第18卷 第2期
2008年2月

计算机技术与发展
COMPUTER TECHNOLOGY AND DEVELOPMENT

Vol. 18 No. 2
Feb. 2008

一种OLAP海量数据载入技术的研究

林 昕,李心科
(合肥工业大学 计算机与信息学院,安徽 合肥 230009)

摘 要:在数据仓库、数据挖掘和联机分析处理系统中,海量数据的载入虽然不是时时发生,但是海量数据的载入效率直接影响着系统性能,如何高效地进行海量数据的载入十分重要。提出了两种技术,即基于UB-Tree的海量数据的初始化载入技术以及海量数据的增量载入技术,阐述了基于UB-Tree的海量数据载入的技术及其算法,提出了海量数据载入模型,建立基于UB-Tree的初始化载入,以及如何在已有的UB-Tree上做增量载入。经过性能分析,算法减少了I/O和CPU代价,为一种有效的海量数据载入方法。

关键词:海量载入;UB-树;数据仓库;联机分析处理
中图分类号:TP311.131 **文献标识码**:A **文章编号**:1673-629X(2008)0051-04

Study on OLAP Mass Data Loading Technology

LIN Xin, LI Xin-ke
(School of Computer and Information, Hefei University of Technology, Hefei 230009, China)

Abstract: In data warehouse, data mining and OLAP large data sets loading occurs not continuously, but the loading efficiency influences the performance of the system. It is necessary to have efficient mass data loading techniques. In this paper, proposes two techniques, one for initial loading based on one new UB-Tree, and one for incremental loading, which adds data to an existing UB-Tree. Proposes a model of mass data loading and two techniques, one for initial loading, which creates a new UB-Tree, and one for incremental loading, which adds data to an existing UB-Tree. Both techniques try to minimize I/O and CPU cost. Analysis demonstrates that the algorithms are efficient and able to handle large data sets.

Key words: mass loading; UB-tree; data warehouse; OLAP

0 引 言

当有海量数据载入数据仓库索引时,根据每一个数据元组的索引值进行插入操作是不可行的,在同一时间大量的页访问将导致系统性能下降。在多数情况下元组以随机顺序出现,而这种顺序并不合适已有的索引,无序的元组插入使本来连续的元组被插入到不同的页,并且在每次新元组插入都会引起索引的搜索和数据页访问。因而数据载入的代价并非随着数据页的增长而线性增长,而是随着新元组数目的增长而增长。

数据的查询性能很大一部分依赖于索引的聚集和页利用率,在进行数据载入时,聚集和页利用率依赖于元组插入的顺序,而K-D-B-Trees无法保持一定的页占有率[1]。

迄今为止已经在多维索引结构上做了大量的工作,如R-Trees[2,3],GridFiles[4],以及quad-Trees[5],文中将讨论UB-Tree。提出基于UB-Tree海量数据载入模型;两种海量数据载入算法,一个用于数据的初始化载入,另一个用于增量载入。分析表明算法减少了I/O和CPU代价。是一种有效的海量数据的载入技术。

