一种基于 Chernoff Bound 的数据流上近似频繁项集的挖掘方法

李海峰 章 宁
(中央财经大学信息学院 北京 100081)

摘 要 数据流高速、无限和动态的特点决定了必须在有限的内存中以尽快的计算速度完成流数据上的频繁项集挖掘。将数据流中的数据按照段进行划分,采用二元组列表的数据结构进行保存,提出了一种基于滑动窗口的近似频繁项集挖掘方法 AFIoDS,以实时获取频繁项集集合的真子集,并引入了概率参数,利用 Chernoff Bound 来动态改变支持度的近似值,保证真子集中的频繁项集被限制在一定的误差范围之内。此外,为了进一步节省内存,AFIoDS 采用闭合项集的形式压缩每个段中获取的频繁项集。通过在 3 种真实数据集上的实验表明,AFIoDS 算法与现有算法相比,在精度没有下降的情况下,具有更快的处理速度,同时其存储开销大大降低。

关键词 Chernoff Bound,数据流,频繁项集
中图法分类号 TP312 **文献标识码** A

Chernoff Bound Based Approximate Frequent Itemset Mining Method over Streams

LI Hai-feng ZHANG Ning
(School of Information, Central University of Finance and Economics, Beijing 100081, China)

Abstract A data stream is fast, unlimited and dynamic, these characteristics constraint the computational resources and storages when mining frequent itemsets. This paper addressed this problem and proposed a simple and effective algorithm AFIoDS, AFIoDS is an approximate algorithm based on sliding window model, which splits stream data into batches and maintains them with 2-tuple lists; thus, a false negative result can be obtained using a probabilistic parameter based on chernoff bound. The approximation will be changed dynamically to guarantee the mining frequent itemsets are error controllable. Plus, a compression of frequent itemsets, the closed frequent itemsets, are employed to represent the results of each batch for further memory saving. Our experimental results on 3 real world data show that without precision reduction, AFIoDS achieves a faster speed and a much reduced memory cost in comparison with the state-of-the-art algorithms.

Keywords Chernoff bound, Data stream, Frequent itemset

1 引言

频繁项集是由 Agrawal 在 1994 年提出来的[1],其动机是寻找超市事务数据的频繁集,用以分析客户的购买行为。近年来随着在众多行业中的应用,频繁项集挖掘受到了广泛关注,逐渐发展成为数据挖掘中的重要角色,其目标是在数据集中寻找用户感兴趣的模式,例如关联规则[2]、数据相互关系[4]、序列模式[3]等,以便为商业分析、网络流量检测与网络监控和信息安全等多种应用服务。

在数据流的环境下,频繁项集挖掘的方法呈现出新的特点。数据流是动态数据的一种典型代表,其高速、无限、连续且动态变化的特点不但要求频繁项集的挖掘算法必须用有限的内存空间去保存和处理数据,而且需要实现对数据的增量分析处理。这些挑战成为研究数据流管理和挖掘的人员关注的焦点。

通常,数据流上的近似频繁项集挖掘技术都是挖掘 false positive 的结果。也就是说,在这些算法中,对于给定的阈值,提供一个缓冲参数,适当降低阈值,将部分当前的非频繁项集作为数据流中未来可能的频繁项集保存,以满足数据流中实时挖掘的特点。但这种技术由于需要保存大量的非频繁项集,因此增加了空间代价。

本文提出了一种 false negative 的方法来解决这一问题,主要贡献如下。

首先,提出了数据流上基于滑动窗口模型的近似频繁项集的方法 AFIoDS。算法只保存真实频繁项集的真子集,并利用 chernoff bound 来保证真子集无限接近真实的频繁项集集合。一方面,由于不再保存非频繁项集,因此节省了大量的内存空间;另一方面,虽然损失了少许的真实结果,但通过参数设置实现误差可控,实现了挖掘的精度和运行代价的折中。

其次,将滑动窗口的数据进行了分段,采用了二元组的数据结构建立数据大纲,并通过小范围数据的挖掘和总体结果的合并,实现了数据在不断到达和离开情况下的增量更新。

到稿日期:2010-06-11 返修日期:2010-10-12 本文受中央财经大学"211 工程"三期资助。
李海峰(1979—),男,博士生,讲师,主要研究方向为数据挖掘、商务智能管理,E-mail:mydlhf@126.com;章 宁(1976—),女,博士生,教授,主要研究方向为信息管理、电子商务。

