第 42 卷 第 6A 期
2015 年 6 月

计 算 机 科 学
Computer Science

Vol. 42 No. 6A
June 2015

并行 Fp-growth 算法在搜索引擎中的应用

黄 剑[1] 李明奇[1] 郭文强[2]

(电子科技大学数学科学学院 成都 611731)[1] (新疆财经大学计算机科学与工程学院 乌鲁木齐 830012)[2]

摘 要 针对用户历史检索过程产生的 Web 日志文件,研究其查询词和点击链接是否为频繁集,以及在分布式条件下频繁集挖掘的效率问题。基于 Hadoop 框架,设计了并行 Fp-growth 算法,对搜索引擎 Web 日志进行挖掘。仿真实验结果显示,满足支持度的查询词和点击链接频繁集在 Web 日志中普遍存在。随着 Hadoop 节点数的增加,并行 Fp-growth 算法性能将得到大幅提高。由此,频繁集挖掘效率得到明显提高,且数据量越大,效率提升越明显。

关键词 日志文件,频繁集,Hadoop,Fp-growth

中图法分类号 TP391 文献标识码 A

Parallel Fp-growth Algorithm in Search Engines

HUANG Jian[1] LI Ming-qi[1] GUO Wen-qiang[2]

(School of Mathematical Sciences, University of Electronic Science and Technology of China, Chengdu 611731, China)[1]

(School of Computer Science and Engineering, Xinjiang University of Finance & Economics, Urumqi 830012, China)[2]

Abstract Web log file is generated for the user history retrieval process. The paper studied whether the query words and click on the link belong to frequent itemsets, and frequent itemsets mining efficiency under the distributed conditions. Based on Hadoop framework, we designed a parallel Fp-growth algorithm to mine the search engine web log. Simulation results show that those query words and click on the link frequent itemsetss satisfying the given support are prevalent in web logs. With the increase of the number of nodes in the Hadoop, the performance of parallel Fp-growth algorithm will be improved greatly. Thus, the mining efficiency of frequent itemsets is significantly improved. Simulation results also show if the amount of data is greater, the improvement is more obvious.

Keywords Log file, Frequent itemset, Hadoop, Fp-growth

1 引言

搜索日志主要是记录用户查询点击过程。用户在每一次检索过程中,其搜索行为将在 cookie 中保留,其中包括了查询词、点击 URL、二跳记录、IP、访问时间戳等信息。同时,搜索行为被搜索引擎服务器解析,最终以结构化数据格式保存在公司的搜索日志中。

国外对搜索引擎研究起步较早,发现英文搜索引擎查询长度具有规律性、普遍存在查询重复性、查询串和 URL 具有相关性等[1]。国内对搜索引擎研究较晚,但在查询串长度规律、查询串长尾效应、热榜分析、点击与排名关系、URL 频次和深度分析等方面[2]都有深入的研究。利用数据挖掘技术进行搜索引擎日志研究,是一个新的方向。近几年,搜索引擎营销已经是一种成熟的盈利模式,商家已经把搜索推广作为主要营销模式之一。搜索推广也就是一个系统推荐过程,当用户输入查询词后,在结果页会出现许多广告信息,这就是推荐结果[3]。商家通过这种方式达到促销的目的。

关联规则作为数据挖掘中的一个重要部分,广泛应用于知识发现和推荐系统。频繁项集的提取直接影响关联规则挖掘算法的知识发现性能,影响频繁集信息收集。1994 年,R. Agrawal 和 R. Srikant 提出了关联规则挖掘的 Apriori 算法,2000 年 J. Han 提出了性能更加优异的 Fp-gowth 算法[4]。由于后期数据量过于庞大,这些算法遇到了内存瓶颈等许多实现问题,以 Hadoop 为代表的分布式平台出现后,许多更优的解决方案使得这种情况得到了很大的改善。

Hadoop 是一个开源分布式框架[5],由 Apache 基金会管理,在很多企业中得到广泛应用。该框架集主要由两部分组成:分布式文件框架 HDFS 和谷歌提出的 MapReduce 的编程模型。在实际应用中,Hadoop 平台具有许多优势。

(1)经济性:集群节点可采用普通 PC;

(2)可扩展性:支持 PB 级别数据处理;

(3)容错性:Hadoop 在处理文件时,采取多副本冗余拷贝机制;

