異種 OS 混在環境におけるコンテナ型仮想化のライブマイグレーション

b1014223 高川雄平
指導教員：松原克弥

Live Migration of Containers among Heterogeneous OS Platforms

Yuhei TAKAGAWA

概要： 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の互換性がない．本研究では，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の検討と実装を行う．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での実現を目的とする．

キーワード： コンテナ型仮想化, ライブマイグレーション, Linux, FreeBSD

Abstract: 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. This research 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. This study clarifies 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, especially for Linux and FreeBSD as the 1st target.

Keywords: the container-based virtualization, live migration, Linux, FreeBSD

1 背景と目的

クラウドコンピューティングを支える技術として，仮想化技術がある．特に，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・ポータビリティによる簡易的な環境構築を目的としたコンテナ型仮想化が注目されている [1]．コンテナ型仮想化は，OS が提供する資源を分離・制限し，単一動作の OS 上に複数の独立した実行環境を構築できる軽量な仮想化を実現する．

また，クラウドコンピューティング実動環境では，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は，Linux では CRIU(Checkpoint and Restore in Userspace)[2, 3]，FreeBSD では FreeBSD VPS(Virtual Private System)[5] が実装として存在する．OS 依存度が大きいコンテナ型仮想化では，OS 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．同一 OS 間のライブマイグレーションでは，システム全体の計算資源を活用することができず，メンテナンス時の可用性が損なわれてしまう．

本研究では，稼働 OS に依存したシステム運用で起きる前述の問題に対処するために，OS 混在環境におけるコンテナ型仮想化実行環境のライブマイグレーションの実現を目的とする．

2 関連技術

2.1 コンテナ型仮想化

コンテナ型仮想化は，アプリケーションが使用する計算資源やアクセス可能な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実行環境を構築できる技術である．コンテナとは分離されたアプリケーション実行環境のことを指す．コンテナの実現

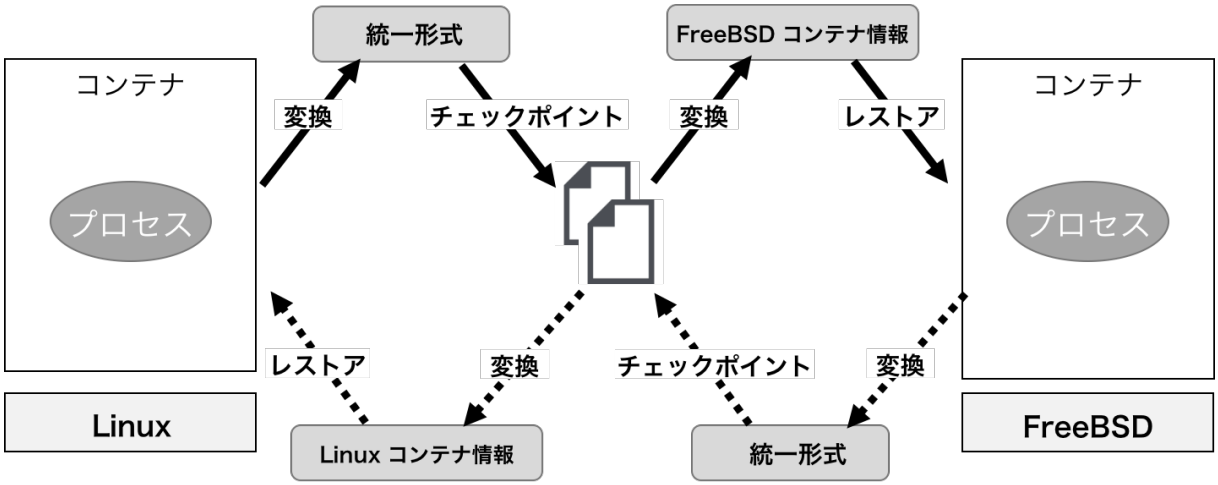

図 1 提案システムの概要

には，Linux では cgroups や Namespace，FreeBSD では Jail という OS 機能が利用されている [4].

cgroups は，Linux のプロセスが使用する資源を制限できる機能である．例えば，CPU 使用率やメモリ利用量などを制限可能である．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 Linux における Namespace にあたる機能で，ディレクトリやネットワークなどの識別子を分離する機能である．分離した空間は他の空間には直接関係しないため，同じ識別子を分離した空間ごとに利用することができる．

2.2 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，負荷分散や可用性の実現を目的に用いられる．ライブマイグレーションには 3 つの手順がある．最初に，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する．次に保存したファイルを転送する．最後に，転送されたファイルから H/W 資源の状態を復元する．H/W 資源の状態を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.

Linux でコンテナのライブマイグレーションを実現可能な CRIU，FreeBSD でライブマイグレーションを実現した FreeBSD VPS について以下に述べる.

