面向批量插入优化的并行存储引擎 MTPower

陈 虎[1)] 唐海浩[2)] 廖江苗[1)] 彭江峰[2)]

[1)](华南理工大学软件学院 广州 510006)

[2)](华南理工大学计算机科学与工程学院 广州 510006)

摘 要 针对多核处理器的特点,文章提出了一个符合 MySQL 接口标准的并行存储引擎 MTPower. 该存储引擎着重利用多核处理器的并行计算能力提升批量插入过程中的索引产生过程,主要包含存储引擎接口、并行批量线性 Hash 索引、并行批量 B^+ 树插入、支持并行访问的磁盘存储缓冲等部分. 测试结果表明,在批量插入记录且需要创建 Hash 和 B^+ 树索引时,MTPower 的性能比经典的单线程存储引擎 MyISAM 最高可以提高 6.1 倍和 4.8 倍;在系统中线程总数略大于处理器核数时,MTPower 可以达到最佳性能;在处理器核的数量增加时,MTPower 的性能也能随之提高.

关键词 并行数据库;Hash 并行索引;B^+ 树并行索引;磁盘缓冲;多核处理器

中图法分类号 TP311 **DOI 号**: 10.3724/SP.J.1016.2010.01492

MTPower: A Parallel Database Storage Engine for Batch Insertion

CHEN Hu[1)] TANG Hai-Hao[2)] LIAO Jiang-Miao[1)] PENG Jiang-Feng[2)]

[1)](*School of Software Engineering, South China University of Technology, Guangzhou* 510006)

[2)](*School of Computer Science and Engineering, South China University of Technology, Guangzhou* 510006)

Abstract This paper presents a parallel MySQL storage engine, MTPower, which focuses on using parallel computing capacity of multi-core processors to improve index insertion in batch insertion. MTPower includes storage engine interface, parallel batch linear hash index, parallel batch B^+ tree index and disk buffer. The test results indicate that MTPower performances are 6.1 times and 4.8 times than MyISAM, a traditional single-threaded storage engine in MySQL, at best in testing batch insertion with hash index and B^+ tree index. MTPower can achieve the best performance, when the total number of threads is close to the core number of processors. With the increase of the number of processor cores, the performance of MTPower increases subsequently.

Keywords parallel database; parallel Hash index; parallel B^+-tree index; disk buffer; multi-core processors

1 引 言

随着多核时代的来临,如何充分利用多核处理器提升数据库系统的处理能力是一个非常有意义的课题. MySQL 是当前最流行的开源关系数据库管理系统之一,它采用插件式存储引擎架构[1-2],分为两层:SQL 层与存储引擎. 其中,SQL 层负责查询的

收稿日期:2010-06-11. 本课题得到广东省基础软件与应用构建实验室支持、广东省科技计划项目基金(2006B80407001)及华南理工大学中央高校基本业务费项目基金(2009ZM0007)资助.**陈 虎**,男,1974 年生,博士,副教授,研究方向为高性能计算、数据库系统、可信计算. E-mail: tommychen74@yahoo.com.cn. **唐海浩**,男,1985 年生,硕士,主要研究方向为并行计算、数据库系统和网络存储. **廖江苗**,男,1985 年生,硕士研究生,主要研究方向为数据库系统和并行计算. **彭江峰**,男,1985 年生,硕士研究生,主要研究方向为高性能计算.

解析、优化与执行等；存储引擎负责数据存储、索引管理、锁、事务管理、恢复等. MySQL 的数据库管理员可以选择最接近自身应用特点的存储引擎，或者定制专用存储引擎，以在特定需求下尽可能达到最佳性能. 本文提出的存储引擎 MTPower 是一种针对多核处理器平台上大批量数据插入优化的 MySQL 存储引擎.

MTPower 基于能进行自动负载均衡的 MSI 并行编程模型[3]开发，支持 Hash 索引和 B^+ 树索引的并行更新，包含能支持多个线程并行访问的磁盘缓冲和内存缓冲，可以在多核处理器平台上有效开发记录批量插入过程中的并行性.

本文第 2 节对 MySQL 常用的存储引擎进行性能分析，找出其不足之处；第 3 节介绍并行存储引擎 MTPower 的总体结构；第 4 节和第 5 节分别介绍针对多核处理器系统的批量并行线性 Hash 和 B^+ 树索引更新算法；第 6 节介绍支持并行访问的磁盘缓冲区；第 7 节给出并行存储引擎的性能测试结果与分析；最后一节为总结.

