第42卷 第11期 计 算 机 学 报 Vol. 42 No. 11
2019年11月 CHINESE JOURNAL OF COMPUTERS Nov. 2019

面向非易失性存储器的多表连接写操作的优化研究

马竹琳[1] 李心池[1] 诸葛晴凤[3] 吴 林[1] 陈咸彰[1,2] 姜炜文[1] 沙行勉[3]

[1](重庆大学计算机学院 重庆 400044)

[2](重庆大学通信工程学院 重庆 400044)

[3](华东师范大学计算机科学与软件工程学院 上海 200062)

摘 要 多表连接操作是嵌入式数据库、数据仓库等系统中的一个重要操作.因此,提升多表连接的性能能够加快数据处理和分析的速度,进而提升系统的整体性能.新型的非易失性存储器(Non-Volatile Memory,NVM)具有内存级读写速度、存储密度高、可字节寻址和持久化等优点,成为补充或替代DRAM的新型存储设备.然而,直接将现有的多表连接算法应用在NVM上会带来两个问题:(1)现有算法不能充分发挥新型非易失性存储器的优势,无法展现较优的性能;(2)连接算法会生成大量中间表,对存储设备造成大量写操作.由于NVM的写耐受度有限,现有多表连接操作极易造成NVM的损坏.该文考虑NVM写耐受度有限的特性,旨在减少多表连接操作引起的对NVM的写操作.首先,该文提出优化连接顺序的NVjoin算法,该算法解析不同表之间的关联性,并通过采样的方法估算中间结果的大小,从而选择较优的连接顺序,尽可能减少NVM上的写操作.其次,该文设计了一个组织中间结果的数据结构——LWTab,该结构充分利用了NVM可字节寻址的特性,通过存储数据的地址而非数据的方式,进一步减少连接过程中中间结果所产生的NVM写操作.该文利用DRAM模拟NVM进行大量的测试实验,结果表明,该文提出的算法在时间性能与NVM写次数两个方面均得到提升:与MySQL所提供的连接顺序相比,NVjoin可以减少104.21倍的NVM写操作并提升65.01%的性能.除此之外,LWTab可以在NVjoin的基础上,进一步减少16.74倍的NVM写操作以及提升71.86%的性能.

关键词 非易失性存储器;多表连接;连接顺序;数据库

中图法分类号 TP311 **DOI号** 10.11897/SP.J.1016.2019.02417

《计算机学报》

A Research of Reducing Write Activities in Multi-Table Join for Non-Volatile Memories

MA Zhu-Lin[1] LI Xin-Chi[1] ZHUGE Qing-Feng[3] WU Lin[1] CHEN Xian-Zhang[1,2]
JIANG Wei-Wen[1] SHA Edwin H-M[3]

[1](*College of Computer Science, Chongqing University, Chongqing* 400044)

[2](*College of Communication Engineering, Chongqing University, Chongqing* 400044)

[3](*School of Computer Science and Software Engineering, East China Normal University, Shanghai* 200062)

Abstract Multi-table join operation is one of the most important operations in lots of systems, such as embedded databases and data warehouses. Hence improving the performance of Multi-table join operation can accelerate the speed of data processing and analysis. The overall performance of the systems can be also enhanced. Emerging non-volatile memories (NVMs) have been widely discussed to complement or substitute DRAM in memory hierarchy in recent years. When the

收稿日期:2018-01-09;在线出版日期:2018-09-25.本课题得到国家“八六三”高技术研究发展计划项目基金(2015AA015304)、国家自然科学基金(61472052)、中国博士后科学基金(2017M620412)资助.**马竹琳**,博士研究生,主要研究方向为数据库的设计、非易失性存储器和优化算法.E-mail:mazhulin1993@gmail.com.**李心池**,硕士研究生,主要研究方向为内存数据库.**诸葛晴凤**(通信作者),博士,教授,博士生导师,中国计算机学会(CCF)会员,主要研究领域为操作系统、嵌入式系统和软件、优化算法.E-mail:qfzhuge@gmail.com.**吴 林**,博士研究生,主要研究方向为文件系统.**陈咸彰**,博士,讲师,中国计算机学会(CCF)会员,主要研究方向为新型内存系统、文件系统、嵌入式系统和软件、云计算.**姜炜文**,博士研究生,主要研究方向为高层次综合、实时系统、供应链管理、非易失性存储器和优化算法.**沙行勉**,博士,教授,博士生导师,中国计算机学会(CCF)高级会员,主要研究领域为新型内存系统、操作系统、云计算和嵌入式系统和软件.

NVM is incorporated into memory hierarchy, it has fast read/write speed, high storage density, byte addressability, low power consumption and persistency. However, employing existing multi-table join algorithms directly on NVM incurs two big problems: (1) existing algorithms cannot fully exploit the benefits of NVM which may stand in the way of improving the performance of systems. (2) Existing algorithms produce large intermediate table size and a large number of copies of data, which causes a lot of NVM writes to the storage device. Since NVM suffers from the limitation of write endurance (i.e., once the number of writes of a memory cell exceeds the endurance limit, the NVM chip is considered worn-out), existing multi-table join algorithms can easily make NVM device breakdown. In this paper, we consider the limited write endurance of NVM and aim to reduce the number of NVM writes caused by multi-table join algorithms. First, we propose an algorithm, NVjoin, to select an optimal order for joining multiple tables. NVjoin algorithm analyzes the relation between tables according to the joining statements entered by user. And then estimate the size of the intermediate tables using sampling techniques. In this way, we can obtain an optimal order so as to reduce the number of NVM writes. Second, a new data structure, LWTab, is designed to keep the intermediate results of multi-table join algorithms. The new structure makes full use of the byte-addressability of NVM. Specifically, instead of storing data, we store the address of data in LWTab, which can significantly reduce the number of copies of data and the size of intermediate tables. In this way, the number of NVM writes can be also significantly reduced. This paper employs DRAM to simulate NVM and a lot of experiments (including timing performance, number of NVM writes) have been carried out with three different join algorithms. Experimental results show that the algorithms proposed in this paper achieve better results not only in timing performance but also in number of NVM writes. Specifically, compared with multi-table join algorithms in MySQL, a widely used database, NVjoin algorithm can reduce the number of NVM writes by 104.21 times and achieve 65.01% improvements in timing performance. Moreover, when LWTab structure is employed, the NVjoin algorithm can further reduce the number of NVM writes by 16.74 times and achieve 71.86% improvements in timing performance.

Keywords non-volatile memory; multi-table join; join order; database

《计算机学报》

1 引 言

随着无线传感网络和物联网技术的发展,学术界提出了多种不同的信息物理系统(Cypber-Physical Systems,CPS)模型. 在信息物理系统中,各种传感器和操作程序构成了一个嵌入式系统. 传感器部署在周围环境中,用来实时监测各种数据. 这些传感器将采集的数据存放到嵌入式数据库中,用于后期数据挖掘等应用的特征分析[1]. 为了加速数据查询,这些嵌入式系统通常使用闪存和非易失性内存NVM,例如相变存储器PCM[2]. 这些非易失性内存具有持久性、高性能、高能量和高效率等特性. 但是,NVM还存在写容忍度受限和读写不对称等问题. 因此,在基于NVM的内存数据库设计中,应着重考虑减少NVM上的写操作.

