Wasm を実行する unikernel と Wasm コンパイラ
- クラウド上での Wasm の実行を最適化 -

1　背景

1.1　WebAssembly（Wasm）

WebAssembly（Wasm）とは Web ブラウザ上で実行可能な仮想命令セットである。従来ブラウザ上で実行されるプログラムには JavaScript がデファクトスタンダードとして用いられてきた。しかし JavaScript にはパフォーマンス上の問題があったため、より高速な実行形態を求めて Wasm が開発された。近年では Wasm の実行環境はブラウザにとどまらず、各種 OS 上で直接実行するランタイムが登場している。Wasm の様々なプログラミング言語からコンパイル可能で、CPU や OS に依存しないポータブルなバイナリ形式であるといった特徴を持ち、応用範囲は IoT やサーバレスコンピューティングなど多岐にわたる。

1.2　コンテナの代替としての Wasm

Wasm はコンテナに代わる次のアプリケーションの実行環境として期待されている。図 1 に仮想化技術とその管理単位の変遷を示す。まず以前から仮想化技術として広く使われていたのは仮想マシンである。仮想マシンとはハイパーバイザというソフトウェアによってエミュレートされる仮想的なコンピュータである。アプリケーション毎に仮想マシン上でゲスト OS を動作させ、その上でライブラリやアプリケーションを実行する。続いて現代のアプリケーション開発の場で広く用いられているのがコンテナである。コンテナではホスト上のファイルシステムやネットワークなどのリソースを名前空間ごとに隔離することで仮想化を実現する。各コンテナはホストカーネルを共有したままライブラリやアプリケーションのみを隔離するため、ゲスト OS が必要な仮想マシンよりも軽量である。しかしコンテナの中には実行するアプリケーションの他に、依存するライブラリや言語ランタイムといった多くのソフトウェアが含まれている。

