·先进计算与数据处理· 文章编号:1000-3428(2018)06-0024-05 文献标志码:A 中图分类号:TP391

基于共享存储的 MPP 数据库连接执行研究

孙庆鑫[1],雷迎春[2],龚奕利[1]

(1. 武汉大学 计算机学院,武汉 430072; 2. 北京达沃时代科技股份有限公司,北京 100020)

摘 要: 为解决哈希分布表转换为随机分布表后连接效率低的问题,提出一种大规模并行处理数据库中哈希表的并行连接操作算法。根据共享存储环境下哈希分布表的数据块分布特性,并结合随机读取的扫描优势,利用数据多副本分布式存储提高本地读比率,且不损失数据块哈希分布的特性。TPC-H 标准测试结果表明,与传统并行连接算法相比,该算法能有效提高连接操作效率,连接查询语句最高可降低 30% 的响应时间。

关键词: 连接操作;并行连接;大规模并行处理;大数据;在线分析处理

中文引用格式: 孙庆鑫,雷迎春,龚奕利. 基于共享存储的 MPP 数据库连接执行研究[J]. 计算机工程,2018,44(6):24-28.

英文引用格式: SUN Qingxin, LEI Yingchun, GONG Yili. Research on MPP database connection execution based on shared storage[J]. Computer Engineering, 2018, 44(6):24-28.

Research on MPP Database Connection Execution Based on Shared Storage

SUN Qingxin[1], LEI Yingchun[2], GONG Yili[1]

(1. Computer School, Wuhan University, Wuhan 430072, China;
2. Beijing Daowoo Time Technology Co., Ltd., Beijing 100020, China)

【Abstract】 In order to solve the problem of low connection efficiency after the Hash distribution table is converted into a random distribution table, a parallel connection operation algorithm for Hash tables in Massively Parallel Processor (MPP) database is presented. According to the data block distribution characteristics of the hash distribution table in the shared storage environment, combining with the scanning advantage of random reading, data multiple copies distributed storage is used to improve the local reading rate without losing the characteristics of data hash distribution. TPC-H standard test results show that compared with the traditional parallel connection algorithm, this algorithm can effectively improve the connection operation efficiency and reduce the response time of the connection query up to 30%.

【Key words】 connection operation; parallel connection; Massively Parallel Processing (MPP); big data; Online Analytical Processing (OLAP)

DOI: 10.19678/j.issn.1000-3428.0047104

0 概述

随着信息技术的发展和应用,在互联网浪潮下产生了海量的数据,发掘海量数据下隐藏的价值成为当前主要任务,于是对于海量数据的分析越来越受到各行业的重视。文献[1-3]阐述了大数据和在线数据分析引擎的现状。文献[4-6]讨论了在线数据分析平台所面临的问题和诸多挑战。文献[7-10]主要介绍了被大量分析平台使用的 Hadoop 分布式文件系统(HDFS)的系统特性以及关系型数据库中连接操作的实现算法。上述系统特性和算法可以加速共享平台下连接操作的执行效率。传统的大规模并行处理(MPP)架构分析引擎是将计算资源和存储资源为同一单位进行并行,如文献[11]提到的 GreenplumDB 架构,每个计算节点只负责处理存储在本地的数据,这种架构借助哈希分布有着高吞吐量和低查询延迟的优点,但是系统扩展性差。文献[12]提出的基于共享存储的 MPP 数据仓库,是将用户海量数据以关系型数据库中关系表的格式存储到 HDFS 中,通过 MPP 架构执行查询语句的在线数据分析平台,它是一种将传统 MPP 架构的计算和存储资源进行分离的新型结构,即它的计算节点和存储节点可以分别按需扩展。在扩展性增强的同时,由于计算节点读取的数据总不在本地,需要通过高延迟的网络读取,导致查询处理速度变慢。文献[12]提出可以通过将哈希分布的数据表转换为随机分布可以获取大量

