

# 基于 MapReduce 的关联规则增量更新算法

朱晓峰,李玲娟,徐小龙,陈建新

(南京邮电大学 计算机学院,江苏 南京 210003)

**摘要:**云计算以其强大的存储和计算能力而成为解决海量数据挖掘问题的有效途径。经典的关联规则增量更新算法 FUP 需要频繁扫描原数据集,不适用于海量数据的处理。文中以提高海量数据上关联规则增量更新效率为目标,将 FUP 算法与云计算的 MapReduce 编程模式相结合,提出了一种基于 MapReduce 的关联规则增量更新算法 MRFPUP。该算法只需扫描原数据集一次,并能充分利用云计算强大的存储和并行计算能力。基于 Hadoop 的实验结果表明,MRFPUP 算法可提高对海量数据的处理能力和效率,适用于海量数据的关联规则挖掘。

**关键词:**海量数据挖掘;云计算;映射/规约;关联规则;增量更新

中图分类号:TP311

文献标识码:A

文章编号:1673-629X(2012)04-0115-04

## MapReduce Based Association Rule Incremental Updating Algorithm

ZHU Xiao-feng, LI Ling-juan, XU Xiao-long, CHEN Jian-xin

(College of Computer, Nanjing University of Posts and Telecommunications, Nanjing 210003, China)

**Abstract:** Cloud computing, with its powerful storage and computing power, has become one of the most effective way for solving the problem of massive data mining. FUP is one of the most classic incremental updating algorithms for association rules. But it can not meet the need of massive data mining very well because it needs to scan the dataset frequently. In this paper, in order to enhance the incremental updating efficiency of association rules for massive data, a MapReduce based incremental updating algorithm for association rules is proposed by combing FUP algorithm and MapReduce programming mode, which is named MRFPUP. MRFPUP scans the original dataset only once, and takes full advantage of the powerful storage and computing power provided by cloud computing. The results of the experiments deployed on Hadoop show that MRFPUP can improve the ability and efficiency of processing massive data; It adapts to mine association rules from massive data.

**Key words:** massive data mining; cloud computing; MapReduce; association rules; incremental updating

## 0 引言

时至今日,海量数据时代的来临已经毋庸置疑,尤其是在互联网、电信、金融等行业,几乎已经到了“数据就是业务本身”的地步。而对海量数据的处理是数据挖掘领域一个常见的问题。关于海量数据挖掘很早就有研究,主要成果有数据约简、分布式并行处理、批处理、增量式处理等策略与算法。其中增量式挖掘算法的核心就是如何利用已挖掘的规则,在变化了的数据集或参数上发现新的规则、删除失效的规则等规则维护更新问题。这些算法高效的关键在于尽可能利用

已有的挖掘结果来生成较小的候选项集或较少次地扫描数据集。目前大多数的关联规则增量式更新算法都是以 Apriori<sup>[1~4]</sup>算法为核心进行改进与优化的。在众多的增量式更新算法中,D. W. Cheung 等提出的 FUP<sup>[5]</sup>算法是最典型、最有效和最实用的算法之一,但是该算法依然存在着频繁扫描数据集等缺点。近年来兴起的云计算<sup>[6,7]</sup>以其超大规模、虚拟化、高可靠性、高扩展性等优点为海量数据的处理带来了新的解决方案。

由于云计算擅长处理大规模数据和进行大规模计算,所以改进传统的数据挖掘算法使之能部署到云计算平台上运行,将是解决海量数据挖掘问题的有效途径。为此,针对关联规则挖掘,文中以 FUP 算法为基础,提出了一种基于云计算的 MapReduce 模式的关联规则增量更新算法——MRFPUP 算法,该算法只需扫描原数据集一次,能有效地克服 FUP 算法需频繁扫描数据集的缺陷。

收稿日期:2011-09-15;修回日期:2011-12-20

基金项目:国家“973”计划资助项目(2011CB302903);国家自然科学基金资助项目(61073189)

