文章编号: 1000-5641(2016)05-0067-08

OceanBase 中基于布隆过滤器的连接算法

茅潇潇, 段惠超, 高 明

(华东师范大学 数据科学与工程研究院, 上海 200062)

摘要: 在大数据时代, "去IOE" 运动的推进以及"双 11"等活动的兴起对分布式数据库系统提出了更高的要求. OceanBase 是阿里巴巴集团自主研发的开源分布式数据库, 支持海量数据跨行跨表事务, 但是对复杂查询的处理性能仍有待提高, 其中连接操作带来的网络传输严重影响了数据库的性能. 本文提出了一种基于布隆过滤器的连接算法, 通过构建布隆过滤器对右表数据进行过滤, 减少了不必要的数据传输开销, 降低了数据处理带来的内存资源的消耗. 本文在 OceanBase 上实现了该算法, 并通过实验证明, 该算法极大提高了连接操作的效率.

关键词: OceanBase; 连接操作; 布隆过滤器

中图分类号: TP311 **文献标识码**: A **DOI:** 10.3969/j.issn.1000-5641.2016.05.008

A join algorithm based on bloom filter in OceanBase

MAO Xiao-xiao, DUAN Hui-chao, GAO Ming

(*Institute for Data Science and Engineering, East China Normal University, Shanghai* 200062, *China*)

Abstract: In the era of big data, the movement of "de-IOE" campaign and the development of activities such as Double 11 have put forward higher request of the performance of distributed database. OceanBase is an open sourced distributed database implemented by Alibaba. It supports for cross-table relational query of massive data but the performance for complex queries remains to be improved. The network transmission overheads caused by join operator seriously influenced the performance of distributed database. This paper proposes a join algorithm based on bloom filter. It filters the data of the right table by constructing a bloom filter on the join column of the left table. The key point of this algorithm is that it reduces the overhead of unnecessary data transmission and the consumption of memory resources by data processing. We implement this algorithm in OceanBase and the experiment results show that the algorithm can greatly improve the efficiency of join operator.

Key words: OceanBase; join operation; bloom filter

收稿日期: 2016-05

基金项目: 国家 863 计划项目(2015AA015307)

第一作者: 茅潇潇, 女, 硕士研究生, 研究方向为分布式数据库. E-mail: jsntmxx@gmail.com.

通信作者: 高 明, 男, 副教授, 研究方向为高可用事务处理优化、数据挖掘等.

E-mail: mgao@sei.ecnu.edu.cn.

0 引 言

为了应对数据规模和业务规模的爆发式增长, 以阿里巴巴公司为代表的互联网行业开始推进“去 IOE”运动, 如何选择数据库产品至关重要. 近几年来, 随着“双 11”活动的发展、“秒杀”应用的兴起、以及云数据服务的推广, 更是对目前开源的分布式数据库系统提出了挑战. 为此, 阿里巴巴集团自主研发了支持海量数据跨行跨表事务的分布式数据库OceanBase[1], 以低成本、高可扩展性、高可用性和高可靠性著称, 已经支持了阿里巴巴集团包括支付宝在内的许多业务.

但是作为主要面向 OLTP 应用的数据库系统, OceanBase 对于复杂查询的处理性能并不高. 当数据量非常庞大时, 连接操作带来的大量数据交互会产生巨大的网络传输开销, 严重影响数据库的性能. 因此, 对于连接操作的优化, 就成为 OceanBase 中查询优化的重点.

迄今为止, 大量的研究学者对分布式数据库中连接操作的查询优化进行了许多研究工作, 遗憾的是, 分布式查询优化技术还很不成熟, 经典的理论不是只局限于某一方面, 就是太过复杂无法应用到实际中. OceanBase 中采用的是经典的排序归并连接算法, 但是它也存在着一些缺陷. 假设, 我们要对关系R和关系S进行连接操作, 当我们在存储关系R数据的节点上进行操作时, 传统的排序归并连接算法要求关系 S 中的所有元组(或者至少为全部元组的一个垂直子集)被发送到该节点上进行计算. 但是在实际的连接计算中, 我们只需要关系R和关系 S 中可能产生连接结果的元组, 不符合连接条件的数据传输浪费了不必要的网络资源, 而且大量数据的排序操作需要较大的内存资源. 因此, 只要拥有足够的信息来判断关系S中各条元组是否符合连接条件, 通过这些信息对关系 S 中的全部元组进行过滤, 仅仅把这个符合条件的子集发送给计算节点, 那么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. 为此, 布隆过滤器是一个理想的选择.

