Vol.2020-DPS-182 No.56
Vol.2020-CSEC-88 No.56
2020/3/13

クラウドとIoTの統合における センサーデータ分散管理方法

高木 優希[1,a)]　串田 高幸[1]

概要：
クラウドコンピューティング (以下「クラウド」と称する) と IoT は異なる技術であるが, 近年, 両者を統合する需要が高まっている. そこで, センサーデータの仕分けや管理を自動化すると同時に, アクセスに関する透過性向上を目的とする IoT 向けマルチクラウドソリューションの開発を行った. センサーの優先度とストレージ容量の降順に従ってデータを自動的に配置している. また, クラウドと IoT デバイスの間に Web サーバを設置することで, クラウドの分散管理を抽象化し, アクセスに関する透過性を向上させた. 検証方法として, 既存の IoT プラットフォームとの比較を行った. この研究の結果, 本実験の環境下では自動仕分けの精度が 100.0 %であり, ケーススタディを重ねることで全ての環境で対応が可能となる. 既存のプラットフォームと比較し, 複数のクラウドに対応する拡張性, 透過性の向上が実現した.

1. はじめに

はじめに, クラウド, IoT, クラウドと IoT の統合の考え方をそれぞれ順に述べる. まず, クラウドの考え方として, 米国国立標準技術研究所によれば, 「共用の構成可能なコンピューティングリソースの集積に, どこからでも, 簡便に, 必要に応じて, ネットワーク経由でアクセスすることを可能とするモデルであり, 最小限の利用手続きまたはサービスプロバイダとのやり取りで速やかに割り当てられ提供されるもの」である [1]. 今日では, サービスモデルや実装モデルが多様化され, 発展を続けている. サービスモデルでは, サーバ管理をユーザーが考慮せず使用可能方向へと進んでいる. そのため, Serverless as a Service というモデルが生み出されつつある. 実装モデルでは, 単一クラウドではセキュリティ・信頼性のリスク上の観点から複数のクラウドを併用するマルチクラウドというモデルが生み出された.

次に, 日本における IoT の考え方として, 特定通信・放送開発事業実施円滑化法によれば, 「インターネットに多様かつ多数の物が接続され, 及びそれらの物から送信され, 又はそれらの物に送信される大量の情報の円滑な流通」とされている. 総務省の情報白書においても 2015 年には「自動車, 家電, ロボット, 施設などあらゆるモノがインターネットにつながり, 情報のやり取りをすることで, モノのデータ化やそれに基づく自動化等が進展し, 新たな付加価値を生み出すというもの」としている [2]. 2019 年度情報白書によれば, 自動車および輸送機器, 医療, 産業用途での IoT の高成長が見込まれる. これらはそれぞれ, コネクテッドカーの普及により, IoT 化の進展が見込まれるという点, デジタルヘルスケアの市場が拡大しているという点, スマート工場やスマートシティが拡大しているという点から予測されている.

これらクラウドおよび IoT の技術を組み合わせることで, それぞれの強みを生かし, お互いの短所を補うことで, ストレージ容量や処理能力, セキュリティ上のリスクといった技術的制約を解決できる. この研究では, これをクラウドと IoT の統合の考え方とする. IoT におけるデバイスでは, ストレージや処理能力が限られているため, 全てのデータを保存することはできない. 対処方法としては, 外部に一度保存し, そこから情報を抽出し, 演算処理をする必要がある. 一方, クラウドは, 実質的無制限のストレージと処理能力を備えた巨大なネットワークを提供している. また, 多種多様なデバイスからの動的なデータ統合を可能にする柔軟性と堅牢性も提供している. したがって, 常に単体としての機能を一切考慮せず IoT デバイスを使用することが可能になる. このような IoT とクラウドの統合は, 近年になって新しい概念として提案されるようになった. Aazam らは両者の統合およびその連係動作を Cloud of Things と

[1] Tokyo University of Technology
Cloud and Distributed Systems Laboratory, Japan
[a)] c0117178@edu.teu.ac.jp

した [3]. また, Botta らは統合に関する考え方を CloudIoT とした [4]. 2 章ではこのような新しい概念についての関連研究と, 一部に特化した先行研究について紹介する. 2 章によって見えてくる課題は, 3 章で詳しく記載する. 4 章では, この論文で提案するモデルについて詳しく記載する. この論文では, 外部ストレージとして各社クラウドプロバイダーの VPS 環境を利用し, データベース (以下「DB」と称する) およびユーザーインタフェース, 管理プログラムを構築した. 5 章では実装部分について詳しく記載する. 検証方法として, 異なるネットワーク上に設定した Raspberry Pi 端末からセンサーデータを送信することを採用した. 6 章では 4 章, 5 章によって得られた結果から評価を行った. 評価方法として, データフロー把握の可否およびユーザビリティ度合いを採用した. 先行研究や既存ツールとの比較を行った. 7 章では, 3 章の課題について 6 章の評価を元に考察を行った. 最後に 8 章では結論と展望を述べた.

