多核处理器环境下内存数据库索引性能分析

郭 超 李 坤 王永炎 刘胜航 王宏安
(中国科学院软件研究所 北京 100190)

摘 要 从20世纪80年代内存数据库出现时的T树到21世纪初出现的缓存感知的CSS、CSB$^+$树等，都适应了当时的硬件发展趋势，具有一定的性能优势. 随着计算机硬件技术的进一步发展，尤其是多核技术的应用与推广，新的多核处理器在提高索引性能的同时又给内存索引结构提出了新的挑战. 文中对B$^+$树、T树、CSS树、CSB$^+$树等几个经典的内存索引结构在多核处理器环境下的性能进行了全面的实验测试，对其在多核处理器环境下不同数据输入、不同节点大小等多种情况下的性能构成与差异进行了比较和分析，总结了在多核处理器中影响索引性能的关键因素，为内存索引结构的进一步改进奠定了坚实的基础.

关键词 内存索引结构；多核处理器；缓存感知
中图法分类号 TP311 **DOI号**：10.3724/SP.J.1016.2010.01512

The Performance Analysis of Main Memory Database Indices on Multi-Core Processors

GUO Chao LI Kun WANG Yong-Yan LIU Sheng-Hang WANG Hong-An
(*Institute of Software, Chinese Academy of Sciences, Beijing* 100190)

Abstract There are more and more advanced technologies used to improve the performance of processors, e.g., SMT and CMP. In one hand, these technologies improve the main memory indices' performance, and in the other hand, they create new challenges for these indices. To design a high-performance index, we should carefully evaluate the behavior of the multi-core processor while searching and updating. We have already known how traditional database indices perform on single-core CPU. This paper chooses some well-known indices including B$^+$ tree, T tree, CSS tree, CSB$^+$ tree and provides a thorough experimental study to show how these indices perform on multi-core processors in different conditions, such as different node size, different data input, and so on. From the results, some useful advices about index improvement can be got.

Keywords main-memory index; multi-core processor; cache conscious

1 引 言

随着技术的发展，内存容量越来越大，内存的价格也越来越低. 将数据库处理一个事务所需要的数据，甚至将整个数据库中的数据放入内存成为了可能[1]. 20世纪80年代提出内存数据库的概念以来，各种各样的内存数据库以及内存缓存前端产品层出不穷. 内存数据库，可将数据库中所有的数据或某几个事务所需要的数据常驻内存，在处理事务的过程中，尽可能减少或者完全消除和硬盘之间的数据交互工作[1]. 在内存数据库中，影响数据库性能的决定

收稿日期：2010-06-11. 本课题得到国家“八六三”高技术研究发展计划项目基金(2008AA04A105)资助. **郭 超**，男，1985年生，博士研究生，主要研究方向为内存数据库索引. E-mail: gchgchgch85@163.com. **李 坤**，男，1982年生，博士，助理研究员，主要研究方向为内存数据库. **王永炎**，男，1978年生，高级工程师，主要研究方向为数据挖掘和实时数据库. **刘胜航**，男，1980年生，硕士，工程师，主要研究方向为内存数据库. **王宏安**，男，1963年生，研究员，博士生导师，主要研究领域为实时智能、实时数据库、人机交互技术.

因素也由传统关系数据库中的磁盘访问转变为计算时间以及内存访问的延迟上.内存数据库和传统数据库的这一重要的差别，为内存数据库在数据组织、索引构建等方面提出了新的要求.

索引是影响数据库性能的一个重要的因素.为了适应内存数据库的特点，除了传统关系数据库中广泛使用的 B/B^+ 树被继续沿用于内存数据库中外，从20世纪80年代开始，研究者为改进内存数据库的索引结构进行了大量的工作.其中，影响较大的有80年代适应内存数据库而提出的T树[2]以及基于缓存敏感进行改进的CSS[3]树、CSB^+ 树[4]等.

为了进一步改进CPU的性能，越来越多的新的技术被应用于现代CPU中，如SMT(Simultaneous Multithreading)和多核技术[5].探知原有内存数据库索引结构在应用新技术的多核处理器中的性能表现，了解影响索引结构性能的重要因素，对改进索引结构，提高内存数据库的整体性能，具有重要的参考价值和指导意义.

本文选取了 B^+ 树、T树、CSB^+ 树、CSS树等几种应用比较广泛的数据库索引结构，针对多核处理器环境，通过实验分析了它们在不同的数据输入、不同的节点大小等多种情况下的主要性能指标，并与其在单核处理器环境下的性能进行比较分析，为进一步改进索引结构，提出适合多核处理器的内存数据库索引，提高内存数据库在多核处理器环境下的查询和更新性能打下坚实基础.

本文第2节为相关研究，主要介绍随硬件发展内存数据库索引结构的发展过程；第3节，分析在程序运行过程中，主要的时间消耗；第4节为主要的实验设计；第5节为具体的实验内容以及实验结果的比较分析；第6节为全文的总结以及未来工作的安排.

2　相关研究

从20世纪80年代提出内存数据库概念以来，为适应内存数据库以及硬件环境的特点，除了传统的 B/B^+ 树被继续使用在内存数据库中之外，研究者在提高内存数据库索引性能方面进行了大量的研究工作，各种各样的内存索引及改进方案被提出.

T树是20世纪80年代提出的面向内存数据库的索引结构.它继承了AVL树的二叉平衡结构和 B^+ 树节点包含多个关键字的特征，因此具有二叉树收敛快、多关键字降低树高的特点，查询过程中执行的指令数较少，在当时的硬件环境下，查询性能较好.在T树和 B^+ 树的基础上，研究者进行了一系列的改进，如基于压缩关键字，提高节点利用率而改进的pk-T树[6]和pk-B树[6]，结合hash结构和 B^+ 树结构的BD树[7]等，都在一定程度上改进了索引性能.

随着CPU技术的进一步发展，CPU的处理速度和内存访问速度之间的差距也越来越大，内存访问的时间延迟，从1980年的不到10个时钟周期，增长到2010年的近1000个时钟周期，CPU cache的缺失成为影响数据库索引性能一个不可忽略的因素.研究表明，超过50%的数据库处理时间浪费在内存访问的时间延迟上[9]，即CPU cache缺失带来的时间消耗.研究者发现，在新的CPU环境中，B/B^+ 树由于较好的缓存感知(cache conscious)特性，表现出良好的查询和更新性能，索引性能甚至好于T树[3].

针对这一趋势，一些新的基于缓存感知的索引被提出，其中影响较大的有基于 B^+ 树、T树和缓存感知思想进行改进的CSS树、CSB^+ 树和CST树[10].

