複数層環境における推定応答時間に基づくオフローディングシステムの検討

谷 遼太郎[1,a)]　小林 裕樹[1,b)]　重野 寛[1,c)]

概要：スマートフォンなどのデバイスはバッテリやプロセッサなどのリソースの点で制約があり，アプリケーション実行の際に負荷軽減が必要とされる．オフローディングでは，クラウドサーバなどのリソースが豊富な実行環境へアプリケーションを移行することでデバイスへの負荷を軽減する．しかし，既存のオフローディング手法ではエッジやクラウドなどから構成される複数層環境に関してさらなる検討が必要である．そこで，本稿では複数層環境における推定応答時間に基づくオフローディングシステムを提案する．本稿で提案するオフローディングシステムでは，各層の実行環境においてネットワークやリソースの状態に基づき推定応答時間を算出し，推定応答時間が最小となる実行環境へとオフローディングする．

1. はじめに

現在，スマートフォンや IoT デバイスなどが広く普及している．しかし，これらのデバイスではバッテリやネットワークの帯域幅，プロセッサの処理能力といったリソースの点で様々な制約がある [1]．そのため，これらのデバイスの限られたリソースへの負荷の軽減が必要とされる．

負荷を軽減する手法の 1 つとしてオフローディングがある．オフローディングではクライアントから，バッテリやプロセッサの処理能力などのリソースが豊富なクラウドサーバやエッジサーバなどのリモート実行環境へとアプリケーションを移行して実行し，クライアントで実行結果を受信する [2]．オフローディングによって制約のあるデバイスの負荷を軽減し，アプリケーションの実行時間を短縮することが可能である．しかし，一般的にオフローディングの際には大きなデータ転送を伴うため，ネットワークの状態やアプリケーションのサイズによってはオフローディングを行うことでデータ転送に掛かる時間が増大し，クライアントで実行する場合よりもアプリケーション実行全体の応答時間が増大してしまう可能性がある．そのため，オフローディングを行う際にはネットワークの状態やアプリケーションのサイズなどに応じて，オフローディングの実行先を動的に決定する必要がある．そこで現在ではクライアントとリモート実行環境の 2 層で構成された Two-tier オフローディングが提案されている．Two-tier オフローディングではネットワークやリソースの状態に応じてオフローディングを行う．Two-tier オフローディングによって制約のあるデバイスの電力消費の削減やアプリケーションの実行時間の短縮することが可能であるが，クライアントとクラウドサーバの 2 層の実行環境に限定されているため，オフローディングによる負荷軽減の効果が不十分である．

そこで，本稿ではクライアント，エッジサーバ，クラウドサーバの実行環境から構成された複数層環境における推定応答時間に基づくオフローディングシステムを提案する．本稿で提案するオフローディングシステムではクライアントからのリクエストによってオフローディングが開始され，各層の実行環境において独立に推定応答時間を算出し，クライアントへと応答を転送する．クライアントではこの応答に基づいて各層の実行環境へオフローディングを行う場合の推定応答時間を比較し，最小の推定応答時間となる層の実行環境へとオフローディングを行う．推定応答時間はアプリケーションの実行時間とデータ転送に掛かる時間の和で表される．また，推定応答時間の算出の際には遅延などのネットワークの状態や CPU 使用率などのリソースの状態を利用する．これによってネットワークやリソースの状態に応じた動的なアプリケーションの実行を実現する．

以下，本稿では，2 章で関連研究について述べ，3 章で複数層環境における推定応答時間に基づくオフローディングシステムを提案し，4 章でプロトタイプによる実験・評価結果を示す．最後に 5 章で結論を述べる．

[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
[a)] tani@mos.ics.keio.ac.jp
[b)] kobayashi@mos.ics.keio.ac.jp
[c)] shigeno@mos.ics.keio.ac.jp

2. 関連研究

本章では，オフローディングの背景と概要について述べ，２層環境におけるオフローディング手法である Two-tier オフローディングについての関連研究と Two-tier オフローディングにおける問題点を挙げる．

