データ集約型 Web サイトにおける静的生成コンテンツの部分更新

石川恭子[†] 有澤達也[†] 遠山元道[††]

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 1 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 HTML 文書を一括生成できる．しかし，更新が必要な場合に部分的な更新ができ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，Web サイトの部分的な更新を実現する．また，データベースの更新によって影響を受けるページを判別する手法，3 つの更新アプローチを提案し，Web サイトの効率的な部分更新を実現する．さらに，3 つの更新アプローチの組合せの中で，ページ生成時間がより小さい組合せを探索する，最適化アルゴリズムを提案し，実験により提案手法の評価を行う．

Partial Maintenance of the Statically Generated Contents in a Data-intensive Web Site

Kyoko Ishikawa,[†] Tatsuya Arisawa[†] and Motomichi Toyama[††]

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. Although this is one of the feature of SuperSQL, there was a problem that it couldn't refresh web pages partially when update to the base relations occurred. 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 show how to specify the pages which are affected by the update of database, and introduce three update approaches. Moreover, we propose an optimization algorithm in order to fine the combination that can generate pages faster than other combinations, and evaluate proposal methods by the experiments.

1. はじめに

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 サイトを構築およ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),6),12),13)]．

Web サイトの生成方法には静的生成と動的生成の 2 種類がある．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．たとえば，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 千件にのぼった[9)]．このような場合，アクセスのたび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 [14),15)] は，1 つのクエリを実行するだけで多段のリンクを持つ HTML 文書を一括して生成できる．これは SuperSQL の特徴の 1 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，すべて動的生成となる．そこで，本研究では SuperSQL

† 慶應義塾大学大学院理工学研究科
Graduate School of Science and Technology, Keio University
†† 慶應義塾大学理工学部
Faculty of Science and Technology, Keio University

に sinvoke（static invocation）関数と FOREACH 句を新たに導入し，SuperSQL クエリを分割し，Web サイトの部分的な更新を実現する．また，データベースの更新によって影響を受けるページを判別する手法，3 つの更新アプローチを提案し，Web サイトの効率的な部分更新を実現する．さらに，3 つの更新アプローチの組合せの中で，ページ生成時間がより小さい組合せを探索する，最適化アルゴリズムを提案し，実験により提案手法の評価を行う．

2. 関 連 研 究

本研究では静的に生成されたデータ集約型 Web サイトにおける効率的な更新手法の提案を目的としており，関連研究として以下のものがあげられる．

- データ集約型 Web サイトの構築と管理
- Web ページのメンテナンス
- ビューのメンテナンス

以下，それぞれについて説明する．

2.1 データ集約型 Web サイトの構築と管理

データ集約型 Web サイトの構築に関する様々なシステムが提案された．

SuperSQL [14),15)] は本研究で利用するシステムであり，Web サイトの動的生成と静的生成のどちらも可能である．静的生成の場合，データベースの更新によって影響を受けるページだけでなく，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．詳しくは 3 章で説明する．

オーストラリアのオンライン旅行予約システムである TIScover [12)] も SuperSQL と同様に静的生成，動的生成どちらも可能である．このシステムでは，頻繁に更新されるデータを含むページをユーザの要求に応じて生成し，その他のページはあらかじめ静的に生成する．データベースが更新された際には，その更新によって影響を受けるページをデータベースに格納されたメタデータを用いて影響を受けるページを判別する．Autoweb [6)] は，WWW アプリケーションのための開発支援ツールであり，ユーザの要求に応じて Web ページを生成し，またある条件に基づいて Web ページをあらかじめ静的に生成することもできる．しかし，データベースの更新によって影響を受けるページを自動的に判別できるかどうかは記述されていない．

一方 ARANEUS [1),13)] は，Web サイトを静的に生成することしかできない．このシステムは TIScover と同様，データベースに格納されたメタデータを用いて影響を受けるページを判別する．また，Strudel [5)] および Torii [4)] は動的生成のみに対応している．

2.2 Web ページのメンテナンスに関する研究

データ集約型 Web サイト上のデータは，データベースのデータとの一貫性を保持する必要があり，データベースが更新された場合には Web ページのデータも更新する必要がある．したがって，Web ページのデータの鮮度をいかに保証するかが問題となる．この問題に対しても様々な研究がなされた．

Liu らは更新クエリを用いて Web ページを更新する手法を提案している [11)]．複数の同種クエリを 1 つに統合することでデータベースへのアクセスを減らし，タイムリーな Web ページの更新を実現している．

Candan らは，キャッシュされた動的コンテンツのうち，データベースの更新によって影響を受けるものの削除方法を提案している [3)]．データベースの更新によって影響を受けるページの判別方法を提案し，さらに同種ページをキャッシュから削除する場合のバッチ処理方法を示している．

本研究では SuperSQL によって静的に生成された Web サイトの更新方法を提案する．データベースの更新によって影響を受けるページの判別方法を示し，そのページを効率的に更新するために複数のページを同時に更新する手法を提案する．SuperSQL は複数のページを一度に生成することができるので，複数のページをまとめて更新することが可能である．

2.3 ビューのメンテナンスに関する研究

データ集約型 Web サイトのメンテナンス問題は，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 ΔV を計算し，ΔV を V に反映させるという手法が広く用いられている [2),7),10)]．

データベースの n 個の関係 $R_i(1 \leq i \leq n)$ から n−1 個の Join を用いてビュー V $(R_1 \bowtie R_2 \bowtie ... \bowtie R_n)$ を作成する場合，この n 個の関係すべてに更新が行われたときのビュー V の変化分 ΔV を求めると式 (1) のようになる [2)]．

$$\begin{aligned}\Delta V = &(\Delta R_1 \bowtie R_2 \bowtie ... \bowtie R_n)\\ &\cup (R_1 \bowtie \Delta R_2 \bowtie ... \bowtie R_n)\\ &\cup ....\\ &\cup (\Delta R_1 \bowtie \Delta R_2 \bowtie ... \bowtie \Delta R_n) \qquad (1)\end{aligned}$$

ΔR_i は R_i の変化分である．ビューの変化分 ΔV を V に適用させることにより更新後のビューを求めることができる．このとき $2^n - 1$ 個の式を計算することになり，非常に計算量が多く効率が悪い．式 (2) のよ

☆ ここでいう Web サイトとは，1 つまたは関連のある複数の SuperSQL クエリによって生成される HTML 文書である．

うに n 個の式で計算することも可能である[7),10)].

$$\begin{aligned}\Delta V = &(\Delta R_1 \bowtie R_2 \bowtie ... \bowtie R_n)\\ &\cup (R'_1 \bowtie \Delta R_2 \bowtie ... \bowtie R_n)\\ &\cup ....\\ &\cup (R'_1 \bowtie R'_2 \bowtie ... \bowtie R'_{n-1} \bowtie \Delta R_n) \quad (2)\end{aligned}$$

R'_i は更新後のテーブルである．

データ集約型 Web サイトの場合，1 つのクエリによって生成される複数のページを 1 つのビューと見なすことができる．しかし，HTML は複雑であり変化分を Web ページに直接反映させることは難しい．よって，Web サイトのメンテナンスの場合は，どのページを更新する必要があるかを判別することが重要になる．

本研究では実体化ビューの変化分の計算方法を応用し，影響を受けるページの判別を行う．従来のビューの変化分 ΔV の計算では更新前の関係 R_i を利用している．更新前の関係を利用する場合，データベースのスナップショットが必要となる．本研究では従来研究のようにデータベースのスナップショットを用いる手法と，スナップショットを用いない手法の 2 通りで影響を受けるページの判別を行う．

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 $<medium><TFE>$ 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 $<medium>$ は，現在では HTML，XML，PDF，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．カンマ (,)，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．HTML 文書の生成において深度方向へのレイアウトはハイパーリンクの生成を指定し，左辺がアンカ，右辺がリンク先のページを生成する TFE となる．

反復子は指定する方向に，インスタンス数だけ繰り返して表示する．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．たとえば，

[氏名，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．

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．たとえば，

[研究室，[氏名，学籍番号]!]!