作者简介:朱晓峰(1986-),男,浙江舟山人,硕士研究生,研究方向为数据挖掘、云计算;李玲娟,教授,博士,研究方向为数据挖掘、信息安全、分布式计算。

# 1 FUP 算法与 MapReduce 计算模型

## 1.1 FUP 算法

FUP 算法的基本框架和 Apriori 算法是一致的。它有多次循环,从 1-项集开始,每次循环找出相同长度的频繁项集。后续的各次循环都根据上一次循环找出的频繁项集生成候选项集。假设  $D$  为原数据集,  $d$  为新增数据集,  $L_k$  为  $D$  中频繁  $k$ -项集的集合,  $L_k^*$  为更新后数据集  $D \cup d$  中的频繁  $k$ -项集的集合,  $C_k$  为第  $k$  次循环生成的候选  $k$ -项集的集合,  $X$ ,  $\text{support}_D$ ,  $X$ ,  $\text{support}_d$ ,  $X$ ,  $\text{support}_{D \cup d}$  分别为项集  $X$  在  $D$ 、 $d$ 、 $D \cup d$  中的支持度计数,  $s$  为最小支持度。

FUP 算法的主要思想及步骤可以概括如下:

(1) 扫描新增数据集  $d$  完成两个任务。

(a) 对所有  $X \in L_k$ , 更新  $X$ ,  $\text{support}_{D \cup d}$ 。当扫描结束时, 若  $X$ ,  $\text{support}_{D \cup d} < s * (D + d)$ , 则从  $L_k$  中删去  $X$ , 剩下的  $L_k$  中的项集则是  $D \cup d$  上的频繁项集。

(b) 在  $d$  中找到所有候选频繁  $k$ -项集集合  $C_k$ , 对所有  $k$ -项集  $X \in C_k$  满足:  $X \subseteq T, T \in d; X \notin L_k$ 。扫描  $d$ , 计算  $C_k$  中每个候选项集  $X$  在  $d$  上的支持度计数。若  $X$ ,  $\text{support}_d < s * d$ , 则  $X$  不可能是  $D \cup d$  上的频繁项集, 从  $C_k$  中修剪掉  $X$ , 这样得到一个较小的候选项集集合  $C_k$ 。

(2) 扫描原数据集  $D$ , 更新  $C_k$  中每个项集的支持度计数, 以发现  $D \cup d$  上的新的频繁项集。这些新的频繁项集和(1)中剩余的  $L_k$  组成了  $D \cup d$  上最终的频繁  $k$ -项集集合  $L_k^*$ 。

FUP 算法的第  $k$  次循环仅扫描整个数据集一次。对于新的频繁项集, 它们的候选项集根据  $d$  上的支持度先进行修剪, 这个开销比在  $D \cup d$  中使用 Apriori 算法的开销小的多。因此, FUP 算法比再次使用 Apriori 算法更新关联规则更快。但 FUP 算法在对候选项集进行匹配来发现所有的频繁项集时仍需多次重复扫描数据集, 当数据集海量时, 会因计算量的增大而速度减慢甚至无法运行。

## 1.2 MapReduce 计算模型

从以上对 FUP 算法的介绍可以看出, 随着数据集的海量化, FUP 算法的挖掘效率将会变低, 因此, 文中借助云计算来提高海量数据的增量更新效率。

广泛采用的云计算编程模式是 MapReduce<sup>[8,9]</sup>, 它是 Google 公司的核心计算模型, 它将复杂的运行于大规模集群上的并行计算的过程高度抽象成两个函数:

Map 和 Reduce。Map/Reduce 编程模式首先会把输入数据分割成若干键/值对  $\langle \text{key}, \text{value} \rangle$  集, 由多个 map 任务来并行地处理, 并将同属于一个键的值组合在一起, 即以  $\langle \text{key}, \text{list}(\text{value}) \rangle$  作为 Reduce 任务的输入, 由 Reduce 计算最终结果并输出。

