「マルチメディア，分散，協調とモバイル
(DICOMO2019)シンポジウム」 令和元年7月

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

佐竹 颯太[1]　谷 遼太郎[1]　重野 寛[1]

概要：現在，モバイルデバイスは処理能力など様々な制約があり，アプリケーションの実行における負荷軽減が必要とされる．エッジコンピューティングではモバイルデバイスの近くにエッジサーバを配置し，アプリケーション実行を移行することにより，負荷軽減と遅延削減をする．また，顔認識アプリケーションの増加に伴い，計算能力，ストレージ容量の要求が高まっている．既存研究のエッジコンピューティングベースの顔認識ではタスクの実行場所が決まっており，計算資源やネットワークの状態を考慮されておらず，タスク配置手法が必要である．そこで，本稿では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タスク配置システムを提案する．本稿で提案するタスク配置システムでは計算資源やネットワーク状況を考慮するタスク配置決定式に基づいて推定応答時間を算出する．そして，推定応答時間が最小となるタスク配置を決定し，各タスクを各計算資源へ配置する．本稿で提案する顔認識アプリケーションのタスク配置システムのプロトタイプを実装して実験を行い，動作確認と評価を行った．

Proposal of Task Placement System for Face Recognition Application in Edge Computing

HAYATA SATAKE[1]　RYOTARO TANI[1]　HIROSHI SHIGENO[1]

1. はじめに

現在，スマートフォンやIoTデバイスなどのモバイルデバイスはバッテリやストレージ容量，ネットワークの帯域幅やプロセッサの処理能力など様々な制約がある [1]．そこで，モバイルデバイスにおけるアプリケーション処理の負荷軽減が必要とされる．負荷軽減手法としてエッジコンピューティング [2] が注目されている．エッジコンピューティングは，エンドユーザの近くにクラウドサーバと同様の役割を持つエッジサーバを分散配置する．エンドユーザとサーバ間の距離の短縮により，遅延を削減することができる．また，クラウドサーバの処理の一部をエッジサーバへ移行することにより，アプリケーション処理にかかる応答時間を削減することができる．

IoTデバイスの増加に伴い，顔画像収集が容易になり，顔認識アプリケーションが増加している．顔認識アプリケーションは高負荷な処理であり，計算能力，ストレージ容量の要求が益々高まっている．そこで，分散処理が必要となる．既存研究としてはクライアント―クラウド間にエッジサーバを配置し，アプリケーションを複数タスクに分割してタスクの一部をエッジサーバで実行するエッジコンピューティングベースの顔認識 [3] が提案されている．エッジコンピューティングベースの顔認識はデータ量をエッジサーバ上で縮小することによって，ネットワーク伝送量を削減し，効率的なアプリケーションの実行を実現する．しかし，既存研究ではタスクの実行場所があらかじめ決定しており，計算資源やネットワークの状態について考慮されておらず，計算資源やネットワークの状態に基づいてタスクを配置する手法が必要である．

本稿ではエッジコンピューティングにおける計算資源やネットワークの状態を考慮した顔認識アプリケーションのタスク配置システムを提案する．提案システムではクライアント，エッジ，クラウドの3層環境を想定しており，クライアントからのリクエストに応じて，各計算資源は推定実行時間を算出する．算出結果に基づいてクライアントは計

[1] 慶應義塾大学大学院理工学研究科
Graduate School of Science and Technology, Keio University,Yokohama, Kanagawa, 223-8522, Japan

算資源やネットワーク状態を考慮したタスク配置決定式に基づいて推定応答時間を算出する．そして，推定応答時間が最小となるタスク配置を決定し，各タスクを各計算資源へ配置する．推定応答時間はアプリケーションの推定実行時間とデータやタスク転送に掛かる推定転送時間のタスク数の総和で表される．計算資源やネットワークの負荷をパラメータ化し，タスク配置決定式に反映させることで計算資源やネットワークの状況を考慮したタスク配置決定式に基づく顔認識アプリケーションのタスク配置を実現する．

