分散 XML に対する XSLT 実行手法

水本　弘貴†　　鈴木　伸崇†

† 筑波大学　〒 305–8550 茨城県つくば市春日 1-2
E-mail: †s0911654@u.tsukuba.ac.jp, ††nszuki@slis.tsukuba.ac.jp

あらまし　近年，複数サーバに XML を分散させて管理する分散 XML の必要性が増加している．分散 XML に関しては，XPath の実行手法は提案されているが，XSLT の実行手法はこれまで提案されていない．本稿では分散 XML の配置状況に応じて各サーバ上で並行して XSLT を実行することで，サーバでの処理を分散させ全体での実行効率を向上させる手法を提案する．XSLT は実行を行うために祖先ノードの情報が必要となる．他サーバに存在する祖先ノードの情報を取得することなく並行実行を可能にするために，配置された XML のルートノードに対し，適用可能なテンプレート全てをスレッドを用い並行に実行した．提案手法に対し評価実験を行った結果，既存手法に対し十分な実行時間の短縮が認められた．

キーワード　分散 XML，XSLT，効率化

1. はじめに

XML(Extensible Markup Language) は Web 上の標準的なデータ形式として幅広く用いられている．近年，一つの組織が扱うデータ量が大幅に増加していること，更にまた地理的または管理上の要因から XML データを複数の断片に分割して管理した方が都合が良いことなどから，複数サーバに分散させて管理すること (分散 XML，図 1，2) の必要性が増加している．分散 XML に関しては，XPath(XML Path Language) の実行手法 [3] [2] やスキーマ設計 [6] については研究されているが，XSLT(XML Stylesheet Language Transformations) の実行手法に関する研究は行われていない．

分散 XML に対する一般的な XSLT 実行手法は，複数に分割された XML データを一つのサーバで統合し XSLT を実行するというものである．しかし，XSLT はサイズの大きい XML データに対して実行効率が著しく悪化することが知られている [1]．そのため上記のような実行手法では，分割配置された個々の XML データのサイズが小さいものであっても統合したデータのサイズが大きいので，XSLT の実行効率が著しく悪化してしまう．そこで本稿では，分散 XML に対して効率よく XSLT を実行する実行手法を提案する．提案手法では分散 XML の配置状況に応じて各サーバ上で並行して XSLT を実行することで，特定のサーバへの処理の集中を回避し，処理の効率化を図っている．XSLT はその変換処理をテンプレートという単位で記述する．XML のノードに対してどのテンプレートを適用するかを判断する際，祖先ノードにテンプレートを実行した結果の情報が必要となる．そのため，他のサーバに祖先ノードがあった場合，どのテンプレートを適用するかの判断ができなくなるため XSLT の同時実行が不可能になる．そこで提案手法では，サーバに配置された XML のルートノードに対して適用する可能性のある全てのテンプレートを並列に実行することで，XSLT の同時実行を実現している．

XSLT はチューリング完全であり [4]，XSLT の動作を完全に解析した上で効率の良い分散実行手法を立てることは困難である．そこで本稿では，XSLT の機能を XSLT の基盤となる下降型木変換機 (top-down tree transducer) の機能に制限している．これにより XSLT スタイルシートに記述可能な命令は apply-templates 命令のみとなり，実行される XSLT はあるノードを別のノードに変換するという機能に制限される．また，分散 XML における断片データ間の参照は DTD の外部実体参照宣言を利用し実現している．提案手法を Ruby で実装し評価実験を行った．その結果，提案手法は既存手法と比較して，十分な実行時間の短縮が認められる結果が得られた．