2　常见存储引擎性能分析

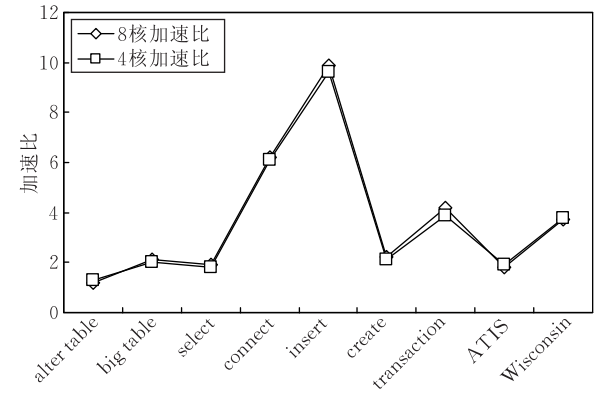
MySQL 有 MyISAM、InnoDB、Merge、Heap、Falcon 等多个常用的存储引擎. 本节将测试和分析 MyISAM 和 Falcon 存储引擎的主要性能. 其中，MyISAM 为单线程的存储引擎，是当前 MySQL 应用最为广泛的存储引擎；Falcon 为多线程的存储引擎，包含 4 个工作线程[4-5]，可以更有效发挥多核处理器的性能优势.

系统测试的平台如表 1 所示. 测试工具采用 MySQL 自带的 bench 套件，包含了 9 个测试，可以测试表的修改、插入、查询、创建和事务等方面的性能. 图 1 给出了 Falcon 引擎和 MyISAM 引擎在 4 核和 8 核平台上相对单核 PC 的加速比.

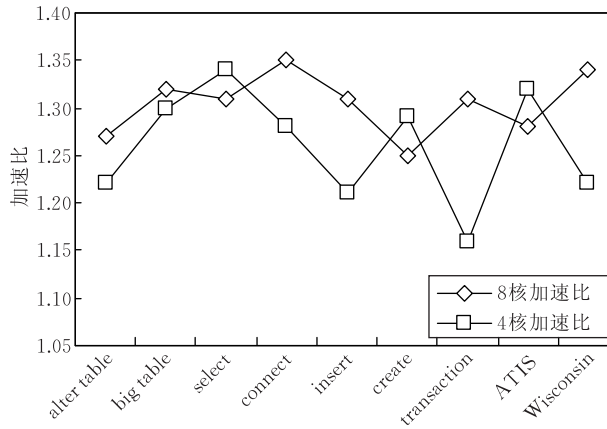
表 1　MySQL 存储引擎测试平台

	CPU	Cache	主存	操作系统
单核 PC	Intel Celeron	L1 16KB L2 256KB	256MB	Linux：2.6.21
4 核服务器	Xeon E5110 1.6GHz×2	L1 64KB L2 4MB	2GB	Linux：2.6.9-22
8 核服务器	Xeon E5310 1.6GHz×2	L1 64KB L2 4MB		

从图 1 可以看出，Falcon 存储引擎在 4 核处理器上的加速效果明显，但是 8 核处理器上的速度与 4 核相当. 这是由于 Falcon 引擎内部固定使用 4 个线程，当处理器核数超过 4 时，无法发挥更多处理器的能力. MyISAM 是单线程存储引擎，所以在 4 核与 8 核处理器上的性能没有显著提升，实验中的少许性能提升得益于多核系统的 Cache 容量比单核系统要大.



(a) Falcon在4核与8核处理器上的加速比对比



(b) MyISAM在4核与8核处理器上的加速比对比

图 1　Falcon 与 MyISAM 在 4 核与 8 核处理器上的加速比对比

为了进一步研究批量插入过程中索引数目对性能的影响，我们使用 KCachegrind 性能分析工具测试不同索引数目下插入 1 千万条记录时，MyISAM 存储引擎中负责记录与索引数据插入的 write_row 函数执行时间占总时间的比例，如图 2 所示.

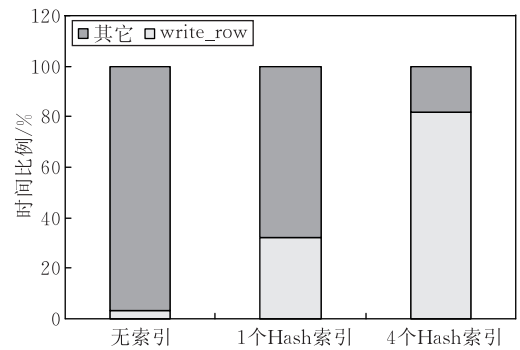


图 2　write_row 函数的执行时间百分比

可以看出，在没有索引的情况下插入 1 千万条

记录，write_row 的执行时间只占总执行时间的 3.5%，但是有 4 个 Hash 索引时，write_row 的执行时间比增加到 82%，可见更新索引对批量插入数据的性能具有重要的影响. 从上述实验结果，可以发现：

(1) Falcon 存储引擎的线程数目固定，导致在 4 核与 8 核处理器上性能相差无几，可扩展性有限；

(2) MyISAM 是单线程存储引擎，处理器核数的增加并不能带来性能提升；

(3) MyISAM 中更新索引是批量记录插入过程的主要性能瓶颈.

本文提出的多核存储引擎 MTPower 正是针对 MySQL 已有存储引擎的缺点，着重提升批量记录插入的索引产生性能，同时提升系统的可扩展能力.

