文章编号: 1000-5641(2016)05-0056-11

面向分布式数据库的相关子查询优化策略

毛思语, 张利军, 张小芳, 高锦涛, 李战怀

(西北工业大学 计算机学院, 西安 710129)

摘要: 子查询是指查询语句作为另一个语句的查询条件出现, 相关子查询是指子查询的查询条件依赖于父查询. 相关子查询要对子查询反复求值, 需要多次访问磁盘, 尤其是在分布式的环境中还会产生大量的通信开销, 导致执行效率低下. 在对现有相关子查询优化策略分析研究的基础上, 综合分布式的特点, 将子查询展开、无用子树切除、聚集函数消除等策略应用于分布式关系数据库系统中, 并在开源分布式关系数据库 OceanBase 中应用这些策略实现对谓词EXISTS的相关子查询的优化. 实验表明这些策略能够明显改善相关子查询的查询性能.

关键词: 分布式数据库; 相关子查询; 子查询优化

中图分类号: TP392 **文献标识码**: A **DOI:** 10.3969/j.issn.1000-5641.2016.05.007

Optimization strategies of correlated subquery for distributed database

MAO Si-yu, ZHANG Li-jun, ZHANG Xiao-fang, GAO Jin-tao, LI Zhan-huai

(*School of Computer, Northwestern Polytechnical University, Xi'an* 710129, *China*)

Abstract: A query which occurs in another query as a filter is called subquery, and if the filtering condition of a subquery depends on its parent query, it is called correlated subquery. Generally, the execution cost of query with correlated subquery is high due to that subquery would be executed multiply, which leads to multiple disk access and extra communications in distributed system. Based on the investigation of the classical optimization strategies of correlated subquery, and according to the characteristics of distributed system, we adopt pulling up subquery, removing useless tree and eliminating aggregation function to optimize correlated subquery in distributed database system. And we implement these strategies in the distributed relational database OceanBase for the correlated subquery predicate EXIST. Experiment results show that these strategies can significantly improve the performance of a correlated subquery.

Key words: distributed database; correlated subquery; subquery optimization

收稿日期: 2016-05

基金项目: 中央高校基本科研业务费专项资金资助(3102015JSJ0004)

第一作者: 毛思语, 女, 硕士研究生, 研究方向为数据库. E-mail: maosiyu@mail.nwpu.edu.cn.

通信作者: 张利军, 男, 讲师, 研究方向为数据库理论, 数据管理技术.
E-mail: zhanglijun@nwpu.edu.cn.

0 引　　言

SQL 语言是关系数据库中一种高度非过程化语言[1], 在SQL语言中, 一个 SELECT-FROM-WHERE 称为一个查询块, 将一个查询块嵌套在另一个查询块条件中的查询称为子查询, 子查询根据是否和父查询有依赖关系分为相关子查询和非相关子查询[2]. 非相关子查询实现较为简单, 即先执行子查询, 父查询绑定子查询的结果执行, 子查询和父查询都只执行一遍; 相关子查询执行较为复杂, 先执行父查询, 从父查询的结果集中取出一个元组, 重写子查询后执行, 简单来说, 父查询中有多少个元组, 将执行多少次子查询. 这种相关子查询执行策略非常耗时, 当子查询嵌套层数较多时, 执行次数将是指数级的增长, 因此对相关子查询的执行进行优化非常重要.

相关子查询的优化方向是减少子查询的执行次数或者是减少子查询的执行时间. 现有对相关子查询的优化策略一般是去相关性, 然后各个部分独立执行. 本文在分析研究相关子查询的优化策略的基础上, 结合分布式的特点, 并将相关子查询的优化策略应用于分布式数据库中, 并以开源分布式关系型数据库 OCEANBASE 为基础, 实现了谓词为 EXISTS 的相关子查询, 并采用子查询展开、无用分支切除和聚集函数消除等优化策略对相关子查询进行优化.

1 相关研究

一般来说, 子查询可以作为过滤条件出现在 WHERE/HAVING/ON 等地方, 或者作为临时表出现在 FROM/SELECT 等子句中[3]. 本文只讨论作为过滤条件使用的子查询, 子查询可以按照关键字和语义划分为以下 3 类[2].

(1) IN 关键字的集合成员查询.