多表连接(multi-table join)是数据库分析数据时的重要操作. 在基于NVM的内存数据库架构下,执行多表连接操作应考虑以下两个方面. 第一,不适当的连接顺序可能影响连接操作的性能并产生过多的NVM写次数. 用户输入的连接顺序并没有考虑到表的大小及关联性等特性. 因此,我们需要用算法来优化多表连接顺序以提升操作的整体性能. 第二,在连接过程中,中间结果同样需要存储在NVM上,这必将导致大量的不必要的NVM写. 因此,我们需要设计一个简洁的数据结构重新组织这些中间结果.

至今,已有一些优化连接顺序的方法. 例如动态规划算法[3]、遗传算法[4-5]、蚁群算法[6]和启发式算法[7-9]. 这些优化算法在基于硬盘的系统中能起到较

好的效果.但是,它们并没有充分考虑 NVM 可字节寻址和受限的写容忍度的特性.

本文提出了一个"NVM 友好"的 NVjoin 算法优化连接顺序.我们发现在两个表连接的过程中,如果没有连接条件,这两个表会做笛卡尔积操作,而该操作会产生大量的 NVM 写次数.在 NVjoin 算法中,我们充分考虑了表之间的关联性,并尽量减少不必要的笛卡尔积操作.该算法包括 3 个阶段:(1)分类;(2)估算并连接;(3)笛卡尔积.第一阶段,算法首先根据输入的连接条件对所涉及的表分类,不同类别之间的表没有任何关联性.第二阶段,对于每一个类,算法迭代执行估算和连接操作,直到类变为空为止.第三阶段,对不同的类得到的连接结果执行笛卡尔积操作,从而得到最终的连接结果.

为了更进一步减少 NVM 的写次数,本文提出了 LWTab.它是一个轻量级的表结构,通过避免在连接过程中冗余数据的拷贝,尽可能地减少中间结果的大小.NVM 可字节寻址的特性使表中的每个元组(表中的一行)都有自己的地址,CPU 可以通过 load/store 指令直接访问这些元组.因此,LWTab 只需存储相应中间结果的元组地址,而不是元组中数据,即可减少对冗余数据的拷贝.本文将 NVjoin 和 LWTab 的结合叫作 NVJoin+LWTab.本文对不同的连接方法,以 NVM 写次数和执行时间为衡量标准做了大量实验.实验结果表明,NVJoin+LWTab 在保证整体性能的同时相比于左深树连接[10]、浓密树连接[10]和 MySQL 连接,NVM 写次数分别减少 59171.06 倍、183.65 倍和 1744.63 倍.

本文的主要贡献如下:

(1)本文提出了一个有效的多表连接优化算法(NVjoin),该算法通过优化连接顺序尽可能地减小中间表的大小和执行笛卡儿积操作的次数,进而提升多表连接的整体性能.此外,对于具有写耐受性的 NVM 介质,该算法还能够尽可能地减少 NVM 的写次数.

(2)本文在 NVjoin 算法的基础上提出新的数据结构——LWTab.该结构充分利用了 NVM 的特性,在多表连接的过程中可以作为辅助结构,减少 NVM 写次数.

(3)实验结果表明,与现有的技术相比,NVjoin+LWTab 技术可以在保证性能的同时极大减少 NVM 的写次数.

2 背景

2.1 非易失性存储器

嵌入式数据库系统早已被广泛应用于 CPS 中,但是嵌入式计算系统经常会受到存储空间、计算能力和能耗等问题限制[11].而新兴的可字节寻址的非易失性存储器(NVM),如 PCM,因其与 DRAM 相比有较高的密度和更低的能量消耗而有望替代 DRAM[1,12-13]作为主存,从而构建非易失的嵌入式内存系统.在 NVM 作主存的架构下,直接将数据库存放在 NVM 上即可保证其非易失性,无需刷回后备存储设备中.此外,该架构还可以直接通过内存总线访问数据.本文称这样的系统为基于 NVM 的内存数据库系统.但是,在设计基于 NVM 的系统时,设计者需根据 NVM 的特性考虑以下两个方面.其一,NVM 具有严重的读写时延不对称的问题.比如 PCM 的写时延比读时延慢 4~20 倍[14].其二,与 DRAM 不同,NVM 有一定的写容忍度,比如 PCM 只能容忍 107 和 108 次写[15],也就是说,一旦 NVM 的某一存储单元的写次数超过了其能承受的上限就会被磨损.NVM 的这两个特性使设计者需要重新考虑现有算法、数据结构等设计是否适用于这种新的架构[16-17].

2.2 连接顺序问题

多表连接在数据库系统中是极为常见的操作.不同的连接顺序会引起不同的开销,因此如何优化多张表的连接顺序至关重要.

当表的数量较小时,可以通过穷举的方式得到一个最优的连接顺序.然而,当表的数量超过一定的大小时,该问题则不能有效地被解决,因为它是一个 NP-hard 问题[18].本文需要依靠算法,在某一搜索空间中找到一个较优的解从而确定连接的顺序.搜索空间是不同连接树的集合,连接树可以描述不同的连接方案但最终都会得到同样的连接结果.为了更容易理解,本文首先对相关的概念进行说明:输入的连接条件中涉及到的表称为基本表,表与表连接产生的中间结果表称为中间表;连接树是一颗二叉树,在该树中,基本表永远是叶子节点,中间表是内部节点,边表示数据的流向;搜索空间可以分为两种,分别是左深树搜索空间和浓密树搜索空间.对于左深树,其内部节点至少有一个叶子节点作为子节点,也就是说,在多表连接的过程中,每一步连接都至少有一个基本表参与连接.图 1(a)为一个左深树

计算机学报

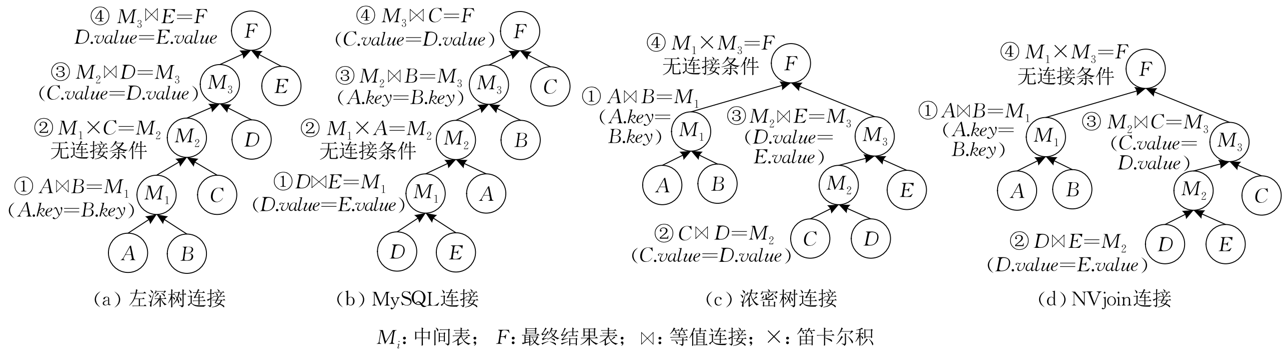


图 1 4 种不同的连接树

的例子,其中 M_i 表示中间表,F 表示最终的连接结果.相反,在浓密树中,有些内部节点是没有叶子节点作为子节点的.图 1(c)为浓密树的一个例子.实际上,左深树是浓密树的一个子集.通常,当表的数量为 n 时,左深树和浓密树搜索空间中连接树的个数分别为 $n!$ 和 $C_{2(n-1)}^{(n-1)} \times (n-1)!$[19].

