ライブマイグレーション可能な WebAssembly ランタイムと WASI をインタフェースとするライブラリ OS の開発
― クラウドとエッジを透過的に連携する効率的な WebAssembly 実行環境 ―

1. 背景

近年、ロボットやドローン等のデータを即時に分析したいというニーズや、クラウド上での高負荷な計算・データ集約のニーズが高まり、クラウドとエッジの連携が重要視されている。しかし、従来の連携サービス開発では、開発者がビジネスロジック以外に、複雑なネットワークプログラミング、RPC、サービスの状態の保存などの通信処理や API 設計を大量に記述する必要があり、開発コストが非常に高いという課題があった。これらの課題を解決するソフトウェアとして、Dapr や Erlang などがあるが、いずれも特定のプログラミング言語やプログラミングパラダイムに強く依存している。

2. 目的

本プロジェクトの目的は、上記の課題を解決し、サービス開発者が単一プロセスのアプリケーションを書くのと同じように、クラウドやエッジなどのさまざまな環境で分散配置されるサービスを記述できるフレームワークを構築することである。サービス配置やオーケストレーション等の詳細を隠蔽し、開発者がサービスの中心な処理（ビジネスロジック）の記述のみに集中できる環境の実現を目指す。

具体的には、開発者が C や C++などで記述したサービスのソースコードに「この処理はどのノードで実行するか」という簡単な目印を追記するだけで、プログラムの実行場所を動的に指定できる仕組みを取り入れる。これにより、複雑な通信処理を記述する必要がなくなる。さらに、軽量で様々な環境でも実行できるプログラム形式である WebAssembly を採用し、実行中のプログラムを止めることなく別のコンピュータへ移動させるライブマイグレーション技術を活用する。従来の仮想マシンのライブマイグレーションでは数秒から数分かかり、同じ種類の機器間でしか移動できなかった。特に、エッジデバイスでは ARM CPU が、クラウドサーバでは Intel CPU が標準的に採用されており、仮想マシンのライブマイグレーションではクラウド・エッジ間のプログラムの引き継ぎが不可能だったが、WebAssembly を用いることでこのような異種環境間のプログラムの移行が可能になる。

これらのアプローチにより、開発者はネットワーク通信やデータ転送といった複雑な処理を一切書くことなく、まるで 1 台のコンピュータ向けのプログラムを作るような手軽さで、高度で効率的な分散システムを構築できるようになる。

3. 開発の内容

本プロジェクトでは、分散配置されるサービスの開発とデプロイをサポートするフレームワーク（SDK）である「Kafu」を開発した。Kafu ではサービスは C/C++言語で記述され、WebAssembly として実行される。この WebAssembly プログラムは、関数呼び出しの境界で別の物理ノードへの実行の切り替え（ライブマイグレーション）を行う専用の実行環境で実行される。Kafu SDK は、単なるツールの集合体ではなく、エッジ・クラウド連携の開発や運用における技術的課題を解決する統合的なフレームワークである。SDK は単一プロセスア

プリケーションのように分散システムを記述することを可能にし、サービスのビルドやデプロイ手順を極限まで簡単化することで開発者体験を向上させる目的で以下のツール・ライブラリを含む。

- C/C++ヘッダーファイル（処理が実行されるノードを宣言するための機能を提供する）
- C/C++から WebAssembly へのコンパイラ（複雑なサービスのコンパイル手順を簡単化）
- WebAssembly 実行環境を備えたサーバ（本番用の実行環境として動作する）
- 単一ノード用の WebAssembly 実行環境（開発用の実行環境として動作する）
- Kubernetes マニフェスト生成ツール（クラスタのデプロイを簡単化する）

C/C++ヘッダーファイルは、特定の関数がどのノードで実行されるかを宣言的に記述するためのマクロを提供する。このマクロを用いて、ユーザは単一プロセスで実行されるプログラムのように分散サービスを記述することができる（図 1）。

```c
#include <stdio.h>
#include "kafu.h"

void f();

int main() {
  // Starts the program on the cloud node.
  printf("Hello, from cloud!\n");
  fflush(stdout);
}

KAFU_DEST(f, "edge")
KAFU_EXPORT(f)
void f() {
  printf("Hello, from edge!\n");
  fflush(stdout);
}
```

図 1：サービスのソースコードの例

C/C++コンパイラはサービスのソースコードを専用の WebAssembly モジュールにコンパイルするツールである。ユーザが指定した関数における実行ノードの切り替えを実現するために、プログラムを中断・再開可能にする WebAssembly モジュールの命令書き換え（バイナリ変換）を開発した。これは WebAssembly モジュールの各関数の入口と出口に中断・再開可能な地点を WebAssembly 命令列として挿入することで、実行途中の関数の状態や変数の状態を保存・復元できるようにするものである。

実行基盤として機能するサーバは、WebAssembly ランタイムを実行し、ノード間のライブマイグレーションやクラスタの制御を行う（図 2）。WebAssembly ランタイムは、WebAssembly 用の機械学習推論のインタフェースである WASI NN をサポートしており、標準的な機械学習モデルを実行することができる。ライブマイグレーション時には、メモリの圧縮転送や、差分転送を行うことで、通信の高速化・効率化を図っている。

