Spark SQL：基于内存的大数据处理引擎

文 / 高彦杰，陈冠诚

作为Shark的下一代技术，Spark SQL的性能已完全超过Shark，且由于底层机制相同，用户可以做到无缝迁移，而受到用户的青睐。本文将深入分析Spark SQL架构思路和优化策略，并与同类产品进行比较。

在刚刚结束的Spark Summit 2014上，Databricks宣布不再支持Shark的开发，全力以赴开发Shark的下一代技术Spark SQL，同时Hive社区也启动了Hive on Spark项目，将Spark作为除MapReduce和Tez之外的新执行引擎。根据Big Data Benchmark测试对比数据，Shark的In Memory性能可达到Hive的100倍，即使是On Disk也能达到10倍的性能提升，是Hive的强有力的替代解决方案。而作为Shark进化版本的Spark SQL，在最新测试中的性能已超过Shark。本文中，统称Spark SQL、Shark和Hive on Spark为SQL on Spark。虽然Shark不再开发，但其架构和优化仍有借鉴意义，因此文章中也会有所涉及。

为什么使用Spark SQL

由于Shark底层依赖于Hive，所以这个架构的优势是传统Hive用户可以将Shark无缝集成进现有系统运行查询负载。但也有一些问题：随着版本升级，查询优化器依赖于Hive，不方便添加新的优化策略，需要进行另一套系统的学习和二次开发，学习成本很高。另一方面，MapReduce是进程级并行，如Hive在不同的进程空间会使用一些静态变量，当在同一进程空间进行多线程并行执行时，多线程同时写同名称的静态变量会产生一致性问题，所以Shark需要使用另一套独立维护的Hive源码分支。为了解决这个问题，AMPLab和Databricks利用Catalyst开发了Spark SQL。在Spark1.0版本中已发布Spark SQL。

机器学习、图计算、流计算如火如荼的发展和流行吸引了大批学习者，那为什么还要重视在大数据环境下使用SQL呢？主要有以下几点原因。

- 易用性与用户惯性。在过去很多年中，有大批程序员的工作是围绕着“DB+应用”的架构来做的，SQL的易用性提升了应用的开发效率。程序员已习惯业务逻辑代码调用SQL的模式去写程序。提供SQL和JDBC的支持会让传统用户像以前一样地编写程序，大大减少了迁移成本。
- 生态系统的力量。很多系统软件性能好，但未取得成功和没落，很大程度上因为生态系统问题。传统的SQL在JDBC、ODBC等标准下形成了一套成熟的生态系统，很多应用组件和工具可以迁移使用，如一些可视化工具、数据分析工具等，原有企业的IT工具可以无缝过渡。
- 数据解耦，Spark SQL正在扩展支持多种持久化层，用户可使用原有的持久化层存储数据，也可体验和迁移到Spark SQL提供的数据分析环境。

Spark SQL架构分析

Spark SQL与传统“DBMS查询优化器+执行器”的架构较为类似，只不过其执行器是在分布式环

境中实现的，并采用Spark作为执行引擎。Spark SQL的查询引擎是Catalyst，其基于Scala语言开发，能灵活利用Scala原生的语言特性方便地进行功能扩展，奠定了Spark SQL的发展空间。Catalyst将SQL语言翻译成最终的执行计划，并在这个过程中进行查询优化。与传统方法的区别在于，SQL经过查询优化器最终转换为可执行的查询计划是一个查询树，传统DB可以执行这个查询计划，而Spark SQL会在Spark内将这棵执行计划树转换为有向无环图（DAG）再进行执行。

Catalyst架构及执行流程分析

图1中是Catalyst的整体架构，可以看到Catalyst是Spark SQL的调度核心，它遵循传统数据库的查询解析步骤对SQL进行解析，转换为逻辑查询计划和物理查询计划，最终转换为DAG进行执行（图2）。

