SuperSQL処理系における グルーピング木に沿った質問分割手法

有澤 達也† 遠山 元道‡

†慶應義塾大学大学院 理工学研究科 計算機科学専攻/ 慶應義塾大学 ITC 本部
‡慶應義塾大学 理工学部 情報工学科
E-mail: † ari@db.ics.keio.ac.jp, ‡ toyama@ics.keio.ac.jp

データベース出版に利用される SuperSQL では構造化データの取得が全体の処理時間に大きく影響する。グルーピング操作のためのソートを SQL の ORDER BY 節で行う手法を提案したが、複数のグルーピングが存在する場合に単一の SQL でデータ取得を行うと、中間結果が膨大になるなど問題が多い。

本稿では、SuperSQL 質問文の構造を示すグルーピング木に従って複数の SQL に分割した上で、その SQL に ORDER BY 節を付加することにより、データ取得を行う手法を提案する。

キーワード : SuperSQL, 関係データベース, 質問処理, 最適化, 構造化

A Grouping Tree based decomposition technique for SuperSQL query processing

Tatsuya Arisawa † Motomichi Toyama ‡

†Department of Computer Science, Faculty of Science and Technology, Keio University.
Information Technology Center, Keio University.
‡Department of Information and Computer Science, Faculty of Science and Technology,
Keio University.
E-mail : †ari@db.ics.keio.ac.jp ‡toyama@ics.keio.ac.jp

SuperSQL is an extended SQL for database publishing, which construct nested data as intermediate result. The time to make nested data is major factor in whole processing time of SuperSQL. Formerly we proposed the technique which adds the proper ORDER BY clause to a SQL to sort for grouping process. But when a SQL has plural groupings, it causes enormous intermediate result.

In this paper, we propose to decompose SuperSQL query into plural SQLs based “Grouping Tree”, which indicates grouping structure of SuperSQL query.

keyword : SuperSQL, relational database,
query processing, optimization, data restructuring

1 はじめに

SuperSQL[1, 2] は、データベース出版を目的とした問い合わせシステムである。SuperSQL では、取得する属性間に階層構造に関する情報を付加することによって関係データベースからの問い合わせの結果をグルーピングし構造化する。また、同時に記述されたメディア指定や次元情報、装飾情報により、この構造化データは HTML 等の様々な媒体用の応用データに変換される。

この構造化データはキー値の等しいものをまとめた簡潔な状態で表現される。しかしながら、関係データベースからの検索結果はもともとフラットなものであって、構造化データを生成するために必要なデータは膨大なものとなる。グルーピング処理にはソートを行う必要があり、フラットな検索結果が大きくなるにつれて、グルーピング処理にかかるコストが膨大となった。特に、SuperSQL 質問文では、複数の検索結果を組み合わせて一つの質問文に表すことが多い。この状況では、検索結果は個々の検索結果の直積を生じていた。この問題に対し、SuperSQL 処理系では、複数の SQL に質問を分割を行いながらデータを取得するして中間結果を抑制する手法 [5, 6] が提案され、その後、ソート処理を ORDER BY 節を付加することによってデータベースの出力時点で行う手法 [4] が提案された。

そこで本稿では、これら二つの手法を組み合わせた Hybrid 手法の提案を行う。すなわち、SuperSQL 処理系の質問処理において、結果のデータ構造を示すグルーピング木を利用することで ORDER BY 節を付加した複数の SQL に質問文に分割し結果を組み合わせるといった手法を提案する。その手法を実装することにより、内部中間結果の抑制およびグルーピング操作の簡略化による構造化データ取得のコストを低減することを目指す。

2 SuperSQL

SuperSQL 質問文は以下のように、SQL の SELECT 節をメディア指定とターゲットリストの拡張である Target Form Expression (以下 TFE) から構成される GENERATE 節へと拡張したものを用いる。

```
GENERATE   < メディア指定 > < TFE >
FROM       < FROM 節 >
WHERE      < WHERE 節 > ...;
```

メディア指定には XML[3],LATEX,HTML,JAVA 等の出力先媒体を指定することができ、同一レイアウト構造を持つ複数の出力媒体への出力を容易に生成可能となっている。

