基于剪切的 XML 数据流自适应发布算法

霍 欢[1)] 陈庆奎[1)] 王国仁[2)] 彭敦陆[1)] 郝聚涛[1)] 高丽萍[1)]

[1)](上海理工大学光电信息与计算机工程学院 上海 200093)

[2)](东北大学信息科学与工程学院 沈阳 110004)

摘 要 XML 数据流上的分片策略是基于剪切的 XML 数据流发布系统面临的首要问题.文中针对基于剪切的 XML 数据流中对 XML 片段解析和连接的操作代价,提出了基于 Hole-Filler 模型的 XML 数据流的基本代价模型,在此基础上提出数据流自适应发布算法 AXF,以期在数据和查询动态变化的情况下自动调整 XML 数据分片策略以获得最佳的系统运行性能、自适应能力和扩展性.实验结果表明 AXF 算法可以提高 XML 片段的有效率,在客户端、服务器及网络传输方面均获得良好的性能.

关键词 XML;数据流;自适应;剪切模型;发布

中图法分类号 TP311 **DOI 号**: 10.3724/SP.J.1016.2010.01953

The Adaptive Fragmentation for XML Stream Dissemination

HUO Huan[1)] CHEN Qing-Kui[1)] WANG Guo-Ren[2)] PENG Dun-Lu[1)] HAO Ju-Tao[1)] GAO Li-Ping[1)]

[1)](*School of Optical-Electrical and Computer Engineering, University of Shanghai for Science and Technology, Shanghai* 200093)

[2)](*School of Information Science and Engineering, Northeastern University, Shenyang* 110004)

Abstract The fragmentation policy over XML stream is the first major problem confronted by the XML stream dissemination system based on document fragmenting. This paper analyzes the features of the XML processing on client, network and server, and brings in the cost-model for fragmented XML stream system based on Hole-Filler model. According to the cost analysis of the parsing operation and join operation over streams on clients, this paper proposes the Adaptive XML Fragmentation algorithm (AXF) on server to dynamically merge and spit the XML fragments with XML data and query characteristics varying constantly over time. The core algorithms, A-Merge and A-Split, monitor and respond to both data and query changes automatically by using online profiler as the indication to the clients' requirements. When XML data and query characteristics were to stabilize, AXF would lead the XML fragments converge to an optimistic fragmentation rapidly. A thorough performance evaluation presents that AXF can sacrifice some fragment validity to balance the cost between transmission amount over network and the cost of query evaluation on client. And the server can achieve 2～2.5x performance improvement in scalability compared with query matching server. To the best of our knowledge, AXF is the first dynamic XML fragmenter considering merging and splitting XML fragments in an adaptive and efficient way.

Keywords XML; stream; adaptivity; fragmentation; dissemination

---

收稿日期:2010-08-22.本课题得到国家自然科学基金(60970012)、上海市重点学科建设项目(S30501)、上海市高校优秀青年教师后备人选基金(slg08012)、上海信息技术领域重点科技攻关项目(09511501000)和上海市教委科研创新项目(08YZ98)资助.**霍 欢**,女,1979 年生,博士,讲师,研究方向为 XML 数据流管理等. E-mail: huohuan_hh@yahoo.com.cn. **陈庆奎**,男,1966 年生,教授,博士生导师,研究领域为网络计算、并行计算等. **王国仁**,男,1966 年生,教授,博士生导师,研究领域为数据库理论和技术、生物信息学等. **彭敦陆**,男,1974 年生,博士,副教授,研究方向为 Web 服务计算、XML 数据管理等. **郝聚涛**,男,1976 年生,博士,讲师,研究方向为无线传感器网络、分布式计算等. **高丽萍**,女,1980 年生,博士,讲师,研究方向为 CSCW、协同计算等.

1 引 言

在基于剪切的 XML[1] 数据流发布/订阅系统[2]中，为了使数据易于同步，满足实时性以及数据重发等需求，一组服务器需要从信息源(一个或多个)不断收集 XML 数据，将大文档剪切为适于网路传送的多个小文档片段，并在异步、无确认的模式下将 XML 片段推送到客户端. 客户端接收到 XML 数据后，在本地进行查询[3]. 这样，如何对 XML 数据进行分片成为基于剪切的 XML 数据流发布/订阅系统的首要问题.

目前，基于服务器/客户端传输模式对 XML 文档分片的研究大多根据文档结构本身进行分片，不考虑数据的变化以及用户查询对分片的需求[4-5]. 文献[4]提出了基于 DOM 的 XML 数据流分片策略，根据节点的扇出度(即元素节点的儿子节点数目)对文档进行剪切，并在此基础上提出了基于标签的 XML 数据流分片算法 TF，对标签结构进行剪切以确定 XML 文档上的剪切点，并提出了优化的剪切策略，以减少碎片的产生. 文献[5]描述了 Xstream 系统中根据无线网络的最大传输单位(MTU)对 XML 文档进行剪切的算法 UF，即按层次遍历顺序对 XML 文档进行剪切，对不能包含进 MTU 的节点，通过增加一个虚拟的父亲节点形成新的 XML 数据单元(XDU).

在实际应用中，系统服务器不断接到来自源端的数据以及来自客户的查询，这必然造成系统资源需求的增长，包括 CPU 处理能力[6]、内存容量[7]以及带宽[8]等. 由于不同的应用对 XML 数据的剪切策略有不同的要求，如何设计 XML 流数据的剪切策略，以使在数据流和查询同时变化的情况下，服务器端、客户端和网络传输都能充分发挥系统性能，是基于剪切的 XML 数据流发布/订阅系统面临的难点. 目前还没有针对 XML 数据流自适应分片算法进行的研究. 本文通过对基于剪切的 XML 数据流发布/订阅系统进行分析，建立服务器端、客户端和网络传输的代价模型，提出了基于 Hole-Filler 模型的 XML 数据流自适应发布算法，以期在 XML 数据流和用户查询动态变化的情况下，算法能自适应调整 XML 数据的分片策略以获得最佳的系统运行性能、自适应能力和扩展性.

