DEIM Forum 2021 G25-3

完全準同型暗号アプリケーションを対象とした計算機資源動的割当手法におけるスケジューリング方式の検討

鈴木　拓也†　　山名　早人††

† 早稲田大学大学院基幹理工学研究科　〒 169–8555 東京都新宿区大久保 3–4–1
†† 早稲田大学理工学術院　〒 169–8555 東京都新宿区大久保 3–4–1
E-mail: †{t-suzuki,yamana}@yama.info.waseda.ac.jp

あらまし　完全準同型暗号を用いることでデータを暗号化したまま計算することができ，データに含まれる秘密を守りながら，アプリケーションを実行することができる．完全準同型暗号を用いたアプリケーションは実行時間が長いため，実行時間短縮を目的に，クライアントサーバモデルにおいては，豊富な計算機資源を活用した並列処理といった様々な手法が提案されている．特に，各クライアントが独立してクエリをクラウドサーバへ送信する状況では，個々のクエリに対応するジョブが並行もしくは並列に実行される．この状況におけるジョブの平均レイテンシと最大レイテンシを短縮することを目的に，著者らは，各ジョブにおける準同型演算ごとに並列化粒度と実行タイミングを動的に決定する手法を提案してきた．本論文では，この既存手法に対して，異なるスケジューリング方式を適用した場合での性能評価をシミュレーションで実施し，得られた結果を元に，最適なスケジューリング方法を検討する．

キーワード　完全準同型暗号, クライアントサーバアプリケーション, タスクスケジューリング, ハイパフォーマンスコンピューティング

1 はじめに

近年，クラウドサーバ上では様々なアプリケーションが実行されている．アプリケーション実行に伴い，クライアントとクラウドサーバ間で送受信されるデータには，個人情報といった機密情報が含まれる場合がある．クライアントとクラウドサーバ間の通信経路上では，AES 暗号といった暗号技術を用いて暗号化されたデータを送受信することで，機密情報漏洩を防いでいる．しかし，アプリケーションの実行のためには，クラウドサーバ上で暗号化されたデータを復号する必要がある．そのため，悪意のあるクラウドサーバ管理者や攻撃者によるクラウドサーバへの攻撃で，クライアントが送受信するデータに含まれる機密データが漏洩する危険性がある．

データを暗号化したまま演算可能である完全準同型暗号 [1] を用いることで，クラウドサーバ上での機密データ漏洩を防ぐことができる．しかし，完全準同型暗号を用いたアプリケーション (以下，完全準同型暗号アプリケーション) は実行時間が長いため，実行時間を短縮させる必要がある．実行時間を短縮させる方法として，実行すべき準同型演算の個数を削減する手法や並列化による高速化といった様々な手法が存在する．特に，Web アプリケーションのようなクライアントサーバアプリケーションにおいては，クライアントから送信されるデータ (以下，クエリ) がクラウドサーバへ到着する頻度によって，各クエリに対応するジョブのそれぞれにどの程度の計算機資源を割り当てるかを，動的に決定する必要がある．メニーコア CPU 搭載サーバ上で，この課題を解決し，ジョブの通信部分を除いた平均レイテンシと最大レイテンシを短縮させることを目的とした，DAMCREM という手法を著者ら提案してきた [2].

DAMCREM では，アプリケーションに対する 1 つのクエリから出力を得るための処理を 1 つのジョブと定義する．また，1 つ以上の完全準同型暗号上での演算 (以下．準同型演算) を 1 つのまとまりであるマクロタスクとして定義する．そして，完全準同型暗号特有の，条件分岐が存在しないことや準同型演算の実行時間の推測が可能であるという性質を利用して，マクロタスクごとに割り当てる計算機資源量や実行タイミングを決定する．その結果，クラウドサーバへの平均クエリ到着間隔が短く，クラウドサーバの負荷が高い状況，すなわちクラウドサーバで複数のジョブを同時に実行する必要がある状況では予め決定しておいたスレッド数を割り当てるナイーブ手法よりも平均レイテンシを短くすることを達成した．しかし，DAMCREM では同時に実行できる完全準同型暗号アプリケーションは 1 種類のみであるため，1 台のサーバで複数種類の完全準同型暗号アプリケーションを同時に実行する場合は，計算機資源を予め分割し，アプリケーションごとに割り当てるといった方法をとる必要がある．この方法では，あるアプリケーション種別用に割り当てた計算機資源が余っていても，他のアプリケーションのジョブに割り当てることができず，計算機資源を十分に活用することができない．そこで，本論文では，DAMCREM におけるマクロタスクの実行順序スケジューリング方式について，2 種類の完全準同型暗号アプリケーションを同時に実行した場合における性能をシミュレーションによって評価することで，DAMCREM における最適なマクロタスク実行順序スケジューリング方式を検討する．

本論文は以下の構成である．2 節で，完全準同型暗号と，

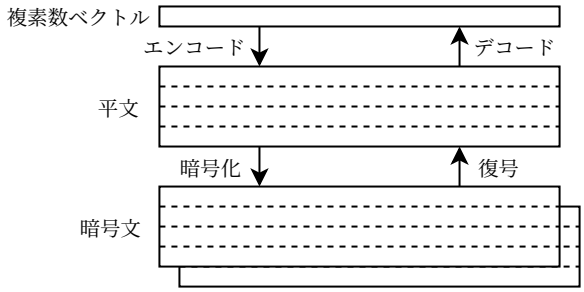

図 1　複素数ベクトルと平文，暗号文の関係 (平文や暗号文における点線で区切られた領域は整数ベクトルを表す)

DAMCREM が使用している完全準同型暗号特有の制約について説明する．その後，3 節で，完全準同型暗号アプリケーションに対するスケジューリング手法に関する関連研究を示す．4 節で本論文での評価実験においてベースラインとなる DAMCREM を説明し，5 節で，同時に 2 種類の完全準同型暗号アプリケーションのジョブを同時に実行する状況において，DAMCREM における最適なマクロタスク実行順序スケジューリング方式の検討について示す．そして，6 節で実施した評価実験を説明し，その結果および考察を示す．最後に，7 節でまとめる．

