大阪大学

1

Adaptable and Evolvable Design of Network-oriented Services based on Core/Periphery Structure

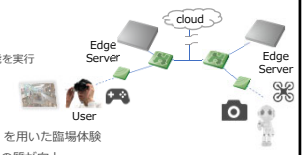
大阪大学 大学院情報科学研究科
情報ネットワーク学専攻 村田研究室

高木 詩織

研究の背景

2

- **ネットワークサービスのソフトウェア化**
  - Multi-access Edge Computing (MEC) への期待
    - ユーザに近い場所にエッジサーバを配置しサービス機能を実行
    - 通信距離の削減、負荷の分散により遅延を軽減
- **ユーザ体感型アプリケーションの登場**
  - Telexistence: 遠隔ロボットや MR (Mixed Reality) を用いた臨場体験
  - MEC によってリアルタイム性が求められるサービスの質が向上
- **MEC 環境におけるサービス機能配置**
  - リソースには限りがありすべてのサービス機能を配置するのは困難
  - 変化するユーザの要求やデバイスの発展などの実環境に合わせたサービスを提供するため、少ないコストで環境変動に対応できる設計が必要



サービス機能のネットワークにおける課題

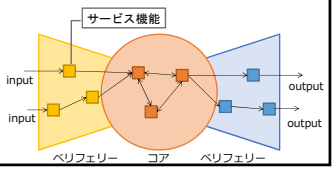
3

- **サービス機能のネットワーク上で多様なサービスを収容**
  - API 等によって利用可能なサービス機能同士が接続されたネットワーク
  - ネットワークを構成するサービス機能の一部を使用しサービスを構成
- **環境変動に対応してサービス機能の構成を変更できるサービス機能の構造**
  - 長期的な開発コストを削減できるサービス機能の構造が必要
    - サービス機能の構成の変更にはサービス機能やインタフェースの開発を伴う
    - 環境変動に応じてサービス全体を作り直す場合、開発コストが大きく増加
  - 将来どのようなサービス要求が発生するかを予測することは困難
- **進化適応性を持つサービス構造**
  - サービスを提供する能力を維持したまま、少ないコストでサービス機能全体の構成を変更し続ける能力
  - サービス機能の構造が、新たなサービスを収容可能な構造に「進化する」ことが必要

研究の目的とアプローチ

4

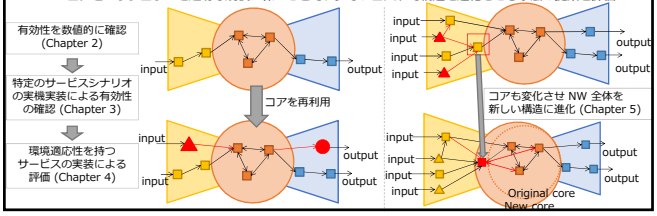
- **研究の目的**
  - 進化適応性を持つサービス構造の実現
    - 進化適応性: サービスの提供能力を維持したまま、少ないコストでシステムを変える能力
- **アプローチ**
  - コア/ペリフェリー構造に着目
    - 生物における柔軟で効率的な情報処理を行うシステム
    - コア: 密に構成され、効率的に情報処理
    - ペリフェリー: 多様な機能を提供できる構造
  - コア/ペリフェリー構造をサービス設計に活用
    - 再利用の可能性が高い機能をコアとすることで、ペリフェリーの接続構造のみを変化させ多様な入出力に対応
    - さらに、ペリフェリー機能の一部をコアに取り入れながら変化させることで新しいサービスを収容できる可能性が上昇



博士論文の構成

5

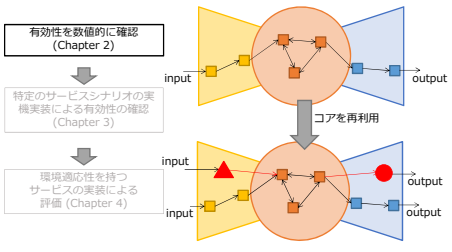
- **環境適応性を持つサービス機能の構造**
  - コアを再利用し、ペリフェリー機能のみを変化させることによって少ないコストで環境変動に適応
- **進化適応性を持つサービス設計（Chapter 5）**
  - コアとペリフェリーを適切な規模に保つことで、少ないコストで構造を進化させる手法の提案と評価



Chapter 2: Design and Placements of Virtualized Network Functions based on a Core/Periphery Structure

6

- Yuki Tsukui, Shin'ichi Arakawa, Shiori Takagi and Masayuki Murata, "Design and placements of virtualized network functions for dynamically changing service requests based on a core/periphery structure," *IEEE Access*, vol. 8, pp. 166 294 –166 303, Sep 2020.