2.3 CRIU

CRIU とは，Linux 上でコンテナのマイグレーションを可能にする OSS 実装である．CRIU の機能は，主にプロセス実行・隔離・制限状態のチェックポイントを行うこととプロセス実行・隔離・制限状態のレストアを行うことである．ライブマイグレーションは状態が移動するだけなので，マイグレーション先のマシンには同じコンテナが存在する必要がある．CPU の復元では，取得したレジスタの値を ptrace システムコールの機能を使って設定することで CPU の状態を復元している．なお，チェックポイント/レストアに必要なユーザ空間では行えない処理のみを Linux カーネルに組み込んでおり，多くの機能をユーザ空間で実現しているという特徴がある.

2.4 FreeBSD VPS

FreeBSD VPS とは，FreeBSD Jail のマイグレーションを可能にしたシステムであり，VPS インスタンスというコンテナを作成する．CRIU はプロセスをマイグレーションしているが，FreeBSD VPS は VPS インスタンスごとマイグレーションしている．CPU の復元には PCB(Process Control Block) を活用しているため，CRIU と全く異なる方法でプロセスの復元を行っている．なお，CRIU とは異なり，チェックポイント/レストアに必要な全ての機能をカーネル内で実現している.

3 異種 OS 間ライブマイグレーション実現方式

本章では，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提案する．本研究では，Linux と FreeBSD を対象とする．図 1 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．本提案では，プロセス実行・隔離状態などの情報を統一形式に変換してチェックポイント時にファイルに保存し，レストア時に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プロセス実行・隔離状態を復元する．本研究では，統一形式を Linux で CRIU を用いて復元できる形式とする．

コンテナ型仮想化実行環境のマイグレーションのためには，プロセス実行状態のマイグレーション (プロセス・マイグレーション) とプロセス隔離状態のマイグレーション (コンテナ・マイグレーション) が必要になる．それぞれにおける技術的課題と解決方法を以下に述べる．

3.1 プロセス・マイグレーションの実現

プロセス・マイグレーションを実現するには，以下の 2 点の課題を解決する必要がある．

3.1.1 システムコールの差異

Linux と FreeBSD の共通なシステムコールの大半は同じ動作である．しかし，システムコール番号やシステムコールの引数・パラメータの値が異なる．そのため，Linux バイナリは FreeBSD 上で動作せず，FreeBSD バイナリは Linux 上で動作しない．

また，システムコール引数の渡し方が Linux と FreeBSD では異なる．FreeBSD/x86 はシステムコール引数をスタック経由で渡すのに対して，Linux/x86 はシステムコール引数をレジスタ経由で渡す (表 1 参照)[7]．Linux/x86 では，システムコールの引数の上限が 5 つに決まっており，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，Linux と FreeBSD ではメモリやレジスタの状態が異なる．

システムコールの差異に関しては，システムコールの変換を行うことで解決する．システムコール番号や引数・パラメータ値に関しては，それぞれを変換することで 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．システムコール引数の渡し方に関しては，FreeBSD 上で Linux バイナリ実行時には引数をレジスタに入れ，Linux 上で FreeBSD バイナリ実行時には引数をスタックに積むというような実装を行うと，異なる OS でもレジスタやメモリの状態を同じにすることができる．

FreeBSD には，Linux バイナリ互換機能 [6] というカーネル機能があり，システムコール番号や引数・パラメータの変換・引数の渡し方の互換を行っている．この機能によって，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，システムコールの差異の吸収を Linux バイナリ互換機能を用いることで実現する．

表 1 x86 における Linux と FreeBSD のシステムコール引数格納場所

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	×	

3.1.2 メモリレイアウトの差異

メモリレイアウトは仮想メモリ空間における領域の配置である．一般的に ASLR(Address Space Layout Randomization) によって，領域の配置アドレスはランダムに割り当てられる．また，ASLR を無効化して領域の配置アドレスを固定した場合でも，Linux と FreeBSD ではスタック領域に 0x1000 番地の誤差がある．この誤差を純粋に修正する場合には，スタック内にある全てのベースアドレス，リターンアドレスなどを見つけ出し 0x1000 番地移動する必要がある．

メモリレイアウトの差異に関しては，メモリレイアウト変更機能を実装することで解決する．Linux では，prctl システムコール機能を利用することでメモリレイアウトをユーザプログラムから変更することができる．そのため，チェックポイントのメモリマップを変更せずに，0x1000 番地の誤差をなくすことができる．また，ASLR のランダムなメモリマップも再現可能である．しかし，FreeBSD にはメモリレイアウトをユーザプログラムから変更できる機能はない．Linux と FreeBSD との相互マイグレーションや ASLR に対応してマイグレーションを実現するには，FreeBSD 上でメモリレイアウトを変更できるシステムコールを作成する必要がある．

