Vol.2013-DBS-157 No.27
Vol.2013-IFAT-111 No.27
2013/7/23

表値型ユーザ定義関数を用いた問合せにおける関数出力オーバヘッド削減手法

藤井　雄規[†1]　　柴田　秀哉[†2]　田村　孝之[†3]　菅野　幹人[†4]

本稿では，RDBMS における表値型ユーザ定義関数のオーバヘッドに関する考察，及び削減手法の提案を行う．表値型関数は複数レコード処理をオーバラップ可能なため，高 CPU 負荷なユーザ定義演算の並列処理をアドオンする用途に使用できるが，出力オーバヘッドにより性能低下を招くという課題がある．本稿では，表値型関数の出力データ量を最小化するように問合せを書き換えるオーバヘッド削減手法を提案し，評価によりその有効性を示す．

1. はじめに

多くの RDBMS は，利用者が独自に定義した処理を実装できるように，ユーザ定義関数と呼ばれる拡張機構を備えている．RDBMS の利用者はユーザ定義関数を利用することで, 独自に定義した演算を SQL インタフェースを介して実行可能となる．ユーザ定義関数は，一般的なスカラ値だけでなく, レコード集合を戻り値とすることもできるため，定義できる処理の自由度が高い．レコード集合を戻り値をする関数は，表値型関数と呼ばれる．

表値型関数は, PostgreSQL の dblink に代表されるように，異なるデータベースサーバ同士を接続するためのモジュールとして利用する用途がよく知られている．PostgreSQL の dblink は分散データベースを実現するための表値型関数であるが，文献 2)のように， RDB と異種 DB の連携を実現する用途に表値型関数を用いている事例もある．PostgreSQL の dblink については 1)を参照されたい．表値型関数の別の応用例として，複数レコード処理をオーバラップできるという特長を活かしたものがある．例えば，文献 3)では表値型関数を用いて，高 CPU 負荷演算の並列処理を PostgreSQL のアドオンとして実現している．

しかしながら，表値型関数はレコード集合を出力とするため，出力データ量が大きくなる傾向にある．実際，いくつかの主流な RDBMS においては，実装上の理由から，表値型ユーザ定義関数の関数出力によるオーバヘッドが無視できないという課題がある．オーバヘッドそのものを解消するには DBMS 内部の実装を改修する必要があるが, ユーザ側で使用方法を工夫することでオーバヘッドを回避できる場合がある．

そこで本稿では，文献 3)で示されている並列処理アドオン機構をターゲットとして，表値型関数の出力データ量を最小化するように問合せ文を書き換えることでオーバヘッドを回避する手法を提案する．また，評価により同アドオン機構に対する提案手法の有効性を示す．

本稿の構成について示す．まず，2 節で表値型関数の概要，及びその応用について述べる．3 節では，表値型ユーザ定義関数の関数出力オーバヘッドについて述べる．4 節では，関数出力オーバヘッドを削減する手法を提案する．5 節では，提案手法の評価について報告する．

2. 背景

本節では，まず表値型関数の概要を述べる．さらに本稿の提案手法でターゲットとする，表値型関数を用いたユーザ定義演算の並列処理アドオン機構について述べる．

2.1 表値型関数について

ユーザ定義関数には，スカラ値型関数と呼ばれるクラスや，さらにそれを包含するクラスとして，表値型関数と呼ばれるものがある．スカラ値関数は，レコードを受け取って何らかの加工を施した後，スカラ値を出力する一般的な関数である．一方の表値型関数は，レコードの集合を出力可能な関数である．

スカラ値型関数にない表値型関数の特長の 1 つに，複数レコード処理をオーバラップできるという点がある．スカラ値型・表値型関数それぞれによる演算の実現イメージを図 1 に示す．スカラ値型関数では，1 回の呼出しで単一の値しか返せないため，複数レコードに対して逐次処理することを強いられる．一方，表値型関数では 1 回の呼出しで複数レコードを出力可能なため，複数レコードを入力として受け取れば，それらを一括処理することが可能である．

