IoT アプリケーション向け通信フレームワークにおけるサーバ選択機構

2014SE118 浜田叶夢アンドレ　2015SE049 道浦舜　2015SE081 竹内滉
指導教員：宮澤元

1　はじめに

近年，IoT(Internet of Things) の分野では，IoT デバイスの近くに配置したエッジサーバで一部の処理を行い，クラウドサーバ，エッジサーバ，IoT デバイスの三層構造で IoT アプリケーションを構成するという考え方が広まっている．IoT デバイスより送られるデータを事前に処理することで，クラウドサーバの負荷を抑えることができる．しかしクラウドと IoT デバイス間で処理を行う二層構造に比べてアプリケーションの開発が複雑になる．

開発を容易にするために多くの企業や団体で IoT 向けのアプリケーションフレームワークの開発が行われている．Liota [1] を始めとするアプリケーションフレームワークでは複数存在する通信プロトコルの処理を行うプログラムをアプリケーション開発者から隠蔽し，IoT デバイスのデータ収集機能，エッジサーバのデータ処理機能などの典型的な機能を抽象化することができる．

IoT アプリケーションを三層構造で開発する上で，開発者は IoT デバイスからのデータ送信先として適切なエッジサーバを選択するようにソフトウェアを設計する必要がある．IoT アプリケーションでは通信レイテンシの低減や負荷分散を意識したサーバ選択が特に重要である．例えば高齢者の容体を常に監視する医療端末では，端末が移動することが考えられる．大きく移動した場合，データの送信先が固定されていると場所によっては通信レイテンシが大きくなってしまう．容体の急変を管理者に知らせる機能が満足に働かなくなることは望ましくない．また，IoT アプリケーションでは Amazon Kinesis Video Streams などの IoT デバイスの機械学習を目的とした動画のストリーミングサービスのように一度に大容量のデータを送信するシステムも存在する．そこでは多くの IoT デバイスがひとつのエッジサーバにデータを要求してしまうと負荷が集中してしまい満足にデータを送信できなくなってしまう．

既存の IoT 向けのアプリケーションフレームワークでは，サーバの選択に関して開発者が最適な接続先を選択するための十分なサポートが提供されていない．IoT では端末の移動，大容量のデータ処理などアプリケーションによって求められることが大きく変わってしまうからだ．最適な接続先サーバを選択する機構が提供されなければ，エッジコンピューティングを活用するために，開発者は IoT デバイスの移動などに対応したエッジサーバの選択機構を自ら作成しなければならない．

そこで我々は，通信フレームワークの機能の中でも最適なエッジサーバの選択方法について着目した．IoT アプリケーションにおいて最適なエッジサーバは常に同じではない．様々な状況に合わせて接続先を選択する必要がある．

本稿では IoT デバイスとエッジサーバの選択に着目し，通信レイテンシの抑制や負荷分散など開発者が意識しなければならない条件を満たすサーバの選択手法を提案する．本選択手法を用いることで IoT 向けのアプリケーションフレームワークにおいて開発者が意識すべき点を満たすサーバの選択を実現することができる．実際に提案手法を用いた選択機構を実装し，選択手法が適切に動作することを確認する．

2　背景技術

ここでは IoT 向けのアプリケーションフレームワークである Liota について示す．

Liota(Little IoT agent) とは IoT アプリケーション開発を容易にするため VMware により開発されたオープンソースのフレームワークである．Liota の主な役割は IoT デバイス，エッジサーバ，データセンタの三層構造に基づく IoT アプリケーションにおける通信処理の支援である．Liota を使用することで開発者は通信プロトコルの内部処理などの導入でかかる手間を減らすことができる．複数の通信プロトコルに対応したアプリケーションを作成する場合，開発者は通信処理を行うプログラムをそれぞれの通信プロトコルの仕様に合わせて個別に作成しなければならない．しかし，Liota が提供するライブラリ関数を使用することで，それぞれの通信プロトコルの仕様を意識することなく通信処理を行うプログラムを作成することができる．また，通信プロトコルの切り替えも提供するライブラリ関数を変えることで実現することができる．新たに通信プロトコルを追加したい場合は Liota 内で新たなライブラリを追加することで対応することができる．

しかし，サーバの選択を始めとした選択を自動化する手段は提供されていない．開発者は通信レイテンシや負荷分散などのサーバ選択に関わる条件を考慮した複雑な設計の IoT アプリケーションを開発する必要がある．

