複数の仮想化基盤に対応する実行環境構築支援ツールの試作

2013SE004 安東凌輔　2014SE091 城川渉

指導教員：宮澤元

1　はじめに

ネットワーク経由でサービスを提供するクラウドコンピューティング (クラウド) が普及している．クラウドはクラウド基盤ソフトウェアを用いて多数の物理サーバを集約することで実現され，多数のユーザが利用することが可能となる．ユーザの利用形態も多様なので，計算リソースを物理的構成にとらわれずに柔軟な形で提供したり，限りあるそれら物理的資源を効率的に活用するなどの必要性がある．そこでクラウドでは，計算リソースを抽象化し，物理ハードウェア上で動作するホストソフトウェアが擬似的なコンピュータ環境を生成する仮想化技術が利用されている．ゲストソフトウェアは，ホストソフトウェアが提供する擬似的なコンピュータ環境を利用して動作する．

クラウドでは仮想化技術を用いることで必要な資源を必要なときに必要なだけ利用できるので，科学技術計算などの ハイパフォーマンスコンピューティング (HPC) 用途に利用することができ，HPC クラウドとして注目されている．しかし，クラウドを HPC に利用する上で仮想化オーバヘッドによる性能低下が問題となる．クラウドで多く使われてきたハイパーバイザ型仮想化では擬似的なコンピュータ環境である仮想マシン (VM:Virtual Machine) とそれを管理するハイパーバイザによって仮想化を実現しているが，ハイパーバイザ型仮想化ではハードウェア環境をエミュレートするオーバヘッドが生じる．また，VM を利用する際，各物理マシンのハイパーバイザを利用するので，ハイパーバイザが VM をスケジューリングし，VM がさらに内部で動いているプロセスのスケジューリングを行うといったスケジューリング処理の重複のオーバヘッドが生じる．こうしたオーバヘッドは VM のパフォーマンスに悪影響を及ぼす．そこで，一般的に使われてきたハイパーバイザ型仮想化に代わり，近年では Docker[1] のようなコンテナ型仮想化技術が主流になりつつある．コンテナ型仮想化はハイパーバイザを用いない通常の OS 上でアプリケーションの実行環境を隔離して動作させる技術なので，ハードウェア環境のエミュレートの必要はなく，ハイパーバイザ型仮想化に比べ，高いパフォーマンスを発揮する．また，ゲスト OS も必要としないので，スケジューリングの重複という問題も解消される．ハイパーバイザ型仮想化のオーバヘッドを低減する別の方法として，クラウドサービスを提供することに特化して，LibraryOS の技術を用いる Unikernels[2] のようなアプローチも提案されている．これは VM 上で動作する OS として軽量 OS を用いることでパフォーマンスの向上を実現している．また，1 つの VM に 1 つのアプリケーションがシングルスレッドで動くので，VM 内でプロセスのスケジューリングを行うために生じるスケジューリングの重複の問題を解消できる．

先行研究 [3] では Unikernels と Docker におけるアプリケーションの性能比較を行い，どちらの性能が高いかはアプリケーションの特性や条件によって異なることを明らかにしている．アプリケーションの特性や条件によってどの技術を用いたクラウドのパフォーマンスが優れているかは変化するので，状況によって最適な仮想化基盤を選択することが重要である．そこで同研究では動作しているアプリケーションの特性に応じて適切な仮想化環境を動的に選択できる IaaS クラウド基盤を提案しており，アプリケーションの動作基盤をコンテナまたは軽量 OS を使った VM に動的に変更することで，アプリケーションの動作パフォーマンスの向上を目指している．

本研究の目的は一つのアプリケーションに対してコンテナ型仮想化基盤と Unikernels のそれぞれの実行環境を構築するツールを作成することである．このような実行環境構築ツールは，我々が取り組んでいるアプリケーションに応じて最適な仮想化方式を選択できるクラウド基盤ソフトウェアの実現に欠かせない．コンテナ型仮想化基盤の実行環境を構築するに当たって Docker を用い，Unikernels として Rumprun[4] を用いる．

