ClickHouse使用姿势系列之分布式JOIN

JOIN操作是OLAP场景无法绕开的，且使用广泛的操作。对ClickHouse而言，非常有必要对分布式JOIN实现作深入研究。

在介绍分布式JOIN之前，我们看看ClickHouse 单机JOIN是如何实现的。

1. ClickHouse单机JOIN实现

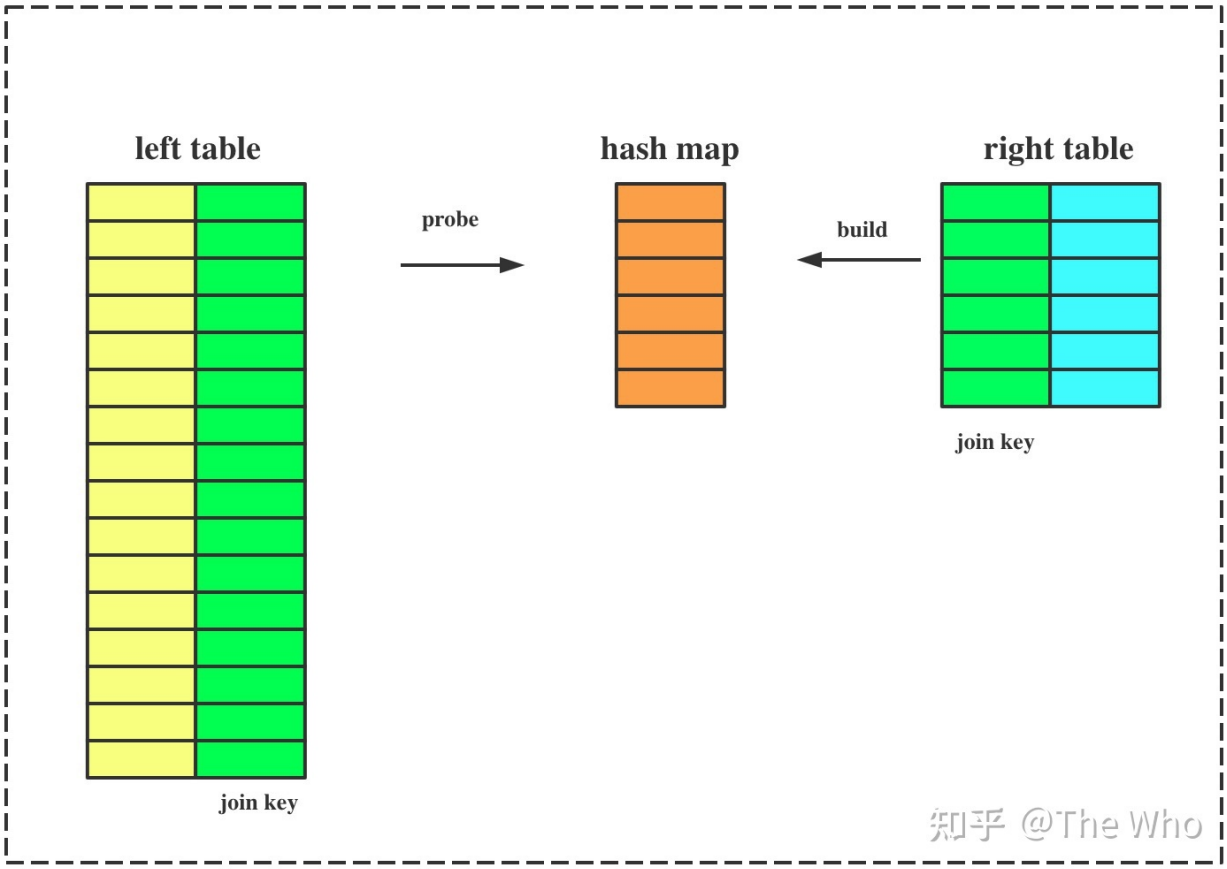
ClickHouse 单机JOIN操作默认采用HASH JOIN算法，可选MERGE JOIN算法。其中，MERGE JOIN算法数据会溢出到磁盘，性能相比前者较差。本文重点介绍基于HASH JOIN算法的实现JOIN操作。

