異種 OS 上のコンテナ型仮想化環境間でのライブマイグレーション実現方式の検討

高川 雄平　松原 克弥

オペレーティングシステム (OS) が提供する計算資源や名前空間をアプリケーション毎に多重化して隔離することで，ひとつの OS 上に独立した複数の実行環境を実現するコンテナ型仮想化が注目されている．一方，ハードウェア資源を仮想化する従来方式と比較して，コンテナ型仮想化の実現方式は OS への依存度が高く，異なる OS 上で実現されたコンテナ環境間では互換性がない．本稿では，仮想化技術のキラーアプリケーションであるクラウドコンピューティング環境において，負荷分散や可用性実現の要として広く利用されているライブマイグレーション技術に着目して，動作中のコンテナ実行環境を異なる種類の OS が動作する別の計算機へ動的に移動して実行の継続を可能にする方式を検討する．OS 毎のシステムコールや ABI の変換，プロセス実行状態の取得と復元方式の統一化，資源隔離機能の対応づけにより，Linux と FreeBSD を対象とした異種 OS 間のライブマイグレーション実現方式を提案する．

The container-based virtualization, that multiplexes and isolates computing resource and name space which operating system (OS) provides for each application, has been recently attracted. As compared with the conventional hardware virtualization, containers run on different OSs may not be compatible because their container implementations must depend on each underlying OS. In this paper, we focus on the live migration since it is one of the most important technology for realizing load balancing and availability in cloud computing, that is a major application of the virtualization. For Linux and FreeBSD as the 1st target, we describe that how system calls and ABI can be converted, how running containers can be captured with an uniform representation and they restore in both OSs, and how an uniformed resource isolation can be realized.

1 はじめに

1.1 背景

クラウドコンピューティングを支える基盤技術として，仮想化技術がある．特に，近年，アプリケーション毎の実行環境の軽量なコンパートメント化を目的として，コンテナ型仮想化が注目されている．コンテナ型仮想化は，OS が提供する資源を分離・制限し，他の OS を起動させずに異なる環境を構築できる軽量な仮想化を実現する．一方，ハードウェア仮想化では，VM と呼ばれる仮想化したハードウェアをハイパーバイザが作成・管理し，VM の上で OS を起動させる必要があり，コンテナ型仮想化と比較するとオーバーヘッドが大きい (図 1 参照)．

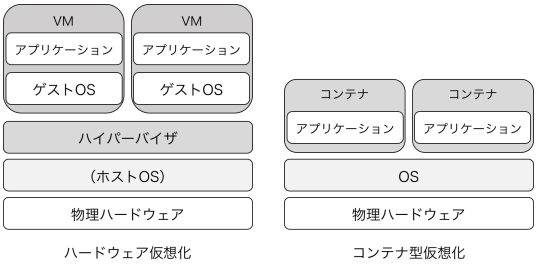

図 1　ハードウェア仮想化とコンテナ型仮想化

また，クラウドコンピューティング実働環境では，負荷分散と可用性を実現するためにライブマイグレーションが活用されている．ライブマイグレーションは，実行中のサービスを動的に別のマシンに移行す

A Study on Implementing Container Live Migration among Heterogeneous OS Platforms

Yuhei Takagawa, Katsuya Matsubara, 公立はこだて未来大学 システム情報科学部, School of Systems Information Science, Future University Hakodate.