基金项目: 国家自然科学基金青年科学基金(61100020);国家自然科学基金面上项目(61572373)。

作者简介: 孙庆鑫(1990—),男,硕士研究生,主研方向为分布式数据库、数据仓库技术;雷迎春,博士;龚奕利,副教授、博士。

收稿日期: 2017-05-08 **修回日期:** 2017-06-09 **E-mail:** qingxinwhu@qq.com

的本地数据读取优势,加速查询的执行,但是由于失去了数据记录分布的特性,导致连接效率较低。在关系型数据库中连接 2 个表有 3 种连接方式,其中的哈希连接由于其具有较低的执行代价,并且适合传统流水线类型的查询执行方式,在查询执行计划中被经常使用。文献[13]提出 Hybrid Hash Join 算法,该算法用于改进普通 Hash Join 连接的执行效率,算法通过最大限度地命中缓存,从而提高查询执行速度。通过以上分析发现,基于共享存储的 MPP 分析引擎存在用户数据表的分布和连接执行方式之间的矛盾,导致连接效率低下。

本文针对基于共享存储环境下的 MPP 数据库分析引擎,提出一个高效的分类并行哈希连接(Classified Parallel Hash Join,CPHJ)操作符操作算法。在 MPP 环境下,一个查询语句的执行需要多个逻辑查询执行进程同时执行,每个进程处理一个关系表的部分数据。CPHJ 算法通过执行节点被查询优化器分配的表数据块的哈希特性进行归类,并为存在的每一类数据块集合启动相应的连接操作线程,连接节点设有缓冲区用于保存中间临时结果。

1　问题描述

基于共享存储的 MPP 分析引擎将数据读写操作分配给分布式文件系统,引擎中的计算节点通过客户端访问关系表。在分布式文件系统中,一个关系表 A 由数据块 $<A_0,A_1,\cdots,A_n>$ 组成,每个数据块由存储在不同节点上的 3 个备份 $<A_{i0},A_{i1},A_{i2}>$ 组成。MPP 查询优化器根据集群计算资源和数据量大小为每个查询语句分配不同数量的逻辑查询执行节点(VSeg)。一个关系表有 2 种分布方式:随机分布和哈希分布。哈希分布的关系表在逻辑上被划分成了多个子文件,每个子文件的数据块被共享存储系统分布到集群中。查询优化器在哈希分布表上做连接时为每个逻辑查询执行节点分配一个子文件。随机分布在逻辑上是一个大文件,它的文件数据块被查询优化器按照一定策略分配给执行节点 VSeg,即查询优化器分配关系表文件数据块到逻辑执行进程上时如果表是哈希分布,则分配单位是子文件,如果表是随机分布,则分配单位是数据块。

如表 1 所示,查询优化器为一个查询语句分配了 8 个执行节点 <VSeg1 ~ VSeg8 >,A 表数据具有 42 个数据块,每个块用 $A_{i,j}(0\leqslant i<42,0\leqslant j\leqslant 2)$ 表示,其中,i 表示块号,j 表示备份号。当表 A 为哈希分布时,每个执行节点被分配一个完整的子文件,那么就必须通过网络读取被分配的子文件的所有数据块,而数据块分布在集群中,所以,扫描表操作具有较高的读取代价。因此,在基于共享存储的 MPP 系统中,查询优化器将哈希分布关系表按照随机分布表对待,根据最大本地读取量原则为每个逻辑执行节点分配本地的数据块。同时,因为数据块有3 个备份,如果一个逻辑执行节点所在的物理存储节点上具有连续的数据块(不同数据块在同一物理节点有备份),则分配连续的数据块。这样可以减少扫描操作符多次打开同一文件的系统消耗。

表 1　A 表数据块分配示例

节点	数据块
1	$A_{1,0}+A_{2,1}+A_{3,2}+A_{4,1}+A_{5,0}+A_{7,1}$
2	$A_{6,1}+A_{8,2}+A_{9,2}+A_{10,1}+A_{11,2}+A_{12,1}+A_{13,0}$
3	$A_{14,0}+A_{15,2}+A_{16,0}+A_{17,1}+A_{19,2}$
4	$A_{18,0}+A_{20,2}+A_{21,0}+A_{22,1}$
5	$A_{23,0}+A_{24,0}+A_{25,2}+A_{26,0}+A_{27,1}+A_{28,2}$
6	$A_{29,0}+A_{30,2}+A_{31,0}+A_{32,1}+A_{33,2}$
7	$A_{34,0}+A_{35,2}+A_{36,0}+A_{39,1}$
8	$A_{37,0}+A_{38,2}+A_{40,0}+A_{41,1}+A_{42,2}$