また、TFE は通常の SQL のターゲットリストにでてくるデータベース属性に対して、結合子によってレイアウト方向を指定したり、反復子によるデータのグルーピング、そして「@{}」により装飾情報を付加することができる。

結合子は属性等を「,」(縦)、「!」(横)、「%」(深さ)、「#」(時間) で区切ることによって、それぞれの方向にレイアウトすることを意味する。また、SuperSQL の持つ強力な表現力の基となるグルーピングは、反復子を用いて記述する。反復子は、反復させたい部分を '[ ]' で囲み、その直後に反復方向を結合子と同じ記号で記述することで指定する。

以下の説明に用いるため、図 1 のようなデータベースにおいて、図 2 に示す SuperSQL 質問文を例示する。(a) は属性 { 店名 } の値ごとに複数のグルーピングがある例、(b) は複数の SQL の検索結果を並べて表示する例である。

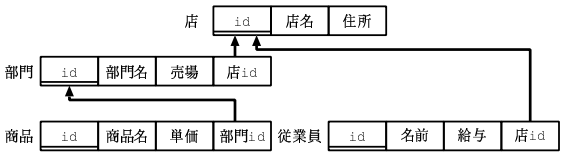

図 1: リレーション例

```
(a) GENERATE HTML
      [s.店名,
         [d.部門名,[i.商品名]!]!, [e.従業員名]!
      ]!
    FROM   店 s, 部門 d, 商品 i, 従業員 e
    WHERE  s.id = d.店id AND d.id = i.部門id
    AND    s.id = e.店id;

(b) GENERATE HTML
      [d.部門名,[i.商品名]!]!, [e.従業員名]!
    FROM   部門 d, 商品 i
    WHERE  d.id = i.部門id
    AND    e.給与 > 20000;
```

図 2: SuperSQL 質問文の例

SuperSQL 処理系では、図 3 に示すような流れで処理を行う。この中で、「データ構造化部」は反復子の指定に従って、キー属性の値ごとに値をまとめるグルーピング操作を行う。この操作にはキー属性でソートすることが必要であり、処理系全体のコストに大きく影響する。このため、グルーピング操作に着目した効率化の研究がなされてきた。

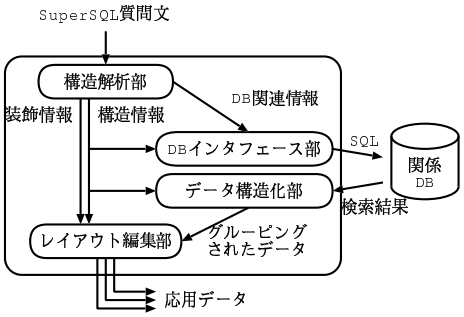

図 3: SuperSQL 処理系の処理図

2.1 グルーピング木

SuperSQL では反復子の入れ子関係のみが属性のグルーピング操作を指定していて、結合子やその他の装飾情報はグルーピング操作には必要ない。そこで SuperSQL 質問文のグルーピングに関する部分だけを表現した「グルーピング木」を定義した [4]。これは、どの属性がどの属性 (集合) によってグルーピングされているかを木構造を用いて表現したものである。グルーピングだけを表現しているので、TFE における結合の順序や結合や反復の次元は無視される。

TFE T に対するグルーピング木 G は再帰的に以下のように定義される。

1. TFE T の反復子で囲まれた部分をキーとなるデータベース属性 $\{a_1, a_2, \cdots\}$ とグルーピングしている TFE$\{T_1, T_2, \cdots\}$ にわける。
2. $\{a_1, a_2, \cdots\}$ のラベルを持ったノード N を生成する。
3. $T_1, T_2, \cdots$ のそれぞれに対して再帰的に 1. に入力し、得たグルーピング木 $\{G_1, G_2, \cdots\}$ を N の子ノードにし、全体をグルーピング木 G とする。

例えば、図 2 の SuperSQL に対するグルーピング木はそれぞれ図 4 のようになる。

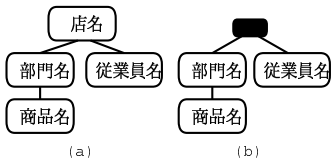

図 4: SuperSQL に対応するグルーピング木

3 従来の質問処理手法

これまで SuperSQL 処理系では、複数の SQL に質問を分割するといった手法、そして ORDER BY 節を SQL に付加する手法といった、質問文処理に関する研究がなされてきた。これらの研究について、以下で説明する。

