Vol.2022-OS-156 No.13
2022/7/28

VM外で実行可能なコンテナの状態保存機構

朝倉 優輝[1]　光来 健一[1]

概要：コンテナは負荷分散などのためにマイグレーションよって他のホストへ移動させることができる．しかし，ホストの負荷が高い場合にはマイグレーション性能が低下し，マイグレーション処理がコンテナ性能にも大きな影響を与える．また，クラウドではコンテナを仮想マシン（VM）内で動作させることが多いため，マイグレーション性能は仮想化の影響も受ける．本稿では，VM 内で動作しているコンテナのマイグレーションを VM 外において実行可能にするシステム OVmigrate を提案する．OVmigrate では，VM 内のコンテナの状態を VM 外で保存し，その状態を移送先ホストに転送する．移送先ホストでは VM 内に新たなコンテナを作成し，移送元のコンテナの状態を復元する．OVmigrate を利用することで VM による仮想化や VM 内の負荷の影響を受けることなく迅速にコンテナマイグレーションが可能となる．OVmigrate におけるコンテナの状態保存機構を VM イントロスペクションを用いて実装し，既存のマイグレーションツールとの性能比較を行った．また，仮想化によるコンテナマイグレーションへの影響についても調べた．

1. はじめに

近年，コンテナを提供するクラウドサービスが増加しており，AI や IoT を支えるビッグデータを扱うためなどに利用されている．コンテナは OS によって提供される軽量な仮想環境であり，いくつかのプロセスとその実行環境によって構成されている．コンテナはマイグレーションと呼ばれる技術を用いて実行を継続したまま別のホストへ移動させることができる．例えば，負荷が高いホストのいくつかのコンテナを別のホストに移動させることで，負荷の分散を行うことができる．

コンテナマイグレーションは負荷分散を行う場合のように，ホストの負荷が高い時に行わざるを得ないことも多い．このような場合には，コンテナマイグレーションがホストの負荷の影響を大きく受け，マイグレーション性能が大きく低下する可能性がある．コンテナマイグレーション自体の負荷も高いため，コンテナマイグレーション処理がコンテナの性能に大きな影響を与えることもある．また，クラウドでは管理の柔軟性を高めるためにコンテナを仮想マシン（VM）内で実行することも多い．そのため，VM 内で実行されるコンテナマイグレーションの性能は VM による仮想化の影響も大きく受ける．

本稿では，VM 内で実行されているコンテナに対して VM 外でマイグレーションを実行可能にするシステム OVmigrate を提案する．OVmigrate は VM イントロスペクション [1] を用いて VM のメモリに格納されているコンテナの状態を VM 外から取得する．保存したコンテナの状態を移送先のホストへ転送し，そのホストの VM 内に移送元のコンテナの状態を復元する．OVmigrate を用いることで，VM 内の負荷や仮想化による影響を受けず，迅速にコンテナマイグレーションを行うことができる．また，コンテナマイグレーションの処理が VM 内の他のコンテナに影響を与えるのを防ぐこともできる．

我々は OVmigrate の状態保存機構を KVMonitor [2] および LLView [1] を用いて実装した．OVmigrate は KVMonitor を用いて共有した VM のメモリから OS データを取得する．また，LLView を用いて OS のソースコードを利用してメモリ上の OS データを解析する．コンテナの状態は主にプロセスの状態によって構成されており，現在のところ，OVmigrate はファイルシステム，プロセス木，ページマップ，ページデータの情報が保存できている．実験により，OVmigrate で保存した状態を用いて CRIU [3] で正常にプロセスの復元処理が実行できることを確認した．また，CRIU と比較して OVmigrate では 10 倍高速にプロセスの状態が保存でき，ばらつきも抑えられることが分かった．

以下，2 章でコンテナマイグレーションの問題点について述べる．3 章では VM 外でコンテナマイグレーションを実行する OVmigrate を提案し，4 章でその実装について述べる．5 章では OVmigrate を用いてプロセスの状態を保存する性能，および，仮想化によるマイグレーション性能への影響について調査した実験について述べる．6 章で関連