图 1 展示了 MapReduce 的工作流程。

Map/Reduce 的主节点 Master 将用户定义的 Map/Reduce 任务分发到计算机集群中的各个工作节点 Worker 上, 各工作节点并行处理自己的任务, 处理后的结果会被收集, 并经过进一步计算得出最终结果。

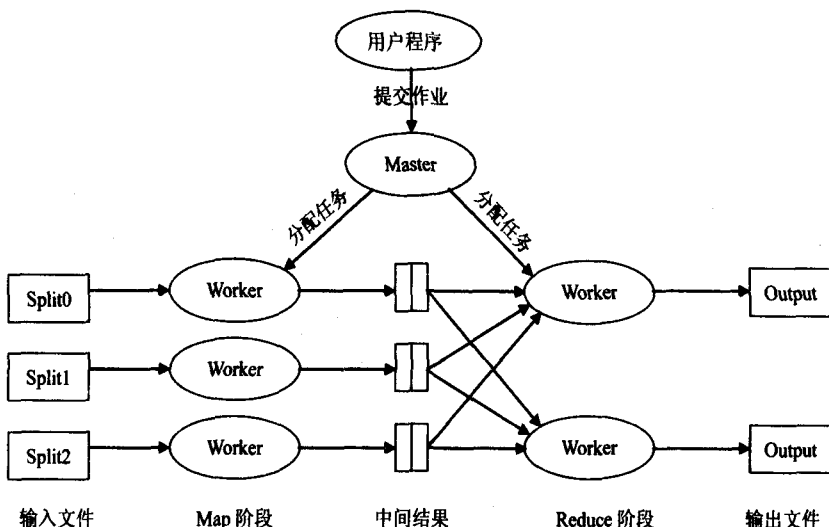


图 1 MapReduce 工作流程

## 2 MRFUP 算法及实现

MRFUP 是针对 FUP 算法存在的问题并利用云计算的特点而提出的。

### 2.1 MRFUP 算法描述

设原数据集为  $D$ , 新增数据集为  $d$ ,  $d$  远小于  $D$ , 原数据集  $D$  的频繁项目集集合为  $L(D)$ 。MRFUP 算法可描述如下:

(1) 对新增数据集  $d$  进行关联规则挖掘, 得到  $d$  的所有频繁项集  $L(d)$ , 比较  $L(d)$  和  $L(D)$ , 找出其公共部分, 将公共部分放入更新后的数据集  $(D + d)$  的频繁项集中, 记为  $L^*$ ,  $L(D) + L(d)$  剩余的项集为  $(D + d)$  的候选频繁项集, 记为  $C_{MR}$ 。

(2) 把数据集  $(D + d)$  分成规模相当的  $P$  个数据子集, 把数据子集和  $C_{MR}$  分别发送到  $P$  个站点。

(3) 每个站点扫描它的数据子集, 与  $C_{MR}$  中的项集进行匹配, 得到  $C_{MR}$  中各项集的支持度计数并标记, 一般标记为 1。

(4) 利用分区函数把  $P$  个站点的  $C_{MR}$  中相同的项集和它的支持度计数发送到  $Q$  个站点。

(5)  $Q$  个站点把相同项集的计数累加起来, 并加上

$C_{MR}$  中的初始支持度计数,产生最后的实际支持度计数,与最小支持度计数  $\min\_sup$  比较,确定局部频繁项集的集合。

(6) 把  $Q$  个站点的输出合并,并加上(1)中的  $L^*$ ,  $L^*$  即为更新后数据集的全部频繁项集的集合。

可见, MRFUP 算法的优势是只需要扫描原数据集一次就能找到所有的频繁项集,并且该算法和 MapReduce 计算模型结合,可充分利用云计算集群强大的存储和计算能力。

## 2.2 MRFUP 算法实现

MRFUP 算法的实现包括两部分:

(1) 在单机上的实现(串行部分)。

伪代码如下:

输入:  $L(D); d; \min\_sup$ 。

输出:  $L^*; C_{MR}$ 。

方法:

1)  $C_1$ :  $\text{init-pass}(d)$ ; //对新增数据集  $d$  进行第一轮扫描

2)  $L_1$ :  $\{f | f \in C_1, f.\text{count}/n \geq \min\_sup\}$ ;

// $n$  是  $d$  中的事务数

3) for( $k=2$ ;  $L_{k-1} \neq \varnothing$ ;  $k++$ ) do //随后的各轮搜索

4)  $C_k$ :  $\text{candidate-gen}(L_{k-1})$ ;

5) for each transaction  $t \in T$  do //对所有事务扫描一遍

6) for each candidate  $c \in C_k$  do

7) if  $c$  is contained in  $t$  then

8)  $c.\text{count}++$ ;

9) endfor

10) endfor

11)  $L_k$ :  $\{c \in C_k | c.\text{count}/n \geq \min\_sup\}$ ;

12) endfor

13)  $L(d)$ :  $L_k$ ; //合并所有的频繁项集

14)  $L^*$ :  $L(d) \cap L(D)$ ; //求  $D$  和  $d$  的公共部分, 即  $L^*$

15)  $C_{MR}$ :  $(L(d) - L^*) \cup (L(D) - L^*)$ ;

其中的  $\text{candidate-gen}(L_{k-1})$  的步骤如下:

1)  $C_k$ :  $\varnothing$ ;

2) for all  $f_1, f_2 \in L_{k-1}$

3) with  $f_1 = \{i_1, \dots, i_{k-2}, i_{k-1}\}$

4) and  $f_2 = \{i_1, \dots, i_{k-2}, i'_{k-1}\}$

5) and  $i_{k-1} < i'_{k-1}$  do

//找出只有最后一项不同的频繁项集对

6)  $c$ :  $\{i_1, \dots, i_{k-2}, i_{k-1}, i'_{k-1}\}$ ; //根据数据字典将  $f_1$  和  $f_2$  合并

7)  $C_k$ :  $C_k \cup \{c\}$ ; //将新项集  $c$  加入  $C_k$

8) for each  $(k-1)$ -subset  $s$  of  $c$  do

9) if  $(s \notin L_{k-1})$  then

10) delete  $c$  from  $C_k$ ;

11) endfor

12) endfor

13) return  $C_k$ ;

(2) 基于 MapReduce 的实现(并行部分)。

伪代码如下:

输入:  $(D+d)$ ; 串行部分得到的  $L^*, C_{MR}$  及其

$\sup_{MR}$ 。

输出: 更新后的数据集  $(D+d)$  的频繁项集  $L_{Dd}$ 。

方法:

1) Map( $Tid, C_{MR}$ ) //  $Tid$ : 事务标识符

2) for each itemsets in  $C_{MR}$ :

3) if (itemsets 匹配  $Tid$ )

4) EmitIntermediate(itemsets, 1);

5) Combiner(itemsets, list(1)) // itemsets:  $C_{MR}$  中的项集

// list(1): 项集对应支持度计数 1 的列表

6) int  $sup = 0$ ;

7) for each  $l$  in list:

8)  $sup++$ ;

9) EmitIntermediate(itemsets,  $sup$ );

10) Reduce(itemsets, list( $sup$ )) // list( $sup$ ): 计数列表

11) int  $result = 0$ ;

12) for each  $sup$  in list:

13)  $result += \text{ParseInt}(sup)$ ;

14) Emit(itemsets,  $result + \sup_{MR}$ );

15) if  $(result + \sup_{MR})/N > \min\_sup$  then

// $N$  为  $D+d$  中事务的数目

16)  $L_{Dd}$ :  $L^* \cup \{\text{itemsets}\}$ ;

17) return  $L_{Dd}$ ;