3.1 質問文を分割する手法

質問文を分割する手法 [5, 6] では、TFE の同一の階層に複数のグルーピングが存在する場合、グルーピング間の組み合わせの情報は構造化された出力に必要がないということに着目している。単一の SQL でこれらの属性を取得した場合に、この組み合わせ情報を含んだ検索結果を得るため、タプル数が膨大になり、ソートにかかるコストが大きくなる。

例えば図 2(a) の SuperSQL 質問文では、出力結果において、属性集合 { 部門名, 商品名 } と { 従業員名 } の組み合わせ情報は必要がない。しかし、単一の SQL で問い合わせた場合はそれぞれ別の表への結合によって検索結果を得るため、店名の値ごとに属性 { 部門名, 商品名 } と { 従業員名 } の直積を検索結果に生じる。

そこでこの研究では、必要のない組み合わせ情報を減少させるために、属性を全て単一の SQL で問い合わせるのではなく、複数の SQL に分割して問い合わせ、その検索結果を組み合わせることで構造化データを生成するといった手法について研究されてきた。この SQL の分割によって、データベースからの検索結果を大幅に削減することにより、比較的大きな質問文に対してグルーピング操作にかかる時間を削減した。

3.2 ORDER BY 節を SQL に付加する手法

一方、ORDER BY 節を SQL に付加する手法 [4] では、グルーピングに必要なデータのソートはデータが関係データベースから表として抽出された時点で必要なソートが終わっていれば、グルーピング時にソートは必要無い [8] ということに着目した手法である。特に SuperSQL では、必要な SQL で 1 対多の結合を伴うことで、データベース上でインデックスを利用していることが多い。

そこでグルーピング操作に必要なソートに従って ORDER BY 節を検索に用いる SQL に付加する手法を提案した。これにより、データベースへの負荷が若干大きくなるものの、グルーピング操作において最もコストのかかっていた、データベースからデータを取得した後のソートを簡略化することが可能になった。フラットな検索結果からのグルーピング

操作では、グルーピング木のルートノードに含まれる属性からソートが必要になるため、ORDER BY 節に属性を付加する順序は、グルーピング木のルートノードに近い順で与えれば良い。

しかしながら、グルーピング木のあるノードに複数の子ノードが存在した場合は、必要なソートを ORDER BY 節で全て指定することができない。このような場合、単一 SQL 内で最も有効的なソート処理を行うために、一つのパスの属性に ORDER BY 節を付加し、他のノードの属性はグルーピング操作時にソートを改めて行うといった手法で行われた。

例えば図 4(a) のグルーピング木においては、

ORDER BY s.店名, d.部門名, i.商品名

と指定し、従業員に関してはグルーピング時にソートを行った。

この手法では、ソートにかかるコストは減少するものの、関係データベースからの検索結果自体は減少しない。

4 グルーピング木を用いた質問文の分割

3 章で述べてきた二つの最適化手法は、それぞれ異なった前提において構造化データの取得までのコストを削減していた。特に適用できる TFE の構造について着目すると、前者は同一階層に複数のグルーピングが存在する際にのみ適用することが可能であるということに対し、後者はそのような TFE では ORDER BY 節のみではソートを完全に行うことができない。

そこで、本稿では同一階層に複数のグルーピングを含むような SuperSQL 質問文が与えられた時に、この二つの手法を組み合わせて利用する Hybrid 法を提案する。つまり、まず質問文の分割を適用することで同一階層に複数のグルーピングが存在しないようにし、その後 ORDER BY 節を付加するといった手法である。

以下では、この質問文の分割に際して、どのような状態のときにどのように属性を分割して取得すべきかをグルーピング木を用いて述べる。そして、このそれぞれの分割取得のための SQL の生成、および分割取得後の構造化データへの合成について述べていく。

4.1 グルーピング木による質問文の分割

グルーピング木の定義から、あるノードに対して複数の子ノードが存在する時、同一階層に複数のグルーピングが存在することになる。つまり、3.1 節で述べたように、これらの子ノード間の組み合わせの関係は出力には不必要である。このような組み合わせが存在しないように以下のように複数の SQL に分割を行う。