(2) ANY/SOME/ALL 等集合比较查询.

(3) 以 EXISTS 为标志的空判断查询. 从语义上分析, EXISTS 不返回任何数据, 只做逻辑空判断, 并且根据证明, 所有其他子查询都可以转化为 EXISTS 子查询[2,4].

由于以 EXISTS 为标志的空判断查询从语义上是一种只取一次的查询, 不同于其他子查询要扫描全表或者是部分表, 从查询语义上说, EXISTS 子查询是一种最优子查询, 并且其他谓词的子查询都可以转为 EXISTS 子查询. 在本文中只讨论 EXISTS 相关子查询, 下文中提到的相关子查询也指谓词为EXISTS的相关子查询.

相关子查询由于执行多次子查询使效率较低, 很多去相关性的方法提出以提高相关子查询性能[5], 常用以下3种方法优化子查询性能.

(1) 子查询展开为半连接, 把子查询展开为半连接, 消除子查询.

(2) 子查询展开为内连接, 把子查询展开为内连接, 消除子查询.

(3) 子查询中相关列上建索引, 减少单次子查询执行时间, 并不降低子查询执行次数.

但是在相关子查询不能展开的情况, 使用原始方法执行子查询还是非常耗时, JUN RAO 等人基于这种情况提出一种将子查询中和父查询相关条件去除后物化中间结果的方法, 填充子查询展开和传统方法中的空缺, 性能优于传统的 NEST-LOOP 方式[6].

1.1 子查询展开为半连接

根据相关子查询语义, 一个相关子查询类似于一个半连接. Bellamokan 等人提出使用窗口函数和半连接的方式消除子查询, 目前 Oracle、Postgre 和 MYSQL 等查询优化器中采用

这种策略对相关子查询进行优化[7-8]. 在执行子查询展开技术是把将子查询转换为半连接, 嵌套层数减 1.

Postgre 将位于 WHERE 子句后的子查询称为子链接, 其在生成执行计划之前有一个优化层处理, 在这一部分处理对相关子查询的优化[8]. 从上层依次调用函数展开子链接, 同时把子链接转换为半连接或者是反半连接完成优化工作[9].

值得注意的是, 并不是所有的子查询都可以进行展开操作, 在 POSTGRE 中如果子查询中含有 WITH 子句、聚集函数、FROM 或者 WHERE 子句为空的情况下并不支持展开操作的处理. 能进行的展开操作即使把子链接FROM子句的表合并到父查询FROM子句中. 把子链接中的 WHERE 子句合并到父查询的 WHERE 子句中. 如例 1 所示, 可将子查询展开为内连接, 其中 T1、T2...表示表, C1 表示表中的列.

例1: SELECT * FROM T1 WHERE T1.C1>1 AND EXISTS (SELECT * FROM T2 WHERE T2.C1=T1.C1);

优化后:

SELECT * FROM T1 SEMMI JOIN T2 WHERE T1.C1>1 AND T2.C1 =T1.C1;

1.2 子查询展开为内连接

除了可以将相关子查询转为半连接, 由 WON KIM 等人提出使用内连接的方式拉平子查询, 减少子查询嵌套层数[1]. 根据: $R \propto_{a=b} S = \prod_R (R \infty_{a=b} S) = R \infty_{a=b} (\prod_b (S))$, 其中 R 和 S 分别代表关系, a 和 b 分别代表关系的连接属性[10]. 把子查询转换内连接后要在输出列上加上去重操作.

Won Kim 将子查询分成五类, 其中相关子查询根据是否含有聚集函数分为两类[1]. 不含聚集函数的相关子查询直接可以展开为内连接, 含有聚集函数的子查询需要借用临时表机制消除聚集函数, 转化为不含聚集函数的相关子查询后展开处理.

其消除子查询的算法是将一个N层子查询拉平为N-1层的算法: 合并所有的 FROM 子句, WHERE 子句的过滤条件用 AND 链接, 如果子查询的谓词是 IS IN 则转换成“=”, 如果是其他的则谓词不变, 对于父查询的 SELECT 子句不做任何变化[1]. 如例 2 所示, 可将子查询展开为内连接.

