DEIM Forum 2017 H6-5

秘匿検索フレームワークOSITにおける最適なクエリプラン選択法

秋山　賢人† 　渡辺知恵美†† 　北川　博之†††

† 筑波大学大学院システム情報工学研究科　〒305–8573 つくば市天王台 1–1–1
†† 筑波大学システム情報系　〒305–8573 つくば市天王台 1–1–1
††† 筑波大学計算科学研究センター　〒305–8573 つくば市天王台 1–1–1
E-mail: †k_akiyama@kde.cs.tsukuba.ac.jp, ††chiemi@cs.tsukuba.ac.jp, †††kitagawa@cs.tsukuba.ac.jp

あらまし　DBaaS を利用したデータの検索サービスでは, データの所有者と管理者が異なるためデータの機密性を保証する必要がある他, 検索者が発行したクエリの内容を保護することも重要である. データやクエリのプライバシを保護しつつ高速な検索処理を行うため, 先行研究でわれわれは暗号化索引を利用した秘匿検索フレームワーク OSIT を提案している. OSIT では索引をクライアントとサーバで分散管理し, クライアントがサーバと通信して索引探索を行うことで問合せを安全かつ効率的に行うことができる. OSIT では索引を利用したクエリ処理のみを想定しているが, データの分布やクエリ選択率, 通信コストによっては索引を利用しない方が高速に処理できる場合もある. そこで, 本研究では索引探索時の通信コストを考慮し, OSIT と索引を利用しない SDB のコストモデルを検討し, コストモデルにもとづいた最適なクエリプランを選択する手法を提案する.

キーワード　プライバシ保護, 秘密計算, Database as a Service

1. はじめに

クラウドコンピューティングの発達により, DBaaS(Database as a Service) が広く提供されている. DBaaS とはネットワークを介してデータベースの機能を提供するサービスであり, データ所有者がサーバ管理者の管理するクラウドデータベースにデータの管理を委託する事ができる. 現在では, Amazon RDS [1], Google Cloud Bigtable [2], Oracle Enterprise Manager 12c [3] などが商用のサービスとして挙げられる. DBaaS では, データベース管理を委託してユーザの負担軽減が可能である他, データ量に応じてデータベースの柔軟な拡張も行うことができる. このように, DBaaS を利用したデータ管理には大きな利点があると言える.

しかし, 機密データを預ける場合には, 機密データやクエリの漏洩によるプライバシの侵害が重大な問題となりうる. 機密データやクエリに関するプライバシ保護を考慮するために, 以下のようなデータやクエリに関するプライバシの要求がある.

データプライバシに関する要求

・データ所有者はサーバ管理者に対し, 預けるデータの中身を知られたくない.

・データ所有者はクライアントに対し, 必要以上の情報を与えたくない.

クエリプライバシに関する要求

・クライアントはデータ所有者およびサーバ管理者に対し, 発行したクエリの内容を秘匿したい.

このような要求を満たすために, 暗号化データベースシステムが提案されている. 暗号化データベースシステムではデータの機密性を保証するために, あらかじめデータを暗号化してからクラウドサーバに保存する. そして, 保存されているデータに対して検索可能暗号を利用することで安全に検索を行う. 暗号化データベースシステムに関する研究としては CryptDB [20], Monomi [21], SDB [22] などがある.

われわれは, これまでにデータとクエリに関するプライバシを保護しつつ暗号化索引を利用した効率的な検索を可能にする検索可能暗号フレームワーク OSIT(Oblivious Secure Index Traversal) [23] を提案している. OSIT では, 索引を暗号化してクライアントとサーバで分散管理する. クライアントは索引探索に必要な情報を所有し, サーバは索引の各エントリを暗号化した情報を所有する. サーバとクライアントはどちらも完全な索引の情報を持たないため, 索引探索の際にはクライアントがサーバにアクセスして探索を行う. 索引の探索は, 暗号化したまま行うことでデータとクエリの機密性を保証する.

現在, OSIT は索引を利用した検索処理のみを想定している. しかし, データ数や通信コストを考慮することで実際には索引を利用しない方が高速に検索処理できるという場合も考えられる. また, OSIT では, 「SELECT * FROM employee, company WHERE age < 30 AND employee.employeeID = company.employeeID」のような複数の演算を含む問合せに対して演算間で相互運用性を持たせることも重要な課題である.

そこで, 本研究では索引を利用する OSIT [23] [24] と索引を利用しない SDB [22] の 2 つについて問合せ処理のコストモデルを検討し, 最適なクエリプランの選択方法を提案する. SDB は準同型暗号を利用したフレームワークであり, OSIT が課題としている問合せ処理における演算間での相互運用性を持つ. われわれは, SDB における演算の一部に OSIT の索引探索手法を利用することで, 相互運用性を持った効率的なフレームワークを形成できると考えられる. また, 実験では提案したコスト

モデルにもとづくプラン選択が適切に行われるかの評価を行い, 提案したコストモデルの有効性を検証する.

本稿の構成は以下のようになっている. まず, 2 節で関連研究について述べ, 3 節で本研究を説明する上で必要となる前提知識について述べる. 次に, 4 節で利用するフレームワークについて述べ, 5 節で提案手法について述べる. 6 節で提案手法の評価実験について述べ, 7 節でまとめと今後の課題について述べる.