[1] 九州工業大学
Kyushu Institute of Technology

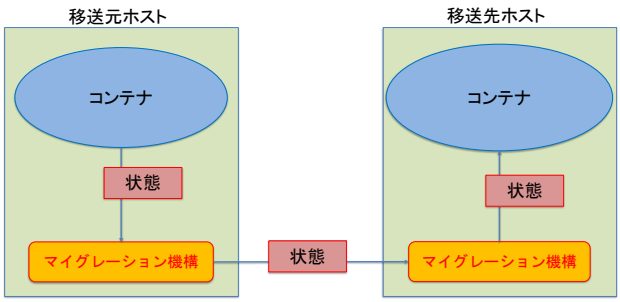

図 1 コンテナマイグレーション

研究に触れ，7 章で本稿をまとめる．

2. コンテナマイグレーション

コンテナ型仮想化は VM を用いた従来の仮想化とは異なり，コンテナと呼ばれる仮想環境を作成する．コンテナはハードウェアの仮想化を行わず，ホストの OS カーネルを利用する．コンテナの中ではいくつかのプロセスが実行され，ファイルシステムやネットワークインターフェースなどの実行環境が提供される．このように，コンテナは少ないリソースで構成されるため，高速に起動や実行を行うことができる．最近は Amazon ECS/EKS，Google GKE，Microsoft AKS などのクラウドでもコンテナが提供されるようになっている．

コンテナはマイグレーションと呼ばれる技術を用いて，VM と同じように別のホストへ移動させることができる．コンテナマイグレーションを実行する流れを**図 1** に示す．まず，移送元ホスト上でコンテナ内部のプロセスの状態などをコンテナの状態として保存する．次に，保存されたコンテナの状態を移送先ホストへ転送し，移送先ホスト上で受信したコンテナの状態から元のコンテナを復元する．コンテナマイグレーションを用いることで，例えば，負荷の高いコンテナを別のホストに移動させることにより，ホスト間で負荷を分散することが可能となる．また，ホストのメンテナンスを行う際にコンテナを別のホストに移動することにより，コンテナを停止させずにホストの保守や再起動などを行うこともできる．

クラウドでは，1 台のホストに多くのコンテナを集約させているため，ホストの負荷が高い時にコンテナをマイグレーションせざるを得ないことも多い．例えば，負荷分散は一般的に，ホストの負荷が高くなったことを検出した時に行われる．移送先ホストもコンテナをマイグレーションしている間に負荷が高くなってしまうことも考えられる．このような場合には，コンテナマイグレーションがホストの負荷の影響を大きく受け，マイグレーション性能が大きく低下する可能性がある．また，コンテナマイグレーション自体の負荷も高いため，コンテナ性能への影響は避けられない．

VM に関しては，マイグレーションを用いた負荷分散について様々な研究が行われている．例えば Sandpiper [4] では，CPU やネットワークの使用率が 75%を超えた時に VM をマイグレーションすることを推奨している．この閾値は，VM マイグレーションによって負荷の高いサーバの性能が 50%低下し，ネットワーク性能が 20%低下することから，マイグレーションのオーバヘッドを吸収できるように決められている．しかし，ホストのリソース使用率が低下するため，クラウドのコスト上昇につながる．また，GCE ではホストの負荷が高い時には VM マイグレーションの速度を調節していると報告されている [5]．しかし，その場合には VM マイグレーションを迅速に行うことができないため，負荷分散などに時間がかかることになる．