CSS树(Cache Sensitive Tree)是Rao Jun等于1998年提出的针对缓存优化的树索引结构.CSS树用一个数组来保存树结构，每个节点的大小和二级缓存块(L2 cache line)的大小保持一致.查询时，通过计算数组下标找到子节点所在的位置.CSS树避免了指针的空间开销，节点利用率高；节点的大小和二级缓存块的大小一致，减少了节点内部查找的缓存缺失；用连续数组存储索引结构，减少了TLB缺失.在查询方面，CSS树具有很好的性能.但是，CSS树的更新操作需要重新构建整个索引结构，因而不适用于更新比较频繁的应用场景.

沿袭了CSS树缓存感知(cache conscious)的思想，Rao Jun等人在 B^+ 树的基础上提出了 CSB^+ 树.CSB^+ 树对 B^+ 树做了两方面的改进：(1) CSB^+ 树的节点大小和二级缓存块的大小保持一致；(2) CSB^+ 树中，每个内部节点对应的子节点都存放在连续区域中.通过这两方面提高 CSB^+ 树的缓存命中(cache hit)率和减小TLB的缺失率.在查询方面，CSB^+ 树较 B^+ 树有更好的性能.和CSS树相比，CSB^+ 树又具备增量更新的能力.但是，由于 CSB^+ 树的内部节点的子节点存放在连续区域中，在更新过程中，需要进行大量的节点复制操作，更新性能较差.

2007年，基于缓存感知的思想，Lee Ig-Hoon等人在T树的基础上提出CST树，也具有一定的性能优势.

处理器频率的大小是评测CPU性能最重要的

指标.在过去的几十年内,处理器设计者也一直致力于提高处理器频率的工作.从1993～2003年期间,CPU的频率以每18个月或2年翻一番的速度增长,当频率增长到4GHz,单核CPU的频率达到了一个极限[11].继续提高处理器频率将导致功耗和热量以指数级别增长,带来的影响弊远大于利.

多核技术(Chip Multiprocessors CMP)有效地解决了单核处理器性能提高和功耗增长的矛盾.多核技术就是将多个CPU内核集成到同一块芯片中,通过多个频率较低的核的并行合作,达到较高的处理效率,有效地解决了功耗的问题.在单核处理器中,往往通过SMT机制来并行处理多个线程:当一个线程处于等待状态时,处理器处理另外的线程,以降低线程延迟给处理器带来的影响,从而提高处理器利用率,实现多个线程并行处理.

多核处理器比单核处理器有更强的并行能力.在多核处理器中,处理器的每个核能够同时并独立处理任务,核与核之间,是真正的并行关系;在核的内部,SMT机制能进一步提高整个处理器的并行处理能力.

在多核处理器中,核与核之间需要进行通信,共享cache是主流的通信机制之一.一般的,对于含有两层cache的多核处理器,每一个核有独立的L1 cache,共享L2 cache(如图1),并通过连接核心的总线进行通信.多个核之间对共享的L2 cache存在的竞争关系,影响处理器的性能.

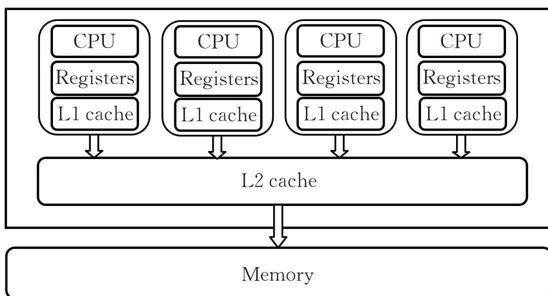


图1 多核处理器结构示意图

多核处理器的新特性给开发人员带来了新的挑战,如何充分利用多核处理器的并行特性,规避资源竞争的影响,尽可能地提高每个核的利用率是软件开发人员面临的一个重要难题.

Ailamaki Anastassia等人在文献[9]中从理论和实验两方面,分析了单核处理器中影响索引结构性能的主要因素;Kim Kyunghwa等人在文献[15]中又从整体时间和L2缓存缺失方面比较了B^+树、CSB^+树等几种主要的内存数据库索引在多核处理器中的性能表现,但缺少对TLB缺失、L1缓存缺失等其它影响索引性能指标的测试及对实验结果的全面分析.

3 程序执行时间消耗分析

索引结构的操作一般分为3种:查询、更新和删除.假设总时间用T表示,有效指令执行时间用T_I表示,存储介质访问时间延迟用T_M表示,分支预测错误延迟用T_{misp}表示,其它资源访问延迟用T_{other}表示,这几个时间延迟的重合部分用$T_{overlap}$表示,我们可以得到索引操作的总时间公式如下:

$$T=T_I+T_M+T_{misp}+T_{other}-T_{overlap} \quad (1)$$

3.1 指令有效执行时间

指令有效执行时间指CPU在执行指令时耗费的时间,是程序执行时间最主要的部分.不同的索引结构的差异关键在于执行的指令不同.指令的有效执行时间和指令数I以及每条指令执行的时间相关,用CPI来表示平均的指令执行时钟周期数,则有

$$T_I=I\times CPI \quad (2)$$

在树结构的索引中,查询过程的指令主要由遍历树过程执行的指令和节点内部查找的指令构成.

更新和删除操作除了遍历树和节点内部搜索比较的指令之外,还包括了对要更新或删除的节点的更新或删除操作指令以及由此引起的可能的数据复制和树的调整转置等指令.

3.2 存储介质访问时延

影响性能的存储介质主要有3种:CPU缓存(CPU cache)、快表(Translation Look-aside Buffer TLB)以及内存.

一般的CPU缓存分为两级:L1 cache和L2 cache,部分处理器还有L3 cache,由于实验中用到的处理器均无L3 cache,因此本文仅讨论包含L1和L2两级缓存的情况.

CPU缓存是为了缩短CPU处理速度和内存访问速度之间的差异而设计的.CPU获取指令或数据时,先从L1 cache获取,如果L1 cache中没有要获取的内容(L1 cache miss),则从L2 cache获取,如果L2 cache包含要获取的内容,则取得该内容并缓存至L1 cache,否则从内存中获取需要的内容(L2 cache miss).一般的,CPU从L1 cache获取数据需要几个时钟周期,从L2 cache获取数据需要几十个时钟周期,而从内存获取内容则需要上百个甚至上千个时钟周期.CPU执行指令的过程中,获取数据的过程就是CPU等待的过程.尽管out of order execution技术[9]能够减少少量的存储访问延迟,但是,在树结构的索引中,这种减少几乎可以忽略不计.L1 cache和L2 cache缺失仍是影响索引性能的

重要因素，如何尽可能地减小 L1 和 L2 cache 的缺失率，是数据库索引研究的一个重要方向.

快表是页表内容的缓存. CPU 要先根据逻辑地址在页表中查找到对应的物理地址，再根据物理地址获取需要的数据或指令. 为减少地址映射过程中的访问延迟，部分页表内容被缓存于快表中. 进行地址映射时，先查找快表，如果要查找的地址项不在快表中，再查找页表，并将找到的页表项存入快表. 快表分为指令地址快表和数据地址快表.

