Vol.2021-CDS-31 No.5
2021/5/21

デバイス主体の自律分散処理サービスフレームワークの提案

中村 藍梨[1,a)]　内藤 克浩[2,b)]　山里 敬也[3,c)]

概要：本研究では，Internet of Things(IoT) を用いたスマートシティに注目する．これまで，広範囲に設置された IoT デバイスから情報をクラウドサーバに収集し，一括処理を行うことにより利活用する広域型スマートシティについての研究が進められている．一方で，地域的に収集した情報を現地で利活用する地域型スマートシティにおいて，既存の研究では，全ての情報がクラウドサーバに集約されるため，必要以上の処理負荷が発生するという問題がある．本論文では，地域型スマートシティに焦点をあてたデバイス主体の分散処理フレームワークの提案を行う．提案システムでは，近隣のデバイス同士が自律的に協調を図ることにより，資源を共有するとともに処理負荷を分散するサービス提供を可能とする．プロトタイプによる評価の結果，システムの基礎的動作を確認した．

1. はじめに

近年の IoT デバイスの増加に伴い，スマートシティが注目を集めている [1–3]．スマートシティは情報を活用し，生活の質を向上させることを目的とする概念であり，大きく 2 つに分類される．広範囲に設置された IoT デバイスから収集した情報を利活用する広域型スマートシティと，範囲を限定した地域内から収集した情報を現地で利活用する地域型スマートシティである．

スマートシティの現状として，広域型スマートシティについての研究が多く進められている [4–6]．広域型スマートシティは，情報をクラウドサーバに一括収集することにより，様々な分野の情報資源を効率的に処理することが可能である [7,8]．しかし情報の用途を考慮せず，全ての情報をクラウドサーバに収集した場合，クラウドサーバには必要以上の処理負荷が発生するという問題が生じる．解決案として，地域型スマートシティのようなデバイス同士の距離が近い場合には，クラウドに情報を収集せず，デバイス同士で直接通信を行う方法が挙げられる [9,10]．既存手法では，地域内のデバイスが相互に情報を交換するように設計を行うことが主流であるが，モビリティ等も着目される近年，IoT デバイスは必ずしも位置が固定されているとは限らない [11,12]．そのため，デバイスは周辺の他のデバイスを探索し，柔軟に連携を図る必要がある．

また，従来のデバイスはスマートシティの特定のサービス専用として設計されることが多く，目的外の用途のためにデバイスを使用することが困難である．例えば，ビニールハウス内部に侵入者検知用のカメラと野菜監視用のカメラがある場合，野菜監視用のカメラでは侵入者の検知がされない．同じ性能のカメラでも搭載するソフトウェアによって提供するサービスが異なり，野菜監視用のソフトウェアが搭載されているカメラは野菜監視専用として機能する．したがって，侵入者検知と野菜監視のサービスを提供するためには，専用のソフトウェアが搭載されたカメラを別々に設置する必要がある．これにより，同じ性能を持つデバイスが同一のエリアに重複して存在するという問題が生じる．解決案としてデバイスの機能を柔軟に更新することにより，様々な用途で活用可能とする方法が挙げられる．

本論文では，地域型スマートシティにおけるデバイス主体の分散処理フレームワークを提案する．本提案システムでは，通信範囲内のデバイス同士が相互に検出を行い，自律的に協調を図ることにより，デバイスが持つセンサやカメラ等の情報資源を共有する．そして，サービスの処理を分散することにより，クラウドを介さないサービスの提供を実現する．また，デバイスが提供するサービスを独立の機能として分割し，自由なサービスの切り替えを実現する

[1] 愛知工業大学大学院経営情報科学研究科
Aichi Institute of Technology Graduate School of Business Administration and Computer Science
Toyota, Aichi, 470-0392, Japan
[2] 愛知工業大学情報科学部
Aichi Institute of Technology Faculty of Information Science
Toyota, Aichi, 470-0392, Japan
[3] 名古屋大学 教養教育院 教養教育推進室
Education Support Center, Institute of Liberal Arts and Sciences, Nagoya University,
Chikusa, Nagoya 464-8601, Japan
[a)] airi-1@pluslab.org
[b)] naito@pluslab.org
[c)] yamazato@nagoya-u.jp