再次，将每段数据的挖掘结果用闭合项集的形式进行保存，既降低了空间代价，又不损失结果的精确度。

最后，通过在3种真实数据集上的实验，与最新近似频繁项集挖掘算法进行了比较，验证了本算法的可行性和有效性。

本文第2节介绍了相关知识，并对需要解决的问题进行了定义；第3节介绍了相关工作；第4节详细描述了AFIoDS算法；第5节进行了实验；最后对本文进行了总结。

2 预备知识与问题定义

2.1 预备知识

2.1.1 频繁项集

用 $\Gamma=\{i_1,i_2,\cdots,i_n\}$ 来表示所有不同项的集合，其中 $|\Gamma|=n$ 表示 Γ 的大小，称长度为 k 的 Γ 的子项集 X 为 k-项集。为了简单起见，可以把项集 $X=\{x_1,x_2,\cdots,x_m\}$ 表示为 $x_1x_2\cdots x_m$。对于数据集 $D=\{T_1,T_2,\cdots,T_v\}$，每一个 $T_i(i=1\cdots v)$ 表示一个基于 Γ 的事务，包含事务 id 和对应的项集 X。对于事务 $T=(\text{tid},X)$ 来说，如果存在项集 Y 使得 $Y\subseteq X$，则称事务 T 支持项集 Y，D 中支持项集 Y 的事务称为 Y 的覆盖，表示为 $cover(Y,D)=\{T\mid T\subseteq D\wedge Y\subseteq X\wedge X\in T\}$。$Y$ 的绝对支持度是其覆盖集合的大小，表示为 $\Lambda(X,D)=|cover(X,D)|$，其相对支持度是 Y 在 D 中发生的概率，即 Y 的绝对支持度与数据集 D 大小的比值，表示为 $\Lambda_r(Y,D)=\Lambda(Y,D)/|D|$。

定义1 给定数据集 D 和最小支持度阈值 λ，如果 $\Lambda_r(Y,D)\geqslant\lambda$，则称项集 Y 为频繁项集。

2.1.2 Chernoff Bound

给定 n 个符合贝努利实验的独立观察值 $o_1,o_2,\cdots,o_n$ 和概率 p，有 $\Pr[o_i=1]=p$，$\Pr[o_i=0]=1-p$，令 r 为 o_i 为1的实际值，则 r 的期望值为 np。对于任给的 $\eta>0$，都有

$$\Pr\{|r-np|\geqslant np\eta\}\leqslant 2e^{\frac{-np\eta^2}{2}}$$

令 $\bar{r}=r/n$，最小支持度 $\lambda=p$，则有

$$\Pr\{|\bar{r}-\lambda|\geqslant\lambda\eta\}\leqslant 2e^{\frac{-n\lambda\eta^2}{2}}$$

令 $\varepsilon=\lambda\eta$，则有

$$\Pr\{|\bar{r}-\lambda|\geqslant\varepsilon\}\leqslant 2e^{\frac{-n\varepsilon^2}{2\lambda}} \tag{1}$$

令不等式(1)的右侧表达式为 δ，则有 o_i 为1的平均概率值以小于 δ 的概率位于 $[\lambda-\varepsilon,\lambda+\varepsilon]$ 之外。其中

$$\varepsilon=\sqrt{\frac{2\lambda\ln(2/\delta)}{n}} \tag{2}$$

也就是说，给定项集 X 的 n 个观察值，其真实支持度 $\Lambda_r(X,D)$ 会以大于 $1-\delta$ 的概率存在于 $[\lambda-\varepsilon,\lambda+\varepsilon]$ 之内。例如，给定 $\lambda=0.1$，$\delta=0.1$ 和 $\varepsilon=0.01$，根据 Chernoff Bound，$n\approx 5991$。也就是说，对于项集 X 的5991个观察值，其真实支持度以高于0.9的概率位于 $[0.09,0.11]$ 之内。

2.2 问题定义

滑动窗口模型能够满足用户关注最近时间所产生的结果，又能够解决流数据无限造成的无法保存的问题，因此适合数据流的挖掘。本文所要解决的问题是在数据流上动态生成最近 n 个数据的近似频繁项集。图1的示例将滑动窗口分为3个段，当数据不断到达时，每次插入一个段的数据，同时删除最早到达段中的数据。

