

# 了解 SparkSQL 运行计划及优化

# 目 录

<b>1 使用HIVE-CONSOLE 了解运行计划.....</b>	<b>3</b>
1.1 运行环境说明 .....	3
1.1.1 硬软件环境.....	3
1.1.2 集群网络环境.....	3
1.2 编译HIVE.....	4
1.2.1 下载Hive源代码包.....	4
1.2.2 上传Hive源代码包.....	4
1.2.3 解压缩并移动到编译目录.....	4
1.2.4 编译Hive.....	5
1.3 首次运行HIVE-CONSOLE .....	6
1.3.1 获取Spark源代码.....	6
1.3.2 配置/etc/profile环境变量.....	6
1.3.3 运行sbt进行编译.....	6
1.4 使用HIVE-CONSOLE .....	7
1.4.1 启动hive-console .....	7
1.4.2 辅助命令Help和Tab键.....	8
1.4.3 常用操作.....	9
1.4.4 查看查询的Unresolved LogicalPlan .....	10
1.4.5 不同数据源的运行计划.....	11
1.4.6 不同查询的运行计划.....	16
1.4.7 优化.....	19
<b>2 SPARKSQL调优.....</b>	<b>20</b>
2.1 并行性.....	21
2.2 高效的数据格式.....	22
2.2.1 数据本地性.....	23
2.2.2 合适的数据类型.....	23
2.2.3 合适的数据列.....	23
2.2.4 优的数据存储格式.....	23
2.3 内存的使用.....	24
2.4 合适的TASK .....	24
2.5 其他的一些建议.....	25

# 了解 SparkSQL 运行计划及优化

## 1 使用 Hive-Console 了解运行计划

前面介绍了 SparkSQL 的运行过程，罗列了很多概念很抽象，比如 Unresolved LogicPlan、LogicPlan、PhysicalPlan，下面介绍一个工具 hive/console，来加深对 SparkSQL 的运行计划的理解。

**【注】以下实验环境为第二课《Spark 编译与部署》搭建 hadoop1 一台机器**

### 1.1 运行环境说明

#### 1.1.1 硬软件环境

- 主机操作系统：Windows 64 位，双核 4 线程，主频 2.2G，10G 内存
- 虚拟软件：VMware® Workstation 9.0.0 build-812388
- 虚拟机操作系统：CentOS6.5 64 位，单核
- 虚拟机运行环境：
  - JDK：1.7.0\_55 64 位
  - Hadoop：2.2.0（需要编译为 64 位）
  - Scala：2.10.4
  - Spark：1.1.0（需要编译）
  - Hive：0.13.1（源代码编译，参见 1.2）

#### 1.1.2 集群网络环境

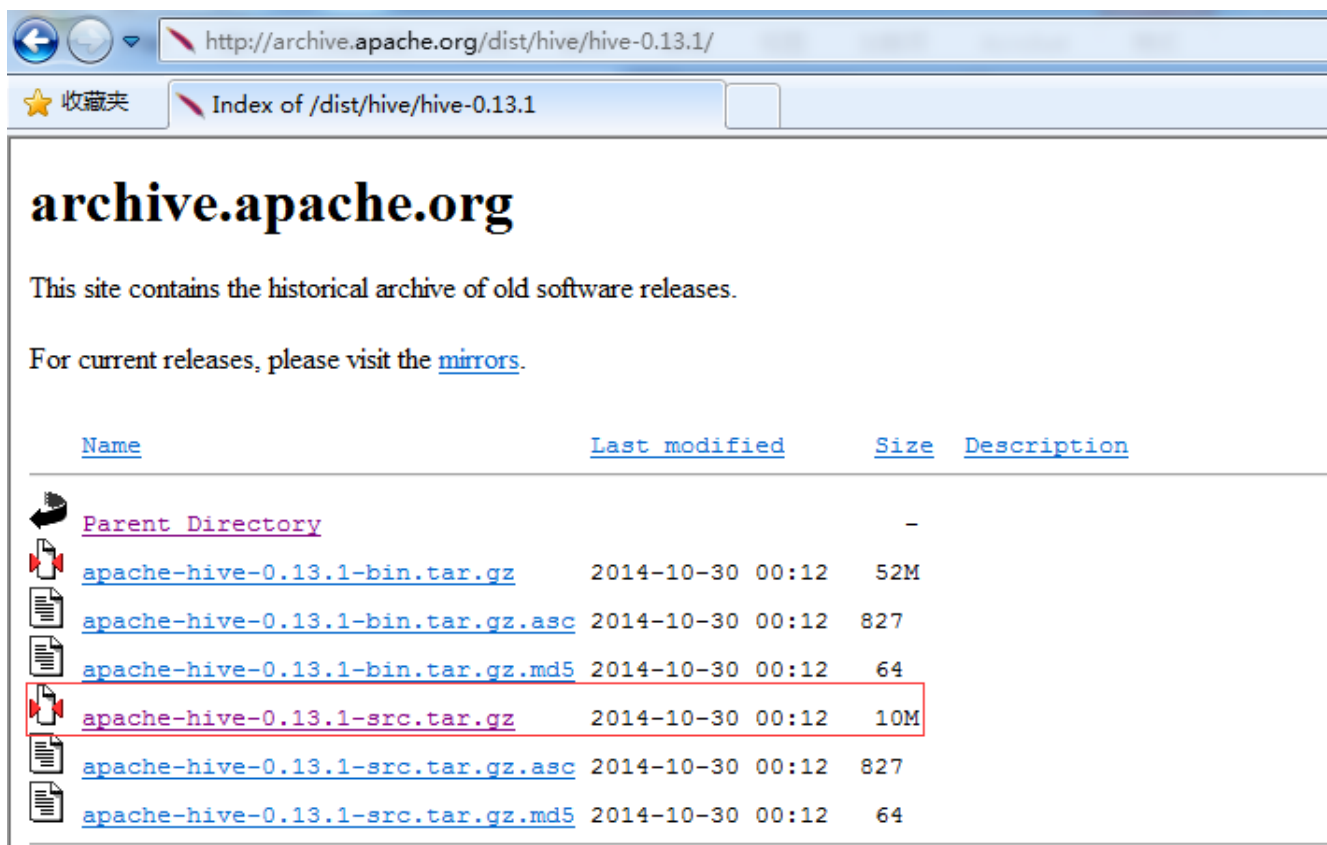
本次实验环境只需要 hadoop1 一台机器即可，网络环境配置如下：

序号	IP 地址	机器名	类型	用户名	目录
1	192.168.0.61	hadoop1	NN/DN	hadoop	/app 程序所在路径 /app/scala-... /app/hadoop /app/complied

## 1.2 编译 Hive

### 1.2.1 下载 Hive 源代码包

这里选择下载的版本为hive-0.13.1，这个版本需要到apache的归档服务器下载，下载地址：<http://archive.apache.org/dist/hive/hive-0.13.1/>，选择 apache-hive-0.13.1-src.tar.gz 文件进行下载：



### 1.2.2 上传 Hive 源代码包

把下载的 hive-0.13.0.tar.gz 安装包，使用 SSH Secure File Transfer 工具(参见第 2 课《Spark 编译与部署（上）--基础环境搭建》1.3.1 介绍）上传到/home/hadoop/upload 目录下。

### 1.2.3 解压缩并移动到编译目录

到上传目录下，用如下命令解压缩 hive 安装文件：

```
$cd /home/hadoop/upload
```

```
$tar -xzf apache-hive-0.13.1-src.tar.gz
```

改名并移动到/app/complied 目录下：

```
$sudo mv apache-hive-0.13.1-src /app/complied/hive-0.13.1-src
```

```
$ll /app/complied
```

## 1.2.4 编译 Hive

编译 Hive 源代码的时候,需要从网上下载依赖包,所以整个编译过程机器必须保证在联网状态。  
编译执行如下脚本:

```
$cd /app/complied/hive-0.13.1-src/
```

```
$export MAVEN_OPTS="-Xmx2g -XX:MaxPermSize=512M -XX:ReservedCodeCacheSize=512m"
```

```
$mvn -Phadoop-2,dist -Dmaven.test.skip=true clean package
```

```
[INFO] Building tar: /app/complied/hive-0.13.1-src/packaging/target/apache-hive-0.13.1-bin.tar.gz
[INFO] Building tar: /app/complied/hive-0.13.1-src/packaging/target/apache-hive-0.13.1-src.tar.gz
[INFO] -----
[INFO] Reactor Summary:
[INFO]
[INFO] Hive ..... SUCCESS [7.440s]
[INFO] Hive Ant Utilities ..... SUCCESS [3.653s]
[INFO] Hive Shims Common ..... SUCCESS [2.608s]
[INFO] Hive Shims 0.20 ..... SUCCESS [1.778s]
[INFO] Hive Shims Secure Common ..... SUCCESS [2.630s]
[INFO] Hive Shims 0.20S ..... SUCCESS [0.932s]
[INFO] Hive Shims 0.23 ..... SUCCESS [4.317s]
[INFO] Hive Shims ..... SUCCESS [0.586s]
[INFO] Hive Common ..... SUCCESS [3.834s]
[INFO] Hive Serde ..... SUCCESS [5.539s]
[INFO] Hive Metastore ..... SUCCESS [22.386s]
[INFO] Hive Query Language ..... SUCCESS [59.027s]
[INFO] Hive Service ..... SUCCESS [5.029s]
[INFO] Hive JDBC ..... SUCCESS [1.738s]
[INFO] Hive Beeline ..... SUCCESS [1.396s]
[INFO] Hive CLI ..... SUCCESS [1.624s]
[INFO] Hive Contrib ..... SUCCESS [0.925s]
[INFO] Hive HBase Handler ..... SUCCESS [2.904s]
[INFO] Hive HCatalog ..... SUCCESS [2:38.359s]
[INFO] Hive HCatalog Core ..... SUCCESS [24.243s]
[INFO] Hive HCatalog Pig Adapter ..... SUCCESS [20.378s]
[INFO] Hive HCatalog Server Extensions ..... SUCCESS [2:37.645s]
[INFO] Hive HCatalog Webcat Java Client ..... SUCCESS [0.912s]
[INFO] Hive HCatalog Webcat ..... SUCCESS [1:36.143s]
[INFO] Hive HCatalog HBase Storage Handler ..... SUCCESS [9.823s]
[INFO] Hive HCatalog Streaming ..... SUCCESS [1.172s]
[INFO] Hive HWI ..... SUCCESS [1.369s]
[INFO] Hive ODBC ..... SUCCESS [0.524s]
[INFO] Hive Shims Aggregator ..... SUCCESS [0.115s]
[INFO] Hive TestUtils ..... SUCCESS [7.724s]
[INFO] Hive Packaging ..... SUCCESS [1:32.914s]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 11:41.289s
[INFO] Finished at: Tue Jul 28 23:04:56 CST 2015
[INFO] Final Memory: 119M/996M
[INFO] -----
```

在编译过程中可能出现速度慢或者中断,可以再次启动编译,编译程序会在上次的编译中断处继续进行编译,整个编译过程耗时与网速紧密相关,网速较快的情况需要 1 个小时左右(上图的时间是多次编译后最后成功的界面)。最终编译的结果为 \$HIVE\_HOME/packaging/target/apache-hive-0.13.1-bin.tar.gz

通过如下命令查看最终编译完成整个目录大小,可以看到大小为 353.6M 左右

```
$du -s /app/complied/hive-0.13.1-src
```

```
[hadoop@hadoop1 hive-0.13.1-src]$ du -s /app/complied/hive-0.13.1-src/
353600 /app/complied/hive-0.13.1-src/
[hadoop@hadoop1 hive-0.13.1-src]$
```



**【注】**已经编译好的 Hive 包在本系列配套资源/install/6.hive-0.13.1-src.tar.gz,读者直接使用

## 1.3 首次运行 hive-console

### 1.3.1 获取 Spark 源代码

由于首次运行hive-console需要在Spark源代码进行编译，关于Spark源代码的获取可以参考第二课《Spark 编译与部署（下）--Spark 编译安装》方式进行获取，连接地址为 <http://spark.apache.org/downloads.html>，获取源代码后把Spark源代码移动到/app/complied目录，并命名为spark-1.1.0-hive

