doi:10.3772/j.issn.1002-0470.2014.09.003

基于滑动窗口的数据流字符串近似查询①

崔 甲②*,**,*** 王伟平③*,** 陈重韬*,**,*** 孟 丹***

(*中国科学院计算技术研究所计算机应用研究中心 北京 100190)

(**中国科学院大学 北京 100049)

(***中国科学院信息工程研究所 北京 100093)

摘 要 针对数据访问模式随着网络技术的发展逐渐从静态磁盘转移到动态数据流的状况,研究了数据流上的字符串近似查询。为了解决数据流的连续性、无边界性、不可预知性和在线计算资源的局限性导致基于静态数据集的近似查询处理方法无法高效地支持数据流的问题,提出了基于滑动窗口数据流的字符串近似查询(AS^3)方法。该方法基于过滤-验证框架和基本窗口索引更新机制,改进并应用非对称特征策略提取数据流和查询关键字的特征,采用了两个新的过滤算法——预剪裁过滤(PPF)算法和流统计(CFS)过滤算法,以及基于矩阵坐标的验证(CV)算法。实验结果表明,AS^3 方法能够高效地支持基于滑动窗口的数据流字符串近似查询,在保证结果准确率的同时具有较高的实时性及峰值处理能力。

关键词 数据流,字符串近似查询,滑动窗口,编辑距离

0 引 言

字符串近似查询是数据库和信息检索领域的一项重要技术,有着广阔的应用前景。典型的应用有数据库的清洗与集成[1]、计算生物学中蛋白质和基因序列的近似匹配[2]、拼写的自动纠正[3]等。目前字符串近似查询领域已经有了很多研究成果,但是大部分是基于静态数据集的。随着网络技术的发展,数据访问模式逐渐从静态磁盘转移到动态数据流,而已有的基于数据流的关键字查询(Stream-based Key Word Search, S-KWS)方法只支持字符串的精确匹配[4,5]。因此,将字符串近似查询技术扩展到基于数据流的查询则成为了一种需求广泛的新型查询处理模式,尤其在信息安全领域,比如垃圾邮件及短信息内容的实时监控和过滤。

当前基于数据流的字符串近似查询的研究工作还很少,而基于静态数据集的近似查询技术无法高效地支持数据流,原因在于静态数据集与数据流之间有本质的区别:静态数据集存储于磁盘,大小有边界,I/O 速率稳定,批量更新;数据流不落地磁盘,连续且无边界,速率波动,实时更新。这给研究带来了极大的挑战:(1)无边界性使得计算机不可能在有限的内存中缓冲全部数据流,缓冲区大小受限于内存;(2)实时性使得数据流具备时效性,暂存于缓冲区的数据流会因过期被移除,因此需要实时更新缓冲区和及时返回查询结果,同时具备较高的峰值处理能力。还有,基于静态数据集的查询是即席查询,即查询是变化的,而数据对象相对固定;基于数据流的查询是连续查询,即查询相对固定,而数据对象是变化的,两者模式明显不同。针对以上挑战,本文提出了基于滑动窗口的数据流字符串近似查询(sliding-window based Approximate String Search on data-Stream,AS^3)方法,简称 AS^3 方法。它基于过滤-验证框架,以编辑距离作为近似度量,采用基本窗口更新机制,改进并应用非对称特征策略提取数据流和查询关键字的特征,采用了两个新的过滤算法和新的验证算法。实验结果表明,AS^3 方法能够高效地支持数据流字符串的近似查询。

① 863 计划(2013AA13204,2012AA01A401),国家自然科学基金(60903047),国家核高基(2013ZX01039-002-001-001)和中国科学院先导专项(DA06030200)资助项目。

② 男,1981 年生,博士生;研究方向:数据流,字符串相似度查询;联系人,E-mail:cuijia@ncic.ac.cn

③ 通讯作者,E-mail:wangweiping@iie.ac.cn

(收稿日期:2014-03-31)

1 相关研究

字符串近似查询分为近似搜索和近似连接,近似搜索是指在数据集上查找与查询关键字近似的字符串,近似连接是指从两个数据集中查询出近似的字符串对。编辑距离[6]是常用的字符串近似度量,它是指将字符串 r 通过三种单字符操作(插入、删除和替换)转换为字符串 s 所需要的最少操作次数,记为 $ed(r,s)$。计算编辑距离的经典算法是动态规划[6],时间复杂度为 $O(|r|\times|s|)$。本文基于编辑距离,研究数据流字符串的近似搜索问题。

已有的字符串近似查询处理方法可分为:

(1)过滤-验证框架[7]。框架分为索引、过滤和验证三个阶段:索引阶段提取字符串特征并创建索引;过滤阶段根据过滤算法和相似度阈值计算特征的重合下限,查询索引并过滤不可能相似的字符串,得到候选集;验证阶段计算候选集与查询关键字的真实相似度,由于特征重合下限是字符串满足相似的必要条件,需要加以验证来消除误报。其中,特征提取策略和过滤算法是影响框架查询性能的关键。已有特征提取策略包括:1)定长特征:q-gram[7]和 q-chunk;2)变长特征:VGRAM[8]和 VCHUNK[9];3)基于分片的特征[10];4)非对称特征[11]。已有的过滤算法包括长度过滤[12]、统计过滤[12]、位置过滤[12]、前缀过滤[1,13]、基于位置不匹配的过滤[7]、基于内容不匹配的过滤[7]等。过滤算法强度直接影响候选集大小以及验证的时间开销。

