Vol.2021-DPS-187 No.12
Vol.2021-MBL-99 No.12
Vol.2021-ITS-85 No.12
2021/5/27

エッジコンピューティング環境での顔認識アプリケーションを対象とした負荷分散機能の実現と評価

永元 陽一[1]　横山 和俊[1]

概要：近年，普及が進むスマートフォンや IoT 機器などのモバイルデバイスには，処理能力が低いという問題がある．そのため，負荷分散を目的としたエッジコンピューティングが注目されている．エッジコンピューティングでは，クライアントやクラウドの処理の一部を，エンドユーザの近くに設置したエッジサーバへ移動させることで，クライアントの負荷を軽減すると共に，通信遅延を削除することができる．エッジコンピューティング環境での負荷分散の取り組みとして，顔認識アプリケーションを対象とした負荷分散手法が提案されている．この手法では，CPU 使用率やネットワークの負荷状況などのパラメータから，実行時間が最小となるタスク配置を計算し，負荷分散している．しかし，パラメータ収集に時間がかかるという問題がある．また，複数クライアントからのアクセスがある場合，実行時間の見積もり精度が悪くなる．本研究では，前回の実行結果から実行時間を推定するために必要なパラメータを取得することで，パラメータ収集に要する時間を短縮する方法を提案する．

Implementation and Evaluation of Load Balancing Function for Face Recognition Applications in Edge Computing Environments

Abstract: Mobile devices such as smartphones and IoT devices, which have become increasing popular in recent years, have the problem of low processing power. Therefore, edge computing for load balancing has been attracting attention. Edge computing reduces the load on the client and communication latency by moving some of the processing in the client and cloud to the edge server located near the end user. As an approach to load balancing in edge computing environments, a load balancing method for face recognition applications has been proposed. In this method, the load is distribution by calculating the task placement that minimizes the execution time based on parameters such as CPU utilization and network load. However, there is a problem that it takes long time to collect parameters. In addition, the accuracy of the execution time estimate becomes worse when there are accesses from multiple clients. In this study, we propose a method to reduce the time required for parameter collection by obtaining the parameters necessary to estimate the execution time from the results of the previous execution.

1. はじめに

近年，スマートフォンや IoT 機器の普及によって，IoT アプリケーションが増加している．また，顔認識技術の普及も進んでおり，空港のゲートで使われるなど顔認識アプリケーションが増加している．しかし，スマートフォンや IoT 機器などのモバイルデバイスには，処理能力やネットワーク帯域幅といった制約がある．そのため，負荷分散を目的としてエッジコンピューティングが注目されている．エッジコンピューティングでは，クライアントやクラウドの処理の一部を，エンドユーザの近くに設置したエッジサーバへ移動させることで，アプリケーションの遅延軽減や計算資源の負荷を軽減することができる [1]．アプリケーションを複数のタスクに分割し，最適な計算資源でタスクを実行するタスク配置について，顔認識アプリケーションを対象とした手法が提案されている [2]．この手法では，計

[1] 高知工科大学
Kochi University of Technology, Kami-city, Kochi 782–8502, Japan

算資源やネットワークの負荷状況などのパラメータから実行時間が最小となる組み合わせを計算し，タスク配置を行うことで負荷分散を行っている．既存手法では，時間のかかるパラメータ収集をクライアントのアクセスごとに毎回行うため，不必要な帯域幅の圧迫が発生する可能性がある．本研究では，実行時間の推定に必要なパラメータ収集時間を削減し，かつ，推定実行時間の精度を改善する手法を提案する．

2. 関連研究

モバイルインターネットや IoT の発展に伴って，物理空間に存在する多くの物理オブジェクトがインターネットからアクセスされるようになっている．顔は，バイオメトリクスに基づく非 ID 技術の識別子として信頼性が高く，物理空間とサイバー空間におけるマッピングの整合性を確保するために使用されている．顔認識に基づくアプリケーションの増加に伴って，コンピュータの計算能力，通信能力，ストレージ能力に対する要求はますます高まっている．これらの問題を解決するために，フォグコンピューティングに基づいた顔識別・解決手法を提案している [3]．具体的には，顔認識処理を複数のタスクに分割し，特定の計算資源に配置して実行する顔認識アプリケーションを提案している．タスクを分散配置することによって，ネットワーク通信量，アプリケーションの応答時間を短縮することが出来ており，フォグコンピューティング環境で顔認識アプリケーションを実行することの有用性を示している．しかし，分割したタスクの実行場所が固定であるため，計算資源やネットワークの負荷状況が考慮できていないという問題がある．

