移动环境下的低开销自动数据收集算法*

周 桓[1,2], 李 京[1], 冯玉琳[1,2]

[1](中国科学院 软件研究所 计算机科学重点实验室,北京 100080);
[2](中国科学院 软件研究所 软件工程技术研究开发中心,北京 100080)
E-mail: {quan,lij}@otcaix.iscas.ac.cn; feng@ios.ac.cn
http://www.iscas.ac.cn

摘要: 移动计算的一个重要难题是断连操作.数据收集过程是指在断连前把用户将来可能访问的数据预先存储到本地缓存的过程.收集过程的结果将明显影响断连操作的性能.提出了一种低开销的通用数据收集算法,在访问数据时同步建立数据之间的关联,并在数据关联的基础上自动选择要收集的数据集.模拟试验结果表明,该算法可以有效地提高断连操作时的 cache 命中率,尤其适用于计算速度慢、存储容量小的手持移动设备.

关 键 词: 移动计算;cache;断连操作;数据收集;概率图
中图法分类号: TP393 文献标识码: A

随着计算机硬件技术和网络通信技术的发展,近年来,计算设备的一个重要发展趋势是小型化和移动化.便携式的计算设备,如笔记本、掌上电脑和智能手机,已经在多种应用领域中越来越广泛地得到使用.这些设备通常通过无线网络连接到移动计算系统中,但易受存储空间和电池容量等条件的限制.这些约束对移动环境中的数据访问技术造成了极大的影响[1].文献[2]概括了移动数据访问技术的最新研究进展.

移动性造成的一个重要影响是断连的经常出现.系统必须允许用户在断连时继续操作以保证计算的连续性,这种操作模式称为断连操作,它是移动计算研究中的一个重要课题.断连不仅在网络覆盖范围无法达到的情况下出现,还会在用户希望主动断开网络连接以节省通信费用或者满足其他特定的需要时出现.当用户预见到断连即将出现时,可以要求系统事先把将来可能访问的数据预取到本地,这个过程就是数据收集.移动数据访问系统对断连操作的支持通常可以分为若干阶段来完成[3],数据收集是其中的第 1 个阶段,它的结果会直接影响到后续阶段操作的性能以及系统的可用性.

早期的移动数据访问系统主要采用用户手工指定数据集的方法来确定将要收集的数据[3].这种方法虽然具有较高的准确性,但要求用户对系统有比较深入的了解.随着手持设备的计算能力的增强以及对多媒体的支持,越来越多的应用开始面向手持设备.这些应用的用户往往只有有限的计算机专门知识,无法确定自己将要访问的数据.因此,自动预测是数据收集中一门必不可少的技术.

最早的关于自动预测技术的系统研究见于 Kunning 和 Popek 的工作[4].他们针对移动文件访问系统提出了基于语义距离(semantic distance)的预测方法.文献[5]提出了一种采用数据挖掘(data mining)技术来抽取关联规则的自动收集算法.上述自动预测算法都具有比较大的额外计算或存储开销.本文提出一种低开销的通用预测算法.它根据用户的访问历史探测不同数据之间的逻辑关联,然后根据数据之间的关联程度把数据对象分成相互独立的组,再推测断连时最有可能访问的簇.

* 收稿日期: 2000-12-19; 修改日期: 2001-03-09
基金项目: 国家自然科学基金资助项目(69833030)
作者简介: 周桓(1976－),男,安徽芜湖人,博士,主要研究领域为移动计算,分布式系统;李京(1966－),男,江苏无锡人,博士,研究员,博士生导师,主要研究领域为分布对象计算,软件工程方法学,移动计算;冯玉琳(1942－),男,江苏泰县人,博士,研究员,博士生导师,主要研究领域为软件工程,形式化方法,分布对象计算.

我们以 Web 访问记录测试了算法的性能,并与缓存系统中通常使用的 LRU(least recently used)算法进行了比较.结果表明,该算法可以有效地提高缓存的命中率,尤其适用于缓存较小的情况.这些特点使其特别适合于在诸如掌上电脑和个人数字代理的小型手持设备上的应用.

本文第 1 节详细描述了基于概率图的数据收集算法.第 2 节给出了模拟试验的方法和结果.第 3 节把我们的工作和其他相关工作进行了比较.第 4 节对本文进行了总结.