は，研究室ごとにグルーピングを行い，それぞれの学生の氏名と学籍番号の一覧が表示される．

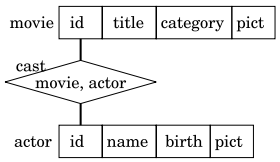

図 1 映画情報データベースのスキーマ
Fig. 1 Schema of the movie database.

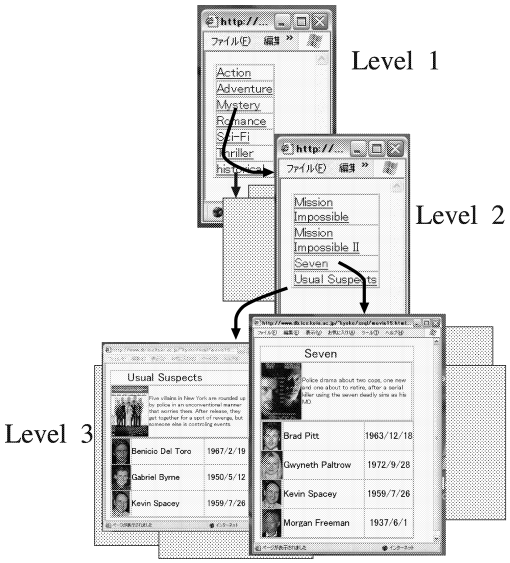

図 2 SuperSQL クエリによる HTML 文書の生成例
Fig. 2 An example of the web pages generated with SuperSQL query.

3.2 SuperSQL による HTML 文書生成例

図 1 の映画情報データベースから以下に示す SuperSQL クエリ Q0 によって得られる HTML 文書を **図 2** に示す．

```
Q0: GENERATE HTML
    [m.category%[m.title%
      {m.title!{imagefile(m.pict),m.story}!
        [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とおりである．

invoke(*TFE*, file="*file_name*", cond=*condition*)

TFE には invoke 関数で生成されるハイパーリンクのアンカとなる TFE を指定する．*file_name* は呼び出されるクエリを含むファイルの名前であり，*condition* は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．

以下に示すクエリ 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 HTML 文書の 1〜3 レベル目を生成する．

```
Q1a: GENERATE HTML
     [invoke(m.category, file="Q1b",
       cond="m.category="m.category)]!
     FROM movie m
Q1b: GENERATE HTML
     [invoke(m.title, file="Q1c", cond="m.id="m.id)]!
     FROM movie m
Q1c: GENERATE HTML
     [m.title!{imagefile(m.pict),m.story}!
       [imagefile(a.pict),a.name,a.birth]!]!
     FROM movie m, actor a, cast c
     WHERE m.id=c.movie and c.actor=a.id
```

Q0 は，HTML 文書の 3 レベルをすべ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．呼び出し側である Q1a の invoke 関数の *condition* で"m.category="m.category が指定されているので，invoke 関数で生成されたハイパーリンクで移動する際に，Q2b に対して，Q1a で選択された category（たとえば xx）を含む条件（m.category="xx"）が動的に追加されたうえで，データベースに問い合わせた結果を出力する．また同様に，Q1c には Q1b で指定された条件（たとえば m.id=yy）が追加される．

3.4 問 題 点

SuperSQL による HTML 文書の生成は，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，およ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，ページアクセスのたび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 HTML 文書の見た目が等しいので等価と見なす．

4.1 分 割 の 例

前章のクエリ 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．

```
Q2a: GENERATE HTML
     [sinvoke(m.category, file="Q2b",
       att1=m.category)]!
     FROM movie m
Q2b: FOREACH m.category
     GENERATE HTML
     [sinvoke(m.title, file="Q2c", att1=m.id)]!
     FROM movie m
Q2c: FOREACH m.id
     GENERATE HTML
     [m.title!{imagefile(m.pict),m.story}!
       [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 文

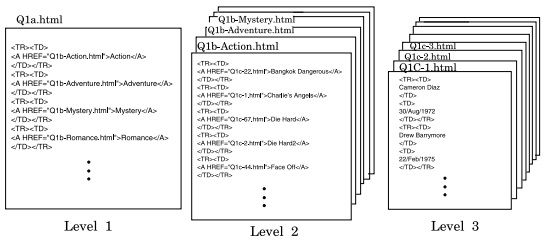

図 3 ハイパーリンクコードの例
Fig. 3 An example of the hyper-link code.

書のハイパーリンクコードの例である．

4.2 Static Invocation（sinvoke）関数

sinvoke 関数は 2 つのクエリ，sinvoke 関数を含むクエリと sinvoke 関数で指定されたクエリを関連付ける．invoke 関数と異なりすべてのページは静的に生成される．sinvoke 関数の構文は以下のとおりである．

sinvoke (TFE, file=”*file_name*”, att1=*attribute1*, att2=*attribute2*, ···)

TFE には invoke 関数と同様に，この sinvoke 関数で生成されるハイパーリンクのアンカとなる TFE を指定する．*file_name* はハイパーリンクで呼び出す SuperSQL クエリのファイル名を指定し，*attribute1*, *attribute2*, ···は *file_name* の SuperSQL 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群を指定する．

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ごとにグルーピングされるので，m.category 属性で一意に識別できる．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 属性で一意に識別できる．

sinvoke 関数で指定されたファイル名と属性の値を用いてリンク先の URL が生成される☆．たとえば Q2a の場合，m.category の値が”Action” ならば

〈 A HREF=”Q2b-Action.html”〉 Action 〈 /A〉

のように Action のリンク先の URL として”Q2b-Action.html” が埋め込まれる．

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 句は GENERATE 句の前に置き，その構文は以下のとおりである．

FOREACH *attribute1*, *attribute2*, ···

attribute1，*attribute2*，··· 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群であり，対応する sinvoke 関数で指定された属性と同じ属性を sinvoke 関数の att1，att2，··· で指定した順に指定する．これにより，sinvoke 関数と FOREACH 句を含む 2 つのクエリを関連付けることができる．例では（ Q2a, Q2b）と（ Q2b, Q2c）の 2 組が sinvoke 関数と FOREACH 句によって関連付けられており，それぞれ m.category と m.id を指定している．