2.1 オフローディングの背景

現在，スマートフォンや IoT デバイスなどのデバイスが広く普及しており，これに伴ってスマートフォンなどのモバイルデバイスで実行可能なアプリケーションも増加している．アプリケーションの例としてはオンラインゲームや SNS，ビデオストリーミングサービスや画像認識などが挙げられ，これらを実行するためにモバイルデバイスは高解像度のディスプレイやマルチコアのプロセッサを搭載し，高速ネットワーク接続を実現するなど，進歩を遂げている．しかし，同時にバッテリ駆動のモバイルデバイスではアプリケーションの実行に伴う電力消費も増大しており，高負荷なアプリケーションを実行した場合ではバッテリを短時間で使い果たしてしまう可能性がある [5]．また，物理的なスペースの制約から搭載できるストレージやプロセッサなどのリソースにも限りがあるため，これらの性能は限られる [6]．そのため，限られたリソースへの負荷を軽減する必要がある．

2.2 オフローディングの概要

モバイルデバイスの負荷を軽減する手法の 1 つとしてオフローディングが挙げられる．図 1 にオフローディングの流れを示す．オフローディングはモバイルデバイスのようなクライアントのローカル実行環境から，リソースが豊富であるクラウドサーバやエッジサーバのようなリモート実行環境へとアプリケーションを移行して実行し，クライアントで実行結果を受信するというものである．一般的に，オフローディングによってモバイルデバイスの電力消費を削減し，アプリケーションの実行時間を短縮することが可能である．しかし，単にローカル実行環境のアプリケーションをリモート実行環境へ移行して実行するだけでは，アプリケーションの実行全体の応答時間が必ずしも削減できるとは限らない．例えば，ローカル実行環境とリモート実行環境の間のネットワークに大きな遅延が発生した場合，オフローディングを行う際のアプリケーションの転送や応答の転送に長時間を要し，結果として応答時間が増大してしまう可能性がある．また，アプリケーションの実行に必要なデータサイズが大きい場合にも同様にデータ転送に長時間を要し，応答時間が増大してしまう可能性がある．また，データ転送に大きな時間を要することで通信を行う時間が増加し，電力消費が増加することで，オフローディングによる負荷軽減の効果が不十分になる可能性がある [7]．

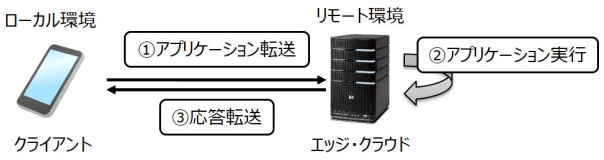

図 1　オフローディングの流れ

以上のことから，ネットワークの状態や実行するアプリケーションに応じて，モバイルデバイスで実行するのか，オフローディングを行うのかを動的に決定するオフローディング手法が必要である [8][9]．

2.3 Two-tier オフローディング

Two-tier オフローディングはローカル実行環境であるモバイルデバイスとリモート実行環境であるクラウドサーバやエッジサーバの２層の環境を想定したオフローディング手法である．Two-tier オフローディングではネットワークの状態やオフローディング先となるリモート実行環境のリソースの状態に応じてローカル環境での実行とオフローディングの決定を動的に行うことで，オフローディングによるモバイルデバイスの電力消費の削減や応答時間の削減などの負荷軽減の効果を高めている．

MAUI[3] ではモバイルデバイスでの電力消費の削減とを目的としており，モバイルデバイスでの実行用とエッジサーバでの実行用の２つのバージョンのアプリケーションを用意し，アプリケーション実行時に電力消費が最小となるようにオフローディングを行うかを決定する．MAUI ではオフローディングの粒度はメソッド単位である．CloneCloud[4] ではアプリケーション実行の高速化と，モバイルデバイスでの電力消費の削減を目的としており，モバイルデバイスのクローン環境をクラウサーバ上のバーチャルマシンに作成し，アプリケーションを分割実行する．CloneCloud ではオフローディングの粒度はスレッド単位である．

2.4 Two-tier オフローディングの問題点

MAUI や CloneCloud のような Two-tier オフローディングではローカル実行環境であるモバイルデバイスと，クラウドサーバなどのリモート実行環境の２層の環境に限定されており，実環境で想定されるクラウドサーバやエッジサーバから構成された複数層の環境について考慮がされていないという問題点がある．MAUI や CloneCloud などの Two-tier オフローディングでは，ネットワークの状態やリモート実行環境のリソースの状態に応じて動的にオフローディングを行うが，アプリケーションの実行先はモバイルデバイスかエッジサーバやクラウドサーバの２つしか存在しないため，例えば，これらの間のネットワークの遅延が大きくなった場合にはモバイルデバイスで実行するしか選択肢がなく，オフローディングによるモバイルデバイスの

