コンテナに対する攻撃からの高速な復旧のための VM 外状態復元機構

木本 翔太[1]　光来 健一[1]

概要：コンテナはチェックポイント機能を用いて定期的に状態を保存しておき，攻撃を受けた時にリストア機能を用いて状態を復元することにより，速やかに復旧を行うことができる．しかし，コンテナを VM 内で動作させるクラウドにおいては，状態の保存・復元処理が仮想化や VM の負荷の影響を受ける．状態保存についてはこれらの影響を軽減する手法が提案されているが，より複雑な処理を必要とする状態復元についてはまだ対処が行われていない．本稿では，攻撃から高速に復旧できるようにするために，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 内に作成した空のコンテナの状態を保存しておいた状態で上書きすることにより VM 外での復元を実現し，仮想化や負荷の影響を軽減する．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と連携する．実験の結果，OVrestorer は VM 内で既存ツールを実行するよりも高速にコンテナを復元できることが分かった．

1. はじめに

近年，Amazon ECS [1] や EKS [2]，Google の 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．コンテナはチェックポイント機能を用いて定期的に状態を保存しておくことにより，攻撃に備えることができる．コンテナが攻撃を受けたことを検知すると，リストア機能を用いて保存しておいた状態を復元することで速やかに復旧を行うことができる．状態をほとんど持たないコンテナの場合には再起動による復旧も可能であるが，ビッグデータ処理を行うコンテナのように大きな状態をもつ場合には再起動後に処理をやり直すのは効率が悪い．

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，コンテナの状態の保存・復元処理は仮想化の影響を受ける．また，コンテナの状態の保存・復元を行う際に VM の負荷が高い場合には，保存・復元処理が負荷の影響を受ける．逆に，コンテナの状態の保存・復元処理によって VM の負荷が増大し，その VM 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．そこで，先行研究 [5] では VM の外でコンテナの状態を保存することで仮想化や VM の負荷の影響を抑えている．しかし，コンテナの状態復元は状態保存と比べて非常に複雑であるため，これらの影響への対処は行えていない．

本稿では，コンテナを高速に復旧するために 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 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チェックポイント時に保存されたコンテナの状態で上書きする．これにより，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影響を与えないようにすることができる．また，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．同時に，状態復元の負荷が VM 内のコンテナの性能に及ぼす影響を抑えることもできる．ディスクなどの VM と共有するリソースに関する負荷については，コンテナ復元時にのみ VM のリソース制限を行うことで影響を抑える．

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 のコンテナ復元機構は VM のメモリ上にある OS のデータを解析して VM 内のプロセスの状態を書き換える．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と連携する．例えば，復元時に新たに必要になる OS データはコンテナ復元支援機構が事前に確保して特定のメモリ領域に格納しておき，コンテナ復元機構はそれを取り出して利用する．実験により，OVrestorer は仮想化や負荷の影

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

響を抑えて，VM 内で動作する既存ツールより高速にコンテナを復元できることが分かった．

以下，2 章でコンテナのチェックポイント・リストア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行う OVrestorer を提案し，4 章でその実装について述べる．5 章では OVrestorer を用いてコンテナの状態を復元する性能について調べ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. コンテナのチェックポイント・リストア

コンテナはチェックポイント機能を用いて定期的に状態を保存しておくことにより，攻撃に備えることができる．コンテナが攻撃を受けた時には，コンテナ全体に攻撃の影響が及んでいる可能性があるため，状態をすべて破棄することが望ましい．しかし，メモリ上に大量のデータを保持してビックデータ処理などを行うコンテナの場合には，状態を破棄することによる損失が大きい．攻撃を受ける前の状態が保存されていれば，攻撃を受けた時にリストア機能を用いてコンテナを復元して速やかに復旧を行うことができる．

クラウドでは VM 内でコンテナを動作させることが多い [6]．これは，物理ホスト上で動作させるよりも VM を用いる方が柔軟に管理を行うことができるためと考えられる．この場合，VM 内で動作するコンテナ管理機構が仮想化の影響を受けるため，コンテナの状態を保存・復元する際の処理性能が低下する．また，VM 内で保存したコンテナの状態を別の VM やホストに転送して保管しておき，リストア時に別の VM で復元するということも考えられる．これはコンテナを別の VM に移動させるコンテナマイグレーションと同様の処理となる．VM 内のコンテナをマイグレーションする際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%になると報告されている [7]．