2　完全準同型暗号

2.1　概　　要

完全準同型暗号は，データを暗号化したまま任意の回数の演算を適用できる暗号方式の総称である．完全準同型暗号には，様々な方式が存在する．本論文では，剰余表現を用いた CKKS 方式 [3] と呼ばれる方式について説明する．

CKKS 方式は，固定小数点数で表された複素数を暗号化して扱うことができる．また，図 1 に示すように，パッキングと呼ばれる手法 [4] を用いることで，1 つの平文や暗号文に対して，複数の複素数をエンコードや暗号化することができる．1 つの平文や暗号文は，複数の整数ベクトルで構成される．準同型演算を構成する処理は，このベクトルごとに実行することができるため，1 つの準同型演算に対して複数のスレッドを割り当てることによる高速化が可能である．このベクトルの数は，平文や暗号文のパラメータによって決まり，準同型演算によっては処理中にベクトルの数が増減する．したがって，平文や暗号文のパラメータや準同型演算によって，並列化した際の 1 スレッドあたりの高速化の効果が異なる．実行時間の短縮のためには，適切なスレッド数を割り当てる必要がある．

平文や暗号文のベクトル数に影響する平文や暗号文のパラメータについて説明する．スロット数と呼ばれるエンコードや暗号化できる複素数の個数や，レベルと呼ばれる後述の Rescaling という演算を適用できる回数，スケールファクタと呼ばれる固定小数点数の小数部の桁数に関するパラメータが存在する．スロット数やレベルを大きくするほど，1 つあたりの平文や暗号文のデータサイズは大きくなる．また，スケールファクタの値を大きくするほど，小数部の桁数が多くなる．ただし，他のパラメータが同一の条件下では，整数部の桁数がスケールファクタの分だけ少なくなり，表せる値の範囲が小さくなる．

完全準同型暗号では，下記の準同型演算が存在する．

- 準同型加算
- 準同型乗算
- Relinearization
- Rescaling
- Bootstrapping

準同型加算と準同型乗算では，平文と暗号文の 1 つずつを入力とすることもでき，2 つの暗号文を入力とする場合と比べて実行時間が短い．

2 つの暗号文を入力とする準同型乗算では，演算結果の暗号文のデータサイズが増加し，その暗号文を入力とする準同型演算の計算量が増加してしまう．そこで，Relinearization を適用することで，暗号文のデータサイズを削減することができる．

CKKS 方式では，固定小数点数をエンコードや暗号化して扱い，整数部と小数部の合計桁数が決まっている．そして，準同型乗算を行う度に，スケールファクタで決まる桁数分だけ小数部の桁数が増えてしまい，桁数が不足した整数部の情報を保つことができなくなる．さらに，値にノイズを加えて暗号化することで，攻撃耐性を実現しており，準同型乗算によって増えた小数部の値に意味は無い．そこで，スケールファクタ分の桁数，すなわち準同型乗算によって増えた小数部を削除するのが Rescaling である．この Rescaling を行うと暗号文のレベルが 1 小さくなる．レベルが 0 の状態では Rescaling を適用することができず，準同型乗算を適用すると，演算結果の暗号文は正しく復号できない．

暗号文に対して Bootstrapping を適用すると，暗号文のレベルを増やすことができる．ただし，Bootstrapping は他の準同型演算よりも実行時間が長い．そのため，機械学習の学習処理のように，準同型乗算や Rescaling を適用する回数を予め知ることができない場合や，レベルが数十以上必要な場合に Bootstrapping が使用される．

2.2　完全準同型暗号特有の制約

完全準同型暗号の制約を利用することで，完全準同型暗号を用いない通常のアプリケーションと比べて，より最適に近いスケジューリングを短時間で行うことが可能である．利用できる完全準同型暗号の制約として，下記の 2 つが挙げられる．

1 つ目は，準同型演算の実行時間は，暗号文に含まれる値に依らないことである．準同型演算の実行時間は入力された平文や暗号文のパラメータや使用する計算機資源によって決まる．なお，平文や暗号文，公開鍵はパラメータによっては，データサイズが 1MB を超える．平文上での計算と異なり，キャッシュヒットによる高速化が望めず，キャッシュヒット率の変化への影響もない．つまり，アプリケーションへの入力暗号文に含まれる値による実行時間の変化がないため，準同型演算の実行時間を予測することが容易である．

2 つ目は，暗号文に含まれる値を条件とした条件分岐が存在しないことである．もし，暗号文に含まれる値を条件に条件分

岐が可能であるとすると，条件分岐でどちらに進んだかを知ることで，暗号文に含まれる値を知ることができてしまう．したがって，暗号文に含まれる値を条件とした条件分岐は完全準同型暗号アプリケーション上には存在しないものとして考えることができる．つまり，アプリケーションへの入力に対する出力を計算するには，そのアプリケーションを構成する全ての準同型演算が実行される必要があるということである．

3 関連研究

完全準同型暗号アプリケーションの実行時間短縮手法としては，静的な最適化手法と動的な最適化手法のどちらかが用いられる．特に，静的な最適化手法では，完全準同型暗号における，準同型演算の実行時間を推測可能であることと，暗号文の値を条件とした条件分岐が存在しないことが活用される．また，完全準同型暗号アプリケーションを構成する準同型演算をグラフとして表して最適化を行う手法が多い．

静的な最適化手法としては，特に実行時間が長い Relinearization や Rescaling，Bootstrapping といった演算の実行タイミングの最適化が挙げられる．佐藤らは，2017 年にループアンローリングを用いた Bootstrapping の実行タイミングの近似解と求める手法 [5] や，2019 年には Relinearization や Bootstrapping の実行タイミングの厳密解を従来よりも高速に求める手法 [6] を提案した．また，完全準同型暗号の知識を持たないユーザでも完全準同型暗号アプリケーションを作成できるようにするためのコンパイラに関する研究も存在する [7] [8] [9] [10] [11]．これらの手法では，完全準同型暗号アプリケーションに変換する際に，Relinearization や Rescaling などの実行タイミングを決定する．また，ニューラルネットワークではモデルパラメータが 1 である場合があり，1 をかけるといった演算は実行しなくても結果に影響しないため削減する [8] [9]．特に，Dathathri らの CHET [10] や EVA [11] では，コンパイラとしての機能で，完全準同型暗号アプリケーションを作成するのに慣れた専門家よりも実行時間が短くなるグラフを生成することを達成した．

