Vol.2023-OS-160 No.3
2023/8/3

コンテナマイグレーションを VM外で実行するための状態復元機構

木本 翔太[1]　光来 健一[1]

概要：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く，コンテナをマイグレーションする際に VM の負荷や仮想化の影響を受け，性能が低下してしまう．先行研究において VM 内のコンテナを VM 外からマイグレーション可能にするシステムが提案されているが，移送元で VM 外からコンテナの状態を保存することによる性能改善に焦点を当てている．移送先でもコンテナは VM 内で動作し，時間のかかるマイグレーション中に VM の負荷が高くなることもあるため，状態復元についても性能改善が必要である．本稿では，VM 外から VM 内にコンテナの状態を復元可能にするシステム OVrestorer を提案する．OVrestorer は VM のメモリ上にあるコンテナの状態を VM 外から書き換えることで，コンテナの状態復元を行う．これにより，VM の負荷や仮想化のオーバヘッドがコンテナの復元性能に与える影響を最小化することができる．コンテナのすべての状態を VM 外から復元するのは容易ではないため，OVrestorer は VM 内のコンテナ復元支援機構と協調して動作する．OVrestorer を用いてコンテナ内で動作するプロセスの復元性能を調べた．

1. はじめに

近年，Amazon ECS [1] や EKS [2]，Google GKE [3]，Microsoft AKS [4] など，コンテナを提供するクラウドサービスが普及してきている．コンテナは OS のプロセスとその実行環境によって構成された軽量な仮想環境である．仮想マシン（VM）と同様に，コンテナもマイグレーションにより別のホストに移動させることが可能である．その際に，ホストの負荷が高い場合にはマイグレーション性能が負荷の影響を受ける．逆に，マイグレーションの負荷がホスト内のコンテナの性能にも影響を与える．また，クラウドではコンテナを VM 内で動作させていることが多く，コンテナのマイグレーション性能は仮想化の影響も受ける [5]．

そこで，VM 内のコンテナを VM 外からマイグレーション可能にする OVmigrate [6] が提案されている．OVmigrate は VM 内で動作していたマイグレーション機構を VM 外で動作させる．それにより，コンテナマイグレーションが VM の負荷や仮想化の影響を受けないようにすることができる．しかし，OVmigrate は移送元で VM 外からコンテナの状態を保存することによる性能改善に焦点を当てており，移送先での状態復元については VM 内で実行している．移送先でもコンテナは VM 内で動作し，時間のかかるマイグレーション中に VM の負荷が高くなることもあるため，状態復元についても性能改善が必要である．

本稿では，VM 外から VM 内にコンテナの状態を復元可能にするシステム OVrestorer を提案する．OVrestorer は移送先の VM 内にあるコンテナの状態を VM 外から書き換えることで，コンテナの状態復元を行う．そのために，状態復元時には VM 内に空のコンテナを作成し，VM のメモリ上にあるそのコンテナの状態を移送元で保存されたコンテナの状態で上書きする．これにより，VM の負荷が高い場合でもその負荷がコンテナの状態復元に及ぼす影響を軽減することができる．また，VM による仮想化のオーバヘッドがコンテナの復元処理に与える影響を最小化することもできる．

コンテナのすべての状態を VM 外から復元するのは容易ではないため，OVrestorer は VM 外で動作するコンテナ復元機構と VM 内で動作するコンテナ復元支援機構を協調させてコンテナの状態復元を行う．現在のところ，VM 外のコンテナ復元機構はコンテナ内で動作するプロセスのファイルシステム情報とメモリを復元することができており，それ以外の状態は VM 内のコンテナ復元支援機構が復元している．コンテナ復元機構が VM のメモリ上にあるコンテナの状態を書き換えられるようにするために，KVMonitor [7] および LLView [8] を拡張して用いた．実験により，OVrestorer を用いてプロセスを保存時の状態に