この問題に対し，動的にタスク配置を決定する手法が提案されている [2]．具体的には，各計算資源の CPU 使用率，ネットワーク帯域幅，ネットワーク遅延をを収集し，これらの値から計算した推定実行時間が最小となる組み合わせでタスク配置を行う．しかし，既存研究では，負荷が高くなった場合の推定実行時間の精度に問題がある．また、実行時間の推定に必要な，CPU 使用率，ネットワーク帯域幅，ネットワーク遅延などのパラメータの収集を，クライアントがアプリケーションを実行するごとに毎回行う．これらパラメータの収集には時間がかかるため，パラメータ収集の影響でアプリケーションの総実行時間が大きくなる可能性がある．本研究では，実行時間の推定に必要なパラメータ収集の時間を削減し，かつ，推定実行時間の精度を改善する手法を提案する．

3. エッジコンピューティングモデル

3.1 顔認識アプリケーション

本研究で使用した顔認識アプリケーションを図 1 に示す．既存研究 [2] を参考に，表 1 に示すパッケージを用いて LBP 特徴量を使用する顔認識アプリケーションを作成した．処理の流れとしては，まず顔認識に使用する顔画像を取得する．この画像に対して OpenCV ライブラリ [4] に含まれる Haar-Like 分類器を適用し，顔領域を特定して顔のみの画像にする．この顔のみの画像に対し前処理として，グレースケール化を行った後，ヒストグラム平坦化を行う．この画像に対して，OpenCV ライブラリに含まれる LBP 特徴分類器を用いて LBP 特徴量を抽出し，予め学習しておいたデータとマッチングを行う．顔画像のサンプルデータとして，Yale Face Database を使用した．Yale Face Database には，1 人につき 11 枚の顔画像が 15 人分用意されている．本研究では，この 11 枚のうち 10 枚を学習用とし，1 枚をテスト用として使用した．

計算資源にタスク配置するために，このアプリケーションにおける顔認証プロセスを，顔検出，前処理，特徴量抽出&マッチング処理の 3 つのタスクに分割した．顔画像取得タスクについては，クライアントでのみ実行可能な処理としてタスク配置の対象としない．

3.2 タスク配置モデル

本研究で想定したタスク配置モデルを図 2 に示す．顔認識アプリケーションを実行するために，エッジサーバとクラウドサーバ上にタスク実行システムを構築した．このシステム上に顔認識アプリケーションをタスクを起動させることでタスク配置された計算資源でアプリケーションを実行する．制約として，処理負荷が大きい特徴量抽出&マッチング処理はクライアントに配置しない．

計算資源に配置されたタスクを実行するために，表 2 に示す構成でタスク実行システムを構築した．アプリケーション実行の流れを図 3 に示す．タスク実行システムでは，データベースを用いて計算機に配置されたタスク情報を管理している．具体的には，クライアント ID，タスク ID，次のタスク ID，次のタスクの実行場所を，クライアン