Chapter 2 の研究目的とアプローチ

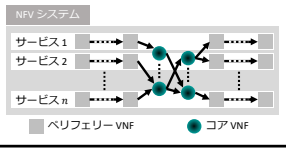
7

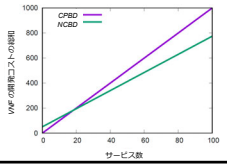
- **研究目的**
  - コア/ペリフェリー構造を用いたサービス構造の有効性を数値的に確認
- **開発コストを削減するアプローチ**
  - NFV (Network Functions Virtualization) を対象とし、コア/ペリフェリー構造を用いた設計 CPBD (Core/Periphery Based Design) を提案
  - CPBD が開発コストを削減可能であることを確認
- **配置コストを削減するアプローチ**
  - CPBD の設計方針に適切な VNF (Virtual Network Function) 配置により、配置コストを削減
  - 2 つの配置方針を比較し配置コストを削減可能な配置ポリシーを確認

CPBD: Core/Periphery Based Design

8

- **コア/ペリフェリー構造を用いた NFV システムの設計**
  - ペリフェリーが積極的に変化することで、システム全体の変化を吸収
    →ペリフェリー VNF: サービスの収容ごとに追加される、そのサービス固有の VNF
  - コアはあまり変化せず、ペリフェリー同士の接続を効率的に媒介
    →コア VNF: 複数のサービスの収容に繰り返し利用可能な VNF
- **コア VNF を繰り返し用いることで全体の開発コストを約 23% 削減**
  - コア VNF を用いない場合（NCBD）との比較結果

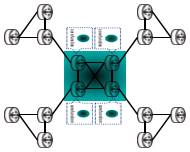




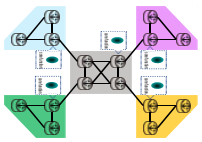
CPBD により設計された VNF の配置

9

- **リソース制約を考慮し 2 つの配置ポリシーを比較**
  - CLCP policy: Center-Located Core/Periphery placement
    - 物理ネットワークの地理的中央にコア VNF を配置
    - 地理的中央は多くの経路が通るため、より多くのサービスの収容に使用
    - 一部のノードや物理回線を集中的に利用し、リソースが枯渇する可能性
  - GDCP policy: Geographically-Distributed Core/Periphery placement
    - コア VNF を地理的に分散



CLCP のコア VNF 配置例



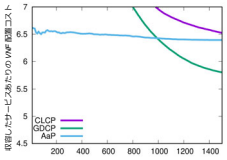
GDCP のコア VNF 配置例

評価

[17] S. Quanying, L. Ping, L. Wei, and Z. Zuqing, "Forecast-assisted NFV service chain deployment based on affiliation-aware vNF placement," in *Proceedings of 2016 IEEE Global Communications Conference (GLOBECOM)*, Dec. 2016, pp. 1–6.

10

- **評価設定**
  - CLCP, GDCP, 既存手法 AaP (Affiliation-Aware vNF Placement)[17] を比較
    - それぞれが収容したサービスあたりの配置コストを算出
- **長期的な視点では GDCP が最もコストを削減可能**
  - 事前に配置したコア VNF を繰り返し利用したことが効果を発揮
    - 多くのサービスを収容するほどサービス当たりの配置コストが小さくなる
  - 約 1000 種類以上のサービスを収容する場合は GDCP のコストが最も優れる
  - CLCP もコア VNF を利用するがサービスあたりに最多のコストを必要
    - リソース制約を受けやすく、収容可能なサービス数が減少
    - 事前に配置したコア VNF の利用回数も減少



7 × 7 の格子状ネットワークに
500 種類のコア VNF を配置した場合

Chapter 2 のまとめ

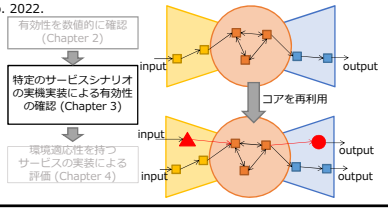
11

- **コア/ペリフェリー構造を用いた設計方針の有効性を数値的に確認**
  - コア VNF を用いて収容を行うことで、追加を含めた全体の VNF 開発コストを削減可能
    - コア VNF を繰り返し収容に使用
  - CPBD を前提とした VNF の配置方針
    - 地理的中央にコア VNF を配置する CLCP
    - コア VNF を地理的に分散させる GDCP
    - GDCPでは追加の VNF 配置コストを長期的に削減可能
- **課題: 実際のサービスを用いた評価**

Chapter 3: Design, Implementation and Evaluation of Core/Periphery-based Network-oriented Mixed Reality Services

12

- Shiori Takagi, Shin'ichi Arakawa and Masayuki Murata, "On the implementation and evaluation of a network- oriented mixed reality service based on core/periphery structure," *IEICE Technical Report (NS2019-218)*, pp. 221–226, Mar. 2020.
- Shiori Takagi, Shin'ichi Arakawa and Masayuki Murata, "Design, implementation and evaluation of core/periphery-based network-oriented mixed reality services," *Journal of Internet Services and Applications*, Feb. 2022.