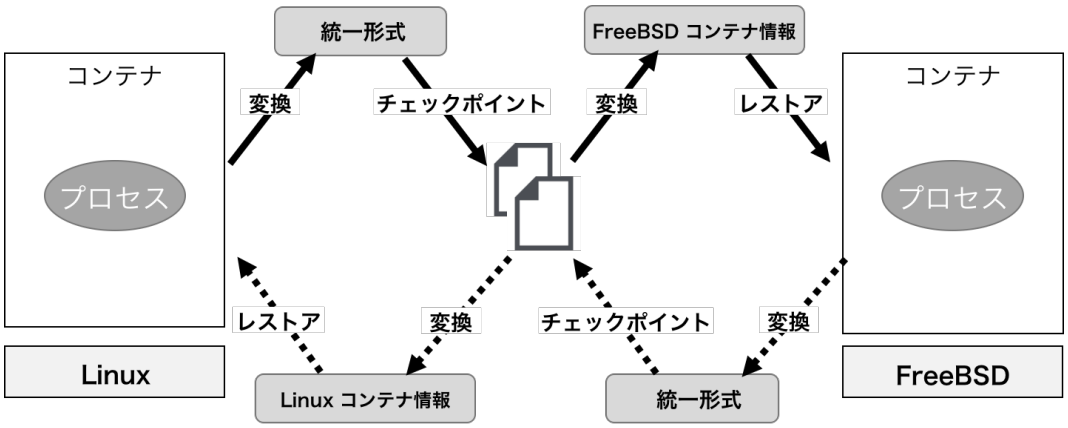

図 2　本提案システムの概要

る技術である．コンテナ型仮想化におけるライブマイグレーションも実現されており，Linux では CRIU [1] [2]，FreeBSD では FreeBSD VPS(Virtual Private System) [6] が実装として存在する．OS への依存度が大きいコンテナ型仮想化では，OS 上の計算資源や API，コンテナ実現方式が異なるため，異なる OS 環境間でコンテナをマイグレーションすることができない．

1.2　提案システム

本稿では，異種 OS 間でのコンテナ型仮想化におけるライブマイグレーションの実現方式を検討する．本検討では，まず Linux と FreeBSD を対象とすることで，技術的課題の洗い出しと実現方式の有効性を確かめる．図 2 は Linux と FreeBSD 間におけるコンテナ型仮想化のライブマイグレーションの概要図である．チェックポイントとはコンテナの状態をファイルに保存することで，レストアでは保存したファイルからコンテナを復元する．本提案では，コンテナの実行状態や環境に関する情報を統一形式に変換してチェックポイントでファイルに保存し，レストアでファイルから取得した情報を OS に適した形式に変換してコンテナの実行状態や環境を復元する．

本提案の異種 OS 間のライブマイグレーションを実現することにより，以下に述べるような効果が期待できる．

- **OS が混在するシステムにおいて全ての計算資源を十分に活用する負荷分散の実現**
  異種 OS が利用している計算資源も活用することができるため，OS 毎の資源だけではなく，システム全体の資源を有効活用できる．
- **メンテナンス時の可用性の向上**
  ある OS を同時にメンテナンスをしなければならない場合に，異種 OS にサービスを移行することができるため，サービスを停止せずにメンテナンスが可能である．

以降，第 2 章では，コンテナ型仮想化とライブマイグレーションに関する既存技術を述べる．第 3 章で，異種 OS 間でのコンテナ・ライブマイグレーションを実現するための技術的課題を示し，その解決法について論じる．第 4 章では，実装状況について述べる．最後に，5 章でまとめと今後の課題について述べる．

2　コンテナ型仮想化

コンテナ型仮想化は，ハードウェア仮想化とは異なり，計算資源や名前空間を分離することでアプリケーションごとに異なる環境を構築できる仮想化である．コンテナは分離されたアプリケーション実行環境のことである．コンテナの実現には，Linux では cgroups や Namespace，FreeBSD では Jail という機能が利用されている [3] [7]．

cgroups は Linux のプロセスに対して使用する資源の制限を設けることができる機能である．例えば，

CPU 利用時間 (CPU 使用率) やメモリ利用量などを制限することができる．Namespace は Linux のプロセス ID やディレクトリ，ネットワークなどを制御するための名前空間を分離できる機能である．

Jail は Namespace にあたる機能で FreeBSD のディレクトリやネットワークなどを制御するための名前空間を分離する機能である．分離した空間は他の空間のプロセス ID やディレクトリには直接関係しないため，同じプロセス ID を分離した空間ごとに利用することができる．

2.1 コンテナ・ライブマイグレーション

ライブマイグレーションは，仮想化環境上で動作しているプロセスを停止させずに仮想化環境ごと別のマシンに移動する技術で，負荷分散や可用性を目的に行われる．CPU やメモリなどのハードウェア資源 (以下，H/W 資源) の状態をファイルに保存し転送，転送されたファイルから H/W 資源の状態を復元することでライブマイグレーションは行われている [8]．H/W 資源の状態保存とファイルの転送は動的に行われ，H/W 資源の変更が一定以下になるまで実行される（プレダンプ，Pre-dump）．