ことにより，1 つのデバイスを様々な用途で利用し，資源を効率的に活用することが可能となる．

提案システムのプロトタイプとして，監視カメラサービスをカメラ周辺に存在するユーザに提供するシステムの開発を行う．評価では，提案システムのプロトタイプが正常に動作することを確認する．また，サービスの要求から提供までにかかる時間を測定する．

2. 提案システムモデル

本提案システムは，サービスを提供する多数のデバイスと，様々なサービスを管理するクラウドにより構成される．センサを搭載したデバイスは周辺のデバイスと協調し，近隣のユーザに対してサービスの提供を行う．デバイスは内部のアプリケーションを定期的に更新し，様々なサービスを提供する．そのため，提案システムでは，サービスアプリケーションを開発し，サービスをクラウドにアップロードするサービス開発者と，デバイスを設置し提供可能なサービスを選択するデバイス管理者，サービスを提供されるユーザが存在する．ユーザが所持するサービスを受ける端末をユーザ端末と呼ぶ．本提案システムが実現すると，ショッピングモール等の比較的小規模なコミュニティにおいて店内カメラや車載カメラの情報を利用し，迷子探しサービスや防犯サービスを同時に提供することが可能となる．また，同様の車載カメラを利用することにより，交通道路において速度違反を犯した車両を検知し，周辺車両や人に対して速やかに警告を発信するサービスを提供することも可能である．

本研究では，デバイス主体の分散処理フレームワークを提案する．センサを搭載したデバイスが周辺デバイスを探索し，デバイスが存在する場合は独自のネットワークを一時的に築く．そして，独自のネットワーク内で各デバイスが保持するセンサから得た情報を共有する．取得可能な情報の組み合わせにより提供可能なサービスを選定し，近隣のユーザ端末に対してサービス提供を行う．1 つのデバイスに処理負荷が集中することのないよう，接続されているデバイス同士で処理を分散する．また，デバイスに搭載されたデータを特定の用途だけに使用せず，様々なサービスとして活用する．デバイス内部は機能ごとに分割することで，定期的にサービスアプリケーションの入れ替えを可能にする．

提案システムを実現するためには，デバイス同士を同一ネットワークに接続し，データの共有や処理を行う必要がある．また，デバイスが提供するサービスを自由に切り替える枠組みが必要となる．デバイス同士の連携において，インフラストラクチャモードのような，接続先デバイスを設置して複数のデバイスが接続する方式では，接続先デバイスが接続不可能になると，デバイス同士の全ての通信が切断される．そのため，デバイス同士の連携にはアドホックネットワークを利用し，デバイス同士の 1 対 1 の独自のネットワークを生成する．通信範囲内に入ったデバイス同士がアドホックネットワークを通じて互いの情報を共有する事により，自律的な連携を行う．またデバイス上で実行されるサービスは，サービスの独立性を担保するために，仮想環境上のサービスとして設計する．仮想環境は，動作の軽さや配布のしやすさから，コンテナと呼ばれる仮想環境の作成や配布，削除をすることが可能なソフトウェアである Docker を利用する．これによりサービス同士の干渉を防ぎ，サービスの自由な切り替えを実現する．

