

異種 OS 間でのコンテナ型仮想化のライブマイグレーションに関する研究

b1014223 高川雄平
指導教員：松原克弥

A Study on Container Live Migration 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は互換性がない。本研究では、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：コンテナ仮想化, ライブマイグレーション, FreeBSD, Linux

Abstract: 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 a uniform representation and they restore in both OSs, and how an uniformed resource isolation can be realized.

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

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・ポータビリティを目的として、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、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。

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[?, ?], FreeBSD では FreeBSD VPS(Virtual Private System)[?] 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上記のような問題を解決するために、異なる OS 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。

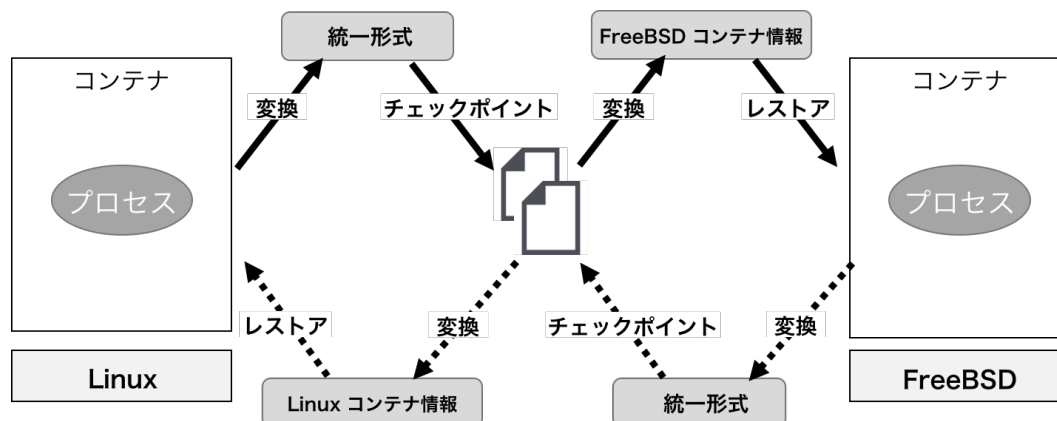


Fig. 1 本提案システムの概要

2 既存技術

2.1 コンテナ型仮想化

コンテナ型仮想化は、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 [?][?]。

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。

2.2 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 [?]。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。サービスを停止時間をゼロにすることはできず、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可能な 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。

2.3 CRIU

CRIU (Checkpoint/Restore in Userspace) は、Linux 上でコンテナ・マイグレーションを可能にする OSS である。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。CPU の復元では、取得したレジスタの値を ptrace でセットすることで復元している。また、Linux カーネルに最低限必要な機能がマージされているため、カーネルを変更する必要がない。

2.4 FreeBSD VPS

FreeBSD VPS (Virtual Private System) は、FreeBSD jail のライブマイグレーションを可能にしたシステムである。FreeBSD VPS は 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 はプロセスではなく 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おこなっている。

3 提案システムの実現方式

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 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、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。

コンテナ型仮想化実行環境のマイグレーションを行うためには、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。第 4, 5 章では、プロセス・マイグレーションとコンテナ・マイグレーションにおける技術的課題と解決方法を示す。

4 プロセス・マイグレーション

4.1 技術的課題

4.1.1 システムコールの差異

Linux と FreeBSD では共通なシステムコールは大半が同じ処理で動作している。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ため、通常であれば、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。

4.1.2 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 Linux と FreeBSD では ABI が異なる。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

Table 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	×	

コール引数をレジスタ経由で渡す(表 1 参照)[?]. 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、FreeBSD と Linux ではメモリやレジスタの状態が異なる。

4.1.3 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の領域の分け方のことである。一般的に ASLR(Address Space Layout Randomization) によって、大まかな配置箇所は同じでも細かい部分が異なる。また、ASLR を無効化した場合でも、FreeBSD と Linux ではスタック領域は 0x1000 ズレてしまう。ズレ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 ズらす必要がある。

4.2 実現手法

4.2.1 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[?]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。

4.2.2 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。

4.2.3 メモリレイアウトの変更機能

Linux カーネルには、システムコール `prctl` 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。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することができる機能はない。Linux と FreeBSD との相互マイグレーションを実現するには、FreeBSD 上で Linux の `prctl` をベースに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。また、`prctl` を実現することにより、ASLR にも対応することができる。

5 コンテナ・マイグレーション

5.1 技術的課題

5.1.1 隔離機能の差異

第 2.1 章で述べたように、FreeBSD と Linux ではコンテナ型仮想化を実現する機能が異なる。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、プロセス実行状態の隔離をそのままの状態では復元することはできない。

5.2 実現手法

5.2.1 隔離機能の差異

FreeBSD の `Jail` と Linux の `cgroup`、`Namespace` 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` の隔離状態の対応付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。

6 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。FreeBSD 上で `ptrace()` を用いることで、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 へ単縦なプロセス・マイグレーションも実現した。第 3 章で述べたように、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と ABI の差異を変換し、Linux の `prctl()` 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した。

7 おわりに

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提案を行った。対象 OS を Linux と FreeBSD とし、異種 OS 間でのプロセス・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、システムコールや ABI の差異を Linux バイナリ互換機能を用い、メモリレイアウトの差異を `prctl()` のような機能を FreeBSD カーネルにも実装することで、それぞれの差異を吸収する。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。

参考文献

- [1] 451 Research, "451 Research: Application containers will be a \$2.7bn market by 2020", 2017.
- [2] A.B.Cddddd, J. Systems Information Science, 11, 1145-1159, 2001.
- [3] ○○××, □□△△, システム情報科学, ☆出版, 1999, 20-21.
- [4] E.Fggg and H.Ijjj, Electrical Engineering, KKPress, 2003, 281-284.