2. 先行研究・関連研究

この章では, この論文で述べるクラウドと IoT の統合に関して, 同様に研究を行うものについて取り上げる. まず, クラウドと IoT の統合に関する調査を行ったものについて取り上げる. Botta らはクラウドと IoT の統合について, CloudIoT という考え方とし, 統合のための課題や統合後の課題について言及した [4]. Qabil らは IoT とクラウドのアーキテクチャ及び両者を統合した概念 Cloud of things(CoT) を調査し, 問題点について言及した [5]. Biswas らはクラウドと IoT の統合における課題に対処する IoT 中心のクラウドスマートインフラストラクチャについて調査を行った [6]. Celesti らはクラウドと IoT の統合サービス IoT as a Service(IoTaaS) の進化についての考察と, 一般的な 3 層 IoT クラウドフェデレーションアーキテクチャの調査と課題について言及した [7]. Siddiqa らはビッグデータ管理に焦点を当て, ストレージ, 処理, セキュリティに重点を置き, 実現可能な管理手法を調査した [8]. 次に, 独自の開発を行ったものについて取り上げる. Persson らは記述, 接続, デプロイ, 管理の 4 つの定義に基づいた分散プラットフォーム Calvin を提案した [9]. 彼らはアプリケーションの開発に着目し, 作成, 展開, 管理についての課題を取り上げ, 解決することに尽力した. Liu らはクラウド及び IoT データでの認証システムベースのデータ整合性検証手法に関する分析を行った [10]. 彼らは, 特にセキュリティにおける整合性に着目し, 新規手法を提案することで課題を解決した. Jaradat らはスマートグリッド分野におけるスマートセンサーネットワークとビッグデータの管理手法に着目し, 言及をしている [11]. Baker らは IoT におけるマイクロサービスを統合し, E2C2 と呼ばれる新しいマルチクラウド IoT サービス構成アルゴリズムを開発した [12]. マルチクラウド向けの構成はマイクロサービスにはなかった手法であり, 個のデータに対する扱い方は今回の研究に取り入れている. Li らは IoT ソリューションプロバイダーがサービスを効率的に提供し, 継続的に拡張をするために不可欠な新しい IoT PaaS フレームワークを提案した [13]. Lea らは IoT ハブをスマートシティ PaaS として一般化するために, マルチクラウドおよびハイブリッドクラウドでの実現手法の提案を行った [14]. Sajid らはクラウドサービスを利用した IoT と物理的サイバーシステム (CPS) における重要なインフラストラクチャのセキュリティ上の課題に焦点を当て, 改善および維持するための提案を行った [15]. Jayaraman らはマルチクラウド環境の全ての IoT サービスに対してセマンティック技術を利用する階層型データ処理アーキテクチャを提案した [16]. どの論文においても, クラウドと IoT の統合に関する課題を解決しているが, アクセスに関する透過性をはじめとした 3 章で挙げる拡張性と透過性, ストレージと割り当てに対する課題は解決されていない. 本研究も同様に, 今後のクラウドと IoT の統合における問題点および課題の解決に貢献することを期待する.

3. 課題

この章では, 2 章で挙げた先行研究・関連研究および調査にしたがって, この論文で取り上げる現状の課題を 3 つ挙げる.

3.1 拡張性と透過性

マルチクラウドに対応させるためには, 拡張性は重要である [4]. この論文での拡張性とは, 単一 DB でデータを管理するだけでなく, 複数の DB での管理においても同様に対応できることを指す. 大手クラウドベンダーでは, 自社のプライベートクラウド環境を前提とした IoT プラットフォームや, それに付随する IoT 向けソフトウェアを提供している. また, 既存のプラットフォームでは, クラウドと IoT デバイスのエンドツーエンドの構成が多く, 複数のクラウドに対応しないことが多い. これらから拡張性がないといえる. また, 拡張性と同時に透過性も重要となる. この論文での透過性とは, データを管理する SQL DB システムが複数の DB から構成されている場合でも, ユーザーに意識させないことを指す. つまり, ユーザーに要求する処理が変化しないということである. データが複数の DB に分割され保存される場合やその保存先のクラウドや DB が変更となった場合でも, ユーザーは常に同じ方法でアクセスが可能ということである. 通常 IoT デバイスとクラウドを直接結び付けるエッジコンピューティングのような構成の場合, エンドツーエンドであり, クラウド上の SQL DB の数や IoT デバイスの数が増減する場合には, ユーザーにより多くの手順が発生する, あるいはデータが分割管理される場合には最初から構成を見直さなければならない. そのため, ユーザに対して構成を意識させると同時に, ユーザー

に要求する処理が変化・増加し, 透過性が低下する.

3.2 ストレージと割り当て

複数の IoT デバイスから送信されるデータを保存するためには, ストレージ容量が十分に備わっている必要がある. IoT デバイスでは, ユーザーがストレージを追加しない限りはストレージ容量の増加はできない. そのため, IoT デバイスに大容量のストレージを予め備える, あるいはクラウドのような外部保存先を造る必要がある. 一方クラウドでは, Amazon Web Service(以下「AWS」と称する) で導入されている EBS のように, ブロック単位でストレージを管理する, あるいは自動的にスケーリングする仕組みがある [4][6]. つまり, クラウドサービスプロバイダーがストレージを管理するため, ユーザーの観点からすれば, ストレージ容量は実質無制限となる. ここで, IoT デバイスから送信されたデータがクラウド上のストレージを要求するため, ストレージの割り当てが課題となる [3]. 特定の IoT デバイスがクラウド上のストレージを必要する量は一意に決定するのは難しいからである. IoT デバイスとその使用目的, データ生成のタイプ, 量, 頻度に応じて, ストレージの割り当てを自動的に決定する必要がある. 現在, これらは手動で行われていることが多く, ユーザーへの負荷が増加する. 単一クラウド以上の構成において, 自動的にデータを管理するシステムが必要である.