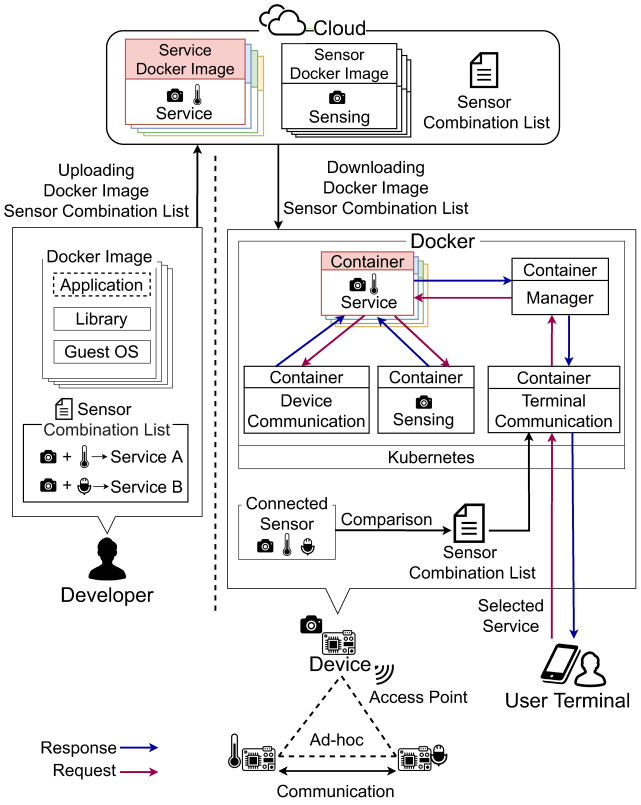

図 1 システムモデル

本研究の提案するシステムモデルを図 1 に示す．提案システムは主に 3 つの特徴を持つ．第一に，デバイス同士の相互検出による柔軟な分散協調サービスの実現である．デバイス同士が通信範囲内に入ると，自律的に連携を取り，現在提供可能なサービスを導き出す．第二に，デバイスのデータ処理方法である．デバイスには役割で細分化された 5 つのコンテナが存在し，データの処理を行う．また，サービスに直接関係する処理を行うサービスコンテナは，提供するサービスに応じて入れ替えを行い，サービスの柔軟な拡張を可能とする．第三に，開発者によるサービスの開発である．サービスの開発者は，サービスを実行する Docker イメージを用意し，コンテナ単位でサービスをクラウドにアップロードすることにより，サービス展開が可能となる．

2.1 デバイスの相互検出

各デバイスは独自のアドホックネットワークへ定期的にメッセージを送信することにより，デバイスが通信範囲内に入ると，相互のデバイスの存在を検出可能である．また，検出したデバイスに対してデバイスに搭載されているセンサの種類を共有するための通信を行うことにより，共有可能な資源を確認する．

デバイス間の連携における通信シーケンスを図 2 に示す．デバイスの位置は移動することも想定しているため，デバイスは一定時間ごとに UDP (User Datagram Protocol) 通信によるブロードキャストアドレスを指定し，同じネットワークに属している全ての機器に対して Hello パケットを送信する．これにより，現在周辺に存在するデバイスを定期的に認識する．Hello パケットには，送信元の IP アドレスとポート番号が含まれている．UDP 通信はコネクションレス型通信であり，一対多の通信を行うことが可能である．ブロードキャストのメッセージを受信したデバイスは，ブロードキャストの送信元に対して UDP 通信を用いて送信先の IP アドレスとポート番号を含めた ACK パケットを送信する．これにより，デバイスが接続可能な周辺デバイスを探知することが可能となる．

次に，デバイスは返信メッセージを受け取ると，SSL(Secure Sockets Layer)/TLS(Transport Layer Security) ハンドシェイクを用いてコネクションの確立を行い，搭載されているセンサの種類を共有する．センサには個人情報を含むデータも含まれるため，デバイスに搭載されているセンサの種類等の情報を秘匿にする必要がある．そのため，SSL/TLS により暗号化されたメッセージの送受信を行う．SSL/TLS ハンドシェイクを行ったデバイスは Sensor Request パケットを通信相手のデバイスに送信する．Sensor Request パケットには，パケットの種類の識別子が含まれる．Sensor Request パケットを受信したデバイスは，デバイスに搭載されているセンサの種類が含まれる Sensor Response パケットを返信する．

