論文 / 著書情報
Article / Book Information

論題(和文)	MapReduceによる大規模なRDFデータ復号化手法の評価
Title(English)	
著者(和文)	Vu Tuan Dat, 横田 治夫
Authors(English)	Tuan Dat Vu, Haruo Yokota
出典(和文)	第7回データ工学と情報マネジメントに関するフォーラム論文集, , ,
Citation(English)	, , ,
発行日 / Pub. date	2015, 3

MapReduceによる大規模なRDFデータ復号化手法の評価

VuTuanDat† 横田 治夫†

† 東京工業大学 大学院情報理工学研究科 計算工学専攻
〒 152-8552 東京都目黒区大岡山 2-12-1
E-mail: dat@de.cs.titech.ac.jp, yokota@cs.titech.ac.jp

あらまし 近年，様々な情報が RDF で記述されることが増えている．RDF データが急激に増加する中，秘匿性のあるデータを保護するため，暗号化が求められる．しかし，RDF データを解析する場合，暗号化したままで処理できることは限られており，復号化してから処理を行わなければならない場合がある．大量データに対する復号化には時間がかかり，並列処理による高速化が望まれる．本研究は MapReduce を用いた大量な RDF データ復号化の並列化アルゴリズムを提案する．RDF は Subject，Predicate，Object の三つ組によって構成される．提案手法では同じ Subject，Predicate，Object をグループ化して，復号化をすることで実行時間の短縮を試みる．MapReduce を実行するとき 1 Job(MapReduce の実行単位) で全要素を復号化すると各トリプル要素が異なるタスクに割り当てる場合があり，トリプル再構築する必要がある．本研究は，再構築する必要がなく，Subject，Predicate，Object それぞれの要素を各 MapReduce Job で復号化をしてトリプルに代入する手法と Subject，Predicate，Object の三つを 1 Job で全要素を復号化してから再構築を行う手法を提案する．さらに，Hadoop 上でプロトタイプを実装し，提案手法の処理効率を評価する．

キーワード RDF，RDF 復号化，MapReduce

1. はじめに

Resource Discription Framework(RDF) は，ウェブ上にある「リソース」を記述するための統一された枠組みであり，World Wide Web Consortium(W3C) により 1999 年 2 月に規格化されている．RDF は意味を損なわずにアプリケーション間で情報を交換することができる．共通の枠組みであるため，アプリケーションの設計者は共通の RDF パーサや処理ツールを有効利用できる．そのため，近年様々な情報が RDF で記述される．

RDF データが急激に増加する中，個人情報のような秘匿性が求められるデータも増えている．秘匿性を保証するためには，情報を暗号化するアプローチや，匿名化を行いノイズ情報を加えるアプローチなどが考えられる．統計情報を集約する場合等にも，匿名化のアプローチは有用であるが，多様な集約処理を前提にしたノイズ情報への制約や，個々の情報が扱いたい場合等の要求を満たすためには，暗号化が有用である．

例えば，災害対応等を考えたとき，そのような秘匿性が求められるデータの中から集約処理を行う必要がある．想定される集約処理としては，単に避難所毎の人数を数えるようなレベルのものから，ある年齢以上の高齢者の人数や，平均年齢，薬を処方するために血圧がある値より高い人の人数，必要な食糧を算出するために成人男性を 1 とした場合に女性は 0.8，高齢者は 0.6，12 歳以下の子供は 0.5 といった比率で算出する値等，様々な場合が考えられる．また，行政機関や医療従事者などが，個々の避難者の情報を利用することも想定すると，ノイズ情報付加ではなく，暗号化が有用となる．本研究では，暗号化されたデータに対して，様々な集約処理を行うことを対象とする．

暗号化を前提とした場合，単純に平均年齢を求めるような場合には準同型暗号 (2. 2. 2 節に参照) を使うことで，復号化せずに算出できる場合がある．しかし，上述したような複雑な条件の場合には全体を一旦復号化して処理することも必要になる．複雑な条件ばかりでなく，単純な平均年齢を求めるような場合も想定して準同型暗号を使う場合には特に全体の復号化に時間がかかる．利用状況として，広域災害等を想定した場合には特に対象データが増えることが予想されることから，膨大なデータに対して処理時間がかかる．そのため，並列化による高速化が求められる．本研究では，復号化に時間を要する暗号を用いた場合に，全体を復号する場合の処理の高速化を対象として，並列処理による高速化の方法を検討する．

並列分散プログラミング環境として，MPI(Message Passing Interface) ライブラリ [3] がよく使われているが，プログラムが大規模になるに従い複雑化し，エラー処理や障害処理などを記述する実装コストが高いことが知られている．一方で，並列プログラミングに慣れていない開発者でも扱いやすい分散処理フレームワーク MapReduce [2] が注目されている．MapReduce では，map 関数と reduce 関数を定義するだけで，データをグループ化することによって大規模なデータの並列処理が可能となるシンプルなフレームワークである．また，計算機の故障や追加にも対応できる．特に，本研究のように，同じようなデータの出現が多いことを想定し，同じデータをまとめてから処理を行うには MapReduce が向いている．

このような背景から，本研究では，MapReduce を用い，暗号化されたデータに対し，平均年齢のような準同型性が利用できる単純な演算の並列化だけでなく，複雑な条件で全体を一旦

復号化して処理する場合の復号化の並列化を行う目的とする．

全体を一旦復号化する必要がある場合の復号化について，前提として復号化に時間がかかることから，本研究では，同じ暗号文をグループ化することによって効率化を行う方法をいくつか提案する．RDF が Subject，Predicate，Object の三つ組みによって構成され，MapReduce を実行するとき 1 Job(MapReduce の実行単位) で全部復号化すると各トリプル要素が異なるタスクに割り当てる場合があり，トリプル再構築する必要がある．本研究は，再構築する必要がなく，Subject，Predicate，Object それぞれの要素を各 MapReduce ジョブで復号化をしてトリプルに代入する手法（Element Job 法）と Subject，Predicate，Object の三つを 1 Job で復号化してから再構築を行う手法（Triple Reconstruction 法）を提案する．

