Vol.2016-OS-136 No.6
2016/2/29

暗号化データベースシステムにおける総和計算の並列化手法に関する評価

堀尾 健太郎[1,3,a)]　川島 英之[2,3]　建部 修見[2,3]

概要：暗号化データベースシステムはデータを暗号化することにより情報漏洩対策を行いながら，正規のユーザは従来のデータベースシステム同様に動作させることができるシステムである．暗号化データベースシステムは様々な性質を持つ暗号化手法を用いることで，データベース上で平文が露わになることなく問合せ処理を行なうことができるが，一方で暗号化によってデータサイズと演算コストが増大するため処理性能の面での劣化が著しい．特に OLAP 等データ分析において重要な演算である総和計算は暗号化データベースシステム上での実現に必要な準同型暗号が必要となり，これは平文での演算と比較して非常にコストが高い．既存研究では暗号手法の特性を用いた総和計算の効率化手法が提案されているが，その並列化については言及されておらず，さらなる高速化の可能性がある．一方で効率化手法の適用によりデータレイアウトが変更され，商用の並列データベースシステムなどで一般に用いられている手法は適していない．本稿では，これまでに我々が提案した暗号化データベースシステムにおける総和計算の並列化手法について，ミクロ的な評価を行なう．

1. はじめに

近年，クラウドコンピューティングの普及により，データベースシステムを第三者が管理するサーバに設置することが多くなってきた．一方でクラウドコンピューティングにはセキュリティに関する懸念が存在する．サーバ管理者はサーバ上のファイルを閲覧する事が可能であるため，第三者でありながら容易に機密データを盗み見る事が可能である．このような情報漏えいに対する防護策としてデータの暗号化がよく用いられている．暗号化データベースシステムはデータを暗号化することにより情報漏洩対策を行いながら，正規のユーザは従来のデータベースシステム同様に動作させることができるシステムである [1]．通常の DBMS の暗号化機能 [2] と異なり，様々な性質を持つ暗号化手法を用いることでデータベース上で平文が露わになることなく問合せ処理を行なうことができるが，暗号化によってデータサイズと演算コストが増大するため性能劣化が著しい．特に OLAP 等データ分析において重要な演算である総和計算は暗号化データベースシステム上での実現に必要な準同型暗号が必要となり，これは平文での演算と比較して非常にコストが高い．しかしながら既存研究ではこのような性能劣化を考慮している物は少ない．暗号化データベースはバックエンドに汎用の RDBMS を利用しているためクエリプロセス中に最適化の可能性がある．クエリプロセスの最適化の 1 つとして処理の並列化が挙げられる．効率的な暗号化データベースシステムである MONOMI[3] で用いられている総和計算処理効率化手法 [4] を適用すると，並列データベースシステムで広く用いられるハッシュ分割には適さないデータレイアウトとなる．そこで我々は暗号化データベースシステムにおけるミクロ的な調査を行い [5]，MONOMI 方式の暗号化データベースシステムにおける総和計算処理に適した並列化手法として，ブロックサイクリック分割を用いた方式を提案した [6]．本稿ではその評価について述べる．

本稿の構成は以下のとおりである．2 節では背景として暗号化データベースシステムにおける総和計算について説明し，問題点とともに述べる．3 節では [6] において提案した総和計算の並列化手法について説明する．4 節では並列化手法の設計と実装について述べる．5 節では各種パラメータを変えて処理性能を計測した実験結果を示す．7 節ではまとめと今後の課題について述べる．

[1] 筑波大学大学院 システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba
[2] 筑波大学 システム情報系
Faculty of Engineering, Information and Systems, University of Tsukuba
[3] 国立研究開発法人 科学技術振興機構 CREST
JST CREST
[a)] horio@hpcs.cs.tsukuba.ac.jp

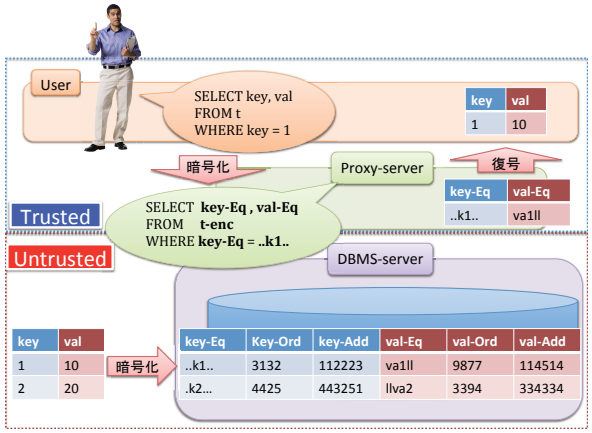

図 1 暗号化データベースシステムの概要

$$Enc(A) \times Enc(B) = Enc(A + B)$$

図 2 Paillier 暗号の持つ性質

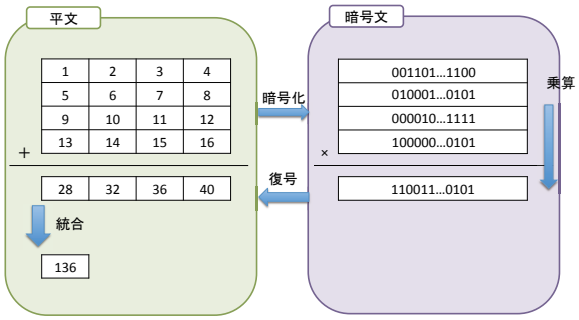

図 3 MONOMI 方式での総和計算の概要

2. 背景

2.1 暗号化データベースシステム

暗号化データベースシステムはデータベースに保管されるデータを暗号化することで情報の秘匿を達成しながら，通常の DBMS のように問い合わせ処理を可能としているセキュアなデータ処理基盤である．

ユーザから発行された問合せと DBMS サーバから返ってきた結果の仲介及び暗号鍵の管理を行なうプロキシサーバ，暗号化されたデータの管理を行なう DBMS が稼働する DBMS サーバの 2 つのサーバにより構成される．ユーザから問合せが発行されると，プロキシサーバは問合せに含まれる変数の値，テーブル名を適切に暗号化して DBMS サーバに送る．DBMS サーバ上では平文が一切露わになることなく問合せが処理され，プロキシサーバに対して暗号化された結果を返す．プロキシサーバは結果を復号してユーザに返す (**図 1**([6] より引用)).

