社団法人 情報処理学会　研究報告
IPSJ SIG Technical Report

2004－DBS－134　(II)　(95)
2004／7／15

# 静的生成されたデータ集約型Webサイトの効率的な更新手法

石川 恭子 †　有澤 達也 †　遠山 元道 ‡

† 慶應義塾大学大学院 理工学研究科 開放環境科学専攻
‡ 慶應義塾大学 理工学部 情報工学科
E-mail: † sachi@db.ics.keio.ac.jp, ‡ toyama@ics.keio.ac.jp

静的に生成されたデータ集約型 Web サイト、特にデータベースの更新数に比べアクセス数が相対的に大きい Web サイトの効率的な更新手法を提案する。本研究では SQL の拡張言語である SuperSQL を利用する。SuperSQL は一つのクエリの実行によって多段のリンクを持つ階層的なハイパーテキストを一括生成できる。しかし、更新が必要な場合 Web サイト全体を再生成しなければならないという問題があった。そこで、本研究では SuperSQL に sinvoke (static invocation) 関数と FOREACH 句を新たに導入し、SuperSQL クエリを分割し、サイトの部分的な更新を実現する。さらに 3 つの更新アプローチ、及び最適な更新組合せの探索アルゴリズムを提案する。

**キーワード**：Web 、 メンテナンス 、 SuperSQL

# Efficient Maintenance of Statically Generated Data-Intensive Web Site

Kyoko ISHIKAWA †　Tatsuya ARISAWA †　Motomichi TOYAMA ‡

†School of Science for OPEN and Environmental Systems,
Faculty of Science and Technology, Keio University.
‡Department of Information and Computer Science, Faculty of Science and Technology,
Keio University.
E-mail : †sachi@db.ics.keio.ac.jp ‡toyama@ics.keio.ac.jp

A methodology for the maintenance of data-intensive web sites, especially for the web sites, which requests occurs very intensively, is introduced. SuperSQL, an extension of SQL, generates a hierachically structured web site by executing only one query. When update to the base relations occurs, it had to regenerate the whole web site even though we need to update just a part of it. In this paper, we show how to perform the maintenance of web sites partially, by introducing a new function sinvoke(static invocation) and a new clause FOREACH into SuperSQL in order to decompose SuperSQL query. We then intruoduce three update approaches to realize efficient partial maintenance of web sites.

keyword : Web , Maintenance , SuperSQL

# 1 背景

近年 WWW では、データ集約型 Web サイト、すなわちデータベース中の大量のデータから構成される Web サイトが急速に広まっている。その結果、このような Web サイトを構築及び更新する手法がより重要となってきており、様々な手法が提案された [1, 5]。

Web サイトの生成方法には静的生成と動的生成のが 2 種類ある。データベースの更新数に比べアクセス数が相対的に大きい Web サイトの場合、静的に生成した方が有利である。例えば、1998 年の長野オリンピックの試合のデータを提供している Web サイトへのアクセス数は、ピーク時には 1 分間に 10 万 3 千件に昇った [4]。このような場合、アクセスの度に動的に生成するよりも静的に生成しておいた方が効率的である。

静的に生成された Web サイトは、データベースが更新された際、効率的に Web サイトに更新を反映する必要がある。そこで、本研究では静的に生成された Web サイト、特にデータベースの更新数に比べアクセス数が相対的に大きい Web サイトの効率的な更新手法を提案する。

SQL の拡張言語である SuperSQL[7, 8] は、一つのクエリを実行するだけで多段のリンクを持つハイパーテキストを一括に生成できる。これは SuperSQL の特徴の一つであるが、データベースが更新された場合、Web サイト全体を再生成することが原則となる。また、invoke 関数により質問文を副質問に分割できるが、全て動的生成となる。そこで、本研究では SuperSQL に sinvoke (static invocation) 関数と FOREACH 句を新たに導入し、SuperSQL クエリを分割し、サイトの部分的な更新を実現する。さらに更新の偏りを考慮した 3 つの更新アプローチを提案する。

# 2 関連研究

データ集約型 Web サイトの構築及び更新する様々な手法が提案された [1, 5]。本論文の特徴は更新の偏りを考慮した部分更新手法の提案であり、更新の偏りを考慮したものは他にない。Sindoni[5] は Web サイトの部分的な更新手法を提案しているが、提案された手法は 1 度の更新で同じページを何度も再生成してしまうことがある。本論文の提案手法は、1 度の更新で同じページ複数回生成することはない。

