基于移动实时事务相关图的数据收集

王敬华[1]，杨进才[1]，刘云生[2]

(1. 华中师范大学 计算机系，湖北 武汉 430079; 2. 华中科技大学 计算机学院，湖北 武汉 430074)

摘　要: 在移动环境下,为了满足实时事务在移动主机断接后数据的可用性,在事务启动前,数据必须收集到移动主机的缓存中。根据实时事务的相关性,给出了实时事务的相关图。介绍了基于时间阈值、空间阈值、数据的有效期阈值的收集算法,有效保证数据的可用性和实时事务的定时限制。

关键词: 移动计算; 实时事务; 事务相关图; 数据收集

中图法分类号: TP14　　**文献标识码**: A　　**文章编号**: 1001-3695(2005) 03-0097-04

Data Hoarding Based on Relation Graph of Mobile Real-time Transaction

WANG Jing-hua[1], YANG Jin-cai[1], LIU Yun-sheng[2]

(1. Dept. of Computer, Central China Normal University, Wuhan Hubei 430079, China; 2. School of Computer, Huazhong University of Science & Technology, Wuhan Hubei 430074, China)

Abstract: In a mobile environment, mobile hosts may disconnect from server frequently. To meet the data availablility during mobile hosts are disconnected, data items accessed by transaction must be hoarded to mobile host. A relation graph of real-time transaction is defined in this paper. According to the relation graph, data hoarding algorithms based on time limit, room limit and validity limit are proposed. Using the algorithms, avaibilitiy of data and the timing constraints of real-time transaction are met.

Key words: Mobile Computing; Real-time Transaction; Transaction Relation Graph; Data Hoarding

1　引言

数据收集(Data Hoarding) 是将数据装入到移动客户机的缓存中,为即将发生的断接作准备[1,2],其目的是使客户机在断接期间使用本地数据自主操作。传统缓存机制中的预取过程通过使数据较快被存取来提高系统性能,预取与数据收集的主要不同之处在于前者是利用网络通信量低的时候将未来使用可能性最高的数据传送到缓存;而后者的执行不依赖于计划断接前网络的通信状况。上述两个过程具有不同的目标: 预取试图提高系统的性能,而数据收集试图提高数据的可用性。

数据收集要解决的问题是: 收集的单元(粒度) ; 收集哪些数据项? 什么时候进行数据收集? 实时数据库系统中,事务和数据都有定时限制。收集时还要考虑数据的时效,对于一个断接周期,如果数据的时效小于断接周期,这样的数据将不被收集。

2　数据收集的相关工作

数据收集最初用于文件系统,大多文件系统中对断接操作的支持是通过扩展缓存管理来实现的。第一个支持断接操作的是 Coda 文件系统[3],用户通过脚本语言指定感兴趣的文件,缓存管理器把这些文件和 LRU 算法得到的缓存数据结合起来构成本地缓存。其收集过程需要用户的指导。另一个文件系统 SPY UTILITY 通过使用收集执行树自动进行收集处理[4],收集执行树是基于文件引用路径建立的。SEER 系统的自动预报数据收集是根据用户过去的行为(文件存取经历) 寻找文件系统间的语义关系。文件用一种称为语义距离的度量聚集在一起,语义距离是对文件之间关联密切度的度量[5,6]。

在数据库系统中数据收集变得非常复杂。有下列原因:

(1) 对于文件系统,收集的单位为文件,它对应于关系数据库中的关系和面向对象数据库中的对象类。但是,这个粒度并不合适,较小的粒度如元组或部分元组将更有意义并能获得更好的性能。

(2) 使用过去引用的经历推导数据库项间的依赖比标志出文件之间的依赖复杂。由于文件系统是按层次组织的,找出相关文件间的相关相对容易。一般而言,同一个目录下的文件比不同目录下的文件有更密切的关系。

(3) 一个文件系统下的用户通常只需要有限数量的文件(整个文件系统的一小部分) 来完成工作,而数据库系统的查询可能涉及大量的数据库。这意味着必须缓存大量的数据以支持数据库系统的断接操作。

(4) 在文件系统中,已有很好支持断接操作的系统实现。如收集算法、日志优化、重新连接后的冲突检测与消解。在数据库系统中,这些问题正处于广泛研究中,仅有为解决数据库断接操作的一些尝试。