ClickHouse JOIN查询语法如下：

```
SELECT <expr_list>
FROM <left_table>
[GLOBAL] [INNER|LEFT|RIGHT|FULL|CROSS] [OUTER|SEMI|ANTI|ANY|ASOF] JOIN
<right_table>
(ON <expr_list>)|(USING <column_list>) ...
```

ClickHouse 的 HASH JOIN算法实现比较简单：

- 从right_table 读取该表全量数据，在内存中构建HASH MAP；
- 从left_table 分批读取数据，根据JOIN KEY到HASH MAP中进行查找，如果命中，则该数据作为JOIN的输出；



从这个实现中可以看出，如果right_table的数据量超过单机可用内存空间的限制，则JOIN操作无法完成。通常，两表JOIN时，将较小表作为right_table.

2. ClickHouse分布式JOIN实现

ClickHouse 是去中心化架构，非常容易水平扩展集群。当以集群模式提供服务时候，分布式JOIN查询就无法避免。这里的分布式JOIN通常指，JOIN查询中涉及到的left_table 与 right_table 是分布式表。

通常，分布式JOIN实现机制无非如下几种：

- Broadcast JOIN
- Shuffle Join
- Colocate JOIN

ClickHouse集群并未实现完整意义上的Shuffle JOIN，实现了类Broadcast JOIN，通过事先完成数据重分布，能够实现Colocate JOIN。