データ集約型 Web サイトのメンテナンス (更新) 問題は、実体化ビューのメンテナンス問題と深い関係がある。

ビュー V を更新する場合、ビューの変化分 $\Delta$ V を計算し、$\Delta$ V を V に反映させるという手法が広く用いられている [2]。ビューが Web ページの場合、HTML は複雑であり変化分を直接反映させることはできないが、変化分によって更新が必要なページを判別することはできる [3]。しかし、変化分の計算ではデータベースへアクセスする必要があり、パフォーマンスが低下する。本論文では更新の必要なページを計算する際 $\Delta$ V を計算しないので、データベースへのアクセスは生じない。

更新の際、複数のビューをまとめて扱うことができれば効率的である。Candan ら [3] は同種クエリのバッチ処理方法を示した。Mistry ら [6] は、特別なビューや索引を実体化し、異なるビューのメンテナンスの計算を共有させた。本論文では SuperSQL を利用し、SuperSQL は複数のページを一度に生成することができるので、簡単に複数のビューをまとめて扱うことができる。

# 3 SuperSQL

SuperSQL はデータベースの内容から出版物を生成することを目的として、SQL のターゲットリストを拡張した言語である。SuperSQL の質問文は、SQL の SELECT 句を GENERATE$< media >< TFE >$ の構文を持つ GENERATE 句で置き換えたものである。目的媒体の指定 $< media >$ は、現在では HTML、XML、Excel、LaTeX などが許されているが、本論文では HTML を対象とする。$< TFE >$ はターゲットリストの拡張である Target Form Expression を表し、結合子、反復子などのレイアウト指定演算子を持つ一種の式である。

## 3.1 結合子と反復子

結合子はデータベースから得られたデータをどの方向 (次元) に結合するかを指定する演算子である。

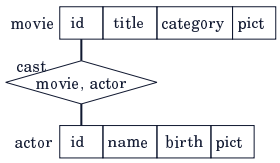


図 1: 映画情報データベースのスキーマ

コン マ (,)、感嘆符 (!)、パーセント記号 (%) の 3 つが、それぞれ第 1〜3 次元に対応する結合子である。属性間をこれらの結合子で区切ることによって、それぞれ水平、垂直、深度方向にレイアウトする。

反復子は指定する方向に、インスタンス数だけ繰りかえして表示する。一対の角括弧 ([ ]) に上記の結合子を添えたものが、それぞれの次元の反復子である。例えば、

[名前, 学籍番号 ]!

は、横方向に連結された学生の氏名、学籍番号を、インスタンス数だけ縦方向に連続的に連結する。

また反復子はただ構造を指定するだけでなく、そのネスト関係によってグルーピングの働きを持つ。例えば、

[研究室, [名前, 学籍番号]!]!

は、研究室ごとにグループ化し、それぞれの学生の氏名と学籍番号の一覧が表示される。

## 3.2 SuperSQL によるハイパーテキスト生成例

映画情報データベース (図 1) から以下に示す SuperSQL クエリ Q0 によって得られる WWW ハイパーテキストを図 2 に示す。

```
Q0: GENERATE HTML
    [m.category%[m.title%
      {{m.title,m.story}!imagefile(m.pict)!
        [imagefile(a.pict),a.name,a.birth]!}]!]!
    FROM movie m, actor a, cast c
    WHERE m.id=c.movie and c.actor=a.id
```