また，現在共有可能なセンサを用いて実行可能なサービスを判別するために， センサの組み合わせとサービスを紐付ける必要がある．そのため，センサの組み合わせによって生成されるサービスの一覧が登録されているセンサ組み合わせリストを用意する．そしてデバイスは，共有したセンサの種類とデバイス内で保持しているセンサ組み合わせリストを照合し，現在提供可能なサービスを算出する．デバイス上のセンサ組み合わせリストは，一定の時間間隔でクラウドを参照することにより更新を行い，最新の状態を維持する．

2.2 デバイスのデータ処理方法

サービスの独立性の実現および，機能のアップデートを容易にするために，デバイス内はユーザ端末通信，管理，

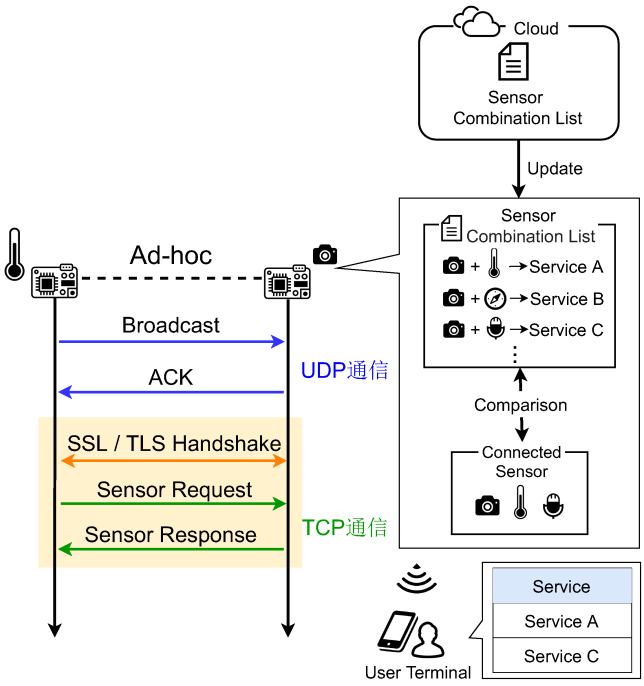

図 2 デバイス間の連携における通信シーケンス

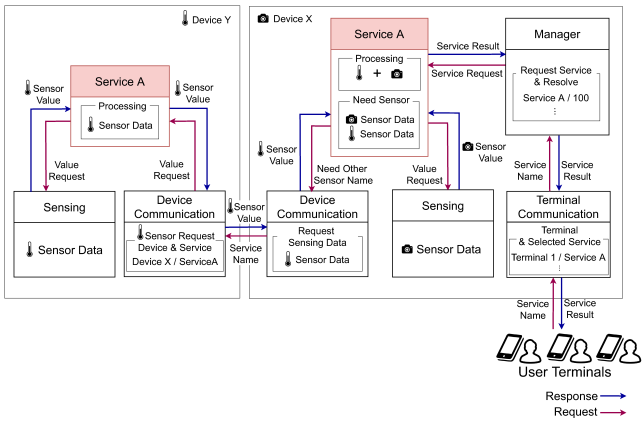

図 3 コンテナの内部構造

センシング，デバイス通信，サービスの 5 つに機能を細分化し，機能をコンテナに割り当てる．以下に詳細を示す．また，コンテナの内部構造を図 3 に示す．

- ユーザ端末通信コンテナ
  デバイスの相互検出時に算出した現在提供可能なサービス名はユーザ端末通信コンテナに送信される．同コンテナは，デバイスに接続するユーザ端末に対して現在提供可能なサービス名の一覧を送信し，ユーザ端末が選択したサービス名を管理コンテナへ転送する，また，管理コンテナから受信したサービス処理結果をユーザ端末に転送する．ユーザ端末通信コンテナでは，ユーザ端末と要求したサービス名を関連付けて管理する．
- 管理コンテナ

管理コンテナは，ユーザ端末通信コンテナが要求したサービスのコンテナにサービス要求を転送する．また，サービスコンテナからサービス処理結果を受信し，ユーザ端末通信コンテナに転送する．管理コンテナでは，要求されているサービス名と結果を関連付けて管理する．