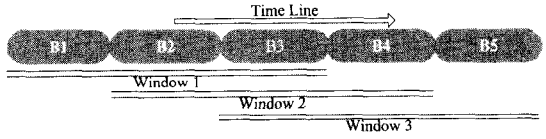

图1 滑动窗口模型

3 相关工作

传统的频繁项集挖掘方法是在静态数据上进行挖掘，挖掘主要基于层次、模式增长和垂直数据格式来优化算法性能；根据不同的应用，提供各种约束条件来提高算法速度。另外，针对频繁项集自身存在的冗余，提出了对频繁项集的多种压缩表示的挖掘方法。数据流上频繁项集挖掘研究集中在如何在有限的内存空间内实时完成计算。

文献[1]采用了CountSketch的数据结构，通过两组哈希函数来估计项目的支持度，并根据数据结构的可增量计算特性，寻找数据流中支持度变化最大的项目。文献[2]提出了MG算法，用来计算 n 个项目中发生次数大于 n/k 的项目，其时间复杂度为 $O(n\log(k))$，空间复杂度为 $O(k\log(k))$；很多算法将其性能进行了改善，其中代表性的算法 EC[3] 能够在存在误差的情况下将时间复杂度降为 $O(n)$，空间复杂度降为 $O(k)$。文献[4]提出了采样算法 StickSampling，它在不同时间段去估计每个项目的频繁度；文献[4]还提出了 LossyCounting 算法，其利用 MG 算法将数据分在多个桶中，并实时根据滑动窗口内容的变化更新桶的内容。文献[5]考虑了在数据流中有数据项目不断离开的情况，采用多层分组的方法消除由于采样产生的误差。文献[7]则利用衰减因子来解决这一问题，通过一次扫描完成在线统计，用分段线性表示来进行数据压缩，以有效地预测支持度的变化趋势。文献[6]考虑了数据流处理中的内存限制，提出了 false negative 的算法 FDPM，采用 Chernoff 方法来估计最小支持度，并通过动态参数来决定候选集的大小。文献[8]尝试对数据流只进行一次扫描来完成挖掘，将数据流分为多个段，将前一个段的频繁项集作为后一个段的候选项集；算法同样采用了 chernoff bound 来计算段的长度，以保证算法的精确性。文献[9]采用了滑动窗口模型，提出了算法 estWin，通过设置支持度的误差范围，提前获取潜在的频繁项集，从而实现实时的频繁项集挖掘。文献[10]为了降低滑动窗口中不断离开的事务对挖掘结果的影响，提出了 ATS 算法，它利用平均时间戳来判断项集是否符合一致性分布；提出了 FCP 算法，其利用频繁度改变点的比较，缩小滑动窗口的范围。文献[11]考虑了数据流上的大型滑动窗口，提出了 SWIM 方法，即通过模式树 PT 直接保存频繁项集，并利用条件计数来完成频繁项集的验证。文献[12]采用衰减因子来反映数据流的变化趋势。文献[13]则利用倾斜窗口来保存过去时间点的频繁项集的支持度，以在满足用户时间相关查询的情况下压缩空间使用情况。文献[14]维护了比前缀树压缩比率更高的后缀树 ISFI-forest，即采用模糊的最小支持度来获取潜在的频繁项集。文献[15]采用了数据分段的思想，以适合项集长度较长的数据。文献[16]提出了算法 StreamMining，即利用 treehash 的数据结构，通过项目统计减少子项集的检测，通过在线检测来减少子项集的插入。

4 AFIoDS 算法

AFIoDS 算法利用二元组列表来保存滑动窗口的数据大

纲，并实现了在事务不断插入和删除时的动态维护。

4.1 数据结构

算法根据 chernoff bound 将项集分为 3 种类型：给定最小支持度 λ、概率参数 δ 和数据集 D，对于项集 X 来说，如果 X 的支持度满足 $\Lambda_r(X,D)\geqslant\lambda$，则称 X 为真实频繁项集；如果 X 的支持度满足 $\lambda>\Lambda_r(X,D)\geqslant\lambda-\sqrt{\frac{2\lambda\ln(2/\delta)}{|D|}}$，则称 X 为可能频繁项集，否则称 X 为非频繁项集。

在任意一段数据上产生的项集，如果其类型为真实频繁项集或可能频繁项集，均有可能在整个滑动窗口成为全局的频繁项集，则采用多个二元组列表 X 和 Y 分别保存滑动窗口中每个段的真实频繁项集和可能频繁项集。当滑动窗口中填满数据后，需要对 X 和 Y 中的数据进行一次合并，得到最终的真实频繁项集和可能频繁项集，并用二元组列表 E 和 F 保存。