3 MTPower 总体结构

MTPower 存储引擎采用并行编程模型 MSI 多线程调度接口[3-4]作为系统的基础. MSI 模型以任务池作为系统基本部件，由内部线程调度器根据各个任务池的负载情况动态分配执行线程. MSI 中的任务池可以根据任务之间的依赖关系连接成流水线或其它结构.

MTPower 存储引擎内部结构如图 3 所示. 其外部接口为 MySQL 的标准存储引擎接口，内部包含：支持并行分配和释放的多线程内存管理器；支持多线程并行访问的磁盘缓冲区；支持批量索引并行更新的任务池.

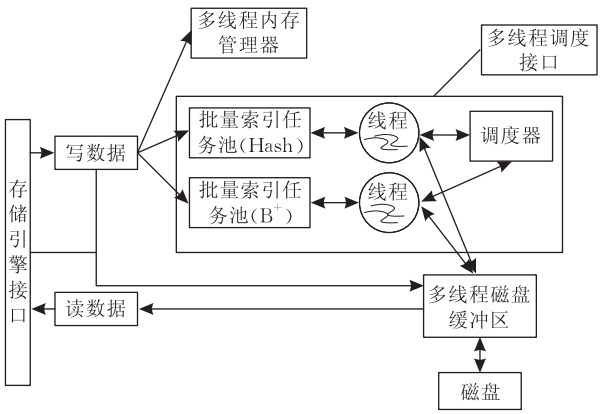


图 3 MTPower 总体结构

在 MTPower 中，插入记录的过程采用基于 MSI 任务池的方法，其主要流程包括：

(1) 从存储引擎接口接收 MySQL 的 SQL 层传递过来要插入的记录数据，分析出记录中的索引字段数据，从内存管理器申请内存块保存需要索引字段的数据，并将内存地址写进批量索引任务池，最后将整条记录写入磁盘缓冲区；

(2) 在批量索引任务池的记录达到了一定数量后，采用并行方式更新 Hash 索引或 B^+ 树索引，并将结果保存到多线程磁盘缓冲区.

上述索引更新过程异步于 SQL 层的记录插入过程，使得 SQL 层的记录插入操作非常简单，从而有效提高了其插入记录的速度，并使得 SQL 层和索引更新过程可以在多核处理器平台上并行执行.

4 批量并行线性 Hash 索引

线性 Hash 索引[5]是一种动态 Hash 方法，其 Hash 表可以随着记录项的插入或删除而扩张或收缩，而且随着 Hash 表的桶数目变化，Hash 函数也随之改变，这种动态处理方法不仅解决了空间利用率问题，而且有效地降低了地址冲突. 并发的线性 Hash 操作由 Ellis 在 1987 年提出[6]，其目的是为了增强线性 Hash 的并发访问操作. Garcia-Monlina 等研究了内存数据库中 Hash 表的访问操作等问题[7].

本文在原有串行线性 Hash 索引更新算法的基础上提出了并行化的批量 Hash 索引更新算法. 其主要特点是：不立刻将单个索引记录插入到 Hash 表中，而在内存中保留这些记录直到达到一定数量再进行批量并行插入.

假定 T 为一次批量插入的记录总数，K 为分裂桶的阈值，b 为桶的容量，B 为当前桶的总数，那么对于 T 条记录的插入应该增加的桶数目为 $B_T = T/(K \times b)$. 根据当前待分裂桶的位置 P_0，有两种情况，如图 4 所示.

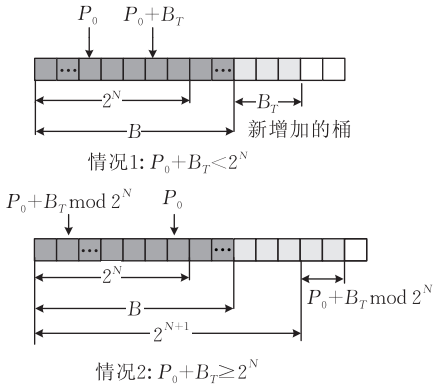


图 4 并行批量 Hash 索引更新算法

情况 1. 此时 $P_0+B_T<2^N$，相应算法如下.

(1) 使用一个线程分裂第 P_0 号桶到第 P_0+B_T-1 号桶，其中 P_0+i 号桶的元素分裂到 P_0+i

号桶和 $B+i$ 号桶($i=0,\cdots,B_T-1$)；

(2) $P_0=P_0+B_T$；

(3) 所有线程并行地将 T 条记录插入 Hash 表的第 0 号桶到第 $B+B_T-1$ 号桶.

情况 2.　此时 $P_0+B_T\geq 2^N$，相应算法如下.

(1) 使用 Hash 函数 H_{N+1} 分裂 P_0 到 2^N-1 号桶，使用 Hash 函数 H_{N+2} 分裂第 0 号到第 $((P_0+B_T) \bmod 2^N)-1$ 号桶；

(2) $P_0=(P_0+B_T) \bmod 2^N$；

(3) 所有线程并行地将 T 条记录使用 Hash 函数 H_{N+1} 插入 Hash 表的第 0 号桶到第 $2^{N+1}+((P_0+B_T) \bmod 2^N)-1$ 号桶.