動的な最適化としては，静的に最適化した準同型演算をグラフで表し，実行可能な準同型演算からリストスケジューリングを用いて実行することで計算機資源を活用した手法や，準同型演算に割り当てるスレッド数を動的に決定する手法が挙げられる．Carpov らの Armadillo [7] や Dathathri らの EVA [11] では，静的な最適化によって得られた準同型演算のグラフに対して，各準同型演算の実行タイミングを動的にスケジューリングして実行することで，静的な最適化のみの手法よりも計算機資源を効率良く使用することができた．また，著者らが 2020 年に発表した DAMCREM [2] は，単一種類の完全準同型暗号アプリケーションを対象に，受信した各クエリに対応するジョブを同時に実行する状況下で，準同型演算の実行順序や準同型演算に割り当てる計算機資源を動的に決定する手法である．なお，著者らは 2019 年に DAMCREM の元となる，平文や暗号文のデータ構造を元に動的に割当スレッド数を決定する手法を発表した [12]．

しかし，いずれの手法も，同時実行ジョブ数が 1 つのみか常に規定数，もしくは単一種類の完全準同型暗号アプリケーションのみを対象としているため，複数種類の完全準同型暗号アプリケーションをクラウドサーバ上で，それぞれ任意の個数のジョブを実行する際には適用することができない．

4 DAMCREM

4.1 概　　要

DAMCREM は，完全準同型暗号を用いたアプリケーションをサーバ上で同時に 1 つ以上のジョブを実行する際に，いくつかの準同型演算をひとまとまりとしたマクロタスクごとに，各マクロタスクの状態に応じて，動的に計算機資源を割り当てる手法である．そして，平均レイテンシと最大レイテンシを短縮することを目指した手法である．DAMCREM では，割り当てる計算機資源の単位としてスレッド数を用いる．このスレッド数は，CPU の物理コア数と同じであり，静的な値である．なお，マクロタスクの実行を管理するのに 1 スレッドを使用し，残りのスレッドをマクロタスクの実行に使用する．そして，高帯域幅メモリの MCDRAM (Multi Channel DRAM) を搭載した Intel の Xeon Phi 7230F を対象としており，十分な帯域幅を持っているという前提のため，使用メモリ帯域幅を考慮していない．また，クライアントとサーバ間の通信部分は考慮していない．本論文でも同様に，メモリ帯域幅と，クライアントとサーバ間の通信部分は考慮しない．

DAMCREM を構成する処理は，「静的解析」と「動的割当」の 2 つに分かれている．静的解析は「マクロタスクグラフ構築」と「各マクロタスクへの割当スレッド数の候補生成」の 2 つに分かれている．動的割当は「マクロタスク実行順序スケジューリング」と「各マクロタスクへの割当スレッド数選択」の 2 つに分かれている．

静的解析では，式 (1) で定義される，マクロタスクに n スレッド割り当てた時の 1 スレッドあたりの効率 $r(n)$(以下，効率) を使用する．

$$r(n) = \frac{T(1)}{n \times T(n)} \tag{1}$$

$T(n)$ は n スレッド割り当てた時の実行時間であり，$n \times T(n)$ はスレッドを占有した合計時間である．式 (1) が表す意味は，1 スレッドのみ割り当てた場合を基準として，n スレッド割り当てた時のスレッドを占有した合計時間の割合の逆数である．すなわち，$r(n)$ が大きいほどスレッドを占有した合計時間が少ないため，利用可能なスレッド数が十分に多い状況下においては，n スレッドを割り当てたことによる実行時間の短縮効果が高いことを意味する．つまり，$n < n'$ において，$r(n) >= r(n')$ である場合では，n' スレッド割り当てるよりも n スレッド割り当てる方が効率が良い．

4.2 静的解析

4.2.1 マクロタスクグラフ構築

マクロタスクグラフ構築では，下記の 2 つの戦略で行う．

（1） 準同型演算の実行順序の決定

（2） マクロタスクの構成

まず，1 つ目の準同型演算の実行順序の決定方法について説明する．完全準同型暗号アプリケーションを構成する準同型演算を DAG (Directed acyclic graph) として表す．この時，アプリケーションで消費するレベルが少なくなるようにする．さらに，Relinearization や Rescaling の数が最少になるようにマクロタスクグラフを構築する．具体的には，同じレベルの暗号文を出力する 2 つの準同型乗算の後に準同型加算が続く場合，各準同型乗算の直後に Relienarization や Rescaling を実行する代わりに，準同型加算後に Relinearization や Rescaling を実行するようにする．すなわち，Relinearization と Rescaling を 2 回ずつ実行する代わりに，1 回ずつ実行することで高速化することができる．

次に，2 つ目のマクロタスクの構成方法について説明する．マクロタスクのスケジューリングやスレッド生成によるオーバヘッドを小さくするために，できるだけマクロタスクを粗く構成する必要がある．しかし，効率が良いスレッド数が異なる準同型演算を 1 つのマクロタスクとして構成すると，並列化による高速化の効果が低くなる．特に，入力となる平文や暗号文のレベルが同じであっても，準同型乗算と Relinearization，Rescaling の 3 つは 1 スレッドあたりの効率が良いスレッド数が異なる．一方で，準同型加算については，他の準同型演算と比べると実行時間が短く，スレッド生成や同期によるオーバヘッドを考慮すると，2 スレッド以上割り当てても並列化の効果が低い．そこで，DAMCREM では，1 つのマクロタスクに準同型乗算か Relinearization，Rescaling のいずれか 1 つと，任意の数の準同型加算を含む．準同型加算は直後の準同型演算と同じマクロタスクに含める．ただし，準同型加算の直後に準同型演算が存在しない場合は，この準同型加算は直前の準同型演算と同じマクロタスクに含める．

4.2.2 各割当スレッド数の候補生成

マクロタスクへの割当スレッド数選択にかかる実行時のオーバヘッドを削減するために，割当スレッド数の候補を予め作成する．2 節で説明した通り，入力された平文や暗号文のパラメータが分かれば，マクロタスクの実行時間は推測できる．