(4)易用性:Hadoop 的并行处理、容错机制、本地优化和负载平衡对用户透明。

HDFS 以 Master/Slave 作为架构[6],由 1 个管理节点(NameNode)和 N 个数据节点(DataNode)组成。NameNode 提供 API 与 Client 之间的交互,在底层实现文件的 block 分

本文受国家自然科学基金(61163066)资助。

黄 剑(1988—),男,硕士,主要研究方向为数据挖掘及应用,E-mail:huangjian502871597@qq.com;李明奇(1970—),男,博士,副教授,主要研究方向为信息论及应用;郭文强(1975—),男,博士,教授,主要研究方向为计算机通信、信号处理。

割,并存储于 DataNode。存储过程引入了冗余存储机制以实现容错容灾。Rack 是机柜的意思,一个 block 的 3 个副本通常会保存到 2 个或者 2 个以上的机柜中,这样做的目的是防灾容错,因为发生一个机柜掉电或者一个机柜的交换机挂了的概率相对较高。Client 先将数据写到第 1 个节点,在第 1 个节点接收数据的同时,又将它所接收的数据推送到第 2 个,第 2 个推送到第 3 个节点,如果有多个节点,依次类推,其架构如图 1 所示。

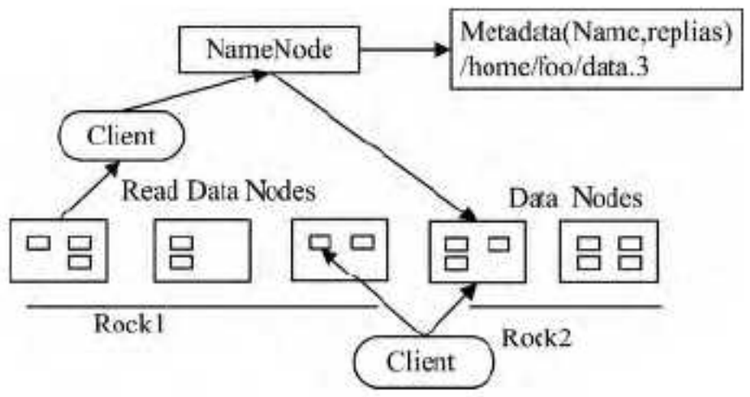

图 1 HDFS 架构

MapReduce 框架是高效的大规模数据计算模型[7],Map/Reduce 是其核心。一个文件被切分成小粒度的块文件。这些块文件分配到对应的 block,同时分配一个 map 任务,所有任务执行完成,进行 shuffle 阶段,然后根据哈希算法分配 reduce 任务,执行完成后输出最后的结果。

本文以查询词(query)和点击链接(url)为维度,设计了基于 Hadoop 的 Fp-growth 算法,挖掘出有效的 query 和 url 频繁项集。同时,给出了在不同节点数目和支持度下的性能分析。

2 基于搜索引擎的并行 Fp-growth 算法设计

下面讨论并行 Fp-growth 算法的设计与其在搜索引擎中的应用。

2.1 并行 Fp-growth 算法设计

给出了一个事务集合,由一系列具有唯一标识数据库事务的 TID 组成,$D=\{t_1,t_2,\cdots,t_m\}$,每个事务 $t_i(i=1,2,\cdots,k)$ 都对应 I 上的一个子集。

定义 1 项目集 I_k 在数据集 D 上的支持度是指包含 I_k 的事务在 D 中所占的百分比:$support(I_k)=\{\|I_k\|\}/\|D\|$。

定义 2 若项目集的支持度大于等于用户给定的最小支持度(minsup),则该项目集称为频繁项集。

定义 3 若存在路径$\langle a,b,c,d,e\rangle$,则$\langle a,b,c,d\rangle$称为以节点 e 为基的前缀路径。

Fp-Growth 算法的设计优越性来自灵巧的数据结构,与 Aprioir 算法相比,能大大降低算法的挖掘代价,减少 IO 时间。它不需要频繁地进行文件 IO 和生成候选集队列。为了实现这样的效果,采用一种叫做 frequent-pattern tree(频繁模式树)的数据结构,通过对该模式树的挖掘,能够得到我们需要的频繁集。其基本思想是对事务进行压缩存储,通过 2 次扫描数据库建立树结构。第 1 次扫描事务能找出频繁一项集合。第 2 次开始动态地建立频繁树,以"Null"为根节点,创建头表。节点由频繁项和频次两部分组成。同时,设置指针指向树中对应的节点,对频繁树进行递归挖掘,找出频繁集合。下面通过一个实例解释该过程。