- ルートノードから葉ノードまでの全てのパスに対して、それぞれのパスに含まれるノードの属性を検索する SQL を生成する。

このように分割することで、生成結果の合成に必要なキー値を同時に取得することができる。

例えば、図 4(a) のグルーピング木の場合では、属性 { 商品名 } と { 従業員名 } の複数の葉ノードが存在するので、図 5(a) のグルーピング木のように、ルートノードからその葉ノードまでのパスに含まれる属性集合 { 店名, 部門名, 商品名 } と { 店名, 従業員名 } の二つの属性集合に分割を行う。

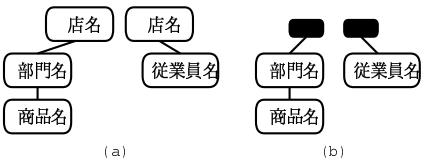

図 5: 分割されたグルーピング木

また SuperSQL 質問文では、独立な複数の SQL からなる検索結果をレイアウトして表示するといった場合に、図 4(b) のようにグルーピング木がルートノードから複数のノードに分かれているものがある。このような質問文の場合、ルートノードから葉ノードまでのそれぞれパスには、元々の SQL ごとに属性が含まれることになる。

4.2 複数の SQL への分割と ORDER BY 節の付加

グルーピング木によって同時取得を行う属性集合を決定した後、SQL の生成を行う。この SQL は元の SuperSQL 質問文の一部のみを取得するため、属性集合とは関連がない条件式を取り除くことが必要である。そこで、‘‘AND’’ によって WHERE 節にある条件式を区切り、この属性集合に関連している部分条件式を選択し、属性集合に対応した SQL を生成する。このアルゴリズムは以下のようになる。

1. 与えられた属性集合を SELECT 節に並べる。
2. 与えられた属性集合の属する表集合を R とする。
3. R に含まれるある表から R に含まれない表への条件式が WHERE 節に存在するかを一通り調べ、存在するならばその表を R に付け加える。これを R が変化しなくなるまで繰り返す。
4. この時の R を FROM 節に並べる。
5. R に含まれる表を参照する条件式を抽出し、WHERE 節に加える。

例えば、図 2(a) の SuperSQL の場合では、二つの属性集合どちらから見ても、全ての条件式が関連しているため、WHERE 節を減らすことができない。一方、図 2(b) の SuperSQL の場合では、{ 部門名, 商品名 } に給与の条件は関連していないことや、{ 従業員名 } に対して結合条件が関連していないため、WHERE 節の条件式から必要な条件式のみを抽出した以下のような SQL を生成する。

```
SELECT  d. 部門名, i. 商品名      SELECT  e. 従業員名
FROM    部門 d, 商品 i            FROM    従業員 e
WHERE   d.id = i. 部門 id;        WHERE   e. 給与 > 20000;
```

そして、このアルゴリズムによって生成された SQL のそれぞれに対して、ORDER BY 節の付加を行う。分割後の属性集合では、グルーピング木においてあるノードの子ノードが一つしか存在しないといった、単純な構造となっている。そこで、3.2 節で述べた方法に従い、グルーピング木のルートノードに近い順に ORDER BY 節に属性を並べる。

4.3 複数の検索結果の合成

複数に分割して取得したデータベースからの検索結果は、それぞれのグルーピング木にしたがって、グルーピング操作を行う。この際に、ORDER BY 節によるソートが完全になされているため、グルーピング操作時にはソートを必要としない。

その後、分割されたグルーピング木の共通部分に着目して、共通部分の属性の値が等しいもの同士を組み合わせ、分割前のグルーピング木の構造へと再構造化を行う。既に、それぞれの分割された構造化データは共通部分の属性に関してソートが完了した状態で与えられるため、この再構成に関してもソートを必要としない。

5 実装・評価

本稿で提案した SQL の分割および ORDER BY 節の付加を行う処理系を JAVA を用いて実装した。実装に用いた環境は以下の通りである。

CPU	UltraSPARC-II 248MHz × 4
メモリ	1GB
OS	Solaris 2.6
言語	Java 1.2
DBMS	PostgreSQL 7.2