本文第 2 节给出本文的背景知识，即 Hole-Filler 模型和基于剪切的 XML 数据流发布/订阅系统模型；第 3 节具体介绍基于剪切的 XML 数据流基本代价模型；第 4 节在基本代价模型的基础上，提出基于剪切的 XML 数据流发布算法；第 5 节介绍性能测试结果；第 6 节总结全文.

2 背景知识

基于剪切的 XML 数据流系统采用 Hole-Filler 模型表达 XML 片段，采用发布/订阅模式发送 XML 片段，本节以数字图书馆为例介绍基于 Hole-Filler 模型的 XML 数据流发布系统模型.

2.1 Hole-Filler 模型

Hole-Filler[9] 模型是 XML 数据片段的关联结构，如图 1 所示. 一个 XML 片段 f 可以表示为 XML 文档上的一棵子树，即 $T_f=(V_f,E_f,\delta_f,\Sigma_f,root_f,fid,tsid_f)$. 其中 V_f 是 XML 片段中元素节点的集合；E_f 是节点之间边的集合；δ_f 是父节点到其所有子节点的一个映射函数. 每一个节点都有一个类型和一个类型标识符相对应，分别属于集合 Σ_f 和 $tsid$，$root_f$ 是一个 XML 片段的根节点，fid 是 XML 片段的标识符. 一个 Hole 节点 h 是 XML 片段中的空节点 $n(n\in\Sigma_f)$，除了对应的类型标识符 $tsid$，还对应唯一的节点标识符 hid.

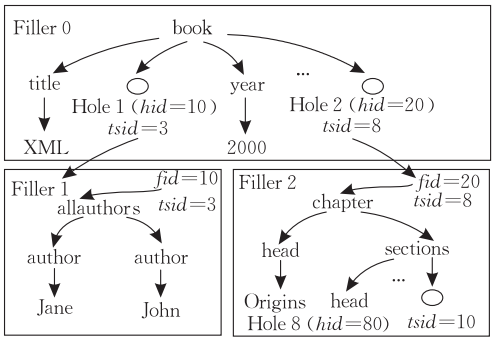

图 1 XML 片段

除了对 XML 剪切片段的关联外，要同步服务器和客户端，还需要定义标签结构[10]. 标签结构是一个独立的模式信息片段，不仅提供每个 XML 文档元素节点间的结构信息，同时也包含 XML 片段的分片信息. 图 2 给出了示例文档对应的标签结构. 其中，每个标签都有一个类型属性，当标签结构中的标签类型为“Filler=true”时，表明 XML 文档中的元素节点为剪切成 XML 片段的根节点，否则为 XML 片段的中间节点.

```
〈stream：structure〉
  〈tag name="book" id="1" 〉
    〈tag name="title" id="2"/〉
    〈tag name="allauthors" id="3" Filler=true〉
      〈tag name="author" id="4"/〉
    〈/tag〉
    〈tag name="year" id="5"/〉
    〈tag name="chapter" id="6" Filler=true〉
      〈tag name="head" id="7"/〉
      〈tag name="sections" id="8" 〉
        〈tag name="head" id="9"/〉
        〈tag name="section" id="10" Filler=true/〉
      〈/tag〉……
  〈/tag〉
〈/stream：structure〉
```

图 2　标签结构

2.2　基于剪切的 XML 数据流发布系统模型

现以数字图书馆为例介绍基于剪切的 XML 数据流发布系统模型，如图 3 所示. XML 分片器对收集的 XML 数据进行分片，XML 片段分发器对标签结构进行广播. 订阅管理器维护客户的订阅请求，包括位置信息、时间戳以及感兴趣的 XML 片段等等，XML 片段分发器据此向客户发送订阅的 XML 数据. 客户在本地进行查询.

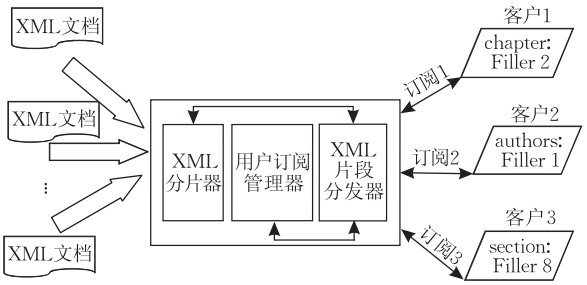

图 3　XML 片段发布/订阅系统模型

由于标签结构提供了所有有效的路径，在目前普遍采用的流水线查询模型中[11-13]，客户端查询可以映射到一组操作符序列，每个操作符对应于 XML 文档上特定层次的特定 XML 元素类型，可通过查询子树的标签 ID($tsid$)，激活和连接相应的操作符，以解析和连接相应的 XML 片段.

3　基于剪切的 XML 数据流系统的基本代价模型

基于剪切的 XML 数据流发布/订阅系统主要基于服务器/客户端传输模式，因此，系统的基本代价由服务器代价、客户端代价以及网络代价 3 部分组成.

3.1　客户端代价模型

由于 XML 片段流查询必须满足低资源占用、实时返回结果的要求，因此客户端查询通常采用流水线结构. 为不失一般性，针对流水线结构上的 XML 片段解析和连接操作得出客户端的代价模型.

定义 1(候选片段，Candidates).　假设客户端 C 接收 XML 片段流 S，在 m 个相关的片段 f_1，f_2，…，f_m 上对查询 Q 进行处理. 如果客户端 C 接收了片段 f_i，则片段 f_i 称为查询 Q 的候选片段.

查询 Q 的查询计划由操作符 O_1，O_2，…，O_n 构成. 当含有标签码 $tsid$ 的片段 f 到达时，对应的操作符 O_i 将片段 f 的信息记录下来，并且把片段 f 和相关联的片段连接起来. 如果没有接受 f 的操作符，则片段 f 为非候选片段，不会被客户端接收.

在 XML 文档的剪切策略 F 下，查询 Q 的代价是获得 XML 文档中所有查询结果的处理时间. 由于 XML 元素的缺失、错误是通过重发 XML 片段修正的，且 XML 片段的接收顺序是无序的，所以这里不考虑等待接收候选片段的时间和重发时间，从而将非候选片段的探测时间排除在外.

处理候选片段的时间代价可以分为两个部分：解析片段阶段的代价和片段连接阶段的代价. 当所有的候选片段完成解析和连接这两个阶段后，客户端 C 得到最终的查询结果.