また，準同型性が利用できる単純な演算の場合，例えば災害対応等を考えたとき，避難者の平均年齢を求める場合，人の情報 (避難者かボランティアか等，年齢の情報) をグループ化し，避難者かどうかを判定してから準同型性を利用して平均年齢を計算する．また，準同型性を利用しない通常の手法を比較して評価を行う．

本稿の構成は以下の通りである．まず，2. 節で前提知識として RDF，暗号の種類と MapReduce を説明し，3. 節で MapReduce を用いた並列化手法を提案する．次に 4. 節で実験を通じて，提案手法の評価を行う．5. 節で関連研究について述べ，最後に 6. 節でまとめを述べる．

2. 前 提 知 識

2.1 RDF

Resource Description Framework(RDF) は，資源に対する構造的なメタデータを記述するための基礎的な枠組である．RDF のメタデータのモデルでは，主語 (subject:S)，述語 (predicate:P)，目的語 (object:O) の 3 つの要素でリソースに関する関係情報を表現し，これをトリプル（triple）と呼ぶ．主語は記述対象のリソースである．述語は主語の特徴や主語と目的語との関係を示す．目的語は主語との関係のある物や述語の値である．トリプルの集合は subject と object を頂点とし，述語が有向辺に対応したラベル付き有向グラフの構造を表現する．RDF データが構成するラベル付き有向グラフを RDF グラフと呼ぶ．図 1 は単純な RDF グラフの例である．RDF のモデルを

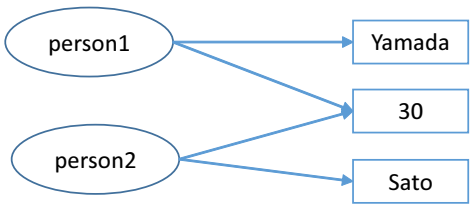

図 1　RDF グラフの例

記述する方法は，グラフ，XML 構文などがあるが，扱いやすく，並列フレームワークで扱いやすい記法として N-Triples がよく使われている．N-Triples は 1 行ごとに 1 つのトリプルを，主語・述語・目的語の順にスペースで区切って表記するフォーマットである．本研究で使用する RDF データのフォーマットは n-triples である．図 1 の RDF グラフは N-Triples で表示すると以下のようになる．

```
Person1 name Yamada .
Person1 age 30 .
Person2 name Sato .
Person2 age 30 .
```

2.2 暗　　号

2.2.1 確率的暗号と確定的暗号

暗号化するに大きく分けると 2 つのタイプがあり，確率的暗号 (probabilistic encryption) と確定的暗号 (deterministic encryption) である．確率的暗号は，平文 m に対する暗号文が複数存在し，同じ平文でも異なる暗号文に暗号化される．このため、確率的暗号では，暗号文同士の比較による同等性チェックすることができない．逆に，同じ平文は同じ暗号になる暗号は確定的暗号と呼ぶ．確定的暗号は確率的暗号より若干弱いが，暗号化したままでもとの平文の同等性チェックすることができ，同等フィルター，GROUP BY，COUNT，DISTINCT などの SELECT 演算が可能となる [7]．本研究は確定的暗号で暗号化されたデータを対象する．

2.2.2 準同型暗号

暗号文同士を演算することで，異なる暗号構えの平文に対応する暗号文が計算できる暗号方式は準同型暗号と呼ぶ．平文 m_1，m_2 を暗号化した二つの暗号文 c_1，$c_2(c_1 = Enc(m_1)$, $c_2 = Enc(m_2))$ とする．Enc が演算子 $\otimes$ と $\odot$ に対して準同型性を持つとは，$c_1 \otimes c_2 = Enc(m_1 \odot m_2)$ をみたすことをいう．

例えば，RSA 暗号 [5] は暗号化したまま平文の乗算を行える準同型性をもつ．RSA 暗号では $c_1 \times c_2 = Enc(m_1 \times m_2)$ となる．また，Paillier 暗号 [6] は暗号化したまま平文の加算を行える準同型性をもつ．Paillier 暗号では $c_1 \times c_2 = Enc(m_1 + m_2)$ となる．本研究は，準同型性を利用し，平均値を求めることを提案している。

2.3 Hadoop と MapReduce

Hadoop [1] とは大規模なデータを複数のコンピューターで分散処理できるようにしたオープンソースソフトウェアである．Hadoop は分散ファイルシステム Hadoop Distributed File System（HDFS），分散処理フレームワーク MapReduce という大きく二つのコンポーネントで構成される．

MapReduce のデータは非常にシンプルで *key* と *value* の組み合わせで管理する．MapReduce の処理は map 処理，shuffle & sort 処理と reduce 処理の 3 段階に分けて実行される仕組みになっている．このうち，ユーザーが外部から処理内容を与えるのが map と reduce である．

それぞれのフェーズでは，「$\langle key, value \rangle$ ペア」または「$\langle key, value$ のリスト$\rangle$ ペア」を入出力として扱っている．MapReduce 処理の流れは図 2 を示し，以降では，各フェーズの詳細を述べる．Map フェーズでは，入力データは細かいブロックに分

