

## XML 形式による効率的 DB キャッシュ法と評価

小林 真也<sup>†</sup> 鈴木 優<sup>††</sup> 川越 恭二<sup>††</sup>

本稿では、Web とデータベースが統合したサービスにおいて、サーバ負荷の増大によるクライアントへの応答速度の低下の問題を解決する。そのために、サーバ側でデータベースキャッシュ（以下 DB キャッシュ）を行い、応答速度の向上を図ることが可能な新しいキャッシュ法を提案する。提案手法では、サーバと直結している DB 内のテーブルをキャッシュとして XML<sup>4)</sup> 形式で保存し、次回アクセス時にそのキャッシュにアクセスする方法である。利用者がサーバにアクセスした時点で、サーバは 1) キャッシュ内に該当テーブルが存在するか、2) テーブルのデータが最新であるかについて調べ、1), 2) の両方を満たしている場合、問合せをキャッシュに対して行い、適合したデータを取得する。また、利用者から DB へデータ更新の命令が送られると、DB はそのまま更新を行い、キャッシュ側は独自に問合せを解析し、木構造を最新状態に更新する。実験の結果、従来の、更新が行われるごとにキャッシュを破棄し、再構築を行う方式と比較して、キャッシュのヒット率を約 30% 向上できた。

### The Efficient DB Cache Method with XML form and Evaluation

SHINYA KOBAYASHI,<sup>†</sup> YU SUZUKI<sup>††</sup> and KYOJI KAWAGOE<sup>††</sup>

In this paper, in order to solve a problem of lower response time as a server load becomes higher, a new cache method called Cache-DB cache, is proposed. The proposed Cache-DB cache can improve response time by employing database cache on the server side is proposed. In the cache method, cache of the table of DB is realized in a kind of databases in which data is represented in XML form. An application server first checks whether a requested table exists in cache or not, and more whether the table is recently updated or not. If the table cache exists and the cache table is the newest, the query request is executed only in the cache DB. If a data update request is sent, both database and cache DB is updated in the same time. Some results of our experiments, as compared with the existing cache method, show that the hit rate of our proposed cache method has been improved about 30%.

#### 1. はじめに

近年のインターネット利用人口の増大や、利用可能回線の拡大により、多くの人が大容量のデータにアクセスすることが可能になった。しかし、コンテンツを提供するサーバ側では、回線拡大に伴いデータの転送量が増加し、それによって負荷が増加するという問題が発生している。これは、膨大なデータアクセスがサーバのレスポンスを遅くする原因となっている。そのため、一般的には通信回線の効率化のために DB サーバ、Web サーバ、クライアント等の様々なポイントにキャッシュが構築されている。ところが、これらのキャッシュ方式では、サーバのデータの更新が行われた場合に、サーバ側の更新がキャッシュ

に反映されないため、迅速にキャッシュに適応できない問題がある。

本稿では Web と DB のサーバ間に設置するキャッシュを対象とし、サーバの負荷を軽減するための新しいキャッシュ手法である Cache-DB キャッシュを提案する。本手法では、DB が更新されたときに迅速なキャッシュへの適応ができない従来の手法とは異なり、DB が更新されると即座にキャッシュへの適応を行い、少ない処理量でキャッシュを最新のデータへと再構築することが可能となる。これによってクライアントはキャッシュへすることによって最新のデータが得られることができるため、DB の負荷の軽減につながる。また、本稿では本キャッシュ手法をより大容量の DB に対応させるため、大規模なテーブルの有効的なキャッシュ法を考える。大規模テーブルでの莫大な量のキャッシュは逆にサーバの負荷を増大させるためである。ここではどのような手法でキャッシュを保管し、いかにして限られた

<sup>†</sup> 立命館大学大学院 理工学研究科

Graduate School of Science and Engineering, Ritsumeikan Univ.

<sup>††</sup> 立命館大学 理工学部

Faculty of Science and Engineering, Ritsumeikan Univ.

