DEWS2004 5-A-01

XMLフィルタ配信システムにおける XPathの特徴量を用いた負荷分散方式

内山　寛之† 　鬼塚　真† 　芳西　崇†

† 日本電信電話株式会社 NTT サイバースペース研究所　〒 239-0847 神奈川県横須賀市光の丘 1-1
E-mail: †{uchiyama.hiroyuki,onizuka.makoto,honishi.takashi}@lab.ntt.co.jp

あらまし　近年，インターネットの普及により情報配信サービスが注目を集めており，大量のユーザにより早く情報を配信することが求められている．我々は，XML に対するクエリ言語の XPath を用いてフィルタリングを行い，大量のユーザがそれぞれに必要な情報だけをリアルタイムに受信できるシステムの実現を目指す．数万ユーザに対してフィルタ配信する場合，現状技術には，複数のフィルタサーバの負荷を均等に分散することが不可欠である．本稿では，配信 XML データを用いて XPath 式の特徴量を定義・利用し，追加される XPath 式がフィルタサーバに与える負荷を予測した上で，複数のフィルタサーバの負荷が分散されるように XPath 式の追加配置を行う手法を提案する．4 台による分散の評価実験を行った結果，ラウンドロビンによる分散手法と比較して，提案手法は XPath 式の登録数に対する配信時間の増加率が 1/2 となることを確認した．

キーワード　XML，e-service，プロファイル，問合せ処理，XPath

Load Sharing for XML Streams Filtering System with XPath Expression Features

Hiroyuki UCHIYAMA†, Makoto ONIZUKA†, and Takashi HONISHI†

† NTT Cyber Space Laboratories, NTT Corporation　1-1 Hikarinooka Yokosuka-Shi Kanagawa 239-0847 Japan
E-mail: †{uchiyama.hiroyuki,onizuka.makoto,honishi.takashi}@lab.ntt.co.jp

Abstract　The publish/subscribe interaction system is now receiving attention and it needs to scale well for a large number of subscribers. Our purpose is to develop a real-time XML streams filtering system that filters the input XML streams with a large number of subscribers' profiles that are written in XPath expressions and publishes the filtered data. Since the existing filtering server techniques do not efficiently proceed a large of subscribers, we have to suitably control the load sharing of filtering servers.In this paper,we propose the XPath distribution method by defining XPath expression features and utilizing them to share the servers' load. The exprimental results show that proposed method improves the increasing rate of publishing time for the number of XPath expressions twice smaller than round-robin method's rate.

Key words　XML,e-service,profile,Query Processing,XPath

1. はじめに

近年，インターネットの普及により広告配信やニュース配信，センサ情報を逐次的に配信するサービスなどの分野に適用可能な SDI システム [7], [9] が注目を集めており，XML でそれらを扱うためのスキーマが策定されている [1]〜[3]．今後ニュースや株価などの膨大な情報がリアルタイムにユーザへ XML データとして配信されることが予測されるが，ユーザの嗜好を XPath 式として表現し，嗜好に応じた情報のみをフィルタリング後に配信できれば，1．配信サーバの配信にかかるコストを減少させる事が可能であり，2．ユーザが情報の選別をする必要がないため，情報配信サービスはより実用的になる．

しかし，大量のユーザが配信サーバを利用する場合，配信サーバの配信にかかるコストが大きくなり，高速な配信を実現する事は困難となる．このため，複数の配信サーバを用意し，効率的に配信にかかるコストを分散化させる手法が必要となる．

配信される XML とユーザのプロファイルを利用したコンテンツベースルーティングは，大量情報の中からユーザに対

して必要な情報のみを配信する技術として注目を集めている [4], [8], [12]〜[14]．Chand らは，XPath 式を用いてコンテンツベースルーティングを行えるようなルータを配置し，XPath 式を登録したユーザに対して適切に XML データを配信するようなプロトコルを提案した [13]．XPath 式の包含関係を利用して登録される XPath 式が増加した際に，上位ルータに対して下位ルータに含まれる XPath 式全てを含むような XPath 式を上位ルータに登録する事で，ルーティングテーブルを減少させる事を可能にした．しかし，制約付きの XPath 式に関する包含関係しかもとめることができないことや，包含関係の計算に時間がかかるなどの問題がある．また，Snoeren らは，冗長にルータを用意し，同じ配信データをいくつものパスを通してクライアントへ配信する事で，高速かつ信頼度の高い配信を行うための手法を提案した [4]．しかし，XML ドキュメントひとつをパケットとして扱っているため，より細かいフィルタができない．

本研究では，大量のユーザへ複数のフィルタリング XML 配信サーバ (フィルタサーバ) を用いてフィルタリング XML データを配信する場合に，配信にかかるコストを均等に分散するように XPath 式の配置を高速に行うための手法を提案し，大量のユーザに対してもリアルタイム配信の実現を目指す．まず，XPath フィルタエンジン [15] を用いたフィルタサーバについて調査した結果，単体サーバでは数千ユーザの処理が限界であり，さらに多くのユーザへリアルタイムに配信を行うためには，複数のフィルタサーバを用意して分散処理する必要があることが分かった．しかし，複数のフィルタサーバを用意したとしても配信にかかるコストがサーバ間で偏ってしまう場合には分散されないという課題を抽出した．上記の課題に対し，入力された XML データに関する XPath 式の特徴量を抽出し，複数のフィルタサーバに配信コストを分散させる手法を考案し，数値実験を行い実データに対する有効性を示した．

本稿の構成を以下に述べる．2 章では，XPath フィルタサーバのアーキテクチャ，性能分析，課題抽出について述べる．3 章では，XPath 式の特徴量抽出及び特徴量を用いた XPath 式の分散配置手法を述べる．4 章では，数値実験を行い提案手法の有効性を検証する．5 章では，配信コストの加重平均を取り入れた手法について述べる．6 章では，まとめと今後の課題について述べる．