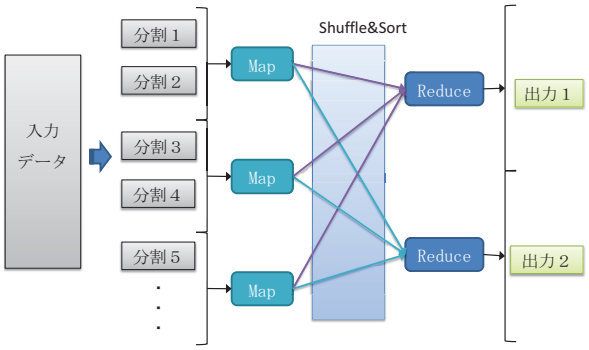

図 2　MapReduce 処理の流れ

割され，多くのマシンに分散される．一つのブロックは一つの map 処理を実行する単位である map タスクへ割り当てられる．map タスクはブロックを読み込み，ユーザーが定義した map 関数を実行する．map 関数は入力された 1 個の〈*key*, *value*〉ペア を内部で処理して，新たな〈*key*, *value*〉ペア として出力する．出力する〈*key*, *value*〉ペア は 2 個以上になることもあるし，1 個もないこともある．

Shuffle & Sort フェーズは，map フェーズが出力した〈*key*, *value*〉ペア をマージしてソートし，同じ key を持つ〈*key*, *value*〉ペア 同士を集約する．key ごとに value をリストアップして，reduce タスクに振り分ける．この処理は Hadoop 側が面倒を見てくれるので，ユーザーは何もする必要がない．

Reduce フェーズは〈*key*, *value* のリスト〉ペア を受け取って，内部で処理した結果を新たな〈*key*, *value*〉ペア として出力する．

3. 提 案 手 法

本研究では，個人情報を記述する枠組みとして RDF を想定する．RDF は，Subject，Predicate，Object の三つ組みで表現され，それぞれに暗号化することができ，本研究がやろうとしていることに向いている．

並列化のアプローチとしては，MapReduce の枠組みを利用することを想定する．MapReduce では，今回のように大量のデータに対して条件を満たすデータをまとめて同様の処理を行うような場合に向いている．

本研究では，準同型性を利用できる単純な集約処理の場合と，複雑な集合処理で全体を一旦復号化して処理する場合を想定する．準同型性が利用できる場合，MapReduce を用いた準同型性を利用した・しない手法を提案し，その比較を行う．全体を一旦復号化して処理する場合 MapReduce を用いた復号化手法を 2 種類提案し，その比較を行う．

以下，f は暗号関数として，各手法を詳細に説明する．

3.1 全体を一旦復号化する必要がある場合の復号化手法

一般に考えると RDF データに対して，RDF トリプルが一個づつ復号化する手法 (手法 0) があるが，復号化時間が長く，また RDF データの中に同じトリプル要素の出現が多いことを想定し，同じ要素をまとめてから復号化を行う手法を提案する．RDF は Subject，Predicate，Object の三つ組みによって構成され，MapReduce を実行するとき 1 job(MapReduce の実行単位) で全要素を復号化すると各トリプル要素が異なるタスクに割り当てる場合があり，トリプル再構築する必要がある．本研究は，再構築する必要がなく，Subject，Predicate，Object それぞれの要素を各 MapReduce ジョブで復号化をしてトリプルに代入する Element Job 法（手法 1）と Subject，Predicate，Object の三つを 1 job で復号化してから再構築を行う Triple Reconstruction 法（手法 2）を提案する．

Element Job 法は MapReduce の 3 job で行い，各 job では各トリプル要素を復号化し，復号化した要素をトリプルに代入する．この手法は RDF データの一部，たとえば Object だけ，を復号化する場合に適する．Triple Reconstruction 法は MapReduce の 2 job で行い，1 job 目はトリプルの Subject，Predicate，Object の同じ要素をグループ化し，復号化を行い，2 job 目はトリプルの再構築を行う．Triple Reconstruction 法は Element Job 法と違って，ある Subject とある Object が同じ場合もまとめられる．あるトリプルの Subject とほかのトリプルの Object が同じ場合が多いと予想しており，この手法は RDF データの全要素を復号化する場合に適する．

3.1.1 Element Job 法

MapReduce を実行するとき 1 job で全要素を復号化すると各トリプル要素が異なるタスクに割り当てる場合があり，トリプル再構築する必要がある．トリプルの Subject，Predicate，Object をそれぞれの要素を復号化してトリプルに代入すると再構築する必要がない．そこで，Element Job 法はそれぞれの要素を各 MapReduce ジョブで復号化を行う．各段で，暗号化されたトリプル要素を key，そのトリプルを value とし，MapReduce により key をグループ化され，key を復号化し，トリプルに代入する．

Element Job 法における MapReduce 1 job 目はアルゴリズム 1 のように行われる．まず，Map フェーズでは，トリプルを解析し（行 2），暗号化された Subject を key，トリプルを value となる〈key，value〉ペアを出力する（行 4）．Shuffle & Sort フェーズは暗号化された Subject をグループ化し，暗号化された Subject（encSubject）とトリプルのリスト（tripleList）を Reduce に振り分ける．次に，Reduce フェーズでは暗号化された Subject を復号化（行 7）し，トリプルのリストの各トリプルに対し，復号化した Subject を代入し，トリプルを出力する（行 8-11）．1 段目では，Subject が復号化されたトリプルが得られる．同様に 2 job 目と 3 job 目で Predicate，Object を復号化してトリプルに代入する．

図 3 は実例である．図は青色を key，紫色を value で表現する．

3.1.2 Triple Reconstruction 法

RDF データの中にあるトリプルの Subject とほかのトリプルの Object が同じ場合もあり，MapReduce の 1 job で全要素を復号化するとそのような Subject と Object がまとめられる．そこで，Triple Reconstruction 法の MapReduce 1 job 目は暗号化されたトリプル要素をグループ化することによりトリプル