(2)基于树形索引的方法。文献[14]提出了基于 Trie 的字符串近似连接方法 TrieJoin。由于 Trie 难以支持实时的插入和删除更新,因此不适合数据流环境。

下面是本文关于数据流的定义。

定义 1 数据流 S 由连续不断传输的元组构成,$S=\{<s_1,ts_1>,\cdots,<s_i,ts_i>,\cdots\}$。其中,$s_i$ 是元组包含的字符串内容,ts_i 是元组的时间戳(可在数据源发送时生成或在元组到达时生成)。

定义 2 滑动窗口是为数据流维护的缓冲区窗口,新元组不断填入,过期元组被移除,在时间轴上看该窗口在数据流上是滑动的。滑动窗口在某一时刻的状态称为滑动窗口快照。

滑动窗口模型提供了数据流连续查询的语义,当滑动窗口发生更新时,则触发对当前滑动窗口快照的查询。滑动窗口一般分为两类:基于时间的滑动窗口或基于计数的滑动窗口。

2 AS^3 方法描述

在数据流上进行字符串近似查询的直接方法是:每接收到一个新元组,直接计算它与查询关键字的编辑距离。显然该方法的瓶颈是动态规划算法的高时间复杂度,当数据流到达速率高于动态规划算法处理的平均速率时便会失效。

2.1 滑动窗口模型

当数据流到达速率较高时,滑动窗口的更新非常频繁,为进行有效的近似查询,我们需要对滑动窗口数据创建索引。若为滑动窗口创建一个独立的索引,更新(插入和删除)维护将非常复杂,而且频繁更新导致查询的触发频率很高。为支持滑动窗口和索引的实时更新,以及降低滑动窗口更新频率,本文引入基本窗口[15]。

定义 3 基本窗口是滑动窗口的非重叠邻接子窗口,每个基本窗口维护相同的时间间隔或相同数量的元组。

定义 3 将滑动窗口定义为一个基本窗口队列,即 $Sw=\cup Bw_i$,Bw_i 是滑动窗口 Sw 的第 i 个基本窗口;同样滑动窗口索引也由一个基本窗口索引的队列构成,AS^3 需要同时维护一个基本窗口队列和一个基本窗口索引队列。当数据流的新元组到达时,若当前基本窗口未满,则将新元组填入基本窗口;若当前基本窗口已满,则创建基本窗口索引,再将基本窗口及索引更新到相应队列中。对于基于时间的滑动窗口而言,基本窗口也具有时间戳,取值为其包含元组的最大时间戳。图 1 是 AS^3 采用的滑动窗口及索引模型。

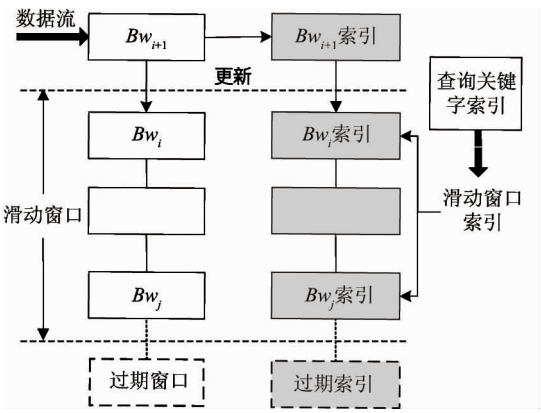

图 1 AS^3 的滑动窗口及索引模型

文献[15]引入基本窗口是为在数据流上支持聚集操作(如 sum、count),而本文引入基本窗口是

为支持滑动窗口及索引的动态更新。在 AS^3 中基本窗口是滑动窗口更新的基本单位,基本窗口大小与滑动窗口的更新频率及查询结果相关:若基本窗口偏大,则导致滑动窗口的更新粒度偏大,可能造成基本窗口被移除时仍包含未过期元组,影响查询准确率;若基本窗口偏小,则导致滑动窗口更新仍然很频繁,未起到分片的作用。

基本窗口大小的确定。假定数据流的最大峰值为 M tuple/s,滑动窗口大小为 N,包含 n 个基本窗口,创建一个基本窗口索引的时间为 t_i,查询一个基本窗口的时间开销为 t_q,为实现数据流上的非阻塞连续查询,基本窗口和滑动窗口应满足以下条件:

$$N/(n \times M) \geqslant t_i + t_q \tag{1}$$

定义 4　在数据流 S 上注册的连续查询 Q 是一个四元组:$Q = \{K, \tau, q, T\}$ 或 $Q = \{K, \tau, q, N\}$,其中 K 是查询关键字集 $\{k_1, k_2, \cdots, k_n\}$,$\tau$ 是编辑距离阈值,q 是提取的特征长度,T 是滑动窗口的时间跨度(N 是滑动窗口包含的元组数)。当基本窗口在时刻 t_k 更新滑动窗口,则触发对当前滑动窗口快照的查询。