2. 関 連 研 究

DBaaS 環境における暗号化データベースシステムに関する研究は, Hacigümüs ら [12] によって提案されて以来, 数多く取り組まれてきた. Hacigümüs らはサーバとクライアント間の問合せ処理モデルを定義した. Hore [13] らは, Hacigümüs [12] らの手法をもとに, 統計による値の推測を困難にするサーバ側のデータ生成法を提案した. Agrawal [4] は数値属性の順序関係を保存できる順序保存暗号 (OPE) を提案し, Lee ら [16] や Hasan ら [15] などにより改良手法が提案されている. また Mykletun ら [18] や Ge ら [9] によって準同型暗号を利用した集約演算を可能にするサーバ側でのデータ暗号化手法や, k-近傍探索なども提案されている. 2011 年には, Popa ら [20] がこれらの手法を組合せて CryptDB を提案しオープンソースパケージとして公開した. CryptDB は,RND, DET, Paillier 暗号 [19] などの異なる暗号化スキームを用いて機密データを幾重にも暗号化することで機密性を保証している. また, 各暗号化スキームごとに異なるクエリの処理を行い, キーワード検索, 結合演算, 大小比較など様々な演算をサポートしている. Stephen らは CryptDB の検索処理最適化機構を改良した Monomi [21] を提案した.

近年では, 準同型暗号や検索可能暗号を利用した検索手法が提案されている. 準同型暗号には, 暗号化したまま暗号文同士の加算が可能な加法準同型暗号や, 暗号文同士の乗算が可能な乗法準同型暗号がある. 代表的な加法準同型暗号には Paillier 暗号 [19] が, 乗法準同型暗号には ElGamal [8] 暗号などがある. また Gentry らにより整数上完全準同型暗号が提案 [10] されたが, この手法はビットごとの演算が必要なため実用的でない [11]. Boneh ら [7] は, 加算と数回の乗算が可能な Somewhat 準同型暗号を利用した共通集合演算を提案している. また, Ma ら [17] は準同型暗号である DBGN 暗号 (deterministic BGN) と Bloom Filter を利用した結合演算を提案している. また, 検索可能暗号には ID ベース暗号をもとにしたキーワード検索公開鍵暗号 (PEKS) [6] があり, キーワード検索を行うことができる.

暗号化データベースシステムでは, 関係代数のようにクエリを構成する各演算の間に相互運用性 (interoperability) を持たせることも重要な課題である. Wong ら [22] は相互運用性のある加法準同型暗号を用いて実装した SDB を提案した. SDB では中間結果を一度クライアントに転送して処理をしなくても, サーバで暗号化したまま演算の出力結果を次の演算の入力とすることができる.

これまでに述べた暗号化データベースシステムは高い安全性を保証する一方で, 大規模データに対する検索を行う場合には良いパフォーマンスを示さない. データ数が大きい場合, これらの暗号化手法では 1 つ 1 つのレコードが検索条件に一致するかを確認しなければならず, 検索時間がかかってしまう.

そこで, Hu ら [14] はデータベース索引を用いた安全な検索フレームワーク OIT を提案した. OIT では, クライアントが索引を所有し, クライアント側で索引探索を行うことにより安全性を確保している. しかし, クライアントに対するプライバシ要件が侵害されるとクライアントに大きな負荷がかかるという問題がある. この問題の解決策として, 篠塚ら [23] はサーバとクライアントで索引を分散配置した OSIT フレームワークを提案している. その後, OSIT の検索コスト最適化 [24] が提案されている.

3. 前 提 知 識

3.1 Paillier 暗号

本稿では, 索引の暗号化に Paillier 公開鍵暗号 [19] を利用する. 平文を m, 公開鍵を $pk = (n, g)$, r をランダムな値とする. m の暗号文を $E(m)$ とすると, 暗号化は以下の式で表すことができる.

$$E(m) = g^m \cdot r^n mod\ n^2$$

Paillier 暗号は加法準同型性を持ち, 2 つの平文 m_1, m_2 の暗号文から $m_1 + m_2$ の暗号文を計算することができる. 暗号文同士の加算は以下の式 1 で表すことができる.

$$\begin{aligned} &E(m_1) \cdot E(m_2) \\ &\quad = (g^{m_1} \cdot {r_1}^n mod\ n^2) \cdot (g^{m_2} \cdot {r_2}^n mod\ n^2) \\ &\quad = g^{m_1+m_2} \cdot (r_1 \cdot r_2)^n mod\ n^2 \\ &\quad = E(m_1 + m_2) \end{aligned} \tag{1}$$

また, k を定数として以下の式 2 が成り立つ.

$$\begin{aligned} E(m)^k &= (g^m \cdot r^n)^k mod\ n^2 \\ &= g^{km} \cdot (r^k)^n mod\ n^2 \\ &= E(km) \end{aligned} \tag{2}$$

4. 利用するフレームワーク

われわれはこれまでに索引を利用して問合せを安全かつ効率的に行うフレームワーク OSIT [23] [24] を提案している. また, 索引を利用しない問合せ手法は CryptDB [20] や Monomi [21], SDB [22] などの手法がある. この中でも SDB は加法準同型性を持つ暗号化スキームを利用し, OSIT で課題とされている演算間での相互運用性があることから「SELECT * FROM employee WHERE age < 30 AND salary > 400,000」のような複数演算を含む問合せに対応している. また, SDB では CryptDB を改良した Monomi との比較実験で高速な処理を行えることも示されているため, 本研究におけるコストモデルの検討に利用する.

本稿では, 索引を利用した OSIT のコストモデルと索引を利用していない SDB [22] のコストモデルを提案し, コストモデルにもとづいた最適なクエリプランの選択を行うことを目標とす

る．そのため，4 節では拡張した OSIT フレームワークと SDB の概要について述べる．

4.1 問題設定

OSIT と SDB において「SELECT * FROM employee WHERE age < 30」のような問合せを行うことを考える．問合せを安全に行うために，満たすべきプライバシ要求を以下に挙げる．

- サーバ管理者は暗号化されたデータ及びクエリ履歴から元の値を推測できない．
- クライアントは発行した問合せ結果以外の情報を得ることができない．

4.2 拡張 OSIT フレームワーク

まず，先行研究 [23] [24] におけるデータモデルについて述べる．先行研究では，データ所有者，クライアント，サーバの 3 者を考える．先行研究では，クライアントとサーバが索引を分散管理し，問合せの際にはクライアントとサーバが通信することで完全な索引の情報を知らないまま索引探索を行うことができる．以降で，問合せ処理に利用する暗号化索引の作成，索引探索の方法ついて説明する．

