第26卷 第3期
2011年6月

郑州轻工业学院学报(自然科学版)
JOURNAL OF ZHENGZHOU UNIVERSITY OF LIGHT INDUSTRY(Natural Science)

Vol. 26 No. 3
Jun. 2011

文章编号:1004-1478(2011)03-0092-04

# 动态散列算法及其改进

李蔚, 陈亚峰, 王艳军

(郑州轻工业学院 计算机与通信工程学院, 河南 郑州 450002)

**摘要**:对2种动态散列算法可扩展散列和线形散列进行研究,提出了允许散列后缀不等长的改进动态散列算法.改进后的动态散列算法不会产生不必要的溢出桶,散列桶的数量因而呈现线性增长,避免了因查找键分布异常而出现频繁的桶分裂及桶地址表更新现象的出现.模拟实验表明,改进后的动态散列算法明显优于可扩展散列和线性散列.

**关键词**:动态散列;可扩展散列;线性散列

**中图分类号**:TP319 **文献标志码**:A

## Dynamic hashing and its improvement

LI Wei, CHEN Ya-feng, WANG Yan-jun

(*College of Comp. and Com. Eng., Zhengzhou Univ. of Light Ind., Zhengzhou* 450002, *China*)

**Abstract**: Extensible hashing and linear hashing were discussed, and an improved algorithm for the hashing suffix length inequality was stated, which avoided unnecessary overflow bucket. The number of hash buckets grow linearly, which avoid splitting buckets and updating bucket address table continually, caused by unusual distribution of search key. The experiments of the simulation method showed that the improved algorithm was significantly better than extensible hashing and linear hashing.

**Key words**: dynamic hashing; extensible hashing; linear hashing

## 0 引言

散列方法是信息存储和查询时使用的一项基本技术,在动态变化的文件特别是数据库中,基于动态散列的文件结构得到了广泛的应用[1].数据处理的主要操作是查找、插入、删除、更改,这就使得数据的存储尤为重要.散列方法是一种基于散列函数的文件构造方法,可实现对记录的快速随机存取,主要是利用给定的查找键 *KEY* 映射到相应的存储单元[2].笔者拟针对可扩展散列和线性散列方法的不完善之处提出自己的改进算法,并进行实验验证.由于可扩展散列方法和线形散列方法已相当成熟,近几年国内外很少再有人研究,但是在嵌入式数据库系统SQL编译器中仍要解决该问题,所以对散列算法的研究有一定的意义.而关于动态散列改进算法国内外尚未见报道.

## 1 可扩展散列方法

在动态散列中选择一单向函数:$key = \mathrm{Hash}(KEY)$,散列值 *key* 为查找键 *KEY* 对应的一个 *K* 位二进制序列.可扩展散列一般结构如图1所示,它由桶地址表和桶组成.桶地址表是一个指向桶的指针数组,

收稿日期:2010-11-19

基金项目:河南省科技厅攻关项目(0424220008)

作者简介:李蔚(1958—),男,河南省驻马店市人,郑州轻工业学院教授,博士,主要研究方向为数据库与信息集成、内存数据库.

桶是存放记录的数据块,通常1个桶就是1个磁盘块,可以存放1条或多条记录,本文假设每个桶最多存放2个记录. 桶地址表上方的 $i$ 表示散列值中有 $i$ 位需要用来正确地定位对应于 $KEY$ 的桶,这个值随着文件的变化而变化. 几个连续的表项可能指向同一个桶,所有这样的表项有一个共同的散列前缀,这个前缀的长度可能小于 $i$. 每一个桶 $j$ 附加一个整数值 $i_j$ 表示共同的散列前缀长度[3].

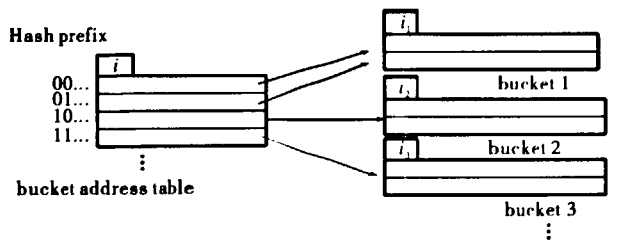


图1　可扩展散列的一般结构

### 1.1　可扩展散列的查找

为了进行一次基于查找键 $K_i$ 的记录的查找,计算 $\mathrm{Hash}(K_i)$ 并确定具有该地址的桶,比如桶 $j$. 假定查找键 $K_m$ 和 $K_n$ 具有相同的散列值,即 $\mathrm{Hash}(K_m)=\mathrm{Hash}(K_n)$,或者 $\mathrm{Hash}(K_m)$ 和 $\mathrm{Hash}(K_n)$ 的前 $i$ 位二进制序列相同. 如果执行对 $K_m$ 的查找,则桶 $j$ 包含的查找键是 $K_m$ 和 $K_n$ 的记录,故必须检查桶中每条记录的查找键值,以确定该记录是否为要查找的记录.