VM の負荷が高い時にコンテナの保存・復元を行うと，VM 内で動作するコンテナ管理機構が負荷の影響を受け，保存・復元の処理性能が低下する可能性がある．例えば，コンテナの状態を定期的に保存するには，コンテナの負荷の高い時に状態を保存したり，負荷の高いコンテナが動作している VM 内で状態を保存したりすることも必要となる．また，コンテナの状態を復元している間に VM 内の別のコンテナの負荷が高くなることも考えられる．復元するコンテナが必要とするデータや協調実行するコンテナが負荷の高い VM 内にあるために，その中で復元を行うことが必要になる場合もある．逆に，コンテナの状態の保存・復元処理によって VM の負荷が増大し，その VM 内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．我々の実験によると，VM が高負荷の場合，コンテナの状態の保存時間は 4.3 倍，復元時間は 5.8 倍に増加した．

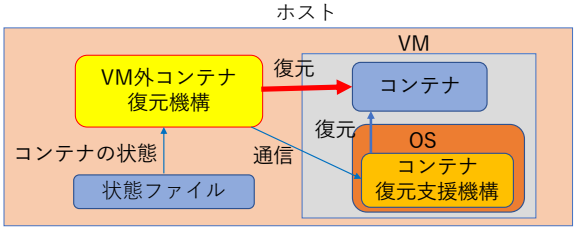


図 1: OVrestorer のシステム構成

そこで，先行研究 [5] ではコンテナの状態を保存するコンテナ保存機構を VM の外で動作させることを可能にしている．VM 内で動作しているコンテナの状態は VM のメモリ上にあるため，VM 外のコンテナ保存機構は VM のメモリを解析してコンテナの状態を取得する．これにより，コンテナの状態の保存処理が仮想化や VM の負荷の影響を受けないようにすることができる．しかし，コンテナの状態の復元処理は保存処理と比べて非常に複雑であるため，状態復元時の仮想化や VM の負荷の影響への対処はまだ行えていない．

3. OVrestorer

本稿では，VM の外から VM 内にコンテナの状態を復元することを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 1 に示す．従来は，コンテナが復元される VM の中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 VM と同じホスト上ではあるが VM の外で動作させる．そして，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チェックポイント時に保存された状態で上書きすることで，コンテナの状態復元を行う．

コンテナ復元機構を VM の外で動作させることにより，復元処理が VM による仮想化の影響を受けずに済む．そのため，ディスクやネットワークなどの仮想化のオーバヘッドなしでコンテナの復元を行うことができる．また，VM 外のコンテナ復元機構の負荷と VM 内のコンテナの負荷は基本的に影響を及ぼし合うことはない．ただし，コンテナ復元機構と VM が共有するリソースについては，コンテナ復元時にのみ VM のリソース制限を行うことでコンテナ復元機構への影響を抑える．例えば，VM 内のコンテナがディスクに頻繁にアクセスすると，チェックポイント時に保存されたコンテナの状態を同じディスクから読み込むコンテナ復元機構の性能が低下する可能性がある．そこで，VM のディスク帯域制限を行うことにより，コンテナ復元機構が利用可能なディスク帯域を確保できるようにする．

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．

具体的には，復元するプロセスに対応する，OS カーネル内のプロセス構造体からポインタをたどって様々な構造体を探索し，それらのメンバ変数をチェックポイント時に保存した値で上書きする．またプロセスのメモリデータを復元するために，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特定し，チェックポイント時に保存したデータを書き込む．VM 外から VM 内の OS データの解析や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，VM 内で動作するコンテナ復元支援機構と協調して復元を行う．例えば，コンテナ復元機構が OS データを新たに確保したり解放したりするにはメモリ管理を行うことが必要になるが，VM 外で OS カーネルのメモリ管理を操作するのは難しい．そこで，コンテナ復元支援機構は必要になる可能性がある OS データを VM 内で事前に確保して特定のメモリ領域に格納しておき，不要になった OS データは後で VM 内で解放する．また，VM 外で VM の仮想ハードウェアを操作するのも難しいため，コンテナ復元機構はコンテナ復元支援機構と通信し，負荷の影響を受けにくい OS カーネル内で復元を行う．