定义 2(解析时间，T_p).　考虑候选片段 $F=f_1,f_2,\cdots,f_k$. 用 t_p 表示一个 XML 单位片段(如 10KB 等)的平均解析时间，$size(f_i)$ 表示返回片段的单位大小值，$freq(f_i)$ 表示片段 f_i 的传送次数. 候选片段的解析时间为

$$T_p=\sum_{i=1}^{k} size(f_i)\,freq(f_i)\,t_p \tag{1}$$

定义 3(连接时间，T_q).　考虑候选片段 $F=f_1,f_2,\cdots,f_k$. 用 t_q 表示两个片段进行连接的单位时间，用 $\delta(O_i)$ 表示 o_i 接收的片段数目，用 $P(O_i)$ 表示 O_i 的父亲片段，则片段连接的时间代价为

$$T_p=\sum_{O_i,P(O_i)\in O)} \delta(O_i)\delta(P(O_i))\,t_q \tag{2}$$

根据 XML 片段的解析时间和连接时间，对文档分片策略 $F=f_1,f_2,\cdots,f_m$，查询 Q 的总时间代价为

$$T_p=\sum_{i=1}^{k} size(f_i)\,freq(f_i)\,t_p+\sum_{O_i,P(O_i)\in O)} \delta(O_i)\delta(P(O_i))\,t_q \tag{3}$$

3.2　网络代价模型

网络代价与 XML 片段流传输的介质、传输模型、采用的传输协议等多种因素有关. 不失一般性，

不考虑 XML 片段流在不同的传输的介质、不同传输协议中的差别. XML 片段流的网络传输模型采用多播介质,客户先根据标签结构在服务器端进行注册,建立逻辑信息通道,之后由服务器在异步、无确认的模式下将 XML 片段推送到客户端. 当 XML 片段产生缺失或者损坏,客户端利用反馈通道向服务器发送显式请求,使服务器对缺损数据进行重传.

这样,网络代价主要与 XML 片段传输代价和维护逻辑通道代价有关. 首先,XML 片段传输代价与传输的所有 XML 片段的大小($size(f_i)$)总和成正比(比例系数设为 K_T). 需要说明的是,一些优先级高或者查询次数多的 XML 片段可能传送多次,如标签结构,因此网络传输代价不能简单等同于 XML 文档的大小,还要考虑 XML 片段重传的频率. 其次,维护逻辑通道的代价与逻辑通道的总数(用 $\delta(channel)$ 表示)相关(系统参数设为 K_M),包括逻辑通道的建立与通信代价. 由于系统采用推送模型,逻辑通道建立后,只有当客户更新订阅或要求数据重传时才发生通信,因此大大降低了客户与服务器间的通信代价. 则网络代价可近似描述为

$$C_n=\sum_{i=1}^{k} size(f_i)\,freq(f_i)K_T+\delta(channel)K_M \quad (4)$$

3.3 服务器端代价模型

如果把服务器端维护客户请求代价表示为 $\delta(sub)K_A$,把XML文档分片代价表示为 $size(D)K_B$,把发布 XML 片段的代价表示为

$$\sum_{i=1}^{k} size(f_i)\,freq(f_i)K_C,$$

其中,K_A,K_B 和 K_C 为相应的系统参数,服务器端代价可以用下面的公式来表示:

$$C_s=\delta(sub)K_A+size(D)K_B+\sum_{i=1}^{k} size(f_i)\,freq(f_i)K_C \quad (5)$$

4 基于剪切的 XML 数据流自适应发布算法

自适应的 XML 发布算法包括 A-Merger 算法和 A-Split 算法. 假设 XML 文档在服务器端已经剪切好,剪切方式不限. 同时设系统最大分片单位为 MFU,不考虑不同网络媒介和终端对数据分片粒度的不同需求.

4.1 自适应合并算法(A-Merge 算法)

A-Merge 算法是服务器端对 XML 片段动态合并的算法. 通过 A-Merge 算法对文档剪切策略的优化,客户端对 XML 片段的查询时间(包括解析时间和连接时间),会随着片段结构的更新而变化. 因此定义分片不等式,用以衡量调整前后客户端的查询代价变化.

定义 4(分片不等式,Fragmentation Invariant). 分片 $F=f_1,\cdots,f_n$ 满足剪切不等式,当下面的不等式成立

$$\Delta T_p+\Delta T_q\leq 0 \quad (6)$$

其中,ΔT_p 表示调整分片策略前后的片段解析时间变化,ΔT_q 表示相应的连接时间变化. 在应用 A-Merge算法前后,客户端应始终满足分片不等式 FI. A-Merge 算法由签名器和处理器构成. 签名器是服务器收集和维护客户签名的数据结构,客户签名包括了客户的订阅信息以及采样信息. 经过签名器处理的统计信息被提交给处理器,用以探测当前的合并分片策略是否满足分片不等式 FI,是则执行合并算法.

4.1.1 A-Merge 签名器

A-Merge 处理器通过签名器对 ΔT_p 和 ΔT_q 进行监测,因此 A-Merge 签名器可看作客户签名在服务器端的视图,如图 4 所示.

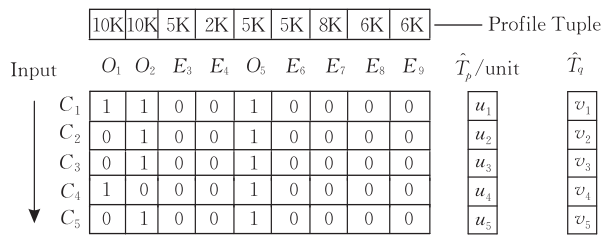

图 4 服务器端的签名器

图 4 中的每行比特序列对应一个客户签名. 每个签名包含 n 位,每位对应一个操作符 O 或 E,表示接收对应的 XML 元素 $t_1,t_2,\cdots,t_n$,且 t 的下标对应标签位置 $tsid$. 如果 $tsid$ 的属性为"Filler=True",则对应的操作符为 XML 片段操作符类型,记为 O,否则为元素类型 E. 如果查询节点落在以 t 为根节点的子树内,则将 t 对应的操作符 O 比特位置 1,否则置 0. 签名的更新可以通过比特位重置实现.