对于基于时间的滑动窗口,则返回结果为

$$R = \{s \in Sw[t_k - T, t_k] = \cup Bw_i \mid Bw_i.ts_i \in [t_k - T, t_k], \forall ed(s, k_j) \leqslant \tau\} \tag{2}$$

对于基于计数的滑动窗口,则返回结果为

$$R = \{s \in Sw[N]t_k = \cup Bw_i \mid \forall ed(s, k_j) \leqslant \tau\} \tag{3}$$

图 2 是以基于计数的滑动窗口为例,对 AS^3 查询过程的伪代码描述。

算法 1　AS^3 查询过程

输入:数据流 S,查询 $Q = \{K, \tau, q, N\}$
输出:结果集 R
① I_K = CreateQueryIndex(K, q)
② While(a new tuple $s \in S$ arrives){
③ **if**(current Bw_i is not full)　insert s to Bw_i;
④ **else**{ Create a new Bw_{i+1} and insert s;
⑤ I_{Bwi} = CreateBwIndex (Bw_i, q);
⑥ append B_{wi} and I_{Bwi}
⑦ remove _ expired _ bw(Sw, N);
⑧ trigger procedure QueryThread} }

Procedure: QueryThread

① Obtain the snapshot of sliding window index I_{Sw};
② Filtering step for candidates C = *filtering*(I_{Sw}, I_K);
③ Verify step for result R = *verify*(C, K, τ);

图 2　AS^3 的查询过程

2.2　特征提取策略

字符串特征提取策略直接影响滑动窗口特征索引创建及更新的速度。为保证基本窗口索引在创建完成时不过期,其创建的时间开销应小于滑动窗口全部更新一次的周期。本文改进并应用了非对称字符串特征[11]提取策略来分别提取数据流和查询关键字的特征。具体描述如下:

提取查询关键字的 q-gram 特征。首先在关键字 r 的尾部添加 $(q-1)$ 个特殊字符‘ $ ’,得到 $r' = r + |q-1|$ ‘ $ ’,提取 r' 的 q-gram 特征,这样确保 r 中的每个字符都对应一个 q-gram,得到的特征集合记为 $G_q(r)$。(算法 1 第①行)。

提取数据流的 q-chunk * 特征。文献[11]提出的 q-chunk 提取方法是:首先在字符串 s 的尾部添加 $|q-(|s| \bmod q)|$ 个特殊字符‘ $ ’得到 s',再提取 s' 的 q-chunk 特征。若字符串长度能被 q 整除,即 $|s| \bmod q = 0$,将在字符串的尾部添加 q 个‘ $ ’,因此提取的特征中存在一个为 q 个‘ $ ’的特征,该特征不可能与查询关键字的 q-gram(最多包含 $|q-1|$ ‘ $ ’)匹配,属于冗余特征。本文对它稍加改进(记为 q-chunk *):在字符串 s 尾部添加 ξ 个特殊字符‘ $ ’,再提取 q-chunk 特征,特征集合记为 $C_q(s)$(算法 1 第⑤行)。ξ 用下式表示:

$$\xi = \begin{cases} q - |s| \bmod q, & (|s| \bmod q) \neq 0 \\ 0, & (|s| \bmod q) = 0 \end{cases} \tag{4}$$

需要注意的是,特征索引采用倒排索引结构,格式为 (*signature*, (*tid*, *pos*)),*tid* 是元组 *id*, *pos* 是特征在字符串中出现的位置。

2.3　过滤算法

基于静态数据集的近似查询方法通常设定一组过滤算法获得中间结果——候选集。由于离线场景对查询返回的实时性要求不高,往往忽略了过滤算法的执行时间也将影响查询的时间开销。因此,本文认为适合数据流环境的过滤算法应满足两点:执行速度快以及过滤能力强。这两点能降低过滤和验证的时间开销,保证查询结果返回的实时性。经过对已有基于静态数据集的过滤算法进行测试和分析,我们认为适合数据流环境的过滤算法有:

长度过滤(length-filtering):根据编辑距离的定义,若数据流 S 中元组 s 的内容与查询关键字 k_i 满足 $ed(s, k_i) \leqslant \tau$,则它们的长度一定满足

$$\min(|k_i|) - \tau \leqslant |s| \leqslant \max(|k_i|) + \tau, \quad k_i \in K, s \in S \tag{5}$$

位置过滤(position-filtering):内容匹配的数据

流 q-chunk 特征和关键字 q-gram 特征的位置之差应小于等于编辑距离阈值 τ，即

$$| q - \text{chunk}.pos - q - \text{gram}.pos | \leqslant \tau \quad (6)$$

本文发现高效于静态数据集的前缀过滤(prefix filtering)算法不再适用于数据流，原因包括：首先，前缀过滤的前提条件是统计一个全局序 O，通常通过统计特征出现频率(term frequency，*tf*)，按特征的 *itf* 降序排列 ($itf = 1/tf$) 获得，之后根据 O 对提取的字符串特征进行排序，选取前 $|S| - T + 1$ 个特征作为前缀($|S|$ 为特征集合大小，T 为特征重合下限)，但是数据流的无边界及不可预知性使得无法统计数据流的全部特征来确定全局序，即使计算得到某段时间内的 O，随着数据流不断到来还将引起 O 的变化，需要重新进行计算。其次，根据全局序对提取的特征进行排序也占用较大的时间开销。第三，假定随意设定一个全局序或以查询关键字特征的 *itf* 作为全局序，可能起不到过滤的作用而又引入了时间开销。因此 AS^3 不采用任何与统计相关的前缀类过滤算法。根据数据流自身的特点，本文提出了以下两个新的过滤算法来降低空间和查询时间开销：