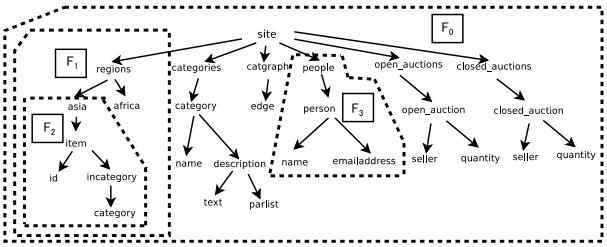

図 1　オークションサイトの XML 例

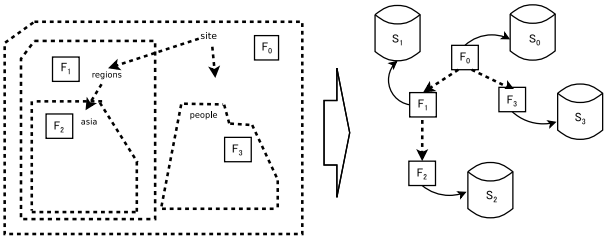

図 2　オークションサイト XML の分割例

2. 諸定義

本章では，XML，XSLT 及び下降型木変換機に関する諸定義を行う．

木，生け垣及び木変換機

我々の実行手法は下降型木変換機に基づいているため，まずは

じめに下降型木変換機に関係する定義を示す．Σ をラベルの有限集合とする．$\mathcal{T}_\Sigma$ は Σ で構成された木の有限集合を表す．$a \in \Sigma$ でラベルされたルートを持つ木が n 個の部分木 $(t_1, \cdots, t_n)$ を持つとき，$a(t_1 \cdots t_n)$ と表す．生け垣は木の有限の系列である．生け垣の有限集合を $\mathcal{H}_\Sigma$ と表す．以降，$t, t_1, t_2, \cdots$ は木を表し，$h, h_1, h_2, \cdots$ は生け垣を表す．$\mathcal{H}_\Sigma(Q)(\mathcal{T}_\Sigma(Q))$ を，Q に属する要素でラベルされた葉ノードを持つことが可能な生け垣 (木) の集合を表すとする．

下降型木変換機 (以下，単に木変換機と書く) は 4 次組 (Q, Σ, q_0, R) で表される [7]．ここで，Q は木変換機の状態を示す変数の有限集合,Σ は入力及び出力で使用するノード名である．$q_0 \in Q$ は木変換機の初期状態を表し，R は $(q, a) \to h$ という形式で記述される変換ルールの有限集合である．ただし，$q \in Q, a \in \Sigma, h \in \mathcal{H}_\Sigma(Q)$ である．$q = q_0$ のとき，h は $\mathcal{T}_\Sigma(Q) \setminus Q$ に属すると仮定する．木変換機の状態は XSLT における mode に対応する．

$Tr = (Q, \Sigma, q_0, R)$ で定義される木変換機による，状態 q での木 t に対する変換は $Tr^q(t)$ として表記され，以下のように帰納的に定義される．

R1: $t = \epsilon$ のとき, $Tr^q(t) := \epsilon$.

R2: $t = a(t_1 \cdots t_n)$ かつ $(q, a) \to h \in R$ が存在するとき, $Tr^q(t)$ は $p \in Q$ でラベルされた全てのノード u を生け垣 $Tr^p(t_1) \cdots Tr^p(t_n)$ に置き換えたものとする．

R3: $(q, a) \to h$ が R に存在しないとき, $Tr^p(t) := \epsilon$.

Tr による t の変換は $Tr(t)$ で表され，$Tr^{q_0}(t)$ で定義される．

例.1 木変換機 $Tr = (Q, \Sigma, p, R)$ を考える．ここで，

$$
\begin{aligned}
Q &= \{p, q\}, \\
\Sigma &= \{a, b, x, y, z\}, \\
R &= \{(p, a) \to x(z), (q, a) \to z(q), \\
&\qquad (p, b) \to x(p\,q), (q, b) \to y(p)\}.
\end{aligned}
$$

T_r は 図 3 で示される XSLT スクリプトに相当する．例として, 変換ルール $(p, a) \to x(z)$ を考える．これは図 3 での最初のテンプレートに相当する．ルール左辺の p はテンプレートの mode 属性に対応し，ルール左辺のラベル a は match 属性に対応する．図 4 における木 t は図 5 における $Tr(t)$ に変換される．

分散 XML

XML 木を $t \in \mathcal{T}_\Sigma$ とし，t の断片である部分木集合を $\mathcal{F}_t \subseteqq \mathcal{T}_\Sigma$ とする．各断片 $F_i \in \mathcal{F}_t$ はそれぞれ異なるサーバに格納されると仮定する．本稿では，その断片が更に断片を持つこと (ネスティング) を許している．

例として，図 1 の $t \in \mathcal{T}_\Sigma$ は 4 つの断片 $\mathcal{F}_t = \{F_0, F_1, F_2, F_3\}$ で構成されている．木 t において，t のルートノードを含む断片を**ルート断片**と呼ぶ．図 2 において，ルート断片は F_0 である．