(1) 连接时间的估算(ΔT_q)

XML 片段的合并策略目的是减少相邻父子片段的总的连接时间. 由于具有相同 $tsid$ 的片段由同一个类操作符接收,相邻片段的连接时间可由相邻操作符接收的片段数目之积进行估算. 并且签名的统计信息包括了客户端单位平均连接时间 v_i,这样,

签名器可以估算任意一个客户端的 T_q.

(2) 解析时间的估算(ΔT_p)

由于服务器在文档分片时可计算片段的大小和数目,因而通过设置签名元组(profile tuple),即为每个操作符接收的 XML 片段的大小和数目进行记录(见图 4),即可估算单位解析时间. 其中,u_i 表示客户 C_i 解析单位 XML 片段的平均时间,乘以接收片段数目后即得到客户端 C_i 平均解析时间.

4.1.2　A-Merge 处理器

当合并策略符合 FI 时,A-Merge 处理器即对 XML 片段进行合并. 但并不是所有的 XML 片段都可以合并,这里首先讨论违反分片不等式 FI 的几种情况,然后提出合并规则.

(1) 违反 FI 的几种特例

如果合并后的分片策略导致解析时间 T_p 和连接时间 T_q 的总和比合并前增大,则称为违反 FI 不等式(FI violation). 以下 3 种情况都可能导致违反 FI 不等式.

① O_i 是 O_j 的父亲操作符. O_i 接收的片段和 O_j 接收的片段在合并后,客户端的连接时间 T_q 减少了,但是解析时间 T_p 增长了. 这是因为不是所有的客户都需要接收 O_i 或 O_j 产生查询结果.

② O_i 是 O_j 的兄弟操作符. O_i 接收的片段和 O_j 接收的片段在合并后,客户端的连接时间 T_q 并没有减少,因为 O_i 和 O_j 的父亲操作符仍然需要与 O_i 和 O_j 进行两次连接操作. 但是由于不是所有的客户都需要接收 O_i 或 O_j 产生查询结果,解析时间 T_p 反而增长了.

③ O_i 与 O_j 既不是父亲操作符,也不是兄弟操作符. O_i 接收的片段和 O_j 接收的片段在合并后,客户端的连接时间 T_q 并没有减少,但是由于不是所有的客户都需要接收 O_i 或 O_j 产生查询结果,解析时间 T_p 增长了.

(2) 合并规则

XML 片段的合并会引入客户并不感兴趣的部分,定义片段有效率来表示片段中与客户查询相关部分的比率.

定义 5(片段有效率,Fragment Validity). 假设客户查询涉及元素 $e_1,\cdots,e_n$,片段 f 的有效率定义为

$$FV=\frac{\sum_{i=1}^{n} size(e_i)k}{size(f)} \tag{7}$$

其中,当片段 f 包含元素 e_i,k 值为 1,否则 k 值为 0. 定义 5 为衡量片段的有效部分提供了量化依据.

在不违反 FI 的情况下,当具有父子关系的 XML 片段同时被客户端查询时,即父亲片段和儿子片段总是同为候选片段或者同为非候选片段,可以对 XML 片段进行合并,FV 的值则为父子片段的 FV 均值. 但由于查询是不断变化的,上面的特例并不总满足条件. 因此可以适当调整 FV 的值,使合并策略可适应查询的变化. 算法 1 描述了 A-Merge 算法对两个片段的判定和合并过程,当且仅当下面的条件成立

$$\frac{C(O_i,O_j)}{\max(|O_i|,|O_j|)}>\frac{1}{\hat{f}v} \tag{8}$$

其中,$C(O_i,O_j)$ 表示 O_i 和 O_j 同时为"1"的父子关系操作符的数目,$|O_i|$ 表示操作符 O_i 对应的比特位为"1"的数目. 有效参数 $\hat{f}v$ 是一个常量,用来避免将两个比特位值相差很多的 XML 片段合并到一起,从而控制 XML 片段的有效部分的比率.

算法 1.　A-Merge 算法.

输入:片段 i,片段 j

输出:合并后的片段

1. $\Delta T_p=0$;
2. if (fragment i is a non-candidate fragment) then
3. 　$\Delta T_p=T_p+Size.i$;
4. end if
5. if (fragment j is a non-candidate fragment) then
6. 　$\Delta T_p=T_p+Size.j$;
7. end if
8. if (the relationship between i and j is parent-child) then
9. 　$\Delta T_q=-t_q*Number.i*Number.j$;
10. end if
11. if (i and j are brothers) then
12. 　$\Delta T_q=-t_q*\min(Number.i,Number.j)$;
13. end if
14. if ($\Delta T_p+\Delta T_q<0$) then
15. 　Insert fragment j into fragment i;
16. 　Delete fragment j;
17. 　Update corresponding *Size* and *Number*;
18. 　Update A-Merge profiler;
19. 　Update tag structure and multicast it;
20. end if

A-Merge 处理器先根据 $|O_i|$ 值对 XML 片段的请求频率进行排序,然后按照频率递减顺序对满足不等式 6 的父子关系片段进行判断,如果符合 FI 并且合并后的片段不大于最大分片单位 MFU,则执行 A-Merge 算法,直到所有满足条件的片段都进

行了合并操作.在合并片段过程中,片段中的节点在标签结构中的结构关系仍保持不变,只是分片信息发生了变化.因此,服务器仅对标签结构的分片信息进行更新,即将被合并片段的根节点的标签属性"Filler"由"true"改为"false",同时将 A-Merge 签名器中对应的比特位改为元素类型 E.

4.2 自适应剪切算法(A-Split 算法)

A-Split 自适应剪切算法利用签名器生成候选元素序列,由 A-Spit 处理器估算分片不等式并从片段分离有效的元素.

4.2.1 A-Split 签名器

A-Split 签名器与 A-Merge 签名器类似,记录客户查询的元素和服务器频繁更新的元素.利用 A-Merge 签名器的比特位,系统仅需标记谓词及查询元素 $tsid$ 的对应位.有些情况需要签名器两次设置比特位,如当查询元素是片段的根节点时,要将对应位置为"-1",当被合并到父片段时,要将比特位置为"1".利用签名器,服务器可收集那些频繁被数据源更新和大量被客户端请求的元素 $tsid$,将这些元素作为片段剪切的候选元素发送到 A-Split 处理器.