量のキャッシュで効果的なアクセスが可能になるかについて考察する。

## 2. 従来のキャッシュ法

本稿で提案する手法は、DB のテーブル、またはその中の行を対象としているが、一般的なキャッシュ手法<sup>3),6)</sup>では、データの更新が行われる時、その時点で DB とキャッシュにデータの相違ができる。つまり、ここでキャッシュは DB へ任意のタイミングでアクセスしてキャッシュを再構築する必要がある。このように、DB を更新する処理が発生すると、キャッシュシステムは効率的なアクセスが行われぬ。

本稿では DB を更新する状態が発生した場合であっても、DB の更新処理とは独立してキャッシュの再構築処理を行い、キャッシュヒット率を向上させることを目的としている。

## 3. 提案キャッシュ法の概要

### 3.1 基本的な考え方

今回提案する手法では、テーブル更新時にキャッシュを破棄せずに、一定の割合でキャッシュの更新を行うことにより、ヒット率の向上を図る。キャッシュを一種の DB として実現するために、サーバ内あるいはサーバと直結している DB 内のテーブルをキャッシュとして保存する。保存形式としては、DB への問合せ処理の実現および DB との連携の必要性から XML<sup>4)</sup>を用いる。なお、効率的なキャッシュ管理の観点から、DB 単位にキャッシュを行うのではなく、1 テーブル毎に一つの XML ツリーフラグメントのキャッシュデータを作成する。

### 3.2 提案するキャッシュ法

先に述べた考え方に基づき、図 1 を用いて提案するキャッシュ方法 (Cache-DB キャッシュ) を説明する。

#### 3.2.1 利用者がキャッシュにデータの問合せを行う場合

利用者がサーバ (通常 Web サーバあるいはアプリケーションサーバ) に参照問合せを行う場合の方法を以下に示す。

- (1) サーバは最初に、問合せに関係するテーブルがキャッシュ内に存在しているか、またそれらが最新のデータであるかを検索する。
- (2) 該当テーブルが存在し、かつデータが最新であれば、サーバは送られてきた参照問合

せをキャッシュを参照できるように適用させ、該当するデータを取得する。ここで、問合せのキャッシュへの適用部分では、利用者から送られてきた SQL 言語をキャッシュの内部構造である XML に適用させるということであり、ツリーフラグメントの検索などに有効である Xpath などに変換してデータの検索を行う。

- (3) キャッシュが存在しない、もしくは最新でないキャッシュが存在していた場合には、システムは最新でないキャッシュを破棄し、DB から最新データを取得してキャッシュの再構成を行う。
- (4) もしキャッシュデータがキャッシュ領域を超えていた場合は、LRU アルゴリズムに基づいて部分的なキャッシュの破棄を行う。
- (5) 最後に キャッシュ情報や DB 情報が記載されている索引に最新データ状況を更新して、一連の処理を終了する。

#### 3.2.2 データベースにデータの更新が発生する場合サーバに利用者からデータ更新命令が送られた場合の方法を以下に示す。

- (1) サーバはまず DB への更新を通常どおり行い、DB の更新情報を更新する。
- (2) 同時に DB 処理とは独立してキャッシュの更新を行う。しかし、本方式ではキャッシュ更新率というものを設けて、キャッシュを更新させない場合も存在させるようにしている。キャッシュ更新率に関しては 3.3.2 節で解説する。
- (3) キャッシュを更新することが確定すると、参照問合せと同じく、SQL 言語で送られてきたデータ更新の問合せの解析を行い、それに適用したキャッシュ内の更新部分を最新データに更新する。この処理は完全に DB に依存せずに行う。
- (4) 最後に、キャッシュの最新情報を更新して作業を終了する。

#### 3.2.3 その他の命令が送られてきた場合

SQL 言語には多数の命令があり、上記の二つの場合以外の命令も送られてくる可能性がある。その中で、DB スキーマを変更する命令が送られてきた場合には、DB と キャッシュ の整合性を考慮し、キャッシュの破棄を行う。それ以外の命令の、本方式だけでは対応してない構文が送られてきた場合は、DB 部分だけの実行を行う。