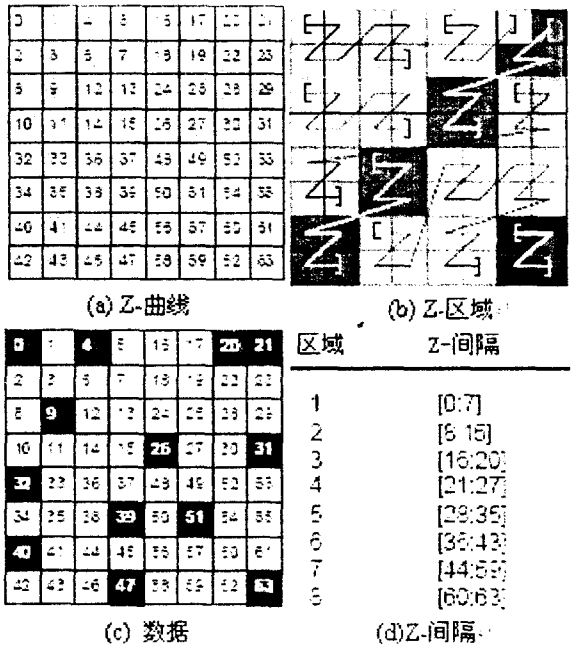
1 UB-Tree简介

UB-Tree(universal B-Tree)是一种多维聚集索引结构,对插入、删除、点查询操作保持了对数级的性能,同时也能保证50%的页利用率。UB-Tree的基本思想是利用空间填充技术将多维数据映射到一维[6]。Z-曲线(见图1(a))保持了多维数据聚集的特性。UB-Tree的创新之处是使用Z-区域(见图1(b))来生成多维空间分离的区域,这将提高多维数据的区域查询的效率。Z-地址是Z-曲线上元组的位置,Z-地址也决定了元组所属的Z-区域。Z-区域

收稿日期:2007-05-27
基金项目:安徽省科技攻关重点项目(0012021A)
作者简介:林 昕(1975-),男,安徽合肥人,硕士研究生,研究方向为软件工程、数据仓库、OLAP;李心科,副教授,博士,研究方向为软件工程、神经网络等。

[a:b]是 Z－曲线上一段曲线所经过的区域，由此段曲线两端的 Z－地址 a，b 来定义这块区域，b 称为 Z－区域[a:b]的区域地址，每个区域对应二级存储的一页，即 UB－Tree 的叶子页。通过搜索 Z－地址来得到存储在 UB－Tree 里的 Z－区域。

举例说明，如果将元组 $O(3,5)$ 插入如图 8×8 的二维空间，首先计算 Z－地址为 27，然后找到包含这个 Z－地址的区域，如图 1(d)所示是区域 4，现在返回和这个区域对应的页并插入该点，必要时还需要进行页分裂，最后存储该页。



(a) Z-曲线 (b) Z-区域

(c) 数据 (d)Z-间隔

图 1 Z－曲线、Z－区域、Z－间隔

2 影响数据载入的因素

当有海量数据需要载入数据仓库索引时，如何高效载入相当重要。在生成一个新的索引时希望它能迅速地被用户使用，也希望能进行高性能的区域查询。根据文献[7]把数据载入数据仓库是初始化和维护数据仓库的一个重要的过程。单就数据容量本身而言就需要一个高性能的载入方法。有效的存储海量数据对于以后数据仓库的维护也将带来重要影响。

在数据仓库和数据挖掘应用系统中，从 OLTP 系统移入 OLAP 系统中时，数据源一般以可携式格式存在，如 ASCII 文本文件，XML，Excel、Table，等等，并不是以二进制存在。

在进行海量数据载入时希望达到以下的目标：

1)减少随机磁盘存取；

2)减少磁盘 I/O；

3)减少 CPU 负载；

4)使聚集达到最优化；

5)使页占有率达到最优化。

数据载入时最主要的代价来自二级存储的存取，或者说载入过程受到 I/O 的限制。因此要减少磁盘 I/O 操作尤其是尽可能减少随机的磁盘存取。连续线性的磁盘访问速度则是很快的。在数据载入时尽量访问磁盘上连续的空间，可以减少 I/O 代价；CPU 负载也同样是数据载入过程中的一个重要因素。

为提高区域查询的效率，采用聚集索引是十分必要的，聚集索引可以减少磁盘随机存取的次数。有两种类型的聚集：一种是元组聚集；另一种是页聚集。对于海量数据载入应同时满足这两种聚集，即一个数据页内的元组应该聚集存储，各个数据页也应该聚集存储。

为了减少区域查询时返回页的数目，必须保证页的利用率。对于数据仓库而言如果数据只载入一次而没有新增数据，比如存入 CD－ROM，那么有必要保持最大的页利用率。否则为了保证数据增量载入，每一页至少保证 50％的利用率。

3 海量数据载入模型

多维海量数据载入通常分为两种类型：

a)将多维数据划分为一定的分区，然后再将这些分区载入索引；

b)将所有待载入数据总排序，然后将排好序的数据载入索引。

在第一种类型里有针对 R－Tree[2]和 Grid－file[4]的方法，而在第二中类型中有针对 R－Tree 使用 Hilbert－曲线进行预排序的方法。这里提出的海量数据载入模型属于第二种类型。在海量数据载入模型中通常有三步(见图 2)：第一步计算每个元组的码值；第二步元组根据关键字排序；第三步将已排好序的数据载入 UB－Tree。