4.2.2 A-Split 处理器

接收到候选元素序列后,如果当前片段满足分片不等式,A-Split 处理器则完成剪切操作.

(1)分片不等式的估计

处理器首先选择片段 f 中的候选元素 $e_1,\cdots,e_n$.如果将这些元素分离,则片段 f 剩余的部分可近似地认为是查询的非候选片段.客户端在解析 XML 片段的时间代价上的变化为

$$\Delta T_p=-\left[size(f)-\sum_{i=1}^{n}size(e_i)\right]t_p \qquad (9)$$

剪切后,由于需要增加新的片段间的连接操作,连接时间代价会有所提高.图 5 展示了候选元素分离的一种情况,当新产生片段是另一个候选片段的孩子片段时,即新片段中的候选元素是片段 f 中候选元素的孩子,这种情况下连接时间代价会增加.将候选元素的集合$\{e_1,\cdots,e_n\}$表示为 E,则连接时间代价可估算为

$$\Delta T_q=\sum_{e_i,P(e_i)\in E}\delta(P(e_i))t_q \qquad (10)$$

其中,$P(e_i)$返回元素 e_i的父节点,$\delta(P(e_i))$表示和 $P(e_i)$具有相同 $tsid$ 元素的数目,即与 e_i相连接元素的数目.在计算出 ΔT_p和 ΔT_q后,处理器能够判断当前的剪切操作是否满足分片不等式.

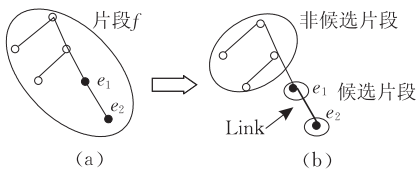

图 5 分离候选元素

(2)剪切规则

在满足分片不等式的条件下,处理器可根据下面的规则完成剪切操作.

(1)候选元素 e 是 f 中的叶子节点.首先产生仅包含元素 e 的新片段,然后从 f 中删除 e,如图 6(a)所示.

(2)候选元素 e 是 f 中的内部节点.首先产生仅包含元素 e 的新片段,然后生成包含 e 后代节点的另一个片段.从 f 中删除 e 和它的后代,如图 6(b)所示.

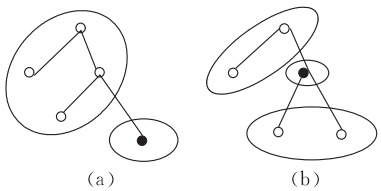

图 6 分离候选元素举例

在完成剪切操作后,元素间仍然保持标签结构中的结构关系,仅仅是分片信息产生了变化.在第 2 种情况下,如果 e 的后代节点包含另一个候选元素,分离元素 e 和它的后代会产生复杂的结果.为了处理这种情况,首先根据候选元素在标签结构中的深度进行排序,然后按照深度递减的顺序完成对元素的分离.这样,分离出来的元素或者是叶子节点或者是内部节点,它们的后代中不会存在任何候选元素.

处理器将候选元素剪切成独立的片段,每个新的片段包含一个元素.然而,新产生的片段可能含有可以合并的候选元素,这时可利用 A-Merge 算法,对产生的新片段进行合并.如果合并后仍满足分片不等式,合并操作就会进行下去,直到得到一个最优方案,图 7 给出了一个寻找最优方案步骤的例子.

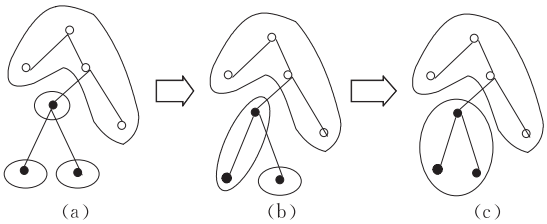

图 7 分离-合并过程举例

处理器对片段中的候选元素进行分离，直到完成对所有候选元素的重新分片操作. 然后服务器将分离元素的标签属性“Filler”修改为“true”，更新标签结构；并将相应比特位由元素类型 E 修改为操作符类型 O，更新 A-Split 签名器. 分离步骤详见 A-Split 算法.

算法 2.　A-Split 算法.

输入：片段 f，候选节点 $e_1,e_2,\cdots,e_n$

输出：新的 XML 片段

1. Sort $e_1,e_2,\cdots,e_n$ in descending depth order;
2. $\Delta T_p=-Size.f*t_p$;
3. for (int $i=0$; $i<n$; $i++$) do
4. $\Delta T_p=\Delta T_p+EleSize.e_i*t_p$;
5. end for
6. $\Delta T_q=0$;
7. for (int $i=0$; $i<n$; $i++$) do
8. if ($Parent.e_i$ is in $\{e_1,e_2,\cdots,e_n\}$) then
9. $\Delta T_q=\Delta T_q+EleNumber.e_i*EleNumber.Parent.e_i*t_q$;
10. end if
11. end for
12. if ($\Delta T_p+\Delta T_q<0$) then
13. for (int $i=0$; $i<n$; $i++$) do
14. Create new fragment g_i with the element of e_i;
15. if (e_i is an internal node) then
16. Create new fragment g_i' with the descendants of e_i;
17. end if
18. Delete element e_i and its descendants from fragment f;
19. Update corresponding *Size* and *Number*;
20. Update A-Merge Profiler and A-Split Profiler;
21. Update and multicast tag structure;
22. end for
23. end if

5　性能测试和分析

实验环境设置为 WindowsXP 系统下，CPU 为 2.4GHz，内存为 512MB，硬盘存储为 80GB 的服务器和客户端. 服务器端实现了自适应的 XML 分片器 AXF，即 A-Merge 和 A-Split 算法，在单一数据流上实现对 XML 数据的分片优化. 为评估不同的分片策略，同时实现了随机分片器(RF)、基于模式的分片器(TF)[4]以及 XDU 分片器(UF)[5]. 客户端用 Java 实现了 XFPro[12]查询引擎. 数据集由 xmlgen 程序生成，利用 Xmark-AuctionDTD[14]产生一组路径表达式查询和相应的标签结构. 签名长度为 16bit，其它参数设置见表 1.

