计算机研究与发展
Journal of Computer Research and Development
ISSN 1000-1239/CN 11-1777/TP
44(1):105~110, 2007

三级存储系统中一种高效的连接算法

刘宝良　李建中　高　宏
(哈尔滨工业大学计算机科学与技术学院　哈尔滨　150001)
(liubl@hit.edu.cn)

An Efficient Join Algorithm for Data on Tertiary Storage

Liu Baoliang, Li Jianzhong, and Gao Hong
(*School of Computer Science and Technology, Harbin Institute of Technology, Harbin* 150001)

Abstract The online use of tertiary storage system provides a costly and feasible scheme for massive data management. In order to extend database system to manage data on tertiary storage, the relational operators, especially the join operation, are one of the key problems that must be resolved. An efficient tertiary join algorithm is presented. Experimental results show that the join algorithm is better than previous ones in performance and scalability. The join algorithm can greatly reduce the I/O cost compared with previous ones. When the data amount is huge, the performance of the join algorithm is even better than that of disk join. The result of the paper shows that tertiary storage can be used to manage massive data as well as disks, solving the key problem of storing and querying massive data.

Key words join algorithm; tertiary storage system; attribute separation; join result index

摘　要　第3级存储器的联机使用为海量数据管理提供了一种廉价可行的方案.为了使数据库管理系统能够联机使用第3级存储设备,第3级存储设备上的关系操作算法,特别是连接操作算法是必须解决的关键问题之一.提出一种高效的连接算法.实验结果表明,该算法无论在性能方面还是在扩展性方面都优于以往算法,极大地减少了I/O代价.当数据量较大时,算法的性能不低于基于磁盘的连接算法.结果表明,第3级存储器可以像磁盘一样在海量数据库系统中联机使用,解决海量数据库存储和联机查询等关键问题.

关键词　连接算法;3级存储系统;属性划分;连接结果索引

中图法分类号　TP311.13

磁盘价格的下降及存储容量的增长始终不能满足飞速增长的海量数据存储需求[1-2].由主存、磁盘及联机使用的第3级存储设备(如机械手磁带库等)构成的3级存储系统为海量数据的存储提供了廉价、可扩展的硬件解决方案[3-4].目前的数据库系统都是基于2级存储设备(主存、磁盘)的,无法有效管理第3级存储设备上的数据.当需要处理第3级存储设备上的数据时,需先将数据复制到磁盘,然后使用基于磁盘的数据库技术进行处理.显然这种方式无法直接管理数据量高于磁盘容量的海量数据[5-6].因此,需要研究适用于3级存储系统的数据操作算法,特别是最常用也最耗时的连接算法.本文目的就是研究适用于第3级存储设备(以机械手磁带库为例)的海量数据连接算法.

以往的连接算法通常具有数据的多遍扫描问题与冗余I/O问题.为了避免第3级存储器过高的

收稿日期:2005-08-03;修回日期:2006-05-09
基金项目:国家自然科学基金项目(60273082);国家"八六三"高技术研究发展计划基金项目(2002AA444110);国家"九七三"重点基础研究发展规划基金项目(G1999032704);黑龙江省自然科学基金项目(zjg03-05)

定位代价,以往算法在对数据进行预处理(比如散列)时,通常需要多遍扫描原始关系数据. 冗余 I/O 问题源于关系数据的存储方式. 磁盘和磁带都是块访问设备. 关系数据按照元组存储,即使仅访问关系中某一元组的某一属性,也需要读取包含该元组的整个数据块. I/O 代价是影响连接操作性能的主要因素,由于磁带是慢速访问设备,更应该避免冗余 I/O问题.

本文提出了一种高效的连接算法 ASJ(attribute separating join). ASJ 算法能够有效地避免以上问题.当数据量较大时,算法的性能不低于基于磁盘的连接算法. 表 1 列出了文中用到的参数:

Table 1 The Parameters Used in This Paper

表 1 本文用到的参数

Parameters	Description
$\|R\|$	Size of relation R (the same with S, M, D, J_R and J_S)
$\|\|R\|\|$	Cadinality of R (the same with S, J_R and J_S)
J_R	Join attribute relation of R (the same with S)
A_R	Non-join attribute relation of R(the same with S)
$c(R)$	Reducing ratio of R (the same with S)
JRI	Join result index
R_C	Semi join relation of R
X_D	Data transfer rate of disk
X_T	Data transfer rate of tape driver
λ	The ratio of S size to R size

1 相关工作

许多数据库工作者在基于 3 级存储系统的海量数据连接算法方面开展了一些研究工作[7-9]. 这些工作可以分为两类:嵌套循环(nested loop)连接和散列(hash based)连接. 循环嵌套连接算法是一种低效的连接算法,它循环读取内关系并将内关系与外关系进行连接. 针对嵌套循环连接的低效问题,散列连接为内关系及外关系构造散列桶,同时将散列值相同的内、外关系的散列桶进行连接. 散列连接能够有效地减少 I/O 代价.