4.2 初始化算法

为了利用 chernoff bound 技术，将数据流的滑动窗口分割为多个段，然后分别对每个段进行挖掘，并在滑动窗口中填满数据时对结果进行合并。为了提高剪枝质量，限定每段数据的数量小于滑动窗口大小与 chernoff bound 参数 ε 的乘积。

引理 1 给定最小支持度 λ、概率参数 δ，观察值 n 一定满足 $n>\frac{2\ln(2/\delta)}{\lambda}$。

证明：由式(2)可知，$\varepsilon=\sqrt{\frac{2\lambda\ln(2/\delta)}{n}}$，因为 $\varepsilon<\lambda$，所以有 $n>\frac{2\ln(2/\delta)}{\lambda}$ 成立。

算法 1 给出了 AFIoDS 的初始化算法。首先根据给定的最小支持度和概率参数计算滑动窗口对应的 chernoff bound 参数，然后对滑动窗口中的数据进行分段。在滑动窗口的每个段中对每个项目进行计数，并根据支持度的范围分别保存。然后生成频繁项集，并根据阈值的不同分为真实频繁项集和可能频繁项集，以闭合项集的形式分别保存。当滑动窗口中的每个段都计算完毕后，将所有项集合并重新计算，将真实频繁项集保存到 E 中，将可能频繁项集保存到 F 中。

算法 1 初始化算法 INITIAL

输入：滑动窗口 W，最小支持度 λ，概率参数 δ；
输出：近似频繁项集集合 E.
方法：
(1)计算 chernoff bound 参数 ε；
(2)for(W 中的每个段 B_i)
(3)　for(每个项目 e)
(4)　　计数 e，得到二元组$\langle e,count\rangle$；
(5)　if($count$ 大于 $\lambda\times|B_i|$)
(6)　　$X_i=X_i\cup\{\langle e,count\rangle\}$；
(7)else if($count$ 大于 $(\lambda-\varepsilon)\times|B_i|$)
(8)　　$Y_i=Y_i\cup\{\langle e,count\rangle\}$；
(9)　根据频繁项生成频繁项集；
(10)　计算频繁项集的闭合项集
(11)　根据闭合项集剪枝 X_i 和 Y_i；
(12)合并 X_i 和 Y_i 到频繁项集集合 E 和 F

4.3 事务插入

为了与初始化算法保持一致性，令每次插入的数据数量与滑动窗口分段的大小相等。在事务插入过程中，部分可能频繁项集会转化为真实频繁项集，也有部分真实频繁项集会转化为可能频繁项集。如算法 2 所示，首先根据最小支持度计算插入事务生成的确定频繁项集和可能频繁项集。然后与已经生成的近似频繁项集合并，将相同的频繁项集的计数进行加和。对真实频繁项集进一步判断，如果其支持度小于最小支持度，则转为可能频繁项集；对可能频繁项集进一步判断，如果其支持度超过最小支持度，则转为真实频繁项集。

算法 2 事务插入算法 INSERT

输入：近似频繁项集集合 E 和 F，新到达事务集 B_n；最小支持度 λ，概率参数 δ
输出：近似频繁项集集合 E
方法：
(1)计算 B_n 的真实频繁项集和可能频繁项集；
(2)计算闭合项集并保存到 X_n 和 Y_n 中；
(3)$E=E\cup X_n$；
(4)if(项集 $I\in E$ 并且 I 的支持度小于 $\lambda\times|W|$ 且大于 $(\lambda-\varepsilon)\times|W|$)
(5)　　$E=E\backslash I$；
(6)　　$F=F\cup I$；
(7)else if(I 的支持度小于 $(\lambda-\varepsilon)\times|W|$)
(8)　　$E=E\backslash I$；
(9)$F=F\cup Y_n$；
(10)if(项集 $I\in F$ 并且 I 的支持度大于 $\lambda\times|W|$)
(11)　　$F=F\backslash I$；
(12)　　$E=E\cup I$；
(13)else if(I 的支持度小于 $(\lambda-\varepsilon)\times|W|$)
(14)　$F=F\backslash I$；