平文や暗号文を構成する整数ベクトルの個数によっては，割当スレッド数を増やしても実行時間が変わらない場合や，実行時間が十分に短縮されない場合が存在する．したがって，効率が良い割当スレッド数のみで候補を生成することで，計算機資源を有効に活用しつつ，実行時のオーバヘッドを削減することが可能となる．同様に，割り当てるスレッド数を多くするほど，大域的には効率が下がる傾向にある．そのため，閾値を設定し，効率が閾値未満となるスレッド数を候補から除く．

4.3 動的割当

4.3.1 マクロタスク実行順序スケジューリング

DAMCREM では，マクロタスクの実行順序を下記の 2 つの戦略で決定する．

（1） クエリ到着時刻が早いジョブを優先して実行する．

（2） 同じかクエリ到着時刻が同じジョブ内のマクロタスクに対しては，1 スレッドで実行した際のそのマクロタスクから最後のマクロタスクまでのクリティカルパス上にあるマクロタスクを合計時間が長いマクロタスクを優先して実行する．

1 つ目の理由は，クエリごとのジョブ実行時間が同じであるためである．2 つ目の理由は，クリティカルパスが長いマクロタスクから先に実行することで，依存関係による実行の遅延を減らすためである．なお，完全準同型暗号アプリケーションでは条件分岐が存在しないため，クリティカルパスは予め確定させることができる．

4.3.2 各マクロタスクへの割当スレッド数選択

マクロタスクごとに予め作成した割当スレッド数の候補から，実行可能なマクロタスクに割り当てるスレッド数を決定するアルゴリズムを説明する．まず，実行可能なマクロタスクの数が利用可能なスレッド数以上である場合は，実行順序スケジューリングに従って，利用可能なスレッド数分の実行可能なマクロタスクを選択し，各マクロタスクに 1 スレッドを割り当てる．次に，実行可能なマクロタスクの数が利用可能なスレッド数よりも少ない場合は，効率の下限を算出し，各実行可能なマクロタスクに対して下限以上の効率を持つ最大のスレッド数を選択する．なお，割当スレッド数の総数が，利用可能なスレッド数を超えない最大値となるように，効率の下限を算出する．

5 DAMCREM における最適なマクロタスク実行順序スケジューリング方式の検討

DAMCREM は，単一種類の完全準同型暗号アプリケーションのみのジョブを同時に実行する状況のみでしか実行することができない [2]．したがって，複数種類の完全準同型暗号アプリケーションのジョブを同時に実行する状況に適応させる必要がある．そこで，本論文の目的は，2 種類の完全準同型暗号アプリケーションのそれぞれからなる複数のジョブを同時に実行する状況下において，DAMCREM のマクロタスク実行順序スケジューリング方式を改良することで，各種類ごとの完全準同型暗号アプリケーションのジョブの平均レイテンシと最大レイテンシを削減することである．

DAMCREM を元に，本論文で適用した変更点について説明する．本論文では，DAMCREM のマクロタスク実行順序スケジューリングに変更を加えることで，2 種類の完全準同型暗号アプリケーションを同時に実行できるようにする．

マクロタスクの実行順序スケジューリング手順として，次の 2 つの戦略を考える．

（1） 1 段階方式：各マクロタスクが属するジョブを考慮せずに，マクロタスク単位でのみ実行順序を決定する方法

（2） 2 段階方式：ジョブ単位で実行順序を決定し，その後ジョブ内のマクロタスク単位で実行順序を決定する方法

ジョブやマクロタスクの実行順序スケジューリング方式として，下記のスケジューリング方式を考える．

（1） FIFO (First-In-First-Out) スケジューリング

（2） デッドラインスケジューリング I (クリティカルパス上のマクロタスクのみを考慮)

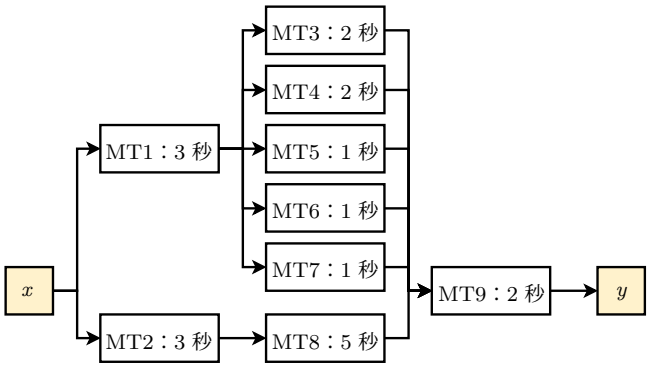

図 2　マクロタスクグラフの例 (x を入力，y を出力，MTi：t 秒 は実行時間が t 秒の i 番目のマクロタスクを表す)

（3） デッドラインスケジューリングⅡ (全てのマクロタスクを考慮)

FIFO スケジューリングでは，ジョブ単位ではクエリが到着した時刻が早いジョブを優先し，マクロタスク単位では先に実行可能状態になったマクロタスクを優先する．デッドラインスケジューリングでは，ジョブのデッドラインを，1 スレッドで実行した際の考慮対象のマクロタスクの合計時間をクエリ到着時刻に足した値とする．デッドラインスケジューリングⅠでは，1 スレッドで実行する際のクリティカルパス上のマクロタスクのみを考慮し，デッドラインスケジューリングⅡでは，全てのマクロタスクを考慮して，それぞれデッドラインを決定する．そして，デッドラインスケジューリングでのマクロタスク単位では，ジョブのデッドラインから考慮すべきマクロタスクの合計実行時間を引いた結果の時刻，すなわち対象のマクロタスクを実行開始しなければならないデッドラインが早いマクロタスクを優先する．なお，いずれも，複数のマクロタスクの優先度が同じである場合は，ランダムに決定される．

デッドラインスケジューリングⅠとⅡの違いについて説明する．デッドラインスケジューリングⅠでは，クリティカルパスのみを考慮するため，マクロタスクグラフ内のマクロタスクを依存関係が許す限り並列に実行できる場合では最適であると考えられる．なお，DAMCREM ではデッドラインスケジューリングⅠを使用している．一方で，デッドラインスケジューリングⅡでは，マクロタスクグラフ内のマクロタスクを逐次実行しなければならない場合で最適であると考えられる．

