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

Load Sharing for XML Streams Filtering System with XPath Expression Features

内山 寛之♥ 鬼塚 真♥ 芳西 崇♥

Hiroyuki UCHIYAMA Makoto ONIZUKA
Takashi HONISHI

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配信負荷を均等に分散することが不可欠である．本稿では，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 XPath 式 4000 本を割り当てた時，提案手法により XPath 式が適切に配置され，4 台の配信時間の差を約 0.8 秒以下に抑えられることを確認した．

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 experimental results show that difference of publishing time of each filter server is smaller than 0.8s by proposed method under assigning 4000 XPath expressions to four filter servers.

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[1,2]が注目を集めている. 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というメリットがある．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[3~7]．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(以後FSとする)を用いてXMLデータを配信する場合に，配信にかかるコストが複数のFSへ均等に分散するようにXPath式の追加割当を高速に行うための手法を提案する．

2. 現状の課題

2.1 システム概要

本研究で用いるXPathエンジン(XPE)[8]は, SAXを用いてストリーミングXMLデータを処理する．この時, 複数のXPath式を等価な決定性有限オートマトン(DFA)に変換することで，フィルタリングをオートマトンの受理問題に帰着させる. 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．SAXベースのXPEには, 非決定性オートマトン(NFA)によるもの[1]とDFAによるもの[8]がある. 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．

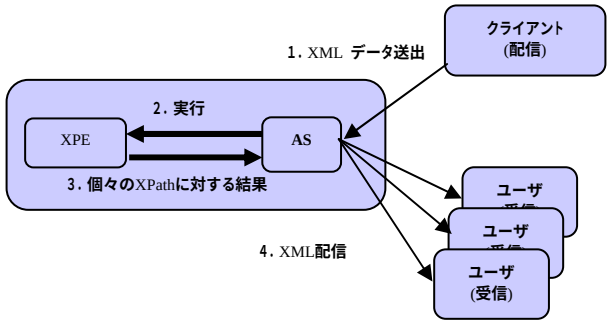

図１ フィルタサーバアーキテクチャ

Fig1. Architecture of filter server

2.2 フィルタサーバのアーキテクチャ

図1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(AS)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，ASは既に登録されているXPath式に対するフィルタリングをXPEに要求する(図中2)．XPE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ASへ伝える(図中3)．AS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, FS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．

総処理時間=配信時間＋フィルタ時間

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

♥ 正会員 日本電信電話株式会社 NTTサイバースペース研究所 {uchiyama.hiroyuki, onizuka,makoto honishi.takashi }@lab.ntt.co.jp

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2である．楕円で囲まれた部分において(フィルタ率が約1%以上の範囲)，配信処理がXPEによる処理に比べて非常に大きい事が分かる．

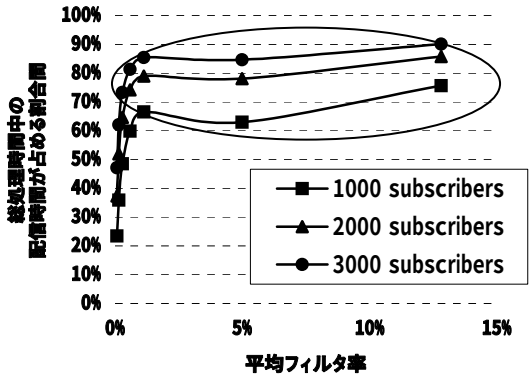

図 2　フィルタ率と配信時間の占める割合
Fig2. Ratio of transfer time to total publishing time

今回採用したDFAベースのXPEでは，XPath式の数が増加してもフィルタ処理速度が一定であるため図2のような結果となった．また，数万ユーザを超える場合には複数のFSによる分散処理が必要であることが分かっている[9]．

2.4 配信コスト分散の課題

FSの分散処理においては，入力XMLデータに対して，配信コストが均等に分散していることが不可欠である．例えば，4000件のXPath式を4台のFSへ均等に分けることを考える．一般に，配信量はXMLデータとXPath式によって変化する．XMLデータ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が無い)．もしも，それらがひとつのFSに登録されているならば，4台のFSを用意しても負荷分散している事にはならない．