在查询优化阶段可以获取如表 1 所示的每个逻辑执行节点所处理的数据块集合,当关系表体积变大后每个逻辑执行节点会被分配大量的数据块,由于按照随机分布方式分配数据块,因此没有哈希关系表数据块的分布特性,且只有一个进程在处理庞大数据块的连接,这将导致在大量数据情况下执行进程响应时间过长的性能问题。这样,就需要新的算法在哈希分布表被随机化对待时通过结合哈希分布特性加速 VSeg 进程本地连接操作。

2　分类并行连接算法分析

针对上述单个进程在执行大量数据连接操作时延迟过高的问题,本文以哈希分布表的数据块分布特性为基础,提出一种分类并行连接算法(CPHJ)。CPHJ 算法分为初始化和执行 2 个步骤。

在初始化阶段,首先查询执行节点在初始化连接操作时通过收集扫描节点扫描的本地数据块元数据信息,获取其中的子文件号信息(Segno)。然后把所有的数据块所属子文件号信息归类整理,将相同子文件号的数据块用链表保存起来,表示一个分类,那么执行单位获得所拥有数据块分类链表 $<C_0,C_1,\cdots,C_m>$。算法通过遍历所有数据块信息将其分类,复杂度为 $O(N)$。最后为连接节点的执行做准备,开启 m 个线程 $<T_0,T_1,\cdots,T_m>$,T_i 线程负责 C_i 类数据块上的连接操作。

算法 1　CPHJ 初始化算法

输入　执行单元 VSeg 的数据块元数据链表(Split Metadata List,SML):$<S_0,S_1,\cdots,S_n>$

输出　属于同一个子文件数据块集合组成的链表

```
1. int T = 0;
2. List splitSet_List
3. For Split s in SML do;
4.      int bucketno = s→segno;
5.      if (find (splitSet_List,bucketno)) do;
```

```
6. //找到了新的子文件编号
7.          splitSet_List. add_set(bucketno);
8.          T++;
9. end if;
10. //写入已经存在的元数据链表中
11.      splitSet_List[bucketno]. append(s);
12. end for;
```

当连接执行线程被创建后,每个线程则开始执行连接算法。每个线程给定了输入数据集合,即初始化算法给定的数据块集合 C_i。每个 C_i 集合中的数据具有一个特点,即所有数据块中的数据记录在分布键上做哈希处理后具有相同的分类值。这是并行连接执行的基础,每个线程只负责当前数据块链表的连接操作,处理本集合数据块和连接操作的另一端输入的连接操作,每个连接工作线程根据连接操作符中的连接约束条件对所属数据集合进行筛选连接,连接结果放入连接操作符的缓冲区供上层执行树节点使用。线程的连接工作就是 CPHJ 算法的执行过程,首先为关系数据记录建立哈希表,然后遍历所有被连接数据块中的记录值。所以,算法复杂度为 $2 \times O(N)$。

算法 2 CPHJ 执行算法

输入 CPHJ 初始化生成的同一类数据块集合 splitSet_List[t]: $<Block_0, Block_1, \cdots, Block_n>$

输出 连接结果 Record

```
1. 数据块分配过程
2. Hash Table htable
3. Join Plan:Node
4. record_l = getRecord(Node→leftNode,t);
5. //节点第一次执行需要建立哈希表
6. if (htable = = NULL) do:
7.      for block in splitSet_List[t] do:
8.          for record_r in block do:
9.              insert record_r into htable;
10.     end for;
11.     end for;
12. end if;
13. //遍历所有数据块进行数据匹配操作
14. while (record_l ){
15.     hv = HASH(record_l)
16.     if (find (htable,hv)&& satisfy join condition)
17.         Return Project(record_l,record_r);
18.     record_l = getRecord(Node→leftNode,t);
19. }
```

下面证明算法的正确性,2 个基本关系表 A 和 B 做连接操作。A 表按照哈希分布,虽然被优化器当作随机表对待,但是其中的记录遵循哈希分布的特性。在 MPP 架构下,表 A 被哈希分布在 Bucket 个子文件中,也就是 $A = A_0 + A_2 + \cdots + A_{Bucket-1}$,$A_i$ 是一个子文件,同一个子文件中的所有记录行都在分布键上具有特性:Record(r)% Bucket = = i。将表 B 在连接键上使用同样的哈希分布,于是 A 表连接 B 表就转换为 $(A_0 + A_2 + \cdots + A_{Bucket-1}) \bowtie (B_0 + B_2 + \cdots + B_{Bucket-1})$。那么,相同属性上具有相同值的记录在哈希运算后一定位于同样的子文件中,即 A 表连接 B 表就只需要将对应的子表进行连接,而省去了大量的无用连接判断,同时可以使用多线程并行执行连接操作,线程间不需要交换数据。于是,A 连接 B 也就等同于:

$$A \bowtie B = A_0 \bowtie B_0 + A_1 B_1 + \cdots + A_{Bucket-1} B_{Bucket-1}$$

同样地,上述定理适用于单个执行进程上分配的哈希分布表中数据块组成的集合上。将查询优化器分配的数据块集合当作一个完整的表,按照所属子文件的不同进行分类,将相同子文件的数据块和外表做连接。CPHJ 算法节约了大量的连接计算,同时拥有并行特性。连接操作符中设置了缓冲区,存放中间结果,当上层操作符获取本执行节点的结果时可以迅速响应。

3 CPHJ 算法实现

本文实现和测试均基于 HAWQ[12] 分析引擎,HAWQ 基于 HDFS 分布式文件系统的 MPP 架构数据库在线分析平台。CPHJ 算法设计实现包括连接节点初始化实现、连接节点中缓冲区队列实现、节点并行连接操作的执行实现 3 个部分。CPHJ 算法对连接节点的初始化工作,首先通过计算数据块元数据信息的分配,确定一个连接执行节点上应该启动并行线程的个数 T。实现缓冲区队列用来存放连接节点的中间结果,连接线程作为生产者向队列中存放中间结果,调用本执行节点的上层节点作为消费者从缓冲区中取出中间结果。执行实现则是具体的连接操作过程用于调用下层执行计划节点产生中间结果。

如图 1 所示,使用 TPC-H[14] 标准数据库测试程序生成 Q3 查询语句的查询计划树,当哈希分布的 Orders 表被当作随机分布时,选择将另外一个连接表 Customer 进行广播操作处理。此时,将两者连接起来的 Hash Join 操作符就可以使用 CPHJ 算法优化连接性能。

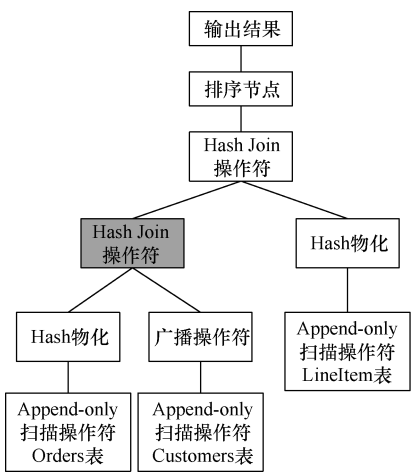

图 1 TPC-H Q3 查询计划树

图 2 所示为 CPHJ 执行算法在哈希连接操作符中的实现。连接操作符中通过无锁队列共享缓冲区将操作符的工作一分为二，就演变成了生产者和消费者的问题。在 CPHJ 算法中，并行执行的线程是生产者，执行计划上层操作符是消费者。生产者将连接操作产生的中间结果放入共享缓冲区，消费者从缓冲区中取走中间结果进行进一步的处理。每个连接线程 T_i 只需要从 CPHJ 初始化算法得出的数据块链表 splitSet_List[t]中获取记录，形成 Hash Table，从另外一端输入中获取记录在 Hash Table 中连接查找，如果满足连接节点的约束条件，则将连接结果放入缓冲区。

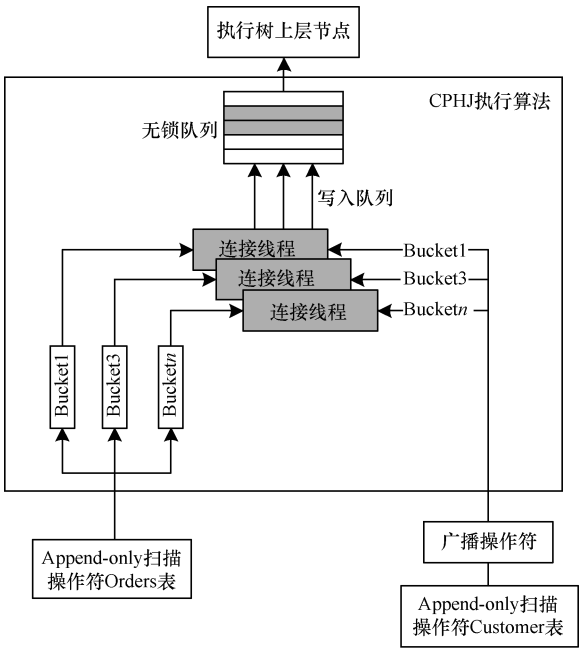

图 2　CPHJ 执行算法实现结构

为了同步生产者和消费者，同时避免加锁对性能产生影响，在实现阶段设计了无锁队列[15]。无锁队列使用数组存储的形式，通过一般 CPU 都支持的 Compare and Swap(CAS)原子指令实现。