2. 現状の課題

2.1 XPath エンジン

本研究で用いる XPath エンジンについて述べる．SAX ベースの XPath エンジンは，ストリーミング XML データを処理する事ができる．　XPath 式をクエリとして，入力された XML データからフィルタリングを行う場合，SAX を用いたアプローチではオートマトンの受理問題に帰着する事が可能である．つまり，XPath 式と等価なオートマトンを構築し，XML データが読み込まれると，要素や属性に対するコールバック関数が呼ばれてオートマトンの状態遷移を行う [17]．

SAX ベースの XPath エンジンには，非決定性オートマトン (NFA) によるもの [6], [9], [16] と決定性オートマトン (DFA) によるもの [5], [15] がある．NFA は，状態遷移が一意に決定しないオートマトンである．一方，DFA は状態遷移先が一意に決定するオートマトンであり，NFA と等価な変換を行うことで構築される．NFA は XPath 式が増加すると処理速度が劣化するが，DFA は XPath 式が増えても処理速度が一定のまま高速に保たれるという特徴がある．

2.2 XPath フィルタサーバのアーキテクチャ

本研究では，DFA ベースの XPath フィルタエンジン [15] を用いて XPath フィルタサーバを開発した．図 1 には，サービス概要図を示す．まず，ユーザの嗜好性を XPath 式としてフィルタサーバに登録しておく．次に，フィルタサーバに XML データが入力されると，ユーザが必要とする部分 XML データのみを配信することが可能である．図 2 は，配信システムのアーキテクチャを示している．アプリケーションサーバは，XML データを配信するクライアントから XML データを受け取る (図中 1)．XML データを配信するクライアントから XML データが送られてくると，アプリケーションサーバは既に登録されている XPath 式に対するフィルタリングを XPath エンジンに要求する (図中 2)．XPath フィルタエンジンは解析処理を行うと同時に，コールバック関数を用いて XPath 式への結果をアプリケーションサーバへ伝える (図中 3)．最後に，アプリケーションサーバは，各ユーザが登録した XPath 式への結果のみをユーザへ配信する (図中 4)．

2.3 フィルタサーバの性能分析

前述のアーキテクチャより，フィルタサーバの処理は大きく分けて，ユーザに XML を配信する処理とオートマトンを利用したフィルタ処理に分けられる．

$$\text{総処理時間} = \text{フィルタ時間} + \text{配信時間}$$

入力された XML データのサイズとユーザがフィルタして受け取る XML データのサイズの比をフィルタ率と呼ぶ．フィルタ率をユーザ間で平均をとったものを平均フィルタ率としたとき，平均フィルタ率に関して，配信時間が総処理時間に占める割合を示したのが図 3 である．この図から，楕円で囲まれた部分の XML の場合 (フィルタ率が約 1%以上の範囲)，配信処理が XPath フィルタエンジンによる処理に比べて非常に大きい事が分かる．次に，平均フィルタ率が 5%の場合についてフィルタサーバの特徴を述べる．図 4 には，フィルタサーバに対して 100 KB の XML データを送り，ユーザそれぞれに 5 KB の配信を行ったとき，ユーザ数を変化させたときの配信時間を示している(注1)．この図より，ユーザ数に比例して処理時間が線形に増加しており，数万ユーザ以上を一台のフィルタサーバでリアルタイムに配信を行う事は難しい．図 5 は，フィルタ時間と配信時間の割合を示している．ユーザ数 2000 以上の時には配信コストが 80%ほどとなり，ユーザ数が小さい場合 (100 以下) においては，フィルタ処理時間が占める割合は大きくなっている．これは，今回採用した DFA ベースの XPath エンジ

(注1)：後述の数値実験と同様の環境であるとする．

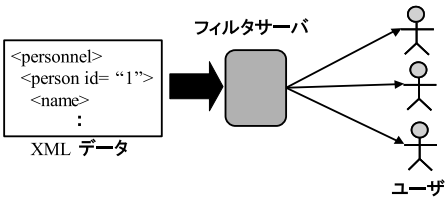

図 1　フィルタサーバサービス概要図

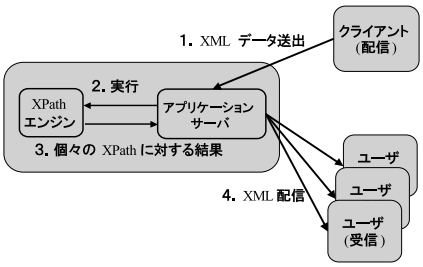

図 2　フィルタサーバアーキテクチャ

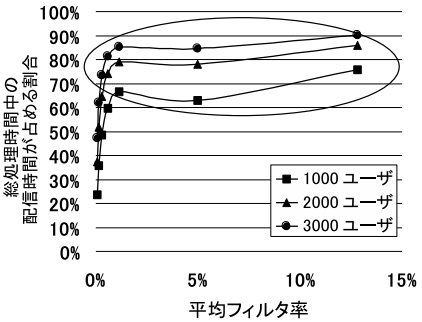

図 3　フィルタ率と配信時間の占める割合

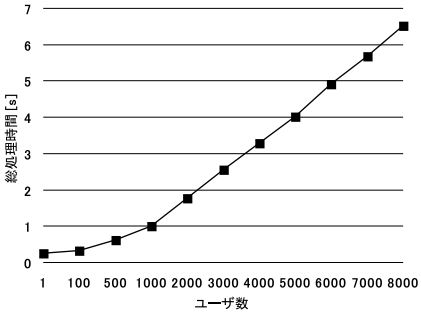

図 4　ユーザ数と総処理時間

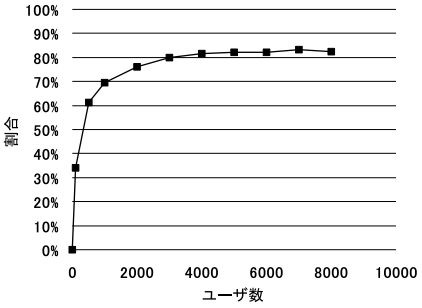

図 5　ユーザ数と配信時間が占める割合

ンでは，XPath 式が増加しても XPath フィルタリング処理に時間がそれほどかからない事から生じている．　図より，実際に配信されるフィルタの割合が入力 XML ドキュメントに対して 1%以上であれば，配信処理が相対的に大きいことを示している．以上より，本研究においては，楕円で囲まれた部分を対象として課題設定を行うが，今回提案する XPath の特徴量を用いて別の性質を持つ入力 XML データに対応可能である．