针对第 3 级存储器的最有效的连接算法是 CTT-GH[8]算法. 它的执行过程分为两个阶段:第 1 阶段是划分阶段,采用相同的方法划分关系 R 及 S,并要求 R 的每个划分 R_i 可以完整地装入内存;第 2 阶段是探测阶段,依次将 R_i 装入内存,并探测对应的 S_i,形成连接结果.

2 基于属性划分的连接算法

2.1 连接属性划分阶段

设 $R(rj,a_1,\cdots,a_m)$和 $S(sj,b_1,\cdots,b_n)$是两个关系,其中 $\{a_1,\cdots,a_m\}$和 $\{b_1,\cdots,b_n\}$分别是 R 和 S 的非连接属性集合,rj 和 sj 是连接属性. 连接属性划分阶段将关系 R 及 S 进行连接属性划分. 对关系 R 进行属性划分的结果为 $J_R(rid,sj)$及 $A\_R(rid,a_1,\cdots,a_m)$. 其中,rid 是在属性划分的过程中,由系统自动添加的自增长的整数,它能够惟一标识 R 中的元组. 由于 J_R 中包含连接属性,称其为连接属性关系. A_R 中不包含连接属性,称其为非连接属性关系.

2.2 连接阶段

连接阶段有两个步骤:在步骤 1 中划分关系 J_R为多个子关系,要求每个子关系可以完整地装入主存. 在步骤 2 中将关系 J_S 读入磁盘缓冲空间,并同时用相同的划分函数对 J_S 进行划分,当磁盘缓冲区满时,将磁盘中数据与关系 R 进行连接.这个过程直到关系 J_S 扫描完. 连接阶段输出的结果为两个连接属性关系的元组标识符对,即(rid,sid). 由于其可以作为两个非连接属性关系的连接索引[10],我们称其为连接结果索引.

下面是连接阶段算法的形式化描述:

算法 1. JoinPhase.

输入 连接属性关系 J_R 与 J_S.

输出 连接结果索引 JRI.

① FOR $i=1$ TO $\lceil |J\_R|/|D| \rceil$ DO

② 在磁盘中形成 $\lfloor |D|/|M| \rfloor$ 个关系 J_R 的子关系;

③ 将这些子关系写入磁带;/* 每个划分在磁带中存储位置连续 */

④ END FOR

⑤ WHILE J_S 未读完 DO(并行地执行⑥,⑦~⑩步)

⑥ 将 J_S 的一部分 J_S_{i+1} 读入磁盘 D,并按照与步骤②相同的方式散列;

⑦ 将前一次读入的 J_S_i 与 J_R 进行连接;得到连接结果 JRI 写入内存缓冲区

⑧ IF 内存缓冲区满 THEN

⑨ 将 JRI 按照 sid 排序后,写入磁盘缓冲区;

⑩ END IF

⑪ END WHILE

2.3 物化阶段

物化阶段利用连接结果索引 JRI,归并两个非连接属性关系 A_R 及 A_S. 下面是物化阶段算法的形式化描述:

算法 2. Materialization.

输入:非连接属性关系 A_R,A_S 及连接结果索引 JRI.

输出:最终连接结果.

① 按照 sid 多路归并排序 JRI;

② WHILE (not end JRI) DO

③ IF 磁盘缓冲区 D 满 THEN

④ 按 rid 多路归并 R_C;

⑤ 归并 R_C 与 A_R,并输出连接结果;

⑥ END IF

⑦ 用 rid 替换 A_S 元组中的 sid;

⑧ 将归并结果写入内存缓冲区 M;

⑨ IF 内存缓冲区 M 满 THEN

⑩ 对 M 中的元组按照 rid 排序,并写回磁盘;

⑪ END IF

⑫ END WHILE

2.4 ASJ 算法的正确性证明

ASJ 算法在物化阶段中,JRI 分别与 A_S 及 A_R 做归并,归并的结果是否正确完全依赖于 JRI 的元组是否与连接结果元组一一对应. 下面的定理保证这种一一对应关系.