Spark SQL优化策略

除了查询优化，Spark SQL在存储上也是进行了优化，下面看看Spark SQL的优化策略。

内存列式存储与内存缓存表

Spark SQL可以通过cacheTable将数据存储转换为列式存储，同时将数据加载到内存进行缓存。cacheTable相当于分布式集群的内存物化视图，将数据进行缓存，这样迭代的或者交互式的查询不用再从HDFS读数据，直接从内存读取数据大大减少了I/O开销。列式存储的优势在于Spark SQL只需要读出用户需要的列，而不需要像行存储那样需要每次将所有列读出，从而大大减少内存缓存数据量，更高效地利用内存数据缓存，同时减少网络传输和I/O开销。并且由于是数据类型相同的数据连续存储，能够利用序列化和压缩减少内存空间的占用。

列存储压缩

为了减少内存和硬盘空间占用，Spark SQL采用了一些压缩策略对内存列存储数据进行压缩。Spark SQL的压缩方式要比Shark丰富很多，例如它支持PassThrough、RunLengthEncoding、DictionaryEncoding、BooleanBitSet、IntDelta和LongDelta等多种压缩方式。这样能够大幅度减少

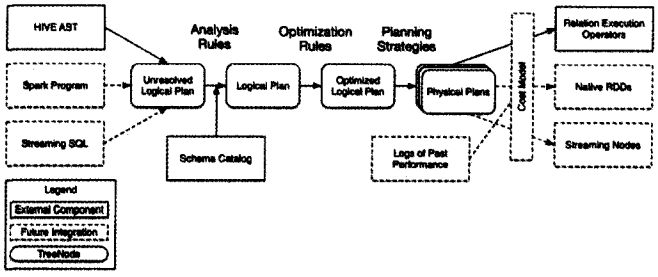

图1 Spark SQL查询引擎Catalyst的架构

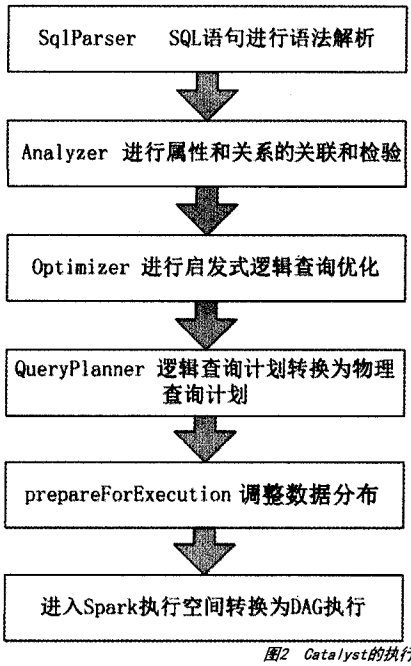

图2 Catalyst的执行流程

内存空间占用、网络传输开销和I/O开销。

逻辑查询优化

Spark SQL在逻辑查询优化上支持列剪枝、谓词下压和属性合并等方法。为了减少读取不必要的属性列及数据传输和计算开销，在查询优化器进行转换的过程中会进行列剪枝优化。

图3是逻辑查询的流程图。下面我们介绍一个逻辑优化例子：

```
SELECT Class FROM (SELECT ID,Name,Class
FROM STUDENT ) S WHERE S.ID=1
```