4.4 事务删除

当新事务到达后，滑动窗口中最早到达的事务将被删除掉。在事务删除的过程中，部分真实频繁项集会转化为可能频繁项集，而部分可能频繁项集会转化为非频繁项集。如算法 3 所示，首先根据最小支持度计算被删除事务生成的确定频繁项集和可能频繁项集，然后将已经生成的近似频繁项集集合中的相应项集进行减数。真实频繁项集，如果其支持度小于最小支持度，则转为可能频繁项集。因为对数据段的大小进行了限制，所以真实频繁项集不可能变为非频繁项集；进一步判断可能频繁项集，如果其支持度小于加入 chernoff bound 参数后的最小支持度，则删除该项集。

算法 3 事务删除算法 DELETE

输入：近似频繁项集集合 E 和 F，最早到达事务集 B_m；最小支持度 λ，概率参数 δ
输出：近似频繁项集集合 E
方法：
(1)计算 B_m 的真实频繁项集和可能频繁项集；
(2)计算闭合项集并保存到 X_m 和 Y_m 中；
(3)$E=E\backslash X_m$；
(4)if(项集 $I\in E$ 并且 I 的支持度小于 $\lambda\times|W|$)
(5)　　$E=E\backslash I$；
(6)　　$F=F\cup I$；
(7)$F=F\backslash Y_m$；
(8)if($I\in F$ 并且 I 的支持度小于 $(\lambda-\varepsilon)\times|W|$)
(9)　　$F=F\backslash I$；

5 实验与性能分析

用 Visual C# 在内存 2G、CPU 为 Xeon 2.0GHz、操作系统为 Windows Server 2003 的机器上实现并运行了 AFIoDS

算法，与目前性能最优的近似频繁项集挖掘算法 FDPM 和 Lossy Counting 进行比较。其中 FDPM 是 false negtive 的算法，而 Lossy Counting 是 false positive 的算法。采用了 3 种标准的数据集作为实验的测试数据集：包括两种 KDDCUP 2000 采用的真实数据集 BMS-POS 和 BMS-Webview，其中 BMS-POS 是商品零售数据，BMS-Webview 是电子商务网站的点击流数据；另外一种是匈牙利在线新闻门户网站的点击流数据 KOSARAK。表 1 列举了 3 种数据集的主要数据特征。

表 1　数据集的主要特征

数据集	事务数量	平均事务	最小事务	最大事务	项目数量
KOSARAK	990002	7.1	1	2497	497
BMS-POS	515597	6.5	1	164	1657
BMS-Webview	59601	2.5	1	267	497

5.1　时空代价比较

在实验中，AFIoDS 算法采用了统一的滑动窗口大小 $|W|=50000$，分段大小 $|B|=|W|/100=500$。令概率参数 $\delta=0.1$，表示最多有 10%的可能性误删真实频繁项集；令最小支持度 λ 从 0.001 到 0.9 变化，则相应的 chernoff bound 参数 ε 从 0.000034 到 0.01 变化。

为了测试空间代价，选取了 KOSARAK 作为测试数据集，并随机选取后续的 5000 数据作为滑动窗口的插入数据。如图 2 所示，由于 Lossy Counting 算法是 false positive 的算法，需要保存部分非频繁项集，因此使用的内存最多；而 FDPM 算法与 AFIoDS 算法是 false negtive 的算法，不需要保存非频繁项集，因此使用的内存较少。与 FDPM 算法相比，AFIoDS 使用了滑动窗口作为模型。

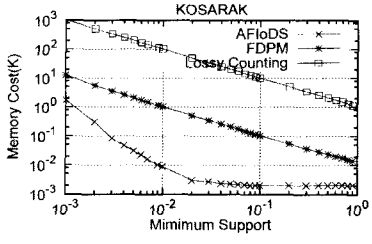

图 2　空间代价比较

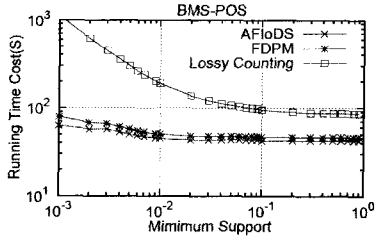

图 3　时间代价比较

为了测试空间代价，选取了 BMS-POS 作为测试数据集，并随机选取后续的 5000 数据作为滑动窗口的插入数据。如图 3 所示，Lossy Counting 算法需要较长的计算时间；而 FDPM 算法和 AFIoDS 算法的计算时间相似。这是因为 AFIoDS 采用了滑动窗口模型，虽然计算的数据量相对减少，但却增加了数据删除的计算代价以及在挖掘过程中对数据压缩产生的计算代价，因此计算性能的改善并不明显。尽管如此，AFIoDS 算法的计算时间还是要少于 FDPM 算法。