DBMS には暗号 Onion と呼ばれる多重に暗号化されたデータが保管される．暗号 Onion には 4 つの種別があり，等号チェックのみ可能な暗号文（Eq-Onion），大小関係チェックのみ可能な暗号文（Ord-Onion），加算演算のみ可能な暗号文（Add-Onion），文章から単語を検索することが可能な暗号文（Search-Onion）が用意される，それぞれ異なる暗号化手法により暗号化される．元データの種類に応じて必要な Onion が生成されるため，1 つの値に対応する暗号文が複数用意されることになる．問合せ処理の際にはオペレータの種類に応じて適切な Onion が参照される．

DBMS サーバ上には暗号化されたデータのみが置かれ，データ処理が必要な際にも平文まで復号されることはない．そのため，DBMS サーバが攻撃され管理者権限を奪取される場合や第三者であるサーバ管理者が情報の不正取得を企てるような場合にも情報漏洩を防ぐことが可能である．

暗号化データベースシステムの先駆け的研究として CryptDB[1] があり，それを効率化したものとして MONOMI[3] がある．

2.2 暗号化データベースシステムにおける総和計算

暗号化データベースシステムにおける総和計算処理と，MONOMI で用いられている効率化手法について述べる．暗号化データベースシステムでは総和計算に Paillier 暗号 [7] を用いている．Paillier 暗号は加法準同型性という性質を持つ暗号化手法である．加法準同型性とは暗号文同士の乗算をすることで，平文を露わにすることなく平文同士の和を暗号化した暗号文を得ることができる性質である (**図 2**)，これを用いて暗号化データベースシステム上での加算処理を実現する．

CryptDB では 1 つの平文に対し 1 つの暗号文を生成している，本稿ではこの方式を CryptDB 方式と呼ぶ．Paillier 暗号では一般に 1024 bit の暗号鍵が用いられ，暗号文は 2048 bit となる．数値型（int 型，32 bit）のデータを暗号化する事を考えるとデータサイズが 64 倍となる．また，加法準同型性によって実現される加算は実際には 2048 bit 整数同士の乗算であり，平文の状態での加算と比べると計算コストが大きくなる．MONOMI では複数の平文を結合して平文ブロックを作り，それを暗号化するという方式 [4] を用いている．本稿ではこの方式を MONOMI 方式と呼ぶ．平文ブロックに含まれる個々の平文をスライスと呼ぶ，平文ブロックを暗号化した暗号文同士を乗算して得られる暗号文を復号すると，各スライス同士が加算された平文ブロックが得られる．復号した後に平文ブロックの各スライスを足し合わせることで総和が求められる（**図 3**([6] より引用)).1 つの平文ブロックに含めるスライスの数を S とする．MONOMI 方式での単純な総和計算処理は，暗号文データサイズは CryptDB 方式の $1/S$，暗号文の状態での乗算の回数が CryptDB 方式の $1/S - 1$ となる．

選択演算を含む総和計算（**図 4**）やグルーピング演算を含む総和計算（**図 10**）の際には，各スライス毎に i 番目の

```
SELECT sum(val) FROM table WHERE key!=5 AND key !=10 AND key!=15;
```

図 4 選択演算を含む総和計算問合せ

暗号文に含まれる値を総和に含めるかどうかを表すビット列を生成する．各スライス毎に，対応するビットが 1 の暗号文のみで総乗計算を行い，平文ブロックの該当するスライスに正しい部分和を含む暗号文を生成する (**図 7**)．プロキシサーバは正しい部分和を含んでいる S 個の暗号文を受け取り，復号しすべての正しい部分和を足し合わせることで結果が得られる．

選択演算またはグルーピング演算を含む総和計算の場合，暗号文の状態での乗算回数は CryptDB 方式と MONOMI 方式で同一であるが，暗号文データサイズについては MONOMI 方式は CryptDB 方式の $1/S$ となる．

2.3 総和計算処理の並列化における問題点

暗号化データベースシステムにおけるデータレイアウトに関して，CryptDB 方式では平文と暗号文は 1 対 1 で対応しているため，行方向のデータをまとめて保持する NSM(N-ary Storage model) の DBMS（MySQL，PostgreSQL 等）に問題なくデータを保持させることができる．一方で MONOMI 方式では複数の平文が 1 つの暗号文に対応するため，特定の列のみ行数が合わなくなることから NSM でのデータの保持は困難である．そこで MONOMI では，MONOMI 方式で暗号化した列のデータのみ別ファイルに保持する DSM 方式 (Decomposed Storage Model)[8] をとっている（**図 5**([6] より引用)）．

一方で，データベースシステムにおける処理の並列化ではデータを何らかの手法で分割し，分割されたデータに対して並列に処理を行い，結果をまとめる，ということが行われる．総和計算については，問合せ処理のはじめにデータを N 個に分割し，分割されたデータそれぞれに対して部分和を計算し，最後に部分和をすべて足し合わせることで総和を N 並列に計算可能である．データの偏りの小さいデータ分割は並列処理の鍵となる要素の 1 つである．主に用いられるデータ分割手法としてハッシュ分割がある．ハッシュ分割はグルーピングキーのハッシュ値で分割することで分割先でユニークなグルーピングユニットが生成されるため，最終的に結果をまとめる際に追加の計算などは必要ない．また比較的偏りの小さいデータ分割が可能である．