综上可知，存储介质的访问时延有如下公式：

$$T_M = N_{hitL1} \times latency_{L1} + N_{missL1} \times latency_{L2} + N_{missL2} \times latency_M + N_{hitTLB} \times latency_{TLB} + N_{missTLB} \times latency_M \quad (3)$$

其中，N_{hitL1} 表示一级缓存的命中次数，$latency_{L1}$ 表示访问一级缓存的延迟，N_{missL1} 表示一级缓存的缺失次数，$latency_{L2}$ 表示访问二级缓存的延迟，N_{missL2} 表示二级缓存的缺失次数，$latency_M$ 表示访问内存的延迟，N_{hitTLB} 表示 TLB 命中次数，$N_{missTLB}$ 表示 TLB 的缺失次数，$latency_{TLB}$ 表示访问 TLB 的延迟. 其中，一级缓存缺失的延迟，二级缓存缺失延迟和数据地址快表的缺失延迟是访问存储介质时间延迟的主要部分，本文的实验也主要关注这三个部分. 在文章接下来的部分中，TLB 均指数据地址快表.

3.3　分支预测错误延迟

CPU 在处理分支指令的时候，总是先预测一个分支并开始执行，当分支条件判断完成之后，再判断分支预测是否正确，如果正确则继续执行分支，否则，将结束预测分支的执行并回滚已经完成的操作. 分支预测能够有效地提高流水线的吞吐量，但是如果分支预测错误(branch mispredictions)频率过高，将影响算法的整体性能. 在数据库索引的查询、更新过程中，节点分支的选择、节点内部的比较操作将产生大量的分支指令，分支预测错误延迟成为影响索引性能不可忽视的重要因素.

在实验中，我们主要从执行时间、L1 和 L2 缓存缺失、TLB 的缺失及分支预测错误数目等方面比较不同索引结构在不同的处理器环境以及不同的数据输入、不同的节点大小等多种情况下的性能差异.

4　实验设计

4.1　实验的软件环境

本文选取了 B^+ 树、T 树、CSS 树、CSB^+ 树进行如下的实验：

(1) 比较这 4 种索引在多核环境下的性能表现，测试内容包括整体的执行时间、缓存缺失、分支预测错误等指标，并分析产生性能差异的原因；

(2) 比较多核环境下，同一索引结构在线性数据和随机数据等不同的数据输入情况下，查询和更新的不同表现，分析数据特点给索引性能带来的影响；

(3) 比较多核环境下，同一索引结构在相同硬件条件下随节点大小变化产生的性能差异，分析节点大小给索引性能带来的影响；

(4) 比较相同条件下同一索引结构在不同机器中的表现，分析不同处理器指标，如缓存大小等对索引性能的影响，分析多核技术对索引性能的影响.

实验中，B^+ 树、CSS 树、CSB^+ 树选择 Rao Jun 等人在文献[4]中使用的版本，T 树选择 FastDB 中 T 树的实现. 由于选取的 CSB^+ 树实现的删除操作为 lazy delete，删除的过程和查询的过程基本一致，故本文的实验均只比较更新和查询操作的性能.

每部分实验均进行 10000000 次，将测得的数据取平均值作为每次操作的指标进行比较分析.

实验系统为 Windows2003 Standard，各项指标通过 Intel VTune Performance Analysis9.0 采集.

4.2　实验的硬件环境

实验选取 3 台处理器不同的机器，型号分别为 Intel® Core™ 2 Quad CPU、Intel® Pentium® D CPU、Intel® Pentium® 4 CPU，用 A、B、C 表示这 3 台机器，通过 Sisoftware Sandra 2010. SP1d 和 RightMark Memory Analyzer V3.8 测得硬件指标如表 1 所示.

表 1　实验机器指标

	指标									
	处理器速度/GHz	Core 数目	L1 cache	L1 cache line /B	L1 latency/cycle	L2 cache	L2 cache line /B	L2 latency/cycle	Memory/MB	Memory latency/cycle	TLB cache/Entry
机器 A	2.33	4	32KBx4 8-way	64	3	2MBx2 8-way	64	17	4096	170	256
机器 B	3.4	2	16KBx2 8-way	64	5	2Mx2 8-way	64	40	1024	450	64
机器 C	2.4	1	8KB 4-way	64	3	512K 8-way	64	23	768	285	64

5 实验结果分析

5.1 4 种索引结构的性能比较

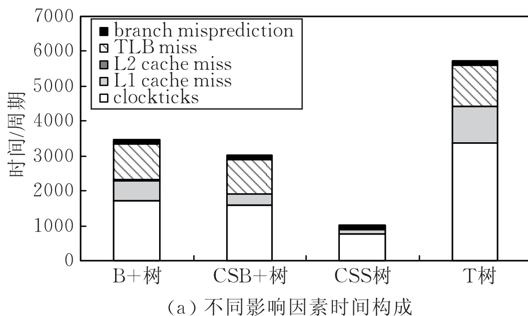
这一部分实验比较在多核环境下，4 种索引结构的性能表现.

实验分成查询和更新两个部分，由于 CSS 树是静态索引结构，所以在更新操作的比较中，我们仅比较 B^+ 树、CSB^+ 树以及 T 树的性能差异.

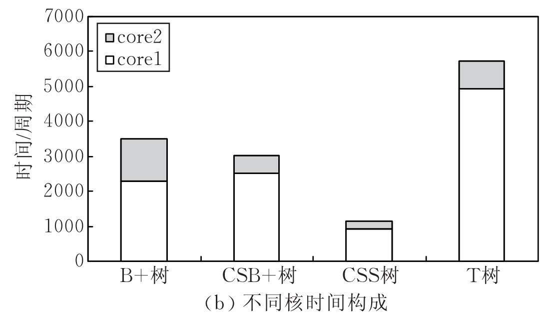
更新操作：从空结构开始，插入 10000000 不重复的随机数.

查询操作：构建包含 10000000 个关键字(key)的树结构，进行 10000000 次查询操作，查询的关键字随机生成.

以时钟周期为单位，可以得到图 2、图 3 所示机器 B 的实验结果.(a)表示平均每次查询或更新时间构成，其中 clockticks 表示处理器两个核执行指令花费的总时钟周期；L1 cache miss 表示总的 L1 缓存缺失的延迟；L2 cache miss 表示总的 L2 缓存缺失延迟；TLB miss 表示总的 TLB 缺失延迟；branch misprediction 表示总的分支预测错误延迟，我们假设每次分支预测错误需花费 10 个时钟周期来处理.(b)表示每个核分别的处理时间.