Hash 函数由式(1)[5]给出，其中 M 为初始桶数目：

$$H_N(Key)=\begin{cases}Key \bmod (2^N M) & (Key \bmod 2^N \geq P)\\ Key \bmod (2^{N+2} M) & (Key \bmod 2^N < P)\end{cases} \quad (1)$$

在 T 个记录并行插入过程中，通过磁盘缓冲区的锁避免插入相同的桶时而产生的互斥问题.

批量并行 Hash 更新算法相比传统串行算法有如下优势：

(1) 预分裂算法减少很多冗余的分裂操作. 在串行插入过程中，记录 r 的初始位置是第 b 号桶. 桶 b 分裂后，r 可能要转放到 $b+M$ 号桶中；同理，$b+M$ 号桶分裂后，记录 r 可能又要转放到 $b+2M$ 号桶中. 以此类推，经过 i 次分裂之后，记录 r 可能最终是放入 $b+iM$ 号桶中. 在批量插入算法中，记录 r 可以一次性定位到第 $b+iM$ 号桶中；

(2) 多线程并行插入记录可以充分利用多核处理器的并行计算能力.

5　批量并行 B$^+$ 树插入

B$^+$ 树是一种适用于磁盘数据库系统的索引机制. B$^+$ 树的每个结点有多棵子树，形状宽而浅，从而只需较少次数的磁盘 I/O 就可以找到目标数据[8]. 但是 B$^+$ 有一个缺点：并发度低. 因此，Lehman 和 Yao 提出了一种 B$^+$ 树的变形——B^{link} 树结构，它具有极高的同步操作性能[9]. 由于 B^{link} 树也是 B$^+$ 树的一种，为方便起见，本文把 B^{link} 树也称为 B$^+$ 树.

当前，人们对 B$^+$ 树索引的并行化操作也作了相关研究. Taniar 和 Rahayu 研究了 B$^+$ 树在分布式系统上的索引处理模式[10]，主要关注在索引的划分和存储方面. Jaluta 等研究了平衡 B^{link} 树下的并发控制方面[11]，其侧重点主要在索引的恢复操作和事务处理. 丁华等对海量数据的索引构建方法进行了研究[12]，其主要特点是自底向上构建 B$^+$ 树，该方法有效提升了数据库索引的建立效率. 本文在 B^{link} 树的算法基础上实现了基于任务池的并行 B^{link} 树的并行批量插入.

通常构建索引树的方法是从空树开始，使用标准的插入算法[13]为每条数据记录插入索引项. 这种方法一次只能插入一个索引项，每次都需要从根开始，一直查找到合适的叶子页进行插入，处理海量数据的效率很低.

本文提出的批量插入算法的基本思想是：先将待插入索引记录预排序，从根结点开始用根结点的 key 将预排序的结果“分割”成多个子序列，并把各子序列插入对应的内结点，层层分割，直至叶结点. 这样就通过根结点中的关键字把待插入的已排好序的索引记录分割为多个子序列. 每个子序列的插入过程就是一个独立的任务并由 B$^+$ 树任务池中的线程并行读取和执行，如图 5 所示.

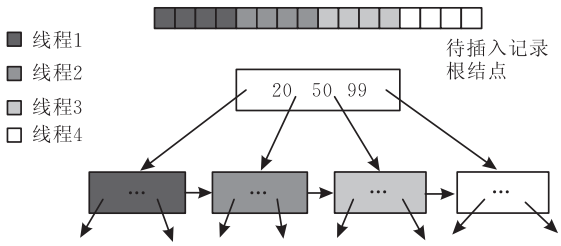


图 5　并行批量 B$^+$ 树插入算法

在 B$^+$ 树的存储结构中，每个结点由一个 4KB 大小磁盘页面存储，其中每页可以存储的最大索引项数为 r. 在线程对当前结点插入长度为 Y 的子序列时采用以下算法.

(1) 若当前结点的页面中不包括高键(每个结点的最小上界)的索引项数为 X，则

$$W=\begin{cases}X+Y, & (\text{当前结点为一层的最右边结点})\\ X+Y+1, & (\text{其它结点})\end{cases}$$

(2) 若 $W\leq r$，则直接将新的索引项插入到当前结点的页面中，算法结束；

(3) 若 $W>r$，求得分裂后的页面数 $K=\left\lceil \dfrac{W-1}{r-1} \right\rceil$；

(4) 从磁盘缓冲中分配 $K-1$ 个新缓冲页；

(5) 将当前结点的项和待插入的子序列重新排序；

(6) 将有序序列分配到 $K-1$ 个新缓冲页和原有的缓冲页中，并将这些页指针和每页高键值上传至父结点；

(7) 若父结点需要分裂，采用此算法继续递归分裂，直至不需分裂为止.