(1)预剪裁过滤(pre-prune filtering，PPF)。该理论依据：已有过滤算法大都是针对索引的查询阶段设计的，若将过滤算法应用于索引创建阶段，就能够提前过滤不可能相似的字符串，减少滑动窗口索引的空间开销。其核心思想是：由于数据流连续查询的关键字相对固定，因此可将关键字的特征作为创建基本窗口索引的参照，即提取查询关键字的 q-gram特征作为基本窗口特征索引的初始值。由于一次字符操作只可能存在于一个 q-chunk 和 q 个 q-gram中，因此一个单字符操作最多能够影响一个 q-chunk与 q-gram 不匹配。例如：假设有字符串 r = "*apple*"，s_1 = "*appde*"，s_2 = "*abace*"，$q = 2$，$\tau = 1$，r 的 2-gram 特征集合 $G_2(r)$ = {ap, pp, pl, le, e \$}，s_1 和 s_2 的2-chunk 特征集合 $C_2(s_1)$ = {ap, *pd*, e \$}，$C_2(s_2)$ = {*ab*, *ac*, e \$}，$C_2(s_1)$ 与 $G_2(r)$不匹配的特征数小于等于 1，则将 s_1 作为候选项，而 $C_2(s_2)$到 $G_2(r)$不匹配的特征数大于 1，则 $ed(r,s_2) > 1$。预剪裁过滤算法的描述如图 3 所示，如果提取到的 q-chunk 特征与倒排索引初始 *token* 不匹配的个数大于 τ，则一定不满足编辑距离小于等于 τ，所以可以忽略该字符串；如果不匹配的 q-chunk 个数小于等于 τ，则可将该字符串的特征加入索引。

算法 2. Pre-Prune Filtering (I_K, Bw, τ)

输入：查询特征索引 I_K，基本窗口 Bw，编辑距离阈值 τ.

输出：基本窗口索引 I_{Bw}.

① get *tokens* of I_K as the initial value of I_{Bw}

② **for** s_i in Bw **do**

③ **for** $chunk_i$ in $C_q(s_i)$ do

④ N = count *mismatched* chunks against I_K;

⑤ **if** ($N > \tau$) continue;

⑥ **else** insert $C_q(s_i)$ into I_{Bw}}}

图 3 预剪裁算法

(2) 流统计过滤(count-filtering on stream，CFS)。基于静态数据集的非对称特征的统计过滤算法(count-filtering，CF)[11]、q-chunk 和 q-gram 特征重合度下限的计算公式如下：

$$| G_q(r) \cap C_q(s) | \geqslant \lceil | s | /q \rceil - \tau \quad (7)$$

式(7)的含义是当特征长度 q 确定时，若字符串满足编辑距离阈值 τ 的约束，则它们的 q-chunk 和 q-gram特征的重合度下限由 q-chunk 所属的字符串长度决定。如前例中，$| G_2(r) \cap C_2(s_1) | = 2$，公式右端结果为 2，满足编辑距离阈值为 1 的重合度下限。如果将式(7)直接应用在数据流上存在的问题是：由于 AS^3 对数据流字符串提取 q-chunk * 特征，所以式(7)需要对数据流上的所有到来的元组都重新计算一次下限，并且需要维护字符串长度，计算开销及查询开销比较大。为解决这个问题，基于 2.2 节的特征提取策略，我们提出了适合数据流的统计算法 CFS，算法的基本思想是：若字符串 r 和 s 满足 $ed(r,s) \leqslant \tau$，则一定满足 $||r| - |s|| \leqslant \tau$，带入式(7)右端得

$$\left\lceil \frac{|r| - \tau}{q} \right\rceil - \tau \leqslant \lceil |s| /q \rceil - \tau \leqslant \left\lceil \frac{|r| + \tau}{q} \right\rceil - \tau \quad (8)$$

式(8)的含义是：若字符串 r 和 s 满足 $ed(r,s) \leqslant \tau$，它们的非对称特征集合重合下限的存在范围，也就是满足长度过滤的字符串 s 的 q-chunk 特征集合，通过 τ 次字符编辑，与 r 的 q-gram 重合剩余的特征数。当式(8)的上限与下限在[0,1]时，特征重合度可以被唯一确定；当式(8)上限与下限之差大于 1 时，即 $2\tau \geqslant q$，而下限应大于等于 1 才有意义。对于短字符串而言，当 τ 取值较大时，重合度下限为 0 的概率很大，因此忽略左端下限为 0 的情况，并收紧式(8)左端得到新的重合度下限为

$$G_q(r) \cap C_q(s)\begin{cases}\geqslant \lceil \dfrac{|r|-\tau}{q} \rceil - \tau, & 2\tau < q \\ > \lceil \dfrac{|r|-\tau}{q} \rceil - \tau, & 2\tau \geqslant q\end{cases} \tag{9}$$