MONOMI 方式の暗号化データベースシステムでは，Add-Onion とそれ以外でデータレイアウトが異なり Add-Onion のみ 1 つの暗号文に複数の値が含まれる．そのため，あるテーブルをグルーピングキーのハッシュ値に基づいて分割することを考えると，異なる複数のグルーピングキーに対応する値が 1 つの Add-Onion 暗号文に含まれる

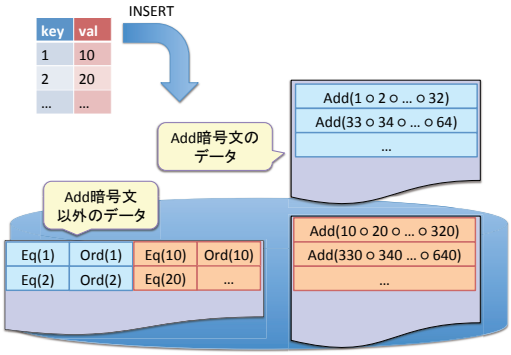

図 5 Decomposed Storage Model

場合がある．そのため，Add-Onion を単純に分割することが出来ない．すなわち，MONOMI 方式を適用した暗号化データベースシステムにおいて総和計算処理並列化を行なう際，データ分割方式にハッシュ分割を用いるのは適さない．

3. 暗号化データベースにおける総和計算並列化技法

本節では暗号化データベースにおける総和計算について，幾つかの並列化技法を述べる．問合せ処理の並列化にはいくつかの種別があり，特にクエリ内の並列性について述べる．

3.1 Round-Robin 分割を用いた並列総和計算

データベースシステムにおける処理の並列化ではデータを何らかの手法を用いて分割し，分割されたデータに対して並列に処理を行い，結果をまとめる，ということが行われる．データの偏りの小さいデータ分割は並列処理の鍵となる要素の 1 つである．多くの商用並列データベースシステムでハッシュ分割が採用されている．

ハッシュ分割を用いた総和計算処理の並列化を考えると，任意のキーのハッシュ値で N 個のバケツに分ける．それぞれのバケツにおいて独立に総和計算をを行うことで，並列に部分和を求めることができる．総和計算の並列度はバケツの数 N となる．異なるデータ分割法を用いた例として Round-Robin 方式が挙げられる．こちらはキーの値に関わらず，タプルを平等に複数のバケツに分ける．こちらの並列度もバケツの数 N となる．ハッシュ分割ではそれぞれのバケツでユニークなキーに対応するタプルが保持されるため，グルーピング演算を伴う総和計算の際，結果をまとめる際に追加の加算などは必要なくなり，効率的で

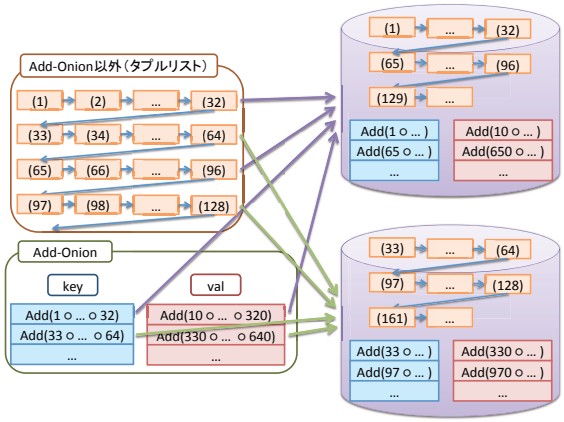

図 6 Round-Robin 分割

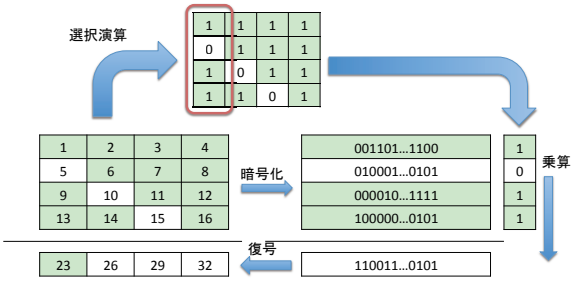

図 7 選択演算を含む総和計算の概要

$$
\begin{aligned}
&\big(\quad Enc\_Add(1 \circ 2 \circ 3 \circ 4) \quad\big)^1 \\
\times&\big(\quad Enc\_Add(5 \circ 6 \circ 7 \circ 8) \quad\big)^0 \\
\times&\big(\ Enc\_Add(9 \circ 10 \circ 11 \circ 12)\ \big)^1 \\
\times&\big(\ Enc\_Add(13 \circ 14 \circ 15 \circ 16)\ \big)^1 \\
=\ & Enc\_Add(23 \circ 26 \circ 29 \circ 32)
\end{aligned}
$$

図 8 あるスライスにおける部分和の導出式

ある．一方で Round-Robin 分割の場合，タプル数は平等に分けられるが結果をまとめる際に追加で加算が必要となる．このことから，多くの商用並列データベースシステムでハッシュ分割が採用されている．

MONOMI 方式を適用した暗号化データベースシステムにおける総和計算処理を考えると，ADD-Onion において異なるキーに対応する値がまとめて 1 つの暗号文に含まれる場合がある．そのためキーのハッシュ値で分割すると，複数のバケツに同一の Add-Onion 暗号文をコピーする必要が生じる場合がある．そのため，MONOMI 方式を適用した暗号化データベースシステムにおいてハッシュ分割による総和計算処理並列化は適さないと考えられる．

Round-Robin 方式を用いることを考える場合, 1 つの平文ブロックに含めるスライスの数を S とすると，Add-Onion 以外のデータは S 個のタプル毎に分割し，タプルに対応する Add-Onion のデータを同様に割り当てる．$S = 32$ として，2 つのバケツにデータを分割する例を**図 6**([6] より引用) に示す．この場合，Add-Onion 以外のデータが含まれるタプルは 32 個毎に各バケツに振り分ける．そのタプルに対応する Add − Onion のデータも同じバケツに振り分ける．キーの値に基づいた分割ではないため，グルーピング演算を伴う総和計算の際には結果をまとめる際に追加の計算が必要になる場合があるが，Add-Onion のコピーを行なうことなくデータを平等に分割することができる．そのため，MONOMI 方式を適用した暗号化データベースシステムには Round-Robin 分割が適していると考えられる．

