移动环境下支持实时事务处理的数据预取

李国徽[1)] 杨 兵[1)] 陈 辉[1)] 杜建强[2)]

[1)](华中科技大学计算机学院 武汉 430074)

[2)](江西中医学院计算机学院 南昌 330006)

摘 要 随着移动通信技术的迅速发展,人们提出了新的应用要求:在移动环境下处理实时事务.而移动通信带宽有限性引起较大的数据访问延迟,有时甚至由于网络传输的断接使得事务得不到所需要的数据,数据预取能够很好地解决这个问题.已有的移动环境下数据预取没有考虑到数据的流行性和事务的时间特性.该文分析影响实时事务数据预取的因素,首先考虑数据易变性、活跃性等因素,获得高价值预取数据集合;然后考虑访问预取数据的事务优先级、数据流行性等因素,构造预取数据的选择函数,通过该函数在前面选取的集合中筛选出对满足实时事务截止期更有价值的数据对象进行预取.实验表明,该数据预取策略能降低移动实时事务满足截止期的比率,更好地支持移动实时事务处理.

关键词 数据预取;移动实时事务;缓存失效;预取策略

中图法分类号 TP311

Data Prefetching to Support Real-time Transactions Processing in Mobile Environments

LI Guo-Hui[1)] YANG Bing[1)] CHEN Hui[1)] DU Jian-Qiang[2)]

[1)](*School of Computer Science & Technology, Huazhong University of Science & Technology, Wuhan* 430074)

[2)](*School of Computer, Jiangxi University of Traditional Chinese Medicine, Nanchang* 330006)

Abstract With the rapid advances in mobile computing technology, there is an increasing demand for processing real-time transactions in mobile environments. Due to the intrinsic limitations of a mobile environment, there can be long time latency for a data access. Sometimes the data can even be unavailable due to the network disconnection. Data prefetching is an approach to this problem. However, most of the previous studies in data prefetching don't consider the timeliness for data objects and deadlines of transactions. Therefore they are inefficient for supporting mobile real-time transaction processing. This paper proposes a data prefetching strategy to support real-time transaction processing in a mobile data broadcast environment. Each data object is assigned a prefetching value determined by its changeability, access frequency. For the data objects with bigger prefetching values, by further considering the priorities of the transactions which access the data objects and the validity intervals for these data objects, the authors determine whether these data objects are actually prefetched. Extensive simulation experiments are conducted and show that when using the proposed data prefetching strategy, more mobile real-time transactions can meet their deadlines.

Keywords data prefetching; mobile real-time transaction; cache invalidation; prefetching schemes

---

收稿日期:2006-06-07;最终修改稿收到日期:2008-06-17.本课题得到国家自然科学基金(60873030)、国家"八六三"高技术研究发展计划项目基金(2007AA01Z300)资助.**李国徽**,男,1973 年生,博士,教授,博士生导师,主要研究领域包括主动、实时数据库、移动计算及并行(并发)程序的同步正确性等.**杨 兵**(通信作者),男,1975 年生,博士研究生,主要研究方向为移动实时数据库系统.E-mail: yangbing@smail.hust.edu.cn.**陈 辉**,男,1976 年生,博士研究生,主要研究方向包括数据流、数据工程.**杜建强**,男,1968 年生,教授,主要研究领域为医学图像处理.

1 引 言

在移动环境下，由于缓存(caching)[1-3]和数据预取(data prefetching)能够显著提高数据访问速度和充分利用广播带宽，已经得到广泛采用. 移动客户机(Mobile Client，MC)缓存里保存数据库服务器部分数据的副本，可以大大减少数据访问时间[4-5]. 传统的数据预取技术[6]，对移动客户机中数据的访问模式进行预测，在数据被请求前预取到 MC 本地缓存中. 与仅仅在需要时才取回数据(fetch-on-demand)这种策略相比较，它能提高数据的可用性(availability)，从而提高系统性能.

在移动实时数据库中[7-8]，一方面数据具有有效期，事务不能访问超过有效期的数据；另一方面，事务有显式的时间限制(如截止期)，事务的执行不能超过截止期. 这就要求移动实时数据库中的预取策略要同时考虑数据的有效期和事务的截止期.