下面进行算法复杂度分析。由于式(7)需要对数据流上的每个新到来的元组都计算一次特征重合度下限,因此其时间复杂度为 $O(n)$,其中 n 为数据流的字符串个数;而流统计过滤(CFS)算法通过查询关键字长度以及特征长度便能够确定特征重合度下限,并且只需一次计算,因此其时间复杂度为 $O(m)(m << n)$,m 为查询关键字个数。CFS 算法有效降低了计算特征重合度下限的时间复杂度,提升了查询的实时性,然而当面对短字符串并且编辑距离较大的情况时,CFS 算法会因收紧下限而损失小部分结果。

2.4　基于矩阵坐标的验证算法

验证候选集的直接方法是采用动态规划算法计算编辑距离,其时间复杂度为 $O(|r| \times |s|)$[6]。文献[10]提出了基于 prefix-pruning 观察的验证方法。在动态规划算法计算过程中,通过判断单元内容来提前终止验证:假设动态规划算法按行计算矩阵 M,$M[i,j]$ 表示子串 $r[1:i]$ 与 $s[1:j]$ 的编辑距离,算法只计算并验证距离主对角线 $[-\tau,\tau]$ 范围内的单元,即 $M[i,[i-\tau, i+\tau]]$。若该范围的所有单元都大于 τ,则说明编辑距离一定大于 τ,可提前终止,而最坏情况需要计算到矩阵最后一行验证 $M[|r|, |s|]$,其时间复杂度为 $O((2\tau+1) \times \min(|r|, |s|))$,仍然存在冗余单元的计算。为避免该情况,我们提出基于矩阵坐标的验证算法 CV(Coordinate-based Verification)。为不失一般性,假设 $ed(r, s) \leqslant \tau$,$|r| \geqslant |s|$ 且 $|r|-|s| \leqslant \tau$,通过下列步骤判断是否计算单元 $M[i,j]$:

(1)若单元坐标满足 $|i-j| > \tau$,说明 $r[1:i]$ 与 $s[1:j]$ 的编辑距离大于 τ,因此不需要计算 $M[i,j]$;

(2)若单元坐标满足 $|i-j| \leqslant \tau$,根据长度过滤可知:$r[1:i]$ 与 $s[1:j]$ 的编辑距离下限 $LB_{r[1:i];s[1:j]} = |i-j|$,同理 $r[i+1:|r|]$ 与 $s[j+1:|s|]$ 的编辑距离下限 $LB_{r[i+1:|r|];s[j+1:|s|]} = |(|r|-i)-(|s|-j)|$。若满足 $ed(r,s) \leqslant \tau$,则以上两个编辑距离下限之和一定小于等于 τ。

(1)若 $i < j$,可得 $LB_{r[1:i];s[1:j]} = j-i$,$LB_{r[i+1:|r|];s[j+1:|s|]} = ||r|-|s|+(j-i)|$,可以推出 $j-i \leqslant ((\tau-(|r|-|s|))/2$。

(2)若 $i \geqslant j$,可得 $LB_{r[1:i];s[1:j]} = i-j$,$LB_{r[i+1:|r|];s[j+1:|s|]} = ||r|-|s|-(i-j)|$。

(i)若 $i-j \leqslant |r|-|s|$,可以推出 $LB_{r[1:i];s[1:j]} + LB_{r[i+1:|r|];s[j+1:|s|]} = |r|-|s|$,由于已知 $|r|-|s| \leqslant \tau$,因此需要计算 $M[i,j]$。

(ii)若 $i-j > |r|-|s|$,可以推出 $i-j \leqslant (\tau+|r|-|s|)/2$。

上述推导过程给出了我们需要计算的矩阵单元 $M[i,j]$ 的坐标应满足

$$-((\tau-(|r|-|s|))/2 \leqslant i-j \leqslant (\tau+|r|-|s|)/2 \tag{10}$$

式(10)为 CV 算法判断是否计算矩阵单元的条件,其时间复杂度为 $O((\tau+1) \times \min(|r|, |s|))$。由于算法避免了计算不相关的矩阵单元,从而降低了验证的时间开销。图 4 是 CV 算法验证过程的伪代码描述。

算法 3. Coordinate-based Verification (r, s, τ)

输入:字符串 r、s,编辑距离阈值 τ

输出:布尔值

```
① for(int i = 0; i ≤| s |; i ++) M[0,i] = i;
② for (int j = 0; j ≤| r |; j ++) M[j,0] = j;
③ for (int i = 1; i ≤| s |; i ++){
④ for (int j = 1; j ≤| r |; j ++){
⑤   if (i - j ≤ (τ +| r | -| s |)/2 and i - j ≥- ((τ -(| r | -
      | s |))/2)
      M[i, j] = min(M[i - 1, j - 1] +1, M[i, j - 1] +1, M[i
      - 1, j]) +1
⑥   if (M[i, * ] > τ)    return false;
⑦   else continue;}
⑧ return true;
```

