基于线性链表的模糊关联规则挖掘

刘青宝 王文熙 王万军
(国防科学技术大学信息系统工程重点实验室 长沙 410073)

摘 要 为改进现有模糊关联规则挖掘算法的不足,提出了一种基于线性链表的模糊关联规则挖掘算法。算法利用线性链表只存储有用的事务数据库信息,并不断利用前期的运算结果对之进行简化,减少了数据的存储开销及扫描时间,降低了算法的时间复杂度,提高了算法的效率。比较分析以及实验表明,该算法对于挖掘模糊关联规则是快速而有效的。

关键词 数据挖掘,模糊关联规则,线性链表

Linear Linklist Based Algorithm for Fuzzy Association Rule Mining

LIU Qing-bao WANG Wen-xi WANG Wan-jun
(Science and Technology on Information System Engineering Laboratory, National University of Defense Technology, Changsha 410073, China)

Abstract In order to improve the efficiency of existing fuzzy association rule mining algorithms, we presented a linear linklist based algorithm for fuzzy association rule mining. Utilizing the linear linklist our algorithm only records the information of the tran-sactions which are useful for counting the support of the frequent itemset, and simplifies the transactions information according to the previous results, which reduces the cost of data storage and increases the running efficiency. Experiments demonstrate that our method is efficient in fuzzy association rule mining.

Keywords Data mining, Fuzzy association rule, Linear linklist

1 引言

由于客观世界的多样性、复杂性以及人类认识规律的限制,许多事物不便于用精确的、确定的概念来表达,而模糊关联规则可以很好地解决这个问题。为了解决定性知识发现的问题,Chan 和 Au 对关系数据库中的数值属性和分类属性应用了语义项,提出了模糊关联规则,并给出了发现模糊关联规则的 F-APACS 算法[1]。算法中,他们利用邻近差异分析和模糊性来获取最小支持度和置信度,而不是由用户来确定。Hong 等人结合模糊集概念和 Apriori 挖掘算法提出了一种从给定的事务数据库中发现感兴趣的模糊关联规则的算法[2],并随后提出了基于 AprioriTid 的模糊关联规则挖掘算法[3]。Lee 等人提出了挖掘具有多重最小支持度的模糊关联规则算法[4]。Papadimitriou 等人提出一种基于 FP-tree 的模糊关联规则挖掘算法[5]。Lin 等人同样提出了一种新的基于 FP-tree 的模糊关联规则挖掘算法[6]。

在模糊关联规则的挖掘方法中,使用较为广泛的有基于 Apriori 的,如 FDMA 算法[2,3];有基于 FP-tree 的,如 Fuzzy FP-tree 算法[6]。其中,基于 Apriori 思想的算法要来扫描数据库的次数与最长的频繁项目集的长度相等,其最典型的缺点是每产生一次频繁项集都要重新扫描一次事务数据库,且每次循环扫描的数据量都不会减少,也就是说算法没有很好地利用前期运算结果,挖掘效率很低。而基于 FP-tree 的模糊关联规则的算法尽管在效率上有了很大提高,但是其处理模糊属性能力较弱,会丢失部分有用信息,不能得到全面的挖掘结果。

为了改进现有模糊关联规则挖掘算法,本文结合 Apriori 算法和 FP-tree 结构的特点,在线性链表的数据结构基础上,提出了一种基于线性链表的模糊关联规则挖掘算法。算法通过线性链表只存储有用的事务数据库信息,并能不断利用前期的运算结果对之进行简化,减少了数据的存储开销及扫描时间,提高了算法的效率。接下来的部分,首先介绍相关理论,然后提出基于线性链表的模糊关联规则挖掘算法,并做性能比较分析和实验验证,最后对本文进行简要总结。

2 基于线性链表的数据结构

线性链表是一种实现存储空间动态管理的链式存储方式。线性链表的存储结构不需要占用一整块事先分配的、固定大小的存储空间,而且其插入、删除、遍历等操作具有简单高效的特点,因此可将其用于提高关联规则挖掘的效率。本节提出一种适用于模糊关联规则挖掘的基于线性链表的数据结构。结构借鉴了 FP-tree 的结构特点,只存储对于计算频繁项集有用的事务信息,提高了运算效率,而且能够很好地处理模糊属性。