1 算法设计

自动预测算法的基本思想是根据用户的访问历史来确定用户的访问模式以推测将来最有可能访问的数据集.我们的设计目标是一个不需要用户和其他计算节点(如数据服务器)干预的客户端自动预测算法.它可以很方便地嵌入到移动数据访问系统的客户端缓存管理器中完成数据收集的功能.整个算法分为两个部分:第 1 步,随着用户的数据访问过程同步建立数据对象之间的逻辑关系,我们称之为关联度;第 2 步,对数据对象进行分组并选择合适的组预先存储到本地的缓存中,它由用户准备断连时启动.

下面我们分别描述算法的各个步骤,然后分析算法的时间和空间复杂度.

1.1 关联度的构造

根据访问历史来建立数据对象之间关联度的方法是建立在用户行为语义局部性(semantic locality)的思想上的[4],即当用户访问了数据对象 A 后,与 A 关联度大的对象在不久的将来也会被访问.因此,我们通过研究数据对象的访问序列来推测它们之间的关联度,给定访问序列 $f_1,f_2,f_3,\ldots,f_n$,数据对象 f_i 到 f_j 的关联度是一个量化的值,它表示在同一个作业期中,在访问了 f_i 之后,f_j 被访问到的可能性.显然,在这个定义下,数据对象之间的关联度不具有对称性.

关联度可以通过同步和异步两种方式来建立.在异步方式下,数据对象的关联度是在断连前根据访问历史静态地完成的,文献[4,5]中采用的都是这种方式.在同步方式下,关联度是在用户访问数据的同时动态地建立的,虽然这种方式可以减少数据收集过程在时间和空间上的开销,但是为了不干扰用户的正常操作,它要求算法有较高的运行效率.为了减少数据收集时的运行开销,我们采用同步方式来建立数据对象之间的关联度.

为了提高文件系统预取操作的准确率,Griffioen 和 Appleton 给出了一种简单而有效的概率图(probability graph)算法来建立文件之间的逻辑关系[6].给定一个访问序列 $f_1,f_2,f_3,\ldots,f_n$,其中 f_i 代表一个数据对象,f_i 的一次出现代表对该数据对象的一次引用,概率图算法把其中的每个数据对象 f_i 定义为一个节点,对于在给定的"向前看间隔"内对其他数据对象 f_j 的每次引用,算法都建立一条从 f_i 指向 f_j 的边,如果已经存在从 f_i 指向 f_j 的边,则将这条边的关联度加 1.因此,通过对访问历史的一次扫描,概率图算法就可以建立一个有向图来描述所有被访问数据对象之间的关联度.概率图算法惟一的参数就是向前看间隔,它的值决定了在多长时间内同时访问过的数据对象被认为是相关的.

概率图算法具有速度快、存储开销小的特点,非常适用于同步构造数据对象之间的关联度.在我们的设计中,访问历史被分为若干个不相交的作业期(session),每个作业期代表了用户的一次完整的数据访问过程,例如从打开浏览器到关闭浏览器的整个 Web 浏览过程.数据对象之间的关系不能跨越作业期,即每当一个新的作业期开始时要清空向前看窗口,但仍然使用同一个概率图.因此,给定访问历史

$$(f_1,f_2,f_1,f_3,f_4),(f_2,f_3,f_4,f_1,f_3),(f_1,f_2,f_1,f_3,f_5),$$

其中括号内的访问序列为同一个作业期内的访问序列,在向前看间隔为 1 的情况下得到的概率图如图 1 所示.

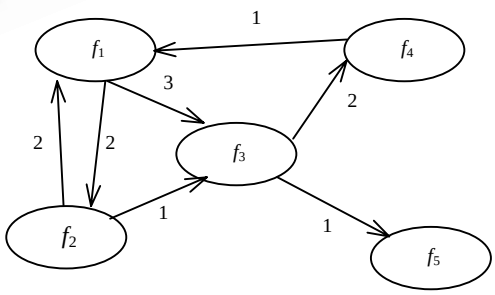


Fig.1 A probability graph example

图 1 概率图示例

1.2 分组算法

概率图可以量化地表示数据对象之间的关联度,但是在选择要收集的数据之前,需要把数据对象分组以保

证用户可以不间断地访问多个逻辑上相关联的数据对象.在概率图的基础上,分组算法的目的是把整个数据对象集合按照关联度分割成为若干个组,每个组内的数据对象具有比较紧密的逻辑关系.这样,当其中某一个数据对象被访问后,其他数据对象有很大的可能在同一个作业期内也被访问.

