Distributed Operator

Assumption

Single Node

- 磁盘IO开销远大于内存计算
- 尽可能减少磁盘IO

Distributed Environment

- 网络开销 > 磁盘IO
- 尽可能减小网络开销，计算逻辑尽可能下放到存储节点
- 在优化生成物理执行计划的时候，应该尽量减少 leader 节点的执行算子复杂度

Physical Operator

单目操作

计算逻辑很容易下推

选择&投影

- 计算下放到存储节点
- 单个节点上可以采用一趟算法
- 结果返回至调度节点并直接返回给用户
- 天然支持流式处理

排序

多路归并排序分布式化

- 单个节点用多路归并排序排好序
- 计算节点维护一个最小K（存储节点数）个缓冲区大小，从每一个存储节点通过RPC读取一个块进入内存
- 类似于二阶段，找到最小的项，进行流式输出

集合

类似于排序

- 单个节点用排序去重算法，生成中间结果，存够一个batch的数据之后发往计算节点
- 计算节点进一步处理收集到的batch，并流式输出内存中小于所有存储节点最近发送的最小项的数据

聚合

不分组

- 在存储节点上进行预聚合生成中间结果发送至计算节点
- 在计算节点上进一步聚合生成最终结果

分组

情况稍微有些复杂

Example

a	b
1	9
1	-8
2	-7
2	6
1	5
2	4

```
select avg(b) from t group by a
```

Hash 聚合

- 维护一个hash表，键为分组的列，值为聚合函数的中间值count和sum

输入数据 `a` `b`	**Hash 表** `[key] (sum, count)`
1 9	`[1] (9, 1)`
1 -8	`[1] (1, 2)`
2 -7	`[1] (1, 2)` `[2] (-7, 1)`
2 6	`[1] (1, 2)` `[2] (-1, 2)`
1 5	`[1] (6, 3)` `[2] (-1, 2)`
2 4	`[1] (6, 3)` `[2] (3, 3)`

- 输入数据输入完后，扫描 Hash 表并计算，便可以得到最终结果：

Hash 表	`avg(b)`
`[1] (6, 3)`	2
`[2] (3, 3)`	1

- 计算可以下推到存储节点，最后由计算节点进行汇总

流式聚合

- 按聚合的列，全局有序/存在索引

输入数据	是否为新 Group 或所有数据输入完成	`(sum, count)`	`avg(b)`	
1 9	是	(9, 1)	前一个 Group 为空，不进行计算	
1 -8	否	(1, 2)		
1 5	否	(6, 3)		
2 -7	是	(-7, 1)	2	
2 6	否	(-1, 2)		
2 4	否	(3, 3)		
	是		1	

- 随机存储
  - 存储节点按照2PMMS进行排序并生成中间结果, 按batch发往计算节点，每一个batch内部是有序的
  - 计算节点仿照单个节点的处理方法，聚合中间结果生成最终结果

比较

- 流式聚合在聚合列存在索引/有序的情况下会很好
- Hash 聚合 性能较差但是不依赖于索引
- Hash 聚合可以采用并行化算法来提高性能

双目操作

由于两个数据表可能分布在不同的机器上，所以导致分布式情况下的双目操作必单机情况下要复杂的多。

将数据统一在计算节点上进行计算是效率不高的一种做法，因为分布式的场景下假设的就是数据量非常大的情况。

另外我们希望是尽可能的利用好存储节点的cpu和内存资源。

连接

基于排序

- 存储节点根据连接属性的值对两个表进行排序，生成中间结果发往计算节点
- 计算节点进一步对两个表排序，并进行连接并输出

基于广播

有某个主节点存储了所有的元信息，知道哪些表存储在哪些节点上，假设表为R和S

- 存储节点先对R和S进行按连接属性进行排序（可选）
- 在主节点的调度下，所有存储了R的节点将数据发往所有存储了S的节点
- 所有存储了S的节点在接收到一部分R时先过滤到范围之外的数据，如何进行连接

- 生成了部分结果之后就发往计算节点

比较

- 基于广播的调度起来可能会比较麻烦，但是可以充分利用存储节点的资源
- 基于排序的方法在存在索引的情况下性能会非常优秀

交&差

基于广播 不支持流式

- 类似于连接操作，将R发往存有S的节点，S的存储节点维护一个计数器
- 在接受完所有R之后，S节点将中间结果+计数器一起发往计算节点进行汇总

基于排序

与之前类似