Chapter 3 の研究の目的とアプローチ

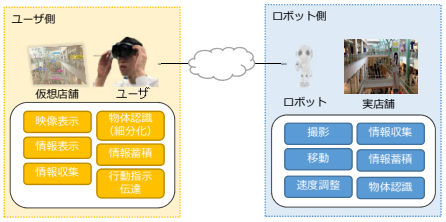
13

- **背景**
  - Chapter 2 では、コア/ペリフェリー構造を用いた設計の有効性を数値的に評価
  - 実際のサービスを用いた評価が必要
    - サービス機能を分割して配置すると、通信経路が長くなりサービスの応答性が損なわれる可能性
- **研究の目的**
  - コア/ペリフェリー構造を実際のサービスに利用し、有効性を評価
- **アプローチ**
  - 1 つのサービスシナリオを MEC 環境に実装することを考え、コア/ペリフェリー構造に基づいて設計
    - 普遍的な機能をコア、環境に応じて変化する機能をペリフェリーとして区別
    - Mixed Reality (MR) と遠隔ロボットを用いたサービスに着目
  - サービスシナリオを実機に実装し、開発コストとサービスの応答性を評価

想定するサービスと機能

14

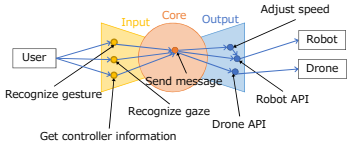
- **Mixed Reality (MR) と遠隔ロボットを用いた買い物体験サービス**
  - ロボットは実店舗の様子を撮影しユーザに提供
  - ユーザは映像を見ながら遠隔地のロボットを操作



コア/ペリフェリー構造に基づいた設計

15

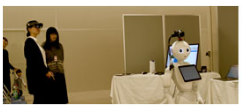
- **コア: 普遍的な機能**
  - 映像の入出力機能
  - ユーザからの行動指示メッセージの送信機能
  - ロボットの周辺の情報を集約する機能
- **ペリフェリー: 環境の変化に応じたサービスを提供する機能**
  - 使っているデバイスに応じたユーザからの行動指示を認識する機能
  - ロボットの API にアクセスする機能



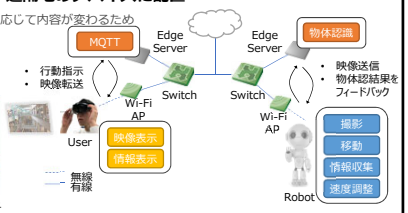
サービス機能の配置

16

- **コア機能: エッジサーバに配置**
  - 共通して使用される機能であるため
- **ペリフェリー機能: ユーザ、遠隔地のデバイスに配置**
  - デバイスの置かれている実環境に応じて内容が変わるため



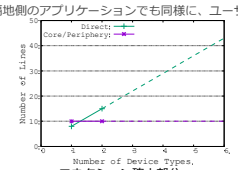
大阪大学共創フェスティバルでの展示の様子



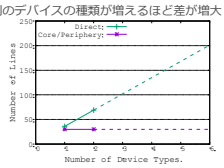
評価: 実装コスト

17

- **ユーザ側のアプリケーションにおいて、必要なソースコードの量を比較**
  - Direct: HoloLens のアプリケーションで直接遠隔地のデバイスを操作する場合
  - Core/Periphery: コア/ペリフェリー構造に基づいて処理を分割した場合
- **遠隔地のデバイスの種類が増えるほど効果が増大**
  - Direct では、遠隔地のデバイスの種類に応じてソースコードを追記する必要
  - 遠隔地側のアプリケーションでも同様に、ユーザ側のデバイスの種類が増えるほど差が増大



コネクション確立部分



メッセージ送信部分

評価: アプリケーションレベルの遅延

18

- **機能を分割し別の端末に配置すると、通信経路が増加しサービスの応答性にペナルティが発生**
- **メッセージを送信してからロボットが動き始めるまでの応答遅延を計測**
  - アプリケーションレベルの遅延は平均 52 [ms] であり、許容可能な大きさ
  - サービス機能を分割することによるペナルティは 31 [ms] 程度
  - コア/ペリフェリー構造を用いることで、少ないペナルティで実装コストを削減

ユーザのメッセージを送信してからロボットが動き始めるまでの時間

	Direct	Core on edge
平均 [ms]	21	52
最大 [ms]	24	263
最小 [ms]	0	18

Chapter 3 のまとめ

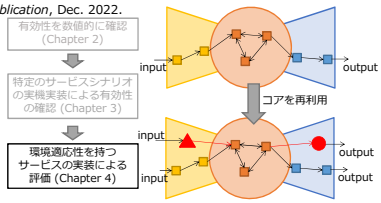
19

- **コア/ペリフェリー構造に基づく MR アプリケーションを設計・実装**
  - 普遍的な機能をコア、環境に応じて変化する機能をペリフェリーとして区別
  - コア機能をエッジサーバに配置し、ペリフェリー機能をユーザの端末とロボットに配置
- **コア/ペリフェリー構造を用いた設計の効果を評価**
  - 実装コスト: 遠隔地のデバイスの種類が増えるほど効果が増大
  - ペナルティは 31 [ms] 程度であり、許容可能な大きさ
- **課題**
  - 複数のサービスシナリオを用いた評価が必要
    - 様々なデバイスが同時に存在し、情報共有を行うサービス
  - より現実的な評価指標が必要
    - ソースコードの量のみでは開発に必要なコストを十分に評価できない

Chapter 4: Design, Implementation and Evaluation of a Network-oriented Service with Environmental Adaptability based on Core/Periphery Structure

20

- Shiori Takagi, Shin'ichi Arakawa and Masayuki Murata, "Implementation and Evaluation of a Network-oriented Service with Environmental Adaptability based on Core/Periphery Structure," *Technical Report of IEICE (2020-158)*, vol. 120, no. 413, pp. 208-213, March 2021 (in Japanese).
- Shiori Takagi, Shin'ichi Arakawa and Masayuki Murata, "Design, implementation and evaluation of a network-oriented service with environmental adaptability based on core/periphery structure," *submitted for publication*, Dec. 2022.



