Journal of Computer Applications
计算机应用,2014,34(8):2175 -2178
ISSN 1001-9081
CODEN JYIIDU
2014-08-10
http://www.joca.cn

文章编号:1001-9081(2014)08-2175-04
doi:10.11772/j.issn.1001-9081.2014.08.2175

并行挖掘频繁项目集新算法——MREclat

章志刚,吉根林*,唐梦梦
(南京师范大学 计算机科学与技术学院,南京 210023)
(*通信作者电子邮箱 glji@njnu.edu.cn)

摘　要:针对 Eclat 算法在挖掘海量数据中的频繁项目集时存在的内存和计算资源不足等问题,提出了基于 Map/Reduce 计算模型的并行挖掘算法——MREclat。首先,将水平型数据库转换成垂直型数据库;然后,将转换后的数据按2-项集的前缀分发到各个计算节点上,且在分发数据时引入了均衡策略;接着,在各个计算节点上求出以某一前缀开头的所有频繁项目集;最后,合并各个节点的结果得到所有频繁项目集。介绍了 MREclat 的设计思想,研究了算法的运行性能。实验结果表明,MREclat 算法效率大约是 PEclat 算法的2倍,加速比性能比 PEclat 算法提高了64%。

关键词:频繁项目集;并行挖掘算法;列存储;Map/Reduce;Eclat 算法
中图分类号:TP311.13　　**文献标志码**:A

MREclat: new algorithm for parallel mining frequent itemsets

ZHANG Zhigang, JI Genlin*, TANG Mengmeng
(*School of Computer Science and Technology, Nanjing Normal University, Nanjing Jiangsu 210023, China*)

Abstract: Aiming at the problem that the memory and computational capability is insufficient while using Eclat algorithm to mine frequent itemsets from massive dataset, a parallel mining algorithm based on Map/Reduce framework, called MREclat (MapReduce Eclat), was proposed. Firstly, MREclat algorithm converted the horizontal database into a vertical one. Secondly, it redistributed the converted dataset according to the first item of each frequent 2-itemset and load-balance was taken into consideration while distributing datasets. Then, all the frequent itemsets prefixed by the same item were computed in each computing node. Finally, MREclat algorithm collected the result of each computing node and generated the whole frequent itemsets. In this paper, the idea of MREclat was introduced and the performance of the algorithm was studied. The experimental results show that MREclat algorithm is twice as efficient as PEclat algorithm, and the speedup performance of MREclat algorithm is 64% higher than that of PEclat.

Key words: frequent itemset; parallel mining algorithm; column-store; Map/Reduce; Eclat algorithm

0　引言

频繁项目集挖掘是数据挖掘领域的重要研究内容,它同时广泛应用于如购物篮分析、网络入侵检测和疾病诊断等领域。频繁项目集挖掘算法根据其所挖掘的数据源呈现形式分为两类:1)基于水平型(即事务型)数据库的算法,所谓水平型是指每条记录在磁盘中按行存储。这类算法中最典型的就是 Apriori 算法[1]和 FP-Growth(Frequent Pattern Growth)算法[2],这类算法常需要多次扫描数据库。2)基于垂直型数据的算法,垂直型的本质是记录在磁盘上以列存储方式存储。Eclat 算法[3]是这类算法中的代表,它只需扫描数据库一次。在列存储的数据库中,数据库管理系统(Database Management System, DBMS)只需读取那些跟查询相关的列,避免了读取不必要的列。而在行存储的数据库中,一次查询往往需要遍历大量的数据记录,因此垂直型数据库在处理数据库"读取"操作上具有较好的优势。实验结果[3-5]显示 Eclat 算法运行效率高于第一类的算法。

近年来,随着信息社会的飞速发展,人们拥有和积累的数据越来越多。在某些领域如天文、地理、气象、智能商业以及互联网等领域,它们所积累的数据已经达到 PB 级,这预示着"大数据"[6]时代的来临,如何从"大数据"中有效地挖掘出有价值的信息,成为目前的研究热点。传统的串行算法已不能满足人们处理数据的需求,目前以"云计算"为代表的并行和分布式计算为从大数据中挖掘信息提供了可能。