図 6: 実験環境

評価実験は、グルーピング木において複数の path で表現される SuperSQL 質問文について、本稿で提案した質問文の分割手法と質問文の分割を行わない手法において、データベース検索時間、データ構造化処理時間およびその合計処理時間について計測を行い、比較を行った。これらの結果について以下に述べていく。

5.1 実験 1：2 つのグルーピング

実験 1 では、一つの項目に対して二つのグルーピングを持つ SuperSQL 質問文に対して実験を行った。実験に用いた SuperSQL 質問文およびそのグルーピング木は図 7 である。なお、表 a のそれぞれのタプルに対して、表 b,c のそれぞれ 10 タプルが結合するデータを用いた。

```
GENERATE HTML
   [a.a1,[b.b1]!,[c.c1]!]!]!
FROM  a,b,c
WHERE a.bid=b.id
AND   a.cid=c.id
```

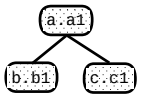

図 7: 実験 1 の SuperSQL 質問文とグルーピング木

実験 1 の結果は図 8 のようになった。グラフの縦軸は、分割を行わない処理系が要した時間を 100% としたときの分割を行った処理系が要した時間を表している。

実験の結果より、二つのグルーピングをもつ SuperSQL 質問文では、本稿で提案した分割手法の方が 60%〜80%程度処理時間を削減することができることがわかる。特に処理すべきデータが増えれば増える程、その効果は大きくなっている。

どちらも ORDER BY 節の付加効果によるソート処理の簡略化によって、データの構造化にかかる時間は相対的に小さくなっているが、分割を行わない処理系の場合、表 c に対するソートが改めて必要な

a.a1 タブル数		10	100	1000
合計処理時間	分割	784	3213	26415
	非分割	2160	19289	189009
DB検索時間	分割	603	2895	24104
	非分割	1812	16533	157313
構造化処理時間	分割	181	318	2311
	非分割	348	2756	31696

一部抜粋, 単位:msec

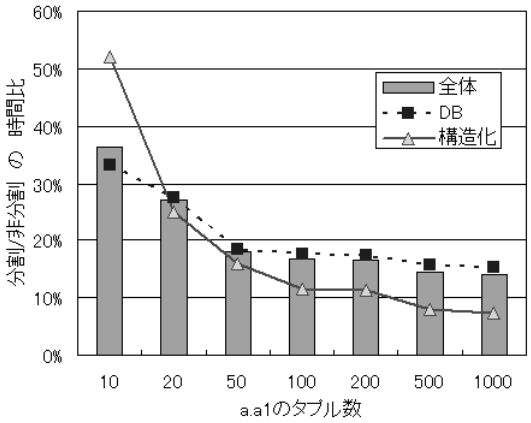

図 8: 実験 1 の結果

分、取得データに構造化の時間がほぼ比例している。それに対して本稿の分割手法では、構造化に必要なソートはすべて行われた状態で検索されるので、データの大きさに比べて構造化にかかる時間の増加の割合はゆるやかになっている。

また、データベースへの検索時間に関してみれば、本稿の分割手法では二つの SQL 文によって検索を行うために検索時間が多くかかると予想できるが、実際には処理時間が短くなっている。この理由としては、データベースからの検索結果のタプル数が、分割手法の方が圧倒的に少なく、その結果 JDBC を通じたデータ転送量が少なくなったことが考えられる。例えば表 a が 10 タプルである場合に、表 a,b,c のすべての結合結果は状況にもよるが最大 1000 タプルになるのに対し、表 a,b の結合結果と表 a,c の結合結果はそれぞれ 100 タプルであり、その合計を比較すれば、転送量が大幅に削減されると考えられる。

逆に、同一表内でグルーピングが行われた場合は、結合結果のタプル数は分割して取得したタプル数程度まで小さくなり、分割することによって処理時間が増えてしまうという可能性がある。

5.2 実験 2：複数のグルーピング

実験 2 では、表 a に結合するグルーピングの数を変化させることによって、分割して行った場合の SQL 発行数による処理時間効率の変化を調べる実験を行った。ここでは、グルーピングを 2 つ、3 つ、4 つとした 3 種類の SuperSQL 質問文を用いて処理時間を計測した。実験に用いた SuperSQL 質問文およびそのグルーピング木は図 9 である。なお、表 a の 1 タプルに対して、表 b,c,d,e のそれぞれ 10 タプルが結合するデータを用いた。

```
GENERATE HTML
   [a.a1,[b.b1]!,[c.c1]!, ...]!]!
FROM  a,b,c
WHERE a.bid=b.id AND a.cid=c.id ...
```