## 3 实验与结果分析

### 3.1 实验平台搭建

Hadoop<sup>[10-12]</sup> 是一个典型的云计算框架,能够对大量数据进行分布式处理,主要是由 HDFS<sup>[11]</sup> 和 MapReduce 两个核心模块组成。HDFS 包括: Client、NameNode 和 DataNode。其工作过程是:用户的各种请求指令包括读和写,由 NameNode 接收;NameNode 将合适的 DataNode 的信息返回给用户,由用户直接与 DataNode 进行数据传送;NameNode 同时处理相关的元数据。Hadoop 具有可靠性、高效性和可伸缩性,对服务器配置的要求也不高。

为此,文中采用 Hadoop 开源软件来搭建实验用的

云计算平台,用了五台 PC 机和一个交换机。实验的软件配置见表 1,集群的 IP 配置见表 2。

表 1 实验的软件环境配置

Linux 版本	Ubuntu 9.10
Hadoop 版本	Hadoop-0.20.2
JDK 版本	jdk-6u13-linux-i586
SSH 版本	OpenSSH 5.8

### 3.2 实验结果分析

实验使用的数据由 IBM 数据生成器<sup>[12]</sup>生成,分别产生大小为 0.1G、0.3G、0.5G、1G、2G、3G、4G 的 7 个原数据集  $D$ ,以及 1 个 0.2G 的新增数据集  $d$ ,最小支持度  $\min\_sup$  置为 20%。用 C++ 实现 FUP 算法和 MRFUP 算法,并通过 Hadoop Streaming 将 MRFUP 算法运行于集群上。

表 2 集群 IP 配置

机器名	IP	用途
U0	10.10.139.21	NameNode, JobTracker
U1	10.10.139.94	DataNode, TaskTracker
U2	10.10.139.34	DataNode, TaskTracker
U3	10.10.139.138	DataNode, TaskTracker
U4	10.10.139.33	DataNode, TaskTracker

#### (1) 单机伪分布环境下的性能比较。

首先在 U0 上搭建单机伪分布环境,运行 MRFUP 算法,对 MRFUP 算法和 FUP 算法进行了性能对比实验,结果如图 2 所示。

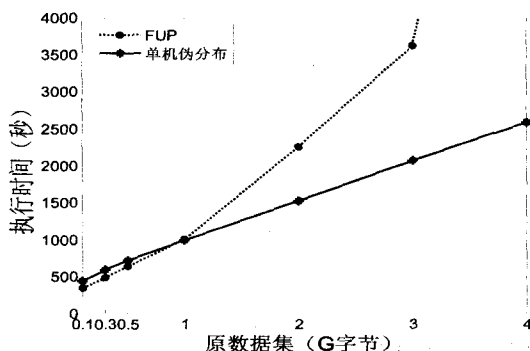


图 2 FUP 和 MRFUP 单机伪分布环境的比较

从图 2 中可以看出,在同样的软硬件环境配置下,当原数据集较小时, FUP 算法的时效性优于 MRFUP 算法,但随着原数据集的增大, FUP 算法的性能明显下降,特别是当原数据集为 4G 时,由于内存调用失败导致 FUP 算法无法执行。而 MRFUP 算法的性能却随着原数据集的增大而保持稳定,原数据集越大, MRFUP 相对于 FUP 算法的优势越明显。这是因为, FUP 算法在执行过程中需要扫描  $k$  遍原数据集  $D$ ,这里的  $k$  视情况而定,在本实验中, FUP 算法需要扫描原数据集  $D$  两遍,  $k$  为 2。当原数据集很大时,多次扫描原数据集

消耗的时间将是无法忍受的。而 MRFUP 算法将原数据集分块执行,用到了 Partition<sup>[7,8]</sup> 的思想,只需要扫描原数据集  $D$  一遍;当数据集较少的时候, Map 和 Reduce 的调度代价在整个算法执行消耗中所占的比例很大,但是随着原数据集的增大,这个比例大大降低。所以从图 2 中可以发现,当面对海量数据时, MRFUP 算法的性能更优于 FUP 算法。