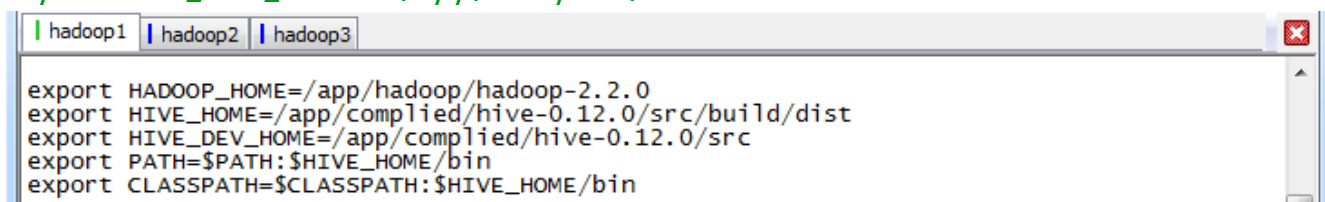
### 1.3.2 配置/etc/profile 环境变量

第一步 使用如下命令打开/etc/profile 文件：

```
$sudo vi /etc/profile
```

第二步 设置如下参数：

```
export HADOOP_HOME=/app/hadoop/hadoop-2.2.0  
export HIVE_HOME=/app/complied/hive-0.13.1-src  
export HIVE_DEV_HOME=/app/complied/hive-0.13.1-src
```



第三步 生效配置并验证

```
$sudo vi /etc/profile  
$echo $HIVE_DEV_HOME
```

### 1.3.3 运行 sbt 进行编译

运行 hive/console 不需要启动 Spark，需要进入到 Spark 根目录下使用 sbt/sbt hive/console 进行首次运行编译，编译以后下次可以直接启动。编译 Spark 源代码的时候，需要从网上下载依赖包，所以整个编译过程机器必须保证在联网状态。编译命令如下：

```
$cd /app/complied/spark-1.1.0-hive  
$sbt/sbt hive/console
```

```
[hadoop@hadoop1 spark-1.1.0-hive]$ sbt/sbt hive/console
Using /usr/lib/java/jdk1.7.0_55 as default JAVA_HOME.
Note, this will be overridden by -java-home if it is set.
Attempting to fetch sbt
##### 100.0%
Launching sbt from sbt/sbt-launch-0.13.5.jar
[info] Loading project definition from /app/complied/spark-1.1.0-hive/project/project
[info] Updating {file:/app/complied/spark-1.1.0-hive/project/project/}spark-1.1.0-hive-build-build...
[info] Resolving org.fusesource.jansi#jansi;1.4 ...
[info] Done updating.
[info] Compiling 1 Scala source to /app/complied/spark-1.1.0-hive/project/project/target/scala-2.10/sbt-0.13/classes...
[warn] there were 2 deprecation warning(s); re-run with -deprecation for details
[info] one warning found
[info] Loading project definition from /home/hadoop/.sbt/0.13/staging/ec3aa8f39111944cc5f2/sbt-pom-reader/project
[warn] Multiple resolvers having different access mechanism configured with same name 'sbt-plugin-releases'. To avoid conflict, Remove duplicate project resolvers ('resolvers') or rename publishing resolver ('publishTo').
[info] Loading project definition from /app/complied/spark-1.1.0-hive/project
[info] Updating {file:/app/complied/spark-1.1.0-hive/project/}spark-style...
[info] Resolving org.fusesource.jansi#jansi;1.4 ...
[info] Done updating.
[info] Updating {file:/app/complied/spark-1.1.0-hive/project/}plugins...
[info] Resolving org.fusesource.jansi#jansi;1.4 ...
[info] Done updating.
[info] Compiling 1 Scala source to /app/complied/spark-1.1.0-hive/project/spark-style/target/scala-2.10/classes...
[info] Compiling 3 Scala sources to /app/complied/spark-1.1.0-hive/project/target/scala-2.10/sbt-0.13/classes...
[warn] there were 1 deprecation warning(s); re-run with -deprecation for details
[warn] one warning found
[info] Set current project to spark-parent (in build file:/app/complied/spark-1.1.0-hive/)
[info] Updating {file:/app/complied/spark-1.1.0-hive/}core...
[info] Resolving org.fusesource.jansi#jansi;1.4 ...
[info] Done updating.
[info] Updating {file:/app/complied/spark-1.1.0-hive/}catalyst...
[info] Resolving org.fusesource.jansi#jansi;1.4 ...
[info] Done updating.
[info] Updating {file:/app/complied/spark-1.1.0-hive/}sql...
[info] Resolving org.fusesource.jansi#jansi;1.4 ...
[info] Done updating.
[info] Updating {file:/app/complied/spark-1.1.0-hive/}hive...
[info] Resolving org.fusesource.jansi#jansi;1.4 ...
[info] Done updating.
[info] Compiling 389 Scala sources and 28 Java sources to /app/complied/spark-1.1.0-hive/core/target/scala-2.10/classes...
```

编译时间会很长，在编译过程中可能出现速度慢或者中断，可以再次启动编译，编译程序会在上次的编译中断处继续进行编译，整个编译过程耗时与网速紧密相关。

```
[warn] Note: /app/complied/spark-1.1.0-hive/core/src/main/java/org/apache/spark/api/java/JavaSparkContextVarargWorkaround.java uses unchecked or unsafe operations.
[warn] Note: Recompile with -Xlint:unchecked for details.
[info] Compiling 62 Scala sources to /app/complied/spark-1.1.0-hive/sql/catalyst/target/scala-2.10/classes...
[info] Compiling 45 Scala sources and 39 Java sources to /app/complied/spark-1.1.0-hive/sql/core/target/scala-2.10/classes...
[info] Compiling 18 Scala sources and 1 Java source to /app/complied/spark-1.1.0-hive/sql/hive/target/scala-2.10/classes...
[warn] there were 9 deprecation warning(s); re-run with -deprecation for details
[warn] one warning found
[info] Starting scala interpreter...
[info]
import org.apache.spark.sql.catalyst.analysis._
import org.apache.spark.sql.catalyst.dsl._
import org.apache.spark.sql.catalyst.errors._
import org.apache.spark.sql.catalyst.expressions._
import org.apache.spark.sql.catalyst.plans.logical._
import org.apache.spark.sql.catalyst.rules._
import org.apache.spark.sql.catalyst.types._
import org.apache.spark.sql.catalyst.util._
import org.apache.spark.sql.execution._
import org.apache.spark.sql.hive._
import org.apache.spark.sql.hive.test.TestHive._
import org.apache.spark.sql.parquet.ParquetTestdata
welcome to scala version 2.10.4 (Java HotSpot(TM) 64-Bit Server VM, Java 1.7.0_55).
Type in expressions to have them evaluated.
Type :help for more information.
```

通过如下命令查看最终编译完成整个目录大小，可以看到大小为 267.9M 左右

```
$du -s /app/complied/spark-1.1.0-hive
```

```
[hadoop@hadoop1 ~]$ du -s /app/complied/spark-1.1.0-hive
267912 /app/complied/spark-1.1.0-hive
[hadoop@hadoop1 ~]$
```

**【注】**已经编译好的 Spark for hive-console 包在本系列配套资源 /install/6.spark-1.1.0-hive.tar.gz，可直接使用

## 1.4 使用 hive-console

### 1.4.1 启动 hive-console

进入到 spark 根目录下，使用如下命令启动 hive-console

```
$cd /app/complied/spark-1.1.0-hive
```

```
$sbt/sbt hive/console
```



```
hadoop@hadoop1 spark-1.1.0-hive]$ sbt/sbt hive/console
using /usr/lib/java/jdk1.7.0_55 as default JAVA_HOME.
Note, this will be overridden by -java-home if it is set.
[info] Loading project definition from /app/compiled/spark-1.1.0-hive/project/project
[info] Loading project definition from /home/hadoop/.sbt/0.13/staging/ec3aa8f39111944cc5f2/sbt-pom-reader/project
[warn] Multiple resolvers having different access mechanism configured with same name 'sbt-plugin-releases'. To a
solvers`) or rename publishing resolver ('publishTo').
[info] Loading project definition from /app/compiled/spark-1.1.0-hive/project
[info] Set current project to spark-parent (in build file:/app/compiled/spark-1.1.0-hive/)
[info] Starting scala interpreter...
[info]
import org.apache.spark.sql.catalyst.analysis._
import org.apache.spark.sql.catalyst.dsl._
import org.apache.spark.sql.catalyst.errors._
import org.apache.spark.sql.catalyst.expressions._
import org.apache.spark.sql.catalyst.plans.logical._
import org.apache.spark.sql.catalyst.rules._
import org.apache.spark.sql.catalyst.types._
import org.apache.spark.sql.catalyst.util._
import org.apache.spark.sql.execution._
import org.apache.spark.sql.hive._
import org.apache.spark.sql.hive.test.TestHive._
import org.apache.spark.sql.parquet.ParquetTestData
welcome to Scala version 2.10.4 (Java HotSpot(TM) 64-Bit Server VM, Java 1.7.0_55).
Type in expressions to have them evaluated.
Type :help for more information.
scala> █
```

## 1.4.2 辅助命令 Help 和 Tab 键

可以使用:help 查看帮助内容

*scala>:help*

```
hadoop@hadoop1 | hadoop2 | hadoop3
scala> :help
All commands can be abbreviated, e.g. :he instead of :help.
Those marked with a * have more detailed help, e.g. :help imports.

:cp <path>                add a jar or directory to the classpath
:help [command]            print this summary or command-specific help
:history [num]             show the history (optional num is commands to show)
:h? <string>              search the history
:imports [name name ...]  show import history, identifying sources of names
:implicit [-v]            show the implicits in scope
:javap <path|class>       disassemble a file or class name
:load <path>              load and interpret a Scala file
:paste                    enter paste mode: all input up to ctrl-D compiled together
:power                    enable power user mode
:quit                     exit the interpreter
:replay                   reset execution and replay all previous commands
:reset                    reset the repl to its initial state, forgetting all session entries
:sh <command line>        run a shell command (result is implicitly => List[String])
:silent                   disable/enable automatic printing of results
:type [-v] <expr>         display the type of an expression without evaluating it
:warnings                  show the suppressed warnings from the most recent line which had a

scala>
```

可以使用 tab 键查看所有可使用命令、函数

```
hadoop@hadoop1 | hadoop2 | hadoop3
scala>
<init>                                DslAttribute                        DslExpression
DslString                             DslSymbol                          ParquetTestData
SqlCmd                                analyzer                           analyzer
applySchema                           applySchemaToPythonRDD             approxCountDistinct
args                                  autoBroadcastJoinThreshold          avg
binaryToLiteral                       booleanToLiteral                    byteToLiteral
cacheTable                            cacheTables                         catalog
classOf                               clear                               clone
codegenEnabled                        columnBatchSize                     configure
convertMetastoreParquet               count                               countDistinct
createParquetFile                     createschemaRDD                     createTable
decimalToLiteral                      defaultSizeInBytes                  describedTable
dialect                               doubleToLiteral                     emptyResult
eq                                     equals                              executePlan
executesql                            execution                           finalize
first                                 floatToLiteral                      functionRegistry
getAllConfs                           getClass                           getConf
getHiveFile                           hashCode                           hiveDevHome
hiveFilesTemp                         hiveHome                           hivePlanner
hiveQTestUtilTables                  hiveconf                            hiveql
hql                                    inRepoTests                         intToLiteral
isCached                              isParquetBinaryAsString             isTraceEnabled
```