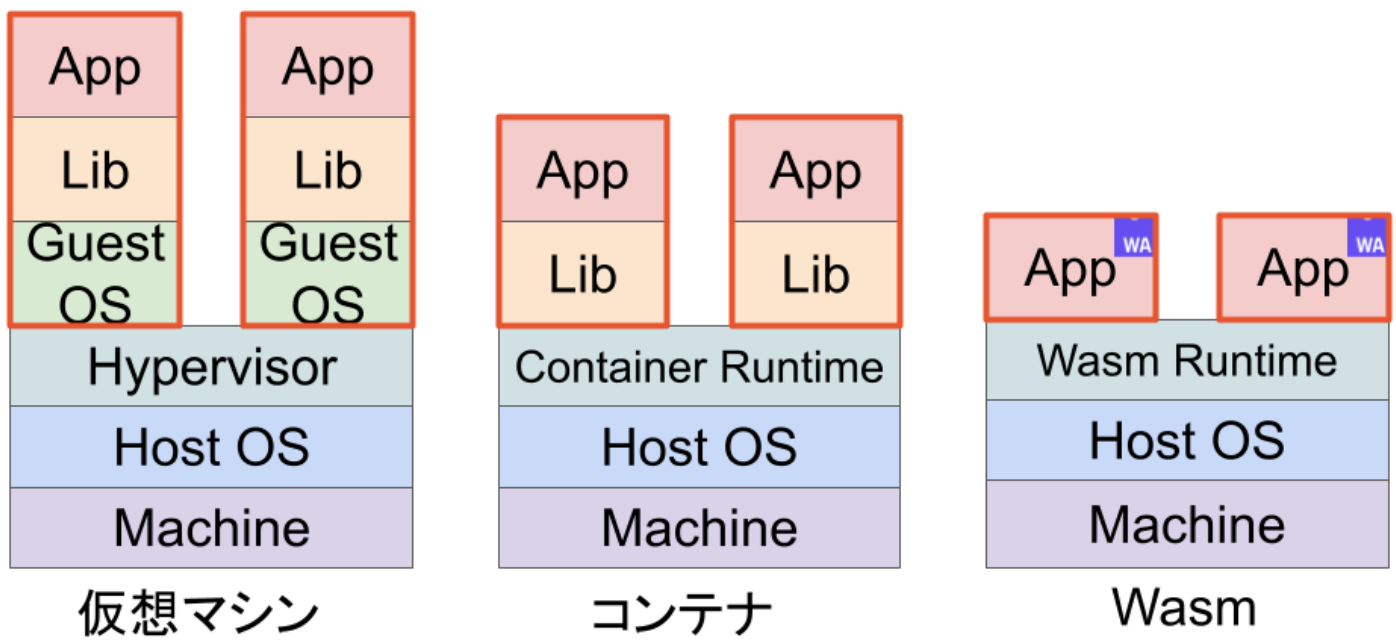

図 1: 仮想マシン、コンテナ、Wasm の仮想化技術とその管理単位

一方 Wasm では、図 1 右図に示すように、アプリケーションは単一のスタンドアロンな Wasm バイナリにビルドされる。この生成された Wasm バイナリは Wasm ランタイムさえあれば他のライブラリ等を必要とせずそのまま実行できる。つまり Wasm バイナリにはそのソフトウェアが実行されるのに必要十分なコードが含まれている。そのため Wasm はコンテナと比較してライブラリなどの依存ソフトウェアがない分、軽量であると言える。

1.3 unikernel

unikernel は軽量なカーネルの構成方法の一種である。unikernel では単一のアプリケーションがビルド時にカーネルと静的にリンクされ、1 つのカーネルイメージにまとめられる。よって unikernel 上では 1 つのアプリケーションのみ実行し、カーネルもアプリケーションも単一のアドレス空間でカーネルモードで動作する。汎用性を重視した Linux や Windows などの一般的な OS と比較して、unikernel はコードサイズやメモリフットプリントの小ささ、常にカーネルモードで動作することによる実行速度の向上、起動時間が短いといった利点を持つ。

2 目的

1.2 節でははは Wasm の軽量性について述べたが、パブリッククラウド上ではこの軽量性について課題がある。それは、パブリッククラウド上では隔離性のため Wasm を仮想マシン上で実行しなければならないことである。パブリッククラウド上では多くのユーザがクラウドベンダーの用意したデータセンター上でワークロードを実行するが、ユーザ間の干渉を防止するためにそれぞれのワークロードを十分に隔離しなければならない。多くのクラウドベンダーでは、ハイパーバイザを用いて仮想マシンとして計算リソースを分離することでこれを実現している。そのため、Wasm をパブリッククラウド上で実行する際には、図 1 とは異なり、仮想マシン上で実行する必要がある。しかしこのような状況では、1.2 節で述べた軽量性とは裏腹に、仮想マシンのレイヤまで含めた実行環境全体は軽量ではなくなる。そのため実行環境の大きさゆえに性能のオーバーヘッドが発生するという問題がある。

本プロジェクトはパブリッククラウド上で仮想マシンによる隔離性を維持したまま、実行環境のオーバーヘッドを削減して Wasm を実行することを目的とする。

3 開発の内容

本プロジェクトでは Wasm の実行に特化した unikernel、Mewz を開発した。従来サーバ上で Wasm を実行する際には、図 2 左図のように、Linux などの汎用的な OS の上で Wasm ランタイムを動作させることで Wasm を実行していた。一方で Mewz は unikernel であるため、図 2 右図のようにカーネルの内部で Wasm が実行される。また Mewz には Linux などの汎用的なカーネルとは異なり Wasm を実行するのに最低限の機能しか持たない。そのため Mewz によって Wasm を unikernel として仮想マシン上で動かすことで、ゲストカーネルのオーバーヘッドを削減することができる。このようにして 2 節で述べた、仮想マシン上で Wasm を動かす際のオーバーヘッドを削減する。

また Mewz 上で Wasm を実行するために、Wasm バイナリをターゲットマシン上のネイティブコードに変換する AoT（Ahead-of-Time）コンパイラ、Wasker を開発した。Wasker は 1 つの Wasm バイナリから 1 つのオブジェクトファイルを出力する。この際、Wasm から呼び出されている WASI（WebAssembly System Interface）の関数を未解決シンボルにしたままにしておく。WASI とは Wasm にシステムのリソースを提供する API の仕様であり、Wasm におけるシステムコールのような役割を持つものである。こうすることによっ