4. 実装

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．

4.1 コンテナ復元機構

VM 外で動作するコンテナ復元機構はプロセスの状態が保存されたイメージファイルを読み込み，VM のメモリ上の OS データを書き換えることによりプロセスの状態の復元を行う．イメージファイルの読み込みについては 4.3 節，VM のメモリ上の OS データの書き換えについては 4.4 節で詳しく説明する．コンテナ復元機構は復元しようとするプロセス 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 のメモリから探し，それを起点として様々な構造体を操作する．

4.1.1 デフォルトのファイルアクセス権の復元

VM 外で実装可能な例として，プロセスがファイルやディレクトリを作成する際にデフォルトで設定されるアクセス権の復元を挙げる．従来は VM 内に作成したプロセスが `umask` システムコールを実行することにより復元が行われていた．ファイルアクセス権の復元の流れを図 2 に示す．VM 外のコンテナ復元機構はまず，ファイルアクセス権に関する情報が格納されている `fs.img` を解析し，チェックポイント時に保存された値を取得する．次に，`task_struct` 構造体からポインタをたどり，ファイルシステムに関する情報が格納された `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` の値をチェックポイント時の値で上書きすることにより復元を行う．

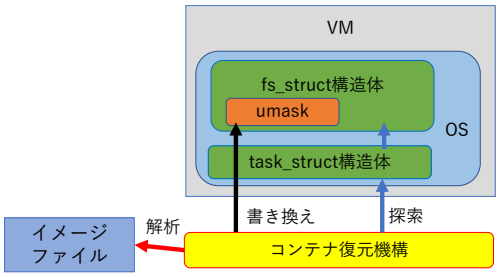


図 2: ファイルアクセス権の復元

4.1.2 リソース制限の復元

復元する状態によっては VM 内のコンテナ復元支援機構との連携が必要になる例として，プロセスのリソース制限の復元を挙げる．従来は VM 内に作成したプロセスが `setrlimit` システムコールを実行することにより復元が行われていた．VM 外のコンテナ復元機構はまず，リソース制限に関する情報が格納されている `core.img` を解析し，チェックポイント時に保存された値を取得する．次に，`task_struct` 構造体からポインタをたどり，シグナルに関する情報が格納された `signal_struct` 構造体を見つける．そして，この構造体のメンバである `rlim` の値をチェックポイント時の値で上書きすることで復元を行う．

`rlim` メンバはリソースごとに現在の制限値を最大の制限値を保持している配列であり，OVrestorer はすべてのリソースについて制限値を上書きする．例えば，CPU 時間やスタックサイズなどがある．CPU リソースについては，復元した現在の制限値が無限ではない場合，CPU タイマーを設定してリソース制限が適用されるようにする必要がある．VM 外で CPU タイマーの設定を行うのは難しいため，この処理が必要となった場合には VM 内のコンテナ復元支援機構を呼び出して処理を実行させる．ただし，復元する現在の制限値は無限である場合も多く，その場合には VM 外のコンテナ復元機構だけで復元が可能である．

4.1.3 メモリデータの復元

プロセスのメモリは一般的に最も大きな状態であり，仮想化や負荷の影響が相対的に大きくなるため，そのデータを VM 外で復元することが重要である．従来は VM 内に作成したプロセスがイメージファイルに保存されたメモリデータを自身のメモリに読み込むことにより復元が行われていた．コンテナ復元機構はまず，`task_struct` 構造体からポインタをたどり，メモリ管理に関する情報が格納された `mm_struct` 構造体を見つける．そして，その構造体から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．その物理アドレスを用いて VM のメモリに