在初次建树时，也采用了批量插入的方法，即对初次插入的 N 个记录进行排序，然后将其直接划分到 L 个叶结点中，其中可根据预先设定的填充因子 $\alpha(0<\alpha<1)$ 来确定叶结点的个数：$L=N/\alpha r$. 由于初次建树过程中无结点分裂，提高了建库效率，且可通过填充因子调整存储空间利用率，可为后续插入数据预留了空间，减少分裂次数，提高插入效率.

6 磁盘缓冲管理

磁盘缓冲的目的是尽可能地将重要的数据保留在内存中，以减少对磁盘的访问. MTPower 中磁盘缓冲的最大特点是能够支持多线程的并发访问，尽量减少多核环境下线程访问的冲突，以发挥多核处理器的性能，其总体结构如图 6 所示. 其中主要包括：(1) 提供外部访问接口、管理磁盘块的磁盘缓冲区；(2) 支持并行内存分配、回收的内存管理器；(3) 保存磁盘缓冲区 I/O 请求的 I/O 任务池；(4) 文件句柄管理器.

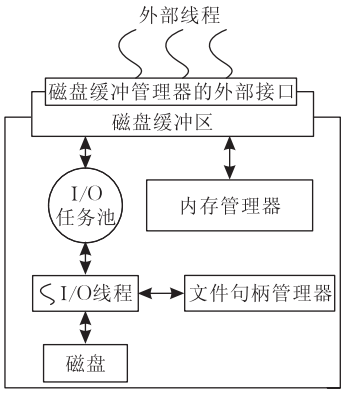


图 6 磁盘缓冲管理器的总体结构

磁盘缓冲的主要功能是完成用户线程的磁盘块读写操作请求. 在此过程中，将首先检查该数据块在磁盘缓冲中是否存在，如果已存在则直接返回给用户线程. 如果不在磁盘缓冲区内，磁盘缓冲区向 I/O 任务池写入一个对应的数据块读请求. 在 I/O 线程返回有效磁盘块前，用户线程等待请求的完成.

与 I/O 任务池相关的 I/O 线程有两个主要作用：处理 I/O 任务池的读写请求；定时扫描磁盘缓冲区的脏数据块并把它们以任务的形式写入 I/O 任务池，然后再处理 I/O 任务池的请求.

文件句柄管理器的作用是管理已经打开的文件句柄. 当磁盘缓冲打开文件的数量达到一定上限时，为了防止系统溢出[4]，由文件管理器根据先入先出的策略关闭最早打开的文件句柄.

磁盘缓冲区的内部结构如图 7 所示，通过文件名和文件中偏移量的散列值来查找相应的内存块. 此外，缓冲中所有内存块的运行时刻信息都以数组形式存储在 Head table 表中，其数据结构如图 8 所示.

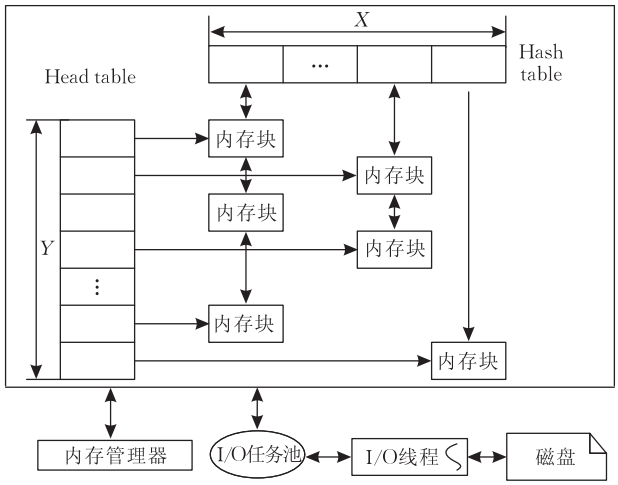


图 7 磁盘缓冲区的总体结构

```
typedef struct _buffer_head{
    u8 bh_dirt_flag:2; //内存块状态标志
    u8 bh_total_use:6; //使用次数
    u8 bh_valid:1; //是否有效
    u8 bh_reference:3; //线程引用计数
    u8 bh_filename_hash:3; //文件名散列
    u8 bh_reading:1; //是否正在读出
} buffer_head;
```

图 8 内存块头 Head table 数据结构

内存块数据的状态有 4 种：DIRTY、CLEAN、DISK_AND_DIRTY 与 DISK. 其中，DIRTY 代表内存块的数据修改过；CLEAN 代表内存块的数据没有修改过；DISK 代表内存块的数据正在写回磁盘；DISK_AND_DIRTY 代表内存块的数据在写回磁盘的过程中又被用户线程修改过. 状态转换如图 9所示.