データ所有者が持つ n レコードのテーブルについて属性 A を例に，図 1 で暗号化索引の作成方法を示す．データ所有者は

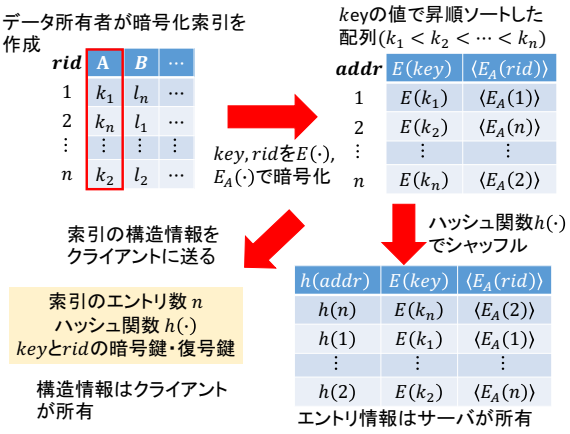

図 1　属性 A における暗号化索引の作成

図 1 上部で属性値 key とレコード ID rid を Paillier 暗号 [19] $E(\cdot)$，加法準同型性を持つ Wong らの暗号化スキーム [22] $E_A(\cdot)$ でそれぞれ暗号化し，key 値で昇順ソートした配列を索引に利用する．配列には $E(key)$ と暗号化されたレコード ID リスト $\langle E_A(rid)\rangle$ が格納されている．

データ所有者は元の索引の順序関係を秘匿するために，配列のアドレス $addr$ をハッシュ関数 $h(\cdot)$ でハッシュし，図 1 下部右側に示すように配列の要素にハッシュ値を加えたエントリ情報をサーバに保存する．また，データ所有者は図 1 下部左側に示すようにクライアントに $h(\cdot)$, key, rid の暗号鍵，復号鍵を与える．このように索引を分散管理することで，サーバは索引のノード間関係を把握できず，クライアントは各ノード対してサーバ上のアドレスのみを知り，ノード自体を取得できない．

次に，索引の探索について説明する．先行研究 [23] [24] のモデルでは 2 台のサーバを利用していたが，索引の探索方法を改善 (詳しくは付録参照) したため，ここでは 1 台のサーバで説明する．(注1) 索引の探索手順を図 2，Algorithm1 に示す．まず，クライアントは探索範囲 $[l, u]$ を $[0, n-1]$ で初期化する (Step 0)．クライアントはクエリの漏洩を防ぐために，Paillier 暗号で暗号化した $E(q)$ をサーバに送る (Step 1)．続いて，クライアントは $(m-1)$ 個のアドレスリスト $h(i_1)$, $h(i_2)$, …, $h(i_{m-1})$ をサーバに送る (Step 2)．サーバはクエリ値とエントリ値の大小比較を行い (Step 3)，クライアントに $E(s_1)$, $E(s_2)$, …, $E(s_{m-1})$ を送る (Step 4)．クライアントは比較結果を復号し (Step 5)，探索範囲を更新する (Step 6)．(Step 2) から (Step 6) を繰り返し行うことで索引を探索してクエリに該当するアドレスリストを得る．

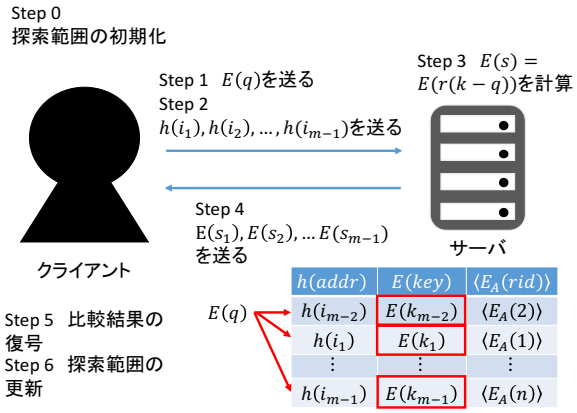

図 2　索引の探索

Algorithm 1 OSIT フレームワークにおける索引の探索

Procedure:

1: $(l, u) \leftarrow (0, n-1)$ // Step.0
2: client sends $E(q)$ to server // Step 1
3: **while** the search is not terminated **do**
4: client sends $h(i_1)$, $h(i_2)$, …, $h(i_{m-1})$ to server // Step 2
5: **for** j = 1 to m-1 **do**
6: server performs $E(s_j) = E(r(k_j - q))$ // Step.3
7: **end for**
8: client decrypts the result // Step.4
9: client updates $[l, u]$ // Step.5
10: **end while**

4.3 SDB

SDB [22] は問合せを安全に行うフレームワークであり，データの相互運用性 (interoperability) を持つ加法準同型性の暗号化スキームを提案している．データの相互運用性とはクライアントが複数演算を含む問合せを発行した際に，各演算の評価結果を次の演算の入力とすることができるという性質のことである．これにより大小比較，結合演算などの様々な演算をサポー

(注1)：[23] [24] では索引の探索に GTSCOT [5] を利用していた．GTSCOT は，送信者 S と受信者 R がお互いの持つ値を知ることなく大小比較を行うことができるプロトコルである．先行研究では，クエリを平文でサーバに送るため，サーバを 2 台に分けてクエリの安全性を保証している．