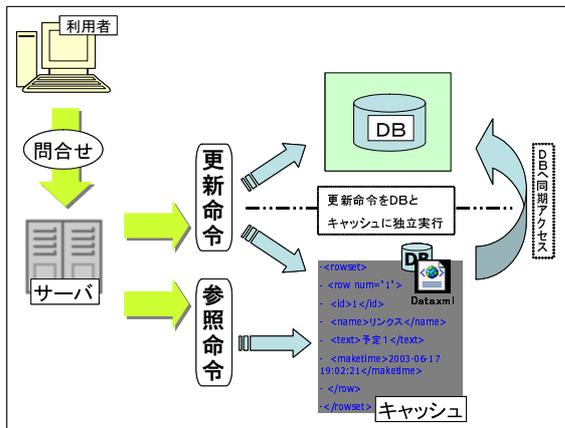


図 1 提案キャッシュ法の概要

Fig. 1 Overview of the proposed cache method.

以上のパターン操作を、一連の図として表現したものを図 2 に示す。

### 3.3 キャッシュ容量と更新率に関して

#### 3.3.1 キャッシュ容量の制御

通常、キャッシュシステムは有限のキャッシュ領域を持っている。これを設定していなければキャッシュが無限に作成される可能性があり、ディスクの容量を圧迫させるだけではなく、メモリが大量のデータを読み込み、キャッシュのアクセス効率にまで影響を及ぼす可能性がある。そこで、提案するシステムでもキャッシュ容量が定められている。ここではキャッシュを更新した際に全体のキャッシュの容量がキャッシュ許容量を超えていた場合に、メモリキャッシュを管理するために一般に利用されキャッシュ利用効率の良い LRU (Least recently used)<sup>5)</sup> に基づいてキャッシュを削除するという手法を用いる。この目的のために、テーブル単位に最新更新時刻をインデックス内の情報として保存する。

#### 3.3.2 キャッシュ更新率

ヒット率を向上させるキャッシュシステムを実現するために、更新作業において DB を更新した際のキャッシュを更新する処理を、一定の間隔で行う<sup>7)</sup>。このため、DB は更新されても、キャッシュは更新されない場合が発生する。このキャッシュ更新を行う割合を“キャッシュ更新率”と呼ぶ。

例えば、キャッシュ更新率を 25% と設定した場合、本システムにデータ更新命令が送られてくると、DB は更新されるが、キャッシュは四分の

```

If (データ取得クエリ){
If (キャッシュが存在し、最新){
・クエリを XML 対応形式に変換
・キャッシュからデータを取得
} else {
・DB から最新データを取得しキャッシュを更新
・キャッシュが一定の容量を超えていたら
LRU に基づき該当キャッシュを削除
・index(キャッシュ情報, DB 情報)を更新
}
} elseif (データ更新クエリ){
・DB にアクセスしてデータを更新
・index(DB 情報)を更新
・キャッシュ更新率によりキャッシュを更新するか否かを決定
If (キャッシュの更新が確定){
・クエリを XML 対応形式に変換
・キャッシュを更新
・キャッシュが一定の容量を超えていたら
LRU に基づき該当キャッシュを削除
}
} else {
・DB に直接実行
}
}

```

図 2 提案キャッシュ法の基本アルゴリズム

Fig. 2 Basic algorithm of the proposed cache method.

一の確率でしか更新されないということになる。このキャッシュ更新率を上げるとキャッシュヒット率は向上するがキャッシュ処理量も多くなる。逆にキャッシュ更新率を下げると、キャッシュ処理量は軽減されるが、キャッシュヒット率は低下することになる。

#### 3.4 DB とキャッシュ間での同期について

本キャッシュシステムでは DB とキャッシュ間で永続的な相違が発生しないように、簡単な同期管理を設ける。ただし、あくまでキャッシュシステムであるため、完全なトランザクション管理を実装して、DB システムに近づけてしまうとキャッシュ自体の柔軟性を損なってしまう可能性がある。そのため、本システムでは DB とキャッシュ間で、想定外のデータの相違などの事態の発生を感知すると、キャッシュ特性を利用してキャッシュの破棄を行う。

本システムでは、キャッシュと DB との間に index と呼ばれる、キャッシュのデータ容量や、DB の更新時間などを記録したキャッシュを持つ。DB が更新されるとまず本システムは、キャッシュにアクセス前に、この index を更新する。それにより、DB 情報をいち早く更新して、キャッシュとの相違性を示し、キャッシュへの異なったアクセスを最低限回避する。