针对这一问题, 本文在 OceanBase 中实现了一种基于布隆过滤器的连接优化算法. 该算法并不将右表的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. 本文的贡献点如下:

(1) 在开源的分布式数据库 OceanBase 上实现了该算法;

(2) 通过一系列测试证明了该算法的高效性;

(3) 提供了一种在 OceanBase 中提高连接操作性能的思路.

本文第 1 节介绍了三种传统的连接算法及其在分布式数据库中的优化技术; 第 2 节形式化定义了该算法所解决的问题, 提出了算法的整体设计并对性能进行了分析; 第 3 节介绍了该算法在开源数据库 OceanBase 上的详细实现; 第 4 节通过一系列实验验证了该算法的性能; 第 5 节总结了论文所做的工作并阐述了未来研究的方向.

1 相关工作

传统的连接算法主要有三种: 嵌套循环连接算法、排序归并连接算法和哈希连接算法, 这三种算法有着各自的特点和使用场景. 随着分布式数据库的产生和发展, 对这三种传统连接算法在分布式数据库中的应用优化技术也被逐渐提出. 下面简要介绍一下这三种连接算法及优化技术.

1.1 嵌套循环连接算法

1977 年, Blasgen 提出了嵌套循环连接算法[2] . 嵌套循环连接算法是一种较简单、稳定的连接算法. 它使用两层嵌套循环, 对于被驱动表的每一行记录, 驱动表的所有记录都会与

其进行比较, 最终得到连接结果. 该算法适用于两表的数据量较小并且内存可以存放的情况, 但如果数据量超过内存大小, 则驱动表就需要进行多次扫描, 扫描的次数为被驱动表大小与可用内存大小的比值. 因此当两表的数据量较大时, 嵌套循环连接算法的效率较为低下.

1.2 排序归并连接算法

针对嵌套循环连接算法的这一缺点, Blasgen 还提出了排序归并连接算法[2]. 在排序归并连接算法中, 两表先根据连接列进行排序, 然后进行顺序扫描, 在扫描的过程中将满足连接条件的元组合并得到最终结果. 该算法适用于大数据量的情况, 曾经被认为是最好的连接算法[3], 但是由于需要对两表进行排序, 排序过程中数据对比操作时间较长, 内存资源消耗也较大. 随着哈希连接算法的提出, 证明排序归并连接算法的优势不一定成立.

1.3 哈希连接算法

第一个以哈希函数为基础的连接算法在 1979 年被 E. Babb 提出[4]. 简单的哈希连接算法的流程如下: 首先选择一张小表作为驱动表, 对小表的连接列使用特定的哈希函数进行计算, 并在内存里产生一张哈希表. 然后, 使用相同的哈希函数对大表(即被驱动表)的连接列进行计算, 将计算的结果在哈希表中进行探测. 如果探测成功, 则一条新的记录将被创建. 在大多数情况下, 哈希连接算法比其他连接算法(如排序归并连接算法)的性能要好[5].

1.4 分布式连接优化技术

分布式查询优化的一个重要目标是减少节点间的数据传输代价. 尽管传统的连接算法的有效性已经在集中式数据库中得到了验证, 但是在分布式环境下, 节点间的数据传输代价是制约查询性能的重要因素. 传统的连接算法在分布式数据库中的应用和优化成为了研究热点.

一种典型的优化方案是采用半连接策略[6]来减少网络传输代价, 降低通信开销. 但是, 半连接需要将驱动表中连接列的值全部传送到被驱动表所在节点, 并且需要在该节点上执行一次额外的连接.

针对以上问题, Chen 等人把布隆过滤器[7]的思想应用到连接算法中[8]. 基于布隆过滤器的连接算法进行了两方面的优化: 首先将驱动表中连接列的值根据不同哈希函数映射到位数组中, 仅将这个位数组进行传输; 其次在被驱动表上执行无序扫描, 不再需要连接操作. L. F. Mackert 等人在论文中指出, 基于布隆过滤器的连接算法比基本的半连接算法性能更为优秀[9]. 因此本文使用了基于布隆过滤器的连接算法, 更好地提高了分布式数据库 OceanBase 连接操作的处理性能.

