基于片上多核的频繁项集并行挖掘算法

张步忠[1] 程玉胜[1] 王则林[2]
(安庆师范学院计算机与信息学院 安庆 246013)[1] (南通大学计算机科学与技术学院 南通 226000)[2]

摘 要 关联规则挖掘中最主要的工作是如何高效地挖掘频繁项集。目前在单机平台上,由于计算量大等原因,大数据集上的关联规则挖掘很难得到理想结果。在分析现有频繁项集挖掘算法的基础上,结合 Eclat 和 dEclat 挖掘算法优点,针对大数据集和片上多核共享内存计算环境,提出一种高效的并行频繁项集挖掘算法 PEclat,算法实现了任务级并行挖掘频繁项集,并在大数据集上进行了多项测试。实验结果表明,无论数据稠密程度如何,该算法均能取得较好的性能。

关键词 片上多核,频繁项集,并行处理,关联规则
中图法分类号 TP391 **文献标识码** A

Frequent Itemset Mining Parallel Algorithm Based on Chip Multi-core

ZHANG Bu-zhong[1] CHENG Yu-sheng[1] WANG Ze-lin[2]
(School of Computer and Information, Anqing Normal University, Anqing 246013, China)[1]
(School of Computer Science and Technology, Nantong University, Nantong 226000, China)[2]

Abstract The main work in association rule mining is how to find mining frequent itemsets efficiently. When the existing frequent itemsets mining algorithms are analyzed, it can be concluded that the computing scale will increase sharply with the increase of data scale. So association rule mining in large data sets on PC platform is difficult to get desired result. Combining the advantages of Eclat and dEclat algorithm, PEclat, an efficient parallel frequent itemset mining algorithm was presented. PEclat runs in the multi-core and shared-memory computing environment. It works as task-level parallel and can process large data sets. The experiment states that better performance can be achieved regardless of the data density.

Keywords Chip multi-core, Frequent itemsets, Parallel process, Association rule

1 引言

关联规则挖掘的主要工作是挖掘数据集中的频繁模式[1],并由 Agrawal 等人[2]开展了 Apriori 算法研究。频繁项集挖掘策略主要有自顶向下和自底向上两种模式。自底向上是从1项频繁集挖掘开始,再逐项递增,直到挖掘到最大项频繁集为止,该模式中,能充分利用前一项集计算信息;而自顶向下模式直接从挖掘最大频繁项集开始,计算量非常大,目前研究较少。

Apriori、FPGrowth 等算法计算时需要水平方式逐项扫描事务项,但主要缺点是每个项的支持度计算均要扫描一次数据集。而以 Eclat 算法[3]为代表的垂直数据格式,首先进行数据变换,然后在计算项集支持度时,只需计算两项集之间的事务标识交集,大大减少了数据集扫描时间。Eclat 算法的缺点在于需要额外的空间记录每个数据项的事务标识集 Tidset,特别是在数据规模较大时,Tidset 需要的空间也变得惊人。为了降低存储需求,Zaki 对 Eclat 算法进行改进,提出记录事务集差集的 dEclat 算法[4],该算法将数据格式变成"属性:事务记录差集",当数据项较密集时能有效降低存储和交集计算时间。

为了提高候选集剪枝效率,Zaki[5]提出了自底向上、基于等价类约束生成的关联规则挖掘方法,但是该方法在利用 K-1 项集生成 K 项集时不能避免剪枝操作。同样,文献[6]提出的基于概念格和等价类的约束方法,也不能避免 K 项集生成时的剪枝操作。文献[7]提出将 K-1 项集计算信息保留为 K 项集剪枝用的 ECLAT+算法,但是数据量大时,保留该类信息比较困难。

为了提高 Tidset 存储效率,以数据属性为行、事务标识 ID 为列,建立布尔矩阵[8],存在标记为1,否则记为0。交集运算转换为布尔运算,从而快速提高计算性能。文献[9]在此基础上引入倒排序,并改进了位运算方法,但在数据规模较大时算法性能下降非常严重。面对稀疏数据时,布尔矩阵存储效率很低。熊忠阳等[10]提出了散列布尔矩阵的改进算法,但该算法在遇到规模较大的数据时,散列冲突加剧,会降低算法性能。