トしている．また，SDB では暗号化したまま暗号鍵を別の鍵に更新する関数が定義されていることで，異なる暗号鍵で暗号化した値同士の比較も行うことができる．

SDB ではデータ所有者がレコードをカラムごと異なる鍵で暗号化してサーバで保存する．暗号化の際には，カラムごとの鍵 $ck = \langle m, x \rangle$ とレコード ID r を利用して式 3 で暗号鍵 v_k を生成する．(ρ_1，ρ_2 を巨大な素数として $n = \rho_1\rho_2$，$\phi(n) = (\rho_1 - 1)(\rho_2 - 1)$ とする)．

$$v_k = mg^{rxmod\ phi(n)} mod\ n \tag{3}$$

v を平文，v_e を暗号化した値とする．SDB では以下の式 4 で暗号化する．

$$v_e = vv_k{}^{-1} mod\ n \tag{4}$$

この暗号化スキームは加法準同型性をもつ．

ここからは，提案手法でコストモデルを検討するために SDB の問合せ処理について説明する．SDB における問合せ処理を図 3 に示す．サーバは図 3 右下にあるように属性ごとに異なる暗号鍵で暗号化したリレーショナルテーブルを持つ．また，サーバで暗号化されたレコード ID をリレーショナルテーブルに追加して持っている．クライアントは，レコード ID，各属性を暗号化した暗号鍵，復号鍵を持つ．まず，クライアントは暗号化したクエリ $E_Q(q)$ と鍵更新に利用するパラメータをサーバに送る (Step 1)．次に，サーバでクエリと対象となる属性の鍵を統一し，すべてのレコードに対して暗号化したままクエリとデータの比較を行うことで問合せを実現する (Step 2)．ランダムな値 r を鍵 R で暗号化した値を $E_R(r)$，鍵 A で属性 A の値を暗号化したものを $E_A(k_i)(i = 1, 2, \ldots, n)$ とし，暗号文 $E_Q(q)$ に関して鍵 Q から鍵 A への鍵更新関数を $ChangeKey(E_Q(q), Q, A)$ とする．$E_{res}(s_i)$ を比較演算の結果として (Step 2) で行う比較演算は式 5 で表せる (鍵更新式，乗算方法については付録を参照)．

$$E_{res}(s_i) = E_R(r)(E_A(k_i) - ChangeKey(E_Q(q), Q, A)) \\ (i = 1, 2, \ldots, n) \tag{5}$$

res は計算後の暗号鍵である．クエリと各レコードの比較結果 $E_{res}(s_i)(i = 1, 2, \ldots, n)$ に対して鍵更新を行い結果を平文で求め (付録参照)，結果の正負により大小関係を判定することができる．クエリは暗号化してサーバへ送るためサーバに知られることはなく，クエリと各レコードの大小比較結果も r によりランダム化されているためクライアントに知られない．

5. 提案手法

ここでは，4 節で説明した索引を利用したフレームワークである拡張 OSIT の検索のコストモデルと索引を利用しない SDB [22] の検索手法のコストモデルを検討する．そのため本研究では，問合せの条件が 1 つの場合について拡張 OSIT と SDB のコストを考える．6 節で最適なプランが選択できるか評価するために，拡張 OSIT，SDB のコストモデルについて順に説明する．

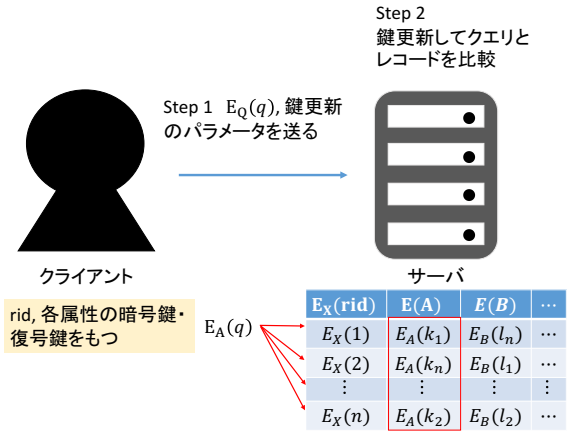

図 3　SDB における問合せ処理

以下，リレーショナルテーブルのレコード数を R，各索引のエントリ数を N，x 割のレコードが結果に含まれるとしてコストモデルを検討する．ここで，統計情報の x は安全に利用できると仮定する．また，OSIT の索引探索には m 分探索を利用する．表 1 に拡張 OSIT と SDB におけるデータの転送時間と計算時間を表すパラメータを示し，コストモデルに利用する．

変数名	説明
t_{query}	クライアントからサーバへの暗号化したクエリの転送時間
t_{hash}	OSIT におけるクライアントからサーバへのハッシュ値 1 つあたりの転送時間
t_{param}	クライアントからサーバへの鍵更新のパラメータ転送時間
t_{keyup}	鍵更新の計算時間
t_{comp}	SDB におけるサーバでのクエリ値と 1 レコードの比較にかかる時間
$t_{idxcomp}$	OSIT におけるサーバでのクエリ値と 1 レコードの比較にかかる時間
$t_{idxcompres}$	OSIT におけるサーバからクライアントへの比較結果 1 つあたりの転送時間
t_{dec}	OSIT におけるクライアントでの暗号文 1 つあたりの復号にかかる時間
R	リレーショナルテーブルのレコード数
N	索引のエントリ数

表 1　拡張 OSIT, SDB のコストモデルに用いた変数名と変数名の説明

5.1 拡張 OSIT におけるコストモデル

まず，拡張した OSIT [23] の索引の探索におけるコストモデルについて考える．図 2 の (Step 1) で t_{query}，(Step 2) で $(m-1)t_{hash}$，(Step 3) で $(m-1)t_{idxcomp}$，(Step 4) で $(m-1)t_{idxcompres}$，(Step 5) で $(m-1)t_{dec}$ のコストがかかる．(Step 1) でのクエリ送信は 1 度のみなのでサイクルに含まないとし，(Step 2) から (Step 5) における 1 サイクルあたりの索引探索のコストは以下の式 6 で表せる．

$$COST_{index}^{OSIT} = (m-1)t_{hash} + (m-1)t_{idxcomp} \\ + (m-1)t_{idxcompres} + (m-1)t_{dec} \tag{6}$$