文献[7] 中,将数据挖掘技术引入到数据收集中,通过建立数据之间的关联规则,提出了一种通用的自动数据收集算法。

文献[8] 中,介绍一种在面向对象数据库中通过扩展缓存

收稿日期: 2004-01-07; 修返日期: 2004-05-16

管理来支持断接操作的方法。它使用了三种级别的粒度作为缓存: 属性缓存、对象缓存和混合缓存。试图在缓存中保存经常查询的数据项。但替代策略要计算单个数据项的存取频率, 因此, 在大型数据库系统中产生很大的计算代价。

文献[9, 10] 提出关系数据库的数据收集方法。数据库设计者定义收集关键字(期望用来获取典型的存取模式)。按关键字关系分成水平段, 以此作为收集单元。其问题是既然收集段是数据库管理员基于收集关键字的定义而产生的, 收集段对于一些移动客户可能没有意义。

此外还有基于过去存取经历的收集方法[11], 以及基于概率图的数据收集算法[12]。

3 基于实时事务相关图的数据收集

前述的数据收集方法的不足概括为: ①数据的收集不完全自动化。②数据收集的内容不能反映临时断接情形下的需要。③数据收集没有考虑实时事务的复杂性和数据的实时性。因此在移动实时数据库系统下, 要设计全新的收集策略。我们提出了基于实时事务相关图的数据收集策略。

3.1 事务相关性分析

实时事务具有复杂的语义和结构特征, 事务之间存在结构、行为、数据、时间上的相关性[13]。因此事务执行的正确性不仅包含数据库的完整性与一致性, 而且包含事务间结构、行为、数据、时间等相关性的满足。为此, 系统必须提取关于事务之间的相关性知识, 为断接操作的数据收集做准备。事务的特性以及部分相关性的知识可在事务启动前经静态预分析提取, 进一步的信息可通过动态预分析提取。

事务在经过预编译时, 可提取关于事务的下列特性:

(1) 主动性, 事务是否有可能触发新的活动。对于主动事务的截止期的计算及优先级的分派都应考虑事务所可能触发的活动。

(2) 定时限制及紧迫性、关键性等知识, 包括事务的价值函数, 软、硬性等。

(3) 资源要求、操作逻辑等, 包括事务的数据集、执行时间估算和事务读写数据库的类型(是只读操作还是既读又写), 操作的顺序。

(4) 可能触发的活动的相关信息。

显然, 上述有关事务的信息不仅能为事务处理(包括调度、优先级分派、并发控制等) 提供必要的支持, 如对于时间要求紧迫的硬实时事务可分派高的优先级, 使之先调度运行而满足其截止期, 也为断接数据收集提供依据。

3.2 实时事务相关图

实时事务之间存在着如下的相关性:

定义 1 如果 t_i 是 t_j 的子事务, 则 t_i 与 t_j 存在结构相关关系 SR(Structure Relevancy), SR = { $t_i \rightarrow t_j$ $|t_i, t_j$ 存在结构相关关系}。

定义 2 如果 t_i 经过用户交互后执行 t_j, 则 t_i 与 t_j 存在行为相关 BR(Behavior Relevancy), BR = { $t_i \rightarrow t_j$ $|t_i, t_j$ 存在行为相关关系}。

定义 3 如 t_i 完成一段时间后执行 t_j, 则 t_i 与 t_j 存在时间相关关系 TR(Time Relevancy), TR = { $t_i \rightarrow t_j$ $|t_i, t_j$ 存在时间相关关系}。

定义 4 如数据被 t_i 存取后才能被 t_j 存取, 则 t_i 与 t_j 存在数据相关关系 DR(Data Relevancy), DR = { $t_i \rightarrow t_j$ $|t_i, t_j$ 存在数据相关关系}。

设一个移动客户应用程序所包含的事务集为 T = { t_1, t_2, …, t_n}

定义 5 事务集 T 的事务相关图为事务间关系的简单(无圈, 无平行边) 有向图:

TRG(T) = <V, E>

其中, V = { $v_1, v_2, \cdots, v_n$} 为顶点集, v_i 表示事务 t_i 以及由静态预分析所提取的事务 t_i 的信息, v_i 为一个四元组:

v_i: : = (tID, attr, timings, interval)