3.2 Add-Onion ファイル読み込み並列化

MONOMI 方式を適用した暗号化データベースシステムでは，Add-Onion は列ごとに独立したファイルで保持される．そのためファイル読み込みの並列化が容易である．ファイル I/O はデータ処理においてボトルネックとなりうる処理であるが，SSD など高速なランダムアクセスを実現したストレージを用いることを想定すると，並列化により高速化が期待できる．

3.3 選択演算を伴う総和計算の並列化技法

MONOMI 方式を適用した暗号化データベースシステムにおいて，選択演算やグルーピング演算のように一部のキーに対応する値のみを選択して総和を求める計算はビットマップ索引を用いて実現される [4]．Add-Onion の暗号文の数分の長さを持つビット列を 1 つの平文ブロックに含めるスライスの数分用意する．各スライスにおいて，選択演算によって n 番目の暗号文に含まれる値が総和に含まれるかどうかをビット列によって表す（総和計算に含まれる場合は 1，そうでなければ 0）これを本稿ではビットマップ索引と呼ぶ．図 7 の例では，Add-Onion の暗号文は 4 つ生成されており，1 つの暗号文に 4 つの値を含んでいるため 4×4 のビットマップ索引が生成される (図 7 上部)．このビットを冪指数としてすべての暗号文を掛け合わせる計算を行なうことで，あるスライスには正しい部分和を含む暗号文を導出することができる (**図 8**)．正しい部分和を含む暗号文を求める計算はスライス毎に独立であるため，並列に処理可能である．

4. 設計と実装

4.1 設計

本研究における提案手法の設計について述べる．概要について**図 9** に示す．図において，緑の四角と青の四角がタスクを表し，矢印が演算を表している．

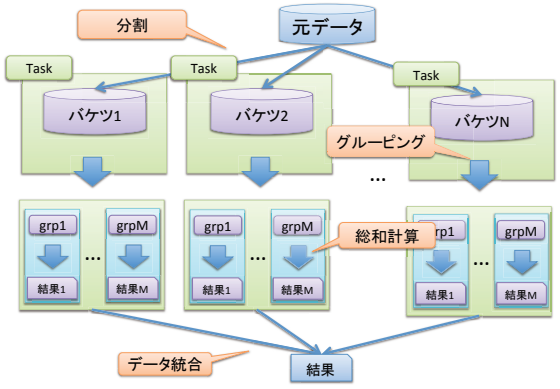

図 9 グルーピング演算を伴う並列総和計算処理の概要

問合せ処理の際，問合せに必要なデータをすべてメモリ上に読み込み，それを N 個のバケツに分割（パーティショニング）する．それぞれのバケツにおいてグルーピング演算を行い，グルーピングキーに応じたビットマップインデックスを生成する．ここで 1 つのバケツにおけるグルーピング演算を 1 つのタスクとして並列化する．ビットマップインデックスが生成されたら，グルーピングキー毎に総和計算を行なう．このグルーピングキー毎の総和計算を 1 つのタスクとして並列処理する．総和計算のタスクの中では更にスライス毎の計算を一つのタスクとして処理している．タスクを細分化することにより，計算負荷がスレッドにバランスよく分散されることが期待できる．

すべてのバケツにおいて計算が終了したら，結果を統合し最終的な結果を出力する．

4.2 実装

3 節で示した並列化手法について，我々が開発しているマイクロ DBMS 上で実装を行った．C++言語を用いて実装し，Paillier 暗号化ライブラリについては [9] を利用した．Paillier 暗号文の処理については GMP ライブラリ [10] を用いた．yacc/lex で生成されたパーサのソースコードを除き，システムのソースコードは 3,830 行であった．コンパイラは gcc 4.7.4 を用いた．グルーピング演算及び総和計算処理の並列化には OpenMP の Task 構文を用いた．

今回は 1 つのノード内でデータを分割し，複数スレッドで並列に処理を行うように実装した．

5. 実験と評価

[6] において提案した総和計算の並列化技法について，評価実験を行った．ワークロードとして図 10 に示す問合せを用意した．

実験に用いるテーブルは**表 1** に示すものを用いた．グルーピングキーとして *key_det* を指定すると 32 個のグループが生成される．*val_add* は 32 個の値をまとめて暗号化しており，1 つの列に対してタプル数の 1/32 の個数の暗号文が配列の形で保持されている．

パラメータとして，1. オペレータ（sum）の数，2. データサイズ，3. スレッド数，4. バケツ数（パーティション数）を設定し，それぞれパラメータを変化させた場合の性能の上下を観察した．また，逐次実行した場合と比較し，評価を行った．結果のグラフにおける処理の名称と処理内容の対応を**表 2** に示す．

実験環境は**表 3** に示す通りである．今回は単一ノードのみでの実験を行った．

5.1 オペレータ数

オペレータの個数を変化させて実験を行った結果について，Q1 の結果を**図 12**，Q2 の結果を**図 13** に示す．縦軸が問合せ処理時間 (msec)，横軸がオペレータの個数となっている．オペレータの個数とは問合せに含まれる総和計算オペレータの数である，例えばオペレータ数が 3 の場合**図 11** のような問合せとなる．

Q1 及び Q2 について，オペレータの個数を 1 から 10 まで変化させて実験を行った．データサイズは 50 万タプル (Add-Onion 40MB，それ以外 8MB)，並列実行の場合バケツ数は 8，スレッド数は 24 に設定した．

