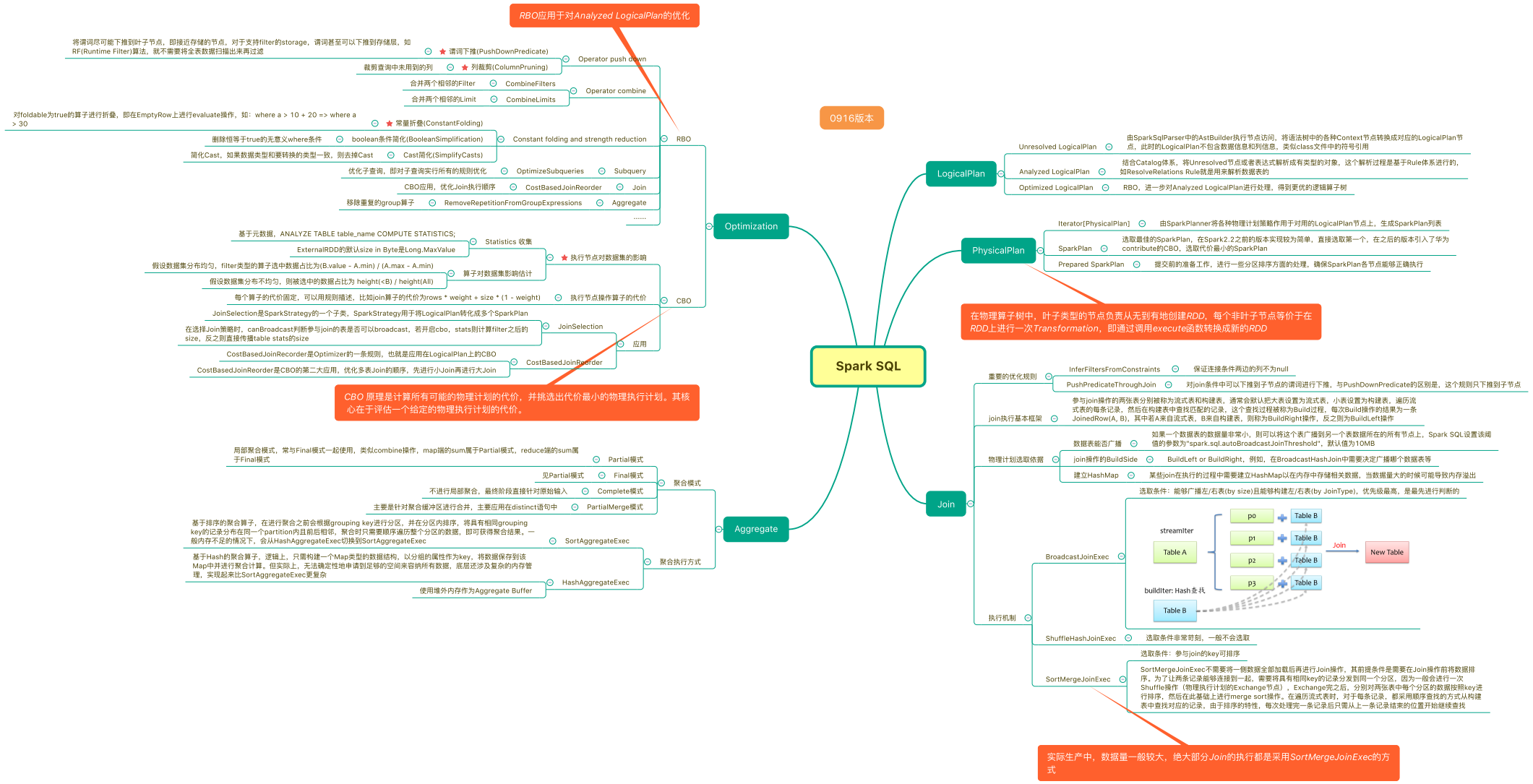
Spark SQL
0916版本
Optimization
RBO应用于对Analyzed LogicalPlan的优化
RBO
Operator push down
★ 谓词下推(PushDownPredicate)
将谓词尽可能下推到叶子节点，即接近存储的节点，对于支持filter的storage，谓词甚至可以下推到存储层，如RF(Runtime Filter)算法，就不需要将全表数据扫描出来再过滤
★ 列裁剪(ColumnPruning)
裁剪查询中未用到的列
Operator combine
CombineFilters
合并两个相邻的Filter
CombineLimits
合并两个相邻的Limit
Constant folding and strength reduction
★ 常量折叠(ConstantFolding)
对foldable为true的算子进行折叠，即在EmptyRow上进行evaluate操作，如：where a > 10 + 20 => where a > 30
boolean条件简化(BooleanSimplification)
删除恒等于true的无意义where条件
Cast简化(SimplifyCasts)
简化Cast，如果数据类型和要转换的类型一致，则去掉Cast
Subquery
OptimizeSubqueries
优化子查询，即对子查询实行所有的规则优化
Join
CostBasedJoinReorder
CBO应用，优化Join执行顺序
Aggregate
RemoveRepetitionFromGroupExpressions
移除重复的group算子
.....