2 问题定义和算法设计

2.1 问题定义

现有 S 表存储在节点$\text{Snode}_i(i = 1, 2, \cdots, x)$, R 表存储在节点 $\text{Rnode}_j(j = 1, 2, \cdots, y)$. 现将 S 表和 R 表在连接属性 a 上做自然连接: select * from S inner join R on S.a=R.a.

2.2 算法设计

本算法基于分布式架构, 使用布隆过滤器对传统的连接算法进行了优化. 优化后的算法流程如下:

(1) 将 S 表的全部数据从节点 Snode_1, $\text{Snode}_2, \cdots$, Snode_x发送到计算节点 M;

(2) 根据 S 表连接列上的数据构建布隆过滤器 BF_S, 并将 BF_S 分别发送到 R 表数据所在的节点 Rnode_1, $\text{Rnode}_2, \cdots$, Rnode_y;

(3) 在Rnode_1, $\mathrm{Rnode}_2, \cdots$, Rnode_y每个节点上对 R 表的数据进行过滤, 并把 R 表中经过过滤的数据发送到计算节点;

(4) 在计算节点上对两表的数据进行排序归并连接, 并把最终结果返回给客户端.

2.3 性能分析

与传统的排序归并连接算法相比, 基于布隆过滤器的连接算法极大地提高了连接效率.

假设 S 表中元组数为 card(S), 元组的长度为 size(S), R 表中元组的个数为 card(R), 元组的长度为 size(R), 则可得到传统的排序归并连接算法下的网络传输开销T为:

$$T_{\mathrm{old}} = \mathrm{card(S)} \cdot \mathrm{size(S)} + \mathrm{card(R)} \cdot \mathrm{size(R)}. \tag{1}$$

假设布隆过滤器 $\mathrm{BF_S}$包含k个相互独立的哈希函数和一个m位长的位向量. 布隆过滤器在判断一个元素是否属于它所代表的集合时会存在误判: 由于存在哈希冲突, 某一个对应于元素 a 的哈希位可能由于元素 b 的插入被设置为1, 因此减小误称率是非常重要的. 可以通过公式推导得出误称率 $\mathrm{p_{err}}$最小的条件为:

$$\mathrm{e}^{-k\cdot\mathrm{card(S)}/m} = \frac{1}{2}, \quad \text{即 } k = m \cdot \ln(2)/\mathrm{card(S)}. \tag{2}$$

此时, 假设连接选择率为α, R 表经过布隆过滤器过滤后缩减为 R′ 表, 基于布隆过滤器的连接算法下的网络传输开销T为:

$$\begin{aligned} T_{\mathrm{new}} &= \mathrm{card(S)} \cdot \mathrm{size(S)} + m + \mathrm{card(R')} \cdot \mathrm{size(R)} \\ &= \mathrm{card(S)} \cdot \mathrm{size(S)} + m + (\mathrm{card(R)} \cdot \alpha \\ &\quad + (\mathrm{card(R)} - \mathrm{card(R)} \cdot \alpha) \cdot (0.5)^{m\cdot\ln(2)/\mathrm{card(S)}}) \cdot \mathrm{size(R)} \\ &= \mathrm{card(S)} \cdot \mathrm{size(S)} + m + \mathrm{card(R)} \cdot (\alpha + (1-\alpha) \cdot (0.5)^{m\cdot\ln(2)/\mathrm{card(S)}}) \cdot \mathrm{size(R)}. \end{aligned} \tag{3}$$

由于布隆过滤器中位向量的大小 m 相比于T_{new}中的其他两项, 可以忽略不计, 从公式 (3) 可以看出, 基于布隆过滤器的连接算法的网络传输开销明显小于传统的排序归并连接算法下的网络传输开销, 并且连接选择率α越小, 基于布隆过滤器的连接算法节省的网络传输代价越大.

3 算法实现

3.1 OceanBase 架构

Oceanbase 系统架构由四种类型的节点服务器组成:

- RootServer: 主控服务器, 负责数据的负载均衡以及集群节点状态管理等.
- ChunkServer: 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.
- UpdateServer: 更新服务器, 实现事务处理, 负责存储一段时间内的增量数据.
- MergeServer: 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.