### 1.4.3 常用操作

首先定义 Person 类，在该类中定义 name、age 和 state 三个列，然后把该类注册为 people 表并装载数据，最后通过查询到数据存放到 query 中

```
scala> case class Person(name:String, age:Int, state:String)
```

```
scala> sparkContext.parallelize(Person("Michael",29,"CA")::Person("Andy",30,"NY")::Person("Justin",19,"CA")::Person("Justin",25,"CA")::Nil).registerTempTable("people")
```

```
scala> case class Person(name:String, age:Int, state:String)
defined class Person

scala> sparkContext.parallelize(Person("Michael",29,"CA")::Person("Andy",30,"NY")::Person("Justin",19,"CA")::Person("Justin",25,"CA")::Nil).registerTempTable("people")
15/07/28 17:02:25 INFO spark.SecurityManager: Changing view acls to: hadoop,
15/07/28 17:02:25 INFO spark.SecurityManager: Changing modify acls to: hadoop,
15/07/28 17:02:25 INFO spark.SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(hadoop, ); users with modify permissions: Set(hadoop, )
15/07/28 17:02:26 INFO slf4j.Slf4jLogger: slf4jLogger started
15/07/28 17:02:26 INFO Remoting: Starting remoting
15/07/28 17:02:28 INFO Remoting: Remoting started; listening on addresses : [akka.tcp://sparkDriver@hadoop1:46330]
15/07/28 17:02:28 INFO Remoting: Remoting now listens on addresses: [akka.tcp://sparkDriver@hadoop1:46330]
15/07/28 17:02:28 INFO util.Utils: Successfully started service 'sparkDriver' on port 46330.
15/07/28 17:02:28 INFO spark.SparkEnv: Registering MapOutputTracker
15/07/28 17:02:28 INFO spark.SparkEnv: Registering BlockManagerMaster
15/07/28 17:02:28 INFO storage.DiskBlockManager: Created local directory at /tmp/spark-local-20150228170228-1a89
15/07/28 17:02:29 INFO util.Utils: Successfully started service 'Connection manager for block manager' on port 53178.
15/07/28 17:02:29 INFO network.ConnectionManager: Bound socket to port 53178 with id = ConnectionManagerId(hadoop1,53178)
15/07/28 17:02:29 INFO storage.MemoryStore: MemoryStore started with capacity 1069.1 MB
15/07/28 17:02:29 INFO storage.BlockManagerMaster: Trying to register BlockManager
15/07/28 17:02:29 INFO storage.BlockManagerMasterActor: Registering block manager hadoop1:53178 with 1069.1 MB RAM
15/07/28 17:02:29 INFO storage.BlockManagerMaster: Registered BlockManager
15/07/28 17:02:29 INFO spark.HttpFileServer: HTTP File server directory is /tmp/spark-1f0a06e0-5548-4009-aaac-0c78fe12c164
15/07/28 17:02:29 INFO spark.HttpServer: Starting HTTP server
15/07/28 17:02:30 INFO server.Server: jetty-8.1.14.v20131031
15/07/28 17:02:30 INFO server.AbstractConnector: Started SocketConnector@0.0.0.0:33998
15/07/28 17:02:30 INFO util.Utils: Successfully started service 'HTTP file server' on port 33998.
15/07/28 17:02:32 INFO server.Server: jetty-8.1.14.v20131031
15/07/28 17:02:32 INFO server.AbstractConnector: Started SelectChannelConnector@0.0.0.0:4040
15/07/28 17:02:32 INFO util.Utils: Successfully started service 'SparkUI' on port 4040.
15/07/28 17:02:32 INFO ui.SparkUI: Started SparkUI at http://hadoop1:4040
15/07/28 17:02:34 INFO util.AkkaUtils: Connecting to HeartbeatReceiver: akka.tcp://sparkDriver@hadoop1:46330/user/HeartbeatReceiver
```

```
scala> val query= sql("select * from people")
```

```
scala> val query= sql("select * from people")
15/07/29 21:46:55 INFO parse.ParseDriver: Parsing command: select * from people
15/07/29 21:46:56 INFO parse.ParseDriver: Parse Completed
query: org.apache.spark.sql.SchemaRDD =
SchemaRDD[3] at RDD at SchemaRDD.scala:103
== Query Plan ==
== Physical Plan ==
ExistingRDD [name#0,age#1,state#2], MapPartitionsRDD[1] at mapPartitions at basicoperators.scala:208
```

#### 1.4.3.1 查看查询的 schema

```
scala> query.printSchema
```

```
scala> query.collect()
```

```
scala> query.printSchema
root
|-- name: string (nullable = true)
|-- age: integer (nullable = false)
|-- state: string (nullable = true)

scala> query.collect()
15/07/29 21:49:09 INFO spark.SparkContext: Starting job: collect at SparkPlan.scala:85
15/07/29 21:49:09 INFO scheduler.DAGScheduler: Got job 6 (collect at SparkPlan.scala:85) with 1 output partitions (allowLocal=false)
15/07/29 21:49:09 INFO scheduler.DAGScheduler: Final stage: Stage 6(collect at SparkPlan.scala:85)
15/07/29 21:49:09 INFO scheduler.DAGScheduler: Parents of final stage: List()
15/07/29 21:49:09 INFO scheduler.DAGScheduler: Missing parents: List()
15/07/29 21:49:09 INFO scheduler.DAGScheduler: Submitting Stage 6 (MappedRDD[10] at map at SparkPlan.scala:85), which has no missing parents
15/07/29 21:49:09 INFO storage.MemoryStore: ensureFreeSpace(2168) called with curMem=13008, maxMem=1111794647
15/07/29 21:49:09 INFO storage.MemoryStore: Block broadcast_6 stored as values in memory (estimated size 2.1 KB, free 1060.3 MB)
15/07/29 21:49:09 INFO scheduler.DAGScheduler: Submitting 1 missing tasks from Stage 6 (MappedRDD[10] at map at SparkPlan.scala:85)
15/07/29 21:49:09 INFO scheduler.TaskSchedulerImpl: Adding task set 6.0 with 1 tasks
15/07/29 21:49:09 INFO scheduler.TaskSetManager: Starting task 0.0 in stage 6.0 (TID 6, localhost, PROCESS_LOCAL, 1485 bytes)
15/07/29 21:49:09 INFO executor.Executor: Running task 0.0 in stage 6.0 (TID 6)
15/07/29 21:49:09 INFO executor.Executor: Finished task 0.0 in stage 6.0 (TID 6). 963 bytes result sent to driver
15/07/29 21:49:09 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 6.0 (TID 6) in 8 ms on localhost (1/1)
15/07/29 21:49:09 INFO scheduler.DAGScheduler: Stage 6 (collect at SparkPlan.scala:85) finished in 0.009 s
15/07/29 21:49:09 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 6.0, whose tasks have all completed, from pool
15/07/29 21:49:09 INFO spark.SparkContext: Job finished: collect at SparkPlan.scala:85, took 0.019672172 s
res9: Array[org.apache.spark.sql.Row] = Array([Michael,29,CA], [Andy,30,NY], [Justin,19,CA], [Justin,25,CA])
```

#### 1.4.3.2 查看查询的整个运行计划

```
scala> query.queryExecution
```

```

hadoop1 | hadoop2 | hadoop3
scala> query.queryExecution
res10: query.sqlContext.QueryExecution =
== Parsed Logical Plan ==
Project [*]
  UnresolvedRelation None, people, None

== Analyzed Logical Plan ==
Project [name#0,age#1,state#2]
  SparkLogicalPlan (ExistingRdd [name#0,age#1,state#2], MapPartitionsRDD[1] at mapPartitions at basicoperators.scala:208)

== Optimized Logical Plan ==
SparkLogicalPlan (ExistingRdd [name#0,age#1,state#2], MapPartitionsRDD[1] at mapPartitions at basicoperators.scala:208)

== Physical Plan ==
ExistingRdd [name#0,age#1,state#2], MapPartitionsRDD[1] at mapPartitions at basicoperators.scala:208

Code Generation: false
== RDD ==

```

## 1.4.4 查看查询的 Unresolved LogicalPlan

*scala> query.queryExecution.logical*

```

hadoop1 | hadoop2 | hadoop3
scala> query.queryExecution.logical
res11: org.apache.spark.sql.catalyst.plans.logical.LogicalPlan =
Project [*]
  UnresolvedRelation None, people, None

```

### 1.4.4.1 查看查询的 Analyzed LogicalPlan

*scala> query.queryExecution.analyzed*

```

hadoop1 | hadoop2 | hadoop3
scala> query.queryExecution.analyzed
res12: org.apache.spark.sql.catalyst.plans.logical.LogicalPlan =
Project [name#0,age#1,state#2]
  SparkLogicalPlan (ExistingRdd [name#0,age#1,state#2], MapPartitionsRDD[1] at mapPartitions at basicoperators.scala:208)

```

### 1.4.4.2 查看优化后的 LogicalPlan

*scala> query.queryExecution.optimizedPlan*

```

hadoop1 | hadoop2 | hadoop3
scala> query.queryExecution.optimizedPlan
res13: org.apache.spark.sql.catalyst.plans.logical.LogicalPlan =
SparkLogicalPlan (ExistingRdd [name#0,age#1,state#2], MapPartitionsRDD[1] at mapPartitions at basicoperators.scala:208)

```

### 1.4.4.3 查看物理计划

*scala> query.queryExecution.sparkPlan*

```

hadoop1 | hadoop2 | hadoop3
scala> query.queryExecution.sparkPlan
res14: org.apache.spark.sql.execution.SparkPlan =
ExistingRdd [name#0,age#1,state#2], MapPartitionsRDD[1] at mapPartitions at basicoperators.scala:208

```

### 1.4.4.4 查看 RDD 的转换过程

*scala> query.toDebugString*

```
scala> query.toDebugString
res8: String =
(1) SchemaRDD[3] at RDD at SchemaRDD.scala:103
== Query Plan ==
== Physical Plan ==
ExistingRDD [name#0,age#1,state#2], MapPartitionsRDD[1] at mapPartitions at basicoperators.scala:208
| MappedRDD[4] at map at HiveContext.scala:360
| MapPartitionsRDD[1] at mapPartitions at basicoperators.scala:208
| ParallelCollectionRDD[0] at parallelize at <console>:42
scala>
```

## 1.4.5 不同数据源的运行计划

上面常用操作里介绍了源自 RDD 的数据，SparkSQL 也可以源自多个数据源：jsonFile、parquetFile 和 Hive 等。

### 1.4.5.1 读取 Json 格式数据

第一步 Json 测试数据

Json 文件支持嵌套表，SparkSQL 也可以读入嵌套表，如下面形式的 Json 数据，可以使用 jsonFile 读入 SparkSQL。该文件可以在配套资源/data/class6 中找到，在以下测试中把文件放到 /home/hadoop/upload/class6 路径中

```
{
  "fullname": "Sean Kelly",
  "org": "SK Consulting",
  "emailaddrs": [
    {"type": "work", "value": "kelly@seankelly.biz"},
    {"type": "home", "pref": 1, "value": "kelly@seankelly.tv"}
  ],
  "telephones": [
    {"type": "work", "pref": 1, "value": "+1 214 555 1212"},
    {"type": "fax", "value": "+1 214 555 1213"},
    {"type": "mobile", "value": "+1 214 555 1214"}
  ],
  "addresses": [
    {"type": "work", "format": "us",
      "value": "1234 Main StnSpringfield, TX 78080-1216"},
    {"type": "home", "format": "us",
      "value": "5678 Main StnSpringfield, TX 78080-1316"}
  ],
  "urls": [
    {"type": "work", "value": "http://seankelly.biz/"},
    {"type": "home", "value": "http://seankelly.tv/"}
```

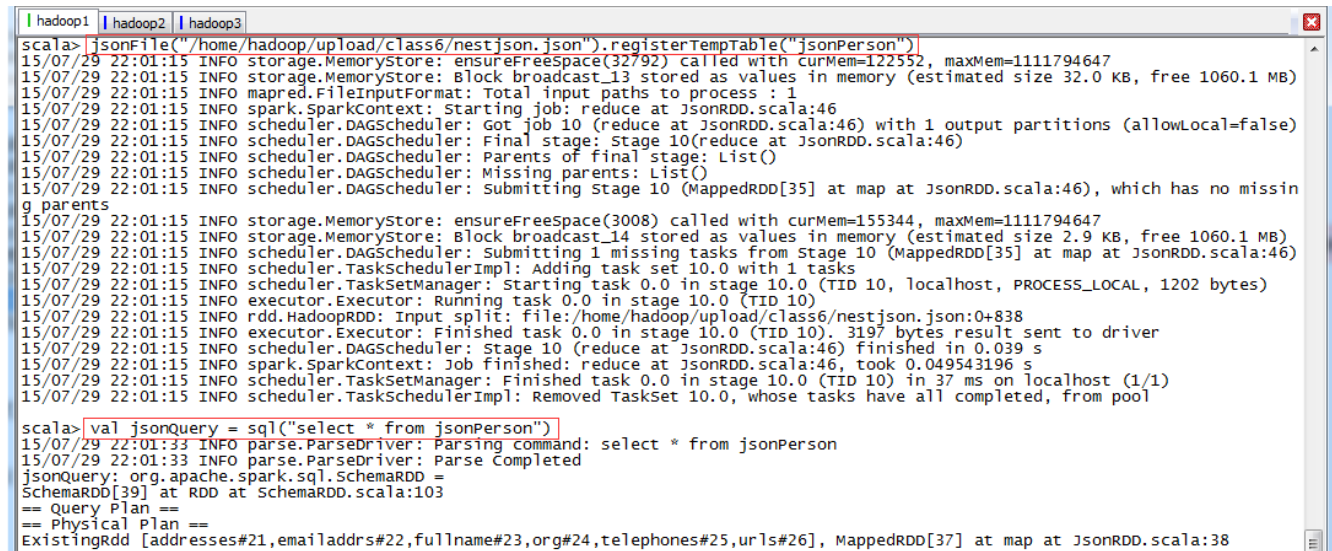
```
}  
}
```

## 第二步 读入 Json 数据

使用 jsonFile 读入数据并注册成表 jsonPerson，然后定义一个查询 jsonQuery

```
scala> jsonFile("/home/hadoop/upload/class6/nestjson.json").registerTempTable("jsonPerson")
```