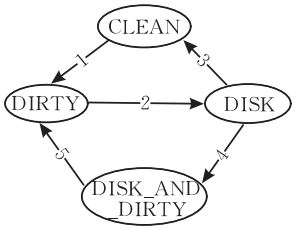


图 9 内存块数据状态转换图

对应转换的含义包括：

转换 1. 对内存块进行写操作，修改内存块的数据；

转换 2. 对脏内存块写回磁盘，脏内存块正处于等待 I/O 线程写回磁盘或者 I/O 线程正在将其写回磁盘，保存在 I/O 任务池中；

转换 3. 脏内存块成功写回磁盘，并且在写回的

过程中没有再次被修改；

转换 4. 脏内存块在写回磁盘的过程中被用户线程重新修改；

转换 5. 脏内存块写回磁盘后，因为写回过程中数据再次被用户线程修改，所以依然标示为脏内存块.

由于内存头数据结构的成员都用位表示，使得每个内存头只占 2 个字节，整个内存头数组仅占用 20KB(在 40MB 缓冲容量下，每个内存块大小为 4KB)，小于一般系统 L1 Cache 的容量. 因此在寻找可替换内存块时，可以大大提高 Cache 命中率，提高运行速度.

磁盘缓冲区采用了 LFU(最不常用使用法)置换策略. 当需要空闲内存块时，将替换内存块头中 bh_total_use 字段最小者(访问次数最少)的内存块. 脏内存块的替换时机有 3 个：

(1) 当用户线程向磁盘缓冲区申请空闲内存块，而磁盘缓冲区里没有干净的内存块时，激发脏内存块的置换过程；

(2) I/O 线程定时激发磁盘缓冲区的脏内存块置换；

(3) 关闭磁盘缓冲区时，磁盘缓冲区会把所有脏内存块写回磁盘.

7　实验结果与分析

对 MTPower 的性能测试分别在 4 核服务器与 8 核多核服务器上进行，其配置如表 1 所示. 图 10 和表 2 分别为测试所使用的数据表和不同记录数量和文件大小. 我们选用最常用的单线程非事务存储引擎 MyISAM 作为 MTPower 的性能比较对象. 实验的主要目标包括：(1) 分析数据量、磁盘缓冲容量、线程数对 MTPower 性能的影响，提供系统参数优化方案；(2) 与 MyISAM 进行性能对比；(3) 评价处理器核数增加时，MTPower 的可扩展能力.

```
CREATE TABLE t1 (col_a INT NOT NULL,
                 col_b INT NOT NULL,
                 col_c VARCHAR(12),
                 col_d VARCHAR(12),
                 col_e VARCHAR(12),
               INDEX index0(col_a) USING HASH)
                 ENGINE=MTPOWER;
```

图 10　测试表的定义

表 2　MTPower 的数据文件大小

插入记录数/万	数据文件大小/MB
50	31.1
100	62.5
150	92.4
200	118.5

数据量和存储缓冲容量对 MTPower 的性能影响

在此实验中，磁盘缓冲块大小固定为 4KB，容量分别设置为 4MB、8MB、20MB 和 40MB. 插入的记录数如表 2 所示. 图 11 给出了在创建一个 Hash 索引(其中 6 个 Hash 更新线程，367 个初始桶数，分裂因子为 0.75)情况下，MTPower 与 MyISAM 在 8 核服务器上的性能对比.

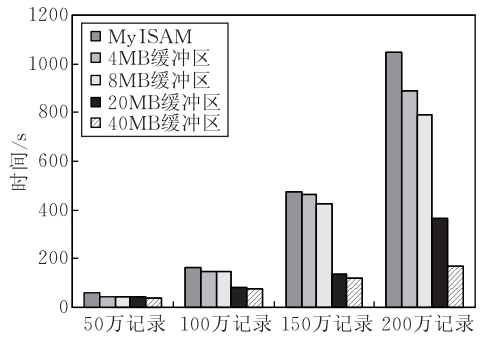


图 11　8 核服务器数据量和磁盘缓冲容量的性能影响(Hash 索引)

由图 11 可以看出，在插入记录数较小(50 万)和缓冲容量较小的情况下(4MB 或 8MB)，MTPower 的性能和 MyISAM 相近. 但是，当磁盘缓冲区容量较大时(20MB 或 40MB)，MTPower 的性能得到了较大的提升. 缓冲容量越大，插入记录的数据量越大，MTPower 性能提升越明显. 相比 MyISAM，其最大加速比可以达到 6.18. 这是因为缓冲容量越大，不仅能够减少磁盘的 I/O 次数，还能够更好地发挥并行批量索引的插入优势. 在后续的实验中，我们都设置磁盘缓冲容量为 40MB.

线程数对 MTPower 的性能影响

在使用 MTPower 存储引擎的 MySQL Server 系统中包括了 MySQL Server 主线程、连接线程、MSI 调度器线程、I/O 线程等 4 个固定线程以及 N 个并行索引插入的线程，即系统的总线程数为 $N+4$. 在 4 个固定线程中，MySQL Server 主线程处于睡眠状态等待新的连接进来唤醒，I/O 线程只有 I/O 任务池有任务或者定时扫描磁盘缓冲区脏内存块才处于运行状态，MSI 调度器线程也是定时唤醒.