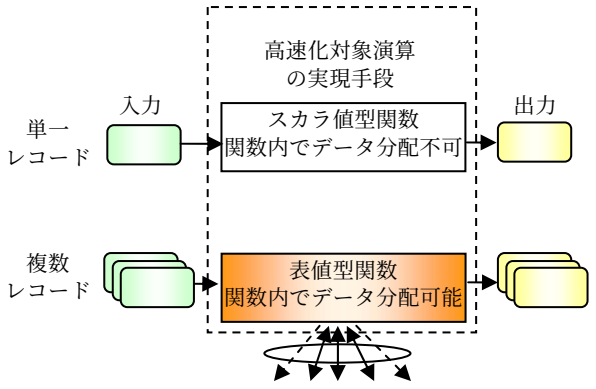

図 1：スカラ値／表値型関数それぞれによる演算の実現

文献 3)ではこのような特徴を活かして，PostgreSQL に高

†1 ～ †4 三菱電機株式会社　情報技術総合研究所

Vol.2013-DBS-157 No.27
Vol.2013-IFAT-111 No.27
2013/7/23

CPU 負荷演算の並列処理をアドオンしている．2.2 節ではこの応用研究について述べる．

2.2 表値型関数を用いた高 CPU 負荷演算の並列化

文献 3)に示されている，PostgreSQL に高 CPU 負荷演算の並列処理をアドオンした応用研究について述べる．なお，本稿では以下，表値型関数のことを単に表関数と呼び，文献 3)に示されている高 CPU 負荷演算の並列処理アドオン機構で利用される表関数を proc_by_set()と表すことにする．

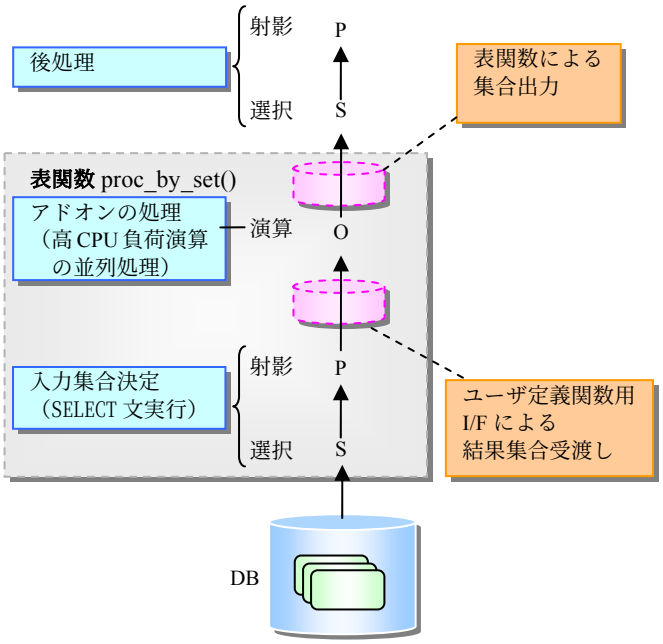

図 2：表関数 proc_by_set()の動作

図 2 には，表関数 proc_by_set()及びその周辺の動作を示している．まず，表関数 proc_by_set()は引数として SELECT 文を受け取る．これを q とすると，次に proc_by_set()は SELECT 文 q を実行する．この SELECT 文 q は，表関数を含む拡張 SQL 文の作成者により指定可能である．続いて SELECT 文 q の実行結果を対象とし，アドオンによる処理を実行する．この処理の部分が並列化可能となる．演算処理の結果は，表関数 proc_by_set()の実行結果表に対して，選択や射影等の後処理を実行する．これらの後処理は表関数の外に記述される SELECT 句や WHERE 句の処理に相当する．以上が表関数 proc_by_set()の動作である．