#### (2) 完全分布环境下的性能比较。

以 U0 为 Master, U1 ~ U4 为 Slaves, 分别搭建 1 个 DataNode、2 个 DataNode、3 个 DataNode、4 个 DataNode 构成的分布集群,在这些集群上运行 MRFUP 算法,结合前面的单机实验,得到如图 3 所示的实验结果。

从图 3 中可以发现当原数据集较小时,集群并行的 MRFUP 算法的处理效率和单机非并行的 FUP 算法的处理效率相似。这是因为集群的启动和运行过程中的交互需要消耗一定的资源,当原数据集很小时,算法实际运算时间在整个消耗时间中所占的比例很小,甚至远小于集群启动和交互的时间。随着原数据集的增大,集群的运行速度比单机 FUP 算法的运行速度明显加快。集群高度的 I/O 并行性和强大的计算能力是出现这种现象的原因。从图中还可以看出, MRFUP 算法具有较好的可伸缩性,即随着数据集的增长,算法的运行时间线性增长。

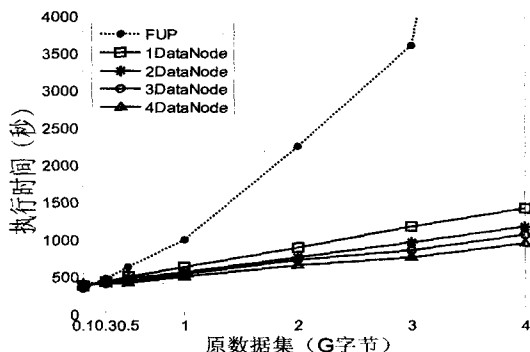


图 3 FUP 和 MRFUP 集群环境的比较

## 4 结束语

文中以 FUP 算法为基础,结合云计算的 MapReduce 模型,提出了一种基于 MapReduce 的关联规则增量更新算法 MRFUP,该算法的特点是只需扫描原数据集一次。实验结果表明: MRFUP 算法能充分利用云计算强大的存储和计算能力,数据量越大,相对于 FUP 算法的性能优势越明显,达到了提高对海量数据的挖掘能力和效率的目的。

#### 参考文献:

- [1] Agrawal R, Srikant R. Fast Algorithms for Mining Association

(下转第 122 页)

呈现出递增的趋势,而附属笔画‘♥’则与之相反,这样,根据其形状信息就可以把‘♣’和‘♥’这两者区分开来。

(4)水平/垂直投影:在对诸如‘1’和‘9’这两个附属笔画进行投影后特征明显,如图 5(a)中所示‘1’的水平投影时的行宽以及垂直投影时的列高变化细微或基本保持不变,而图 5(b)中所示‘9’水平投影时的行宽以及垂直投影时的列高则出现剧减的情况。图 5(c)、5(d)则分别表示两个附属笔画的垂直投影。

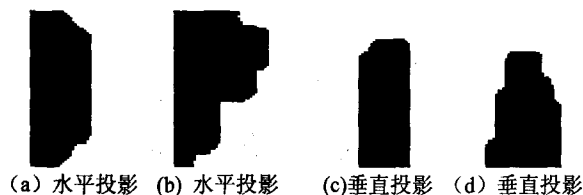


图 5 两种附属笔画的水平投影及垂直投影

#### 4 结束语

文中的算法通过提取维吾尔文字母特征量,对待识别的字符进行带有冗余的分类(粗分类),力求在保证分组准确的基础上尽可能缩小字符匹配范围,以提高识别的过程的运算速度。同时分组后的模板匹配算法从各组中识别出最后结果,保证了识别的正确率。经实验测得在含有 400 多个样本的测试集上的识别率达到 94%。实验结果表明两者的结合使系统具有较高的识别正确率与识别速度。

#### 参考文献:

[1] 袁保社,吾守尔·斯拉木.一种手写维吾尔文字母识别算

法[J].计算机工程,2010,36(2):186-188.