3 相关工作

目前,已有很多学者在基于新型存储介质的数据库系统方面做了相关的研究.Kuan 等人[11]针对基于 PCM 的嵌入式数据库提出了空间有效的索引方案;Wang 等人[1]为基于闪存的嵌入式数据库设计了新型的多版本 B+tree 索引结构;Arulraj 等人[14]实现了数据库系统中现有的三种存储引擎架构,并针对 NVM 分别对它们进行了优化;Wang 等人[20]针对 SCM 提出了 key-value 存储系统;Chen 等人[21]提出了基于 NVM 的 Key-Vaule 数据库框架,并将 Redis 数据库应用在该框架下进行了性能的测试.但目前还没有基于 NVM 的数据库连接顺序优化的研究.

针对多表连接的研究由来已久.文献[3]在左深树搜索空间中利用动态规划方法优化连接顺序,但该方法只适用于连接表的数量较小的情况.文献[7]提出 KBZ 算法,分两个阶段寻找时间开销最小的连接序列,从而得到最终的优化结果.文献[5]为环型查询图提出了基于遗传算法的优化器.文献[18]为大规模(大于 20 张表)的查询优化提出了新的范式,先简化查询图,再用动态规划算法优化连接顺序,尽量减小连接表的基数(表中不同值的个数).文献[9]提出 SAM 算法,该方法可以减少优化过程中的时间复杂度.然而,这些算法没有充分考虑 NVM 可字节寻址和受限的写容忍度特性,并不适用于基于 NVM 的内存数据库.也有一些研究针对基于闪存的固态硬盘(SSD)对 join 算法(两个表的连接)做优化[22-23],但都是以 SSD 替代硬盘,使得系统的整体架构为 DRAM+SSD,并着重研究两个表连接过程中的优化问题,其目标是减少连接过程中的 I/O 操作.而本文是以 NVM 作主存的存储架构,着重研究如何充分利用 NVM 的优势优化多表连接的顺序问题,目标是通过减少中间表的字节数来减少 NVM 的写次数,并得到较优的性能.

减少对 NVM 写操作的研究可以分为硬件与软件两个层面.在电路和控制器等硬件层面,文献[24]提出一种基于 PCM 作为主存的电路级写优化技术,用于减少冗余的比特位的写操作,提升非易失性存储器的写性能;文献[25]提出针对 PCM 存储电路和内存控制器的优化技术,包括冗余位免写技术、行位移技术以及段交换技术,以减少对 NVM 的写操作,提升 NVM 的写性能.对于上层应用而言,减少 NVM 的写次数虽然不能提升 NVM 的写性能,但它对提升应用的整体性能有着重要的影响.例如,文献[26]提出基于 NVM 的索引结构,减少维护索引一致性的代价,提升索引的整体性能.文献[20]提出写友好的解决哈希冲突的机制,提升索引的性能.本文从上层应用的角度出发,研究数据库中多表连接操作的写优化技术.而对于上述底层技术而言,仍可以应用在本文提出的算法下,进一步提高应用的整体效率.

4 研究动机

在执行多表连接查询时,不同的连接顺序严重影响中间结果的大小.由于这些结果都存储在 NVM 介质上,因此,如何减小中间结果的大小显得尤为重要.在这一部分中,本文将通过一个例子进一步

步说明这一问题.

随机生成五张表：A，B，C，D 和 E. 图 2 展示了每张表的大小以及表中部分属性的信息. 假设查询语句为："create table R select * from A，B，C，D，E where $A.key=B.key$ and $C.value=D.value$ and $D.value=E.value$". 在展示不同的连接树之前，本文先介绍一些必要的符号. '$\bowtie$'和'$\times$'分别表示等值连接和笛卡尔积操作. M_i 表示在第 i 次连接过程中得到的中间结果，F 表示最终的连接结果. 图 1 分别展示了 4 个不同连接顺序的连接树，下面本文会逐一进行解释.

表A(10B/元组)

key	…
a_1	…
a_2	…
a_3	…
a_4	…

表B(30B/元组)

key	…
a_1	…
a_2	…
a_3	…
a_4	…

表C(150B/元组)

…	value	…
…	11	…
…	11	…
…	33	…
…	5	…
…	60	…
…	97	…
…	30	…

表D(100B/元组)

value	…
11	…
68	…
90	…

表E(200B/元组)

value	…
11	…
33	…
100	…

图 2　表 A，B，C，D，E 的示意图

图 1(a)为左深树的连接方式，它是完全按照用户输入的查询顺序连接基本表的. 表 1 展示了左深树的连接过程以及对应中间表的大小. 可以看出由于左深树的连接规则是将上一步的连接结果与一张新的表连接，所以即使表 M_1 与 C 没有任何的关联性(因为没有需要连接的共有属性)，也必须先对这两张表做笛卡尔积. 由于笛卡尔积操作需要将两个表中的所有数据进行连接，因此得到的中间表 M_2 的是两个表中数据大小的乘积. 此外，中间表 M_2 仍会参与到后续的连接操作中，因此较早的执行笛卡尔积使得过大的中间结果一直参与到连接过程中直至结束，从而引起较多的 NVM 写操作，在本例中左深树的连接方式产生的所有中间表的大小为 7800 字节(不考虑最终结果的大小).

表 1　左深树连接过程表

连接表	结果大小
$A\bowtie B=M_1$	4×(10+30)=160
$M_1\times C=M_2$	4×7×(40+150)=5320
$M_2\bowtie D=M_3$	8×(190+100)=2320
$M_3\bowtie E=F$	8×(290+200)=3920

图 1(b)展示了 MySQL 的连接方式，它同样采用左深树的结构进行连接，不同的是表的连接顺序有所改变. 该顺序是由 MySQL 的执行计划(数据库会对用户输入的查询语句做解析和优化，得到相应的执行计划)确定的. MySQL 根据基本表中涉及到的行数等因素通过贪心算法逐步将待连接的表以左深树的形式连接起来，从而得到一个较优的连接顺序. 表 2 展示了 MySQL 的连接过程，产生的所有中间表的大小为 2900 字节. 可以发现，通过 MySQL 优化后的多表连接顺序产生的中间表大小比没有优化的左深树连接减少了近 3 倍. 但是 MySQL 的连接顺序并没有充分考虑表与表之间的关联性(在连接的第二步，表 M_1 先与表 C 连接更为合理)，所以在执行计划的第二步就出现了笛卡尔积操作.

表 2　MySQL 连接过程表

连接表	结果大小
$D\bowtie E=M_1$	1×(100+200)=300
$M_1\times A=M_2$	4×(300+10)=1240
$M_2\bowtie B=M_3$	4×(310+30)=1360
$M_3\bowtie C=F$	8×(340+150)=3920

图 1(c)展示了浓密树的连接方式，顺序执行用户输入查询条件的连接树. 与左深树的连接方式不同，浓密树无需在上一次连接结果的基础上执行连接，因此可以避免不必要的笛卡尔积操作. 例如在本例中，当执行第二个连接条件"$C.value=D.value$"时，并不需要像左深树一样将 M_1 与 C 用笛卡尔积进行连接，而是直接利用连接条件把 C 和 D 连接起来，并且将笛卡儿积推迟到最后一步执行. 表 3 展示了浓密树的连接过程，这种连接方式所产生的中间表的大小为 1560 字节. 正因为避免了不必要的笛卡尔积操作，该方法相比 MySQL 连接方式进一步减少了中间结果的大小.

