BloomFilter Join 功能开发文档

修订历史

版本	修订日期	修订描述	作者	备注
Cedar 0.2	2016-07-01	BloomFilter Join 功能开发文档	茅潇潇	无

1 总体设计

1.1 综述

> Cedar是由华东师范大学数据科学与工程研究院基于OceanBase 0.4.2 研发的可扩展的关系数据库，实现了巨大数据量上的跨行跨表事务。业务中存在大量的多表连接查询，在处理多表连接时，Cedar 0.1中进行Merge Join运算时，由Merge Server（MS）分发请求给相应的 Chunk Server（CS），CS合并UPS上的增量数据后把所有数据返回给MS，MS进行排序后做 Merge 操作。若对大表进行查询，CS将传输大量不会产生连接关系的无效数据到MS，浪费了大量的网络传输时间和无效数据的排序时间等，导致查询缓慢。
>
> 针对这一问题，Cedar 0.2版本实现了一种基于布隆过滤器的连接优化算法（Bloomfilter Join）。该算法并不将右表数据全部数据发送到计算节点，而是使用布隆过滤器对右表数据进行过滤，再对过滤后的数据进行排序归并连接，这样网络通讯代价将会大大减少，在过滤后的数据集上进行排序操作所消耗的内存资源也会下降。

1.2 名词解释

- **CS**：ChunkServer，Cedar系统中的基线数据服务器，提供分布式数据存储服务，负责存储基线数据。
- **MS**：MergeServer，Cedar系统中的查询处理服务器，负责接收和解析SQL请求、生成和执行查询计划以及将所有节点的查询结果合并并返回给客户端。
- **Bloom Filter**：布隆过滤器，由k个相互独立的哈希函数和一个m位长的位向量组成，可以用于检索一个元素是否在一个集合中。

1.3 功能

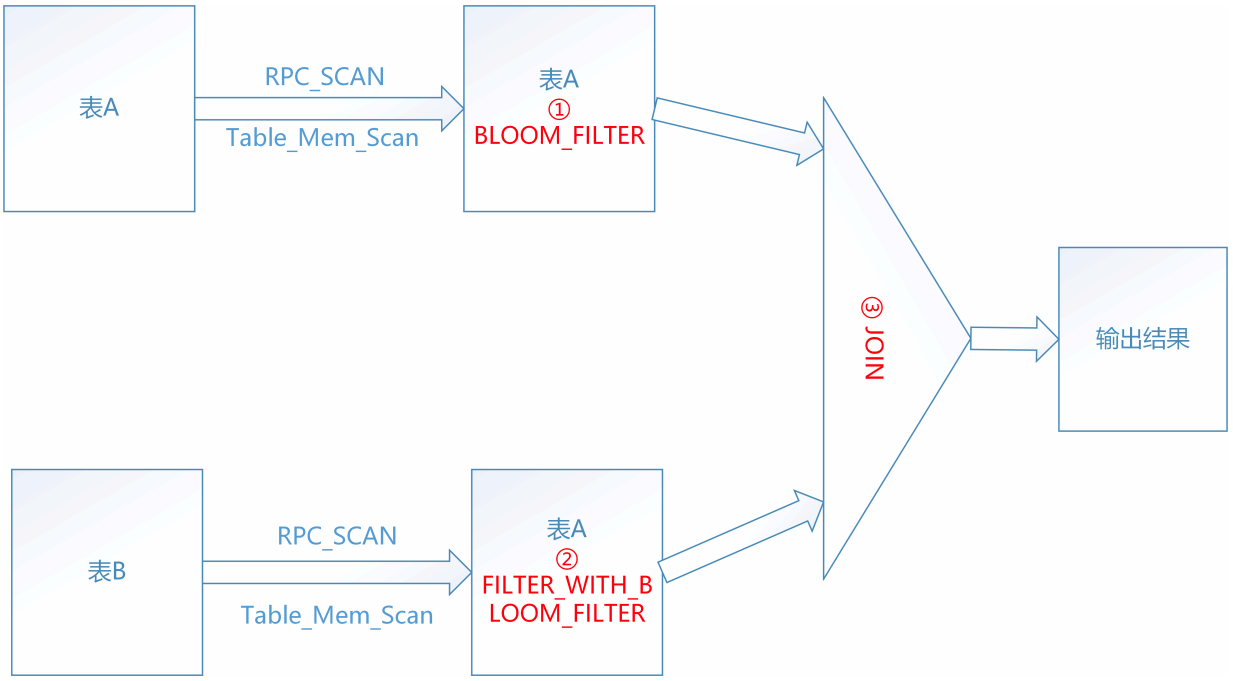
BloomFilter Join将关系S在公共属性B上的投影∏B(S)表示在一个布隆过滤器中。假设，节点1上的关系R和节点2上的关系S，在属性R.A=S.B上做连接操作，它的工作流程如下：首先，节点2将关系S的每个元组在属性B上的值插入到一个布隆过滤器BFs里；然后，将BFs发送到节点1，节点1根据BFs过滤掉关系R中不符合连接条件的记录，把R中符合连接条件的记录传送给节点2；最后，在节点2上过滤掉布隆过滤器误判的记录，并进行连接操作。