一方，クラウドでは物理ホストよりも柔軟に管理を行えるようにするために，VM 内でコンテナを動作させていることが多い [6]．この場合には，VM 内で行われるコンテナマイグレーションの処理が VM による仮想化の影響を受ける．実際に，ネットワーク仮想化がオーバヘッドの主な原因であり，CPU 使用率が移送元 VM で 70%，移送先 VM で 118%になるという報告がある [7]．Portkey [7] ではコンテナマイグレーションのネットワークアクセスを高速化しているが，VM 内のマイグレーション機構やゲスト OS への変更が必要である．また，6 章で述べる実験によると，仮想ネットワーク以外のプロセス間通信や仮想ディスクへの書き込みでも仮想化による影響が見られている．

3. OVmigrate

本稿では，VM 内で動作しているコンテナのマイグレーションを VM 外において実行可能にするシステム OVmigrate を提案する．OVmigrate のシステム構成を**図 2** に示す．2 章で述べたように，従来のコンテナマイグレーションではコンテナが動作する VM 内でマイグレーション機構が実行されていた．それに対し，OVmigrate のマイグレーション機構はコンテナが動作する VM と同じホスト内の VM 外部で動作する．移送元ホストのマイグレーション機構は VM 内のコンテナの状態を保存し，移送先ホストへ転送する．移送先ホストのマイグレーション機構は VM 内に新しいコンテナを作成し，受信したデータを用いて移送元ホストのコンテナの状態を復元する．

OVmigrate は，コンテナの状態の保存・復元を VM の外部で実行するため，マイグレーション機構が VM による仮想化の影響を受けない．また，VM 外のマイグレーション機構は VM 内のシステムの負荷の影響も受けない．VM 内のシステムの負荷が高い場合でも，VM を動作させているホストにはリソースが余っていることが多いことから，OVmigrate はコンテナマイグレーションを迅速に実行できる．加えて，VM 内でコンテナマイグレーションに関する処理を行わないことから，VM 内で動作している他のコン

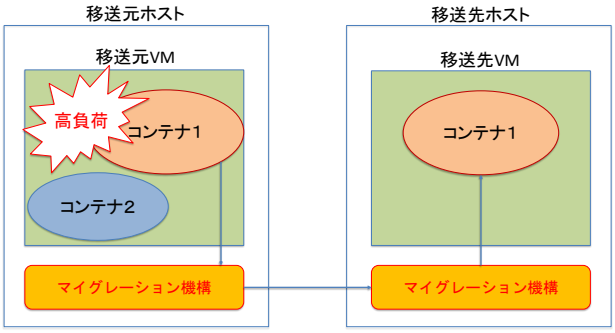

図 2 OVmigrate のシステム構成

テナの性能に影響を与えない．

コンテナはコンテナ内部で動作するプロセスとその実行環境で構成されており，コンテナの状態の大部分はプロセスの状態によって占められている．また，コンテナの実行環境に関する情報の多くもプロセスの状態に含まれている．そのため，OVmigrate はコンテナの状態を保存する際に主にプロセスの状態の保存を行う．プロセスの状態としては，プロセスに割り当てられている仮想メモリに関する情報やプロセス間の関係を表すプロセス木などがある．一方，プロセスの中のコンテナの実行環境に関する情報としては，cgroups やファイルシステムの情報などがある．

OVmigrate は VM 内のプロセスの状態を VM イントロスペクション [1] を用いて取得する．VM イントロスペクションは VM のメモリ上にある OS やプロセスのデータを解析することによって，VM 外から VM 内の情報を取得する手法である．OVmigate はプロセス ID を基にカーネルメモリ上のプロセス構造体を見つける．そして，プロセス構造体からポインタを辿っていくことにより，プロセスに関する様々な情報を取得する．VM イントロスペクションを用いることで，VM 内で情報取得のためのプロセスを動作させたり，VM 内の OS に変更を加えたりすることなくプロセスの状態保存を行うことができる．

OVmigrate は VM 内のコンテナのプロセスに疑似的にシグナルを送信 [8] [9] することにより，VM 外から VM 内のプロセスの制御を行う．このシグナル疑似送信は VM のカーネルメモリ上のプロセス情報を書き換えてシグナルが送られた状態に変更することによって実現する．プロセスの状態保存時には STOP シグナルを疑似送信することにより，プロセスを停止させることができる．また，プロセスの状態保存後には KILL シグナルを疑似送信することにより，プロセスを終了させることができる．