定理 1. 设 R 与 S 是两个关系,J_R 与 J_S 分别为其对应的连接属性关系,JRI 为 J_R 与 J_S 的连接结果索引,C 为 R 与 S 的连接结果,则 JRI 中的元组与 C 的元组间存在一一对应关系.

证明. 略.

2.5 对 ASJ 算法的优化

在连接属性划分阶段,可以根据统计信息,预先估算连接属性关系 J_R 及 J_S 的大小,分情况将他们保存到磁盘或磁带中,从而对基本 ASJ 算法进行优化.

1) J_R 与 J_S 可同时保存到磁盘缓冲区

在这种情况下,连接属性划分阶段将 J_R 与 J_S 直接写入磁盘,采用磁盘连接算法进行连接,生成连接结果索引. 其中,写入磁盘操作代价与扫描关系 R 及 S 的代价重叠,因此写入磁盘代价可以忽略.显然在磁盘上进行连接操作比在磁带上进行连接操作快得多.

2) J_R 与 J_S 之一可保存到磁盘缓冲区

在这种情况中,连接属性划分阶段将可以保存在磁盘中的关系(假设为 J_R)直接写入磁盘. J_R 与 J_S 的连接转变为磁盘-磁带连接. 优化的连接阶段算法同样分为两个步骤:步骤 1 划分 J_R,此时对 J_R 的划分不需要多遍扫描,可以极大地节省磁带 I/O. 步骤 2 将 J_S 读入缓冲区,并用相同的划分函数对 J_S 进行划分. 当缓冲区满时,将缓冲区中数据与 J_R 进行连接. 步骤 2 执行到 J_S 读完为止. 要根据磁盘剩余空间与内存空间的大小情况来决定采用磁盘剩余空间作缓冲区,还是采用内存做缓冲区.

2.6 性能分析

CTT-GH 算法是第 3 级存储设备上性能最好的海量数据连接算法. 当下面条件满足时 ASJ 算法性能优于 CTT-GH 算法.

定理 2. 设 $\lambda = |S|/|R|$ ($|S| > |R|$) 为关系 S 与关系 R 的大小的比值,$c = \min\{c(R), c(S)\}$ 为缩减比的最小值. 则当满足条件 $c \geqslant \sqrt{1+\lambda}$ 时,ASJ 算法的性能优于 CTT-GH 算法的性能.

证明. 略.

3 实验结果

本文在 Pentium Ⅲ 计算机上实现了 ASJ 算法. 计算机的 CPU 主频为 Pentium Ⅲ 550MHz,操作系统为 Linux 7.2,内存为 128MB,20GB 磁盘空间. 计算机通过一个 INITIO SCSI-2 总线连接 Exabyte 220L 磁带库. 磁带库具有两个 Eliant 820 磁带驱动器,磁带容量为 10GB(未压缩),磁带架上可以放 20 盘磁带. 磁带库的详细性能参数参见表 1.

在所有的实验中,设定关系 R 和 S 的大小满足 $|S| = 2|R|$,元组数满足 $\|S\| = 2\|R\|$,缩减比满足 $c(R) = c(S)$. R 的连接属性值惟一,S 的连接属性参照 R 的连接属性,但 S 的连接属性随机生成. 这样,连接结果索引 JRI 的势在没有其他过滤条件的情况下为 $\|S\|$. 除非特别说明,我们在实验中设定的主存容量为 40MB,磁盘容量为 650MB.

3.1 数据集大小对 ASJ 算法和 CTT-GH 算法性能的影响

第 1 组实验比较 ASJ 算法和 CTT-GH 算法的性能随数据集变化的情况. 在这组实验中,数据集从 3GB 增长到 30GB(关系 R 从 1GB 增长到 10GB). $c(R) = c(S) = 10$,$\|JRI\| = 0.1\|S\|$. 在图 1 中

列出了两种算法的执行时间,显然 ASJ 算法的执行时间大大优于 CTT-GH 算法. 从图中可以看到,即使两个连接属性关系都不能保存到磁盘中(即 $|R|>6$GB 时),ASJ 算法的性能仍然比 CTT-GH 好得多. 比较 ASJ 与 CTT-GH 的性能随数据量增长的变化情况,可以清楚地看出 ASJ 算法执行时间随数据集的增长比 CTT-GH 算法慢得多,说明 ASJ 算法的扩展性极大地优于 CTT-GH 算法.