5.2　近似度比较

选取了 BMS-Webview 作为测试数据集。在实验中，AFIoDS 算法采用的滑动窗口大小 $|W|=10000$，分段大小 $|B|=|W|/100=100$。令概率参数 $\delta=0.1$，最小支持度 λ 从 0.01 到 0.1 变化，则相应的 chernoff bound 参数 ε 从 0.0024 到 0.007 变化。

采用精确度和召回度来比较算法的近似度。给定真实频繁项集集合 A 和算法计算得到的频繁项集集合 A' 两个集合，定义精确度为 $P=\frac{|A\cap A'|}{|A'|}$，召回度为 $R=\frac{|A\cap A'|}{|A|}$。精确度和召回度越大，表示计算所得结果越接近于真实结果。当精确度为 1 时，意味着 $A'\subseteq A$，即计算所得结果均为真实结果；当召回度为 1 时，意味着 $A\subseteq A'$，即计算所得结果包含了所有真实结果。

一方面，如图 4 所示，FDPM 算法和 AFIoDS 算法的精确度相同，都达到 100%。也就是说，它们挖掘的结果都是正确的。但 Lossy Counting 算法的精确度在 60%左右，这是因为该算法需要保存部分非频繁项集。另一方面，FDPM 算法和 AFIoDS 算法在理论上获得的结果只是真实结果的子集，但实际上，如图 5 所示，3 种算法得到的结果的召回度都达到了 100%。

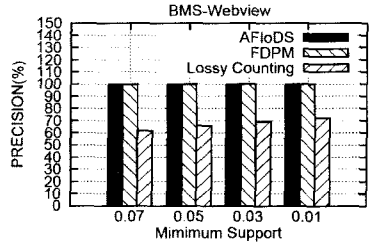

图 4　精确度比较

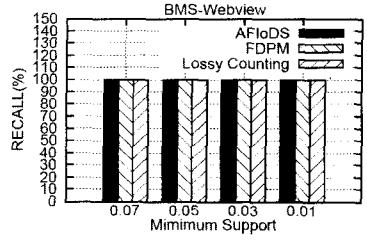

图 5　召回度比较

结束语　本文考虑了数据流上挖掘近似频繁项集的问题，提出了基于滑动窗口模型的 AFIoDS 算法，只保存真实频繁项集的真子集。并利用 chernoff bound 来保证真子集无限接近于真实频繁项集集合，节省了内存空间，实现了精确度和运行代价的折中。对 AFIoDS 算法滑动窗口的数据进行了分段，采用二元组的数据结构建立数据大纲，并用闭合项集的形式保存，然后通过小范围数据的挖掘和总体结果的合并，实现了数据在不断到达和离开情况下的增量更新。实验证明，AFIoDS 算法在保证精度没有下降的情况下降低了时间和空间代价，是可行、有效的。

参 考 文 献

[1] Charikar M, Chen K, Farach-Colton M. Fining frequent items in data stream[C]// International Colloquium on Automata, Languages, and Programining. 2002

[2] Misra J, Gries D. Finding repeated elements[J]. Science of Computer Programming, 1982, 2: 143-152
[3] 王伟平，李建中，张冬冬，等. 一种有效的挖掘数据流近似频繁项算法[J]. 软件学报，2007，8(4)
[4] Manku G S, Motwani R. Approximate frequency counts over data streams[C]// International Conference on Knowledge Discovery and Data Mining. 2006
[5] Cormode G, Muthukrishnan S. What's hot and what's not: Tracking most frequent items dynamically[C]// ACM Symposium on Principles of Database Systems. 2003
[6] Yu J X, Chong Z, Lu H, et al. False positive or false negative: Mining frequent itemsets from high speed transactional data streams[C]// International Conference of Very Large Databases. 2004
[7] Teng W, Chen M, Yu P S. A regression-based temporal pattern mining scheme for data streams[C]// International conference of Very large Databases. 2003
[8] Sun X, Orlowska M E, Li X. Finding frequent itemsets in high-speed data streams[C]// SIAM International Conference on Data Mining. 2006
[9] Chang J H, Lee W S. estWin: Adaptively monitoring the recent change of frequent itemsets over online data streams[C]// International Conference on Information and Knowledge Management. 2003
[10] Koh J, Shin S. An approximate approach for mining recently frequent itemsets from data streams[C]// International Conference on Data Warehousing and Knowledge Discovery. 2006
[11] Mozafari B, Thakkar H, Zaniolo C. Verifying and mining frequent patterns from large windows over data streams[C]// International Conference on Data Engineering. 2008
[12] Chang J H, Lee W S. Finding recent frequent itemsets adaptively over online data streams [C] // International Conference on Koowledge Discovery and Data Mining. 2003
[13] Giannella C, Han J, Pei J, et al. Mining frequent patterns in data streams at multiple time granularities[M]// Kargupta H, et al., eds. Next Generation Data Mining. AAAI/MIT, 2003
[14] Li H, Lee S, Shan M. An efficient algorithm for mining frequent itemsets over the entire history of data streams[C]// International Workshop on Frequent Itemset Ming Implementations. 2004
[15] 刘学军，徐宏炳，董逸生，等. 挖掘数据流中的频繁模式[J]. 计算机研究与发展，2005，12
[16] Jin R, Agrawal G. An algorithm for in-core frequent itemset mining on streaming data[C]// IEEE International Conference on Data Mining. 2005