在移动实时数据库中，有两方面的因素会影响数据预取[9]，一方面是数据本身的特性，另一方面是事务的特性：

(1) 数据易变性，指数据的变更速率. 实时数据有着不同的变化速率，有的随时间而频繁地变化，有的很长时间才变，而有的甚至是静态的. 令 $d_{evi}(i)$ 表示数据 d_i 的“外部有效期”[10]，$|d_{evi}(i)|$ 表示 $d_{evi}(i)$ 的长度，若 $|d_{evi}(i)|<t_1$，则说它是高易变的；若 $t_1\leq|d_{evi}(i)|\leq t_2$，则说它是低易变的；若 $|d_{evi}(i)|>t_2$，则是稳定的，t_1 和 t_2 为由应用语义决定的两个阈值.

(2) 活跃性，指数据被访问的频率. 对数据访问也有所谓的“80-20”规律，即大部分的访问集中在少部分的数据上，所以这种数据的访问频率较高，成为活跃数据. 显然，应该优先考虑预取这些活跃数据.

(3) 数据的流行性，是指数据所反映的状态与现实世界真实状态两者间的时间一致性程度，它可以用当前时间 τ_c 与 $d_{evi}(i)$ 之间的时序关系 $<_t$ 来描述：若 $\tau_c\in d_{evi}(i)$，则它是“当前流行”的；$d_{evi}(i)<_t\tau_c$，则它是陈旧的. 当前流行的数据只要出现在广播通道，就应该考虑预取，而陈旧数据完全可以不预取. 流行数据的预取还要考虑事务对其访问的时机，对于那些当前流行而在访问时可能过期的数据在预取空间紧张时可以考虑不予预取.

(4) 事务的时间特性. 高优先级事务访问的数据应该优先被预取，并且尽量不要被置换出 MC 缓存.

传统的预取策略[11-13]没有考虑访问数据的事务特性和数据的时间特性. VP(Value-based Prefetch)预取策略[14]注意到 MC 电源能量限制和数据大小、数据访问率及数据更新率，对于每个数据 d_i，VP 计算出一个预取值 $value(i)$，然后根据 $value(i)$ 来决定哪些数据应该被预取. 假设一个 MC 能够预取 N_p 个数据，那么，VP 策略就会预取 $value(i)$ 值最大的 N_p 个数据. 随后，扩展 VP 策略，提出 AVP(Adaptive Value-based Prefetch)策略，基于不同 MC 要求，在系统性能和能量消耗之间寻找一个平衡. 基于阈值[15]的预取策略通过设置一个预取阈值来决定哪个数据对象被预取，它仅仅考虑数据的访问概率，对于访问概率大于预取阈值的数据将被预取到本地缓存中. 这个预取阈值是根据无线带宽和时间代价动态计算的，随不同的无线网络环境发生变化. 尽管它在预取阈值的设置上考虑到了系统资源，但相对于固定预取阈值的方法来说，是一个很大的进步. CMIP(Cache-Miss-Initiated Prefetch)预取策略[16]动态生成两个集合：总是预取数据集合和失败预取数据集合，对于总是预取集合中的数据，只要出现在数据广播中，就考虑预取到 MC 本地缓存；对于失败预取数据集合中的数据，如果 MC 需要访问的数据在本地缓存中找不到时，就要向数据库服务器发送请求，这时不仅仅请求没有找到的数据，而且要请求预取相关联的若干数据到本地缓存，大大提高 MC 缓存命中率. 但在移动实时数据库中，这些是不够的. 对于一个数据，如果它占用的空间较大，且即将过时，尽管具有较大的访问概率，把它预取到 MC 本地缓存不但消耗很多系统资源，而且可能过期而不被 MC 使用. 总之，这些数据预取策略由于没有考虑数据的有效期及事务的截止期，显然不能满足事务的实时性要求.

本文通过综合考虑数据的有效期、事务的截止期、数据活跃性等因素，研究给出了移动环境下支持实时事务处理的数据预取策略. 文章第 2 节是系统模型；第 3 节详细讨论数据的预取策略；第 4 节对提出的预取策略进行详细的性能分析；第 5 节是全文总结.

2 系统模型

我们使用一个基于请求的广播模型. 数据库服务器上的数据库由 n 个数据 $d_1,d_2,\cdots,d_n$ 组成. 服务器负责维护数据库和响应 MC 的数据请求. 一个 MC 缓存存放频繁访问的数据. 当客户端需要访问的数据在本地缓存中找不到时，它就通过上行带宽向服务器发一个请求. 一旦接收到 MC 的请求，服

务器通过广播通道广播数据.