- センシングコンテナ
  センシングコンテナは，デバイスに搭載されているセンサを制御し，センサ値を取得する．そのため，センシングコンテナの処理は搭載されているセンサに応じて異なる．サービスコンテナからセンサ値を要求されると値を取得し，サービスコンテナに送信する．
- デバイス通信コンテナ
  デバイス通信コンテナは，サービスコンテナの要求に応じて他のデバイスにセンサ値の要求を行い，返信されたセンサ値をサービスコンテナに送信する．また，他のデバイスからセンサの要求をされた場合は，サービスコンテナへセンサ値の要求を行い，要求結果を要求したデバイスに転送する．そのため，デバイス通信コンテナでは，要求されたデバイスと要求されたサービスを関連付けて管理する．
- サービスコンテナ
  サービスコンテナは，要求に応じてセンサ値の処理を行う．サービスコンテナが受信する処理要求は 2 通りある．1 つ目は，デバイス通信コンテナからのセンサ値要求である．センサ値要求を受信すると，要求されたサービスコンテナで，センサ値を加工する処理を行い，処理結果のセンサ値をデバイス通信コンテナに転送する．2 つ目は，管理コンテナからのサービス要求である．サービス要求を受信すると，1 つ以上のセンサ値を用いたサービス全体的な処理を行う．処理結果は管理コンテナに転送する．

2.3 サービス開発者によるサービスの作成

サービスの開発者はクラウド上にサービスで用いる Docker イメージおよび，センサ組み合わせリストを格納する．新規サービスの Docker イメージを追加する際にそのサービスと用いるセンサが，センサ組み合わせリストに自動で追加される．デバイスはコンテナ単位でサービスの入れ替えを行うため，開発者はコンテナをクラウド上にアップロードするのみでサービスの提供が可能である．

2.4 デバイス管理者によるサービスの選択

サービス開発者がクラウドにサービスをアップロードし，デバイスがサービスの取得を行うと，意図しない利用をされる可能性がある．そのため，デバイスを設置したデバイス管理者が取得可能なサービスコンテナの選択を行うことにより，任意のサービスのみをユーザ端末に提供する．

3. システム運用形態

3.1 サービスモデル

ユーザ端末はデバイスが提供する Wi-Fi アクセスポイントに接続し，アプリケーションを起動する．接続されたデバイスは，現在提供可能なサービスの一覧をユーザ端末に送信し，アプリケーション上で表示する．利用したいサービスを選択すると，デバイスのコンテナが実行され，ユーザ端末のアプリケーションに対してサービスの提供を行う．

3.2 コンテナの取得および実行

ユーザ端末が選択したサービスのコンテナが，デバイス内に保持されていない場合，クラウドから Docker ファイルの取得を行う．デバイスとクラウド間の通信では SSL/TLS を使用し，Docker ファイルとセンサ組み合わせリストの改ざん検知および盗聴の防止を行う．クラウドには Docker ファイルの形式でサービスのコンテナが保持されているため，デバイスは Docker ファイルの取得をした後 Docker イメージに展開し，Docker コンテナとして使用する．

3.3 コンテナの削除

Wi-Fi アクセスポイントに接続しているユーザ端末がサービスの要求を止めた場合は，プロセッサリソースおよびメモリリソースの抑圧を防ぐために，そのサービスを提供していたコンテナを停止する．そして現在サービス要求がされていないサービスは，Docker イメージとして一定時間保管する．これにより，再度サービスが要求された際にコンテナを起動することが可能である．サービスの要求がされない際にコンテナを保持すると，デバイスが保持するコンテナが増えるにつれて，稼働していないコンテナによりストレージリソースが抑圧されるため，コンテナは適当なタイミングにおいて削除する必要がある．保管期間が過ぎサービスを削除する場合は，最後にサービスの提供をした日付で Docker イメージを管理し，新しい Docker イメージがデバイスに追加される度に，最後にサービスの提供をした日時が古い Docker イメージを消去する．これにより，サービスの Docker イメージを取得するにつれて，長期間使用されないサービスの Docker イメージが蓄積されることによる，デバイスのストレージリソース不足を防ぐ．