Eclat 算法具有良好的执行效率,因此将其并行化具有较高的可行性和研究价值。目前,频繁项目集的并行策略主要有 CD(Count Distribution)、CaD(Candidate Distribution)和 DD(Data Distribution)[7]等,但这些策略在通信和同步方面需要大量的额外开销。Zaki 等[8]提出了多种基于 MC(Memory Channel)的 Eclat 并行算法,由于这些方法均需将数据加载进入内存,导致其在对大数据进行频繁项目集挖掘时缺乏可行性。基于 Map/Reduce 计算模型的 Eclat 并行化研究文献目前较少,文献[9]提出了基于 Map/Reduce 的 Eclat 并行算法——PEclat(Parallel Eclat)算法。该算法需要多次迭代运行 Map/Reduce 任务,导致性能较低,同时算法没有考虑负载均衡等问题。本文提出了一种基于前缀分组的并行挖掘算法——MREclat(MapReduce Eclat),它将项目按前缀进行分组,在分组的过程中考虑了负载的均衡性。实验结果表明,MREclat 算法有着较好的可扩展性和加速比,且比文献[9]的

收稿日期:2014-04-10;**修回日期**:2014-05-10。　**基金项目**:江苏省自然科学基金资助重点项目(BK2011005)。
作者简介:章志刚(1988 -),男,江苏海安人,硕士研究生,主要研究方向:云计算、数据挖掘;　吉根林(1964 -),男,江苏海安人,教授,博士生导师,博士,CCF 高级会员,主要研究方向:数据挖掘;　唐梦梦(1990 -),女,江苏连云港人,硕士研究生,主要研究方向:空间轨迹挖掘。

PEclat 算法效率更高。

1 背景知识

1.1 频繁项目集

Agrawal 等[1]首次提出频繁项目集挖掘相关概念，它的形式化定义如下：设 $I = \{i_1, i_2, \cdots, i_n\}$，$I$ 是数据库中全体项目的集合，频繁项目集挖掘的数据库记作 D，$D = \{t_1, t_2, \cdots, t_k, \cdots, t_m\}$，其中 t_k 称为事务，每个事务包括一个唯一的标识符和一个项目集合，该项目集合是 I 的子集。

定义 1 设 $I_1 \subseteq I$，项目集 I_1 的支持度是数据集 D 中包含项目集 I_1 的事务的比例，即 $support(I_1) = \| \{t \in D \mid I_1 \subseteq t\} \| / \| D \|$。

定义 2 若项目集 I_1 的支持度大于用户设定的最小支持度阈值 min_sup，则项目集 I_1 为频繁项目集。

1.2 行存储和列存储

目前，数据存储有两种方案可供选择：行存储和列存储。在行存储的架构中，磁盘将一条记录的所有字段的数据聚合存储，行存储所对应的数据库称为水平型数据库[3]。行存储架构的数据库在执行“写”操作时具有较高的性能，在类似于 OLTP(On-Line Transaction Processing)的应用中具有较好的应用效果；与此相对的，列存储将同一字段的数据聚合存储。与行存储相比，列存储有以下两个优点：1)每个字段的数据聚集存储，在查询只需要少数几个字段的时候，能大幅度减少读取的数据量；2)可以为列存储设计出压缩/解压算法，来减少数据的存储量[10]。在频繁项目集挖掘概念中，将行存储所实现的数据库称为水平型数据库，水平型数据库的每个元组(又称事务)由事务唯一标识符(TID)和项目(item)组成，一条事务可以包含一个或多个项目。列存储所对应的数据库称为垂直型数据库，垂直数据中每个记录由项目集和所有包含该项目集的事务标识组成。传统的频繁项目集挖掘算法如 Apriori 和 FP-Growth 算法针对水平型数据表示，它们往往需要多次扫描数据库以完成对项目集频繁的统计。Eclat 是一种典型的基于垂直数据的频繁项目集挖掘算法，它只需扫描数据库一次就能完成频繁项目集求解。