2.4　配信コスト分散の課題

2.1 節より，数万ユーザを超える場合には分散処理が必要であるが，総処理の 8 割はサーバの配信コストであることが分かった．従って，フィルタサーの負荷分散において，どのような XML データが入力されたとしても，配信コストが均等に分散していることが不可欠である．図 6 は，複数のフィルタサーバを並列分散した場合の模式図である．図のように，複数のフィルタサーバを用意して単純に XPath 式を割り振る場合，偏りが生じて適切な配信コストの分散ができない可能性がある．例えば，4000 件の XPath 式を 4 台のフィルタサーバへ均等に分けることを考える．この時，配信量は XML データと XPath 式によって変化する．ある XML ドキュメントが入力されたときにその XML ドキュメントの大部分を要求する XPath 式が 1000 本あったとする (他の 3000 本はほんとんど配信量が無い)．もしも，それらがひとつのフィルタサーバに登録されているならば，4 台のフィルタサーバを用意しても負荷分散している事にはならない．図 6 は，全てのフィルタサーバが均等に配信している場合であり，図 7 は，一部のフィルタサーバに配信処理が偏った場合である．図 7 から，全体の処理は配信コストが大きいフィルタサーバの性能劣化に影響を受ける事が分かる．

以上より，本研究において解決するべき課題を，リアルタイムにフィルタリング XML データを配信する事を目指し，図 6 のようなアーキテクチャで構成されるシステムにおいて配信コストの分散を行うこととする．

配信コストを分散させるためには，XML ドキュメントに対して複数の XPath 式がどれほど同時に配信しているかを知る必要がある．なぜなら，ストリーミング XML データに対する配信処理は，XML データを解析すると同時に行われるからである．入力された XML データに対して複数の XPath 式により要求される部分 XML データが重複すればするほど単位時間に配信する量が増加する事になる．　複数の XPath 式に対する結果 XML データの重複を判定する手段として，XPath 式の包含関係を利用する事が考えられるが，XPath 式そのものを比較して包含関係を取得する事は不可能であることが分かっている [11]．さらに，値の比較無し，かつスキーマ有りであるような XPath 式のサブセット XP(DTD,/,//,+,[],*,|) を考えた場合においても指数オーダ以上の計算量を必要とすることが分かっている [11].

提案手法は，入力 XML データから XPath 式の特徴を取得し，特徴量を利用して追加 XPath 式がフィルタサーバに与える影響を予測し，どのフィルタサーバへ配置するかを決定する．その実現のために，XPath 式そのものを評価するのではなく，特定の XML ドキュメントに対する XPath 式の結果を特徴として利用する．後述するように，特徴量の算出と算出された特徴量を用いて負荷分散を行う手法はリアルタイム性を崩すことなく可能となる．

XPath 式による配信コストの分散について，例を用いて説明する．図 6 では，上位フィルタサーバは，下位フィルタサーバに対して送られてきた XML ドキュメントをそのまま配信することが示されている．下位フィルタサーバは，上位サーバから送られてきたストリーミング XML データをフィルタリングし，ユーザへ配信する．XPath 式がこのシステムに追加されたときには，上位フィルタサーバがどの下位フィルタサーバへ登

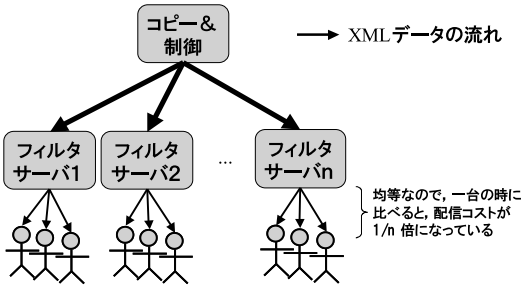

図 6　分散フィルタリング概略図

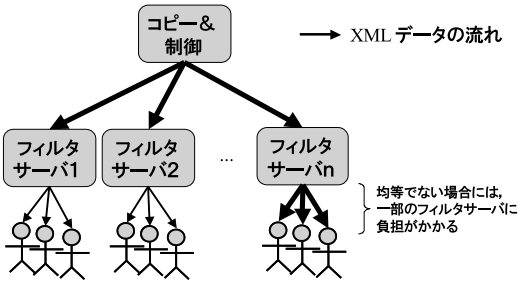

図 7　配信処理が偏っている場合

録させるかを決定する．フィルタサーバで利用している XPath フィルタエンジンは，DFA ベースであるため XPath 式の本数が増加しても XML 解析処理速度は劣化しない．そのため，上位フィルタサーバで下位フィルタサーバの持っている XPath 式を全て保持し，下位フィルタサーバの XPath 式を管理可能である．