## 3.3 invoke 関数による動的質問呼び出し

SuperSQL には、クエリの中から別のクエリを呼び出すために invoke 関数が用意されている。invoke 関数の構文は以下の通りである。

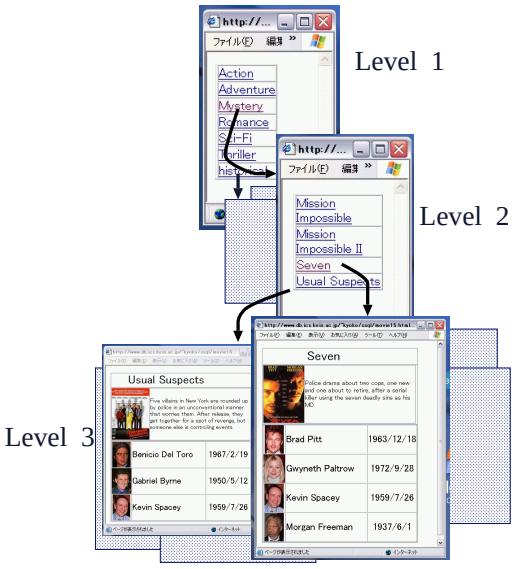


図 2: SuperSQL クエリによるハイパーテキストの生成例

invoke(file, string1=att1, string2=att2, ... )

file は呼び出されるクエリを含むファイルの名前であり、string1=att1, string2=att2, ... はそれぞれ、現文脈における属性 att の値が string に連結され、呼び出されるクエリの WHERE 句に連言で追加される条件である。

クエリ Q1a、Q1b、Q1c は、前節のクエリ Q0 を invoke 関数を用いて 3 つに分割したものであり、それぞれハイパーテキストの 1〜3 レベル目を生成する。Q0 は、ハイパーテキストの 3 レベルを全て静的に一括生成するのに対し、Q1a、Q1b、Q1c は、Q1a が 1 レベル目を生成し、2、3 レベルの 1 ページを Q1b、Q1c がそれぞれ動的に生成する。Q1a の invoke 関数で分かる通り、Q1b には選択されたジャンル (例えば xx) を含む条件 (m.category=”xx”) が動的に追加される。同様に、Q1c には条件 (m.id=yy) が追加される。

```
Q1a: GENERATE HTML
     [m.category%invoke(Q1b,m.category=m.category)]!
     FROM movie m

Q1b: GENERATE HTML
     [m.title%invoke(Q1c,m.id=m.id)]!
     FROM movie
     (WHERE m.category="xx") ; dynamically added
```

```
Q1c: GENERATE HTML
     [{m.title,m.story}!imagefile(m.pict)!
       [imagefile(a.pict),a.name,a.birth]![]!
     FROM movie m, actor a, cast c
     WHERE m.id=c.movie and c.actor=a.id
     (and m.id=yy) ; dynamically added
```

### 3.4 問題点

SuperSQLによるハイパーテキストの生成は、Webサイト全体の一括静的生成とinvoke関数による動的生成の2通りある。しかし、静的生成の場合にはWebページの更新が必要な場合、Webサイト全体を再生成しなければならない。またWebページを動的生成する場合、Webドキュメントを格納するスペース、及び更新するコストを削減でき、データベースが頻繁に更新される場合動的生成は有利である。しかし、データベースがたまにしか更新されない場合、ページアクセスの度にデータベースへもアクセスされ、データベースに大きな負荷がかかるという欠点がある。

そこで本論文では、データベースの更新数に比べアクセス数が相対的に大きいWebサイトの効率的な更新手法を提案する。

## 4 クエリの分割

SuperSQLで静的生成されたWebサイトの部分更新を実現するため、SuperSQLにsinvoke (static invocation) 関数とFOREACH句を導入する。sinvoke関数とFOREACH句は2つのクエリ、sinvoke関数を含むクエリとsinvoke関数で指定されたFOREACH句を含むクエリを関連付ける。ここで、分割されたクエリと分割前のクエリは、それぞれによって生成されたハイパーテキストの見た目が等しいので等価とみなす。

### 4.1 分割の例

前節のクエリQ0を3つのクエリQ2a、Q2b、Q2cに分割する。各クエリは別々に実行することができ、それぞれ第1レベル、第2レベル、第3レベルのページを生成する。