ClickHouse 的分布式JOIN查询可以分为两类，带GLOBAL关键字的，和不带GLOBAL关键字的情况。

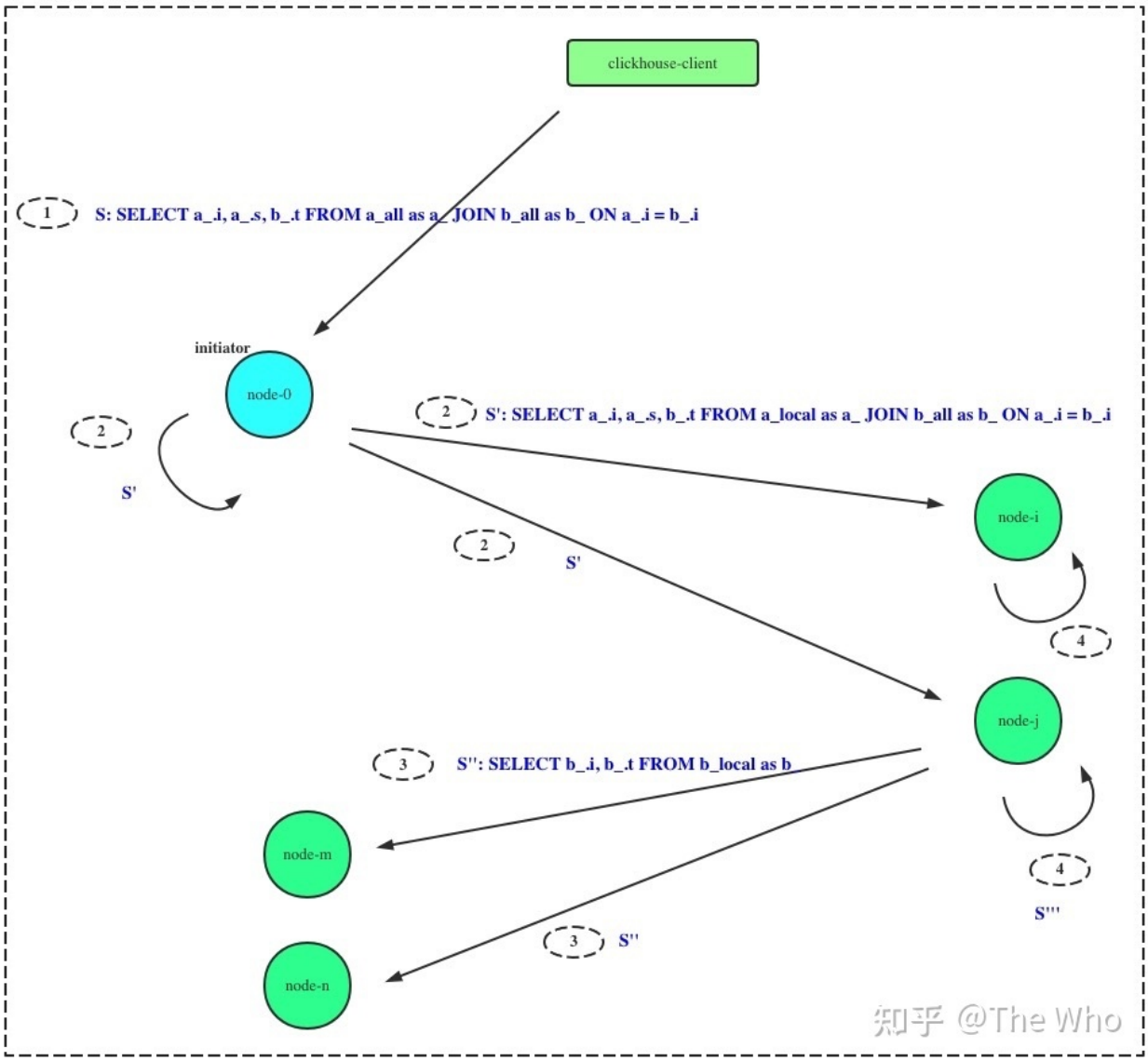
2.1 普通JOIN实现

2.1 中描述了GLOBAL JOIN的实现。接下来看看无GLOBAL关键字的JOIN如何实现的：

- a. initiator 将SQL S中左表分布式表替换为对应的本地表，形成S'
- b. initiator 将a.中的S'分发到集群每个节点
- c. 集群节点执行S'，并将结果汇总到initiator 节点

- d. initiator 节点将结果返回给客户端

如果右表为分布式表，则集群中每个节点会去执行分布式查询。这里就会存在一个非常严重的读放大现象。假设集群有N个节点，右表查询会在集群中执行N*N次。



如图所示，假设执行的SQL为：

```
SELECT a_.i, a_.s, b_.t FROM a_all as a_ JOIN b_all AS b_ ON a_.i = b_.i
```

其中，a_all, b_all为分布式表，对应的本地表名为a_local, b_local。则改SQL在分布式执行的时序为：

- 1）initiator 收到查询请求
- 2) initiator 执行分布式查询，本节点和其他节点执行`SELECT a_.i, a_.s, b_.t FROM a_local AS a_ JOIN b_all as b_ ON a_.i = b_.i`即左表分布式表更改为本地表名。该SQL在集群范围内并行执行。
- 3）集群节点收到2）中SQL后，分析出右表时分布式表，则触发一次分布式查询： `SELECT b_.i, b_.t FROM b_local AS b_` 集群各节点并发执行，并合并结果，记为subquery.

- 4）集群节点完成3）中SQL执行后，执行 `SELECT a_.i, a_.s, b_.t FROM a_local AS a_ JOIN subquery as b_ ON a_.i = b_.i`其中subquery表示2中执行的结果
- 5) 各节点执行完成JOIN计算后，向initiator节点发送数据

可以看出，ClickHouse 普通分布式JOIN查询是一个简单版的Shuffle JOIN的实现，或者说是一个不完整的实现。不完整的地方在于，并未按JOIN KEY去Shuffle数据，而是每个节点全量拉去右表数据。这里实际上是存在着优化空间的。