以上研究处理的数据规模都较小。为了处理更大规模数

到稿日期:2013-05-15 返修日期:2013-07-29 本文受安徽省自然科学基金(070412061,10040606Q42),安庆师范学院青年科研基金项目(KJ201112)资助。

张步忠(1980—),男,硕士,副教授,主要研究方向为多核计算、分布式计算,E-mail:zhbzhong@aqtc.edu.cn;程玉胜(1969—),男,博士,教授,主要研究方向为数据挖掘。

据，CD[11]等算法给出了并行或分布式环境下的数据挖掘算法，但其通信开销非常大。文献[12]给出了数据并行化的Apriori并行算法。文献[13]给出了高性能集群计算环境下的关联规则挖掘算法，但其数据规模没有超过16MB。文献[14]描述了MapReduce云计算模式下的Apriori并行算法，但实验数据同样没有超过百兆，而且只分析了一个数据一个支持度下的效率。

目前，多核CPU已广泛运用，多核之间任务(task)[15]并行执行，共享内存，然而以上研究均未能发挥多核并行共享内存的优势。因此，本文在分析已有研究的基础上，综合Eclat、dEclat算法的优势，提出一个共享内存多核环境下关联规则挖掘的并行算法PEclat。该算法用垂直数据格式处理数据，对数据集进行预处理获得1项集后，根据数据稠密程度决定存储事务记录集Tidset还是事务记录差集Diffset，从2项集起并行产生K项集，算法采用先计算支持度后剪枝的优化存储策略，保证一次产生频繁项集。数据存储和记录集存储时，采用不规则矩阵存储数据，有效利用内存。该算法采用任务级并行策略，充分发挥多核CPU的优势，数据存储共享内存，因此避免了CD等算法的通信开销；并行计算时，各自任务处理相互独立，保证有更好的并行性能。实验结果表明，对较大规模数据集，无论数据稠密程度如何，PEclat算法均能获得较好的性能。

2 关联规则相关概念

$I=\{I_1,I_2,\cdots,I_m\}$是属性项集，数据集$T=\{T_1,T_2,\cdots,T_n\}$是事务集合，$T\subseteq I$。其中T_i是T中的一条事务，由若干属性组成，有唯一的标识符ID。事务标识集是全集合TID。

设A和B是项集，关联规则是形如$A\Rightarrow B$的蕴涵式[1]，主要性能指标是支持度和置信度。该项集若满足预定义的最小支持度，则称为频繁项集。频繁项X的事务标识差集$Diffset$，指不含X的事务标识集合。显然，$Diffset=TID-Tidset$。

定义1(稀疏因子) 事务集中含K项集的某项L_i的事务标识集合$L_{i_tid}=\{T_1,T_2,\cdots,T_n\}$。$|L_{i_tid}|$是$L_i$在事务集中出现的次数，$|T|$为总的事务数目。$K$项集的事务记录存储稀疏因子如下：

$$\alpha=\frac{1}{n}\sum_{i=1}^{n}\frac{|L_{i\_tid}|}{|T|} \tag{1}$$

同等情况下，稀疏因子越小，需要的存储空间越少。

性质1 向下封闭性。频繁项集的非空子集一定是频繁的[1]。反之，非频繁项集的超集一定是非频繁的。

该性质在候选项集剪枝时非常有用，能提升算法执行效率。

性质2 先产生K项集候选项集再剪枝，与一边生成频繁项集一边剪枝是等价的。

传统的关联规则算法中，K项集计算时先产生候选项集C_k，然后将非频繁项从中剪去。但可以在项L_{k1}产生后立即计算其支持度，若不满足，直接舍弃，计算完毕保留的即是K项集。

该性质意义在于不需要存储C_{K1}及其附属数据，在数据集较大时，能降低存储需求。

性质3 K项集的支持度不会超过其子集的支持度。

设K项集中某个项L由$K-1$项集L_1和L_2连接而成。L_1、L_2的事务标识记录集分别为L_{1_tid}、L_{2_tid}，根据Eclat算法，L的事务集$L_{tid}=L_{1_tid}\bigcap L_{2_tid}$，$|L_{k_tid}|\leqslant\min\{|L_{1_tid}|,|L_{2_tid}|\}$。