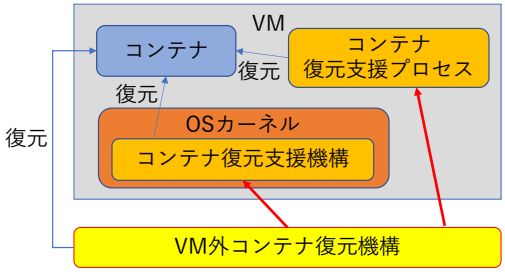


図 3: コンテナ復元支援プロセス

アクセスし，プロセスのメモリ領域の仮想アドレスを物理アドレスに変換する．この物理アドレスを用いて VM のメモリにアクセスし，pages.img から読み込んだメモリデータを 4KB 単位で書き込む．

4.2 コンテナ復元支援機構

VM 内で動作するコンテナ復元支援機構は VM 専用の VM ソケット（Vsock）を用いてコンテナ復元機構から情報を受け取り，プロセスの状態の復元を行う．VM の仮想化や負荷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 の OS カーネル内で動作させ，OS カーネルの機能を用いて復元を行う．それに加えて，現在の実装では，図 3 のように VM の OS カーネル上で動作するコンテナ復元支援プロセスとも協調してコンテナの復元を行う．コンテナ復元支援プロセスはコンテナ復元機構にも OS カーネル内のコンテナ復元支援機構にも実装できていない復元処理を実行する．コンテナ復元支援プロセスは従来ツールの CRIU [8] の復元処理をベースに実装しており，システムコールを用いて復元を行う．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．

4.2.1 非同期での OS データの確保・解放

OVrestorer は復元処理に必要となる OS データの確保・解放をコンテナ復元機構の実行とは非同期に行う．VM 外のコンテナ復元機構が VM 内の OS データを新たに確保したり，不要になった OS データを解放したりするのは容易ではない．しかし，OS データの確保・解放が必要になるたびに Vsock を用いてコンテナ復元支援機構と通信を行うと，通信のオーバヘッドで復元性能が低下する．この問題を解決するために，復元処理を開始する際に VM 内のコンテナ復元支援機能が新たに確保される可能性がある OS データを事前に確保しておく．確保した OS データは確保用のメモリ領域に格納する．コンテナ復元機構が新たに OS データを必要とした際には，VM のメモリを解析して確保用のメモリ領域から必要な OS データを取得する．一方，使われている OS データが不要になった際には，VM のメモリを解析して解放用のメモリ領域にその OS データを格納する．VM 内のコンテナ復元支援機構は復元処理を終了する際に解放用のメモリ領域に OS データがあれば解放処理を行う．

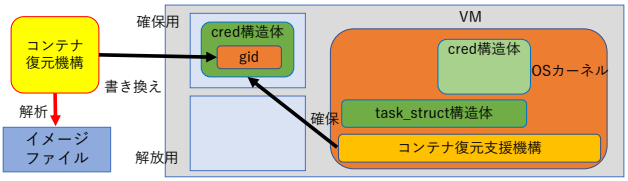


図 4: グループ ID の復元

4.2.2 グループ ID の復元

非同期での OS データの確保・解放を利用して実装した例として，プロセスのグループ ID の復元を挙げる．従来は VM 内に作成されたプロセスが setresgid システムコール等を用いることにより復元が行われていた．グループ ID は task_struct 構造体からたどれる cred 構造体の gid メンバに格納されている．しかし，この cred 構造体は OS カーネルの様々な箇所から参照されている可能性があるため，そのメンバを直接，上書きするとカーネル内での整合性が取れなくなる場合がある．アトミックに書き換えを行うために，OS カーネル内では新たに cred 構造体を確保し，現在の cred 構造体の中身をコピーして書き換え，新しい cred 構造体を task_struct 構造体のメンバに代入している．

コンテナ復元機構は OS カーネルと同様の方法で cred 構造体の gid メンバを書き換える．コンテナ復元機構はまず，グループ ID に関する情報が格納されている core.img を解析し，チェックポイント時に保存された値を取得する．次に，図 4 のように，コンテナ復元支援機構が事前に確保しておいた cred 構造体を確保用の VM のメモリ領域から取得し，その gid メンバをチェックポイント時の値で上書きする．この構造体を task_struct 構造体のメンバに代入することによってグループ ID の復元を行う．そして，不要になった古い cred 構造体は解放用の VM のメモリに格納する．後で，コンテナ復元支援機構がその cred 構造体の解放処理を行い，他の参照がなくなった時点で解放が行われる．