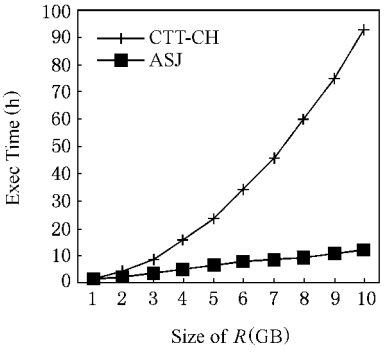

Fig. 1 The influence of dataset size on performance.

图 1 数据集大小对算法性能的影响

图 2 中给出了连接阶段的执行时间占 ASJ 算法总的执行时间的比例. 在所有情况下,连接阶段占 ASJ 算法总的执行时间不到 2%. 通过图 1 及图 2 可以得出结论:ASJ 可以有效地避免冗余 I/O 的问题,并能够极大地减少关系的扫描遍数,极大地提高连接算法的执行效率及改善连接算法的扩展性.

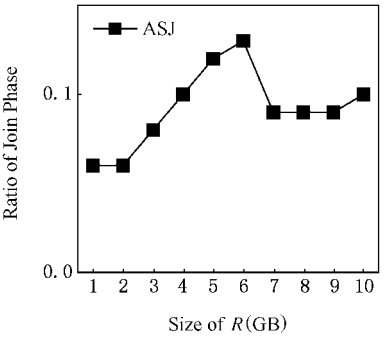

Fig. 2 The ratio of join phase.

图 2 连接阶段占总执行时间比例

3.2 缩减比对性能的影响

在第 2 组实验中,使用 30GB($|R|=10$GB)的测试数据集. $\|JRI\|=0.1\|S\|$. 我们固定连接属性的大小,通过增大关系 R 及 S 的元组大小的方法来增大缩减比,为了保持数据集合大小不变,我们相应地减少元组数目. 由于 CPU 代价可以忽略不计,所以执行时间的减少肯定不是由于元组数目的下降所致. 实验结果如图 3 所示. 从图中可以看出,算法的执行时间随着缩减比的提高而降低. 缩减比越大两个连接属性关系就越有可能放入磁盘,进而提高连接的执行时间. 当增大缩减比时,算法的执行时间趋向于收敛到一个定值. 该值即第 3 级存储设备上进行连接操作的最小 I/O 代价.

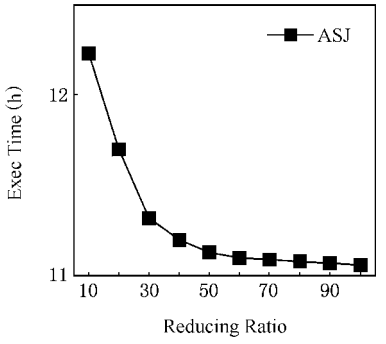

Fig. 3 The influence of reducing ratio on performance.

图 3 缩减比对性能的影响

3.3 JRI 的势对 ASJ 性能的影响

第 3 组实验选用 30GB($|R|=10$GB)的测试数据集. $c(R)=c(S)=10$. 实验结果如图 4 所示. JRI 的势越高物化阶段生成的临时关系 R_C 越大,R_C 很可能不能放在磁盘缓冲区中,从而在与 A_R 归并时,需要多遍扫描 A_R. 实验中关系 S 的大小为 20GB,即使 JRI 的势为 $0.1\|S\|$ 时,物化阶段的临时关系 R_C 也不能完全存放在磁盘缓冲区,从而需要多边扫描 A_R. 但是即使这样,ASJ 算法的性能也要大大优于 CTT-GH 算法. 从图 4 中可以看到,即使 JRI 的势等于 $\|S\|$ 时,ASJ 算法的执行时间只有 CTT-GH 算法的一半左右.

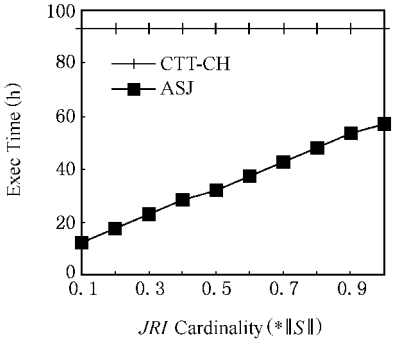

Fig. 4 The influence of $|JRI|$ on the performance.