表関数 proc_by_set()の利用イメージを擬似 SQL により説明する．2 項演算 op を真偽値を返す高 CPU 負荷な比較演算とし，演算 op のアドオンが表関数 proc_by_set()により実現されているとする．ここである定数値 l に対して，

```
SELECT a FROM r WHERE b op l　（A）
```

という問合せ文を考える．問合せ文（A）は proc_by_set()を用いることで，

```
SELECT a' FROM proc_by_set(
 'SELECT a, b FROM r, l, b
) AS r' (a' typeof(a), b' boolean)
WHERE b' = true　　（B）
```

のように書き換えることができる．ここで AS 句は表関数が返す結果表のレコード型に対するキャストを意味し，"typeof"は対象列のデータ型を表現している．この例では，表関数は SELECT 文と 1 つの定数パラメータ l を引数とし，結果表として，列 a'及び演算結果列である b'を返す．

3. 課題

2 節では，表関数を使用することで，ユーザ定義演算の並列処理をアドオン可能であることを述べた．一方で，このような拡張機構を利用すると，RDBMS の実装形態によっては，RDBMS 本体との間でデータのやり取りによるオーバヘッドが問題となることがある．実際，いくつかのよく知られた RDBMS においては，各々の実装上の理由から，表関数から RDBMS 本体への出力受渡しのオーバヘッドが無視できないという課題がある．本節では，いくつかの RDBMS で採用されている，表関数の実装形態を 2 種類示し，それぞれに対して表関数の関数出力オーバヘッドが発生する原因を示し，考察を行う．

3.1 プロセス分離型

一部の RDBMS では，C 言語や JAVA 等の汎用プログラミング言語で実装されたユーザ定義関数に対して，RDBMS 本体のアドレス空間とは別のアドレス空間を割り当て，外部プロセスとして扱う実装形態を採用している．本稿では，このような実装形態をプロセス分離型と呼ぶことにする．プロセス分離型の実装形態では，表関数から RDBMS 本体へ結果集合が返却される際に，プロセス間通信によるオーバヘッドが発生する．

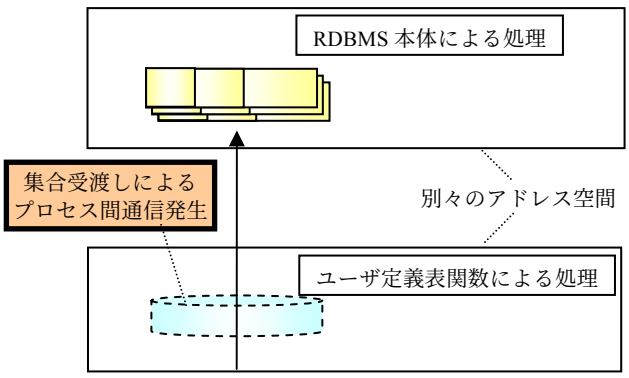

図 3：表関数の関数出力オーバヘッド（プロセス分離型）

このような実装形態には，プロセス間通信によるオーバヘッドというデメリットがある反面，ユーザ定義関数側の

処理でエラーが起きても，RDBMS 本体に割り当てられたアドレス空間内のデータが被害を受けないという，安全面のメリットもある．よって，今後プロセス分離型の実装形態が採用されなくなるとは言い切れない．

3.2 出力データ蓄積型

表値型ユーザ定義関数の出力データが一定量を超えるとディスクに 1 次退避する実装形態を採っているような RDBMS も存在する．このような実装形態を本稿では出力データ蓄積型と呼ぶ．出力データ蓄積型の実装形態では，この 1 次退避によるディスク入出力が関数出力のオーバヘッドとなる．