4.2.3 インターバルタイマーの復元

VM の仮想ハードウェアを操作する必要があるために OS カーネル内に実装した例として，インターバルタイマーの復元を挙げる．インターバルタイマーはプロセスが設定し，設定した時間が経過するとシグナルを送信する機能である．従来は VM 内に作成されたプロセスが setitimer システムコールを実行することにより復元が行われていた．コンテナ復元機構はまず，タイマー情報が格納されている core.img を解析し，チェックポイント時に保存された値を取得する．次に，取得したタイマー情報を Vsock を用いてコンテナ復元支援機構に送信する．VM 内のコンテナ復元支援機構は復元するプロセスに対応する task_struct 構造体を探し，signal_struct 構造体のメン

バである real_timer の値を上書きすることで復元を行う．また，必要に応じてハードウェアにタイマーの設定を行う．この処理は setitimer システムコールと同様であるが，システムコールを呼び出したプロセスに対してではなく，指定されたプロセスに対して処理を行う．

4.2.4 メモリマッピングの復元

VM 外で実装するのが難しいために VM 内のコンテナ復元支援プロセスに実装した例として，プロセスのメモリマッピングの復元を挙げる．従来は VM 内に作成されたプロセスが mmap システムコールを用いることにより復元が行われていた．コンテナ復元機構はまず，pagemap.img 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領域の情報を取得し，Vsock を用いてコンテナ復元支援プロセスに送信する．コンテナ復元支援プロセスはその仮想アドレスとサイズを指定して mmap システムコールを発行し，プロセスに仮想メモリ領域を割り当てる．その仮想メモリ領域に VM 外で物理メモリを割り当てるのも難しいため，mmap システムコールに MAP_POPULATE フラグを指定することで物理メモリの確保を同時に行う．この処理を OS カーネル内で実装することは難しくはないと考えられる．

4.3 イメージファイルからの状態読み込み

OVrestorer は CRIU がプロセスの状態を保存するために用いているイメージファイルと同じフォーマットを用いてプロセスの復元を行う．CRIU はプロトコルバッファ [9] 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 はこの proto ファイルを利用し，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．VM 外でプロセスの状態を保存する OVmigrate も同じフォーマットを用いているため，OVrestorer は OVmigrate が VM 外に作成したイメージファイルを用いて復元を行うこともできる．

4.4 VM 内の OS データの読み書き

OVrestorer は KVMonitor [10] を用いて VM のメモリを読み書きする．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することにより共有する．次に，コンテナ復元機構は QEMU と通信することにより，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．この物理アドレスを用いてコンテナ復元機構にマッピングされた VM のメモリにアクセスし，ページテーブルをたどることにより VM 内の OS データの仮想アドレスを物理アドレスに変換する．そして，この物理アドレスを用いてメモリファイル上の OS データにアクセスする．

VM のメモリ上にある OS データの解析を容易にするために，LLView フレームワーク [11] を用いてコンテナ復元機構を開発した．LLView は Linux カーネルのヘッダファイルで定義されている構造体やグローバル変数，インライン関数，マクロなどを用いて解析プログラムを作成することを可能にする．LLView を用いてコンテナ復元機構のプログラムをコンパイルし，生成された LLVM の中間表現を変換する．具体的には，メモリの読み書きを行うための load 命令および store 命令の直前にコードを埋め込み，上で述べたように OS データの仮想アドレスを変換して VM のメモリの読み書きを行わせる．

4.5 コンテナの復元

コンテナの復元を行う際には，VM 外のコンテナ復元機構から Vsock 経由で要求を受け取った VM 内のコンテナ復元支援プロセスがコンテナ復元のための Docker コマンドを実行する．この Docker コマンドは新しいコンテナを作成し，その中で CRIU を実行することにより復元に用いる新しいプロセスを作成する．その後，VM 外のコンテナ復元機構がプロセスの復元を開始し，必要に応じて VM 内のコンテナ復元支援機構を呼び出す．プロセスの復元が完了すると，CRIU の実行完了を待っていた Docker コマンドが VM 内でコンテナの残りの状態を復元する．