该性质表明，在整个计算周期中，可以开辟一个不小于1项集最大支持度的空间，作为计算支持度的缓冲区。

性质4 K项集挖掘时，只需用到K-1项集的$Tidset$集，与前K-2项集事务标识集无关。

数据规模较大时，$Tidset$集合需要非常多的存储空间，该性质能有效减少内存空间使用。

性质5 基于差集的K项集支持度[4]计算。给定$P\subseteq I$、现有数据项X和Y，P是PX、PY和PXY的公共前缀。令$t(X)$表示X的事务集，$|t(X)|$则是X的支持度，$d(X)$表示X的差集。则有：

$$t(PX)=t(P)\bigcap t(X)$$
$$t(PY)=t(P)\bigcap t(Y)$$
$$t(PXY)=t(PX)\bigcap t(PY)$$
$$d(PX)=t(P)-t(X)$$

项集PX的支持度$|t(PX)|=|t(P)|-|d(PX)|$，项集PXY的支持度$|t(PXY)|=|t(PX)|-|d(PXY)|$。项集$PXY$的事务标识差集：

$$\begin{aligned}d(PXY)&=t(PX)-t(PY)\\&=t(PX)-t(PY)+t(P)-t(P)\\&=(t(P)-t(PY))-(t(P)-t(PY))\\&=d(PY)-d(PX)\end{aligned}$$

因此，有：

$$d(PXY)=d(PY)-d(PX) \tag{2}$$
$$|t(PXY)|=|t(PX)|-|d(PXY)| \tag{3}$$

该性质简化了频繁项集计算，通过差集计算可以获得支持度。

性质6 K项集的事务集稀疏因子较大，其差集稀疏因子必然较小，反之亦然。

K项集中项L_i的事务集合为L_{i_tid}，差集$L_{i_diff}=Tid-L_{i_tid}$，设差集稀疏因子为β，则

$$\alpha+\beta=1 \tag{4}$$

稀疏因子小，需要的存储空间亦小，因此在存储记录集时，可以根据稀疏因子决定存储记录集还是差集。

例如，图1是数据水平格式的数据集，图2是以数据属性项为索引的数据垂直格式，括号内为支持度，最小支持度为2。图3－图5分别是采用差集计算的1项集、2项集和3项集，其中3项集为空。

Tid	Items			
1	I1	I2	I3	
2	I1	I3	I5	
3	I1	I5	I6	I7
4	I2	I3	I5	I7
5	I4	I5		

图1 数据水平格式

Items	I1(3)	I2(2)	I3(4)	I4(1)	I5(4)	I6(1)	I7(2)
Tidset	1	1	1	5	2	3	3
	2	4	2		3		4
	3		4		4		
			5		5		

图2 数据垂直格式

Items	I1(3)	I2(2)	I3(4)	I5(4)	I7(2)
	4	2	3	1	1
Diffset	5	3			2
		5			5

图 3　事务标识差集表示的 1 项集

Items	I1I3(2)	I1I5(2)	I2I3(2)	I3I5(3)	I5I7(2)
Diffset	3	1	Φ	Φ	2
					5

图 4　事务标识差集表示的 2 项集

该例中，稀疏因子为 0.6。用差集方法能有效降低存储需求。

3　片上多核的并行化关联规则挖掘算法

3.1　片上多核的并行化编程

并行计算可以将被求解的问题分解为若干部分，每个部分分别由不同的处理器同时进行计算，从而提高计算效率。通常做法是把任务分解，由多个处理器执行，或者数据分解到多个处理器处理。并行程序设计方法有：编译器将程序代码并行化、数据并行、共享变量、消息传递等。

片上多核处理器将多个内核集成在一个处理器芯片中，从而提高计算能力。多个核心可以并行工作，共享内存。OpenMp、MPI 等并行包也可以在多核系统中使用，但并不能充分发挥多核性能。