デッドラインスケジューリングⅠとⅡの違いについて，例を用いて具体的に説明する．図 2 のマクロタスクグラフは，入力 x を受信した時に 1 つジョブとして実行され，出力 y を求めるとする．

ジョブ単位で実行順序を決定する場合について説明する．デッドラインスケジューリングⅠでは，ジョブの実行開始からデッドラインまでの時間は，図 2 のクリティカルパス上にある MT2 と MT8，MT9 の合計 10 秒である．一方，デッドラインスケジューリングⅡでは，全てのマクロタスクを考慮するため，MT1〜9 の合計 20 秒が，ジョブの実行開始からデッドラインまでの時間となる．

マクロタスク単位では，ジョブの開始直後では，MT1 と MT2 のマクロタスクが実行可能であるため，MT1 と MT2 の実行順序を決める必要がある．デッドラインスケジューリングⅠでは，クリティカルパス上のマクロタスクのみを考慮するため，MT1 の実行開始からジョブのデッドラインまでの時間は，MT1 の 3 秒と MT3 もしくは MT4 の 2 秒，MT9 の 2 秒の合計の 7 秒となる．同様に，MT2 の実行開始からジョブのデッドラインまでの時間は，MT2 の 3 秒と MT8 の 5 秒，MT9 の 2 秒の合計 10 秒となる．したがって，ジョブ内における実行開始しなければならないデッドラインがより早い MT2 を優先して実行することになる．一方，デッドラインスケジューリングⅡでは，対象のマクロタスクの依存先全てのマクロタスクを考慮するため，MT1 の実行開始からジョブのデッドラインまでの時間は，MT1 と MT3〜7，MT9 の合計 12 秒である．MT2 の実行開始からジョブのデッドラインまでの時間は，デッドラインスケジューリングⅠと同様に合計 10 秒である．したがって，実行開始しなければならないデッドラインがより早い MT1 の方を優先して実行することになる．

6 評価実験

6.1 実験目的

本評価実験を実施する目的は，複数種類の完全準同型暗号アプリケーションを同時に実行可能な状況下で，DAMCREM におけるマクロタスクの実行順序スケジューリング方式をどのように設計すべきかを明らかにすることである．最適にスケジューリング方式を明らかにすることで，DAMCREM の欠点である，単一種類の完全準同型アプリケーションしか同時に実行できない，という問題点を解決することができる．

6.2 実験方法

本評価実験はシミュレーションにて実施する．5 節で説明した，2 種類のスケジューリング手順と 3 種類のスケジューリング方式をそれぞれ組み合わせ，計 6 通りの方法を比較する．

完全準同型暗号アプリケーションとして，下記の 2 種類を使用する．1 つ目は，図 3 に示す，指数関数のような 1 入力関数に対するテイラー展開による多項式近似を計算するアプリケーション (以下，アプリケーション 1) である．定数係数は予めクラウドサーバ上に保存されているとし，入力暗号文の初期レベルは 8 とした．2 つ目は，図 4 に示す，4 層のニューラルネットワークの推論を行うアプリケーション (以下，アプリケーション 2) である．各層の重みパラメータはサーバ上に保存されている暗号文とし，入力は 3 次元とした．また，入力暗号文の初期レベルは 7 とした．なお，図 3 および図 4 で用いられている記号の凡例を図 5 に示す．

クラウドサーバへの負荷を決定する要素として，アプリケーションの種類ごとに，平均クエリ到着間隔を決め，ポアソン分布に従って，各アプリケーション種類に対応するクエリ到着時刻を独立に決定する．アプリケーション 1 の平均クエリ到着間隔 dt_{pa} は，200[ms]，280[ms]，360[ms] のいずれかを使用し，アプリケーション 2 の平均クエリ到着間隔 dt_{nn} は，320[ms]，

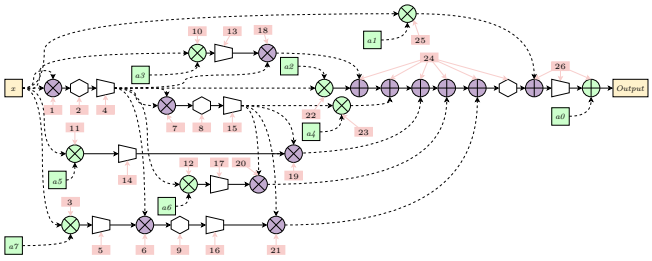

図 3　アプリケーション 1(テイラー展開による多項式近似を計算するアプリケーション) のマクロタスクグラフ

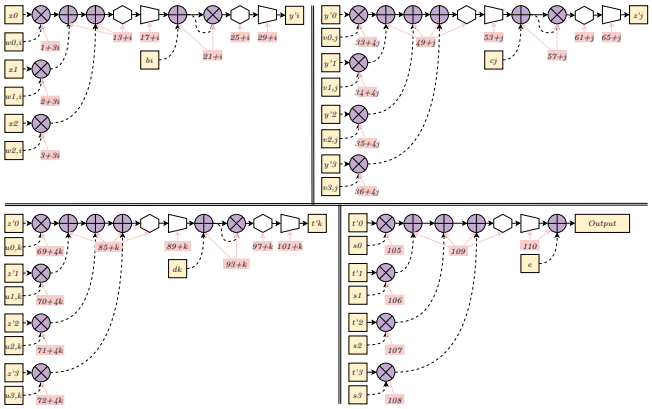

図 4　アプリケーション 2(4 層ニューラルネットワークの推論を行うアプリケーション) のマクロタスクグラフ ($0 \leqq i < 4$, $0 \leqq j < 4$, $0 \leqq k < 4$)