4　实验结果与分析

4.1　实验设计

本文实验在由 1 个 Master 节点和 3 个 Slave 节点组成的集群系统中进行，每个节点具有相同的软硬件配置(Xeon CPU 2.6 GHz，4 GB 内存，200 GB 硬盘)。测试工具使用 TPC-H[11]关系数据库标准测试程序。CPHJ 算法是针对连接查询语句的优化，所以采用了 Q3、Q5 和这 Q11 这 3 个连接类型的查询语句对算法进行性能测试。查询语句所涉及的表均采用哈希分布方式来存储。本次测试的评判标准是响应时间 D，响应时间 D 由测试起止 $T_{start} - T_{end}$ 时间得到。本文实验采用了 4 种不同大小规模的数据集合进行测试(2 GB、4 GB、8 GB、16 GB)并行连接算法优化表连接操作在查询响应上的性能提升。

4.2　结果分析

基于数据块分布的并行连接算法以提取数据块分布特征为基础。在测试对比方面，本文选择文献[13]。因为文献[13]是关系型数据库领域中应用最广、最具代表性的连接算法。Q3、Q5 响应时间对比测试结果如图 3、图 4 所示。

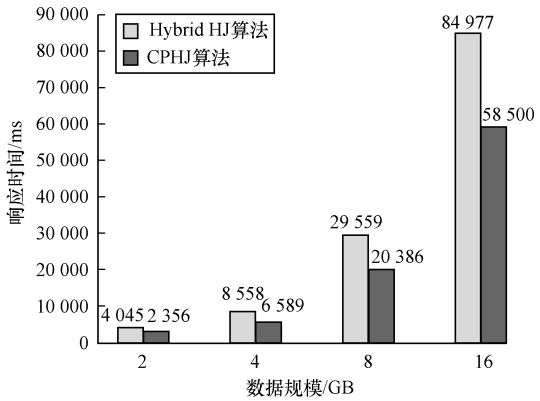

图 3　Q3 响应时间测试结果

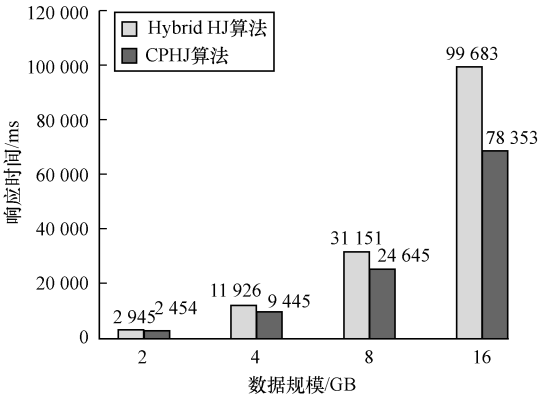

图 4　Q5 响应时间测试结果

由实验数据可知，本文连接方法在 2 个连接查询语句中均展现出了较好的查询性能。这是由于基于数据块分布的连接算法提取了数据块分布特征，将相同子类的数据块进行连接，在转换为随机分布表情况下节省大量无用的记录匹配过程。对于大文件的连接操作提升更为明显，在 8 GB 数据规模以上有 30% 的性能提升。基于共享存储的 MPP 数据库中执行节点是根据数据量大小弹性变化的，当数据量增大后，由于随机分布表带来的广播操作不具有哈希分布特性，因此在大文件上文献[13]的连接操作失去了带来性能的数据局部性。而本文算法在数据分布上的分类连接则降低了广播操作对连接操作的局部性影响，对于在大文件上的连接操作有更高效的表现。如图 5 所示，本文算法在相同数据量下对 Q3 的查询响应提升在不同数据量下均略高于 Q5，这是因为 Q5 具有 6 个连接表，而 Q3 只有 3 个连接表。连接查询语句的执行由于连接表的增多和连接键的选择不同，会导致执行计划中高代价的广