图 12 和图 13 分别给出了 4 核与 8 核服务器上不同线性 Hash 索引插入线程数对 MTPower 存储引擎的性能影响，图 14 给出了 8 核服务器上不同 B^+ 树索引插入线程数对 MTPower 的影响以及其与 MyISAM 的性能比较.

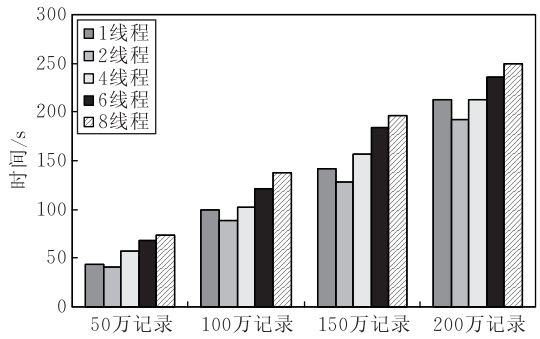


图 12 4 核服务器上不同线程数的性能影响(Hash 索引)

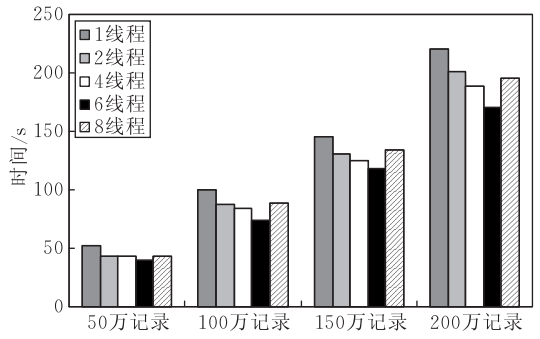


图 13 8 核服务器上不同线程数的性能影响(Hash 索引)

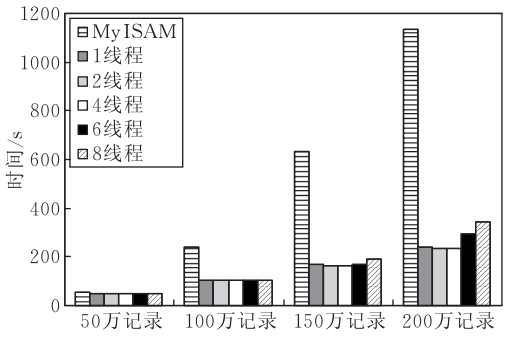


图 14 8 核服务器上不同线程数的性能影响(B^+ 树索引)

从上述 3 个图中可以看出,在 4 核服务器上 2 个索引插入线程的性能最好,在 8 核服务器上 4 或 6 个索引插入线程的性能最好. 考虑到系统中 4 个固定线程(其中有 3 个为不活跃线程,1 个为活跃线程),可以得到并行索引线程数的优化配置方案:在总的线程数略大于处理器的核数时,MTPower 的性能最优. 总线程数过小,不能为多个核提供足够的计算负载;总线程数过多,则会因为过多的线程切换开销导致性能反而下降.

此外,从图 14 中也可以看出,MTPower 在 B^+ 树索引方面的性能也较 MyISAM 有较大提高,在大数据量情况下,加速比最大可以达到 4.8 倍.

扩展性测试

图 15 给出了最佳参数设置时,MTPower 在 4 核和 8 核服务器上 Hash 索引的性能对比. 4 核和 8 核服务器上的索引插入线程数分别为 2 个和 6 个.

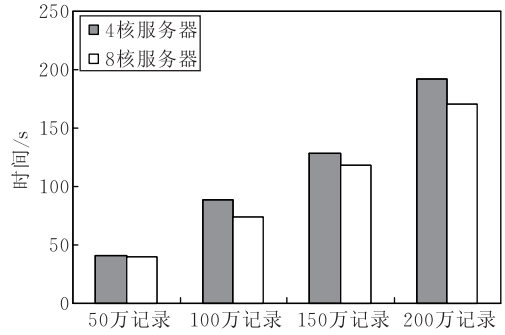


图 15 在 4 核和 8 核服务器上的性能对比(Hash 索引)

从图 15 可以看出,在插入相同数据量时,8 核处理器的性能要比 4 核处理器的性能高,而且随着数据量的增加,性能提升更为显著. 在 200 万条记录数量情况下,MTPower 在 8 核服务器上的性能提升了约 12%,具有一定的可扩展能力.

我们还在 4 核服务器平台上比较了 4 线程存储引擎 Falcon 和 MTPower 在插入 200 万条记录,创建一个 B^+ 树索引(Falcon 不支持 Hash 索引)的性能,其中 Falcon 的插入时间为 203s,MTPower 为 225s,两者基本相当.

8 结 语

作为当今最为流行的开源数据库之一,MySQL 采用了插件式存储引擎架构. 但 MySQL 已有的存储引擎大部分只是以单线程的方式处理数据,不能充分利用多核处理器计算能力. 本文实现了一个符合 MySQL 存储引擎接口规范,支持多核处理器并行计算的存储引擎 MTPower. 该存储引擎总体结构基于多线程动态调度机制,支持 Hash 索引和 B^+ 树索引的并行批量插入,同时包含了磁盘缓冲,以减少系统 I/O 开销.