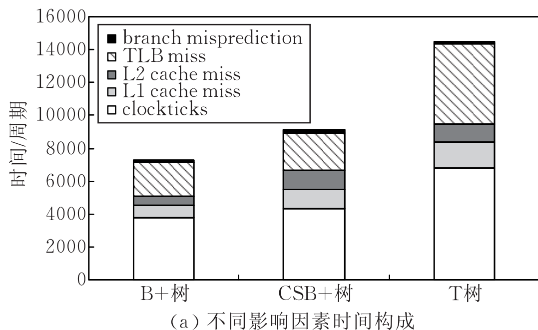


(a) 不同影响因素时间构成

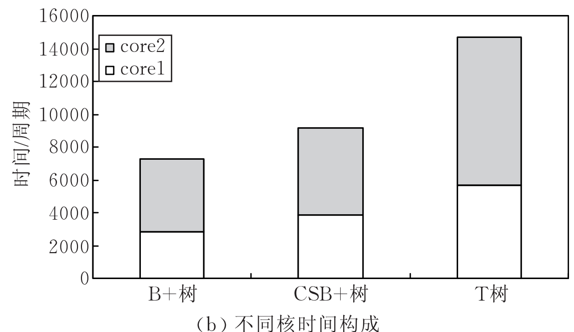


(b) 不同核时间构成

图 2 机器 B 查询时间比较



(a) 不同影响因素时间构成



(b) 不同核时间构成

图 3 机器 B 更新时间比较

从图 2(a)中可以看到，在多核环境中，TLB 以及 L1、L2 缓存缺失带来的时间延迟在整体的时间中仍占很大比重，以 B^+ 树为例，平均每次查询中，TLB 缺失带来的时间延迟约为 1038 个时钟周期，约占整体时间的 1/3；L1 cache 缺失带来的时间延迟约为 566 个时钟周期，约占整体时间的 1/6；由于机器 B 有较大的二级缓存，因此 L2 cache miss 带来的时间延迟相对较少. 由分支预测错误及缓存缺失带来的时间延迟，占整体时间的 1/3～1/2 左右.

从图 2 中可以知道，在查询方面，CSS 树的性能是最好的. 这是因为 CSS 树树结构的连续存储提高了 TLB 的命中率和缓存命中率，因此，在各项指标上，CSS 都有着比较突出的优势. CSB^+ 树采用连续的区域来存储节点的子节点，TLB 缺失比 B^+ 树少；同时减少了指针的使用，提高了节点的利用率，在查询过程中，CSB^+ 树访问的内存页较少，产生 L1、L2 缺失概率也相应较小；而 T 树的 TLB 以及 L1、L2 缓存缺失最多，体现出最差的性能.

图 3 是以时钟周期为单位的平均每次更新的时间构成. 从图 3 中可以看到，在索引结构的更新操作中，存储介质的访问延迟对索引性能影响很大，L2 cache 缺失的影响也更为明显. 这是因为在更新操作过程中，需要进行大量的节点分裂(如 B^+ 树，CSB^+ 树)，或者节点的旋转调整(如 T 树)，这些操作带来大量的内存的分配和释放，更新过程访问的内存页比较多，产生更多的 TLB 及 L1、L2 缓存

缺失.

对于 CSB^+ 树而言，由于将所有的节点的子节点都存放在连续区域，在节点分裂的时候，较 B^+ 树需要更多的内存复制的操作，所以 CSB^+ 树更新过程的指令执行时间及内存访问次数都比 B^+ 树多，产生的缓存缺失和 TLB 缺失的也比较多.

从本节实验可以看到，当 L2 cache 达到 2Mx2 大小时，L2 cache 缺失对索引查询性能的影响很小；而无论是在查询还是更新操作，索引 TLB 缺失和 L1 cache 缺失对索引性能的影响都很大，因此，对索引性能的改进应更注意考虑减少 TLB 和 L1 cache 的缺失，尤其要减小 TLB 缺失带来的时间延迟.

5.2　不同输入对索引性能的影响

这部分的实验主要考察在多核环境中，不同的数据输入特征给索引结构性能带来的影响.

实验也分为查询操作和更新操作两部分.

查询操作：构建包含 10000000 个关键字的树结构，查询的数据有 3 种：① 线性数据. 数据按照 1～10000000 顺序查询；② 随机数据. 随机生成 10000000 个 10000000 以内的随机数据进行查询；③ 按概率生成数据. 概率参考文献[4]中所用的概率函数，$N_i=\frac{i^{1.3}}{10}$，其中 i 表示待查询关键字序号.

更新操作：更新的数据类型分为随机不重复的数据和连续的线性数据两种. 两种数据都为 10000000 条，从空的树结构开始更新.

图 4 和图 5 分别表示机器 B 中平均每次查询和更新所有核总的相关指标值. 其中，(a)表示总的时钟周期，(b)表示总的缓存缺失（包括 L1 cache、L2 cache、TLB 之和）带来的时间延迟（以时钟周期为单位），(c)表示分支预测错误数目.

从图 4 的 3 个图可以看到，线性数据和以概率产生的数据由于相邻两个查询关键字比较接近甚至重合，查询过程中遍历的路径的重合度比较高，访问相同节点的概率也相应增高，因此具有较高缓存命中率；临近的查询关键字值比较接近，选择同样分支的概率也大大增高，分支预测正确率也随之提高. 由此可见，在批量查询中，处理顺序的或数据重合度较高的数据时，索引的整体性能要比处理随机数据好. 在大部分情况下，由于概率产生的数据重合度比较高，产生的缓存命中率和分支预测正确率都较线性数据高，索引具备有更好的性能.