在移动实时数据库中，数据只在一定时间内是“流行”的，其值随着外部环境状态变化而频繁地改变，所以除了考虑数据库内部状态的一致性，还必须考虑外部状态与内部状态之间的一致性. 数据库中每个数据 d_i 的时间特性定义为一个二元组 $\langle d_{evi}(i), d_{tp}(i)\rangle$. 其中 d_{tp} 为取得 d_i 值的时间；d_{evi} 是外部（或绝对）有效期，如果 $\tau_c - d_{tp}(i) \leq d_{evi}(i)$（$\tau_c$ 为当前时间），则认为 d_i 是有效的；否则 d_i 是过期的.

MC 缓存数据可能由于其在服务器上的更新变得无效. 为了保证缓存数据和数据库数据的内部（或逻辑）一致性，服务器每 L 秒广播失效报告（Invalidation Report，IR）. IR 包含当前时间 τ_c 和一对数据 (d_i, t_u)，并且要求 $t_u > (\tau_c - w \times L)$，在这里，$d_i$ 是某一数据的 id，t_u 是 d_i 最新更新时间，w 是失效广播窗口大小. 也就是说，IR 包含过去 w 个广播间隔的更新历史. 任何客户端如果断开的时间超过 w 个间隔时，IR 将不能使用，此时就必须丢弃掉 MC 中缓存中的所有数据. 只要处于活动状态，每个 MC 都要监听 IR，使相应的缓存数据失效.

为了减少查询延迟，人们提出在每一个广播周期中复制 m 次 IR，称为 Update Invalidation Report（UIR）[17]. 此时，一个客户端仅仅需要等待广播周期的 $1/m$ 来响应一个查询. 每个 UIR 仅仅包含上一次 UIR 广播后更新的数据信息.

系统有一个事务集 $TS = \{T_1, T_2, \cdots, T_m\}$，系统的主要目标是满足事务的截止期，为了保证高优先级的事务优先被调度，本文采用可达截止期最早最优先（Earliest Feasible Deadline First，EFDF）策略分派事务优先级，如果所有的事务都具有不可达截止期，则按照截止期最早最优先（Earliest Deadline First，EDF）策略分派优先级.

3　缓存预取策略

3.1　主要思想

我们设计在移动环境下支持实时事务处理的数据预取策略的主要思想是：把整个数据库视为一个数据集 DB，将 DB 根据预取选择函数获得一个它的子集 S，对任意数据 $d_i \in S$ 出现在广播通道时，MC 预取到本地缓存.

我们提出的缓存预取策略分两步获得预取数据集合 S，首先考虑数据易变性、活跃性等因素，计算数据预取代价函数，然后考虑访问预取数据的事务优先级、数据流行性等因素，使预取的数据尽量满足事务的截止期.

设数据库 $DB = \{d_1, d_2, \cdots, d_n\}$，$d_i$ 的预取代价函数值 $COST_i$ 定义为当 d_i 没有被 MC 预取时存取 d_i 的代价与 MC 预取 d_i 时存取 d_i 的代价之差与 d_i 的大小之比.

$$COST_i = (C_{no_prefetch}(i) - C_{prefetch}(i))/s_i \tag{1}$$

这里的 $C_{no_prefetch}(i)$ 表示 d_i 没有被预取时 MC 访问 d_i 的代价，$C_{prefetch}(i)$ 表示 d_i 被预取时 MC 访问 d_i 的代价.

为了方便讨论，我们用下面的符号来进行说明：

p_{ai} 表示事务访问数据 $d_i (i=1,2,\cdots,n)$ 的概率.

r_i 表示从数据库服务器取数据 $d_i (i=1,2,\cdots,n)$ 到 MC 的代价.

v 表示数据到服务器端进行一致性验证所花的平均代价.

F 表示当前考虑预取的候选数据集.

C 表示 MC 缓存中当前缓存的数据集.

D 表示从缓存中置换出的数据集合. 置换的目的是为预取数据腾出空间.