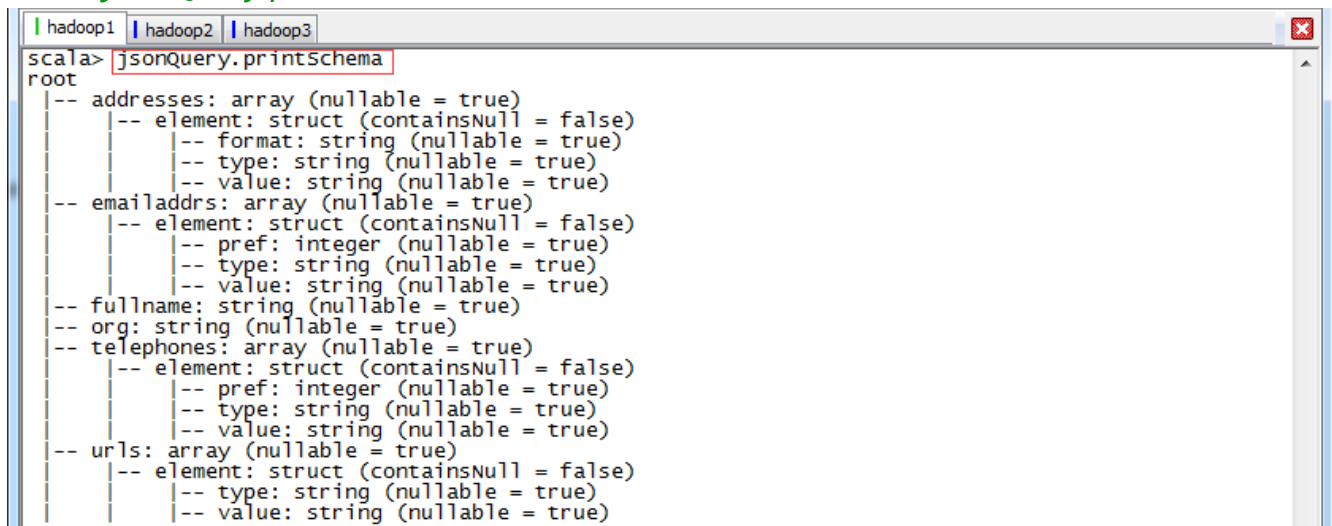
```
scala> val jsonQuery = sql("select * from jsonPerson")
```



```
scala> jsonFile("/home/hadoop/upload/class6/nestjson.json").registerTempTable("jsonPerson")  
15/07/29 22:01:15 INFO storage.MemoryStore: ensureFreeSpace(32792) called with curMem=122552, maxMem=1111794647  
15/07/29 22:01:15 INFO storage.MemoryStore: Block broadcast_13 stored as values in memory (estimated size 32.0 KB, free 1060.1 MB)  
15/07/29 22:01:15 INFO mapred.FileInputFormat: Total input paths to process : 1  
15/07/29 22:01:15 INFO spark.SparkContext: Starting job: reduce at JsonRDD.scala:46  
15/07/29 22:01:15 INFO scheduler.DAGScheduler: Got job 10 (reduce at JsonRDD.scala:46) with 1 output partitions (allowLocal=false)  
15/07/29 22:01:15 INFO scheduler.DAGScheduler: Final stage: Stage 10(reduce at JsonRDD.scala:46)  
15/07/29 22:01:15 INFO scheduler.DAGScheduler: Parents of final stage: List()  
15/07/29 22:01:15 INFO scheduler.DAGScheduler: Missing parents: List()  
15/07/29 22:01:15 INFO scheduler.DAGScheduler: Submitting Stage 10 (MappedRDD[35] at map at JsonRDD.scala:46), which has no missing parents  
15/07/29 22:01:15 INFO storage.MemoryStore: ensureFreeSpace(3008) called with curMem=155344, maxMem=1111794647  
15/07/29 22:01:15 INFO storage.MemoryStore: Block broadcast_14 stored as values in memory (estimated size 2.9 KB, free 1060.1 MB)  
15/07/29 22:01:15 INFO scheduler.DAGScheduler: Submitting 1 missing tasks from Stage 10 (MappedRDD[35] at map at JsonRDD.scala:46)  
15/07/29 22:01:15 INFO scheduler.TaskSchedulerImpl: Adding task set 10.0 with 1 tasks  
15/07/29 22:01:15 INFO scheduler.TaskSetManager: Starting task 0.0 in stage 10.0 (TID 10, localhost, PROCESS_LOCAL, 1202 bytes)  
15/07/29 22:01:15 INFO executor.Executor: Running task 0.0 in stage 10.0 (TID 10)  
15/07/29 22:01:15 INFO rdd.HadoopRDD: Input split: file:/home/hadoop/upload/class6/nestjson.json:0+838  
15/07/29 22:01:15 INFO executor.Executor: Finished task 0.0 in stage 10.0 (TID 10). 3197 bytes result sent to driver  
15/07/29 22:01:15 INFO scheduler.DAGScheduler: Stage 10 (reduce at JsonRDD.scala:46) finished in 0.039 s  
15/07/29 22:01:15 INFO spark.SparkContext: Job finished: reduce at JsonRDD.scala:46, took 0.049543196 s  
15/07/29 22:01:15 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 10.0 (TID 10) in 37 ms on localhost (1/1)  
15/07/29 22:01:15 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 10.0, whose tasks have all completed, from pool  
  
scala> val jsonQuery = sql("select * from jsonPerson")  
15/07/29 22:01:33 INFO parse.ParserDriver: Parsing command: select * from jsonPerson  
15/07/29 22:01:33 INFO parse.ParserDriver: Parse Completed  
jsonQuery: org.apache.spark.sql.SchemaRDD =  
SchemaRDD[39] at RDD at SchemaRDD.scala:103  
== Query Plan ==  
== Physical Plan ==  
ExistingRDD [addresses#21,emailaddrs#22,fullname#23,org#24,telephones#25,urls#26], MappedRDD[37] at map at JsonRDD.scala:38
```

## 第三步 查看 jsonQuery 的 schema

```
scala> jsonQuery.printSchema
```



```
scala> jsonQuery.printSchema  
root  
|-- addresses: array (nullable = true)  
|   |-- element: struct (containsNull = false)  
|   |   |-- format: string (nullable = true)  
|   |   |-- type: string (nullable = true)  
|   |   |-- value: string (nullable = true)  
|   |-- emailaddrs: array (nullable = true)  
|   |   |-- element: struct (containsNull = false)  
|   |   |   |-- pref: integer (nullable = true)  
|   |   |   |-- type: string (nullable = true)  
|   |   |   |-- value: string (nullable = true)  
|   |-- fullname: string (nullable = true)  
|   |-- org: string (nullable = true)  
|   |-- telephones: array (nullable = true)  
|   |   |-- element: struct (containsNull = false)  
|   |   |   |-- pref: integer (nullable = true)  
|   |   |   |-- type: string (nullable = true)  
|   |   |   |-- value: string (nullable = true)  
|   |-- urls: array (nullable = true)  
|   |   |-- element: struct (containsNull = false)  
|   |   |   |-- type: string (nullable = true)  
|   |   |   |-- value: string (nullable = true)
```

## 第四步 查看 jsonQuery 的整个运行计划

```
scala> jsonQuery.queryExecution
```



```
hadoop1 | hadoop2 | hadoop3
scala> jsonQuery.queryExecution
res13: jsonQuery.sqlContext.QueryExecution =
== Parsed Logical Plan ==
Project [*]
  UnresolvedRelation None, jsonPerson, None

== Analyzed Logical Plan ==
Project [addresses#3,emailaddrs#4,fullname#5,org#6,telephones#7,urls#8]
  SparkLogicalPlan (ExistingRDD [addresses#3,emailaddrs#4,fullname#5,org#6,telephones#7,urls#8], MappedRDD[10] at map at JsonRDD.scala:38)

== Optimized Logical Plan ==
SparkLogicalPlan (ExistingRDD [addresses#3,emailaddrs#4,fullname#5,org#6,telephones#7,urls#8], MappedRDD[10] at map at JsonRDD.scala:38)

== Physical Plan ==
ExistingRDD [addresses#3,emailaddrs#4,fullname#5,org#6,telephones#7,urls#8], MappedRDD[10] at map at JsonRDD.scala:38

Code Generation: false
== RDD ==
```

### 1.4.5.2 读取 Parquet 格式数据

Parquet 数据放在配套资源/data/class6/wiki\_parquet 中，在以下测试中把文件放到/home/hadoop/upload/class6 路径下

第一步 读入 Parquet 数据

parquet 文件读入并注册成表 parquetWiki，然后定义一个查询 parquetQuery

```
scala> parquetFile("/home/hadoop/upload/class6/wiki_parquet").registerTempTable("parquetWiki")
```

```
scala> val parquetQuery = sql("select * from parquetWiki")
```

```
hadoop1 | hadoop2 | hadoop3
scala> parquetFile("/home/hadoop/upload/class6/wiki_parquet").registerTempTable("parquetwiki")
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
15/07/29 22:03:52 INFO parquet.ParquetTypesConverter: Falling back to schema conversion from Parquet types
Buffer(id#27, title#28, modified#29L, text#30, username#31)

scala> val parquetQuery = sql("select * from parquetwiki")
15/07/29 22:04:02 INFO parse.ParseDriver: Parsing command: select * from parquetwiki
15/07/29 22:04:02 INFO parse.ParseDriver: Parse Completed
parquetQuery: org.apache.spark.sql.SchemaRDD =
SchemaRDD[41] at RDD at SchemaRDD.scala:103
== Query Plan ==
== Physical Plan ==
ParquetTableScan [id#27,title#28,modified#29L,text#30,username#31], (ParquetRelation /home/hadoop/upload/class6/wiki_parquet, Some(Configuration: core-default.xml, core-site.xml, mapred-default.xml, mapred-site.xml), org.apache.hadoop.mapred.TestHive$@62d9de3b, []), []
```

有报错但不影响使用

```
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
15/02/28 17:29:54 INFO parquet.ParquetTypesConverter: Falling back to schema conversion from Parquet types; result: ArrayBuffer(id#0, title#1, modified#2L, text#3, username#4)
```

第二步 查询 parquetQuery 的 schema

```
scala> parquetQuery.printSchema
```

```
hadoop1 | hadoop2 | hadoop3
scala> parquetQuery.printSchema
root
|-- id: integer (nullable = true)
|-- title: binary (nullable = true)
|-- modified: long (nullable = true)
|-- text: binary (nullable = true)
|-- username: binary (nullable = true)
```

第三步 查询 parquetQuery 的整个运行计划

## scala>parquetQuery.queryExecution

```
scala> parquetQuery.queryExecution
res3: parquetQuery.sqlContext.QueryExecution =
== Parsed Logical Plan ==
Project [*]
  UnresolvedRelation None, parquetwiki, None

== Analyzed Logical Plan ==
Project [id#0,title#1,modified#2L,text#3,username#4]
  ParquetRelation /home/hadoop/upload/wiki_parquet, Some(Configuration: core-default.xml, core-site.xml, mapred-default.xml, mapred-site.xml), org.apache.spark.sql.hive.test.TestHive$@2088e6cd, []

== Optimized Logical Plan ==
ParquetRelation /home/hadoop/upload/wiki_parquet, Some(Configuration: core-default.xml, core-site.xml, mapred-default.xml, mapred-site.xml), org.apache.spark.sql.hive.test.TestHive$@2088e6cd, []

== Physical Plan ==
ParquetTableScan [id#0,title#1,modified#2L,text#3,username#4], (ParquetRelation /home/hadoop/upload/wiki_parquet, Some(Configuration: core-defa...
```