从图 5 可以看到，由于更新过程中树的遍历操作在整体操作中占很大比重，线性数据的更新，较随机数据的更新，临近更新遍历树过程中访问相同节

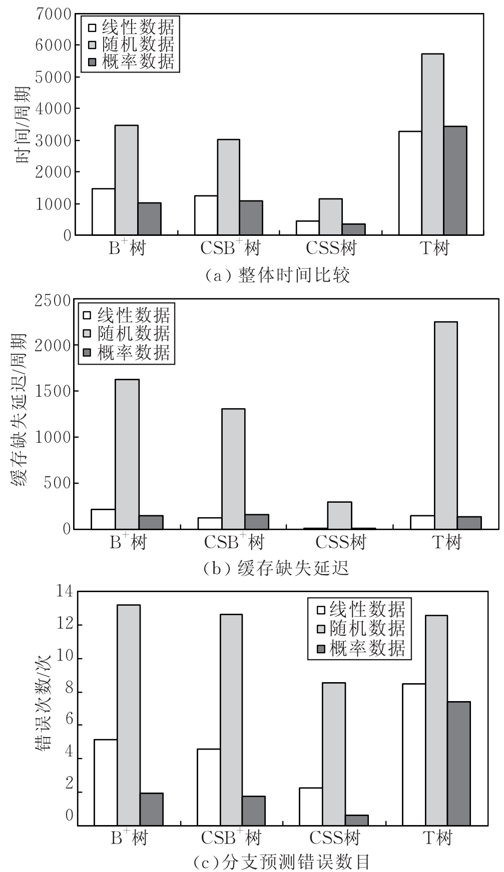


图 4　不同数据特征查询操作比较

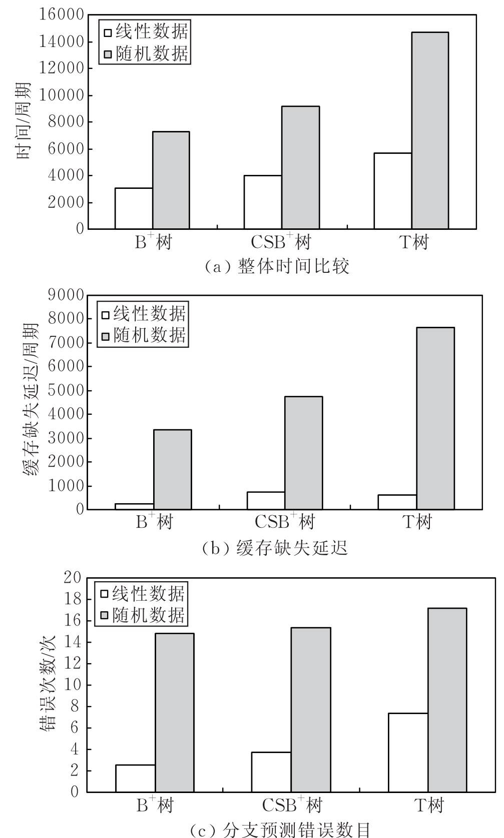


图 5　不同数据特征更新操作比较

点的概率增大,因此在缓存命中等指标上性能优势都较突出,具有更好的整体性能.

从本实验可以知道,在批量查询或者更新的时候,对关键字进行排序,是提高整体处理性能的一个较为简单而有效的方法.这种方法同样适用于单核处理器的机器 C 和四核处理器的机器 A 中.

5.3 节点大小对索引性能的影响

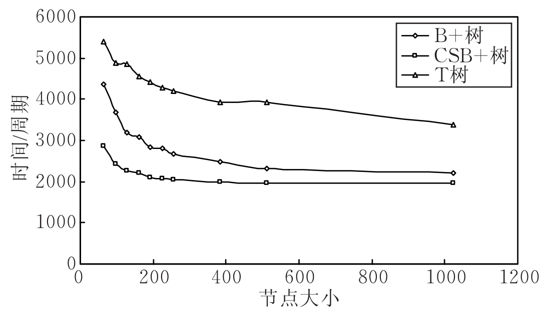
从文献[8]中可以知道,在单核环境中,索引性能随着节点的增大而提高;本节实验,我们将比较分析在多核环境中,B^+ 树、CSB^+ 树、T 树随着节点大小的变化,查询和更新性能的变化趋势.

查询操作:构建包含 10000000 个关键字的树结构,进行 10000000 次查询,查询关键字随机生成,所有查询操作使用统一的实验数据.

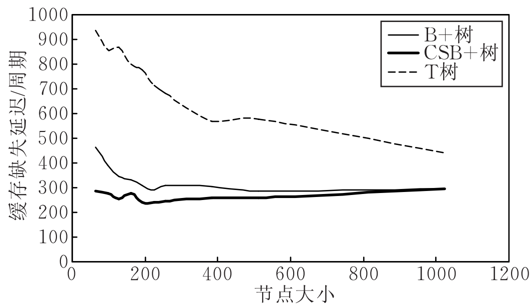
更新操作:更新 10000000 个不相等的随机关键字,所有更新操作使用统一的实验数据.

节点的大小从 64bytes 开始,每次增加 32bytes,直至 1024bytes,由于机器 A 和机器 B 中的实验结果基本一致,故仅选择机器 B 的结果进行分析.

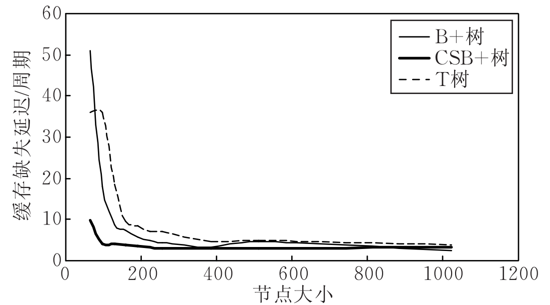
图 6、图 7 分别表示在查询和更新操作中随着节点大小变化指标发生的变化.(a)表示每个查询或更新操作所有核花费的总时钟周期变化情况,(b)表示每个查询或更新操作总的 L1 缓存缺失延迟,(c)表示总的 L2 缓存缺失变化,(d)表示总的 TLB 缺失延迟的变化情况,(e)表示总的分支预测指令错误数变化情况,延迟均以时钟周期为单位.



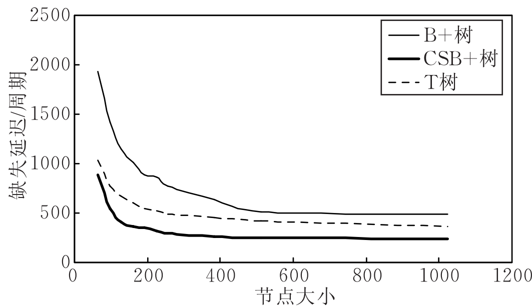
(a) 整体时间变化



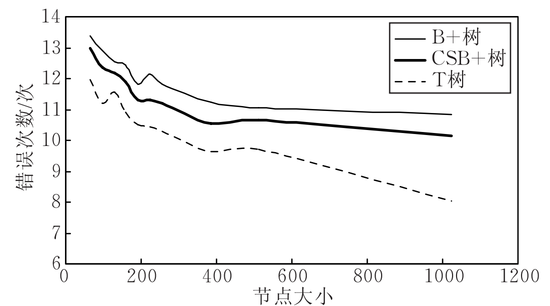
(b) L1 缓存缺失



(c) L2缓存缺失



(d) TLB缺失延迟

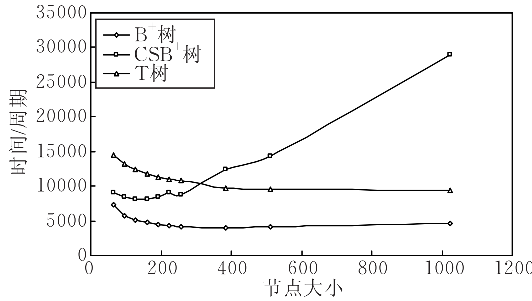


(e) 分支预测错误数目

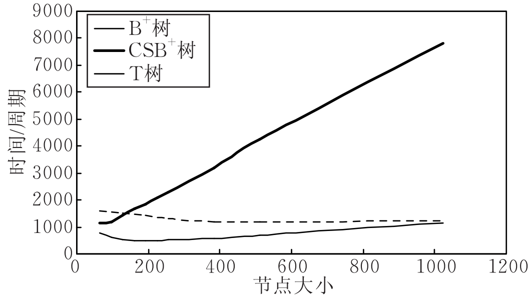
图 6 不同节点大小的查询性能比较

从图 6(b)、(c)中可以看到,随着节点的增大,3 种索引结构的 L1 缓存和 L2 缓存缺失延迟都随之减少.这是因为当节点增大时,每个节点能存放的关键字的数目增大,节点被重复选择的概率增大;同时节点增大,树高减小,查询遍历树的过程中访问节点的数目也减少,尽管随着节点增大访问每个节点所产生的缓存缺失率可能提高,但当树高及每个节点关键字数目增多带来节点重复访问的影响更大时,L1 缓存和 L2 缓存的命中率就会随之增大.