(上接第 134 页)

假设阈值上限是 26，按照单点检测方法，采样点 2，5，6，7 均超过阈值，则通报次数为 4，自律管理器 AM 在 10 分钟内将收到 4 次来自同一台服务器的性能故障通报，并采取策略来修复故障，导致同样的策略执行 4 次。而第一个策略执行完毕后，可能服务器性能值已经恢复正常，因此其他策略执行属于无效操作，加大了系统开销。

表 1 ESX2.5 Server 的 CPU Usage%

个数	采样值
1	23.92613
2	41.1516
3	25.37086
4	22.78824
5	26.58733
6	27.66686
7	26.18363
8	23.53114
9	23.80006
10	23.17188

按照多数据点阈值检测方法，假设阈值上限相同，连续 3 个采样点越界才通报，则数据点 2 属于偶然越界值，不通报。只有在 5，6，7 这 3 个数据点判断完毕后才认为性能故障发生，向自律系统通报一次，策略执行一次。与单点判断相比，系统无效操作减少了 75%。

结束语 “监视，分析，计划，执行”是自律系统的 4 个主要特征。要进行自主决策，首先要能够自己发现问题，因此“自我监视”应该是自律计算中要研究的核心问题。本文主要针对自律计算中的性能监视故障，提出了一种多数据点的阈值检测和恢复方法，其目的是改善单点检测所存在的准确度不高、影响自律系统策略执行效率的问题。多数据点检测方法能够显著改善检测的准确度，从而提高系统的决策效率。本方法已经应用到了自律监视系统的设计中，所开发的产品也已经在国外投入了使用。实践证明，采用本方法的系统具有较高的决策效率。

参 考 文 献

[1] IBM Company. An architectural blueprint for autonomic computing [EB/OL]. Autonomic Computing White Paper. Third Edition. http://www.ibm.com/autonomic, June 2005
[2] Kephart J, Chess D. The Vision of Autonomic Computing [M]. IEEE Computer Society, 2003: 41-59
[3] Sterritt R, Curran E P, Song H. HACKER: Human And Computer Knowledge discovered Event Rules for Telecommunications Fault Management[C]// Proc. IEEE Int. Conf. Systems Man & Cybernetics. Oct. 2002
[4] Diaconescu A, Mos A, Murphy J. Automatic Performance Management in Component Based Software Systems[C]// Proceedings of the International Conference on Autonomic Computing (ICAC'04). 2004
[5] Wang Xiao-ying, Lan Dong-jun, Fang Xing, et al. A resource management framework for multi-tier service delivery in autonomic virtualized environments[C]// Network Operations and Management Symposium(NOMS 2008). IEEE, April 2008: 310-316
[6] Marin L. A Performance Analysis Method for Autonomic Computing Systems [J]. ACM Transactions on Autonomous and Adaptive Systems, 2007, 2(1)
[7] 廖备水，李石坚，姚远，等. 自主计算概念模型和实现方法[J]. 软件学报，2008，19(4)：779-802
[8] 曹敏，等. 基于自适应阈值的网络流量异常检测算法[J]. 计算机工程，2009，35(19)：164-166
[9] 刘文洁，李战怀. 一种基于自律计算的性能监视工具[J]. 微电子学与计算机，2008，25(3)：130-133