2.1 基于线性链表的结构设计

本文提出的基于线性链表的结构如图 1 所示,主要包括

到稿日期:2011-04-07 返修日期:2011-07-14 本文受国家自然科学基金(70771110)资助。

刘青宝(1967-),男,博士,研究员,主要研究方向为数据仓库技术和数据挖掘,E-mail:qbliulidd@263.net;王文熙 男,硕士,主要研究方向为数据仓库技术和数据挖掘;王万军 男,硕士,主要研究方向为数据仓库技术和数据挖掘。

项头表和线性链表两部分。该结构采用项头表存储候选项集或频繁项集，使用线性链表存储有用的模糊化后的事务数据信息。线性链表的遍历访问是通过项头表上的结点链头 Link 定位进行的。

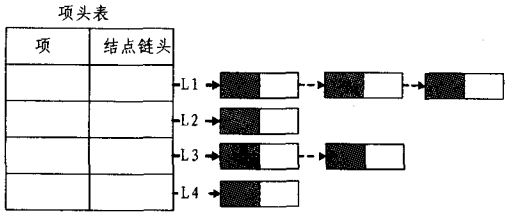

图 1　基于线性链表的存储结构

图中项头表的结构与 FP-tree 算法中所采用的项头表结构相同，都由两个域组成：项集名称和结点链头。其中结点链头指向项集所对应的线性链表。与 FP-tree 结构不同的是：项头表中项集并非按照支持度大小排序，而是按照字典顺序排序，这是为了方便此后在模糊频繁项集挖掘中的操作。所定义的线性链表 L，存储的是某项集在事务数据库中的隶属度取值分布信息。线性链表的结点分为数据域和指针域。其中，数据域存储的数据形如$(i, f^{(i)})$，i 为事务的序号，$f^{(i)}$ 为第 i 条事务上此项集的隶属度取值。结点按其数据域中事务的序号 i 升序排列。而指针域则存储指向后续结点的指针，若无后续结点，则指针域为 NULL。

显然，同一线性链表中的结点所存储的是链头对应项集在某条事务上的隶属度取值信息，且后续结点的事务序号永远比其前驱结点的事务序号要大。同时，此项集在事务数据库中所有的隶属度取值信息均能反映在其对应的线性链表上来。因此，所提出的线性链表的结构能够完全表达出事务数据库的数据信息及其分布，保证了挖掘结果的正确性。

本文所用到的链表为单向链表，只设置一个指针域，用以指向其直接后续结点，并通过头结点的指针唯一标识该链表。

2.2　基于线性链表的基本操作

为了能够高效地挖掘模糊频繁项集，根据上面所定义的线性链表结构，本文定义了线性链表的以下几种基本操作：SUM 操作、INTERSECT 操作以及 DELETE 操作。

(1)SUM 操作。从结点链头开始遍历线性链表，对链表 L 中所有结点的数据域存储的数据的第二分量求和，记为 SUM(L)，表示对应项集的支持度。

(2)INTERSECT 操作。对于链表 $L1$ 和 $L2$，分别将指针指向第一个结点。比较指针所指向的两结点：如果存储的事务序号 i 相同，则新建结点插入到链表 L，且该结点存储的事务序号亦为 i，其存储的数据的第二分量取值为所比较的两结点存储的数据的第二分量的较小值，并同时将两指针后移；否则，将事务序号较小的结点所在链表的指针往后移，然后对两指针指向的结点再进行比较，直到某一链表的指针为空。最终，得到新项集对应的新链表 L，记为 $L=L1 \wedge L2$，表示此新项集在事务数据库中的隶属度取值分布信息。此操作的目的是生成新项集所对应的线性链表。

(3)DELETE 操作。此操作的目的是释放线性链表所占用的内存空间，该操作在频繁项集挖掘中用于删除无用的线性链表。

3　基于线性链表的模糊关联规则挖掘算法

本节提出了基于线性链表的模糊关联规则挖掘算法来挖掘模糊关联规则。算法通过线性链表只存储有用的事务数据库信息，并能不断利用前期的运算结果对之进行简化，减少了数据的存储开销及扫描时间，降低了算法的时间复杂度，提高了算法的效率。

