一种适应性的流式数据聚集计算方法

侯东风 刘青宝 张维明 邓 苏
(国防科学技术大学信息系统与管理学院 长沙 410073)

摘 要 针对流式数据聚集查询问题,提出了一种基于适应性层次聚集树的计算方法。适应性层次聚集树结构基于多层次时间窗口模型,将距离当前时刻较近的数据保存为细粒度数据,而相对久远的数据仅保留高层聚集信息;适应性层次聚集树中粒度的划分取决于相应时间间隔的数据密度。稀疏密度的时间间隔对应粗粒度的划分,而高密度的间隔对应细粒度的划分。并且提出了相应的构建维护以及聚集查询计算方法。实验结果表明,该方法在非均匀分布条件下的流式数据聚集计算中具有较为明显的优势。

关键词 流式数据,聚集计算,适应性层次聚集树,时间窗口
中图法分类号 TP311.13 **文献标识码** A

Adaptive Method of Computing Data Stream Aggregation

HOU Dong-feng LIU Qing-bao ZHANG Wei-ming DENG Su
(School of Information Systems & Management, National University of Defense Technology, Changsha 410073, China)

Abstract A method based on Adaptive Hierarchy Aggregation tree(AHA-Tree) was presented for computing aggregation of data stream. The structure of AHA-Tree borrowed the idea of multiple time granularities hierarchical window model, the recent data was kept in fine granularity and the older in rough. In addition, the partition of granularities was determined by density of time unit, the sparse time unit was kept in rough granularity and the denseness in fine. Moreover, the method of maintenance and aggregate computing was proposed for aggregation query. Experiment shows that the method is efficient in processing the data under non-uniform distribution.

Keywords Data stream, Aggregate computation, Adaptive hierarchy aggregation tree, Time window

在网络监控、通信数据管理、传感器网络数据管理及金融分析等应用中,流式数据分析技术已经成为研究者广泛关注的热点内容[1]。这些流式数据通常表现为快速变化、连续动态更新的无限数据序列[1,2],其巨大的数据量和固有的动态特征给数据处理带来巨大的挑战。在各领域的应用中,多数分析者更关心流式数据的高层趋势与异常模式,而不是单一个体元素。聚集计算则是解决这一问题的重要途径。

在多年的研究中,各研究机构的学者提出了多种流式数据聚集计算方法。López 对现有的计算方法进行了综合分析比较[2,3],将处理模型总结为滑动窗口模型和完全模型。滑动窗口模型仅关心最近一段时间内的数据,而完全模型则包含整个数据集合。Datar 提出了指数直方图方法[4],用于近似计算 COUNT 和 SUM 聚集值。Zhang 等人基于传统的时空聚集计算方法提出了 2SB-Tree 结构[5],用于计算多时间粒度的聚集值。Arasu 等从查询共享的角度研究了滑动窗口上的聚集查询问题[6]。文献[7]将整个滑动窗口划分为互不相交的格(pane),通过计算每个格的聚集值来支持多粒度的流式数据聚集计算。张冬冬等提出了基于 HDS-Tree 的历史数据聚集计算方法[8],通过多层递阶抽样方法处理大量的流式数据。Qin Shou-ke 则以分形为基础,倒转直方图作为概要存储结构,监控流式数据上多粒度时间窗口的近似聚集查询[9]。Jiao Yishan 提出了一种多度量滑动窗口模型[10],用于估计流式数据上的统计值。在最近的研究中,Nagaraj 提出了预先计算中间聚集值和合并过滤条件的计算方法[11]。此外,应用完全模型的典型方法包括基于 RHist 的计算方法[12]、基于小波的计算方法[13]以及基于草图的计算方法[14]等,这些方法通常忽略了数据的时间特征,主要应用在一些特定的背景中。