コンテナ型仮想化環境のライブマイグレーションでは H/W 資源の状態を全て保存されたファイルと保存するための処理を「チェックポイント」と呼び，H/W 資源の状態を復元することを「レストア」と呼ぶ．サービスを全く停止させないことはできず，チェックポイントの転送中は停止することになる．停止時間はサービスに影響を与えない範囲にしなければならない．コンテナ・ライブマイグレーションを行うには，コンテナ内で動作しているプロセスのマイグレーションとコンテナの隔離・分離状態のマイグレーションが必要である．また，Linux でライブマイグレーションを実現できる CRIU，FreeBSD でライブマイグレーションを実現した FreeBSD VPS が存在する．

2.2 CRIU

CRIU(Checkpoint/Restore in Userspace) は，OpenVZ や LXC，Docker コンテナのライブマイグレーションを可能にするオープンソースソフトウェアで Linux 上で動作する．CRIU の機能は主にプロセスのチェックポイントを作成することとプロセスのレストアを行うことである．ライブマイグレーションを行うマシンには同じコンテナが存在することが重要である．また，OS の再起動やカーネルの再ビルドを必要としない．

CRIU の内部処理を簡単に説明する．プレダンプとチェックポイントは，ptrace() や procfs から H/W 資源の情報を収集し，Google Protocol Buffers [4] の形式でファイルに書き込みを行う．レストアは，ファイルから Google Protocol Buffers を読み取り ptrace() を用いて H/W 資源の状態を復元している．Google Protocol Buffers は異なるプログラム言語でもデータの受け渡しが簡単にできるようにしたデータ形式である．H/W 資源以外にディレクトリやプロセスツリーといったディレクトリ構成やプロセスの親子関係などの情報も対象となっている．

2.3 FreeBSD VPS

FreeBSD VPS(Virtual Private System) は，FreeBSD jail[†3] のライブマイグレーションを可能にしたシステムであり，FreeBSD 上で動作する．FreeBSD VPS は VPS インスタンスというコンテナを作成する．Jail をそのままライブマイグレーションしても，プロセス ID が重複して正常に動作しないため，FreeBSD VPS ではプロセステーブルを VPS インスタンスごとに作成することで重複を回避している．CRIU はプロセスをライブマイグレーションしているが，FreeBSD VPS はプロセスではなく VPS インスタンスごとライブマイグレーションしている．

FreeBSD VPS におけるチェックポイントとレストアの内部処理を簡単に説明する．チェックポイントは，VPS インスタンスに利用しているディレクトリの同期を 2 回行い，1 回目と 2 回目の間でプロセスを停止させ，H/W 資源のチェックポイントを作成する． レストアは，メモリを復元した際に取得できる PCB[†4] を利用し，実行可能プロセスとして OS に処理の続きを行わせている．PCB はカーネル空間の情報である

†3 FreeBSD 上で作成できるコンテナ

†4 Process Control Block，プロセス制御ブロック

ため，CRIU とは異なりカーネル空間に対する処理が発生する．

3 異種 OS 間でのコンテナ・ライブマイグレーション実現方式

異種 OS 間でのコンテナ・コンテナマイグレーションを行うためには，OS による差異を吸収しなければならない．本章では，異種 OS 間でのコンテナ・ライブマイグレーションを実現するにあたっての課題と解決手法を述べる．また，本検討の対象 OS は Linux と FreeBSD とする．

3.1 技術的課題

3.1.1 システムコールの差異

Linux と FreeBSD とで大半の共通なシステムコールは同じ処理で動作する．しかし，FreeBSD バイナリは Linux では動作せず，Linux バイナリも FreeBSD では通常は動作しない．以下が 2 点が原因である．