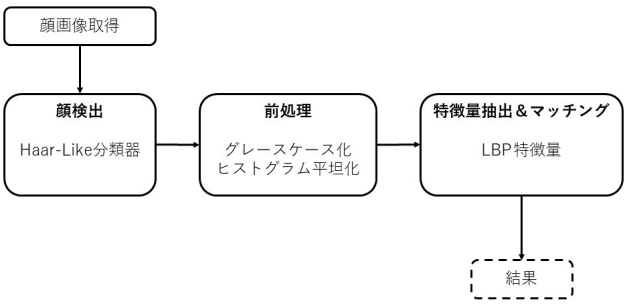

図 1 顔認識アプリケーション

表 1 顔認識アプリケーションの構築に用いたパッケージ等

	パッケージ名	バージョン
使用言語	Python	3.6 ～
ライブラリ	OpenCV	4.5.0
サンプルデータ	Yale Face Database	

Vol.2021-DPS-187 No.12
Vol.2021-MBL-99 No.12
Vol.2021-ITS-85 No.12
2021/5/27

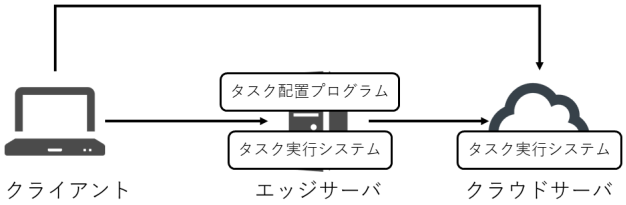

図 2 タスク配置モデル

表 2 タスク実行システムの構築に用いたパッケージ等

	パッケージ名	バージョン
使用言語	Python	3.6 ～
フレームワーク	Django	3.1.3
Web サーバ接続モジュール	mod wsgi	
Web サーバ	Apache	2.4
データベース	MariaDB	15.1

ト ID とタスク ID の組み合わせをプライマリーキーとして管理している．このシステムにタスク配置を行う際は，クライアント ID，タスク ID，次のタスク ID，次のタスクの実行場所の情報を，タスク配置システムへ送信することでデータベースにタスク情報が追加される．クライアント ID は，タスク配置要求がされた際のセッション ID を使用し，セッションは 1 分間で終了するようにしている．また，タスク実行システムでは，タスク完了時にそのタスクに関する情報をデータベースから削除するようにした．これにより，クライアント ID の重複が回避できると考えられる．

アプリケーション実行の流れとして，まずクライアントがエッジ―サーバに対してタスク配置要求を行う．エッジサーバは，この要求に対してタスク配置プログラムを実行してタスク配置先を決定し，タスク配置を行う．タスク配置が完了すると，クライアントに実行先を返却する．クライアントは，指示された計算資源でアプリケーションを実行する．

4. 提案手法

4.1 タスク配置アルゴリズム

本手法では，既存手法 [2] を以下の点で改善する．

(1) パラメータ収集にかかる時間を短縮するために前回の実行結果から求めたパラメータを使用する．
(2) パラメータの計算はクライアントが行い，エッジサーバで管理を行う．

各計算資源はタスクを実行する際に，自身の処理時間と以降の実行時間を記録し実行結果に追加して返却する．クライアントはアプリケーション実行後，実行結果から単位データ量あたりの処理時間と単位時間あたりのデータ転送速度を求めて，エッジサーバに送信することでパラメータ

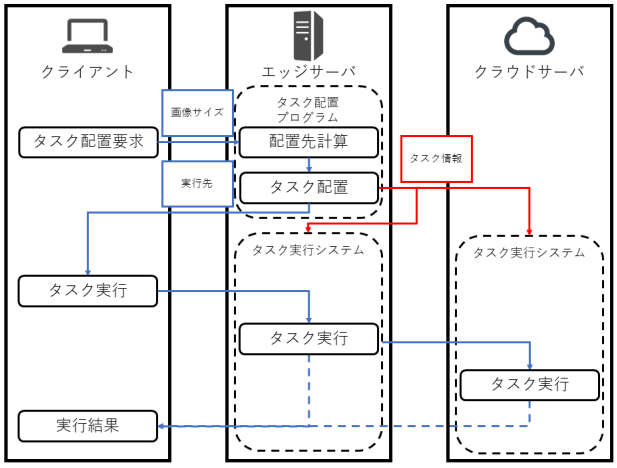

図 3 アプリケーション実行フロー

の更新を行う．

式 (1) は，あるタスクにおける処理する画像データサイズである．S はサイズ，A はタスク，ω はデータ圧縮率を表している．既存手法と同じように，予め各タスクのデータ圧縮率を調べ，データサイズとデータ圧縮率を乗算することで求める．計算資源 R_{A_i} で A_i のタスクを実行した場合の単位データ量あたりの処理時間 $P_r(A_i, R_{A_i})$ は，データサイズを S_{A_i} とすると，式 (2) のようにを用いてタスクの実行時間 $T_r(R_{A_i})$ と処理したデータサイズの除算で求める．計算資源 $A_{R_{A_{i-1}}}$ から R_{A_i} までの単位時間あたりのデータ転送速度 $P_t(R_{A_{i-1}}, R_{A_i})$ は，転送時間を $T_t(R_{A_{i-1}}, R_{A_i})$ とすると，式 (3) のように転送したデータサイズと転送時間の除算で求める．