単純な総和計算問合せである Q1 においては，オペレータの個数を増やしても並列化による性能向上はほぼ一定であり，今回の実験では平均して 3.70 倍の高速化を達成した．処理時間の内訳を見ると，処理時間の大部分を占める総和計算処理が並列化により 4 倍から 5 倍の高速化をしており，並列化によって生まれる分割と統合の処理時間を加味しても一定の性能向上を示した．

一方でグルーピング演算を伴う総和計算問合せである Q2 においては，オペレータの個数を増やすことで並列化による性能向上が低下し，今回の実験では最大でオペレータ 1 つの時に 19 倍の高速化を達成した．処理時間の大部分を占めるグルーピング演算処理はオペレータの個数によらず一定の処理時間を要し，並列化による性能向上も 58 倍程度で一定であった．今回グルーピング演算について，グルーピングキーがマッチしたタプルをグループが保持するタプルリストに連結する際，グループが保持するタプルリストの先頭から順にリストをたどって末尾を見つけ出し末尾のタプルに連結するというナイーブな実装を行っている．そのためグルーピング演算がタプル数に応じて指数関数的に処理時間の増大する処理となっている．パーティショニングによって各スレッドが参照するタプルリストの長さが短くなるため，58 倍というコア数以上の性能向上を示している．総和計算についてはオペレータの個数に比例して処理時間も増大したが，性能向上は一定であった．一方でオペレータの個数に比例して統合処理にかかる時間が増加しており，問合せ処理全体での性能向上率が低下したのは結合

```
Q1. SELECT sum(val_add) FROM table;
Q2. SELECT sum(val_add) FROM table GROUP BY key_det;
```

図 10 想定する問合せ

表 1 table の内容

カラム名	型	内容
key_det	ENC_DET(uint64_t)	自然数を暗号化した暗号文 (64 bit)．32 種類のランダムな値を取る
val_det	ENC_DET(uint64_t)	自然数を暗号化した暗号文 (64 bit)
val1_add	ENC_ADD(mpz_t)	自然数を暗号化した暗号文 (2048 bit)
val2_add	ENC_ADD(mpz_t)	自然数を暗号化した暗号文 (2048 bit)
val3_add	ENC_ADD(mpz_t)	自然数を暗号化した暗号文 (2048 bit)
val4_add	ENC_ADD(mpz_t)	自然数を暗号化した暗号文 (2048 bit)
val5_add	ENC_ADD(mpz_t)	自然数を暗号化した暗号文 (2048 bit)
val6_add	ENC_ADD(mpz_t)	自然数を暗号化した暗号文 (2048 bit)
val7_add	ENC_ADD(mpz_t)	自然数を暗号化した暗号文 (2048 bit)
val8_add	ENC_ADD(mpz_t)	自然数を暗号化した暗号文 (2048 bit)
val9_add	ENC_ADD(mpz_t)	自然数を暗号化した暗号文 (2048 bit)
val10_add	ENC_ADD(mpz_t)	自然数を暗号化した暗号文 (2048 bit)

表 2 処理の種別

inittable	ファイルからデータを読込，タプルリスト生成
readcolumn	ファイルからデータを読込，Add 暗号文配列生成
split	メモリ上のタプルリスト及び Add 暗号文配列の分割
grouping	グルーピング処理
aggregate	総和計算処理
merge	バケツごとに生成された結果の統合
decrypt	結果の復号処理

表 3 実験環境

OS	CentOS release 6.6 (Final)
プロセッサ	Intel(R) Xeon(R) CPU E5-2695 v2
プロセッサ速度	2.40GHz
コア数	24
メモリ	64 GB

処理のためだと考えられる．

5.2 データサイズ

データサイズを変化させて実験を行った結果について，Q1 の結果を**図 14**，Q2 の結果を**図 15** に示す．縦軸が問合せ処理時間 (msec)，横軸がタプル数となっている．

Q1 及び Q2 について，データサイズを 10 万タプルから 200 万タプルまで変化させて実験を行った．オペレータの個数は 5，並列実行の場合バケツ数は 8，スレッド数は 24 に設定した．

Q1 においては並列化による性能向上はほぼ一定であり，今回の実験では平均 3.69 倍の高速化であった．処理時間の内訳を見ると，データサイズが増加するに従ってデータ分割及び総和計算の処理時間が増加するものの，その増加率は一定であり，並列化による性能向上は一定率得られた．

Q2 においては，データ数増加に従ってグルーピング演算処理の時間が指数関数的に増加している．並列化を行なうとデータ分割処理時間も指数関数的に増大するが，グルーピング演算処理及び総和計算処理による性能向上が大きく，処理全体では性能向上した．グルーピング演算の性能向上率はデータ数増加に伴い大きくなっており，処理全体でもデータ数増加にともなって並列化による性能向上率は大きくなっている．

5.3 スレッド数

スレッド数を変化させて実験を行った結果について，Q1 の結果を**図 16**，Q2 の結果を**図 17** に示す．また，オペレータ数 5，バケツ数 8，データサイズ 50 万タプルのときの処理時間の内訳をそれぞれ**図 18**，**図 19** に示す．縦軸が問合せ処理時間 (msec)，横軸がスレッド数となっている．

Q1，Q2 共に，スレッド数 8 まではスレッド数を増やすに従って問合せ処理時間が削減されたが，その後はスレッド数を増やしてもほぼ一定の処理時間を示した．バケツ数，データサイズ，オペレータ数を増やしてもほぼ同様に 8 スレッドまで性能向上し，その後一定の処理時間を示した．実験を行ったマシンのコア数よりも少ないスレッド数で最大の性能を示す結果となった．

5.4 バケツ数

次に，バケツ数（パーティション数）を変化させて実験を行った結果について，Q1 の結果を**図 20** 及び**図 21**，Q2 の結果を**図 22** 及び**図 23** に示す．縦軸が問合せ処理時間 (msec)，横軸がバケツ数となっている．