FOREACH 句を含むクエリによって生成される HTML 文書のファイル名は，自分自身のクエリ名と FOREACH 句で指定された *attribute* の値を用いて生成される．たとえば Q2b では m.catogory の値が Action の場合，ファイル名が”Q2b-Action.html” の HTML 文書が生成される．

このファイル名と sinvoke 関数によって生成されるハイパーリンクの URL が等しくなるので，関連付けられたクエリ間にハイパーリンクを自動生成することができる．したがって，関連付けられたクエリを別々に実行しても，INVOKE 句で指定された属性群の値と FOREACH 句で指定された属性群の値とが等しいものは，ハイパーリンクを張ることができる．

4.4 クエリ分割の利点

sinvoke 関数と FOREACH 句を用いてクエリを分割する最大の利点は，次章で述べる Web サイトの部分更新が可能になることだが，さらにいくつかの利点がある．

- 再帰的なリンクの実現：あるクエリ Q と直接 sinvoke 関数で関連付けられている，もしくは間接的に関連付けられているクエリ中に sinvoke 関数でクエリ Q を指定することで，ツリー型の 1 方向なリンクだけでなく，再帰的なリンクを実現することができる．たとえば映画のページと俳優のページの相互リンクは，それぞれを生成するクエリの sinvoke 関数でもう一方のクエリを指定するだけで実現できる．invoke 関数を用いた動的生成では実現できていたが，静的生成の場合，sinvoke 関数を用いない 1 つのクエリでは相互リンクは実現できない．FOREACH 句と sinvoke 関数による HTML の生成は本質的にバッチ処理であり，生成された HTML 上で利用者が再帰的なナビゲーションを行っても，その時点で sinvoke 関数が実行されるわけではない．invoke 関数による動的生成では再帰で同じページの戻る際には冗長に生成が行われるが，sinvoke ではこのような冗長性は発生しない．
- 重複の除去：1 つのクエリで Web サイトを生成した場合，まったく同じページが存在することがある．たとえば，M という映画がアクションとコメディの 2 つのカテゴリに分類されている場合，M はアクション映画の一覧とコメディ映画の一覧の 2 ページに表示される．通常では M を内容とするまったく同じページが 2 ページ生成され，それぞれからリンクされることになる．FOREACH 句を利用する場合，FOREACH 句に指定された属性群の値ご

☆ ファイル名と属性の値を用いてリンク先の URL を生成するとしているが，実際には URL エンコーディングを行い UTF-8 にエンコードしたものを URL として利用している．

Site	Query
movie	Q2a
movie	Q2b
movie	Q2c

(a) Site_Query Table

Query	Relation
Q2a	movie
Q2b	movie
Q2c	movie
Q2c	actor
Q2c	cast

(b) Query_Relation Table

Query	FOREACH
Q2b	m.category
Q2c	m.id

(c) Query_FOREACH Table

図 4 クエリ情報データベースの例
Fig. 4 An example of query information database.

とにページが作成されるので，このような重複を除去できる．

- クエリ記述の簡単化：1 つのクエリによって巨大な Web サイトを生成する場合，クエリは長く複雑になる．Web サイトの各レベルごとにクエリを分割することでクエリを簡単化できる．

5. 部分更新手法

本章では SuperSQL で生成された Web サイトを部分的に更新する手法を説明する．部分更新は以下の順に処理が行われる．

(1) 関係データベースを更新する．
(2) 更新ログを生成する．データベースが更新された場合，更新されたタプルに関する一時テーブルを作成し，トリガによりその一時テーブルに更新されたタプルが挿入される．この一時テーブルを更新ログとして利用する．トリガとは関係データベースに更新が行われたときに，自動的に関係データベース側で処理を行うものであり，本システムでは更新が行われた際に自動的に一時テーブルにタプルを挿入する．一時テーブルには元のテーブルの属性に Operation という属性を加えたものであり，その値は，タプルが挿入された場合は”insert”，削除された場合は”delete”，更新された場合は”update” として挿入する．
(3) 部分更新を実行する Web サイトを指定する．
(4) 更新ログとクエリ情報データベースを用いて指定された Web サイトの更新の必要なページを判別する．クエリ情報データベースについては後述する．
(5) 削除の必要なページを削除する．
(6) 3 つの更新アプローチの組合せのうち，最適な組合せを探索し部分更新クエリを生成する．
(7) 部分更新クエリを SuperSQL システムが処理し，Web ページを部分的に新規生成/再生成する．

5.1 クエリ情報データベース

クエリ情報データベースとは，SuperSQL で Web サイトを生成する際に，その Web サイトを生成するために必要なクエリの様々な情報を格納したデータベースであり，以下の 3 つのテーブルからなる．

(1) Site_Query Table
(2) Query_Relation Table
(3) Query_FOREACH Table

Site_Query Table は SuperSQL で生成された Web サイトと，その Web サイト生成に利用したクエリの情報を格納しているテーブルである．Query_Relation Table は，各クエリで利用しているテーブルの情報を格納しているテーブルである．Query_FOREACH Table は，各クエリの FOREACH 句で指定された属性の情報を格納しているテーブルである．このクエリ情報データベースを用いて，更新の必要なページを判別する．

図 4 はクエリ情報データベースの例である．

5.2 更新の必要なページの判別

更新の必要なページの判別は更新ログとクエリ情報データベースを用いて行う．まず再生成の必要なページと削除の必要なページの違いについて述べる．そして，再生成または削除の必要なページの判別方法について説明する．

5.2.1 ページの再生成と削除

データベースが更新された場合，その更新によって影響を受けるページを再生成（新規生成も含む），または削除する必要がある．データベースの更新のうち insert，update はすべてページの再生成となるが，delete の場合にはページの再生成となる場合とページの削除になる場合があり，再生成と削除のどちらを行う必要があるのかを判断する必要がある．

ページの削除は以下の 2 つの場合に必要となる．このほかのすべての場合はページの再生成が必要となる．

(1) FOREACH 句で指定された属性を持つテーブルからタプルが delete され，かつ FOREACH 句

で指定された属性がそのテーブルの候補キーである場合

(2) FOREACH 句で指定された属性を持つテーブルからタプルが delete され，かつ FOREACH 句で指定された属性がそのテーブルの候補キーではないが，delete されるタプルの FOREACH 句で指定された属性の値と同じ値のタプルが他に存在しない場合

たとえば，4.1 節のクエリ Q2b と Q2c を考える．Q2b はカテゴリーごとに映画一覧を表示するページを生成し，Q2c は映画ごとにその詳細情報を表示するページを生成する．Q2b と Q2c ではどちらも movie テーブルが利用されており，movie テーブルから以下のようなタプルが delete された場合とする．

（id=7,title="Seven",category="Mystery",pict="7.jpg"）

この delete によって影響を受けるページは，ファイル名が"Q2b-Mystery.html" である Q2b によって生成されたページと，ファイル名が"Q2c-7.html" である Q2c によって生成されたページの 2 ページである．まず"Q2c-7.html" について考える．movie テーブルは Q2c で指定された属性を持つテーブルであり，かつ Q2c の FOREACH 句で指定されている属性 m.id は movie テーブルの候補キーである．よって，"Q2c-7.html" はページを削除する必要がある．一方"Q2b-Mystery.html" は，ページの再生成が必要な場合とページの削除が必要な場合がある．Q2b で指定された属性は movie テーブルの属性であるが，候補キーではない．よって，movie のタプルのうち category="Mystery" であるものが delete されたタプル以外にも存在する場合はページの再生成，存在しない場合はページの削除となる．