### 第四步 查询取样

## scala>parquetQuery.takeSample(false,10,2)

```
scala> parquetQuery.takeSample(false,10,2)
15/02/28 17:35:24 INFO storage.MemoryStore: ensureFreeSpace(36894) called with curMem=0, maxMem=1120995901
15/02/28 17:35:24 INFO storage.MemoryStore: Block broadcast_0 stored as values in memory (estimated size 36.0 KB, free 1069.0 MB)
15/02/28 17:35:25 INFO spark.SparkContext: Starting job: collect at SparkPlan.scala:85
15/02/28 17:35:25 INFO input.FileInputFormat: Total input paths to process : 10
15/02/28 17:35:25 INFO hadoop.ParquetInputFormat: Total input paths to process : 10
15/02/28 17:35:25 INFO hadoop.ParquetFileReader: reading summary file: file:/home/hadoop/upload/wiki_parquet/metadata
extraMetadata: {} readSupportMetadata: {org.apache.spark.sql.parquet.row.metadata=StructType(List(StructField(id,IntegerType,true), StructField(title,BinaryType,true), StructField(modified,LongType,true), StructField(text,BinaryType,true), StructField(username,BinaryType,true))), org.apache.spark.sql.parquet.row.requested_schema=StructType(List(StructField(id,IntegerType,true), StructField(title,BinaryType,true), StructField(modified,LongType,true), StructField(text,BinaryType,true), StructField(username,BinaryType,true)))}}
15/02/28 17:35:44 WARN hadoop.ParquetRecordReader: Can not initialize counter due to context is not a instance of TaskInputOutputContext, but is org.apache.hadoop.mapreduce.TaskAttemptContext
15/02/28 17:35:44 INFO hadoop.InternalParquetRecordReader: RecordReader initialized will read a total of 3930 records.
15/02/28 17:35:44 INFO hadoop.InternalParquetRecordReader: at row 0. reading next block
15/02/28 17:35:44 INFO compress.CodecPool: Got brand-new decompressor
15/02/28 17:35:44 INFO hadoop.InternalParquetRecordReader: block read in memory in 333 ms. row count = 3930
15/02/28 17:35:45 INFO executor.Executor: Finished task 9.0 in stage 2.0 (TID 20). 254081 bytes result sent to driver
15/02/28 17:35:45 INFO scheduler.DAGScheduler: stage 2 (collect at SparkPlan.scala:85) finished in 13.880 s
15/02/28 17:35:45 INFO spark.SparkContext: Job finished: collect at SparkPlan.scala:85, took 13.911563332 s
15/02/28 17:35:45 INFO scheduler.TaskSetManager: Finished task 9.0 in stage 2.0 (TID 20) in 1135 ms on localhost (10/10)
15/02/28 17:35:45 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 2.0, whose tasks have all completed, from pool
res4: Array[org.apache.spark.sql.Row] = Array([27429380,[B@157b98ab,1366749867,[B@65640d91,[B@bb6de98], [512036,[B@12cfd32a,1387279152,[B@a30740c,[B@7312c50d], [2326787,[B@55471380,1366458930,[B@2d51643d,[B@3a8bf809], [7004148,[B@3b4934e6,1393201300,[B@43300672,[B@5f748d4], [151026,[B@2dce595b,1398784088,[B@464c7cdb,[B@479931eb], [30618034,[B@1a9c6006,1348078744,[B@10643790,[B@237bd758], [26377030,[B@1776e49b,1397572318,[B@7e9a8825,[B@53607612], [24413778,[B@61e7c281,1353016963,[B@3692d55e,[B@61f62a4f], [25301252,[B@6853dac7,1391366818,[B@59407453,[B@169522ec], [14428870,[B@1927bf89,1345726557,[B@8fc846,[B@5fa56bf3]])
[B@8fc846,[B@5fa56bf3])
```

### 1.4.5.3 读取 hive 内置测试数据

在 TestHive 类中已经定义了大量的 hive0.13 的测试数据的表格式，如 src、sales 等等，在 hive-console 中可以直接使用；第一次使用的时候，hive-console 会装载一次。下面我们使用 sales 表看看其 schema 和整个运行计划。

### 第一步 读入测试数据并定义一个查询 hiveQuery

## scala>val hiveQuery = sql("select \* from sales")

```
scala> val hiveQuery = sql("select * from sales")
15/07/29 22:09:30 INFO parse.ParseDriver: Parsing command: select * from sales
15/07/29 22:09:30 INFO parse.ParseDriver: Parse Completed
15/07/29 22:09:30 INFO metastore.HiveMetaStore: 0: get_table : db=default tbl=sales
15/07/29 22:09:30 INFO HiveMetaStore.audit: ugi=hadoop ip=unknown-ip-addr cmd=get_table : db=default tbl=sales
15/07/29 22:09:30 INFO storage.MemoryStore: ensureFreeSpace(149147) called with curMem=181243, maxMem=1111794647
15/07/29 22:09:30 INFO storage.MemoryStore: Block broadcast_16 stored as values in memory (estimated size 145.7 KB, free 1060.0 MB)
hiveQuery: org.apache.spark.sql.SchemaRDD =
SchemaRDD[44] at RDD at SchemaRDD.scala:103
== Query Plan ==
== Physical Plan ==
HiveTableScan [key#41,value#42], (MetastoreRelation default, sales, None), None
```

### 第二步 查看 hiveQuery 的 schema



*scala>hiveQuery.printSchema*

```
scala> hiveQuery.printSchema
root
|-- key: string (nullable = true)
|-- value: integer (nullable = true)
```

第三步 查看 hiveQuery 的整个运行计划

*scala>hiveQuery.queryExecution*

```
scala> hiveQuery.queryExecution
res1: hiveQuery.sqlContext.QueryExecution =
== Parsed Logical Plan ==
Project [*]
  UnresolvedRelation None, sales, None

== Analyzed Logical Plan ==
Project [key#4,value#5]
  MetastoreRelation default, sales, None

== Optimized Logical Plan ==
MetastoreRelation default, sales, None

== Physical Plan ==
HiveTableScan [key#4,value#5], (MetastoreRelation default, sales, None), None

Code Generation: false
== RDD ==
```

第四步 其他 SQL 语句的运行计划

*scala>val hiveQuery = sql("select \* from (select \* from src limit 5) a limit 3")*

```
scala> val hiveQuery = sql("select * from (select * from src limit 5) a limit 3")
15/02/28 22:11:06 INFO parse.ParseDriver: Parsing command: select * from (select * from src limit 5) a limit 3
15/02/28 22:11:06 INFO parse.ParseDriver: Parse Completed
15/02/28 22:11:06 INFO metastore.HiveMetaStore: 0: get_table : db=default tbl=src
15/02/28 22:11:06 INFO HiveMetaStore.audit: ugi=hadoop ip=unknown-ip-addr cmd=get_table : db=default tbl=src
15/02/28 22:11:06 INFO storage.MemoryStore: ensureFreespace(149166) called with curMem=595850, maxMem=1120995901
15/02/28 22:11:06 INFO storage.MemoryStore: Block broadcast_4 stored as values in memory (estimated size 145.7 KB, free 1068.4 MB)
hiveQuery: org.apache.spark.sql.SchemaRDD =
SchemaRDD[4] at RDD at SchemaRDD.scala:103
== Query Plan ==
== Physical Plan ==
Limit 3
  Limit 5
    HiveTableScan [key#12,value#13], (MetastoreRelation default, src, None), None

scala>
```

*scala>val hiveQuery = sql("select \* FROM (select \* FROM src) a")*

```
scala> val hiveQuery = sql("select * FROM (select * FROM src) a")
15/02/28 22:11:33 INFO parse.ParseDriver: Parsing command: select * FROM (select * FROM src) a
15/02/28 22:11:33 INFO parse.ParseDriver: Parse Completed
15/02/28 22:11:33 INFO metastore.HiveMetaStore: 0: get_table : db=default tbl=src
15/02/28 22:11:33 INFO HiveMetaStore.audit: ugi=hadoop ip=unknown-ip-addr cmd=get_table : db=default tbl=src
15/02/28 22:11:33 INFO storage.MemoryStore: ensureFreespace(149198) called with curMem=745016, maxMem=1120995901
15/02/28 22:11:33 INFO storage.MemoryStore: Block broadcast_5 stored as values in memory (estimated size 145.7 KB, free 1068.2 MB)
hiveQuery: org.apache.spark.sql.SchemaRDD =
SchemaRDD[5] at RDD at SchemaRDD.scala:103
== Query Plan ==
== Physical Plan ==
HiveTableScan [key#14,value#15], (MetastoreRelation default, src, None), None
```

*scala>hiveQuery.where('key === 100).queryExecution.toRdd.collect*

```

| hadoop1 | hadoop2 | hadoop3
scala> hiveQuery.where('key == 100').queryExecution.toRdd.collect
15/02/28 22:12:34 INFO metastore.HiveMetaStore: 0: get_table : db=default tbl=src
15/02/28 22:12:34 INFO HiveMetaStore.audit: ugi=hadoop ip=unknown-ip-addr cmd=get_table : db=default tbl=src
15/02/28 22:12:34 INFO storage.MemoryStore: ensureFreeSpace(149254) called with curMem=1051284, maxMem=1120995901
15/02/28 22:12:34 INFO storage.MemoryStore: Block broadcast_8 stored as values in memory (estimated size 145.8 KB, free 1067.9 MB)
15/02/28 22:12:34 INFO mapred.FileInputFormat: Total input paths to process : 1
15/02/28 22:12:34 INFO spark.SparkContext: Starting job: collect at <console>:41
15/02/28 22:12:34 INFO scheduler.DAGScheduler: Got job 1 (collect at <console>:41) with 2 output partitions (allowLocal=false)
15/02/28 22:12:34 INFO scheduler.DAGScheduler: Final stage: Stage 1(collect at <console>:41)
15/02/28 22:12:34 INFO scheduler.DAGScheduler: Parents of final stage: List()
15/02/28 22:12:34 INFO scheduler.DAGScheduler: Missing parents: List()
15/02/28 22:12:34 INFO scheduler.DAGScheduler: Submitting Stage 1 (MappedRDD[17] at map at HiveContext.scala:360), which has no missing pare
15/02/28 22:12:34 INFO storage.MemoryStore: ensureFreeSpace(7848) called with curMem=1200538, maxMem=1120995901
15/02/28 22:12:34 INFO storage.MemoryStore: Block broadcast_9 stored as values in memory (estimated size 7.7 KB, free 1067.9 MB)
15/02/28 22:12:34 INFO scheduler.DAGScheduler: Submitting 2 missing tasks from Stage 1 (MappedRDD[17] at map at HiveContext.scala:360)
15/02/28 22:12:34 INFO scheduler.TaskSchedulerImpl: Adding task set 1.0 with 2 tasks
15/02/28 22:12:34 INFO scheduler.TaskSetManager: Starting task 0.0 in stage 1.0 (TID 2, localhost, PROCESS_LOCAL, 1216 bytes)
15/02/28 22:12:34 INFO executor.Executor: Running task 0.0 in stage 1.0 (TID 2)
15/02/28 22:12:34 INFO rdd.HadoopRDD: Input split: file:/tmp/sparkHivewarehouse1642017435544903306/src/kv1.txt:0+2906
15/02/28 22:12:34 INFO executor.Executor: Finished task 0.0 in stage 1.0 (TID 2). 1952 bytes result sent to driver
15/02/28 22:12:34 INFO scheduler.TaskSetManager: Starting task 1.0 in stage 1.0 (TID 3, localhost, PROCESS_LOCAL, 1216 bytes)
15/02/28 22:12:34 INFO executor.Executor: Running task 1.0 in stage 1.0 (TID 3)
15/02/28 22:12:34 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 1.0 (TID 2) in 41 ms on localhost (1/2)
15/02/28 22:12:34 INFO rdd.HadoopRDD: Input split: file:/tmp/sparkHivewarehouse1642017435544903306/src/kv1.txt:2906+2906
15/02/28 22:12:34 INFO executor.Executor: Finished task 1.0 in stage 1.0 (TID 3). 1952 bytes result sent to driver
15/02/28 22:12:34 INFO scheduler.DAGScheduler: Stage 1 (collect at <console>:41) finished in 0.095 s
15/02/28 22:12:34 INFO spark.SparkContext: Job finished: collect at <console>:41, took 0.116700063 s
res4: Array[org.apache.spark.sql.Row] = Array([100,val_100], [100,val_100])

```

## 1.4.6 不同查询的运行计划

### 1.4.6.1 聚合查询

*scala> sql("select name, age, state as location from people").queryExecution*

```

| hadoop1 | hadoop2 | hadoop3
scala> sql("select name, age, state as location from people").queryExecution
<console>:40: warning: inferred existential type _10.sqlContext.QueryExecution forSome { val _10: org.apache.spark.sql.SchemaRDD }, which ca
nnot be expressed by wildcards, should be enabled
by making the implicit value scala.language.existentials visible.
This can be achieved by adding the import clause 'import scala.language.existentials'
or by setting the compiler option -language:existentials.
See the scala docs for value scala.language.existentials for a discussion
why the feature should be explicitly enabled.
sql("select name, age, state as location from people").queryExecution
15/02/28 22:28:14 INFO parse.ParseDriver: Parsing command: select name, age, state as location from people
15/02/28 22:28:14 INFO parse.ParseDriver: Parse Completed
res14: _10.sqlContext.QueryExecution forSome { val _10: org.apache.spark.sql.SchemaRDD } =
== Parsed Logical Plan ==
Project [name, age, state AS location#45]
  UnresolvedRelation None, people, None
== Analyzed Logical Plan ==
Project [name#25, age#26, state#27 AS location#45]
  SparkLogicalPlan (ExistingRdd [name#25, age#26, state#27], MapPartitionsRDD[20] at mapPartitions at basicOperators.scala:208)
== Optimized Logical Plan ==
Project [name#25, age#26, state#27 AS location#45]
  SparkLogicalPlan (ExistingRdd [name#25, age#26, state#27], MapPartitionsRDD[20] at mapPartitions at basicOperators.scala:208)
== Physical Plan ==
Project [name#25, age#26, state#27 AS location#45]
  ExistingRdd [name#25, age#26, state#27], MapPartitionsRDD[20] at mapPartitions at basicOperators.scala:208
Code Generati...
scala>

