完全準同型暗号を用いた頻出パターンマイニング委託システムの分散処理等による高速化手法の実装と評価

理学専攻・情報科学コース　山本百合

1　はじめに

近年，データ所有者が外部機関にデータマイニング等の分析調査を委託することが一般的であるが，データのプライバシ保護の観点から，有用なデータであるにもかかわらず，データマイニング計算の外部委託が困難な環境が多く見られる．そのため，データを暗号化した状態で乗算と加算の操作が可能な完全準同型暗号を利用することで，安全な委託計算システムの構築を目指す研究が近年盛んである．

先行研究 [1] では，完全準同型暗号を Apriori アルゴリズムによる秘匿データマイニングに適用し，アルゴリズムの高速化を進めている．しかしながら，完全準同型暗号演算は計算量が大きいためにサーバ上の計算負荷が大きくなりやすい．本研究では，秘匿データマイニング計算委託システムのサーバ上の演算に対し，マスタ・ワーカ型の分散処理を実装し，高速化を試みた．また Apriori アルゴリズムに対して，データベースの更新に伴う再計算を高速化する FUP アルゴリズム [2] を適用し，秘匿データマイニング計算委託システムを構築し，クラウドコンピューティングを想定した環境下で実験を行った．

2　完全準同型暗号

完全準同型暗号 (Fully Homomorphic Encryption) は，式 (1), (2) のように暗号文同士の加算と乗算の演算が成立する性質を持ち，暗号化した状態で平文と同様の多項式演算が可能な暗号である．完全準同型暗号は公開鍵暗号方式の機能を持ち，データの暗号化・復号に秘密鍵を，暗号文同士の演算を公開鍵で使用することで，データを秘匿化した状態で，暗号文の平文状態に対する演算結果を暗号化した状態で導くことができる．そのため，秘匿データマイニングに完全準同型暗号を適用することで，データを秘匿した状態で，サーバがデータを用いた統計計算を行えるようなシステムへの応用が期待されている．

完全準同型暗号

$$Encrypt(m) \oplus Encrypt(n) = Encrypt(m+n) \quad (1)$$
$$Encrypt(m) \otimes Encrypt(n) = Encrypt(m \times n) \quad (2)$$

完全準同型暗号の概念自体は，1978 年に Rivest らによって議論され [3]，privacy homomorphism と呼ばれていた．これは Rivest らが，現在公開鍵暗号として広く用いられている RSA 暗号を発見した翌年である．RSA 暗号や ElGamal 暗号は乗法においては準同型暗号の性質を持つが，加法と乗法の両方において準同型暗号として成立するような性質を持ち，更に演算に対して回数制限の無い暗号は，しばらくの間，実行可能な手法が現れなかった．そのため，暗号研究者の間では完全準同型暗号の具体的な実装を考案することが，未解決問題のような扱いであった．2009 年に Gentry が完全準同型暗号を実現する手法を提案したことにより [4]，様々な研究者によって完全準同型暗号の改良が活発に進められるようになり，ライブラリなどの実装も多く見られるようになる．その結果，当初は計算量の大きさから実用性が乏しかった完全準同型暗号は，高速化が進められ，簡単な計算あるいは一部の方式において，十分な性能レベルを示している．しかし依然として，複雑な計算や大きなデータに対する演算では暗号文サイズが大きくなる傾向にあり，実用化を考慮すると現段階も計算量が大きい印象を受ける．

3　提案手法

3.1　FUP アルゴリズム

FUP（Fast UPdate）アルゴリズム [2] は 1996 年に Cheung らによって考案された手法であり，データベースのトランザクション追加後の Apriori 計算時に効率良くデータベース走査をするアルゴリズムである．Apriori は静的なデータを扱うことを前提にサポート値算出を行っていたが，データベースにトランザクションが追加される場合，そのトランザクションの偏り次第では各段階で頻出と判定されるアイテムセットが変化することとなるため，サポート値の再計算が求められる．FUP アルゴリズムではデータベース更新前の頻出アイテムセットの判定を再利用し，候補アイテムセットを減少させることで，再計算コストを削減する．

3.2　マスタ・ワーカ型分散処理

本研究ではシステムがクラウドコンピューティング環境で利用されることを想定し，マスタ・ワーカ型分散処理を実装した．サーバとクライアントの間はインターネットを中継するため，通信が遅くなることが想定される．そのためクライアントとの通信を担当するマスタを作成することで，クライアントとの通信を極力削減した．またタスクの分割に関しては，クエリごとにマスタに送信されるアイテムセットを分割する手法を用いる．サーバサイドで完全準同型暗号の暗号文同士の演算が行われている箇所は，(a) 相関性を調べたい複数のアイテムごとのパッキングベクトル同士の掛け算，(b) パッキングベクトル内の頻出度の合計値算出，(c) パッキングベクトルごとの合計値をさらにアイテムセットごとで足し合わせた合計値算出の 3 ヶ所である．アイテムセットごとの分割では，それら全ての計算が分割されるため効率的である．