3. 配信コストの分散手法

本章では，まず，入力 XML データに対する XPath 式の特徴量の定義と算出する手法について述べる．次に，XPath 式の特徴量を用いて XPath 式の包含関係及び重複度を算出するための関数，下位フィルタサーバの配信コストを算出するための関数を定義する．最後に，追加 XPath 式を下位フィルタサーバへ割り振るための XPath 式配置アルゴリズムを提案する．提案する下位フィルタサーバへの XPath 式配置手法は，配信コストの分散化に関して二つの評価軸がある．配信される XML データがストリーミングであるという視点から見た分散化手法と，XML がドキュメントであるという視点から見た分散化手法である．ストリーミングであるということを考えると，XML データをユーザへ配信する時に，一部の下位フィルタサーバへ一時的な配信コストが集中することを避ける手法をとるべきであり，下位フィルタサーバに登録された XPath 式から得られる結果ができるだけ重複していないことが必要となる．しかし，重複のみを考慮したアルゴリズムでは，重複度は低いが XML ドキュメント単位での配信量が大きい場合に対応できない．つまり，一時的な配信コストに関しては最適であっても，XML ドキュメントとしてみたときには配信量に偏りが生じる可能性がある．したがって，XML ドキュメントに対して，各フィルタサーバ間に配信量が差が出ないようにする必要がある．

3.1 節では，ストリーミング XML データに対する XPath 式の特徴量取得手法について述べる．3.2 節では，XPath 式の重複度計算と下位フィルタサーバの選択関数について述べ，XPath 式の配置アルゴリズムを提案する．

3.1 XPath 式の特徴量

XML ドキュメントの集合を $D = \{d_k | 1 \leqq k \leqq \infty\}$ とおく．添え字 k は，XML ドキュメントがストリーミングとして送られてくるために上限が無い．特徴量は，ある特定の XML ドキュメント d_k に対して特定されるが，今後 d_k に対することが明らかである場合には，煩雑さを避けるために省略する事とする．入力 XML データに対する XPath 式により抽出されたノードセットから特徴量を取得する．具体的には，抽出されるノードのはじめと終わりの場所，ノードの文字数である．ノードの文字数は配信する対象そのものなので，配信コストを表している．

XPath 式を表す集合を $P := \{p_i \in \mathcal{N} | 1 \leqq i \leqq |P|\}$ とおく．ここで，$|P|$ は，集合 P に含まれる要素数とし，p_i はひとつの XPath 式を表すものとする．$\mathcal{N}$ は，自然数を表す．次に，XPath 式 p_i に対応するノードセットに対する特徴量を $a^{p_i}(d_k)$ とする．ノードに対する特徴量は，$a_j^{p_i}(d_k) \in \Re^3 (1 \leqq j \leqq \epsilon)$ で表現されるものとする．$a_j^{p_i}(d_k) := (a_{j1}^{p_i}(d_k), a_{j2}^{p_i}(d_k), a_{j3}^{p_i}(d_k))$ は，ノードのはじめと終わりの場所，及びノードの文字数をそれぞれ表している．但し，$a_{j1}^{p_i}(d_k), a_{j2}^{p_i}(d_k), a_{j3}^{p_i}(d_k) \in \Re$．ここで，XPath 式 p_i に対応するノードセットに対する特徴量を $a^{p_i}(d_k) \in \Re^{\epsilon \times 3}$ とし，以下のように定義する事ができる (d_k は省略している)．

$$
\begin{aligned}
a^{p_i} &:= \begin{pmatrix} a_1^{p_i} \\ a_2^{p_i} \\ \vdots \\ a_\epsilon^{p_i} \end{pmatrix} \\
&= \begin{pmatrix} a_{11}^{p_i} & a_{12}^{p_i} & a_{13}^{p_i} \\ a_{21}^{p_i} & a_{22}^{p_i} & a_{23}^{p_i} \\ \vdots & \vdots & \vdots \\ a_{\epsilon 1}^{p_i} & a_{\epsilon 2}^{p_i} & a_{\epsilon 3}^{p_i} \end{pmatrix},
\end{aligned}
\tag{1}
$$

ここで，ϵ は，抽出されるノードセットの大きさを示しており，XPath 式 p_i と XML ドキュメント d_k により変化するため $\epsilon := \epsilon(p_i, d_k)$ と書く事ができる．煩雑さを避けるために，引数が明らかな場合には省略する．この ϵ は，フィルタリングが行われると同時に得る事ができる．図 8 は，入力 XML データに対し XPath 式 a^{p_i} の解釈の結果得られるノードセットの例を示している．図 8 上部に記してある XPath 式

/personnel/person/name

は，personnel 要素の子として person 要素をもち，さらにその子要素として name を持つような XML データを取得する事を示している．図 8 から，入力された XML データに対して XPath 式 p_i は特徴量行列 a^{p_i} をもち，下記のような行列となっている．

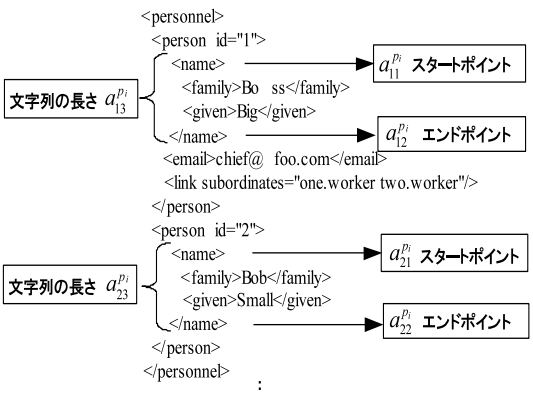

図 8　XML と特徴量の関係図

1.**startDocument()**
$g = 0$;initilalize S;

2.**startElement(len)**
$++g$; $\forall p_i \in S, a^{p_i}_{\epsilon 3}$+=len;

3.**endElement()**
$++g$;

4.**characters(len)**
$\forall p_i \in S, a^{p_i}_{\epsilon 3}$+=len;

5.**endDocument()**

6.**startContext(p_i)**
$++\epsilon$;$a^{p_i}_{\epsilon 1} = g$; S=$S \cup \{p_i\}$;

7.**endContext(p_i)**
$a^{p_i}_{\epsilon 2} = g$; S=S-$\{p_i\}$;