3　エッジサーバの選択手法

本節では，我々が提案する IoT 向けの開発フレームワークにおける最適な接続先エッジサーバの選択手法について述べる．

3.1　サーバ選択の仕組み

本選択手法では第一に通信レイテンシを基準にした切り替えを行う．これにより通信レイテンシを抑制し，リアルタイム性を向上することができる．IoT では最新の情報を要求されるものが多い．IoT は医療現場など急を要する状況で要求されることもあるので，許容できない通信レイテンシは決して起こしてはならない．よって通

信レイテンシの抑制は第一に意識しておくべきである．

第二に負荷分散を意識した切り替えを行う．IoT では動画のストリーミングのように大容量のデータ送信を複数の IoT デバイスがエッジサーバに要求をするシステムが存在する．しかし，一つのエッジサーバに IoT デバイスからの要求が集中してしまうと通信速度が遅くなってしまう．負荷分散を行うことで，一つのエッジサーバにかかる負荷が下がり，帯域幅の圧迫や処理速度の低下を防ぐことができる．それによって，エッジサーバにかかる負荷が下がり，帯域幅の圧迫や処理速度の低下を防ぐことができる．しかし，エッジサーバの負荷を下げることは，アプリケーションによっては動画の画質を落とすなどのサービスの品質を低下させることでもできる．なので，通信レイテンシに比べるとサーバ選択の際に考慮すべき条件としての優先度は低い．

よって以下の条件に従ってエッジサーバを選択する．

1. 通信レイテンシが開発者が設定した値を超えた場合，超えていないサーバへ切り替えを行う
2. 負荷分散のために複数の項目のいずれかを比較し切り替えを行う
    - 各サーバの帯域幅使用率の比較を行い，使用率の低いサーバがある場合切り替えを行う
    - 各サーバの CPU 使用率の比較を行い，使用率の低いサーバがある場合切り替えを行う

通信レイテンシ，帯域幅使用率，CPU 使用率の計測は IoT デバイスと接続しているエッジサーバが担当する．IoT デバイスが接続するエッジサーバを切り替えるかどうかの判断もエッジサーバが行い，IoT デバイスに対してエッジサーバを切り替えるように指示する．これは，比較的計算リソースが少ない IoT デバイスに極力負荷をかけないためである．

3.2　通信レイテンシによる切り替え

通信レイテンシを基準にした切り替えでは，各エッジサーバは自身と IoT デバイスの間で発生する通信レイテンシを計測する．切り替え回数を最低限に抑えるために，接続されているエッジサーバは利用者が設定したしきい値を超えていた場合に IoT デバイスに対して切り替え命令を出す．接続先の切り替えが頻繁に行われてしまうと処理が必要以上に増加し，IoT デバイスやエッジサーバに負荷を与えてしまう恐れがあるからである．しかし，しきい値を超えたかどうかだけを条件にしてしまうと，現在はしきい値を超えていないが，このままだと超えてしまうエッジサーバを選択してしまう可能性がある．そこで， IoT デバイスとの通信レイテンシがしきい値未満かつその通信レイテンシが現在のものより十分小さいようなサーバに接続先を切り替える．

3.3　エッジサーバの負荷分散

負荷分散を意識した切り替えでは帯域幅使用率と CPU 使用率を用いる．帯域幅使用率，CPU 使用率は値が大きくなるとサーバの持つ処理能力に影響を及ぼすので，抑えることで効率の良いデータ送信を実現することができる．帯域幅使用率と CPU 使用率を用いた切り替えでは，各エッジサーバは自身の帯域幅使用率と CPU 使用率を計測する．接続されているエッジサーバは一定の間隔で計測を続け，設定したしきい値を 3 回連続で超えている状況が続く場合，自身よりも帯域幅使用率が少ないエッジサーバへ切り替え命令を IoT デバイスへ行う．しきい値を設けることで，想定以上に値が高くなった場合即座に切り替えを行うようにし，IoT アプリケーションに与える影響を低減させることができるからである．

4　シミュレーションを用いた実装と実験

接続しているサーバの通信レイテンシがしきい値に達した時，最適なサーバへ切り替えられることを ns-3 [2] を用いたシミュレーションで確認する．これにより実機を用いた切り替え実験の目安にすることができる．

4.1　シミュレーションの設定

実装ではシミュレーションの対象のノードとして IoT デバイスを 1 台，エッジサーバを 2 台用意した．IoT デバイスは異なったインターネット経路でそれぞれのエッジサーバと接続する．シミュレーション時間が開始すると，IoT デバイスは一つのエッジサーバに接続し，データを繰り返し送信する．切り替えが行われる場合，データの送信を中断し，もう一方のエッジサーバへデータの送信を開始および再開する．