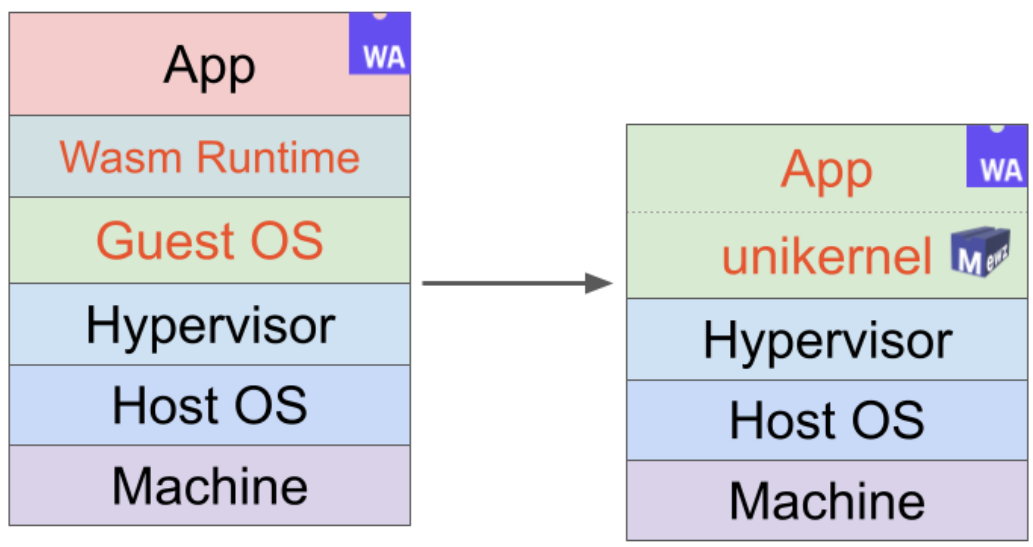

図 2: 仮想マシン上で Wasm を実行するアーキテクチャの比較

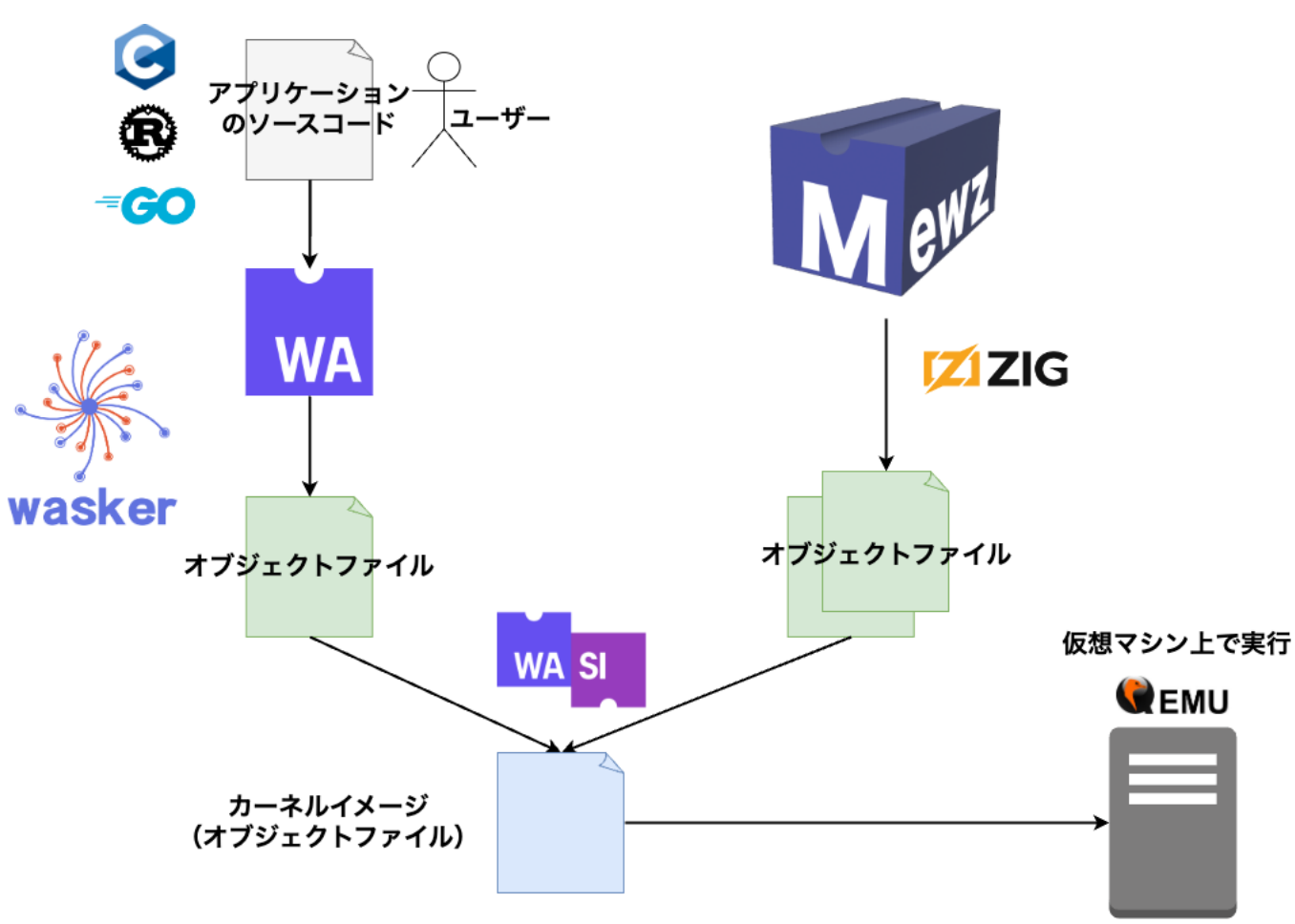

図 3: Mewz と Wasker が動作する様子

て、図 3 に示すように Mewz が提供する WASI の関数と Wasm からの WASI の関数への呼び出しをリンク時に結びつけることができる。

さらに、Kubernetes などのコンテナ管理ソフトウェアで Mewz を管理することを可能にするソフトウェア、containerd-shim-mewz を開発した。

4　従来の技術との相違

既存の Wasm ランタイムの多くは Linux などの汎用的なカーネル上でユーザプロセスとして動作する。そのため WASI を呼び出す度にシステムコールを発行し、カーネルモードからユーザモードへの切り替えが発生する。一方、Mewz は常にカーネルモードで動作するため、WASI の実行は関数呼び出しという形で呼び出され、その際に大きなオーバーヘッドは発生しない。実際に WasmEdge という既存の Wasm ランタイムを仮想マシン上で実行する場合との比較を行い、静的ファイル配信のワークロードにおいて、Mewz は約 30% のスループットの向上を確認した。

既存の unikernel は互換性と軽量さのトレードオフがある。アプリケーションに対して独自の API を提供する unikernel は、互換性のためのレイヤを削減できるが、アプリケーションの移植性が低い。