2. 関連研究

本章では，エッジコンピューティング，タスク配置，顔認識アプリケーションの分散処理の関連研究について述べ，これらから考えられる既存の分散処理における顔認識アプリケーションの分割実行手法の問題点を抽出する．

2.1 エッジコンピューティング

モバイルデバイスでのアプリケーション実行はプロセッサの処理能力の低さ，ストレージ容量や電力消費により制限される．そこで，モバイルデバイスにおけるアプリケーション処理の負荷軽減が必要とされる．負荷軽減手法としてエッジコンピューティングが注目されている．図 1 にエッジコンピューティングの概要の図を示す．エッジコンピューティングの特徴はエンドユーザの近くにクラウドサーバと同様の役割を持つエッジサーバを分散配置する．エンドユーザとサーバ間の距離を短縮することにより，遅延を削減することができる．また，クラウドサーバの処理をエッジサーバへ移行することにより，アプリケーション処理にかかる応答時間を削減することができる．

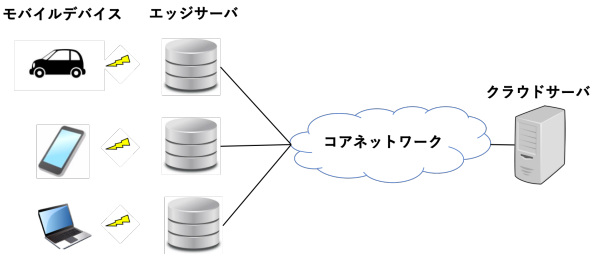

図 1 エッジコンピューティングの概要

2.2 タスク配置

アプリケーションを粒度の小さいタスクに分割し，タスクを効率的に実行できるサーバを選択し，配置するタスク配置がある．文献 [4] ではネットワーク情報を集約するコントローラがモバイルデバイスの電力消費を考慮してアプリケーション実行の遅延を最小化するシステムを提案されている．タスク配置問題を NP 困難な混合整数非線形問題として定義し，最適化問題を解くことにより遅延の最小化をする．タスク配置問題を 2 つの副問題であるタスク移行問題と資源割り当て問題に変換する．2 つの副問題を定式化し，解を導出する．解に基づいて，コントローラがタスク移行と資源割り当てすることにより，効率的なアプリケーション実行ができるタスク配置を実現する．

2.3 エッジコンピューティングベースの顔認識

既存の顔認識アプリケーションの負荷軽減手法としてエッジコンピューティングベースの顔認識 [3] が提案されている．クライアント—クラウド間にエッジサーバを配置し，クライアントやクラウド上で全てが完了していたアプリケーションを複数タスクに分割する．その分割したタスクをエッジサーバに分散配置する．データ量をエッジサーバ上で縮小することによって，ネットワーク伝送量を削減し，効率的なアプリケーションの実行を実現する．エッジコンピューティングベースの顔認識ではアプリケーションの分割実行を行うことによって，クラウドへの負荷を削減したアプリケーションの実行ができる．

既存のエッジコンピューティングベースの顔認識において，アプリケーションを分割したタスクの実行場所がそれぞれ固定されており，各計算資源の計算能力や負荷，ネットワークの状況について考慮されていないという問題点がある．また，アプリケーションの応答時間についての評価もされておらず，各計算資源の計算能力や負荷，ネットワークの状況によっては応答時間が大幅に増大してしまう可能性が考えられる．そのため，従来の顔認識アプリケーション分割実行手法において考慮や評価がなされていない，各計算資源の計算能力や負荷，ネットワークの状況を考慮し，アプリケーションの応答時間を最小とするタスク配置を行う手法が必要であると考えられる．

3. 提案手法

本章では，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タスク配置システムについて提案する．

3.1 タスク配置システムの概要

本稿で提案する顔認識アプリケーションのタスク配置システムでは，各計算資源において自計算資源でタスクを実行する場合の推定のタスク実行時間を算出する．その結果をクライアントのコントローラが集約し，タスク配置決定式に基づいて推定応答時間が最小となるタスク配置を決定する．クライアントのコントローラは各タスクを計算資源に配置し，実行を命令する．推定応答時間の算出は推定実行時間と推定転送時間の総和で算出する．推定実行時間は各計算資源の計算能力や負荷などの計算資源の状態や各タスクにおける入力データサイズに基づき算出する．推定転送時間は各タスクにおける入力データサイズ，帯域幅や遅延などのネットワークの状態に基づき算出する．