图 4 JRI 的势对算法性能的影响

3.4 ASJ 算法与 3 种磁盘连接算法的比较

为了充分说明第 3 级存储设备管理海量数据的有效性,我们分别实现了磁盘中的 3 种连接算法,它

们分别是嵌套循环连接、排序归并连接及散列连接,并将这些连接算法与 ASJ 算法进行性能对比. 在实验中固定内存大小为 40MB. 在前 3 种算法中,设定磁盘空间足够大. ASJ 算法的磁盘缓冲空间为 1GB. 实验数据集大小从 3GB 变化到 30GB(其中关系 R 从 1GB 变化到 10GB),$c(R)=c(S)=10$. $\|JRI\|=0.1\|S\|$. 实验结果如图 5 所示. 在 4 种算法中,嵌套循环连接的性能最差,排序归并连接与散列连接的性能相近,并且散列连接的性能稍稍优于排序归并连接. 当数据量超过 18GB($|R|>6$GB)时,ASJ 算法的性能优于其他 3 种磁盘连接算法. 为了更清楚地说明 ASJ 算法性能优越的原因,我们将 4 种算法的执行时间与扫描一遍数据集的时间进行对比,结果如图 6 所示. 从图中可以看到,嵌套循环连接扫描数据集的遍数远远高出其他 3 种算法,排序归并连接及散列连接都需要对数据集进行多遍扫描,且它们扫描数据集的遍数相近,但都高于 ASJ 算法. 这个实验还进一步表明,对于海量数据连接操作来说,选用高性能的算法比选用的更高级的设备的收益更大.

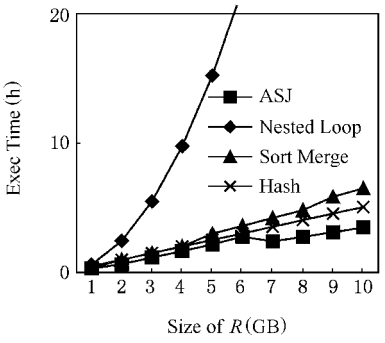

Fig. 5 Performance comparison between ASJ and disk joins.

图 5 ASJ 算法与磁盘连接算法的性能比较

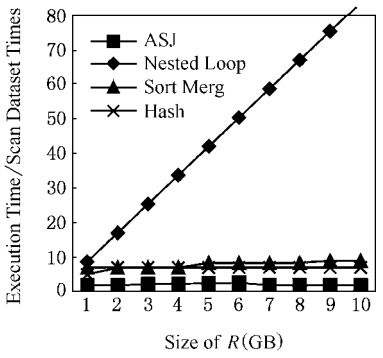

Fig. 6 The ratio of execution time to the times of scanning dataset.

图 6 算法执行时间与扫描数据集时间的比值

3.5 磁带传输速度对算法性能的影响

不同厂商磁带库系统的数据传速率差别很大. 在以上实验中都是采用慢速的磁带库设备,这必然使实验结果向有利于磁盘算法的方向偏斜. 在下面的仿真实验中,我们测试磁带传输率对 ASJ 算法性能的影响. 实验数据集为 30GB($|R|=10$GB),$c(R)=c(S)=10$,$\|JRI\|=0.1\|S\|$. 我们用磁盘来模拟磁带库设备,并通过插入冗余数据的方法来改变数据传输率. 实验结果如图 7 所示. 从图中可见,磁带数据传速率对 ASJ 算法性能影响很大. 当磁带数据传输率为磁盘数据传输率的 1/10 时,ASJ 算法性能不如排序归并算法及散列算法,当提高磁带数据传输率为磁盘数据传输率的 3/10 时,ASJ 算法开始优于磁盘算法. 目前带光线通道的高端磁带库磁带驱动器的数据传输率为 70MB/s,而个人计算机上的磁盘传输率仅为 10MB/s 左右. 在高端磁带库设备上运行 ASJ 算法会比磁盘上的连接算法效率更高. 这又进一步说明了采用第 3 级存储设备管理海量数据的可行性及有效性.

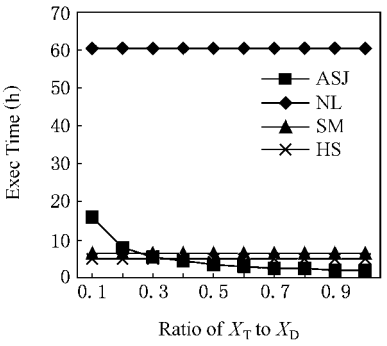

Fig. 7 The influence of tertiary data transfer rate on the join performance.