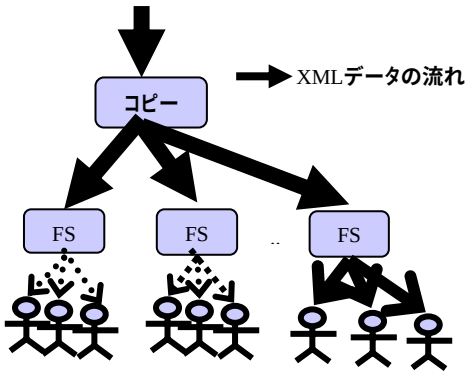

図 3 配信処理が偏った場合
Fig 3. Unequal load sharing case

図3は，一部のFSに配信処理が偏った場合を示している．全体の処理は配信コストが大きいFSの性能劣化に影響を受ける事が分かる．本研究では，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3 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負荷分散手法を提案する．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FSに与える影響を予測し，どのFSへ追加XPath式を配置するかを決定する．提案手法は，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．

3. 配信コストの分散手法

本章では，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，FS の配信コストを算出するための関数を定義する．最後に，XPath 式割当アルゴリズムを提案する．割当アルゴリズムは，XML データを配信する時に，一部の FS へ配信コストが集中することを避けることを目的とする．

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 FS 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 $D=\{d_k \mid 1\le k\le\infty\}$ とおく．添え字 k は，特徴量は，ある特定の XML ドキュメント d_k に対して得られるが，明らかな場合は省略する．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 $P:=\{p_i \mid 1\le i\le|P|\}$ 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．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，$a_j^{p_i}(d_k)\in\Re^3(1\le j\le\varepsilon)$ 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．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 $a^{p_i}(d_k)\in\Re^{3\times\varepsilon}$ とし，以下のように定義する事ができる．

$$a^{p_i}(d_k)=\begin{pmatrix}a_1^{p_i}\\a_2^{p_i}\\\vdots\\a_\varepsilon^{p_i}\end{pmatrix}=\begin{pmatrix}a_{11}^{p_i}&a_{12}^{p_i}&a_{13}^{p_i}\\a_{21}^{p_i}&a_{22}^{p_i}&a_{23}^{p_i}\\\vdots&\vdots&\vdots\\a_{\varepsilon1}^{p_i}&a_{\varepsilon2}^{p_i}&a_{\varepsilon3}^{p_i}\end{pmatrix}$$

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 により変化する変数であり，フィルタリングが行われると同時に得る事ができる．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 Algorithm1 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．

Algorighm1

1.startDocument()
　g; initialize S;
2.startElement(len)
　$++g$; $\forall p_i \in S$; $a_{\varepsilon 3}^{p_i} += len$;
3.endElement()
　$++g$;
4.characters(len)
　$\forall p_i \in S$; $a_{\varepsilon 3}^{p_i} += len$;
5.endDocument()
6.startContext(p_i)
　$++\varepsilon$; $a_{\varepsilon 1}^{p_i} = g$; S=S∪{p$_i$};
7.endContext(p_i)
　$a_{\varepsilon 2}^{p_i} = g$; S=S-{p$_i$};

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．特徴量ベクトル $a_s^{p_i}, a_t^{p_j}$ を引数としてとり，重複の度合いを返す関数 NodeLap を下記のように定義する．

$$\mathrm{Nodelap}(a_s^{p_i}, a_t^{p_j}) = \begin{cases} a_{t3}^{p_i} & \text{if} \quad a_{s1}^{p_i} \le a_{t1}^{p_j} \wedge a_{t2}^{p_j} \le a_{s2}^{p_i} \\ a_{s3}^{p_i} & \text{if} \quad a_{t1}^{p_j} \le a_{s1}^{p_i} \wedge a_{s2}^{p_i} \le a_{t2}^{p_j} \\ 0 & \text{(others)} \end{cases}.$$

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_i} 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 を以下のように定義する．

$$\mathrm{PathLap}(a^{p_i}, a^{p_j}) = \sum_{s=1}^{\varepsilon_i} \sum_{t=1}^{\varepsilon_j} \mathrm{Nodelap}(a_s^{p_i}, a_t^{p_j}) \qquad \cdots(1)$$

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．そのまま演算すればこの計算量は $O(\varepsilon_i \times \varepsilon_j)$ となる．そこで，ノードセットの重複度を $O(\varepsilon_i + \varepsilon_j)$ で算出するアルゴリズムを Algorithm2 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 式配置アルゴリズム

図 4 は，本方式のアーキテクチャを示している．図中のコピー＆コントロールサーバを CCS とする．割当アルゴリズムは CCS 上で実行される．CCS は，FS の全ての XPath 式を保持しており，FS の全ての XPath 式に対する特徴量を算出可能である．どの FS に XPath 式を割り当てるかを決定する．本研究で用いた手法は，XPath 式の追加に関する greedy アルゴリズムであって 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 FS を選択する．FS の集合を $F = \{f_i \mid i = 1,...,m\}$ で表す．