4. 実装

我々は QEMU-KVM 4.2.0 上で動作する VM 内のゲスト OS である Linux 5.4 に対して OVmigrate を実装した．現在のところ，OVmigrate が保存できるプロセスの状態はファイルシステム，プロセス木，ページマップ，ページデータの情報であり，他の状態については実装中である．

```
message pagemap_entry {
        uint64 vaddr    = 1;
        uint32 nr_pages = 2;
        uint32 flags    = 3;
}
```

図 3 .proto ファイルによるメッセージの定義の例

4.1 .proto ファイルに基づく状態保存

OVmigrate は CRIU 3.16 [3] によって保存されているものと同じプロセスの状態を保存する．CRIU はプロセスの保存・復元を行う既存のツールである．CRIU はプロトコルバッファ [10] を用いてプロセスの状態を保存しており，保存するプロセスの状態が.proto ファイルで定義されている．OVmigrate でもこの.proto ファイルを利用し，C 言語用のプロトコルバッファ [11] を用いてプロセスの状態を保存する．.proto ファイルでは**図 3** のように状態の種類ごとにメッセージが定義されており，その中のフィールドで個別の状態が定義されている．

4.2 VM イントロスペクション

OVmigrate は KVmonitor [2] を用いて VM イントロスペクションを行う．KVMonitor を用いた OVmigrate のシステム構成をに**図 4** に示す．まず，VM に割り当てる物理メモリをホスト上のファイルとして作成し，VM とマイグレーション機構の両方にマッピングをして共有する．次に，QEMU と通信して仮想 CPU の CR3 レジスタに格納されているページテーブルのアドレスを取得する．このページテーブルを用いて，プロセスの状態が格納されているカーネルデータの仮想アドレスを物理アドレスに変換し，VM 内の情報を取得する．

VM のメモリ上のカーネルデータの解析を容易にするために，OVmigrate は LLView フレームワーク [1] を用いる．Linux カーネルのヘッダファイルで定義されているカーネルの構造体やグローバル変数などを用いて，プロセスの状態を取得するプログラムを記述する．このプログラムをコンパイルして生成された LLVM の中間表現を変換することで，必要に応じて VM のメモリにアクセスするコードを埋め込む．

4.3 ファイルシステム情報の保存

ファイルシステムの情報として，カレントディレクトリおよびルートディレクトリに付けられる固有の ID と umask の値を保存する．umask には新規ファイルや新規ディレクトリの作成時に設定されるアクセス権が格納されている．ディレクトリに付けられる固有の ID は CRIU と同様にして連番で割り当てる．umask の値を取得するためにまず，プロセス ID を基に init_task 変数からプロセスリストをた

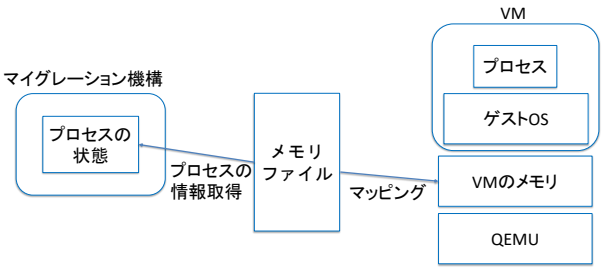

図 4 VM 外からのメモリの取得

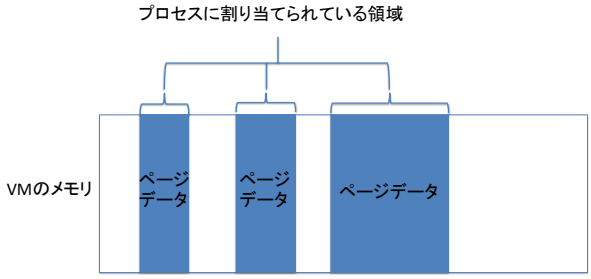

図 5 プロセスに割り当てられているメモリ領域

どり，task_struct 構造体を探索する．この構造体にはプロセスに関する情報が格納されている．次に，task_struct 構造体からポインタをたどって fs_struct 構造体を見つける．この構造体には inode やパスなどの情報が保存されている．そして，この構造体のメンバ umask に格納されている値を取得する．