### 1.2　可扩展散列的插入

要插入一个查找键为 $K_i$ 的记录,首先按照1.1所述过程进行查找,最终定位到某个桶(比如桶 $j$). 如果该桶有剩余空间,将该记录插入桶 $j$ 即可;如果桶 $j$ 已满,系统必须分裂这个桶并将该桶中现有记录和新记录一起进行重新分布. 为了分裂该桶,首先根据 $i$ 和 $i_j$ 的关系确定是否增加 $i$ 的位数[4].

1)如果 $i=i_j$,在桶地址表中只有1个表项指向桶 $j$. 这时需要增加桶地址表的大小,$i$ 自加1,桶地址表翻倍. 现在桶地址表中有2个表项指向桶 $j$,这时,系统分配一个新的桶 $z$,并让第2个表项指向此新桶,将 $i_j$ 和 $i_z$ 置为 $i$. 接下来,桶 $j$ 中的各条记录被重新散列,根据前 $i$ 位来确定该记录是放在桶 $j$ 中还是放到新创建的桶 $z$ 中. 系统再次尝试插入该新记录,通常这一尝试会成功. 如果桶 $j$ 中原有的所有记录和新插入的记录具有相同的散列值前缀,该桶就必须被再次分裂,这样的分裂有时要进行多次才能完成插入. 如果桶 $j$ 中所有记录搜索码和要插入的记录搜索码相同,那么多少次分裂也不能解决问题. 这种情况下采用溢出桶来存储记录.

2)如果 $i>i_j$,那么在桶地址表中有多个表项指向桶 $j$. 这时系统不需要增加桶地址表的大小就能分裂桶 $j$. 如果指向桶 $j$ 的所有表项的索引前缀的最左 $i_j$ 位相同,系统就分配一个新桶 $z$,将 $i_j$ 和 $i_z$ 置为原 $i_j$ 加1后得到的值. 接下来系统需要调整桶地址表中原来指向桶 $j$ 的表项,系统让这些表项的前一半保持原样(指向桶 $j$),而使后一半指向新创建的桶 $z$. 桶 $j$ 中的各条记录被重新散列,分配到桶 $j$ 或新桶 $z$ 中. 此时,系统重新尝试插入记录,方法同1).

### 1.3　可扩展散列的删除

要删除一个查找键为 $K_i$ 的记录,系统按照1.1所述方法找到相应的桶,把记录从文件中删除. 如果这时桶成为空的,那么桶也需要被删除[5]. 此时某些桶可能被合并,桶地址表的大小也可能减半.

### 1.4　可扩展散列的更新

要更新一个搜索码值为 $K_i$ 的记录,系统可以按前面的查找过程找到该记录,更新该记录即可. 更新操作不涉及桶的分裂和合并,桶地址表也不发生任何变化.

## 2　线性散列方法

与可扩展散列不同,线性散列[6]的桶地址表是线性增减的. 线性散列的一般结构如图2所示,它使用散列值的后 $i$ 位,$i$ 表示散列后缀的长度,$n$ 表示散列地址表的表项数(表项和桶是一一对应的,故所有的散列后缀长度都相同),$r$ 表示所有桶中的记录总数. 表项数 $n$ 的选择桶空间的利用率不超过某一个常数,比如80%. 由于桶并不总是可以分裂,所以允许溢出桶. 散列后缀的长度 $i=\lceil\log_2 n\rceil$. 设查找键为 $K$ 的记录的散列值的后 $i$ 位二进制为 $a_1a_2\cdots a_i$,它的十进制值为 $m$. 插入该记录时,如果 $m<n$,则插入编号为 $m$ 的表项所指向的桶;如果 $n\leqslant m<2^i$,即编号为 $m$ 的表项还不存在,把记录插入编号为 $m-2^{i-1}$ 的表项所对应的桶中.

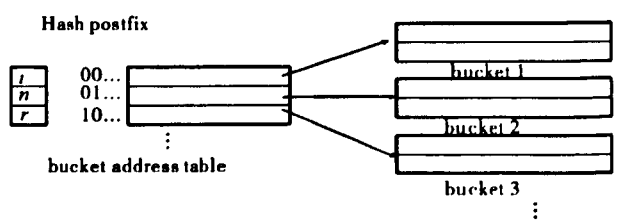


图2　线性散列的一般结构

### 2.1　线性散列的查找

线性散列的查找与可扩展散列类似,唯一不同的一点是,当 $n\leqslant m<2^i$ 时,记录在编号为 $m-2^{i-1}$ 的

表项所对应的桶中查找.