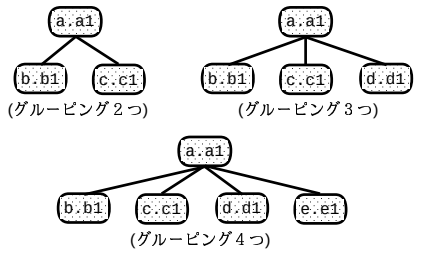

図 9: 実験 2 の SuperSQL 質問文とグルーピング木

実験 2 の結果は図 10 のようになった。グラフの縦軸は実験 1 と同じで、処理時間の比率で表している。

グルーピング数		2	3	4
合計処理時間	分割	784	2256	22378
	非分割	2160	22265	306154
DB検索時間	分割	603	2104	22200
	非分割	1812	18619	248916
構造化処理時間	分割	121	152	178
	非分割	348	3646	57238

単位:msec

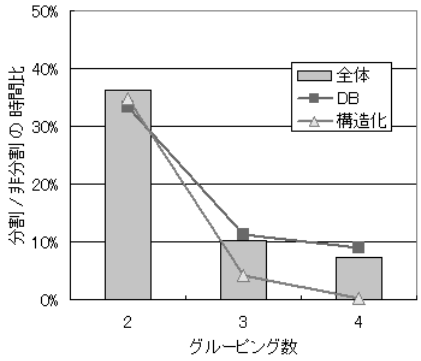

図 10: 実験 2 の結果

実験の結果より、同一階層にあるグルーピングが多くなればなるほど、本稿で提案した分割する処理

系がより有利になることがいえる。グルーピングの数が一つ増えると、全ての表を結合した場合における検索結果のタプル数が最大 10 倍にもなるにもかかわらず、分割して取得した中間結果は 100 タプルしか増えないことからも推測できる。

さらに分割する処理系では、構造化にかかる時間はグルーピングの大きさにほとんど影響していないことが、結果からわかる。これは、分割取得した結果がソート済みかつグルーピングが一つしか存在しないなど十分小さくなっているため、それぞれのデータの構造化や結果の合成にかかる時間を大幅に簡略化できたと考えられる。

5.3 実験 3 : 2 つのグルーピングの片方に 2 つのグルーピングが存在

実験 3 では、2 つのグルーピングの片方にさらに 2 つのグルーピングが存在する SuperSQL 質問文に対して、処理時間を計測した。実験に用いた SuperSQL 質問文およびそのグルーピング木は図 11 である。表 a と c、表 b と d および表 b と e はそれぞれ 1 対 10 の割合で結合していて、表 a に対する表 b の結合するタプル数を変化させて実験を行った。

```
GENERATE HTML
 [a.a1,[b.b1,[c.c1]!,[d.d1]!]!,
       [e.c1]!
 ]!
FROM  a,b,c,d,e
WHERE a.bid=b.id AND a.eid=e.id
AND   b.cid=c.id AND b.did=d.id
```

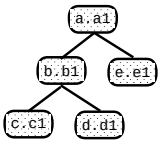

図 11: 実験 3 の SuperSQL 質問文とグルーピング木

この実験は、再帰的に質問文の分割を適用できる質問文であり、分割を適用すると図 12 のように 3 つの SQL が発行されることになる。

```
SELECT a.a1,b.b1,c.c1
FROM  a,b,c,d,e
WHERE a.bid=b.id AND a.eid=e.id
AND   b.cid=c.id AND b.did=d.id

SELECT a.a1,b.b1,d.d1
FROM  a,b,c,d,e
WHERE a.bid=b.id AND a.eid=e.id
AND   b.cid=c.id AND b.did=d.id

SELECT a.a1,e.e1
FROM  a,b,c,d,e
WHERE a.bid=b.id AND a.eid=e.id
AND   b.cid=c.id AND b.did=d.id
```