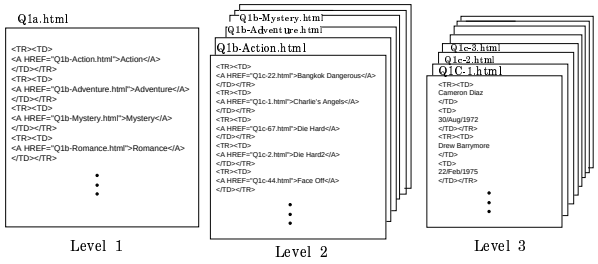


図 3: ハイパーリンクコードの例

```
Q2a: GENERATE HTML
     [m.category%sinvoke(Q2b,m.category)]!
     FROM movie m

Q2b: FOREACH m.category
     GENERATE HTML
     [m.title%sinvoke(Q2c,m.id)]!
     FROM movie m

Q2c: FOREACH m.id
     GENERATE HTML
     [{m.title,m.story}!imagefile(m.pict)!
       [imagefile(a.pict),a.name,a.birth]![]!
     FROM movie m, actor a, cast c
     WHERE m.id=c.movie and c.actor=a.id
```

図3はSuperSQLによって生成されたHTMLファイルのハイパーリンクコードの例である。

### 4.2 Static Invocation (sinvoke) 関数

invoke関数と異なり全てのページは静的に生成される。sinvoke関数を含むクエリを実行して生成されるHTMLファイルには、sinvoke関数で指定されたクエリによって生成されるページのURLが埋め込まれる。sinvoke関数の構文は以下の通りである。

sinvoke (query, attribute)

一つ目の引数は呼び出すSuperSQLクエリの名前であり、二つ目は呼び出されるクエリによって生成されるページを一意に識別できる属性である。データベース設計時に、一意に識別できる属性 (idなど) を作成してあるものとする。

例ではクエリQ2aがクエリQ2bを呼び出し、Q2bによって生成されるページは映画のカテゴリーごとにグルーピングされるのでm.category属性で一意

に識別され。同様に Q2b は Q2c を呼び出し、Q2c によって生成されるページは m.id 属性で一意に識別される。

### 4.3 FOREACH 句

sinvoke 関数で指定されたクエリは、そのクエリによって生成されるページのファイル名を一意に指定するために、FOREACH 句を持たねばならない。FOREACH 句の構文は以下の通りである。

FOREACH ⟨*attribute*⟩

attribute は FOREACH 句を含むクエリによって生成されるページを一意に識別できる属性であり、対応する sinvoke 関数で指定された属性と同じ属性を指定する。これにより、sinvoke 関数と FOREACH 句を含む二つのクエリを関連付けることができる。

## 5 部分更新手法

この節では Web サイトを部分的に更新する手法を説明する。まず、更新が必要なページの判別方法について述べ、それから 3 つの更新アプローチを紹介する。

### 5.1 更新が必要なページの判別方法

更新が必要なページを判別するために、更新ログファイルとクエリ情報データベースを用いる。クエリ情報データベースは Web サイトを最初に生成したする際に生成する。図 4 に更新ログファイルとクエリ情報データベースの例を示す。更新ログには、オペレーション (insert, update, delete)、関係、属性とその値が記述される。またクエリ情報データベースには、各クエリで使われている関係と FOREACH 句で指定された属性が格納されている。

データベースが更新された場合、どのページが影響を受けるかを知る必要がある。図 5 は更新の必要なページを判別するアルゴリズムである。プロシージャ SpecifyPagesUpdate は、まず更新された関係 R を利用しているクエリ Q を探す。もし Q が存在した場合、クエリ情報データベースを調べそのクエリの FOREACH 句で指定されている属性 a を探す。そして pageList にファイル Q-v.html (v は a の値) を追加する。R を利用しているクエリがなくなるまで繰り返し、最終的に pageList に入っているファイル名を持つページが更新するページとなる。

```
insert  actor  (id, name, birth)  (210 , Ken Watanabe,1959/10/21)
insert  actor  (id, name, birth)  (211, Koyuki, 1976/12/18)
update  actor  (id, name, birth)  (93, Tom Cruise, 1962/7/3)
insert   movie  (id, title, category)  (1001, The Last Samurai, Historical)
insert  cast  (movie, actor)  (1001, 93)
insert  cast  (movie, actor)  (1001, 210)
insert  cast  (movie, actor)  (1001, 211)
delete  actor  (id, name, birth)  (200, Kyoko Ishikawa, 1979/11/7)
                                        .
                                        .
                                        .
```