3.1　算法基本思想

基于线性链表的模糊关联规则挖掘算法，首先通过给定的模糊隶属函数集合对数据的属性进行模糊化处理，得到模糊化数据(隶属度取值)；然后建立链表结构存储 1-项集在事务数据库中的隶属度取值分布信息；接着基于建立的链表结构挖掘模糊频繁项集；最后利用所得到的模糊频繁项集生成模糊关联规则。规则生成过程采取与 Hong 等人相同的算法[3]，但采用兴趣度阈值控制规则的生成。

3.2　算法过程描述

具体过程描述如下。

算法 1　基于线性链表的模糊关联规则挖掘算法

输入：数值型事务数据库 $T=\{t_1, t_2, \cdots, t_n\}$，隶属函数集合，最小支持度 α，最小置信度 λ，最小兴趣度 min_interest。

输出：模糊关联规则集合。

Step1　将每条事务数据 $t_i(i=1\sim n)$，对于每个项目 $I_j(j=1\sim m)$，利用对应的隶属度函数将数量型数值 $v_j^{(i)}$ 转换为一个模糊集合 $f_j^{(i)}$，表示为

$$f_j^{(i)}=\{\frac{f_{j1}^{(i)}}{R_{j1}}+\cdots+\frac{f_{jl}^{(i)}}{R_{jl}}\}$$

式中，R_{jk} 是项目 I_j 的第 k 个语义项，$f_{jk}^{(i)}$ 是 $v_j^{(i)}$ 在区域 R_{jk} 上的模糊隶属度值，$l(=|I_j|)$ 是 I_j 的语义项数。

Step2　建立项头表 HC1(Region, Link)。字段 Region 用于存储 R_{jk} $(1\leqslant j\leqslant m, 1\leqslant k\leqslant |I_j|)$，并对之分别建立相应的线性链表。对于每个语义项 R_{jk}，其所对应的线性链表进行 SUM 操作。如果小于 $n*\alpha$，则删除记录及对应链表，最终得到频繁项头表 HL1。否则将之加入模糊频繁 1-项集 L1。

Step3　令 $r=1$，r 表示当前产生项集为 r-项集。

Step4　由频繁项头表 HLr 生成候选项头表 HC($r+1$)。

建立候选项头表 HC($r+1$)，其中字段 Region 用于存储 HLr. Region×HLr. Region 产生的($r+1$)-项集，两个 r-项集对应的线性链表进行 INTERSECT 操作，得到($r+1$)-项集对应的线性链表、Link 存储所得到的($r+1$)-项集对应的线性链表的头指针。

产生($r+1$)-项集的条件为：(1) HLr. Region 与 HLr. Region 中元素的 $r-1$ 项相同；(2)第 r 项为不同属性的语义项(模糊集合)；(3)其所有 r-项子集都属于集合 HLr. Region。

若无满足条件的($r+1$)-项集产生，则转到 Step7。

Step5　对候选项头表 HC($r+1$)中($r+1$)-项集对应的线性链表进行 SUM 操作。如果支持度小于 $n*\alpha$，则删除记录及对应链表，最终得到频繁项头表 HL($r+1$)。将支持度计数大于 $n*\alpha$ 的项集加入($r+1$)-模糊频繁项集 $L(r+1)$，并删除频繁项头表 HLr。

Step6　如果 HL($r+1$)=∅，则执行下一步。否则，令 $r=r+1$，重复执行 Step4－Step6。

Step7　按照以下步骤从所有的频繁 q-项集 $I_k(I_{k_1}, I_{k_2}, \cdots, I_{k_q})$ 中产生关联规则($q\geqslant 2$)：

(1)生成所有的准关联规则：

$I_{k_1} \wedge \cdots \wedge I_{k_{j-1}} \wedge I_{k_{j+1}} \wedge \cdots \wedge I_{k_q} \rightarrow I_{k_j}$，$1\leqslant j\leqslant q$

(2)按照下式计算准关联规则的置信度：

$$\frac{S_{I_k}}{S_{I_{k_1} \wedge \cdots \wedge I_{k_{j-1}} \wedge I_{k_{j+1}} \wedge \cdots \wedge I_{k_q}}}$$

(3)按照以下公式计算准关联规则的兴趣度：

$$\frac{S_{I_k}}{S_{I_{k_1}\wedge\cdots\wedge I_{k_{j-1}}\wedge I_{k_{j+1}}\wedge\cdots\wedge I_{k_q}}\times S_{I_{k_j}}}$$