例 2: SELECT C3 FROM T1 WHERE T1.C1 IS IN (SELECT C1 FROM T2 WHERE T2.C2=T1.C3);

优化后:

SELECT DISTINCT C3 FROM T1 INNER JOIN T2 WHERE T1.C1=T2.C1 AND T2.C1=T1.C3;

1.3 子查询中相关列上建索引

相关子查询除了可以减少内查询执行的次数, 还可以优化内查询的执行, 总体提升子查询性能. Takamitsu Shioi 等人从数据存储角度出发, 提出针对使用行存储的数据库系统, 可通过在 WHERE 中的属性列上建立索引的方式对子查询进行优化[11]. MYSQL 中对于 EXISTS 相关子查询没有进行优化, 而是采取索引的方式减少子查询执行时间.

在父查询表很小的情况下, 子查询表很大的情况下, 借用索引进行执行, 缩短每次执行子查询的时间, 总体开销较小. MYSQL 本质上没有对 EXISTS 做优化, 但是通过借助索引减少子查询的执行时间, 提升子查询执行性能.

2 面向分布式系统的相关子查询优化

2.1　面向分布式系统的相关子查询优化策略

数据库查询的优化主要分为两个方面: 路径优化和扫描次数的优化. 查询耗时主要是传送磁盘块次数和搜索磁盘块次数[4]. 对于路径方面的优化可以减少传送磁盘块的次数, 对扫描次数的优化可以减少搜索磁盘块的次数. 对于分布式数据库而言, 通信是影响查询响应的主要因素, 在处理查询语句时要尽量避免过多的通信开销, 可以通过减少通信时间实现子查询的优化[12].

由子查询执行策略可以看出, 相关子查询的复杂度主要是因为子查询和父查询有依赖关系, 对子查询的扫描次数过多. 最坏的情况是父查询有多少个元组, 则要进行多少次子查询表扫描. 所以, 针对相关子查询优化的方向主要是减少子查询表的扫描次数和借助索引减少子查询表的扫描时间.

2.1.1　子查询展开

把子查询置于外层的父查询中, 作为连接关系与外层父查询并列, 其实质是把某些子查询重写为等价的多表连接操作[5], 这样有关的访问路径、连接方法和连接顺序可能被有效利用, 使得查询语句的层次尽可能地减少. 因为 EXISTS 子查询语义是半连接语义, 所以可以根据公式把子查询展开为内连接执行. 如上述例 2 所示的转化. 将子查询的 FROM 子句以内连接的形式合并到父查询中, 将子查询的 WHERE 子句以 AND 连接的形式合并到父查询中, 对父查询的输出列做去重操作.

但是这种优化并不是全部适应, 在子查询中有 WITH 子句、FROM 或者 WHERE 子句为空的情况下并不能进行子查询展开[7].

2.1.2　无用子树切除

根据 EXISTS 语义来说, 只做空判断, 并不关心结果集具体是什么内容, 所以可以去除子查询的 SELECT 子句去及 SELECT 子句上的所有操作. 在可以去除子查询的 SELECT 子句、对 SELECT 子句的排序去重或限制输出等. 减少了物理计划执行时路径长度和无用操作. 如例 4 所示, 是无用分支切除后的语句.

例4: SELECT * FROM T1 WHERE EXISTS (SELECT DISTINCT C2 FROM T2 WHERE T1.C1=T2.C1 ORDER BY (C3));

优化后:

SELECT * FROM T1 WHERE EXISTS (SELECT C2 FROM T2 WHERE T1.C1=T2.C1);

2.1.3　聚集函数消除

结合聚集函数的语义, 不管一张表里有没有数据, 聚集函数都会输出一行, 例如 SUM(X) 会返回 NULL, COUNT(*)会返回 0[13], 子查询的最终结果集不可能为空, 对于 EXISTS 过滤条件, 其结果是永真的. 因此输出列中存在聚集函数的子查询是无意义的, 可以直接去除. 如例 5 所示, 可以直接去除子查询.