また、キャッシュが作成される条件として、必ず DB で SQL のコミットが確定された後となっている。万が一 DB で SQL がアボートした場合、

システムはキャッシュの破棄を遂行する。

#### 4. 大規模キャッシュへの対応

本章では大規模テーブルをキャッシュ化の際の問題点と、それについての解決方法を示す。

##### 4.1 大規模テーブルのキャッシュ化の問題点

3.2 節で DB テーブルをキャッシュ化してキャッシュヒット率を向上させる手法を述べたが、ここで、キャッシュ化される DB テーブルは DB テーブル一つに対して一つのキャッシュテーブルデータが構築される。つまり、テーブル数の数だけキャッシュが存在する。

キャッシュ対象の DB テーブルが小規模な場合は、テーブル全体をキャッシュ化する方法でも効率的なキャッシュは可能だが、キャッシュ対象のテーブルが大規模な場合は、提案手法では有効なキャッシュは望めない。なぜなら、本キャッシュ手法では、キャッシュを読み込む際に大量のデータを読み込んでしまい、メモリに負荷が生じてしまう可能性があるためである。

##### 4.2 大規模キャッシュの解決案

大規模テーブルをキャッシュ対象とする場合では、メモリに負担が大きく、効率性が悪い。そのため、この場合では 1 テーブル全体をキャッシュ化するのではなく、別の要素をキャッシュ対象としてキャッシュ化しなければならない。

DB テーブルに関係するキャッシュ対象として以下に三つの手法を挙げる。

- テーブル全体をキャッシュとする手法
- SQL 命令によって得られた結果行をキャッシュ対象とする手法
- SQL 命令とそれによって得られた結果をキャッシュの 1 要素とする手法

この三手法についてのそれぞれの利点と欠点を解説していく。

##### 4.2.1 テーブル全体をキャッシュ対象とした場合

この手法では、テーブル全体のキャッシュ化を行う。

利点: 1 テーブルすべてがキャッシュ化されるので、利用者から送られてきた参照問合せによって参照されるデータは、キャッシュ容量によって一時的にキャッシュが削除されていない限り、ほぼ確実にキャッシュテーブルに存在している。よって、キャッシュヒット率は極めて高い結果となる。

欠点: 先にも述べたが、大規模な DB テーブル

をキャッシュ化し、利用者がキャッシュに参照する場合には、サーバはテーブル内の全データに対して、検索を施行するので、メモリの負担が増大し、処理効率が減少することがある。

この手法では小規模な DB テーブルをキャッシュシステムとして扱う場合に有効である。

##### 4.2.2 SQL 命令によって得られた結果行をキャッシュ対象とする手法

この手法は、SQL 命令で得られた結果の行をキャッシュとして保管する方法である。

利点: 4.2.1 節の手法のように、テーブル全体をキャッシュせず、得られた結果順にキャッシュとして格納していく手法のため、大規模な DB テーブルの場合でもメモリに負担がかからず、処理量が少ないという利点がある。

欠点: 最新のアクセスデータがキャッシュに保存されるが、ここで、別の SQL 言語が利用者から送られてきた場合に、返されるべき結果がすべてキャッシュだけでは存在していない可能性が高い。そのためサーバは、実行結果をキャッシュとその差分を DB から取得することになる。

##### 4.2.3 SQL 命令とその結果行をキャッシュ対象とする手法

最後に提案する格納手法では、利用者から送られてきた SQL 命令をキャッシュの検索キーとして、その結果一覧を一つの要素として扱う手法である。この格納手法についての概要図を図 3、4 に示す。

利点: 4.2.2 節の手法の利点と同じく、大規模 DB のキャッシュにも適用が可能となる。さらに、一つの SQL 命令をキーとして、結果行すべてを一つのキャッシュ要素としているため、異なる SQL 命令が送信されても、キャッシュと DB の二つに分散してアクセスするという処理が無くなり、どちらかに一意にアクセスされる。