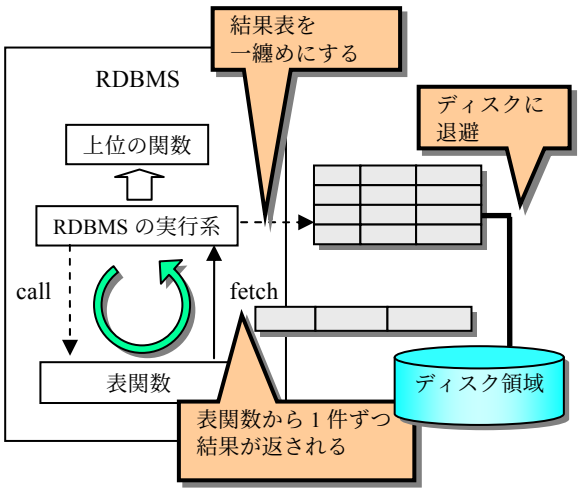

図 4：表関数の関数出力オーバヘッド（出力データ蓄積型）

出力データ蓄積型の RDBMS における，表関数の動作を図 4 に示す．出力データ蓄積型の表関数では，出力結果である表のデータが 1 件（行）作成されるたびに，表関数の実態である汎用プログラミング言語関数が呼出される仕組みとなっており，1 件毎に RDBMS 本体へ行データが返却される．このような仕組みを採用しているにも関わらず，出力データ蓄積型の RDBMS は表関数からの全出力結果が揃うまで一時バッファに結果を溜めておき（結果表を実体化し），結果が揃い次第，後の処理を再開する．この際，一時バッファ用に確保されたメモリ領域が足りなくなると，ディスクに結果が書き出すため，余分なディスク入出力が発生する．

この課題は，当該 RDBMS の改版に伴い，近い将来解消される可能性があるが，当面のオーバヘッド回避策が求められる．

4. オーバヘッド削減手法

本節では，文献 3)で示されている高 CPU 負荷演算の RDBMS アドオン機構をターゲットとして，表関数を含む問合せ文を，関数の出力データ量が最小となるように書き換える手法を提案する．

4.1 問合せ文への要件

提案手法の適用対象となる問合せ文は全く任意ではなく，ある要件を満たす必要がある（図 5）．

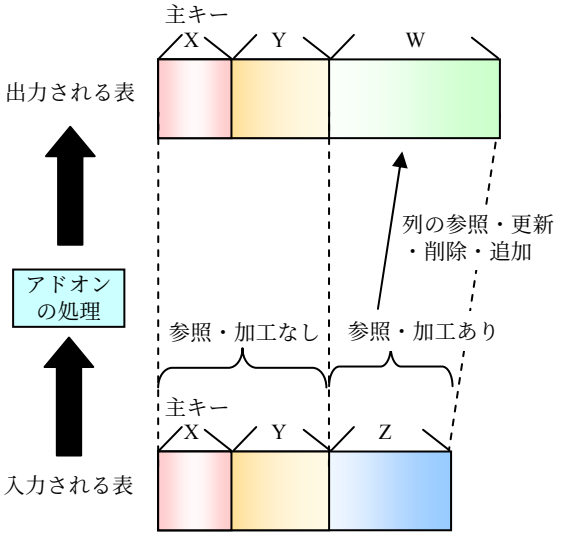

図 5：表関数の動作（提案手法適用の要件）

以下，図 5 に基づいて問合せ文の要件を説明する．

- 表関数には，主キーを持つ表が入力される．入力される表を R とし，R の主キーを X とする．
- 表関数は，R の特定の列の集合 Z だけを参照・加工するように指定する．アドオンの処理は，Z のみを参照し，列の更新・削除・追加等を実行する．Z を加工してできる列を W とする．
- 主キーX を構成する任意の列は，参照・加工されない．
- R の列の内，X・Z 以外の部分を Y とすると，表関数は結果表は R'（X, Y, W）を出力する．

4.2 オーバヘッド削減方針

アドオンの処理を含む問合せについて，提案手法適用前，適用後それぞれの実行計画を図 6（提案手法適用前）及び図 7（提案手法的用後）に示す．図 6 では，テーブル 1 から「選択・射影 1」によって入力集合を決定し，アドオンの処理を実行した後，「選択・射影 2」で後処理を実行する計画を示している．