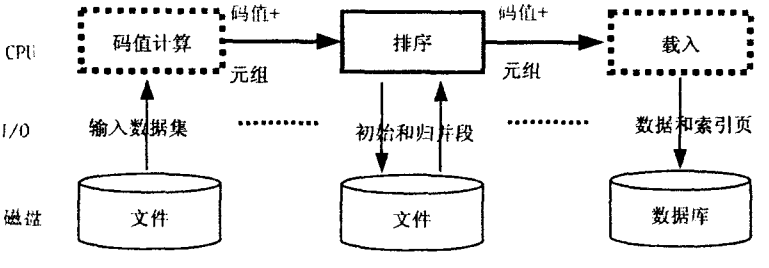


图 2 海量数据载入过程模型

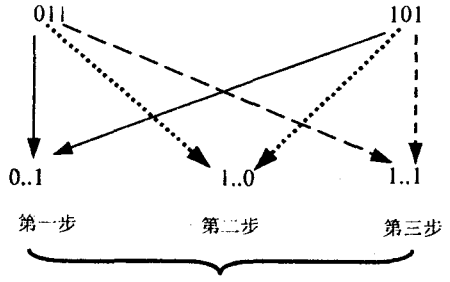
为了加速数据载入过程速度也可以使用 I/O 通道，这样可以避免载入过程产生的临时数据写入二级

存储器。中间结果使用二进制也可以提高载入效率,避免了内二进制码数据表示和外 ASCII 码数据表示的相互转化的时间。既节省了 I/O 代价又减少了 CPU 时间。

3.1 码值计算

码值是数据仓库中元组的唯一标识,通常是一个元组属性集合的子集。然而在内部的索引结构会使用不同的表示方法,在 UB－Tree 中使用空间填充曲线(Z－曲线)来计算曲线上元组的位置数值(Z－地址),从而得到码值。UB－Tree 中通过位插入算法可以很有效地计算码值[8]。

如果要计算图 1(c)元组 $O(3,5)$ 的 Z－地址,位插入算法通过以下步骤完成 Z－地址的计算(见图 3)。



元组 O(3,5)的 Z-地址=27

图 3　位插入算法计算 Z－地址

3.2 归并排序

对海量数据进行外排序的最好的方法是使用归并排序。先按照内存的大小,将元组数据读入内存并利用有效的内部排序算法对进行排序,然后将数据读回磁盘。对于内排序使用堆排序可以一次获得 $2M$ 长度的归并段,M 是内存一次可以载入的元组数。为了减少 I/O 代价在归并的同时计算码值,并把码值整合进归并段。

3.3 初始化载入

初始化载入是生成一个新的索引的过程。一旦数据按照 Z－地址排序,就可以使用 B－Tree 技术生成索引。

在初始化算法中使用了一种特殊的数据结构,大数据页,类似于磁盘上的页,但是容量是标准数据页的两倍。大数据页只在内存中使用,其页利用率是标准页的两倍。使用这样的数据结构是为了简化算法。

初始化载入算法描述如下:首先定义一个空白的大数据页 page。然后开始读入元组,直到 page 的利用率达到指定的标准数据页利用率的两倍。当达到这个限度的时候对当前大数据页从中间进行分裂,即将 page 分裂为 pg1 和 pg2,pg1 和 pg2 中各自包含 page 中一半的元组。分裂后 pg1 达到了指定的页利用率,写回磁盘。分裂地址随即被插入到索引中,不需要在索引中进行搜索,直接插入索引页的末尾,在索引页满时也会引起页分裂。使用大数据页结构同样可以保证索引页的利用率。