アルゴリズム 1 Element Job 法 MapReduce 1 job 目 Subject の復号化

入力: n-triples 形のトリプル，トリプルの Id
出力: 暗号化された Subject，トリプル

1: **function** MAP($tripleString, tripleId$)
2: $triple \leftarrow parse(tripleString)$ ▷ トリプルを解析
3: $subject \leftarrow triple.getSubject()$
4: $emit(subject, triple)$ ▷ 暗号化された Subject，トリプルを key-value として出力
5: **end function**

入力: 暗号化された Subject，トリプルのリスト
出力: 暗号化された Subject，トリプル

6: **function** REDUCE($encSubject, tripleList$)
7: $subject \leftarrow decrypt(encSubject)$ ▷ 暗号化された Subject を復号化
8: **for all** $triple \in tripleList$ **do**
9: $subject$ を $triple$ に代入▷ 復号化した Subject をトリプルに代入
10: $emit(subject, triple)$
11: **end for**
12: **end function**

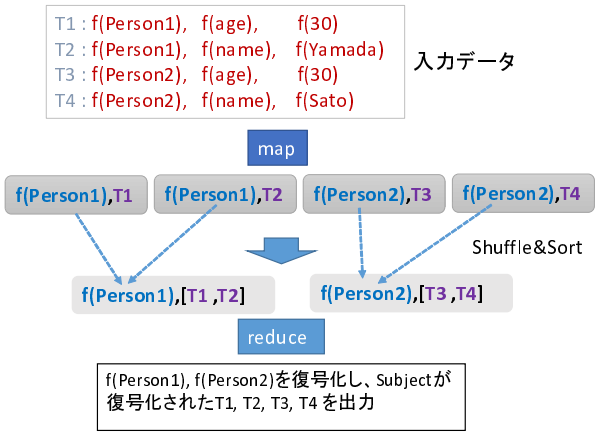

図 3 Element Job 法 MapReduce 1 job 目 Subject の復号化

要素を復号化する．2 job 目はトリプル id をグループ化することによりトリプルの再構築を行う．

Triple Reconstruction 法における MapReduce の手順はアルゴリズム 2 とアルゴリズム 3 のように行われる．まず，MapReduce 1 job 目（アルゴリズム 2）の Map フェーズでは，トリプルを解析し（行 2），各トリプルの要素に対して，暗号化されたトリプルの要素を key，トリプル要素の情報（termInfo= トリプル Id，その要素の位置 (S か P か O か)）を value となる〈key，value〉ペアを出力する（行 3-11）．Shuffle & Sort フェーズは暗号化されたトリプル要素をグループ化し，暗号化された要素（encTerm）と要素情報のリスト（termInfoList）を Reduce に振り分ける．次に，Reduce フェーズでは暗号化されたトリプル要素（term）を復号化し（行 14），復号化した要素を key，トリプル要素情報のリストを value となる〈key，value〉ペアを出力する（行 15）．

MapReduce 2 job 目（アルゴリズム 3）の Map フェーズでは，トリプル要素のリストの各トリプル要素（termInfo）に対して，termInfo からトリプル要素の位置とトリプルの id を取得する．トリプルの id を key，termObject（トリプル要素，その要素の位置）を value となる〈key，value〉ペアを出力する（行 6）．Shuffle & Sort フェーズはトリプルの id をグループ化し，トリプルの id（tripleId）と termObject のリスト（termObjectList）を Reduce に振り分ける．次に，Reduce フェーズでは termObject のリストの各 termObject に対して，termObject から要素の内容と位置を取得し，トリプル (triple) に代入する（行 10-18）．最後に，トリプルを出力する．

図 4 と図 5 は実例である．

アルゴリズム 2 Triple Reconstruction 法 MapReduce 1 段目 全要素の復号化

入力: n-triples 形のトリプル，トリプルの Id
出力: 暗号化されたトリプル要素，termInfo(トリプル Id，その要素の位置 (S か P か O か))

1: **function** MAP($tripleString$, $tripleId$)
2: $triple \leftarrow parse(tripleString)$ ▷ トリプルを解析
3: $subject \leftarrow triple.getSubject()$
4: $termInfo \leftarrow (tripleId, \text{“S”})$
5: $emit(subject, termInfo)$
6: $predicate \leftarrow triple.getPredicate()$
7: $termInfo \leftarrow (tripleId, \text{“P”})$
8: $emit(predicate, termInfo)$
9: $object \leftarrow triple.getObject()$
10: $termInfo \leftarrow (tripleId, \text{“O”})$
11: $emit(object, termInfo)$
12: **end function**

入力: 暗号化されたトリプルの要素，(トリプル id と要素の位置) のリスト
出力: 復号化されたトリプルの要素，(トリプル id と要素の位置) のリスト

13: **function** REDUCE($encTerm$, $termInfoList$)
14: $term \leftarrow decrypt(encTerm)$ ▷ 暗号化されたトリプル要素を復号化
15: $emit(term, termInfoList)$
16: **end function**

3.2 準同型暗号データ間の演算を用いる場合の手法

避難所の情報から，避難者 (evacuee) の平均年齢を計算する集約を例として説明する．集約のクエリーが以下のようになる．

```
SELECT AVG(?X) WHERE {
    ?Person  rdf:type  :evacuee .
    ?Person  :age      ?X
}
```

MapReduce により，人の情報 (避難者かボランティアか等，年齢の情報) をグループ化し，避難者かどうかを判定し，避難者の年齢の暗号が得られる．準同型性が利用できない場合，復号化してから平文を利用して平均を算出する手法があるが，準同