この手法で SQL 命令をキーにすることにより、さらに検索効率も向上する。ここではキーを参照命令のデータを特定する部分 (SQL 言語では Where 句以降) としている。そのため、キャッシュを参照する SQL 命令が送られてきても、従来のように構文の中身を詳しく検証してキャッシュに適用しなくても、構文自体をそのままキーとして検索を行

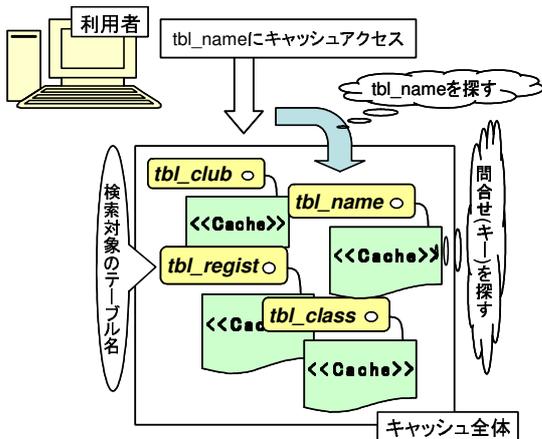


図3 SQL と結果行をキャッシュ要素とした格納手法の概要図  
 Fig.3 Overview of the storing method using cache elements and SQL queries.

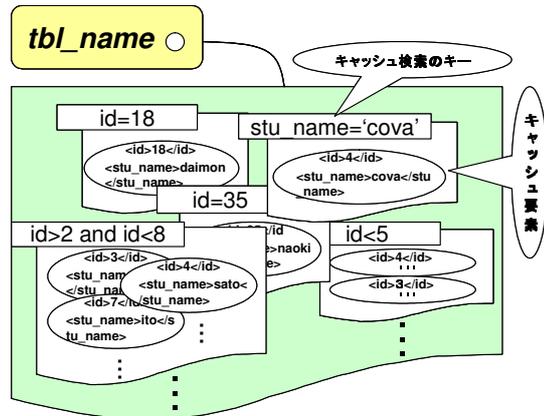


図4 図3のキャッシュテーブルの内部の例  
 Fig.4 An example of an inside of cahce with fig 3.

うことができるので、処理効率が向上し、また、いままで処理できなかった複雑な SQL 命令も扱えることが可能となった。

欠点: SQL 命令をキーとしているため、利用者がアクセスしても、キャッシュされている要素の中でその SQL のキーが存在しなかった場合はキャッシュミスとなる。当然、そのアクセスは新たにキャッシュ要素として保存されるが、キーが存在しない場合は、例えばキャッシュ内に望むデータが存在していてもキャッシュからは得られないという欠点がある。そのため、SQL 命令の異なるアクセスが多い場合はあまり効果的ではない。さらに、同じデータが複数存在し、キャッシュデータの冗長性が高くなる可能性もある。

つまり、アクセスされる SQL 命令が局所的な場合に有効性が高い。

今回の実験では大規模 DB に関しては 4.2.3 節の手法を、そうでない場合には 4.2.1 節の手法を用いて評価を行う。ここで、4.2.3 節の手法についての実装に関する基本アルゴリズムを図 5 に示す。この図は、図 2 に部分的な変更を加えたものである。

### 5. 評価および結果

提案方式のキャッシュヒット率の効率性を調べるため、(1) 更新が発生するたびにキャッシュを破棄する従来方式 (Check and destroy)、(2) 提案方式であるキャッシュを連続的に更新していく方式 (Continuous update) について、両者を比較

```

If (データ取得クエリ){
  If (送られてきた問合せがキャッシュ内のキーに存在し、かつ最新){
    ・キャッシュからデータを取得
  }elseif (問合せのキーが存在しないが、キャッシュテーブルは最新){
    ・DB から最新データを取得しキャッシュを更新
    ・キャッシュが一定の容量を超えていたら
      LRU に基づき該当キャッシュの一部を削除
    ・index(キャッシュ情報)を更新
  }else {
    ・キャッシュを削除
    ・index(キャッシュ情報)を更新
  }
}elseif (データ更新クエリ){
  ・DB にアクセスしてデータを更新
  ・index(DB 情報)を更新
  ・クエリを XML 対応形式に変換
  ・キャッシュを部分更新
  ・キャッシュが一定の容量を超えていたら
    LRU に基づき該当キャッシュを削除
  ・index(キャッシュ情報)を更新
}else {
  ・DB に直接実行
}
  
```

図5 大規模キャッシュ法の基本アルゴリズム  
 Fig.5 Basic algorithm of the large-scale cache method.