また, [24] から, 暗号化索引が N 個のエントリからなる場合の m 分探索における索引探索のサイクルは $\lceil \frac{log(N)}{log(m)} \rceil$ であることから, 拡張 OSIT における総コストは以下の式 7 で表せる.

$$\begin{aligned} COST_{OSIT} &= t_{query} + \lceil \frac{log(N)}{log(m)} \rceil COST_{index}^{OSIT} \\ &= \lceil \frac{log(N)}{log(m)} \rceil (m-1)(t_{hash} + t_{idxcomp} \\ &\quad + t_{dec} + t_{compres}) + t_{query} \end{aligned} \tag{7}$$

5.2 SDB におけるコストモデル

続いて, 索引を利用しない場合のコストモデルについて検討する. SDB [22] ではクエリに対して 1 つ 1 つのレコードと比較を行わなければならない. 図 3 から, SDB は (Step 1) で t_{query}, t_{param}, (Step 2) で t_{keyup}, Rt_{comp} のコストがかかる. 図 3 の (Step 1) における通信コストは以下の式 8 で表せる.

$$COST_{comm}^{SDB} = t_{query} + t_{param} \tag{8}$$

図 3 の (Step 2) における計算コストは以下の式 9 で表せる.

$$COST_{calc}^{SDB} = R(t_{keyup} + t_{comp}) \tag{9}$$

式 8, 9 より, SDB における総コストは以下の式 10 で表せる.

$$COST_{SDB} = t_{query} + t_{param} + R(t_{keyup} + t_{comp}) \tag{10}$$

5.3 コストモデルについての考察

拡張 OSIT, SDB での総コストはそれぞれ式 7, 10 で表すことができた. ここでは, 定式化したコストモデルについての考察を行う.

まず, 式 7, 10 において t_{query}, t_{hash}, t_{param}, $t_{idxcompres}$ はクライアント, サーバ間で通信を行うことで事前に計算して情報を保持しておくことが可能である. t_{dec} についてはクライアントで, $t_{idxcomp}$, t_{keyup}, t_{comp} についてはサーバで事前に計算して情報を保持しておくことが可能である. m は式 7 で通信コストが最小となる m を計算して使用する.

従って, 定数項 $t_{hash} + t_{idxcomp} + t_{idxcompres} + t_{dec}$ を t_{const1} で置き換えて式 7 は式 11 で表せる.

$$\begin{aligned} COST'_{OSIT} &= \lceil \frac{log(N)}{log(m)} \rceil (m-1) t_{const1} \\ &\quad + t_{query} \end{aligned} \tag{11}$$

定数項 $t_{query} + t_{param}$ を t_{const2}, $t_{keyup} + t_{comp}$ を t_{const3} で置き換えて式 10 は式 12 で表せる.

$$COST'_{SDB} = t_{const2} + Rt_{const3} \tag{12}$$

コストモデルとしては式 11, 12 で表わせ, N, m, R が決まることでコスト計算を行うことができる. クライアントがクエリを発行する前にサーバと通信を行い, 通信コストと計算コストを測定してクライアント側で保持する. そして, クライアントがクエリを発行したら事前に測定した値をもとにクライアント側の最適化機構により索引利用の有無を以下で判定することができる.

$$\begin{cases} \text{OSIT を利用 } (COST'_{OSIT} < COST'_{SDB}) \\ \text{SDB を利用 } (otherwise) \end{cases}$$

6. 評価実験

6 節では, 5 節で提案したコストモデルをもとに最適なクエリプランを選択できるかの評価を行う.

6.1 実験設定

本実験には, クライアントに Mac OSX, Intel Core i7 @ 3GHz CPU, 16GB RAM で構成された PC を, サーバに Ubuntu, Intel Core i7-2600 @ 3.40GHz CPU, 16GB RAM で構成された PC を使用した.

データセットは属性「A」, 「B」からなり, データ数は 100, 1,000 と 10,000 から 100,000 まで 10,000 ごと増やした 12 種類の人工データを作成した. 各属性値は 0 から 1,000 までの整数値をとる. 索引のエントリ数 N については, データ数 100 では 96, 1,000 では 637, 10,000 から 100,000 では 1,000 となっている. クエリは「SELECT * FROM dataset WHERE A < 10」とし, レコードの選択率は 1%程度とする. 拡張 OSIT と SDB における各データの最大サイズについては表 2 に示す. また, クライアントとサーバの間の通信速度は 3.5Mbps, 32Kbps の場合を考える.

データ名	データサイズ
クエリ	128 bit
ハッシュ値	256 bit
比較結果	128 bit
鍵更新のパラメータ	126 bit

表 2 拡張 OSIT と SDB における各データサイズの最大値

6.2 実験 1

実験 1 では OSIT と SDB について, 実測値が定義したコストモデルに従っているかどうかを検証する. OSIT と SDB について, 3.5Mbps と 32Kbps の場合で事前に計測した値を表 3, 4 に示す. OSIT における m 分探索の $m = 2$ として実験を行った.