負荷軽減が不十分となる可能性がある．

そのため，Two-tier オフローディングはオフローディング先の選択肢が限定的と言え，より実環境に近い複数層で構成された環境で適用可能なオフローディング手法が必要となる．

3. 提案手法

本章では，複数層環境における推定応答時間に基づくオフローディングシステムについて提案する．

3.1 オフローディングシステムの概要

本稿で提案するオフローディングシステムでは，各層の実行環境において独立に自層の実行環境でアプリケーションを実行する場合の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層の実行環境へとオフローディングを行う．推定応答時間はアプリケーションのサイズ，ネットワークの帯域幅や遅延などのネットワークの状態，CPU 使用率などのリソースの状態などに基づき算出する．推定応答時間が最小となる実行環境へとオフローディングを行うことで，同時にクライアントにおける電力消費の削減も期待される．本稿で提案するオフローディングシステムでは Two-tier オフローディングでは考慮していなかったエッジやクラウドから構成される複数層の環境を想定している．オフローディングの際には遅延や CPU 使用率を考慮することでネットワークに遅延が生じている場合や，オフローディング前にすでに実行環境のリソースへ負荷が掛かっている場合にも適用可能にする．

3.2 オフローディングシステムの想定環境

図 2 に本稿で提案するオフローディングシステムの構成を示す．本稿で提案するオフローディングシステムではクライアント，エッジ，クラウドからなる複数層の実行環境から構成される環境を想定する．Tier 1 はクライアント（R_1），Tier 2 はエッジサーバ（R_2），Tier 3 はクラウドサーバ（R_3）から構成される．

3.3 オフローディングシステムの構成

図 3 に本稿で実装するオフローディングシステムのコンポーネントの構成を示す．各層の実行環境は Manager によってオフローディング時の制御が行われる．

- Client Manager
  Offload Controller としてオフローディングを管理する役割を担っており，隣接するエッジへとリクエストを送信する Request Sender，各層の実行環境からの応答を受信する Resource Offer Receiver，オフローディングを行う層の実行環境へと最終的なオフローディングのリクエストを転送する Offload Request Sender，Java のクラスローダによってアプリケーションのクラスファイルをエッジやクラウドへロードするためのサーバとなる Class Load Server，アプリケーションの実行結果を受信する Result Receiver から構成される．アプリケーション実行時には Client Manager が起動され，リクエストをエッジへと送信することでオフローディング手順が開始される．また，各層の実行環境での推定応答時間の算出結果に基づき，最小の推定応答時間となる層の実行環境へとアプリケーションの実行を移行する．
- Edge Manager，Cloud Manager
  オフローディング先となるエッジとクラウドでは，それぞれ Edge Manager と Cloud Manager がオフローディングを制御している．Edge Manager と Cloud Manager はリクエストの受信や応答の転送，推定応答時間の算出などのオフローディングを行うまでの処理を担う Resource Manager と，クラスロードや実行結果の転送などのオフローディングを行う際の処理を担う Offload Execution Engine で構成されている．Resource Manager はリクエストを受信し，自層