$$S_{A_i} = S_{A_{i-1}} \cdot \omega_s(A_{i-1}) \tag{1}$$

$$P_r(A_i, R_{A_i}) = \frac{T_r(R_{A_i})}{S_{A_i}} \tag{2}$$

$$P_t(R_{A_{i-1}}, R_{A_i}) = \frac{S_{A_i}}{T_t(R_{A_{i-1}}, R_{A_i})} \tag{3}$$

計算資源 A_i における推定タスク実行時間 T_{exe} の計算は，式 (4) のようにデータ圧縮率を考慮したデータサイズ $S(A_i)$ と，計算資源 R_{A_i} におけるタスク A_i の単位データ量あたりの実行時間 $P_r(A_i, R_{A_i})$ の乗算で求める．計算資源 $R_{A_{i-1}}$ から計算資源 R_{A_i} への推定転送時間 T_{trans} の計算は，式 (5) のようにデータ圧縮率を考慮したデータサイズ $S(A_i)$ と単位時間あたりの転送速度 $P_t(R_{A_{i-1},A_i})$ の除算で求める．この推定タスク実行時間と，推定転送時間を合わせた推定実行時間が最小となる組み合わせを求めために，全ての組み合わせの推定実行時間を計算し比較する．

$$T_{exe} = \sum_{i=1}^{n} S(A_i) \cdot P_r(A_i, R_{A_i}) \tag{4}$$

$$T_{trans} = \sum_{i-1}^{n} \frac{S(A_i)}{P_t(R_{A_{i-1}}, R_{A_i})} \tag{5}$$

4.2 推定例

4.2.1 パラメータの計算

表 3 のように顔検出タスクをクライアント，前処理タスクをエッジ，特徴量抽出& マッチング処理をクラウドで行った場合の推定例を示す．このとき，クライアントにおける顔検出タスクの単位データサイズあたりの実行時間は，処理の実行時間を処理したデータ量で割り 0.04[s/MB] となる．全ての計算資源にタスク配置を行ってアプリケーションを実行した場合，クライアント–クラウド間でデータ転送を行わない．そのため，クライアント–エッジ間とエッジ–クラウド間のデータ転送結果から，クライアント–クラウド間の単位時間あたりのデータ転送速度 $P_t(client, cloud)$ を，式 (6) をのようにして推定する．表 3 の場合，クライアント–エッジ間のデータ転送時間は，クライアントで処理した顔検出タスク以降にかかった時間 1.205[s] から，エッジで処理した前処理タスクの実行時間 0.005[s] と，前処理タスク以降の実行時間 0.7[s] を引いて 2 で割り 0.25[s] となる．エッジ–クラウド間のデータ転送時間は，エッジで処理した前処理タスク以降にかかった時間 0.7[s] から，クラウドで処理した特徴量抽出&マッチング処理にかかった時間 0.3[s] を引いて 2 で割り 0.2[s] となる．よって，この場合のクライアント–クラウド間の単位時間あたりの推定データ転送速度は，式 (6) から 0.22[MB/s] となる．このようにして計算した表 3 の場合のパラメータを，表 4 に示す．

$$P_t(client, cloud) = \frac{1}{\frac{T_t(client,edge)}{S_{A_2}} + \frac{T_t(edge,cloud)}{S_{A_3}}} \quad (6)$$

4.2.2 実行時間の推定

顔認識を行う画像データのサイズを 0.7[MB] とし，推定のためのパラメータを 4 とした場合の実行時間の推定を示す．顔検出と前処理をクライアントに，特徴量抽出&マッチング処理をエッジに配置する場合の推定実行時間は，それぞれのタスクの実行時間を処理するデータサイズとパラメータから求めた推定タスク実行時間と，転送するデータサイズとパラメータから求めた推定転送時間の和で求める．推定タスク実行時間は，それぞれデータサイズとパラメータの積から 0.028[s]，0.007[s]，0.42[s] となる．推定転送時間は，転送を行うのがクライアント–エッジ間のみなので，転送するデータサイズとパラメータの商から 0.105[s] となる．よって，顔検出タスクと前処理タスクをクライアントに，特徴量抽出&マッチング処理をエッジに配置する場合の推定実行時間は，これらを足し合わせて 0.56[s] となる．このようにして全ての組み合わせの推定実行時間を求め，実行時間が最小となる配置先を決定する．