Step8　输出满足最小置信度和最小兴趣度的模糊关联规则。

4　算法比较与分析

本节通过与 FDMA 及 Fuzzy FP-tree 算法在时间复杂度上的分析，证明了本文所提出的算法具有较高的效率和优越的性能。

4.1　与 FDMA 算法的性能分析比较

设定一个事务数据库 T 中有 n 条事务，事务的属性经过模糊化后共有语义项数目为 m。设每条事务中平均含有 $x*m$ 个项目；每个项目平均出现 $x*n$ 次($0<x\leqslant 1$)。又设频繁 1-项集集合平均含有 $y*m$ 个元素($0<y\leqslant 1$)。

基于以上假设，FDMA 算法的计算过程中产生候选 2-项集个数为 C^2_{y*m}，每条事务平均产生的 2-项集个数为 C^2_{x*m}，则对每条事务的扫描需要$[2\times(C^2_{x*m}+C^2_{y*m})]$次，因此计算候选 2-项集的支持度的执行次数为$[2\times(C^2_{x*m}+C^2_{y*m})\times n]$。本文方法中，产生的候选 2-项集个数亦为 C^2_{y*m}，而每个频繁 1-项集对应的线性链表平均含有 $x*n$ 个元素，对线性链表进行 INTERSECT 操作需要比较$(x*n+x*n)$次，因此计算候选 2-项集的支持度的执行次数为$[(x*n+x*n)\times C^2_{y*m}]$。

根据以上分析，FDMA 算法与本文提出的算法的时间复杂度之比(记作 θ)为

$$\theta=\frac{T(2\times(C^2_{x*m}+C^2_{y*m})\times n)}{T((x*n+x*n)\times C^2_{y*m})}。$$

θ 可简化为 $\theta\approx\frac{1}{y^2}[\frac{(x-y)^2}{x}+2y]$。若 y 大小固定，则当 $x=y$ 时，θ 取值最小，为 $2/x$。

θ 取最小值 2 时，x,y 均为 1。此时，意味着每条事务均包含所有 m 个项目，亦即每个项目在每条事务中均出现，且所有的项目均为频繁 1-项集。也就是对于最稠密的数据集和较小的支持度阈值，本文提出的算法在挖掘频繁 2-项集的效率上至少是 FDMA 算法的 2 倍。而当数据集越稀疏，支持度阈值越大，x 和 y 的取值就越小，本文提出的算法效率比 FDMA 算法就越高。

推广到频繁 k-项集的挖掘，可以预见随着 k 值的增大，数据将会变得稀疏，本文提出的算法将具有更高的效率。一般而言，若本阶段对于上一阶段的稀疏率和频繁项集率保持不变，则有

$$\theta=\frac{T(k\times(C^k_{x*m}+C^k_{y^{k-1}*m})\times n)}{T((x^{k-1}*n+x^{k-1}*n)\times C^k_{y^{k-1}*m})}$$

若 $m>>k$，则 θ 最小取值为 $\frac{k}{2}\cdot\frac{1}{x^{k-1}}\cdot(\frac{1}{x^{k(k-2)}}+1)$。显然，与挖掘频繁 2-项集相比，在挖掘频繁 k-项集上，本文提出的算法比 FDMA 算法更具效率。

4.2　与 Fuzzy FP-tree 算法的性能分析比较

Lin 等人提出的 Fuzzy FP-tree 算法为了使用树结构而只能使用属性的语义项中支持度最高的一项，这就丢失了许多有用的信息，使其只适合处理高偏度的数据(每个属性模糊化后只有一个语义项占主导地位)，具有很大的局限性。虽然其采用树结构减少了内存空间的占用，但是在挖掘频繁模式时不可避免地需要递归地创建附加数据结构，并且每当模式增长时就要创建一次，这将耗费大量的时间和空间。

本文采用的方法中，链表的结构是树结构的扩展，虽牺牲了一定的内存空间，但可以处理所有的语义项，因此本文的算法更适合处理模糊属性。

结合 4.1 节和 4.2 节，表 1 列出了这些算法与本文算法比较分析的结果。总的来说，本文的算法比基于 Apriori 的算法挖掘效率要高，比 FFP-tree 算法的模糊属性处理能力强，因此是一种有效的模糊关联规则挖掘方法。