表 3　浓密树连接过程表

连接表	结果大小
$A\bowtie B=M_1$	4×(10+30)=160
$C\bowtie D=M_2$	2×(150+100)=500
$M_2\bowtie E=M_3$	2×(250+200)=900
$M_1\bowtie M_3=F$	8×(40+450)=3920

图 1(d)为根据本文所提出的 NVjoin 算法构建的浓密树，该算法根据表的信息确定了一个较为合理的连接顺序，如表 4 所示. 可以注意到，该连接方式中并没有不必要的笛卡尔积，并且将"$D\bowtie E$"操作先于"$C\bowtie D$"操作执行. 该连接顺序使得最终所有中间结果的大小只有 1360 字节. 相比其它 3 种连接方式，这是一个较优的连接方案，且随着表中数据量的增加，算法的优越性会更加明显.

《计算机学报》

表 4 NVjoin 连接过程表

连接表	结果大小
$A \bowtie B = M_1$	4×(10+30)=160
$D \bowtie E = M_2$	1×(200+100)=300
$M_2 \bowtie C = M_3$	2×(300+150)=900
$M_1 \bowtie M_3 = F$	8×(40+450)=3920

此外，传统的存储中间结果的方案是把相关的数据拷贝到中间表中. 这种方式对基于 NVM 的内存数据库而言会因大量冗余数据的拷贝而产生过多的 NVM 写次数. 若针对图 1(d)的连接树，充分利用 NVM 可字节寻址的特性，改变存储中间结果的方式，即中间表中存储数据的地址而非数据本身，如表 5 所示，这种方式产生的中间表的大小仅仅为 128 字节.

表 5 改变中间表存储结构的 NVjoin 连接过程表

连接表	结果大小
$A \bowtie B = M_1$	4×(8+8)=64
$D \bowtie E = M_2$	1×(8+8)=16
$M_2 \bowtie C = M_3$	2×(16+8)=48
$M_1 \bowtie M_3 = F$	8×(16+24)=320

综上所述，与图 1(a)、图 1(b)和图 1(c)的连接方式相比，我们通过优化连接顺序以及改变存储方式将连接过程中产生的中间结果大小分别减少了 98.4%、95.6%和 91.8%，这也意味着本文提出的算法和结构能够极大地减少 NVM 的写次数.

5 设计与实现

本文主要的设计与优化目标是减少 NVM 的写次数. 鉴于此，本文从两个方面进行考虑. 一方面，连接顺序严重影响中间结果的大小. 为减少写中间表产生的 NVM 写次数，本文提出 NVJoin 算法，可以有效地优化连接顺序. 另一方面，连接两个表的过程中并不需要把所有的数据都拷贝到中间表中. 为此本文提出一个轻量级的“NVM 友好”的结构，LWTab. 该结构充分利用 NVM 可字节寻址的特性，以此减少 NVM 的写次数. 下面本文将分别介绍 NVjoin 和 LWTab.

5.1 多表连接算法：NVjoin

NVM 受到写容忍度的限制，一个最大的挑战就是如何减少 NVM 的写次数. 多数的连接顺序优化算法没有考虑 NVM 寿命这一因素，且没有利用 NVM 可字节寻址的特性，因此并不适用于基于 NVM 的内存数据库. 本文提出的算法充分考虑了表之间的关联性，尽量避免不必要的笛卡尔积操作，利用浓密树的方式构建连接树，尽可能减小中间表的字节数，即减少 NVM 的写次数. NVjoin 算法主要包含 3 个阶段：(1) 分类；(2) 估算和连接；(3) 笛卡尔积. 详细步骤见算法 1. 算法中用 T_i 表示任意的表，它可以是基本表或者是中间表.

算法 1. NVjoin 算法.

输入：JP“连接条件的集合”

输出：PT“连接树”

1. 将 JP 根据表之间的关联性分为不同的类 DC_i
2. FOR 每一个类 DC_i DO
3. WHILE DC_i 不为空 DO
4. FOR DC_i 中的每一个表 T_i DO
5. 以比率 P_i 对表 T_i 抽样得到抽样表 S_i
6. FOR 每一个连接条件 DO
7. 用 T_i 和 T_j 的抽样表 S_i 和 S_j 估算连接大小 ES_{ij}
8. 选择 ES_{ij} 中估算值最小的连接
9. 执行 $T_i \bowtie T_j$，创建中间表 T_k，并将该过程添加到 PT 中
10. 在 DC_i 中删除 $T_i \bowtie T_j$ 并用 T_k 替代 DC_i 中的 T_i 和 T_j，T_k 仍旧会被当成待连接的 T_i 或 T_j 继续参与循环
11. 对每个类 DC_i 的结果做笛卡尔积
12. 返回 PT

(1) 分类(算法 1 中第 1 行). 多表连接查询可以看作是一个连接图 $G=(V,E)$，其中 V 是基本表的集合，边 $(T_i, T_j) \in E$，表示表 T_i 和 T_j 之间有连接条件. 由于一张连接图可能有多个连通分支，我们把输入的连接条件根据表与表之间的关联性分成不同的类(用 DC_i 表示). 也就是说，每个连通分支可以看作是一个单独的类，类中包含了该连通分支中相应的连接条件. 例如，查询命令为“select * from A, B, C, D, E, F where $A.1=B.1$ and $B.2=C.2$ and $A.3=D.3$ and $D.1=C.3$ and $E.1=F.1$”. 该查询语句的连接图如图 3 所示，其中包含两个连通分支. NVjoin 算法先根据表之间的关联性对这 6 个连接条件进行分类. 第一个类是 $DC_1=\{A \bowtie B, B \bowtie C, C \bowtie D, D \bowtie A, A \bowtie C\}$，第二类是 $DC_2=\{E \bowtie F\}$.

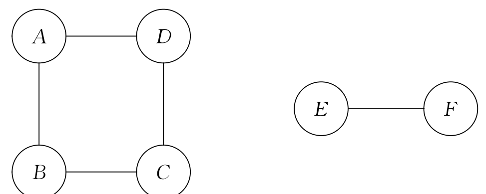


图 3 查询语句的连接图

(2) 估算和连接(算法 1 中 2～10 行). 对于每一个类，算法迭代执行下述①、②步骤，直到该类为空，即所有的连接条件都已经被执行.

①估算. 对于每一个连接条件 $T_i \bowtie T_j$，算法采用抽样技术估算其连接结果的大小. 首先，算法对连接条件涉及到的每张表进行抽样，得到其抽样表 S_i 和 S_j. 在这里，我们以 P_i 的比率对表 T_i 进行抽样得到 S_i. 例如，对一张有 50 个元组的 T_1 表以 10% 的比率进行抽样，抽样的方式为均匀抽取，即从第 0 个元组开始，每隔 50×10%＝5 个元组抽取一个，则得到的抽样表 S_1 如图 4(a) 所示，该表中包含 5 个元组. 同理，对 T_2 进行抽样，得到抽样结果 S_2 如图 4(b) 所示. 然后，算法计算两个抽样表 S_i 和 S_j 连接结果的大小，即按照两个表等值连接的方式计算连接结果中元组的个数再乘以元组的字节数(域大小)，表示为 SS_{ij}. 那么 $T_i \bowtie T_j$ 的估算大小 ES_{ij} 可以表示为 $ES_{ij}=SS_{ij}/(P_iP_j)$. 例如，在上述例子中，S_1 和 S_2 连接的结果共有 4 个元组，如图 4(c) 所示. 假设表 T_1 和 T_2 的元组字节数分别为 30 和 25 字节，那么，可以求得 $SS_{12}=4\times(30+25)=220$ 字节，$ES_{12}=SS_{12}/(P_1P_2)=220/(0.1\times0.1)=22\,000$ 字节. 需要说明的是，图 4(c) 只是为了说明如何计算 S_1 和 S_2 连接结果的大小，在算法中并不需要真正将这两个表做连接，只需做相应的计算.