図 9　特徴量取得アルゴリズム

$$a^{p_i} = \begin{pmatrix} a^{p_i}_{11} & a^{p_i}_{12} & a^{p_i}_{13} \\ a^{p_i}_{21} & a^{p_i}_{22} & a^{p_i}_{23} \end{pmatrix}.$$

例では二つのノードがフィルタされるので $\epsilon(p_i, d_k) = 2$ となる．

次に，特徴量を取得するための手法を述べる．SAX ベースでアルゴリズムを作成するため，コールバック関数が呼ばれたときの処理を記す事でアルゴリズムとした．

XML ドキュメント d_k の位置情報を表すグローバル変数 g を用意する．引数の len は startElement の引数の場合には，要素の文字数を表し，characters の引数の場合には，内容の文字数を表す．また，複数の XPath 式を格納できるグローバル変数 S を用意する．以下にストリーミング処理による特徴量取得アルゴリズムを図 9 に示す．

ϵ は，startContext() が呼び出されるたびに拡張されている．アルゴリズム中のインデックス 1〜5 については，一般的な SAX パーサのコールバック関数である．startContext 及び endContext コールバック関数は，XPath 式 p_i を引数として持っており，それぞれ引数の XPath 式に対してフィルタリングの開始と終わりを通知する．endDocument 関数が呼び出されると 3.2 節で提案するアルゴリズムを開始する．次に二つの XPath 式の重複度を特徴量によって計算する手法について述べる．

図 10 には，XPath 式 p_1(/personnel/person/name) と p_2(/personnel/person/name/family) に対するフィルタ範囲を中括弧でそれぞれ記してある．この場合には，XPath 式 p_2 によって得られるノードセットが重複している． 特徴量ベクトル $a^{p_i}_s, a^{p_j}_t$ を引数としてとり，重複の度合いを返す関数 NodeLap:$\Re^3 \times \Re^3 \to \Re$ を下記のように定義する．

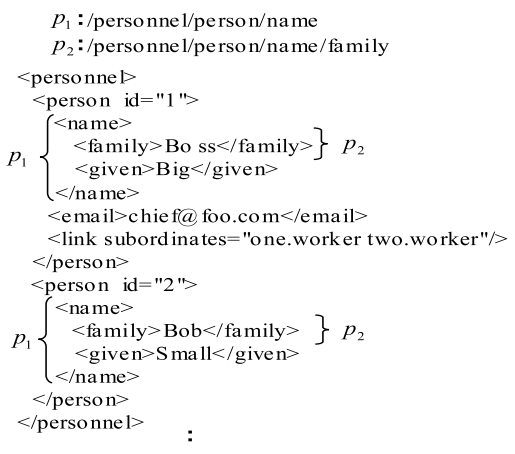

図 10　二つの XPath 式 p_1 と p_2 の範囲

$$\mathrm{NodeLap}(a^{p_i}_s, a^{p_j}_t) := \begin{cases} a^{p_j}_{t3} & \text{if } a^{p_i}_{s1} \leqq a^{p_j}_{t1} \ \wedge a^{p_j}_{t2} \leqq a^{p_i}_{s2} \\ a^{p_i}_{s3} & \text{if } a^{p_j}_{t1} \leqq a^{p_i}_{s1} \ \wedge a^{p_i}_{s2} \leqq a^{p_j}_{t2} \\ 0 & \text{(others)} \end{cases}.$$

関数 NodeLap は，二つの XPath 式の特徴量からその包含関係を取得する．包含関係が成立するのであれば，含まれる側の文字数を返す．包含関係が成立しない場合には，0 を返す．XML データは，木構造であるため包含関係がなければ，必ず結果ノードは交わらないため包含関係が無ければ出力が 0 となる．

関数 NodeLap を用いて，二つの XPath 式に対する特徴量行列 a^{p_i}, a^{p_j} を引数とし，それらの XPath 式間の重複度を出力するような関数 PathLap: $\Re^{\epsilon_i \times 3} \times \Re^{\epsilon_j \times 3} \to \Re$ を以下のように定義する．

$$\mathrm{PathLap}(a^{p_i}, a^{p_j}) := \sum_{s=1}^{\epsilon_i} \sum_{t=1}^{\epsilon_j} \mathrm{NodeLap}(a^{p_i}_s, a^{p_j}_t), \qquad (2)$$

Step 0(初期化)
$rw = s = t = 0;$

Step 1(条件分岐)
if($a_{s1}^{p_i} \leqq a_{t1}^{p_j} \ \wedge a_{t2}^{p_j} \leqq a_{s2}^{p_i}$)
rw+=$a_{t3}^{p_j}$;++t;
else if($a_{t1}^{p_j} \leqq a_{s1}^{p_i} \ \wedge a_{s2}^{p_i} \leqq a_{t2}^{p_j}$)
rw+=$a_{s3}^{p_i}$;++s;
else if($a_{t2}^{p_i} <= a_{s1}^{p_j}$)
++t;
else if($a_{s1}^{p_j} <= a_{t2}^{p_i}$)
++s;

Step 2(終了条件)
if($s > \epsilon_i \vee t > \epsilon_j$)
return rw;
else
goto Step 1;