4.4 プロセス木の情報の保存

プロセス木の情報として，対象プロセスの ID，そのプロセスの親プロセスの ID，グループ ID，セッション ID，対象プロセスが持つスレッド数，各スレッドのスレッド ID を保存する．まず，task_struct 構造体からポインタをたどって signal_struct 構造体を見つける．この構造体にはシグナルに関連して使われるプロセスの情報が格納されている．次に，signal_struct 構造体からポインタをたどって pid 構造体を見つける．この構造体にはプロセス ID，グループ ID，セッション ID が格納されているが，それぞれの ID は階層化されたプロセスの名前空間ごとに割り当てられている．そのため，名前空間の階層を考慮して ID が格納された upid 構造体を見つける．また，現在のところ，1 つのスレッドのみを持つプロセスを想定し，プロセス ID と同じ値をスレッド ID として保存している．

4.5 ページマップの情報の保存

ページマップの情報として，**図 5** のようにプロセスに割り当てられているメモリページの開始アドレスと連続するページ数の組を順番に保存する．まず，task_struct 構造体からポインタをたどって mm_struct 構造体のリストを見つける．この構造体にはプロセスのメモリの仮想アドレスやページに関する情報が格納されている．vma_area_struct 構造体のリストをたどり，対応するメモリ領域の先頭ページの仮想アドレスとページ数を取得する．メモリ領域のすべてのページがプロセスに割り当てられているわけではないため，プロセスのページテーブルを調べることで，メモリ領域に含まれるすべてのページについて存在の有無を調べる．ページが存在し，かつ，連続している領域を 1024 ページ以下になるように分割して保存する．ファイルがマッピングされているメモリ領域については，匿名メモリページでない場合は対象としない．

4.6 ページデータの情報の保存

ページデータの情報として，プロセスに割り当てられており，かつファイルのデータではないページの内容を保存する．ページマップの情報の保存処理において存在することが分かったページについて，そのページの 4KB のデータを取得する．CRIU ではページデータの保存にプロトコルバッファを用いていないため，CRIU に合わせて直接ファイルに保存する．

5. 実験

OVmigrate を用いて VM 内のプロセスの状態を正常に保存できることを確認する実験を行った．また，メモリを多く使用するプロセスの状態を保存するのにかかる時間を測定した．比較として，VM 内で CRIU を用いてプロセスの状態を保存するのにかかる時間も測定した．ホストとして，Intel Core i7-10700 の CPU，64GB のメモリ，2TB の SATA HDD を搭載したマシンを用いた．このホストでは OS として Linux 5.4，仮想化ソフトウェアとして QEMU-KVM 4.2.0 を動作させた．VM には仮想 CPU を 2 個，メモリを 30GB 割り当て，ゲスト OS として Linux 5.4 を動作させた．

5.1 動作確認

VM 内で 1 秒ごとにカウンタ値を増加させながら表示するプログラムを実行し，OVmigrate を用いてそのプロセスの状態を保存した．その際にプロセスを終了させないようにし，プロセスに実行を継続させた．次に CRIU を用いて同じプロセスの状態の保存を行い，保存後にプロセスを終了させた．CRIU によって状態が保存されたファイルの内，ファイルシステム，プロセス木，ページマップ，ページデータの情報に関するファイルを OVmigrate を用いて保存したファイルに差し替えた．その後，CRIU を用いてプロセスを復元した．