- [2] 靳简明,王 华,丁晓青.维汉英混排文档识别[J].电子与信息学报,2006,28(7):1188-1191.
- [3] 王 华,丁晓青,哈力木拉提.多字体多字号印刷维吾尔文字符识别[J].清华大学学报(自然科学版),2004,44(7):946-949.
- [4] 靳简明,丁晓青,彭良瑞,等.印刷维吾尔文本切割[J].中文信息学报,2005,18(5):76-83.
- [5] Amin A, Mari J F. Machine recognition and correction of printed Arabic text[J]. IEEE Transactions on Systems, Man and Cybernetics, 1989, 19(5): 1300-1306.
- [6] Al-Muallim H, Yamaguchi S. A method of recognition of Arabic cursive handwriting[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1987, 9(5): 715-722.
- [7] Olivier C, Miled H, Romeo-Pakker K, et al. Segmentation and coding of Arabic handwritten words [C]//Proceedings of 13th International Conference on Pattern Recognition. Vienna, Austria: [s. n.], 1996: 264-268.
- [8] Romeo-Pakker K, Miled H, Lecourtier Y. A new approach for Latin/Arabic character segmentation [C]//Proceedings of the 3rd International Conference on Document Analysis and Recognition. Montréal, Canada: [s. n.], 1995: 874-877.
- [9] 吴伟伟,王小红,周亚南.字符识别中两种改进的模板匹配算法[J].传感器世界,2008,14(6):35-37.
- [10] 崔 政,李 壮.两种改进的模板匹配识别算法[J].计算机工程与设计,2006,27(6):1083-1085.
- [11] 张 晶,李志敏,黄 凡.一种改进的自适应模板匹配法[J].微计算机信息,2008,24(9):166-167.
- [12] 哈力木拉提,阿孜古丽.多字体印刷维吾尔文字符识别系统的研究与开发[J].计算机学报,2004,27(11):1480-1484.

(上接第 118 页)

- Rules [C]//Proceedings of 20th International Conference on Very Large Database. Santiago Chile: [s. n.], 1994: 487-499.
- [2] 范 明,孟小峰.数据挖掘:概念与技术[M].北京:机械工业出版社,2001.
- [3] 程舒通,徐从富.关联规则挖掘技术研究进展[J].计算机应用研究,2009,26(9):3210-3213.
- [4] Savasere A, Omiecinski E, Navathe S. An Efficient Algorithm for Mining Association Rules in Large Databases [C]//Proc. 21st Int'l Conf. on Very Large Databases. San Francisco: Morgan Kaufmann, 1995: 432-444.
- [5] Cheung D W. Maintenance of Discovered Association Rules in Large Database: An Incremental Updating Technique [C]//Proc. of 1996 Intl. Conf. on Data Engineering. [s. l.]: IEEE Computer Soc. Press, 1996: 106-114.
- [6] Weiss A. Computing in Clouds [J]. ACM Networker, 2007, 11(4): 18-25.
- [7] 刘 鹏.云计算[M].北京:电子工业出版社,2010.
- [8] Dean J, Ghemawat S. Mapreduce: simplified data processing on large clusters [C]//Proceedings of the 6th Symposium on Operating System Design and Implementation. San Francisco, California, USA: [s. n.], 2004: 137-150.
- [9] Rajaraman A, Ullman J D. Mining of Massive Data [M]. Stanford: [s. n.], 2010.
- [10] Venner J. Pro Hadoop [M]. [s. l.]: Apress, 2009.
- [11] White T. Hadoop: The Definitive Guide [M]. [s. l.]: O'Reilly Media, Yahoo! Press, 2009.
- [12] Ghemawat S, Gobioff H, Leung S. The Google Filesystem [C]//Proc. of ACM Symposium on Operating Systems Principles. Lake George, NY: [s. n.], 2003: 29-43.