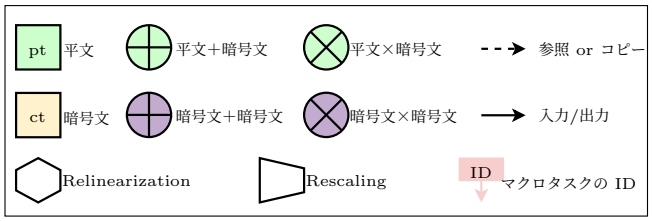

図 5　図 3 および図 4 で用いる記号の凡例

400[ms]，480[ms] のいずれかを使用した．各クエリに対応するジョブは，そのクエリ到着時刻に実行開始されるとし，ジョブ開始時刻 (クエリ到着時刻) からジョブの実行完了までをそのジョブのレイテンシとした．そして，それぞれの平均クエリ到着間隔の組み合わせごとに 5 回ずつ実験を実施し，アプリケーションごとの平均レイテンシと最大レイテンシを比較した．

平均レイテンシと最大レイテンシを算出する際には，実験開始直後と実験終了直前は，同時に実行されるジョブ数が少なくなり，平均レイテンシが小さくなってしまう可能性がある．そこで，実験開始してから一定時間経過後から平均レイテンシや最大レイテンシを算出する対象のジョブを測定するようにする．図を用いて具体的に説明する．図 6 は，実験開始からの経過時間と各アプリケーションの実行開始したジョブの個数を示している．赤色の線は両方のアプリケーションにおいて，アプリケーション 1 とアプリケーション 2 において，遅い方の 512 番目のジョブの実行開始時刻を表している．実験開始から赤色の線の時間が経過してから測定を開始し，赤色の線の時刻以前

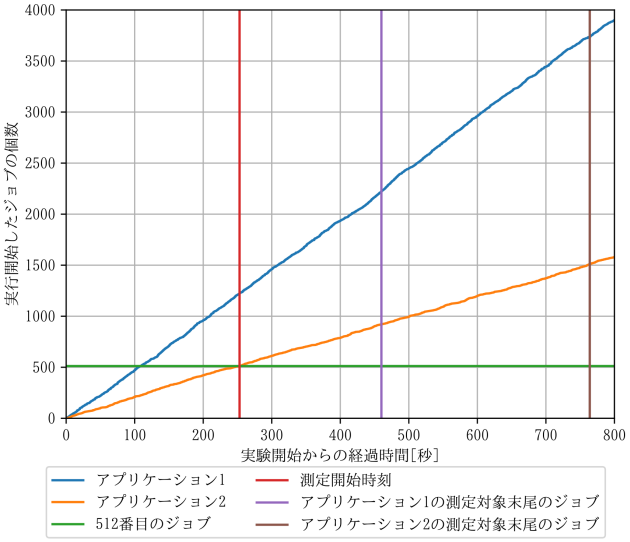

図 6　実験開始からの経過時間とアプリケーションごとの実行開始したジョブの個数

に実行開始したジョブは測定対象から除外した．また，紫色の線はアプリケーション 1 の測定終了の時刻を表しており，茶色の線はアプリケーション 2 の測定終了時刻を表す．したがって，赤色の線と紫色の線の間にあるアプリケーション 1 のジョブと，赤色の線と茶色の線の間にあるアプリケーション 2 のジョブが測定対象となる．各アプリケーションごとに 1000 個のジョブを測定対象として，平均レイテンシと最大レイテンシを算出した．なお，測定終了時刻を過ぎても，負荷を保つために，そのアプリケーションのジョブの実行は継続する．

シミュレータにおいては，各マクロタスクの実行時間はランダム性を持たせず，予め実測した値を使用した．また，DAM-CREM のアルゴリズムやマクロタスクのスケジューリング等におけるオーバヘッドは存在しないものとして扱った．

6.3　結　　果

図 7〜10 に実行結果を箱ひげ図で示す．各図における各グラフは，アプリケーションごとの平均クエリ到着間隔の組み合わせごとの結果を示している．例えば，各図における左下のグラフは，多項式近似を計算するアプリケーションの平均クエリ到着間隔が 200[ms] であり，ニューラルネットワークの推論をするアプリケーションの平均クエリ到着間隔が 320[ms] である．また，それぞれのグラフの横軸の「X-Y」はスケジューリング手順及びスケジューリング方式を表す．X については，1 は「1 段階方式」を意味し，2 は「2 段階方式」を表す．Y については，1 は「FIFO スケジューリング」を意味し，2 は「デッドラインスケジューリング I」を表し，3 は「デッドラインスケジューリング II」を表す．

6.4　考　　察

図 7 の左下のグラフより，デッドラインスケジューリング II では，他のスケジューリング手法よりも，アプリケーション 1 の平均レイテンシを短縮することができていることが分かる．

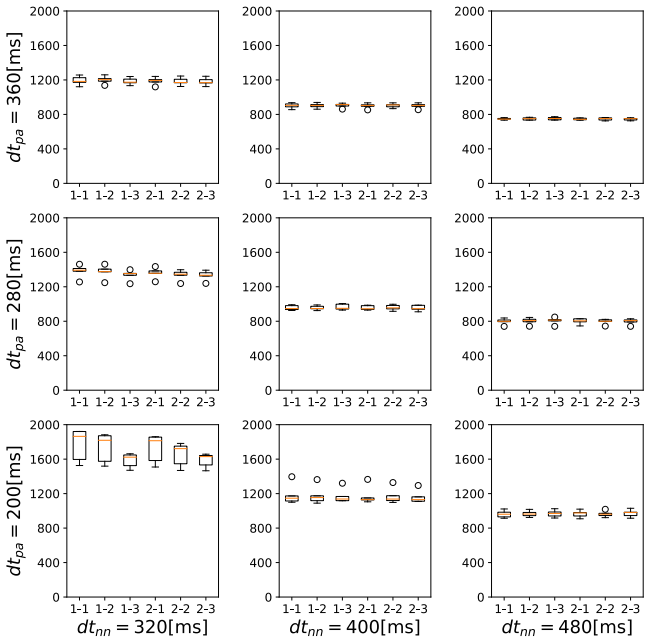

図 7　スケジューリング手順とスケジューリング方式の組み合わせごとに対するアプリケーション 1 の平均レイテンシ [ms] (9 つのグラフは，行方向ではアプリケーション 1 の平均クエリ到着間隔 dt_{pa} が同じであり，列方向ではアプリケーション 2 の平均クエリ到着間隔 dt_{nn} が同じであり，交差した位置にあるグラフは各平均クエリ到着間隔を組み合わせた時の結果を示す)