Q1 及び Q2 について，バケツ数を変化させて実験を行った．オペレータの個数は 5，スレッド数は 24，データサイズは 50 万タプルと 10 万タプルで実験した．

```
SELECT sum(val1_add), sum(val2_add), sum(val3_add) FROM table;
SELECT sum(val1_add), sum(val2_add), sum(val3_add) FROM table GROUP BY key_det;
```

図 11　オペレータ数 3 の問合せ

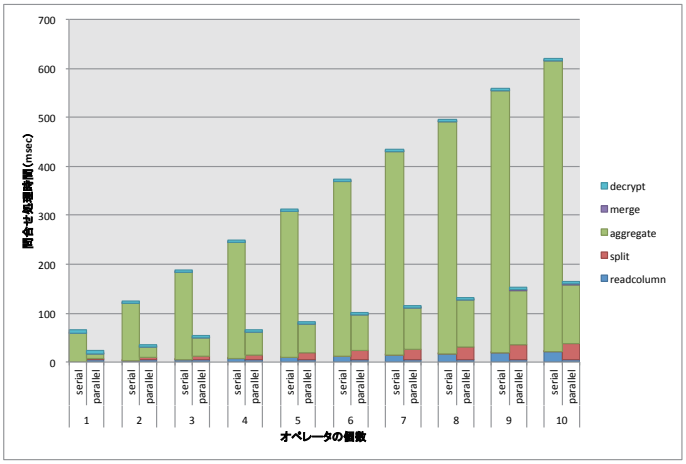

図 12　オペレータ数変更実験 (Q1)

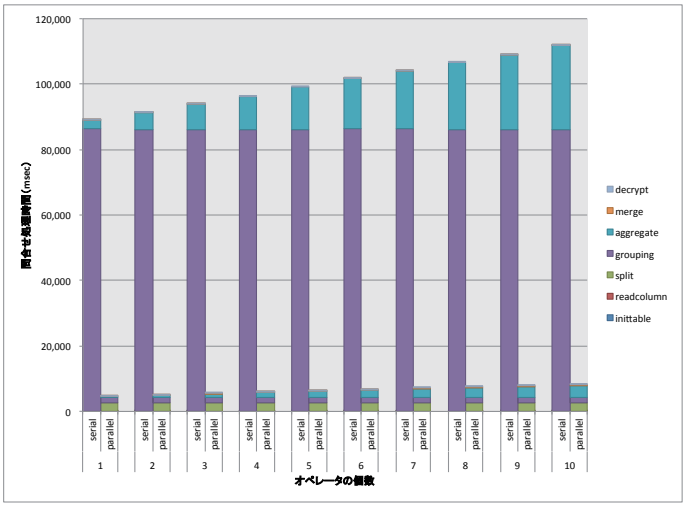

図 13　オペレータ数変更実験 (Q2)

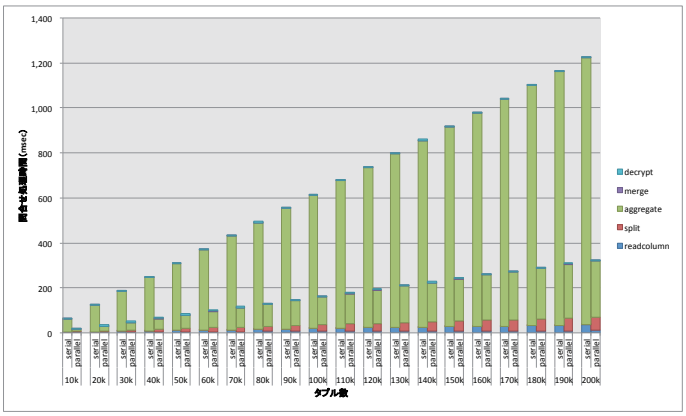

図 14　データサイズ変更実験 (Q1)

単純な総和計算問合せにおいては，データ数を大きくするとバケツ数を増やすことによる分割・統合コストの増大

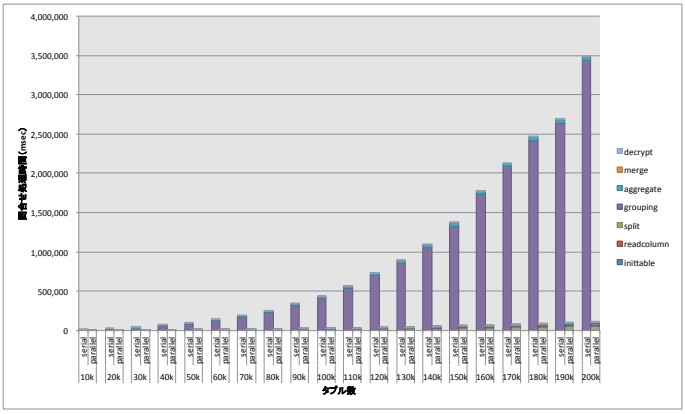

図 15　データサイズ変更実験 (Q2)

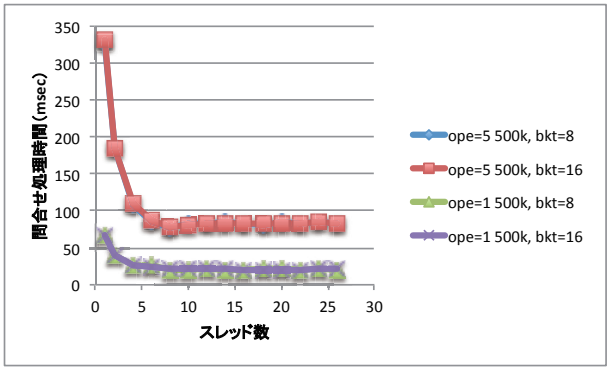

図 16　スレッド数変更実験 (Q1)