変数	計測値 (ms)(3.5Mbps)	計測値 (ms)(32Kbps)
t_{query}	10.74	9.8
t_{hash}	0.16	0.15
$t_{idxcomp}$	0.04	0.03
$t_{idxcompres}$	0.86	0.81
t_{dec}	0.16	0.17

表 3 拡張 OSIT のコストモデルに用いた変数と計測値

変数	計測値 (ms)(3.5Mbps)	計測値 (ms)(32Kbps)
t_{query}	21	21.9
t_{param}	2.5	2.3
$t_{comp} + t_{keyup}$	0.026	0.027

表 4 SDB のコストモデルに用いた変数と計測値

図 4, 5 に通信速度 3.5Mbps での OSIT, SDB における各レ

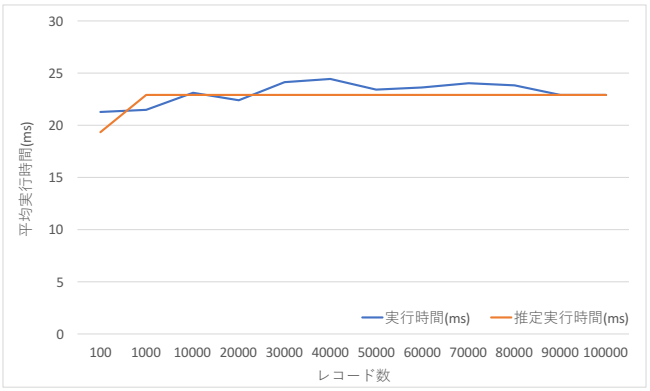

図 4　通信速度 3.5Mbps での OSIT における実行時間と推定実行時間

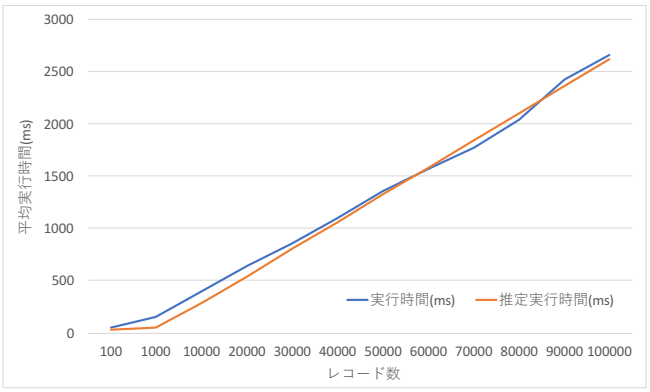

図 5　通信速度 3.5Mbps での SDB における実行時間と推定実行時間

コード数での実行時間とコスト式にもとづく推定実行時間を示す．図 6，7 に通信速度 32Kbps での OSIT, SDB における各レコード数での実行時間とコスト式にもとづく推定実行時間を示す．

OSIT では索引のエントリ数 N, 各処理時間の計測値, 最適な m の値も事前に得ている．これらをコスト式 11 に代入すると図 4，6 のようにエントリ数が一定であれば推定実行時間もレコード数に依存せず一定になることがわかる．図 4，6 より，各レコード数での OSIT の実行時間についても推定実行時間と同様にエントリ数が一定の場合 21.3ms から 24.5ms の範囲に収まっており, コスト式 11 を用いて実際の実行時間を推定することができると考えられる．

SDB では, 各処理時間の計測値, レコード数 R を事前に得ている．これらをコスト式 12 に代入することで図 5，7 で示すように推定実行時間を求めることができる．図 5，7 ではレコード数 100, 1,000 の場合以外で実行時間は R に対して線形に増加していることがわかり, コスト式 12 より, 推定実行時間も R に対して線形に増加する．SDB でもコスト式で求めた推定実行時間と実際の実行時間の差が小さい事がわかるのでコスト式による実行時間の推定は可能であると考えられる．

6.3　実　験　2

実験 2 では, クエリが与えられたら式 11, 12 を計算した値と, 実際に索引を使用した検索時間, 索引を使用しない検索時間を計測し, コストモデルから正しくプランが選択できるかを評価する．実験 2 でも表 3, 4 にあるパラメータを利用し, $m = 2$