4. 提案モデル

この章では, 提案するモデルおよび構成について述べる. 全体的な構成図を図 1 に示す.

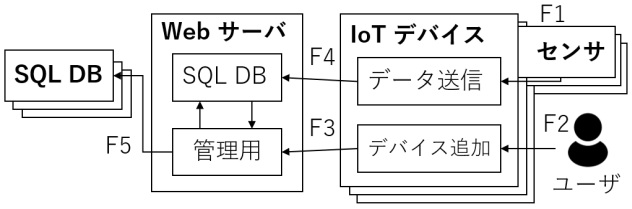

図 1 提案モデルの全体構成図

IoT デバイスでセンサから測定値を取得し, Web サーバへ送信することで, 各センサの優先順位およびストレージ容量の降順に, クラウド上の DB へ自動的に格納する. ここで, 図 1 の SQL DB はクラウド上にある DB を指し, Web サーバ内の SQL DB は優先順位やマシン情報, キャッシュデータを格納するものである. この研究では複数の SQL DB が複数のクラウドに分散管理されることを前提としている. Web サーバの役割は, 1 つ以上のクラウドで構成される複数の DB に対応させると同時に, 透過性を向上させることである. Web サーバを設けることで, センサやクラウドの個数に依存しないモデルが構築可能となる. さらに, 図 1 の Web サーバを複数個設置することで, IoT デバイスの数が増加した場合にも対応ができる. 図 1 における具体的な流れについて説明を行う.

図 1 の流れ

(F1) センサーからの測定値を IoT デバイスで取得可能にするように設定を行う
(F2) 初期設定として, IoT デバイスの追加を行う
(F3) Web サーバ上の DB に IoT デバイスの数および優先順位を記録する
(F4) IoT デバイスから Web サーバへセンサから取得した測定値が送信される
(F5) Web サーバから格納先 SQL DB へ (F4) データが転送される

(F2) における初期設定や (F3) の記録とは, 新規追加を行うセンサーの種類, 個数, 優先順位といったデータを Web サーバ上へ送信することを指す. 優先順位が故意に指定されない場合は, センサーの個数が多いほど優先順位が高くなることとした. この研究では, 全てのセンサーが一定間隔ごとに同じ量のデータを送信すると仮定している. センサーの個数が増えるほど, 送信されるデータ量が増加するため, 要求されるクラウドの SQL DB のストレージ量も増加する. ここで, この研究で使用した優先順位をルール 1 に示す. ルール 1 によって, センサーごとに明確な優先順位付けを行うためである. 時系列データである点や測定値の変動が発生しやすい順に設定をしている. これらによって, Web サーバに各センサーデータを管理し, ユーザーに対する負荷を軽減させるとともに, センサーの個数が増減した場合でも柔軟に反映させることを可能とする. (F5) に関して, 自動的な処理を行うために, 大きく分けて 2 つの仕組みを取り入れた. まず, クラウドの自動選択である. 各クラウドから一定期間ごとにマシン情報を取得し, Web サーバ上に記録を行い, その情報を元に優先順位付けを行う. 優先順位について, ルール 2 に示す.

ルール 1 : センサーデータの優先順位

1. 気温データ
2. 湿度データ
3. 気圧データ
4. 高度データ
5. その他センサーデータ

ルール 2 : マシン情報 (SQL DB) の優先順位

1. 利用可能なストレージ容量
2. 全ストレージ中利用可能なストレージ容量の割合
3. メモリ容量
4. CPU コア数

主に SQL DB を使用するため, 利用可能なストレージ容量が大きいほど優先順位は上がる. 複数の DB と比較を

行うため, 単なるストレージ容量との比較のほかに, 全ストレージ中利用可能なストレージ容量の割合を考慮する. また, よりデータ処理を円滑に行うために, メモリ容量やCPU コア数も考慮する. さらに, IoT デバイスが 1 つ追加される毎に, 1GB のストレージ容量を消費すると仮定し, 予め現在の空き容量から減少させる. これらによって, 単一の DB に偏ることのない分散管理が可能となり, 単一のクラウドに偏りにくくなる. どのセンサーデータがどのクラウドに格納されているかが明確に把握できるというメリットがある.

次に, データ仕分けの自動化である. データ仕分け方法として, 2 つ提案を行う.

第一に, ラベリングを行う手法である. 構成図を図 2 に示す. ラベリング手法では, IoT デバイスと Web サーバにラベル付けのためのラベルデータセットを設置する. 1 日ごとに同期を行い, 設定を統一化させることによって, 整合性を保つことが実現可能である. Web サーバに送信された後, このラベルを元にキャッシュ用 DB で照会され, 送信先 DB が決定する.