随后 pg2 的内容被拷贝回 page 中,并继续向 page 中载入新的元组,达到利用率限度再分裂,如此循环,直到没有元组剩下为止。在结束的时候需要进行检查,看 page 中是否还需要进行最后的分裂,因为在 page 中数据可能超过标准页的的数据容量,如果超过标准页的容量则还要进行一次分裂,并返回 pg1,pg2。否则直接存储页。存储最后一页的算法如下所示:

```
void writeLastPages(LargePage &page) {
  if(page.getUtilization(page)>100%) {
  Page pg1, pg2;
    splitPage(page,&pg1,&pg2 );
    storePage (pg1);
    storePage (pg2);
  else {
storePage(page); //隐含把大数据页转化为标准页
} }
```

算法既提供了元组聚集又提供了页聚集,因为输入的数据已经按照 Z－地址进行了排序。如果采用 CD－ROM 来存贮数据,则可以选择 100% 的页利用率。

算法也可以用来对已存在的 UB－Tree 进行重新组合。如果在归并排序阶段加入已存在 UB－Tree 中的元组,那么将得到一个包含新、旧数据的 UB－Tree,在初始化载入时这种方法的效率比把新元组插入到现有 UB－Tree 中的效率要高。当然如果初始载入的数据较少时,采用增量载入的效率还是很高的。虽然这并不能保证一定的页利用率。

3.4 增量载入

当需要对现有的 UB－Tree 进行扩展时候,称其为增量载入。增量载入不同于初始化载入,因为它不仅要生成新页,而且要更新已有的页。

增量载入算法描述如下:首先将第一个元组根据其码值插入 UB－Tree,将插入的那一个数据页载入到设置的大页 page 中去。然后开始以下循环。

读入输入数据的下一个元组。检查元组是否属于当前的大数据页 page,如果不属于,则根据 writeLastPage 函数存储大页 page。包含这个元组的数据页返回到 UB－Tree 中,通过在 UB－Tree 中点查询找到该元组所在的数据页,将该数据页拷进大数据页 page 中。元组也随即插入到 page 中。如果 page 超过设计的页利用率,则将 page 分解成两个标准页 pg1 和 pg2,包含插入元组的页继续留在 page 中。而另一个写回

磁盘。剩余的元组不断地载入到 page 中，检查利用率、分裂直到所有元组载入为止，最后的 page 再使用 writeLastPage 写回磁盘。

和初始化载入算法相比，增量算法有两个新的地方需要注意：其一是检查元组是否属于当前页；其二是页分裂后保存页的策略不同。

4 性能分析

在归并排序中，对于一个有 P 页，n 个元组的输入数据，需要 P 次读入数据，P 次写归并段，P 次读初始归并段，P 次写输出结果，这就需要 $2P$ 次顺序读和 $2P$ 次顺序写。CPU 代价包括码值计算，开销为 $O(n)$，再加上内排序，其时间复杂度为 $O(n\log n)$。

假设 A 是页读写数，T 是每页元组数，n 是新元组数，o 是在索引中旧的元组数，分析 I/O 代价如下：

随机插入数据：$A = n/T + 2n$，因为需要进行 n/T 页的聚集，且每插入一个元组需要取出和写回，故需要 $2n$ 次。

初始化载入：$A = 2(n + o)/T$，因为需要进行 n/T 页的聚集；从已存在的 UB－Tree 中读 o/T 页；同时生成 $(n + o)/T$ 页。

增量载入：最差情况 $A = n/T + 2n$，这是每次插入元组属于另一页时的情况，最好的情况是几乎或不需要更新现有页，这时候 $A = 2n/T$，n/T 是需要进行页的聚集次数，而 n/T 是新生成页的页数。

当用 o/T 时假设页的利用率是 100%，当然这不是实际的情况，可以把页的利用率 ρ 计算进去，$o/\rho T$，然而为了简化忽略了。

图 4 显示了将 1000 个新元组插入旧元组时 I/O 复杂度。图 5 显示了将一定数目新元组插入 1000 个元组时 I/O 复杂度。两幅图显示了不同插入技术 I/O 复杂度。