```

*scala> sql("select name from (select name, state as location from people) a where location='CA').queryExecution*

```

| hadoop1 | hadoop2 | hadoop3
scala> sql("select name from (select name, state as location from people) a where location='CA').queryExecution
<console>:40: warning: inferred existential type _11.sqlContext.QueryExecution forSome { val _11: org.apache.spark.sql.SchemaRDD }, which ca
nnot be expressed by wildcards, should be enabled
by making the implicit value scala.language.existentials visible.
This can be achieved by adding the import clause 'import scala.language.existentials'
or by setting the compiler option -language:existentials.
See the scala docs for value scala.language.existentials for a discussion
why the feature should be explicitly enabled.
sql("select name from (select name, state as location from people) a where location='CA').queryExecution
15/02/28 22:28:53 INFO parse.ParseDriver: Parsing command: select name from (select name, state as location from people) a where location='CA
15/02/28 22:28:53 INFO parse.ParseDriver: Parse Completed
res15: _11.sqlContext.QueryExecution forSome { val _11: org.apache.spark.sql.SchemaRDD } =
== Parsed Logical Plan ==
Project [name]
  Filter ('location = CA)
    Subquery a
      Project [name, state AS location#47]
        UnresolvedRelation None, people, None
== Analyzed Logical Plan ==
Project [name#25]
  Filter (location#47 = CA)
    Project [name#25, state#27 AS location#47]
      SparkLogicalPlan (ExistingRdd [name#25, age#26, state#27], MapPartitionsRDD[20] at mapPartitions at basicOperators.scala:208)
== Optimized Logical Plan ==
Project [name#25]
  Filter (state#27 = CA)
    SparkLogicalPlan (ExistingRdd [name#25, age#26, state#27], MapPartitionsRDD[20] at mapPartitions at basicOperators.scala:208)
== Physical Plan ==
Project [name#25]
  Filter (state#27 = CA)
    ExistingRdd [name#25, age#26, state#...
scala>

```

*scala> sql("select sum(age) from people").queryExecution*

*scala>sql("select sum(age) from people").toDebugString*

```
scala> sql("select sum(age) from people").queryExecution
<console>:40: warning: inferred existential type _12.sqlContext.QueryExecution for some { val _12: org.apache.spark.sql.SchemaRDD }, which cannot be expressed by wildcards, should be enabled
by making the implicit value scala.language.existentials visible.
This can be achieved by adding the import clause 'import scala.language.existentials'
or by setting the compiler option -language:existentials.
See the scala docs for value scala.language.existentials for a discussion
why the feature should be explicitly enabled.
    sql("select sum(age) from people").queryExecution
    ^
15/02/28 22:29:30 INFO parse.ParseDriver: Parsing command: select sum(age) from people
15/02/28 22:29:30 INFO parse.ParseDriver: Parse Completed
res16: _12.sqlContext.QueryExecution for some { val _12: org.apache.spark.sql.SchemaRDD } =
== Parsed Logical Plan ==
Project [SUM('age) AS c_0#49]
  UnresolvedRelation None, people, None

== Analyzed Logical Plan ==
Aggregate [], [SUM(CAST(age#26, LongType)) AS c_0#49L]
  SparkLogicalPlan (ExistingRDD [name#25,age#26,state#27], MapPartitionsRDD[20] at mapPartitions at basicOperators.scala:208)

== Optimized Logical Plan ==
Aggregate [], [SUM(CAST(age#26, LongType)) AS c_0#49L]
  Project [age#26]
    SparkLogicalPlan (ExistingRDD [name#25,age#26,state#27], MapPartitionsRDD[20] at mapPartitions at basicOperators.scala:208)

== Physical Plan ==
Aggregate false, [], [SUM(PartialSum#52L) AS c_0#49L]
  Exchange SinglePartition
    Aggregate true, [], [SUM(CAST(age#26, LongType)) AS PartialSum#52L]
      ...
scala>
```

```
scala> sql("select sum(age) from people").toDebugString
15/02/28 22:30:09 INFO parse.ParseDriver: Parsing command: select sum(age) from people
15/02/28 22:30:09 INFO parse.ParseDriver: Parse Completed
res17: String =
(1) SchemaRDD[39] at RDD at SchemaRDD.scala:103
== Query Plan ==
== Physical Plan ==
Aggregate false, [], [SUM(PartialSum#59L) AS c_0#56L]
  Exchange SinglePartition
    Aggregate true, [], [SUM(CAST(age#26, LongType)) AS PartialSum#59L]
      Project [age#26]
        ExistingRDD [name#25,age#26,state#27], MapPartitionsRDD[20] at mapPartitions at basicOperators.scala:208
      | MappedRDD[46] at map at HiveContext.scala:360
      | MapPartitionsRDD[45] at mapPartitions at Aggregate.scala:128
      | MappedRDD[44] at map at Exchange.scala:97
      | ShuffledRDD[43] at ShuffledRDD at Exchange.scala:95
      +- (1) MapPartitionsRDD[41] at mapPartitions at Aggregate.scala:86
        | MapPartitionsRDD[40] at mapPartitions at basicOperators.sc...
```

*scala>sql("select state,avg(age) from people group by state").queryExecution*

*scala>sql("select state,avg(age) from people group by state").toDebugString*

```
scala> sql("select state,avg(age) from people group by state").queryExecution
<console>:40: warning: inferred existential type _13.sqlContext.QueryExecution for some { val _13: org.apache.spark.sql.SchemaRDD }, which cannot be expressed by wildcards, should be enabled
by making the implicit value scala.language.existentials visible.
This can be achieved by adding the import clause 'import scala.language.existentials'
or by setting the compiler option -language:existentials.
See the scala docs for value scala.language.existentials for a discussion
why the feature should be explicitly enabled.
    sql("select state,avg(age) from people group by state").queryExecution
    ^
15/02/28 22:30:57 INFO parse.ParseDriver: Parsing command: select state,avg(age) from people group by state
15/02/28 22:30:57 INFO parse.ParseDriver: Parse Completed
res18: _13.sqlContext.QueryExecution for some { val _13: org.apache.spark.sql.SchemaRDD } =
== Parsed Logical Plan ==
Aggregate ['state], ['state,AVG('age) AS c_1#63]
  UnresolvedRelation None, people, None

== Analyzed Logical Plan ==
Aggregate [state#27], [state#27,AVG(CAST(age#26, LongType)) AS c_1#63]
  SparkLogicalPlan (ExistingRDD [name#25,age#26,state#27], MapPartitionsRDD[20] at mapPartitions at basicOperators.scala:208)

== Optimized Logical Plan ==
Aggregate [state#27], [state#27,AVG(CAST(age#26, LongType)) AS c_1#63]
  Project [state#27,age#26]
    SparkLogicalPlan (ExistingRDD [name#25,age#26,state#27], MapPartitionsRDD[20] at mapPartitions at basicOperators.scala:208)

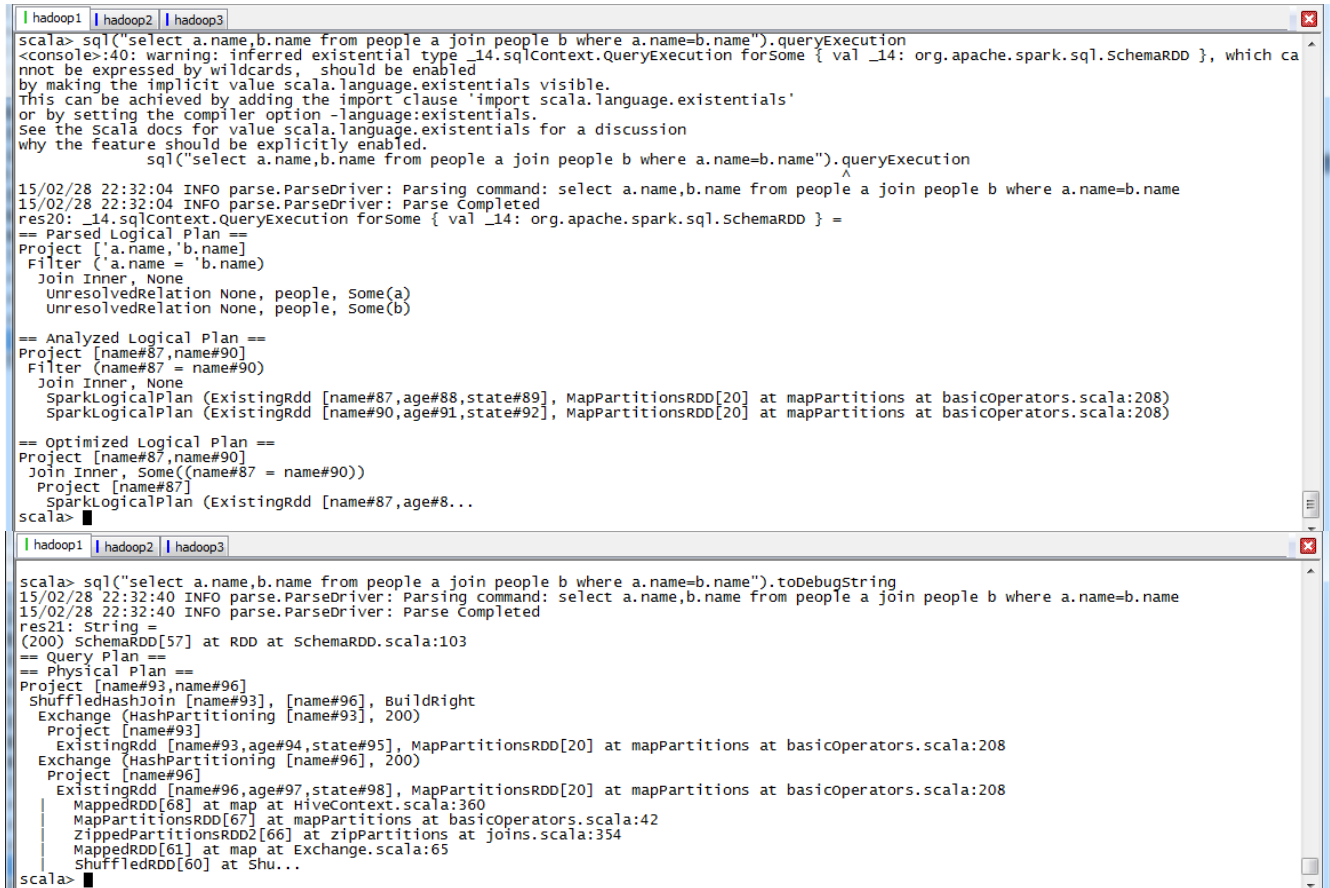
== Physical Plan ==
Aggregate false, [state#27], [state#27,(CAST(SUM(PartialSum#67L), DoubleType) / CAST(SUM(Pa...
scala>
```

```
scala> sql("select state,avg(age) from people group by state").toDebugString
15/02/28 22:31:33 INFO parse.ParseDriver: Parsing command: select state,avg(age) from people group by state
15/02/28 22:31:33 INFO parse.ParseDriver: Parse Completed
res19: String =
(200) SchemaRDD[48] at RDD at SchemaRDD.scala:103
== Query Plan ==
== Physical Plan ==
Aggregate false, [state#27], [state#27,(CAST(SUM(PartialSum#79L), DoubleType) / CAST(SUM(PartialCount#80L), DoubleType)) AS c_1#75]
  Exchange (HashPartitioning [state#27], 200)
    Aggregate true, [state#27], [state#27,COUNT(CAST(age#26, LongType)) AS PartialCount#80L,SUM(CAST(age#26, LongType)) AS PartialSum#79L]
      Project [state#27,age#26]
        ExistingRDD [name#25,age#26,state#27], MapPartitionsRDD[20] at mapPartitions at basicOperators.scala:208
      | MappedRDD[55] at map at HiveContext.scala:360
      | MapPartitionsRDD[54] at mapPartitions at Aggregate.scala:151
      | MappedRDD[53] at map at Exchange.scala:65
      | ShuffledRDD[52] at ShuffledRDD at Exchange.scala:63
      +- (1) MapParti...
```