Vol.2013-DBS-157 No.27
Vol.2013-IFAT-111 No.27
2013/7/23

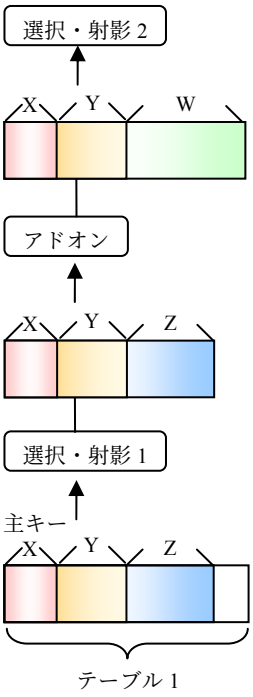

図 6：アドオンの処理を含む問合せ（提案手法適用前）

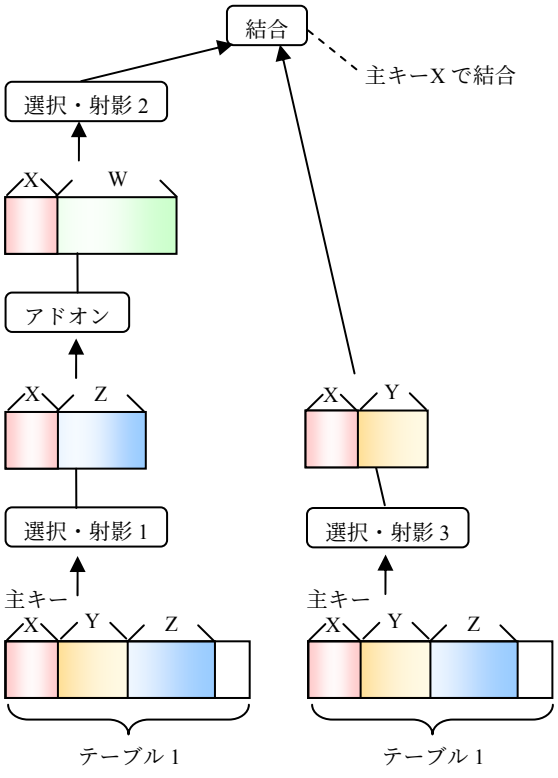

図 7：アドオンの処理を含む問合せ（提案手法適用後）

図 6 に示している問合せ実行計画において，入力される表の列の内，列の集合 Y はアドオンの処理に不要である．そこで，提案手法では，問合せを 2 つに分け，Y を入力から除去する．Y は別途テーブル 1 から読み出し，表関数によって出力される表と結合するように実行計画を変更する．図 7 には，提案手法適用後の問合せ実行計画を示している．表関数が出力する表は，図 6 から図 7 で，(X, Y, W)から（X, W)になり，データ量が削減されていることがわかる．

4.3 問合せ文の書換え方法

4.2 小節のオーバヘッド削減方針に従って問合せ文を書き換える方法を説明する．2.2 小節で説明した問合せ文（A）の SELECT 句に，列 c を追加した問合せ文

SELECT a, c FROM r WHERE b op l （C）

を考える．ここで a はテーブル r の主キーであると仮定する．問合せ文（C）を，表関数 proc_by_set()を用いて単純に書き換えると，

```
SELECT a', c' FROM r proc_by_set(
 'SELECT a, b, c FROM r', l, b
) AS r' (a' typeof(a), b' boolean, c' typeof(c))
WHERE b' = true                    （D）
```

となる．この問合せ文では，proc_by_set()の結果表は r'(a', b' , c)である．提案手法では，問合せ文（D）を

```
SELECT a', c'
FROM proc_by_set(
 'SELECT a, b FROM r', l
) AS r' (a' typeof(a), b' boolean, c' typeof(c))
INNER JOIN r ON
r'.a' = r.a                        （E）
```