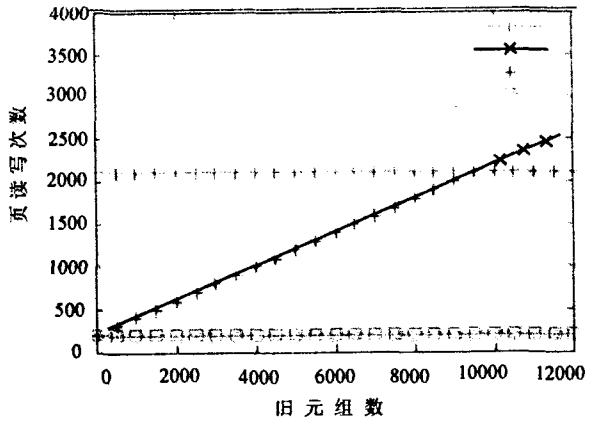


图 4 载入 1000 个新元组

根据新旧元组数的关系可以简化计算 A 的方法。在初始化操作时如果 $o = 0$ 或者 $o \ll n$，那么 $A = 2n/T$，这说明载入的时间取决于新元组的数目，当 $o \gg n$ 时，$A = 2o/T$，即载入时间取决于旧元组的数目。增量载入要稍微复杂点，因为要考虑到是将元组载入已有的 UB－Tree 的页中还是新页中，最差的情况，每个元组插入不同页，其 I/O 代价和随机插入相同。然而如果新元组主要载入到新页中，那么其载入性能和初始化载入相似。

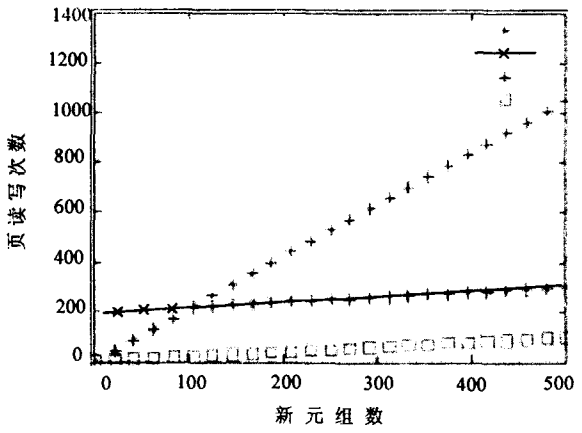


图 5 载入到 1000 个旧元组中

5 结 论

阐述了基于 UB－Tree 的海量数据的载入方法，提出了海量数据载入模型，两种数据载入算法。初始化数据载入提供了元组和页聚集，可以提高区域查询的性能。该算法也可以用来重组 UB－Tree，或将一个 UB－Tree 和新元组进行合并。如果只是一部分数据需要载入 UB－Tree 中则可采用增量算法，这优于随机插入。分析表明模型能有效地提高数据载入效率。

参考文献：

[1] Robinson J T. The K－D－B－Tree: A Search Structure For large multidimensional dynamic Indexes[C]// In Lien Y E. Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data. Ann Arbor, Michigan, USA: ACM Press, 1981.

[2] Garcia Y J, Lopex M A, Leutenegger S T. A Greedy Algorithm for Bulk Loading R－Trees[C]// In ACM International Workshop on Advances in Geographic Information Systems. Boston: ACM Press, 1998: 163－164.

[3] Kamel I, Faloutsos C. On Packing R－trees[C]// In CIKM. Jerusalem, Israel: Springer－Verlag, 1993: 490－499.

[4] Leutenegger S T, Nicol D M. Efficient Bulk－Loading of Gridfiles[J]. In Proceedings of the IEEE Transactions on Knowledge and Data Engineering, 1997, 9(3): 410－420.