上述各种方法从不同的角度研究了流式数据聚集计算问题。在一些流式数据应用中,数据在时间维上的分布是非均匀的,具有一定的突发性,例如网络访问流式数据。本文针对这一特点,以流式数据的分布密度为依据,在多层次时间窗口模型[15]的基础上,提出了一种基于适应性层次聚集树(Adaptive Hierarchy Aggregation Tree, AHA-Tree)的聚集计算方法,其能够根据数据分布调整占用的存储空间,从而提高了空间利用效率,同时具有增量式更新的特征,因此提高了数据的维护效率和查询效率。

本文第1节分析了流式数据的特点和聚集计算中面临的主要难点;第2节详细阐述了适应性层次聚集树结构和计算

到稿日期:2009-04-10 返修日期:2009-06-15 本文受国家自然科学基金(70771110)资助。

侯东风(1978-),男,博士生,主要研究方向为流式数据管理与多维数据分析,E-mail:dongfeng.hou@gmail.com;刘青宝(1967-),男,博士,副教授,主要研究方向为数据仓库技术和数据挖掘;张维明(1962-),男,教授,博士生导师,主要研究方向为军事信息系统、信息综合处理与辅助决策;邓 苏(1963-),男,教授,博士生导师,主要研究方向为信息综合处理与辅助决策。

方法；第 3 节进行了性能分析和实验验证；最后总结了本文的工作，并指出进一步的研究方向。

1 基本概念

流式数据集 $S=\{a_1,\cdots,a_i,\cdots\}$，其中 $a_i=(v_i,t_i)$ 为顺序到达的数据元素，$v_i\in R$ 表示 a_i 的取值，t_i 表示 a_i 的时间属性。在本文中，令 $S[p,q]$ 表示 S 从时间点 p 到 q 的数据子集，若 $t_i\in[p,q]$，则 $a_i\in S[p,q]$。

由定义可知，流式数据具有动态变化、无限增长等特点，并且数据通常处在较低的抽象层次。而用户感兴趣的信息通常处在不同的粒度层次，例如观察不同时间粒度上的发展趋势。流式数据聚集计算通过在可快速访问的主存中维持预先计算的概要存储结构，以支持用户提出的各种聚集查询和分析请求。

流式数据的固有特点使得预先聚集计算面临很多困难，主要包括①存储空间约束：流式数据在时间跨度上是无限的，并且在计算中需保存大量的聚集信息，而可用的主存空间却是有限的，因此无法存储全部数据和聚集信息。②更新维护复杂性约束：流式数据随着时间的推移而不断动态变化，相应的聚集信息也必然随之更新，从而对概要存储结构的更新维护效率提出了更高要求。③查询复杂性约束：流式数据聚集查询由用户即席提出，且要求快速响应，因此如何利用概要存储结构获得查询结果成为了关注的焦点。

本文针对上述的约束提出了一种适应性的聚集计算方法。第一，通过多层次时间窗口[15]进行约束，保存最近一段时间内的数据信息。多层次时间窗口表示为 $\xi=\{(W_i,\omega_i,g_i)\mid i=1,\cdots,m\}$，其中 W_i 为第 i 层时间窗口，对应的时间粒度为 g_i，时间窗口宽度为 ω_i，m 为时间窗口的层次数。在任意时刻 t，对应的第 i 层的数据落在时间窗口 $[t-\omega_i+1,t]$ 中，对应的流式数据子集为 $S[t-\omega_i+1,t]$。若时间间隔 I 落在时间窗口中，则记为 $I\subset\xi$。第二，充分考虑流式数据的突发性特征，以单元时间内的数据密度为依据，建立适应性的存储结构，以提高空间存储效率。第三，概要数据存储在可快速访问的主存空间中，通过一次扫描动态更新维护存储结构，并且支持增量式更新，减少数据处理的时间复杂性。第四，通过支持渐进式查询，提供时间效率和精确性的折中。

2 适应性层次聚集树结构与计算方法

2.1 数据结构

定义 1(适应性层次聚集树，AHA_Tree) 适应性层次聚集树包含两个部分：一是子窗口索引结构 SWI，另外一部分是子窗口内部的聚集树结构。描述如下：

①索引结构 $SWI=\{SWT_1,\cdots,SWT_p\}$ 包含 p 个子窗口，SWT_i 是第 i 个子窗口对应的聚集树结构。根节点 $Root$ 为索引结构顶层的节点，对应整个时间窗口 ξ，V_{Root} 表示 ξ 的聚集值集合。