Chapter 4 の研究目的とアプローチ

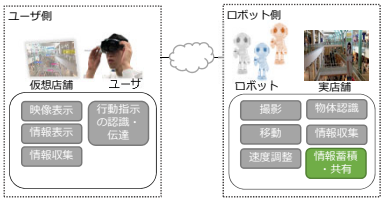
21

- **背景**
  - Chapter 3 では、ユーザ→ロボットやロボット→ユーザの情報処理を行うサービスを実装
    - 実際のサービスでは、遠隔地に複数の端末が存在し、情報を共有する可能性
  - Chapter 3 で用いた評価指標であるソースコードの量のみでは開発コストを評価するには不十分
    - サービスを開発し続けるためには、ソースコードの量だけでなくどれだけ単純なプログラムであるかが重要
- **研究目的**
  - コア/ペリフェリー構造の有効性をより実用的な指標で評価
- **アプローチ**
  - 遠隔地に複数のデバイスが存在し情報を共有するサービスに注目し、環境に合わせたサービスを提供するアプリケーションを実装
  - 新たな評価指標を導入し、コア/ペリフェリー構造の有効性の評価
    - プログラムの複雑度
    - 情報共有にかかるオーバーヘッド

想定するサービス

22

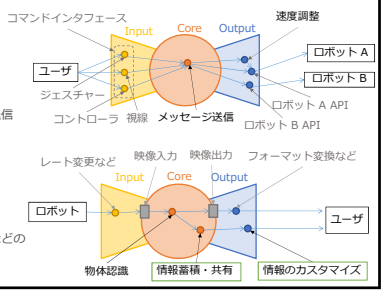
- **基本サービス**
  - ロボットは実店舗の様子を撮影ユーザに提供
  - ユーザは映像を見ながら遠隔地のロボットを操作
- **環境に合わせてサービスを向上させる機能追加**
  - 情報蓄積・共有
    - 遠隔地の情報を集約し、ユーザの注目度の変化に応じて必要な情報を提供



設計指針

23

- **共通の機能をコア機能とし、その他の機能をペリフェリー機能としてコア機能に接続**
  - ペリフェリー機能のみの変更で環境変動に適応
- **コア機能**
  - ユーザからの行動指示メッセージの送信
  - 映像中の物体を認識
  - 物体の情報を蓄積
- **ペリフェリー機能**
  - ロボットの速度調整
  - ユーザの注目度に応じてレコメンドなどの情報をカスタマイズし提供



評価方法

24

- **検討したサービスシナリオを実機で実装し評価**
- **設計シナリオ**
  - no core: コア機能が存在せず、機能が各端末に特化した設計
  - core/periphery: コア機能が存在する設計
    - core on robot: コア機能をロボット内に実装
    - core on edge: コア機能をエッジサーバに実装
- **評価指標**
  - Cyclomatic Complexity: 循環的複雑度
    - 環境変動への適応に必要なコストを表す指標
    - 新しい機能を追加する際にプログラムを読み解くことが必要であり、プログラムが複雑であるほど追加が困難
  - 遠隔ロボット間の情報共有に必要な情報送信の回数
    - 情報共有にかかるオーバーヘッドを決める指標
    - 回数が多いほど、必要なリソースや時間が増大

Cyclomatic Complexity の評価

25

- **ユーザ側・遠隔地側の端末の種類が増加した場合に必要な追加実装**
  - ユーザのデバイスから遠隔地のデバイスに接続する部分のコードを抜粋
  - コア機能がない場合 (no core): 各デバイスに応じた処理をすべてユーザ側のアプリケーションに追加
    - ユーザ側のアプリケーションから遠隔地のデバイスに直接アクセスするため
  - コア機能がある場合 (core/periphery): 各デバイスにペリフェリー機能を追加
- **コア/ペリフェリー構造を用いた設計では、デバイスの種類が増加してもプログラムは複雑にならない**

no core

```
分岐
if(!string.IsNullOrEmpty(pepperIP)){
    session = QiSession.Create(tcpPrefix + pepperIP + portSuffix);
    if (!_session.IsConnected){
        Debug.Log("Failed to establish connection");
        return;
    }
}
else if(!string.IsNullOrEmpty(droneIP)){
    session_drone = DroneSession.Create(tcpPrefix + droneIP + portSuffix);
    if (!_session.IsConnected){
        Debug.Log("Failed to establish connection");
        return;
    }
}
```

core/periphery

```
try{
    client.Connect(clientId);
}
catch (Exception e){
    Debug.Log(string.Format("Exception MQTT {0} ", e ));
    throw new Exception("Exception MQTT", e.InnerException);
}
mqttc = mqtt.Client()
mqttc.connect("192.168.10.73",1883, 60)
```