表 1　实验参数缺省值

	参数							
	元素类型个数	文档最大深度	最大重复次数	查询深度	相似度 γ	客户签名个数	平均解析时间 t_p/(ms/1M)	平均连接时间 t_q/(ms/1M)
缺省值	77	16	5	12	0.6	1000000	145.83	273.5

运行代价测试. 实验研究了在 XML 数据流和用户签名稳定时算法的性能. 影响因素主要在于客户端的处理能力，签名的大小以及它们之间的相似度. 系统使用一个相对简单的参数 γ 表示签名的相似度. 当 $\gamma=1$ 时，两个签名是完全相同的. 当 $\frac{C(O_i,O_j)}{\max(|O_i|,|O_j|)}>\gamma$ 时，认为两个签名是相似的. 实验固定 γ 的值并测试客户端的平均连接时间和平均解析时间.

图 8 显示了对不同大小的文件，服务器分别采用 A-Merge、A-Split、AXF、RF、TF、UF 算法的总处理时间，包括 XML 片段的剪切和分发时间. 对于 A-Merge、A-Split 和 AXF，处理时间也包括调整分片所需代价. 由于静态算法 RF 和 UF 没有考虑到查询需求，从而导致发布的分片数量增加. 而 TF 将每个元素封装入一个片段并且只发布被查询的元素，可作为最佳分发数量，因此 AXF 的服务器代价可看作接近理想状态的工作负载.

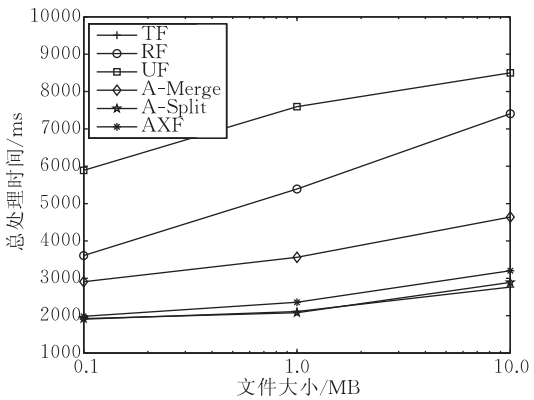

图 8　服务器端运行代价

图 9 显示了不同分片策略对客户端的影响，纵轴表示客户端平均查询时间. 由于 AXF 能自适应地将所需求的元素合并入分片中，并分离不相关的部分，因此要胜过其它的策略. 而 UF 则导致对不相

关内容的解析时间增大，TF 导致连接时间增大，因而性能较差.

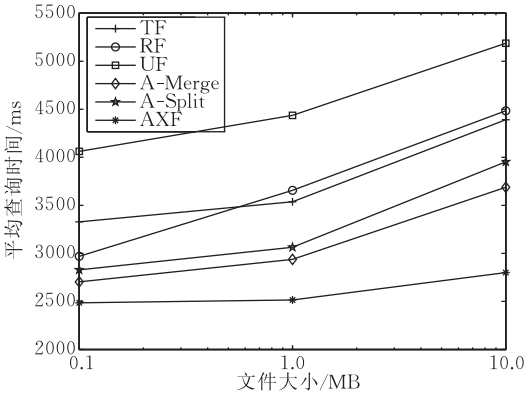

图 9 客户端运行代价

图 10 显示了不同分片策略下网络的传输量. 其中 UF 的传输量差不多是 AXF 和 TF 的 6 倍，这是由于查询结果没有均匀分布，增加了需要传输的分片数量. 由于 TF 只发布查询的元素，可看作最小传输量，而由于 A-Split 分离了不相关元素，所以很接近于最佳传输量.

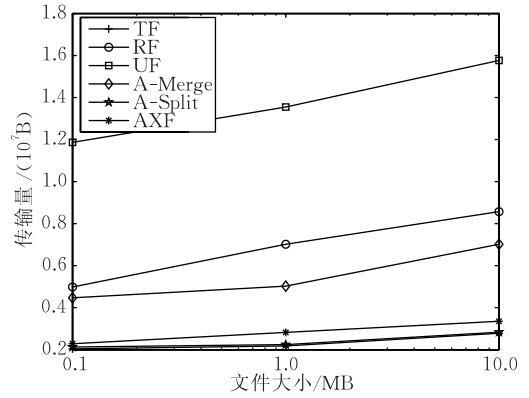

图 10 网络传输代价

片段有效率测试. 由定义 5 以及 γ 的定义可知参数 γ 与 FV 是成反比的，可通过变化不同的 γ 值来分析片段有效率与客户性能以及网络传输量之间的关系，如图 11 所示. 服务器端采用 AXF 分片算法，利用不同的 γ 值生成不同的分片策略. γ 值越小，被合并的分片和查询共享的公共部分就越少，因此网络传输量就更大，处理代价就更高. 当 γ 增加到 1 时，网络传输达到最小值，此时 XML 片段不包括任何不相关的元素. 但在 γ 调整到 1 之前，客户端代价就达到了下界. 这是因为不相关元素只影响客户端的解析时间，与连接时间相比可以忽略. 因此可以在一定程度上牺牲 FV 来得到客户端更高的处理效率.

自适应性测试. 图 12 显示了当 $\gamma=0.6$ 时，随着 XML 数据流和客户签名信息的不断变化，自适

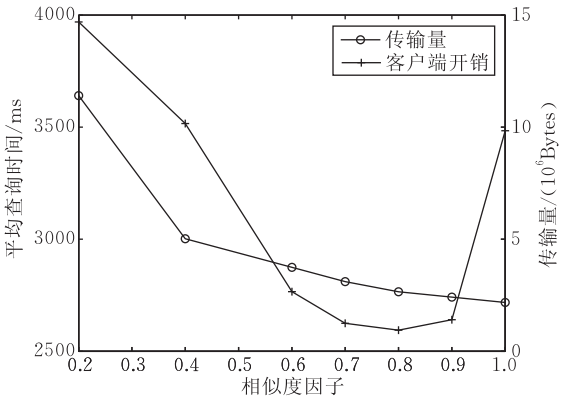

图 11 片段有效率与查询代价

应性测试下的分片有效率. 其中，AXF 和 A-Split 的分片有效率都高于 A-Merge. 这是因为随着合并操作的进行，不相关元素不断被引入到分片中，因而降低了分片有效率. 所以 AXF 的分片有效率也低于 A-Split.