5.2.2 スナップショット法を用いた ΔP の計算

次に，更新が必要なページ群 ΔP を計算する．従来研究では更新前のデータベースのスナップショットを用いて計算している．このような従来の手法をスナップショット法と呼ぶ．スナップショット法では，以下のような手順で ΔP を計算する．

$R1$ から Rn の n 個の関係を利用したクエリ Q によって生成されるページ群を $P = R1 \bowtie R2 \bowtie \cdots \bowtie Rn$ とする．このページ群 P のうち，関係 Ri $(1 \leq i \leq n)$ へのタプルの insert および delete によって影響を受けるページをそれぞれ ΔPi^+，ΔPi^- とすると，たとえば $i = 1$ のとき $\Delta P1^+$，$\Delta P1^-$ はそれぞれ式 (3)，(4) で計算される．

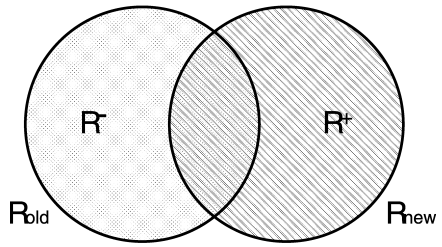

図 5 関係 R の変化分の表記法
Fig. 5 Notation for the delata of relation R.

$$\Delta P1^+ = R1^+ \bowtie (R2_{old} \cup R2^+) \bowtie ... \bowtie (Rn_{old} \cup Rn^+) \quad (3)$$

$$\Delta P1^- = R1^- \bowtie R2_{old} \bowtie ... \bowtie Rn_{old} \quad (4)$$

ここで，**図 5** は関係 R の変化分の表記法を示しており，Ri_{old}，Ri_{new} は更新前後の Ri を，Ri^+，Ri^- は insert および delete されたタプルの集合を表す．update されたタプルについては，update される前のタプルが Ri^- に，update された後のタプルが Ri^+ に含むものとする．

そして，ページ群 P のうち，関係 Ri（$1 \leq i \leq n$）へのタプル操作によって更新が必要なページ群を ΔPi とすると，

$$\Delta Pi = \Delta Pi^+ \cup \Delta Pi^- \quad (5)$$

と表すことができるため，更新によって影響を受けるページの全体は式 (6) で表される．

$$\Delta P = \bigcup_{i=1}^{n} \Delta Pi = \bigcup_{i=1}^{n} (\Delta Pi^+ \cup \Delta Pi^-) \quad (6)$$

式 (3)，(4) はどちらも Ri_{old}，すなわち更新前のデータベースを用いているため，スナップショット法と呼ぶ．

本研究では，スナップショット法に対して，更新前のデータベースを用いない手法である置換法を提案する．

5.2.3 置換法を用いた ΔP の計算

本研究では，影響を受けるページ ΔP を計算するにあたり，更新前のデータベースを参照しないで計算する置換法を提案する．式 (3) を計算する際に，Ri_{old} を利用する代わりに，更新後のテーブル Ri_{new} を利用する．

insert の場合

$Ri_{old} \cup Ri^+$ を Ri_{new} に置き換える．すると式 (3) は，

$$\Delta P1^+ = R1^+ \bowtie R2_{new} \bowtie ... \bowtie Rn_{new} \quad (7)$$

となる．$Ri_{old} \cup Ri^+ = Ri^- \cup Ri_{new} \supseteq Ri_{new}$ であり，Ri_{new} に置き換えることで Ri^- の分だけ更新が

表 1 置換法とスナップショット法の計算量の比較
Table 1 Comparison of the performance of substitution approach and snapshot approach.

	insert	delete
置換法	O(n)	$O(n^2)$
スナップショット法	$O(n^2)$	O(n)

必要と判別されるページが不足する可能性がある．しかし，Ri^- の部分は delete によって影響を受けるページを判別する際に計算されるので，insert によって影響を受けるページはすべて判別される．

delete の場合

Ri_{old} を $Ri_{new} \cup Ri^-$ に置き換える．すると式 (4) は，

$$\Delta P1^- = R1^- \bowtie (R2_{new} \cup R2^-) \bowtie ... \bowtie (Rn_{new} \cup Rn^-) \quad (8)$$

となる．$Ri_{old} \subseteq Ri_{old} \cup Ri^+ = Ri_{new} \cup Ri^-$ であり，Ri^+ の分だけ過剰に更新が必要と判別される可能性がある．しかし，過剰に判別される分には冗長な再生成をするだけのことだけであり，delete によって影響を受けるページはすべて正確に判別される．

5.2.4 置換法とスナップショット法の比較

本研究で提案した，データベースのスナップショットを用いないで ΔP を計算する置換法と，データベースのスナップショットを用いて計算するスナップショット法について，比較する．**表 1** は 2 つの手法の計算量を比較した表である．n はクエリで使用されているテーブル数である．

本研究で提案した置換法では，insert による影響 ΔPi^+ を算出する際に，Ri_{new} は計算なしで得ることができるため，計算量は $O(n)$ となるが，delete による影響 ΔPi^- を算出する際には，$Ri_{new} \cup Ri^-$ を計算する必要があるため，計算量は $O(n^2)$ となる．一方，従来研究のスナップショット法では，insert による影響 ΔPi^+ を算出する際に，$Ri_{old} \cup Ri^+$ を計算する必要があるため，計算量は $O(n^2)$ となるが，delete による影響 ΔPi^- を算出する際には，Ri_{old} は計算なしで得ることができるため，計算量は $O(n)$ となる．このことから，本研究で提案した置換法と従来研究のスナップショット法は，insert と delete の割合によって有利さが変わり，特に insert と delete の割合が等しければどちらの手法を用いても計算量はほぼ同程度であるといえる．

スナップショット法は，通常のデータベースのほかに更新前のデータベースのスナップショットが必要なため，ほとんど同じデータベースを 2 つ保持しなくてはならないという欠点がある．一方置換法は，delete によって影響を受けるページを計算する際に，過剰にページが判別されてしまう場合があり，この場合本来更新しなくてもよいページを更新してしまうという欠点がある．

Web サイトが参照するデータベースがどのように更新されているかによって，insert が多い場合は置換法を，delete が多いときは従来手法であるスナップショット法をとるよう，Web サイトを設計する際に管理者が決定する．

5.3 3 つの更新アプローチ

再生成の必要なページが判別されたら，それらのページを部分的かつ効率的に再生成する必要がある．

データベースの更新要求には偏りがある場合が多い．Labrindis らは New York Stock Exchange（NYSE）における株価データベースの更新履歴を分析し，たとえば “何万銘柄の株のうち最も更新頻度の高い 10 銘柄の更新が全体の更新の 15%を占める” などのように，データベースの更新には偏りがあることを確認したと報告している[8]．