する実験を行った。ここで、大規模 DB のキャッシュの効果を評価するため、両キャッシュ法に対し、(a) テーブル全体をキャッシュして保存する格納方式と、(b) SQL 命令とその結果行をキャッシュする方式についても評価を行った。

#### 5.1 キャッシュ手法の実現

本提案方式におけるキャッシュは木構造である XML 形式で構築する。この木構造の操作を容易にするため、開発言語として XI (ザイ)<sup>1)</sup> を用いた。XI は、Web アプリケーション開発に特化したプログラミング言語であり、XML との親和性が非常に高いという特徴を持つ。

本提案方式では、DB に入力された SQL 言語による問合せを分解し、XI のツリー制御の構文に

変換する．特に，SQL 言語での対象となるデータを特定する構文（Where 句など）は，XI におけるツリー内のノードの位置を指定する記述形式（XPath）に変換する．

## 5.2 実験環境

この実験では，(2)のキャッシュシステムでは(a)の格納方式に関して，キャッシュ更新率を50%に設定した．これは，データベースが更新される際に50%の割合でキャッシュも同時に更新することを意味する．(b)の方式に関しては，キャッシュ容量の許す範囲ではキャッシュ更新率を100%に設定している．これは，(b)の方式ではキャッシュは最初は存在せず，参照のSQL命令が実行されるたびに逐次追加されているため，途中でキャッシュを削除することによってキャッシュ効率が著しく悪化すると考えたためである．

なお，実験に使用するDBとしてHSQL<sup>2)</sup>を用いた．HSQLはピュアJavaによる単純な仕組みのDBのため，一般的なDBMSより動作が速いという特徴がある．また，実験で行う上での基礎となるサーバアプリケーションは横浜ベイキット<sup>1)</sup>で配布されているBayServerを使用した．

## 5.3 実験データ

実験データとして，単純なデータを含む5種類のテーブルを作成した．各テーブルは数個の整数型あるいは文字列型の列を有する．その作成したテーブルに，ランダムなDBへのアクセス命令（更新命令を含むSQL命令）を連続で実行し，キャッシュヒットによるキャッシュへのアクセス回数を測定した．DBへのランダムな命令では，一定の割合でDBの更新を行う命令を含んでいる．ここで言う更新命令とは削除(DELETE)，挿入(INSERT)，更新(UPDATE)の命令を指す．データ数に関しては，大規模DBの実験では各テーブルごとに10000件とし，それ以外では500件から2000件とした．

## 5.4 実験結果と考察

図6，図7，図8に実験結果を示す．また，図9図10に作成されたキャッシュの一部を示す．

### 5.4.1 キャッシュヒット率の結果

図6はキャッシュ更新率を変えずに，データの件数を変化させたときのヒット率である．双方の結果における左部のグラフ棒が従来のキャッシュ破棄方式(Check and destroy)，右部が今回提案方式(Continuous update)による結果である．図中の縦軸はキャッシュのヒット率を示しており，

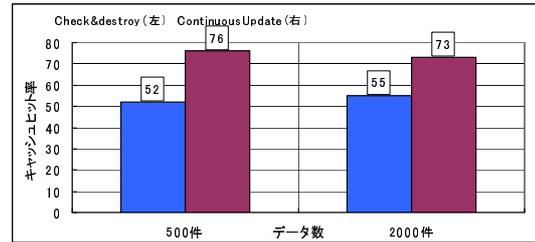


図6 (a)手法のDBのデータ数の変化によるキャッシュヒット率  
Fig. 6 Cache hit rate with change of DB data size of (a).