1.3 Eclat 算法

Eclat 算法有两种求解频繁项目集策略，它们分别是自底向上搜索和混合遍历搜索。由于混合遍历搜索只能挖掘最大频繁项目子集，因此，本文采用自底向上搜索策略来发现所有的频繁项目集。

```
输入:F_k = {I_1, I_2, ..., I_n}。                    //k-项频繁项目集
输出:l-项频繁项目集,l > k。
Bottom-Up(F_k) {
1)    for all I_i ∈ F_k
2)      F_{k+1} = ∅;
3)      for all I_j ∈ F_k, i < j
4)        N = I_i ∩ I_j;
5)        if N.sup ≥ min_sup then
6)          F_{k+1} = F_{k+1} ∪ N;
7)        end
8)      end
9)    end
10)   if F_{k+1} ≠ ∅ then
11)     Bottom-Up(F_{k+1});
12)   end
13)   }
```

自底向上的搜索过程和 Apriori 算法非常类似，但是在第 5)行中，Apriori 算法需要扫描整个数据库来计算项目集 N 的支持数，而 Eclat 算法只需要统计 N 的 *tid-list* 的元素个数。本文在实现 Eclat 时使用了两种改进策略：1)在求交集时引用了 Apriori 算法中的剪枝思想，以减少交集运算；2)引用了文献[11]中“差集”思想，以减少每个项目集的 *tid_list* 长度，并有效减少了求解交集的运算时间。

1.4 Map/Reduce 框架

Map/Reduce[12]是 Google 提出的一种用于处理大规模数据集的并行编程模型，它是无共享软件架构的典型实现。它采用“Divide & Conquer”思想，把对大规模数据的操作分发到一个主节点管理下的各分节点共同完成，然后通过整合各分节点的中间结果，得到最终结果。它把所有的运算过程，高度抽象成 map 和 reduce 这两个函数，所以用户不必考虑如工作调度、容错处理、网络通信、负载平衡等细节。

Map/Reduce 程序在执行过程时首先把数据集分成若干固定大小的独立数据块(block)，同时为每个数据块保存多个副本。Map/Reduce 在每个数据块上运行一个独立的 Mapper 任务。由于每个数据块的大小固定，因此运行的 Mapper 数由数据源大小决定。Mapper 将输入的每个元组通过 map 函数转换成〈*key*,*value*〉形式的键值对并输出，所有的键值对根据其 *key* 的值被 Hash 分送到不同的 Reduce 运算节点上。由于所有的 Mapper 采用相同的 Hash 函数，*key* 值相同的键值对会被分送到同一个 Reduce 节点，同时，同一个 Reduce 节点可能会收到多个不同 *key* 值的键值对(它们 Hash 后值相同)。在每个 Reduce 计算节点上运行一个 Reducer 任务用于处理 Mapper 发送过来的数据，Reducer 的 reduce 函数每次将一个 *key* 值对应的所有键值对转换成新的键值对，Reducer 的输出键值对集合即为本文所需结果。

2 MREclat 算法设计与实现

MREclat 算法主要分为两步：1) 初始化步骤。该步骤用于将行存储的水平型数据库转换成列存储的垂直型数据库。2) 并行挖掘步骤。首先，将第 1) 步获得的数据按项目集的前缀(项目集排序后的第一个 *item*) 分发到不同的计算节点上，为了获得较好的负载平衡，本文将所有的前缀(1-项频繁集)进行均衡分组；然后，使用改进的 Eclat 算法在各计算节点上求解其所对应的分组中 *item* 开头的所有频繁项目集。

2.1 初始化

由于所获得商务数据源往往采用水平数据库，因此需要先将水平数据库转换成垂直数据库。目前已有的方法直接将水平数据库变成以每个频繁 1-项集为列的垂直数据库，这样做主要有 3 个缺点：1)本身转换需要时间；2)转换后的 1-项集所对应的列长度较大；3)由 1-项集求解 2-项集时需要较多的交运算。

假设一个数据集有 1 000 000 条事务数据，共 1 000 个项目，并且平均每条事务的长度为 10。那么可计算出该数据集的每个 *item* 所对应的 *tid_list* 的平均长度为 10 000。求解 2-项频繁项目集的过程中需要进行 $C_{1000}^2 \times (2 \times 10000) \approx 10^9$ 次运算。本文采用的方法是从水平数据库中直接求解 2-项集。对水平数据的每条记录产生所用可能的 2-项集，然后以 2-项