其中, tID 为事务号, 标志一个事务; attr 表示事务的属性, 如是否是子事务、硬实时事务等; timings 为事务的定时限制; interval 为事务执行的估算时间。

事务的执行时间与数据库的状态、事务所含的操作, 以及处理机的数据处理能力如 CPU 的处理速度有关, 精确估算事务的执行时间是一件非常困难的事, 但它是实时事务调度的一个依据, 我们假定有些事务具有估算执行时间, 且它是准确的。

E = { $e_1, e_2, \cdots, e_m$} 为有向边集, 任一有向边 e_i 表示它关联的事务间的关系, e_i 为一个三元组:

e_i: : = (type, delay, value)

其中, type 表示关联类型: type = { SR |BR |TR |DR} ; delay 为时间间隔, 如果边 e_i 关联的有向边为 $t_i \rightarrow t_j$ 且 t_i 与 t_j 时间相关, 则 delay 表示 t_i 完成后经过 delay 时间执行 t_j。

value 为相关值, $0 \leq value \leq 1$, 值为 1 表示必然相关, value < 1, 表示选择相关或概率相关。

定义 6 必然相关: 如果事务 t_i 的后继事务 t_j 唯一, t_j 为 t_i 的必经之路, t_i 与 t_j 必然相关。相关值 value = 1。

定义 7 选择相关: 如果事务 t_i 的后继事务只有多个, 具体执行哪个事务由用户的交互或前阶段运行产生的不同数值决定。如 t_j 在 t_i 后可能执行, 称 t_i 与 t_j 选择相关, 值 value 为可能执行到的概率。

定义 8 在事务相关图 TRG 中, 如果从某个节点 t 有有向边指向 t_i 与 t_j, 则称 t_i, t_j 的互为兄弟事务; t 为 t_i, t_j 的前趋事务; t_i, t_j 为 t 的后继事务。

结构相关与时间相关一般为必然相关, 行为相关与数据相关为选择相关。

图 1 为一个包含 13 个实时事务的相关图。

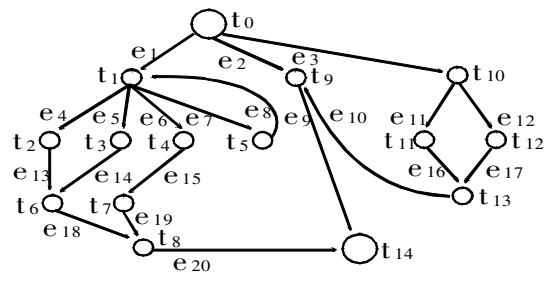

图 1 实时事务相关图

图 1 中, t_0 为开端事务。所有有向边中 e_8, e_9, e_{10}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}, e_{19}, e_{20} 对应的为必然相关, 其他边对应的关联为选择相关。

3.3　事务相关图的存储结构

列表 $T = \{t_1, t_2, \cdots, t_n\}$ 存储事务相关图顶点的信息, T 的第 i 个元素 t_i 存储事务 t_i 的信息。

结构域 t_i. tID 存储第 i 个事务的标志;

t_i. attr 存储第 i 个事务的属性;

t_i. timings 存储第 i 个事务的定时限制;

t_i. interval 存储第 i 事务执行的估算时间。

采用关联矩阵 ***MRT*** 来表示事务相关图, 矩阵的元素 $\boldsymbol{MRT}_{ij}$ 描述事务 t_i 与事务 t_j 的相关信息。

$\boldsymbol{MRT}_{ij}$. value 存储事务 t_i 与事务 t_j 的关联度。若其值为 0, 表示事务 t_i 到事务 t_j 无直接关联。

$\boldsymbol{MRT}_{ij}$. type 存储事务 t_i 与事务 t_j 的关联类型。

$\boldsymbol{MRT}_{ij}$. delay 存储事务 t_i 与事务 t_j 的时间间隔。

矩阵 ***MRT*** 具有如下性质:

(1) 因为事务相关图为简单有向图, 矩阵对角线元素的值为 0, 即

$$\boldsymbol{MRT}_{ij}.\text{value} = 0,\ (1 \le i \le n)$$

(2) 事务相关图每个节点射出边的所有关联度之和为 1, 即

$$\sum_{j=1}^{n} \boldsymbol{MRT}_{ij}.\text{value} = 1,\ (1 \le i \le n)$$

4　基于事务相关图的数据收集

4.1　收集事务及其构造方法

按事务的经历模型及其投影概念[13], 考虑事务 t 的经历 H_t 在其数据操作集 OP_t 上的投影, 有

$$H_{OP} = P(H_t, OP_t) = \langle OE_t, <_t \rangle$$

$$OE_t = \{op_i(D_i) \mid op_i(D_i) \in OP_t \wedge D_i < DB, i = 1, 2, \cdots, n\}$$

显然, $\forall op_i(D_i) \in OP_t$ 有一自数据库 DB 中存取其数据集 D_i 的存取过程 $p_i(D_i)$, 于是有:

定义 9　$\forall op_i(D_i) \in OP_t$, 称它所对应的存取过程 $p_i(D_i)$ 为事务的一存取过程, 其存取的数据集为 p_i 的存取集, 记为 $DS(p_i)$。

令事务 t 的所有存取过程集为 $P_t = \{p_i(D_i) \mid 1 \le i \le n\}$, 要保证 t 在断接时所存取内容在缓存中, 就必须保证 $\forall p_i \in P_t$, p_i 开始执行时 $DS(p_i)$ 均在缓存。数据收集的任务就是确定每个存取过程 $p_i \in P_t$ 的存取集, 为此, 它为各个存取过程 $p_i(i = 1, 2, \cdots, n)$ 创建一些收集过程 $P'_i(i = 1, 2, \cdots, m, m \le n)$, 它具有以下性质:

(1) $DS(P'_i) \subseteq DS(p_i)$。

(2) P'_i 为 $DS(P'_i)$ 只读过程。

我们称 P'_i 为 t 的收集过程(Hoarding Process), 称它所要收集的数据集为 t 的一个收集集。

定义 10　称事务 t 的收集过程 P'_i 所组成的事务(只读事务)为其收集事务, 记为 Hoard(t), 它是各 P'_i 的一个偏序集: $Hoard(t) = \langle P', <_{th} \rangle$, 其中 $P' = \{P'_i \mid P'_i$ 为事务 t 的收集过程$\}$; t 为收集事务的主事务; $<_{th}$ 为收集顺序。

4.2　数据收集算法

数据收集算法根据事务相关矩阵, 构造事务执行的一个次序, 同时依次序收集各个事务存取的数据集。

定义 11　设应用程序中各事务集 $T = \{t_1, t_2, \cdots, t_n\}$, 由 T 所生成的收集事务序列记为 TH(T), 它是 T 中各事务子集 T' 的一个偏序集: $TH(T) = \langle T', <_t \rangle$, 其中, $T' \subseteq T$。$\forall t_i, t_j \in TH(T)$, if $t_i <_t t_j$, 则称 t_i 为 t_j 的前趋, t_j 为 t_i 的后继。

定义 12　收集事务序列中各个事务 t_i 所存取的数据集记为 $DS(t_i)$, DS 为收集数据总集。$DS = \bigcup_{t_i \in TH_i} DS(t_i)$。

收集算法的输入和输出分别为

输入: 初始事务队列 InitQ, 存有初始事务的事务标志; 事务信息列表 T; 事务相关矩阵 ***MRT***。

输出: 收集事务序列 TQ, 收集数据总集 DS。

4.2.1　简单收集算法

简单收集算法不考虑各个事务本身的特性, 事务之间的相关特性, 只简单将断接期间可能执行的事务的数据集收集到缓存。算法 SH 是一个广度优先的递归收集算法, 设置一个辅助队列 TmpQ, 用来临时存放事务号。