1	2
a_1	b_1
a_2	b_2
a_1	b_3
a_2	b_3
a_3	b_4

(a) 表 T_1 的抽样表 S_1

1	2
a_1	c_1
a_2	c_2
a_4	c_3

(b) 表 T_2 的抽样表 S_2

1	2	3	4
a_1	b_1	a_1	c_1
a_2	b_2	a_2	c_2
a_1	b_3	a_1	c_1
a_2	b_3	a_2	c_2

(c) 表 S_1 与 S_2 连接结果

图 4　抽样举例

② 连接. 根据估算出的每个连接条件预计产生的中间表大小，算法首先选择估算结果最小的连接条件，假设为 $T_i \bowtie T_j$. 然后执行相应的连接操作，并在该连接过程中创建一个新的中间表 T_k. 最后算法更新 DC_i 中存储的连接条件. 该更新过程包括两方面. 首先移除已经被执行过的连接条件 $T_i \bowtie T_j$. 其次，如果 DC_i 中剩余的连接条件有涉及到表 T_i 或 T_j 的，则用 T_k 来替代. 该表作为待连接的表(即可以被当作 T_i 或 T_j) 参与下一轮的抽样、估算与连接. 在本例中，假设当前的估算结果中"$A \bowtie B$"的连接结果最小，那么算法就执行"$A \bowtie B$"并得到中间表 M. DC_1 也会被更新为 $DC_1=\{M\bowtie C, C\bowtie D, D\bowtie M, M\bowtie C\}$.

(3) 笛卡尔积(算法 1 中第 11 行). 如果连接图有多个连通分支，即连接条件被分为了多个类，需要对各个类得到的连接结果执行笛卡尔积从而得到最终的结果. 本例中，需要对 DC_1 和 DC_2 得到的结果执行笛卡尔积，进而得到最终查询的结果.

NVjoin 算法考虑表之间的关联性，避免不必要的笛卡尔积操作，提升多表连接操作的整体性能，在 DRAM、SDD 等非 NVM 介质上同样适用. 此外，NVjoin 算法通过减小中间表的大小，极大地减少 NVM 的写次数. 然而，在存放中间结果的过程中需要拷贝很多重复的数据，这仍会引起不必要的 NVM 写操作. 因此，本文需要一个全新的存储结构减小这方面的开销.

5.2　轻量级中间表结构：LWTab

这一部分中，本文提出一个可以存放中间结果，并且是"NVM 友好"的数据结构. 由于 NVM 与 DRAM 一样具有可字节寻址的特性，内存总线可以直接访问表中的每个元组，因此我们并不需要像传统的方式一样把所有的数据都拷贝到中间表中，只需把元组的内存地址保存在新的结构中即可，本文将这一轻量级结构命名为 LWTab. 通过这种方式，可以进一步减少 NVM 的写次数.

(1) LWTab 结构. 我们用 BT_i 表示一个基本表. 如图 5 所示，LWTab 作为中间表，存储基本表中符合连接条件的元组地址. 每一列中的地址是同一个基本表中不同元组的地址，而每一行的地址表示不同表的元组中满足连接条件的元组地址.

BT_i 中元组地址	…	BT_j 中元组地址
0X004F1540	…	0X009B4240
…		…
0X008A1640	…	0X009B425C

图 5　LWTab 结构示意图

(2) LWTab 举例. 如图 6(a) 所示，两个待连接的基本表 A 和 B，连接条件为"$A.3=B.4$". 可以看出，表 A 和表 B 中每一个元组都有自己的首地址，假设表 A 一个元组大小为 100 字节，表 B 的一个元组大小为 200 字节. 图 6(b) 展示了利用传统的存储中间结果的方式构建的表 T_c，该方案把所有满足连接条件的数据都拷贝到 T_c 中. 由图可知，共有 3 个元组，每个元组的大小为 300 字节，表 T_c 的总大小为 900 字节. 值得注意的是，当元组的字节变得更大的时候表 T_c 也会随之增大. 相比之下，如图 6(c) 所示，新结构 LWTab 构建的中间表 T_p 只需要存储对

《计算机学报》

应的元组首地址即可. 考虑在 64 位系统中，一个地址的大小为 8 字节. 则 T_p只有 48 字节. 这一例子充分说明了 LWTab 技术的有效性.

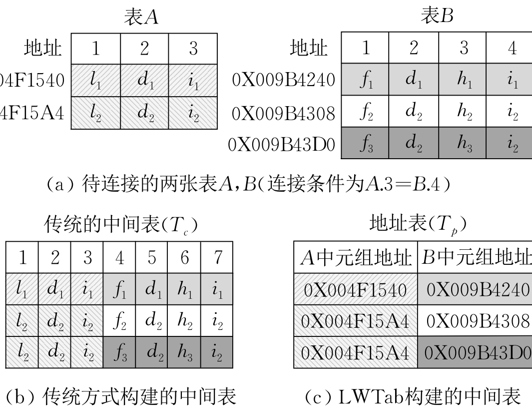


图 6 传统存储方式与 LWTab 比较

与传统的构建中间表的方式相比，LWTab 结构有以下两个优势：① LWTab 结构通过记录元组的首地址而非拷贝元组数据的方式，减少 NVM 的写次数，尤其当元组较大时，效果尤为明显；② 在访问中间表中的数据时，LWTab 需要根据地址寻找对应的数据. 这虽然增加了一步访存的操作，但却极大地减少拷贝数据的操作. 这对于读写延迟极不对称的 NVM 而言，LWTab 能够通过减少 NVM 写操作所带来的性能提升弥补额外的访存的开销，甚至获得整体性能的提升. 综上所述，无论是在减少 NVM 写次数方面还是提升整体的性能方面，LWTab 结构都能起到一定的作用. 本文将 LWTab 和 NVjoin 这两个技术进行结合，并把它命名为 NVjoin+LWTab.

6 实 验

本文将 NVjoin 和 NVjoin+LWTab 与其它三种连接方法进行比较：(1) 左深树连接. 通过左深树的方式构建连接树，连接的顺序取决于用户输入；(2) 浓密树连接，通过浓密树的方式构建连接树，连接的顺序同样取决于用户输入；(3) MySQL 连接. 该连接方式是由 MySQL 的执行计划确定的. MySQL 在左深树搜索空间中利用贪心算法优化连接顺序，进而得到一棵左深连接树.