CRIU によるプロセスの保存・復元時のログをに**図 6**,

```
a = 214
a = 215
a = 216
a = 217
Killed
```

図 6　CRIU による保存処理のログ

```
(00.031538) Running post-resume scripts
a = 105
a = 106
a = 107
a = 108
```

図 7　ファイル差し替え後の CRIU による復元処理のログ

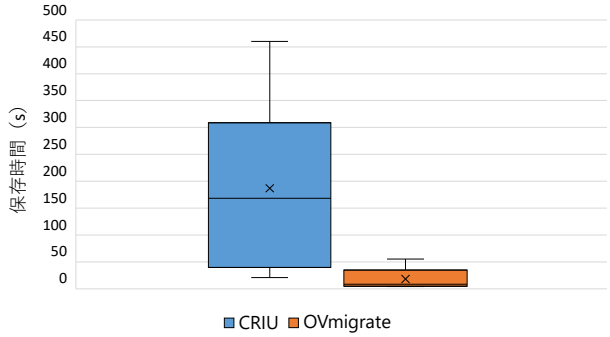

図 8　メモリ情報の保存時間の比較

図 7 示す．OVmigrate によって保存されたファイルに差し替えてもプロセスの復元処理が正しく実行され，プロセスはカウントを再開した．カウンタの値がプロセスの終了時点と開始時点で異なるのは，OVmigrate がプロセスの状態を保存した時点から正しく再開されたためである．

5.2　状態保存の性能

VM 内で 5GB のメモリを使用するプログラムを実行し，そのプロセスの状態を保存する時間を測定した．OVmigrate ではページマップとページデータの情報を保存する処理のみを実行し，CRIU ではプロセスのすべての状態を保存してページマップとページデータの情報の保存にかかった時間を測定した．測定結果は図 8 のようになり，OVmigrate は CRIU より平均で 10 倍高速にプロセスのメモリ情報を保存できることが分かった．また，CRIU に比べて保存時間のばらつきも抑えられていることが確認できた．

5.3　仮想化の影響の分析

CRIU に対する仮想化の影響を調べるために，5GB のメモリを使用するプロセスのメモリ情報を CRIU で取得するのにかかる時間をホスト上と VM 内のそれぞれで測定した．測定結果は図 9 のようになり，ホスト上ではメモリ情報を高速に保存することができ，ばらつきも小さいことが

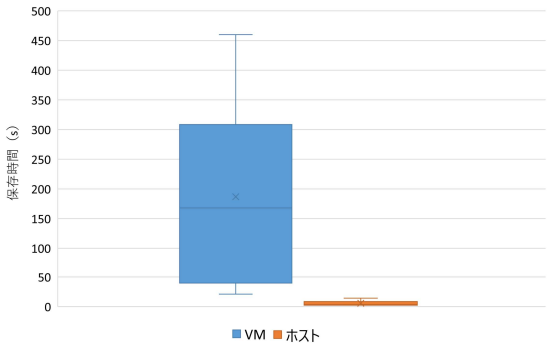

図 9　CRIU によるメモリ情報の保存時間の比較

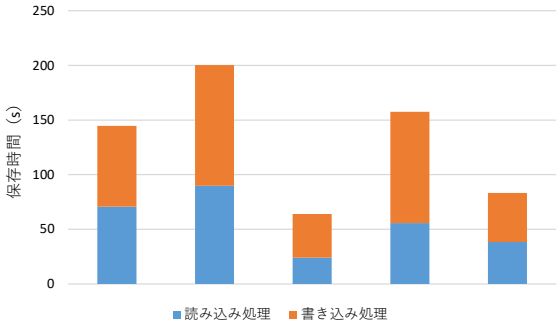

図 10　CRIU におけるメモリ情報の保存時間の内訳

分かった．

次に，VM 内の CRIU によるメモリ情報の保存においてどの処理に時間がかかっているかを調べた．CRIU によって対象プロセスに送り込まれたパラサイトコードが取得・送信したメモリ情報をパイプから読み込む時間とそれをディスクに書き込む時間を測定した．測定結果は図 10 のようになり，どちらにも長い時間がかかっているが，書き込みにより長い時間がかかっていること分かった．これは仮想ディスクへの書き込み処理が仮想化の影響を大きく受けたためだと考えられる．