②对于任意的 SWT_i，所包含的树节点 $Node_k$ 表示为五元组 $(I_k,g_k,\widetilde{Node_k},V_k,count(k))$，其中 I_k 为 $Node_k$ 对应的时间间隔，g_k 为对应的时间粒度，$\widetilde{Node_k}$ 为子节点构成的划分集合，V_k 表示聚集值集合，$count(k)$ 为节点所包含的数据元素计数值。若 $\widetilde{Node_k}\neq\phi$，则 $Node_k$ 为聚集节点，拥有 α 个子节点，V_k 与 $count(k)$ 通过其子节点计算获得。若 $\widetilde{Node_i}=\phi$，则 $Node_k$ 为叶节点，对应的时间间隔不可再划分，V_k 与 $count(k)$ 通过所包含的数据元素计算获得。若 $\widetilde{Node_k}=\phi$ 且 $count(k)=0$，则 $Node_k$ 为虚节点，不包含任何数据元素，V_k 为空。

叶节点所包含的数据元素保存为列表，定义如下。

定义 2(数据元素列表) 若 $Node_i$ 为叶节点，包含的数据元素列表定义为 $Node_i.List$，对应的时间间隔为 I_i，对应的流式数据子集为 $S[I_i]$，即节点 $Node_i$ 涵盖的数据元素集合。$Node_i.List$ 中的数据元素是依据时间顺序排列的。所有叶节点的数据列表中数据元素的并集即为时间窗口中所有原始数据构成的集合。

由定义可知，AHA_Tree 是一种基于时间间隔的层次分解树结构，层次划分的主要依据是时间间隔上的数据密度分布。高密度的间隔对应精细的粒度划分，而稀疏的间隔则对应相对粗糙的粒度划分。若某个节点所涵盖的数据元素列表长度超出了阈值 $\varepsilon\omega$，则需进一步划分为更精细的时间间隔，直至基本时间粒度，其中 ω 为时间窗口长度，ε 为用户定义参数。

AHA_Tree 存在以下几个性质：

①对于任意两个不同的节点 $Node_i$ 和 $Node_j$，存在 $I_i\neq I_j$，即任何两个不同节点对应的时间间隔不同。

②若 $g_i=g_j$，则 $Node_i$ 和 $Node_j$ 属于同一层次的时间窗口。

③若 $Node_j$ 为 $Node_i$ 的子节点，则 $Node_j\in\widetilde{Node_i}$。

④若 g_i 不为基本时间粒度，则 $count(i)<\varepsilon\omega$。

假设输入的流式数据实例 S 为{(12,2),(10,4),(23,5),(10,5),(5,6),(8,6),(22,8),(24,8),(10,13),(11,15),(22,18),(4,25),(6,27),(3,27),(6,28),(23,30),(5,32)…}，令时间粒度的层次数 $m=3$，基本时间粒度为 1s，每一层次按照倍数 $\alpha=4$ 递增。建立 AHA_Tree 结构如图 1 所示。其中 $Root$ 为根节点，子节点为子聚集树的根节点。AHA_Tree 中每个节点均对应于时间间隔的划分，令 $SWT_i\in SWI$，对于 $\forall Node_k\in SWT_i$，总存在 $I_k\subset\xi$ 与之对应，如子聚集树 SWT_1 的根节点 N_1 对应时间间隔 I_1，N_4 则对应时间间隔 I_4。时间间隔依据划分的覆盖关系构成了层次结构。若 $Node_k\in\widetilde{Node_j}$，则有 $I_k\subset I_j$，且 I_k 的长度为 I_j 的 $1/\alpha$，如节点 N_6 为 N_4 的子节点。不包含任何数据元素的节点为虚节点，如图中标记阴影的节点。节点 N_5 涵盖了两个数据点，则形成一个长度为 2 的数据列表，存储数据的时间戳和取值。