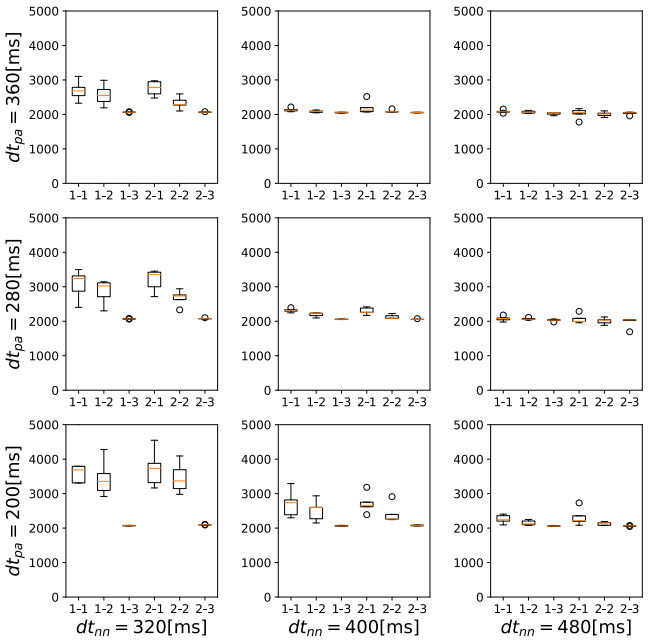

図 8　スケジューリング手順とスケジューリング方式の組み合わせごとに対するアプリケーション 1 の最大レイテンシ [ms] (9 つのグラフは，行方向ではアプリケーション 1 の平均クエリ到着間隔 dt_{pa} が同じであり，列方向ではアプリケーション 2 の平均クエリ到着間隔 dt_{nn} が同じであり，交差した位置にあるグラフは各平均クエリ到着間隔を組み合わせた時の結果を示す)

一方で，図 9 の左下のグラフでは，デッドラインスケジューリング I もしくは 2 段階方式の FIFO スケジューリングの方が，デッドラインスケジューリング II よりもアプリケーション 2 の平均レイテンシを短縮した．図 8 及び図 10 においても同様の傾向が見られる．これは，同時に各アプリケーションのジョブが実行開始された場合，デッドラインスケジューリング II における，アプリケーション 1 のデッドラインがアプリケーション 2 のデッドラインよりも短く，アプリケーション 1 のジョブのマクロタスクの方が優先的に実行されたためと考えられる．本評価実験においては，他のスケジューリング方式と比べて，両方のアプリケーションのレイテンシが低いスケジューリング方式が優れていると判断できる．また，試行ごとのばらつきが小さく，クエリ到着時刻のばらつきの影響を低く抑えることができるスケジューリング方式が優れていると判断できる．したがって，2 段階方式でのマクロタスク単位でのデッドラインスケジューリング II が適している．

5 回の試行における最大レイテンシの中央値においても，2 段階方式でのデッドラインスケジューリング II が優れていることが分かる．まず，アプリケーション 1 においては，FIFO スケジューリングやデッドラインスケジューリング I では，デッドラインスケジューリング II と比べて 1.3〜1.7 秒長い．また，アプリケーション 2 では，マクロタスク単位でのデッドラインスケジューリング II と最小値を達成したスケジューリング方式との差は 1.2 秒であった．

なお，各図の右上のグラフのように，平均クエリ到着間隔が長い場合は，いずれのスケジューリング手順とスケジューリング方式の組み合わせでも平均レイテンシの差は数 [ms] 程度であり，最大レイテンシは数十 [ms] の差であった．このことから，負荷が低い場合は，負荷が高い場合と比べて，スケジューリング手順やスケジューリング方式の影響が小さいことが分かる．DAMCREM では，負荷を予め知らない状況において，ジョブのレイテンシを短縮することを目的としている．したがって，負荷が低い場合よりも，負荷が高い場合においてジョブのレイテンシを短縮できるスケジューリング手順とスケジューリング方式の組み合わせを用いるのが適している．

7　おわりに

本論文では，2 種類の完全準同型暗号アプリケーションについて，同時に複数のジョブを実行する状況下において，DAMCREM でのマクロタスクの実行順序を動的に決定するスケジューリング方式に対して，最適なスケジューリング方式を明らかにすることを目的とした．スケジューリングの手順として，各マクロタスクが属するジョブを考慮せずに，マクロタスク単位でのみ実行順序を決定する方法である 1 段階方式と，ジョブ単位で実行順序を決定し，その後ジョブ内のマクロタスク単位で実行順序を決定する方法である 2 段階方式の 2 種類の方法を用いた．また，スケジューリング方式として，FIFO スケジューリングとクリティカルパス上のマクロタスクのみを考慮するデッドラインスケジューリング I，全ての依存先マクロタスクを考慮するデッドラインスケジューリング II の合計 3 種類を用いた．そして，評価実験にて，上記のスケジューリング手

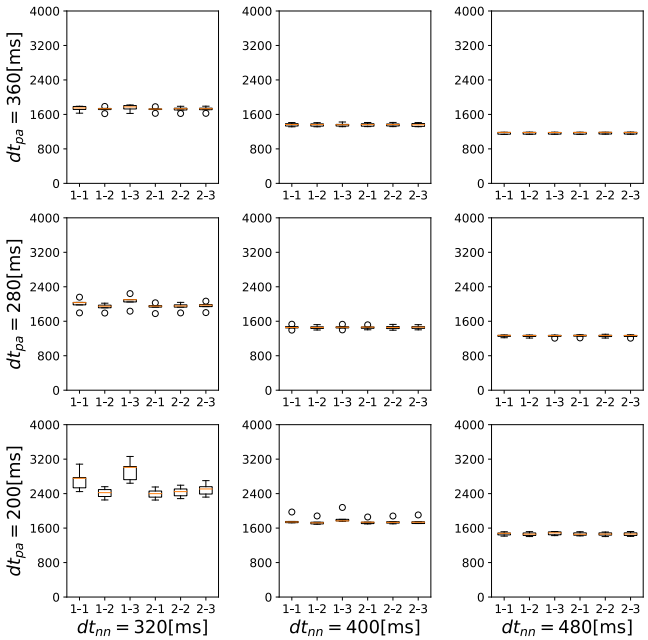

図 9　スケジューリング手順とスケジューリング方式の組み合わせごとに対するアプリケーション 2 の平均レイテンシ [ms] (9 つのグラフは，行方向ではアプリケーション 1 の平均クエリ到着間隔 dt_{pa} が同じであり，列方向ではアプリケーション 2 の平均クエリ到着間隔 dt_{nn} が同じであり，交差した位置にあるグラフは各平均クエリ到着間隔を組み合わせた時の結果を示す)