2 つの断片 F_i, F_j について．F_j の断片内のルートノードを w とし，もし F_i のノード $v \in F_i$ が全断片を統合した XML 木 t において w を子に持っていた場合，断片 F_j は断片 F_i の**子断片**と呼ぶ．F_i における v の子ノード w が F_j に含まれていることを示すために，w の位置に実体参照ノードを挿入する．

例として，図 2 の右図のように各断片 $\mathcal{F}_T$ をサーバ $\mathcal{S} = \{S_0, S_1, S_2, S_3\}$ に配置していく．F_0 は S_0 に F_1 は S_1 に，F_2 は S_2 に F_3 は S_3 に, とそれぞれ 1 つのサーバに 1 つの断片を配置する．ルート断片である F_0 を持つ S_0 はルートサーバと呼ぶ．F_0 のノードである “site” ノードは T において子ノード “regions”,“people” を持つが，断片単体でみた場合，それらの位置に実体参照ノードである “&fragment1;”，“&fragment3;” が挿入されている (図 6)．

全ての断片はサーバに格納される．ルート断片を持つサーバを**ルートサーバ**と呼び，それ以外のサーバを**スレーブサーバ**と呼ぶ．例として，図 2 において S_0 はルートサーバであり，S_1, S_2, S_3 はスレーブサーバである．本稿では，2 つ以上の断片が同じサーバに格納されることはないと仮定する．もし断片がサーバ S に格納され，断片の子断片がサーバ S' に格納されたとき，S' は S の子サーバ (S は S' の親サーバ) と呼ぶ．例

```
<xsl:template match="a" mode="p">
  <x>
    <z/>
  </x>
</xsl:template>

<xsl:template match="a" mode="q">
  <z>
    <xsl:apply-templates mode="q" />
  </z>
</xsl:template>

<xsl:template match="b" mode="p">
  <x>
    <xsl:apply-templates mode="p" />
    <xsl:apply-templates mode="q" />
  </x>
</xsl:template>

<xsl:template match="b" mode="q">
  <y>
    <xsl:apply-templates mode="p" />
  </y>
</xsl:template>
```

図 3　XSLT スクリプト

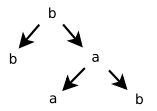

図 4　入 力 木 t

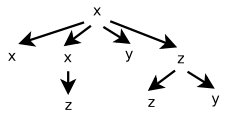

図 5　出力木 $Tr(t)$

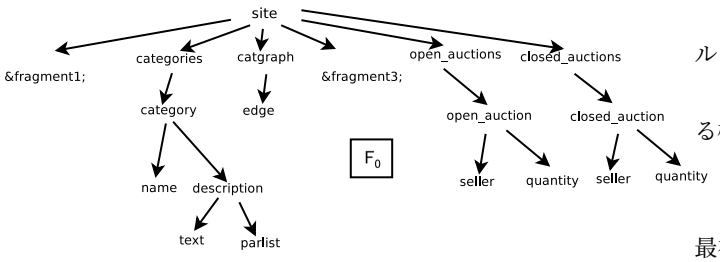

図 6　断　片 F_0

として，図 2 S_0 は 2 つの子サーバ S_1 及び S_3 を持つ．

3. 分散実行戦略

3.1 概　　要

提案手法の実行戦略は，全てのサーバが並行して自身の持つ断片に XSLT 変換を行うというももである．それにより，一つのサーバに処理が集中することを避け，処理の高速化を図る．ただし，この戦略を実現するために，解決しなければならない次の問題がある．

> 断片 F に対して，F に適用するテンプレート (変換ルール) は F の親断片の変換が完了するまで決定することができない．

例として，図 4 のルートノード右側の子ノード a について考える．ノードは a でラベルされているため，このノードには 2 つのテンプレート $(p, a) \to x(z)$，及び，$(q, a) \to z(q)$ が適用される可能性がある．どちらが適用されるかはルートノード b の変換が終わるまでは決定できない．従って，断片 F 及び断片 F に適用し得るルール $r_1, \cdots, r_n$ に対して，提案手法では以下のように変換処理を進めていく．

（1） サーバ S に F が格納されているとする．S はネイティブスレッドを使用し，並列にルール $r_1, \cdots, r_n$ を F に適用させる．それによって S は n 個の変換結果を得る．

（2） S の親サーバ S' において F の親断片への変換処理が完了したとき，F に適用させるべきテンプレートのモードが判明する．S' は判明したモードを S に送信する．