CBO
★ 执行节点对数据集的影响
Statistics 收集
基于元数据，ANALYZE TABLE table_name COMPUTE STATISTICS;
ExternalRDD的默认size in Byte是Long.MaxValue
算子对数据集影响估计
假设数据集分布均匀，filter类型的算子选中数据占比为(B.value - A.min) / (A.max - A.min)
假设数据集分布不均匀，则被选中的数据占比为 height(<B) / height(All)
执行节点操作算子的代价
每个算子的代价固定，可以用规则描述，比如join算子的代价为rows * weight + size * (1 - weight)
应用
JoinSelection
JoinSelection是SparkStrategy的一个子类，SparkStrategy用于将LogicalPlan转化成多个SparkPlan
在选择Join策略时，canBroadcast判断参与join的表是否可以broadcast，若开启cbo，stats则计算filter之后的size，反之则直接传播table stats的size
CostBasedJoinReorder
CostBasedJoinRecorder是Optimizer的一条规则，也就是应用在LogicalPlan上的CBO
CostBasedJoinReorder是CBO的第二大应用，优化多表Join的顺序，先进行小Join再进行大Join
CBO 原理是计算所有可能的物理计划的代价，并挑选出代价最小的物理执行计划。其核心在于评估一个给定的物理执行计划的代价。
Aggregate
聚合模式
Partial模式
局部聚合模式，常与Final模式一起使用，类似combine操作，map端的sum属于Partial模式，reduce端的sum属于Final模式
Final模式
见Partial模式
Complete模式
不进行局部聚合，最终阶段直接针对原始输入
PartialMerge模式
主要是针对聚合缓冲区进行合并，主要应用在distinct语句中
聚合执行方式
SortAggregateExec
基于排序的聚合算子，在进行聚合之前会根据grouping key进行分区，并在分区内排序，将具有相同grouping key的记录分布在同一个partition内且前后相邻，聚合时只需要顺序遍历整个分区的数据，即可获得聚合结果。一般内存不足的情况下，会从HashAggregateExec切换到SortAggregateExec
HashAggregateExec
基于Hash的聚合算子。逻辑上，只需构建一个Map类型的数据结构，以分组的属性作为key，将数据保存到该Map中并进行聚合计算。但实际上，无法确定性地申请到足够的空间来容纳所有数据，底层还涉及复杂的内存管理，实现起来比SortAggregateExec要复杂
使用堆外内存作为Aggregate Buffer
LogicalPlan
Unresolved LogicalPlan
由SparkSqlParser中的AstBuilder执行节点访问，将语法树中的各种Context节点转换成对应的LogicalPlan节点，此时的LogicalPlan不包含数据信息和列信息，类似class文件中的符号引用
Analyzed LogicalPlan
结合Catalog体系，将Unresolved节点或者表达式解析成有类型的对象，这个解析过程是基于Rule体系进行的，如ResolveRelations Rule就是用来解析数据表的
Optimized LogicalPlan
RBO，进一步对Analyzed LogicalPlan进行处理，得到更优的逻辑算子树
PhysicalPlan
Iterator[PhysicalPlan]
由SparkPlanner将各种物理计划策略作用于对用的LogicalPlan节点上，生成SparkPlan列表
SparkPlan
选取最佳的SparkPlan，在Spark2.2之前的版本实现较为简单，直接选取第一个，在之后的版本引入了华为contribute的CBO，选取代价最小的SparkPlan
Prepared SparkPlan
提交前的准备工作，进行一些分区排序方面的处理，确保SparkPlan各节点能够正确执行
在物理算子树中，叶子类型的节点负责从无到有地创建RDD，每个非叶子节点等价于在RDD上进行一次Transformation，即通过调用execute函数转换成新的RDD
Join
重要的优化规则
InferFiltersFromConstraints
保证连接条件两边的列不为null
PushPredicateThroughJoin
对join条件中可以下推到子节点的谓词进行下推，与PushDownPredicate的区别是，这个规则只下推到子节点
join执行基本框架
参与join操作的两张表分别被称为流式表和构建表，通常会默认把大表设置为流式表，小表设置为构建表，遍历流式表的每条记录，然后在构建表中查找匹配的记录，这个查找过程被称为Build过程，每次Build操作的结果为一条JoinedRow(A, B)，其中若A来自流式表，B来自构建表，则称为BuildRight操作，反之则为BuildLeft操作
物理计划选取依据
数据表能否广播
如果一个数据表的数据量非常小，则可以将这个表广播到另一个表数据所在的所有节点上，Spark SQL设置该阈值的参数为"spark.sql.autoBroadcastJoinThreshold"，默认值为10MB
join操作的BuildSide
BuildLeft or BuildRight，例如，在BroadcastHashJoin中需要决定广播哪个数据表等
建立HashMap
某些join在执行的过程中需要建立HashMap以在内存中存储相关数据，当数据量大的时候可能导致内存溢出
执行机制
BroadcastJoinExec
选取条件：能够广播左/右表(by size)且能够构建左/右表(by JoinType)，优先级最高，是最先进行判断的
streamIter
Table A
p0
p1
p2
p3
Table B
Join
New Table
buildIter: Hash查找
ShuffleHashJoinExec
选取条件非常苛刻，一般不会选取
SortMergeJoinExec
选取条件：参与join的key可排序
SortMergeJoinExec不需要将一侧数据全部加载后再进行Join操作，其前提条件是需要在Join操作前将数据排序，为了让两条记录能够连接到一起，需要将具有相同key的记录分发到同一个分区，因为一般会进行一次Shuffle操作（物理执行计划的Exchange节点），Exchange完之后，分别对两张表中每个分区的数据按照key进行排序，然后在此基础上进行merge sort操作。在遍历流式表时，对于每条记录，都采用顺序查找的方式从构建表中查找对应的记录，由于排序的特性，每次处理完一条记录后只需从上一条记录结束的位置开始继续查找
实际生产中，数据量一般较大，绝大部分Join的执行都是采用SortMergeJoinExec的方式