Spark SQL 分享

Baofeng Zhang

Previously, I …

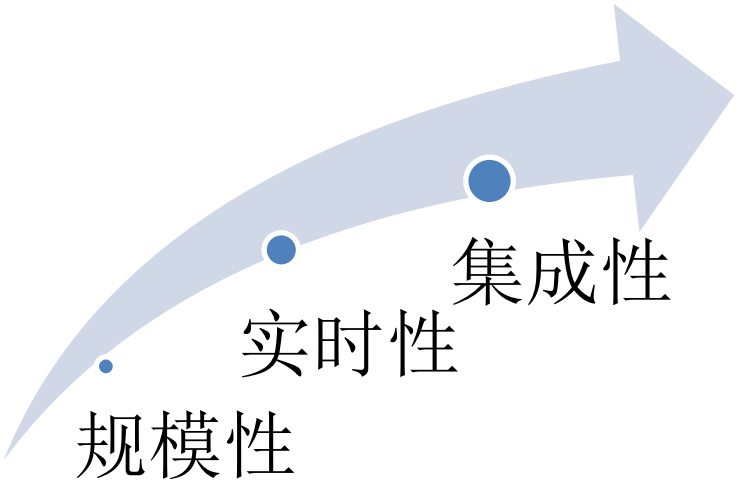
- 2013.8, Spark Ecosystem
  - Spark 0.7.2, Shark 0.7, Mesos 0.9.0
- 2014.3, Flare Project
  - Pig, Hive, Spark Sql各个层次的实现
- 2014.7, 业务推广
  - Spark Sql & Streaming
- 2014.8, 增量计算引擎
  - SQL, etc.

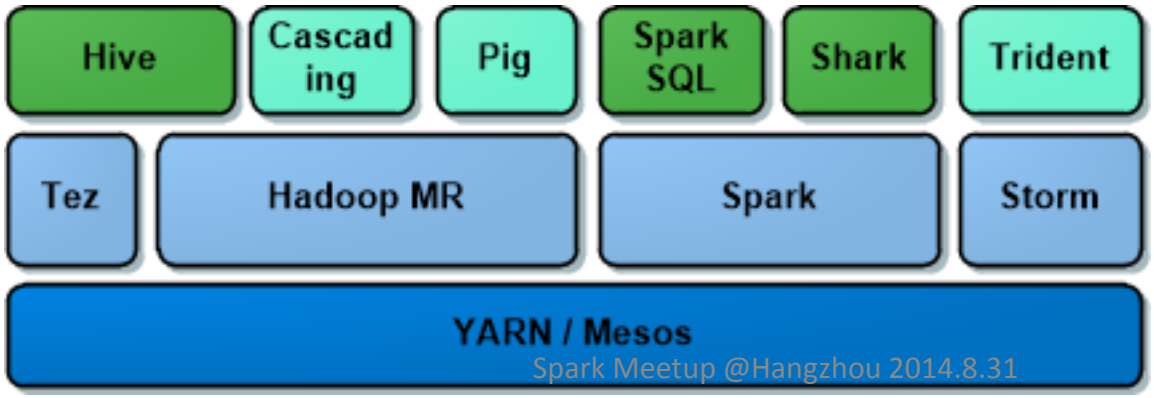
计算框架与算子层

DSL ON FRAMEWORKS

Distributed SQL

- Parallel, Shared-nothing DB/DW
  - GreenPlum, InfoBright, Vertica
- Sql on Hadoop
  - Hive, Impala
- Other System
  - Tenzing, Drayd, HadoopDB
- Frameworks
  - Spark, Storm, Tez, Tajo