### 1.4.6.2 Join 操作

```
scala> sql("select a.name,b.name from people a join people b where  
a.name=b.name").queryExecution
```

```
scala> sql("select a.name,b.name from people a join people b where  
a.name=b.name").toDebugString
```



```
scala> sql("select a.name,b.name from people a join people b where a.name=b.name").queryExecution
<console>:40: warning: inferred existential type _14.sqlContext.QueryExecution forSome { val _14: org.apache.spark.sql.SchemaRDD }, which ca
nnot be expressed by wildcards, should be enabled
by making the implicit value scala.language.existentials visible.
This can be achieved by adding the import clause 'import scala.language.existentials'
or by setting the compiler option -language:existentials.
See the scala docs for value scala.language.existentials for a discussion
why the feature should be explicitly enabled.
    sql("select a.name,b.name from people a join people b where a.name=b.name").queryExecution
                                     ^
15/02/28 22:32:04 INFO parse.ParseDriver: Parsing command: select a.name,b.name from people a join people b where a.name=b.name
15/02/28 22:32:04 INFO parse.ParseDriver: Parse Completed
res20: _14.sqlContext.QueryExecution forSome { val _14: org.apache.spark.sql.SchemaRDD } =
== Parsed Logical Plan ==
Project ['a.name','b.name]
  Filter ('a.name = 'b.name)
    Join Inner, None
      unresolvedRelation None, people, Some(a)
      unresolvedRelation None, people, Some(b)

== Analyzed Logical Plan ==
Project [name#87,name#90]
  Filter (name#87 = name#90)
    Join Inner, None
      SparkLogicalPlan (ExistingRdd [name#87,age#88,state#89], MapPartitionsRDD[20] at mapPartitions at basicoperators.scala:208)
      SparkLogicalPlan (ExistingRdd [name#90,age#91,state#92], MapPartitionsRDD[20] at mapPartitions at basicoperators.scala:208)

== Optimized Logical Plan ==
Project [name#87,name#90]
  Join Inner, Some((name#87 = name#90))
    Project [name#87]
      SparkLogicalPlan (ExistingRdd [name#87,age#88,state#89], MapPartitionsRDD[20] at mapPartitions at basicoperators.scala:208)
scala>
```

```
scala> sql("select a.name,b.name from people a join people b where a.name=b.name").toDebugString
15/02/28 22:32:40 INFO parse.ParseDriver: Parsing command: select a.name,b.name from people a join people b where a.name=b.name
15/02/28 22:32:40 INFO parse.ParseDriver: Parse Completed
res21: String =
(200) SchemaRDD[57] at RDD at SchemaRDD.scala:103
== Query Plan ==
== Physical Plan ==
Project [name#93,name#96]
  ShuffledHashJoin [name#93], [name#96], BuildRight
    Exchange (HashPartitioning [name#93], 200)
      Project [name#93]
        ExistingRdd [name#93,age#94,state#95], MapPartitionsRDD[20] at mapPartitions at basicoperators.scala:208
    Exchange (HashPartitioning [name#96], 200)
      Project [name#96]
        ExistingRdd [name#96,age#97,state#98], MapPartitionsRDD[20] at mapPartitions at basicoperators.scala:208
      MappedRDD[68] at map at HiveContext.scala:360
      MapPartitionsRDD[67] at mapPartitions at basicoperators.scala:42
      ZippedPartitionsRDD[66] at zipPartitions at joins.scala:354
      MappedRDD[61] at map at Exchange.scala:65
      ShuffledRDD[60] at ShuffleRDD at ShuffleRDD.scala:103
scala>
```

### 1.4.6.3 Distinct 操作

```
scala> sql("select distinct a.name,b.name from people a join people b where  
a.name=b.name").queryExecution
```

```
scala> sql("select distinct a.name,b.name from people a join people b where  
a.name=b.name").toDebugString
```



```
scala> sql("select distinct a.name,b.name from people a join people b where a.name=b.name").queryExecution
<console>:40: warning: inferred existential type _15.sqlContext.QueryExecution forSome { val _15: org.apache.spark.sql.SchemaRDD }, which cannot be expressed by wildcards, should be enabled
by making the implicit value scala.language.existentials visible.
This can be achieved by adding the import clause 'import scala.language.existentials'
or by setting the compiler option -language:existentials.
See the Scala docs for value scala.language.existentials for a discussion
why the feature should be explicitly enabled.
    sql("select distinct a.name,b.name from people a join people b where a.name=b.name").queryExecution
15/02/28 22:33:00 INFO parse.ParseDriver: Parsing command: select distinct a.name,b.name from people a join people b where a.name=b.name
15/02/28 22:33:00 INFO parse.ParseDriver: Parse Completed
res22: _15.sqlContext.QueryExecution forSome { val _15: org.apache.spark.sql.SchemaRDD } =
== Parsed Logical Plan ==
Distinct
  Project ['a.name','b.name]
    Filter ('a.name = 'b.name)
      Join Inner, None
        UnresolvedRelation None, people, Some(a)
        UnresolvedRelation None, people, Some(b)

== Analyzed Logical Plan ==
Distinct
  Project [name#99,name#102]
    Filter (name#99 = name#102)
      Join Inner, None
        SparkLogicalPlan (ExistingRdd [name#99,age#100,state#101], MapPartitionsRDD[20] at mapPartitions at basicoperators.scala:208)
        SparkLogicalPlan (ExistingRdd [name#102,age#103,state#104], MapPartitionsRDD[20] at mapPartitions at basicoperators.scala:208)

== Optimized Logical Plan ==
Distinct
  Project [name#99,name#102]
    Join Inner, Some((name#99 = name#102))
      Project [name...]
scala>
```

```
scala> sql("select distinct a.name,b.name from people a join people b where a.name=b.name").toDebugString
15/02/28 22:33:39 INFO parse.ParseDriver: Parsing command: select distinct a.name,b.name from people a join people b where a.name=b.name
15/02/28 22:33:39 INFO parse.ParseDriver: Parse Completed
res23: String =
(200) SchemaRDD[70] at RDD at SchemaRDD.scala:103
== Query Plan ==
== Physical Plan ==
Distinct false
  Project [name#105,name#108]
    ShuffledHashJoin [name#105], [name#108], BuildRight
      Exchange (HashPartitioning [name#105], 200)
        Project [name#105]
          ExistingRdd [name#105,age#106,state#107], MapPartitionsRDD[20] at mapPartitions at basicoperators.scala:208
      Exchange (HashPartitioning [name#108], 200)
        Project [name#108]
          ExistingRdd [name#108,age#109,state#110], MapPartitionsRDD[20] at mapPartitions at basicoperators.scala:208
    MappedRDD[83] at map at HiveContext.scala:360
    MapPartitionsRDD[82] at mapPartitions at basicoperators.scala:256
    MapPartitionsRDD[81] at mapPartitions at basicoperators.scala:256
    MapP...
scala>
```

## 1.4.7 优化

### 1.4.7.1 CombineFilters

CombineFilters 就是合并 Filter，在含有多个 Filter 时发生，如下查询：

*sql("select name from (select \* from people where age >=19) a where a.age <30").queryExecution*

```
scala> sql("select name from (select * from people where age >=19) a where a.age <30").queryExecution
<console>:40: warning: inferred existential type _16.sqlContext.QueryExecution forSome { val _16: org.apache.spark.sql.SchemaRDD }, which cannot be expressed by wildcards, should be enabled
by making the implicit value scala.language.existentials visible.
This can be achieved by adding the import clause 'import scala.language.existentials'
or by setting the compiler option -language:existentials.
See the Scala docs for value scala.language.existentials for a discussion
why the feature should be explicitly enabled.
    sql("select name from (select * from people where age >=19) a where a.age <30").queryExecution
15/02/28 22:36:05 INFO parse.ParseDriver: Parsing command: select name from (select * from people where age >=19) a where a.age <30
15/02/28 22:36:05 INFO parse.ParseDriver: Parse Completed
res24: _16.sqlContext.QueryExecution forSome { val _16: org.apache.spark.sql.SchemaRDD } =
== Parsed Logical Plan ==
Project ['name]
  Filter ('a.age < 30)
    Subquery a
      Project [*]
        Filter ('age >= 19)
          UnresolvedRelation None, people, None

== Analyzed Logical Plan ==
Project [name#25]
  Filter (age#26 < 30)
    Project [name#25,age#26,state#27]
      Filter (age#26 >= 19)
        SparkLogicalPlan (ExistingRdd [name#25,age#26,state#27], MapPartitionsRDD[20] at mapPartitions at basicoperators.scala:208)

== Optimized Logical Plan ==
Project [name#25]
  Filter ((age#26 >= 19) && (age#26 < 30))
    SparkLogicalPlan (ExistingRdd [name#25,age#26,state#27], MapPartitionsRDD[20] at mapPartitions at basicoperators.scala:208)

== Physical Plan ==
Project [name#25]
  Filter ((age#26 >= 19) && (age#26 < 30))
scala>
```

上面的查询，在 Optimized 的过程中，将 age>=19 和 age<30 这两个 Filter 合并了，合并成 ((age>=19) && (age<30))。其实上面还做了一个其他的优化，就是 project 的下推，子查询

使用了表的所有列，而主查询使用了列 name，在查询数据的时候子查询优化成只查列 name。

### 1.4.7.2 PushPredicateThroughProject

PushPredicateThroughProject 就是 project 下推，和上面例子中的 project 一样

*sql("select name from (select name,state as location from people) a where location='CA']").queryExecution*

```
scala> sql("select name from (select name,state as location from people) a where location='CA']").queryExecution
<console>:40: warning: inferred existential type _17.sqlContext.QueryExecution forSome { val _17: org.apache.spark.sql.SchemaRDD }, which cannot be expressed by wildcards, should be enabled
by making the implicit value scala.language.existentials visible.
This can be achieved by adding the import clause 'import scala.language.existentials'
or by setting the compiler option -language:existentials.
See the Scala docs for value scala.language.existentials for a discussion
why the feature should be explicitly enabled.
sql("select name from (select name,state as location from people) a where location='CA']").queryExecution

15/02/28 22:37:49 INFO parse.ParseDriver: Parsing command: select name from (select name,state as location from people) a where location='CA'
15/02/28 22:37:49 INFO parse.ParseDriver: Parse completed
res25: _17.sqlContext.QueryExecution forSome { val _17: org.apache.spark.sql.SchemaRDD } =
== Parsed Logical Plan ==
Project [name]
  Filter (location = CA)
    Subquery a
      Project [name,state AS location#111]
        unresolvedRelation None, people, None
== Analyzed Logical Plan ==
Project [name#25]
  Filter (location#111 = CA)
    Project [name#25,state#27 AS location#111]
      SparkLogicalPlan (ExistingRdd [name#25,age#26,state#27], MapPartitionsRDD[20] at mapPartitions at basicOperators.scala:208)
== Optimized Logical Plan ==
Project [name#25]
  Filter (state#27 = CA)
    SparkLogicalPlan (ExistingRdd [name#25,age#26,state#27], MapPartitionsRDD[20] at mapPartitions at basicOperators.scala:208)
== Physical Plan ==
Project [name#25]
  Filter (state#27 = CA)
    ExistingRdd [name#25,age#26,sta...
scala>
```

### 1.4.7.3 ConstantFolding :

ConstantFolding 是常量叠加，用于表达式。如下面的例子：

*sql("select name,1+2 from people").queryExecution*

```
scala> sql("select name,1+2 from people").queryExecution
<console>:40: warning: inferred existential type _18.sqlContext.QueryExecution forSome { val _18: org.apache.spark.sql.SchemaRDD }, which cannot be expressed by wildcards, should be enabled
by making the implicit value scala.language.existentials visible.
This can be achieved by adding the import clause 'import scala.language.existentials'
or by setting the compiler option -language:existentials.
See the Scala docs for value scala.language.existentials for a discussion
why the feature should be explicitly enabled.
sql("select name,1+2 from people").queryExecution

15/02/28 22:39:59 INFO parse.ParseDriver: Parsing command: select name,1+2 from people
15/02/28 22:39:59 INFO parse.ParseDriver: Parse completed
res26: _18.sqlContext.QueryExecution forSome { val _18: org.apache.spark.sql.SchemaRDD } =
== Parsed Logical Plan ==
Project [name,(1 + 2) AS c_1#113]
  unresolvedRelation None, people, None
== Analyzed Logical Plan ==
Project [name#25,(1 + 2) AS c_1#113]
  SparkLogicalPlan (ExistingRdd [name#25,age#26,state#27], MapPartitionsRDD[20] at mapPartitions at basicOperators.scala:208)
== Optimized Logical Plan ==
Project [name#25,3 AS c_1#113]
  SparkLogicalPlan (ExistingRdd [name#25,age#26,state#27], MapPartitionsRDD[20] at mapPartitions at basicOperators.scala:208)
== Physical Plan ==
Project [name#25,3 AS c_1#113]
  ExistingRdd [name#25,age#26,state#27], MapPartitionsRDD[20] at mapPartitions at basicOperators.scala:208
Code Generation: false
== RDD ==
scala>
```