3.3　概要

本研究では，以下の完全準同型暗号を用いた秘匿 FUP 計算のマスタ・ワーカ型分散処理システムを提案する．本システムの概観を図 1 に示す．

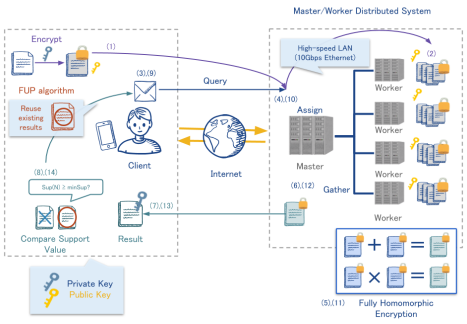


図 1: 提案手法概観

(1) クライアントはデータを暗号化し，公開鍵と共にマスタへ送信．
(2) マスタは受け取ったデータと公開鍵を各ワーカへ送信．
(3) クライアントは更新前データベースに関する Apriori 計算結果から，更新前に頻出ではないアイテムセットのサポート値計算を更新分データベースに対して行う依頼をマスタに送信．
(4) 更新分データベースに関して，サポート値計算をワーカに割り振る．
(5) 各ワーカは完全準同型暗号を用いてサポート値計算を暗号化されたデータに対して行う．

(6) マスタは各ワーカから結果を収集し，クライアントに送信.
(7) クライアントはデータを受け取り，秘密鍵でデータを復号.
(8) クライアントは結果と閾値の比較を行う.
(9) クライアントは手順 (8) で更新前は頻出ではなかったが，更新分において新規に頻出と判定されたアイテムセットと手順 (3) で計算依頼された更新前の時点で既に頻出であるアイテムセットに関して，サポート値計算を依頼.
(10) マスタは各ワーカにタスクを割り振る.
(11) 各ワーカは完全準同型暗号計算を用いてサポート値を計算.
(12) マスタは結果を収集し，クライアントに送信.
(13) クライアントはデータを受け取り，秘密鍵でデータを復号.
(14) クライアントは結果と閾値の比較を行い，データベース更新後の頻出アイテムセットを得る.
(15) (3)〜(14) の手順を繰り返し，最終的に閾値を超えるアイテムセットが無くなるか，頻出アイテムセットが最長になるまで繰り返す.

4 実験

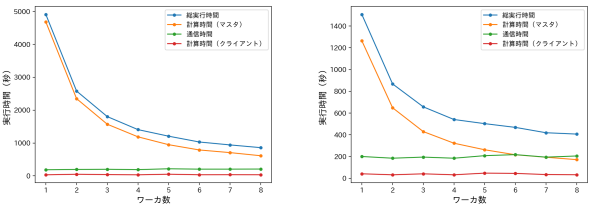
4.1 実験環境

C++ で実装し，完全準同型暗号計算には HElib [5] を，分散化における各マシンの制御のために Open MPI [6] を用いた．実装したマスタ・ワーカ側のプログラムをお茶の水女子大学内に構築されたサーバルーム内の同一性能の 4 台の Linux マシンに設置した．またクライアント側のプログラムを Amazon Web Services の EC2 m4.4xlarge インスタンス 1 台に設置した．マスタ・ワーカ側のマシン 4 台それぞれに対し，最大 2 スロット分のワーカの演算を行わせ，そのうち 1 台にマスタの機能を持たせる．最大で 8 スロット分のワーカを稼働させてワーカ数ごとの実行時間を比較する実験を行った．

4.2 実験結果

本手法実装システムにて，アイテム数 50，トランザクション数 19,800 のデータセットを用いて秘匿 Apriori 計算を，更新前トランザクション数 19,470，更新後トランザクション数 19,800 のデータセットを用いて秘匿 FUP 計算を行う実験を行った．ワーカ数ごとのそれぞれの実行時間を図 2 に，ワーカ数ごとの秘匿 Apriori 計算と秘匿 FUP 計算の計算時間の比較とそれぞれの実行時間・計算時間の両対数グラフを図 3 に，分散処理化の評価として高速化率を 逐次実行時間 (秒)/並列実行時間 (秒) で算出し，式 (3) に基づいた Amdahl の法則による並列度と高速化率の関係 [7] と共に図 4 に示す．