通过谓词下压，Catalyst优先执行选择操作ID=1，以过滤大部分数据。然后，通过属性合并，最后的投影将只做一次，并最终保留Class属性列。

Join优化

Spark SQL深度借鉴传统数据库的查询优化技术的精髓，同时也在分布式环境下进行特定的

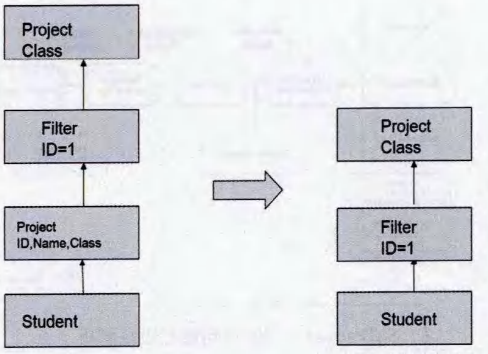

图3 逻辑查询优化

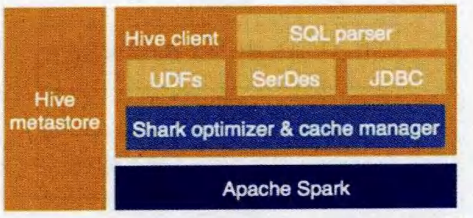

图4 Shark架构

优化策略调整和创新。Spark SQL对Join进行了优化，以支持多种连接算法，因此其连接算法比Shark丰富，且很多原来Shark的元素也逐步迁移过来，如BroadcastNestedLoopJoin、HashJoin和LeftSemiJoin等。

下面介绍其中一个Join算法——BroadcastHashJoin，它将小表转化为广播变量进行广播，以避免Shuffle开销，最后在分区内做Hash连接。这里用的就是Hive中Map Side Join的思想。同时用了DBMS中的Hash连接算法做连接。

随着Spark SQL的发展，未来会有更多的查询优化策略加入进来。同时后续Spark SQL会支持像Shark Server一样的服务端及JDBC接口，兼容更多的持久化层，如NoSQL和传统的DBMS等。

如何使用Spark SQL

```
val sqlContext = new org.apache.spark.sql.
SQLContext(sc)
// 在这里引入 sqlContext 下所有的方法我们就可以直
接用 sql 方法进行查询
import sqlContext._
case class Person(name: String, age: Int)
// 下面的 people 是含有 case 类型数据的 RDD，会默认
由 Scala 的 implicit 机制将 RDD 转换为 SchemaRDD，
SchemaRDD 是 SparkSQL 中的核心 RDD
val people = sc.textFile("examples/
src/main/resources/people.txt").map(_.
split(",")).map(p => Person(p(0), p(1).trim.
toInt))
// 在内存的元数据中注册表信息
people.registerAsTable("people")
//sql 语句就会触发上面分析的 Spark SQL 的执行过程
val teenagers = sql("SELECT name FROM
people WHERE age >= 13 AND age <= 19")
// 最后生成的 teenagers 也是一个 RDD
teenagers.map(t => "Name: " + t(0)).
collect().foreach(println)
```

下面我们看下Shark的原理，虽然Shark已完成学术使命终止开发，但其中的架构和优化策略还是有借鉴意义的。

Shark

图4是Shark的架构。在整体架构中Shark复用了Hive Metastore和Hive SerDe以及查询解析器和优化器，但用Spark重写了Hive的执行Operator，并实现了基于内存的优化策略。最初为了缩短开发周期，Shark复用Hive的查询优化器。但这样不得不维护一个单独的Hive分支用来支持Shark。随着系统复杂性提升及优化策略的不断扩充，维持Hive的查询优化器代价越来越大，最终Databricks宣布终止Shark开发。

执行流程

Shark读取用户的查询表达式，运用Hive的解析器和查询优化器形成查询树进行语法解析和逻辑物理优化，最终形成等待执行的执行计划。执行器遍历执行计划树到叶子节点的Operator进行执行，执行后再回溯到父母节点继续执行，直到完全执行完整个查询计划。Operator中不再用Hadoop的MapReduce进行分布式计算，而是用Spark重写Operator进行分布式计算。

容错性

Spark记录了RDD的Lineage，也就是RDD的依赖关系，功能类似于传统数据库中redo日志。当有分区丢失或者出错时，Spark可以从源头的基础数据重做运算恢复分区数据。这也是和Impala进行对比的一个优势——在任务失败时，Impala需要整体重做全部任务。