图 4　基于矩阵坐标的验证算法

3　实验结果与分析

为搭建数据流测试环境,我们采用 LAN 的两台机器,其中一台机器从磁盘读取数据并封装为数据包连续发送到另一台机器,接收数据的机器在连续的数据流上进行字符串近似查询。测试机器硬件为双核 AMD CPU 2100MHz,内存 2GB,操作系统为 Linux Cent OS 6.3,内核版本号 2.6.32。AS^3 原型系统采用 Java 实现,JDK version 1.6.0,测试采用数据集 English-Dict①,它包含 150k 个字符串,平均长

① http://dbgroup.cs.tsinghua.edu.cn/wangjn/data/word.format.tar.gz

度为 9,图 5 是数据集 English-Dict 的字符串长度分布示意图。

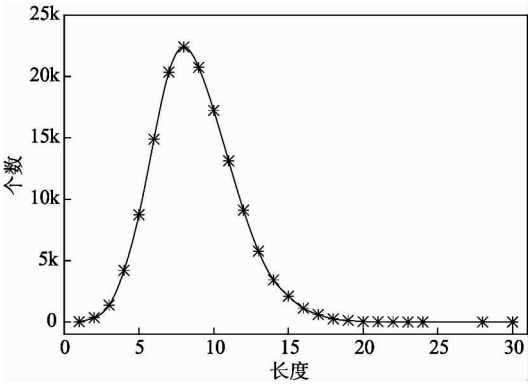

图 5 English-Dict 的字符串长度分布

3.1 特征索引的创建

对比三种特征提取方法——q-gram、非对称特征中的 q-chunk 以及改进的 q-chunk *(未加入预剪裁过滤(PPF)。采用基于计数的滑动窗口,在相同条件下测试创建特征索引的性能。滑动窗口大小为 10^5,基本窗口为 10^4,特征长度 q 的取值为 2 和 3。测试并对比创建基本窗口索引的平均时间开销和平均索引大小。图 6 给出了滑动窗口索引创建的平均时间对比。

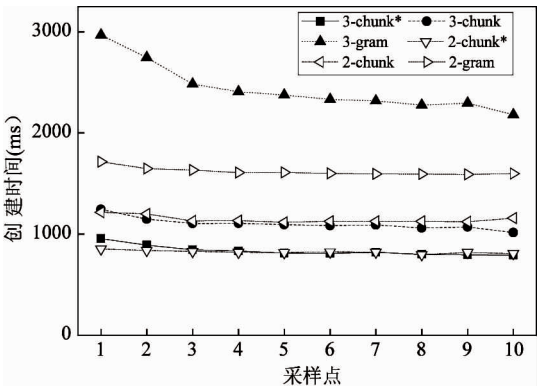

图 6 滑动窗口索引创建的平均时间对比

图 6 中我们固定基本窗口大小为 1000,测试创建滑动窗口索引的时间,其中横坐标为创建索引的采样点。我们可知,采用 q-chunk * 提取并创建基本窗口特征索引的时间开销最少,比 q-gram 降低了 50%,比 q-chunk 降低了 15% ~20%。图 7 是固定滑动窗口,改变基本窗口大小的索引创建时间,可以看到 q-chunk * 随着基本窗口的增大,创建所需的时间小于 q-gram 和 q-chunk。图 8 是基本窗口索引大小,由图可知采用 q-chunk * 创建基本窗口索引比 q-gram降低了 40% 的空间开销,比 q-chunk 降低了约 10% 的空间开销。改进的非对称特征提取策略的q-chunk * 在时间和空间开销上更适合在数据流上创建特征索引。

