大阪大学 1

# Implementation and Evaluation of a Network-oriented Mixed Reality Service 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

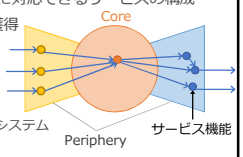
高木 詩織

## 研究の背景 2

- **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
  - 高負荷な処理やリアルタイム性が求められるサービス
- **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
- **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
  - ユーザの要求や実環境の変動に対して少ないコストで対応できる配置

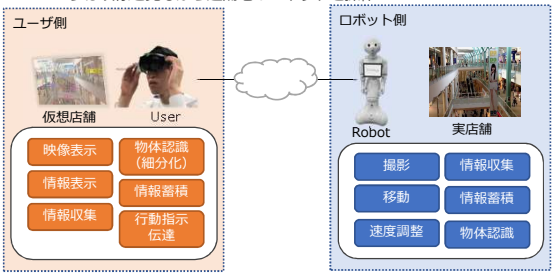
## 研究の目的とアプローチ 3

- **研究の目的**
  - 実装コストを抑えつつ実環境の変化に柔軟に対応できるサービスの構成
  - MEC 環境でのサービス機能配置の指針を獲得
- **アプローチ**
  - Core/Periphery 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
    - Core: 密に構成され、効率的に情報処理
    - Periphery: 多様な構成を持つことが可能
  - サービス機能を適切に分割し MEC 環境に配置
- **研究ステップ**
  1. Core/Periphery 構造に基づいた MR サービスを設計
  2. 設計したサービスを実機を用いて実装
  3. Core/Periphery 構造を用いた設計の効果を評価



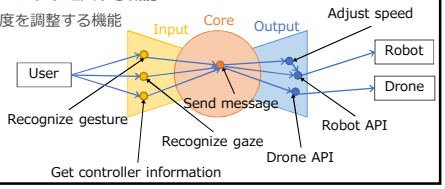

## 想定するサービスと機能 4

- **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



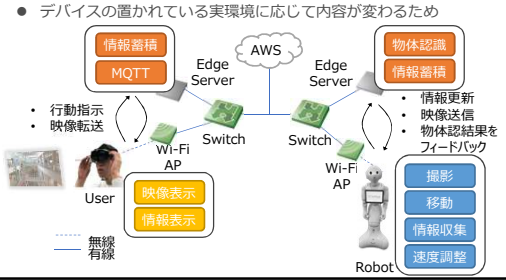

## Core/Periphery の構成 5

- **Core: 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
- **Periphery: 環境の変化に応じて内容が変わる機能**
  - 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
  - ロボットの速度を調整する機能




## サービス機能の配置 6

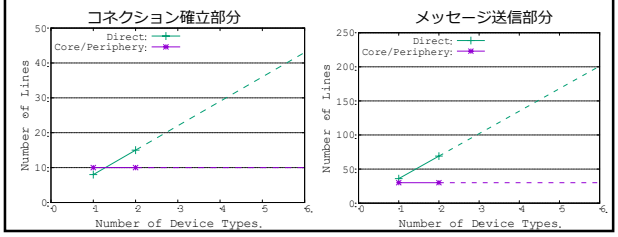
- **Core 機能: エッジサーバに配置**
  - 複数の端末での情報共有、画像認識などの高負荷な処理を行うため
- **Periphery 機能: エンド端末に配置**
  - デバイスの置かれている実環境に応じて内容が変わるため



## 評価: 実装コスト

7

- **Core/Periphery 構造を取ることでソースコードの量がどれだけ減少するかを比較**
  - 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 vs Core/Periphery 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**

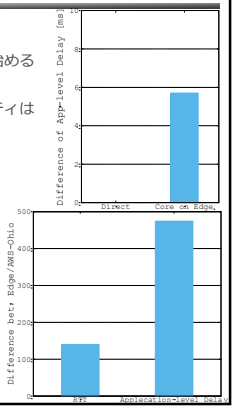



## 評価: アプリケーションレベルの遅延

8

1. **機能の分割によるペナルティ**
   - メッセージを送信してから Pepper が動き始めるまでの応答遅延を計測
   - サービス機能を分割することによるペナルティは 5.7 [ms] 程度
   - サービスの応答性にほぼ差はない
2. **エッジ vs クラウド**
   - AWS に MQTT 機能を配置、エッジ配置時との応答遅延を比較
   - ping の応答時間の差の 3.5 倍の応答遅延差
   - コア機能をエッジに配置する意義は大きい

Core/Periphery 構造に基づいてサービス機能を分割し、コア機能をエッジサーバに置くことでサービスの応答性を保ったまま実装コストを削減







## まとめと今後の課題

9

- **まとめ**
  - Core/Periphery 構造に基づく MR アプリケーションを設計・実装
  - Core/Periphery 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: 5.7 [ms] 程度に抑制可能
- **今後の課題**
  - 他シナリオでの実装コスト・ペナルティの評価
    - ロボット間での現在地などの情報共有
    - ロボットの周辺の状況に合わせた速度調整
    - 物体認識を実行するマシンからのフィードバック

## 付録: ソースコード（抜粋）

10

- **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（Core/Periphery 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
  - Core/Periphery 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

```
//Pepper
if( string.IsNullOrEmpty(pepperIP)){
    _session = QiSession.Create(tcpPrefix + pepperIP + portSuffix);
    if (!_session.IsConnected){
        Debug.Log("Failed to establish connection");
        return;
    }
//Drone (fictious)
}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