在没有预取 d_i 的情况下，MC 对数据 d_i 的请求必须到数据库服务器取回数据来满足需求，所以数据 d_i 的平均访问代价是

$$C_{no_prefetch}(i) = p_{ai} \cdot r_i / s_i \tag{2}$$

如果数据 d_i 被预取到 MC 的缓存中，MC 对它的请求必须花费代价 v 到数据库服务器进行一致性验证，并且数据库服务器的数据 d_i 可能发生变化，所以必须考虑它的外部有效期 $d_{evi}(i)$，同时，发生变化的数据 d_i 又必须花费代价 r_i 取回. 所以在数据 $d_i \notin F \cup C \backslash D$ 的情况下，花费的代价如下：

$$C_{prefetch}(i) = p_{ai} \cdot (v + |d_{evi}(i)| \cdot r_i)/s_i \tag{3}$$

我们把式(2)、(3)代入到式(1)中，就可以得到计算预取代价函数 $COST$ 的表达式：

$$COST_i = p_{ai} \cdot r_i / s_i - p_{ai} \cdot (v + |d_{evi}(i)| \cdot r_i)/s_i \tag{4}$$

假定 MC 中缓存中空闲空间是 S_{free}，为了通过数据预取提高系统整体性能，考虑到候选预取数据并不都能满足事务截止期，取系数 $\rho = 0.8$，构造如下目标函数值：

$$\begin{aligned} &\text{(a)}\ \sum_{d_i} COST_i \text{ 取得最大值，并且满足} \\ &\text{(b)}\ \rho \times \sum_{d_i \in F} s_i \leq \sum_{d_i \in D} s_i + S_{free},\ \rho = 0.8 \end{aligned} \tag{5}$$

即使目标函数值：

$$\begin{aligned} &\text{(a)}\ \sum_{d_i \in F \cup c - d} p_{ai} \cdot (v + |d_{evi}(i)| \cdot r_i)/s_i \text{ 取得最大值} \\ &\text{(b)}\ \rho \cdot \sum_{d_i \in F \cup C - D} s_i \leq Size(C),\ \rho = 0.8 \end{aligned} \tag{6}$$

这里的 $Size(C)$ 表示 MC 缓存的大小. 按照上面的分析,我们预取策略的目的就是解决下面的问题:

(1) 找到集合 $DB-C$,使得能够获得最大

$$\left\{\sum_{d_i\in DB-C} p_{ai}\cdot r_i/s_i\right\}.$$

(2) 找到集合 $F\cup C-D\subseteq DB-C$,使得能够获得最大

$$\left\{\sum_{d_i\in F\cup C-D} p_{ai}\cdot(v+|d_{\mathrm{evi}}(i)|\cdot r_i)/s_i:\ \sum_{d_i\in F\cup C-D} s_i\leq Size(C)\right\}.$$

第一个问题实际上是计算可能预取数据的访问代价,第二个问题计算预取到 MC 缓存数据的访问代价,我们第一步的目的是为了获得集合 $F\cup C-D$.

到此为止,我们已初步得到了所有可能被预取的候选数据对象,在此基础上,综合考虑候选数据对象的流行性以及访问它们的事务优先级等因素,决定每个数据 $d_i\in F\cup C-D$ 是否真正被预取. 为此,定义如下的数据预取选择函数:

$$select(i)=\partial_1\cdot f(\tau_c)+\partial_2\cdot p_t(i)+\partial_3\cdot|d_{\mathrm{evi}}(i)|,\quad d_i\in F\cup C-D,$$

这里,

$$f(\tau_c)=\begin{cases}\dfrac{e^{\tau_c-al(i)}}{al(i)-\tau_c}, & al(i)\neq\tau_c\\ 0, & al(i)=\tau_c\end{cases},$$

其中 $\partial_i(i=1,2,3)$ 为加权因子,τ_c 为数据预取时当前时刻;$al(i)$ 为数据 d_i 外部有效期下限,即 d_i 进入系统的时间;$|d_{\mathrm{evi}}(i)|$ 为数据 d_i 外部有效期的长度,$p_t(i)$ 是访问数据 d_i 的事务的最大优先级.

我们获得的预取数据集合 S 满足下面条件:

$$\sum_{d_i} select(i),\quad \text{取得最大值,并且}$$

$$d_i\in F\cup C-D,\ \tau_c-d_{\mathrm{tp}}(i)\leq d_{\mathrm{evi}}(i).$$

当数据 $d_i\in F\cup C-D$ 出现在广播通道时,选择函数最大的数据被考虑优先预取到 MC 本地缓存中.