4.2　シミュレーション結果

はじめに，2 台の固定されたエッジサーバ A とエッジサーバ B の間で A から B に IoT デバイスが移動することを想定した．ケース 1 では，10 秒ごとに通信レイテンシが 50ms 増加するエッジサーバ A と，10 秒ごとに通信レイテンシが 25ms 減少するエッジサーバ B を用意した．エッジサーバ A とエッジサーバ B の通信レイテンシの初期値はそれぞれ 0.8ms と 300ms である．はじめに IoT デバイスはエッジサーバ A に接続を行う．通信往復時間のしきい値は 700ms 以下とした．シミュレーション時のエッジサーバ A と IoT デバイス間，エッジサーバ B と IoT デバイス間，IoT デバイスと IoT デバイスが送信しているエッジサーバ間での計測結果を**図 1** に示す．

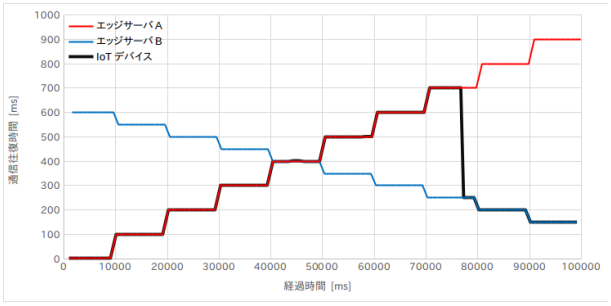


図 1　ケース 1：A から B へ IoT デバイスの移動を想定したシミュレート結果

シミュレーション時間のおよそ 76 秒にて，IoT デバイスと送信先のエッジサーバ間の通信往復時間が 701ms か

ら 250ms に下降している．設定したしきい値に基づいた送信先切り替えを確認できた．

次に移動先のエッジサーバ B がケース 1 よりも遠い状況を想定した．ケース 2 では，10 秒ごとに通信レイテンシが 50ms 増加するエッジサーバ A と，10 秒ごとに通信レイテンシが 30ms 減少するエッジサーバ B を用意した．エッジサーバ A とエッジサーバ B の通信レイテンシの初期値はそれぞれ 0.8ms と 600ms である．IoT デバイスはエッジサーバ A に初期送信を行う．通信往復時間のしきい値は 700ms 以下とした．シミュレーション時のエッジサーバ A と IoT デバイス間，エッジサーバ B と IoT デバイス間，IoT デバイスと IoT デバイスが送信しているエッジサーバ間での計測結果を**図 2** に示す．

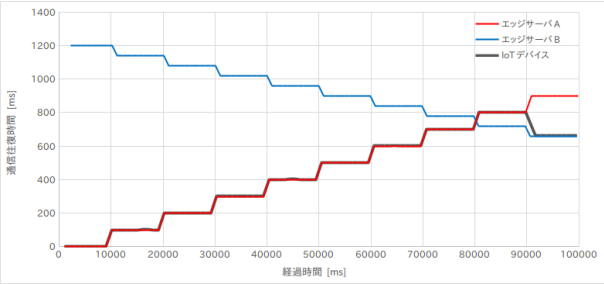


図 2　ケース 2：B の距離がケース 1 よりも遠方の場合を想定したシミュレート結果

およそ 76 秒では，エッジサーバ B の通信往復時間はエッジサーバ A のものより長いので，エッジサーバ A の通信往復時間がしきい値を超えているにも関わらず送信先切り替えを行っていない．およそ 80 秒では，エッジサーバ B の通信往復時間はしきい値を超えているので，エッジサーバ B の通信往復時間はエッジサーバ A のものより短くなったにも関わらず送信先切り替えを行っていない．およそ 90 秒にて，エッジサーバ B の通信往復時間がしきい値を下回っているので，IoT デバイスと送信先のエッジサーバ間の通信往復時間が 800ms から 660ms に下降している．よって，実装した判断機能に基づいた送信先切り替えを確認できた．

5　エッジサーバ選択機構の実装と実験

IoT デバイスとエッジサーバに関わる選択機構を Python を用いて実装した．実装したサーバ選択機構を用いることで，提案した選択方式に従って機能することを示す実験を行った．エッジサーバの役割を持つ PC を二台用意し，提案した基準での切り替えが実際に可能かどうかを確認する．