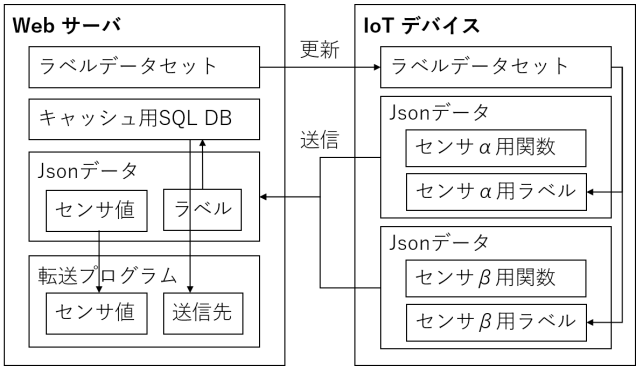

図 2 ラベリング手法の構成図

第二に, 機械学習を用いた手法について提案を行う. この研究では, 実験に使用するデータ数が少ないが, 教師ありの事前学習が可能である点を考慮し, k-近傍法による分類を行う. k-近傍法による分類手法を行う上での構成図を図 3 に示す. 予め学習用のプログラムを実行し, k-近傍法モデルを作成し, ここにセンサ値を入力することで, 結果が出力される. ラベリング手法と同様に, 出力結果のラベルは照会され, 送信先が決定する.

ラベリング手法では, 必ず期待通りの分類が可能な反面, センサーごとにユーザー入力を必要としている. このユーザー入力数をさらに減少させることを可能とするのが, 機械学習を用いた手法である. しかしながら, 分類の精度はデータ量に依存するため, 一定期間ごとに学習を繰り返す必要がある. それぞれの特徴を表 1 に示す. 表 1 では, ラベリング手法と機械学習手法それぞれの機能について比較を行った. 両者ともデータをユーザーが手動で分類する必要はなく, Web サーバ内でデータの自動分類が可能である. ラベリング手法では, ラベルに従って分類されるため, 必ず期待通りに仕分けが行われるため, ラベル分類精度は 100.0 %である. しかし, IoT デバイス側でデータに対してラベル付けを行うため, 初期設定としてユーザー入力処理を求める必要がある. さらに, Web サーバと IoT デバイス間でラベルデータを統一化する必要があるため, 同期が必要である. 一方, 機械学習手法では, ユーザー入力や同期を行う必要はない. しかし, 学習が必要となるため, 事前に学習を行うためのセンサーデータが必要である. また, ラベル分類精度は学習量に依存し, 必ず期待通りの結果になるとは限らない.

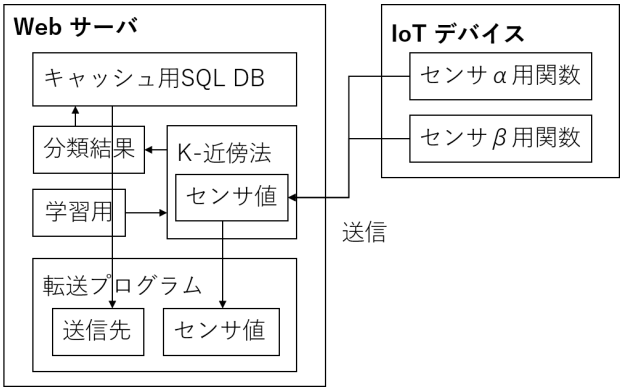

図 3 k-近傍法を用いた手法の構成図

表 1 データ仕分け提案モデルの比較

機能	ラベリング手法	機械学習手法
データの自動分類	○	○
ラベル分類精度	○	△
事前データ不要	○	×
ユーザー入力不要	×	○
同期不要	×	○

5. 実装

5.1 実験環境

本研究で使用した IoT デバイス, Web サーバ, クラウドについてそれぞれ記す. この研究では, 各種センサーデータで検証実験を行うため, 単一のコンピュータで複数のセンサーを付けることが可能な Raspberry Pi を IoT デバイスとする. センサーには温度, 湿度, 気圧, 高度といった一般的なセンサーデータを取得できるものを採用した. OS は, パッケージ管理のし易さを重視し, Linux の Debian 系ディストリビューションである Ubuntu や Raspbian を採用した.

IoT デバイス

Raspberry Pi 3 model B+と Raspberry Pi Zero を使用した. OS は Raspbian10.2 を採用, センサーには ADT7410, DHT11, BMP085 を使用した. また, センサーから値を取

得及びデータ送信に使用するプログラムは Python3.7.3 を使用している.

Web サーバ

Google Cloud Platform(以下「GCP」と称する) の Compute Engine を利用し, VM インスタンスを作成した. OS は Ubuntu 18.04 LTS を採用し, データを送受信するプログラムは PHP7.2.24 を使用している.

クラウド (SQL DB)

クラウドサービスとして主流とされる GCP, AWS, Microsoft Azure(以下「Azure」と称する) を使用した. 実験環境では, DB は全て MySQL を使用した. GCP は Web Server 同様に VM インスタンスを作成し, OS は Ubuntu 18.04 LTS を採用した. AWS では 2 パターンを作成した. 一方を, Amazon EC2 において GCP と同様に VM インスタンスを作成し, OS は Amazon Linux 2 を採用した. もう一方を, Amazon RDS で MySQL DB を作成した. Azure も同様に 2 パターンを作成した. 一方を, Virtual Machine から OS が Ubuntu 16.04 LTS の VM インスタンスを作成した. もう一方を, Azure Database for MySQL を利用し, DB を作成した.