数据预取和缓存中数据的替换策略是相互关联的. 在冷启动情况下,即移动客户机中初始缓存为空时,采用我们的策略决定预取哪些数据到客户机本地缓存中,随着预取数据的不断增加,客户机的缓存会填满数据. 此时就存在客户机缓存数据淘汰问题. 数据的淘汰也是按照预取相类似的方法,计算每个数据的预取代价函数值,选择对应函数值最小的数据淘汰出去.

3.2 移动客户机的数据预取算法

下面给出支持实时事务处理的数据预取算法,如图 1 所示.

```
Notations:
• DB,C,D,F,S_free: 前面已经定义;
• Q: 移动客户端请求数据集合;
• L_data: 移动客户端向服务器发送请求数据的 id 队列,初始化
  L_data = ∅;
• L_heap: 存放移动客户端数据 id 的堆栈,按照最近最少使用的
  原则,越少使用的数据越靠近栈顶.

Procedure query(Q)
  L_data = ∅;
  for each d_i ∈ Q do
    if d_i ∈ C then
      Wait for the validation report UIR or IR;
    else L_data = L_data ∪ {d_i};
  send request(L_data) to the server.

Procedure replacement(i)
  get d_i from the broadcast channel;
  if d_i ∈ C then
    update d_i in the cache;
    adjust the position of d_i in the L_heap;
  else if S_free != 0 then
    insert d_i into the cache and L_heap;
  else
    while 0.8×S_free ≤ Size(i) do
      remove the top item from the heap and clear the related
      data from the cache;
  insert d_i into the cache and the heap.

Procedure prefetch( )
  find C, s.t. maximize(Σ_{d_i∈DB−C} p_ai·r_i/s_i),
        here Σ_{d_i∈C} s_i ≤ Size(C);
  find F∪C−D ⊆ DB−C,
        s.t. maximize(Σ_{d_i∈F∪C−D} p_ai·(v+|d_evi(i)|·r_i)/s_i);
  find maximize(select(i));
  if d_i in broadcast channel then
    if d_i in maximize(select(i)) then
      replacement(i);
```

图 1 移动客户端缓存管理算法

4 性能评价

4.1 参数估计

我们采用一个滑动平均[18]方法来估算数据 d_i 的"外部有效期"$d_{\mathrm{evi}}(i)$. 对每个数据 d_i,在缓存中我们保持 k 个最近更新的时间戳滑动窗口 $\{t_i^1,t_i^2,\cdots,t_i^k\}$. 数据 d_i 的"外部有效期"$d_{\mathrm{evi}}(i)$ 就可以用下面的公式得到

$$d_{\mathrm{evi}}(i)=\frac{k}{\tau_c-t_i^k},$$

在这里,τ_c 表示当前时间 t_i^k 在滑动窗口里对数据 d_i 的最远一次更新的时间戳.

我们采用在 TCP[8] 中估算环路循环延迟的指数衰变方法来估算 r_i. 初始设置 $r_i=0$,当服务器响应对数据 d_i 的查询时,按如下公式估算系统取回数

据 d_i 的代价：

$$r_i = \partial \cdot (\tau_c - t_{qi}) + (1-\partial) r_i^l,$$

这里 ∂ 表示计算 r_i 的系数，τ_c 表示当前时间，t_{qi} 表示对数据 d_i 的查询时间，等式右边的 r_i^l 表示前次计算得到的数据 d_i 取回代价.

4.2　实验模型

为了评价我们的缓存预取策略，参照文献[14,19]，设计了一组实验数据，并且和 AVP、基于阈值 TB (Threshold-based)的预取算法和不预取(UIR)这 3 种方式进行比较，分析我们的数据预取策略在支持移动实时事务方面的有效性. 由于这 4 种算法的执行，系统 CPU 的平均使用率没有超过 35%，所以我们的模拟实验忽略算法本身的时间开销.

表 1　实验关键参数和其缺省值

参数	缺省值	范围
数据服务器库中数据个数(n)	1000	
移动客户机事务集的个数	200	
事务长度(操作数据个数)	8	4～12
移动客户机事务平均到达率	1 per 800000 bit-times	1 per 2000000 to 1 per 200000 bit-times
移动客户机个数	100	
数据外部有效期长度	100s	50～300s
广播间隔(L)	20s	
广播窗口大小(w)	10 interval	
广播带宽	144Kbps	
上行带宽	14.4Kbps	
缓存空间大小	20% total database size	(10%～30%) total database size
复制 IR 次数($m-1$)	4(=5−1)	
最小数据大小(S_{min})	1KB	
最大数据大小(S_{max})	100KB	
事务优先规则	EFDF,EDF	

在移动客户机端，假定每个移动客户机只产生两个只读事务，事务对数据库服务器数据 d_i 的访问模型类似 ZIPF 分布[20]，按照该模型，事务访问数据 d_i 的概率如下：

$$p_{ai} = \frac{1}{i^{\theta} \sum_{k=1}^{n} \frac{1}{k^{\theta}}},$$

这里的 $0 \le \theta \le 1$，当 $\theta=1$ 时，就是严格的基夫分布，$\theta=0$ 时，符合正态分布.