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

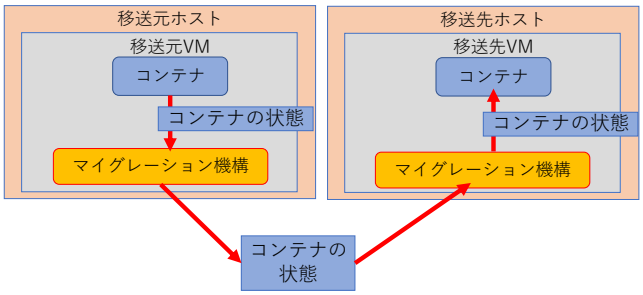

図 1 コンテナマイグレーション

復元できることが確認できた．また，OVrestorer は仮想化の影響を受けず，VM の負荷が高い時でも高速にプロセスのメモリを復元できることが分かった．

以下，2 章でコンテナマイグレーションの性能低下について述べる．3 章では VM 外からコンテナの復元を実行する OVrestorer を提案し，4 章でその実装について述べる．5 章では OVrestorer を用いてプロセスの状態を復元する性能，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の性能低下

コンテナはマイグレーション技術を用いることで，VM と同様に別のホストに移動することが可能である．コンテナマイグレーションの流れを図 1 に示す．まず，移送元ホストでコンテナの状態を保存する．次に，保存したコンテナの状態を移送先ホストに転送する．そして，転送されたコンテナの状態を基に，移送先ホストでコンテナの状態を復元する．コンテナマイグレーションを用いる例として，負荷分散やホストのメンテナンスが挙げられる．ホストの負荷が高くなった時に，いくつかのコンテナを他のホストに移動させることによって，ホストにかかる負荷を分散させることができる．また，コンテナを他のホストに移動させることによって，コンテナを止めることなく，ホストのメンテナンスを行うことができる．

移送元ホストの負荷が高い時にコンテナのマイグレーションを行うと，ホストの CPU，メモリ，ネットワークなどの負荷がマイグレーション性能に影響を与えてしまう可能性がある．移送先ホストとしては一般に負荷の低いホストが選ばれるが，マイグレーションには時間がかかることが多いため，マイグレーション中に移送先ホストの負荷が高くなることもある．このような場合には，移送先ホストの負荷もマイグレーション性能に影響を与える可能性がある．逆に，コンテナマイグレーションによって移送元ホストや移送先ホストの負荷が増大し，そのホストで実行されているコンテナの性能に影響を及ぼす可能性もある．

負荷分散のために VM マイグレーションを用いる場合については，様々な研究が行われている．Sandpiper [9] では，CPU やネットワークの使用率が 75%を超えた時に VM マイグレーションを実行することを推奨している．リソース使用率にまだ余裕がある内に VM マイグレーションを行うのは，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%，ネットワーク性能が 20%低下するためである．しかし，このような運用を行うと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 [10]．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．

クラウドでは VM 内でコンテナを動作させていることが多い [11]．これは，VM の方が物理ホストより柔軟に管理を行うことができるためである．この場合，VM 内で動作するコンテナマイグレーション機構が仮想化の影響を受けるため，コンテナマイグレーションの性能が低下してしまう．マイグレーションのオーバヘッドの主な原因はネットワーク仮想化であり，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 [5]．Portkey [5] 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 内のゲスト OS やマイグレーション機構の変更が必要である．

そこで，VM 内のコンテナを VM 外からマイグレーション可能にする OVmigrate [6] が提案されている．OVmigrate は VM 内で動作していたコンテナマイグレーション機構を VM 外で動作させる．それにより，コンテナマイグレーションが VM の負荷や仮想化の影響を受けないようにすることができる．しかし，OVmigrate は移送元で VM 外からコンテナの状態を保存することによる性能改善に焦点を当てており，現在のところ，移送先での状態復元は VM 内で実行している．多くのクラウドでは移送先でもコンテナは VM 内で動作し，コンテナマイグレーション中に移送先 VM の負荷が高くなることもあるため，コンテナの状態復元についても性能改善が必要である．

3. OVrestorer

本稿では，移送先 VM の外から VM 内にコンテナの状態を復元可能にするシステム OVrestorer を提案する．OVrestorer のシステム構成を図 2 に示す．従来は，コンテナが復元される移送先 VM 内でコンテナ復元機構が動作し，コンテナの状態を復元していた．それに対して，OVrestorer はコンテナ復元機構をコンテナが復元される移送先 VM と同じホスト上で動作させ，VM 内のコンテナの状態を VM 外から書き換える．OVrestorer は VM 内に空のコンテナを作成し，移送先 VM のメモリ上にあるそのコンテナの状態を移送元で保存したコンテナの状態で上書きすることで，コンテナの状態復元を行う．