（3） S は F に適用させるべきテンプレートのモードを受信する．S は n 個の変換結果のうちから適切な変換結果を選択し，ルートサーバに変換結果を送信する．

3.2 提 案 手 法

提案手法の詳細について示す．提案手法では 2 つの XSLT プロセッサ Master-XSLT と Slave-XSLT を使用する．Master-XSLT はルートサーバで使用され，以下の機能を持つ．

- ルート断片を変換する機能．
- ルート断片の変換結果から，子サーバに適切なモードを送信する機能．
- ルート断片の変換結果，及び，スレーブサーバより受け取った変換結果をマージする機能．

Slave-XSLT はスレーブサーバで使用され，以下の機能を持つ．

- スレーブサーバに格納された断片 F を変換する機能．
- サーバのネイティブスレッドを使用し，並列に変換ルールを F に適用させる機能．
- F の変換結果により，子サーバに適切なモードを送信する機能．
- F の変換結果をルートサーバに送信する機能．

まずはじめに，Master-XSLT について示す．この手続きは最初にスレーブサーバに木変換機 (XSLT スタイルシート) を送信する (1〜3 行目), そしてルート断片に対して後に示す関数 Transform を使用し，変換を行っていく (5 行目), スレーブサーバから変換結果の断片を受け取り (6 行目)，最後に全ての断片をマージする (7 行目).

Master-XSLT
Input: 木変換機 $T_r = (Q, \Sigma, q_0, R)$ 及び ルート断片 F.
Output: 木 $T \in \mathcal{T}_\Sigma$.

（1） **for** each スレーブサーバ S **do**
（2）　　木変換機 T_r を S に送る.
（3） **end**
（4） $v \leftarrow F$ のルートノード;
（5） $F' \leftarrow \mathrm{Transform}(T_r, F, v, q_0)$;
（6） 全てのスレーブサーバ S_i から変換結果の断片 F_i を受け取るまで待機.
断片 $F_1, \cdots, F_k$ を受信.
（7） F' と $F_1, \cdots, F_k$ をマージし T を作成.
（8） T を返す;

次に, 5 行目で使用した関数 Transform について示す．この関数は与えられた断片を再帰的に変換していく．4 行目から 21 行目は F のノード v' をルートノードとして持つサブ断片の T_r による変換を行っている．4 行目から 13 行目は木変換機の定義におけるルール R2 を適用しており，21 行目はルール R3 を適用している．1 行目から 3 行目には v' が実体参照ノードだった場合の処理であり，断片 $F(v')$ に適用するべきモード q をサーバ $S(v')$ に送信する．$F(v')$ は実体参照ノード v' が指す断片を表し，$S(v')$ は $F(v')$ が格納されているサーバを表す．

Procedure Transform
Input: 木変換機 $T_r = (Q, \Sigma, q_0, R)$, 断片 F, F のコンテキストノード v', 及び, モード q.
Output: モード q での F の変換結果.

（1） **if** v' が実体参照ノード **then**
（2）　　$F(v')$ に適用するべきモード q を求める.
（3）　　$S(v')$ にモード q を送信.
（4） **else if** $(q, v') \to h \in R$ となる h が存在する **then**
（5）　$Q' \leftarrow \{q' \mid q'$ は h の葉ノード $\} \cap Q$;
（6）　**if** v' が子ノード $v_1, \cdots, v_k$ を持つ **then**
（7）　　**for** each $q' \in Q'$ **do**
（8）　　　**for** each v' の子ノード $v_i \in \{v_1, \cdots, v_k\}$ **do**
（9）　　　　$F_i \leftarrow F$ における v_i をルートとする部分木 ;
（10）　　　　$T_i \leftarrow \mathrm{Transform}(T_r, F_i, v_i, q')$;
（11）　　　**end**
（12）　　　h のノード q' を生け垣 $T_1 \cdots T_k$ で置き換える.