3.2 コンテナ・マイグレーションの実現

第 2.1 章で述べたように，FreeBSD と Linux ではコンテナを実現する機能が異なる．隔離・制限する対象や名称，値が異なるため，チェックポイントの状態をそのまま復元することはできない．

隔離・制限機能の差異に関しては，Linux cgroup，Namespace と FreeBSD Jail との機能の対応付けや数値の変換を行い，それぞれの機能で隔離・制限状態を再現することで解決する．資源の制限に関しては，Linux cgroups と FreeBSD RCTL の制限状態の対応付けと相互変換 (表 2 参照) を行うことで再現す

表 2 プロセス制限機能対応

制限機能	Linux	FreeBSD
CPU リソース	cgroups	RCTL+cpuset(1)
メモリリソース		RCTL
ファイル入出力		
デバイスアクセス		devfs(8)
一時停止/再開		△ SIGSTOP

表 3 プロセス隔離機能対応

隔離機能	Linux	FreeBSD
プロセス間通信	Namespace	Jail
マウントポイント		
ユーザ ID		
ホスト名		
ネットワーク		
プロセス ID		△ Jail

る．例えば，CPU 使用率を 50%に制限するのであれば，Linux 上では cgroups に以下のように設定する．

```
cpu.cfs_quota=50
cpu.cfs_period_us=100
```

FreeBSD 上では，RCTL に以下を設定することで同様の制限を実現できる．

```
jail:<jail id>:pcpu:deny=50
```

プロセスがアクセス可能な資源の隔離は，プロセス ID 空間の隔離を除いて，Linux Namespace と FreeBSD Jail で同様の実現が可能である (表 3 参照)．プロセス ID 空間の隔離機能の違いは，すべてのコンテナに含まれるプロセスに対して，システム内で重複しない一意な ID を付与することで対処する．

4 実装状況

FreeBSD で FreeBSD VPS を用いない単純なプロセス・マイグレーションを実現した．ptrace システムコールを用いてレジスタの状態を取得し，procfs の mem からプロセスのユーザ空間のメモリを取得する．復元時には，プロセスをフォークしプログラムを実行開始前で停止させ，レジスタやメモリの状態を書き込むことで，計算処理の続きからプロセスを再開することができる．ただし，プロセスを動作させるために最低限必要な情報しかないため，ネットワークやファイル書き込みの状態を復元することはできない．また，FreeBSD から Linux へ単純なプロセス・マイグレーションを実装している．Linux バイナリを対象として，Linux バイナリ互換機能を活用することでシステムコールの差異を変換し，Linux の prctl システムコールを利用することでメモリレイアウトの差異を吸収することで実現する．

5 おわりに

本研究は，異種 OS 混在環境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実現を目的としている．対象 OS を Linux と FreeBSD とし，プロセス・マイグレーションとコンテナ・マイグレーションの検討を行った．プロセス・マイグレーションでは，システムコールの差異を Linux バイナリ互換機能を用いて解決し，メモリレイアウトの差異を prctl システムコール機能を用いることで解決する．コンテナ・マイグレーションでは，制限機能は cgroup と RCTL との対応付けと数値の変換，隔離機能は Namespace と Jail の対応付けと変換を行い，チェックポイント時の隔離・制限状態を各種 OS 機能で再現する．

今後の課題として，ネットワークソケットやファイルポインタなどのカーネル等プロセス外で管理されているプロセス関連情報を対象プロセスのチェックポイント・レストアに含める技術の実現がある．これらのプロセス関連情報には，ネットワーク接続状態やファイル読み書き位置などの対象プログラムの実行状態の一部が含まれており，これらのチェックポイント・レストアを実現することで，実行中の任意の時点でのライブマイグレーションを可能にする．

参考文献

[1] 451 Research, ”451 Research: Application containers will be a $2.7bn market by 2020”, 2017.
[2] A. Mirkin, A. Kuznetsov and K. Kolyshkin: Containers checkpointing and live migration, In Proceedings of the 2008 Ottawa Linux Symposium, Vol. 2, 2008, pp. 8590.
[3] CRIU Project: CRIU Main page, https://criu.org/, 2010 (accessed November 1 2017).
[4] Docker Inc.: Docker overview, https://docs.docker.com/engine/docker-overview/, 2017 (accessed November 7, 2017).
[5] Klaus P. Ohrhallinger: Virtual Private System for FreeBSD, http://www.7he.at/freebsd/vps/, 2010 (accessed November 6, 2017). EuroBSDCon 2010.
[6] Jim Mock: Linux Binary Compatibility, https://www.freebsd.org/doc/handbook/linuxemu.html, 1999 (accessed November 2, 2017).
[7] 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.