(a) 更新ログファイル

| Query | Relation |
|---|---|
| Q1a | movie |
| Q1b | movie |
| Q1c | movie |
| Q1c | actor |
| Q1c | cast |

| Query | FOREACH |
|---|---|
| Q1b | m.category |
| Q1c | m.id |

(b) クエリ情報データベース

図 4: 更新ログファイルとクエリ情報データベースの例

```
1.  PROCEDURE SpecifyPagessUpdate
2.    RECEIVE update
3.
4.    /* Initialize */
5.    pageList:  the list of pages needed to updated
6.    R: relalation which update occurred
7.
8.    WHILE (Q=(FIND query using relation R) != null) DO
9.      a = FIND attribute of FOREACH clause in Q
10.     pageList.append(Q-v.html) (v: value of a)
11.   END WHILE
12.
13.   /* As a result, we get a list of pages needed to be updated */

14. END PROCEDURE
```

図 5: 更新が必要なページを判別するアルゴリズム

### 5.2 ページの削除

更新の必要なページが判別されたら、それらのページは再生成または削除される。削除は、FOREACH 句で指定されている属性を持つ関係に対し、オペレーション delete が実行された時のみ起きる。再生成するページと削除するページを識別するため図 5 のアルゴリズムの 9 行目と 10 行目の間で次の処理を行う: もし、属性 a が関係 R に属し、オ

ペレーションが delete の場合、Q-v.html は pageList ではなく removeList に追加する。removeList のページはプロシージャ終了後ただちに削除される。pageList のページの再生成方法については次節で説明する。

## 5.3 ページの再生成

データベースの更新は、Web サイト上には 2 種類の偏りとなって現れると考えられる。Web サイトの葉ページへの偏りと部分木への偏りである。このような偏りを考慮し、3 つの更新アプローチを提案する。

1. 個別更新アプローチ
2. 兄弟一括更新アプローチ
3. 部分木更新アプローチ

それぞれのアプローチについて詳しく説明する。

### 5.3.1 個別更新アプローチ

このアプローチは pageList のページを一つずつ個別に更新する。更新は、SuperSQL クエリの GENERATE $\langle medium \rangle$ 句を REGENERATE $\langle filename \rangle$ で置き換えたクエリを実行して行う。以下のクエリは Q2c-1.html(Q1c によって生成され、m.id の値が 1 であるようなページ) を再生成する場合の例である。GENERATE 句を REGENERATE 句で置き換えるのに加え、WHERE 句に条件 m.id=1 が追加されている。

```
REGENERATE Q2c-1.html
 [imagefile(a.pict), a.name, a.birth]!
FROM movie m, actor a, cast c
WHERE m.id=c.movie and c.actor=a.id and m.id=1
```

上記のクエリによって生成されるページは 1 ページのみ (Q2c-1.html) である。データベースの更新後すぐに Web ページを更新する場合、一つのデータベースの更新によって影響を受けるページは少ないので、このアプローチが適している。

### 5.3.2 兄弟一括更新アプローチ

Web ページの更新を定期的に行う場合、複数のページをまとめて更新できれば効率的である。一つのクエリによって生成されるページは兄弟ページと考えられる。このアプローチは兄弟ページをまとめて扱う、すなわちある一つのクエリによって生成されるページを全て再生成する。このアプローチは、最初にハイパーテキストを生成するのに用いたクエリを実行すればよいので、とても簡単である。例えば、Q2c によって生成されたページの大部分を再生成する必要がある場合、Q2c を実行して兄弟ページ全てを再生成する。

このアプローチは必要のないページまで再生成してしまうが、再生成する兄弟ページが多い場合でも実行するクエリは一つであり、一つのクエリ処理には固定的な時間がかかるので、再生成するページの数だけクエリを実行する個別更新に比べ低コストで更新できると考えられる。

### 5.3.3 部分木更新アプローチ

このアプローチも複数のページをまとめて扱うが、兄弟ページではなく Web サイトの同じ部分木に属するページをまとめて扱う。以下の 3 つのクエリ Q3a、Q3b、Q3c で生成される Web サイトを用いて説明する。

```
Q3a: GENERATE HTML
      [R1.a % sinvoke(Q3b, R1.b)]!
     FROM R1

Q3b: FOREACH R1.b
     GENERATE HTML
      [R2.c % sinvoke(Q3c, R2.d)]!
     FROM R1, R2
     WHERE R2.d=R1.b

Q3c: FOREACH R2.d
     GENERATE HTML
      [R3.e]!
     FROM R2, R3
     WHERE R3.f=R2.d
```