集为列构造垂直数据库。这一过程只需要 $C_{10}^2 \times 1\,000\,000 = 4.5 \times 10^7$ 操作,比已有方法效率更高。同时,为了进一步减少计算操作,在构造数据库之前,MREclat 先扫描数据库,得出所有的 1- 项频繁集(只需较少的时间);然后再对水平数据库的每条记录扫描其出现在 1- 项频繁集中的 *item* 两两合并构成一个 2- 项集。求解 1- 项频繁集的过程的本质就是词频统计,这是 Map/Reduce 框架的一个经典应用因此不作介绍。水平数据库转换成垂直数据库过程的伪代码描述如下。

```
输入:key 是事务的标识符,value 是事务的项目集。
map(key,value) {
  从value 中获得所有的项目,删除不在 1- 项频繁集中出现的项
    目;对剩下的项目按字典序进行排序;items 数组用来存放排
    序后的结果,项目集的长度为 n
  for (i = 0;i < n - 1;i ++)
    for (j = i + 1; j < n; j ++)
      output( items[i] ∩ items[j],key);
                          // 生成 2- 项集,输出 2- 项集和 tid
    end
  end
}
```

相应地,reduce 函数的输入是 2- 项集和包含该 2- 项集的 *tid_list*。通过 reduce 函数可以获得所有的 2- 项频繁项目集和它的 *tid-lists*,并将结果存储在以垂直数据布局的数据库中。

```
reduce(key,value) {
  tid_list = ∅;
  sum = 0;
  for each tid in value
    sum ++;
    tid_list = tid_list ∪ tid;
  end
  if sum/transnum ≥ min_sup then
    output (key,tid_list);
    //只输出 2-项频繁项目集
  end
}
```

2.2　负载平衡

为了在从 2-项频繁集中求解剩下频繁项集时获得较好的负载平衡,MREclat 将 1-项频繁集进行了有效的分组。Eclat 算法主要的运算就是交集运算,且计算两个项目集交集运算复杂度可用这两个项目集 *tid_list* 长度的和表示。本文使用 w_i(w 表示权重)来表示获得以项目 A 为前缀的所有频繁项目集的操作复杂度(i 是 A 在 1- 项频繁项目集中的索引)。w_i 的定义如下:

$$w_i = \lg(n-1) + \lg\left(\sum_{j=0}^{j<n} len_j\right)$$

其中 n 表示以 A 为前缀的 2- 项频繁集的数量。MREclat 将以 A 为前缀的 2- 项频繁项目集组成的集合按字典序进行排序,并用 len_j 来表示第 j 个 2- 项频繁集的长度。

在计算出所有的 1-项频繁集的权重后,MREclat 需要将由 1-项频繁项目集和其权重构成的二元组集合按照权重的降序进行排序。MREclat 算法使用贪心策略来将这些二元组划分到不同组。分组策略如下所示。

```
输入:数组 a 用来存储包含项目和权重,即〈item,weight〉的元组;
      M 为 1- 项集的个数;N 为分组的数量。
输出:groups 数组,每个元素包含一组项目集。
divide(a,groups,M,N)
  if (M <= N) then
    for each tuple t in a
      add t. item to groups[i];
    end
  else
    Initial array sum[ ] of length N with 0;
    for (i = 0;i < M;i ++)
      add the item of a[i] to groups[i];
      sum[i] += a[i]. weight;
    end
    for (i = M;i < N;i ++)
      k = min_index(sum);// 从 sum 数组中找到最小值的下标,记为 k
      add the item of a[i] to groups[k];
      sum[k] += a[k]. weight;
    end
  end
```

2.3　并行 Eclat

在这一步中,首先,要将相同前缀的 2-项频繁项目集分发到相同的节点;然后,在这些各节点上使用改进后的 Eclat 算法并行挖掘频繁项目集。该步骤需要 Map/Reduce job 来完成,mapper 阶段完成数据的再划分,reducer 阶段运行 Eclat 算法。在 mapper 和 reducer 之间需要重写 partition 过程以确保属于同一分组的前缀所对应的列被划分到同一个计算节点。该过程伪代码如下所示。

```
输入:key 是 2- 项集,value 是 2- 项集的 tid_list。
map(key,value) {
  Get the prefix p from key;
  output (a,〈key,value〉);
}
```