例5: SELECT * FROM T1 WHERE EXISTS (SELECT MAX (C1) FROM T2 WHERE T2.C1=T1.C1);

优化后:

SELECT * FROM T1;

2.1.4　借用索引

在不能进行子查询展开的情况下, 可以选择建合适的索引, 减少子查询的执行时间. 并

且由于 EXISTS 相关子查询是空判断, 并不关心具体得到了多少个元组, 应该对子查询加 LIMIT 1 限制. 使子查询在执行时只得到一行可以满足条件的元组后就会输出, 而不是一直阻塞直到得到所有元组. 如例 6 所示, T2 表的 C1 列是主键列, 则填充后可以改变扫描方式.

例6: SELECT * FROM T1 WHERE EXISTS (SELECT * FROM T2 WHERE T2.C1 =T1.C2);

优化后:

SELECT * FROM T1 WHERE EXISTS (SELECT * FROM T2 WHERE T2.C1=T1.C2 LIMIT 1);

2.2 分布式数据库 OceanBase 中相关子查询优化实现

2.2.1 分布式数据库系统 OceanBase

OceanBase 是由阿里巴巴开发的分布式关系型数据库. OceanBase 有 4 种类型的服务器: 主控服务器、基准服务器、增量服务器和合并服务器. OceanBase 将数据分为基准数据和增量数据分别存放在基准服务器和增量服务器中, 增量服务器中的数据可以通过合并的方式生成新的基准数据. 客户端在进行查询时, 需要通过合并服务器把需求发送到两个服务器中, 根据主控服务器中的信息取到相关数据, 然后进行过滤合并后返回给客户端.

OceanBase 对某些查询可以不用将数据取到合并数据库上进行过滤, 将过滤条件下压到基准数据库上进行, 减少传输的元组数和空间占用, 在基准数据库上的过滤是一次扫描所有所需元组, 然后输出第一行满足条件的元组. OceanBase 元组的扫描方式有两种: 一种基于主键的 GET 方法, 利用主键索引直接取到所需元组; 另一种是逐行 SCAN, 扫描表中所有元组是否满足过滤条件.

目前开源的 OceanBase 版本中并不支持子查询, 实验只实现了应用较多且性能较优化的 EXISTS 相关子查询.

2.2.2 EXISTS 的执行策略

对于相关子查询的执行策略, 需要先执行父查询, 把和子查询相关的列传给子查询, 经过子查询过滤, 决定是否输出这一个元组. 图 1 为 OceanBase 中的 EXISTS 相关子查询执行流程.

如图 1 所示, 在物理计划执行开始时, 从父查询取出一个元组, 填充子查询, 如果子查询中有合适的元组, 即子查询的结果集不为空, 则输出父查询的元组; 否则, 不输出.

父查询过滤规则: 本阶段, 在父查询物理操作符打开完成后, 需要过滤除子查询外其他元组, 即假设子查询结果恒为真的情况下过滤表中元组, 去除不满足其他条件的元组. 这里主要有两种情况: 一种是<子查询>OR <其他过滤条件 P>, 这种情况不执行 P, 直接把元组输出; 另一种是<子查询>AND<其他过滤条件 P>, 如果不满足P过滤条件, 则不用取出这行元组, 被过滤掉. 为了识别子查询, 需要在物理计划构建的时给每个过滤条件打上标记. 在执行时直接假设这个条件为真.

子查询执行: 执行子查询, 采用管道方式进行, 从父查询取出一个元组, 需要在子查询执行前填充子查询中和父查询相关的列, 填充完成后, 打开子查询物理操作符, 执行子查询物理计划. 因为 EXISTS 条件的特殊性, 只需执行一次子查询就可以知道是否满足 EXISTS 条件. EXISTS 是空判断不是 NULL 判断, 即使执行子查询后输出结果是 NULL 仍然满足 EXISTS 条件. 这里需要对于父查询中出现<子查询>OR <其他过滤条件P>的过滤条件重

新进行判断, 对不满足子查询的元组进行判断是否可以满足 P.