（13） **end**
（14） **else**
（15） **for** each $q' \in Q'$ **do**
（16） $q' \leftarrow \epsilon$;
（17） **end**
（18） **end**
（19） F における v' をルートとする部分木を h で置き換える.
（20） **else**
（21） F における v' をルートとする部分木を ϵ で置き換える.
（22） **end**
（23） F を返す;

最後に Slave-XSLT について示す．Slave-XSLT はスレーブサーバ S で実行され，S に格納された断片 F に変換を行う．処理はルートサーバから木変換機を受信してから開始され，スレッドを使用し F に適用する全てのルールを並列に適用させていく (3〜8 行目). 従って一つ以上の変換結果が得られることとなる．その後親サーバから F に適用させるべきモード q を受信するまで待機し (9 行目), 受信しだいルートサーバにモード q に対応する変換後の断片を送信する (１１行目). もし F に実体参照ノード v が含まれるとき，$F(v)$ に適用するべき適切なモードを $S(v)$ に送信する．この処理は 12 行目から 15 行目で行われている．

Procedure Slave-XSLT
Input: サーバ自身の持つ断片 F.
Output: 無し
（1） ルートサーバから木変換機 $T_r = (Q, \Sigma, q_0, R)$ を受信するまで待機.
（2） $v_r \leftarrow F$ のルートノード;
（3） $Modes \leftarrow \{q \mid (q, v_r) \rightarrow h \in R$ となる $h\}$;
（4） **for** each $q \in Modes$ **do**
（5） Thread **start**
（6） $T_q \leftarrow \mathrm{Transform}(T_r, F, v_r, q)$;
（7） Thread **end**
（8） **end**
（9） モード p を親サーバから受信するまで待機.
（10） **if** $T_p \neq null$ **then**
（11） ルートサーバに T_p を送信.
（12） **for** each F 内の実体参照ノード v**do**
（13） v に適用するモード q' を識別する.
（14） q' を $S(v)$ に送信.
（15） **end**
（16） **else**
（17） ルートサーバに ϵ を送信.
（18） **end**

例として，2 つの断片 m 及び f からなる分散 XML について考える (図 7). 木変換機として，例 1 で定義した T_r を使用する．また m が格納されるサーバを S_r, f が格納されるサーバを S_s とする．Master-XSLT が S_r で実行され，T_r をサーバ $S(\&fragment) = S_s$ に送信する．そして m の変換を初期モード p で開始する (図 8 は m の変換結果を示している)．Slave-XSLT が S_s で実行され，T_r を受信次第 f に変換を開始する．この場合，2 つの変換ルール $(p, a) \rightarrow x(z)$，及び，$(q, a) \rightarrow z(q)$ が f のルートノード a に適用される可能性がある．従って，Slave-XSLT はこの 2 つのルールを並列に適用していき，図 9 に示される結果が得られる．Master-XSLT の処理が m の実体参照ノード "$\&fragment$" に移ったとき，適切なモードを $S(\&fragment)$ に送信する．この場合 p 及び q の両モードが $S(\&fragment)$ に送信される．Slave-XSLT が f の変換を完了したあと，変換後の断片を送信する．この場合，２つの断片 (図 9) がルートサーバ S_r に送信される．最後に S_r は３つの断片をマージし出力木 (図 10) が得られる．

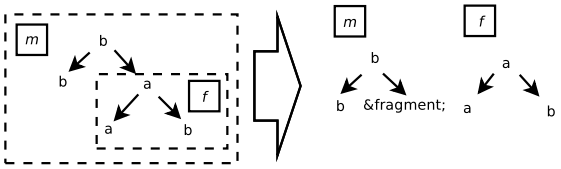

図 7　m と f からなる分散 XML

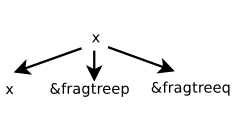

図 8　断片 m の変換結果

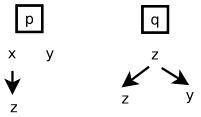

図 9　断片 f の変換結果

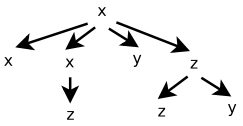

図 10　分散 XML の変換結果

4. 評価実験

本章では，第 3 章の提案手法に対する評価実験について述べる．

4.1 概　　要

まず，第 3 章の XSLT 分散実行の提案手法を Ruby を用いて実装した．次に，XMark [5] と呼ばれるベンチマークソフトで作成した XML データを分割し, 分散 XML データを作成した．そして，このデータに適用する XSLT スタイルシートを複数作成した．これらの分散 XML データとスタイルシートを用いて，提案手法での XSLT 実行時間と, 分散 XML データを統合してから一つのサーバで XSLT を実行するという既存手法での XSLT 実行時間を比較する．なお，XMark は "f オプション" を使用することによりサイズを指定した XML データを作成できる．本評価実験では "f オプション" を 0.5 から 2.5 まで 0.5 刻みで指定し，約 50MB，約 100MB，約 150MB，約 200MB，