5.2 IoT デバイス側の処理

IoT デバイス側では, ユーザーが IoT デバイスに各種センサーを取り付けた後, センサーの種類が何かをユーザーが入力を行う. センサーの種類として入力できるものは, センサーデータとして一般的である温度, 湿度, 気圧, 高度の 4 種類とした. この 4 種類以外のデータが送信された場合には, その他データとして処理を行う. 図 1 の (F2) の実装方法として, まず, 対話方式の入力を行う. 次に, IoT デバイスから送信されるデータの種類や使用されているセンサーの数を別ファイルに保存する. その後, Web サーバへ送信し, DB に保存する. このユーザーからの入力情報を元に, ルール 1 に従い各センサーデータに優先順位を設定する. この研究では, 使用したセンサーの数が多い順としている. これは, センサー数が増加することにより, 1 日あたりに要求するストレージ数が線形に増加するためである. この優先度は, IoT デバイス側の設定ファイルおよび Web サーバ上の DB によって管理されている. IoT デバイスを追加および削除する際には, IoT デバイスと Web サーバどちらでも対応可能であるため, 柔軟な運用が可能である. Web サーバに送信された優先度とセンサーの対応付けの情報は, クラウドの選択に利用される. つまり, 優先度の高いセンサーデータから順にストレージ容量大きいクラウドが選択されていくということである.

5.3 クラウド側の処理

図 1 のように, IoT デバイスから送信されたセンサーデータは, キャッシュ用の DB に保存される. また, 各クラウドから現在のストレージ容量を毎日送信させ, キャッシュ用の DB に保存する. これらのマシン情報を元に, ルール 2 に基づいてどのクラウドに IoT デバイスを割り当てるかを判断する. ルール 2 は, データ保存に関する情報ほど優先度を高く設定している. センサーデータは蓄積され, 膨大なデータとなることが予想されるため, ストレージに関わるものほど優先度が高い. つまり, マシン情報の最終更新時の状態で最もストレージ容量に十分な空きがあるものが選択される. この時, 登録済みのセンサーの種類とクラウドの対応表が DB 内で作成され, 以降のセンサーデータ受信時に使用される. 対応表が作成され, クラウドが割り当てられた後, IoT デバイス側からデータを受信するようになる. この時, 現在時刻に加え, 前後 1 時間の 3 つのデータを 7 日間分取得し, 最大値と最小値を閾値とする. これは, 時系列データかつ値のばらつきが最も大きいセンサーを基準に行っているためである. なお, この実験では, 1 時間毎にデータを送信することとしている. ここで, この研究におけるばらつきが大きいとは, 統計学で用いられる変動係数が大きいことを指す. 標準偏差を平均値で割ることによって, ばらつきを数値として求めた. この論文では小数点以下 4 桁で示す. 時系列データである気温・湿度・気圧についての変動係数の比較を表 2 に示す.

表 2 2020/1/16〜2020/2/10 の取得データ数と変動係数

種類	気温データ	湿度データ	気圧データ
データ数	623	596	627
変動係数	0.1734	0.1300	0.0075

表 2 から分かるように, 2020 年の 1 月 16 日から 2 月 10 日の 25 日間の計測では, 気温データが最も数値が大きいため, 最もばらつきが大きいことがわかる. 気温データは湿度データの約 1.3 倍, 気圧データの約 2.3 倍ばらつきが大きいということである. さらに, 気温データのうち, 最もばらつきの大きい期間を Web サーバ内の DB にキャッシュする期間とすることで, 閾値を計算する際に精度を向上させる. 気温データの 1 週間以内の変動係数の比較を表 3 に示す.

表 3 一定期間ごとの気温データの変動係数比較

期間	最大値	最小値	平均値
2 日ごと	0.2350	0.0540	0.1445
3 日ごと	0.2056	0.0579	0.1318
4 日ごと	0.2029	0.0777	0.1403
5 日ごと	0.1885	0.0881	0.1383
6 日ごと	0.2018	0.0868	0.1443
7 日ごと	0.2020	0.0997	0.1508

表 3 から, 変動係数の最小値 0.0997 および平均値 0.1508 が最も高いという点から, ばらつきが最も大きくなるのは 7 日ごとである. これらの結果をもとに, 1 週間分のデータを保持し, キャッシュとして使用する. 閾値である最大値・

最小値の範囲内であれば正常値となり, 保存用のテーブルに格納される. 範囲外であれば, 例外用のテーブルに格納され, Slack に通知が送信される. その後, ラベリングおよび機械学習の手法によってデータが分類され, 各データに合った SQL DB へ送信される.

5.4 ラベリング手法

ラベリングモデルでは事前にユーザーが IoT デバイス側でプログラムを実行し, ラベルの設定を行う. この論文でのラベルとは, ルール 1 のように, センサーの種類に合わせて割り振る属性値のことである. また, 設定とはそれぞれのセンサーごとにラベルを割り当てることである. 実際には, IoT デバイスの初期設定時に登録されているセンサー種類に合わせて付与されたラベルが測定値と共に Web サーバへ送信される. 具体的な例を図 4 に示す.