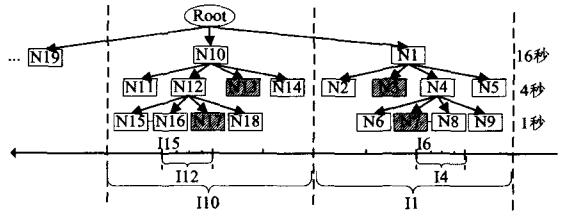


图 1 适应性层次聚集树结构

2.2 构建与维护方法

基于上述定义，随着流式数据元素的不断到达，新到达数据元素 e_{new} 将会影响 AHA_Tree 结构。首先将 $e_{new}.t$ 所属的时间间隔加入到相应的节点 $Node_k$ 下，并且更新从根节点到

$Node_k$ 路径上的所有聚集值。若 $count(k)>\varepsilon\omega$，则需进一步划分该节点。另外，基于多层次时间窗口模型思想，随着数据量的增加，一些相对久远的历史数据不再保留详细信息，而仅保留高层聚集信息，以减少数据占用的存储空间，因此AHA_Tree树结构随着时间的推移会删除部分陈旧的节点数据。在本节中主要阐述这两种条件下的更新维护操作。

以图1的AHA_Tree结构为例，在初始状态，仅包含一个根节点 $Root$。当接收到一个数据元素时，首先建立第一个子树结构 SWT_1，并依据顶层时间粒度增加节点 N_1。将到达的数据元素加入到节点 N_1 下，形成数据列表，并且更新节点与祖先节点的聚集值，如图2(a)所示。图2(a)中 $count(N_1)=4$，令 $\varepsilon\omega=4$，$\alpha=4$，根据规则，若增加一个节点，则需将节点 N_1 拆分为 α 个子节点，N_1 转化为聚集节点。将数据单元列表中的数据分别加到划分的节点下，形成新的数据单元列表，并且更新创建节点的聚集值，图2(b)所示。没有覆盖任何数据单元的节点为虚节点，如节点 N_2 和 N_3。若经过划分之后仍然有节点超出阈值，则继续进行划分，直至划分至最细粒度。图2(b)中 $count(N_4)=4$，新到达的元素需进一步划分节点 N_4，如图2(c)所示。后面的元素同此操作，若超出 N_1 所涵盖的时间间隔，将新建子聚集树 SWT_2。

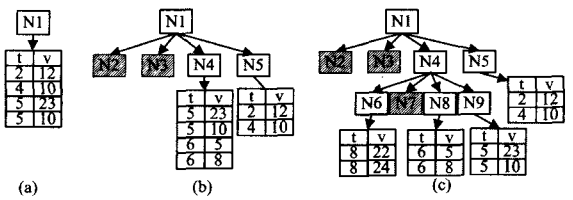


图2 适应性层次聚集树构建过程

算法描述如下：

算法1 构建AHA_Tree树结构

输入：新加入数据元素 e_{new}，原始聚集树结构 AHA

输出：更新后的聚集树结构 AHA_{new}

```
BEGIN
  /* AHA 的索引结构为 SWI，根节点为 Root，εω 为划分阈值 */
  /* 在 SWI 确定 e_new 所对应的子聚集树结构，若不存在则创建新的聚集树结构 */
  FIND(SWT_i, SWI);
  /* 在 SWT_i 中定位 e_new 对应的叶节点 Node_k，若为空树，则建立子树根节点 */
  LOCATE (Node_k);
  /* 将 e_new 插入到 Node_k.List 中，更新 count(Node_k)。并更新从 Node_k 到 Root 路径上所有节点的聚集值集合 */
  INSERT(e_new, Node_k.List);
  count(Node_k)=count(Node_k)+1;
  UPDATE(Root→Node_k);
  /* 若涵盖的数据元素超出阈值且未达到基本时间粒度，则执行划分操作 */
  IF(count(Node_k)>εω)
    /* 函数 SPLIT 递归执行划分操作，直至 count 值小于 εω 或者到达基本时间粒度 */
      SPLIT(Node_k);
  RETURN AHA_new;
END
```