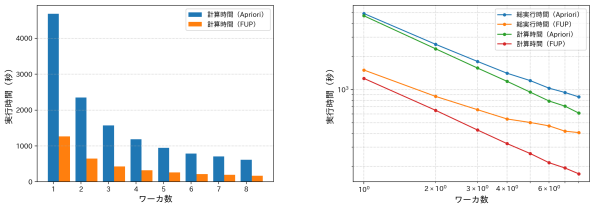
$$\text{高速化率} \leq \frac{1}{(1 - \text{並列実行時間の割合}) + \frac{\text{並列実行時間の割合}}{\text{ワーカ数}}} \tag{3}$$



(a) Apriori (b) FUP

図 2: 秘匿 Apriori 計算と秘匿 FUP 計算の実行時間 (秒)

図 2 より，秘匿 Apriori 計算と秘匿 FUP 計算どちらもワーカ数が増加するにつれて総実行時間は減少し，特にマスタ側の計算時間の分散化効果が顕著であることが示された．また通信時間とクライアント上の計算時間は，ワーカ数の増加に対してほとんど変化しない．図 3(a) より，FUP アルゴリズムによる再計算は Apriori アルゴリズムによって計算を最初からやり直す再計算よりも，計算時間が約 3〜4 倍高速化される結果となった．また図 3(b) は秘匿 Apriori の再計算と秘匿 FUP 計算それぞれの総実行時間を両対数グラフの形式で示している．それぞれのグラフはほぼ直線であるため，それぞれの実行時間とワーカ数の関係は冪乗則に従う．図 4 より，マスタ側の完全準同型暗号の暗号文同士の計算時間はほぼ 99% 並列時に近い高速化率で



(a) 計算時間比較 (b) 両対数グラフによる比較

図 3: 秘匿 Apriori 計算と秘匿 FUP 計算の比較

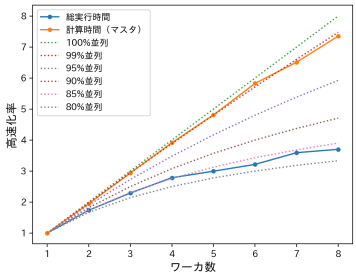


図 4: 秘匿 FUP 計算の高速化率と Amdahl の法則による並列度に対する高速化率

分割されていることが示された．ただし，通信時間，クライアント側での暗号化や復号に伴う計算時間，ファイル入出力等に必要な時間も含めた総実行時間では，85% 並列時の高速化率に近い．したがって，今回の実験環境下において本システムは式 (3) より，ワーカ数を最大限に増やす場合，最大で約 6.7 倍の高速化率が期待できる．

5 まとめと今後の課題

完全準同型暗号を用いた秘匿データマイニング計算の高速化を目指し，データベース更新時の Apriori アルゴリズムの高速化を行う FUP アルゴリズムを用いた秘匿データマイニングシステムを実装した．またマスタ・ワーカ型分散処理を適用し，システムの高速化を試みた．その結果，データベース更新時において FUP アルゴリズムによる再計算は Apriori アルゴリズムによる再計算と比較して，約 3〜4 倍の計算時間の短縮が可能になった．また分散処理化によって，マスタ側の計算時間が分散台数に応じて減少した．

本システムの応用として，人間の動作ログデータに関するデータマイニングへの適用を提案している．今後は更に動作ログデータに対する秘匿委託データマイニングに最適な手法を検討していきたい．

参考文献

[1] Hiroki Imabayashi, Yu Ishimaki, Akira Umayabara, Hiroki Sato, and Hayato Yamana. Secure frequent pattern mining by fully homomorphic encryption with ciphertext packing. In *International Workshop on Data Privacy Management*, pp. 181–195. Springer, 2016.
[2] D. W. Cheung, Jiawei Han, V. T. Ng, and C. Y. Wong. Maintenance of discovered association rules in large databases: an incremental updating technique. In *Proceedings of the Twelfth International Conference on Data Engineering*, pp. 106–114, Feb 1996.
[3] Ronald L Rivest, Len Adleman, and Michael L Dertouzos. On data banks and privacy homomorphisms. *Foundations of secure computation*, Vol. 4, No. 11, pp. 169–180, 1978.
[4] Craig Gentry, et al. Fully homomorphic encryption using ideal lattices. In *STOC*, Vol. 9, pp. 169–178, 2009.
[5] Shoup V. and Halevi S. HElib. `http://shaih.github.io/HElib/index.html`. Accessed: 2019-1.
[6] Open MPI. `https://www.open-mpi.org/`. Accessed: 2019-1.
[7] Clay Breshears. *The Art of Concurrency: A Thread Monkey's Guide to Writing Parallel Applications*. O'Reilly Media, Inc., 2009.