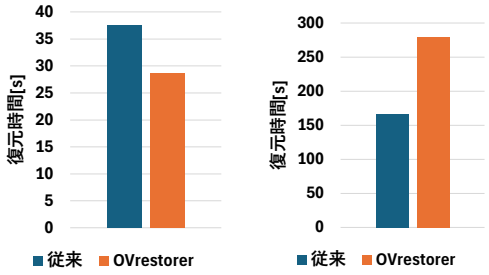
コンテナ復元時のリソース競合を避けるために，コンテナ復元機構は復元時にのみ VM のリソース制限を行う．現在の実装では，virsh blkdeviotune コマンドを用いて VM の仮想ディスクの帯域を制限することにより，VM 外でのイメージファイルの読み込みが VM 内のディスクアクセスの影響を受けにくくなるようにしている．コンテナの復元が完了すると，コンテナ復元機構は VM のリソース制限を解除する．

5. 実験

OVrestorer を用いて VM 外からコンテナの復元が行えることを確認し，復元性能を調べる実験を行った．比較として，VM 内で既存の Docker-27.4.1 と CRIU-3.17.1 を用いてコンテナの復元を行った場合についても測定を行った．この実験には，Intel Xeon W-2245 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 コンテナの状態復元の確認

キャッシュサーバの memcached [12] が動作するコンテナを復元する実験を行った．まず，コンテナ内の memcached にデータを格納し，VM 内で Docker のチェックポイント機能を用いてコンテナの状態を保存した．その後，状態が



(a) 低負荷時　　(b) 高負荷時

図 5: プロセス復元時間

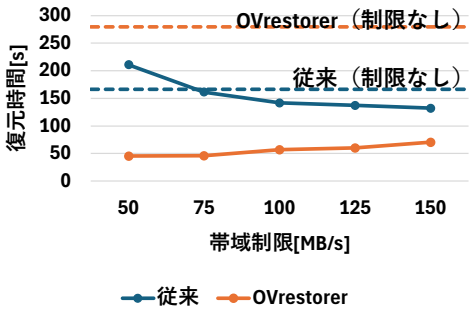


図 6: 帯域制限ごとのプロセス復元時間

保存されたイメージファイルを VM 外に転送し，コンテナへの攻撃が行われたことを想定してコンテナを停止させた．次に，VM 外のコンテナ復元機構と VM 内のコンテナ復元支援機構およびコンテナ復元支援プロセスを協調させてコンテナの復元を行った．その結果，VM 内にコンテナを復元することができ，復元されたコンテナ内の memcached に格納されているデータが保存前と一致していることが確認できた．

5.2　プロセスの復元性能

まず，OVrestorer を用いてプロセスのみを復元するのにかかる時間を測定した．そのために，VM 内でメモリを 5GB 使用するプロセスを実行して従来ツールの CRIU を用いて状態を保存し，作成されたイメージファイルを用いて復元を行った．比較のために，VM 内で CRIU を用いてこのプロセスを復元した場合についても測定を行った．

VM に負荷をかけなかった時の復元性能を図 5a に示す．VM が低負荷の場合，OVrestorer は従来ツールより 25% 高速にプロセスを復元できることが分かった．この結果より，OVrestorer は VM による仮想化の影響を抑えることができているといえる．次に，VM に負荷をかけた時の復元性能を調べた．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．高負荷時の復元性能を図 5b に示す．OVrestorer では，VM にディスク帯域制限をかけなかった場合，VM 外のコンテナ復元機構が VM 内の負荷の影響を受けて従来ツールよりも復元性能が低下することが分かった．

VM に様々なディスク帯域制限をかけた場合の復元性能を図 6 に示す．この結果より，少しでも VM のディスク帯域制限を行うと OVrestorer の復元性能が大幅に改善することが分かった．帯域を 150MB/s に制限した場合，従来ツールの 1.8 倍高速にプロセスの復元を行うことができ，帯域制限を強くするほど性能差が開いた．ディスク帯域制限が VM 内のコンテナの性能に及ぼす影響を調べるために，コンテナ内で fio コマンドを実行してディスクアクセス性能を測定した．実験結果は図 7 に示すようになり，100MB/s まではディスク帯域制限をかけてもコンテナのディスクアクセス性能はほとんど低下しないことが分かった．