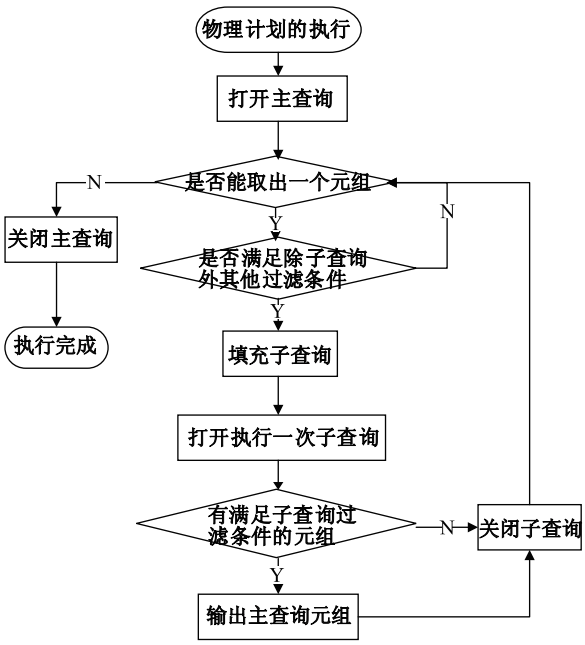

图 1　EXISTS相关子查询执行流程

Fig. 1　Execution process of correlated subquery for predicate EXISTS

2.2.3　EXISTS 的优化策略

使用 2.1 节所述的优化策略对 EXISTS 进行优化, 如算法 1 所示, 其中参数Q表示待优化查询.

算法1:

```
Optimize_subquery(Q)
{
    for each subquery_i in Q
    {
        if(subquery_i中存在无用子树)
        {
            切除无用子树;
        }
        else if(subquery_i中存在聚集函数)
        {
            从Q中移除subquery_i;
        }
        else if(subquery_i可以展开)
        {
            if(subquery_i中还有子查询)
            {
```

```
                Optimize_subquery(subquery_i);//递归调用展开子查询
            }
            else
            {
                将 subquery_i的from 子句合并到父查询中;
                将 subquery_i的where 子句合并到父查询中;
                给父查询的输出列加上 distinct;
                从Q中移除 subquery_i;
            }
        }
        else
            {
                使用主键索引进行查询;
            }
        } // end for
    }
```

Optimize_Subquery( )函数实现对查询 Q 的优化, 对 Q 中的每个子查询, 首先判断其有无无用分支, 若有则直接切除无用分支, 然后判断子查询中是否有聚集函数, 如果存在则直接去除子查询, 否则判断子查询是否可展开, 若可展开同时子查询中不再包含子查询, 则对子查询进行展开处理, 否则递归地调用 Optimize_Subquery( ) 函数对该子查询进行同样的处理. 最后若有索引, 则使用索引对查询进行处理.

3 实　　验

3.1 实验设计

实验在服务器上部署 OceanBase 分布式数据库系统, 其中服务器软硬件配置如下: 16G 内存, 4 核 CPU, 1T 硬盘. Red Hat 6.2 操作系统.

基于前述对 OceanBase 分析, 为了减少通信的时间和空间开销, 过滤条件最好可以下压到基准服务器中, 过滤条件存在主键索引时速度比较快. 根据优化策略可以得知: 展开减少通信开销, 主键索引减少子查询执行时间. 由于无用子树切除和聚集函数消除都是在特殊场景下的应用, 虽然会对性能产生一定影响, 但是不算在优化的范围内, 所以本次实验并不包括这些因素对实验的影响. 设计实验如下.

(1) 展开有效性验证, 当父查询表很大, 子查询表很小, 执行会有很多次的通信时间开销, 使用子查询展开技术应该优化性能更好一点.

(2) 主键索引有效性验证, 当父查询表很小, 子查询表很大, 执行时间主要是子查询表的查询时间, 通过主键索引可以更好的优化性能.

(3) 展开稳定性验证, 当父查询和子查询表都很大的情况下, 分布式的通信开销和子查询的执行时间开销都很大的情况下, 主键索引不会明显的优化性能, 但是使用子查询展开应该可以有效的提升性能, 并且不会有很大的波动.

实验所用的表结构如下:

STUDENT (SNO, SNAME, SSEX, SAGE, SDEPT);

COURSE (CNO, CNAME, CPNO, CCREDIT);

SC (SNO, CNO, GRADE).

其中加下划线的列为主键列.

3.2　子查询展开有效性验证

该实验测试在父查询表很大, 子查询表很小的情况下, 有无主键索引性能上的差异, 另外在无主键索引的情况下, 使用子查询展开优化对性能的影响. 测试父查询表为 SC 表, 表中有 100 万行元组, 下面语句中使用到的 SNO 列为主键列. 子查询表为 COURSE 表, 表中 130 行元组, 其中 CNO 为主键列.