通过 reduce 函数中,获得所有以 a 为前缀的 2-项频繁项目集,其伪代码如下:

```
输入:key 表示前缀;value 表示以 key 为前缀的 2- 项集。
reduce(key, value) {
  a = key;
  Get all C_a2 from value;          //C_a2 表示以 a 为前缀的 2- 项集
  Bottom-Up (C_a2)
  Output all the frequent itemsets with the prefix a;
}
```

最后,通过收集 reduce 函数的输出,得到所有的频繁项目集。

3　实验结果及算法性能分析

本实验所运行的云计算实验环境共有 11 台服务器(1 个 master 节点,10 个 slave 节点),每个节点配置相同。节点的处理器为 Intel Xeon CPU E5620,主频为2.40 GHz,操作系统为 64 位 Debian Linux 操作系统。Hadoop 平台版本为 Hadoop-0.20.2,数据存放在 HBase 上。在实验中使用了两组合成数据集和一个真实数据集。第 1 组数据集的容量分别为 560 MB、1 126 MB、1 740 MB 和2 356 MB,分别包含3 000 000,6 000 000,9 000 000和12 000 000条事务;第 2 组数据集的容量分别为186 MB、374 MB、560 MB,分别包含1 000 000,2 000 000 和 3 000 000 条事务。这两组数据集均是由 IBM DataGenerator 产生,每个数据集包含 1 000 个不同的项目,平均事务长度是 10;真实数据集 WebDocs 来自 FIMI[13],该数据集共包含 1 692 082条事务和5 267 656个不同的项目,最大的事务长度为 71 472。

图 1 显示了 MREclat 算法在挖掘不同大小数据下的性能所示,实验使用第 1 组合成数据集,使用了 0.01%、0.015% 和 0.02% 这 3 个最小支持度阈值(*min_sup*)。从图 1 可以看出,由于节点个数确定,随着总数据量的增加,每个节点的数据量也随之增加,算法运行总时间随之成线性增长;同时随着支持度阈值的增大,算法所需运行时间随之减少。

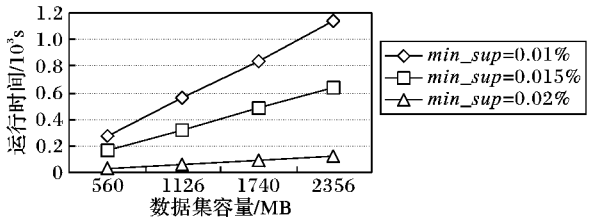

图 1 MREclat 在合成数据集(节点数为 10)的性能

图 2 展示了 MREclat 算法在第 2 组合成数据集下的运行结果。从图 2 中可以看出,随着计算节点的增加,3 个数据集的计算时间相应减少。同时也发现当计算节点增加到一定的量后,通信时间将会占用主要时间,导致总运行时间不再减少。

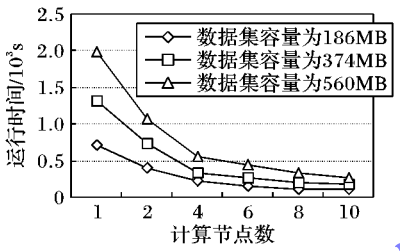

图 2 不同计算节点下的运行时间(*min_sup* = 0.01%)

图 3 ~4 为 MREclat 算法与文献[9]的 PEclat 算法在分别合成和 WebDocs 数据集上不同计算节点下的运行时间比较。从图 3 ~4 中可以看出,MREclat 算法执行效率高于 PEclat 算法。

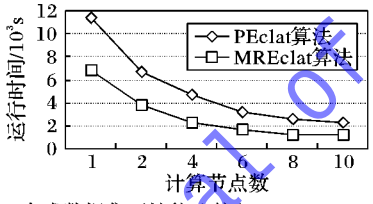

图 3 合成数据集下性能比较(*min_sup* = 0.01%)