表 1　算法比较结果

算法名称	模糊属性处理能力	挖掘效率
本文算法	较好	高
FDMA 算法	较好	低
FFP-tree 算法	差	——

5　实验结果及分析

本节以 UCI 数据集 glass[7] 中第 7 类 glass——headlamps 的数据为测试数据来验证算法的可行性，数据是对玻璃的折射率以及各种元素含量的描述。实验平台采用的计算机 CPU 为 Intel(R) CPU @2.13GHz，内存 1GB；软件平台为 Windows XP；处理程序由 Visual 2005 编写。

5.1　可扩展性测试

从数据集中分别选取 1k，2k，3k，4k 和 5k 条记录，并在支持度分别为 0.55，0.6，065 和 0.7 时进行挖掘。各自的运行时间如图 2 所示。

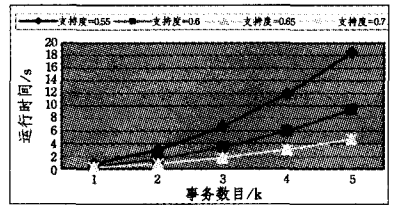

图 2　数据集事务数目及支持度与频繁项集挖掘时间的关系

图 2 显示：(1)随着事务数目的增加，挖掘频繁项集所需的时间基本呈线性增长。(2)算法效率随着支持度增大而减小，当支持度达到一定阈值时，算法效率无明显变化。这说明本文提出的算法具有很好的可扩展性。

5.2　各参数设置对挖掘规则的影响测试

实验通过对置信度、支持度、兴趣度等参数进行设置，观察各参数设置对关联规则挖掘数目的影响。实验结果如图 3—图 5 所示。

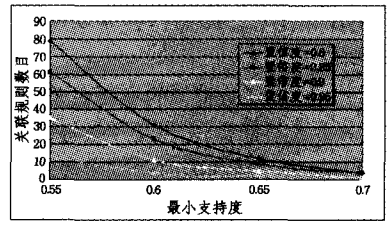

图 3　最小支持度与关联规则数目的关系

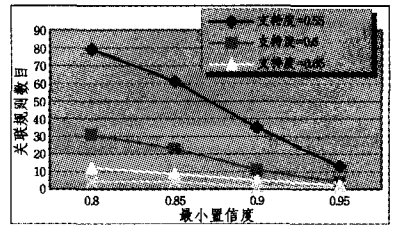

图 4　最小置信度与关联规则数目的关系

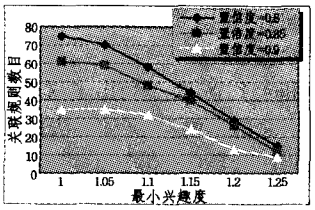

图 5　最小兴趣度与关联规则数目的关系

图 3 描述了最小支持度对关联规则产生数目的影响。在相同的置信度水平下，随着最小支持度的增大，发现规则的数目不断减少；当最小置信度处于一个较低的水平时，曲线的变化比较陡，说明此时最小支持度对最终产生的关联规则数目影响较大；而当最小置信度处于一个较高的水平时，曲线的变化比较平缓，说明此时最小支持度对产生规则的数目影响较小。随着最小支持度的取值趋向 1，产生的规则数目趋向 0。

图 4 描述了最小置信度对关联规则产生数目的影响。在相同的最小支持度水平下，随着最小置信度的增大，所得到的关联规则数目不断减少；当最小支持度设置较小时，曲线变化比较陡，此时最小置信度对最终产生的关联规则数目具有较大的影响；而当最小支持度设置较大时，曲线变化比较平缓，此时最小置信度对产生的关联规则数目影响较小。随着最小置信度的取值趋近 1，规则的数目趋向 0。

图 5 描述了最小兴趣度与产生的关联规则数目之间的关系（最小支持度为 0.55）。从图中可知，在相同的置信度水平下，随着兴趣度的增大，所产生的规则数目不断减少。

5.2 节的实验得出的各参数设置对挖掘结果的影响效果均符合预期。结合 5.1 节的实验，可以得出本文所提出的算法是有效的。

结束语　本文借鉴线性链表的存储结构的优点，提出了基于线性链表的模糊关联规则挖掘算法。算法能够充分利用前期计算的结果有效地减少存储和计算的花销，具有较高的效率；并且通过引入兴趣度剔除那些无意义或者意义不大的规则，很好地控制了产生规则的有效性和准确性。比较分析和实验表明，算法快速而有效。