使用主键测试语句为: SELECT* FROM SC WHERE SNO<X AND EXISTS(SELECT * FROM COURSE WHERE CNO=SC.CNO);其执行结果和展开后的执行结果见图2.

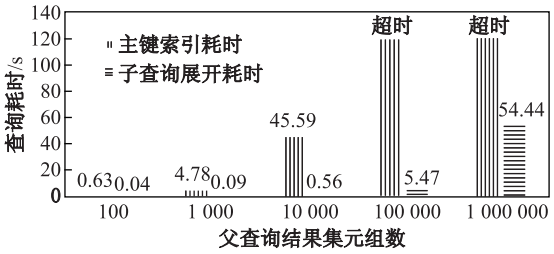

图 2　主键索引的情况下优化后的性能

Fig. 2　Performance of using primary key index and pull up subquery

由图 2 可以看出, 在父查询表很大, 子查询表很小的情况下, 即使有主键索引的情况, 对于分布式而言, 通信开销仍然是不可忽略的. 将子查询展开后性能提升的比较明显.

不使用主键索引的语句为: SELECT* FROM SC WHERE SNO<X AND EXISTS(SELECT * FROM COURSE WHERE CPNO=SC.CNO);其执行结果和展开后的执行结果见图3.

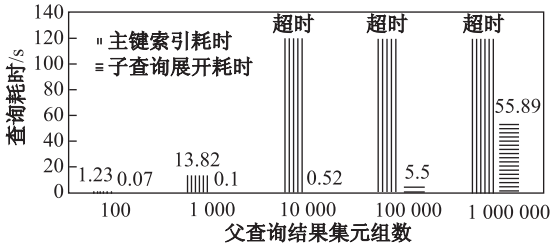

图 3　无主键索引的情况下优化后的性能

Fig. 3　Performance of without primary key index and pull up subquery

从两次实验结果可以看出, 是否使用了主键索引, 性能上并没有提升了很多. 因为在子查询表很小的情况下, 主键索引优化的空间很小. 并且在这种情况下, 通信时间开销是不可回避的. 所以这种情况下, 使用子查询展开技术是非常有效的优化手段.

3.3　主键索引有效性验证

该实验测试在父查询表很小, 但是子查询表很大的情况下, 主键索引和子查询展开对性能的影响. 实验环境同上. 父查询使用 COURSE 表, 子查询使用 SC 表.

使用主键测试语句为: SELECT* FROM COURSE WHERE CNO <X AND EX-

ISTS(SELECT * FROM SC WHERE CNO=COURSE.CNO); 其执行结果和展开后的执行结果见图 4.

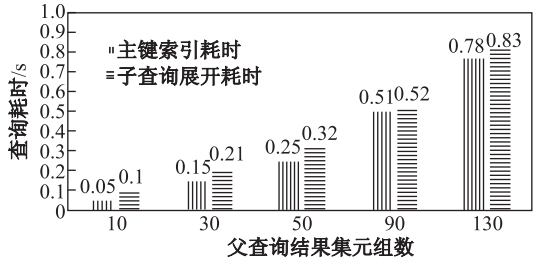

图 4 有主键索引的情况下优化后的性能

Fig. 4 Performance of using primary key index and pull up subquery

由图 4 可以看出, 有主键索引的情况下, 主键的性能优于子查询展开. 因为在父查询表很小的情况下, 在分布式系统中, 通信开销不是很大. 但是由于子查询的表很大, 所以对子查询的优化就有很大的空间. 并且展开子查询后需要很大的空间开销.

不使用主键索引的语句为: SELECT* FROM COURSE WHERE CNO <X AND EXISTS(SELECT * FROM SC WHERE SGRADE=COURSE.CNO); 其执行结果和展开后的执行结果见表 5.