Algorithm2

Step0
　$rw = s = t = 0$;
Step1
　if($a_{s1}^{p_i} \le a_{t1}^{p_j} \wedge a_{t2}^{p_j} \le a_{s2}^{p_i}$)　$rw += a_{t3}^{p_j}; ++t$;
　else if ($a_{t1}^{p_j} \le a_{s1}^{p_i} \wedge a_{s2}^{p_i} \le a_{t2}^{p_j}$)　$rw += a_{s3}^{p_i}; ++s$;
　else if($a_{t2}^{p_i} \le a_{s1}^{p_j}$)　$++t$;
　else if($a_{s1}^{p_j} \le a_{t2}^{p_i}$)　$++s$;
Step2
　if($s > \varepsilon_i \vee t > \varepsilon_j$)　return rw;
　else　goto Step1;

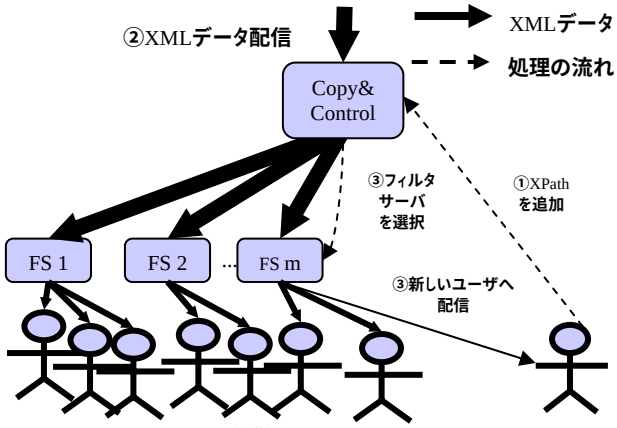

図 4 分散アーキテクチャ
fig 4. Multiple architecture

$f_i (1 \le i \le m)$ が処理している XPath 式の集合を $X^{f_i} := \{x_j \mid 1 \le j \le |X^{f_i}|\}$ とする．x_j は X^{f_i} に含まれる XPath 式を示している．今，追加される XPath 式を p_{new} とする．$a^{p_{new}}$ を p_{new} の特徴量とする．各 FS に既に割り当てられた XPath 式全てと p_{new} の結果の重複度が最も小さい FS を返す関数 select を次のように定義する．

$$\mathrm{select}(F, p_{new}) := \arg\min_{1 \le i \le m} \{ \sum_{x_j \in X^{f_i}} \mathrm{PathLap}(a^{x_j}, a^{p_{new}}) \}$$

次に，d_k が入力された時に最も小さい配信量を持つ FS を求める関数 select2 を次のように定義する．

$$\mathrm{select2}(F) := \arg\min_{1 \le i \le m} (\sum_{x_j \in X^{f_i}} \sum_{k=1}^{\varepsilon} a_{k3}^{x_j})$$

この関数は，追加 XPath に関係なく FS の持つ XPath 式の負荷のみを考慮して，最も負荷の低い FS を返す．次に，追加 XPath 式割り当てアルゴリズムを以下に示す．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でに Algorithm2 によって取得されているものとする．選択された FS 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．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．

Step 1 では，FS 間の総配信量が乖離しないようにするため，FS 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，FS の配信コスト

Algorithm3

Step0

initialize f; $\forall f_i \in F, w^{f_i} = \sum_{x_j \in X^{f_i}} \sum_{k=1}^{\varepsilon} a_{k3}^{x_j}$; $w^{p_{new}} = \sum_{k=1}^{\varepsilon} a_{k3}^{p_{new}}$;

Step1

if($\max_{1 \le i \le m}(w^{f_i}) \le \alpha \min_{1 \le i \le m}(w^{f_i})$)

f = select(F, p_{new}); goto Step 2;

else

if($\beta w^{p_{new}} \le \min_{1 \le i \le m}(w^{f_i})$) goto Step 1.1;

else f =select2(F); goto Step2;

Step1.1

$\forall f_i \in F, w^{f_i} = w^{f_i} / \beta$; goto Step1;

Step2

Add p_{new} to f; $w^f += w^{p_{new}}$;

による選択(select2 関数)を行うことを示している．Step1.1 は，Step1 において FS の重みによる選択が行われる時に追加 XPath に対する配信コスト $w^{p_{new}}$ と FS の配信コストが乖離している場合に平均化することを示している．