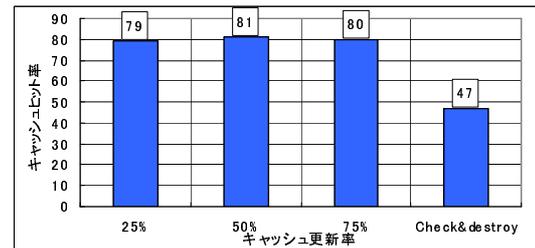


図7 (a)手法のキャッシュ更新率の変化によるキャッシュヒット率  
Fig. 7 Cache hit rate with change of cache update rate of method (a).

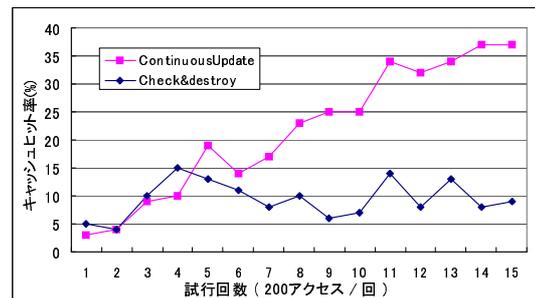


図8 (b)手法の，SQLと結果をキャッシュ要素としたときのキャッシュヒット率  
Fig. 8 Cache hit rate with use as the cache element and SQL of method (b).

棒上の数字は実際のヒット率の値である．横軸は二つのデータ件数比較である．左部が500件のデータ数で行った実験の結果，右部が2000件のデータ数で行った実験の結果である．図に示すように提案方法のキャッシュヒット率が従来手法と比べて約30%高い結果を得た．

図7はキャッシュの更新率を変化させたときの結果である．グラフは左からキャッシュ更新率を20%，50%，75%としたときの結果であり，最も右に示すグラフは従来手法(Check and destroy)の結果である．ここではデータ数を1000件とした結果である．提案方式のキャッシュヒット率は，従来の方式と比較して高いことがわかる．また，

この結果からではキャッシュ更新率によって大きな変化は見られなかった。しかし、キャッシュ更新率が 50% 付近に、わずかにレスポンス時間が短い結果となった。

図 8 は大規模 DB のキャッシュを考慮した、SQL 命令とその結果をキャッシュ要素としたときのキャッシュヒット率を表したグラフである。この実験では大規模 DB を考慮してデータ数を 10000 件として評価を行った。グラフの縦軸はキャッシュヒット率を示しており、横軸では DB への試行回数を表している。この試行回数では DB への 200 回のアクセスを 1 試行回としている。グラフでは、従来手法 (Check and destroy) ではキャッシュヒット率に大きな変化は見られないが、提案方式 (Continuous update) では試行回数を増すごとにキャッシュヒット率が上向いていく結果となった。

#### 5.4.2 効率、非効率パターンについての考察

この実験では完全にランダムな SQL 命令を DB に与えた。ただ、現実ではシステムによって SQL の命令は偏った場合が多い。そこで、この実験を通じてキャッシュの特性を活かした効率のよい命令パターン、逆に非効率なアクセスの命令パターンについてを考察する。

本キャッシュシステムで、最も効率のいい命令パターンは、更新が少なく、また局所的な参照が多い場合である。これは、従来手法にも当てはまるが、今回の手法では多くの更新要求があった場合でも、キャッシュが削除される従来手法と異なり、キャッシュの領域を超えない範囲ではアクセス効率が保たれる点がある。逆にキャッシュ対象になっているデータへの包括的な参照は効率が悪くなる。アクセスするユーザが、すべて別々のデータを参照するパターンはワーストケースである。このような包括的参照の場合では、キャッシュ領域が限られている本キャッシュシステムは有効ではない。

さらにこの実験で、参照命令と更新命令の処理手順によって、処理効率が変化することがわかった。最も良いパターンでは参照命令と更新命令がそれぞれ固まっている場合であった。逆に参照命令と更新命令が互いに実行される場合は最も効率が悪くなった。これは、キャッシュの更新率によるところが大きく、更新命令によって一定の確率でキャッシュが更新されなかった場合では、次に発生する参照命令は必ずキャッシュミスになるためである。