このようなデータベースの更新の偏りを，本論文で扱うような構成においては，Web サイト上の 2 種類の偏りとなって現れると仮定する．Web サイトの葉ページへの偏りと部分木への偏りである．株価データベースに基づく株価 Web サイトで，木構造の根で業種別に部分木を作り，1 つの部分木の葉がその業種の各企業のページとする．葉ページへの偏りとは，各企業のページが頻繁に更新され，上位が影響を受けないようなページデザインを想定する．一方，部分木への偏りは，特定業種の取引が他の業種より相対的に多い状況を想定する．

もちろん，偏りはこの 2 パターンに限られるわけではないが，複雑なケースはこの基本形の組合せ的な現象として理解できると考える．

本研究ではこの 2 種類の偏りを考慮し，効率的に部分更新を実現する 3 つの更新アプローチを提案する．

- 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

以下，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

このアプローチは再生成が必要と判別されたページを 1 ページずつ個別に更新する．更新は，もとのクエリに条件を付加したものを実行して行う．以下のクエリは Q2c-1.html（Q2c 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．Q2c の WHERE 句に条件 m.id=1 が追加されて

いる．

```
Q2c': GENERATE Q2c-1.html
      [m.title!{imagefile(m.pict),m.story}!
        [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，1 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 ページの更新時におけるクエリ処理では 1 つのクエリごとに固定的な時間がかかるため，定期的に更新を行う場合には，複数のページをまとめて更新できれば効率的である．ここで，1 つのクエリによって生成されるページを兄弟ページと定義し，この兄弟ページをまとめて扱う☆．すなわちある 1 つのクエリによって生成されるページをすべて再生成する．このアプローチは最初に Web サイトを生成するために用いたクエリをそのまま実行すればよい．たとえば，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すべてを再生成する．

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 1 つであり，1 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．簡単のため，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
      [sinvoke(R1.a, file="Q3b", cond="R1.b="R1.b)]!
     FROM R1
Q3b: FOREACH R1.b
     GENERATE HTML
      [sinvoke(R2.c, file="Q3c", cond="R2.d="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
```

☆ 議論の単純化のために 1 クエリが 1 レベルを生成すると仮定する．実際には SuperSQL の 1 クエリで生成される多レベルは 2 クエリでも生成できるから，その木のリーフ集合に着目すればこの仮定の下での議論を容易に一般化できる．

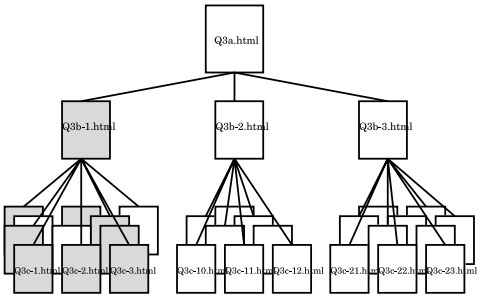

図 6 部分木更新の例
Fig. 6 An example of subtree update.

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 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.d 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，Q3b および Q3c 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，部分木をまとめて再生成できる．

5.4 更新プランの最適化

定期的または要求に応じて Web サイトの更新を行う場合，Web サイト上の複数のページを更新することになる．3 つの更新アプローチのどれか 1 つを適切に選ぶことにより，Web サイト全体の再生成に比べると効率的な部分更新が実現できる．しかし，3 つの更新アプローチのうち 2 つまたは 3 つのアプローチを組み合わせることによって，より効率的に更新を行うことができる．

5.4.1 最適化の必要性

ある Web サイトの更新を行う際，再生成の必要なページが複数の場合には，適用できる更新アプローチの組合せも複数存在する．

図 7 はクエリ 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．この例で考えら

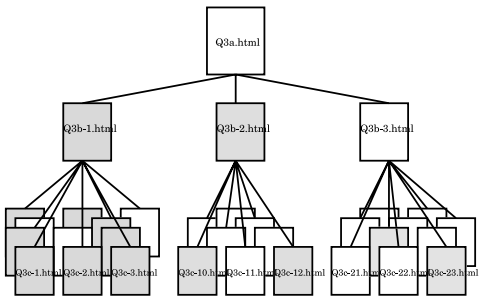

図 7 更新の必要なページの例
Fig. 7 An example of the affected pages.

れる更新の組合せは次の 5 通りある．

(1) すべて個別更新．

(2) 第 2 レベルと第 3 レベルをそれぞれ兄弟一括更新．

(3) 1 つの部分木更新とその他を個別更新．

(4) 1 つの部分木更新と第 2 レベルを兄弟一括更新，残りの第 3 レベルを個別更新．

(5) 1 つの部分木更新と第 3 レベルを兄弟一括更新，残りの第 2 レベルを個別更新．

このように小規模な Web サイトでも，更新の必要なページの分布の仕方によっては考えられる更新組合せの数は大きくなり，それぞれの場合で再生成にかかる時間も異なるので，ページ生成時間が最も小さい更新組合せを探索する必要がある．探索の際，次章の実験で得られた各アプローチの生成時間の近似式を利用し，各更新組合せの生成時間の推定値を求める．

5.4.2 最適化アルゴリズム

以下の 2 つのプランにおいて，それぞれ生成時間の推定値が最小となる組合せを探索し，最後に 2 つのプランで得られた組合せで推定値が小さい方を更新プランとする．

plan1 では，3 つのアプローチのうち個別更新と兄弟一括更新の 2 つのアプローチのみの場合の組合せ，plan2 では 3 つのアプローチのうち部分木更新アプローチを必ず含む組合せを探索する．

以下は最適化の処理手順である．

(1) plan1 の生成時間の推定値が最小となる組合せを探索し，その組合せの生成時間の推定値 $Time_{plan1}$ を計算する．

(1-1) 各レベルごとに個別更新と兄弟一括更新のどちらを適用するかを判定する．

(1-2) それぞれのレベルの生成時間の推定値を計算し，その和を $Time_{plan1}$ とする．

(2) plan2 の生成時間の推定値が最小となる組合せを探索し，その組合せの生成時間の推定値 $Time_{plan2}$ を計算する．

(2-1) ΔP を更新の必要なページのリストとする．ΔP のページがすべて同じ 1 レベルのページの場合は $Time_{plan2}$=$Time_{plan1}$ とし，(2-3) へ．それ以外は (2-2) へ．

(2-2) ΔP のうち，まだ調べていないページでより上位レベルのページを 1 ページ探し p とする．p が葉ページの場合は (2-5) へ．それ以外は (2-3) へ．

(2-3) p を親とする部分木を sub_p とする．ΔP の中で sub_p に含まれるページを調べ，この部分木のみを更新するとした場合，個別更新と部分木更新のどちらが適しているかを判定する．個別更新が適している場合は (2-2) へ戻る．部分木更新が適している場合は (2-4) へ．

(2-4) sub_p の p 以外の更新の必要なページの集合を $\Delta Psub_p$ とする．$\Delta Psub_p$ について (2-1, 2-2, 2-3) を繰り返し，sub_p の推定値の最小となる更新プランを決める．(2-2) へ戻る．