本ツールを用いると，アプリケーションのソースから，Docker コンテナと Rumprun の実行環境の両方を構築できる．ツールを用いて，実際に Web サーバの実行環境をそれぞれの仮想化方式で構築できることを確認する．

2　関連研究

ここでは，本研究で用いるハイパーバイザ型仮想化，コンテナ型仮想化や軽量 OS を用いた VM，またハイパーバイザの Xen，コンテナ型仮想化ツールである Docker や Unikernels の一つである Rumprun といった技術について述べる．

2.1　ハイパーバイザ型仮想化

ハイパーバイザ型仮想化では，ハイパーバイザと呼ばれる仮想化ソフトウェアを物理マシン上で直接起動し，ハイパーバイザによって複数の VM が作成される．OS とアプリケーションは，VM 上で動作する．図 1 にハイパーバイザ型仮想化におけるアプリケーション実行環境の構成を示す．問題点として，ハイパーバイザが VM をスケジューリングし，VM 上で OS がスケジューリングを行うという二重のスケジューリングが発生している点や．ハイパーバイザが介在することにより，オーバヘッドが生まれ性能が低下する点などが挙げられる．

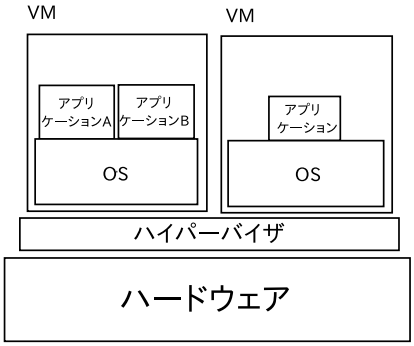

図 1 ハイパーバイザ型仮想化の構成

2.1.1 Xen

Xen はオープンソースのハイパーバイザである．Xen 上ではコントロールドメインと呼ばれる Domain-0 とゲストドメインと呼ばれる Domain-U の 2 種類の VM が動作する．図 2 に Xen におけるアプリケーション実行環境の構成を示す．Domain-0 では Domain-U を管理するための管理コマンドを備えており，ホスト OS が動作する．Domain-U ではゲスト OS が動作し，Xen の管理権限は持たない．

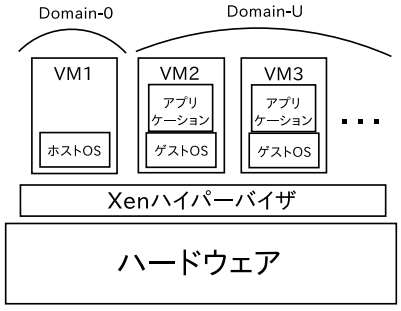

図 2 Xen の構成

2.2 コンテナ型仮想化

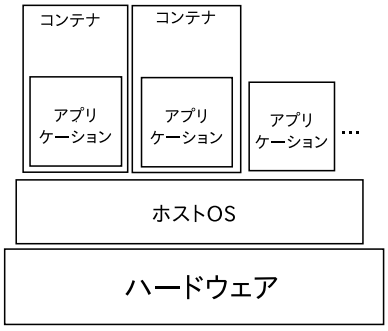

図 3 コンテナ型仮想化の構成

コンテナ型仮想化とは OS 上に隔離されたアプリケーション実行環境 (コンテナ) を構築する技術である．図 3 にコンテナ型仮想化を用いたアプリケーション実行環境の構成を示す．コンテナ型仮想化では起動する全てのプロセスはホスト OS 上で直接起動する．図 1 のようにハイパーバイザにより複数の VM が仮想化され，その上で OS とアプリケーションを動作させるハイパーバイザ型仮想化に対し，コンテナ型仮想化はホスト OS 上のプロセスの一部をグループ化し，他のグループやグループに属していないプロセスから隔離された空間 (コンテナ) で動作させる．