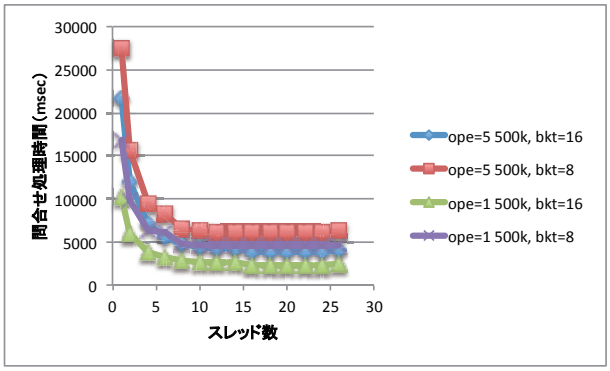

図 17　スレッド数変更実験 (Q2)

の影響は小さくなり，バケツ数によらずほぼ一定の結果が得られた．

グルーピング演算を伴う総和計算問合せにおいては，問合せ処理時間の大部分を占めているグルーピング演算の処理時間がバケツ数を増やすことによって大幅に短縮された．一方でバケツ数増大に従って統合コストが増大するため，性能向上率は低下していった．バケツ数を増やすことにより，グルーピング演算のコストは大幅に削減されるも

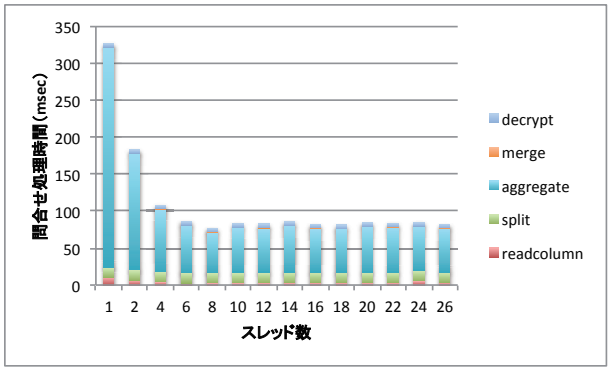

図 18 スレッド数変更実験 (Q1-内訳)

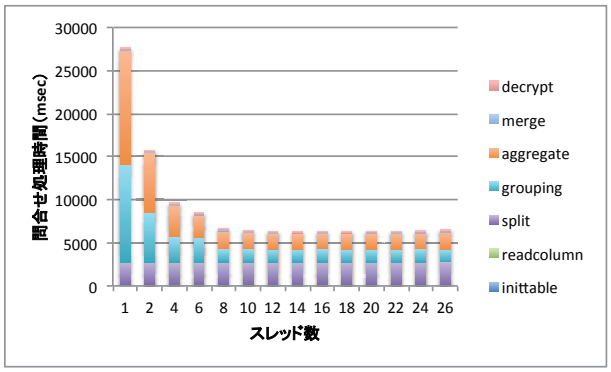

図 19 スレッド数変更実験 (Q2-内訳)

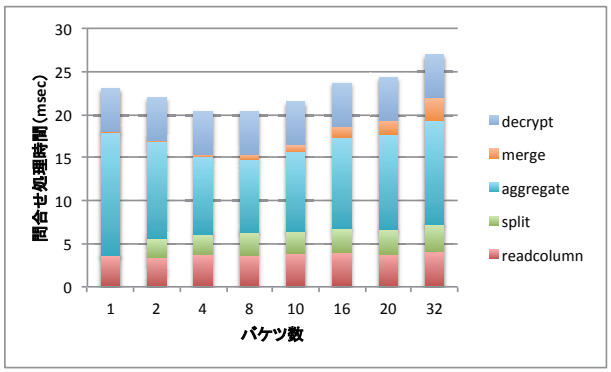

図 20 バケツ数変更実験 (Q1-100k)

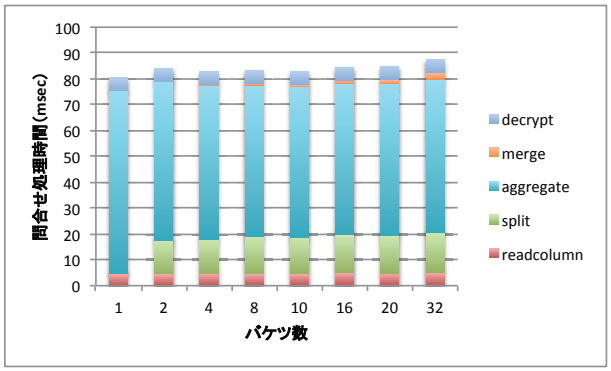

図 21 バケツ数変更実験 (Q1-500k)

ので，最後の統合処理にかかるコストが増大することが結果から示された．しかしながら，データ数を増やすことで

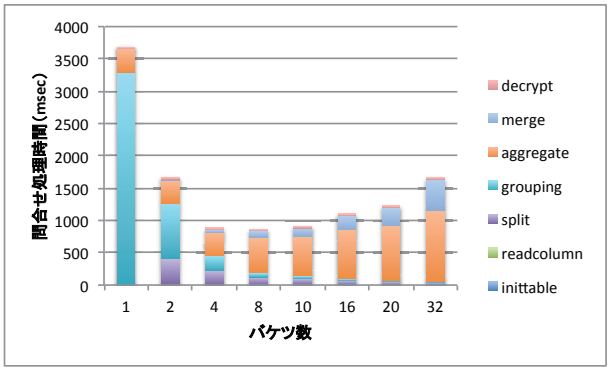

図 22 バケツ数変更実験 (Q2-100k)

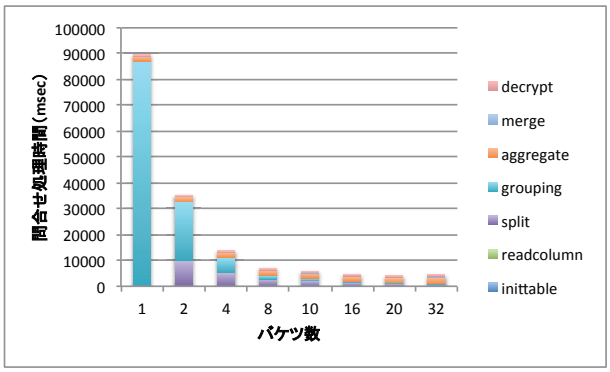

図 23 バケツ数変更実験 (Q2-500k)