随着时间的推移，数据总量不断增加，主存中无法保存全部数据。因此当数据量达到一定程度后，不再保留相对陈旧的数据的详细信息。首先从最细的粒度开始，若节点超出了对应时间窗口的范围，则删除节点及其数据元素列表，依照时间窗口的顺序逐层维护。最后根据时间窗口的范围，删除子树的根节点，这一单元的数据完全转化为静态历史存储。为了避免频繁操作，删除操作按照一个时间单元的间隔进行。以图1中的AHA_Tree结构为例，以16s为周期进行删除。若时间粒度为1s的层次超出了时间范围，则沿着树结构从右至左的顺序搜索该层次的节点。图1中对应的节点为 N_6，N_7，N_8，N_9，删除这些节点及其数据列表。其他的层次亦是如此操作。

算法描述如下：

算法2 维护AHA_Tree树结构过期数据

输入：维护的时间粒度层次 $Level(i)$，原始聚集树结构 AHA

输出：更新后的聚集树结构 AHA_{new}

```
BEGIN
  /* AHA 的索引结构为 SWI，根节点为 Root，w(i)为对应窗口的长度，g(i)为对应窗口的时间粒度，当前时间为 Now */
  /* 依据 w(i)和 Now 在 SWI 中定位需要维护的子聚集树结构 */
  FIND(SWT_i, SWI);
  /* 遍历 SWT_i，搜索粒度为 g(i)的子节点并删除 */
  SEARCH (SWT_i, g(i));
  RETURN AHA_new;
END
/* 函数 SEARCH 遍历子聚集树结构 */
FUNCTION SEARCH (SWT_i, g(i))
BEGIN
  /* 定位 SWT_i 子树中时间粒度为 g(i)的子节点 Node_j */
  LOCATE(Node_j, SWT_i, g(i));
  DELETE(Node_j)
/* 若为顶层粒度，则更新根节点的聚集值 */
  IF(g(i)为顶层时间粒度)
    UPDATE(Root)
END
```

2.3 基于适应性层次聚集树的查询方法

在实际应用中，流式数据上多时间粒度的聚集查询是非常普遍的，如发现某个时间段内流式数据不同粒度层次的发展趋势或者异常点。然而在多数情况下并不需要精确的查询结果，而是快速的响应速度，因此流式数据上的聚集查询是渐进式(Progressive)的。首先在高层粒度的节点上搜索，若需查询更精确的结果，则进一步搜索下层节点，类似于OLAP中的下钻操作。

流式数据上的聚集查询定义为 $Q=(S,f_A,I,g(i),P)$，其中 S 为查询的流式数据集合，f_A 为对应的聚集函数，$I=[p,q]\in\xi$ 为查询的时间范围，$g(i)$ 为查询的最低时间粒度，P 为约束条件集合。文献[8]将聚集查询分为简单聚集查询和复杂聚集查询。若 P 为空，即为简单聚集查询；否则，为复杂聚集查询。在本文中 f_A 通常定义为分布式与代数聚集函数[16]。

从查询的形式上还可分为快照查询和连续查询。若 I 为时间窗口内部的固定间隔，则为快照查询，搜索当前时间窗口内的数据集，返回满足条件的查询结果。而在连续查询中，I 对应一个半开的时间间隔，从注册查询时间开始直到查询终止，针对动态更新的流式数据集合，随着时间推移不断返回满