コンテナ型仮想化を用いた場合のメリットとして，起動する全てのプロセスはホスト OS 上で直接起動するので，仮想的なハードウェア環境から仮想化するハイパーバイザ型仮想化に比べ，オーバヘッドが小さいという点が挙げられる．また，ハイパーバイザが動作していないのでハイパーバイザ型仮想化における VM をスケジューリングするオーバヘッドも解消される．一方ホスト OS は固定されるので，プロセスごとに様々な OS を利用することはできないというデメリットがある．

2.2.1 Docker

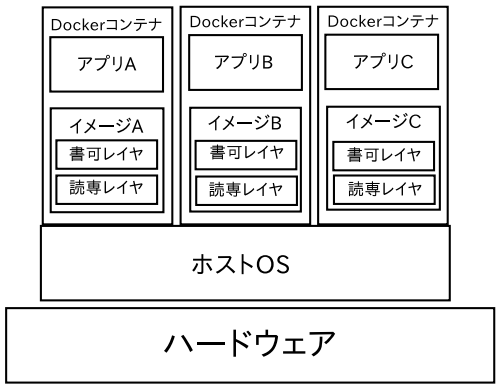

図 4 Docker の構成

Docker はオープンソースのコンテナ型仮想化ソフトウェアである．コンテナの実行に必要な部品すべてを含む「イメージ」をメモリ内に展開し，実際に実行する．ホスト上のリソースにハイパーバイザを通してしかアクセスできない仮想マシンと比べ，コンテナはアプリの実行に何らかのプログラムによる仲介を必要としないので，性能向上が期待できる．図 4 に Docker によるアプリケーション実行環境の構成を示す．コンテナはホスト OS のカーネルを共有するが，コンテナ同士は，通常はそのまま相互に参照できない．イメージはファイルの実体やメタ情報を含むレイヤの集合であり，各レイヤは読み込み専用である．コンテナの環境を更新するときには新しいレイヤに，古いレイヤとの差分情報を書き込むことができる．

2.3 軽量 OS を用いた VM

ハイパーバイザ型仮想化におけるスケジューリングの重複によるオーバヘッドを低減するために VM 上で軽量 OS を動作させ，その上で単一のアプリケーションを動作させる Unikernels と呼ばれるクラウドの構成法が提案されている [2]．図 5 に軽量 OS を用いた VM におけるアプリケーション実行環境の構成を示す．基本構成はハイパーバ

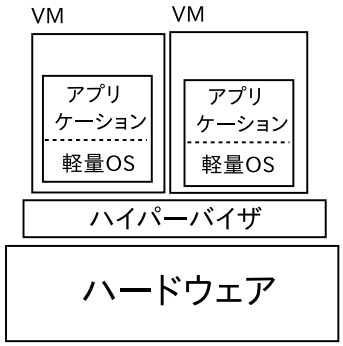

図 5 軽量 OS を用いた VM の構成

イザ型仮想化と同様だが，VM では軽量 OS 上で単一のアプリケーションが動く．

軽量 OS を用いた VM を用いるメリットとして，必要最低限の機能をもった軽量 OS をハイパーバイザ上で動作させることにより，ハイパーバイザ型仮想化に比べて高いパフォーマンスを発揮することができるという点がある．また，一つの VM で一つのアプリケーションをシングルスレッドで動かすという特性からハイパーバイザ型仮想化における VM のスケジューリング多重化によるオーバヘッドは存在しない．一方，軽量 OS 上のアプリケーションは使用する OS によって，実装言語やライブラリが制限されるので，ユーザーが自由にアプリケーションを実装できないというデメリットがある．