- システムコールの番号が異なる
- システムコールの引数・パラメータの値が異なる

システムコールの open() を例にあげる．パラメータとしてパスに指定されたファイルを開く際のオプションを引数に渡す．open() のオプションの *O_CREAT* は FreeBSD では値が 0x0200 と定義されているが，Linux では値は 0x0040 と定義されている．そのため，異なる OS のプロセスを復元したとしてもパラメータの影響で正しく動作しない．なお，open() のシステムコール番号は共通である．

3.1.2 ABI の差異

ABI(Application Binary Interface) とは，システムコールやプログラムの「呼び出し方」のことで，FreeBSD と Linux では ABI が異なる．FreeBSD/x86 はシステムコールの引数をスタック経由で渡すのに対して，Linux/x86 はシステムコールの引数をレジスタ経由で渡す (表 1 参照) [9]．Linux/x86 では，システムコールの引数の上限が 5 つにが決まっており，それ以上は利用できない．関数の引数はスタック経由であるため，6 つ以上の引数を持つことができる．そのため，同じプログラムが動作していても，FreeBSD と Linux ではメモリやレジスタの状態が異なる．

表 1 x86 における Linux と FreeBSD のシステムコール引数の渡し方

引数	Linux	FreeBSD
第 1 引数	EBX	スタック
第 2 引数	ECX	
第 3 引数	EDX	
第 4 引数	ESI	
第 5 引数	EDI	
第 6 引数	×	

3.1.3 プロセス実行状態の取得と復元

本提案ではライブマイグレーションの共通フレームを CRIU とする．CRIU は ptrace() や procfs を用いることで，プロセス実行状態の取得と復元を行っている．FreeBSD に ptrace() や procfs は存在するが，Linux よりも取得できる情報が少なく，マウント情報やネットワーク情報などを procfs から取得できない．加えて，取得できるメモリやレジスタの情報が OS で異なるため，レジスタやユーザ空間の状態をそのまま復元したとしてもプロセスは正しく動作しない．

3.1.4 プロセス実行環境の隔離

第 3.1.3 章のプロセス実行状態の取得・復元によりプロセス・ライブマイグレーションを可能にしたとしても，コンテナ・ライブマイグレーションにはならない．プロセスのライブマイグレーションに加え，プロセスを隔離している状態を復元することで，コンテナ・ライブマイグレーションが可能になる．Linux と FreeBSD では，資源を隔離するために使っている機能が異なる．Linux は cgroups と Namespace，FreeBSD は Jail を用いている．隔離・制限する資源の対象や名称，値が異なるため，プロセス実行環境の隔離を復元することはできない．

3.2 実現手法

3.2.1 システムコールの変換

システムコールテーブルの順番やシステムコールの引数のパラメータを変換することで，OS が異なってもシステムコールを動作させることができる．FreeBSD に関しては，Linux エミュレータ [5] というカーネル

機能があり，システムコールテーブルとシステムコールの引数のパラメータを実際に変換し，FreeBSD で Linux バイナリが実行できる．本提案では，システムコールに関する変換には Linux エミュレータを用いる．

3.2.2 ABI の変換

カーネル機能として FreeBSD で Linux バイナリを実行する際は引数をレジスタに入れ，Linux で FreeBSD バイナリを実行する際は引数をスタックに入れるというような実装を行うと，異なる OS でもレジスタやユーザ空間の状態を同じにすることができる．これは上記した FreeBSD の Linux エミュレータ機能が実現している．Linux バイナリであれば，Linux と FreeBSD で実行ファイルを一切変更せずに動作させることができる．本提案では，ABI の変換は Linux エミュレータを用いることで実現する．

3.2.3 プロセス実行状態の取得と復元

FreeBSD でプロセス実行に最低限必要なレジスタとメモリの情報は ptrace() の GETREGS や procfs から取得する．取得できるメモリやレジスタの情報が異なる点は，Linux エミュレータであれば，Linux で取得できる情報と同様になるため，Linux エミュレータを用いて解決する．

プロセスの状態保存を 1 つの OS に統一することで，管理や運用時のオーバーヘッドを削減することができる．本提案では統一形式を Linux とし，Linux エミュレータを用いて FreeBSD プロセスの状態を Linux プロセスの状態に変換することで統一する．