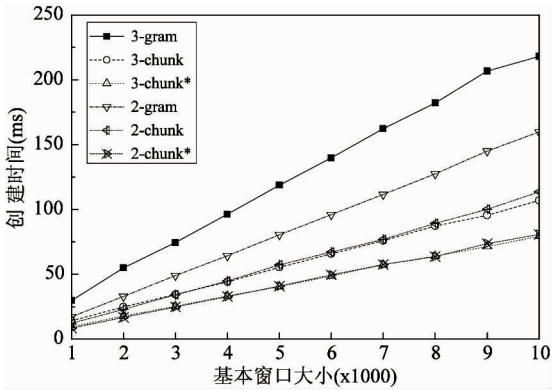

图 7 基本窗口索引创建的平均时间对比

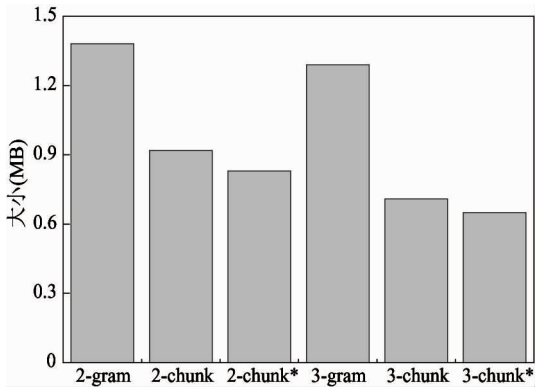

图 8 基本窗口索引的平均大小对比

3.2 过滤算法性能

测试预剪裁过滤(PPF)和流统计过滤(CFS)的性能。我们随机选取 50 个字符串作为查询关键字,编辑距离阈值取 1 和 2。

PPF 算法。图 9 是对比加入 PPF 过滤算法前

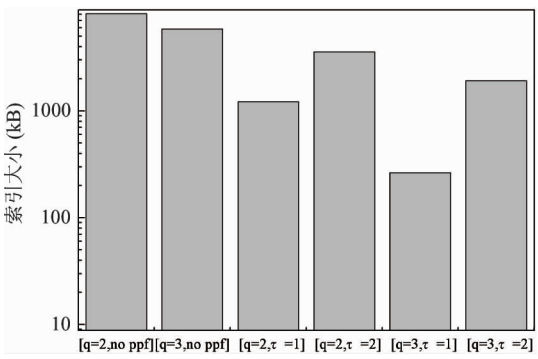

图 9 加入 PPF 算法前后的滑动窗口索引大小对比

后,滑动窗口索引的平均大小。由图可知,加入 PPF 算法后有效降低了索引空间开销,索引大小与查询关键字集、特征长度以及编辑距离阈值相关。查询关键字越多,则生成的索引占用空间越大;特征长度越大,索引展占用空间越小;编辑距离阈值越小,过滤强度越大,被过滤掉的字符串也越多。

下面对比原非对称特征的统计过滤 CF 算法与 CFS 算法的性能。图 10 是数据流滑动窗口发生更新时,触发查询后两个算法的执行时间对比,横坐标为采样点。可以看到 CFS 算法对基本窗口执行的时间开销大约是 CF 算法的 50%,并且时间曲线更加平缓,这是因为 CFS 的计算复杂度低于 CF 算法。

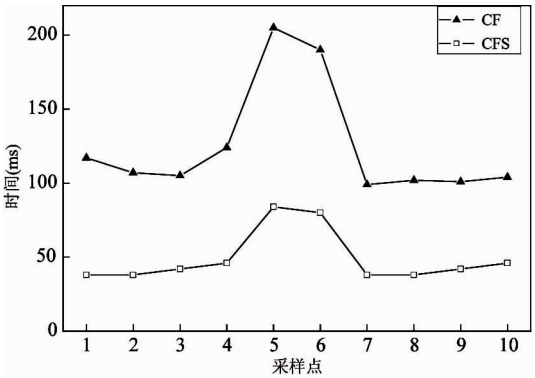

图 10　CF 与 CFS 的执行时间对比

表 1 是算法 CF 和 CFS 在数据流连续查询时,关键字对应的候选集的平均大小,合并倒排链表采用 ScanCount 算法[16]。由表 1 可知,CFS 算法生成的候选项更少,说明它的过滤强度好于 CF 算法。

表 1　算法生成候选集的平均大小

	CF	CFS
$q=2,\tau=1$	399	56
$q=2,\tau=2$	2977	2408
$q=3,\tau=1$	1805	287
$q=3,\tau=2$	26274	4690

表 2 列出了验证候选集的部分查询结果,可以看到当 q 和 τ 较小时,CFS 能够保证查询结果的完整性和准确率;当 q 和 τ 较大时,CFS 损失了小部分结果,其原因是对短字符串而言,单字符编辑操作将破坏了所有 q-chunk,使得无法与 q-gram 特征索引重合,而 CFS 因收紧重合度下限而将其过滤。实验对 50 个字符串构成连续查询,当 $q=2$ 时,查询结果的准确率达到 95%,当 $q=3$ 时,$\tau\geqslant 2$ 时,准确率下降至 83% 左右。

表 2　部分查询结果与真实结果的对比

	$q=2$		$q=3$		真实
	CF	CFS	CF	CFS	结果
happy, $\tau=1$	7	7	7	7	7
happy, $\tau=2$	60	60	60	46	60
hello, $\tau=1$	9	9	9	9	9
hello, $\tau=2$	110	110	110	85	110
instrumentation, $\tau=1$	2	2	2	2	2
instrumentation, $\tau=2$	3	3	3	3	3

通过以上实验,可得到如下结论:在 AS^3 中当查询和数据流中涉及的字符串长度较短时,并且查询编辑距离阈值较大时,应尽量采用较小的 q 值;而对长字符串的查询而言,若编辑距离阈值较小时,可适当增大 q 值,不会对查询结果带来影响。

3.3　峰值测试

测试对比三种查询处理方法的最大处理峰值:

(1)对新到元组直接计算它与查询关键字的编辑距离;

(2)采用基于基本窗口的滑动窗口模型,原非对称特征提取和过滤算法(长度、位置、统计);

(3)本文提出的 AS^3。

验证过程均采用 CV 算法。测试结果如表 3 所示。可以看到方法(2)的峰值处理能力比方法(1)提升了 100%;AS^3 的峰值处理能力比方法(1)提升了约 4 倍,比方法(2)提升了约 45%。由此可以看出 AS^3 在数据流上具备更好的峰值处理能力。

表 3　峰值测试结果

	平均时间(μs/tuple)	峰值(tuple/s)
方法(1)	4.1	3.2×10^5
方法(2)	1.3	7.9×10^5
方法(3)	0.8	1.2×10^6

4　结　论

随着网络技术的发展,越来越多的数据流实时地在网络中传输。面对数据流上的字符串近似查询的挑战,本文提出了基于滑动窗口的数据流字符串近似查询(AS^3)方法。它基于过滤与验证的框架和编辑距离,采用基本窗口的更新机制,改进并应用非对称特征策略创建特征索引,提出了适合数据流的新过滤算法 CFS 和 PPF 以及新验证算法 CV。实验