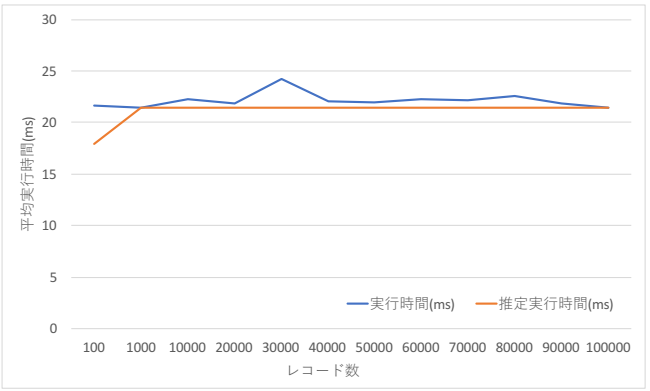

図 6　通信速度 32Kbps での OSIT における実行時間と推定実行時間

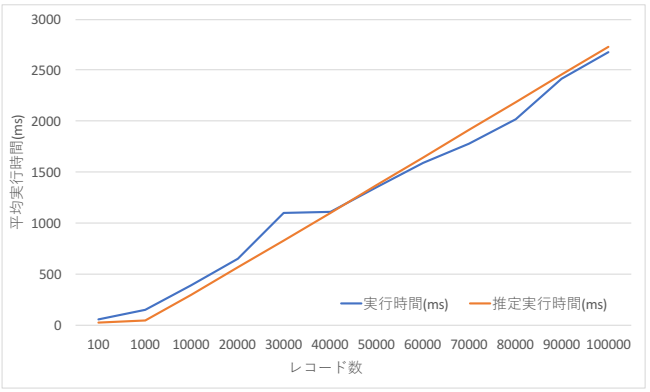

図 7　通信速度 32Kbps での SDB における実行時間と推定実行時間

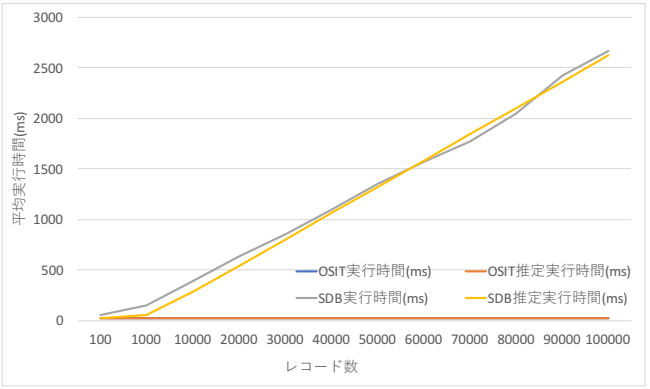

図 8　通信速度 3.5Mbps での OSIT と SDB における推定実行時間と実行時間の比較

とした．図 8 に通信速度 3.5Mbps における OSIT, SDB の実行時間と推定実行時間を示す．図 9 に通信速度 32Kbps における OSIT, SDB の実行時間と推定実行時間を示す．式 11, 12 から計算した推定実行時間を用いることで図 8，9 において, クエリが発行された際にレコード数が 100, 1,000, 10,000 から 100,000 のすべてのデータセットに対して OSIT を利用すれば最適な処理が可能である．本実験では, 通信速度が 3.5Mbps と 32Kbps の 2 つの場合についての評価を行った．実験から, 通信速度が変わっても OSIT と SDB の検索時に送るパラメータのサイズが小さいためレコード数 100 程度でも OSIT が選択されることが示された．

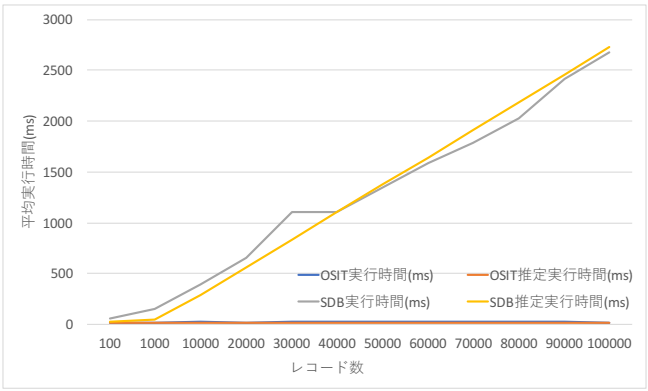

図 9　通信速度 32Kbps での OSIT と SDB における推定実行時間と実行時間の比較

7. おわりに

本研究では, OSIT と SDB について単一条件の問合せに関して通信コスト及び計算コストからなるコストモデルを定義し, 検討したコストモデルにもとづく最適なクエリプラン選択法を提案した. 実験 1 から, OSIT と SDB のそれぞれに対して定義したコストモデルにもとづく計算時間の推測値に OSIT と SDB の実測値が従っていることを示せた. また, 実験 2 から, 問合せの際に索引利用の有無についてコスト計算を行うことで, より高速に問合せ処理を行うプランを選択できるということが示せた. なお, OSIT, SDB において, どちらの場合も通信で送るパラメータのサイズが小さいため通信速度が遅い環境においても OSIT が選択されると示されたが, レコード数が極めて小さい場合には SDB が選択されると考えられる.

謝　辞

本研究の一部は NICT 高度通信・放送研究開発委託研究「欧州との連携による公共ビッグデータの利活用基盤に関する研究開発」, JSPS 科研費 JP16K00149 の助成を受けたものです.

文　献

[1] Amazon RDS. `https://aws.amazon.com/jp/rds`.

[2] Cloud Bigtable - Google Cloud Platform. `https://cloud.google.com/bigtable/?hl=ja`.

[3] Oracle Enterprise Manager 12c. `http://www.oracle.com/technetwork/jp/oem/enterprise-manager/overview/index.html`.

[4] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order-preserving encryption for numeric data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, June 13-18, 2004*, pp. 563–574, 2004.

[5] Ian F. Blake and Vladimir Kolesnikov. Strong conditional oblivious transfer and computing on intervals. In *Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings*, pp. 515–529, 2004.

[6] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, pp. 506–522, 2004.

[7] Dan Boneh, Craig Gentry, Shai Halevi, Frank Wang, and David J. Wu. Private database queries using somewhat homomorphic encryption. In *Applied Cryptography and Network Security - 11th International Conference, ACNS 2013, Banff, AB, Canada, June 25-28, 2013. Proceedings*, pp. 102–118, 2013.

[8] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Information Theory*, Vol. 31, No. 4, pp. 469–472, 1985.

[9] Tingjian Ge and Stanley B. Zdonik. Answering aggregation queries in a secure system model. In *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*, pp. 519–530, 2007.

[10] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pp. 169–178, 2009.

[11] Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, pp. 465–482, 2012.

[12] Hakan Hacigümüs, Balakrishna R. Iyer, Chen Li, and Sharad Mehrotra. Executing SQL over encrypted data in the database-service-provider model. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin, June 3-6, 2002*, pp. 216–227, 2002.

[13] Bijit Hore, Sharad Mehrotra, and Gene Tsudik. A privacy-preserving index for range queries. In *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004*, pp. 720–731, 2004.

[14] Haibo Hu, Jianliang Xu, Xizhong Xu, Kexin Pei, Byron Choi, and Shuigeng Zhou. Private search on key-value stores with hierarchical indexes. In *IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014*, pp. 628–639, 2014.

[15] Hasan Kadhem, Toshiyuki Amagasa, and Hiroyuki Kitagawa. A secure and efficient order preserving encryption scheme for relational databases. In *KMIS 2010 - Proceedings of the International Conference on Knowledge Management and Information Sharing, Valencia, Spain, October 25-28, 2010*, pp. 25–35, 2010.

[16] Seungmin Lee, Tae-Jun Park, Donghyeok Lee, Taekyong Nam, and Sehun Kim. Chaotic order preserving encryption for efficient and secure queries on databases. *IEICE Transactions*, Vol. 92-D, No. 11, pp. 2207–2217, 2009.

[17] Sha Ma, Bo Yang, and Mingwu Zhang. PPGJ: A privacy-preserving general join for outsourced encrypted database. *Security and Communication Networks*, Vol. 7, No. 8, pp. 1232–1244, 2014.