3.2 想定環境

本稿で実装するタスク配置システムではクライアント—エッジ—クラウドの 3 層環境を想定している．図 2 は本稿で提案するタスク配置システムの想定環境を示す．Tier 1 はクライアント，Tier 2 はエッジサーバ，Tier 3 はクラウドサーバで構成される．クライアントはアプリケーションとアプリケーションで処理されるデータを保持する．クラウドは特徴量を格納しているデータベースと接続する．

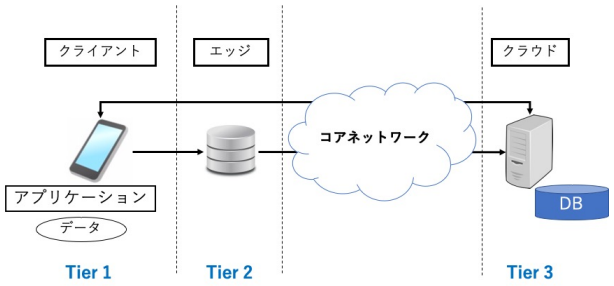

図 2　タスク配置システムの想定環境

3.3 タスク配置システムの構成

タスク配置システムにおいて，各計算資源はコントローラを持つ．クライアントは Client Controller，エッジは Edge Controller，クラウドは Cloud Controller である．図 3 はタスク配置システムの構成コンポーネントを示す．各計算資源のコントローラはリクエスト送受信やタスク実行などを行うコンポーネントで構成される．

Client Controller はタスク配置システムを主導で動かし，エッジ，クラウドに Request を送信することでタスク配置手順を開始する．また，各計算資源で各タスクを実行するのに掛かる推定実行時間の算出結果に基づき，推定応答時間を算出し，最小の推定応答時間となるタスク配置における計算資源へとタスクやデータを転送することや自計算資源でタスクを実行する役割を担う．

Edge Controller，Cloud Controller はリクエストの受信や応答の転送，推定実行時間の算出などのタスク配置を行うまでの処理やタスクとデータ転送，実行結果の応答転送などの実際にタスク配置を行う際の処理や自計算資源でタスクを実行する役割を担う．また，Cloud Controller はデータベースと接続し，マッチング処理も行う．

3.4 タスク配置決定式

推定応答時間 $T(R_{A_i}, S_{A_i})$ の算出はデータやタスク転送に掛かる推定転送時間とアプリケーションの処理に掛かる推定実行時間の和で表す．A_i は実行するタスク，R_{A_i} はタスク A_i を実行する計算資源であり，S_{A_i} はタスク A_i への入力データサイズである．$T(R_{A_i}, S_{A_i})$ は式 (1) で表す．

$$T(R_{A_i}, S_{A_i}) = T_{exe}(R_{A_i}, S_{A_i}) + T_{tr}(R_{A_i}, S_{A_i}) \quad (1)$$