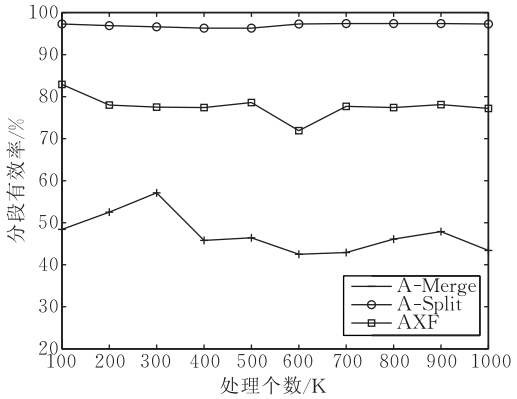

图 12 片段有效率的自适应性

图 13 显示了经 A-Merge、A-Split 和 AXF 优化后的客户端平均执行时间. 在服务器处理 200K 个签名后 AXF 所优化的 XML 分片表现出较稳定的处理效率，表明 AXF 很快确定了优化的分片策略. 而经 A-Merge 和 A-Split 优化的分片则随着签名的

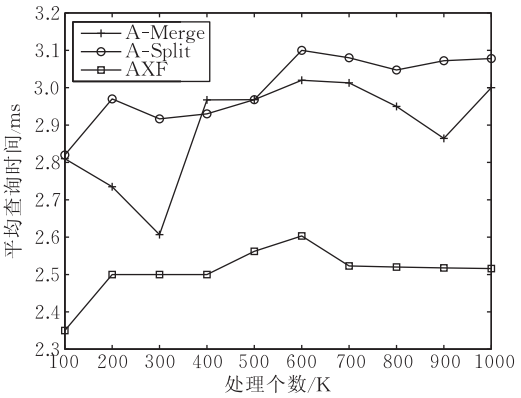

图 13 查询代价的自适应性

增加而变化显著，这是因为 A-Merge 和 A-Split 的优化策略必须协调进行，任何一个都不能单独达到优化的目的.

扩展性测试. 本测试比较基于 XML 片段分发的系统模型和基于查询匹配的系统模型间的系统扩展性. 对查询匹配系统模型，实现了基于树型模式(Tree pattern)聚合的贪婪算法 AGGR[15] 以及路径表达式过滤算法 XFiler[1] 和 YFilter[16]，并产生了一组 1000K 树型查询，使用 1MB 文件用于测试.

表 2 分析了这 4 种系统模型的特点. XFilter 和 YFilter 将查询构建为自动机对数据进行匹配，服务器直接将查询结果发给客户端，因此客户不需要精化查询结果，但随着 XML 数据量的增加，服务器端的自动机状态会急速增长，限制了系统的扩展性. AGGR 在此基础上对查询进行聚合，降低查询的复杂度，但导致查询精度的下降，需要客户端进行二次查询. 以上几种方式的缺点是当查询进行更新时，系统需要重新计算查询计划，因而系统扩展性仍有提升空间. 而 AXF 将查询处理推送至客户端处理，服务器只需根据客户请求对 XML 数据片段进行分发，无需对查询进行聚合，大大提高了系统的扩展性.

表 2　不同数据流系统的查询特点分析

匹配算法	查询结果精确度/%	客户端查询	查询聚合度	查询更新代价
XFilter	100	不需要	无	大
YFilter	100	不需要	低	大
AGGR	≤100	需要	高	大
AXF	≤100	需要	—	小

图 14 给出 4 种数据流系统的服务器响应时间. 当签名增加到近 400K 时，XFilter 的响应时间已经是 AXF 的 7 倍，YFilter 和 AGGR 由于对查询进行了聚合，响应时间仅为 AXF 的 2.5 倍和 2 倍. 但随着签名数量增长到 1000K，YFilter 和 AGGR 的响应时间也增长到 AXF 的 3 倍和 2.5 倍，这说明查询聚合只能在一定程度上缓解数据量和查询增长带来的负载. 在 AXF 中，由于分片可以被不同的查询重复使用，服务器不需要进行查询处理，因此可以获得较高的扩展性能.

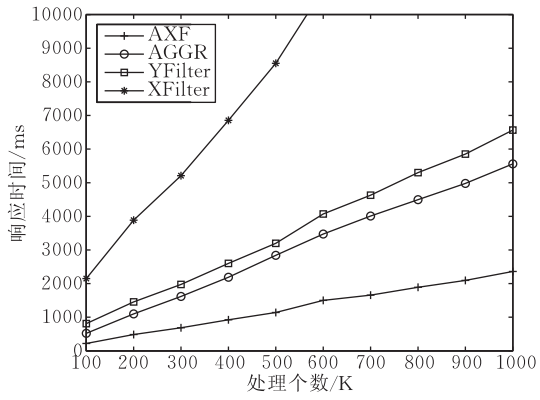

图 14　不同数据流系统的扩展性能比较

6　结　论

本文通过对基于剪切的 XML 数据流发布系统进行研究，建立服务器端、客户端和网络传输的代价模型，深入分析了系统性能的影响因素，提出了一系列优化 XML 分片策略的标准，以期获得最佳的系统运行性能、自适应能力和扩展性. 实验证明，基于 XML 数据流系统基本代价模型提出的动态自适应的 XML 数据流发布算法可以提高系统的总体性能. 未来的工作将进一步优化自适应算法的调度策略，当数据和查询急速变化时，使算法在自适应性和健壮性方面对瞬时产生的变化作出合适的反应，以降低分片策略更新带来的系统开销.