OVrestorer はコンテナ復元機構を移送先 VM の外で動

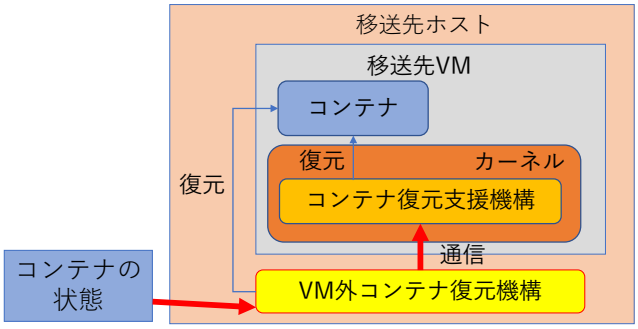

図 2 OVrestorer のシステム構成

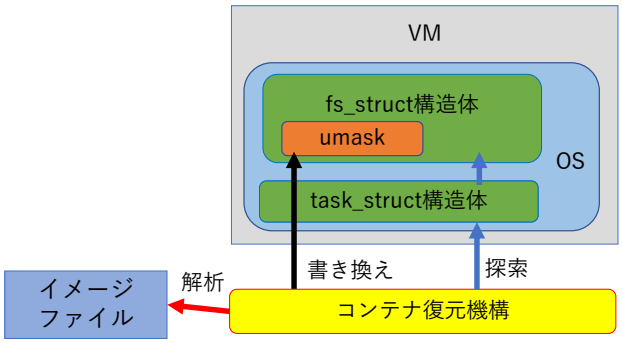

図 3 ファイルシステム情報の復元

作させているため，コンテナ復元機構は VM の負荷の影響を受けない．したがって，移送先 VM の負荷が高い場合でもコンテナの復元性能に対する影響を軽減することができる．また，移送先 VM による仮想化のオーバヘッドもコンテナ復元機構には影響しない．さらに，コンテナ復元機構が移送先 VM に負荷をかけないため，VM 内で実行されているコンテナの性能が低下することもない．

コンテナの状態は主にその中で動作しているプロセスの状態であるため，OVrestorer は VM のメモリ上にある OS データを解析して VM 内のプロセスの状態を書き換える．例えば，ファイル作成時のアクセス権限の設定を復元する際には，プロセス構造体からファイルシステム構造体を探索し，そのメンバ変数を書き換える．プロセスのメモリの内容を復元する際には，VM のメモリの中からプロセスに割り当てられているメモリ領域を探し，復元するデータを書き込む．VM 外から VM 内の OS データの解析と書き換えを容易にするために，OVrestorer は OS のソースコードを用いて復元処理を記述し，透過的に VM のメモリ上のデータを読み書きすることを可能にする．

VM 外のコンテナ復元機構が VM 内のコンテナのすべての状態を復元するのは容易ではないため，OVrestorer は VM 内で動作するコンテナ復元支援機構を用いる．例えば，VM 内に新しいプロセスを作成したり，プロセスにメモリを割り当てたりする処理は VM のメモリ書き換えだけでは実現するのが難しい．VM 内で動作するコンテナ復元支援機構は VM の負荷や仮想化の影響を受けるが，VM 内の OS カーネルの中で動作させることによりこれらの影響を軽減する．VM 外で動作するコンテナ復元機構は VM 内のコンテナ復元支援機構と協調して動作することでコンテナの状態の復元を行う．そのために，コンテナ復元機構は必要に応じてコンテナ復元支援機構と通信し，コンテナ復元支援機構が OS カーネル内で状態の復元を行う．

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 5.15 に対して OVrestorer を実装した．現在のところ，OVrestorer のコンテナ復元機構が VM 外から復元できる状態はプロセスのファイルシステム情報とメモリであり，他の状態については VM 内のコンテナ復元支援機構を用いて復元を行っている．

4.1 イメージファイルからの状態読み込み

OVrestorer は既存ツールの CRIU [12] によって作成されたイメージファイルから復元対象のプロセスの状態を取得する．CRIU はプロセスの状態の保存・復元を行うツールであり，コンテナの状態の保存・復元にも用いられている．CRIU はプロトコルバッファ [13] 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は proto ファイルで定義されている．OVrestorer ではこの proto ファイルを利用し，C 言語用のプロトコルバッファを用いてプロセスの状態をイメージファイルから読み込む．