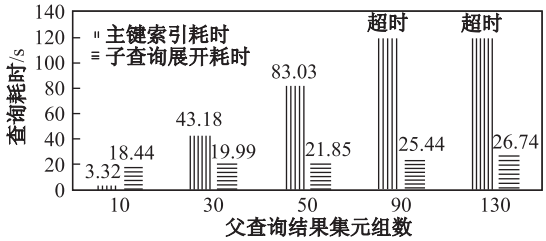

图 5 无主键索引的情况下优化后的性能

Fig. 5 Performance of without primary key index and pull up subquery

由图 5 可知, 子查询展开技术和主键索引都对性能有很大的提升, 但是从子查询展开技术效果比较明显. 但是在有主键索引的情况下, 不展开子查询效果更好一点. 在没有主键索引的情况下, 使用子查询展开较好.

3.4 子查询展开稳定性验证

该实验测试在父查询和子查询表很大的情况下, 主键索引和子查询展开对性能的影响. 测试环境同上. 测试父查询表为 STUDENT 表, 表中存在 100 万行元组, 子查询是 SC 表.

使用主键测试语句为: SELECT* FROM STUDENT WHERE SNO<X AND EXISTS(SELECT * FROM SC WHERE SNO=STUDENT.SNO); 其执行结果和展开后的执行结果见图 6.

由图 6 可知, 在有主键索引的情况下, 如果父查询的结果集比较小, 主键索引的效果更好一点. 这时候对于展开反而不是很好的选择. 但是当子查询的结果集增大时, 分布式环境中的通信开销随之增大, 耗时和子查询的结果集成正比. 但是可以看出, 子查询的结果集对展开的情况没有太大的影响.

不使用主键索引的语句为: SELECT* FROM STUDENT WHERE SNO <X AND EXISTS(SELECT * FROM SC WHERE SNO=STUDENT.SAGE); 其执行结果和展开后的执行结果见图 7.

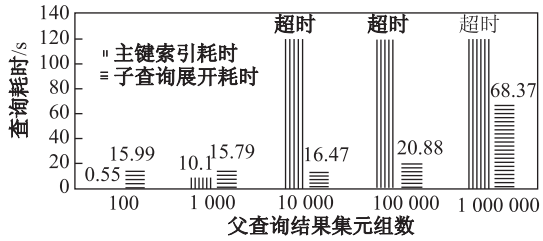

图 6　有主键索引的情况下优化后的性能

Fig. 6　Performance of using primary key index and pull up subquery

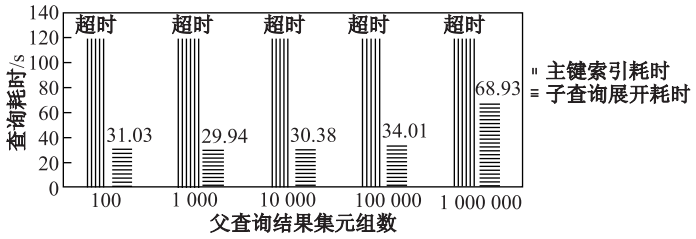

图 7　无主键索引的情况下优化后的性能

Fig. 7　Performance of without primary key index and pull up subquery

由图 7 可知, 主键索引的耗时和父查询的元组数成正比, 在数据集比较小的情况下, 表现突出. 但是父查询的元组数对子查询展开没有太大的影响, 所以这种情况下, 比较适用于选择子查询展开避免过多的通信开销.

4　总　　结

相关子查询是关系数据库的重要功能之一, 按照传统方式执行的相关子查询是一种非常耗时的操作, 对于相关子查询的优化是有重要意义的. 本文研究分析现有相关子查询的优化策略, 并结合分布式数据库的特点, 将无用分支切除、子查询展开, 索引、聚集函数消除等相关子查询优化策略应用于分布式数据库中. 在分布式环境中传统执行策略会产生大量的通信开销, 因此选择子查询展开技术减少通信时间开销, 使用索引降低通信空间开销. 实验了基于OCEANBASE分布式数据库的EXISTS相关子查询, 并在不同的应用场景下比较了不同优化策略对查询性能的影响.

根据实验结果可以得知, 当父查询执行其它过滤条件后的结果集较小, 并且子查询有索引的情况下, 其性能优于子查询展开后的性能. 当父查询结果集比较大, 或者是没有主键索引的情况下, 使用子查询展开技术对性能提升很大. 使用索引的情况, 执行时间和父查询的结果集大小成正比, 但是子查询展开技术相对来说比较稳定, 在大数据的情况下比较适用.