结果证明了,AS^3 方法能够有效地降低滑动窗口索引的创建时间和空间开销,支持索引实时更新,其过滤算法有效降低了索引空间及候选集大小,验证算法 CV 具有更低的计算时间复杂度,从而使其能够很好地支持数据流上的近似字符串查询,具备较高的实时性和峰值处理能力。下一步研究工作是基于滑动窗口的数据流字符串相似连接问题。

参考文献

[1] Chaudhuri S, Ganti V, Kaushik R. A primitive operator for similarity joins in data cleaning. In: Proceedings of the 22nd International Conference on Data Engineering, Atlanta, USA, 2006. 5-5

[2] Williams H E, Zobel J. Indexing and retrieval for genomic databases. *IEEE Transactions on Knowledge and Data Engineering*, 1998, 14(1): 63-78

[3] Chaudhuri S, Kaushik R. Extending autocompletion to tolerate errors. In: Proceedings of the 35th ACM International Conference on Management Of Data, Rhode Island, USA, 2009. 707-718

[4] Hristidis V, Valdivia O, Vlachos M, et al. A system for keyword search on textual streams. In: Proceedings of the 7th SIAM International Conference on Data Mining, Minneapolis, Minnesota, USA, 2007. doi: 10. 1137/1. 9781611972771. 52

[5] Hristidis V, Valdivia O, Vlachos M, et al. Continuous keyword search on multiple text streams. In: Proceedings of the 15th ACM International Conference on Information and Knowledge Management, Arlington, Virginia, USA, 2006. 802-803

[6] Navarro G. A guided tour to approximate string matching. *ACM computing surveys* (*CSUR*), 2001, 33(1): 31-88

[7] Xiao C, Wang W, Lin X. Ed-Join: an efficient algorithm for similarity joins with edit distance constraints. *Proceedings of the Very Large Data Bases Endowment*, 2008, 1(1): 933-944

[8] Li C, Wang B, Yang X. VGRAM: Improving performance of approximate queries on string collections using variable-length grams. In: Proceedings of the 33rd International Conference on Very Large Data Bases, Vienna, Austria, 2007. 303-314

[9] Wang W, Qin J, Chuan X, et al. VChunkJoin: An Efficient Algorithm for Edit Similarity Joins. *IEEE Transactions on Knowledge and Data Engineering*, 2012, 25(8): 1916-1929

[10] Li G, Deng D, Wang J, et al. Pass-join: A partition-based method for similarity joins. *Proceedings of the Very Large Data Bases Endowment*, 2011, 5(3): 253-264

[11] Qin J, Wang W, Lu Y, et al. Efficient exact edit similarity query processing with the asymmetric signature scheme, In: Proceedings of the 37th ACM International Conference on Management Of Data, Athens, Greece, 2011. 1033-1044

[12] Gravano L, Ipeirotis P G, Jagadish H V, et al. Approximate string joins in a database (almost) for free. In: Proceedings of the International Conference on Very Large Data Bases, Rome, Italy, 2001. 491-500

[13] Arasu A, Ganti V, Kaushik R. Efficient exact set-similarity joins. In: Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, South Korea, 2006. 918-929

[14] Wang J, Feng J, Li G. Trie-join: Efficient trie-based string similarity joins with edit-distance constraints. *Proceedings of the Very Large Data Bases Endowment*, 2010, 3(1-2): 1219-1230

[15] Golab L, Garg S, Özsu M. On indexing sliding windows over online data streams. Advances in Database Technology-EDBT, 2004. 547-548

[16] Li C, Lu J, Lu Y. Efficient merging and filtering algorithms for approximate string searches. In: Proceedings of the 24th IEEE International Conference on Data Engineering, Cancun, Mexico, 2008. 257-266

Sliding-window based approximate string search on data streams

Cui Jia[*,**,***], Wang Weiping[***], Chen Zhongtao[*,**,***], Meng Dan[***]

([*] CCAR, Institute of Computing Technology, Chinese Academy Of Sciences, Beijing 100190)

([**] University of Chinese Academy of Sciences, Beijing 100049)

([***] Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093)

Abstract

The approximate string search on data streams was studied to face the gradual transfer of the data accessing from the static disk mode to the dynamic data stream mode with the development of network technologies. To solve the problem that current static-dataset based approximate searching methods cannot work efficiently on data streams because data streams are continuous, boundless and unpredictable, and the resources for online computing are limited, a sliding-window based method for Approximate String Search on data Streams, called the AS^3, was proposed based on a filter-and-verification framework. The AS^3 adopts a basic window mechanism to facilitate real-time index update, and improves the asymmetric signature scheme for the construction of basic window signature index. Furthermore, it uses the pre-prune filtering (PPF) algorithm, the count-filtering on stream (CFS) algorithm and the coordinate based verification (CV) algorithm. The experimental results show that the AS^3 can achieve the great query performance on data streams while ensuring the accuracy of results with the higher real-time response and peak processing capacity on data streams.

Key words: data stream, approximate string search, sliding window, edit distance