本文模拟实现这 5 种算法，并从 NVM 写次数和整体运行时间两个方面来进行评估. 本文提出的算法中由于抽样表同样需要存储在 NVM 上，所以在统计 NVM 写次数的时候也需要把抽样表大小加进去. 因此，为了达到减少 NVM 写次数的目的，抽样表的大小不能过大，然而，过小的抽样比率可能影响估算中间表大小的精确度. 因此，本文针对不同的抽样比率做了大量实验(见 6.1 节)，实验结果显示当抽样比率设定为 10% 时即可达到一个较为准确的估算精度，同时又能达到减少 NVM 写次数的效果. 因此，实验将抽样比率均设为 10%. 对于每一种多表连接方法，本文都采用相同的连接算法(嵌套循环算法)对两个表进行连接. 对于 MySQL 连接，本文利用 MySQL 较新的版本(ver. 5.7.17)获得执行计划并在模拟器中模拟该执行方案. 实验根据文献[27]提出的方法生成测试的数据集，该方法早已在查询优化算法中[27-29]被应用为基准测试集. 此外，不失一般性，本文分别生成具有不同连通分支个数的查询语句. 具体数据集的分布如表 6 所示.

表 6 实验工作集

工作集	表的数量	连通分支数
A	8	2
B	9	2
C	10	2
D	8	3
E	6	1
F	5	1
G	4	1
H	6	1

本文的实验环境与文献[30]类似，利用 DRAM 模拟 NVM. 实验平台的配置有 32GB DRAM、3.2GHz 的 Intel i5-4570 四核处理器. 实验模拟 NVM 非对称的读写特性. 文献[31]指出 NVM 的读延迟与 DRAM 相近，约为 60 ns；文献[26]指出 NVM 的写延迟比 DRAM 高 100 ns；因此本文将 DRAM 的读写延迟分别设置为 60 ns 和 160 ns.

在展示实验结果之前，本文先定义一些必要的符号. 对于 NVM 写次数，假设 N_a 和 N_b 分别表示不同的连接方法产生的 NVM 写次数，计算 $\frac{N_a}{N_b}=\omega$，并用“ωX”表示 N_b 相比 N_a 所取得的 NVM 写次数的减少程度. 比如，MySQL 连接和 NVjoin 所产生的 NVM 写次数分别是 1.93×10^7 和 3.49×10^5，那么 NVjoin 相比 MySQL 连接可减少 $\frac{1.93\times10^7}{3.49\times10^5}=55.3$X 的 NVM 写次数. 类似地，令 T_a 和 T_b 分别表示不同连接方法的执行时间，计算 $\frac{T_a-T_b}{T_a}=u\%$，并用“$\mu\%$”表示 T_b 所取得的加速比. 比如，MySQL 连接和 NVjoin 的执行时间分别是 11.4338 s 和 1.6511 s，则 NVjoin 相比 MySQL 连接可以得到

$\frac{11.4338-1.6511}{11.4338}=85.56\%$的加速比. 接下来，我们分别对 NVM 写次数和执行时间的实验结果进行说明.

6.1 抽样比率实验结果

抽样比率的大小不仅会影响 NVM 的写次数，而且会影响估算中间表大小的精确度. 因此，为了获得较优的抽样比率，我们利用表 6 中的工作集，分别测试了抽样比率为 10%、30%、50% 和 100% 时，NVjoin＋LWTab 对每种数据集产生的 NVM 写操作(中间表和抽样表引起的 NVM 写之和)的实验结果(见图 7). 从实验结果中可以发现，在大部分工作集中(除工作集 E 外)，抽样比率设为 10% 时，NVjoin＋LWTab 对 NVM 的写次数最少. 主要原因可以总结为以下两点：(1) 随着抽样比率的增加，抽样表中的数据不断增多，对 NVM 的写操作次数也随之增加. 因此，在实验中，抽样比率为 10% 时，其抽样表引起的写次数最少；(2) 抽样比率对估算中间表的大小有着重要影响，进而会影响表的连接顺序. 从实验中可以发现，当抽样比率设定为 10% 或 30% 时，可得到较好的多表连接的顺序，从而获得较少的写次数. 综合上述因素，本文认为将抽样比率设定为 10%，在 NVM 写方面均能取得较好的结果.

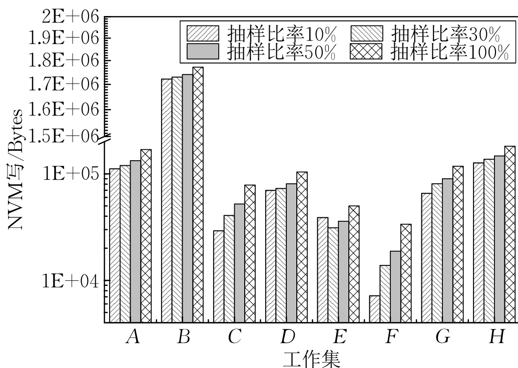


图 7 NVjoin＋LWTab 在不同的抽样比率下引起的 NVM 写

6.2 NVM 写次数实验结果

多表连接操作过程中产生的 NVM 写次数主要为存储中间结果所导致的 NVM 写. 图 8 展示了 NVM 写次数的实验结果. 可以看出，不同的连接顺序严重影响中间表的大小，对于工作集 A, B, C 和 D，由于连接图中都有多个连通分支，所以在连接过程中笛卡尔积是不可避免的. 那么只能尽量延迟笛卡尔积操作. 对于这 4 个数据集，我们可以总结出两点：(1) 选用左深树方式的左深树连接和 MySQL 连接产生的 NVM 写次数比其它方法高很多. 与左深树相比，浓密树可以尽可能地将迟笛卡尔积推迟到最后一步执行. 通常情况下，前面连接步骤产生的中间结果的大小会影响后面的中间结果，较早地执行笛卡尔积，所产生的较大的中间结果很可能一直维持到最后. 所以，推迟笛卡尔积的执行可以有效减少 NVM 写次数；(2) 左深树连接产生的 NVM 写次数比 MySQL 连接多. 在同样是利用左深树的方式构建连接树的情况下，MySQL 利用贪心算法对连接顺序做了优化，所以其产生的中间表大小要小很多. 类似地，与浓密树连接相比，NVjoin 可以减少很多 NVM 写次数.

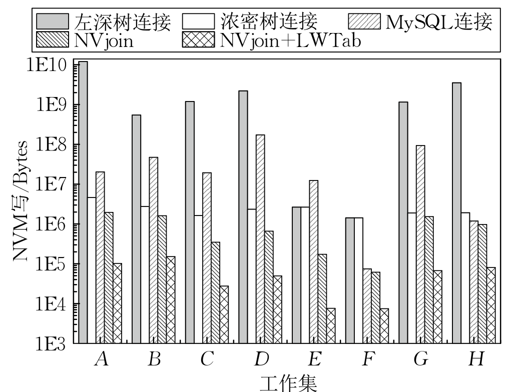


图 8 不同连接方法在不同工作集下产生的 NVM 写

对于工作集 E, F, G 和 H，它们的连接图都是连通图. 也就是说，在连接过程中应该是没有笛卡尔积的. 然而，左深树连接和 MySQL 连接仍然存在笛卡尔积. 主要原因是，左深树连接的顺序完全取决于用户输入，而 MySQL 虽然对连接顺序有一定的优化，但是并没有考虑表与表之间的关联性，所以同样会产生笛卡尔积. 对于浓密树连接来说，虽然可以避免笛卡尔积，但其连接顺序仍然取决于用户输入，同样会产生过多的 NVM 写次数. 这些原因使得上述三种连接方法在 NVM 写次数上表现得很不稳定.

相反，NVjoin 和 NVjoin＋LWTab 在不同的工作集下所产生的 NVM 写次数均少于其它三种连接方法. 主要原因可以总结为以下 4 点：(1) NVjoin 算法充分考虑了表之间的关联性，从而避免了不必要的笛卡尔积；(2) 算法选用浓密树来构建连接树，从而减轻了笛卡尔积带来的影响；(3) 算法中的估算过程可以帮助我们得到较优的连接顺序，有效减小连接过程中产生的中间表的总大小；(4) LWTab 避免了冗余的数据拷贝，极大地减少了 NVM 写次数.