4.2 ファイルシステム情報の復元

OVrestorer はファイルシステム情報の一つで，ファイルや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 umask を復元する．umask の復元の流れを図 3 に示す．まず，VM 外のコンテナ復元機構は umask の値が格納されているイメージファイルを解析し，保存された umask の値を取得する．次に，コンテナ復元機構は復元するプロセスの ID と一致するプロセスの情報が格納された task_struct 構造体を VM のメモリ上の OS データから探す．その後，task_struct 構造体からポインタをたどり，ファイルシステムの情報が格納された fs_struct 構造体を見つける．そして，この構造体のメンバである umask の値をイメージファイルから取得した値に書き換えることで復元を行う．

4.3 プロセスメモリの復元

プロセスメモリの復元の流れを図 4 に示す．OVrestorer はプロセスのメモリを復元するために，まずメモリマッピングの復元を行う．メモリを復元する時点ではまだ復元するプロセスに仮想メモリが割り当てられていない．OVrestorer はイメージファイルから仮想アドレスとサイズの組で構成されるプロセスのメモリ情報を取得する．VM

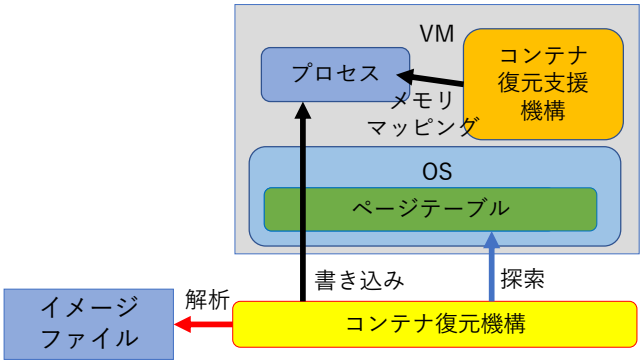

図 4 プロセスメモリの復元

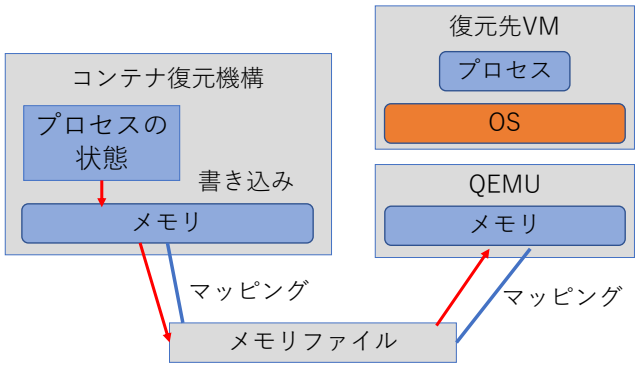

図 5 VM 外からのメモリ取得

外からプロセスに仮想メモリを割り当てるのは難しいため，メモリマッピングは VM 内のコンテナ復元支援機構が復元する．プロセスへの仮想メモリの割り当ては mmap システムコールを用いて行うが，その際に MAP_POPULATE フラグを指定することで物理メモリの確保まで行う．VM 外から物理メモリを確保するのも難しいためである．

次に，復元した仮想メモリに対してプロセスのメモリデータの復元を行う．VM 外のコンテナ復元機構は VM のメモリ上にある対象プロセスの task_struct 構造体からそのプロセスが用いるページテーブルの仮想アドレスを取得し，カーネルのページテーブルを用いてそれを物理アドレスに変換する．プロセスのページテーブルを用いて，復元するプロセスメモリの仮想アドレスを物理アドレスに変換する．そして，この物理アドレスに対応する VM のメモリ領域にイメージファイルから対応するメモリデータを 4KB 単位で読み込む．

4.4 VM 内のプロセス状態の書き換え

OVrestorer は KVmonitor [7] を用いて VM のメモリを読み書きする．KVMonitor を用いた OVrestorer のシステム構成を図 5 に示す．まず，VM に割り当てる物理メモリをホスト上のメモリファイルとして作成し，VM とコンテナ復元機構の両方に読み書き可能でマッピングして共有する．次に，コンテナ復元機構は QEMU と通信することによって，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物理アドレスを取得する．このページテーブルを用いて，VM 内の OS データの仮想アドレスを物理アドレスに変換し，さらにコンテナ復元機構内のアドレスに変換してアクセスする．

VM のメモリ上の OS データの解析を容易にするために，OVrestorer は LLView フレームワーク [8] を用いる．コンテナ復元機構として，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読み書き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 の中間表現を変換し，load 命令および store 命令の直前に