为了使程序有更好的并行性能，Intel TBB[15] 中调度的最小单位是任务(Task)，比线程更轻。TBB 采用 fork-join 模型，即主程序中，将任务分解到多个任务中(fork)执行，最终由主程序回收结果(join)。类似 TBB，Java SE7[16] 也集成了多核并行计算功能。考虑算法实现程序的可移植性，采用 Java SE7 作为算法实现环境。

3.2　PEclat 算法

算法描述如下：

```
输入：数据集文件 file，最小支持度 min_sup
输出：满足 min_sup 的关联规则，即频繁项集
PEclat(file,min_sup){
  initDataSet(T);//将数据集读入内存
  genEclatCk1(Ck1);//根据 Eclat 产生 1 项集
  compute(α);//计算稀疏因子
  if(α>0.1) genDiffset(DiffSet);
  for(K=2;LK≠Φ;K++){
    parallelGenLk(Lki);//并行产生 K 项集
    Lk=shuffle(Lk1,…Lkn);
  }
}
```

(1)1 项集产生。计算 1 项集时，逐条扫描数据集，将“标识：属性”转换成“属性：标识”，导致内存写操作频繁。并行化后，需要对共享区域频繁互斥写，这会大大降低并行任务的执行性能，因此 1 项集处理使用串行方式。

(2)稀疏因子。面向大数据集计算时，Tidset(Diffset)存储空间需求大。事务集稀疏因子大时，采用差集则能减少存储；同样，当差集稀疏因子大时，采用事务集则较好。

(3)并行化产生 K 项集。K 项集通过 K-1 项集自身连接生成，算法如下：

```
for(i=1;i<|Lk-1|;i++)
  for(j=i+1;j<=|Lk-1|;j++){
    i 项与 j 项连接，并计算支持度；　满足条件保留 Tidset(Diffset)，加入 K 项集　}
```

这里每个 K 项集候选项的计算均是独立的，分割循环并行执行，能取得较好的并行性能。

(4)由于 K 项集计算需要以 K-1 项集为基础，K 项集与 K-1 项集计算很难并行化处理。这里没有并行处理。

(5)结果汇总。并行任务之间数据是相互独立的，因此 shuffle(L_{k1},…,L_{kn})直接对局部结果做集合合并，生成全局 K 项集。

3.3　算法程序实现

考虑到 JDK 即时优化非常出色，不同语言实现的算法受语言环境影响比较大。因此，为了使算法在同一基准下比较，在实现 PEclat 算法时，重写了 Apriori 和 Eclat 算法，并做了数据读入和内存写操作优化。

这里所有算法均将数据集一次性全部读到内存中，存储在二维锯齿型数组中。将数据矩阵转换成属性矩阵时，常规做法是逐行扫描属性，再对 1 项集排序，做法描述如下：

```
ArrayList<EclatItem> tmpL1Set ;
……
for(int row=1;row<=transCount;row++){//逐行
  for(int col=0;col<trans[row].length;col++){//逐个
    dataItem=trans[row][col];
    al[dataItem].add(row);
```

逐行扫描时，只需将事务标识 row 写到属性 dataItem 对应的 al[dataItem]链表中，自动升序存在，无需再排序。计算支持度可直接统计 al[dataItem]中元素个数。该程序避免了 $O(n^2)$的排序操作。

程序并行化部分主要代码如下：

```
class ParallelTask extends RecursiveAction{
  //内部类
  protected void compute( ) {//并行任务入口
  计算 K 项集结果存储到 ArrayList<EclatItem>中
  }
}
parallelGenLk( ){
  ForkJoinPool fjpool=new ForkJoinPool();
  ParallelTask fjt1=new ParallelTask(new ArrayList<EclatItem>(),
  K,k_1Length,start,end);
  fjpool.submit(fjt1);//fork 操作，提交任务
  fjt1.join();//join，等待执行完毕
  读取 ArrayList<EclatItem>数据即 K 项集集合
  内存清理
  ……
  fjpool.shutdown();//计算完毕关闭线程池
}
```