2.3.1 Rumprun

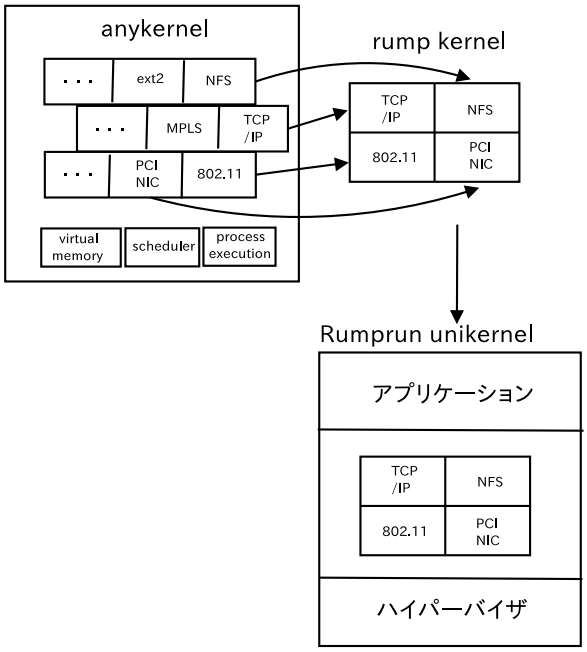

図 6 anykernel，rumpkernel，Rumprun 間の関係

Rumprun は変更されていない既存の POSIX ソフトウェアを Unikernels として実行できるソフトウェアスタックであり，rumpkernel はフリーでポータブルなコンポーネント化されたカーネル品質ドライバを提供する rumpkernel[6] に基づいて実現されている．rumpkernel のドライバは，anykernel である NetBSD のリポジトリから取得できる．anykernel とはカーネルのドライバを変更せずにさまざまな構成で実行できるようにするカーネルコードの構成のことである．現状では NetBSD が anykernel である唯一のオペレーティングシステムカーネルである．

図 6 は anykernel，rumpkernel，Rumprun 間の関係を示したものである．Rumprun は，anykernel から必要なコンポーネントを選択し，抽出した rumpkernel により軽量な仮想環境を構築する．

Rumprun のモデルではシングルプロセスしか扱うことができない．従って，fork 関数を用いるマルチプロセスアプリケーションは適合しない．また，アプリケーションで使用されるビルドシステムがクロスコンパイルをサポートしなければならないという制限が存在する．

3 実行環境構築支援ツール

アプリケーションの実行状況に合わせて最適な仮想化方式を選択できるクラウド基盤ソフトウェアでは，複数の仮想化基盤上で同じアプリケーションを動作させる必要がある．ここではアプリケーションに対してコンテナ型仮想化と Unikernel 型仮想化の双方の実行環境の構築を支援するツールについて述べる．

3.1 仮想化基盤における実行環境の構築

本研究ではコンテナ型仮想化基盤として Docker を用いる．Docker は OS のイメージの種類が豊富なので，動作させるアプリケーションへの対応がし易い．また，Unikernels として Rumprun を用いる．Rumprun は NetBSD から必要なコンポーネントを抽出し，軽量 OS を構築できるので，Rumprun 上で動作させるためのアプリケーションの変更が少なくてすむ．

3.2 ツールの実装

VM，コンテナの生成までの一連の UNIX コマンドをシェルスクリプトに記述して，ツールを作成した．ツールは一つのアプリケーションから Docker コンテナと Rumprun の VM をどちらも生成する．ツールでの各仮想化基盤の働きは，Docker はコンテナを生成し，コンテナ上に必要な環境とプログラムを入れ，動作させる．Rumprun は，Xen 上に VM を作り，その上で VM のイメージファイルに変換したプログラムを動作させる．

アプリケーションの性質によってツールが各基盤の実行環境を構築する手順，即ちスクリプトに記述する UNIX コマンドは変わるが，これは主に選択する Docker のベースイメージと Rumprun のツールチェインをアプリケーションに対応させるコマンドに記述を変更する必要があるからである．

また Rumprun において，いくつかのアプリケーション

はパッケージでサポートされているが，パッケージが存在せず，Rumprun に対応していないアプリケーションを動作させる場合には修正プログラムを作成する必要がある．