図 11　区間比較アルゴリズム

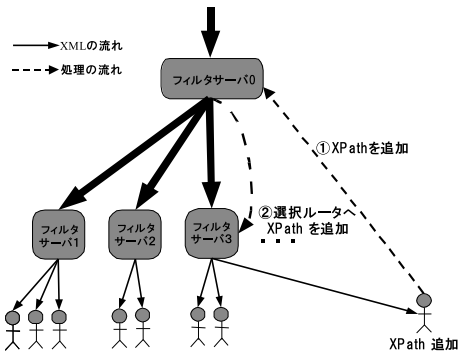

図 12　XPath 式が追加されるときの模式図

ここで，$\epsilon_i = \epsilon(p_i, d_k), \epsilon_j = \epsilon(p_j, d_k)$ である．関数 PathLap の返値は，XPath 式 p_i, p_j の XML ドキュメント d_k 上における重複度を示している．この関数は，XPath 式 p_i, p_j が持つ結果ノードの特徴量を比較し，重複しているならば和をとることを示している．式 (2) をそのまま演算すれば $O(\epsilon_i \cdot \epsilon_j)$ となる．この計算量は，ノードセットが大きくなると非常に大きな計算量がかかることを示している．そこで，ノードセットの重複度を $O(\epsilon_i + \epsilon_j)$ で算出するアルゴリズムを図 11 に示す．rw は，重複度を保存する変数であり，s, t は，カウンタ変数である．このアルゴリズムは，ノードセットが順序集合であることを利用して比較演算を減少させている．

3.2　XPath 式の配置アルゴリズム

ここでは，関数 PathLap を用いて下位フィルタサーバへ追加された XPath 式をどのように配置していくかを述べていく．上位フィルタサーバを u であるとする．下位フィルタサーバの集合を $F = \{f_i \in \mathcal{N} | i = 1, ..., m\}$ で表す．u は，下位フィルタサーバの全ての XPath 式を保持している．よって，u は下位フィルタサーバの全ての特徴量を持つ事が可能となり，新しく XPath 式が追加されたときにどの下位フィルタサーバに XPath 式を追加するかを決定する事ができる．

本研究で用いた手法は，greedy アルゴリズムであって新たな XPath 式が追加されたとき，その追加による配信コストの増加が最も小さい下位フィルタサーバを選択する．既に登録されている XPath 式の移動を含めた最適化は行わない．図 12 には，あるユーザが新たな XPath 式を追加するときの模式図が示されている．まず，u に XPath 式を追加する．u は追加された XPath 式に対する特徴量を XML ドキュメントに対して算出し，$\{f_i | 1 \leqq i \leqq m\}$ の中から最も重複度が少ないものを選択，XPath 式を追加する．

フィルタサーバ $f_i (1 \leqq i \leqq m)$ が処理している XPath 式の集合を $X^{f_i} := \{x_j | 1 \leqq j \leqq |X^{f_i}|\}$ とする．但し，x_j は X^{f_i} に含まれる XPath 式を示している．$|X^{f_i}|$ は，X^{f_i} の要素数を示すものとする．上位フィルタサーバは，全ての XPath 式を持つので XPath 配置アルゴリズムは，上位フィルタサーバで計算可能である．

今，追加される XPath 式を p_{new} とする．この時，p_{new} に対する特徴量は $a^{p_{new}}$ と書く事ができる．各フィルタサーバと XPath 式 p_{new} の重複度を演算する関数 AllLap:$\mathcal{N} \times \Re^{\epsilon \times 3} \rightarrow \Re$ は，式 (2) を用いて次のように定義できる．

$$\mathrm{AllLap}(f_i, a^{p_{new}}) := \sum_{x_j \in X^{f_i}} \mathrm{PathLap}(a^{x_j}, a^{p_{new}}). \qquad (3)$$

この関数は，フィルタサーバ f_i に登録されている XPath 式全てと追加される XPath 式 p_{new} 間の重複度の和を返すものである．この関数を用いて，重複度の最も小さいフィルタサーバを選択する select:$\mathcal{N}^m \times \mathcal{N} \rightarrow \Re$ を式 (3) を用いて次のように定義する．

$$\mathrm{select}(F, p_{new}) := \underset{1 \leqq i \leqq m}{\mathrm{argmin}} \{\mathrm{AllLap}(f_i, a^{p_{new}})\}. \qquad (4)$$

この関数は，引数として下位フィルタサーバの集合と追加 XPath 式 p_{new} をとり，最も重複度が小さいフィルタサーバを返す関数である．

次に，下位フィルタサーバの XML ドキュメント d_k に対する負荷を示す関数 weight:$\mathcal{N} \rightarrow \Re$ を次のように定義する．この関数は，引数としてフィルタサーバ $f_i (1 \leqq i \leqq m)$ を入力すると f_i の総配信量 (に比例する値) を返す．

$$\mathrm{weight}(f_i) := \sum_{x_j \in X^{f_i}} \sum_{k=1}^{\epsilon} a_{k3}^{x_j}. \qquad (5)$$

次に，式 (5) を用いて，下位フィルタサーバの選択関数 select2:$\mathcal{N}^m \rightarrow \Re$ を下記のように定義する．

$$\mathrm{select2}(F) := \underset{1 \leqq i \leqq m}{\mathrm{argmin}} \{\mathrm{weight}(f_i)\}.$$

この関数は，追加 XPath に関係なく下位フィルタサーバの持つ負荷のみを考慮して，最も負荷の低い下位フィルタサーバを返す．

最後に，XML ドキュメント単位での偏りを減少させるための条件を取り入れたアルゴリズムを図 13 に示す．まず，追加 XPath 式と 既に登録されてる XPath 式に対する特徴量はす

Step 0(初期化)
initialize f;
$\forall f_i \in F, w^{f_i} = \mathrm{weight}(f_i)$;
$w^{p_{new}} = \sum_{k=1}^{\epsilon} a_{k3}^{p_{new}}$;

Step 1(下位フィルタサーバの選択)
if $\left(\max_{1 \leqq i \leqq m}(w^{f_i}) \leqq \alpha * \min_{1 \leqq i \leqq m}(w^{f_i})\right)$
$f = \mathrm{select}(F, p_{new})$; goto Step 2;
else
if $\left(\beta * w^{p_{new}} \leqq \min_{1 \leqq i \leqq m}(w^{f_i})\right)$
goto Step 1.1;
else
f=select2(F); goto Step 2;

Step 1.1(フィルタサーバと追加 XPath 式の配信コストを平均化)
$\forall f_i \in F, w^{f_i} = w^{f_i}/\beta$;
goto Step 1;