(2-5) ΔP のページのうち，(2-2, 2-3, 2-4) で部分木更新と判定されたページ，または判定されたページを親とする部分木に含まれないページについて，各レベルごとに個別更新と兄弟一括更新のどちらを適用するかを決める．

(2-6) それぞれの部分木更新の生成時間の推定値と (2-5) で判定された部分木に含まれないページの更新アプローチの生成時間推定値の和を $Time_{plan2}$ とする．

(3) $Time_{plan1} \leq Time_{plan2}$ ならば plan1 の更新組合せを更新プランとし，$Time_{plan1} > Time_{plan2}$ ならば plan2 の更新組合せを更新プランとする．

6. 実験と評価

提案した部分更新手法を Java（JDK 1.4）で実装し，3 つの更新アプローチの比較，および最適化アルゴリズムによって得られた組合せとその他の組合せの比較実験を行った．この 2 つの比較では再生成の必要なページ数がパフォーマンスに大きな影響を与えるため，データベースへの更新によって，テーブルのタプル数が変更されたり全体ページ数が変わったりしないように，データベースへの更新は update のみに限定した．実験では 4 つのクエリによって生成される 4 レベルの Web サイトを用い，すべてのページは 4 KB となるように設定した．実験を行った環境は，CPU Pentium（R）Ⅲ 1.4 GHz ×2，メモリ 2 GB，OS Linux（RedHat7.3），DBMS PostgreSQL 7.4.3 である．

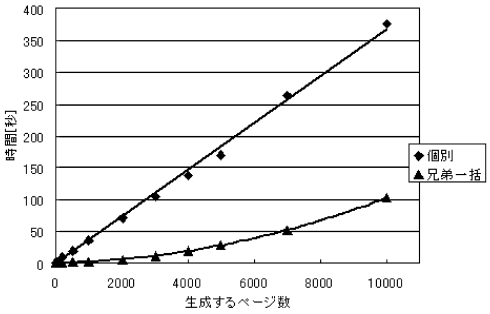

図 8 兄弟一括更新と個別更新のページ生成時間の比較：1
Fig. 8 Generation time of sibling update and one-by-one update.

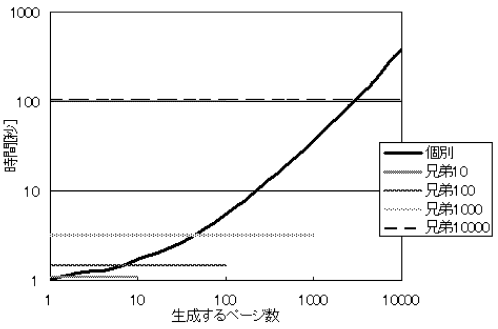

図 9 兄弟一括更新と個別更新のページ生成時間の比較：2
Fig. 9 Generation time of sibling update and one-by-one update: 2.

6.1 予備実験：各アプローチのページ生成時間の近似式

まず実験によって 5.3 節で述べた 3 つのアプローチのページ生成時間の近似式を求めた．**図 8** は兄弟一括更新と個別更新で，生成するページ数を 1 から 10,000 まで変化させたときのページ生成時間を比較したグラフである．

この実験から，兄弟一括更新の生成時間 $T_{sibling}$ と個別更新の生成時間 T_{one} は，クエリによって生成するページ数を n とした場合，以下の式で近似できる．

$$T_{sibling}(n) = (7 \times 10^{-9})n^2 + (6 \times 10^{-4})n + 1.025 \quad (9)$$

$$T_{one}(n) = 0.037n + 1.025 \quad (10)$$

部分木更新の生成時間はその部分木の構造によって異なるため，個別更新や兄弟一括更新のように実験から生成時間の近似式を得ることはできない．部分木更新は，各レベルを生成するクエリにある条件を加えることによって Web サイトの一部分を生成するので，条件を加えた各レベルのクエリを実行することは，その条件を加えたクエリによって生成される兄弟ページを一括して生成するのとほぼ等しいと考えられる．したがって，部分木更新の生成時間 $T_{subtree}$ は，兄弟一括更新の生成時間 $T_{sibling}$ を用いて近似することができる．ここで，木全体のレベル数が l，部分木の根ページが k レベルに属しているとし，$(k \leq i \leq l)$ の各レベル i で生成するページ数を n_i とすると，$T_{subtree}$ は以下のようになる．

$$T_{subtree} = \sum_{i=k}^{l} T_{sibling}(n_i) \quad (11)$$

以下の実験では，この $T_{sibling}$，T_{one}，$T_{subtree}$ の式を生成時間として，5.4.2 項の最適化アルゴリズムに従い，更新アプローチの選択を行った．

6.2 実験 1：3 つの更新アプローチの比較

まず，兄弟更新と個別一括更新，部分木更新と個別更新の組に対して，パフォーマンスの比較を行った．

6.2.1 兄弟一括更新アプローチと個別更新アプローチの比較

図 9 は兄弟ページが 10，100，1,000，10,000 ページの 4 通りの場合の兄弟一括更新のページ生成時間と，個別更新のページ生成時間を比較したグラフである．兄弟一括更新は再生成の必要なページ数にかかわらず，生成時間は一定となる．たとえば兄弟ページが 100 ページの場合，再生成の必要なページが 10 ページでも 50 ページでも兄弟ページすべてを再生成するので，ページ生成時間は 100 ページ生成する時間と等しくなる．このグラフから，兄弟ページが 10 ページの場合，再生成するページが 2 ページ以上のときは兄弟一括更新の方が生成時間が短いので，兄弟一括更新の方が有利であるといえる．同様に，兄弟ページが 100，1,000，10,000 ページの場合は，再生成の必要なページがそれぞれ 7，50，3,000 ページ以上なら兄弟一括更新の方が有利である．

6.2.2 部分木更新アプローチと個別更新アプローチの比較

図 10 は 4 レベルの Web サイトのうち，レベル 2〜4 までの 3 レベルの部分木を考え，そのページ数を **表 2** のように変化させたときの部分木更新と個別更新の生成時間を比較したグラフである．

このグラフから，部分木のページが 31 ページの場合，再生成するページが 3 ページ以上のときは部分木更新の方が生成時間が短いので，部分木更新の方が有利であるといえる．同様に，部分木のページが 211，2011 ページの場合は，再生成の必要なページがそれぞれ 15，180 ページ以上なら部分木更新の方が有利である．

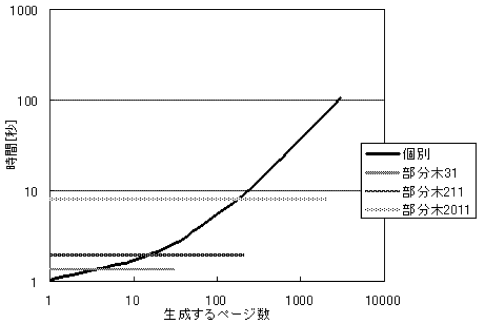

図 10 部分木更新と個別更新のページ生成時間の比較
Fig. 10 Generation time of subtree update and one-by-one update.

表 2 部分木のページ数
Table 2 Page number of subtrees.

合計ページ数	レベル 2	レベル 3	レベル 4
31	1	10	20
211	1	10	200
2011	1	10	2000

表 3 各レベルのページ数
Table 3 Page number of each levels.

