WebAssemblyのライブマイグレーションにより透過的にクラウドとエッジを連携するサービス実行環境の検討

田村 来希[1,a)]

1. はじめに

クラウドとエッジを連携させたサービス実行基盤は，レイテンシ削減や計算資源の効率的活用の観点から重要性が高まっている．クラウドからエッジへのタスク移行は応答時間の短縮やクラウド側の負荷削減に寄与し，エッジからクラウドへのタスク移行は計算オフロードを実現する．これらを適切に組み合わせることで，高効率なサービスの実行が可能となる．

しかし，異種ノード間でタスクを移行するには，従来の単一ノード向けプログラミングパラダイムでは不十分である．たとえば AWS IoT Greengrass v1 はイベント駆動型の記述モデルを採用しており，v2 ではコンポーネント指向を採用することで，開発者に分散システムの構成を意識した設計を要求する．CEF [2] は関数が実行されるリージョンを関数アノテーションとして記述するアスペクト指向プログラミング (AOP) を導入した分散サービス実行環境であり，単一プロセスアプリケーションと同様のコード記述で分散実行を可能にする点で有用である．

一方で, CEF には以下の制約がある．(1) アノテーション付き関数呼び出しのみが RPC に変換されるため，Python の意味論が分散環境で完全には保証されない．具体的には，移行元ノードと移行先ノードでグローバル変数が同期されず，グローバル変数を利用するプログラムが正しく動作する保証はない．(2) 関数境界でのみノード切り替えが行われるため，関数途中でのより細粒度なオフロードがサポートされない．

本研究では，これらの課題を解決するために，CEF の AOP を拡張した WebAssembly(Wasm) 実行環境 **Kafu** を提案する．Kafu は Python の代わりに Wasm を用い，RPC の代わりに Wasm ランタイムのライブマイグレーションを行うことで，開発者がアノテーションにより指定した方法でサービスのノード切り替えを実現する．本システムでは Wasm バイナリ変換ベースのライブマイグレーション手法を用いており，使用する Wasm ランタイムやノードの OS/CPU アーキテクチャに依存せず，異種環境間での透明な移行を実現する．したがって，クラウドでは高性能な JIT コンパイラを，エッジでは軽量なインタプリタを利用するといった柔軟な構成を可能とし，システム全体の性能の最大化や余剰計算資源の有効的な活用を実現する．

[1] 京都大学
Kyoto University, Kyoto, Japan
[a)] tamura.raiki.48a@st.kyoto-u.ac.jp

2. 設計

各ノードは軽量な RPC サーバを実行し，マイグレーション要求を処理する．移行元のサーバは，Wasm プログラムのスナップショットを移行先ノードへ送信し，移行先サーバは，スナップショットから Wasm プログラムの実行を再開する．スナップショットには，Wasm プログラムのスタック，グローバル変数，メモリ，テーブルといった実行状態が含まれ，ノード間のプログラムの完全な移行を保証する．

2.1 Wasm のライブマイグレーション

Wasm のライブマイグレーションには複数のアプローチが存在する.Nurul‐Hoque [3], Fujii [1] らの手法はインタプリタループでマイグレーション要求をハンドリングする方式を採用している．この方式は実装が容易である一方，ランタイムがインタプリタに限定される点が制約となる．

田村ら [4] の手法は AOT コンパイル時に機械語を計装することで，JIT/AOT を維持したままライブマイグレーションを実現する．しかし，ランタイム内部を改変する必要があり，実装負荷やランタイム更新との整合性維持が課題となる．

図 1 Kafu によるリアルタイム物体認識プログラム

```
#include "kafu.h"

KAFU_DEST("edge1")
image_t get_image_from_camera() {
    image_t image = /* Get image from camera */;
    return image;
}

KAFU_DEST("edge1")
KAFU_MAY_OFFLOAD("cloud1")
int count_people() {
    image_t image = get_image_from_camera();
    // Count people in the image with machine
     learning model.
    return inference(image);
}

KAFU_DEST("cloud1")
int main() {
    // Count people every 1 second
    for(;;) {
        printf("count: %d\n", count_people());
        sleep(1);
    }
}
```

本研究では，Wasm バイナリ変換に基づく新たなライブマイグレーション手法を採用する.Wasm バイナリ自体にチェックポイント・リストア処理を挿入することで，インタプリタ・JIT・AOT のいずれの実行方式にも対応し，かつランタイムの修正を不要とする．この特性により，既存ランタイムとの互換性を保ちながら実装・保守コストを抑えた移行機構を実現する．

2.2 AOP

AOP はアノテーションなどにより，開発者が RPC やライブマイグレーションなどのサービスに直接関係のないロジックを意識することなく，サービスの開発に集中することを可能にする．Kafu では，C/C++ プログラムの AOP を実現するために，以下の C マクロを提供する．どちらも関数定義の直前において利用することができる．

- `KAFU_DEST`：関数の実行ノードを指定する．
- `KAFU_MAY_OFFLOAD`：関数実行中にリソース状況に応じて指定ノードへオフロード可能であることを示す．

これにより，単一アプリケーションとして記述したコードをクラウド／エッジ分散環境へ透過的にデプロイすることができる．図 1 は，クラウドとエッジの両方で実行されるリアルタイム物体認識のプログラムの例である．

3. 実装

アプリケーションは通常の C/C++ プログラムとして記述され，Clang を用いて Wasm モジュールへコンパイルされる. 生成された Wasm バイナリに対し，本研究で開発したバイナリ変換ツール *Snapify* を適用する．Snapify は Binaryen/Asyncify をベースにしたフォークであり，Kafu 用のマイグレーションポイント挿入パスを実装している．

Snapify は AOP マクロによるメタデータを解析し，以下の規則に従ってマイグレーションポイントを挿入する．

- `KAFU_DEST`：関数先頭と `ret` 命令の直前にマイグレーションポイントを挿入する．
- `KAFU_MAY_OFFLOAD`：ループのバックエッジにマイグレーションポイントを挿入する．

4. 課題

Wasm プログラムのライブマイグレーションはの実行中の関数以外の情報を含むためスナップショットのサイズが大きく，RPC よりもコストが高い．今後はライブマイグレーションの所要時間と RPC の所要時間を比較し，サービスの性能を評価したい．また，CEF ではデータのプライバシーを考慮していたが，提案手法ではこれを達成することができないため改善が必要である．

参考文献

[1] D. Fujii, K. Matsubara, and Y. Nakata. Stateful vm migration among heterogeneous webassembly runtimes for efficient edge-cloud collaborations. In *Proceedings of the 7th International Workshop on Edge Systems, Analytics and Networking*, EdgeSys '24, page 19–24, New York, NY, USA, 2024. Association for Computing Machinery.

[2] Y. Nakata, M. Takai, H. Konoura, and M. Kinoshita. Cross-site edge framework for location-awareness distributed edge-computing applications. In *2020 8th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, pages 63–68, 2020.

[3] M. Nurul-Hoque and K. A. Harras. Nomad: Cross-platform computational offloading and migration in femtoclouds using webassembly. In *2021 IEEE International Conference on Cloud Engineering (IC2E)*, pages 168–178, 2021.

[4] R. Tamura, D. Kotani, K. Shudo, and Y. Okabe. Bringing together cross-isa checkpoint/restoration and aot compilation of webassembly programs. In *Proceedings of the 22nd ACM SIGPLAN International Conference on Managed Programming Languages and Runtimes*, MPLR '25, page 114–124, New York, NY, USA, 2025. Association for Computing Machinery.