## 2 SparkSQL 调优

Spark 是一个快速的内存计算框架，同时是一个并行运算的框架，在计算性能调优的时候，除了要考虑广为人知的木桶原理外，还要考虑平行运算的 Amdahl 定理。



木桶原理又称短板理论，其核心思想是：一只木桶盛水的多少，并不取决于桶壁上最高的那块木块，而是取决于桶壁上最短的那块。将这个理论应用到系统性能优化上，系统的最终性能取决于系统中性能表现最差的组件。例如，即使系统拥有充足的内存资源和 CPU 资源，但是如果磁盘 I/O 性能低下，那么系统的总体性能是取决于当前最慢的磁盘 I/O 速度，而不是当前最优越的 CPU 或者内存。在这种情况下，如果需要进一步提升系统性能，优化内存或者 CPU 资源是毫无用处的。只有提高磁盘 I/O 性能才能对系统的整体性能进行优化。



Amdahl 定理，一个计算机科学界的经验法则，因吉恩·阿姆达尔而得名。它代表了处理器平行运算之后效率提升的能力。并行计算中的加速比是用并行前的执行速度和并行后的执行速度之比来表示的，它表示了在并行化之后的效率提升情况。阿姆达尔定律是固定负载（计算总量不变时）时的量化标准。可用公式：

$$\frac{W_s + \frac{W_p}{p}}{W_s + \frac{W_p}{p}}$$
 来表示。式中  $W_s, W_p$  分别表示问题规模的串行分量（问题中不能并行化的那一部分）和并行分量， $p$  表示处理器数量。当  $p \rightarrow \infty$  时，

上式的极限是  $\frac{W}{W_s}$ ，其中  $M = M^s + M^p$ 。这意味着无论我们如何增大处理器数目，加速比是无法高于这个数的。

SparkSQL 作为 Spark 的一个组件，在调优的时候，也要充分考虑到上面的两个原理，既要考虑如何充分的利用硬件资源，又要考虑如何利用好分布式系统的并行计算。由于测试环境条件有限，本篇不能做出更详尽的实验数据来说明，只能在理论上加以说明。

## 2.1 并行性

SparkSQL 在集群中运行，将一个查询任务分解成大量的 Task 分配给集群中的各个节点来运行。通常情况下，Task 的数量是大于集群的并行度。比如前面第六章和第七章查询数据时，shuffle

的时候使用了缺省的 `spark.sql.shuffle.partitions`，即 200 个 partition，也就是 200 个 Task：

### Spark Stages

Total Duration: 4.9 min  
Scheduling Mode: FIFO  
Active Stages: 2  
Completed Stages: 7  
Failed Stages: 0

#### Active Stages (2)

Stage Id	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	Shuffle Read	Shuffle Write
6	<a href="#">mapPartitions at Exchange.scala:48</a>	<a href="#">+details</a> (kill)	2014/10/23 05:01:42	2 s	<div><div></div></div> 4/200		40.1 KB	90.7 KB
11	<a href="#">mapPartitions at Exchange.scala:48</a>	<a href="#">+details</a> (kill)	2014/10/23 05:01:07	36 s	<div><div></div></div> 1/2	5.9 MB		2.4 MB

而实验的集群环境却只能并行 3 个 Task，也就是说同时只能有 3 个 Task 保持 Running：

20	1233	0	SUCCESS	PROCESS_LOCAL	hadoop1	2014/10/23 05:02:46	0.5 s			25.4 KB	1 ms	5.7 KB	
21	1234	0	SUCCESS	PROCESS_LOCAL	hadoop3	2014/10/23 05:02:46	0.4 s			24.6 KB	1 ms	5.8 KB	
22	1235	0	SUCCESS	PROCESS_LOCAL	hadoop2	2014/10/23 05:02:46	0.2 s			24.3 KB	1 ms	5.7 KB	
23	1236	0	RUNNING	PROCESS_LOCAL	hadoop3	2014/10/23 05:02:46	0.4 s						
24	1237	0	RUNNING	PROCESS_LOCAL	hadoop1	2014/10/23 05:02:46	0.4 s						
25	1238	0	SUCCESS	PROCESS_LOCAL	hadoop2	2014/10/23 05:02:46	0.4 s			26.5 KB		5.8 KB	
26	1239	0	RUNNING	PROCESS_LOCAL	hadoop2	2014/10/23 05:02:46	27 ms						

这时大家就应该明白了，要跑完这 200 个 Task 就要跑  $200/3=67$  批次。如何减少运行的批次呢？那就要尽量提高查询任务的并行度。查询任务的并行度由两方面决定：集群的处理能力和集群的有效处理能力。

- 对于 Spark Standalone 集群来说，集群的处理能力是由 `conf/spark-env` 中的 `SPARK_WORKER_INSTANCES` 参数、`SPARK_WORKER_CORES` 参数决定的；而 `SPARK_WORKER_INSTANCES*SPARK_WORKER_CORES` 不能超过物理机器的实际 CPU core；
- 集群的有效处理能力是指集群中空闲的集群资源，一般是指使用 `spark-submit` 或 `spark-shell` 时指定的 `--total-executor-cores`，一般情况下，我们不需要指定，这时候，Spark Standalone 集群会将所有空闲的 core 分配给查询，并且在 Task 轮询运行过程中，Standalone 集群会将其他 spark 应用程序运行完后空闲出来的 core 也分配给正在运行中的查询。

综上所述，SparkSQL 的查询并行度主要和集群的 core 数量相关，合理配置每个节点的 core 可以提高集群的并行度，提高查询的效率。

## 2.2 高效的数据格式

高效的数据格式，一方面是加快了数据的读入速度，另一方面可以减少内存的消耗。高效的数据格式包括多个方面：

## 2.2.1 数据本地性

分布式计算系统的精粹在于移动计算而非移动数据，但是在实际的计算过程中，总存在着移动数据的情况，除非是在集群的所有节点上都保存数据的副本。移动数据，将数据从一个节点移动到另一个节点进行计算，不但消耗了网络 IO，也消耗了磁盘 IO，降低了整个计算的效率。为了提高数据的本地性，除了优化算法（也就是修改 spark 内存，难度有点高），就是合理设置数据的副本。设置数据的副本，这需要通过配置参数并长期观察运行状态才能获取的一个经验值。

下面是 Spark webUI 监控 Stage 的一个图：

- PROCESS\_LOCAL 是指读取缓存在本地节点的数据
- NODE\_LOCAL 是指读取本地节点硬盘数据
- ANY 是指读取非本地节点数据
- 通常读取数据 PROCESS\_LOCAL > NODE\_LOCAL > ANY，尽量使数据以 PROCESS\_LOCAL 或 NODE\_LOCAL 方式读取。其中 PROCESS\_LOCAL 还和 cache 有关。

26	60	0	SUCCESS	PROCESS_LOCAL	hadoop3	2014/09/21 20:52:44	0.2 s			164.0 MB (memory)	52.0 B	
30	61	0	SUCCESS	PROCESS_LOCAL	hadoop1	2014/09/21 20:52:44	0.2 s			164.5 MB (memory)	52.0 B	
27	62	0	SUCCESS	PROCESS_LOCAL	hadoop3	2014/09/21 20:52:44	0.2 s			164.3 MB (memory)	52.0 B	
33	63	0	SUCCESS	PROCESS_LOCAL	hadoop1	2014/09/21 20:52:45	0.2 s			164.6 MB (memory)	52.0 B	
29	64	0	SUCCESS	NODE_LOCAL	hadoop3	2014/09/21 20:52:48	13 s	36 ms		128.0 MB (hadoop)	52.0 B	
23	65	0	SUCCESS	NODE_LOCAL	hadoop2	2014/09/21 20:52:48	10 s	45 ms		128.0 MB (hadoop)	52.0 B	
24	66	0	SUCCESS	ANY	hadoop1	2014/09/21 20:52:52	14 s	44 ms		128.0 MB (hadoop)	52.0 B	
25	67	0	SUCCESS	NODE_LOCAL	hadoop2	2014/09/21 20:52:58	8 s	41 ms		128.0 MB (hadoop)	52.0 B	
32	68	0	SUCCESS	NODE_LOCAL	hadoop3	2014/09/21 20:53:01	13 s	56 ms		128.0 MB (hadoop)	52.0 B	
28	69	0	SUCCESS	NODE_LOCAL	hadoop2	2014/09/21 20:53:06	13 s	40 ms		128.0 MB (hadoop)	52.0 B	
31	70	0	SUCCESS	ANY	hadoop1	2014/09/21 20:53:10	15 s	37 ms		128.0 MB (hadoop)	52.0 B	
34	71	0	SUCCESS	ANY	hadoop3	2014/09/21 20:53:14	13 s	53 ms		137.1 MB (hadoop)	52.0 B	

## 2.2.2 合适的数据类型

对于要查询的数据，定义合适的数据类型也是非常有必要。对于一个 tinyint 可以使用的数据列，不需要为了方便定义成 int 类型，一个 tinyint 的数据占用了 1 个 byte，而 int 占用了 4 个 byte。也就是说，一旦将这数据进行缓存的话，内存的消耗将增加数倍。在 SparkSQL 里，定义合适的数据类型可以节省有限的内存资源。

## 2.2.3 合适的数据列

对于要查询的数据，在写 SQL 语句的时候，尽量写出要查询的列名，如 Select a,b from tbl，而不是使用 Select \* from tbl；这样不但可以减少磁盘 IO，也减少缓存时消耗的内存。

## 2.2.4 优的数据存储格式

在查询的时候，最终还是要读取存储在文件系统中的文件。采用更优的数据存储格式，将有利

于数据的读取速度。查看 SparkSQL 的 Stage，可以发现，很多时候，数据读取消耗占有很大的比重。对于 sqlContext 来说，支持 textFile、SequenceFile、ParquetFile、jsonFile；对于 hiveContext 来说，支持 AvroFile、ORCFile、Parquet File，以及各种压缩。根据自己的业务需求，测试并选择合适的数据存储格式将有利于提高 SparkSQL 的查询效率。

## 2.3 内存的使用

spark 应用程序最纠结的地方就是内存的使用了，也是最能体现“细节是魔鬼”的地方。Spark 的内存配置项有不少，其中比较重要的几个是：

- SPARK\_WORKER\_MEMORY 在 conf/spark-env.sh 中配置 SPARK\_WORKER\_MEMORY 和 SPARK\_WORKER\_INSTANCES，可以充分的利用节点的内存资源， $SPARK\_WORKER\_INSTANCES * SPARK\_WORKER\_MEMORY$  不要超过节点本身具备的内存容量；
- executor-memory，在 spark-shell 或 spark-submit 提交 spark 应用程序时申请使用的内存数量；不要超过节点的 SPARK\_WORKER\_MEMORY；
- spark.storage.memoryFraction spark 应用程序在所申请的内存资源中可用于 cache 的比例
- spark.shuffle.memoryFraction spark 应用程序在所申请的内存资源中可用于 shuffle 的比例

在实际使用上，对于后两个参数，可以根据常用查询的内存消耗情况做适当的变更。另外，在 SparkSQL 使用上，有几点建议：

- 对于频繁使用的表或查询才进行缓存，对于只使用一次的表不需要缓存；
- 对于 join 操作，优先缓存较小的表；
- 要多注意 Stage 的监控，多思考如何才能更多的 Task 使用 PROCESS\_LOCAL；
- 要多注意 Storage 的监控，多思考如何才能 Fraction cached 的比例更多

### Storage

RDD Name	Storage Level	Cached Partitions	Fraction Cached	Size in Memory	Size in Tachyon	Size on Disk
HiveTableScan [dt#1,webSession#2,word#3,s_seq#4,c_seq#5,website#6]. (MetastoreRelation default, sogouq, None), None	Memory Deserialized 1x Replicated	27	77%	4.3 GB	0.0 B	0.0 B

## 2.4 合适的 Task

对于 SparkSQL，还有一个比较重要的参数，就是 shuffle 时候的 Task 数量，通过 spark.sql.shuffle.partitions 来调节。调节的基础是 spark 集群的处理能力和要处理的数据量，

spark 的默认值是 200。Task 过多，会产生很多的任务启动开销，Task 多少，每个 Task 的处理时间过长，容易 straggle。

## 2.5 其他的一些建议

优化的方面的内容很多，但大部分都是细节性的内容，下面就简单地提提：

- 想要获取更好的表达式查询速度，可以将 `spark.sql.codegen` 设置为 `Ture`；
- 对于大数据集的计算结果，不要使用 `collect()`，`collect()` 就结果返回给 driver，很容易撑爆 driver 的内存；一般直接输出到分布式文件系统中；
- 对于 Worker 倾斜，设置 `spark.speculation=true` 将持续不给力的节点去掉；
- 对于数据倾斜，采用加入部分中间步骤，如聚合后 `cache`，具体情况具体分析；
- 适当的使用序化方案以及压缩方案；
- 善于利用集群监控系统，将集群的运行状况维持在一个合理的、平稳的状态；
- 善于解决重点矛盾，多观察 Stage 中的 Task，查看最耗时的 Task，查找原因并改善；