由于记录事务集(差集)需要的内存较多，这里对 Eclat、PEclat 算法均按照性质 4 做优化。

4　实验与分析

4.1　实验环境

实验环境为 Intel B960 双核 2.2GHz，4GB 内存、Windows 7 HB 操作系统，Java SE7。所对比的算法均用 Java 实现。实验数据集来自 http://fimi.ua.ac.be，如表 1 所列。

表 1　实验用数据集

数据集	文件大小	事务数	总记录数	平均事务长度	最大属性值	稀疏因子@支持度
accidents	33.8MB	340183	11500870	33.8	468	0.75@0.5
chess	334KB	3196	118252	37	75	0.8@0.6
pumsb_star	10.7MB	49046	2475947	50.5	7116	0.52@0.35
retail	3.97MB	88162	908576	10.3	16469	0.0035@0.001
T10I4D100K	3.8MB	100000	1010228	10	999	0.02@0.01
T40I10D100K	14.7MB	100000	3960507	39.6	999	0.087@0.05

4.2　实验结果

实验中，由于不同的数据集稠密程度不同，因此设定的支持度也不同。实验测试了 Apriori、Eclat 和 PEclat 算法的运行时间，表 2—表 7 给出实验结果，对运行时间超过 1500s 的做超时处理。

表 2　accidents 结果(s)

支持度	Apriori	Eclat	PEclat
0.5	800.9	溢出	16.3
0.55	386.3	溢出	8.9
0.6	202.8	溢出	6.5
0.65	102.8	13.6	5.6
0.7	51.8	8.6	4.7
0.8	14.9	4.7	4.3

表 3　chess 结果(s)

支持度	Apriori	Eclat	PEclat
0.6	278.1	361.7	310.2
0.65	85.8	54.1	49.4
0.7	30.4	7.1	7.9
0.75	11.8	1.5	1.4
0.8	4.5	0.4	0.3
0.85	1.5	0.1	0.1

表 4　pumsb_star 结果(s)

支持度	Apriori	Eclat	PEclat
0.35	1377.9	溢出	45.6
0.4	340.1	12.3	3.5
0.45	43.1	3.2	1.1
0.5	14.9	1.4	1.1
0.55	7.8	1.0	0.9

表 5　T10I4D100K 结果(s)

支持度	Apriori	Eclat	PEclat
0.35	1377.9	溢出	45.6
0.4	340.1	12.3	3.5
0.45	43.1	3.2	1.1
0.5	14.9	1.4	1.1
0.55	7.8	1.0	0.9

表 6　retail 结果(s)

支持度	Apriori	Eclat	PEclat
0.001	超时	14.2	8.8
0.002	超时	4.7	3.0
0.003	1152.86	2.1	1.6
0.004	425.346	1.3	0.9
0.005	175.784	0.9	0.8
0.006	113.006	0.7	0.6

表 7　T40I10D100K 结果(s)

支持度	Apriori	Eclat	PEclat
0.05	641.4	8.7	5.4
0.06	398.8	5.8	3.8
0.07	240.9	4.4	3.3
0.08	140.5	3.5	2.4
0.09	89.1	2.5	2.4
0.1	52.2	2.2	1.7

鉴于部分结果数量级相差太大，很难用图直观展示，这里全部用表形式给出。实验结果表明：

(1)Apriori 算法内存需求稳定，但是执行时间长。Eclat 算法能获得较好性能，但在数据规模变大、低支持度时，内存消耗大，部分测试导致内存溢出。PEclat 算法降低了内存需求，并提升了执行性能。

(2)在数据规模较大时，算法性能提升效果明显。如测试用的 accidents 数据集文件大小为 33.8MB、支持度为 0.5～0.6 时，Eclat 算法内存溢出，PEclat 亦能获得很好的性能。

(3)并行化 PEclat 算法能明显改善关联规则挖掘性能。当数据规模较小(如 chess 数据集)时，算法性能提升不明显。PEclat 算法数据读入和 1 项集产生均是串行执行，并行任务也需要时间开销。