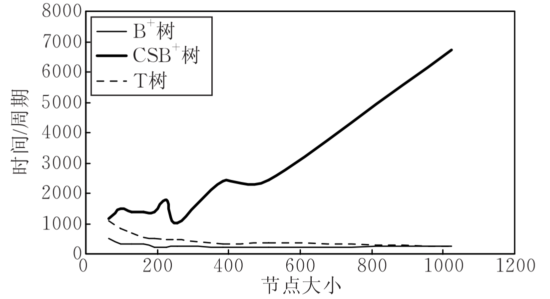
当节点较小时,同样规模关键字索引结构的节点数目比较多.由于节点的内存是动态分配的,所以节点占用内存页的数目也相应较多,查询操作要访问的



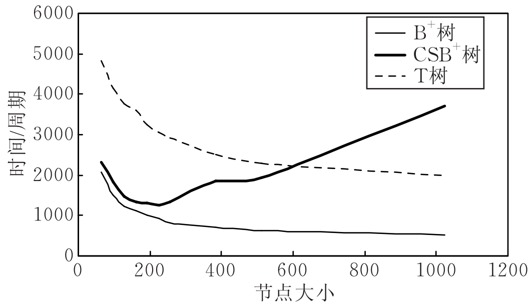
(a) 总时间比较



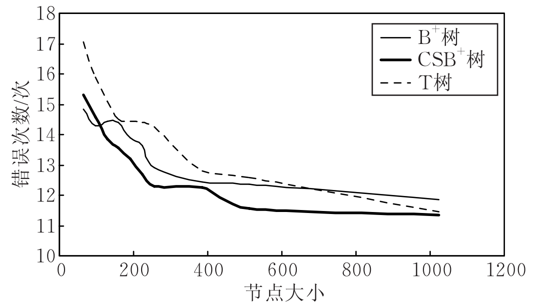
(b) L1缓存缺失延迟



(c) L2缓存缺失延迟



(d) TLB缺失延迟



(e) 分支预测错误数目

图 7　不同节点大小的更新性能比较

内存页也相应增多，当节点增大时，访问的内存页的数目减小，产生 TLB 缺失的概率也就相应的比较小. 如图 6(d)所示，随着节点的增大，TLB 缺失减小.

从图 6(e)中可以看到，随着节点增大，树高减小，查询过程中产生的分支指令相应的减少，分支预测错误的概率也随之减小.

B^+ 树和 T 树的更新过程主要是树的遍历查找过程，因此如图 7 中(b)～(e)所示，在 L1、L2 缓存缺失延迟、TLB 缺失延迟以及分支预测错误方面，B^+ 树和 T 树的查询和更新所表现出来的趋势基本一致，更新操作的整体性能也随着节点的增大有所提高；节点的增大，也给 B^+ 树的分裂过程以及 T 树的旋转过程带来不利的影响，节点越大，分裂和旋转过程中需要复制的数据越多，因此在更新的操作中，节点的增大带来的性能优化不甚明显.

从图 7(b)、(c)可以看到，CSB^+ 树的 L1 和 L2 缓存缺失随着节点的增大而增多，这是因为，CSB^+ 树更新时可能的节点分裂操作，需进行大量的节点复制操作，访问较多的内存页. 节点越大，复制的内容越多，访问的内存页也就越多，由此带来的 L1、L2 缓存缺失也就越多. 从图 7(d)中可以看到，当节点小于某个值时，节点增大没有增加内部查找时内存页的访问数，但随着节点增大，树的高度减小，因此更新时遍历树的过程中访问的总的内存页数目减少，TLB 缺失随之减小；随着节点继续增大，节点内部查找及节点复制所访问的内存页增多，超过了树高减小带来的好处，TLB 缺失就随之增大. 当复制操作成为影响索引性能的最主要因素时，索引性能降低. 如图 7(a)所示，CSB^+ 树的更新性能随着节点的增大而变差.

综合图 6、图 7 的(b)～(e)可以知道，随着节点的增大，B^+ 树和 T 树节点查询和更新过程中所产生的 L1、L2 缓存缺失、TLB 缺失、分支预测错误都相应的减少，查询的整体性能随之提高. 但是，随着节点不断增大，节点的利用率逐渐降低，节点大小的增大带来有利影响(包括树高减小和节点被重复选择的概率增大)和不利影响(访问单个节点产生更多的缓存缺失)差距越来越小，L1、L2 缓存缺失和快表

缺失的减小趋于平缓;而且随着节点增大,每个节点的关键字增多,节点内部二分查找的时间消耗也越来越多,整体性能提高也趋于平缓.

此外,节点增大,节点利用率降低,构建树所耗用的内存也逐渐增加,节点大小的选择,应该综合考虑性能提高的速率和节点的利用率.从实验可以看到,在节点大小是二级缓存块大小 1～3 倍时,索引查询和更新性能提高最快.多核环境中索引性能随节点大小变化的趋势和单核环境基本一致.

从 5.1～5.3 节的实验中,我们可以看到,在多核环境中,传统的内存数据库索引之间性能对比及同一索引在不同的数据输入和节点大小等情况下的性能变化,表现出和单核基本一致的结果.这是因为,现有的索引的查询和更新操作,均是单线程操作,多核处理器环境中,操作系统在执行指令时,能够根据核的负载情况,选择合适的核来执行.从 5.1 节的实验中,我们看到,指令是被分发到不同的核中进行处理的,多核协作提高了索引操作线程的指令被选择的概率,一定程度上减少了线程执行的时间跨度.但是,除了和单核处理器一样由 out of order execution 技术带来少量的指令并行以外,同一个操作线程中的不同阶段的指令是串行执行的,因而在多核处理器中表现出和单核处理器基本一致的趋势和现象.

为了充分利用多核处理器的并发能力,需要对原有的索引进行并发改进,改进可以从以下几个方面入手:(1) 改进索引结构,将单路查询转为多路查询,每一路查询用独立的线程处理;(2) 分解每个查询或更新任务,如并行处理二分查找过程等;(3) 对于批量查找或者更新的任务,将任务分组,并行处理每个任务组.