図 6 は Q3a、Q3b、Q3c で生成される Web サイトの木構造を表した図であり、灰色の四角は更新する必要があるページを表している。灰色の四角は Web サイトの部分木に偏っている。部分木のルートは Q3b-1.html であり、これは属性 R1.b の値が

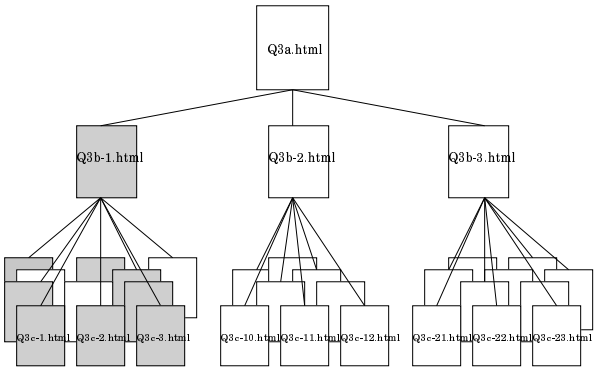


図 6: 部分木更新の例

1 であることを意味する。3 レベル目のページは、2 レベル目で R1.b でグルーピングされた後、R2.b でグルーピングされているので、3 レベル目の灰色の四角も条件 R1.b=1 を満たしている。よって、Q2b 及び Q2c の WHERE 句に条件 R1.b=1 を付加して実行すると、部分木が再生成される。このようにして、ある部分木全体が満たす条件を付加することによって、部分木をまとて再生成できる。

## 6 パフォーマンスの比較

3 つのアプローチのパフォーマンスを比較するため、[8] で提案した線形のコスト評価式を用い、拡張する。評価式は単純ではあるが、複雑なクエリの処理コスト (時間:sec) を実用的な精度で見積もることができる。生成されるページ数を P、ページの平均サイズ (KB) を S としたとき、クエリの処理コストを、

$$C_{query} = C_0 + C_1 \cdot P + C_2 \cdot S \cdot P \qquad (1)$$

と定義する。ただし、$C_0$ は一つのクエリ処理に固定的にかかる時間、$C_1$ は 1 ページを生成し格納するのにかかる時間、$C_2$ はページのサイズに掛けられる係数を表す。例として、CPU Pentium III 698MHz、PostgreSQL7.3.4、Linux2.4.18 の環境で SuperSQL を実行して得られた係数の例を以下に示す。

$C_0 = 0.2307[sec], C_1 = 0.001[sec/page], C_2 = 0.0004[sec/KB]$

3 つのアプローチのうち、個別更新アプローチと兄弟一括更新アプローチには (1) 式をそのまま用いることができるが、部分木更新アプローチは複数のクエリを同時に実行するので、(1) 式を単純に用いることができない。そこで (1) 式を (2) 式のように拡張する。ここで、$Q_s$ は部分木を生成するクエリの集合を表すものとするものとする。

$$C_{subtree} = \sum_{q_i \subset Q_s} \{C_0 + C_1 \cdot P_{q_i} + C_2 \cdot S_{q_i} \cdot P_{q_i}\} \qquad (2)$$

パフォーマンスの比較ではページの削除にかかるコストは全てのアプローチに共通であるので、再生成にかかるコストのみを比較する。3 つのケースにおけるそれぞれのアプローチの処理コストを比較する。各ケースでは 3 階層の Web サイトを用い、それぞれのレベルのページ数とサイズを表 1 のように仮定する。表 2 は 3 つのケースにおける各アプローチの処理時間を示している。

表 1: Web サイトの設計例

| レベル | ページ数 | ページサイズ [KB] |
|---|---|---|
| 1 | 1 | 15 |
| 2 | 30 | 30 |
| 3 | 6000 | 10 |

表 2: 3 つのケースにおけるクエリ処理時間 [sec]

| | 個別 | 兄弟一括 | 部分木 |
|---|---|---|---|
| a | 0.24 | 29.23 | - |
| b | 260.7 | 29.23 | - |
| c | 23.81 | 29.85 | 0.97 |