また，プロセスの復元は ptrace() の SETREGS や procfs を用いることで行う．取得時と同様に，Linux エミュレータを用いることで情報の変換を行う．

3.2.4 プロセス実行環境の隔離

FreeBSD の Jail と Linux の cgroups，Namespace の機能の対応づけや数値の変換を行い，それぞれの機能で隔離している状態を再現できれば解決できる．資源の隔離に関しては，FreeBSD の Jail と Linux の Namespace の隔離状態の対応づけと相互変換を行うことで再現する．資源の制限に関しては，FreeBSD jail に対して資源の制限をつける RCTL と Linux の cgroup の制限状態の対応づけと相互変換を行うことで再現する．

4 実装状況

まず，FreeBSD で CRIU 形式のライブマイグレーションを可能にする必要がある．FreeBSD 上で ptrace() を用いてシステムコール実行前後のレジスタ情報を取得し，procfs の mem からプロセスのユーザ空間のメモリを取得可能にした段階である．

今後は取得したレジスタ状態とメモリ状態を復元し，プロセスを計算途中から実行できるようにする必要がある．また，メモリやレジスタの情報以外にも，プロセスに関するパイプ情報やファイルディスクリプタなど様々な情報がなければ，実践的な利用ができないため，その実装も行う必要がある．

5 おわりに

5.1 まとめ

本稿では，異種 OS 上のコンテナ型仮想化環境間でのライブマイグレーションの実現方式の提案を行った．対象 OS を Linux と FreeBSD とし，ライブマイグレーションの共通フォーマットを CRIU として，異種 OS 間でのプロセスとコンテナのマイグレーション方式の検討を行った．プロセスのライブマイグレーションにおける OS 毎のシステムコールや ABI の違いは，FreeBSD の Linux エミュレータを用いることで解決する．コンテナのマイグレーションにおけるコンテナ作成機能の違いは各 OS にある機能でコンテナの制限・分離を再現することで解決する．提案するシステムがもたらす効果はサーバ運用に関することが多く，OS が混在するシステムでも計算資源の活用や可用性の向上を見込める．

5.2 今後の課題

今後，FreeBSD 上でプロセスの情報を取得する方式の検討，および，CRIU 形式でプロセスをライブマイグレーションする機能を実装する必要がある．加えて，Linux と FreeBSD におけるコンテナ機能の差異の調査と対応づけ，ネットワークの移行方法を検討する．また，実働環境への対応のため，Linux エミュレータを利用しない場合や Linux エミュレータで不足

しているシステムコールへの対処の検討，Windows などの他の OS とのライブマイグレーションの実現方式の検討がある．

参 考 文 献

[1] A. Mirkin, A. Kuznetsov and K. Kolyshkin: Containers checkpointing and live migration, *In Proceedings of the 2008 Ottawa Linux Symposium*, Vol. 2, 2008, pp. 85–90.

[2] CRIU Project: CRIU Main page, `https://criu.org/`, 2010 (accessed August 14, 2017).

[3] Docker Inc.: Docker overview, `https://docs.docker.com/engine/docker-overview/`, 2017 (accessed August 16, 2017).

[4] Google Inc.: Protocol Buffers, `https://developers.google.com/protocol-buffers/`, 2017 (accessed August 16, 2017).

[5] Jim Mock: Chapter 10. Linux Binary Compatibility, `https://www.freebsd.org/doc/handbook/linuxemu.html`, 1999 (accessed August 16, 2017).

[6] Klaus P. Ohrhallinger: Virtual Private System for FreeBSD, `http://www.7he.at/freebsd/vps/`, 2010 (accessed August 14, 2017). EuroBSDCon 2010.

[7] LiWenHsu: Docker on FreeBSD, `https://wiki.freebsd.org/Docker`, 2017 (accessed August 16, 2017).

[8] Yamada, H.: Survey on Mechanisms for Live Virtual Machine Migration and its Improvements, *Information and Media Technologies*, Vol. 11(2016), pp. 101–115.

[9] 坂井弘亮: ハロー“*Hello,World*”：*OS* と標準ライブラリのシゴトとしくみ：たった 7 行の *C* プログラムから解き明かす, 秀和システム, September, 2015.