在数据库服务器端，服务器周期性向客户机广播 IRs 和 UIRs 信息，广播周期时间间隔 L 设置为 20s，并且在每个时间间隔内 UIRs 被复制 4 次. 设服务器端有 n 个数据，每个数据大小 s_i 在 S_{min} 到 S_{max} 平均分布. 我们采用 bit-times 作为时间单位，即无线广播带宽传送 1bit 所需时间，例如，广播带宽是 64Kbps，那么 1M bit-times 大约就是 15s. 表 1 包含实验数据的关键参数和其缺省值. 在实验中，有些参数是固定的，如广播时间间隔；有些参数在一定范围内变化，如数据外部有效期长度.

4.3　实验结果

在我们设计的实验中用平均访问时间、事务重启率及事务没有达到截止期的比率来评价这 4 种算法的性能. 平均访问时间指事务对数据库服务器数据发出请求信息到获得请求的数据之间的时间间隔，度量单位是 bit-times. 事务重启率是指事务在重新提交前由于数据过期而夭折并且重启的平均数.

(1) 平均访问时间实验结果

从图 2 我们可以看出，在平均访问时间上，我们的算法显然不是最优的，AVP 算法由于考虑数据大小、数据访问率及数据更新率等因素进行数据预取，使得事务请求的数据能够在移动客户机缓存获得，从而使平均访问时间最低；TB 算法由于预取了访问频率较高的数据，一定程度上减少了访问时间，但是访问频率高的数据不一定都是小数据，这样，对于访问频率低，但是数据较大，没有预取到移动客户机缓存就会产生较大访问时间；对于没有预取到移动客户机本地缓存的数据，事务对数据的每个请求，就不得不到服务器数据库中获得，它的平均访问时间和 TB 算法相差无几，并且随着事务访问率的提高，增幅较快.

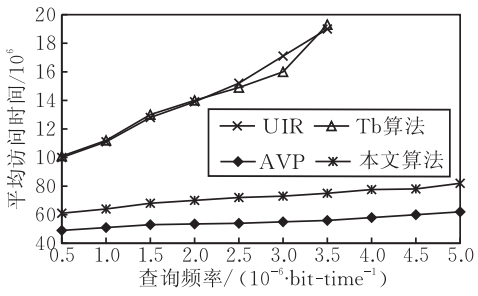

图 2　平均访问时间相对查询频率性能比较

(2)事务重启率实验结果

事务的重启是由于事务在提交前被它请求的数据过期而中止并且被迫重启.从图 3 中可以看出,我们提出的预取算法首先考虑数据易变性、活跃性等因素,获得较高预取价值的数据集合,然后在这个集合中筛选数据的事务优先级、数据流行性高的数据,使预取的数据尽量满足事务的截止期.优先级高的事务预取的数据往往也是当前流行的、易变的.所以我们的算法事务重启率最低.AVP 算法基于值预取的数据,往往也是事务大部分请求的数据,这样使平均访问时间大大减少,也使得绝大部分的数据能够在它的有效期内满足事务请求,所以也获得了一个较低的重启率.不管 UIR 或者是 TB 方法,移动客户机中读取广播通道数据的事务,我们都指定一个截止期,UIR 和 TB 算法完全没有考虑数据有效期和满足事务截止期的影响,因此事务的重启率较高,甚至在事务的执行过程中没有错过截止期的前提下的重启不止一次,在事务到达率较高时,超过 100%[21].