足条件的动态聚集结果。

快照查询算法描述如下：

算法 3 快照查询

输入：聚集查询 $Q=(S,f_A,I,g(i),P)$，聚集树结构 AHA

输出：查询结果 U

BEGIN

/* 在 AHA 的索引结构 SWI 中确定落在 I 中的子聚集树集合 N */

LOCATE(N,I,SWI)；

/* 遍历 N 中的子聚集树 $SWT_{k_1},\cdots,SWT_{k_l}$，计算部分查询结果 $U_{k_1},\cdots,U_{k_l}$ */

FOR $j=k_1,\cdots,k_l$

U_j=QUERY($SWT_j,I,g(i)$)；

/* 根据 $U_{k_1},\cdots,U_{k_p}$ 计算查询结果 U，其中⊕为合并计算符 */

$U=U_{k_1}\oplus U_{k_2}\oplus\cdots\oplus U_{k_p}$；

END

/* 函数 QUERY 遍历聚集树结构 SWT_j，返回 U_j */

FUNCTION QUERY ($SWT_j,I,g(i)$)

BEGIN

/* $Node_{j0}$ 为 SWT_j 的根节点，判断 $Node_{j0}$ 被 I 所覆盖的部分，若为部分覆盖则需下钻计算至粒度 $g(i)$，若为全部覆盖则仅需获取预先计算的聚集值 */

IF($Node_{j0}.I\subset I$)

GETAGG($V_{Node_{j0}}$)；

ELSE IF($Node_{j0}.I\subset_p I$)

递归查询被 I 所覆盖的子节点，直至时间粒度 $g(i)$；叶节点则读取叶节点的数据列表进行计算。

RETURN U_j；

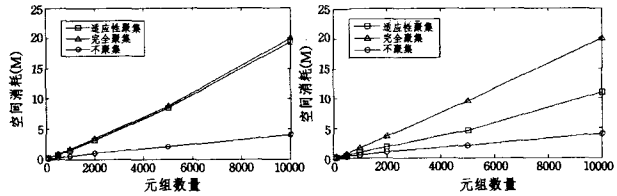
END

连续查询属于累积型操作。对于不同的时间点，查询得到的结果是不同的。令 $U(t)$ 表示在时间点 t 的快照聚集查询结果，则连续查询结果表示为 $U=\bigcup_t(U(t)-U(t-1))\oplus U(0)$。从查询注册开始，首先计算当前时刻的时间窗口中满足条件的快照查询结果，然后随着时间的推移，依据查询条件搜索新到达的数据子集，得到增量的连续查询结果，最终将连续的查询结果返回给用户。

3 性能分析

为了分析 AHA_Tree 的性能，本文针对顺序存储、完全预先计算和适应性计算等 3 种不同计算策略进行比较。顺序存储将流式数据按照时间顺序存储为数据列表(不聚集)，完全预先计算策略[15]计算所有时间粒度上的聚集信息并将其保存在相应的数据结构中，而适应性聚集方法(AHA-Tree)则根据数据的分布舍弃部分时间粒度的存储。数据主要采用生成的人工数据，分为均匀分布的数据和非均匀分布的数据两种。在实验中主要验证 3 种策略在存储空间、计算效率和查询效率上的差别。

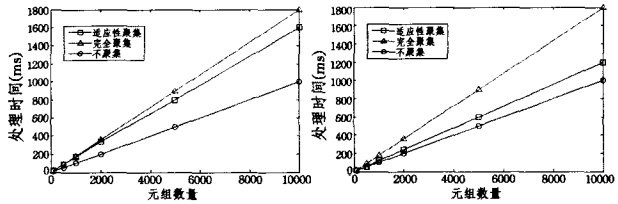
如图 3(a)所示，在均匀的数据分布条件下，AHA-Tree 占用的存储空间接近于完全聚集。而图 3(b)所示，在非均匀分布的条件下所占用的存储空间却大大减少，因为在构建过程中它能够根据数据在时间上的分布密度调整树结构。在很多情况下，流式数据中稀疏的数据往往会占用大量的存储空间，采用适应性的调整策略能够有效利用存储空间，进一步压缩整个概要存储结构的存储空间。