アルゴリズム 3 Triple Reconstruction 法 MapReduce 2 job 目 トリプルの再構築

入力: 復号化されたトリプルの要素，(トリプル id と要素の位置) のリスト
出力: トリプルの id，復号化されたトリプルの要素とその要素の位置

```
1: function MAP(term, termInfoList)
2:     for all termInfo ∈ termInfoList do
3:         termPosition ← termInfo.getPosition()   ▷ トリプル要素の位置 (S か P か O か) を取得
4:         tripleId ← termInfo.getTripleId()
5:         termObject ← (term, termPosition)
6:         emit(tripleId, termObject)
7:     end for
8: end function
```

入力: トリプルの id，(復号化されたトリプルの要素とその要素の位置) のリスト
出力: トリプル

```
9: function REDUCE(tripleId, termObjectList)
10:    for all termObject ∈ termObjectList do
11:        if termObject.getPosition =“S” then
12:            triple.subject ← termObject.getTerm()
13:        else if termObject.getPosition =“P” then
14:            triple.predicate ← termObject.getTerm()
15:        else if termObject.getPosition =“O” then
16:            triple.object ← termObject.getTerm()
17:        end if
18:    end for
19:    emit(triple)
20: end function
```

型暗号が利用できる場合，加法準同型性を利用して暗号化したままの暗号文で年齢の合計が求められ，人数で割ることで平均年齢が計算できる．

3.2.1 準同型性を利用した場合

集約処理は 2 ステップに分けられる．ステップ 1 は MapReduce で結合を実行し，暗号化された年齢が得られる．ステップ 2 はステップ 1 から得られたデータを集計し，平均年齢を求める．ステップ 1 における MapReduce の手順はアルゴリズム 4 のように行われる．ステップ 2 はアルゴリズム 5 のようになる．

まず，ステップ 1 の MapReduce（アルゴリズム 4）の Map フェーズでは，そのトリプルは年齢の情報（Predicate は “:age” の暗号と等しい）（行 3）があれば，トリプルの subject（人），object（年齢）を出力する．そのトリプルは避難者を指すトリプル（Predicate は “rdf:type” の暗号と等しく，かつ Object は “:evacuee” の暗号と等しい）であれば，トリプルの subject（人），object（避難者）を出力する．

Shuffle & Sort フェーズは暗号化された Subject をグループ化し，暗号化された Subject（encSubject）と暗号化された Object のリスト（encObjectList）を Reduce に振り分ける．このフェーズは人ごとに年齢の情報と避難者かどうかの情報をグループ化する．

次に，Reduce フェーズでは encObjectList のサイズでまとめた情報が避難者の情報かどうかを判定する（行 18:サイズが 2 以上なら避難者である）．避難者の場合，年齢の情報を取り（行 20），避難者数を加算され（行 21），加法準同型性を利用して合計年齢の暗号を計算する．各 Reduce タスクが終わる前に求めた避難者数と合計年齢の暗号を出力する．

ステップ 2（アルゴリズム 5）では，各 Reduce タスクから，全体の避難者数を計算し，加法準同型性を利用して全体の合計年齢の暗号を計算（行 6）して復号化する（行 8）．全体の避難者数と全体の合計年齢から平均年齢を計算（行 9）して結果を出力する．

図 6 は MapReduce 実行例である．

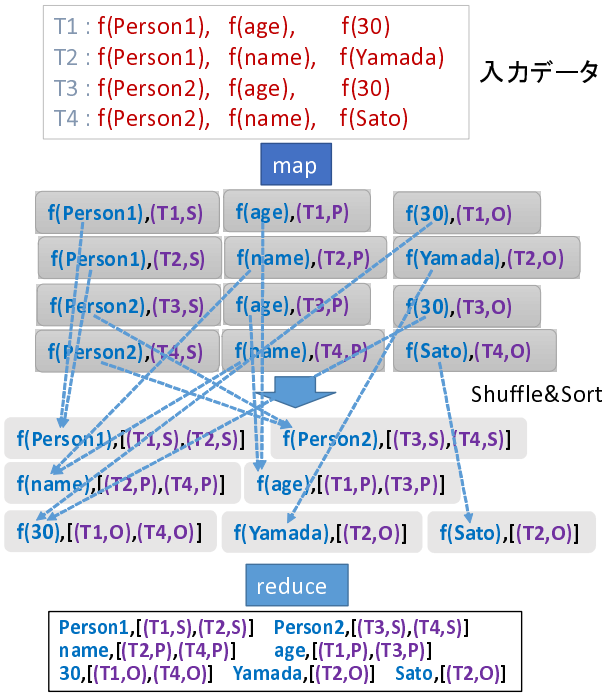

図 4 Triple Reconstruction 法 MapReduce 1 job 目 全要素の復号化

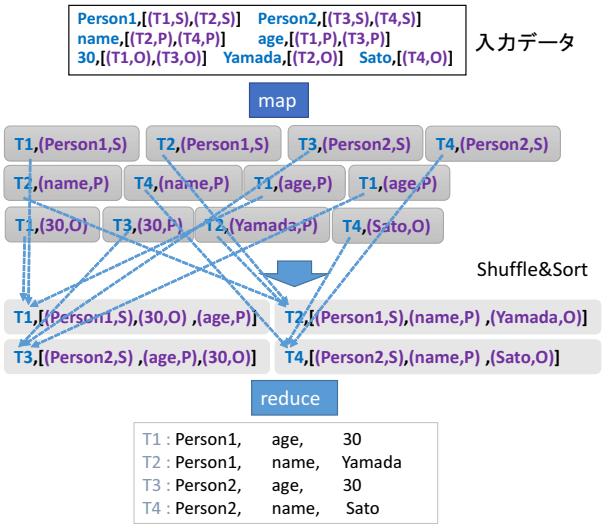

図 5 Triple Reconstruction 法 MapReduce 2 job 目 トリプル再構築

アルゴリズム 4 準同型性を利用した場合の手法のステップ 1 MapReduce job

入力: n-triples 形のトリプル，トリプルの Id
出力: 暗号化された Subject，暗号化された年齢，または暗号化された避難者の URI(:evacuee)

```
1: function MAP(tripleString, tripleId)
2:     triple ← parse(tripleString)                ▷ トリプルを解析
3:     if triple.getPredicate() = f(": age") then
4:         subject ← triple.getSubject()
5:         object ← triple.getObject()
6:         emit(subject, object)
7:     else if triple.getPredicate() = f("rdf : type") then
8:         if triple.getObject() = f(": evacuee") then
9:             subject ← triple.getSubject()
10:            object ← triple.getObject()
11:            emit(subject, object)
12:        end if
13:    end if
14: end function
```