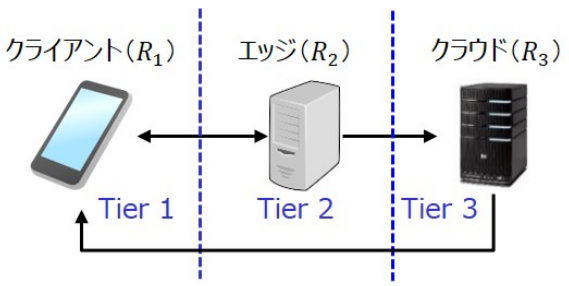

図 2 オフローディングシステムの想定環境

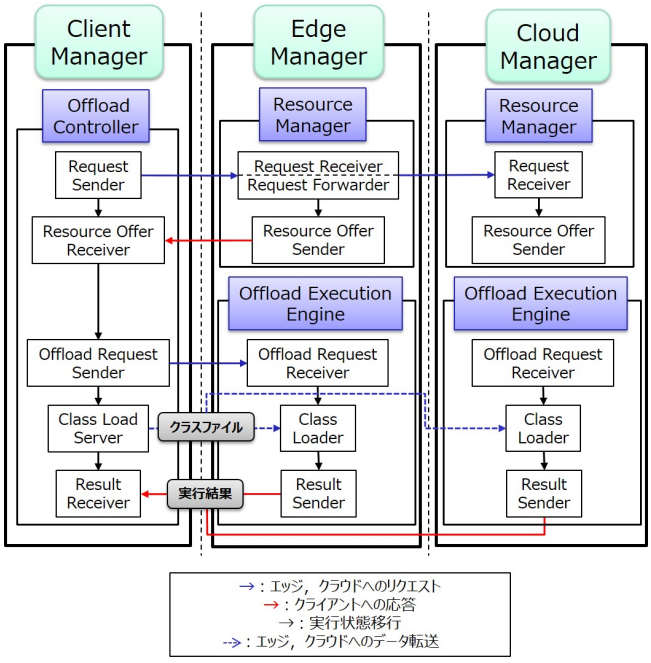

図 3 オフローディングシステムの構成コンポーネント

の実行環境における推定応答時間を算出する Request Receiver，エッジからクラウドへとリクエストを転送する Request Forwarder，クライアントへ応答を転送する Resource Offer Sender によって構成される．Offload Execution Engine はクライアントからの最終的なオフローディングのリクエストを受信する Offload Request Receiver，クラスローダによってクライアントに存在するアプリケーションのクラスファイルをロードする Class Loader，実行結果をクライアントへと転送する Result Sender から構成される．

3.4 推定応答時間の算出

推定応答時間はデータ転送に掛かる時間とアプリケーションの実行時間の和である式（1）によって算出する．

$$T(A_{a_0}, R_l) = T_{transmit}(A_{a_0}, R_l) + T_{exec}(A_{a_0}, R_l) \quad (1)$$

ここで，A_{a_0} は実行するアプリケーション，R_l は Tier l の実行環境を表しており，第一項の $T_{transmit}(A_{a_0}, R_l)$ は，実行環境 R_l へアプリケーション A_{a_0} のデータを転送するに掛かる時間，第二項の $T_{exec}(A_{a_0}, R_l)$ は実行環境 R_l におけるアプリケーション A_{a_0} の実行時間を表している．また，$T_{transmit}(A_{a_0}, R_l)$ は式（2）のように，$T_{exec}(A_{a_0}, R_l)$ は式（3）のように表される．

$$T_{transmit}(A_{a_0}, R_l) = \frac{IS(A_{a_0})}{BW_{spd}(R_{1\to l})} + D_{link}(R_{1\to l}) \quad (2)$$

$$T_{exec}(A_{a_0}, R_l) = w_{CPU}(R_l) \cdot ET_{exec}(A_{a_0}, R_l) \quad (3)$$

$IS(A_{a_0})$ は実行するアプリケーション A_{a_0} のサイズ，$BW_{spd}(R_{1\to l})$ はクライアント R_1 と実行環境 R_l 間のネットワークの帯域幅，$D_{link}(R_{1\to l})$ は Offloading Request 送信時のクライアント R_1 と実行環境 R_l 間におけるネットワークのリンク遅延，$w_{CPU}(R_l)$ は実行環境 R_l におけるアプリケーションが使用可能な CPU 使用率に基づく重み，$ET_{exec}(A_{a_0}, R_l)$ は実行環境 R_l において，アプリケーションが CPU を 100%使用可能な場合の推定実行時間を表している．また，$ET_{exec}(A_{a_0}, R_l)$ はアプリケーションを事前に静的解析することによって式（4）のような形に簡易的に定式化し，推定応答時間の算出に用いる．

$$ET_{exec}(A_{a_0}, R_l) = -a \cdot f_{R_l} + b \quad (4)$$

式（4）において a と b はアプリケーションの静的解析によって得られる傾きと切片，f_{R_l} は実行環境 R_l の CPU の動作周波数である．各実行環境において利用可能な CPU 使用率を用いることで，オフローディング前にすでに負荷が高い実行環境へのオフローディングを回避する．