表 1 给出了一份商品清单,一共有 5 条事务。原来记录为 item bought,在扫描一次表 1 对应的数据库后,会得到事务项目的频繁次数。对于不满足支持度的项目,进行删除,以生成一项频繁集。表 1 中(ordered)frequent item 这一列是按照事务的频次进行了一次降序排列后的结果。

表 1 事务排序并过滤

TID	item bought	(ordered) frequent item
100	f,a,c,d,g,i,m,p	f,c,a,m,p
200	a,b,c,f,l,m,o	f,c,a,b,m
300	b,f,h,j,o	f,b,
400	b,c,k,s,p	c,b,p
500	a,f,c,e,l,p,m,n	f,c,a,m,p

接下来,将每一条记录添加到频繁树中,有重合路径的,将节点的频次做叠加处理。根据表 1 建树完成,可以得到图 2 的频繁树。

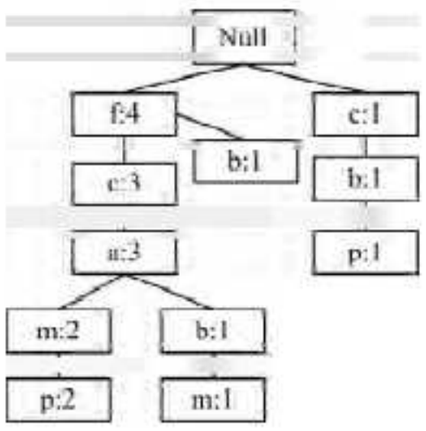

图 2 事务频繁树

如图 3 所示,当表头建立完之后,分别从节点 p 开始建立子模式频繁树,过程与建立初始频繁树同理。

图 3 表头

以图 3 中节点 p 为例,$(p:3)$就是一个简单的频繁集。根据链状关系,它的两个节点分别位于路径$\langle c:1,b:1,p:1\rangle$和路径$\langle f:4,c:3,a:3,m:2,p:2\rangle$中。可以观察到,$\langle f,c,a,m,p\rangle$出现了 2 次,$\langle f,c,a\rangle$出现了 3 次,$\langle f\rangle$出现了 4 次。

在以上所提路径中我们需要关注路径$\langle f,c,a,m,p\rangle$,只有通过这条路径我们才能找出包含 p 的所有频繁集合。前缀$\langle f,c,a,m\rangle$也叫做 p 的条件模式基(subpattern-base)。接下来为这个条件模式基构建一棵 Fp 树。建树过程和初期的建树过程是一样的,通过迭代产生下一次的条件模式基。

图 4 中,p 的子模式树就是一次迭代的结果。如果取频繁树的阈值为 3,那么这棵树经过剪枝之后只剩下一个分支$(c:3)$。因此,从这棵条件 Fp 树上只能派生出一个频繁项目集$\{cp:3\}$。

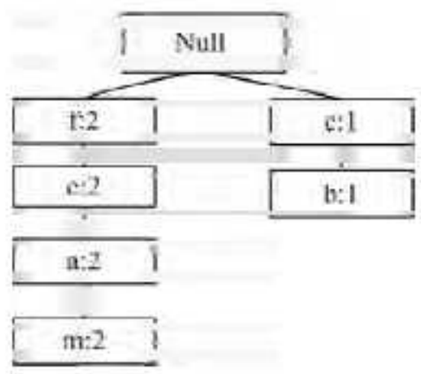

图 4 p 的子模式树

同时,从表 1 中已经可以直接得到频繁集$(p:3)$,这样$\{cp:3\}$,$(p:3)$就是以 p 为基的所有频繁集合。以此类推可

以得到基于 m,b,a,c 构建的子模式树，这样就可以找出所有的频繁集。

Fp-growth 算法可能会因为频繁树过大，导致内存溢出等问题，不利于挖掘。因此，采取“分而治之”思想，可以使用多个局部 Fp-Tree 代替整个的 Fp-tree。基于 Hadoop 的 Fp-growth 算法设计如下：