そこで，メモリ情報をディスクに書き込まないようにした場合にメモリ情報の保存処理にかかる時間を測定した．測定結果は図 11 となり，ディスクへの書き込み処理とは関係なく読み込み処理に長い時間がかかることが分かった．保存処理のほとんどはパラサイトコードからのメモリ情報の読み込み時間であったことから，プロセス間通信でも仮想化の影響を大きく受けていることが考えられる．

6.　関連研究

Portkey [7] はネットワーク転送を最適化することにより，VM 内のコンテナを効率よくマイグレーションするこ

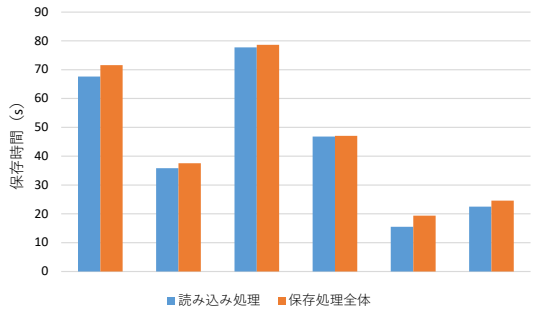

図 11 CRIU においてディスクへの書き込みを行わない場合の保存時間

とを可能にしている．CRIU が保存したコンテナの状態を転送する際には Portkey のカーネルモジュールを呼び出し，カーネルモジュールはゲスト OS でのネットワーク処理をバイパスしてハイパーバイザを呼び出す．ハイパーバイザが移送先ホストへコンテナの状態を転送し，CRIU はカーネルモジュール経由でハイパーバイザから状態を受け取る．ハイパーバイザで転送処理を行うことでコンテナマイグレーション中の CPU 使用率を抑えることはできているが，マイグレーションは高速化されていない．OVmigrate はコンテナの状態を完全に VM 外で転送するため，さらに CPU 使用率を削減し，マイグレーションの高速化も実現することができると考えられる．

mWarp [12] は同一ホストの VM 間でメモリを再配置することにより，コンテナマイグレーションにおいて時間がかかるプロセスメモリのコピーや転送を行わないようにする．mWarp では，移送元 VM 内の CRIU はシステムコール経由でハイパーコールを呼び出し，コンテナ内のプロセスのメモリ情報をハイパーバイザに通知する．移送先 VM 内の CRIU がハイパーバイザを呼び出すと，ハイパーバイザが移送元 VM のメモリを移送先 VM へ再マッピングし，転送を完了させる．しかし，mWarp では同じホスト内の VM 間でのみコンテナマイグレーションが実行可能である．OVmigrate でも移送元と移送先の VM が同一ホスト上にあれば，同様の最適化が可能であると考えられる．

VMBeam [13] はホスト VM 内で動作しているゲスト VM を対象として，mWarp と同様にゼロコピーでのマイグレーションを可能としている．VMbeam では，移送元と移送先のホスト VM 内のマイグレーション機構がハイパーバイザを呼び出し，移送元ホスト VM のメモリを移送先ホスト VM とスワップすることでゲスト VM のメモリ転送を高速に実行する．VMBeam も同一ホスト内でのみ利用可能な最適化である．

Sledge [14] は Docker コンテナの効率のよいライブマイグレーションを実現している．Sledge は移送元ホストのコンテナが使っている階層的なイメージの中の冗長なレイヤを転送しない．また，CRIU のインクリメンタル・チェックポイントを利用してメモリの差分だけを繰り返し転送する．さらに，移送先ホストで時間のかかる Docker デーモンの再ロードを行わず，管理コンテキストのロードだけを行う．これらの最適化技術は OVmigrate でも利用できる可能性がある．