4. 実装及び検証

4.1 実装環境

本研究で提案するサービスフレームワークとして，監視カメラを想定したプロトタイプの実装を行った．プロトタイプは，デバイスとユーザ端末，クラウドにより構成される．プロトタイプではカメラを搭載したデバイスに対してユーザ端末がサービス要求を行う．デバイスは必要に応じてコンテナをクラウドから取得する．そして，コンテナを

表 1　実装環境

デバイス	Raspberry Pi 4 Model B (Raspbian10.7)
クラウド	Raspberry Pi 4 Model B (Raspbian10.7)
カメラ	Logicool Webcam C270
無線 LAN 子機	Archer T4U AC1300
ユーザ端末	iPod touch (iOS version 12.5.1)

用いて搭載カメラの映像を取得し，ユーザ端末の iOS アプリケーションに表示を行う．今回はクラウドに相当するものとして Raspberry Pi を使用した．

実装環境を表 1 に示す．仮想環境として Docker Ver.20.10.1 を用いる．本研究で提案するシステムでは，デバイス間の連携を行うために，アドホックモードを使用し，デバイスとユーザ端末の通信を行うために，アクセスポイントを使用する．また，デバイスとクラウドの通信を行うために，インフラストラクチャモードを使用する．全ての通信機能をサポートするには，Raspberry Pi が持つ標準の無線 LAN モジュールではインタフェースが不足する．そのため無線 LAN 子機を接続し，ネットワークインタフェースの増設を行う．またカメラの映像を表示させるために，MJPG-Streamer を使用する．MJPG-Streamer はカメラから取得した映像を，Web ページを介してストリーミングする動画配信ソフトウェアである．

4.2　サービス提供アプリケーション

監視カメラサービスのプロトタイプでは，iPod touch から Device Raspberry Pi に対してリクエストを送信すると，Device Raspberry Pi の端末通信コンテナが受信する．端末通信コンテナはリクエストを受けたサービスのコンテナが存在するかを判別し，コンテナが存在する場合はユーザ端末に対してサービス提供可能パケットを送信する．サービスのコンテナは存在しないがサービスの Docker イメージが存在する場合は，Docker イメージを展開し，コンテナとして実行した後でユーザ端末に対してサービス提供可能パケットを送信する．サービスの Docker イメージが存在していない場合は，Cloud Raspberry Pi にリクエストを送信する．Cloud Raspberry Pi との間で SSL/TLS ハンドシェイクを行い，Cloud Raspberry Pi に保存されているサービスコンテナの Docker ファイルを受信する．端末通信コンテナは Docker ファイルを受診するとホスト OS にコピーし，ホスト OS で Docker イメージに展開後，Docker コンテナとして実行する．実行した後に，端末通信コンテナからユーザ端末に対してサービス提供可能パケットを送信する．

4.3　サービス受信アプリケーション

サービスの受信を行うための iOS アプリケーションを実装した．Device Raspberry Pi のアクセスポイントに iPod touch を接続し，アプリケーションを起動させる．画面には現在利用可能なサービス名がボタンとして表示されている．サービス名を選択すると，サービスの結果として，Device Raspberry Pi に搭載されているカメラの映像が表示される．