(1)统计频繁 1 项集

所有事务中的项目频次利用 MapReduce 任务来统计[8]。文件被切分成 split，分配到 block，然后调用 map 任务进行事务解析。每一条 $pair(key,value)$ 中，key 是事务的 id，$value$ 是事务 String。Map 里面需要设计函数进行字段分割，最后循环将每一个项集进行输出，格式为 $(itme,1)$。中间经过 shuffle，把 key 相同的聚集到一起。Reduce 函数统计项目集出现的次数，其中需要考虑支持度的过滤。最后，将项目集出现的次数依从高到低的顺存入到 Flist 中。

Mapper 过程：

```
Mapper(key,value=T) {T=(t1,t2,…,tm)}
for each ti in T do
output(ti,1);
end
```

Reducer 过程：

```
Reduce(ti,1)
Set C=0;
Pre=ti-1;
If(pre==ti)
    C+=1;
Else
  if C/‖D‖
      minsup then
      output(key,C);//输出 1 项频繁集及支持度
end
```

(2)节点事务分组的创建，并进行挖掘

在统计了 1 项频繁集合 Flist 以后，进行分组操作。主要是根据事务中的 Flist 支持度降序进行数据分组[9]，创建 Glist 表。将数据分发后，每个 Glist 表中的事务是完备的，其中 Glist 用 gid 标识，这是分治策略的核心所在。利用 m 个节点并行处理 n 个频繁集，就空间复杂度而言，它会降到 $O(n/m)$。按照平衡分组的策略[10]把数据分成 n 组，使得当数据输入时，一个 map 函数对应一个分组，分别对各分组进行挖掘。这样，就达到了数据分解的效果。利用 reduce 把 map 处理的结果汇总起来。每个 map 任务的运行是基于在平衡分组的过程中产生的 Glist 表。

本文建立的 Glist 表是 Hash 结构，每个项映射到该项所在的事务数。每个 reduce 任务归约 map 任务产生的频繁模式，最后得到挖掘的结果。算法的伪代码如下：

Mapper 过程：

```
Mapper(key,value=T)
Load G-Llist
Generate hash table h from Glist
for j=T-1 to 0 do
  if getHashNum(H,a[j])=true
  //判断该事务中，是否命中了 gid 中的频繁集
  output(gid;a[0]+a[1]+…+a[j]);
End
```

Reducer 过程：

```
Reducer(key=gid,value=分发的对应事务)
//这里的主要任务是解析所有事务，完成后进行 LocalFptree 的建立
foreach 事务数据 Ti in DB(gid) doCall
  insert(Ti,tree)将 Ti 保存到 tree 中;
End
Calloutput(gid;tree)
```

(3)聚集融合

这一步通过一个 MapReduce 过程融合多个第(2)步的输出，最终输出全局结果。

```
Mapper(key,value=频繁模式)
  foreach 频繁项 i in 频繁模式 do
    Calloutput(i;频繁模式);
  End
Reducer(key=i,value=包含 i 的频繁模式 Vi)
  foreach 模式 v in Vi do
    Calloutput(gid,tree)
```

2.2 基于搜索引擎日志的用户行为分析

用户每一次检索，都会形成 Web 日志，通过这些日志，可以进行用户行为分析。用户在搜索时，输入查询词，点击查询以后，在自然结果中会呈现许多网站信息，然后用户根据喜好进行点击。一个网站是由主页、二级页面，甚至三级页面组成，每一个页面对应一个特定的 url。在搜索引擎中该 url 称为着陆页。比如，在一个网站内检索了“三星手机”，很有可能接下来会检索“HTC 手机”；检索“全聚德烤鸭”还会检索“便宜坊烤鸭”。用户行为分析如图 5 所示。