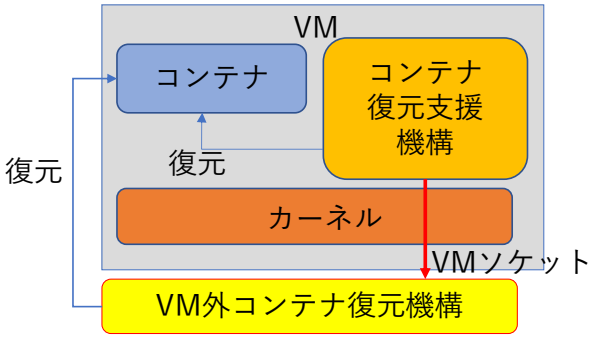

図 6 実装した OVrestorer の構成

OS データの仮想アドレスを変換して VM のメモリにアクセスするコードを埋め込む．

4.5 VM 内のコンテナ復元支援機構

OVrestorer は VM 内で動作するコンテナ復元支援機構と協調してコンテナの復元を行う．第一に，VM 外のコンテナ復元機構が VM のメモリ書き換えによって VM 内のコンテナのすべての状態を復元するのは容易ではないためである．第二に，コンテナのすべての状態を復元できないと正しく復元できていることを確認できないためである．VM の負荷や仮想化の影響を最小限に抑えるために，コンテナ復元支援機構は VM 内の OS カーネルの中で動作させ，VM ソケット（Vsock）を用いて VM 外のコンテナ復元機構と通信する．

現在のところ，VM 内のコンテナ復元支援機構は CRIU をベースにプロセスとして実装している．また，コンテナ復元機構が必要に応じてコンテナ復元支援機構を呼び出すのではなく，基本的には VM 内で動作する CRIU が状態の復元を行い，実装できた復元処理を実行する際に VM 外のコンテナ復元機構を呼び出している．この呼び出し時には Vsock を用いてコマンドと復元に必要な情報を送信し，実行結果を受信する．

5. 実験

OVrestorer を用いて VM 内のプロセスの状態を正常に

図 7 CRIU による状態保存時の出力結果

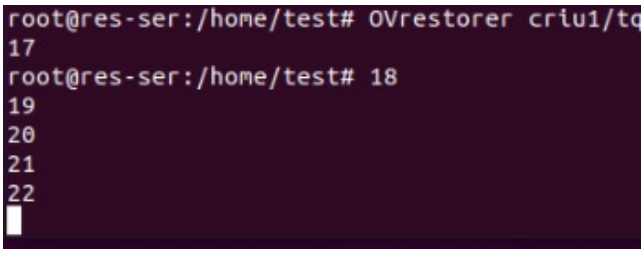

図 8 OVrestorer による状態復元時の出力結果

復元できることを確認する実験を行った．また，多くのメモリを使用するプロセスの状態を復元するのにかかる時間を測定した．比較として，VM 内で既存ツールの CRIU を用いてプロセスの状態を復元するのにかかる時間も測定した．この実験には，Intel Xeon W-2245 の CPU，64GB のメモリ，2.5TB の HDD を搭載したマシンを用いた．このマシンではホスト OS として Linux 5.15，仮想化ソフトウェアとして QEMU-KVM 4.2.0 を動作させ，VM ではゲスト OS として Linux 5.15 を動作させた．

5.1 プロセスの状態復元の確認

OVrestorer を用いてプロセスの状態を復元する実験を行った．まず，VM 内で CRIU を用いてプロセスの状態を保存し，プロセスのメモリと umask の情報が含まれるイメージファイルについては VM 外に転送した．次に，VM 外のコンテナ復元機構がプロセスのメモリと umask を復元し，VM 内のコンテナ復元支援機構がそれ以外の状態を復元した．この実験では 5 秒ごとにカウンタ値を 1 ずつ増加させて表示するプロセスを用い，VM には仮想 CPU を 4 個，メモリを 45GB 割り当てた．

CRIU によるプロセスの状態保存時の出力結果を図 7 に，OVrestorer による状態復元時の出力結果を図 8 に示す．プロセスの状態復元時には状態保存時のカウンタ値から表示が再開されることが確認できた．このことより，OVrestorer は VM 外からプロセスのメモリを正しく復元できているといえる．また，プロセスの状態保存前と状態復元後で umask の値が一致していることも確認できた．