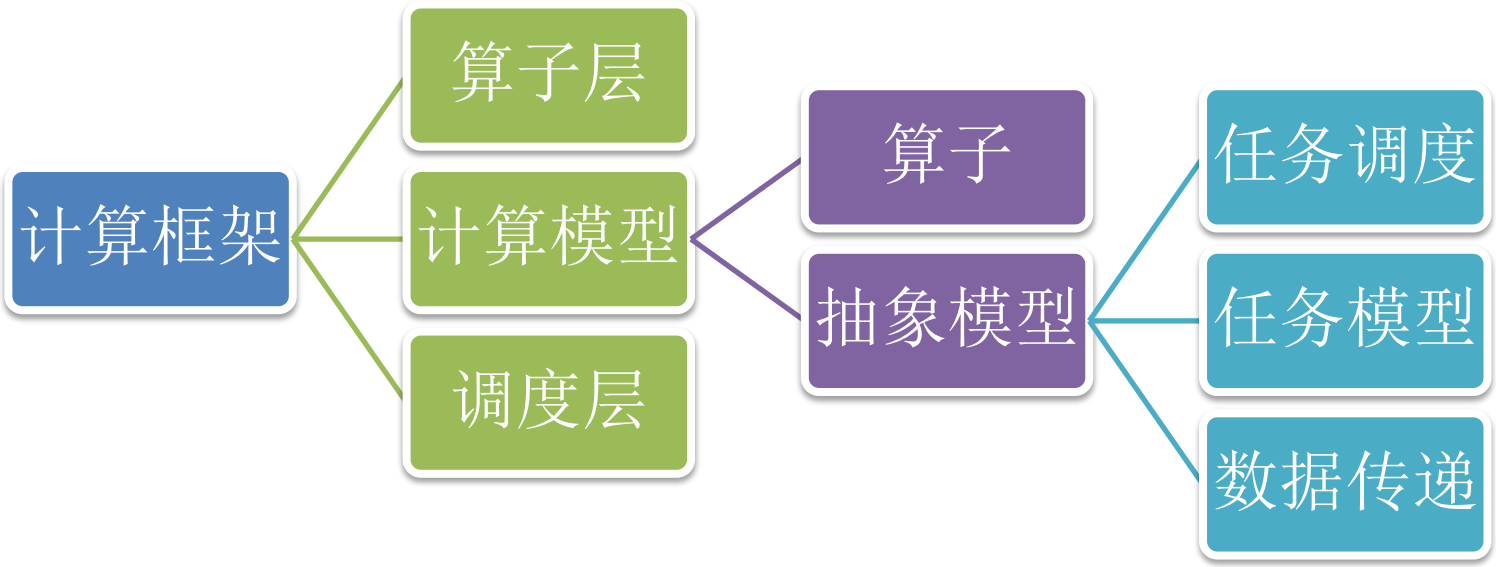




Point of View

- 资源管理层及存储层基本上被YARN和HDFS及辅助列式存储统一

- 计算框架，尤其是实时计算方向，还在不断革新



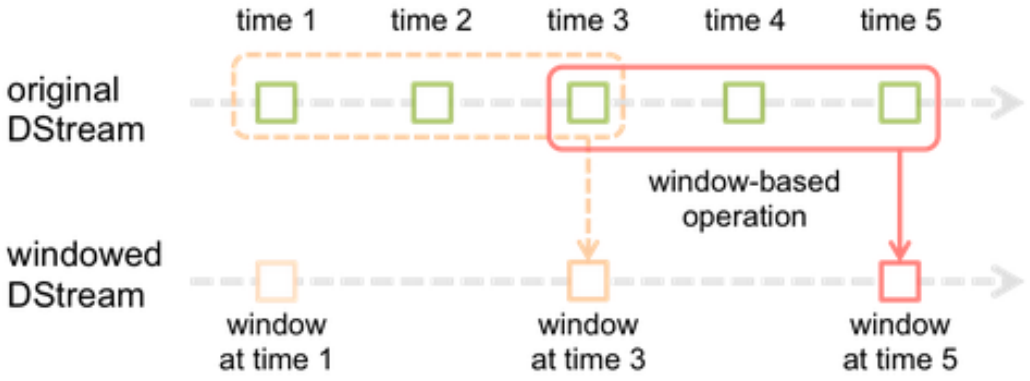
计算模型

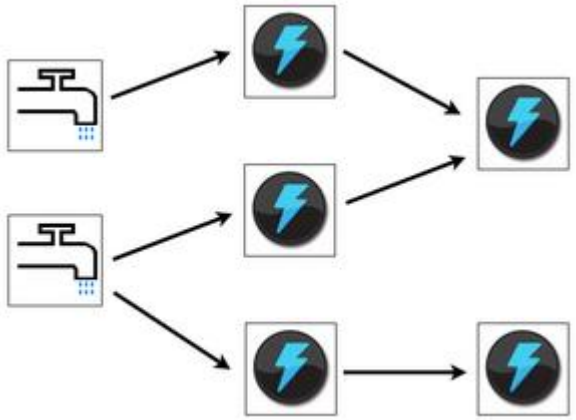
- 计算模型 = 抽象模型 + 算子
  - 抽象模型: Topology, DAG
  - 基础算子: map, groupByKey, reduce, union
  - 高级算子: count, sum, join, filter, sort, top => DSL, ML, Graph
  - 高级抽象算子: each, groupBy, aggregate, shuffle
- “计算模型”
  - MapReduce, MapOnly, MSCR
  - 计算框架的优化以计算模型为单位展开
- 抽象模型 = 任务调度 + 任务模型 + 数据传递
  - 任务模型
    - Pipeline；计算框架内的容错
  - 数据传递(RPC & Channel)
    - Pull方式: 运行时, 容错性好
    - Push方式: 编译时, 实时性好
- 计算场景
  - 实时计算，离线计算
  - 批量计算，流式计算 (加上事务和容错会怎么样?)
  - 全量计算，增量计算
  - 持续计算，迭代计算

As a Framework

~~Fault tolerant, Storage, Scheduler~~

- Spark