(a)均匀数据分布条件下的存储空间消耗　(b)非均匀数据分布条件下的存储空间消耗

图 3

概要存储结构的构建和更新效率是另外一个非常重要的因素。如图 4(a)所示，在均匀分布条件下，AHA-Tree 的性能接近于完全聚集，而图 4(b)则显示在非均匀数据分布条件下存储结构的构建效率有所提高。AHA_Tree 树结构中根节点索引的是顺序的时间单元，便于增量式更新和维护。同时在构建树的过程中，它根据数据密度适应性调整数据存储粒度，舍弃了部分稀疏的节点，从而在空间上提高了效率，同时不必维护过多的中间聚集节点，因此在更新的时间上也提高了效率。



(a)均匀数据分布条件下的处理时间　(b)非均匀数据分布条件下的处理时间

图 4

如图 5 所示，根据随机生成的不同长度的范围查询，3 种策略的表现是大不相同的。顺序存储虽然节省了存储空间和处理时间，但是在处理范围查询上却有较高的时间复杂性。AHA-Tree 与完全聚集两种预先计算策略的性能是非常接近的，在 AHA-Tree 树结构中维持了预先计算的多时间粒度聚集结果，并且依据数据密度适应性调整存储粒度，在稀疏部分数据中也能够快速获得计算结果，从整体上提高了查询效率。

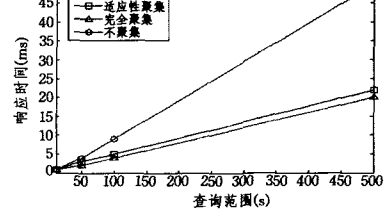


图 5　不同长度的范围查询响应时间

实验表明，AHA-Tree 在非均匀分布条件下的流式数据聚集计算中具有较为明显的优势。

结束语　流式数据聚集计算是进行流式数据分析的重要基础。如何寻求存储空间、处理时间和查询响应时间的折中，是需要考虑的问题。本文提出的基于适应性层次聚集树的计算方法充分考虑了流式数据的突发性特征，根据流式数据在时间维度上的分布密度适应性调整聚集树的存储结构，无论是从存储空间上还是处理时间上均提高了计算效率。

在以后的研究工作中，将进一步拓展适应性存储结构的研究，考虑查询负载等因素影响下的结构调整问题和多维聚集查询的计算问题。

参 考 文 献

[1] Muthukrishnan S. Data Streams: Algorithms and Applications [M]. Hanover, MA, USA: Now Publisher Inc, 2005

(下转第 169 页)

[16] Salamah S, Gates A Q, Roach S. Improving Pattern-Based LTL Formulas for Automata Model Checking[C]// Proceedings of the 5th International Conference on Information Technology: New Generations. 2008:9-14

[17] Flake S. Temporal OCL Extensions for Specification of Real-Time Constraints[C]// Workshop Specification and Validation of UML Models for Real Time and Embedded System (SVERTS'03). 2003

[18] Rammig F J. OCL Goes Real-time[C]// Proceedings of the 5th IEEE International Symposium on Object-oriented Real-time Distributed Computing (ISORC'02). 2002

[19] 余金山. 实时 UML 与 Rational Rose RealTime 建模案例剖析[M]. 北京:电子工业出版社,2007

[20] 彭中,毛晓光. 基于 OCL 的面向方面监控框架[J]. 计算机工程,2009

[21] Database Systems Group Bremen University. USE A UML based Specification Environment[EB/OL]. http://www.db.informatik.uni-bremen.de/projects/USE/,2009-05

[22] Konermann A. The Parser Subsystem of the Dresden OCL 2 Toolkit Design and Implementation[M]. Dresden, Germany: Technische Universitat Dresden,2005

[23] Briand L C, Dzidek W, Labiche Y. Using Aspect-oriented Programming to Instrument OCL Contracts in Java[M]. Ottawa, Canada: Carleton University,2004

[24] Ochoa O. Towards a Tool for Generating Aspects from MEDL and PEDL Specifications for Runtime Verification[J]. Lecture notes in Computer Science,2007,4839:75-86

[25] Chen F, D'Amorim M, Rosu G. A Formal Monitoring-based Framework for Software Development and Analysis[C]// Proceedings of the 6th International Conference on Formal Engineering Methods (ICFEM'04). 2004