約 250MB とサイズの異なる 5 つの XML データを用意した．これらの XML データを分散 XML データとして断片に分割する際には，図 2 の様に “site” をルートとしたルート断片 F_0，“regions” をルートとしたサブ断片 F_1，“asia” をルートとしたサブ断片 F_2，“people” をルートとしたサブ断片 F_3 の 4 つに分割した．各 XML データでの各断片のファイルサイズを表 1 に示す．評価に使用する XSLT スタイルシートは，特性の異なる 3 つのものを用意した．

Sheet-1: 2 つの mode を使用し，XML データのノード全てに少なくとも一度テンプレートを適用する，137 のテンプレートで構成されたスタイルシート．

Sheet-2: 1 つの mode を使用し，XML データのノード全てにテンプレートを適用する，68 のテンプレートで構成されたスタイルシート．

Sheet-3: 2 つの mode を使用し，XML データのいくつかのノードに少なくとも一度テンプレートを適用する，94 のテンプレートで構成されたスタイルシート．

4.2 評価実験の手順

評価実験手順は以下の通りである．

（1） XMark の “f オプション” を使用しサイズの異なる 5 つの XML ファイルを作成

（2） XML ファイルをそれぞれ 4 つの断片に分割し分散 XML を作成

（3） それぞれ 4 つの断片を 4 つのサーバに配置

（4） 配置した分散 XML に対して sheet-1，sheet-2，sheet-3 を提案手法で実行し実行時間を計測

（5） 配置した分散 XML に対して sheet-1，sheet-2，sheet-3 を既存手法で実行し実行時間を計測

（6） (4) の実行時間と (5) の実行時間を比較する

4.3 実 験 環 境

実験には以下の環境のサーバを 4 台使用した．

サーバ：DELL PowerEdge R310
CPU：Xeon X3430 2.4 GHz
メモリ：4G
OS：CentOS 5.8(64bit)
Ruby：1.9.3p194 (x86_64-linux)

4.4 評価実験の結果及び考察

評価実験の結果，断片の合計ファイルサイズごとの実行時間の変化は図 11，図 12，図 13 となった．使用したスタイルシートに関わらず，提案手法は既存手法と比較して 4 倍程度の高速化を達成している．よって，提案手法は既存手法に対して十分な実行時間の短縮が実現されたと考えられる．

5. む す び

本稿では，分散 XML に対する効率の良い XSLT 実行手法を提案し，評価実験を行った．今後の課題としては本稿で課している XSLT に対する制限の緩和が挙げられる．表 2 は XSLT1.0 の命令及び関数の数である，36 の関数及び命令は操作対象がカレントノードや定数であるものであり，これらは現在の下降型木変換機の制限下でも比較的容易に使用可能にすることができると判断したものである．一方，使用困難とある 33 の命令や関数は．操作対象がノードの集合であるような関数及び命令である．これらについては現在の制限下では使用が難しい．制限下でも比較的容易に使用可能にすることができる命令の例として “xsl:text” 命令が挙げられる．これは現在の出力木にテキストノードを追加するという命令であり，入力 XML のノードに

表 1 各断片のサイズ

	各断片のサイズ			
XML サイズ	F_0	F_1	F_2	F_3
約 50MB	23MB	25MB	5.5MB	2.6MB
約 100MB	46MB	50MB	11MB	5.1MB
約 150MB	69MB	75MB	17MB	7.6MB
約 200MB	92MB	100MB	22MB	11MB
約 250MB	115MB	125MB	28MB	13MB