3.5 オフローディング手順

図 4 にエッジへオフローディングする場合のオフローディング手順例を，以下にその流れを示す．

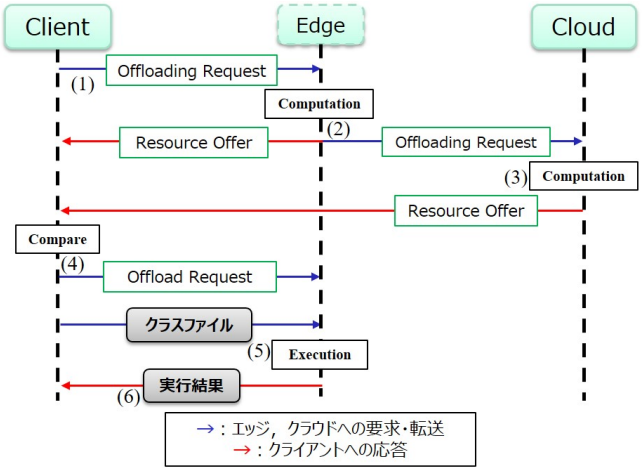

図 4 エッジへオフローディングする場合のオフローディング手順例

（1）クライアントから実行するアプリケーションのサイズ，クライアントにおける推定応答時間が格納されたリクエストをエッジへと送信する．

（2）クライアントからのリクエストを受信したエッジではリクエストに基づき，エッジにおける推定応答時間の算出を行い，エッジにおける推定応答時間が格納された Resource Offer をクライアントへと転送する．また，同時にリクエストに格納された推定応答時間をエッジにおける推定応答時間に置き換え，クラウドへとリクエストを転送する．

（3）エッジからのリクエストを受信したクラウドではリクエストに基づき，クラウドにおける推定応答時間の算出を行う．この例では，エッジにおける推定応答時間の方がクラウドにおける推定応答時間よりも小さいため，クラウドへはオフローディングを行えないことを示した Resource Offer をクライアントへと送信する．

（4）エッジとクラウドからの Resource Offer を受信したクライアントではすべての推定応答時間を比較し，最も推定応答時間が小さいエッジへと Offload Request を送信する．

（5）Offload Request を受信したエッジでは，クラスローダによってクライアントからアプリケーションのクラスファイルをロードし，実行する．

（6）エッジにおいてアプリケーションの実行が終了すると，実行結果をクライアントへと転送する．

4. プロトタイプ実装・評価

本章では，オフローディングシステムのプロトタイプ実装と，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 実装環境

図 5 にオフローディングシステムの実装環境を，表 1 に

使用機材を示す．各機材同士は有線接続されており，クライアントとエッジの間にはネットワークの状態を制御できるネットワークエミュレータを配置し，リンク遅延を付与しない場合と，意図的にリンク遅延を付与した場合の 2 つの状況で実験を行った．遅延未付与の場合と遅延付与の場合のそれぞれの帯域幅を Iperf[10] を用いて測定し，実測値を実験パラメータとして利用した．付与したリンク遅延の設定値とネットワーク帯域幅の実測値による実験パラメータを表 2 に示す．

式（3）の CPU 使用率に基づく重み w_{CPU} は今回の実験では cpulimit[11] を用いることで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，実行環境に負荷が掛かった状態を再現した．また，実験においては式（5）のように w_{CPU} を設定した．

$$w_{CPU} = 1 - \frac{CPU_{set}}{100} \tag{5}$$

式（5）において，CPU_{set} は cpulimit によって制限した CPU 使用率であり，例えば，CPU 使用率が 30%の場合にはアプリケーション実行に 70%の CPU を使用できるため，$w_{CPU} = 0.7$ となる．

推定実行時間を表す式（4）の a と b は CPU の動作周波数と実行時間の関係を調べた予備実験で図 6 のような結果が得られ，これに基づき今回の実験においては式（6）のように設定した．

$$ET_{exec}(A_{a_0}, R_l) = -1949.3 \cdot f_{R_l} + 7924.4 \tag{6}$$

実行するアプリケーションとしては，100,000 までの整数の中に素数がいくつ存在するかをカウントする素数判定プログラムを用いた．この素数判定プログラムはクライア

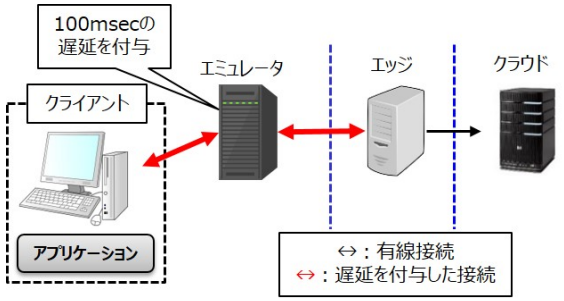

図 5　実装環境

表 1　使用機材

	OS	CPU	メモリ
Client	Ubuntu 14.04 LTS	Intel Pentium G4400@ 3.30GHz	4GB
Edge	Ubuntu 14.04 LTS	Intel Pentium G3240@ 3.20GHz	4GB
Cloud	Ubuntu 16.04 LTS	Intel Core i5-2400@ 3.30GHz	2GB
エミュレータ	Ubuntu 14.04 LTS	Intel Core i7-4790@ 3.60GHz	8GB