图 7 磁带的传输速度对算法性能的影响

4 结 论

针对传统连接算法的多遍关系扫描问题及冗余 I/O 问题,本文提出了基于属性划分的 ASJ 连接算法. 理论分析及实验表明,ASJ 算法可以极大地降低影响连接算法性能的主要因素——I/O 代价. 并且随着数据量的增长,ASJ 算法具有很好的扩展性,说明 ASJ 算法比以往的磁带连接算法更适用于对海量数据的处理. 本文还通过大量的实验说明了采用第 3 级存储器管理海量数据的可行性及有效性,并说明在海量数据管理中,算法的选择对性能的影响比采用的设备对性能的影响大得多. 并且在高端

磁带库设备上运行 ASJ 算法的性能甚至比磁盘连接算法的效率更高.因此我们的结论是,3 级存储器辅以有效的数据处理算法是解决海量数据存储和管理的有效途径.

参 考 文 献

[1] L B Michael, B Adam, R H James, *et al*. Data management challenges in very large enterprises [C]. In: Proc of the 28th Int'l Conf on Very Large Data Bases. San Francisco: Morgan Kaufmann, 2002

[2] M J Carey, L M Haas, M Livny. Tapes hold data, too: Chanllenges of tuples on tertiary store [C]. Association for Computing Machinery Special Interest Group on Management of Data, Washington, 1993

[3] S Sarawagi. Database systems for efficient access to tertiary memory [C]. IEEE Symp on Mass Storage Systems, Monterey, CA, 1995

[4] S Sarawagi. Query processing in tertiary memory databases [C]. The 21th Conf on Very Large Databases, Zurich, Switzerland, 1995

[5] J Yu, D De Witt. Processing satellite images on tertiary storage: A study of the impact of tile size on performance [C]. The 5th National Aeronautics and Space Administration Goddard Conf on Mass Storage Systems and Technologies, College Park, MD, 1996

[6] D De Witt, J Yu. Query pre-execution and batching in Paradise: A two-pronged approach to the efficient processing of queries in tape-resident data sets [C]. The 9th Int'l Conf on Scientific and Statistical Database Management, Olympia, Washington, 1997

[7] J Myllymaki, M Livny. Disk-tape joins: Synchronizing disk and tape access [C]. ACM SIGMETRICS, Lausanne, Switzerland, 1996

[8] J Myllymaki, M Livny. Relational joins for data on tertiary storage [C]. Int'l Conf on Data Engineering, Birmingham, UK, 1997

[9] Kraiss, P Muth, M Gillmann. Tape-disk strategies under disk contention [C]. Int'l Conf on Data Engineering, Birmingham, Australia, 1999

[10] P Valduriez. Join indices [J]. ACM Trans on Database System, 1987, 12(2): 218-246

Liu Baoliang, born in 1976. Ph. D. His main research interests are query processing techniques for massive data.

刘宝良,1976 年生,博士,主要研究方向为海量数据的查询处理技术.

Li Jianzhong, born in 1950. Professor and Ph. D. supervisor. Senior member of China Computer Federation. His main research interests are database, parallel, etc.

李建中,1950 年生,教授,博士生导师,中国计算机学会高级会员,主要研究方向为数据库、并行计算等.

Gao Hong, born in 1966. Professor and Ph. D. supervisor. Senior member of China Computer Federation. Her main research interests are database, data warehouse system, etc.

高 宏,1966 年生,教授,博士生导师,中国计算机学会高级会员,主要研究方向为数据库、数据仓库等.

Research Background

The management of DBMS on tertiary storage is becoming more and more important with the development of applications, not only because tertiary devices have been used to archive data, but also the amount of data that application had to deal with is increasing rapidly. Despite the decrease in secondary storage prices, the data storage requirements of these applications cannot be met economically using secondary storage alone. Tertiary storage offers a lower-cost alternative. But till now, commercial database systems are optimized for performance with primary and secondary storage. Tertiary storage, if included in the system, serves only as a backup medium or slow store from which data is first moved onto disks and then the DBMS operates on it as though it were disk resident data. Tertiary devices differ significantly from magnetic disks, in particular. They are not random access devices and media switch times are several orders of magnitude higher. Due to these differences, optimizations that work well for magnetic disks can result in disastrous performance on tertiary storage. So we need to study data operation methods required by tertiary storage system, especially the join method. The purpose of this paper is to study the join method for massive data on tertiary storage system.