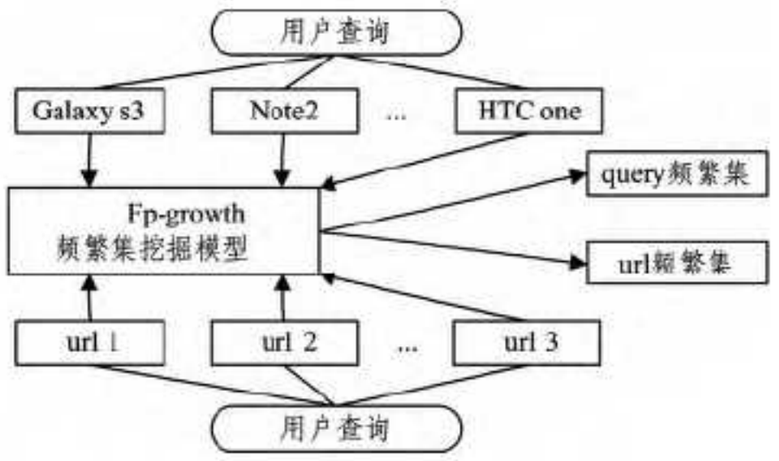

图 5 用户行为分析

3 仿真实验结果与分析

使用 4 台一般配置的计算机构建 Hadoop 分布式集群，其中 1 台作为 NameNode 和 TaskTracker，其余 3 台作为 DataNode 和 TaskTracker。硬件环境：Intel(R) Core2 Duo CPU @2.00GHz、2GB 内存、250G 硬盘、100Mbps 网口。软件环境：Ubuntu 10；JDK1.6.0-θ。

实验数据来自搜狗实验室，其字段的格式如表 2 所列。频繁一项集合分布如表 3 所列。频繁二项集合分布如表 4 所列。

表 2 数据字段说明

字段名称	说明
SesstionID	cookie 信息
Query Term	查询词
Rank	被点击 url 排名
Sequence Number	用户点击顺序号
URL	用户点击 url

(下转第 483 页)

Retrieval. 1996:4-11

[9] Xu J, Croft B. Improving the effectiveness of information retrieval with local context analysis[J]. ACM Transaction on Information Systems, 2000, 18(1):79-112

[10] Kelly D, Teevan J. Implicit feedback for inferring user preference:a bibliography[C]// ACM SIGIR Forum. 2003:18-28

[11] Shen X, Tan B, Zhai C. Context-sensitive information retrieval using implicit feedback[C]//Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval. 2005:43-50

[12] Jones K S. Automatic keyword classification for information retrieval[M]. 1971

[13] Jing Y F, Croft W B. An association thesaurus for information retrieval[C]// RIAO Conference Proceedings. 1994:146-160

[14] Yahia S B, Jaoua A. Discovering knowledge from fuzzy concept lattice[M]// Data mining and computational intelligence. Springer, 2001:167-190

[15] Fonseca B M, Golgher P B, De Moura E S, et al. Discovering search engine related queries using association rules[J]. Journal of Web Engineering, 2003, 2(4):215-227

[16] Latiri C C, Yahia S B, Chevallet J P, et al. Query expansion using fuzzy association rules between terms[J]. Proceedings of JIM. 2003

[17] Cui H, Wen J, Nie J, et al. Query expansion by mining user logs [J]. IEEE Transactions on Knowledge and Data Engineering, 2003, 15(4):829-839

(上接第 461 页)

表 3 频繁一项集

名称	事务数	url 频繁数	query 频繁数
data1	1×10^7	1782	7231
data2	2×10^7	3461	13282
data3	3×10^7	5113	20891
data4	4×10^7	6922	29182
data5	5×10^7	8211	37261

表 4 频繁二项集

名称	事务数	url 频繁数	query 频繁数
data1	1×10^7	688	3892
data2	2×10^7	1361	7912
data3	3×10^7	1929	1125
data4	4×10^7	2416	15271
data5	5×10^7	3213	17828

搜狗实验室的搜索引擎日志中,设置 $minsup=0.001$ 时,从表 3 可以看出,频繁一项集随着事务数的增加而基本上呈现正相关,同时 query 频繁集明显多于 url 频繁集合。从表 4 可以看出,频繁二项集比频繁一项集明显减少,url 减少的幅度大于 query 减少的幅度。找到了频繁一项集和频繁二项集,就可以根据关联规则得经典应用,进行推荐。