对于概率图中一条从数据对象 A 指向数据对象 B 的边,如果其关联度和所有从 A 出发的边的关联度的总和之比大于一个事先定义的阈值,我们称 B 和 A 是紧密关联的.这种紧密关联的定义是不对称的,但是它可以反映用户访问数据对象的方向性.例如,编译程序总是首先分析用户指定的源文件,然后才访问源文件中引用的其他文件,如 C 语言的头文件等;在浏览 Web 信息时通常也是首先访问站点的入口页面,然后才按照链接一步步地浏览感兴趣的页面.

通用的分组算法通常具有 $O(n^2)$的时间复杂度[7],对于计算能力较弱的手持设备来说,这些算法的计算开销太大.为了能够对概率图进行一次扫描就得出分组结果,我们设计分组算法如下.首先确定一个“入口数据对象”列表,然后对于每个入口数据对象执行下列操作:

(1) 创建一个组 ψ,并把数据对象 A 添加到该组中;

(2) 对于每个和 A 紧密关联的数据对象 B,执行第(3)、第(4)中的一步操作;

(3) 如果 B 已经属于某个组,则跳过 B,处理下一个数据对象;

(4) 否则,以 B 为起始点递归调用分组算法.

其中,入口数据对象可以通过多种信息得到,例如,根据数据对象的访问次数等.在实际情况中我们发现,仅根据数据对象的访问顺序就可以得到比较好的结果.

1.3 选择算法

数据收集算法的最后一步是在从分组算法中得到的候选组中选择用户最有可能访问的组并预先存储到本地缓存中.我们的算法主要面向整个作业期的断连操作,即移动客户在发生断连后才开始新的作业期.因此我们无法通过作业期内的信息来选择组.从用户以前访问的作业期中可以得到的启发信息主要有以下几种:

(1) 组中数据对象的总共访问次数(TREF);

(2) 组中数据对象的平均访问次数(AREF);

(3) 组中起始点数据对象的访问次数(SREF);

(4) 组中数据对象的平均字节访问次数(BREF).

其中 TREF 倾向于选择数据对象较多的组;BREF 倾向于选择占用缓存较小的组;AREF 和 SREF 倾向于选择用户最经常访问的组.

与使用固定设备的用户相比,使用手持设备的用户访问数据往往更具有明确的目的,如定期浏览电子邮件、股票信息以及少数几个固定的网站等等.因此,选择算法应该优先选择用户经常访问的组.由于 SREF 对起始点的选择有很大的依赖性,我们以 AREF 作为选择组的标准.

我们通过试验对上述 4 种方法进行了比较,试验结果显示,AREF 优于其他 3 种.这和我们的分析是一致的.

1.4 算法描述和复杂度分析

算法 **1**. 概率图的构造.

初始化概率图 Graph 和窗口 Window 为空; //这里 Window 是已访问对象窗口,记录 m 个最近访问的节点,m 是向前看间隔.

```
foreach (请求 request) {
    new=search(Graph, request);
    if (new==NULL) then new=create(request);
    foreach (Window 中的节点 old) {add_arc(old,new);}
    Window 向前滑动一格,并把 new 添加到空格中;
}
```

算法 2. 数据收集过程.

```
foreach (Graph 中的节点 node) {
    if (node.clustered !=true) then
        cluster(node, null);
}
把组按对象的平均访问次数降序排列;
while (cache size>0) {
        从列表中弹出第 1 个组;
        并把组中的对象加入到缓存中;
}
其中递归过程 cluster 描述如下:
function cluster (node, group) {
    if (node.clustered == true) then 返回;
        把节点 node 添加到组 group;
        node.clustered = true;
        foreach (从 node 发出的边 arc) {
            if (arc.value/node.value>阈值) then
                cluster(arc.dest, group);
        }
}
```

概率图的构造过程是与用户的访问过程同步进行的.对于用户的每次访问,算法需要在概率图中搜索所访问数据对象对应的节点.我们用散列表来存储概率图的节点,因此处理一个访问记录的时间复杂度为 $O(1)$.分组算法只需要对概率图节点扫描一次就可以完成,时间复杂度为 $O(n)$,组的排序过程的复杂度为 $k\log(k)$,因此,数据收集过程的时间复杂度为 $O(n+k\log(k))$,其中 n 和 k 分别为节点和组的数目,$k<n$.