播和重分布操作变多,从而连接操作在查询语句整体的执行代价中比例变小,优化提升略低于连接表较少的 Q3。

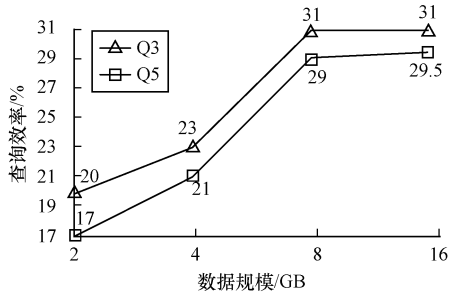

图 5 Q3 和 Q5 效率提升对比

综合 2 个连接语句查询可以看出,在数据规模 4 GB 以上,本文使用的基于数据块分布特征的并行连接算法的优势较为明显,平均较传统方法有 25% 的性能提升,最大有 30% 的性能提升。

5 结束语

本文提出一种基于数据块分布的并行连接算法(CPHJ)。CPHJ 算法根据共享存储环境下哈希分布表的数据块分布特性,结合随机读取的扫描优势,解决了哈希分布表转换为随机分布表后连接效率低的问题。首先在初始化阶段归类数据块,然后并行执行一个逻辑执行节点上的连接操作。该算法具有低延迟连接操作高效的特点,能够有效提高连接类查询语句的执行效率,降低响应延迟时间。通过在集群环境下数据库标准测试工具 TPC-H 的测试,结果表明,CPHJ 算法适合在大数据量下的执行拥有较少关系表连接查询语句,与传统的逻辑查询执行节点上单进程执行连接操作相比,连接查询执行性能提升 30% 。本文算法只适合哈希分布的表模式,如何将随机分布模式下的表在单个逻辑执行进程上实现多线程并行连接是下一阶段的工作。

参考文献

[1] 李国杰,程学旗. 大数据研究:未来科技及经济社会发展的重大战略领域——大数据的研究现状与科学思考[J]. 中国科学院院刊,2012,27(6):5-15.

[2] 黄晓云. 基于 HDFS 的云存储服务系统研究[D]. 大连:大连海事大学,2010.

[3] 肖 凌,刘继红,姚建初. 分布式数据库系统的研究与应用[J]. 计算机工程,2001,27(1):33-35.

[4] 朱 珠. 基于 Hadoop 的海量数据处理模型研究和应用[D]. 北京:北京邮电大学,2008.

[5] 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(1):34-36.

[6] 刘大昕,张春林,聂亚杰,等. 数据仓库与技术[J]. 计算机仿真,2003,20(5):40-43.

[7] CHAUDHURI S, DAYAL U. An overview of data warehousing and OLAP technology[J]. ACM Sigmod Record,1997,26(1):65-74.

[8] FLORATOU A, MINHAS U F, ZCAN F. SQL-on-Hadoop: full circle back to shared-nothing database architectures[J]. VLDB Endowment, 2014, 7(12): 1295-1306.

[9] ABADI D, BABU S, ZCAN F. SQL-on-hadoop systems: tutorial[J]. VLDB Endowment,2015,8(12):2050-2051.

[10] BORTHAKUR D. The Hadoop distributed file system: architecture and design[J]. Hadoop Project Website, 2007,11(11):1-10.

[11] WAA S, FLORIAN M. Beyond conventional data warehousing——massively parallel data processing with Greenplum database[C]//Proceedings of International Workshop on Business Intelligence for the Real-time Enterprise. Berlin, Germany: Springer, 2008: 235-243.

[12] CHANG L, WANG Z, MA T, et al. HAWQ: a massively parallel processing SQL engine in hadoop[C]// Proceedings of ACM SIGMOD International Conference on Management of Data. New York, USA: ACM Press, 2014: 1223-1234.

[13] LIU L C H, BAERENWALD L L, PLASEK J M, et al. Hybrid hash join process: USA, US 6263331 B1[P]. 2001-05-31.

[14] BONCZ P, NEUMANN T, ERLING O, et a. TPC-H analyzed: hidden messages and lessons learned from an influential benchmark[C]//Proceedings of Conference on Performance Evaluation and Benchmarking. Berlin, Germany: Springer, 2013: 124-132.

[15] GIACOMONI J, MOSELEY T, VACHHARAJANI M. Fast forward for efficient pipeline parallelism: a cache-optimized concurrent lock-free queue[C]//Proceedings of ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming. New York, USA: ACM Press, 2008: 43-52.

编辑 索书志