5.2 仮想化や負荷が復元性能に与える影響

仮想化がプロセスの復元性能に与える影響を調べるために，VM 内で CRIU を用いてプロセスの状態を復元するのにかかる時間を測定した．比較として，VM 外のホスト上で CRIU を用いてプロセスの復元にかかる時間を調べた．また，VM の負荷がプロセスの復元性能に与える影響を調べるために，高負荷の VM 内で CRIU を用いてプロセスの状態を復元するのにかかる時間を測定した．VM を高負荷にするために，VM 内で stress コマンドを実行して CPU に負荷をかけた．この実験ではメモリを 4GB 使用するプロセスを用い，VM には仮想 CPU を 2 個，メモリを 8GB 割り当てた．

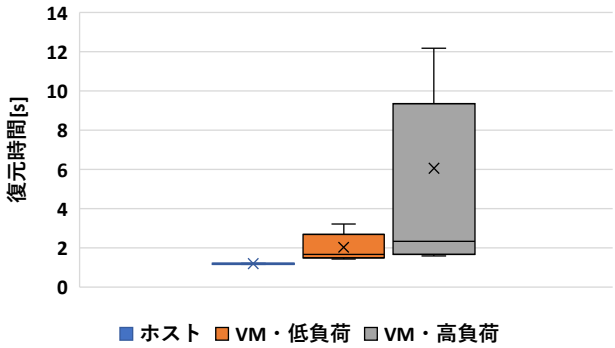

図 9 プロセスの復元時間

測定結果を図 9 に示す．実験結果より，ホスト上でプロセスを復元する場合と比べて，VM 内でプロセスの状態を復元すると仮想化の影響を受け，復元時間が平均で 70%増加することが分かった．それに加えて，復元時間のばらつきが大きくなった．また，VM が高負荷になると VM の負荷が低い時と比べて，復元時間のばらつきが非常に大きくなり，復元時間が平均で 200%増加することが分かった．

5.3 OVrestorer の復元性能

OVrestorer を用いてメモリを 5GB 使用するプロセスを復元するのにかかる時間を測定した．特に，コンテナ復元機構が VM 外からプロセスの umask とメモリを復元するのにかかる時間を詳細に調べた．プロセスメモリの復元については，コンテナ復元支援機構が VM 内でプロセスのメモリマッピングを復元するのにかかる時間も測定した．これらの復元時間を VM が低負荷の場合と高負荷の場合について調べた．VM を高負荷にするために，VM 内で stress-ng コマンドを実行して仮想ディスクに負荷をかけた．比較のために，VM 内で CRIU を用いてプロセスを復元した場合についても低負荷の場合と高負荷の場合について測定を行った．この実験では，VM に仮想 CPU を 4 個，メモリを 45GB 割り当てた．

umask の復元にかかった時間の平均を図 10 に示す．実験結果より，OVrestorer を用いて復元を行うと VM 内で CRIU を用いて復元するよりも大幅に時間がかかることが分かった．これは CRIU における umask の復元処理が umask システムコールを呼び出すだけであり，非常に単純