5.4 不同硬件环境对索引结构性能的影响

这一部分实验,比较不同的硬件环境下索引结构的性能差异.实验分为查询和更新两种操作.

查询操作:构建包含 10000000 个关键字的索引并进行 10000000 次查询,查询关键字随机生成;

更新操作:从空结构开始,更新 10000000 个不重复的随机数.

由于未进行并发改进的索引在多核处理器中指令的执行基本是串行的,指令执行的速度主要和处理器频率相关而与多核的特性基本无关,因此,在这个实验中,我们仅比较 CSB^+ 树、B^+ 树、T 树每次查询和更新的 L1、L2 缓存缺失次数、TLB 缺失次数、分支预测错误数目 4 种指标.机器 A 的 TLB entries 是机器 B 和 C 的 4 倍,为了更直观的比较,将机器 A 实际测得的 TLB 缺失数乘以 4 作为实验结果.不同索引的对比结果基本一致,鉴于篇幅限制,仅选取 B^+ 树的结果进行分析.

在多核系统中,不同的核拥有独立的 L1 缓存,L1 缓存的缺失主要和缓存的大小以及访问的内存大小有关,当访问内存基本一致时,核对应的 L1 缓存越大,产生的 L1 缓存缺失越小.如图 8、图 9 中(a)所示,机器 A 产生的总的 L1 缓存缺失小于机器 B,机器 C 的 L1 缓存最小,产生的 L1 缓存缺失最大.

多核处理器不同的核共享 L2 缓存,不同的核对 L2 缓存的访问存在竞争关系,当核越多时,对共享 cache 的访问竞争越大,可能产生的缓存缺失也越多.如图 8、图 9 的(b)所示,在 L2 缓存大小一致的情况下,四核的机器 A 产生的总的 L2 缓存缺失比双核的机器 B 大.L2 缓存的缺失率和缓存大小密切相关,L2 缓存越大,L2 缓存缺失越少,因此,机器 C 产生的缓存缺失远大于机器 A 和机器 B.

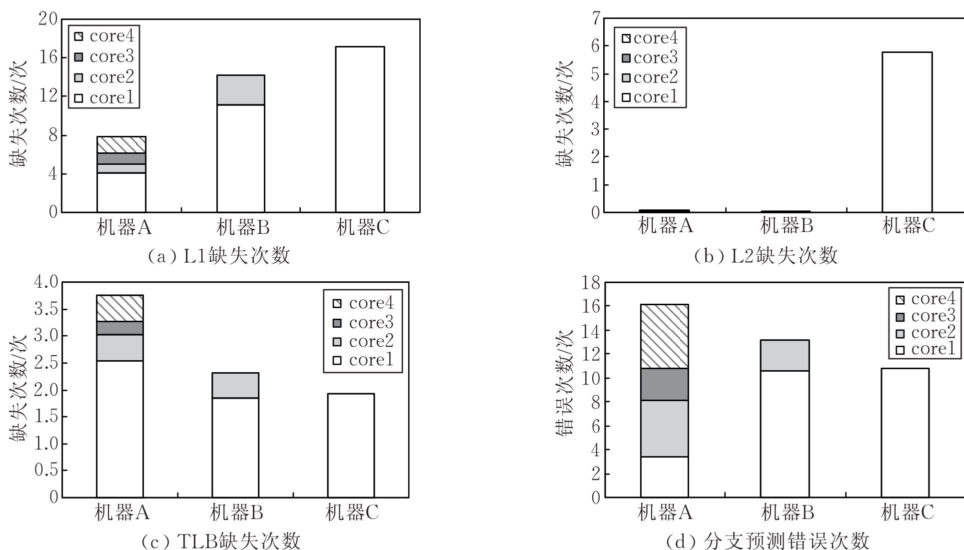


图 8 不同处理器 B^+ 树的查询性能比较

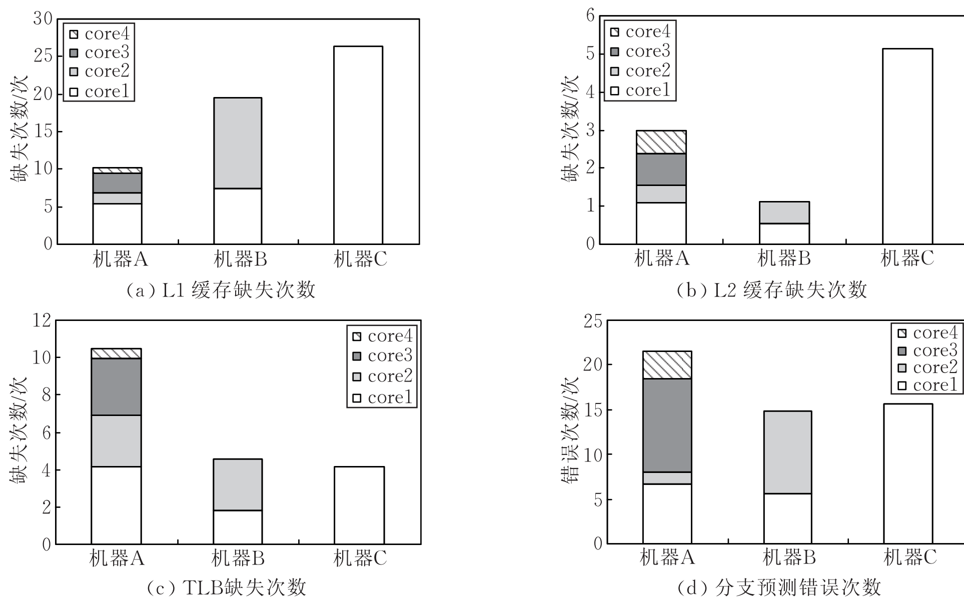


图 9　不同处理器 B^+ 树的更新性能比较

TLB 缺失和内存访问次序和 TLB 的大小等因素相关. 多核环境提高了处理器的并行能力，增加了处理器能够同时处理的线程数，不同核不同线程访问内存次序不尽相同，多核加强了处理器不同核不同线程对 TLB 的使用竞争. 一般的，如图 8、图 9 的(c)所示，核越多，并行能力越强，TLB 的使用竞争越大，TLB 缺失越多.

从图 8、图 9 的(e)我们可以知道，多核环境中，由于执行线程指令的核不是固定的，在大部分情况下，多核提高了分支预测指令错误产生的概率.

在多核环境中，我们在改进索引结构，增强索引操作的并发性，充分利用多核处理器的并行处理能力，提高每个核的利用率的同时，还应当尽可能地减少甚至避免多核并行处理导致的 L2 缓存和 TLB 竞争所带来的额外的时间延迟，这样才能充分提高索引的性能.