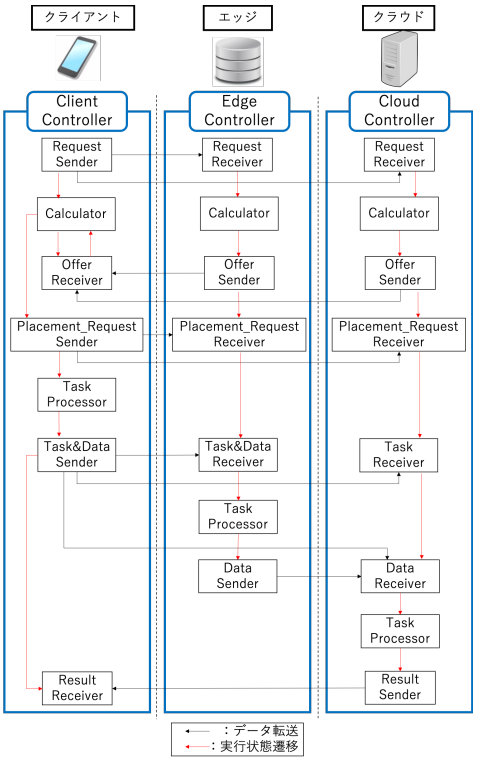

図 3　タスク配置システムの構成コンポーネント

$T_{exe}(R_{A_i}, S_{A_i})$ は計算資源 R_{A_i} において，タスク A_i を実行する際に掛かる推定実行時間の総和を表す．また，$T_{tr}(R_{A_i}, S_{A_i})$ は，クライアント R_{A_0} から計算資源 R_{A_i} へタスク A_i を転送する際に掛かる時間と 1 つ前のタスクを実行した計算資源 $R_{A_{i-1}}$ から計算資源 R_{A_i} へタスク A_i によって処理されるデータを転送する際に掛かる時間の総和を表す．さらに，$T_{exe}(R_{A_i}, S_{A_i})$ は式 (2)，$T_{tr}(R_{A_i}, S_{A_i})$ は式 (3) にそれぞれ表す．

$$T_{exe}(R_{A_i}, S_{A_i}) = \sum_{i=1}^{n} (S_{A_i} \cdot T_{cpu}(R_{A_i})) \quad (2)$$

$$T_{tr}(R_{A_i}, S_{A_i}) = \sum_{i=1}^{n} (T_{task}(R_{A_i}) + T_{data}(R_{A_i}, S_{A_i})) \quad (3)$$

また，タスク A_i への入力データサイズ S_{A_i} を式 (4) で表す．

$$S_{A_i} = S_{A_{i-1}} \cdot w_s(A_{i-1}) \quad (4)$$

$w_s(A_{i-1})$ はタスク A_{i-1} 実行前後のデータの縮小率，$S_{A_{i-1}}$ は 1 つ前のタスク A_{i-1} への入力データサイズである．また，計算資源 R_{A_i} の CPU 使用率から導出されるタスク A_i の処理にかかる推定時間 $T_{cpu}(R_{A_i})$ を式 (5) で表す．

$$T_{cpu}(R_{A_i}) = T_{est}(A_i, R_{A_i}) \cdot \frac{1}{1 - C_{usage}(R_{A_i})} \quad (5)$$

$T_{est}(A_i, R_{A_i})$ は計算資源 R_{A_i} においてアプリケーションが利用可能な CPU 使用率が 100%の時において，タスク A_i がデータ 1Byte を処理する際に掛かる推定時間であり，

$C_{usage}(R_{A_i})$ は計算資源 R_{A_i} における CPU 使用率である．式 (5) において，各計算資源における利用可能な CPU 使用率をパラメータ化することで，すでに負荷が高い計算資源へとタスク配置を行うことを回避する．また，タスクの転送時間 $T_{task}(R_{A_i})$ を式 (6) で表す．

$$T_{task}(R_{A_i}) = \frac{S(A_i)}{B(R_{A_0 \leftrightarrow A_i})} + D(R_{A_0 \leftrightarrow A_i}) \qquad (6)$$

$S(A_i)$ はタスク A_i のサイズ，$B(R_{A_0 \leftrightarrow A_i})$ はクライアント R_{A_0} と計算資源 R_{A_i} 間のネットワークの帯域幅，$D(R_{A_0 \leftrightarrow A_i})$ はクライアント R_{A_0} と計算資源 R_{A_i} 間のネットワークのリンク遅延である．また，処理データの転送時間 $T_{data}(R_{A_i}, S_{A_i})$ を式 (7) で表す．

$$T_{data}(R_{A_i}, S_{A_i}) = \frac{S_{A_i}}{B(R_{A_{i-1} \leftrightarrow A_i})} + D(R_{A_{i-1} \leftrightarrow A_i}) \quad (7)$$