のように書き換える．

5. 評価

本節では，文献 3)で示されている並列処理アドオン機構に対して，提案方式の有効性を評価した結果を報告する．

5.1 評価条件

(1) アプリケーション

評価で使用するアプリケーションは，文献 3)で示されている，検索可能暗号を用いた検索アプリケーションとする．検索可能暗号とは，データ及び検索キーワードを暗号化した状態のまま，両者が平文情報として一致するか否かのみを判定可能とする暗号技術であり，文献 4)に端を発している．検索可能暗号を用いた検索処理では，単純なバイナリ比較による一致判定ではなく，確率的暗号を用いた複雑な処理が必要となる．そこで文献 3)では，検索可能暗号における一致判定処理を高 CPU 負荷な演算と見做して，表関数 proc_by_set()を用いた並列処理アドオンを実現している．

以下 proc_by_set()を，検索可能暗号による一致判定処理を組み込んだ表関数として，問合せ文（F）によりその利用イメージを説明する．表関数 pro_by_set()は，アドオンの入力集合を決定する問合せ文・暗号化キーワード・一致判

Vol.2013-DBS-157 No.27
Vol.2013-IFAT-111 No.27
2013/7/23

定対象の列名（問合せ文（F）では tag）を入力として受け取り，比較演算を実行する．この際，レコードを分配し一致判定処理を並列化する．出力する結果表は，一致判定結果の列（ブール型）を入力表に追加したものである．

```
SELECT a', b' FROM proc_by_set(
 'SELECT a, b, tag FROM t', enc(keyword), tag
) AS r' (a' typeof(a), b' typeof(b), c' boolean)
WHERE c' = true                    （F）
```

(2) 問合せとスキーマ

評価で使用するスキーマは，評価で使用する問合せと共に，図 8 及び図 9 に示している．以下，図 8 を用いてスキーマの説明を行う．評価では，テーブル 1 からテーブル 25 の合計 25 個のテーブルを使用する．テーブル 1 は，検索可能暗号の一致判定に用いる暗号化列を含んでおり，テーブルの規模は，列数 46，データ件数 100,568 である．テーブル 2 からテーブル 25 は, テーブル 1 の情報に関して付加的な情報を格納するテーブルである．

図 8 の問合せは，提案方式適用前の問合せである．図 8 の問合せは，「選択・射影 1」から「選択・射影 2」で検索可能暗号の一致判定アドオンを含む検索処理を行った後，その結果集合にテーブル 2 からテーブル 25 の付加的な情報を結合して最終的な結果表を作成する．