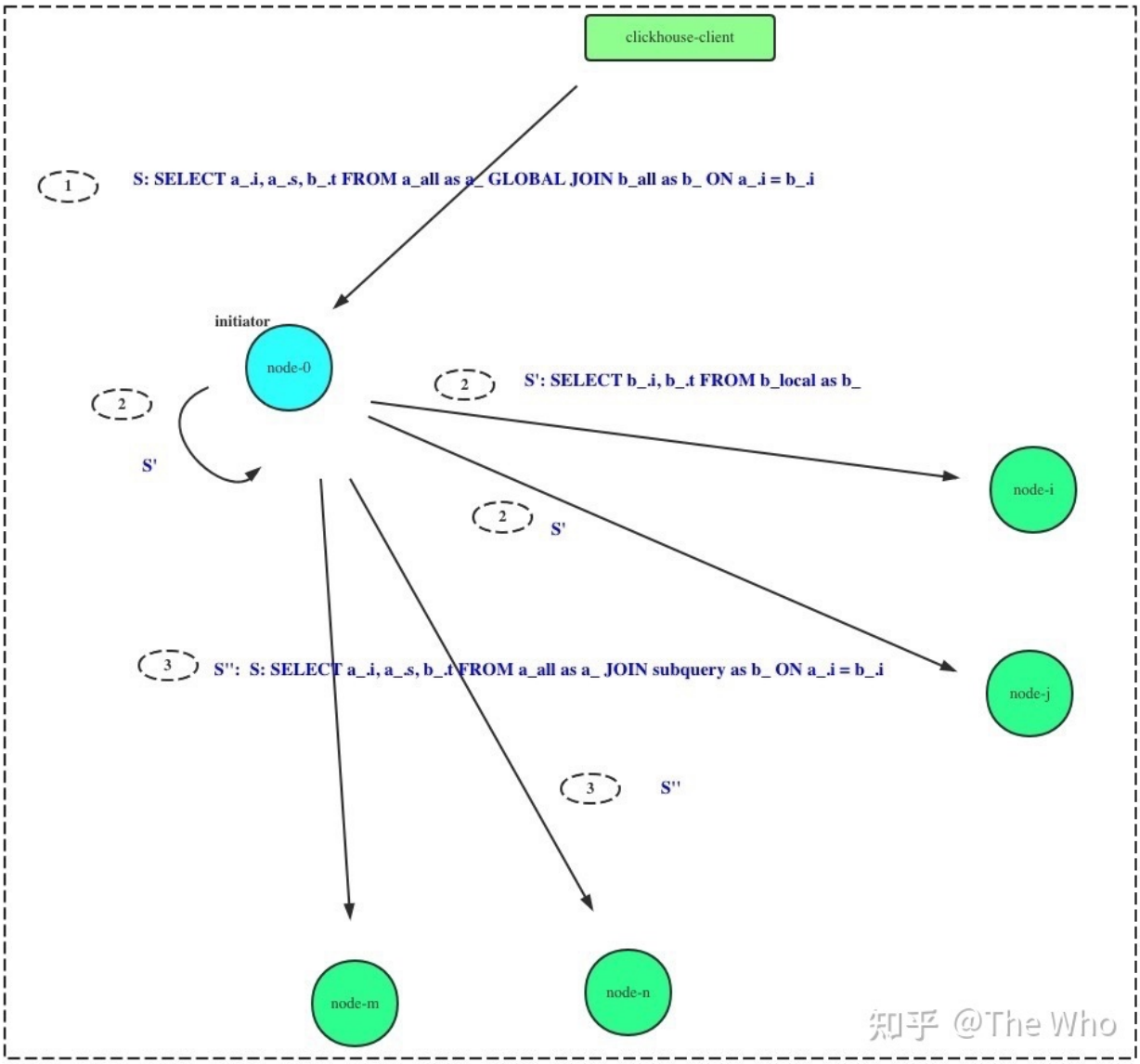
在生产环境中，查询放大对查询性能的影响是不可忽略的。

2.2 GLOBAL JOIN 实现

GLOBAL JOIN 计算过程如下：

- a. 若右表为子查询，则initiator完成子查询计算；
- b. initiator 将右表数据发送给集群其他节点；
- c. 集群节点将左表本地表与右表数据进行JOIN计算；
- d. 集群其他节点将结果发回给initiator节点；
- e. initiator 将结果汇总，发给客户端；