### 2.2 线性散列的插入

要插入一个查找键为 $K_i$ 的记录,按照2.1所述过程进行查找,最终定位到某个桶(比如桶 $j$). 如果该桶有剩余空间,将该记录插入桶 $j$ 即可;如果桶 $j$ 已满,就创建一个溢出桶,并把它链接在桶 $j$ 上,记录就存入该溢出桶中. 每次插入后都要计算当前的桶空间利用率,如果大于规定值,并且 $n<2^i$,就直接增加下一个表项到桶地址表,它指向一个新增加的桶. 新增加的桶与发生插入的桶之间没有任何联系. 如果新增加的表项为 $1a_2\cdots a_i$,就分裂表项 $0a_2\cdots a_i$ 所指向的桶,让该桶(及其溢出桶,如果有的话)中的记录在2个表项所指向的桶之间重新分配;若 $n=2^i$,首先 $i$ 递增1,所有的表项前面都增添一个0,然后增加下一个表项到桶地址表,后面的操作和 $n<2^i$ 相同.

### 2.3 线性散列的删除和更新

线性散列的删除和更新与可扩展散列基本相同,只不过删除桶时,该桶对应的桶地址表表项也要被删除.

## 3 改进的动态散列算法

对于动态散列,当桶地址表翻倍时,要做大量的工作,使某些插入花费很长时间,并且会阻止对数据文件的访问. 当 $i$ 较大时,桶地址表翻倍后可能在主存装不下,数据访问所需的磁盘I/O也翻倍,性能明显下降. 有时一次插入会导致桶地址表的多次翻倍,甚至 $i$ 从1连续增加到32才能完成插入,这时桶地址表表项从2变为 $2^{32}$(40亿),尽管存放记录的桶只有3个. 对于线性散列,频繁出现的溢出桶,会使磁盘I/O增加. 由于 $1a_2\cdots a_i$ 和 $0a_2\cdots a_i$ 对应的记录可能在同一个桶中,会增加查找插入的开销. 如果连续插入若干个散列值的后 $i-1$ 位都相同的记录,这些记录要放在同一个桶及其溢出桶中. 如果这样的记录数量较大,会出现很多溢出桶,桶地址表表项也会增加许多,新增加的桶与发生插入的桶之间没有任何联系,桶的分裂不能有效解决溢出桶问题. 为了解决上述问题,提出了改进的动态散列算法. 改进算法对溢出的处理和可扩展散列相同,1次插入最多增加1个桶地址表表项,并且记录的重新散列只发生在新增加的桶和发生插入的桶之间,其余的桶和表项不发生任何变化,它的一般结构如图3所示. 为了方便,假设每个记录散列值的长度都是4位. 它的桶地址表是个结构体数组,每个表项由2个域组成. 1个域存放1个长度为 $i$ 的二进制序列,它是正在使用的散列值的后 $i$ 位,不同表项的 $i$ 值可以不同. 另一个域是一个指针,指向存放记录的桶. 散列值的后 $i$ 位相同的记录存放在同一个桶中. 只有散列值完全相同的记录一个桶容纳不下时,才允许溢出桶的出现.

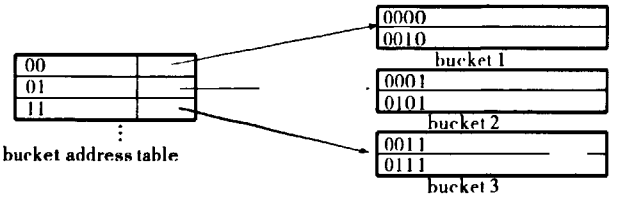


图3 改进的动态散列的一般结构

### 3.1 改进算法的查找

每一个记录的查找键的散列值都是相同长度的二进制序列. 为了进行一次基于查找键 $K_i$ 的记录的查找,计算 Hash($K_i$),然后与桶地址表中的二进制序列进行匹配. 若某表项中二进制序列的长度为 $i$,它正好与 Hash($K_i$) 的后 $i$ 位完全匹配,则此表项指向的桶就是要查找的桶.

### 3.2 改进算法的插入

要插入一个查找键 $K_i$ 的记录,按照3.1所述过程进行查找,假设找到的桶地址表表项中二进制序列长度为 $i$,最终定位到某个桶(比如桶 $j$). 如果该桶有剩余空间,将该记录插入桶 $j$ 即可;如果桶 $j$ 已满,并且桶 $j$ 中所有记录的散列值和要插入的记录的散列值完全相同,就要创建溢出桶,将该记录插入溢出桶,并将溢出桶链接在桶 $j$ 后面. 否则,假设欲插入的记录和桶 $j$ 中所有记录散列值的后 $k(k\geq i)$ 位二进制 $a_1a_2\cdots a_k$ 完全相同,则把原表项中的二进制序列变为 $0a_1a_2\cdots a_k$,仍指向桶 $j$. 增加一个二进制序列为 $1a_1a_2\cdots a_k$ 的表项,指向一个新创建的桶 $z$. 桶 $j$ 中的各条记录被重新散列,分配到桶 $j$ 或桶 $z$ 中. 最后把新记录插入桶 $j$ 或桶 $z$ 中.