表 3 実行例

タスク名	実行場所	タスク実行時間 [s]	以降の処理にかかった時間 [s]
顔検出	クライアント	0.02	1.205
前処理	エッジ	0.005	0.7
特徴量抽出&マッチング	クラウド	0.3	0
データサイズ [MB]		0.5	
データ圧縮率			
顔検出から前処理		前処理から特徴量抽出	
0.4		0.3	

表 4 パラメータ

実行時間のパラメータ		転送時間のパラメータ	
顔認識	0.04[s/MB]	クライアント–エッジ	0.8[MB/s]
前処理	0.025[s/MB]	エッジ–クラウド	0.3[MB/s]
特徴量抽出&マッチング	5[s/MB]	クライアント–クラウド	0.22[MB/s]

5. 評価

5.1 評価環境

表 5 に評価で用いたコンピュータを示す．クライアント端末として，Raspberry Pi 4 Model B を評価用と負荷用に 2 つ使用した．エッジサーバには，intel core i5 を搭載したコンピュータを，クラウドサーバには，AWS EC2 の t2.xlarge インスタンスを比較のため東京とオレゴンの 2 つ使用した．

5.2 評価方法

負荷用クライアントから 1 秒ごとに 2，4，8 アクセスを行った状態で，評価用クライアントから負荷ごとに 30 回アクセスを行う．この時評価用クライアントは，アプリケーション総実行時間，タスク配置のオーバーヘッド，顔認識にかかった時間，推定実行時間を記録する．負荷用クライアントでは，1 秒ごとに決まった数のリクエスト非同期で実行する．また，既存手法と提案手法ではタスク配置先を計算する場所が異なるため，今回は提案手法の有用性を確認するために，既存手法のタスク配置先決定はエッジサーバで行うこととした．評価内容を示す．

(1) アプリケーション総実行時間
アクセス数が増加した際に，どの程度アプリケーション総実行時間が増加するのか確認するために，アクセス数ごとのアプリケーション総実行時間を比較する．

(2) タスク配置のオーバーヘッド
アクセス数が増加した際に，既存手法と提案手法のタスク配置のオーバーヘッドの変化を確認するために，アクセス数ごとのタスク配置のオーバヘッドを比較する．

(3) 顔認識の処理時間
アクセス数が増加したことによるアプリケーションへの影響を確認するために，アクセス数ごとの顔認識の実行時間を比較する．

表 5　評価環境

計算資源	プロセッサ	CPU クロック	コア数	メモリ
クライアント	ARM Cortex-A72	1.50 GHz	4	1.8 GB
エッジサーバ	Intel(R) Core(TM) i5-9400	2.90 GHz	6	7.6 GB
クラウドサーバ	Intel Xeon プロセッサ	3.3 GHz	4	16 GB