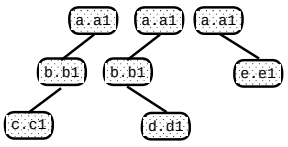

図 12: 実験 3 の質問文に対する分割手法の適用結果

実験 3 の結果は図 13 のようになった。グラフの縦軸は実験 1 と同じで、処理時間の比率で表している。

b.b1 タプル数		10	20	50	100
合計処理時間	分割	2720	3468	3823	28051
	非分割	20935	21344	26043	364322
DB 検索時間	分割	2504	3258	3610	27331
	非分割	17608	18026	21936	304967
構造化処理時間	分割	216	210	213	720
	非分割	3327	3318	4107	59355

単位:msec

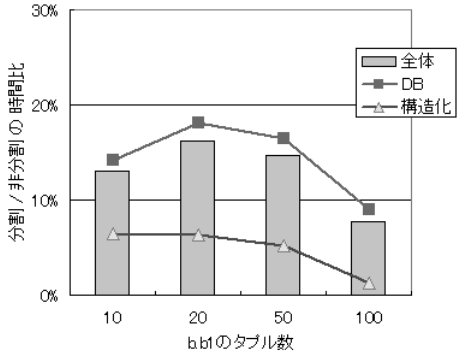

図 13: 実験 3 の結果

実験の結果から、分割を完全に行った処理系が、分割を全く行わなかった方と比較して五分の一程度の時間で行い、有利であることがわかった。

しかしながら、b.b1 のタプル数が 10 から 20 に増えた際に処理時間比が大きくなっていることから、検索対象が大きくなるほど分割による効果が大きくなるとは一概にいえないこともわかる。実際の検索においてはさまざまな状況が考えられるため、この実験で用いたデータと比較して、データが均一に分布していない場合や、同一表内での結合により、分割を行わない方が有利に可能性があると考えられる。

6 おわりに

本稿では、複数のグルーピングを持つ SuperSQL 質問文処理において、グルーピング木を用いた質問文の分割について述べた。グルーピング木のルートノードから葉ノードへのパス毎に属性集合を抽出し、その属性集合に対して SQL を発行することによって、データベースからの検索結果を抑制することができた。

また、これら分割された属性集合はグルーピング構造が単純なため、それらを取得する SQL に対して適切な ORDER BY 節を付加する手法を組み合わせることによって、グルーピング時に必要なソート処

理をデータベースからの検索結果を得る時点で全て行うことが可能になった。

この結果、複数のグルーピングを持つ SuperSQL 質問文において、データベースからグルーピングされたデータを取得するまでのコストを大幅に低減することを、評価実験を行った結果により示した。

参考文献

[1] Motomichi Toyama: SuperSQL: An Extended SQL for Database Publishing and Presentation, *Proc. of ACM SIGMOD '98*, pp. 584-586, 1998.

[2] SuperSQL HOME PAGE, http://ssql.db.ics.keio.ac.jp/index.html

[3] 赤堀正剛, 有澤達也, 遠山元道: SuperSQL による関係データベースと XML データの統合利用, 情報処理学会論文誌:データベース, Vol. 42, No. SIG8(TOD 10), pp. 66–95, 2001.

[4] 有澤達也, 遠山元道: SuperSQL 処理系におけるグルーピング操作の効率的な実装, データ工学ワークショップ (DEWS2001), 2001.

[5] 有澤達也, 実政宏幸, 遠山元道: TFE 処理系における問い合わせ文の分割による最適化, 情報処理学会第 55 回全国大会論文集,2X-03, 1997.

[6] 実政宏幸, 遠山元道: TFE 処理系におけるマルチセッション高速化, 情報処理学会第 51 回全国大会論文集,2D-6, 1995.

[7] Jayavel Shanmugasundaram, et. al.: Relational Databases for Querying XML Documents: Limitations and Opportunities, *Proc. of the 25th VLDB Conference*, pp. 302-314, 1999.

[8] Michael Carey, et. al.: Efficiently Publishing Relational Data as XML Documents, *Proc. of the 26th VLDB Conference*, pp. 65-76, 2000.

[9] Goetz Graefe: Query Evaluation Techniques for Large Databases, *ACM Computing Surveys 25(2)*, pp. 73-170, 1993.

[10] Mary Fernandez et al.: Efficient Evaluation of XML Middle-ware Queries, *Proc. of ACM SIGMOD 2001*, pp. 103-114, 2001.