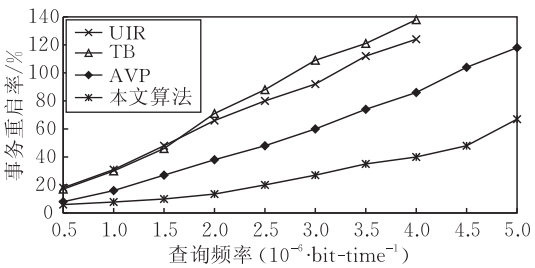

图 3 4 种算法的事务重启率比较

(3)事务没有达到截止期的比率实验结果

图 4 比较了 4 种算法的事务没有达到截止期的比率.对于 UIR 和 TB 这两种算法,事务没有达到截止期的比率几乎相同,并且性能最差,原因就是这两种算法完全没有考虑事务的截止期.AVP 算法能够获得比较好的性能指标是其基于给定值预取的策略虽然没有考虑实时事务的特性,但是通过考虑数据更新率、数据访问率和数据大小等因素预取的数据,由于事务请求的数据绝大部分都能够在移动客户机缓存中得到满足,这样就大大减少了事务的执行时间,从而能够使大部分事务在其截止期内完成.我们提出的算法由于直接考虑到了事务的截止期,它的事务没有达到截止期的比率是最低的.总的来说,对于这 4 种算法,事务没有达到截止期的比率随着事务到达率增加而有上升趋势.

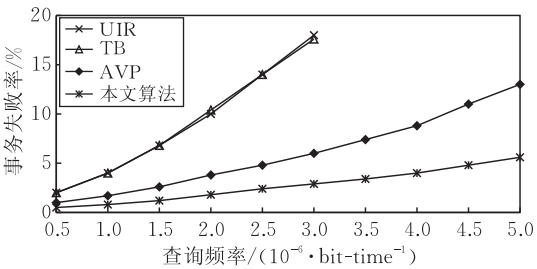

图 4 4 种算法的事务失败率比较

移动环境下事务截止期的满足是支持实时事务处理数据预取的主要目标,实时事务的时间要求(典型地为截止期)的满足甚至比事务执行的逻辑结果完全正确更为重要,宁愿要部分正确但及时的结果而不愿意要绝对正确但过时的结果[22].尽管我们提出的算法平均访问时间比 AVP 算法长,但是在满足事务截止期和事务重启率方面显然优于 AVP 算法.

5 结 论

在移动实时数据库系统中,数据预取是减少系统平均访问时间和使更多事务满足截止期的一个有效办法.本文的工作主要是:在讨论影响预取数据因素的基础上,考虑数据易变性、活跃性等因素,计算数据预取代价函数,然后再考虑访问数据的事务优先级、数据流行性等因素,使预取的数据尽量满足事务的截止期.最终确定应该预取到 MC 本地缓存的数据集合.

然而,数据预取需要消耗能量.在无线广播环境中,能量是十分有限的.我们提出的预取算法没有考虑能量的消耗,我们下一步将研究数据预取策略在支持移动实时事务处理的同时降低移动客户机数据管理的能耗.

参 考 文 献

[1] Hung Jen-Jou, Leu Yungho. Efficient index caching for data dissemination in mobile computing environments. The Journal of Systems and Software, 2006, 79(1): 93-106

[2] Kumar Ajey, Misra Manoj, Sarje A K. A predicted region based cache replacement policy for location dependent data in mobile environment//Proceedings of the 2nd Annual International Conference on Wireless Communications, Networking and Mobile Computing. Wuhan, China, 2006: 1-4

[3] Peng Wen-Chih, Chen Ming-Syan. Design and performance studies of an adaptive cache retrieval scheme in a mobile computing environment. IEEE Transactions on Mobile Computing, 2005, 4(1): 29-40

[4] Tuah N J, Kumar M, Venkatesh S. Resource-aware speculative prefetching in wireless networks. Wireless Networks, 2003, 9(1): 61-72

[5] Savvas Gitzenis, Nicholas Bambos. Efficient data prefetching for power-controlled wireless packet networks//Proceedings of

the 1st Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous'04). Boston, MA, USA, 2004: 64-73

[6] Stylianos Drakatos, Niki Pissinou, Kia Makki, Christos Douligeris. A context-aware cache structure for mobile computing environments. The Journal of Systems and Software, 2007, 80(7): 1102-1119

[7] Lee V C S, Lam Kwok-Wa, Son S H, Chan E Y M. On transaction processing with partial validation and timestamp ordering in mobile broadcast environments. IEEE Transactions on Computes, 2002, 51(10): 1196-1211