$B(R_{A_{i-1} \leftrightarrow A_i})$ は 1 つ前のタスク A_{i-1} を実行した計算資源 $R_{A_{i-1}}$ とタスク A_i を実行する計算資源 R_{A_i} 間のネットワークの帯域幅，$D(R_{A_0 \leftrightarrow A_i})$ は 1 つ前のタスク A_{i-1} を実行した計算資源 $R_{A_{i-1}}$ とタスク A_i を実行する計算資源 R_{A_i} 間のネットワークのリンク遅延である．式 (6)，式 (7) において，データサイズ，遅延，帯域幅をパラメータ化することで，転送に時間が計算資源へのタスク配置を回避する．

3.5 タスク配置手順

エッジに顔検出タスクと前処理タスク，クラウドに特徴量抽出タスクを配置する場合を例に提案するタスク配置システムのタスク配置手順について述べる．図 4 はエッジに顔検出タスク，前処理タスクを配置し，クラウドに特徴量抽出タスクを配置する場合のタスク配置手順例を示す．

(1) Client Controller はアプリケーションで処理される顔画像のサイズ，顔検出タスク，前処理タスク，特徴量抽出タスクの各タスクにおけるデータサイズの入出力比が格納された Request を Edge Controller と Cloud Controller へ転送する．

(2) Edge Controller と Cloud Controller は Client Controller からの Request を受信し，自計算資源における Request 中のタスクの推定実行時間の算出を行う．算出結果の推定実行時間が格納された Offer を Client Controller へと転送する．

(3) Client Controller は Edge Controller と Cloud Controller からの Offer を待つ間に，自計算資源における各タスクの推定実行時間の算出を行う．Offer を受信すると，すべてのタスク配置パターンにおける推定実行時間の和を算出し，各パターンにタスクを配置する推定転送時間を足し合わせ，推定応答時間を求める．

(4) Client Controller は各タスク配置パターンの推定応答時間の比較を行い，推定応答時間が最小となるタスク配置を決定し，各計算資源で実行するタスク名を格納した Placement_Request を Edge Controller と Cloud Controller に転送する．

(5) Client Controller は Placement_Request を Edge Controller と Cloud Controller に転送後，顔画像と顔検出タスクと前処理タスクを Edge Controller へと転送し，特徴量抽出タスクを Cloud Controller へと転送する．

(6) Placement_Request と元の顔画像と顔検出タスクと前処理タスクを受信した Edge Controller は，顔検出タスクと前処理タスクを実行する．タスク実行による出力データを Cloud Controller へと転送する．

(7) Client Controller から Placement_Request と特徴量抽出タスクを受信し，Edge Controller から出力データを受信した Cloud Controller は，受信した出力データを入力データとして特徴量抽出タスクを実行する．

(8) Cloud Controller は特徴量抽出タスクの実行が終了すると，データベースとマッチングを実行し，得られた結果を Client Controller へと転送する．