一方 Linux などとの互換を提供する unikernel は、300 以上のシステムコールや標準 C ライブラリとの互換を保つための実装が必要であり、unikernel の利点である軽量さが損なわれる。Mewz では WASI をカーネルインタフェースとして採用しているため、Linux のシステムコールよりも少ない WASI を実装することで、アプリケーションに対して WASI という仕様で定められた互換性の提供が可能である。

5　期待される効果

Mewz は隔離性を維持しつつ高パフォーマンスに Wasm を実行することができる。パブリッククラウドなど隔離性が求められる場所で Wasm を実行する需要は今後も高まると考えられ、Mewz はその実行基盤として一つの選択肢となることが期待される。

Wasker の仕組みは Mewz のみならず任意の OS 上で Wasm を実行することを可能にする。Wasker によって WASI 互換な OS の開発が容易になるため、カーネルインタフェースとして WASI が普及することへの貢献が期待される。

6　普及の見通し

本プロジェクトの成果をカンファレンスや論文での発表を通じて Wasm コミュニティに発信し、Wasm という技術全体の発展に貢献する。また、これらの活動を通して新たなユーザやコントリビュータを獲得し、OSS プロジェクトとして発展させていく。

7　クリエータ名（所属）

- 上田 蒼一朗（京都大学工学部情報学科）
- 野崎 愛（東京大学情報理工学系研究科システム情報学専攻）

（参考）関連 URL

- Mewz のソースコード：`https://github.com/mewz-project/mewz`
- Wasker のソースコード：`https://github.com/mewz-project/wasker`