[8] Lee V C S, Lam Kwok-Wa, Son S H. Real-time transaction processing with partial validation at mobile clients//Proceedings of the Real-Time Computing Systems and Applications. Cheju Island, Korea, 2000: 473-477

[9] Liu Yun-Sheng, Li Guo-Hui. Loading of a real-time main memory database. Journal of Software, 2000, 11(6): 829-835(in Chinese)
(刘云生,李国徽.实时内存数据库的装入.软件学报,2000,11(6):829-835)

[10] Liu Yun-Sheng. Real-time database system. Computer Science, 1994, 21(3): 42-46

[11] Cao Guo-Hong. Proactive power-aware cache management for mobile computing system. IEEE Transactions on Computers, 2002, 51(6): 608-621

[12] Cao Guo-Hong. A scalable low-latency cache invalidation strategy for mobile environments. IEEE Transactions on Knowledge and Data Engineering, 2003, 15(5): 1251-1265

[13] Grassi V. Prefetching policies for energy saving and latency reduction in a wireless broadcast data delivery system//Proceedings of the 3rd ACM International Workshop Modeling, Analysis, and Simulation of Wireless and Mobile System. Boston, MA, USA, 2000: 77-84

[14] Yin Liang-Zhong, Cao Guo-Hong. Adaptive power-aware prefetch in wireless networks. IEEE Transactions on Wireless Communications, 2004, 3(5): 1648-1658

[15] Jiang Zhi-Mei, Kleinrock Leonard. Web prefetching in a mobile environment. IEEE Personal Communications, 1998, 5(5): 25-34

[16] Song Hui, Cao Guo-Hong. Cache-miss-initiated prefetch in mobile environments. Computer Communications, 2005, 28(7): 370-381

[17] Song Hui, Cao Guo-Hong. On improving the performance of cache invalidation in mobile environments. Mobile Networks and Applications, 2002, 7(4): 291-303

[18] Shim J, Scheuermann P, Vingralek R. Proxy cache design: Algorithms, implementation and performance. IEE Transactions on Knowledge and Data Engineering, 1999, 11(4): 549-562

[19] Li Guo-Hui, Yang Bing. Efficient optimistic concurrency control for mobile real-time transactions in a wireless data broadcast environment//Proceedings of the 11th IEEE International Conference on Embedded and Real-time Computing Systems and Applications. Hong Kong, China, 2005: 17-19

[20] Breslau L, Fan P, Philips G, Shenker S. Web caching and Zipf-like distributions: Evidence and implications//Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Society. New York, USA, 1999: 126-134

[21] Shanmugasundaram J, Nithrakashyap A, Sivasankaran R, Ramamritham K. Efficient concurrency control for broadcast environments//Proceedings of the ACM SIGMOD International Conference on Management of Data. Philadelphia, Pennsylvania, USA, 1999: 85-96

[22] Li Guo-Hui, Chen Ji-Xiong et al. Relative time correctness for mobile real-time transactions and its implementation. Chinese Journal of Computers, 2005, 28(1): 137-141 (in Chinese)
(李国徽,陈基雄等.移动实时事务的相对时间正确性及其实现.计算机学报,2005,28(1):137-141)

LI Guo-Hui, born in 1973, Ph. D., professor, Ph. D. supervisor. His main research interests include active databases, real-time database, mobile computing and synchronization for parallel/concurrent programming.

YANG Bing, born in 1975, Ph. D. candidate. His main research interests include mobile real-time database systems.

CHEN Hui, born in 1976, Ph. D. candidate. His main research interests include data stream, data engineer.

DU Jian-Qiang, born in 1968, professor. His main research interests focus on medical imaging.

Background

The work in this paper is supported by the National Science Foundation of China under grant No. 60873030, the National High-Tech Research and Development Plan of China under grant No. 2007AA01Z300. The research projects aim to get some efficient methods to support real-time transaction processing in mobile distributed computing environments. Till now authors have analyzed environmental factors that have direct impacts on the distributed and mobile real-time transaction processing. A relative timing correctness criterion and the strategy to implement it have been investigated. Based on the checkpoint-based transaction execution model that may benefit transactions to satisfy their timing constraints, a distributed cooperative real-time transaction model for a kind of distributed real-time database applications is proposed. Considering the undependable and asymmetric wireless communication channel and the limited power capacity, a data broadcast strategy to support mobile real-time transaction processing is under consideration. This paper focuses on data prefetching to support real-time transaction in mobile environment.