表 2　実験パラメータ

	遅延未付与	遅延付与
ネットワーク帯域幅 $BW_{spd}(R_{1\to l})$ (測定値)	940Mbps	80Mbps
リンク遅延　$D_{link}(R_{1\to l})$ (設定値)	0msec	100msec

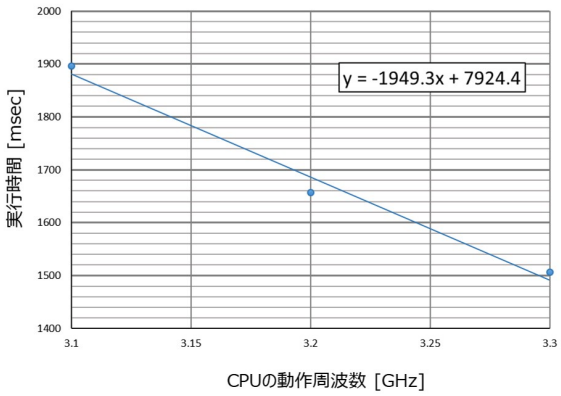

図 6　CPU 動作周波数と実行時間の関係

ントにおいて CPU 使用率が 0%の場合に約 1,500msec で実行が終了し，今回の実験においてはクライアントにのみ配置されている．また，このプログラムのクラスファイルのサイズは 923Byte である．

評価項目としては遅延なしの場合と遅延ありの場合のそれぞれについて，アプリケーション実行全体の応答時間を測定した．

4.2　動作確認と評価

図 7 にクライアントとエッジ間に遅延を付与していない場合の各実行環境における推定応答時間とオフローディングが実行されたエッジにおける実際の応答時間を示す．ここで CPU 使用率はクライアントで 40%，エッジで 0%，クラウドで 20%に設定した．図 7 において Client，Edge，Cloud は各層の実行環境における推定応答時間を表しており，Client = 2,352msec，Edge = 1,687msec，Cloud = 2,352msec であり，エッジの推定応答時間が最小となった．これに基づき，エッジへオフローディングが実行され，実際の応答時間は Ex_Edge = 1,738msec であった．Ex_Edge にはアプリケーションを実行するのに掛かった実行時間とデータ転送に掛かった時間が含まれる．