[26] Sadjadi S, McKinley P K, Cheng B H C, et al. TRAP/J: Transparent generation of adaptable Java programs[C]// Proceedings of the International Symposium on Distributed Objects and Applications (DOA'04). Agia Napa, Cyprus,2004

[27] Stolz V, Bodden E. Temporal Assertions using AspectJ[C]// 5th Workshop on Runtime Verification (RV'05). 2005

[28] Karaorman M, Freeman J. jMonitor: Java Runtime Event Specification and Monitoring Library[J]. Electronic Notes in Theoretical Computer Science,2005,113(3):181-200

[29] Richters M, Gogolla M. Aspect-Oriented Monitoring of UML and OCL Constraints[C]// AOSD Modeling with UML Workshop,6th International Conference on the United Modeling Language (UML). San Francisco, USA,2003

(上接第 155 页)

[2] Babcock B, Babu S, Datar M, et al. Models and issues in data stream systems[C]// Proceedings of the 21st ACM Symposium on Principles of Database Systems. Madison, Wisconsin, USA, 2002:1-16

[3] López I F V, Snodgrass R T, Moon B. Spatiotemporal Aggregate Computation: A Survey[J]. IEEE Transactions on Knowledge and Data Engineering,2005,17(2):271-286

[4] Datar M, Gionis A, Indyk P, et al. Maintaining Stream Statistics over Sliding Windows[J]. SIAM J. COMPUT, 2002, 31(6): 1794-1813

[5] Zhang D, Gunopulos D, Tsotras V J, et al. Temporal and spatio-temporal aggregations over data streams using multiple time granularities[J]. Information Systems,2003,28:21-84

[6] Arasu A, Widom J. Resource Sharing in Continuous Sliding-window Aggregates[C]// Proceedings of the 30th International Conference on Very Large Data Bases. Toronto, Canada, 2004: 336-347

[7] Li J, Maier D, Tufte K, et al. No Pane, No Gain: Efficient Evaluation of Sliding-Window Aggregates over Data Streams[J]. SIGMOD Record,2005,34(1):39-44

[8] 张冬冬,李建中,王伟平,等. 数据流历史数据的存储与聚集查询处理算法[J]. 软件学报,2005,16(12):2089-2098

[9] Qin S, Qian W, Zhou A. Approximately Processing Multi-granularity Aggregate Queries over Data Streams[C]// Proceedings of the 22nd International Conference on Data Engineering. Atlanta, GA, USA: IEEE Computer Society,2006

[10] JIAO Y. Maintaining Stream Statistics over Multiscale Sliding Windows[J]. ACM Transactions on Database Systems,2006,31(4):1305-1334

[11] Nagaraj K, Naidu K V M, Rastogi R, et al. Efficient Aggregate Computation over Data Streams[C]// Proceedings of the 24th International Conference on Data Engineering. Cancún México, 2008:1382-1384

[12] Qiao L, Agrawal D, Abbadi A E. RHist: Adaptive Summarization over Continuous Data Streams[C]// Proceedings of the 11th ACM International Conference on Information and Knowledge Management. McLean, Virginia, USA,2002

[13] Fan Xiao-bo, X T-T, Li Cui-ping, et al. MRST--An Efficient Monitoring Technology of Summarization on Stream Data[J]. Journal of Computer Science & Technology,2007,22(2):190-196

[14] Rusu F, Dobra A. Fast Range-summable Random Variables for Efficient Aggregate Estimation[C]// Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data. Chicago, Illinois, USA,2006

[15] 刘青宝,金燕,侯东风,等. 数据流层次窗口模型及聚集查询算法[J]. 计算机科学,2007,34(5):194-196

[16] Gray J, Bosworth A, Layman A, et al. Data cube: a relational aggregation operator generalizing group-by, cross-tab, and sub-totals[C]// Proceedings of the 12th International Conference on Data Engineering. Los Alamitos, CA,1996:152-159