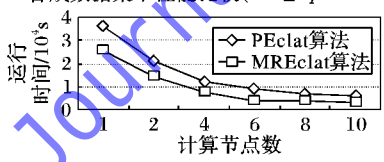

图 4 WebDocs 数据集下性能比较(*min_sup* = 0.01%)

图 5 为 MREclat 和 PEclat 算法在 WebDocs 数据集下的加速比对比。

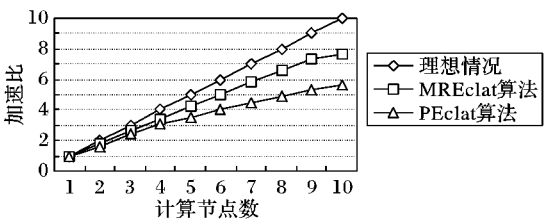

图 5 加速比比较

从图 5 中可知,MREclat 算法具有较好的加速比,与 PEclat 算法相比可扩展性更高。

4 结语

本文提出了一种基于 Map/Reduce 框架的 MREclat 并行算法。MREclat 算法先将水平型数据库,转换成适合 Map/Reduce 模型的垂直型数据库;然后将数据按前缀进行重新分组,使前缀相同的能够被分发到同一个节点上进行计算。MREclat 引入了均衡的分组方案以获得较好的负载平衡。实验结果表明,MREclat 算法有着很好的加速比和良好的可扩展性,运行效率比文献[9]的算法更高。

参考文献:

[1] AGRAWAL R, SRIKANT R. Fast algorithms for mining association rules [C]// Proceedings of the 20th International Conference on Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers, 1994: 487 – 499.

[2] HAN J W, PEI J, YIN Y. Mining frequent patterns without candidate generation [C]// Proceedings of the 2000 ACM SIGMOD International Conference on Management of data. New York: ACM Press, 2000: 1 – 12.

[3] ZAKI M J, PARTHASARATHY S, OGIHARA M, *et al*. New algorithms for fast discovery of association rules [C]// Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining. Menlo Park: AAAI Press, 1997: 283 – 286.

[4] LI Z, LIU X, CAO X. A study on improved Eclat data mining algorithm [J]. Advanced Materials Research, 2011, 328/329/330: 1896 – 1899.

[5] KOTIYAL B, KUMAR A, PANT B, *et al*. User behavior analysis in Web log through comparative study of Eclat and Apriori [C]// Proceedings of the 7th International Conference on Intelligent Systems and Control. Piscataway: IEEE Press, 2013: 421 – 426.

[6] HOWE D, COSTANZO M, FEY P, *et al*. Big data: the future of biocuration [J]. Nature, 2008, 455(7209): 47 – 50.

[7] ASHRAFI M Z, TANIAR D, SMITH K. ODAM: an optimized distributed association rule mining algorithm [J]. IEEE Distributed Systems Online, 2004, 5(3): 1 – 18.

[8] ZAKI M J, PARTHASARATHY S, OGIHARA M, *et al*. Parallel algorithms for discovery of association rules [J]. Data Mining and Knowledge Discovery, 1997, 1(4): 343 – 373.

[9] LI W, ZHAO H, ZHANG Y, *et al*. Research on massive data mining based on MapReduce [J]. Computer Engineering and Applications, 2013, 49(20): 112 – 117. (李伟卫, 赵航, 张阳, 等. 基于 MapReduce 的海量数据挖掘技术研究[J]. 计算机工程与应用, 2013, 49(20): 112 – 117.)

[10] STONEBRAKER M, ABADI D J, BATKIN A, *et al*. C-store: a column-oriented DBMS [C]// Proceedings of the 31st International Conference on Very Large Data Bases. New York: ACM Press, 2005: 553 – 564.

[11] ZAKI M J, GOUDA K. Fast vertical mining using diffsets [C]// Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York: ACM Press, 2003: 326 – 335.

[12] DEAN J, GHEMAWAT S. MapReduce: simplified data processing on large clusters [J]. Communications of the ACM, 2008, 51(1): 107 – 113.

[13] LUCCHESE C, ORLANDO S, PEREGO R, *et al*. WebDocs: a real-life huge transactional dataset [EB/OL]. [2013-12-10]. http://fimi.ua.ac.be/data/webdocs.pdf.