5.1　切り替え機能の実装

通信レイテンシの計測は ping を用いて計測を行う．通信レイテンシを計測するために Python 上で ping を使用することを可能にする pyping [3] を使用した．帯域幅使用率と CPU 使用率は SNMP(Simple Network Management Protocol) を用いて入手したデータより計算することが可能である．しかし，計測した瞬間の CPU 使用率は値が 0 と 100 のように極端な値を算出してしまう．そこで CPU 使用率は過去 60 秒の平均を用いる．帯域幅使用率と CPU 使用率を計測するために Python 上で SNMP を使用することを可能にする PySNMP [4] を使用した．実装では 3.2 節で説明した「通信レイテンシが現在のものより十分小さいようなサーバに接続先を選択する」部分が未実装であるが，実験結果には大きな影響を与えていない．

5.2　通信レイテンシによる切り替え実験

pyping を使用した通信レイテンシの計測・切り替えが問題なく動作するかを確認する．10 秒ごとに通信レイテンシが 50ms 増加する擬似負荷を与えたエッジサーバ A と，負荷を一切与えないエッジサーバ B を用意した．IoT デバイスははじめにエッジサーバ A に接続を行う．監視カメラを使用したシステムを想定し，IoT デバイスは動画ストリーミングサービスのデータ送信量の目安となる 1Mbps をエッジサーバへ送信する．また切り替えの基準となる通信レイテンシは 350ms 以下とした．IP 電話事業者のクラス分けの基準となる遅延時間を参考にし，最低クラスの品質を保証する 400ms 以下を常に維持するためである．実験時のエッジサーバ A と IoT デバイス，エッジサーバ B と IoT デバイス，IoT デバイスと接続しているエッジサーバ間で通信レイテンシを計測した結果を**図 3** に示す．IoT デバイスと接続しているエッジサーバ間の通信レイテンシが 350ms の時の 78 秒の時点で大幅に下降し，約 0ms に落ちている．こちらで設定したしきい値の条件を IoT デバイスが上回ったので切り替えが発生しエッジサーバ B と接続された．エッジサーバ B は一切負荷を与えていないので大きな通信レイテンシは発生していない．この結果から，本サーバ選択システムを用い，通信レイテンシを基準にして適切なエッジサーバに切り替えを行うことができることを確認できた．

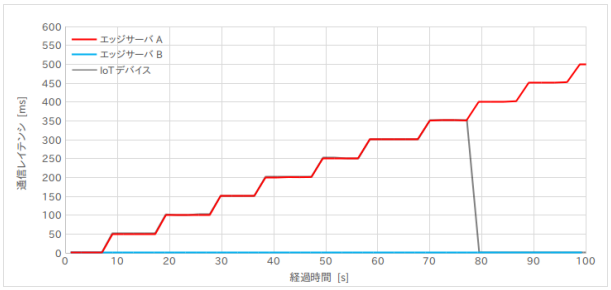
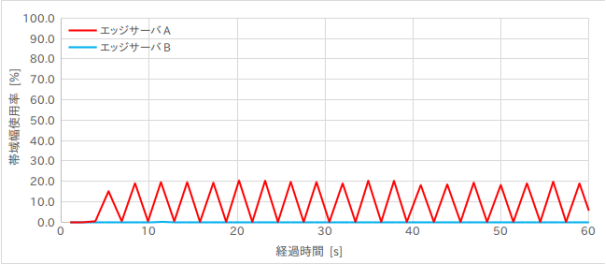


図 3　通信レイテンシを基準にした切り替え実験

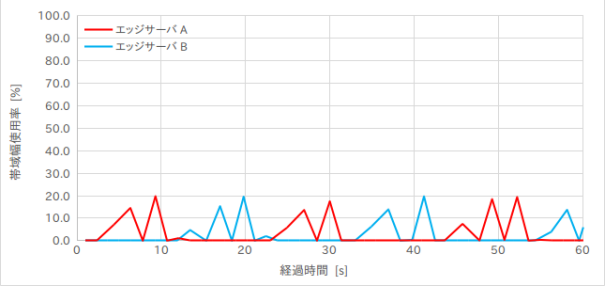
5.3　帯域幅使用率による切り替え実験

次に帯域幅を基準にして切り替えをすることが可能であるかを確認する．2 台のエッジサーバを用意し，はじめに IoT デバイスはエッジサーバ A に対して接続を行う．5.2 節と同じように監視カメラを想定し，IoT デバイスは 1Mbps をエッジサーバに送信する．実験では切り替え機能を実装していない場合と実装した場合の比較を行うことで確認する．切り替え機能ではしきい値を 0.1%とした．データを受信していない場合の帯域幅使用率は 0%なので使用されていることを確認するためである．およそ 1 秒の間隔で計測を繰り返し，3 回の連続した計測の全てで