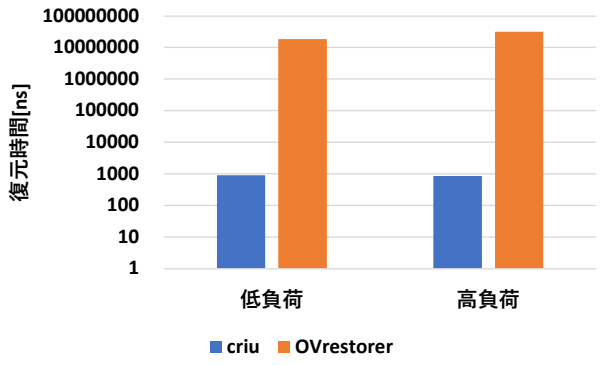

図 10　umask の復元時間

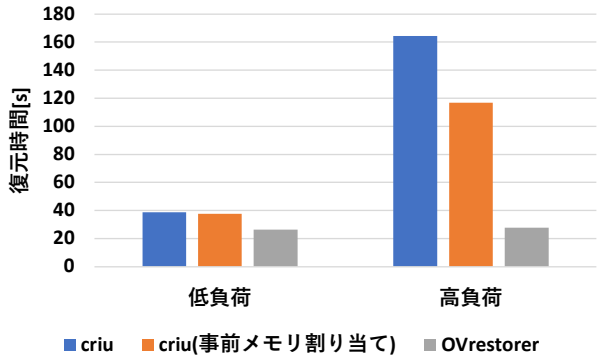

図 11　プロセスメモリの復元時間

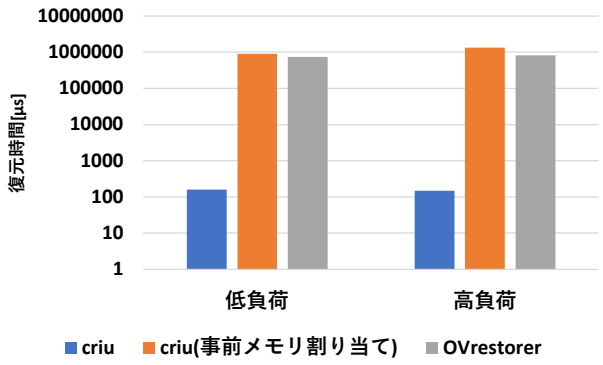

図 12　メモリマッピングの復元時間

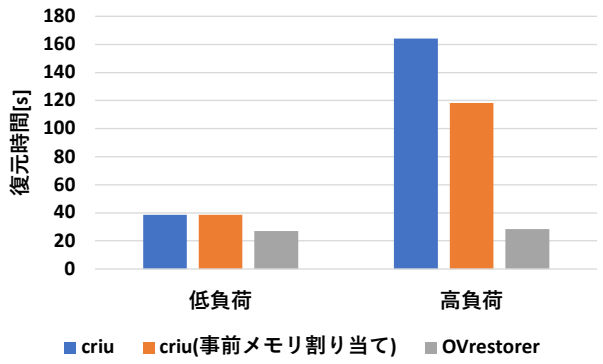

図 13　プロセスの復元時間

であるためと考えられる．ただし，OVrestorer による復元時間の増加は 20〜30ms 程度であった．CRIU を用いた場合は VM の負荷の影響をほぼ受けなかったが，OVrestorer の場合は影響が大きかった．この原因は現在，調査中である．

次に，プロセスメモリの復元にかかった時間の平均を図 11 に示す．実験結果より，VM が低負荷の時には OVrestorer の方が VM 内の CRIU よりも 30%性能がよかった．このことから，OVrestorer は仮想化の影響を軽減できていると考えられる．一方，VM が高負荷の時は OVrestorer の方が CRIU よりも 5.9 倍高速であった．これは，OVrestorer によるプロセスメモリの復元処理が VM にかかっている負荷の影響をほとんど受けていないということである．この実験では VM の仮想ディスクに負荷をかけたため，VM 内の CRIU がメモリデータをファイルから読み込む時間が増加したと考えられる．

一方，プロセスメモリの復元前に行うメモリマッピングの復元にかかった時間の平均を図 12 に示す．実験結果より，OVrestorer は CRIU よりも復元にかかる時間が非常に長いことが分かった．これは，OVrestorer も CRIU も VM 内でプロセスのメモリマッピングを復元するが，OVrestorer では物理メモリの割り当てまで行うのに対し，CRIU はメモリの復元時に物理メモリの割り当てを行うためである．ただし，復元時間の増加は 0.7〜0.8 秒程度であり，プロセスメモリの復元時間と比べると非常に小さい．

OVrestorer と同様に，CRIU でもメモリマッピングの際に物理メモリを割り当てるようにした場合，図 12 のようにメモリマッピングにかかる時間は 0.9〜1.3 秒増加し，OVrestorer と同程度となった．その一方で，図 11 のようにプロセスメモリの復元時間は短縮された．VM が低負荷の場合は元の CRIU と同程度であったが，高負荷の場合には 40 秒も減少した．しかし，まだ OVrestorer の方が 4.2 倍高速であった．

最後に，プロセスのすべての状態の復元にかかった時間の平均を図 13 に示す．実験結果より，VM が低負荷の場合は OVrestorer の方が 30%高速であり，高負荷の場合は 5.8 倍高速であった．また，CRIU でメモリマッピングの際に物理メモリをプロセスに割り当てるようにしてもまだ 4.1 倍高速であった．プロセスメモリの復元時間がほとんどの時間を占めており，umask やメモリマッピングの復元のオーバヘッドはプロセスメモリの復元の性能改善によって打ち消せていることが分かる．

6. 関連研究

Portkey [5] は VM 内のコンテナを効率よくマイグレーションできるようにするために，ネットワーク転送を最適化している．移送元 VM 内の CRIU はコンテナの状態を保存した後，Portkey のカーネルモジュール経由でハイパー