(4)频繁内存操作对性能影响较大。如 chess 集上支持度为 0.6 时，Eclat 和 PEclat 性能均较差。在小数据集上频繁内存申请和写操作影响了算法性能。数据集规模较大时，内存操作影响比重下降。

(5)在不同的数据稠密程度下，PEclat 算法均能获得较好的性能。PEclat 算法综合了 Eclat 事务集和 dEclat 差集的优势，避免了各自的缺点。

(6)同一数据集上，支持度较高时，算法性能较差。随着支持度变大，满足条件的频繁项集变少，而并行计算的开销变得明显，影响算法性能，如 pumsb_star 数据集中支持度为 0.55时，PEclat 性能不如 Eclat。

(7)本文没有给出并行化常用的性能分析指标加速比。PEclat 算法在对 Eclat 算法并行化的同时，还针对数据集稠密程度，优化了事务集存储，因此无法用加速比分析算法性能。

结束语　关联规则的主要问题在于频繁项集的发现，文章在研究 Apriori、Eclat、dEclat 算法的基础上，综合多种并行计算实现方法，以多核计算环境为平台，实现了改进的多核并行化关联规则算法，充分发挥了多核平台的优势，使用了多个存储优化方法。该算法对其他数据挖掘算法的并行化研究有较好的借鉴意义。限于普通计算机的系统环境限制，本文未能对 100MB 以上大规模数据进行处理和验证，课题的下一步工作是对大规模数据集上的挖掘进行并行化研究，并改进 PEclat 算法中尚存的串行部分。

参 考 文 献

[1] Han J, Kamber M. Data mining: Concepts and techniques [M]. Morgan Kaufmann, 2005

[2] Agrawal R, Srikant R. Fast Algorithms for Mining Association Rules[C]// Proc. of the 20th Int. Conf. on Very Large Databases. 1994: 487-499

[3] Zaki M J, Parthasarathy S, et al. New algorithms for fast discovery of association rules[C]// Proc. of the 3rd Int. Conf. on Knowledge Discovery and Data Mining. 1997: 283-286

[4] Zaki M J, Gouda K. Fast Vertical Mining Using Diffsets[C]// Proc. ACM SIGKDD Knowledge Discovery and Data Mining. 2003: 326-335

[5] Zaki M J. Scalable Algorithms for Association Mining[J]. IEEE Transactions on Knowledge and Data Engineering, 2000, 12: 372-390

[6] 杜剑峰，李宏，陈松乔，等. 单调和反单调约束条件下关联规则的挖掘算法分析[J]. 计算机科学，2005，32(6)：142-144，166

[7] 宋长新，马克. 改进的 Eclat 数据挖掘算法的研究[J]. 微计算机信息，2008，24：92-94

[8] Wur S, Leu Y. An Effective Boolean Algorithm for Mining Association Rules in Large Databases[C]// Proc. of the 6th Int. Conf. on Database Systems for Advanced Applications. 1999: 179-186

[9] 傅向华，陈冬剑，王志强. 基于倒排索引位运算的深度优先频繁项集挖掘[J]. 小型微型计算机系统，2012(8)：1747-1751

[10] 熊忠阳，陈培恩，张玉芳. 基于散列布尔矩阵的关联规则 Eclat 改进算法[J]. 计算机应用研究，2010(4)：1323-1325

[11] Agrawal R, Shafer J C. Parallel Mining of Association Rules[J]. IEEE Transactions on Knowledge and Data Engineering, 1996, 8(6): 962-969

[12] 张诤，王惠文. 一种高效的并行频繁集挖掘算法[J]. 计算机工程，2008(11)：55-57，60

[13] Zhang Y, Zhang F, Bakos J. Frequent Itemset Mining on Large-Scale Shared Memory Machines[C]// IEEE International Conference on Cluster Computing. 2011: 585-589

[14] 黄立勤，柳燕煌. 基于 MapReduce 并行的 Apriori 算法改进研究[J]. 福州大学学报：自然科学版，2011(5)：680-685

[15] Threading Building Blocks SDK [EB/OL]. http://www.threadingbuildingblocks.org

[16] Java SDK[EB/OL]. http://www.oracle.com