帯域幅使用率がしきい値を超えた場合，切り替え命令を出す．



(a) 切り替え無し



(b) 切り替え有り

図 4　帯域幅使用率を基準にした切り替え実験

図 4(a) では IoT がエッジサーバ A に常にデータの送信を行うため，エッジサーバ A の帯域幅使用率が高い状態が続くが，帯域幅使用率による切り替え機能を使用した場合，**図 4(b)** の結果となった．帯域幅使用率がしきい値を超えた状態が続くことによって接続先のサーバが切り替わっており，帯域幅使用率の計測及び帯域幅使用率を元にした切り替え機能が想定通り動作していることがわかる．

5.4　通信レイテンシと帯域幅使用率を基準にした切り替え実験

基準として通信レイテンシと帯域幅使用率の両方を用いるサーバ切り替え実験を行う．5.2 節と同様に通信レイテンシが 350ms を超えた時に切り替えが起こるように設定し，10 秒ごとに通信レイテンシを 50ms 増加する擬似負荷をエッジサーバ A に与え続ける．帯域幅使用率は 5.3 節と同様に切り替え確認できるようにしきい値を 0.1%と設定した．およそ 1 秒の間隔で計測を繰り返し，3 回連続で超えた場合に切り替えが起こるようにする．その時の通信レイテンシと帯域幅使用率の状態を確認し，切り替えが想定通りにできているかを確認した．

図 5 では実線が通信レイテンシ，点線は帯域幅使用率を表している．IoT デバイスと接続しているエッジサーバはデータを常に受け取るので，帯域幅使用率が大きくなる．よって，IoT デバイスがどちらのエッジサーバと接続されているのかが判断できる．また IoT デバイスが計測した通信レイテンシを確認することでも接続先を判断できる．今回の結果として通信レイテンシのしきい値に到達する 350ms(経過時間 70s ～ 80s) 以降，エッジサーバ A へ切り替えが発生しなかった．しかし，帯域幅使用率による切り替えが行われる経過時間 80s 以前の通信レイテンシは**図 3** と比べると大きくゆらいでしまっている．その点から帯域幅使用率による切り替えが通信レイテンシに負荷を与えてしまっていると考えられる．

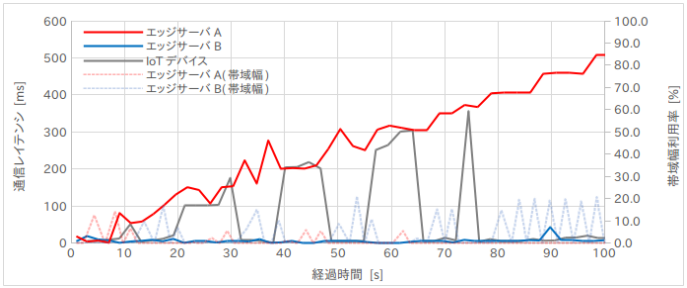


図 5　通信レイテンシと帯域幅使用率を基準にした切り替え実験

6　おわりに

我々は，通信レイテンシと帯域幅使用率， CPU 使用率を用いたエッジコンピューティングでのサーバ選択機構を提案した．提案した手法をシミュレータと実機で実装し，各切り替え機能の確認を行った．

提案したサーバ選択機構を用いた実験により，接続先のサーバを適切に切り替えられることを確認した．しかし，基準として通信レイテンシと帯域幅使用率の両方を用いた実験では，通信レイテンシに大きなゆらぎが発生した．帯域幅使用率を基準にした切り替え動作が通信レイテンシに負荷を与えてしまっているからだと考えられる．今後，通信レイテンシに負荷を与える要因を調べ，より安定した切り替え手法を検討する．

また，本選択手法は既存の通信フレームワークに組み込むことで，サーバ接続に関わる開発を良い物にすることができると考えている．実現するために実際のアプリケーションを用いた動作確認をおこない，更に最適な選択機構を開発する必要がある．

参考文献

[1] VMware:“ Liota ”，2018 年 11 月 23 日，https://github.com/vmware/liota

[2] ns-3:“ ns-3 tutorial ”，2019 年 1 月 11 日，https://www.nsnam.org/docs/release/3.29/tutorial/ns-3-tutorial.pdf

[3] “ pyping ”，2018 年 12 月 28 日，https://pypi.org/project/pyping

[4] “ SNMP library for Python ”，2019 年 1 月 3 日，http://snmplabs.com/pysnmp/index.html