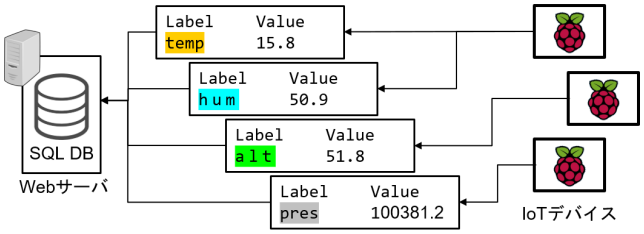

図 4 ラベリング手法の例

複数台の IoT デバイスからデータが送信, また, 1 台の IoT デバイスで複数のセンサーデータを送信することも考慮している. 各 IoT デバイスは, Web サーバに対し, ラベルとセンサ値を一緒に送信している. ラベルのデータセットについては, 図 2 の様に, IoT デバイス側で cron を実行し, Web サーバからラベルデータを取得することで, 整合性を維持している. IoT デバイス側が自動で行うことで, IoT デバイス数の増加した場合でも Web サーバの管理が困難にならないようにするためである. この動作を 1 日 1 回の頻度で行うことで, Web サーバと IoT デバイスの両方で最新のラベルデータを共有することが可能となる.

5.5 k-近傍法による分類手法

k-近傍法による分類手法では, ラベリング手法のようにラベルと一緒に事前に学習を行う. 教師あり学習により, 新しいデータが入力された際にどのラベルに近いかを予測させるためである. データは特徴量化され, パラメータとして与えられる k 個の最近傍が標本として選択される. この k 個の標本のラベルの多数決が行われ, 結果が出力される. 具体例を図 5 に示す.

教師あり学習を行うため, 正解ラベルと値をセットにしたデータを与え, 学習を行う. IoT デバイスから送信されたデータ (図 5 では 51.8) に対して分類を行い, k 個の最近傍のデータ群で最も一般的なラベルを割り当てる. この研究では, 使用言語は Python, ライブラリは scikit-learn を使用し, k-近傍法モデルを構築した. 標本数 k にあたるパラメータ n_neighbors を 5 と設定した. これは, センサーの種類がその他データを含めて 5 種類であることと, 標本数を小さく設定することで, 誤った判定をしないようにするためである. ラベリング手法と同様に, 分類が終了したセンサーデータは適切なクラウドへ送信される. この実験における k-近傍法を用いた精度を図 6 に示す.

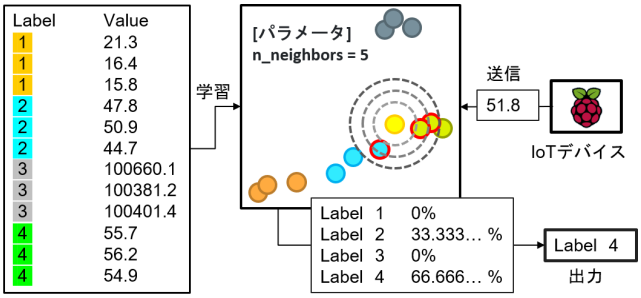

図 5 k-近傍法の例

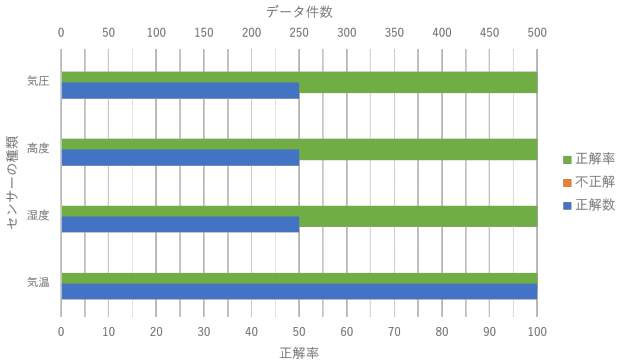

図 6 k-近傍法による仕分け精度

2020 年 1 月 16 日から ADT7410, DHT11, BMP085 を用いて毎時東京都八王子市の気温, 湿度, 気圧, 高度のセンサ値を取得し続け, これを学習用データセットとして用いた. 学習用データは, 気温データ 2111 件, 湿度データ 718 件, 気圧データ 715 件, 高度データ 714 件, 合計 4258 件を使用した. 学習用データから最大値と最小値を求め, 最小値-1 から最大値+1 の間で疑似乱数を生成し, テストデータを作成した. 気温データ 500 件, 湿度, 気圧, 高度のデータを各 250 件ずつ作成し, 仕分け精度の実験を行った. 図 6 に示すように, 正解率は 100.0 %であり, 4 種類のセンサーにしっかりと分類されていることが分かる.