简单收集算法 SH:

```
Procedure SH ( MT MTR, QUEUE InitQ)
    //MT 为事务相关矩阵类型, QUEUE 为队列类型
    TmpQ = ;
    While ! Empty( InitQ) Do      //Empty( QUEUE) 判别队列是否空
      ti = OutQueue( InitQ) ;      //OutQueue( ) 元素出队列
      If ! In( TQ, ti)             //In( QUEUE, ti) 判别队列是有元素 ti
      EnterQueue( TQ, ti) ;    //EnterQueue( ) 元素进队列
      EnterQueue( TmpQ, ti) ;
      D = GetData( ti) ∪ D; //DetData( Ti) 获取事务 ti 存取的数据集
      Endif
    EndDo
    While ! Empty( TmpQ) Do
      ti = OutQueue( TmpQ) ;
      For j = 1 to n
        If MRT ti tj . value ≠ 0
          EnterQueue( InitQ, tj) ;
        Endif
      EndFor
    EndDo
    If! Empty( InitQ)
        SH( MTR, InitQ) ;
    Endif
EndProc
```

4.2.2　收集的阈值

简单收集算法 SH 简单地将断接后应用程序的所有可能执行到的事务的数据集进行收集。事实上, 断接不是长期的, 可能只有一小段时间, 因此只需为断接期的操作收集数据。同时, 由于缓存大小有限, 数据也具有有效期, 这些为收集算法应考虑到的限制。

(1) 时间阈值

假定断接期为 DisconInterval, 它将作为收集的时间阈值, 超过其阈值的事务的数据将不被收集。

假定断接点事务 t_0 到事务 t_i 的一条路径为 $PATH(t_i) = \langle t_0, t_1, \cdots, t_i \rangle$。事务 t_j 的估计执行时间 t_i. interval。

若 $SumInterval(t_j) = \sum_{j \in path(t_j)} t_j.\text{Interval} \le \text{DisconInterval}$, 则事务 t_i 有可能在断接期中执行, 否则 t_i 不可能在断接期执行, t_i 及其后继事务的数据将不被收集。

(2) 空间阈值

设供数据收集的缓冲区的大小为 HoardSize, 当事务 t_i 的数据集 D_i 与已收集的数据集 D 的并 $D \cup D_i$ 所占空间 SumSize

(t_i) = size(D∪ D_i) > HoardSize, 事务 t_i 的数据集将不被收集。

(3) 数据的有效期阈值

设事务 t_i 存取的数据集为 D_i, D_i 中数据从断接开始时计算的最短有效期为 MinTimings(D_i) 。如果经过一条执行路径到事务 t_i 时, 事务 t_i 存取的数据已过时, 则事务 t_i 的数据集将不被收集, 即 $\sum_{j\in path(t_i)} t_j$. Interval > MinTimings(D_i) 事务 t_i 存取的数据已错过截止期。

4.2.3 基于收集阈值的收集算法

当一个事务 t_i 超过了时间阈值时, TRG 中事务 t_i 的及其后继事务的数据将不再收集, 但其兄弟事务及后继的执行时间可能没有超过时间阈值, 因此还要对兄弟事务进行阈值判断。当一个事务 t_i 超过了空间阈值时, 事务 t_i 的及其后继事务的数据将不再收集, 但其兄弟事务及后继所存取的数据集可能没有超过空间阈值。这里有两个原因: ①兄弟事务所存取的数据集较 t_i 小; ②兄弟事务所存取的数据集与已收集的数据集的重复部分较多, 使追加收集的数据量少。因此还要对兄弟事务进行阈值判断。

当一个事务 t_i 超过了数据有效期阈值时, 事务 t_i 的及其后继事务的数据将不再收集, 此时同样还要对兄弟事务进行阈值判断。

为了对事务 t_i 进行阈值计算, 需要为事务 t_i 的前趋事务维护一个路径队列。记所有当前事务的前趋事务路径集合为 PathSet, 其中每个元素为一个路径队列 PathQ(t_i) 。

PathSet = { PathQ(t_0) , PathQ(t_1) , …, PathQ(t_n) }

进行断接收集时, 如所有事务还没开始, 路径集合 PathSet = 。在数据收集过程中 PathSet 中的路径始终为已进行数据收集的事务的路径, 即将要进行收集的事务的前趋事务的路径。算法 LH:

```
If ! In( TQ, t_i)
For j = 1 to n
If MRT t_j t_i. value≠0 and SumInterval( t_i) ≤DisconInterval And SumSize( t_i) ≤HoardSize
And SumInterval( t_i) ≤MinTimings( D_i) //如果三个收集阈值均满足;
If ! Exsist( path( t_i) ) //PathQ( t_i) 为收集开始时到事务 t_i 的路径
NewPath( t_i) ; //建立新路径;
add path( t_j) To path( t_i) ; //与前趋事务的路径进行连接;
PathSet = PathSet∪ path( t_i) //加入到路径集中
EnterQueue( TQ, t_i) ;
EnterQueue( TmpQ, t_i) ;
D = GetData( t_i) ∪D;
Endif
Endif
Endfor
Endif
```