3.3 ツールの動作例

3.2 節で示したスクリプトを作成し，以下の環境で実際にツールを動作させ，HTTP サーバの動作を確認した．表 3.1 に示した PC2 台を実験に使用した．仮想化基盤に Xen と Docker を使用し，動作させる通常の OS に Ubuntu，軽量 OS に Rumprun を使用した．Xen を用いる場合の VM は表 3.2 に示す仕様とした．Docker を用いたコンテナは表 3.3 に示す仕様とした．

表 1 PC の仕様

CPU	インテル®Core™i7-3770 プロセッサー
コア数	4 コア 8 スレッド
メモリ	8GB
HDD 容量	1TB
OS	Ubuntu 16.04LTS 64bit
通信規格	1000BASE-T

表 2 VM の仕様

CPU コア数	1
メモリ	8GB
OS	Ubuntu 16.04LTS 64bit Rumprun
ハイパーバイザ	Xen 4.6.5

表 3 コンテナの仕様

CPU コア数	4
メモリ	7.68GB
コンテナ	Docker version 17.09.0-ce
ベースイメージ	library/ubuntu

3.3.1 独自 HTTP サーバ

作成した実行環境構築ツールの動作を単純なアプリケーションを用いて確認するために，1 つの C 言語ソースファイルからなる HTTP サーバプログラムを作成した．Docker では，ベースイメージ Ubuntu をもとにコンテナを生成，擬似ターミナルを割り当て，コンテナはホスト OS にポートを開く．bash を起動し，gcc など必要な環境をインストールした後，C 言語プログラムをコンパイル，実行する．Rumprun では，まずツールチェインへの PATH を通してコマンドの実行を可能にし，C 言語プログラムのコンパイルをする．その実行ファイルから VM イメージファイルへの変換を行い，IP アドレスを指定し，変換された VM イメージファイルからプログラムの実行を行う．

3.3.2 Nginx

実用的に利用される HTTP サーバの例として，Nginx の実行イメージを本実行環境構築支援ツールを用いて作成した．Docker では，C プログラムと同様にポートを開き，擬似ターミナルを割り当てたコンテナを生成する．bash を起動し，必要な環境をインストールした後，Nginx のインストールを行う．Rumprun では，パッケージから Nginx をビルドし，Nginx の実行ファイルを VM イメージファイルに変換して IP アドレスを指定し，各 VM で実行する．ただし，実際には Nginx を Rumprun 上で動作させるための修正が必要なので，ツールを用いた仮想化基盤の生成が完全に自動化されていない．

4 おわりに

本研究では各仮想化基盤に対応できるアプリケーションの実行環境の構築を支援するツールを作成した．本ツールでは，一つのアプリケーションから Docker コンテナと Rumprun を用いた VM の両方が生成できる．実際に Web サーバの実行環境をそれぞれの仮想化方式で構築できることを確認した．

今後の課題は，様々なアプリケーションに対してコンテナ型仮想化と軽量 OS を用いた VM の双方の実行環境を生成するより一般的な枠組みを作ることである．現状では軽量 OS を用いた VM の制限により動作させられないアプリケーションが存在するので，軽量 OS を用いた VM のシステムにおいてマルチスレッドアプリケーションを効率的に動作させるシステムが必要となる．その後，アプリケーションを動的に別の仮想化環境で動作させるシステムについて検討する．

参考文献

[1] “Docker Documentation — Docker Documentation，”https://docs.docker.com/，(access 2017-10-4).

[2] A. Madhavapeddy， et al: “Unikernels: Library Operating Systems for the Cloud，” SIGPLAN Not.，vol. 48，no. 4，pp. 461-472，Mar. 2013.

[3] 田尻翔太: “IaaS 環境におけるマルチプロセスアプリケーションを考慮した仮想化基盤の動的構成管理に関する研究”，南山大学大学院，理工学研究科，2016 年度 修士論文要旨集.

[4] “rumpkernel/wiki，” https://github.com/rumpkernel/wiki/wiki，(access 2017-10-3).

[5] “The Xen Project，” https://www.xenproject.org/，(access 2018-01-06).

[6] “Rump Kernels，” rumpkernel.org/，(access 2018-01-06).