表 2 からも分かるように, 湿度データのばらつきも多いため, 高度データの値と近くなることがある. Web サーバ内にある SQL DB のキャッシュデータを用いて同様の実験を行うと, 正解率 80.2 %であった. これは, 異なるセンサーの種類であっても, 測定値が近い値になる場合がある

からである．しかし, 各クラウドの SQL DB 内にあるデータを用いて学習を行う場合, 図 6 の様に精度が十分に得られる．これは, キャッシュデータの際の 2 倍以上のデータ量を学習させ, k-近傍法における最近傍のデータ群が変化したからであると考えられる．毎日 1 回以上の頻度で学習に用いるデータを変更し, 事前学習済みモデルを作成しておくことで精度の維持が可能となる．

6. 評価

実験の検証および評価として, 2 つの観点から述べる．

第一に, 提案したモデルについて, 実験の結果を元に課題に対する評価を行う．ラベリングモデルのような事前にデータセットを用意する場合, 提案モデルでは, IoT デバイスを利用するユーザーに対して多少の手間をかけることになるが, 必ず正確な仕分けが可能である．k-近傍法による分類では, ユーザーはデータを Web インタフェースに対して送信するだけで良い．しかし, 分類は学習量に依存するため, 必ず正しい仕分けが成されるわけではないが, 図 6 の様にこの研究では精度は十分に示された．これは, 元々多くのデータを必要としない k-近傍法というアルゴリズムを用いた点や, 東京都八王子市という限定的な地域である点, 標本数を 5 と設定した点が要因であると考えられる．世界中のデータを学習させることで, 精度がさらに向上し, 全世界に対応できると予想される．この研究では, データの格納場所をユーザーが任意で決定するため, クラウドの SQL DB の数が増加した場合でも, ユーザーへ求める処理は変化しないという透過性が最も優れている．既存モデルの多くは, データの流れには着目せず, データそのものに対して監視を行うからである．また, IoT 向けプラットフォームとして, マルチクラウドを前提とした構成であることから, 拡張性にも優れている．しかし, リアルタイムで変更が反映されない点や SQL DB の数が増加する際は設定ファイルを書き換えなければならないといった柔軟性, 学習コストの高さに課題が残っている．

第二に, 既存のプラットフォームとの比較を行い, 課題に対する評価を行う．既存のプラットフォームやアプリケーションを使用し, 提案モデルとの違いを比較した．この論文で挙げた課題に加え, 提案モデルとの明確な違いについて表にまとめ, 表 4 に示す．提案モデルでは, 複数のクラウド上の SQL DB および IoT デバイスを前提とした構成であり, かつストレージの自動割り当てを可能としている．IoT Toolkit は自動処理を行うシステムがないが, クラウドや IoT デバイスに対する柔軟な対応が可能であった．OpenIoT はオープンソースのプラットフォームであり, 拡張性や柔軟性が高い特徴がある．しかし, 複数のクラウドが前提ではないため, 複数のクラウドに対応させる仕組みを作成する必要がある．また, ThingsSpeak や CloudPlugs はエンドツーエンドのモデルであり, 自動化処理のような高度なシステムは実装されていなかった．また, 複数のクラウドに対応させることが難しいため, ThingsSpeak や CloudPlugs を管理するプラットフォームが必要であると考えられる．AWS, Azure, GCP の IoT プラットフォームでは, 自社のクラウドをユーザーに提供することが前提であるために, 他社のクラウドとの連携や複数のクラウドを今回提案したモデルより容易に連携させることが難しい．提案モデルは, 既存のプラットフォームと比較すると, この研究で取り上げた課題に対しては最も解決に尽力している．

表 4 提案モデルと既存プラットフォームの比較

	既存クラウド	複数クラウド	複数 IoT デバイス	ストレージ自動割当
提案モデル	○	○	○	○
IoT Toolkit	○	○	○	×
OpenIoT	○	△	○	○
ThingsSpeak	○	△	○	×
CloudPlugs	○	×	○	△
AWS/Azure/GCP IoT	○	△	○	○

7. 考察

この章では, 3 章で挙げた拡張性と透過性, ストレージと割り当てに対し, 5 章で述べた結果と照らし合わせ, 考察を述べる．この研究の実験により, 3 章の課題が改善された．

7.1 拡張性と透過性

提案手法を用いたマルチクラウド環境において, クラウド数の増減によるユーザーに要求する処理の増加や透過性に関する影響はなかった．これは, 提案モデル (1) 自体が IoT デバイスや SQL DB の数に依存しない構成をしているからである．この研究で構成した Web サーバが単一であっても SQL DB や IoT デバイスの数が増加してもある程度は許容が可能である．ここで, Web サーバを複数台用意することで, 実質無制限まで許容が可能となる．これによって, 拡張性が保証されると同時にユーザーに要求する処理は一定以上に増加しないため, 透過性も保証される．

7.2 ストレージと割り当て

提案手法を用いてマルチクラウド環境を構築することで, 実質的無制限のストレージ容量となった．ユーザーが各クラウドのストレージ容量を増設あるいは, マルチクラウド構成におけるクラウド数の増加を行うことで, ストレージ容量がなくなることがはない．IoT デバイスをクラウドに接続することで, ユーザーは IoT デバイス上のストレージ容量を考慮する必要がなくなる．この研究においても, データの格納先はクラウドストレージであり, ローカル環境に格納先を作成する必要がない．また, この研究ではクラウドや DB の選択がリソースの割り当てに直結している．そのため, 柔軟性の高い自動的なリソースの割り当てが実現された．これは, マルチクラウドだけでなく, ハイブリッドクラウドにおいても実現が容易である．この研究では大ま

かなストレージに対するリソースの割り当てを行ったが, 単一のリソースをさらにパーティションで区画し, 効率的な分散アプローチと組み合わせることで, より効果を発揮する. 大まかな部分から詳細までユーザーに選択しを与えることでユーザビリティの向上につながるといえる.