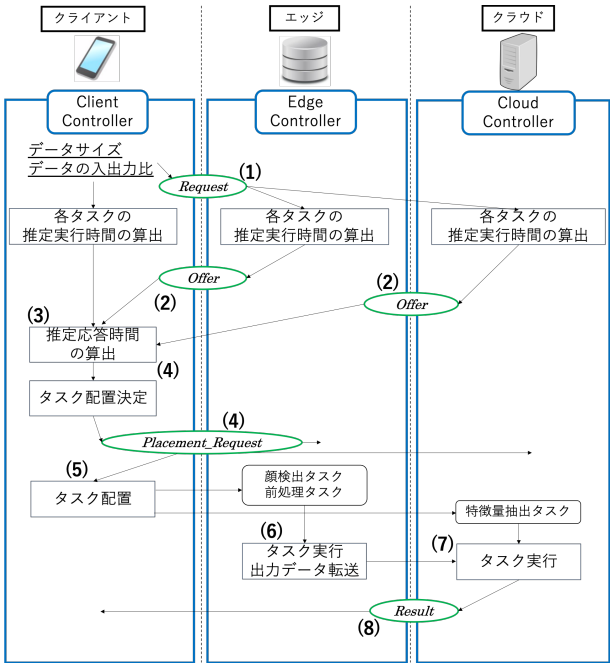

図 4 提案機構におけるタスク配置手順例

4. 実験・評価

本章では提案システムのプロトタイプを用いた実験による動作確認と評価について述べる．

4.1 実験環境

実装したタスク配置システムを検証する実験の実験環境を図 5 に示す．実験では 3 台のコンピュータを計算資源として，それぞれをクライアント，エッジ，クラウドと想定し，図 5 のように有線接続した．実環境における遅延の影響を想定し，クライアント—クラウド間，エッジ—クラウド間に tc コマンド [5] で擬似的にクラウドを想定した遅延を発生させた．実験ではタスク配置システムの顔認識アプリケーションのタスク配置および応答時間を確認する．

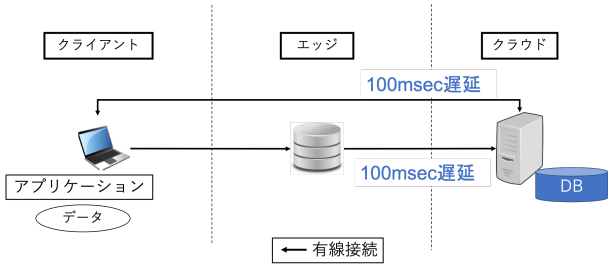

図 5　実験環境

本稿で実装するタスク配置システムにおいて，各コンポーネントの実装には Python [6] を用いた．表 1 にタスク配置システムの実装における使用機材の性能を示す．

表 1　使用機材

	OS	CPU	Memory
クライアント	Ubuntu 16.04.3 LTS	Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz	2GB
エッジ	Ubuntu 16.04.5 LTS	Intel(R) Pentium(R) CPU G4560 @ 3.50GHz	8GB
クラウド	Ubuntu 16.04.5 LTS	Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz	8GB

タスク配置システム上で動作する顔認識アプリケーションを複数タスクに分割し，各タスクを個別に実装した．図 6 にタスク分割とタスク実行における流れとデータの遷移を示す．顔認識アプリケーションは顔画像取得タスク，顔検出タスク，前処理タスク，特徴量抽出タスク，マッチングタスクの 5 つのタスクに分割した．顔画像取得タスクはクライアントで実行され，顔画像データベース中からランダムで 1 枚抽出し，その顔画像をタスク配置システムに入力する．顔検出タスク，前処理タスク，特徴量抽出タスクはクライアントから各計算資源に配置されて実行される．顔検出タスクは OpenCV ライブラリ [7] に含まれている Haar-like 検出器 [8] を用いて顔の領域のみを切り出す．前処理タスクは顔領域のカラー画像を 256 階調のグレースケール化を行うことで特徴量抽出の精度を向上させる．特徴量抽出タスクは uniform LBP 特徴 [9] によって抽出する．uniform LBP を抽出する際は Python ライブラリの scikit-image [10] を利用した．マッチングタスクはクラウド上で実行され，特徴量抽出タスクで抽出した特徴量とデータベースに格納されている特徴量の比較を行い，マッチングの成否をクライアントに返す．

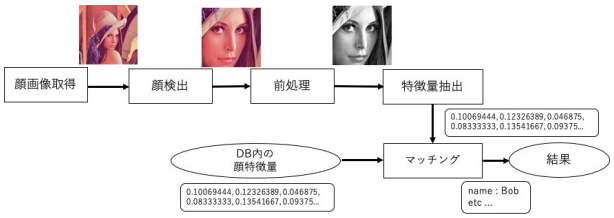

図 6　タスク実行の流れとデータ遷移

4.2 実験パラメータ

本稿の実験では帯域幅を Iperf [11] を用いて測定を行い，実測値を利用した．実験パラメータを表 2 に示す．

表 2　実験パラメータ

クライアント—エッジ間の遅延	0	[msec]
クラウドへの遅延	100	[msec]
データサイズ	300	[KByte]