  - 与 Hadoop 同为MapReduce的实现
  - 优势: Memory, DAG, **RDD**, Iterative
- Incremental Computing
  - 可以支持Batch, Streaming, Iterative
  - Spark Streaming是小批计算



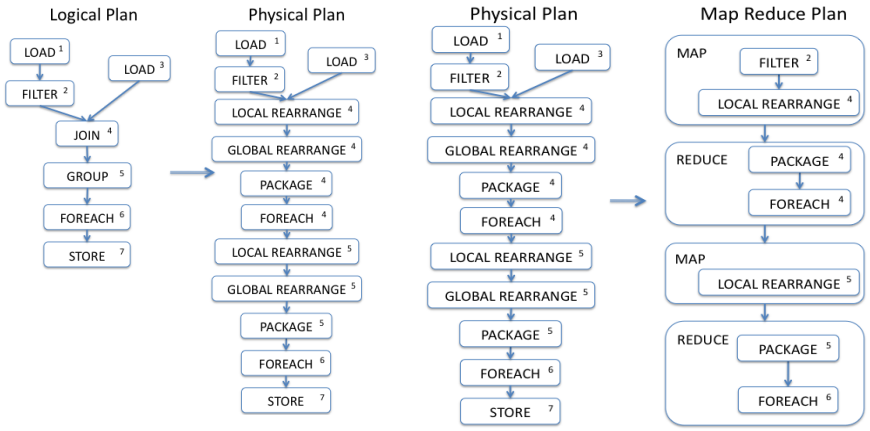


RDD

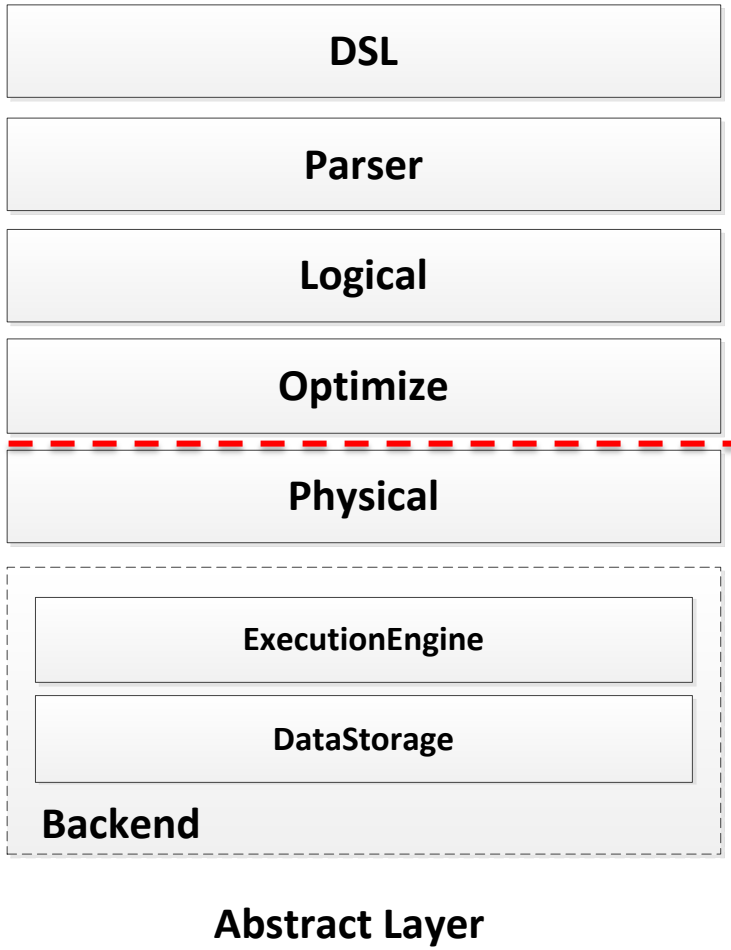
- 算子层
  - Primitives: transforms & actions
    - 组合算子如distinct, intersection
  - 类比FlumeJava:
    - Pcollection<T>, PTable<K, V>
    - parallelDo(), groupByKey(), combineValues(), flattern()
  - Trident更抽象:
    - Stream, TridentState
    - each(), shuffle(), groupBy(), aggregate(), stateQuery()
- 数据表示
  - 计算, 关联, 依赖, 分区, 感知, 缓存
    - Iterator, Lineage, Dependency, Partitions, Location-awareness, StorageLevel
  - Batch之间的计算结果形态
    - 迭代计算
- 衍生成特殊操作对象
  - SchemaRDD, DStream, RDD in Mllib/GraphX, RDD ⇔ Files