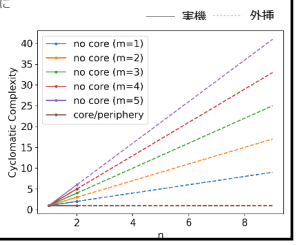
ユーザのデバイスから遠隔地のデバイスに接続する部分

Cyclomatic Complexity の変化

26

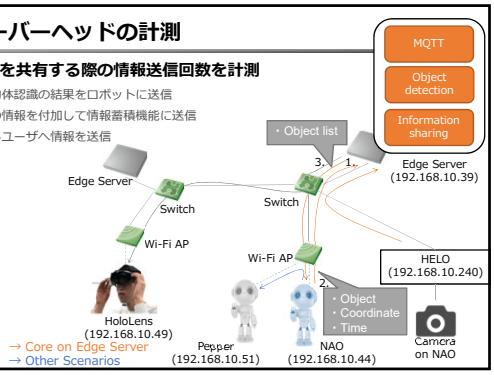
- **no core: *O(m*n)***
  - ユーザのデバイスから直接遠隔ロボットにアクセスするため、ロボットの種類が増えると全ての機能で必ず分岐処理が必要
  - ユーザ側のデバイスの種類が増えると、そのデバイスに各ロボットに対応する機能の実装が必要
- **core/periphery: 一定**
  - コア機能によってデバイスの差異を吸収
  - ペリフェリー機能の追加実装のみが必要
- **コア/ペリフェリー構造を用いた設計ではデバイスの種類の増加への対応が容易**
  - デバイスの種類が増えるほど差が顕著

変数	意味
m	ユーザ側のデバイスの種類数
n	遠隔地側のデバイスの種類数



情報共有のオーバーヘッドの計測

- **ロボットが情報を共有する際の情報送信回数を計測**
  1. エッジサーバが物体認識の結果をロボットに送信
  2. ロボットは物体の情報を付加して情報蓄積機能に送信
  3. 情報蓄積機能からユーザへ情報を送信

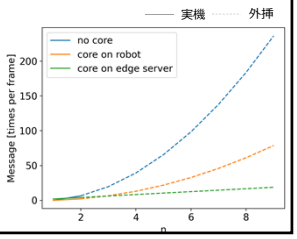


情報共有のオーバーヘッドの変化

28

- **no core: *O(k*n²)***
  - 各ロボットが自身以外のロボットに送信
  - 情報の蓄積方法が統一されておらず情報の種類毎に送信が必要
- **core on robot: *O(n²)***
  - 全てのロボット間での情報送信が必要
- **core on edge: *O(n)***
  - 各ロボットからエッジサーバにのみ情報を送信
  - エッジサーバからの送信頻度は低
- **コア/ペリフェリー構造を用いた設計では少ないオーバーヘッドで多くの情報を共有**
  - ロボットが 2 種類以下の場合オーバーヘッド大
  - ロボットの種類が増えるほど差が顕著

変数	意味
k	共有する情報の種類数 (ここでは 3)
n	遠隔地側のデバイスの種類数



Chapter 4 のまとめ

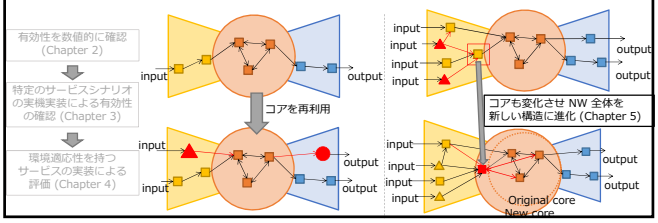
29

- **コア/ペリフェリー構造を活用したネットワークサービスを設計**
  - 環境変動に適応してサービスを向上させるサービスシナリオを検討
  - コア/ペリフェリー構造を用いてサービスを設計、実装
- **コア/ペリフェリー構造を活用した設計の有効性を評価**
  - プログラムの複雑度の指標を導入し、コア/ペリフェリー構造によってプログラムが単純になり開発コストを削減可能であることを示した
  - 端末同士で情報共有を行うサービスに着目し、コア/ペリフェリー構造を用いた設計とコアのエッジサーバへの配置によって効率的に情報共有を行うことができることを示した
- **課題: サービス機能ネットワークの進化適応性の獲得**
  - サービス機能ネットワーク上で様々なサービスを収容する可能性を考慮
  - 変化するサービスの要求に備え、コアとペリフェリーを適切な規模に保ち構造を進化させる手法

Chapter 5: Evolvable Design of Network-oriented Services based on Core/Periphery Structure

30

- Shiori Takagi, Shin'ichi Arakawa and Masayuki Murata, "Evolvable Design of Network-oriented Services based on Core/Periphery Structure," *submitted for publication*, Jan. 2023.



Chapter 5 の研究目的とアプローチ

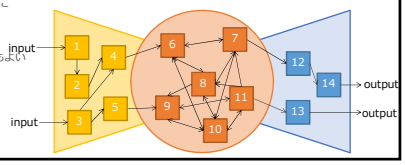
31

- **背景**
  - コア/ペリフェリー構造に基づいた設計により、ペリフェリー機能のみの変更で環境変動に適応可能
  - サービス機能ネットワーク全体を進化させ、新しいサービスを収容する必要
    - ペリフェリー機能の著しい増加によりサービスの効率が低下
    - コアではなかった機能の利用の増加
- **研究目的**
  - 進化適応性（サービスの提供能力を維持したまま、少ないコストでシステムを再構成する能力）を獲得
- **アプローチ**
  - ペリフェリー機能の一部をコアとして利用できるようにサービス機能の構成を変更
  - コアとペリフェリーを適切な大きさと密度に調整
    - サービス機能を密に接続する（コアを大きくする）→サービスを効率よく収容することができるが、高コスト
    - サービス機能を疎に接続する（コアを小さくする）→コストは低いが、サービスの効率が悪化
    - 適切な大きさと密度に調整することで、長期にわたりサービス機能の構成を変え続けることが可能