### 3.3 改进算法的删除和更新

除了删除和更新前的查找过程不同外,改进算法的删除和更新操作与可扩展散列相同.

## 4 算法对比实验

### 4.1 实验的设计

为了简化实验,本文采用的Hash函数能将一个记录的查找键转变为一个4位的二进制序列,故桶地址表最多有 $2^4=16$ 个表项:假定每个桶最多存放2个记录,以随机的10个源数据进行测试,主要关心所用桶数、溢出桶数、表项数3个方面的数据. 其中所用桶数反映空间复杂度,溢出桶导致磁盘I/O

增加，反映时间复杂度，桶地址表太大时可能要放在磁盘中，导致磁盘I/O增加，或者把主存中的其他数据挤出主存，导致系统性能下降. 由于可扩展散列使用散列前缀，而线性散列和改进算法使用散列后缀，为了让数据对3个算法一致，实验中针对可扩展散列的二进制序列 $a_1a_2a_3a_4$，对线性散列和改进算法，理解为 $a_4a_3a_2a_1$. 实验没有设计相同记录超过2个的情况，因为这样的情形对3个算法的影响是一样的.

### 4.2 实验的执行和数据结果分析

表1为测试结果. 从实验结果可以看出，改进算法明显优于可扩展散列算法和线性散列算法. 这是因为改进算法的桶地址表是线性递增或递减的，并且桶地址表常驻主存中. 所以一般不会出现溢出桶，但相同查找键的记录多到1个桶也存放不下时，可能会出现溢出桶的现象. 此外允许散列后缀不等长，即使在极端情况下，1次插入最多增加1个桶地址表表项和1个桶. 实验证明了这一点，由于篇幅所限，实验仅仅列举了10个数据，在实际实验中，我们共模拟了几千个数据，均证明了改进算法的性能相当优越.

表1　3种算法所用桶数、表项数、溢出桶数比较

| 比较项 | 算法 | 数据 | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0001 | 1100 | 0000 | 0010 | 1111 | 1000 | 1110 | 0000 | 0011 | 0010 |
| 桶数 | 可扩展散列 | 1 | 1 | 2 | 4 | 4 | 5 | 6 | 7 | 7 | 8 |
| | 线性散列 | 1 | 2 | 2 | 3 | 3 | 5 | 5 | 6 | 7 | 8 |
| | 改进算法 | 1 | 1 | 2 | 3 | 3 | 4 | 4 | 5 | 5 | 6 |
| 表项数 | 可扩展散列 | 1 | 1 | 2 | 8 | 8 | 8 | 8 | 16 | 16 | 16 |
| | 线性散列 | 1 | 2 | 2 | 3 | 3 | 4 | 5 | 5 | 6 | 6 |
| | 改进算法 | 1 | 1 | 2 | 3 | 3 | 4 | 4 | 5 | 5 | 6 |
| 溢出桶数 | 可扩展散列 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 线性散列 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 2 |
| | 改进算法 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 5 结语

改进算法桶地址表的增减是线性的，一般情况下桶地址常驻主存. 除了相同查找键的记录多到1个桶也存放不下外，没有溢出桶的出现. 改进算法的核心是允许散列后缀不等长，即使在极端情况下，1次插入最多增加1个桶地址表表项和1个桶. 改进算法的不足之处是，查找时Hash(*KEY*)要与多个桶地址表表项进行比较，由于操作是在主存中进行的，相对于改进算法的优点，增加的开销可以接受.

**参考文献：**


[1] 王忠效，范植华. 汉字异或动态散列分组查找算法[J]. 中文信息学报，1998，12(4)：60.

[2] 严蔚敏，吴伟民. 数据结构[M]. 北京：清华大学出版社，1996：251－262.

[3] 李晓明，凤旺森. 两种对URL的散列效果很好的函数[J]. 软件学报，2004，15(2)：60.

[4] Hector Carcia-Molina，Jeffrey D Ullman，Jennifer Widom. Database System Implementation[M]. 北京：机械工业出版社，2002：170－184.

[5] 李新社，杜晓辉，尹毅峰，等. 多态密码机制的改进及其严格雪崩特性分析[J]. 北京工业大学学报，2009(6)：35.

[6] Witold Litwin. Linear hashing：a new tool for file and table addressing[C]//Proc 6th Conf on Very Large Databases，New York：ACM Press，1980：212－223.