测试结果表明,与经典单线程存储引擎 MyISAM 相比,MTPower 在包含单个 Hash 索引和 B^+ 树索引的批量插入时,性能最高可以提高 6 倍和 4.8 倍. 同时,通过实验也找到了最佳线程数的配置方案:系统中的线程总数应略大于处理器核数. 此外,通过对 MTPower 在 4 核和 8 核服务器上的性能对比,说明其具有一定的可扩展能力. 当然 MTPower 也有一些缺陷,例如可扩展性还不够强、尚不支持并行的查询等,这些将是进一步的研究目标.

参 考 文 献

[1] Bell C A. Expert MySQL. New York: Springer-Verlag New York Inc., 2005

[2] Pachev S. Understanding MySQL Internals. O'Reilly & Associates Inc., 2005

[3] Peng Jiang-Feng, Chen Hu, Xi Jian-Qing. MSI a new parallel programming model//Proceedings of the 2009 WRI World Congress on Software Engineering. Xiamen, China, 2009: 56-60

[4] Stevens W R, Rago S A. Advanced programming in the UNIX environment. 2nd Edition. New Jersey: Addison-Wesley, 2008

[5] Litwin W. Linear hashing: A new tool for file and table addressing//Proceedings of the 6th International Conference on VLDB. Montreal, Quebec, Canada, 1980: 212-223

[6] Ellis C S. Concurrency in linear hashing. ACM Transactions on Database Systems, 1987, 12(2): 195-217

[7] Garcia-Monlina H, Lipton R, Valdes J. A massive memory machine. IEEE Transaction on Computer, 1984, 33(5): 391-399

[8] Lehman T J, Carey M J. A study of index structures for main memory database management systems//Proceedings of the 12th International Conference on Very Large Database. San Francisco, CA, USA, 1986: 297-302

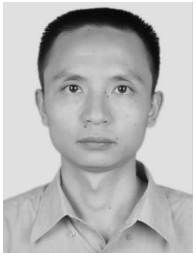
[9] Lehman P L, YAO S B. Efficient locking for concurrent operations on B-trees. ACM Transactions on Database Systems, 1981, 6(4): 650-670

[10] Taniar D, Rahayu J W. A taxonomy of indexing schemes for parallel database systems. Distributed and Parallel Databases, 2008, 3(1): 79-90

[11] Jaluta I, Sippu S, Soisalon-Soininen E. Concurrency control and recovery for balanced B-link trees. The International Journal on Very Large Data Bases, 2005, 14(2): 257-277

[12] Ding Hua, Liao Xue-Jun, Zhang Zhi-Wei, Wang Rong-Feng. Study on index constructing method based on mass attribute data. Journal of the Academy of Equipment Command and Technology, 2005, 16(6): 83-87(in Chinese)
(丁华，廖学军，张志威，汪荣峰. 基于海量属性数据的索引构建方法研究. 装备指挥技术学院学报，2005，16(6)：83-87)

[13] Ramakrishnan R, Gehrke J. Database Management Systems. 3rd Edition. New York: McGraw-Hill, 2003



CHEN Hu, born in 1974, Ph. D., associate professor. His current research interests include high performance computing, database systems and trusted computing.

TANG Hai-Hao, born in 1985, M. S.. His current research interests include parallel computing, database systems and network storage.

LIAO Jiang-Miao, born in 1985, M. S. candidate. His current research interests include database systems and parallel computing.

PENG Jiang-Feng, born in 1985, M. S. candidate. His current research interests include high performance computing and parallel computing.

Background

This research is partly supported by the Guangdong Science and Technology Plan under grant No. 2006B80407001, and the Fundamental Research Funds for the Central Universities, SCUT under grant No. 2009ZM0007.

How to use multi-core processors to enhance performance of database systems is an important research topic in database field. As one of the most popular open source database systems, MySQL has many storage engines to manage data and index storage, transaction and recovery. However, the existing storage engines always adopt single thread model, like MyISAM, or fixed threads number, like Falcon. In such storage architecture, it is hard to exploit potential parallel computing capacity of multi-core processors.

We developed a multi-threaded storage engine for MySQL, MTPower, which is optimized for batch index insertion. Based on multithreaded dynamic scheduler interface (MSI), MTPower has strong scalability on multi-core processors. This storage engine mainly includes storage engine interface, parallel batch linear hash index, parallel batch B^+ tree index and disk buffer. The storage engine interface totally follows MySQL storage engine interface standard. Parallel batch linear hash and B^+ tree index can effectively use parallel computing capacity to improve performance of index insertion in batch record insertion. The disk buffer also supports multi-threaded parallel access and LFU replace strategy.

As a parallel storage engine on the multi-core processor systems, MTPower can enhance performance of MySQL, especially for batch record insertion.