式 (5) の計算資源 $R(A_i)$ の CPU の使用率 $C_{usage}(R_{A_i})$ は今回の実験では cpulimit [12] を用いることで顔認識アプリケーションの実行に使用可能な CPU 使用率を任意の値に制限し，計算資源に負荷が掛かった状態を再現する．cpulimit で指定した CPU 使用率を CPU_{set} とすると，実験で用いる値 $C_{usage}(R_{A_i})$ をの式 (8) で表す．

$$C_{usage}(R_{A_i}) = \frac{CPU_{set}}{100} \tag{8}$$

CPU_{set} は cpulimit によって制限した CPU 使用率である．taskset コマンド [13] と cpulimit コマンドを組み合わせて，負荷を掛ける計算資源の CPU のコアを指定する．

式 (5) の計算資源 R_{A_i} においてアプリケーションが利用可能な CPU 使用率が 100%の時において，タスク A_i がデータ 1Byte を処理する際に掛かる推定時間 $T_{est}(A_i, R_{A_i})$ は事前に各タスクを各計算資源で実行し，測定したものを使用する．表 3 に $T_{est}(A_i, R_{A_i})$ の値を示す．表 3 より実装環境における各計算資源の計算能力はクラウド，エッジ，クライアントの順で高いものとする．

表 3　$T_{est}(A_i, R_{A_i})$ の値

計算資源	$T_{est}(A_i, R_{A_i})$[msec]		
	顔検出タスク	前処理タスク	特徴量抽出タスク
クライアント	0.0045	0.00016	0.0014
エッジ	0.0041	0.00015	0.0010
クラウド	0.0039	0.00011	0.0008

4.3 動作確認と評価

図 7 はクライアント—エッジ間に遅延がなく，クライアント—クラウド，エッジ—クラウド間の遅延が 100msec 時の推定応答時間と実際の応答時間を示す．各計算資源の CPU 使用率は，クライアント 30%，エッジ 20%，クラウド 20%である．図 7 における (a) のグラフはすべてのタス

クを 1 つの計算資源で固定して実行する場合の推定応答時間と提案手法のタスク配置決定式で算出された最小の推定応答時間を示す．提案手法は顔検出タスクをエッジ，前処理タスクをクラウド，特徴量抽出タスクをクラウドで実行する場合が推定応答時間が最も小さくなるタスク配置であると判断し，以上のタスク配置でタスクを実行することを確認した．図 7 における (b) のグラフはすべてのタスクを 1 つの計算資源で固定して実行する場合の実際の応答時間と提案手法のタスク配置における実際の応答時間を示す．応答時間は顔認識アプリケーションを実行するのに掛かった実行時間と，タスク転送やデータ転送に掛かった転送時間が含まれている．提案のタスク配置における推定応答時間と実際の応答時間の差異は 13.9%であった．図 7 の (b) のグラフの提案手法以外は各計算資源における実際の応答時間を測定するために恣意的に各計算資源において全てのタスクを実行した場合の応答時間を示している．提案のタスク配置でタスクを実行した場合，応答時間は他のどのタスク配置の応答時間よりも小さくなった．

以上より，タスク配置システムでは計算資源の CPU 使用率を考慮したタスク配置決定式による推定応答時間の算出に基づき，推定応答時間が最も小さくなるタスク配置でタスクが実行されることを確認した．また，データやタスクの転送時間が大きくなる転送を避けたタスク配置で提案のタスク配置においてタスクが実行されることも確認した．

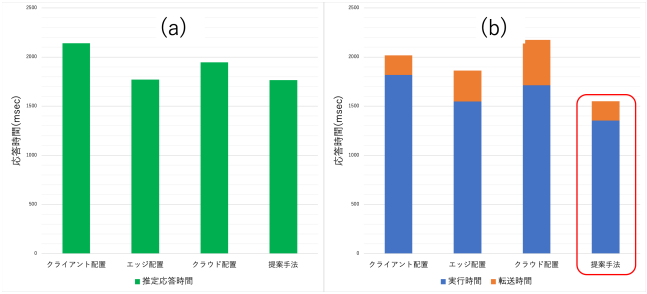

図 7 CPU 使用率がクライアント 30%，エッジ 20%，クラウド 20% の時の (a) 推定応答時間，(b) 実際の応答時間

5. おわりに

本稿はエッジコンピューティングにおける計算資源やネットワーク状態を考慮した顔認識アプリケーションのタスク配置システムを提案した．推定応答時間が最小となるタスク配置を決定し，各計算資源へ配置する．計算資源やネットワーク負荷をパラメータ化し，タスク配置決定式に反映させ，計算資源やネットワーク状態を考慮した顔認識アプリケーションのタスク配置システムを実現した．