レベル	ページ数
1	1
2	20
3	500
4	10,000

6.3 実験 2：最適化アルゴリズムによって得られた組合せとその他の組合せの比較

5.4.2 項で述べた最適化アルゴリズムによって得られる組合せの生成時間と，その他の組合せの生成時間を比較する実験を行った．再生成の必要なページを葉ページへ偏らせた場合と，部分木へ偏らせた場合の 2 通りで実験を行った．

この実験では，Web サイトの各レベルのページ数を **表 3** とした．このときレベル 2 のページはレベル 3 の 25 ページを子として持ち，さらにレベル 3 のページはレベル 4 の 20 ページを子として持つ．よってレベル 2 のページを親とする部分木は，レベル 2，3，4 の各ページ数が 1，25，500 で合計 526 ページとなる．

6.3.1 葉ページへの偏りがある場合

この実験ではデータベースの更新によって **表 4** のように 4 パターンにおけるページ生成時間の比較を行った．このとき，影響を受けるページが部分木に偏らないようにデータベースの更新を行った．

表 5 はそれぞれのパターンにおける最適化アルゴリズムによって得られた組合せの探索時間，生成時間，全体（探索時間+生成時間）に対する探索時間の比を示す．さらに，**表 6** は，得られた組合せとその他の組合せの生成時間を比較した表である．各パターンの最上段で太字で書かれたものが最適化アルゴリズムによって得られた組合せである．

表 4 各パターンによって影響を受けるページの数
Table 4 Number of the affected pages in each pattern.

パターン	レベル 3	レベル 4
1	10	10
2	100	100
3	500	1,000
4	500	10,000

表 5 探索時間と生成時間
Table 5 Search time and generation time.

パターン	探索時間 [秒]	生成時間 [秒]	$\frac{\text{探索}}{\text{探索+生成}}$
1	0.645	1.360	0.322
2	2.396	8.309	0.224
3	5.516	38.055	0.127
4	5.806	44.838	0.115

表 6 最適化アルゴリズムによって得られた組合せとその他の組合せの比較（葉ページへの偏りがある場合）
Table 6 Comparison of the combination obtained by the optimization algorithm and other combinations (the case that affected pages has clustered on leaf pages).

パターン	生成時間 [秒]	レベル 3	レベル 4
1	**1.360**	**個別**	**個別**
	2.202	部分木	—
	108.674	兄弟一括	兄弟一括
2	**8.309**	**兄弟一括**	**個別**
	9.698	個別	個別
	17.591	部分木	—
	108.674	兄弟一括	兄弟一括
3	**38.055**	**部分木**	**—**
	41.558	兄弟一括	個別
	49.924	個別	個別
	108.674	兄弟一括	兄弟一括
4	**44.838**	**部分木**	**—**
	106.167	兄弟一括	兄弟一括
	426.481	兄弟一括	個別
	441.079	個別	個別

表 5 から，影響を受けるページ数が多いほど生成全体の時間に対する探索時間の比は小さく，すべての場合で 3 分の 1 以下となっていることが分かる．影響を受けるページ数が少ない場合，生成時間が小さいので生成時間全体に対する探索時間の比は当然大きくなる．しかしどの場合でも探索時間は数秒程度であり，Web サイトの静的な再生成にかかる時間としては探索アルゴリズムは十分に実用的であるといえる．

次に 表 6 から，すべての場合で最適化アルゴリズム

によって得られた組合せはその他の組合せよりも生成時間が小さいことが分かる．これは，6.2 節の実験によって得られた近似式を利用することによって，ページ生成時間が最小となる組合せを探索できることを示している．

また 2 つの表から，探索時間と最適化アルゴリズムによって得られた組合せの生成時間の和を 2 番目に小さい生成時間と比較する．パターン 1，4 は前者の方が小さく，またパターン 2，3 の場合も後者よりそれぞれ約 1 秒，約 2 秒大きいだけである．同じように最も大きい組合せの生成時間と比較すると，探索時間と最適な組合せの生成時間の和の方が 55〜390 秒小さい．

Web サイトの静的生成では，動的生成とは異なり，利用者が介在しないなかでの更新となるため，少量のオーバヘッドであれば十分実用的である．探索を行わなければ，実用的でない非常に生成時間の長くなってしまう手法を選択してしまう可能性がある．したがって，少しのオーバヘッドでページの生成時間が最も小さい組合せを探索可能である点において最適化アルゴリズムが有用であるといえる．

この実験では影響を受けるページを Web サイトの葉ページに偏らせた．しかし実験結果から分かるようにパターン 3，4 のどちらの場合も最適化アルゴリズムによって得られた更新組合せは兄弟一括更新ではなく部分木更新が選択された．これは，SuperSQL システムではデータベースへの問合せ結果のサイズが大きくなるほど二次関数的に生成時間がかかることが要因であると考えられる．このため，兄弟ページが多いこの実験では，問合せ結果が大きくなる兄弟一括更新よりも，個々の問合せ結果が小さい部分木更新を何回も実行した方が，全体の生成時間が小さくなったと考えられる．この実験では，葉レベルとその上のレベルの 2 レベルのページが影響を受けたため部分木更新が選択されたが，葉レベルのページのみが影響を受けた場合，部分木更新を適用することはできないので，このような場合は兄弟一括更新が有効であると考えられる．

6.3.2 部分木への偏りがある場合

この実験ではデータベースの更新によって影響を受けるページをすべて同じ部分木に偏らせ，最適な更新組合せの探索を行った．ケース 1 とケース 2 の 2 通りの場合で実験を行った．各ケースにおける影響を受けるページ数を **表 7** に示す．

表 8 はそれぞれのケースにおける最適化アルゴリズムによって得られた組合せの探索時間，生成時間，全体（ 探索時間+生成時間）に対する探索時間の比を示す．さらに，**表 9** は，得られた組合せとその他の組合せの生成時間を比較した表である．各ケースの最上段で太字で書かれたものが最適化アルゴリズムによって得られた組合せである．

表 7 各ケースの影響を受けるページの数
Table 7 Number of the affected pages in each case.

ケース	レベル 2	レベル 3	レベル 4
1	—	1	40
2	1	10	400

表 8 探索時間と生成時間
Table 8 Search time and generation time.

ケース	探索時間 [秒]	生成時間 [秒]	$\frac{\text{探索}}{\text{探索+生成}}$
1	0.378	0.492	0.414
2	0.439	1.597	0.216

表 9 最適化アルゴリズムによって得られた組合せとその他の組合せの比較（部分木への偏りがある場合）
Table 9 Comparison of the combination obtained by the optimization algorithm and other combinations (the case that affected pages has clustered on a subtree).