7. まとめ

本稿では，VM 内で動作しているコンテナのマイグレーションを VM 外において実行可能にするシステム OVmigrate を提案した．OVmigrate を用いることで，VM による仮想化や VM 内の負荷がコンテナマイグレーションに及ぼす影響および，コンテナマイグレーションがコンテナ性能に及ぼす影響を小さくすることができる．OVmigrate は VM イントロスペクションを用いて VM のメモリを解析することで，VM 内のコンテナの状態を VM 外で保存する．実験により，ファイルシステム，プロセス木，ページマップ，ページデータの情報を正しく保存できていることが確認できた．また，OVmigrate は VM 内で CRIU を用いる場合より 10 倍高速にメモリ情報を保存することができ，保存時間のばらつきも抑えられることが分かった．

今後の課題は，コンテナマイグレーションに必要なすべての状態を保存できるようにすることである．保存する必要がある状態としては，様々なプロセスの状態およびコンテナエンジンが管理しているコンテナの状態がある．次に，保存した状態を用いてコンテナの復元も行えるようにする．その際に，新しいコンテナやプロセスを VM 外から作成することも課題である．

謝辞 本研究の一部は，JST，CREST，JPMJCR21M4 の支援を受けたものである．

参考文献

[1] Ozaki, Y., Kanamoto, S., Yamamoto, H. and Kourai, K.: Detecting System Failures with GPUs and LLVM, *Proceedings of the 10th ACM SIGOPS Asia-Pacific Workshop on Systems*, pp. 47–53 (2019).
[2] Nakamura, K. and Kourai, K.: Efficient VM Introspection in KVM and Performance Comparison with Xen, *IEEE 20th Pacific Rim International Symposium on Dependable Computing*, pp. 192–202 (2014).
[3] Team, O.: CRIU, `https://criu.org/Main_Page`.
[4] Wood, T., Shenoy, P., Venkataramani, A. and Yousif, M.: Black-box and Gray-box Strategies for Virtual Machine Migration, *4th USENIX Symposium on Networked Systems Design & Implementation* (2007).
[5] Ruprecht, A., Jones, D., Shiraev, D., Harmon, G., Spivak, M., Krebs, M., Baker-Harvey, M. and Sanderson, T.: VM Live Migration At Scale, *Proceedings of the 14th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pp. 45–56 (2018).
[6] VMware, Inc.: VMware tanzu, `https://tanzu.vmware.com/application-platform`.

[7] Prakash, C., Mishra, D., Kulkarni, P. and Bellur, U.: Portkey: Hypervisor-Assisted Container Migration in Nested Cloud Environments, *Proceedings of the 18th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pp. 3–17 (2022).
[8] Kimura, K. and Kourai, K.: GPU-based First Aid for System Faults, *In Proceedings of the 13th ACM Asia-Pacific Workshop on Systems* (2022).
[9] 木村健人，光来健一：システム外部からの OS メモリの書き換えによるシステム障害からの復旧，第 33 回コンピュータシステム・シンポジウム (2021).
[10] Google: Protocol buffers, `https://developers.google.com/protocol-buffers`.
[11] Benson, D.: Protocol Buffers implementation in C, `https://github.com/protobuf-c/protobuf-c`.
[12] Sinha, P., Doddamani, S., Lu, H. and Gopalan, K.: mWarp: Accelerating Intra-Host Live Container Migration via Memory Warping, *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops* (2019).
[13] Ooba, H. and Kourai, K.: Zero-copy Migration for Lightweight Software Rejuvenation of Virtualized Systems, *Proceedings of the 6th Asia-Pacific Workshop on Systems* (2015).
[14] Xu, B., Wu, S., Xiao, J., Jin, H., Shi, G., Rao, J., Yi, L. and Jiang, J.: Sledge : Towards Efficient Live Migration of Docker Containers, *IEEE International Conference on Cloud Computing*, pp. 321–328 (2020).