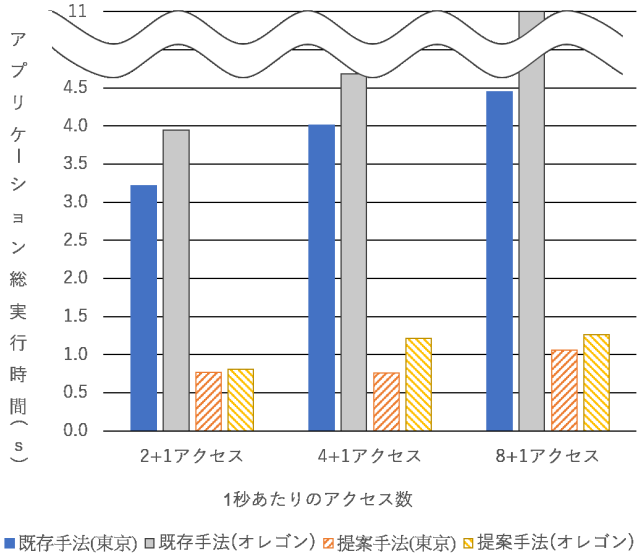

図 4　アプリケーション総実行時間

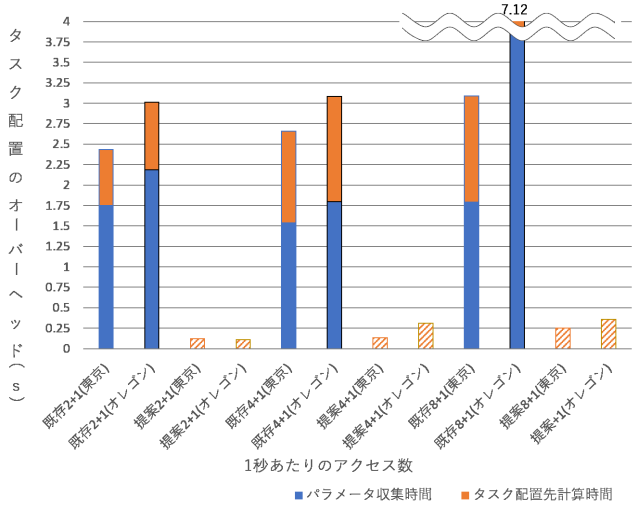

図 5　タスク配置のオーバーヘッド

(4) 推定実行時間の精度
実行時間の推定がどの程度正しいか確認するために，アクセス数ごとの推定実行時間と実際の実行時間の差を比較する．

5.3　評価結果と考察

5.3.1　アプリケーション総実行時間

アクセス数が増加した際のアプリケーション総実行時間の増加を確認するために，アクセス数ごとのアプリケーション総実行時間を比較した結果を図 4 に示す．アクセス数が増加するにつれて，既存手法と提案手法の両手法でアプリケーションの総実行時間が増加している．既存手法と提案手法を比べると，既存手法の方がアプリケーションの総実行時間が大きく，アクセス数が増加した場合も既存手法の方が大きく増加している．アクセス数を 10+1 とした場合の評価を試みたが，既存手法ではパラメータ収集に使用するツールでエラーが頻発し，正確に測定することが出来なかった．提案手法においては，エッジサーバでタスクの偏りが生じサーバダウンが発生した．

5.3.2　タスク配置のオーバーヘッド

アクセス数が増加したことによるアプリケーションへの影響を確認するために，アクセス数ごとの顔認識の実行時間を比較した結果を図 5 に示す．既存手法において，パラメータ収集に使用するツールの実行時間はできるだけ短くし，測定に失敗した場合は再測定を行うようにしている．

図 5 からわかるように，既存手法はパラメータ収集にかかる時間が大きいことが確認できる．CPU 使用率の測定が計算資源への負荷になるとは考えにくく，ネットワーク遅延を測定するツールが帯域幅を圧迫するとは考えにくいため，既存手法においてパラメータ収集時間が増大する原因は，ネットワーク帯域幅を測定するツールであると考えられる．また，図 5 から提案手法と比べて既存手法の方がタスク配置にかかる時間が大きいことが確認できる．推定実行時間の計算量は，既存手法と提案手法で変わらないため，タスク配置においても帯域幅の圧迫より処理時間が増加していると考えられる．