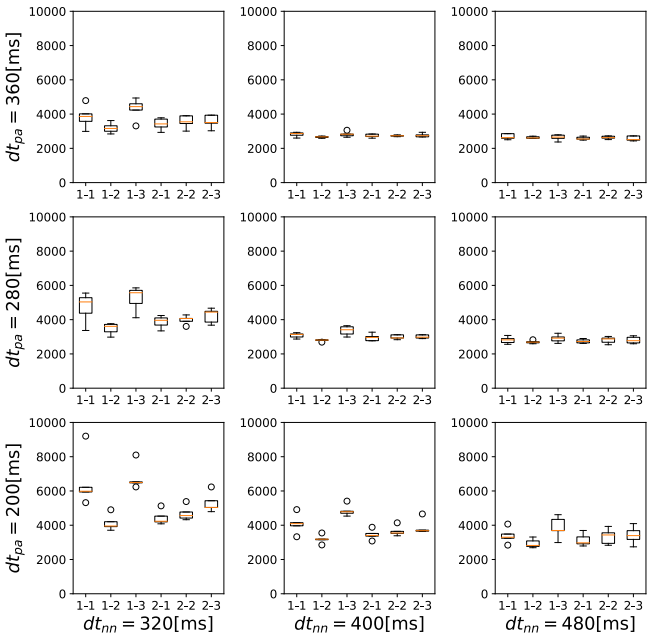

図 10　スケジューリング手順とスケジューリング方式の組み合わせごとに対するアプリケーション 2 の最大レイテンシ [ms] (9 つのグラフは，行方向ではアプリケーション 1 の平均クエリ到着間隔 dt_{pa} が同じであり，列方向ではアプリケーション 2 の平均クエリ到着間隔 dt_{nn} が同じであり，交差した位置にあるグラフは各平均クエリ到着間隔を組み合わせた時の結果を示す)

順とスケジューリング方式をそれぞれ組み合わせて，6 種類の組み合わせをシミュレーションによって比較した．評価実験の結果から，2 段階方式とデッドラインスケジューリングⅡを組み合わせることで，他のスケジューリング手順とスケジューリング方式の組み合わせと比べて，負荷が高い場合における平均レイテンシや最大レイテンシを全体的に低くなることが分かった．今後の課題として，実際に計算機上でアプリケーションを実行して評価することや様々なアプリケーションを対象に評価することが挙げられる．

謝辞　本研究は，JST，CREST，JPMJCR1503 の支援を受けたものである．

文　　献

[1] C. Gentry, “Fully Homomorphic Encryption Using Ideal Lattices,” In Proc. of the 4th Annual ACM Symposium on Theory of Computing, pp. 169–178, 2009.

[2] T. Suzuki, Y. Ishimaki and, H. Yamana, “DAMCREM: Dynamic Allocation Method of Computation REsource to Macro-Tasks for Fully Homomorphic Encryption Applications,” In Proc. of 2020 IEEE Intl. Conference on Smart Computing (BITS2020), pp. 458–463, 2020.

[3] J.H. Cheon, K. Han, A. Kim, M. Kim and Y. Song, “A Full RNS Variant of Approximate Homomorphic Encryption,” Selected Area of Cryptography 2018, LNCS, vol. 11349, pp. 347–368, 2019.

[4] N. P. Smart and F. Vercauteren, “Fully Homomorphic SIMD Operations,” Designs. Codes, and Cryptography, vol. 71, no. 1, pp. 57–81, 2014.

[5] 佐藤　宏樹，馬屋原　昂，石巻　優，山名　早人，“完全準同型暗号による秘密計算回路のループ最適化と最近傍法への適用，” 第 9 回データ工学と情報マネジメントに関するフォーラム，no. H6-3，pp. 1–8，2017.

[6] 佐藤　宏樹，石巻　優，山名　早人，“完全準同型暗号における bootstrap problem 及び relinearize problem の厳密解法の高速化，” 第 9 回データ工学と情報マネジメントに関するフォーラム，no. I5-4，pp. 1–6，2019.

[7] S. Carpov, P. Dubrulle and R. Sirdey, “Armadillo: A compilation chain for privacy preserving applications,” In Proc. of the 3rd Intl. Workshop on Security in Cloud Computing, pp. 13–19, 2015.

[8] F. Boemer, Y. Lao, R. Cammarota and C. Wierzynski, “nGraph-HE: A Graph Compiler for Deep Learning on Homomorphically Encrypted Data,” In Proc. of the 16th ACM Intl Conference on Computing Frontiers, pp. 3–13, 2019.

[9] F. Boemer, A. Costache, R. Cammarota and C. Wierzynski, “nGraph-HE2: A High-Throughput Framework for Neural Network Inference on Encrypted Data,” In Proc. of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography, pp. 45–56, 2019.

[10] R. Dathathri, O. Saarikivi, H. Chen, K. Laine, K. Lauter, S. Maleki, M. Musuvathi and T. Mytkowicz, “CHET: an optimizing compiler for fully-homomorphic neural-network inferencing,” In Proc. of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 142–156, 2019.

[11] R. Dathathri, B. Kostova, O. Saarikivi, W. Dai, K. Laine and M. Musuvathi, “EVA: an encrypted vector arithmetic language and compiler for efficient homomorphic computation,” In Proc. of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 546–561, 2020.

[12] 鈴木　拓也，石巻　優，山名　早人，“メニーコア CPU 環境における準同型暗号演算高速化を目的とするタスクスケジューリング手法の検討，” 研究報告ハイパフォーマンスコンピューティング (HPC)，vol. 2019-HPC-1，no. 28，2019.