サービス機能ネットワークとサービスチェイン

32

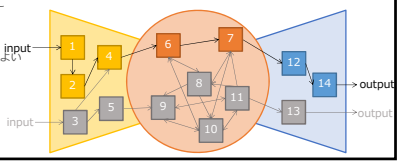
- **サービス機能ネットワーク**
  - 入力側のペリフェリー、コア、出力側のペリフェリーで構成
    - コア: 互いに利用可能なサービス機能が多く高密度
    - ペリフェリー: 低密度
- **サービスチェイン**
  - 使用したいサービス機能と順序を示した組み合わせ
  - 与えられたサービス機能をその順に使用する経路が存在するとき、**収容可能である**と定義
    - 不必要なサービス機能が含まれてもよい
  - サービスチェインを収容した際の経路の長さを**チェイン長**と定義



サービス機能ネットワークとサービスチェイン

33

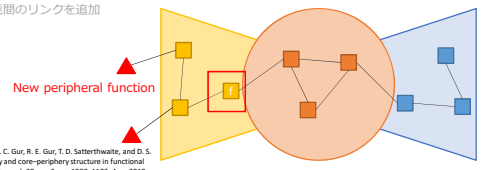
- **サービス機能ネットワーク**
  - 入力側のペリフェリー、コア、出力側のペリフェリーで構成
    - コア: 互いに利用可能なサービス機能が多く高密度
    - ペリフェリー: 低密度
- **サービスチェイン**
  - 使用したいサービス機能と順序を示した組み合わせ
  - 与えられたサービス機能をその順に使用する経路が存在するとき、**収容可能である**と定義
    - 不必要なサービス機能が含まれてもよい
  - サービスチェインを収容した際の経路の長さを**チェイン長**と定義



サービス機能の密度の制御

34

- **コアの密度 > コア/ペリフェリー間の密度 > ペリフェリーの密度 になるように密度を制御**
  - コアとペリフェリーの密度が分離しているほど、各ブロックの機能の発展を促進[56]
  1. コアに隣接するペリフェリー機能から注目するサービス機能 f をランダムに選択
  2. f をコア機能とみなせるよう f とコアの間にリンクを追加
  3. 密度の大小関係を保つように、f を含む新しいコアとペリフェリーの間のリンク
  4. ペリフェリー機能間のリンクを追加



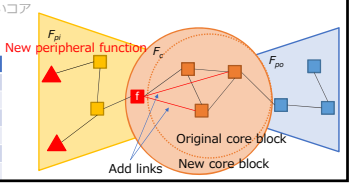
[56] S. Gu, C. H. Xia, R. Ciric, T. M. Moore, R. C. Gur, R. E. Gur, T. D. Satterthwaite, and D. S. Bassett, "Unifying the notions of modularity and core–periphery structure in functional brain networks during youth," Cerebral Cortex, vol. 30, no. 3, pp. 1087–1102, Aug. 2019.

サービス機能ブロックの密度の制御

35

- **コアの密度 > コア/ペリフェリー間の密度 > ペリフェリーの密度 になるように密度を制御**
  1. コアに隣接するペリフェリー機能から注目するサービス機能 f をランダムに選択
  2. f をコア機能とみなせるよう f とコアの間にリンクを追加
     - コアの大きさ *coresize* は $coresize_{min} < coresize < coresize_{max}$
     - コアの密度 d_c は $dmin_c < d_c$ かつ $d_c - d_{c,cp} > th_{c,cp}$
  3. 密度の大小関係を保つように f を含む新しいコアとペリフェリーの間のリンクを追加
  4. ペリフェリー機能間のリンクを追加

変数名	説明
dc, dp	コア、ペリフェリーの密度
$d_{c,cp}$	コア/ペリフェリー間の密度
$coresize$	コアの大きさ
$coresize_{min}$, $coresize_{max}$	コアの大きさの最小・最大
$dmin_c$, $dmin_p$	密度の最小値
$th_{c,cp}$, $th_{cp,p}$	密度の差の閾値

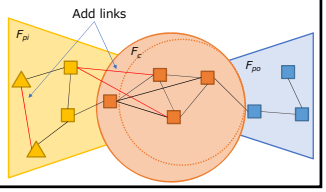


サービス機能ブロックの密度の制御

36

- **コアの密度 > コア/ペリフェリー間の密度 > ペリフェリーの密度 になるように密度を制御**
  1. コアに隣接するペリフェリー機能から注目するサービス機能 f をランダムに選択
  2. f をコア機能とみなせるよう f とコアの間にリンクを追加
  3. 密度の大小関係を保つように、f を含む新しいコアとペリフェリーの間のリンクを追加
     - コア/ペリフェリー間の密度とペリフェリーの密度の差は $d_{c,cp} - d_p > th_{cp,p}$
  4. ペリフェリー機能間のリンクを追加
     - ペリフェリーの密度 d_p は $d_p > dmin_p$