Step 2(選択されたフィルタサーバへ追加)
$f += p_{new}$; $w^f += w^{p_{new}}$;

図 13 XPath 式配置アルゴリズム

でに特徴量算出アルゴリズムによって取得されているものとする．選択された下位フィルタサーバを格納するための変数 f を用意する．追加される XPath 式を p_{new} とする．また，下位フィルタサーバの集合を $F := \{f_i | 1 \leqq i \leqq m\}$ とする．フィルタサーバ f_i の配信コストを記憶するための変数を w^{f_i} とする．また，定数 $\alpha, \beta > 1$ を用意する．また，XPath 式 p_{new} に対する配信コストを $w^{p_{new}}$ とおく．

Step 0 では，$f, w^{f_i}, w^{p_{new}}$ の初期化を行う．Step 1 では，下位フィルタサーバ間の総配信量が乖離しないようにするため，フィルタサーバ間の配信コストの差が小さい場合には，重複度による選択 (select 関数) を行い，逆に大きい場合には，フィルタサーバの配信コストによる選択 (select2 関数) を行うことを示している．Step 1.1 では，Step 1 においてフィルタサーバの重みによる選択が行われるときに追加 XPath に対する配信コスト $w^{p_{new}}$ とフィルタサーバの配信コストが乖離している場合には平均化を行うことを示している．これは，Step 1 の if 文においてフィルタサーバの最大値と最小値が乖離しているかどうかを最小値の定数倍と最大値の比較を行ったために必要となる．もし，Step 1.1 がないとフィルタサーバの配信コストが大きな値になったときに，追加 XPath 式によるフィルタサーバの配信コストへの影響が少ないために，比較的最大値と最小値が離れていても重複度による配置のみ，つまり，select 関数による配置が行われてしまい，配信コストの分散化がされない．Step 2 では，選択されたフィルタサーバへ p_{new} を追加するとともに，p_{new} の配信コストを選択されたフィルタサーバへ加える．こうすることで，複数の XPath 式が追加されたときにも対応可能となる．

4. 数 値 実 験

この章では，3.2 節で述べたアルゴリズムを検証するために行った数値実験の結果を示し，その考察を行う．サーバ及びユーザプログラムは，GNU C++(ver.3.2) で実装し，-O3 オプションを利用した．ハードウェアは，サーバプログラムは，RedHat8.0(CPU Pentium-IV Xeon 2.4GHz × 2, mem 6GB) で実行した．また，ユーザプログラムは，Solaris8(Sun CPU UltraSPARC-III 900MHz × 2,mem 4GB) 上で実行した．実データに対するアルゴリズムの有効性を確認するために，DBLP [10] で提供されているデータを用いて $\{d_k | 1 \leqq k \leqq 20\}$ を作成した．$d_k (1 \leqq k \leqq 20)$ に対して XPath 式を作成し，$|P| = 4000$ である．下位フィルタサーバの台数が増えたとしても提案手法は適用できるため，下位フィルタサーバの数は 4 つとした．d_k は，それぞれ 100 KB 程度の大きさである．$\alpha = 2, \beta = 2$ とした．実験方法は，XML ドキュメントが与えられたときに，XPath 式を 200 本ずつ追加する．それを 20 回続ける事で，用意された XPath 式が全て下位フィルタサーバへ登録される事になる．d_k の入力後に登録された XPath 式が d_{k+1} 以降の XML ドキュメントに対して分散して配置されていれば配信時間がほぼ同じとなることが予測される．今後示す図の横軸は，全てこの XML の番号を示している．ラウンドロビン方式は，XPath 式が追加された順番に下位フィルタサーバへ追加するものである．

まず，各フィルタサーバの配信時間の最大値と最小値の差を示したのが，図 14 である．ラウンドロビンでは，18 秒ほどの差が付いているにもかかわらず，提案手法では，0.5 秒程度で収まっている．提案手法を用いれば，ユーザの XPath 式がどのフィルタサーバに登録されていたとしても最も早く配信されているユーザに比べても 0.5 秒以内に XML データが配信される事になる．

XML データ d_1〜d_{20} までの累積した場合，どの程度差が生じるかについて示したのが図 15 である．この図から，ラウンドロビン方式は XML d_{20} を配信完了するまでに 160 秒以上かかっているが，提案方式では，50 秒程度で配信が完了している．このことから，フィルタサーバ 4 台を用いた分散ではラウンドロビンに比べて 1/3 の時間で配信可能である事が分かる．また，図 15 のグラフを二次関数で近似した結果の微分をとったものはそれぞれ下記のようになった．

- ラウンドロビン：$2.0 \times 10^{-5}x - 1.0 \times 10^{-2}$
- 提案手法：$1.0 \times 10^{-5}x - 4.0 \times 10^{-3}$

提案手法の配信時間の増加率がラウンドロビンに比べて 1/2 である事が分かる．このことから，XPath 式が追加されればされるほど，提案手法の配信時間とラウンドロビンの配信時間の差は大きくなる．

最後に図 16 は，XPath 式が追加されるときのフィルタサーバへの配置を決定する際に要する時間を示している．各フィルタサーバに配置されている XPath 式がある程度大きくなっても一本あたりの追加時間が 0.01 秒程度である事が分かる．本研究で提案するアルゴリズムが，極めて高速である事を示して