4. 数値実験

この章では，数値実験の結果を示す．サーバ及びユーザプログラムは，GNU C++(ver.3.2)で実装し，サーバプログラムは，RedHat8.0(CPU Pentium4 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 から $d_k (1 \le k \le 20)$ を作成し，XPath 式の数|P|は，4000 とした．FS の数 m は 4 とした．d_k は，それぞれ 100KB 程度の大きさである．α=β=2 とした．実験方法は，XML ドキュメントが入力されたときに，XPath 式を 200 本ずつ追加する．交互に 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} 以降に負荷分散されていれば各 FS の配信時間がほぼ同じとなることが予測される．ラウンドロビン方式では，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．図 5 は，追加 XPath 式に対して，4 台の配信時間の差がどのように変化するかを示すものである．ラウンドロビンでは，XPath 式の本数が増加するに従い，FS 間の配信時間が大きく開いており，負荷分散できてないことが分かる．一方，提案手法では，FS 間に配信時間の差はほとんど無く，負荷分散が出来ていることがわかる．また，Algorithm3 を実行するにあたり要した時間は，XPath 式 4000 本が追加された場合でも 1 本の XPath 式を追加する時にかかる時間は，高々0.01 秒程度であり[9]，割り当てに要する計算コストは低いといえる．

5. まとめ

本研究では，過去の XML データに対して XPath 式の特徴量をとらえ，追加された XPath 式を配信負荷が均等になるように複数の FS へ割り当てる手法を提案した．

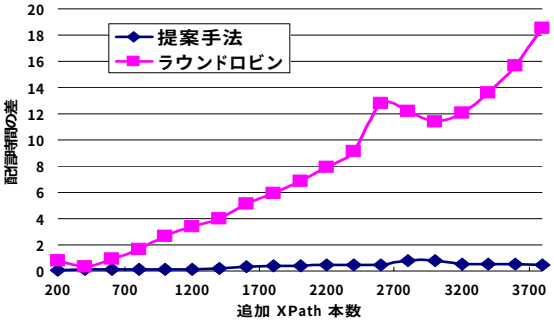

図 5 各 FS に対する配信時間の最大と最小の差
Fig 5. Difference for each FS's publishing time

[文献]

[1] M. Altinel and M. J. Franklin, "Efficient Filtering of XML Documents for Selective Dissemination of Information", In *Proc. VLDB*, 2000.

[2] J. Pereira, F. Fabret, H.-A. Jacobsen, F. Llirbat and D. Shasha, "WebFilter: A High-throughput XML-based Publish and Subscribe System", In *Proc. VLDB*, 2000.

[3] R. Chand and P. Felber, "Scalable Protocol for Content-Based Routing in Overlay Networks", *Second IEEE International Symposium on Network Computing and Applications(NCA)*, 2003.

[4] A. C. Snoeren, K. Conley and D. K. Gifford, "Mesh Based Content Routing using XML", In *Proc. ACM Symposium on Operating Systems Principles*, 2001.

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

[6] P. T. Eugster, P. Felber, R. Guerraoui and A.-M. Kermarrec, "The many faces of publish/subscribe", *ACM Computing Surveys*, 2003.

[7] L. Golab and M. T. Ozsu, "Issues in data stream management", *SIGMOD Record*, 2003.

[8] T. J. Green, G. Miklau, M. Onizuka and D. Suciu, "Processing XML streams with deterministic automata", In *Proc. ICDT*, 2003.

[9] 内山 寛之，鬼塚 真，芳西 崇, "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", *DEWS*, 2004.

内山 寛之 Hiroyuki UCHIYAMA

2000 年阪大・基礎工・システム卒．2002 年同大大学院・基礎工・修士課程修了．同年 NTT 入社．XML ストリーム処理の研究に従事．日本データベース学会正会員．

鬼塚 真 Makoto ONIZUKA

1991 年東工大・工・情工卒．同年 NTT 入社．
XML 変換，XML ストリーム処理，データベースの研究開発に従事．2000 年 ワシントン州立大学 客員研究員．ACM，情報処理学会各会員．

芳西 崇 Takashi HONISHI

1983 年東工大大学院・工・修士課程修了．同年，日本電信電話公社（現 NTT）入社．以後，DBMS，オペレーションシステム，メタデータサービス、オープンソースソフトウェアの研究開発等に従事．現在，NTT サイバースペース研究所主幹研究員．IEEE，情報処理学会各会員．