入力: 暗号化された Subject，暗号化された Object のリスト
出力: 避難者数，暗号化された年齢の合計

```
15: numEvacuee ← 0
16: encSumAge ← 1
17: function REDUCE(encSubject, encObjectList)
18:     if encObjectList.size() ≧ 2 then  ▷ 避難者かどうかを判定
19:         for all encObject ∈ objectList do
20:             if encObject ≠ f(": evacuee") then ▷ Object は暗号化された年齢かどうかを判定する
21:                 numEvacuee ← numEvacuee + 1
22:                 encSumAge ← encSumAge × object ▷ 加法準同型性を利用し，年齢の合計の暗号を計算する．この場合は Paillier 暗号の場合の計算である．
23:             end if
24:         end for
25:     end if
26: end function
27: function REDUCECLEANUP ▷ reduce タスクの全部 reduce 関数の実行が終わった後，Reduce タスクが終わる前に呼ばれる．
28:     emit(numEvacuee, encSumAge)
29: end function
```

3.2.2 準同型性を利用しない場合

準同型性を利用した場合と同じく，個人情報をもとめて，避難者の数を数え，避難者の年齢の暗号を取るが年齢の暗号を一旦復号化してから平均年齢を計算する．

4. 実　　験

4.1 実験の環境

マシン台数 1 台〜4 台 (表 1) を利用して実験を行った．

4.1.1 暗号アルゴリズム

本実験は RSA-2048bit と Paillier-2048bit を利用して，実験が行った．但し，Paillier 暗号は確定的暗号ではないので乱数を定数にして暗号化した．

アルゴリズム 5 準同型性を利用した場合の手法のステップ 2 平均年齢の計算

入力: 各 reduce タスクからの避難者数，暗号化された年齢の合計
出力: 平均年齢

```
1: function GETAVERAGEAGE(reduceTaskOutputList)
2:     totalEvacuee ← 0
3:     encTotalAge ← 0
4:     for all out ∈ reduceTaskOutputList do
5:         totalEvacuee ← totalEvacuee+out.getNumEvacuee()
6:         encTotalAge ← encTotalAge × out.getSumAge()
7:     end for
8:     totalAge ← decrypt(encTotalAge)
9:     averageAge ← totalAge ÷ totalEvacuee
10:    return averageAge
11: end function
```

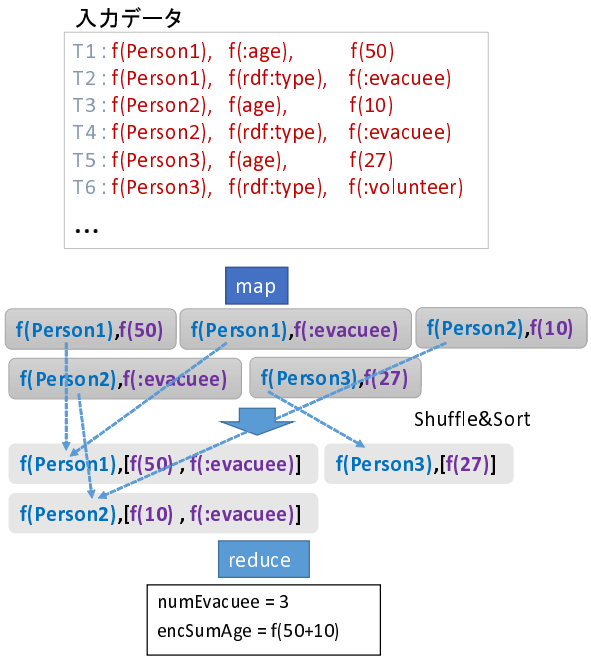

図 6　準同型暗号データ間の演算を用いる場合の手法　ステップ 1

表 1　実 験 機 材

CPU	Intel Xeon E5620OS
Core 数	8(16 threads)
CoreFrequency	2.40GHz
メモリ	24GB
OS	Ubuntu11.10
Java	1.6
Hadoop	Version 1.0.4
Ethernet	1GB/s

4.1.2 データセット

本実験で使用するデータセットは RDF ベンチマークの Lehigh University Benchmark(LUBM [11]) である．仮想的な大学組織の大学数をパラメータとして生成した RDF データである．

全体を一旦復号化する必要がある場合の復号化手法の実験では 50 大学のデータ (LUBM50) を生成した．また，準同型暗号

データ間の演算を用いる場合の手法の実験では学部生の平均点を計算するため各学生に平均点 (ub:averageScore) を追加し，50 大学のデータ (LUBM50EX) を生成した．生成したデータを Jena [4] により n-triples の形に変換し，RSA，Paillier アルゴリズムで暗号化したデータを入力データとする．データの詳細は表 2 で示す．