変数名	説明
dc, dp	コア、ペリフェリーの密度
$d_{c,cp}$	コア/ペリフェリー間の密度
$coresize$	コアの大きさ
$coresize_{min}$, $coresize_{max}$	コアの大きさの最小・最大
$dmin_c$, $dmin_p$	密度の最小値
$th_{c,cp}$, $th_{cp,p}$	密度の差の閾値



比較手法と評価方法 37

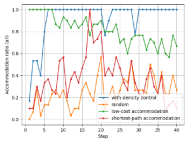
- **比較手法**
  - Density Control: 提案手法
  - Random: ランダムにノードの組を選択し、リンクを追加
  - Low-Cost Accommodation: サービスチェイン収容率が最も上昇するノードの組を選択しリンクを追加
    - 低コストで多くのサービスチェインを収容できるが、チェイン長が長い
  - Shortest-Path Accommodation: チェイン長が最も短くなるようにリンクを追加
- **評価方法**
  1. 環境変動を想定し、サービス機能ネットワークとサービスチェインが変化
     - ペリフェリー機能の増加
     - ランダムなサービスチェインの発生
  2. 各手法を適用しサービス機能ネットワークを更新
  3. 1. と 2. を 40 ステップが経過するまで反復
  4. 40 ステップの実行を 100 回試行
     - サービス機能ネットワークが 100 通りの進化を行った場合について評価

評価指標 38

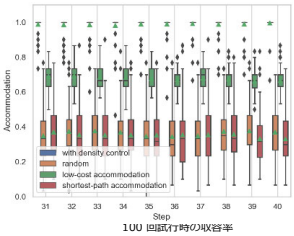
- **サービスチェインの収容率**
  - 各ステップ 30 個生成したサービスチェインのうち、収容可能であったものの割合
  - 新しいサービスチェインをどれだけ収容できるようになったかを評価
  - Random と Shortest-Path Accommodation では、Density Control とほぼ同じ量のコストを使用
- **開発コスト**
  - 各手法での目標を達成するために追加したリンクの数
  - Density Control, Random, Low-Cost Accommodation では追加したリンクの総数
  - Shortest-Path Accommodation で使用するコストは無制限とし、各ステップで生成したサービスチェインを全て最短のチェイン長で収容する場合のコストを計算
- **収容可能であったサービスチェインについて、チェイン長**
  - 最短のチェイン長（サービスチェインのノード数 -1）との比
  - 短いほど余計な通信経路が少ないため効率よくサービスを提供可能

評価結果: サービスチェイン収容率 39

- **Density Control で安定して高い収容率を達成（青）**
  - コア（密に接続されたブロック）を一定の大きさに保ったことによる効果
  - Shortest-Path Accommodation（赤）では、Density Control と同等のコストを使用した場合、収容率のばらつきが大
    - 長いサービスチェインが多いと収容しきれないため
  - Low-Cost Accommodation（緑）では収容率が低
    - コア機能を増やす進化をしておらず、双方向に利用できるサービス機能が少ないため



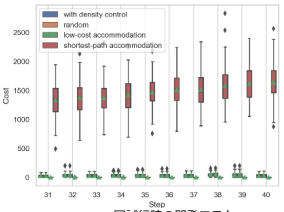
収容率の変化の一例



100 回試行時の収容率

評価結果: 開発コスト 40

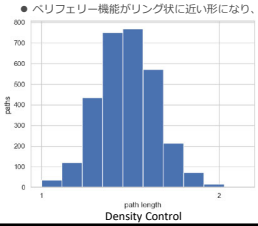
- **Density Control は Shortest-Path Accommodation よりも少ないコストで同等の収容率を達成**
  - Shortest-Path Accommodation では、過去に追加したリンクが再利用されることが少ない
  - Low-Cost Accommodation と比較すると 10 倍程度多くのコストが必要
- **Density Control の開発コストはサービスチェインの数や長さに左右されない**
  - サービスチェインが 30 個/ステップの場合の結果
  - Low-Cost, Shortest-Path Accommodation は発生したサービスチェインの数や長さに依存
    - 新しいサービスチェインが多く発生するほどコストが増大



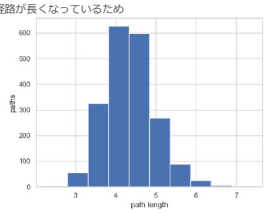
100 回試行時の開発コスト

評価結果: チェイン長 41

- **Density Control では短いチェイン長で効率よくサービスを提供可能**
  - Density Control では最短チェイン長の 2 倍以下でチェインを構成
    - ペリフェリー機能とコア機能を接続するリンクが用意されるため
  - Low-Cost Accommodation では最短チェイン長の 3 ~ 7 倍程度のチェイン長
    - ペリフェリー機能がリング状に近い形になり、通信経路が長くなっているため



Density Control



Low-Cost Accommodation

Chapter 5 のまとめ 42

- **進化適応性を持つサービス機能ネットワークの構造を実現**
  - コアとペリフェリーの密度を制御することで、少ないコストで多様なサービスチェインを収容可能
  - サービス構造がどのような進化を行っても、安定して高い収容率を達成
  - 短いサービスチェインを構成することができるため、効率的なサービスを提供可能
  - 開発コストはサービスチェインの数や長さに左右されず安定
- **課題**
  - コアとペリフェリーの配置問題
    - ペリフェリーからコアに変化したサービス機能をどのように配置するか
  - サービス機能が削除された場合への対応