1.4 性能指标

当左表在连接列上不重复值的个数不超过100万，并且右表的数据行数减去左表选择出的右表行数，即过滤掉的右表数据越大时，bloomfilter join的优化效果越好。当过滤掉的右表数据大于500万行时，Bloomfilter Join比原Merge Join极大地减少了Cedar对连接操作的处理时间。

2 模块设计

2.1 基本原理



如上图所示，BloomFilter Join处理可以分三个步骤：

1. 生成Bloom Filter：将参与join的左表生成Bloom Filter。先将左表所有符合查询条件的数据获取到Merge Server，并生成Bloom Filter。
2. Bloom Filter过滤：将参与JOIN的左表生成的Bloom Filter作为SQL Expression的一个参数，序列化后传入右表的ChunkServer，过滤右表数据。无等值连接条件时，Bloom Filter为空，右表进行全表扫描。

3. Join处理： 按照不同规则进行等值Join操作。

2.2 解析词法语法子模块设计

在Hint中对指定的Join算法进行解析，并指定两两表之间的Join规则，把用户输入的 SQL 语句解析成语法树。

例如：Select /*+ join(Bloomfilter_join,merge_join,Bloomfilter_join,merge_join) */ * from tab1 left join tab2 on tab1.col1=tab2.col2 right join tab3 on tab2.col1=tab3.col2 full outer join tab4 on tab1.col1=tab4.col2 inner join tab5 on tab3.col1=tab5.col2;
/*+ join(Bloomfilter_join,merge_join,Bloomfilter_join,merge_join) */括号内需要符合“join_type(,join_type)”规则，按照join顺序，表明两两表之间的Join使用的算法规则。

- 若hint中的join类型个数多于后面数据表的两两连接数，则忽略hint中多余的join类型；若hint中的join类型个数少于后面数据表的两两连接数，则后面未指定的连接都执行merge join。
- 对于select * from tab1,tab2 where tab1.col1=tab2.col2; 此类Join查询需要DBA改为Inner Join后再指定Hint，完成Bloomfilter Join；否则将执行原Merge Join操作。

> **修改内容：**
>
> 1. 分别向词法定义文件sql_parser.l和语法定义文件sql_parser.y中添加Bloomfilter Join和Merge Join的词法规则和语法规则，使解析器能够识别Bloomfilter Join和Merge Join的语法。

2.3 生成逻辑计划子模块设计

根据语法树生成中缀表达式和Hint信息，判断hint中指定的join信息是否符合规则，存到逻辑计划中。

- 在生成逻辑计划时，判断Hint信息是否与SQL中Join运算一一对应，若不对应则按照原Merge Join查询。

> **修改内容：**
>
> 1. 对语法树生成中缀表达式和Hint信息，增加hint是否有指定join信息的分支case T_JOIN_TYPE_LIST。
> 2. 增加方法oceanbase::sql::generate_join_hint()对Hint中的Join信息进行判断是否符合规则，并将Hint信息中每一个join类型有序地存到一个动态数组中，供生成物理计划时使用。

2.4 生成物理计划子模块设计

根据逻辑计划里面存的相应信息，然后按照指定的Join算法生成相应的物理操作符，包括Merge Join运算符和BloomFilter Join运算符，并按照Join先后顺序确定操作符之间的父子关系。从逻辑计划中取出中缀表达式，转成后缀表达式存到相应的物理操作符里面。

- 添加add_Bloomfilter_join_expr表达式

```
int ObTransformer :: add_Bloomfilter_join_expr (
    ObLogicalPlan *logical_plan,
    ObPhysicalPlan *physical_plan,
    ObBloomfilterJoin& join_op,
    ObSort& l_sort,
    ObSort& r_sort,
    ObSqlRawExpr& expr,
    const bool is_table_expr_same_order)
|--->// 添加join表达式 join_expr.fill_sql_expression(join_op_cnd, thi
s, logical_plan, physical_plan);
|--->// 添加等值连接条件 join_op.add_equijoin_condition(join_op_cnd);
```