全体を一旦復号化する必要がある場合の復号化手法の実験では LUBM50 のデータセットを利用し，RSA，Paillier アルゴリズムでそれぞれの暗号化したデータに対して，Element Job 法と Triple Reconstruction 法を比較する．準同型暗号データ間の演算を用いる場合の手法の実験は，LUBM50EX のデータセットを利用し，加算準同型性をもっていない RSA 暗号で通常の手法と加法準同型性をもつ Paillier 暗号で準同型を利用した手法を比較する．

表 2　実験データ

データセット	LUBM50	LUBM50EX
トリプル数	6,865,225	7,425,365
人数	549,818	618,235
暗号しないサイズ	1.09 GB	1.16 GB
RSA 暗号サイズ	6.32 GB	6.71 GB
Paillier 暗号サイズ	13.1 GB	14.2 GB

4.2　実験の結果

4.2.1　全体を一旦復号化する必要がある場合の復号化手法の実験結果

50 大学のデータ (LUBM50)(表 2) を RSA，Paillier アルゴリズムで暗号化されたデータを利用し，マシン台数を増やした場合の実行時間をそれぞれ図 7 と図 8 に示す．手法 0 は同じ要素をまとめなく，各トリプルの一個づつを復号化した場合の実行時間である．

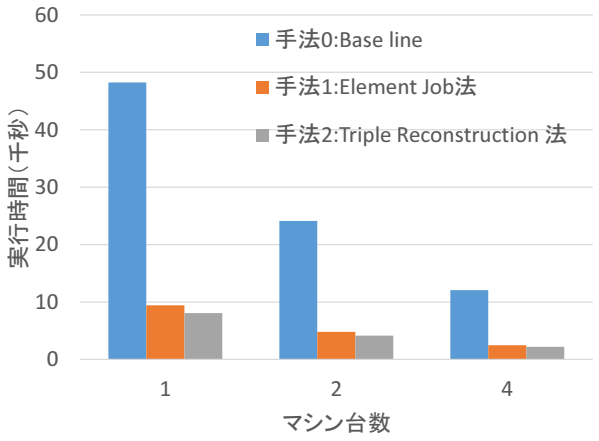

図 7　全体を一旦復号化する必要がある場合の復号化手法 RSA 暗号

図 7 と図 8 より，RSA 暗号が Paillier 暗号より復号化時間が早いが，どちらの暗号アルゴリズムでもシン台数が増えることによって手法 1 と手法 2 とも実行時間が短くなり，並列化の効果が見える．また，Element Job 法（手法 1）と Triple Reconstruction 法（手法 2）は手法 0 より実行時間が約 5 倍短

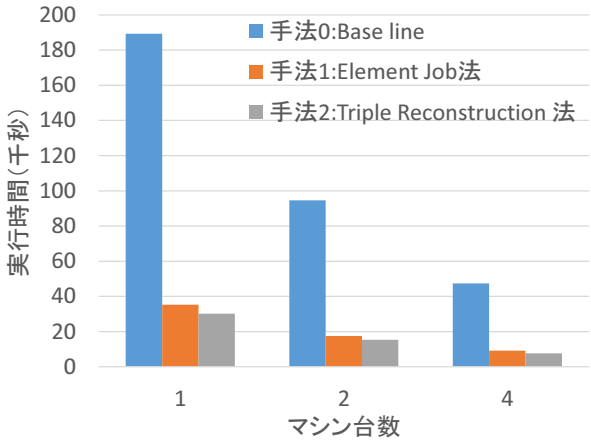

図 8　全体を一旦復号化する必要がある場合の復号化手法 Paillier 暗号

くなり，同じ暗号文のまとめてから復号化する手法の効果があることが分かる．

また，Triple Reconstruction 法は Element Job 法より早いと分かった．その理由については，図 9 に示すように Element Job 法の場合は各要素が別々の MapReduce Job で復号化が行うのである Subject とある Object の同じ暗号文になる場合でもまとめることにならなく，Triple Reconstruction 法より復号化回数が多くなり，実行時間が遅くなったと考えられる．

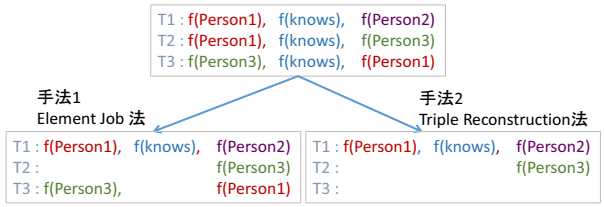

図 9　Element Job 法と Triple Reconstruction 法のグループ化

4.2.2　準同型暗号データ間の演算を用いる場合の手法の実験結果

各学生に平均点 (ub:averageScore) を追加した 50 大学のデータ (LUBM50EX) を RSA，Paillier アルゴリズムで暗号化されたデータを利用した．集約のクエリーが以下のようになる．

```
SELECT AVG(?X) WHERE {
    ?Person rdf:type ub:UndergraduateStudent ,
    ?Person ub:averageScore ?X
}
```

準同型性の手法は Paillier アルゴリズムで暗号化されたデータ，通常の手法は RSA アルゴリズムで暗号化されたデータを利用し，マシン台数を増やした場合の実行時間を図 10 に示す．図 10 より，どちらの手法でもマシン台数が増えることによって実行時間が短くなり，並列化の効果が見える．また，マシン台数が増えると準同型性を利用しない場合と準同型性を利用した場合の実行時間の比が小さくなり，同じ暗号文をまとめる効果があると考えられる．