博士論文のまとめ

- **コア/ペリフェリー構造を用いた進化適応性を持つサービス構造を実現**
  - コア機能を再利用することで開発コストを削減し、ペリフェリーのみの変更で様々な入出力に対応
    - コア機能をエッジサーバに配置することで、効率よく端末間の情報共有が可能
  - ペリフェリー機能の一部をコアに変化させることで、ペリフェリー機能の著しい増加や新しいサービスの要求に対応
    - 少ないコストで高い収容率を達成
    - 短いサービスチェインを構成し効率的なサービスを提供
  - サービス機能ネットワークが様々な進化を行った場合でも安定した収容率
- **今後の課題**
  - コアとペリフェリーの配置問題
  - サービス機能が削除された場合への対応

付録

Chapter 3: ソースコード（抜粋）

- **2 種類のロボットへの接続に対応する場合の コネクション確立部分**
  - 直接ロボットに接続する（コア/ペリフェリー 構造でない）場合、ロボットごとの API に合わせた処理が必要
  - コア/ペリフェリー 構造に基づいた場合、ロボットとの通信は MQTT を介して行うため、ロボットの種類が増えても追記が不必要

Direct

```
//Direct
if(!string.IsNullOrEmpty(pepperIP)){
    _session = QiSession.Create(tcpPrefix + pepperIP + portSuffix);
    if (!_session.IsConnected){
        Debug.Log("Failed to establish connection");
        return;
    }
//Core/Periphery
}else if(!string.IsNullOrEmpty(droneIP)){
    _session_drone = DroneSession.Create(tcpPrefix + droneIP + portSuffix);
    if (!_session.IsConnected){
        Debug.Log("Failed to establish connection");
        return;
    }
}
```

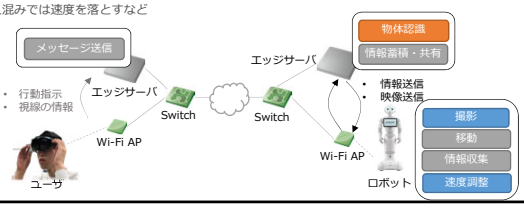
Core/Periphery

```
try{
    client.Connect(clientId);
}
catch (Exception e){
    Debug.Log(string.Format("Exception MQTT {0} ", e ));
    throw new Exception("Exception MQTT", e.InnerException);
}
```

API に合わせた追記が必要

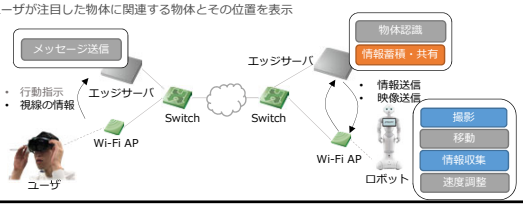
Chapter 4: 速度調整部分の実装

- **ロボットが撮影した映像に対して物体認識を行い、その結果をロボットに送信**
  - コア機能としたことでデバイス外に配置可能
  - 高負荷な処理のためエッジサーバで実行
- **ロボットは物体の情報をもとに速度を調整**
  - 人混みでは速度を落とすなど



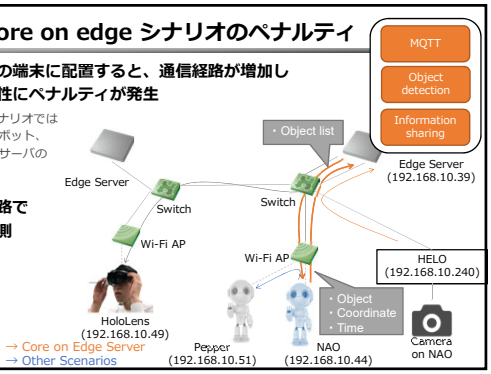
Chapter 4: 情報蓄積・共有部分の実装

- **物体認識の結果とロボットが持つ位置情報をもとに物体の情報を蓄積**
  - 周辺のロボットで共有するのためエッジサーバで実行
  - 各ロボットがエッジサーバに情報を送信
- **ユーザの注目度に応じて情報をカスタマイズ・提示**
  - ユーザが注目した物体に関連する物体とその位置を表示



Chapter 4: Core on edge シナリオのペナルティ

- **機能を分割し別の端末に配置すると、通信経路が増加し サービスの応答性にペナルティが発生**
  - Core on edge シナリオでは エッジサーバ→ロボット、 ロボット→エッジサーバの 通信が必要
- **増加した通信経路で かかる遅延を計測**



Chapter 4: ペナルティの計測

- **Core on edge シナリオで実行する処理を 5 つのフェーズに分け、新たな通信経路であるフェーズ 2, 4 でかかる遅延を計測**
  - フェーズ 2: ロボットが物体認識の結果を受信
  - フェーズ 4: ロボットが情報を付加しエッジサーバに送信
- **通信経路が増加したことによるペナルティは最小 4 [ms]、平均 99 [ms]**
  - ロボットは Wi-Fi 環境で通信しており、障害物やアクセスポイントからの距離などで遅延のばらつきが大
  - 通信の遅延については、5G によって提供される超低遅延の通信によって改善されると期待