DSL on Spark

- Hive/Pig on Spark
  - Backend Engine的替换
  - 物理层 MRCompiler -> SparkCompiler



- SQL-Like / SQL on Catalyst
  - Spark Sql
  - Spark hive

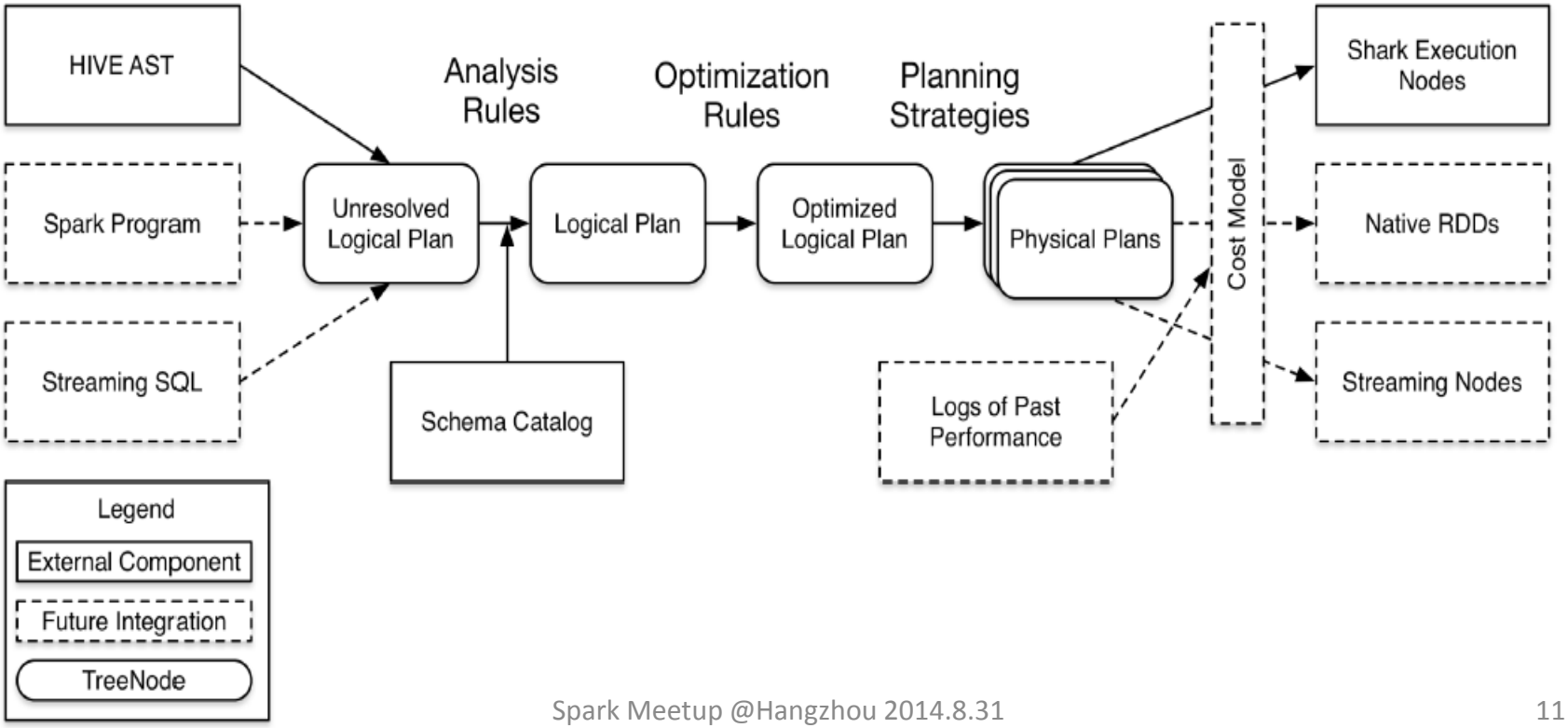


架构与设计

SPARK SQL

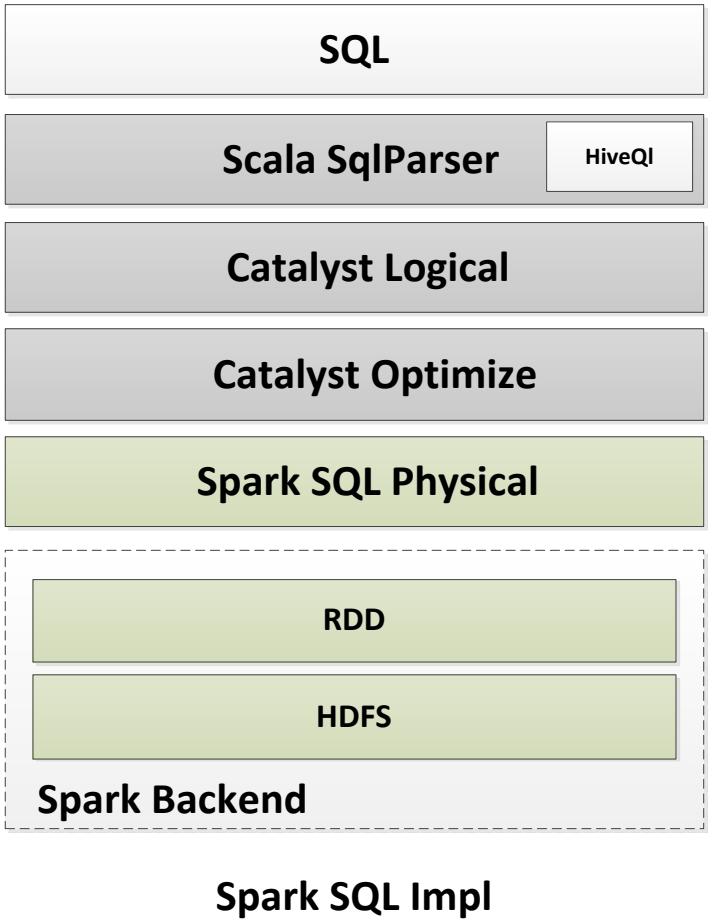
Catalyst

- 与Spark解耦, Scala写成的框架
- 提供类结构，执行计划表示模型和规则处理规范



Layers

- 语言
  - SQL92 vs. Pig-latin
- 词法语法解析
  - Simple Sql Parser vs. Antlr
- 执行计划表示
  - TreeNode lib vs. Hive AST
- 基于规则的优化
  - 优化规则有限，可扩展
    - Rule-based (编译时，逻辑层面)
    - Cost-based (运行时，物理层面)
    - 传统数据库非常完善