[18] Einar Mykletun and Gene Tsudik. Aggregation queries in the database-as-a-service model. In *Data and Applications Security XX, 20th Annual IFIP WG 11.3 Working Conference on Data and Applications Security, Sophia Antipolis, France, July 31-August 2, 2006, Proceedings*, pp. 89–103, 2006.

[19] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, pp. 223–238,

1999.
[20] Raluca A. Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: processing queries on an encrypted database. *Commun. ACM*, Vol. 55, No. 9, pp. 103–111, 2012.
[21] Stephen Tu, M. Frans Kaashoek, Samuel Madden, and Nickolai Zeldovich. Processing analytical queries over encrypted data. *PVLDB*, Vol. 6, No. 5, pp. 289–300, 2013.
[22] Wai Kit Wong, Ben Kao, David Wai-Lok Cheung, Rongbin Li, and Siu-Ming Yiu. Secure query processing with data interoperability in a cloud database environment. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pp. 1395–1406, 2014.
[23] 篠塚 千愛, 渡辺 知恵美, 北川 博之. DaaS 環境におけるデータとクエリ双方のプライバシ保護を実現する効率的な秘匿検索. DEIM Forum 2015 G2-6 (2015).
[24] 篠塚 千愛, 渡辺 知恵美, 北川 博之. 秘密計算による秘匿検索フレームワーク OSIT における検索コストと最適化. DEIM Forum 2016 F5-6 (2016).

付　　録

OSIT フレームワークにおける索引探索の改善策

OSIT [23] [24] では, 索引の探索の際に GT-SCOT プロトコル [5] を使用して安全に大小比較を行うことができた. しかし, GT-SCOT プロトコル利用の際に, 比較する値をビット表現して Paillier 暗号で暗号化するため, 鍵長が n ビット, 比較する値が m ビットのとき, クライアントはサーバに最大で $2mn$ ビットを送信しなければならならず, 通信コストが大きくなる. そこで, 本稿では通信コスト削減のため Paillier 暗号のみを用いて大小比較を行う. これにより, 鍵長が n ビットだとしても, クエリやエントリの値はビット表現しないものを用いるため, 最大 $2n$ ビットを送信するだけですむため, 通信コストを削減できる. クエリの値を q, エントリの値を k, Paillier 暗号の暗号化関数を $E(\cdot)$, r をサーバで生成したランダムな正の整数とし, サーバ上で以下の式 A·1 を計算してクエリ値とエントリ値の大小比較を行うことができる.

$$\begin{aligned} E(s) &= E(r(k-q)) \\ &= E(k-q)^r \\ &= (E(k)E(-q))^r \end{aligned} \tag{A·1}$$

式 A·1 を利用して暗号化したままサーバで計算し, クライアントで比較結果の復号のみを行うため安全に索引の探索を行うことができる. しかし, Paillier 暗号は負の整数を扱うことはできないので, Paillier 暗号で負の整数を扱うために, 暗号文 $E(m)$ を復号した値が $[0, \frac{n}{2}]$ である場合を正の値とし, 暗号文 $E(m)$ を復号したときに $\frac{n}{2}$ を超えていたら負の値と判定する.

SDB の鍵更新式と乗算方法

ここでは Wong らの暗号化スキーム及び鍵更新式について述べる. まず, 暗号化のためにクライアントは 2 つの秘密の数 g, n を準備する. ここで g と n は互いに素, n は 2 つの大きなランダムな素数 ρ_1, ρ_2 の積である. また, $\varphi(n)$ を以下の式 A·2 で定める.

$$\varphi(n) = (\rho_1 - 1)(\rho_2 - 1).v \tag{A·2}$$

以降で鍵更新式と乗算方法について述べる.

鍵 更 新 式

以下で鍵の更新演算について述べる. まず, データ所有者がデータを預ける際にカラム内の値がすべて 1 のカラム S を暗号化してサーバに保存する.

S のカラムキーを $ck_S = \langle m_s, x_s \rangle$, カラム A の暗号化値を a_e, カラムキーを $ck_A = \langle m_A, x_A \rangle$, 更新後のカラムを C, 暗号化値を c_e, カラムキーを $ck_C = \langle m_C, x_C \rangle$ とする. クライアントは以下の式 A·3 で p, q の値を計算してサーバに送る.

$$\begin{aligned} p &= x_s^{-1}(x_C - x_A) mod\ \varphi(n) \\ q &= m_A {m_S}^p {m_C}^{-1} mod\ n \end{aligned} \tag{A·3}$$

サーバはクライアントから p, q を受け取り, 以下の式 A·4 で c_e 計算する.

$$c_e = q a_e {s_e}^p \tag{A·4}$$

また, カラムキー $ck_C = \langle 1, 0 \rangle$ で鍵更新を行うと式 4 を変形して式 A·5 で表すように平文の値を入手できる. v は平文, v_e は暗号文, v_k は v の暗号鍵を表す.

$$\begin{aligned} v_e &= v v_k^{-1}\ mod\ n \\ &= v \cdot m(g^{rx})^{-1}\ mod\ n \\ &= v \cdot 1 \cdot g^0 mod\ n \\ &= v \end{aligned} \tag{A·5}$$

乗　　算

A, B の暗号化値をそれぞれ a_e, b_e とし, カラムキーを $ck_A = \langle m_A, x_A \rangle$, $ck_B = \langle m_B, x_B \rangle$ とする. a_e と b_e の乗算方法について述べる. クライアントは乗算後の鍵を式 A·6 で計算する.

$$\begin{aligned} m_C &= m_A m_B\ mod\ n \\ x_C &= x_A + x_B\ mod\ \varphi(n) \end{aligned} \tag{A·6}$$

乗算結果を c_e とし, サーバは式 A·7 で暗号化された値を書き換える.

$$c_e = a_e b_e\ mod\ n \tag{A·7}$$