参　考　文　献

[1] Bray T, Paoli J, Sperberg-McQueen C M et al. Extensible markup language (XML) 1.0(Second Edition) W3C recommendation. World Wide Web Consortium, Technical Report: RECxml-20001006, 2000

[2] Altinel M, Franklin M J. Efficient filtering of XML documents for selective dissemination of information//Proceedings of the International Conference on Very Large Databases (VLDB). Cairo, Egypt, 2000: 53-64

[3] Fegaras L, Levine D, Bose S, Chaluvadi V. Query processing of streamed XML data//Proceedings of the International Conference on Information and Knowledge Management (CIKM). McLean, VA, USA, 2002: 126-133

[4] Huo Huan, Hui Xiao-yun, Wang Guo-Ren et al. Document fragmentation for XML streams based on Hole-Filler model. Journal of Huazhong University of Science & Technology (Natural Science Edition), 2005, 33(12): 249-252 (in Chinese)
(霍欢，回晓云，王国仁等. 基于 Hole-Filler 模型的 XML 流数据剪切分片技术. 华中科技大学学报(自然科学版)，2005，33(12)：249-252)

[5] Wong E Y C, Chan A T S, Leong H V. Efficient management of XML contents over wireless environment by Xstream//Proceedings of the ACM Symposium on Applied Computing (SAC). Nicosia, Cyprus, 2004: 1122-1127

[6] Tatbul N, Çetintemel U. Load shedding in a data stream manager//Proceedings of the International Conference on Very Large Databases (VLDB). Berlin, Germany, 2003: 309-320

[7] Babcock B, Babu S. Chain: Operator scheduling for memory minimization in data systems//Proceedings of the ACM-SIG-

MOD International Conference on Management of Data. San Diego, California, USA, 2003: 253-264
[8] Cheng R, Kalashnikov D, Prabhakar S. Evaluating probabilistic queries over imprecise data//Proceedings of the ACM-SIGMOD International Conference on Management of Data. San Diego, California, USA, 2003: 551-562
[9] Bose S, Fegaras L, Levine D, Chaluvadi V. A query algebra for fragmented XML stream data//Proceedings of the International Workshop on Data Base Programming Languages (DBPL). Potsdam, Germany, 2003. Springer, 2004: 6-8
[10] Bose S, Fegaras L. XFrag: A query processing framework for fragmented XML data//Proceedings of the International Workshop on the Web and Databases. Baltimore, Maryland, USA, 2005: 16-17
[11] Huo H, Wang G, Hui X et al. Efficient query processing for streamed XML fragments//International Conference on Database Systems for Advanced Applications. Singapore, 2006: 468-482
[12] Huo H, Zhou R, Wang G et al. Efficient evaluation of multiple queries on streamed XML fragments//Proceedings of the International Conference on Web-Age Information Management (WAIM). Hong Kong, China, 2006: 61-72
[13] Hui X, Wang G, Huo H et al. Region-based coding for queries over streamed XML fragments//Proceedings of the International Conference on Web Information Systems Engineering (WISE). Wuhan, China, 2006: 487-498
[14] Schmidt A, Waas F, Kersten M, Carey M J, Manolescu I, Busse R. XMark: A benchmark for XML data management//Proceedings of the International Conference on Very Large Databases (VLDB). Hong Kong, China, 2002: 974-985
[15] Chan C Y, Fan W, Felber P, Garofalaki M, Rastogi R. Tree pattern aggregation for scalable XML data dissemination//Proceedings of the International Conference on Very Large Databases (VLDB). Hong Kong, China, 2002: 20-23
[16] Diao Y, Franklin M J. High-performance XML filtering: An overview of Yfilter. IEEE Data Engineering Bulletin, 2003, 26(1): 41-48

HUO Huan, born in 1979, Ph. D., lecturer. Her main research interests focus on XML fragment streams management technology.

CHEN Qing-Kui, born in 1966, professor, Ph. D. supervisor. His main interests include network computing, parallel computing, and etc.

WANG Guo-Ren, born in 1966, professor, Ph. D. supervisor. His main research interests include database theory and technology, bioinformatics and etc.

PENG Dun-Lu, born in 1974, Ph. D., associate professor. His research interests include Web applications and service-oriented computing, XML data management and etc.

HAO Ju-Tao, born in 1976, Ph. D., lecturer. His main research interests include wireless sensor network and distributed computing.

GAO Li-Ping, born in 1980, Ph. D., lecturer. Her main research interests include CSCW and collaborative computing.

Background

Effectively processing XML streams is an important issue in data stream management systems. Unlike in traditional databases, since streaming data is endless and transient, queries on XML streams are bounded not only by memory but also by real time processing. Recently proposed Hole-Filler model is promising for information transmission and publication, by slicing XML data into low consuming and easy synchronized fragments. The research based on Hole-Filler model involves the problems in document fragmentation based XML stream management, and queries processing on disordered XML fragments. This paper considers the problem of XML fragmentation, where a continuous stream of XML fragments is generated by a fragmenter. Existing methods for XML fragmentation are limited to simply slicing XML data into fragments in terms of the contents and the structure of the document itself, without taking query information into account[4-5]. However, queries over streaming XML fragments require appropriate fragmentation of XML data to speed up query execution. According to the parsing operation and join operation in fragmented XML stream system, this paper analyzed the features of the data processing on client, network and server, and brought in the adaptive XML fragmentation algorithm (AXF) for scalable XML stream dissemination and proposed a series of fragmentation policies for XML streams to improve the system efficiency, adaptivity and scalability where XML data and query characteristics very constantly over time. A thorough performance evaluation presented that AXF can monitor and respond to both data and query changes automatically, improve fragment validity and achieve a good performance among the client, network and server. To the best of our knowledge, the AXF is the only XML fragmenter considering merging and splitting XML fragments in an effective and efficient way. The work is part of the project Study on key techniques in XML stream pub/sub systems based on document fragmenting and supported by the National Natural Science Foundation of China (60970012), Innovation Program of Shanghai Municipal Education Commission (08YZ98), Shanghai Key Science and Technology Project in Information Technology (09511501000), Shanghai Leading Academic Discipline Project (S30501) and the Nominated Excellent Youth-Teacher Program of Shanghai Higher Education (slg08012). As the other parts of the contribution to the project, several efficient query processing algorithms over streamed XML fragments were developed to speed up the query execution on clients[11-13] in our previous work, which intrigued the fragmentation problem through query analysis on client and finally helped to complete the performance evaluation for scalable XML fragments dissemination.