ケース	生成時間 [秒]	レベル 2	レベル 3	レベル 4
1	**0.492**	**−**	**部分木**	**−**
	2.777	—	個別	個別
	108.674	—	兄弟一括	兄弟一括
2	**1.597**	**1 部分木**	**−**	**−**
	2.489	個別	部分木	—
	19.922	個別	個別	個別
	109.423	個別	兄弟一括	兄弟一括

表 8 を見るとどちらも探索時間が 0.5 秒以下である．6.3.1 項の実験に比べ探索時間が小さい理由は，影響を受けるページ数が少なく，またすべてのページが 1 つの部分木に含まれるため部分木の探索は一度しか行われないからである．たとえば 6.3.1 項の実験のパターン 2 の場合，第 3 レベルの影響を受ける 100 ページを親とする 100 個の部分木を探索するのでケース 1，2 に比べ探索に時間がかかっている．この実験では生成するページ数が少ないため，全体時間に対する探索時間の比は 6.3.1 項の実験と比べると大きくなっている．

次に 表 9 から，本実験においてもすべての場合で，最適化アルゴリズムによって得られた組合せはその他の組合せよりも生成時間が小さいことが分かる．

また 2 つの表から，探索時間と最適化アルゴリズムによって得られた組合せの生成時間の和を 2 番目に小さい生成時間と比較すると，どちらのケースでも前者の方が小さいことが分かる．同じように最も大きい組合せの生成時間と比較すると，どちらも探索時間と最適化アルゴリズムによって得られた組合せの生成時間

の和の方が約 106 秒小さい．よってこの実験からも，少しのオーバヘッドでページ生成時間が最も小さい更新組合せを探索することができ，Web サイトの静的生成の点では，最適化アルゴリズムが有用であるといえる．

この実験ではデータベースの更新によって影響を受けるページを 1 つの部分木に偏らせた．そして最適化アルゴリズムによってケース 1，2 のどちらも部分木探索が選択され，その他の更新組合せと比較し部分木更新が最適であることが分かった．よって影響を受けるページが部分木に偏っている場合，本研究で提案した部分木更新アプローチは有用であるといえる．

7. おわりに

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，実装した．SuperSQL に新たに sinvoke 関数と FOREACH 句を導入し，SuperSQL クエリを分割することによって SuperSQL で生成された Web サイトの部分生成を実現した．また，データベースの更新によって影響を受けるページを判別する手法，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．さらに，3 つのアプローチ組合せの中でページの生成時間がより小さい組合せを探索する，最適な更新組合せ探索アプローチを提案した．

実験では 3 つのアプローチのページ生成時間を比較し，各アプローチがそれぞれ異なるシナリオに適していることを示した．また，最適な更新組合せ探索アルゴリズムによって，ページの生成時間が最も小さくなるような更新組合せを探索できることを示した．

実験から生成するページ数が多い場合，兄弟一括更新アプローチよりも部分木更新アプローチの方が有効であることが分かった．しかし，本論文における最適化アルゴリズムでは，部分木更新アプローチは親ページが更新が必要と判別されていないと適用することができない．今後の課題として，親ページが判別されていなくても部分木更新アプローチを適用できるようにする必要があると考えている．たとえば，最初に SuperSQL で Web サイトを生成する際にすべてのページの親子関係を保持しこれを利用するという方法を検討している．また，本研究ではデータベースの更新によって影響を受けるページの更新はページの再生成と削除のみに限定した．Web サイトは SuperSQL で生成されるため，生成時に付加的な情報を HTML ファイルに記述することによってページの再生成，削除ではなく HTML ファイルの部分的な更新を行うこともできると考えている．

参考文献

1) Atzeni, P., Mecca, G. and Merialdo, P.: Design and Maintenance of Data-Intensive Web Sites, *EDBT'98*, pp.436–450 (1998).
2) Blakeley, J.A., Larson, P.A. and Tompa, F.W.: Efficiently updating materialized Views, *ACM SIGMOD*, pp.61–71 (1986).
3) Candan, K.S., Agrawal, D., Li, W.-S., Po, O. and pin Hsiung, W.: View Invalidation for Dynamic Content Cashing in Multitiered Architectures, *VLDB*, pp.562–573 (2002).
4) Ceri, S., Fraternali, P. and Paraboschi, S.: Data-Driven, One-To-One Web Site Generation for Data-Intensive Applications, *VLDB '99: Proc. 25th International Conference on Very Large Data Bases*, pp.615–626 (1999).
5) Fernandez, M., Florescu, D., Kang, J., Levy, A. and Suciu, D.: Catching the Boat with Strudel: Experiences with a WebSite Management System, *ACM SIGMOD*, pp.414–425 (1998).
6) Fraternali, P. and Paolini, P.: A Conceptual Model and a Tool Environment for Developing More Scalabe, Dynamic, and Customizable Web Applications, *EDBT'98*, pp.421–435 (1998).
7) Gupta, A., Mumick, I.S. and Subrahmanian, V.: Maintaining views incrementally, *ACM SIGMOD*, pp.157–166 (1993).
8) Labrinidis, A. and Roussopoulos, N.: Update Propagation Strategies for Improving the Quality of Data of the Web, *VLDB*, pp.391–400 (2001).
9) Lassettre, E.R.: Olympic Records for Data at the 1998 Nagano games, *ACM SIGMOD*, p.537 (1998).
10) Lee, K.Y., Son, J.H. and Kim, M.H.: Efficient Incremental View Maintenance in Data Warehouses, *CIKM*, pp.349–356 (2001).
11) Liu, H., Ng, W.-K. and Lim, E.-P.: Query Integration for Refreshing Web Views, *DEXA*, pp.557–566 (2001).
12) Proll, B., Strack, H., Retschitzegger, W. and Sighart, H.: Ready for Prime Time — Pre-Generation of Web Pages in TIScover, *WebDB*, pp.67–72 (1999).
13) Sindoni, G.: Incremental Maintenance of Hypertext Views, *WebDB'98*, pp.98–117 (1998).
14) Toyama, M.: SuperSQL: An Extended SQL for Database Publishing and Presentation,

ACM SIGMOD '98 International Conference on Management of Data, pp.584–586 (1998).
15) Toyama, M. and Nagafuji, T.: Dynamic and Structured Presentation of Database Contents on the Web, *EDBT'98*, pp.451–465 (1998).

(平成 17 年 3 月 20 日受付)
(平成 17 年 7 月 5 日採録)

(担当編集委員 岩山 真)

石川 恭子

平成 15 年慶應義塾大学理工学部情報工学科卒業．現在同大学大学院開放環境科学専攻修士課程在学中．データベースの研究に従事．

有澤 達也（学生会員）

平成 9 年慶應義塾大学理工学部管理工学科卒業．平成 11 年同大学大学院理工学研究科管理工学専攻修士課程修了．現在同大学院理工学研究科開放環境科学専攻博士課程在学中．データベースと構造化文書，データベース質問処理に関する研究に従事．ACM 会員．

遠山 元道（正会員）

昭和 54 年慶應義塾大学工学部管理工学科卒業．昭和 56 年同大学大学院修士課程修了．昭和 59 年慶應義塾大学工学部管理工学科助手．平成 4 年専任講師．平成 8 年情報工学科に移籍．現在に至る．博士（工学）．平成 8 年オレゴン大学院大学客員研究員．平成 10〜13 年科学技術振興事業団さきがけ研究 21「情報と知」領域研究員．主にデータベースの研究に従事．電子情報通信学会，日本ソフトウェア科学会，IEEE Computer Society，ACM 各会員．