**更新が頻繁に起こらない場合**　これはデータベースの更新の度に Web ページも更新する場合、または更新するページ数が少ない場合である。

3 レベル目の 1 ページを更新する場合、個別更新と兄弟木一括更新のクエリ処理時間はそれぞれ 0.24[sec] と 29.23[sec] であり、個別更新の方が兄弟一括更新に比べ 120 倍速い。

**葉ページへの偏り**　これは更新が葉ページに偏っている場合、つまり 3 レベル目のページのみ更新が必

要でその数が多い場合である。例えば、いくつかの店の商品価格比較表を提供しているようなサイトでは、価格比較表はサイトの葉ページにある場合が多く価格は頻繁に変動する。

3 レベル目の 1000 ページを更新する場合、個別更新のクエリ処理時間は 260.7[sec] であり、兄弟一括更新のクエリ処理時間はケース 1 と同じで 29.23[sec] である。このケースでは兄弟一括更新の方が個別更新より 9 倍速い。

**部分木への偏り**　これは更新がサイトのある一部分に偏っている場合、つまり部分木に偏っている場合である。ある親ページのデータが更新された場合、その子ページのデータも更新されることが多いと考えられる。

2 レベル目の 1 ページと、そのページと同じ部分木に存在する 3 レベル目の 100 ページを更新する場合、個別更新、兄弟一括更新、部分木更新のクエリ処理時間はそれぞれ 23.81[sec]、29.85[sec]、0.97[sec] である。この場合、部分木更新は個別更新にくらべ 25 倍、兄弟一括更新に比べ 31 倍速い。

以上 3 つのケースについてパフォーマンスの比較を行ったが、他にも様々なケース (例えば、ケース 1 とケース 2 が同時に現れるケースなど) が考えられる。しかし、3 つの更新を適切に組み合わせれば多くの応用ケースについても対処できると考えられる。

## 7　まとめ

SuperSQL で静的に生成されるデータ集約型 Web サイトの効率的な部分更新手法を提案した。SuperSQL で生成された Web サイトの部分更新を実現するために、SuperSQL に新に sinvoke 関数と FOREACH 句を導入した。また、更新の偏りを考慮した 3 つの更新アプローチ、個別更新アプローチ、兄弟一括更新アプローチ、部分木更新アプローチを提案した。パフォーマンスの比較では、兄弟一括更新アプローチと部分木更新アプローチが 2 種類の更新の偏り、Web サイトの葉ページへの偏りと部分木への偏り、に対処できることを示した。

本論文では 3 つのアプローチを別々に扱ったが、3 つの更新を組み合わせることにより更新をより効率的に行うことができると考えられる。今後は、3 つのアプローチの最適な組み合わせを探索するアルゴリズムの提案する予定である。

## 参考文献


[1] Paolo Atzeni, Giansalvatore Mecca, Paolo Merialdo. Design and Maintenance of Data-Intensive Web Sites. In *Proc. EDBT'98*, pp.137-157, 1998.

[2] J. A. Blakeley, P. A, Larson, F. W. Tompa. Efficiently updating materialized Views. In *Proc. ACM SIGMOD*, pp.61-71,1986.

[3] K. Selcuk Candan, Divyakant Agrawal, Wen-Syan Li, Oliver Po Qang-in Hsiung. View Invalidation for Dynamic Content Cashing in Multitiered Architechtures. In *Proceedings of VLEB*, pp. 562-573, 2002.

[4] Edwin R. Lassettre. Olympic Records for Data at the 1998 Nagano games. In *Proc. ACM SIGMOD*, p.537, 1998.

[5] Giuseppe Sindoni. Incremental Maintenance of Hypertext Views. In *Proc. of Web*, 1998.

[6] Hoshi Mistry, Prasan Roy, S Sudarshan, Krithi Ramamritham. ¡aterialized View Selection and Maintenance Using Multi-Query Optimization. In *Proc. ACM SIGMOD*. pp. 307-318, 2001.

[7] Motomichi Toyama. SuperSQL: An Extended SQL for Database Publishing and Presentation. In *Proceedings of ACM SIGMOD '98 International Conference on Management of Data*, pp. 584-586, 1998.

[8] Motomichi Toyama, Takuhiro Nagafuji. Dynamic and Structured Presentation of Database Contents on the Web, in *Proc. EDBT'98*, pp.451-465, 1998.