## 6. おわりに

本稿では、サーバ側での新しい DB キャッシュ方法 (Cache-DB キャッシュ) を提案した。本試作システムによりキャッシュヒット率が向上し、従来手法に比べ、より効率的なアクセスが可能となった。

今後の主な課題は、システムの処理速度の向上と、より複雑な SQL 命令の対応である。システムの処理速度に関しては XML キャッシュの読み込みとパースを処理量を軽減することと、システムのコア部分の開発言語の変更を目標とする。今回、開発に使用した XI という言語は XML 操作には非常に容易で有効であるが、他の言語に比べて処理速度がやや遅いという欠点がある。複雑な SQL 命令の対応については、XML と SQL の関連性<sup>8),9)</sup>などを研究して、本システムの改善を目指していく。

また、今後の目標として、複数のサーバによる XML キャッシュの分散処理なども行っていく。これは、キャッシュを複数の場所に配置し、ユーザがより近く、より迅速にキャッシュにアクセスできるようにするためである。

## 参 考 文 献

- 1) 横濱ベイキット。  
<http://www.baykit.org/index.xi>.
- 2) Hypersonic SQL。  
<http://hsqldb.sourceforge.net/>.
- 3) Barish, G. and Obraczka, K.: *World Wide Web Caching: Trend and Techniques*, Internet Technology, IEEE Communications (2000).
- 4) Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E. and Yergeau, F.: *Extensible Markup Language (XML) 1.0 (Third Edition)*, Technical report, World Wide Web Consortium (2004). <http://www.w3.org/TR/2004/REC-xml-20040204>.
- 5) Mattson, R. L., Gecsei, J., Slutz, D. R. and Traiger, I. L.: Evaluation techniques for storage hierarchies, *IBM System Journal*, Vol. 9, No. 2, pp. 78 - 117 (1970).
- 6) Selcuk, K., Li, W.-S., Luo, Q., Hsiung, W.-P. and Agrawal, D.: Enabling Dynamic Content Caching for Database-Driven Web Sites, *SIGMOD RECORD*, Vol. 30, No. 2, pp. 532 - 543 (2001).
- 7) 林秀樹, 原隆浩, 西尾章治郎: アドホックネットワークにおけるデータ更新間隔を考慮した

キャッシュ無効化について, 情報処理学会 データベースシステム研究報告, Vol. 2002, No. 1, pp. 1 – 8 (2003).

- 8) 油井誠, 森嶋厚行: PostgreSQL を用いた多機能な XML データベース環境の構築, 情報処理学会 論文誌データベース, Vol. 44, No. SIG12 TOD19 (2003).
- 9) 久保田和己, 金政泰彦, 石川博: XML 問合せ処理システム ( XQues ) の問合せ処理系, 情報処理学会報告書, Vol. 99, No. 61, pp. 25 – 30 (1999).

```
<rowset>
<ut>2004/05/17 18:20:11</ut>
<row num="1">
  <id>1</id>
  <stu_name>name1</stu_name>
</row>
<row num="2">
  <id>2</id>
  <stu_name>name2</stu_name>
</row>
<row num="3">
  <id>3</id>
  <stu_name>name3</stu_name>
</row>
```

図 9 (a) 手法によって作成されたキャッシュ (一部)

Fig. 9 An example of cache data of method (a).

```
<rowset>
<ut>2004/06/02 13:57:06</ut>
<q query="id=2068">
  <ut>2004/06/02 13:58:57</ut>
  <row num="1">
    <id>2068</id>
    <stu_name>name2068</stu_name>
  </row>
</q>
<q query="id=2026 and id=732">
  <ut>2004/06/02 13:58:57</ut>
  <row num="1">
    <id>2026</id>
    <stu_name>name2026</stu_name>
  </row>
  <row num="2">
    <id>732</id>
    <stu_name>name732</stu_name>
  </row>
</q>
<q query="name='name1378'">
  <ut>2004/06/02 13:58:57</ut>
  <row num="1">
    <id>1378</id>
    <stu_name>name1378</stu_name>
  </row>
</q>
```

図 10 (b) 手法によって作成されたキャッシュ (一部)

Fig. 10 An example of cache data of method (b).