通用搜索引擎的数据较为稀疏;垂直型搜索引擎,例如电商、团购网站,其数据会更加的稠密。频繁集合挖掘的效果与数据稠密性紧密相关。

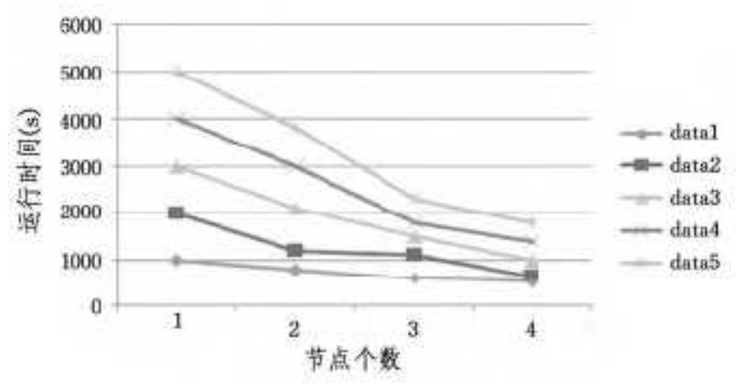

图 6 不同节点数的运行时间对比

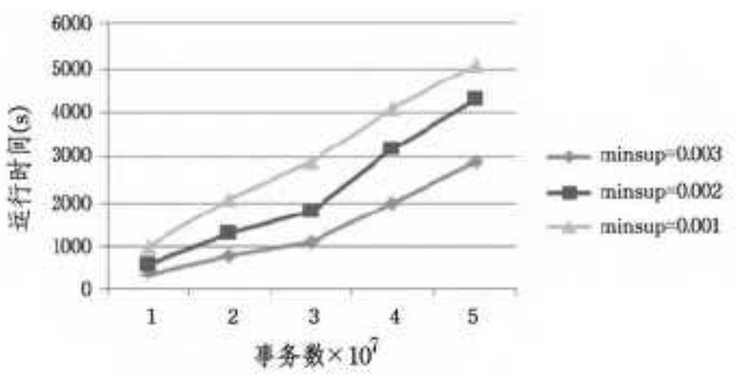

图 7 不同事务数的运行时间对比

从图 6 可以看出,随着节点数增加,算法的运行效率会得到很大提高,数据规模越大效率提升越明显,在第一个和第二个节点的提升较快,后续节点提升变缓。从图 7 可以看出,在不同的支持度下,运行时间随着事务数的增加,几乎呈线性增加。

因此,可根据数据集的大小选择合适的节点数目并设置合适的支持度阈值。

结束语 在搜索引擎中根据 Web 日志,寻找 query 和 url 的频繁集,有利于发现用户获取信息的习惯,帮助站点提供更好的个性化服务。本文设计了基于 Hadoop 的并行 Fp-growth 算法数据挖掘模型,有利于性能、架构优异性的迁移。通过对日志的处理效率和结果分析,发现 Hadoop 分布式架构能够很好地满足大规模日志处理的需求。当数据量越大时,并行 Fp-growth 算法的运算性能提升越明显。

参 考 文 献

[1] 董志安,吕学强. 基于百度搜索日志的用户行为分析[J]. 计算机应用与软件,2013,7(2):17-20

[2] 陈富赞,刘青,李敏强,等. 一种基于会话聚类算法的 Web 使用挖掘方法[J]. 系统工程学报,2012,1(7):129-136

[3] 刘建国,周涛,汪秉宏. 个性化推荐系统的研究进展[J]. 自然科学进展,2009,1(10):1-15

[4] 蓝祺花,吴博. 频繁项集挖掘算法研究[J]. 计算机与现代化,2009,3(9):60-65

[5] 吕婉琪,钟诚,唐印浒,等. Hadoop 分布式架构下大数据集的并行挖掘[J]. 计算机技术与发展,2014,24(1):22-25,30

[6] 周诗慧,殷建. Hadoop 平台下的并行 Web 日志挖掘算法[J]. 计算机工程,2013,6(3):43-46

[7] 张俊,李鲁群,周熔. 基于 Lucene 的搜索引擎的研究与应用[J]. 计算机技术与发展,2013,23(6):230-232

[8] Naganathan E R, Narayanan S, Kumar K R. FP-Growth Based New Normalization Technique for Subgraph Ranking[J]. International Journal of Database Management Systems, 2011, 31

[9] Jiao Ming-hai, Yan Ping, Jiang Hui-yan. Research and application on Web information retrieval based on improved FP-growth algorithm[J]. Wuhan University Journal of Natural Sciences, 2006, 11(5):1065-1068

[10] 章志刚,吉根林. 一种基于 FP-Growth 的频繁项集并行挖掘算法[J]. 计算机工程应用,2014,2(2):103-106