本稿における提案システムのプロトタイプを実装して実験を行い，動作確認と評価を行った．実験により，タスク配置決定式による推定応答時間の算出に基づき，推定応答時間が最小となるタスク配置においてタスクが実行されることを確認した．また，推定応答時間が最小となるタスク配置で各タスクを実行した場合の応答時間がすべてのタスク配置パターンの中で最小であった．

以上より，提案のタスク配置システムはネットワークと計算資源の状態を考慮した顔認識アプリケーションのタスク配置が行われるシステムであることを示した．

参考文献

[1] Huber Flores, Pan Hui, Sasu Tarkoma, Yong Li, Srirama, Satish Narayana and Rajkumar Buyya: Mobile code offloading: from concept to practice and beyond, *IEEE COMMUNICATIONS MAGAZINE*, Vol. 53, No. 3, pp. 80-88 (2015).

[2] Blesson Varghese, Nan Wang, Sakil Barbhuiya, Peter Kilpatrick and Dimitrios S. Nikolopoulos: Challenges and Opportunities in Edge Computing, *2016 IEEE International Conference on Smart Cloud (SmartCloud)*, pp. 20-26 (2016).

[3] Pengfei Hu, Huansheng Ning, Tie Qiu, Yanfei Zhang and Xiong Luo: Fog Computing Based Face Identification and Resolution Scheme in Internet of Things, *IEEE Transactions on Industrial Informatics*, Vol. 13, No. 4, pp. 1910-1920 (2017).

[4] Min Chen, Yixue Hao: Task Offloading for Mobile Edge Computing in Software Defined Ultra-Dense Network, *IEEE Journal on Selected Areas in Communications*, Vol. 36, No. 3, pp. 587-597 (2018).

[5] Manpage maintained by bert hubert: "tc-show/manipulate traffic control settings", 入手先 ⟨http://man7.org/linux/man-pages/man8/tc.8.html⟩ (参照 2019-01-26).

[6] Python Software Foundation: "Python", 入手先 ⟨https://www.python.org/⟩ (参照 2019-01-26).

[7] Intel Corporation: "OpenCV (Open Source Computer Vision Library)", 入手先 ⟨https://opencv.org⟩ (参照 2019-01-26).

[8] T. Mita and T. Kaneko and O. Hori: Joint Haar-like features for face detection, *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, pp. 20-26 (2005).

[9] T. Ojala, M. Pietikainen and T. Maenpaa: Multiresolution gray-scale and rotation invariant texture classification with local binary patterns, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 24, No. 7, pp. 971-987 (2002).

[10] van der Walt, Stéfan, Schönberger, Johannes L, Nunez-Iglesias, Juan, Boulogne, François, Warner, Joshua D, Yager, Neil, Gouillart, Emmanuelle, Yu, Tony, the scikit-image contributors: scikit-image: image processing in Python, *PeerJ*, Vol. 2, pp. e453 (2014).

[11] Iperf.fr: "Iperf - The TCP/UDP Bandwidth Measurement Tool", 入手先 ⟨https://iperf.fr/⟩ (参照 2019-01-26).

[12] Angelo Marletta: "cpulimit", 入手先 ⟨https://github.com/opsengine/cpulimit⟩ (参照 2019-01-26).

[13] Robert M. Love: "taskset-set or retrieve a process's CPU affinity", 入手先 ⟨http://manpages.ubuntu.com/manpages/xenial/man1/taskset.1.html⟩ (参照 2019-01-26).