表 7 总结了 NVjoin＋LWTab 产生的 NVM 写次数相比其它连接方法减少的倍数. 可以看出，左深树连接、浓密树连接、MySQL 连接和 NVjoin 所

《计算机学报》

产生的 NVM 写次数平均是 NVjoin＋LWTab 的 59171.06倍、183.65 倍、1744.63 倍和 16.74 倍.

表 7　NVjoin＋LWTab 相比其它方法减少 NVM 写次数总结

	最小	最大	平均
左深树连接	190.88X	118151.25X	**59171.06X**
浓密树连接	18.22X	349.08X	**183.65X**
MySQL 连接	10.04X	3479.23X	**1744.63X**
NVjoin	10.65X	22.83X	**16.74X**

6.3　运行时间实验结果

图 9 展示了对于不同的数据集每一种方法整体的执行时间(其中包括了 NVjoin 和 NVjoin＋LWTab 的优化顺序时间). 由于笛卡尔积不仅会导致产生大量的 NVM 写次数,同样会产生较高的计算开销. 所以,当连接图中有多个连通分支的时候,左深树连接和 MySQL 连接的运行时间会比其它三种连接方法高很多. 在剩下的三种连接方法中,NVjoin 和 NVjon＋LWTab 比浓密树连接的运行时间低很多. 主要是因为一个较优的连接顺序可以极大地降低连接过程的执行时间. 从整体上来看,NVjoin 通过优化连接顺序降低运行时间,而 NVjoin＋LWTab 节省了拷贝数据的时间,进一步减少了整体的运行时间.

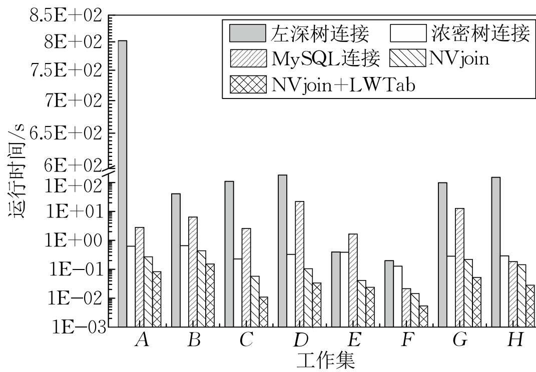


图 9　不同连接方法在不同工作集下的运行时间

表 8 总结了 NVjoin＋LWTab 相比其它方法在运行时间上所取得的提升. NVjoin＋LWTab 的运行时间相比左深树连接,浓密树连接,MySQL 连接和 NVjoin 的运行时间平均提升了 96.97%、88.48%、87.6%和 71.86%.

表 8　NVjoin＋LWTab 相比其它方法运行时间加速比总结

	最小/%	最大/%	平均/%
左深树连接	93.97	99.98	**96.97**
浓密树连接	81.74	95.21	**88.48**
MySQL 连接	74.67	99.84	**87.26**
NVjoin	62.78	80.93	**71.86**

6.4　LWTab 访问数据开销实验结果

LWTab 通过存储数据的地址,避免对数据的拷贝操作,能够极大地减少 NVM 的写次数. 但是这种组织数据的方式使得在访问中间表时需要根据地址寻找对应的数据,增加一次额外的访存开销. 针对 LWTab 结构实验分别统计了每个工作集中通过指针访问数据的开销,并与整体的执行时间进行比较,实验结果如表 9 所示,结果表明 LWTab 造成的开销极低,可以忽略不计.

表 9　LWTab 访问指针开销与整体开销的占比

	整体执行时间	访问指针开销	占比/%
A	0.69	0.0001	0.15
B	0.64	0.006	3.92
C	0.81	0.00003	0.27
D	0.68	0.0001	0.42
E	0.42	0.00006	0.27
F	0.63	0.00002	0.35
G	0.76	0.00065	1.20
H	0.81	0.00028	0.99

根据表 9 中的数据可知,对于工作集 A、C、D、E、F、H 而言,访问指针开销的占比不超过整体执行时间的 1%. 而对于工作集 B 和 G 而言,访问开销的占比不超过 5%. 这是由于工作集 B 和 G 在连接过程中待访问的数据要远多于其它的工作集,因此访问指针的次数也会相应增多. 但从总的占比而言,访问指针的开销是微乎其微的,主要有以下两个原因:

(1) NVjoin 算法通过优化连接顺序尽可能地减少中间表的大小以及执行笛卡尔积的次数,因此,访问中间表中的指针次数较少.

(2) 随着硬件技术的不断发展,访存的速度越来越快,因此根据地址访问数据并不会引起很大的开销.

综上所述,LWTab 通过存储数据地址的方式虽然会引起额外的访存开销,但与整体的执行时间相比此开销只占据很小的部分. 此外,这种方式节省了大量拷贝重复数据的开销,所以,LWTab 在减少 NVM 写次数的同时提升了多表连接的性能.

7　结束语

针对基于 NVM 的内存数据库,本文为连接顺序问题提出一个"NVM 友好"的优化方案(NVjoin)和一个新的轻量级结构(LWTab). NVjoin 算法充分考虑了表之间的关联性,避免了不必要的笛卡尔积,并通过估算的方式尽量减少中间表的总大小. 此

《计算机学报》

外，LWTab 通过利用 NVM 可字节寻址的特性避免了冗余数据的拷贝.实验结果表明，与其它连接方法相比，NVjoin 可以减少大量 NVM 写次数.通过将 LWTab 技术与 NVjoin 进行结合，可以在保证性能的同时更近一步减少 NVM 的写次数.

由于大部分数据库都是利用左深树的方式构建连接树的，其操作方式与浓密树完全不一样.本文是通过模拟的方式来进行实验.在未来的工作中，我们会将 NVjoin＋LWTab 嵌入到 MySQL 中，进一步验证算法的有效性.

参　考　文　献

[1] Wang J, Lam K Y, Chang Y H, et al. Block-based multiversion B+-tree for flash-based embedded database systems. IEEE Transactions on Computers, 2015, 64(4): 925-940

[2] Lee B C, Ipek E, Mutlu O, et al. Architecting phase change memory as a scalable dram alternative//Proceedings of the International Symposium on Computer Architecture. Austin, USA, 2009: 2-13

[3] Selinger P G, Astrahan M M, Chamberlin D D, et al. Access path selection in a relational database management system//Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data. Boston, USA, 1979: 23-34

[4] Goldberg D E. Genetic algorithms in search, optimization and machine learning. Ethnographic Praxis in Industry Conference Proceedings, 1989, xiii(7): 2104-2116

[5] Muntésmulero V, Aguilarsaborit J, Zuzarte C, et al. CGO: A sound genetic optimizer for cyclic query graphs. Lecture Notes in Computer Science, 2006, 3991: 156-163

[6] Li N, Liu Y, Dong Y, et al. Application of ant colony optimization algorithm to multi-join query optimization. Advances in Computation and Intelligence, Third International Symposium, 2008, 5370: 189-197

[7] Krishnamurthy R, Boral H, Zaniolo C. Optimization of nonrecursive queries//Proceedings of the VLDB'86 Twelfth International Conference on Very Large Data Bases. Kyoto, Japan, 1986: 128-137