然而，由于时间原因及条件限制，本文仅在模糊关联规则挖掘算法研究上做了一些工作。下一步，将在隶属函数的确定与优化方面做深入的研究，因为隶属函数选择的好坏对挖掘结果具有非常大的影响。

参 考 文 献

[1] Chan K C C, Au W-H. Mining fuzzy association rules [C]// Proceedings of the 6th ACM International Conference on Information and Knowledge Management. Las Vegas. Nevada, USA, 1997: 209-215

[2] Hong T P, Kuo C S, Chi S C. Mining association rules from quantitative data[J]. Intelligent Data Analysis, 1999, 3: 363-376

[3] Hong T P, Kuo C S, Wang S L. A fuzzy AprioriTid mining algorithm with reduced computational time[J]. Applied Soft Computing, 2004, 5: 1-10

[4] Lee Y C, Hong T P, Lin W Y. Mining fuzzy association rules with multiple minimum supports using maximum constraints [J]. Lecture Notes in Computer Science, 2004, 3214: 1283-1290

[5] Papadimitriou S, Mavroudi S. The frequent fuzzy pattern tree [C]// Proceeding of the 9th WSEAS International Conference on Computers. 2005

[6] Lin C W, Hong T P, Lu W H. Linguistic data mining with fuzzy FP-trees[J]. Expert Systems with Applications, 2010, 37: 4560-4567

[7] http://archive.ics.uci.edu/ml/machine-learning-databases/glass/

[8] 陆建江，张亚非，宋自林. 模糊关联规则的研究与应用[M]. 北京：科学出版社，2008

（上接第 127 页）

框架 FILiC。FILiC 能够在占用较少的 GPU 计算资源的情况下，实现 CUDA 设备端的打印、文件操作等交互函数。另一方面，FILiC 提供了一系列交互模板和底层 API，以支撑用户根据自身需求开发新的交互型库函数。FILiC 框架在交互过程中能够正确有效地完成用户的代理请求，而且执行效率很高，与 CUDA 打印相比平均有 1.14 的加速比，而且能够实时地对用户的打印给出反馈。

我们认为，FILiC 库函数框架的意义主要有两点：一是对于 CUDA 应用开发人员而言，本框架不仅提供了新的交互型库函数功能，而且提供了新的实时调试手段。二是编译器开发者如果需要将用户程序转换为 CUDA 代码，那么可以利用 FILiC 框架，来实现原用户程序中的 I/O 功能、消息发送接收、socket 操作甚至一个完整的 Web 服务器。可以说，FILiC 是对 CUDA 功能拓展的一次有益探索。

参 考 文 献

[1] 张舒，褚艳丽，赵开勇，等. GPU 高性能运算之 CUDA[M]. 北京：中国水利水电出版社，2009

[2] CUDA 3.1 Downloads. developer. nvidia. com/object/cuda_3_1_downloads. html

[3] Ocelot: An Open Source Debugging and Compilation Framework for CUDA[OL]. http://code.google.com/p/gpuocelot/

[4] cuda-waste: Why Another Simple Trivial Emulator for CUDA [OL]. http://code.google.com/p/cuda-waste/

[5] NVIDIA Compute——PTX: Parallel Thread Execution ISA Version 1.1[OL]. http://www.nvidia.com/object/io_1195170102263.html

[6] OpenHMPP[OL]. http://en.wikipedia.org/wiki/HMPP_Open_Standard

[7] PyCUDA[OL]. http://mathema.tician.de/software/pycuda

[8] Breitbart J. Cupp-a framework for easy cuda integration [C]// IPDPS '09: Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing. Washington, DC, USA, IEEE Computer Society, 2009: 1-8

[9] CUDA 4.0 Downloads. developer. nvidia. com/object/cuda_4_0_downloads. html

[10] Kirk D B, Hwu W-M W. Programming Massively Parallel Processors: A Hands-on Approach[M]. ELSEVIER Press, 2010

[11] Next Generation CUDA Architecture [OL]. www.nvidia.com/object/fermi_architecture.html

[12] 尤洪涛，姜小成，陈左宁. 基于动态任务划分的降级机制[J]. 微计算机信息，2006，10(3)：72-76