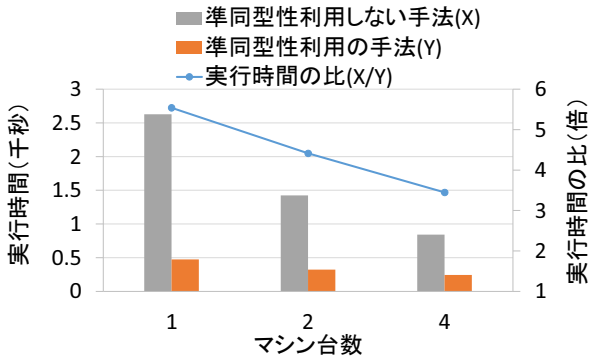

図 10　準同型暗号データ間の演算を用いる場合の手法

5. 関 連 研 究

データベースにおいて暗号化されたデータに対して演算を実現するための研究としては，CryptDB [7] が挙げられる．CryptDB は，R. A. Popa ら [5] によって提案された手法である．暗号化されたデータに対して全ての演算を可能とする暗号化アルゴリズムは現在現実的ではない．CryptDB では，暗号化されたデータに対して複数の演算が行えるために複数の暗号化アルゴリズムを利用して，複数の暗号を作成する．CryptDB は様々な演算が実行できるが “$WHEREsalary > age * 2 + 10$” のような複雑な条件の演算は実行できない [7].

RDF 暗号化の研究としては，M. Giereth により提案した PRE [9] が挙げられる．XML 暗号化手法を利用して，RDF データを暗号化する．PRE は部分的 RDF 暗号化と復号化手法を提案したが膨大な RDF データの復号化を考慮していない．

MapReduce を用いた大規模な RDF データの処理については，M.F. Husain によって提案した RDF データに対してクエリ処理を高速化する HadoopRDF がある。RDF データを事前に Predicate，Object によって複数のファイルに分け，複数の MapReduce ジョブにより集約を行う。また，J. Urbani らにより提案した RDF データ圧縮手法は辞書を利用し，各 RDF トリプルの要素は id で代入し，同じ要素をまとめて id で代入することでデータ量を減少する．これらの研究は暗号化データを考慮していない．

6. ま と め

本研究では，MapReduce を用いた暗号化されたデータに対し，平均年齢のような準同型性が利用できる単純な演算の手法と複雑な条件で全体を一旦復号化する必要がある場合の復号化手法を提案し，提案手法を評価した．

実験により全体を一旦復号化する必要がある場合の復号化手法について，マシン台数が増えるとどちらの手法も実行時間が短くなり，並列化の効果が確認できた．同じ暗号文をまとめることにより実行時間が約 5 倍早くなり，同じ暗号文のまとめてから復号化する手法の効果があることが分かった．

準同型暗号データ間の演算を用いる場合，準同型性を利用した手法，利用しない手法のどちらの手法もマシン台数が増えるとどちらの手法も実行時間が短くなり，並列化の効果が確認できた．また，マシン台数が増えると準同型性を利用しない場合と準同型性を利用した場合の実行時間の比が小さくなり，同じ暗号文をまとめる効果があると考えられる．

今後の課題として，暗号アルゴリズムを変えた実験を行う必要がある．暗号アルゴリズムによって復号化時間が異なるので，データ転送時間よりどのぐらい効果があるかを確認することが必要である．また，各プライバシーが異なる部分データに対して異なる暗号 key で暗号化されるレベル別暗号もあるので，そのような暗号に向けて，復号化手法も検討する必要がある．

謝　　辞

本研究の一部は，日本学術振興会科学研究費補助金基盤研究 (A)(#25240014) の助成により行われた．

文　　献

[1] Apache Hadoop. http://hadoop.apache.org, 2014
[2] Jeffrey Dean, Sanjay Ghemawat, “Mapreduce: Simplied data processing on large clusters”, Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI ’04), pp.137-149, 2004
[3] Message Passing Interface Forum http://www.mpi-forum.org, 2012
[4] Jena http://jena.apache.org, 2014
[5] R. L. Rivest, A. Shamir and L. Adleman, “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems”, Communication of the ACM, 21, 2, pp. 120—126, 1978.
[6] P. Paillier, “ Public-Key Cryptosystems Based on Composite Degree Residuosity Classes”, EUROCRYPT 1999, pp223-238, 1999
[7] R.A. Popa, N. Zeldovich, H. Balakrishnan, “CryptDB: A Practical Encrypted Relational DBMS”, Technical Report MIT-CSAIL-TR-2011-005, MIT (2011)
[8] M. Giereth, “PRE4J - A Partial RDF Encryption API for Jena”, Academic Medicine, 70(3):216—223, 2006.
[9] M. Giereth, “Partial Encryption of RDF Graphs” ISWC 2005, pp. 308—322, Springer, 2005.
[10] J. Urbani, J. Maassen, N. Drost, F. Seinstra, H. Bal, “Scalable RDF data compression with MapReduce. Concurrency and Computation”, Practice and Experience 25(1), 24—39, 2013.
[11] Y. Guo, Z. Pan, J. Heflin, “LUBM: A Benchmark for OWL Knowledge Base Systems”, Journal of Web Semantics 3(2), 158—182, 2005.
[12] M.F. Husain, J.P. McGlothlin, M.M. Masud, L.R. Khan, B.M. Thuraisingham, “Heuristics-based query processing for large RDF graphs using cloud computing”, IEEE Trans. Knowl. Data Eng. 23(9), 1312—1327, 2011.
[13] C. Gentry, “Fully homomorphic encryption using ideal lattices”, Proceedings of the 41st ACM Symposium on Theory of Computing (STOC 2009), pp. 169-178, ACM, 2009.
[14] HElib https://github.com/shaih/HElib, 2014
[15] S. Halevi, “Performance of HElib”, 2012 http://mpclounge.files.wordpress.com/2013/04/hespeed.pdf
[16] C. Moore, M. O’Neill, E. O’Sullivan, Y. Doroz, B. Sunar, “Practical homomorphic encryption: A survey”, Circuits and Systems (ISCAS), 2014 IEEE International Symposium on , vol., no., pp.2792,2795, 2014