[8] Wong E, Youssefi K. Decomposition—A strategy for query processing. ACM Transactions on Database Systems, 1976, 1(3): 223-241

[9] Zeng Y, Lu A N, Xia L, et al. SAM: A sorting approach for optimizing multijoin queries//Proceedings of the Database and Expert Systems Applications. Valencia, Spain, 2015: 367-383

[10] Ioannidis Y E, Kang Y C. Left-deep vs. bushy trees: An analysis of strategy spaces and its implications for query optimization//Proceedings of the ACM SIGMOD International Conference on Management of Data. Denver, USA, 1991: 168-177

[11] Kuan Y H, Chang Y H, Huang P C, et al. Space-efficient multiversion index scheme for PCM-based embedded database systems//Proceedings of the Design Automation Conference. San Francisco, USA, 2014: 1-6

[12] Awad A, Blagodurov S, Solihin Y. Write-aware management of NVM-based memory extensions//Proceedings of the International Conference on Supercomputing. Istanbul, Turkey, 2016: 9

[13] Pathak S. Status of Phase Change Memory in Memory Hierarchy and Its Impact on Relational Database [Ph.D. dissertation]. National University of Singapore, Singapore, 2011

[14] Arulraj J, Pavlo A, Dulloor S R. Let's talk about storage & recovery methods for non-volatile memory database systems //Proceedings of the ACM SIGMOD International Conference on Management of Data. Melbourne, Australia, 2015: 707-722

[15] Mittal S, Vetter J. A survey of software techniques for using non-volatile memories for storage and main memory systems. IEEE Transactions on Parallel & Distributed Systems, 2016, 27(5): 1537-1550

[16] Chang J, Ranganathan P, Mudge T, et al. A limits study of benefits from nanostore-based future data-centric system architectures//Proceedings of the 9th Conference on Computing Frontiers. Cagliari, Italy, 2012: 33-42

[17] DeBrabant J, Arulraj J, Pavlo A, et al. A prolegomenon on OLTP database systems for non-volatile memory. Proceedings of the VLDB Endowment, 2014, 7(14): 57-63

[18] Neumann T. Query simplification: Graceful degradation for join-order optimization//Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data. Rhode Island, USA, 2009: 403-414

[19] Zhou Z. Using heuristics and genetic algorithms for large-scale database query optimization. Journal of Computing & Information Science in Engineering, 2007, 2: 261-280

[20] Wang Z, Huang L, Zhu Y. SCMKV: A lightweight log-structured key-value store on SCM//Proceedings of the IFIP International Conference on Network and Parallel Computing. Hefei, China, 2017: 1-12

[21] Chen X, Sha H M, Abdullah A, et al. UDORN: A design framework of persistent in-memory key-value database for NVM//Proceedings of the Non-Volatile Memory Systems and Applications Symposium. Hsinchu, China, 2017: 1-6

[22] Li Y, On S T, Xu J, et al. Optimizing nonindexed join processing in flash storage-based systems. IEEE Transactions on Computers, 2013, 62(7): 1417-1431

[23] Tsirogiannis D, Harizopoulos S, Shah M A, et al. Query processing techniques for solid state drives//Proceedings of the ACM SIGMOD International Conference on Management of Data. Rhode Island, USA, 2009: 59-72

[24] Cho S, Lee H. Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance//Proceedings of the IEEE/ACM International Symposium on Microarchitecture. Washington, USA, 2010: 347-357

《计算机学报》

[25] Zhou P, Zhao B, Yang J, et al. A durable and energy efficient main memory using phase change memory technology. ACM SIGARCH Computer Architecture News, 2009, 37(3): 14-23

[26] Yang J, Wei Q, Chen C, et al. NV-Tree: Reducing consistency cost for NVM-based single level systems//Proceedings of the Usenix Conference on File and Storage Technologies. Santa Clara, USA, 2015: 167-181

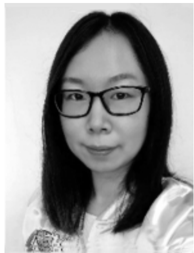
[27] Steinbrunn M, Moerkotte G, Kemper A. Heuristic and randomized optimization for the join ordering problem. The VLDB Journal, 1997, 6(3): 191-208

[28] Trummer I, Koch C. Multi-objective parametric query optimization. The VLDB Journal, 2015, 45(1): 1-18

[29] Bruno N, Galindo-Legaria C, Joshi M. Polynomial heuristics for query optimization//Proceedings of the 2010 IEEE 26th International Conference on Data Engineering. Long Beach, USA, 2010: 589-600

[30] Ou J, Shu J, Lu Y. A high performance file system for non-volatile main memory//Proceedings of the 11th European Conference on Computer Systems. Paris, France, 2016: 12

[31] Volos H, Tack A J, Swift M M. Mnemosyne: Lightweight persistent memory. ACM SIGPLAN Notices, 2011, 47(4): 91-104



MA Zhu-Lin, Ph. D. candidate. Her research interests include database design, non volatile memories and optimization algorithms.

LI Xin-Chi, M. S. candidate. Her research interests focus on main memory database.

ZHUGE Qing-Feng, Ph. D., professor. Her research interests include operating systems, embedded systems and software, and optimization algorithms.

WU Lin, Ph. D. candidate. His research interests focus on file system.

CHEN Xian-Zhang, Ph. D., lecturer. His research interests include new-generation memory systems, file systems, embedded systems and software, and cloud computing.

JIANG Wei-Wen, Ph. D. candidate. His research interests include high-level synthesis, real-time systems, supply-chain management, non-volatile memories, and optimization algorithms.

SHA Edwin H-M, Ph. D., professor, Ph. D. supervisor. His research interests include new-generation memory systems, cloud computing, operating systems, embedded systems and software, and communication security.

《计算机学报》

Background

Join operations combine the information from separate data sets in terms of data analysis, which are widely employed in the data intensive systems, such as relational databases, Hadoop, and Spark. With the development of storage technology and the increasing demand on high performance, non-volatile memories (NVMs) with the DRAM-like performance and disk-like persistency have the potential to replace both DRAM and disk to maintain data sets. In the novel storage architecture, the design of join operations faces a lot of emerging problems. One of the most challenging problem is that the write activities on NVM should be minimized due to the limited endurance and the write disturbance of NVM.

However, existing join algorithms, such as left-deep join, bushy-join and join algorithm of MySQL, incur large number of write activities on NVM, which will not be appropriate for NVM.

This paper studies what is a good join algorithm for NVM and we have two observations. First, different join orders may greatly affect the number of NVM writes. Second, the intermediate data stored in NVM may result in a lot of NVM writes. By understanding the relevance among different data sets, we devise an efficient algorithm, which is called NVjoin, to determine the join orders that can significantly reduce the number of NVM writes. To further reduce write activities on intermediate results, we propose a novel structure, called LWTab, to organize data that can take full advantage of the byte-addressable property of NVM. Compared with join orders generated by MySQL, NVjoin can achieve 104.21 times reductions on the number of NVM writes. In addition, LWTab can further achieve 16.74 times reductions.

This paper is supported by the National High Technology Research and Development Program (863 Program) of China under Grant No. 2015AA015304, the National Natural Science Foundation of China under Grant No. 61472052, the Chinese Postdoctoral Science Foundation under Grant No. 2017M620412. These projects aim to optimize big data systems with emerging persistent memories. This paper is part of the topic for optimizing join algorithms with persistent memory.