本算法实际分为四个阶段:

(1) 将左表即 S 表的所有数据发送到一台 MergerServer 上, 其中既包括 ChunkServer 存储的基线数据又包括 UpdateServer 存储的增量数据;

(2) 在 MergerServer 上根据 S 表的数据在连接属性 S.a 上生成布隆过滤器 BF_S, 并将该布隆过滤器 BF_S 传入右表即 R 表所在的 ChunkServer;

(3) R表所在的 ChunkServer 通过合并 UpdateServer 上的增量数据获得R表的全部数据后, 使用 BF_S 对数据进行过滤, 并将过滤后的数据发送到 MergerServer;

(4) MergerServer 对两张表的数据根据连接类型进行等值连接操作, 再根据不等值条件进行过滤, 并把最终结果返回给客户端.

3.2　布隆过滤器构建

在算法的第二阶段, S 表的数据全部发送到 MergerServer 后, MergerServer 会根据这些数据构建一个能够表示 S 表所有元组的布隆过滤器 BF_S. 通过公式, 我们可以根据设定的误称率和数据量的大小计算出 BF_S中位数组的大小和哈希函数的个数.

布隆过滤器的一个重要问题在于如何使用正确的哈希函数来确保过滤器生效. 在哈希函数的选择方面, 本算法采用了 MurmurHash 函数来提高布隆过滤器的性能和效率. MurmurHash 由 Austin Appleby 于 2008 年创立, 是一种非加密型哈希函数, 适用于一般的哈希检索操作, 具有高运算性能和低碰撞率的特点. 在 Google 的 Guava 开源项目[10]和 LevelDB 高效键值数据库[11]中, BloomFilter(布隆过滤器)类就是基于 MurmurHash 函数来实现的. 它的好处在于, 不需要根据计算得出的k的大小来确定具体的哈希函数, 仅需要对一个哈希函数迭代k次, 就能获得有效的布隆过滤器. 布隆过滤器的构建算法如下:

算法1 布隆过滤器构建算法

输入: S 表元组集合 S, |S|=n
输出: 布隆过滤器 BF_S

```
1  根据误称率 p_err和元素个数 n 计算 MurmurHash 函数的迭代次数 k 和位数组大小 m
2  生成一个 m 位的位数组 BF_S, 并将每一位初始化为 0
3  for 读取 S 表的一条记录 s do
4      if s 不为空 then
5          for i from 1 to k step 1 do
6              将 s 代入 MurmurHash 函数, 计算 h_i(s)的值 V_i;
7              将 BF_S位数组的 V_i位设为 1;
8          end for
9      end if
10 end for
11 布隆过滤器构建结束
```

如第 1 行所示, 先根据设定的误称率p_{err}和元素个数n, 计算布隆过滤器所需位数组的大小m以及 MurmurHash 函数的迭代次数k; 然后如第 3 行至第 10 行所示, 对S表的每一条记录 s 依次作判断, 如果 s 不为空, 则如第 5 行至第 8 行所示, 将 s 依次代入 MurmurHash 函数迭代k次, $h_1(s)$, $h_2(s)$, $\cdots$, $h_k(s)$得到k个值V_1, V_2, $\cdots$, V_k; 再将BF_S位数组的V_1, V_2, $\cdots$, V_k位设为 1, 其余位维持初始化的 0 状态. 当把 S 表的所有记录遍历过后, 布隆过滤器的构建完成.

3.3　布隆过滤器查找

当算法进行到第三阶段, R 表所在 ChunkServer 上的基线数据经过与 UpdateServer 上的增量数据合并获得 R 表的全部数据后, 需要使用BF_S对数据进行过滤, 找出有可能符合等值连

接条件的记录. 布隆过滤器的查找算法如下:

算法2 布隆过滤器查找算法

输入: R表元组集合R, 布隆过滤器BF_S
输出: 符合等值条件的R表记录集合R′

```
1  生成空集R′
2  对于R表中的每一条记录, 执行如下流程
3  for 读取R表的一条记录 r do
4      if r 不为空 then
5          for i from 1 to k step 1 do
6              将 r 代入 MurmurHash 函数, 计算 h_i(r)的值V_i;
7              检查 BF_S位数组的V_i位是否为 1;
8          end for
9          if BF_S位数组的 V_1位至V_k位均为1 then
10             R′=R′∪{r}
11         end if
12     end if
13 end for
14 布隆过滤器查找结束
```

如第 3 行至第 13 行所示, 依次读取 R 表的一条记录 r, 如果 r 不为空, 则如第 5 行至第 8 行所示, 将 r 依次带入 MurmurHash 函数迭代 k 次, $h_1(r)$, $h_2(r)$, $\cdots$, $h_k(r)$得到k个值V_1, V_2, $\cdots$, V_k, 再检查 BF_S位数组的V_1, V_2, $\cdots$, V_k位是否为 1, 如第 9 行至第 11 行所示, 如果全部k个位都为 1, 则将记录 r 添加到集合R′中. 当遍历完 R 表的所有记录后, 输出符合等值条件的R表记录集合 R′.