算法为访问历史中的每个数据对象创建一个节点,只有关联度不为 0 的数据对象之间才有边,所以存储概率图的空间复杂度为 $O(n+e)$.我们研究的问题所得到的图是稀疏图,因此实际的空间复杂度接近 $O(n)$.保存分组信息所需的空间也是 $O(n)$.

2 试验方法和结果

为了测试算法的性能,我们在 Linux 平台上实现了基于概率图的数据收集算法和 LRU 算法,并以真实的 Web 访问记录作为原始访问记录比较了它们在断连操作时的缓存命中率.

2.1 试验方法

试验原始数据来自 http://ita.ee.lbl.gov,是从 Cunha 等人的工作[8]中得到的.通过修改客户端浏览器,他们记录了从 1994 年 11 月~1995 年 5 月这段时间内 Boston 大学计算机系实验室中用户 Web 访问的历史记录.整个原始记录由一组日志文件组成,每个日志文件代表一个用户的一个作业期.文件中的每一条记录代表一次 Web 对象请求,记录了请求发出时间、被请求对象的 URL(uniform resource locator)以及对象大小等相关信息.由于浏览器缓存命中的请求也包含在日志文件中,因此它是关于用户访问历史的完整记录,非常适用于模拟测试.

在模拟测试之前,我们对原始记录进行了预处理.由于程序动态生成的页面通常不能被缓存,所以删除了所有向 CGI 程序发送的请求和带参数的请求,同时我们还删除了所有非 HTTP 协议的请求.由于这些请求只占总请求数的 4.4%,我们相信这个预处理过程不会对结果造成实质性影响.

模拟程序以用户访问作业期为单位工作,它有两种状态:连接状态和断连状态.在连接状态中,模拟程序读入作业期的每个访问记录并构造概率图,这是一个学习过程;在进入断连状态时,模拟程序先执行数据收集算法

构造本地缓存,然后完全根据本地缓存的内容来响应作业期中的请求.我们以断连操作时的缓存命中率来比较算法性能的优劣.

我们首先把日志文件按用户进行分组,然后从中选取了作业期最多的 10 个用户进行测试.在整个记录跨越的时间段内,这些用户的作业期数都在 50 个以上.其中第 1 个作业期被用来初始化数据收集算法,对于剩下的每个作业期,模拟程序首先进入断连状态测试数据收集算法的缓存命中率,然后再进入连接状态学习该作业期.这种方式既可以保证学习过程的连续性,又可以最大限度地测试算法在各种断连情况下的性能.

在建立关联度时,由于一个 Web 页面通常包含一个或多个内嵌文件,如内嵌声音和图像文件等.对这些内嵌文件的访问总是紧跟在对页面的访问之后,这会影响页面之间的逻辑关系的构造.这个问题可以通过把内嵌文件和页面组合为一个整体,算法以这些整体数据对象为单位构造概率图的方法来解决.在不影响试验结果的情况下,为了简化实现,我们忽略了对内嵌文件的请求.

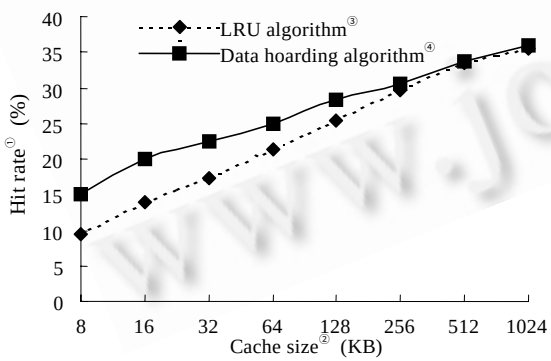
2.2 试验结果

2.2.1 数据访问命中率比较

在上述试验方法下,首先每个用户计算出断连操作时的平均访问命中率,然后对这 10 个用户求其算术平均值,得到总的平均命中率.图 2 比较了基于概率图的数据收集算法和 LRU 算法在不同缓存大小下的数据访问命中率.由于原始数据是在有线网络下获取的,因此,每个作业期中的用户都有可能访问新的数据文件,即历史记录中没有出现的文件.我们在假定缓存无穷大的情况下运行模拟算法得到了命中率的理论最优值作为参考数据,当缓存大小超过 512K 时,所有用户的访问命中率都接近于理论上的最优值.结果表明,基于概率图的数据收集算法在各种缓存大小时的访问命中率均优于 LRU 算法.