バイザを呼び出すことにより，ゲスト OS のネットワーク処理をバイパスする．ハイパーバイザが移送先 VM にコンテナの状態を転送すると，その VM 内の CRIU はカーネルモジュール経由でコンテナの状態を受け取り，コンテナの状態を復元する．これにより，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．OVrestorer はコンテナの状態の中で最も大きな割合を占めるプロセスメモリのデータを VM 内に転送しないため，さらに CPU 使用率を削減し，マイグレーションを高速化できる可能性がある．

mWarp [14] 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けるプロセスメモリのコピーや転送にかかる時間を削減する．mWarp では，移送元 VM 内の CRIU は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 へマッピングし直し，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．OVrestorer 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 [15] 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 [16] 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 OVrestorer と組み合わせて利用することができると考えられる．

GPUfas [17] は GPU からメインメモリの書き換えによって OS データを変更し，システム障害からの復旧を行う．例えば，シグナル関連のデータを書き換えることでプロセスにシグナルを送信したり，スケジューラのデータを書き換えることでプロセススケジューリングを変更したりすることができる．GPUfas はホストから VM のメモリを書き換えて復旧を行うこともでき，OVrestorer も同様の手法を用いている．GPUfas のように，状態を復元したプロセスにシグナルを送信することにより，プロセスを再開することができると考えられる．

7. まとめ

本稿では，VM 外から VM 内にコンテナの状態を復元可能にするシステム OVrestorer を提案した．OVrestorer は VM のメモリ上にあるコンテナの状態を VM 外から書き換えることにより，コンテナの状態復元を行う．これにより，VM の負荷や仮想化のオーバヘッドがコンテナの復元性能に与える影響を最小化することができる．すべての状態を VM 外から復元するのは容易ではないため，OVrestorer は VM 外で動作するコンテナ復元機構と VM 内で動作するコンテナ復元支援機構を協調させる．実験から，OVrestorer が VM 内のプロセスを保存時の状態に復元できることを確認し，VM の負荷が高い時でも高速にプロセスメモリを復元できることが分かった．

今後の課題は，コンテナやその中で動作するプロセスの様々な状態を VM 外で復元できるようにすることである．また，VM 内のコンテナ復元支援機構を OS カーネルの中で動かせるようにして VM の負荷の影響を受けにくくする必要がある．

謝辞 本研究の一部は，JST，CREST，JPMJCR21M4 の支援を受けたものである．また，本研究の一部は，国立研究開発法人情報通信研究機構の委託研究（05501）による成果を含む．

参考文献

[1] Amazon Web Services, Inc.: Amazon Elastic Container Service (Amazon ECS), `https://aws.amazon.com/ecs/`.
[2] Amazon Web Services, Inc.: Managed Kubernetes Service, `https://aws.amazon.com/jp/eks/`.
[3] Google: Google Kubernetes Engine(GKE), `https://cloud.google.com/kubernetes-engine`.
[4] Microsoft Corporation: Managed Kubernetes Service, `https://azure.microsoft.com/en-us/products/kubernetes-service`.
[5] 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).
[6] 朝倉優輝，光来健一：VM 外で実行可能なコンテナの状態保存機構，*SWoPP* (2022).
[7] Nakamura, K. and Kourai, K.: Efficient VM Introspection in KVM and Performance Comparison with Xen, *Proceedings of the 20th IEEE Pacific Rim International Symposium on Dependable Computing*, pp. 192–202 (2014).
[8] 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).
[9] Wood, T., Shenoy, P., Venkataramani, A. and Yousif, M.: Black-box and Gray-box Strategies for Virtual Machine Migration, *Proceedings of the 4th USENIX Symposium on Networked Systems Design & Implementation* (2007).

[10] 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).
[11] VMware,Inc: VMware Tanzu, `https://tanzu.vmware.com/application-platform`.
[12] The CRIU Team: CRIU, `https://criu.org/Main_Page`.
[13] Google LLC: Protocol buffers, `https://protobuf.dev/`.
[14] Sinha, P., Doddamani, S., Lu, H. and Gopalan, K.: mWarp: Accelerating Intra-Host Live Container Migration via Memory Warping, *Proceedings of IEEE INFOCOM 2019* (2019).
[15] 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).
[16] 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, *Proceedings of the 13th IEEE International Conference on Cloud Computing*, pp. 321–328 (2020).
[17] 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).