GLOBAL JOIN 可以看做一个不完整的Broadcast JOIN实现。如果JOIN的右表数据量较大，就会占用大量网络带宽，导致查询性能降低。



如图所示，假设执行的SQL为：

```
SELECT a_.i, a_.s, b_.t FROM a_all as a_ GLOBAL JOIN b_all AS b_ ON a_.i = b_.i
```

其中，a_all, b_all为分布式表，对应的本地表名为a_local, b_local。则改SQL在分布式执行的时序为：

- 1）initiator 收到查询请求
- 2) initiator 和集群其他节点均执行`SELECT b_.i, b_.t FROM b_local AS b_` 即左表分布式表更改为本地表名。该SQL在集群范围内并行执行。汇总结果，记录为subquery。
- 3）initiator 将2)中subquery发送到集群中其他节点，并触发分布式查询： `SELECT a_.i, a_.s, b_.t FROM a_local AS a_ JOIN subquery as b_ ON a_.i = b_.i`其中subquery表示2)中执行的结果
- 4) 各节点执行完成JOIN计算后，向initiator节点发送数据

可以看出，GLOBAL JOIN 将右表的查询在initiator节点上完成后，通过网络发送到其他节点，避免其他节点重复计算，从而避免查询放大。

3. 分布式JOIN最佳实践

在清楚了ClickHouse 分布式JOIN查询实现后，我们总结一些实际经验。

- **一、尽量减少JOIN右表数据量**

ClickHouse根据JOIN的右表数据，构建HASH MAP，并将SQL中所需的列全部读入内存中。如果右表数据量过大，节点内存无法容纳后，无法完成计算。

在实际中，我们通常将较小的表作为右表，并尽可能增加过滤条件，降低进入JOIN计算的数据量。

- **二、利用GLOBAL JOIN 避免查询放大带来性能损失**

如果右表或者子查询的数据量可控，可以使用GLOBAL JOIN来避免读放大。需要注意的是，GLOBAL JOIN 会触发数据在节点之间传播，占用部分网络流量。如果数据量较大，同样会带来性能损失。