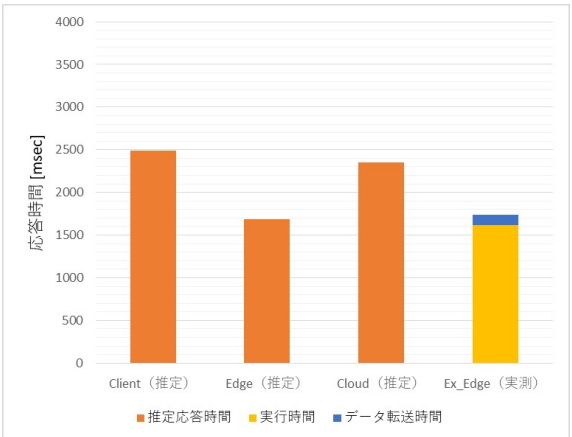

図 7　エッジへオフローディングされる場合の応答時間（遅延未付与）

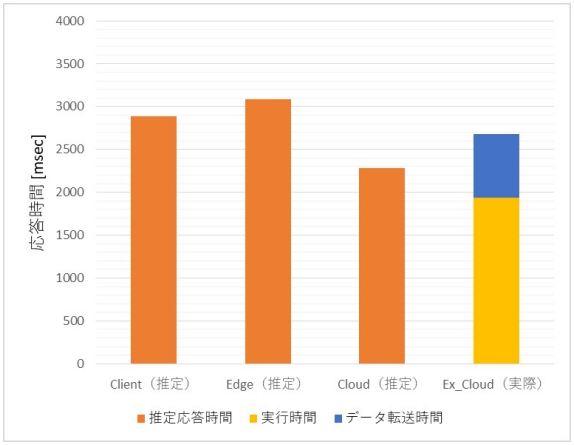

図 8 クラウドへオフローディングされる場合の応答時間（遅延付与）

図 8 にクライアントとエッジ間に 100msec の遅延を付与した場合の各実行環境における推定応答時間とオフローディングが実行されたクラウドにおける実際の応答時間を示す．ここで CPU 使用率はクライアントで 40%，エッジで 0%，クラウドで 0%に設定した．図 8 において Client，Edge，Cloud は各層の実行環境における推定応答時間を表しており，Client = 2,886msec，Edge = 3,087msec，Cloud = 2,282msec であり，クラウドの推定応答時間が最小となった．これに基づき，クラウドへオフローディングが実行され，実際の応答時間は Ex_Cloud = 2,682msec であった．Ex_Cloud にはアプリケーションを実行するのに掛かった実行時間とデータ転送に掛かった時間が含まれる．

本稿で提案したオフローディングシステムのプロトタイプの実装による実験から，遅延を付与していない場合と遅延を付与した場合の両方で推定応答時間の算出に基づき，最小の推定応答時間となる層の実行環境へオフローディングが実行されることが確認できた．

5. おわりに

本稿ではクライアント，クラウドサーバやエッジサーバなどの複数層環境における推定応答時間に基づくオフローディングシステムの提案を行った．本稿で提案したオフローディングシステムでは，各層の実行環境において独立に推定応答時間を算出し，最小の推定応答時間である層の実行環境へとオフローディングを行う．推定応答時間はアプリケーションの実行時間とデータ転送に掛かる時間の和で表される．また，推定応答時間の算出の際には遅延などのネットワークの状態や CPU 使用率などのリソースの状態を利用する．これによってネットワークやリソースの状態に応じた動的なオフローディングを実現する．

また，提案したオフローディング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，本稿で提案したオフローディングシステムにおいて，各層の実行環境おける推定応答時間の算出に基づき，最小の推定応答時間となる層の実行環境へオフローディングが実行されることを確認した．

以上より，本稿で提案したオフローディングシステムによって，複数層環境においてネットワークの状態やリソースの状態に応じた動的なオフローディングが可能であることを示した．

参考文献

[1] Flores, H., Hui, P., Tarkoma, S., Li, Y., Srirama, S. and Buyya, R.: Mobile code offloading: from concept to practice and beyond, *IEEE COMMUNICATIONS MAGAZINE*, Vol. 53, No. 3, pp. 80–88 (2015).

[2] Benedetto, J. I., Neyem, A., Navon, J. and Valenzuela, G.: Rethinking the Mobile Code Offloading Paradigm: From Concept to Practice, in *Proceedings of the 4th International Conference on Mobile Software Engineering and Systems*, MOBILESoft '17, pp. 63–67 (2017).

[3] Cuervo, E., Balasubramanian, A., Cho, D.-k., Wolman, A., Saroiu, S., Chandra, R. and Bahl, P.: MAUI: Making Smartphones Last Longer with Code Offload, in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, pp. 49–62 (2010).

[4] Chun, B.-G., Ihm, S., Maniatis, P., Naik, M. and Patti, A.: CloneCloud: Elastic Execution Between Mobile Device and Cloud, in *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, pp. 301–314 (2011).

[5] Sekar, K.: Power and Thermal Challenges in Mobile Devices, in *Proceedings of the 19th Annual International Conference on Mobile Computing; Networking*, MobiCom '13, pp. 363–368 (2013).

[6] Chen, M. and Hao, Y.: 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).

[7] Kumar, K., Liu, J., Lu, Y.-H. and Bhargava, B.: A Survey of Computation Offloading for Mobile Systems, *Mob. Netw. Appl.*, Vol. 18, No. 1, pp. 129–140 (2013).

[8] Shi, C., Habak, K., Pandurangan, P., Ammar, M., Naik, M. and Zegura, E.: COSMOS: Computation Offloading As a Service for Mobile Devices, in *Proceedings of the 15th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, MobiHoc '14, pp. 287–296 (2014).

[9] Balan, R., Flinn, J., Satyanarayanan, M., Sinnamohideen, S. and Yang, H.-I.: The Case for Cyber Foraging, in *Proceedings of the 10th Workshop on ACM SIGOPS European Workshop*, EW 10, pp. 87–92 (2002).

[10] Iperf.fr, : "Iperf - The TCP/UDP Bandwidth Measurement Tool", `https://iperf.fr/` (Accessed : 2018-01-22).

[11] Marletta, A.: "cpulimit", `https://github.com/opsengine/cpulimit` (Accessed : 2018-01-22).