SchemaRDD

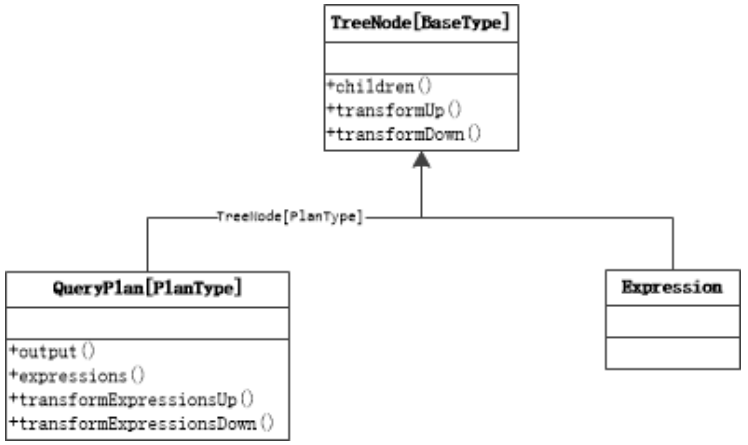
- 功能定位
  - 为泛型RDD的数据关联Columns => “Table”
  - Column = 列名 + 数据类型
- 数据来源
  - 已有RDD
  - 其他支持: Parquet, JSON, etc.
- Schema来源
  - 显式指定 => Reflect from Case Class, Hive metadata
  - 非显式的 => JsonRDD.inferSchema
- 数据表示
  - GenericRow: Seq[Any]
- Schema传递性
  - references: Set[Attribute] 携带在LogicalPlan内
- 提供额外的DSL方法
  - select(), where(), join(), limit(), orderBy(), etc.
  - 更具化的”算子层”

SQLContext

- 功能定位
  - 整体Sql执行流程的控制
  - 读取不同格式文件转换为SchemaRDD
  - Cache/Uncache Table
- 代码层面
  - 内部成员SparkContext
  - 暴露sql()方法, 返回SchemaRDD
  - 子类HiveContext
    - 暴露hql()
    - Hive客户端交互

Catalyst - TreeNode

- 二叉树
  - 集合操作能力: foreach, collect, map
  - 遍历能力: transformDown, transformUp
    - 满足PartialFunction(制定规则)的节点被替换
  - 遍历、递归 => 替换(类比Pig访问者模式)
- 执行计划体系
  - QueryPlan执行计划表示
    - LogicalPlan逻辑执行计划
    - SparkPlan物理执行计划
  - Expression表达式
    - Cast, Projection, 四则运算, 逻辑操作符运算
    - 简单表达式可在优化过程中可预处理完



LogicalPlan & SparkPlan & Rules

- LogicalPlan及代表性子类
  - LeafNode:　Command体系(non-query)
  - UnaryNode: Distinct, Filter, Limit, Project, Sort, Aggregate
  - BinaryNode: Join, Union
- SparkPlan与物理引擎有关
  - 执行引擎(如Spark Sql)制定子类
  - 实现execute()方法
  - Catalyst分区模型
- RuleExecutor
  - Seq[Batch], Batch is (name, strategy, rules: Rule[TreeType]*)
  - Rule的子类复写apply()逻辑
- DOT representation