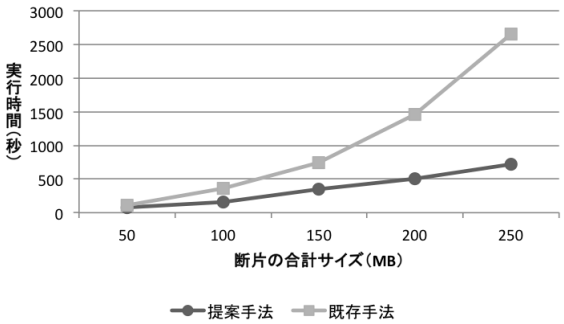

図 11 sheet-1 での実行結果

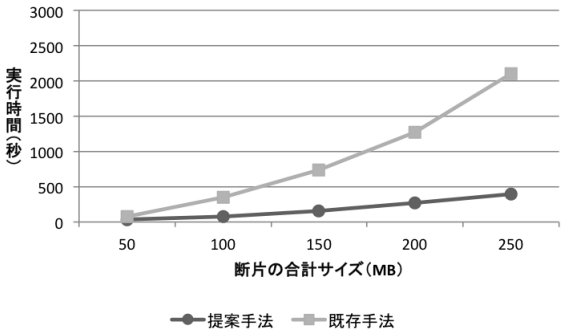

図 12 sheet-2 での実行結果

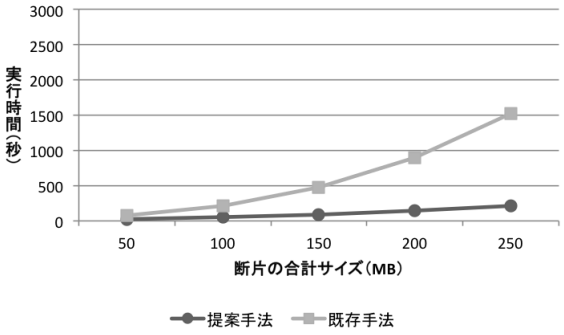

図 13 sheet-3 での実行結果

関係なく使用することが可能である．一方使用困難である命令の例としては “xsl:for-each” 命令が挙げられる．“xsl:for-each” 命令は select 属性で指定した XPath 式で表現されたノード全てに処理を行うというものである． select=“para” とあった場合はコンテキストノードの子ノードである “para” ノード全てに処理を行う．しかし，コンテキストノードの子要素として外部実体参照ノードが存在した場合は，他のサーバのルートノードの情報を得なければ “xsl:for-each” 命令の実行が完了しないため，使用困難だと判断される．

現在の制限下でも使用が可能であると判断した関数及び命令をまず提案手法で使用可能にし，その後，使用不可能であると判断した関数及び命令に関しても，操作対象を子孫ノードのみに制限し，また子断片を持つ断片と持たない断片で制限する関数及び命令を変更することにより，できる限りの関数及び命令を提案手法で使用できるように改善していきたいと考えている．

表 2　XSLT の関数及び命令の数

	命令	関数	合計
使用可能	18	18	36
使用困難	17	16	33

文　　献

[1] Filip Zavoral and Jana Dvorakovam,“ Perfomance of XSLT processors on large data sets ”. Applications of Digital Information and Web Technologies, 2009. ICADIWT ’09. Second International Conference. London, UK, 2009-10-02/08-23. 2009, p.110–115.

[2] Gao Cong, Wenfei Fan, Anastasios Kementsietsidis, Jianzhong Li and Xianmin Liu, ”Partial Evaluation for Distributed XPath QueryProcessing and Beyond”. ACM Transactions on Database Systems. 2012, Volume 37, Article No. 32.

[3] Gao Cong,Wenfei Fan, and Anastasios,“ Distributed Query Evaluation with Performance Guarantees ”. SIGMOD ’07 Proceedings of the 2007 ACM SIGMOD international conference on Management of data . NY, USA,p.509–520.

[4] Kepser S,“ A simple proof for the Turing-completeness of XSLT and XQuery ”. Proceedings of the Extreme Markup Languages, 2004.

[5] Schmidt A, Waas F, Kersten M, Carey M.J, Manolescu I, and Busse R,“ XMark: A benchmark for XML data managemet ” VLDB ’02 Proceedings of the 28th international conferenceon Very Large Data Bases. Hong Kong, China, 2002–08–20/08?23. VLDB, 2002, p.974–985.

[6] Serge Abiteboul, Georg Gottlob, and Marco Manna,“ Distributed XML design ”. PODS ’09 Proceedings of the twenty-eighth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems .NY, USA,p.247–258.

[7] Wim Martens, Frank Neven, ”Typechecking Top-Down Uniform Unranked Tree Transducers ”. Lecture Notes in Computer Science, 2002, Volume 2572/2002, p.64–78.

[8] World Wide Web Consortium (W3C).“ XSL Transformations (XSLT) Version 2.0 ”. (online), available from 〈 http://www.w3.org/TR/xslt20/〉, (accessed 2012-05-30).