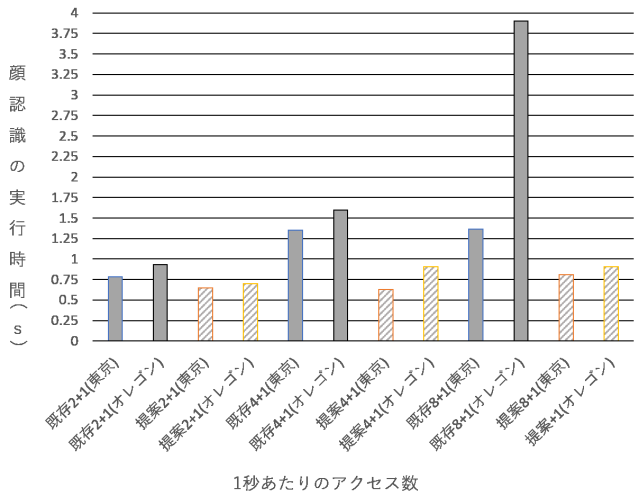

図 6　顔認識の実行時間

Vol.2021-DPS-187 No.12
Vol.2021-MBL-99 No.12
Vol.2021-ITS-85 No.12
2021/5/27

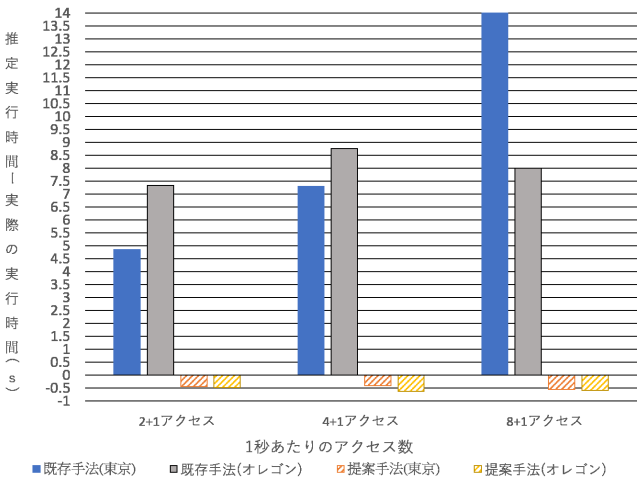

図 7 推定実行時間と実際の実行時間の差

5.3.3 顔認識の実行時間

アクセス数が増加したことによるアプリケーションへの影響を確認するために，アクセス数ごとの顔認識の実行時間を比較した結果が図 6 である．既存手法よりも提案手法の方が，アクセス数が増加しても顔認識の実行時間が小さいことが確認できる．提案手法よりも既存手法の方が，リアルタイムで計算資源やネットワークの負荷状況を測定しており，アクセス数の少ない状況では両手法に大きな差が見られないことから，既存手法の方が顔認識の実行時間が大きくなる原因は，パラメータ収集により帯域幅が圧迫されることでデータ転送にかかる時間が大きくなることであると考えられる．

5.3.4 推定実行時間と実際の実行時間の差

実行時間の推定がどの程度正しいか確認するために，アクセス数ごとの推定実行時間と実際の実行時間の差を比較した結果が図 7 である．既存手法は，アクセス数が増加し計算資源への負荷が増加するにつれて推定実行時間が大きくなり，推定値としては信頼できないほど実際の実行時間との差が大きくなっている．提案手法では，アクセス数増加による推定実行時間の増加が大きくないものの，実際の実行時間よりも小さい．これは，提案手法において転送時間の推定を顔認識に使用する画像データサイズで行うため，実際に転送しているデータよりも小さいからであると考えられる．

6. おわりに

本研究では，エッジコンピューティング環境における顔認識アプリケーションの負荷分散法を提案した．提案手法では，前回の実行結果から推定実行時間の計算に使用するパラメータを求め，実行時間を推定する．タスク配置を決定する際には，タスク配置の全ての組み合わせについて，実行時間を推定し，推定実行時間が最も短くなる配置を選択する．また，前回の実行結果から実行時間を推定するため，パラメータ収集にかかる時間を短縮する．既存研究との比較では．アプリケーションの総実行時間，顔認識のみの実行時間ともに提案手法の方が小さいことが確認できた．

参考文献

[1] 総務省，“5G 時代に向けての各レイヤーの動向”，<https://www.soumu.go.jp/johotsusintokei/whitepaper/ja/r02/html/nd114220.html>，(参照 2021 年 2 月 24 日).
[2] 佐竹颯太，谷遼太郎，重野寛，“エッジコンピューティングにおける顔認識アプリケーションのためのタスク配置システムの提案”，マルチメディア，分散，協調とモバイルシンポジウム 2019 論文集，pp.1190–1195 (2019).
[3] Pengfei Hu，Huansheng Ning，Tie Qiu，Yanfei Zhang，Xiong Luo，“Fog Computing Based Face Identification and Resolution Scheme in Internet of Things”，IEEE Transactions on Industrial Informatics，pp.1910–1920 (2017).
[4] Intel Corporation，“OpenCV（Open Source Computer Vision Library）”，<https://github.com/opencv>，(参照 2021 年 2 月 24 日).