（下转第 58 页）

3 协议验证

3.1 验证思想

验证的主要目的是比较文中算法与已有协议的通信开销。由于现有检查点软件中没有设计协调式检查点协议的完整接口,而文中考虑的重点仅限于通信协议,故文中实验基于 MPI 并行环境[6]编程模拟各种协议来进行通信开销比较,而不是将协议实现在检查点软件中再进行比较。

比较的侧重点在消息驱赶机制(MsgClear)、消息计数机制(MsgCount)、文中所提一次同步机制(OnceSync)三种协议的协调开销方面。

3.2 实验及结果

实验使用基于 MPI 的 1024 * 1024 数值矩阵的 Jacobi 叠代程序,迭代 4000 次,每次叠代完成后都调用一次协调协议,迭代过程中的矩阵分割数为参与协调协议的进程数(包括 Manager)。因通信是对称的,故可使用各进程实际执行时间的平均值来代表协议的开销。实验结果如表 1 及图 4 所示。

表 1 三种协议通信开销实验结果

进程数	一次同步	消息驱赶	消息计数
2	113.58	114.80	117.47
4	53.91	54.28	55.07
8	96.13	119.20	131.37
16	193.19	273.68	321.40

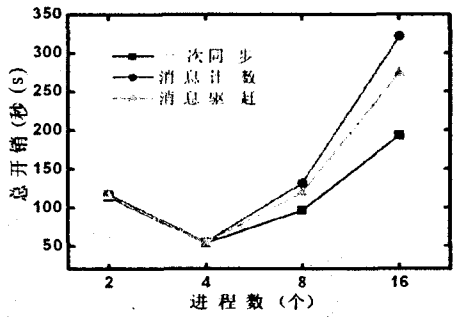


图 4 三种协议通信开销比较曲线图

由图 4 可看出,消息计数协议由于其附加信息的计算和记录、发送开销而导致性能较差;消息驱赶协议由于每条通信信道上都需要发一个驱赶消息,消息复杂度为 $O(N_2)$[5],性能也不理想;文中所提一次同步协议的消息复杂度为 $O(N)$,且没有引入其他的计算和记录开销,所以在各种叠代次数下性能都优于其他两种机制。

4 总 结

提出了一种应用于通信信道不可靠环境下的开销较小的协调式检查点协议——一次同步协议,其主要依据是在不可靠信道环境中,中途消息和由于信道不可靠而导致的消息丢失在用户应用程序看来是没有区别的。通过把中途消息当作消息丢失交给用户程序去处理,减少协调所需要的开销。实验证明它比传统的协议有更好的性能。但因其要求应用程序本身具有消息丢失的处理机制,对该协议的应用范围构成了一定的限制。

参考文献:

[1] Elnozahy E N, Alvisi L, Wang Y M, et al. A Survey of Rollback-Recovery Protocols in Message-Passing Systems[J]. ACM Computing Surveys, 2002, 34(3): 375-408.

[2] Alvisi L, Rao S, Husain S A, et al. An Analysis of Communication-Induced Checkpointing[C]// Proceedings of the Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing. Washington, DC., USA: IEEE Computer Society, 1999.

[3] 魏晓辉,鞠九滨. 分布式系统中的检查点算法[J]. 计算机学报, 1998(4): 367-375.

[4] Helary J M. Communication-Induced Determination of Consistent Snapshots[J]. IEEE Transactions on Parallel and Distributed Systems, 1999, 10(9): 856-877.

[5] 汪东升,邵明珑. 具有 O(n)消息复杂度的协调检查点设置算法[J]. 软件学报, 2003, 14(1): 43-48.

[6] The MPI Forum. The MPI Message-passing Interface Standard[EB/OL]. 1995-05. http://www.mcs.anl.gov/mpi/standard.htm.

(上接第 54 页)

[5] Hjaltason G R, Samet H, Sussmann Y J. Speeding up Bulk-Loading of Quadtrees[C]// In ACM International Workshop on Advances in Geographic Information Systems. Indiana, USA: Addison-Wesley, 1997: 50-53.

[6] Bayer R, Markl V. The UB-Tree: Performance of Multidimensional Range Queries[R]. Germany: Institut f"ur Informatik, TU Munchen, 1997.

[7] Paller A. Rely on Red Brick's Performance for Data Warehouse Applications[R]. Massachusetts: DataWarehousing Institute, Informix Corporation, 2000.

[8] Orenstein J A, Merrett T H. A Class of Data Structures for Associative Searching[C]// In Proceedings of the Third ACM SIGACT-SIGMOD Symposium on Principles of Database Systems. Waterloo, Ontario, Canada: ACM, 1984: 181-190.