多数据计算范式混合

与其他SQL on Hadoop产品对比，Shark一大优势还源于其可以和其他多种计算范式混合计算。当用Shark通过SQL建立内存表时，既可以通过MLlib进行机器学习的运算，也可以用GraphX进行大规模图计算等。这样能让用户方便地进行一站式的数据流水线计算，而不需要有一个持久化层（如HDFS）进行中间数据暂存。这无疑会大幅度减少性能开销，同时提升开发效率和复杂度，更减少了不同系统间兼容的代价。

下面我们看一个SQL和机器学习结合的例子：进入Spark Shell进行交互式查询，方便用户将想法迅速变现。

```
$./bin/shark-shell
// 通过 SparkContext 的 SQL2rdd 方法运行 SQL 查询，
从 Shark 中读出表并在内存建立 RDD
scala> val youngUsers = sc.SQL2rdd("SELECT
* FROM users WHERE age < 20")
// 对内存 youngUsers RDD 进行 map 操作，提取数据
中的 feature，并转化为可以用于机器学习的数据格式
scala> val featureMatrix = youngUsers.
map(extractFeatures(_))
// 调用 MLlib 中的 kmeans 进行用户数据聚类
scala> kmeans(featureMatrix)
```

综合看来，Spark SQL和Shark相比其他SQL on Hadoop产品有以下的几点优势。

- 依托内存计算框架Spark，利用内存计算大幅度提升性能。
- 支持Spark Shell做交互式查询，用户想法可以快速变现。
- 依托Spark生态系统，可以方便构建全栈数据解决方案。

Hive on Spark

随着Hive on Spark的立项，未来的Hive会支持MapReduce、Tez和Spark三大执行引擎。相比Shark和Spark SQL，Hive on Spark会全面支持现有Hive，也就意味着原来使用Hive的用户可以无缝过渡，数据不需要迁移，原来针对业务逻辑写的Hive QL脚本不需要重写，只是换了个更加快速的执行引擎。语法上支持全部的Hive QL语法和扩展特性，同时会集成Hive的权限管理、运行监控、审计和其他基于Hive的管理工具。基于Hive的生态系统的组件可以过渡到Spark执行引擎使用。

Hive on Spark的设计方向及潜在问题如下。

- Shuffle和Join：由于Spark的Shuffle是不进行分组排序的，所以Hive的Join是基于MapReduce的Shuffle来做的MapSideJoin和ReduceSideJoin，Spark社区已开始发起改变或者提供相应的Shuffle API。同时原有的Hive Join算法会迁移过来。
- 线程安全：Spark执行任务和分区是在一个JVM空间执行多线程，而传统Hive的Map端操作树或者Reduce端操作树将任务的每个线程分在不同的JVM，因为Hive的操作中有静态变量，这样就会产生并发和线程安全问题。这里也是需要重新设计的地方。
- Java API：Hive on Spark需要社区提供Job监控和RDD扩展的API，这样就能够和原有组件融合。

未来展望

Spark SQL提供了对RDD的SQL支持，同时支持其他的数据源，例如Parquet文件和Hive表。统一这些强大的数据存储模型能够让用户更方便地进行复杂的数据分析。统一的Spark数据平台能够让用户选择需要的工具去处理数据，而不需要再构建另一套系统。未来Databricks会继续在Spark SQL生成自定义字节码加速表达式解析，提供更多数据源的支持如Avro和HBase及更丰富的语言API。Databricks和AMPLab会继续投资Spark SQL希望使其成为结构化数据分析的标准。

高彦杰

中国人民大学计算机系硕士研究生，IBM中国研究院实习生，Spark SQL Contributor，研究方向为Spark、查询优化和Benchmark。微信公共号：Spark大数据。新浪微博：@高彦杰gyj

陈冠诚

IBM中国研究院高级研究员，大数据项目经理，主要技术方向为大数据系统的软硬件协同设计和优化。个人博客为并行实验室（www.parallellabs.com），新浪微博：@冠诚

责任编辑：杨爽（yangshuang@csdn.net）