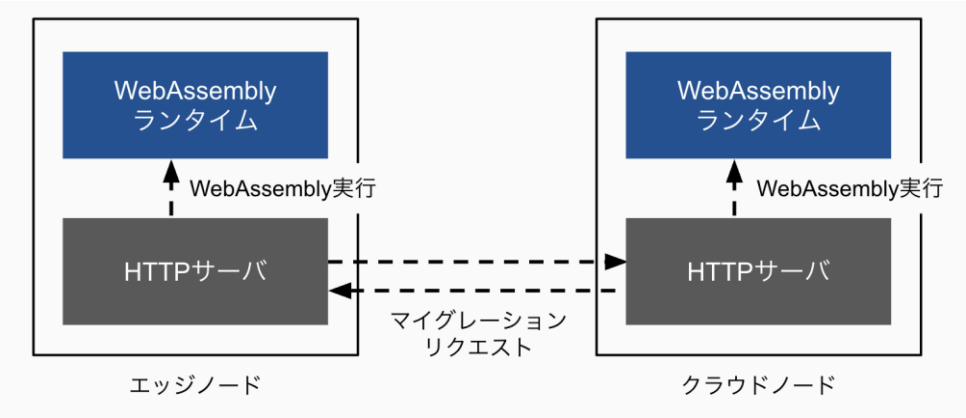

図 2:サーバによるサービス実行

また、開発効率の向上やサービスのデプロイのための周辺ツールも開発した。まずは、SDK でコンパイルした WebAssembly プログラムを単一ノード上で実行するためのツールである。このツールを使うことで、実際にクラスタを構築することなく、開発環境で分散サービスの実行をエミュレートし、デバッグすることができる。また、デプロイ面では、クラスタ設定ファイルから Kubernetes マニフェストを生成するツールを開発した。これにより、ユーザは複雑な設定をすることなく、開発したサービスを Kubernetes クラスタにデプロイすることができる。

4. 従来の技術(または機能)との相違

(i) バイナリ変換による WebAssembly ランタイムのライブマイグレーション

WebAssembly は 2017 年に登場したプログラム形式であり、多言語からのコンパイルに対応し、多様な環境で高速かつ効率的に動作するように設計されている。本プロジェクトの特徴は、この WebAssembly のポータビリティ(環境非依存性)を活かし、従来のプロセスや OS のライブマイグレーションに存在する移行元と移行先の環境同一性という制約を取り払い、異なる環境間でのプログラムの移行を可能にした。特に、独自のバイナリ変換ツールを用いて WebAssembly モジュール自体に中断・再開機能を付与することで、特定のランタイム実装に依存せず、ハードウェア特性やリソース制約に応じた最適なランタイムの選択が可能になる。

(ii) アスペクト指向プログラミングによる通信処理の隠蔽

SDK では複雑なネットワーク通信や状態移送の処理を隠蔽する C/C++マクロを提供している。従来では、複雑なネットワークプログラミングを記述する必要があったが、ユーザはアスペクトを記述するだけでサービスの分散実行を実現することができ、容易にサービスを開発することができる。

5. 期待される効果

本プロジェクトの成果により、クラウドとエッジを横断する分散システムの開発手法が大きく変革されることが期待される。特に、移動体(ドローンや自律走行ロボット)が通信圏内を移動しながら計算リソースを動的に切り替えるようなユースケースにおいて、100ms 以下

のダウンタイムで任意の環境の間でプログラムの実行状態を移送できる技術は、リアルタイム性が要求される産業分野での強力な基盤となる。

また、WASI NN のサポートにより、エッジ側での軽量な AI 推論と、クラウド側での高負荷なデータ解析をシームレスに統合することが可能になる。サービス開発者が複雑な通信プロトコルや状態管理を設計する必要がなくなり、AI と IoT を組み合わせた高度なサービスのプロトタイピングから実運用までの期間が大幅に短縮され、関連技術分野の活性化に寄与することが見込まれる。

6. 普及（または活用）の見通し

本ソフトウェアは、GitHub を通じたオープンソースソフトウェア（OSS）としての公開に加え、導入を容易にするインストーラや詳細なドキュメンテーションを整備しており、広範な開発者コミュニティでの活用を想定している。今後は特定のランタイムに依存しない WebAssembly バイナリ変換の汎用性を活かし、特定の分野に限らない利用を推進していきたい。

今後の展望としては、本プロジェクトを研究として発展させていき、国際会議や論文誌への論文投稿を目指すことを考えている。開発したバイナリ変換技術は、異種環境間かつ異種ランタイム間でライブマイグレーションを実現することができる点で革新的だが、プログラムサイズや実行時オーバーヘッドの増加というデメリットもある。この 2 つの指標でバイナリ変換の性能を評価し、改善を加えていきたいと考えている。また、ファイルシステムやネットワークなどのプログラム外部の状態をライブマイグレーション実施後に維持する機能はまだサポートされていない。ライブラリ OS は、このような OS の機能を WebAssembly ランタイム側に実装する技術であり、この課題の解決策になると考えている。

7. クリエータ名（所属）

田村 来希（京都大学 大学院情報学研究科 社会情報学コース 修士課程）

（参考）関連 URL

- Kafu SDK のリポジトリ：https://github.com/tamaroning/kafu/
- バイナリ変換ツールのリポジトリ：https://github.com/tamaroning/binaryen/tree/snapify
- デモのリポジトリ：https://github.com/tamaroning/kafu-demo
- ドキュメンテーション：https://tamaroning.github.io/kafu