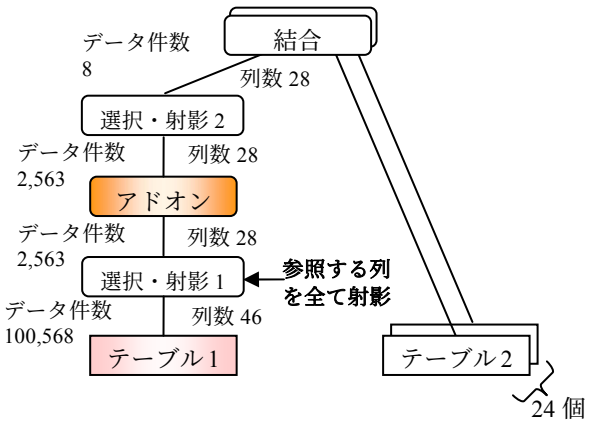

図 8：提案方式適用前の問合せ

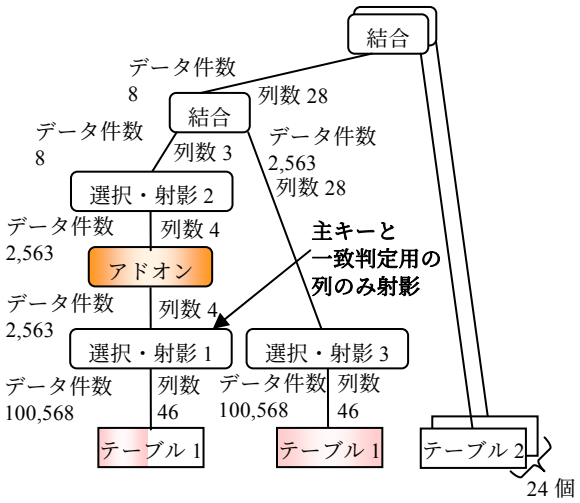

図 9：提案方式適用後の問合せ

図 9 の問合せ 2 は，図 8 の問合せを提案方式により書き換えたものである．実際に書き換えている部分は，図 8 の問合せにおける，「選択・射影 1」から「選択・射影 2」の副問合せの部分である．提案方式適用前と適用後それぞれにおける，表関数の出力データ詳細を表 1 に示す．

表 1：表関数の出力データ詳細（問合せ毎）

	問合せ	
	適用前	適用後
出力サイズ（合計）	4,557KB	33KB
1 行のサイズ	1,778B	13B
行数	2,563	2,563
列数	28	4

(3) 評価環境

評価環境を表 2 に示す．

表 2：評価環境

構成要素			内容
DB サーバ	H/W	CPU	Xeon X5650 2.66GHz（6 コア）×2 Hyper-Threading：無効
		メモリ	4GB
		HDD	1TB（7.2krpm 3.5inch SATA）
	S/W	DBMS	プロセス分離型
		OS	Windows Server 2008R2 (x64)
並列処理ノード	H/W, OS 構成等		DB サーバを利用
	DBMS との通信手段		パイプ

Vol.2013-DBS-157 No.27
Vol.2013-IFAT-111 No.27
2013/7/23

5.2 評価結果

問合せ毎の，表関数出力及び検索時間を表 3 に示す．まず, 表関数出力時間は, 提案手法適用前の問合せで 0.7 秒，適用後で 0.1 秒となっている．また，検索処理全体については，提案手法適用前の問合せで 2.2 秒，適用後の問合せで 1.5 秒かかっている．よって，提案手法適用により表関数の関数出力オーバヘッドが 0.6 秒分削減されていることがわかる．

表 3：表関数出力と検索全体の時間（問合せ毎）

処理内容	問合せ	
	適用前	適用後
表関数出力	0.7 秒	0.1 秒
処理全体	2.2 秒	1.5 秒

提案手法適用前と適用後それぞれにおける表関数による出力データの詳細（表 1）と表 3 と比較することで，出力データ量が小さい程，表関数の出力時間が短くなることが確認できる．ただし，今回の評価結果だけでは，表関数の出力時間を小さくするために，出力データについて，行数や列数等どういった要素を小さくすれば良いのかまではわからない．表関数の関数出力オーバヘッドの特性については，今後詳細に分析する必要がある．

6. おわりに

本稿では，RDBMS の表値型ユーザ定義関数の関数出力オーバヘッドについて考察し，削減手法を提案した．提案手法は，表値型ユーザ定義関数を含む問合せ文に対して，関数の出力データ量が最小となるように書き換えるものである．また，検索可能暗号を用いた検索アプリケーションを対象として評価を実施し, 提案手法の有効性を実証した．

今後はオーバヘッドの特性を詳細に分析し，表値型ユーザ定義関数を含む問合せに対して，実行計画最適化の指針を得ることを目指す．

参考文献

1) http://www.postgresql.org/docs/9.2/static/dblink.html
2) 竹内丈志，山岸義徳，中村隆顕，立床雅司，郡光則: オープンソース DBMS 拡張によるセンサデータベースの実現，情報処理学会第 74 回全国大会 2C-3，Mar. 2012
3) 柴田秀哉，田村孝之: RDBMS へアドオン可能な高 CPU 負荷演算の分散並列処理手法, DEIM Forum 2013 F10-4, Mar.2013
4) D. X. Song, D.Wagner and A. Perrig, ¥Practical Techniques for Searches on Encrypted Data", Proc. 2000 IEEE Symposium on Research in Security and Privacy, pp.44-55, 2000