統合コストの影響が小さくなり 50 万タプルでの実験では，バケツ数 20 の時点でもっともよい性能が得られた．

総和計算処理に注目すると，データ数が少ない時にはバケツ数を増やすことで処理時間が増大した．一方でデータを増やすと，バケツ数を増やしても処理時間にほとんど変化が見られなかった．データ数が少ない時に処理時間が増大したのは，1 タスクの仕事量が小さく，タスクの分配や同期のためのコストが明らかになってしまっているためだと推測される．一方データ数が大きくなると 1 タスクの仕事量が大きくなり，かつスライス毎の総和の計算を 1 タスクとして並列化していることで仕事が細粒度となることから，各スレッドにバランスよく振り分けられられると考えられる．そのため，データ数が大きい場合はデータの分割を増やしてもスレッド間の仕事量のバランスに影響がなく処理時間が一定になると推測される．

以上の結果からバケツ数については，データサイズに応じて適切に設定する必要があると言える

6. 関連研究

本研究は準同型暗号で暗号化したデータにおける総和計算を効率化するものである．

[4] は MONOMI[3] で採用されている総和計算の効率化手法を提案している．本研究はこの手法の演算部分を並列化するものである．

[11] は暗号化データベースにおける効率的な集計処理を

目的とした研究であるが，Paillier 暗号ではなく Privacy homomorphism という暗号手法を用いている．また，並列化などについては議論されていない．

[12] は CryptDB の仕組みをストリームデータ処理に適用したものである．本研究と同様に加算のための暗号手法として Paillier 暗号を用いているが，その並列化については言及されていない．

7. まとめ

本稿では，我々がこれまでに提案した暗号化データベースシステムにおける総和計算の並列化手法について，オペレータの数，データサイズ，スレッド数，バケツ数（パーティション数）を設定し，それぞれパラメータを変化させた場合の性能の上下を観察した．また，逐次実行した場合と比較し，評価を行った．

単純な総和計算問合せについてはデータ数及びオペレータ数を増加させても一定率で性能向上することを示した．グルーピング演算を伴う総和計算問合せについては，オペレータ数を増やすと性能向上率が小さくなる一方で，データサイズを増やすと性能向上率が大きくなるという結果が得られた．一方でスレッド数およびバケツ数についてはデータ数，問い合わせの種類及び問合せを実行するマシンによって最適数が異なることが考えられ，網羅的に調査する必要がある．

今後はより大きなデータや実データを用いた実験，異なるコア数の環境での実験，Paillier 以外の暗号手法での評価，複数ノードを用いた並列並列問合せ処理などを検討している．

謝辞 本研究の一部は，JST CREST「ポストペタスケールデータインテンシブサイエンスのためのシステムソフトウェア」，JST CREST「EBD：次世代の年ヨッタバイト処理に向けたエクストリームビッグデータの基盤技術」，JST CREST「広域撮像探査観測のビッグデータ分析による統計計算宇宙物理学」，科研費「#25280043HA」，JST A-STEP(#AS262Z02895H) による．

参考文献

[1] Popa, R. A., Redfield, C. M. S., Zeldovich, N. and Balakrishnan, H.: CryptDB: Protecting Confidentiality with Encrypted Query Processing., *SOSP*, pp. 85–100 (2011).
[2] Oracle Corporation: *MySQL 5.7 Reference Manual*, https://dev.mysql.com/doc/refman/5.7/en/encryption-functions.html.
[3] Tu, S., Kaashoek, M. F., Madden, S. and Zeldovich, N.: Processing Analytical Queries over Encrypted Data., *PVLDB 6(5)*, pp. 289–300 (2013).
[4] Ge, T. and Zdonik, S. B.: Answering Aggregation Queries in a Secure System Model., *VLDB*, pp. 519–530 (2007).
[5] 堀尾健太郎，川島英之，建部修見：暗号化データベースシステムにおける効率的な集約処理の評価，情報処理学会研究報告システムソフトウェアとオペレーティング・システム（OS）， Vol. 2015-OS-133, No. 19, pp. 1–10 (2015).
[6] 堀尾健太郎，川島英之，建部修見:：暗号化データベースシステムにおける総和計算処理の並列化，情報処理学会研究報告ハイパフォーマンスコンピューティング（HPC），Vol. 2015-HPC-150, No. 23, pp. 1–6 (2015).
[7] Paillier, P.: Public-Key Cryptosystems Based on Composite Degree Residuosity Classes., *EUROCRYPT*, pp. 223–238 (1999).
[8] Copeland, G. P. and Khoshafian, S.: A Decomposition Storage Model, *ACM SIGMOD*, pp. 268–279 (1985).
[9] Bethencourt, J.: Paillier Library, Advanced Crypto Software Collection (online), available from ⟨http://acsc.cs.utexas.edu/libpaillier/⟩ (accessed 2015-04-20).
[10] GNU project: The GNU MP Bignum Library, GNU-project (online), available from ⟨https://gmplib.org/⟩ (accessed 2015-04-20).
[11] Hacıgümüş, H., Iyer, B. and Mehrotra, S.: *Database Systems for Advanced Applications: 9th International Conference, DASFAA 2004, Jeju Island, Korea, March 17-19, 2003. Proceedings,*, chapter Efficient Execution of Aggregation Queries over Encrypted Relational Databases, pp. 125–136 (2004).
[12] Tomiyama, K., Kawashima, H. and Kitagawa, H.: A Security Aware Stream Data Processing Scheme on the Cloud and Its Efficient Execution Methods, *Proceedings of the Fourth International Workshop on Cloud Data Management*, CloudDB '12, ACM, pp. 59–66 (2012).