8. おわりに

最後に, これまでの章をまとめ, 結論を述べる. この研究での環境下では, 3 章で挙げた拡張性と透過性, ストレージと割り当てについては解消されたといえる. これは, 7 章で述べたように, DB やクラウドの数が変化した場合でも, ユーザーに要求される処理は常に変化せず, 割り当ての自動化も実現ができた. さらに, データの仕分け精度も十分なものである. しかし, Web サーバを設けることによって, ネットワーク遅延やオーバヘッドといったクラウドサービスプロバイダーに依存する問題点が解消できないという点がある. このように, クラウドと IoT それぞれの技術のみで解決するのではなく, 両者を共に考慮した際に起こりうる問題に取り組むべきである. この研究のように, 複数のストレージを管理し, 最も効率かつ分散させる手法の研究に取り組むことで, 従来のオンプレミス環境での運用より遥かにデメリットの少ない運用が可能になる. また, 透過性については, クラウドを使用する以上, データの監視やデータフローの可視化といった別の手法を使用する以外に向上させることはできない. そのため, 新規プラットフォームの開発も検討していきたい. ストレージに関する課題や拡張性, 透過性に関する解決方法をさらに改善していくことで, クラウドと IoT の統合が促進されると予想される. この研究はその先駆けとして, IoT デバイスとクラウドの間に Web サーバを構築することで, ストレージ分散管理や割り当ての課題と拡張性, 透過性といった課題を解消した.

参考文献

[1] Mell, P. and Grance, T.: The NIST Definition of Cloud Computing, Technical report, National Institute of Standards and Technology (2011).
[2] Anonymous: WHITE PAPER, Ministry of Internal Affairs and Communications, Japan (2015).
[3] Aazam, M., Khan, I., Alsaffar, A. A. and nam Huh, E.: Cloud of Things: Integrating Internet of Things and cloud computing and the issues involved, *Proceedings of 2014 11th International Bhurban Conference on Applied Sciences and Technology (IBCAST) Islamabad, Pakistan, 14th - 18th January, 2014* (2014).
[4] Botta, A., Donato, W., Persico, V. and Pescapé, A.: Integration of Cloud computing and Internet of Things: A survey, *Future Generation Computer Systems*, Vol. 56, pp. 684–700 (2016).
[5] Qabil, S., Waheed, U., Awan, S. M., Mansoor, Y. and Khan, M. A.: A Survey on Emerging Integration of Cloud Computing and Internet of Things, *2019 International Conference on Information Science and Communication Technology (ICISCT)* (2019).
[6] Biswas, A. R. and Giaffreda, R.: IoT and cloud convergence: Opportunities and challenges, *2014 IEEE World Forum on Internet of Things (WF-IoT)* (2014).
[7] Celesti, A., Fazio, M., Giacobbe, M., Puliafito, A. and Villari, M.: Characterizing Cloud Federation in IoT, *2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA)* (2016).
[8] Siddiqa, A., Hashem, I. A. T., Yaqoob, I., Marjani, M., Shamshirband, S., Gani, A. and Nasaruddin, F.: A survey of big data management: Taxonomy and state-of-the-art, *Journal of Network and Computer Applications*, Vol. 71, pp. 151–166 (2016).
[9] Persson, P. and Angelsmark, O.: Calvin – Merging Cloud and IoT, *Procedia Computer Science*, Vol. 52, pp. 210–217 (2015).
[10] Liu, C., Yang, C., Zhang, X. and Chen, J.: External integrity verification for outsourced big data in cloud and IoT: A big picture, *Future Generation Computer Systems*, Vol. 49, pp. 58–67 (2016).
[11] Jaradat, M., Jarrah, M., Bousselham, A. and Ayyoub, M. A.: The Internet of Energy: Smart Sensor Networks and Big Data Management for Smart Grid, *Procedia Computer Science*, Vol. 56, pp. 592–597 (2015).
[12] Baker, T., Asim, M., Tawfik, H., Aldawsari, B. and Buyya, R.: An energy-aware service composition algorithm for multiple cloud-based IoT applications, *Journal of Network and Computer Applications*, Vol. 89, pp. 96–108 (2017).
[13] Li, F., Voegler, M., Claessens, M. and Dustdar, S.: Efficient and Scalable IoT Service Delivery on Cloud, *2013 IEEE Sixth International Conference on Cloud Computing* (2013).
[14] Lea, R. and Blackstock, M.: City Hub: A Cloud-Based IoT Platform for Smart Cities, *2014 IEEE 6th International Conference on Cloud Computing Technology and Science* (2014).
[15] Sajid, A., Abbas, H. and Saleem, K.: Cloud-Assisted IoT-Based SCADA Systems Security: A Review of the State of the Art and Future Challenges, *IEEE Access*, Vol. 4 (2016).
[16] Jayaraman, P. P., Perera, C., Georgakopoulos, D., Dustdar, S., Thakker, D. and Ranjan, R.: Analytics - as - a - service in a multi - cloud environment through semantically - enabled hierarchical data processing, *Software : Practice and Experience*, Vol. 47, No. 8 (2016).