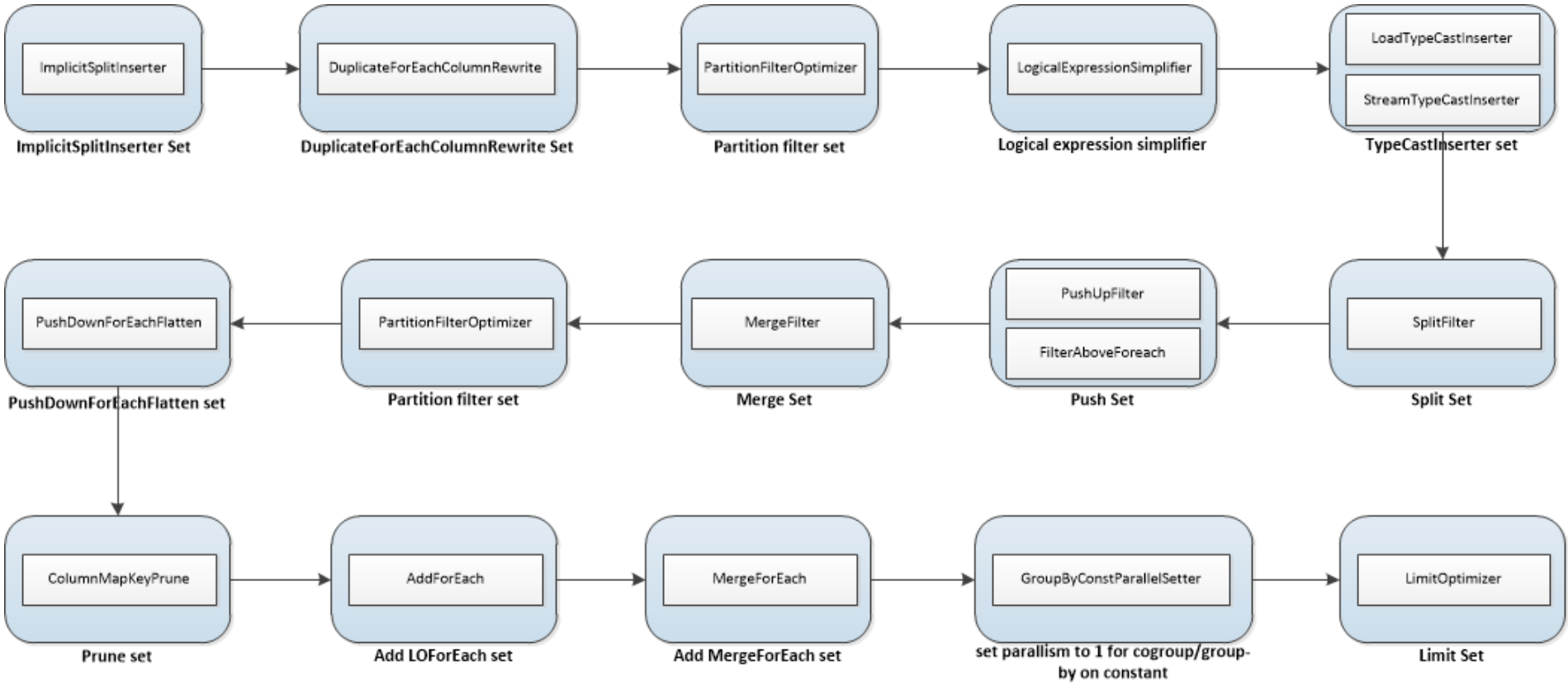
执行计划处理

- Analyzer
  - 通过Catalog和FunctionRegistry来翻译Unresolved Attributes/Relations
    - Reference(表达式), Relation(表与列), Function(UDF)
- Optimizer
  - 三套优化规则
    - CombineLimits, ConstantFolding, Filter Pushdown
- QueryPlanner
  - 实现逻辑在Spark Sql项目的SparkStrategy内
  - 简单逻辑, 如basicOperation, 直接映射
    - Distinct, Sort, Project, Filter, Aggregate, Limit
  - 复杂逻辑
    - TopK, CartesianProduct, PartialAggregate, SparkEquiInnerJoin

Optimization is Complicated!

~~I know little and just don't care~~

- Pig`s Optimization Rules



SQL

```
select distinct st.sno, sname
from student st
  join score sc on (st.sno = sc.sno)
where sc.cno in ('c001', 'c002', 'c003') and  sc.sno<>'s001'
```

Logical

```
Distinct
 Project ['st.sno,'sname]
  Filter ('sc.cno IN (c001,c002,c003) && NOT ('sc.sno = s001))
    Join Inner, Some(('st.sno = 'sc.sno))
      UnresolvedRelation None, student, Some(st)
      UnresolvedRelation None, sc, Some(sc)
```

Analyzed

```
Distinct
 Project [sno#0,sname#1]
  Filter (cno#10 IN (c001,c002,c003) && NOT (sno#9 = s001))
    Join Inner, Some((sno#0 = sno#9))
      SparkLogicalPlan (ExistingRdd [sno#0,sname#1,sage#2,ssex#3], MapPartitionsRDD[2])
      SparkLogicalPlan (ExistingRdd [sno#9,cno#10,score#11], MapPartitionsRDD[14])
```

Optimized

```
Distinct
  Project [sno#0,sname#1]
    Join Inner, Some((sno#0 = sno#9))
      Project [sno#0,sname#1]
        SparkLogicalPlan (ExistingRdd [sno#0,sname#1,sage#2,ssex#3], MapPartitionsRDD[2])
      Project [sno#9]
        Filter (cno#10 IN (c001,c002,c003) && NOT (sno#9 = s001))
          SparkLogicalPlan (ExistingRdd [sno#9,cno#10,score#11], MapPartitionsRDD[14])
```