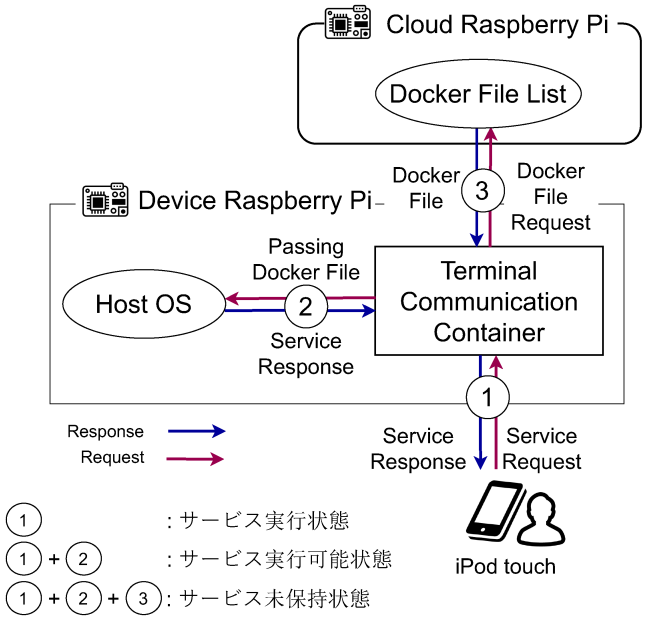

図 4　サービス提供の手順

表 2　サービス提供における処理時間の計測結果

	処理時間 [ms]
サービス実行状態	149
サービス実行可能状態	4,525
サービス未保持状態	221,672

4.3.1　サービス提供時間の測定

検証では，iPod touch がサービスの要求をした後に，Device Raspberry Pi に搭載されたカメラの映像が，iPod touch に出力されるまでの時間を測定する．今回の評価では，サービスコンテナが実行しているサービス実行状態，サービスコンテナは存在しないが，Docker イメージを保持しているサービス実行可能状態，Docker イメージを保持していないため，Cloud Raspberry Pi から Docker ファイルを取得する必要のあるサービス未保持状態の 3 通りについて各 10 回計測し，平均を求めた．サービス提供の手順を図 4 に示す．また，計測結果を表 2 に示す．サービス実行状態と比較し，サービス未保持状態は 221 秒程処理が遅い．サービス未保持状態は Docker ファイルの取得および展開，Docker イメージの実行が 98%を占めており，大半の時間を要していることがわかる．

4.3.2　内部処理時間の測定

サービス未保持状態において内部処理時間を各 10 回計測し，平均を求めた．内部処理の手順を図 5 に示す．内部処理では，Device Raspberry Pi において，iPod touch からのサービス要求を受信してから Cloud Raspberry Pi へ

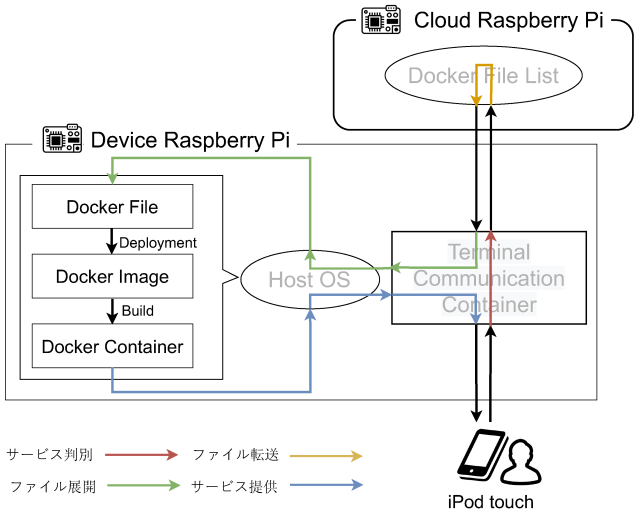

図 5 内部処理の手順

表 3 内部処理時間の計測結果

	処理時間 [μs]
サービス判断	197
ファイル転送	86,428
ファイル展開	1,288
サービス提供	182
total	88,095

Docker ファイルの要求を行うまでをサービス判断時間，Cloud Raspberry Pi において，Docker ファイルの要求を受信してからファイルの転送を行うまでをファイル転送時間，Device Raspberry Pi において，Docker ファイルを受信してからファイル展開を行うまでをファイル展開時間，Docker コンテナが起動してから iPod touch にサービス提供を行うまでをサービス提供時間と定義する．計測結果を表 3 に示す．サービス未保持状態における処理時間に内部処理はほとんど影響していないことがわかる．