将简单收集算法 SH 进行扩充阈值限制, 成为基于阈值的收集算法 LH, 算法 LH 为 SH 算法的扩充部分。

5 总结

本文介绍了基于实时事务相关图的简单收集算法和基于阈值的收集算法, 但实时事务除定时限制外, 还具有不同的优先级, 其优先级也决定收集的先后次序。

事务的兄弟事务与后继事务之间存在收集优先级, 它决定是先进行兄弟事务的数据收集还是先进行后继事务的数据收集。当决定后继事务的收集优先级高于兄弟事务收集的优先级时, 所采用的对相关图的遍历算法为深度优先; 反之, 采用广度优先。

兄弟事务之间也存在收集优先级, 有以下几种决定收集优先级的尺度:

(1) 事务的优先级, 优先级高的事务给予高的收集优先级。

(2) 频率, 执行频率高的事务在兄弟事务间给予较高的收集优先级。

(3) 数据增量, 兄弟事务间相对于前趋事务数据集的增量越少的事务给予较高的收集优先级。

在收集过程中如何将优先与收集过程的状态动态进行调和, 以决定后续收集过程是今后进一步研究的内容。

参考文献:

[1] Pitoura, G Samaras. Data Management for Mobile Computing[C]. Kluwer, Dordrecht, 1997.

[2] M Satyanarayan. Mobile Information Access[J]. IEEE Personal Communications, 1996, 3(1): 102-109.

[3] J Kisler, M Satyanarayan. Experience with Disconnected Operation in a Mobile Computing Environment[C]. USENIX Association Conference Proceedings, 1993. 88-93.

[4] C D Tait, H Lei, S Acharya, *et al*. Intelligent file hoarding for mobile computing[C]. New York, NY: Proceedings of the 1st ACM International Conference on Mobile Computing and Networking(Mobicom '95). ACM Press, 1995. 119-125.

[5] G H Kuenning. The Design of the Seer Predictive Caching System [C]. Proceedings of IEEE Workshop on Mobile Computing System and Applications, Santa Cruz, CA, 1994. 253-270.

[6] G H Kuenning, G J Popek. Automated Hoarding for Mobile Computers [C]. Proceedings of the 16th ACM Symposium on Operating Systems Principles, ACM Press, 1997. 264-275.

[7] Y Saygin, O Ulusoy, A Elmagarmid. Association Rules for Supporting Hoarding in Mobile Computing Environments[C]. Proceedings of the 10th International Workshop on Research Issues in Data Engineering. IEEE Computer Society, 2000. 71-78.

[8] B Y Chan, A Si, H V Leong. Cache Management for Mobile Databases: Design and Evaluation[C]. Proceedings of the 14th International Conference on Data Engineering, Orlando, FL, 1998. 280-289.

[9] B R Badrinath, S Phatak. Database Server Organization for Handling Mobile Clients[R]. Technical Report DCS-342, Rutgers University, 1997.

[10] B R Badrinath, S Phatak. An Architecture for Mobile Databases[R]. Technical Report DCS-342, Rutgers University, 1997.

[11] N Pissinou, C Dunu, K Makki. A New Framework for Handling Mobile Clients in a Client-Server Database System[J]. Computer Communication, 2000, 23: 936-941.

[12] 周桓, 李京, 冯玉琳. 移动环境下的低开销自动数据收集算法[J]. 软件学报, 2002, 13(10): 1962-1968.

[13] 刘云生, 胡国玲, 舒良才. 实时主存数据库事务的预处理[J]. 软件学报, 1997, 8(3): 204-209.

作者简介:

王敬华(1965-), 男, 副教授, 硕士, 主要研究方向为现代信息系统; 杨进才(1967-), 男, 副教授, 博士, 主要研究方向为移动数据库; 刘云生(1940-), 男, 教授, 博士生导师, 主要研究方向为现代数据库系统。