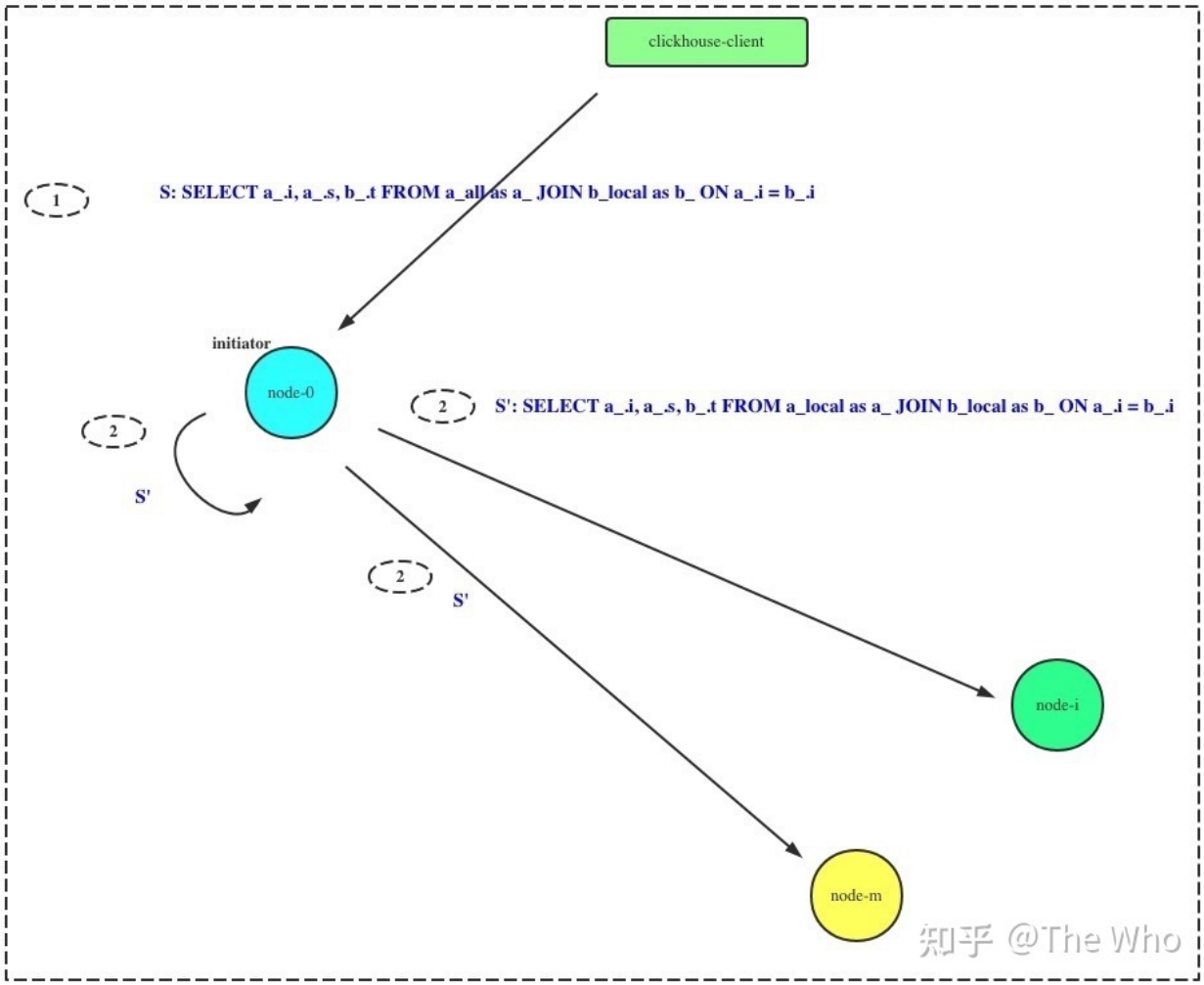
- **三、数据预分布实现Colocate JOIN**

当JOIN涉及的表数据量都非常大时，读放大，或网络广播都带来巨大性能损失时，我们就需要采取另外一种方式来完成JOIN计算了。

根据“相同JOIN KEY必定相同分片”原理，我们将涉及JOIN计算的表，按JOIN KEY在集群维度作分片。将分布式JOIN转为为节点的本地JOIN，极大减少了查询放大问题。

如果如下操作：

- 将涉及JOIN的表按JOIN KEY分片
- 根据2.2节描述，将JOIN预计中右表换成相应的本地表



如图所示，执行的SQL为：

```
SELECT a_.i, a_.s, b_.t FROM a_all as a_ JOIN b_local AS b_ ON a_.i = b_.i
```

其中，a_all, b_all为分布式表，对应的本地表名为a_local, b_local。则改SQL在分布式执行的时序为：

- 1）initiator 收到查询请求
- 2）initiator 发起一次分布式查询，本机以及其他节点执行： `SELECT a_.i, a_.s, b_.t FROM a_local AS a_ JOIN b_local as b_ ON a_.i = b_.i`
- 3) 各节点执行完成JOIN计算后，向initiator节点发送数据

由于数据以及预分区了，相同的JOIN KEY对应的数据一定在一起，不会跨节点存在，所以无需对右表做分布式查询，也能获得正确结果。

4. 总结

本文介绍了ClickHouse JOIN实现原理，并根据原理介绍了相关的最佳实践：

- 减少JOIN右表数据量
- 避免查询放大带来性能损失

- 数据预分布实现Colocate JOIN;