5. まとめ

本研究では，地域型スマートシティにおけるデバイス主体の分散処理フレームワークの提案及び実装を行った．既存の問題であるクラウドに処理が集中する点を解決するために，クラウドを介さずにデバイス間で連携して処理を行う必要がある．本提案システムでは，デバイス同士が直接通信可能な距離に設置されている際に，デバイス間で自律的に探索や認識，協調を行う．これにより，クラウドへの処理負荷を軽減させることが可能である．プロトタイプとして，監視カメラを想定したサービスを実装した．検証では，サービスの提供において，サービス実行状態とサービス実行可能状態，サービス未保持状態に分類し，提供に必要な時間を測定した．また，Docker ファイルの受信から展開までの時間およびコンテナ立ち上げからサービス提供までの時間，内部処理にかかる時間も測定した．検証結果から，Docker ファイルの展開や Docker イメージの実行にかかる時間が大半であることを明らかにした．また，サービス提供にかかる時間において，内部処理時間はほとんど影響していないことも確認した．

謝辞 本研究の一部は JSPS 科研費 (21K11877) の助成を受けたものである．記して謝意を表する．

参考文献

[1] Arindam Giri, Subrata Dutta, Sarmistha Neogy, Keshav Dahal, and Zeeshan Pervez. Internet of things (iot): A survey on architecture, enabling technologies, applications and challenges. New York, NY, USA, 2017. Association for Computing Machinery.

[2] Charith Perera, Yongrui Qin, Julio C. Estrella, Stephan Reiff-Marganiec, and Athanasios V. Vasilakos. Fog computing for sustainable smart cities: A survey. Vol. 50, No. 3, 2017.

[3] Bowen Zhou and Rajkumar Buyya. Augmentation techniques for mobile cloud computing: A taxonomy, survey, and future directions. Vol. 51, No. 1, 2018.

[4] Hadi Habibzadeh, Cem Kaptan, Tolga Soyata, Burak Kantarci, and Azzedine Boukerche. Smart city system design: A comprehensive study of the application and data planes. Vol. 52, No. 2, 2019.

[5] Muhammad Usman, Mian Ahmad Jan, Xiangjian He, and Jinjun Chen. A survey on big multimedia data processing and management in smart cities. Vol. 52, No. 3, 2019.

[6] Vaia Moustaka, Athena Vakali, and Leonidas G. Anthopoulos. A systematic review for smart city data analytics. Vol. 51, No. 5, 2018.

[7] Toni Mastelic, Ariel Oleksiak, Holger Claussen, Ivona Brandic, Jean-Marc Pierson, and Athanasios V. Vasilakos. Cloud computing: Survey on energy efficiency. Vol. 47, No. 2, 2014.

[8] Zhi-Hui Zhan, Xiao-Fang Liu, Yue-Jiao Gong, Jun Zhang, Henry Shu-Hung Chung, and Yun Li. Cloud computing resource scheduling and a survey of its evolutionary approaches. Vol. 47, No. 4, 2015.

[9] Ilija Hadžić, Yoshihisa Abe, and Hans C. Woithe. Edge computing in the epc: A reality check. New York, NY, USA, 2017. Association for Computing Machinery.

[10] Ju Ren, Deyu Zhang, Shiwen He, Yaoxue Zhang, and Tao Li. A survey on end-edge-cloud orchestrated network computing paradigms: Transparent computing, mobile edge computing, fog computing, and cloudlet. Vol. 52, No. 6, 2019.

[11] Noura Aljeri and Azzedine Boukerche. Mobility management in 5g-enabled vehicular networks: Models, protocols, and classification. *ACM Comput. Surv.*, Vol. 53, No. 5, September 2020.

[12] Fei Miao, Sihong He, Lynn Pepin, Shuo Han, Abdeltawab Hendawi, Mohamed E Khalefa, John A. Stankovic, and George Pappas. Data-driven distributionally robust optimization for vehicle balancing of mobility-on-demand systems. *ACM Trans. Cyber-Phys. Syst.*, Vol. 5, No. 2, January 2021.