- 按照SQL中的Join顺序生成物理查询计划

```
int ObTransformer::gen_phy_joins(
    ObLogicalPlan *logical_plan,
    ObPhysicalPlan *physical_plan,
    ErrStat& err_stat,
    ObSelectStmt *select_stmt,
    ObJoin::JoinType join_type,
    oceanbase::common::ObList<ObPhyOperator*>& phy_table_list,
    oceanbase::common::ObList<ObBitSet<> >& bitset_list,
    oceanbase::common::ObList<ObSqlRawExpr*>& remainder_cnd_list,
    oceanbase::common::ObList<ObSqlRawExpr*>& none_columnlize_alia
s)
|--->//原Join逻辑不变
|--->//while (ret == OB_SUCCESS && phy_table_list.size() > 1)
|------>判断phy_table_list.size()是否大于0
|------>并对Hint中Join信息join_array_.size()和指定inner(left) join判断
        如果没有Hint信息或不是inner(left) join则执行原来的Merge Join
|------>根据不同Join类型建立物理操作符
        CREATE_PHY_OPERRATOR(bloomfilter_join_op, ObBloomFilterJoi
n, physical_plan,err_stat);
        CREATE_PHY_OPERRATOR(join_op, ObMergeJoin, physical_plan, e
rr_stat);
|------>提取ObQueryHint中的信息，判断Join使用的算法
|--->//最后释放join_array_中的join列
```

修改内容：

1. 添加表达式sql::ObTransformer ::add_Bloomfilter_join_expr()。
2. 修改SQL中的Join物理查询计划sql::ObTransformer ::gen_phy_joins()，增加判断Join类型的语句，识别Bloomfilter Join类型，建立Bloomfilter Join的物理操作符。

2.5 Bloomfilter Join物理操作符子模块设计

Bloomfilter Join主要实现三个函数：Open()、Close()、Get_next_row()。

- 添加ObBloomfilterJoin:open()方法

```
int ObBloomfilterJoin:open(){
|--->// Rpc Scan->Open()或Table Mem Scan->Open()拉取前表的所有信息
|--->//left_op_->open() ;
|--->//while(left_op_->get_next_row(row))
|--------->bloom_filter->insert(row);
|--------->迭代前表每一行数据，建立Bloomfilter
|--------->left_cache_.add_row(*row, stored_row);
|--------->维护前表数据
|--->//right_op_->open() ;打开后表
|--->// Rpc Scan->Open()或Table Mem Scan->Open()，构造参数和信息
}
```

- 添加ObBloomfilterJoin: get_next_row ()方法

```
根据join_type选择具体的join算法（left、right、full out、inner join）
int ObBloomfilterJoin::get_next_row(const ObRow *&row){
|--->//const ObRow *temp_row = NULL;
|--->//while(OB_SUCCESS == ret)
|--------> //每次迭代后表的get_next_row()取一行数据
|--------->ret = right_op->get_next_row(temp_row);
|--------->compare_equijoin_cond();计算当前行是否能够与前表连接，判断等值
连接
|--------->curr_row_is_qualified ();判断非等值连接
|--------->output
|--------->row = &curr_row_;
}
```

- 添加ObBloomfilterJoin: close ()方法

```
int ObBloomfilterJoin:close(){
|--->//释放空间，重置运算符中的数据
|--->//ob_free(store_buf);
|--->//row_desc_.reset();
|--->//ret = ObJoin::close();
}
```

3 模块接口

对两个或多个数据表使用hint, 指定join类型(bloomfilter join或merge join)的连接。语句如下：
select /*+ join(bloomfilter_join,merge_join,…)*/ <查询内容> from <表名> left (inner, right, full outer) join <表名> on <join条件>；
/*+ join (bloomfilter_join, merge_join, bloomfilter_join, merge_join) */括号内需要符合“join _ type (, join_ type)”规则，按照join顺序，表明两两表之间的join使用的算法规则。

若hint中的join类型个数多于后面数据表的两两连接个数，则忽略hint中多余的join类型；若hint中的join类型个数少于后面数据表的两两连接个数，则后面未指定的连接都默认执行merge join。

4 使用限制条件和注意事项

- bloomfilter join不支持在不同类型的连接列上做等值连接。
- bloomfilter join不支持right、full out join算法，若指定了bloomfilter join类型而连接类型为right、full outer join，还是默认执行merge join。
- 对于select * from tab1,tab2 where tab1.col1=tab2.col2; 此类Join查询需要DBA改为inner join后再指定Hint，完成bloomfilter join；否则将执行原merge join操作。
- 只有当左表在连接列上不重复值的个数不超过100万，并且右表的数据行数减去左表选择出的右表行数，即过滤掉的右表数据越大，bloomfilter join的优化效果越好。建议过滤掉的右表数据大于500万行时使用bloomfilter join。
- 多表连接时，根据连接顺序对每两张表使用以上规则，以达到优化效果。