针对子查询不满足展开条件的情况, 如何对其进行优化, 本文没有提出切实的解决方法, 这将在以后的工作中进一步深入研究.

[参 考 文 献]

[1] KIM W. On optimizing an SQL-like nested query[J]. ACM Transactions on Database Systems (TODS), 1982, 7(3): 443-469.

[2] 萨师煊, 王珊. 数据库系统概论[M]. 北京: 高等教育出版社, 2000.

[3] 李海翔. 数据库查询优化器的艺术[M]. 北京: 机械工业出版社, 2014.

[4] SILBERSCHATZ A, KORTH H F, SUDARSHAN S. Database System Concepts[M]. New York: McGraw-Hill, 1997.

[5] CAO B. Optimization of complex nested queries in relational databases[C]//Proceedings of 22nd International Conference on Data Engineering Workshops. [S.l.]: IEEE, 2006: X137.

[6] RAO J, ROSS K A. Reusing invariants: A new strategy for correlated queries [C]//SIGMOD, 1998, 27(2): 37-48.

[7] BELLAMKONDA S, AHMED R, WITKOWSKI A, et al. Enhanced subquery optimizations in oracle[C]//Proceedings of the VLDB Endowment. Germany: DBLP, 2009, 2(2): 1366-1377.

[8] 彭智勇. PostgreSQL 数据库内核分析[M]. 北京: 机械工业出版社, 2012.

[9] KHAN M, KHAN M N A. Exploring query optimization techniques in relational databases[J]. International Journal of Database Theory & Application, 2013, 6(3): 11-20.

[10] 魏士伟, 黄文明, 康业娜, 等. 分布式数据库中基于半连接的查询优化算法研究[J]. 计算机应用, 2007, 27(B06): 34-36.

[11] SHIOI T, HATANO K. Query processing optimization using disk-based row-store and column-store[C]//Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services. New York: ACM, 2015: 69.

[12] CHEN G, WU Y, LIU J, et al. Optimization of sub-query processing in distributed data integration systems[J]. Journal of Network and Computer Applications, 2011, 34(4): 1035-1042.

[13] GALINDO-LEGARIA C, JOSHI M. Orthogonal optimization of subqueries and aggregation[C]//ACM SIGMOD Record. New York: ACM, 2001, 30(2): 571-581.

(责任编辑: 张 晶)

(上接第 44 页)

[5] CORBETT J C, DEAN J, EPSTEIN M, ET A L. Spanner: Google's globally distributed database[J]. ACM Transactions on Computer Systems (TOCS), 2013, 31(3): 8.

[6] CHEN G, VO H T, WU S, et al. A framework for supporting DBMS-like indexes in the cloud[J]. Proceedings of The Vldb Endowment, 2011, 4(11): 702-713.

[7] 翁海星, 宫学庆, 朱燕超,等. 集群环境下分布式索引的实现[J]. 计算机应用, 2016, 36(1): 1-7.

[8] CHANG F, DEAN J, GHEMAWAT S, et al. Bigtable: A distributed storage system for structured data[J]. ACM Transactions on Computer Systems, 2008, 26(2): 4.

[9] TAN W, TATA S, TANG Y, et al. Diff-index: differentiated index in distributed log-structured data stores[C]. Extending Database Technology, 2014: 700-711.

[10] 阳振坤. OceanBase 关系数据库架构[J]. 华东师范大学学报(自然科学版), 2014 (5): 141-148.

[11] 孟必平, 王腾蛟, 李红燕, 等. 分片位图索引: 一种适用于云数据管理的辅助索引机制[J]. 计算机学报, 2012, 35(11): 2306-2316.

[12] 黄贵, 庄明强. OceanBase分布式存储引擎[J]. 华东师范大学学报(自然科学版), 2014 (5): 164-172.

[13] ALIBABA INC. OceanBase[Z/OL].[2016-07-07]. https://github.com/alibaba/oceanbase_0.4.

[14] 杨传辉. 大规模分布式存储系统: 原理解析与架构实战[M]. 北京: 机械工业出版社, 2013.

[15] COOPER B F, SILBERSTEIN A, TAM E, et al. Benchmarking cloud serving systems with YCSB[C]// Proceedings of the 1st ACM Symposium on Cloud Computing. ACM, 2010: 143-154.

(责任编辑: 李万会)