6　总　结

本文通过不同的实验，比较分析了多核环境中，B^+ 树、CSB^+ 树、CSS 树、T 树等主要的内存数据库索引结构的性能差异，实验内容主要包括在多核环境中索引更新和查询的时间构成、索引结构在不同的数据输入、节点大小、硬件环境中的性能差异.

从实验结果中可以得到如下结论：

(1) 在多核环境下，存储介质的访问延迟和分支预测错误的延迟在索引的查询和更新所消耗时间中仍占有较大的比重，有效地减少这些延迟，对提高索引的整体性能有着重要的作用. 当 L2 缓存达到 2M×2 甚至更大的时候，L2 缺失产生的影响变小，查询的优化应该由原来的减小 L2 缓存缺失转向减小 TLB 和 L1 缓存缺失.

(2) 一般的，对于树结构的索引，节点的增大是一个简单有效的提高查询性能的方法，但随着节点的不断增大，整体性能提高的增长率越来越小，最后趋于平缓. 在节点大小是二级缓存块大小的 1～3 倍时，索引性能提高最快. 由于 CSB^+ 树更新操作中节点的分裂，要进行大量的内存复制操作，节点的增大反而降低了更新性能.

(3) 在批量操作中，对数据进行排序是提高整体性能的简单而有效的方法.

(4) 在多核环境中，增强索引操作的并发性是利用多核处理器特性提高索引性能的重要手段. 主要可以从改进索引结构、分解单个查询或更新任务、将批量查询或者更新任务分组等方面入手.

(5) 在提高索引并发性的同时，还要注意多核多线程引起的二级缓存和 TLB 等资源的竞争，尽可能避免多核并发处理带来的额外的时间延迟，这也是在多核环境中提高索引性能的一个突破点和难点.

在将来的工作中，我们将进一步分析在多核环境中影响索引结构性能的因素，尤其是要分析并行处理时影响索引性能的因素，并在此基础上改进索引结构，将索引结构上的操作并行化，提高每个核的利用率，并尽量减少并发访问 L2 和 TLB 引起的冲突，以适应新的多核处理器的发展.

参　考　文　献

[1]　Garcia-Molina H, Salem K. Main-memory database systems:

An overview. IEEE Transactions on Knowledge and Data Engineering, 1992, 4(6): 509-516

[2] Lehman Tobin J et al. A study of index structures for main memory database management systems//Proceedings of the 12th VLDB Conference. Kyoto, Japan, 1986: 294-303

[3] Rao Jun, Ross Kenneth A. Cache conscious indexing for decision-support in main memory//Proceedings of the 25th VLDB Conference. Edinburgh, Scotland, UK, 1999: 78-89

[4] Rao Jun, Ross Kenneth A. Making B^+-trees cache conscious in main memory//Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data. Dallas, Texas, USA, 2000: 475-486

[5] Zukowski Marcin. Balancing vectorized query execution with bandwidth-optimized storage [Ph. D. dissertation]. Universiteit van Amsterdam, Amsterdam, The Netherlands, 1999

[6] Bohannon Philip, Mcllroy Peter et al. Main-memory index structures with fixed-size partial keys//Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data. Santa Barbara, California, USA, 2001: 163-174

[7] Cui Bin, Ooi Beng Chin et al. Main memory indexing: The case for BD-tree. IEEE Transactions on Knowledge and Data Engineering, 2004, 16(7): 870-874

[8] Hankins Richard A, Patel Jignesh M. Effect of node size on the performance of cache-conscious B^+-trees//Proceedings of the 2003 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems. San Diego, CA, USA, 2003: 283-294

[9] Ailamaki Anastassia et al. DBMSs on a modern processor: Where does time go?//Proceedings of the 25th VLDB Conference, Edinburgh, Scotland, UK, 1999: 266-277

[10] Lee Ig-Hoon, Shim Junho et al. CST-trees: Cache sensitive T-trees//Proceedings of the 12th International Conference on Database Systems for Advanced Applications. Bangkok, Thailand, 2007: 398-409

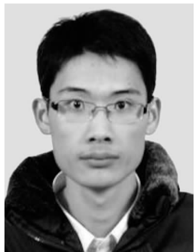
[11] Parkhurst Jeff, Darringer John et al. From single core to multi-core: Preparing for a new exponential//Proceedings of the 2006 IEEE/ACM International Conference on Computer-Aided Design. San Jose, California, USA, 2006: 67-72

[12] DeWitt D J et al. Implementation techniques for main memory database systems//Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data. Boston, Massachusetts, 1984: 1-8

[13] Bayer R, McCreight E. Organization and maintenance of large ordered indices//Proceedings of the 1970 ACM SIGFIDET (now SIGMOD) Workshop on Data Description, Access and Control. Houston, Texas, USA, 1970: 107-141

[14] Hsu Lisa R, Reinhardt Steven K et al. Communist, utilitarian, and capitalist cache policies on CMPs: Caches as a shared resource//Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques. Seattle, Washington, USA, 2006: 13-22

[15] Kim Kyunghwa, Shim Junho et al. Cache conscious trees: How do they perform on contemporary commodity microprocessors?//Proceedings of the 2007 International Conference on Computational Science and Its Applications. Kuala Lumpur, Malaysia, 2007: 189-200



GUO Chao, born in 1985, Ph. D. candidate. His current research interests focus on main memory database indices.

LI Kun, born in 1982, Ph.D., assistant researcher. His current research interests focus on main memory database.

WANG Yong-Yan, born in 1978, senior engineer. His current research interests include data mining and real-time database.

LIU Sheng-Hang, born in 1980, M. S., engineer. His current research interests focus on main memory database.

WANG Hong-An, born in 1963, researcher, Ph. D. supervisor. His current research interests include real-time intelligence, real-time database, human-computer interaction technology.

Background

This research is supported by the National High Technology Research and Development Program (863 Program) of China under grant No. 2008AA04A105.

As the development of semiconductors and integrated circuits technologies, the volume of the main memory becomes larger and larger. It has a profound impact on database management system as it will be possible, and in some cases, to store the entire database in main memory. In 80s of 20th century, researchers proposed a new concept called "Main Memory Database". In main memory database system, the most important difference from traditional database is that the data one transaction or all the transactions need is resident in main memory. And just the new feature of main memory database gives new challenges to database system designs in storage, indices and so on. Index effectively affects the performance of database. Until now, researchers have found out varieties of novel indices for main memory database, for instants, T-tree, CSS-tree, CSB-tree, etc.

In recent years, multi-core is much more popular in Processor Manufacturing, it helps to improve the performance of indices and also creates new challenges. In multi-core processors, different cores share the secondary cache and other resources, and this brings the competition for resources. We have already known that how traditional indices perform in single-core processors. In our research, we choose some popular indices and use series of experiments to show that how these indices perform and find out the features and reasons which affect the performance in multi-core processors, we also give some useful advices about index improvement in future work from the results.