试验的另一个结果是,基于概率图的数据收集算法在小缓存下的优越性.通过对原始记录的分析,我们发现在大多数作业期内,用户所访问的所有非内嵌文件大小总和在 100K 以内.因此我们着重考察缓存大小在 100K 以内的结果.当缓存大小为 16K,32K,64K 和 128K 的时候,基于概率图的数据收集算法在 LRU 算法的基础上分别有 45%,29%,17%,12%的提高.由于我们在设计时考虑的一个重要因素就是手持设备存储空间的限制,这个结果表明算法达到了预期的设计目标.

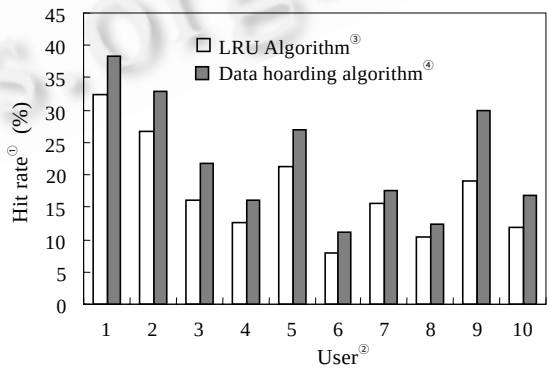
图 3 列出了缓存大小为 32K 时每个用户命中率的比较.总的来看,基于概率图的数据收集算法对用户的访问命中率都有所提高.但由于原始记录中不同用户的访问模式差别很大,因此算法对不同的用户提高的幅度不一样,其中对用户 9 的效果最好,为 57%,对用户 7 的效果最差,为 13%.



①命中率,②缓存大小,③LRU 算法,④数据收集算法.

Fig.2 Hit rate vs. cache size

图 2 算法命中率随缓存大小的变化



①命中率,②用户,③LRU 算法,④数据收集算法.

Fig.3 Hit rate in 32K cache

图 3 32K 缓存下算法命中率的比较

2.2.2 参数对算法的影响

基于概率图的数据收集算法的两个参数是向前看间隔和关联度阈值.图 4 显示了它们对算法命中率的影响,其中缓存大小设为 32K.向前看间隔决定了在多长时间段内访问的数据对象是相关的,增大它的值会在更多不相关的数据对象之间建立关联,从而导致概率图的扩大.从图 4 的实线可以看出,命中率随着向前看间隔的增大而缓慢降低.因此,在试验中我们选择向前看间隔为 1.

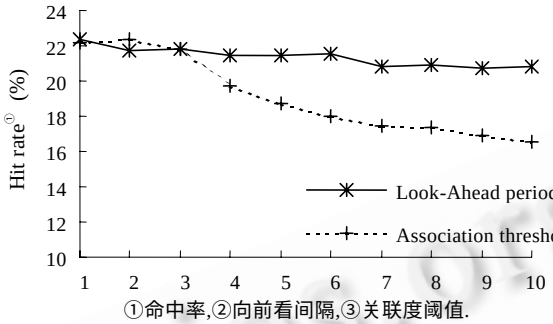


Fig.4 Hit rate vs. parameters

图 4 参数对命中率的影响

关联度阈值决定哪些边连接的数据对象是紧密关联的,它的值会直接影响组的大小.关联度阈值的选择在一定程度上和应用相关,但我们在试验中发现,当阈值小于 30%时,命中率随阈值的降低而单调减.如图 4 中虚线所示,关联度阈值的横坐标是阈值的另一种表达形式,它和阈值的换算公式为

阈值=1/(横坐标值+1).

3 相关工作

数据收集是移动数据访问系统中的一个关键技术,也是移动计算研究中的难点和热点之一.

Coda 移动文件系统主要采用用户指定的方法来进行数据收集[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集结合起来构成断连前的本地缓存.虽然这种方法可以满足长期断连的需要,但它要求用户指导收集过程,从而增加了用户的负担.Tait 等人通过构造程序树(program tree)的方式提出了一种半自动化的数据收集方法[9].通过检测应用程序和数据文件之间的调用关系,该算法把一个项目中相关的文件组织成一棵或多棵树,用户通过图形化工具选择需要收集的树.