3.4 等值连接

在算法的最后一个阶段, 将经过布隆过滤器 BF_S 过滤的R表记录集合 R′ 发送到 MergeServer 后, 由于布隆过滤器存在误判, 因此这时获得的 R 表数据是 R 表最终可以进行等值连接操作的元组集合的超集, 所以 MergeServer 还必须对数据进行一次过滤, 以获得最终结果集. 本算法选择使用经典的排序归并连接算法, 在 MergeServer 的内存里对两张表的数据根据连接列排序, 然后对排完序的结果做归并连接, 最后再根据其余不等值条件进行过滤, 并把最终结果返回给客户端.

4 实验评估

为了验证本算法的效率, 本文设计了三组实验, 从选择率、连接列和数据分布对性能的影响三个方面, 通过观察对相同查询语句的处理时间, 分析了基于布隆过滤器的连接算法的性能, 并得出了结论.

4.1 实验环境

实验使用的 OceanBase 集群测试环境是在 OceanBase 开源的 0.4.2 版本上实现上述算法经过优化的版本, 本文使用 4 台虚拟机组成的集群作为测试环境, 每台虚拟机的配置相同, 包括 4 核 1.2 GHz 主频CPU、100 GB 内存、3 000 GB 磁盘, 虚拟机上安装了 CentOS release 6.5 系统, 相互之间通过千兆以太网连接. 集群中的一台虚拟机被配置为 RootServer、MergeServer 和 UpdateServer, 另外三台虚拟机被配置为 ChunkServer. 实验采用的数据是使用数据生成器随机生成的数据.

4.2 实验结果

4.2.1 选择率对性能的影响

该实验对比查询语句中连接列的选择率对 OceanBase 中传统的排序归并连接算法与基于布隆过滤器的连接算法的性能影响. 测试左表包含 10 万条记录, 右表的数据量从 10 万到 1 000

万条记录不等, 两表连接列均为 [1, MAX]的整数, MAX 为记录数.

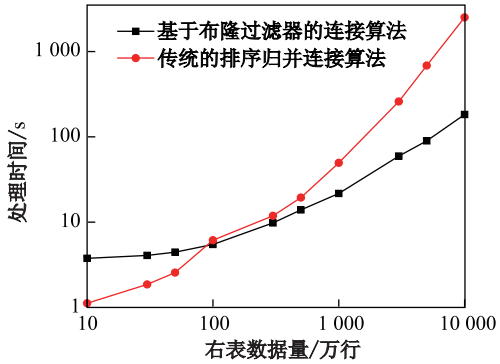

图 1　选择率对性能的影响结果

Fig. 1　Effect of the selectivity on performance

从图 1 中可以看出, 当右表的数据量较小, 即左表对右表的选择率较高时, 布隆过滤器的构建和查找增加了计算开销, 右表数据传输的网络开销降低得并不明显, 基于布隆过滤器的连接算法的处理时间比传统的排序归并连接算法的处理时间要多. 但是当右表的数据量达到 100 万行以上, 即左表对右表的选择率越来越低时, 传统的排序归并连接算法的处理时间大大增加, 而使用布隆过滤器的连接算法的处理时间则呈近似线性增长. 这是因为在选择率较小时, 数据传输的网络代价将会占查询处理时间的主要部分. 布隆过滤器以极低的计算代价, 极大地降低了网络开销, 提高了连接操作的性能.

4.2.2　连接列对性能的影响

该实验对比查询语句中连接列的个数对 OceanBase 中传统的排序归并连接算法和基于布隆过滤器的连接算法的性能影响. 测试左表包含100 万条记录, 右表包含 1 000 万条记录, 连接列的个数从 1 到 7 个不等, 两表连接列均为 [1, MAX] 的整数, MAX为记录数.