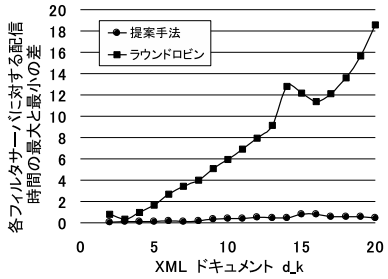

図 14　各フィルタサーバに対する配信時間の最大と最小の差

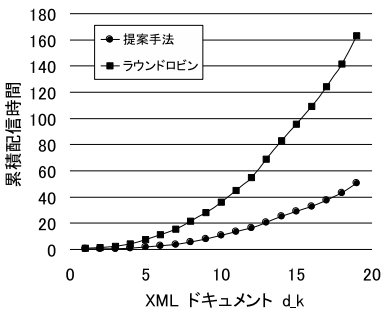

図 15　累積配信時間の比較

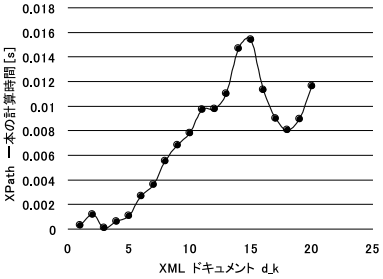

図 16　XPath 式一本あたりの配置処理のための計算時間

いる．

5. 配信コストの加重平均

ここまでの議論では，入力されるひとつの XML ドキュメント $\{d_k\}$ のみから特徴量を計算していた．しかし，XML が変化すれば，同じ XPath に対する重複度も変化する．実際には XML ドキュメント d_k の前には，$d_1 \ldots d_{k-1}$ の XML ドキュメントが入力されている．それらの XML ドキュメントから得られた情報を d_k に対して得られる XPath 式の特徴量に反映させるために加重平均を利用して特徴量を算出する．$a_{j3}^{p_i}(d_k)'$ は，$d_1, ..., d_{k-1}$ に関する情報も取り入れた配信コストであるとすれば，

$$a_{j3}^{p_i}(d_k)' := \frac{a_{j3}^{p_i}(d_k) + \gamma a_{j3}^{p_i}(d_{k-1})}{1+\gamma},$$

と書くことが出来る．$\gamma \in \Re$ は，$d_1, ..., d_{k-1}$ の影響力を示す定数である．γ が小さくなるに従い，古い XML の影響力が小さくなる．このような配信コストを特徴量取得アルゴリズムに適用することで，XPath 式配置アルゴリズムを XML データの変化に対応させることが可能となる．

6. まとめと今後の課題

本研究では，まず，XPath エンジンを用いた XML 配信システムを構築した．その結果，XPath 処理コストに比べて配信コストの割合が高い事が分かった．大量のユーザをリアルタイムに処理するためには，フィルタサーバを並列分散処理させる必要がある．この時，配信コストを適切に分散させなければならないという課題が生じた．実データに対する XPath 式の特徴量抽出という新技術を開発し，それを利用して下位フィルタサーバ選択アルゴリズムを提案した．数値実験の結果から，実データに対して提案手法が有効である事が示され，大規模ユーザへ適用可能であるという見込みを得たものと考える．

今後の課題としては，負荷分散アーキテクチャにプリフィルタを取り入れた多段処理の高速化があげられる．プリフィルタ処理は，ユーザが全ての配信 XML データから 1%未満の情報量を取得するような XPath 式を登録する場合に有効である．なぜなら，図 3 の楕円で囲まれた部分の左側は，急激に下がっているがこれは，配信コストよりも XPath フィルタコストのほうが大きい事を示している．上位フィルタサーバから配信する XML データを下位フィルタサーバの必要な XML データのみとすることで，下位フィルタサーバの XPath 式フィルタ処理を減らす事が可能となるからである．また，今回の提案技術とプリフィルタ処理はトレードオフの関係があるので，多段システム全体としての最適化を目指したい．

文　献

[1] “http://www.informatik.uni-trier.de/~ley/db/xpath”. Vis-Analysis Systems Technologies at The University of Alabama in Huntsville.

[2] “http://www.naa.org”. Newspaper Association of America(NAA).

[3] “http://www.newsml.org/pages/index.php”. International Press Telecommunications Council(IPTC).

[4] A. C. Snoeren and K. Conley and D. K. Gifford. “Mesh Based Content Routing using XML”. *SOSP*, 2001.

[5] A. K. Gupta, D. Suciu. “Stream Processing of XPath Queries with Predicates”. *SIGMOD*, 2003.

[6] C. Y. Chan, P. Felber, M. N. Garofalakis, R. Rastogi. “Efficient Filtering of XML Documents with XPath Expressions”. *ICDE*, 2002.

[7] J. Pereira, F. Fabret, H.-A. Jacobsen, F. Llirbat, D. Shasha. “WebFilter: A High-throughput XML-based Publish and Subscribe System”. *VLDB*, 2001.

[8] L. Golab and M. T. Ozsu. “Issues in data stream management”. *SIGMOD Record*, 2003.

[9] M. Altinel and M. J. Franklin. “Efficient Filtering of XML Documents for Selective Dissemination of Information”. *VLDB*, 2000.

[10] M. Ley. “http://www.informatik.uni-trier.de/~ley/db/xpath”.

[11] F. Neven and T. Schwentick. “XPath containment in the presence of disjunction, DTDs, and variables,”. *ICDT*, 2003.

[12] P. T. Eugster and P. Felber and R. Guerraoui and A.-M. Kermarrec. “The many faces of publish/subscribe”. *ACM Comput. Surv.*, 2003.

[13] R. Chand and P. Felber. “Scalable Protocol for Content-Based Routing in Overlay Networks”. *NCA*, 2003.

[14] R. Shah, R. Jain and F. Anjum. “Efficient Dissemination of Personalized Information Using Content-based Multicast”. *INFOCOM*, 2002.

[15] T. J. Green and G. Miklau and M. Onizuka and D. Suciu. “Processing XML streams with deterministic automata”. *ICDT*, 2003.

[16] Y. Diao, M. J. Franklin. “High-Performance XML Filtering: An Overview of YFilter”. *IEEE Data Eng. Bull.*, 2003.

[17] 富田悦次, 横森貴（編）. オートマトン・言語理論. 森北出版株式会社, 1992.