Kenning 等人最早开始研究移动环境中的自动数据收集算法.它们提出通过计算数据之间的语义距离来刻画数据之间的逻辑关联,并针对文件访问开发了 Seer 自动预测缓存系统[4],其中两个文件之间的语义距离通过计算引用序列中对这两个文件的引用之间的其他引用数目得到.在文献[5]中,作者把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,他们提出了一种通用的自动数据收集算法.

我们的思想部分来自分布式系统中对预取缓存技术的研究,在概率图算法[6]的基础上,我们提出了一个完整的自动数据收集算法.与上述研究相比,该算法不仅以提高断连操作的访问命中率作为目标之一,还着重考虑了算法开销对移动设备,尤其是小型手持设备可能造成的影响.

4 结 论

移动性对传统的分布式系统造成的一个深刻影响就是计算节点断连的经常出现.数据收集过程是支持断连操作的第 1 步,它的性能直接影响到用户操作的成功率.在研究自动数据收集算法时不仅要考虑断连操作的成功率,还要考虑算法开销对低计算和低存储能力的移动计算设备造成的影响.本文提出了一种基于概率图的移动环境数据收集算法.该算法不仅具有接近线性的时间和空间复杂度,而且通过同步建立关联度的方式进一步减少了断连前数据收集过程的运算时间.我们以用户 Web 访问记录测试了算法的性能.试验结果表明,和 LRU

算法相比,基于概率图的数据收集算法可以有效地提高断连操作中数据访问的命中率,尤其适用于缓存较小的情况,是一种非常实用的算法.

References:

[1] Satyanarayanan, M. Fundamental challenges in mobile computing. In: Burns, J., Moses, Y., eds. Proceedings of the 15th ACM Symposium on Principles of Distributed Computing. New York, NY: ACM Press, 1996. 1~7.

[2] Jing, J., Helal, A., Elmagarmid, H. Client-Server computing in mobile environments. ACM Computing Surveys, 1999,31(2): 117~157.

[3] Kistler, J., Satyanarayanan, M. Disconnected operation in the coda file system. ACM Transactions on Computer Systems, 1992, 10(1):213~225.

[4] Kuenning, G., Popek, G. Automated hoarding for mobile computers. In: Banatre, M., Levy, H., eds. Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP-16). New York, NY: ACM Press, 1997. 264~275.

[5] Saygin, Y., Ulusoy, O., Elmagarmid, A. Association rules for supporting hoarding in mobile computing environments. In: Joshi, A., Helal, A., eds. Proceedings of the 10th International Workshop on Research Issues in Data Engineering. San Diego, CA: IEEE Computer Society, 2000. 71~78.

[6] Griffioen, J., Appleton, R. Reducing file system latency using a predictive approach, In: Proceedings of the 1994 Summer USENIX Technical Conference. 1994. http://www.usenix.org/publications/library/proceedings/bos94/index.html.

[7] Zupan, J. Clustering of Large Data Sets. Hertfordshire, UK: Research Studies Press, 1982.

[8] Cunha, C., Bestavros, A., Crovella, M. Characteristics of WWW traces. Technical Report, TR-95-010, Department of Computer Science, Boston University, 1995.

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

A Low-Cost Automatic Data Hoarding Algorithm for Mobile Environment*

ZHOU Huan[1,2], LI Jing[1], FENG Yu-lin[1,2]

[1](*Key Laboratory for Computer Science, Institute of Software, The Chinese Academy of Sciences, Beijing* 100080, *China*);

[2](*Software Engineering Technology Center, Institute of Software, The Chinese Academy of Sciences, Beijing* 100080, *China*)

E-mail: {quan,lij}@otcaix.iscas.ac.cn; feng@ios.ac.cn

http://www.iscas.ac.cn

Abstract: Disconnected operation is a challenging issue in mobile computing. Data hoarding is the process of prefetching important data into local cache for future operation. The result of hoarding process will dramatically affect the performance of disconnected operation. A low-cost general hoarding algorithm is introduced to exploit relationship among the data items simultaneously when processing data requests and to select data for hoarding automatically. Simulation results show that this algorithm effectively improves cache hit rate in disconnected operation. And it is especially applicable for handheld mobile devices with low storage capacity and slow computing speed.

Key words: mobile computing; cache; disconnected operation; data hoarding; probability graph

* Received December 19, 2000; accepted March 9, 2001
Supported by the National Natural Science Foundation of China under Grant No.69833030