Physical

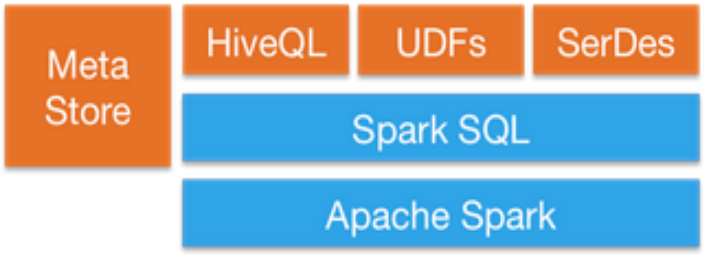
```
Aggregate false, [sno#0,sname#1], [sno#0,sname#1]
  Project [sno#0,sname#1]
    HashJoin [sno#0], [sno#9], BuildRight
      Project [sno#0,sname#1]
        ExistingRdd [sno#0,sname#1,sage#2,ssex#3], MapPartitionsRDD[2])
      Project [sno#9]
        Filter (cno#10 IN (c001,c002,c003) && NOT (sno#9 = s001))
          ExistingRdd [sno#9,cno#10,score#11], MapPartitionsRDD[14])
```

Summary

	Spark SQL	Hive Support
Parser 词法语法解析，产出语法树	Catalyst SqlParser	HiveQl.parseSql
Analyzer 解析出初步的逻辑执行计划	Catalyst Analyzer	Hive MetaStore & Hive UDFs
Optimizer 逻辑执行计划优化	Catalyst Optimizer	Catalyst Optimizer
QueryPlanner 逻辑执行计划映射物理执行计划	spark-sql SparkPlanner & SparkStrategy	HivePlanner
Execute 物理执行计划树触发计算	spark-sql SparkPlan.execute()	SparkPlan.execute()

~~Shark~~

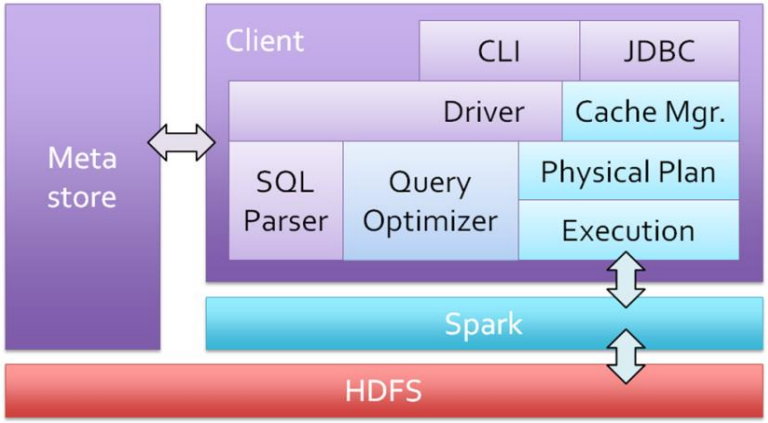
- Spark-hive项目
  - 复用Metadata, SerDes, UDF/UDAF/UDTF
  - Query无关的cmd走Hive Client
  - HiveContext.hql()
  - 重点在ASTNode <-> LogicalPlan
  - 目的在数据层面的”整合”



Spark SQL can use existing Hive metastores, SerDes, and UDFs.

- Hive on Spark
  - Spark as Backend Engine
  - 出世前还得用Shark
  - 目的在Hive用户”舒适度”

- Shark在Spark Sql的传承体现
  - Cache Mgr.
  - CLI & JDBC



Features

- 内存列存储(Shark)
  - 行转列；多种压缩方式(ratio>0.8)；Structed ByteBuffer
- UDF, UDAF, UDTF
  - 目前Spark Sql的实现不支持UDF
  - Analyzer步骤: FunctionRegistry.lookupFunction()
- Hive, Parquet, JSON, etc.
  - 尝试过ORCFile
- JDBC & CLI
  - 参考Hive CLI实现

使用场景

- 实时性, Sql on Hadoop替代者?
  - 数据仓库(ETL层Shark)
  - 数据平台(BI, 其他数据服务)
- 集成性, Lamda Architecture?
  - Streaming中融入SQL，表达方便
  - ML中融入 SQL

Q & A

Thx