45-v2.4

q46-v2.4

q47-v2.4

q48-v2.4

q49-v2.4

q50-v2.4

q51-v2.4

q52-v2.4

q53-v2.4

q54-v2.4

q55-v2.4

q56-v2.4

q57-v2.4

q58-v2.4

q59-v2.4

q60-v2.4

q61-v2.4

q62-v2.4

q63-v2.4

q64-v2.4

q65-v2.4

q66-v2.4

q67-v2.4

q68-v2.4

q69-v2.4

q70-v2.4

q71-v2.4

q72-v2.4

q73-v2.4

q74-v2.4

q75-v2.4

q76-v2.4

q77-v2.4

q78-v2.4

q79-v2.4

q80-v2.4

q81-v2.4

q82-v2.4

q83-v2.4

q84-v2.4

q85-v2.4

q86-v2.4

q87-v2.4

q88-v2.4

q89-v2.4

q90-v2.4

q91-v2.4

q92-v2.4

q93-v2.4

q94-v2.4

q95-v2.4

q96-v2.4

q97-v2.4

q98-v2.4

q99-v2.4

**执行所有查询并记录数据**

// ================== 环境与查询集 ==================

import com.databricks.spark.sql.perf.tpcds.TPCDS

spark.sql("USE tpcds\_new") //数据库名称

val tpcds = new TPCDS(sqlContext = spark.sqlContext)

val queries = tpcds.tpcds2\_4Queries // v2.4 查询集（103条）

// ================== 文件归档工具==================

import java.nio.file.{Files, Paths, StandardOpenOption, DirectoryStream, Path}

import java.nio.file.StandardCopyOption.REPLACE\_EXISTING

import java.nio.charset.StandardCharsets

import java.util.regex.Pattern

import scala.util.Try

import scala.util.control.NonFatal

import scala.jdk.CollectionConverters.\_

object TrainFileArchiver {

private val tmpDir = Paths.get("/tmp")

private val trainDir = Paths.get("/tmp/train")

private val TMP\_JSON = tmpDir.resolve("gnn\_training.jsonl")

private val TMP\_CSV = tmpDir.resolve("gnn\_training.csv")

private val IDX\_JSON = Pattern.compile("""(\d+)\.jsonl""")

private val IDX\_CSV = Pattern.compile("""(\d+)\.csv""")

def ensureTrainDir(): Unit = {

if (Files.notExists(trainDir)) Files.createDirectories(trainDir)

}

/\*\* 删除 /tmp/gnn\_training.jsonl / .csv（如果存在） \*/

def cleanTmpFiles(): Unit = {

Try(Files.deleteIfExists(TMP\_JSON))

Try(Files.deleteIfExists(TMP\_CSV))

}

/\*\* 返回 /tmp/train 下“下一个可用”的整数编号（最大编号+1；若目录不存在则为1） \*/

def nextFreeIndex(): Int = {

ensureTrainDir()

val ds: DirectoryStream[Path] = Files.newDirectoryStream(trainDir)

try {

val used: Set[Int] = ds.asScala.flatMap { p =>

val n = p.getFileName.toString

val m1 = IDX\_JSON.matcher(n)

val m2 = IDX\_CSV.matcher(n)

if (m1.matches()) Some(m1.group(1).toInt)

else if (m2.matches()) Some(m2.group(1).toInt)

else None

}.toSet

if (used.isEmpty) 1 else (used.max + 1)

} finally ds.close()

}

/\*\* 把 /tmp/gnn\_training.\* 移动为 /tmp/train/{index}.jsonl / {index}.csv（存在哪个就移动哪个） \*/

def moveToTrainWithIndex(index: Int): Unit = {

ensureTrainDir()

val jsonDst = trainDir.resolve(s"$index.jsonl")

val csvDst = trainDir.resolve(s"$index.csv")

var movedAny = false

if (Files.exists(TMP\_JSON)) {

Files.move(TMP\_JSON, jsonDst, REPLACE\_EXISTING)

println(s"[move] ${TMP\_JSON} -> ${jsonDst}")

movedAny = true

} else {

println(s"[warn] tmp json not found, skip.")

}

if (Files.exists(TMP\_CSV)) {

Files.move(TMP\_CSV, csvDst, REPLACE\_EXISTING)

println(s"[move] ${TMP\_CSV} -> ${csvDst}")

movedAny = true

} else {

println(s"[warn] tmp csv not found, skip.")

}

if (!movedAny) {

println(s"[warn] nothing moved for index " + index + " (both tmp files missing?)")

}

}

}

// ================== 执行动作==================

def runOne(sqlText: String): Long = {

val df = spark.sql(sqlText)

val t0 = System.nanoTime()

val \_ = df.count()

val t1 = System.nanoTime()

(t1 - t0) / 1000000L

}

// ================== 50 个 Profile 组合==================

case class Profile(name: String, kv: Map[String,String])

val bool = Seq("true","false")

val brThr = Seq("-1","64MB","128MB","256MB")

val shuf = Seq("64","128","200","400","800")

val allProfiles: Seq[Profile] = (for {

aqe <- bool

cbo <- bool

smj <- bool // spark.sql.join.preferSortMergeJoin

bt <- brThr

sp <- shuf

} yield Profile(

s"aqe=$aqe|cbo=$cbo|smj=$smj|br=$bt|sp=$sp",

Map(

"spark.sql.adaptive.enabled" -> aqe,

"spark.sql.cbo.enabled" -> cbo,

"spark.sql.cbo.joinReorder.enabled" -> cbo,

"spark.sql.join.preferSortMergeJoin" -> smj,

"spark.sql.autoBroadcastJoinThreshold" -> bt,

"spark.sql.shuffle.partitions" -> sp

)

)).take(50)

// ================== 断点续跑主循环 ==================

import scala.util.Try

// 计算从哪个全局编号开始（= /tmp/train 下“下一个可用编号”）

val startIndex = TrainFileArchiver.nextFreeIndex()

println(s"\n[resume] detected next free index in /tmp/train = $startIndex (will skip earlier ones)\n")

var globalIdx = 0 // 全局序号：按 profile×query 的顺序从 1 递增

var pIdx = 0

allProfiles.foreach { prof =>

pIdx += 1

// 应用本 profile 的配置

prof.kv.foreach{ case (k,v) => spark.conf.set(k, v) }

println(s"\n[profile $pIdx/${allProfiles.size}] ${prof.name}")

var qIdx = 0

queries.foreach { q =>

qIdx += 1

globalIdx += 1

if (globalIdx < startIndex) {

// 已完成的编号：跳过

if (globalIdx % 100 == 0) println(s"[skip] global=$globalIdx (already done)")

} else {

// 1) 每条查询执行前：删除 /tmp/gnn\_training.jsonl / .csv

TrainFileArchiver.cleanTmpFiles()

// 2) 运行查询（你的源码会把样本追加写入 /tmp/gnn\_training.\*）

val name = q.name

val sql = q.sqlText.get

val ms = Try(runOne(sql)).getOrElse(-1L)

println(f"[ok] global=$globalIdx%05d p=$pIdx%02d q=$qIdx%03d $name runtime=${ms}ms")

// 3) 运行后：把 /tmp/gnn\_training.\* 移到 /tmp/train/{globalIdx}.\*

Try(TrainFileArchiver.moveToTrainWithIndex(globalIdx)) recover {

case NonFatal(e) => println(s"[warn] move failed @global=$globalIdx: ${e.getMessage}")

}

}

}

}

println("\n[done] resume-run finished. New files appended to /tmp/train/.")