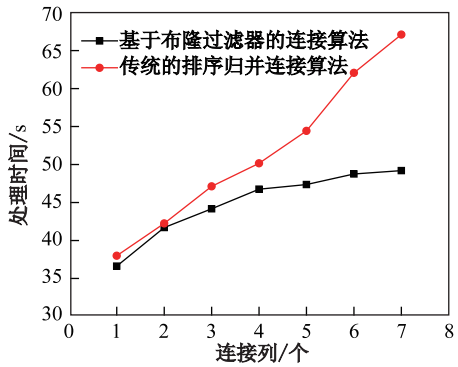

图 2　连接列对性能的影响结果

Fig. 2　Effect of the number of join columns on performance

图 2 表明, 查询语句中的连接列个数增多时, 基于布隆过滤器的连接算法优势更加明显. 布隆过滤器将多个连接列映射到一组对应的位数组, 随着连接列的增多, 布隆过滤器对数据的描述更加准确, 对数据的过滤也更加有效.

4.2.3　不同数据分布对性能的影响

该实验对比连接列中数据不同的分布情况对基于布隆过滤器的连接算法的性能影响. 测试左表包含 10 万条记录, 右表的数据量从 10 万到 1 000 万条记录不等. 为避免数据分布情况对两

表连接结果的大小产生影响, 控制三种数据分布下两表连接的结果集大小相等.

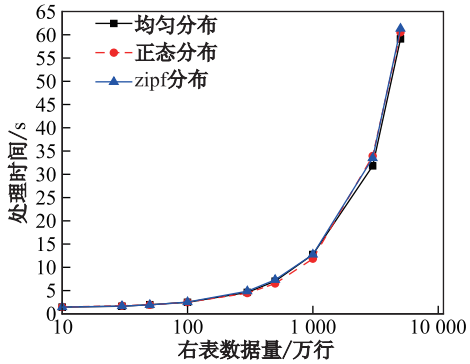

图 3 不同数据分布对性能的影响结果

Fig. 3 Effect of the data distribution on performance

如图 3 所示, 数据的分布情况对布隆过滤器的性能几乎没有影响. 布隆过滤器构建和查找的性能主要由前表和后表的元组数决定, 因此在不同的数据分布下, 布隆过滤器的性能是较为稳定的.

通过以上实验得出, 改进后的连接算法极大地减少了 OceanBase 对连接操作的处理时间, 并且随着选择率的降低和连接列个数的增加, 性能的提高也更加明显.

5 总 结

本文实现了一种分布式数据库中基于布隆过滤器的连接算法, 并在开源数据库 OceanBase 上进行了实验验证. 该算法充分利用了布隆过滤器低空间代价和快速响应的特点, 通过对右表数据使用布隆过滤器进行过滤, 减少了分布式环境下不必要数据的网络传输代价, 降低了数据操作带来的内存资源的消耗, 在连接列的选择率较低的情况下显著提高了连接操作的处理性能.

本文基于目前 OceanBase 数据库的架构, 使用了布隆过滤器对连接算法进行了优化, 但该算法仍有优化空间. 如何对传统的布隆过滤器模型进行拓展, 如何进一步使用 MPP 架构, 将布隆过滤器的计算和数据的连接任务并行地分散到多个节点上, 以及如何根据统计信息选择具体的连接算法, 都是以后对连接优化算法研究工作的重点.

[参 考 文 献]

[1] 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.

[2] BLASGEN M W, ESWARAN K P. Storage and access in relational data bases[J]. IBM Systems Journal, 1977, 16(4): 363-377.

[3] MERRETT T H. Why sort-merge gives the best implementation of the natural join[J]. ACM SIGMOD Record, 1983, 13(2): 39-51.

[4] BABB E. Implementing a relational database by means of specialized hardware[J]. ACM Transactions on Database Systems, 1979, 4(1): 1-29.

[5] SCHNEIDER D A, DEWITT D J. A performance evaluation of four parallel join algorithms in a shared-nothing multiprocessor environment[C]//Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data. ACM, 1989: 110-121.

[6] BERNSTEIN P A, GOODMAN N, WONG E, et al. Query processing in a system for distributed databases (SDD-1)[J]. ACM Transactions on Database Systems, 1981, 6(4): 602-625.

[7] BLOOM B H. Space/time trade-offs in hash coding with allowable errors[J]. Communications of the ACM, 1970, 13(7): 422-426.

(下转第 102 页)