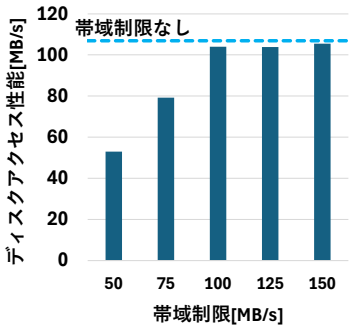


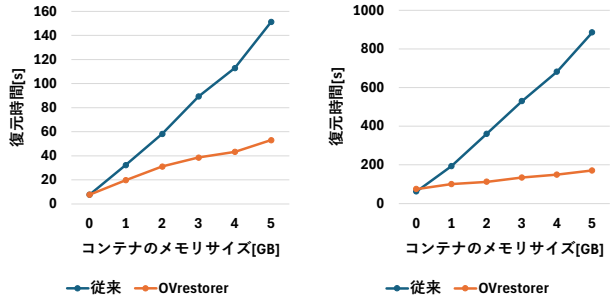
図 7: 帯域制限ごとのコンテナのディスクアクセス性能

5.3　コンテナの復元性能

memcached プロセスを含むコンテナ全体の復元性能を調べる実験を行った．この実験では，VM が低負荷の場合と高負荷の場合についてコンテナの復元にかかる時間を測定し，VM には 150MB/s のディスク帯域制限をかけた．比較のために，VM 内で従来ツールの Docker を用いてコンテナを復元した場合についても測定を行った．

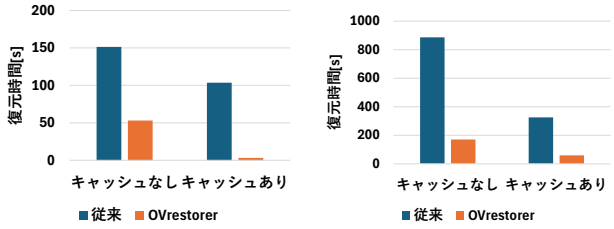
5.3.1　メモリサイズごとの復元性能

コンテナのメモリサイズが復元性能に与える影響を調べるために，コンテナ内の memcached が保持するデータ量を変えながら，コンテナ復元時間を測定した．VM を高負荷にする場合には VM 内で仮想ディスクに負荷をかけた．低負荷時のコンテナ復元時間を図 8a に，高負荷時のコンテナ復元時間を図 8b に示す．実験結果より，OVrestorer と従来ツールどちらの場合でもメモリサイズに比例して復元時間が長くなることが分かった．また，OVrestorer はメモリサイズが大きくなるほど，復元性能が改善できることが分かった．これは仮想化のオーバヘッドが削減できたことに加えて，Docker によるイメージファイルのコピーが削減できたためである．Docker はコンテナ内のプロセスを復元する際に状態が保存されたイメージファイルをコピーするため，この処理がディスクの負荷の影響を大きく受けた．OVrestorer はこのようなコピーを行わないため，大幅に性能が改善できたと考えられる．



(a) 低負荷時　　(b) 高負荷時

図 8: メモリサイズごとのコンテナ復元時間



(a) 低負荷時　　(b) 高負荷時

図 9: ファイルキャッシュがある場合のコンテナ復元時間

5.3.2　ファイルキャッシュがある場合の復元性能

プロセスを復元する際に用いるイメージファイルがファイルキャッシュが残っている場合のコンテナの復元性能について調べた．この場合には，コンテナ復元機構はディスクにアクセスせずにイメージファイルを読み込むことができる．コンテナ内の memcached には 5GB のデータを格納し，コンテナの復元にかかる時間を測定した．VM を高負荷にする場合には VM 内で仮想ディスクに負荷をかけた．低負荷時のコンテナ復元時間を図 9a に，高負荷時のコンテナ復元時間を図 9b に示す．実験結果より，低負荷時にはファイルキャッシュがあると 50 秒程度性能がよくなることが分かった．高負荷時も同様に，ファイルキャッシュがある方が性能が 65%程度よくなった．しかし，従来の Docker はイメージファイルをコピーするために復元にはまだ長い時間がかかった．

5.3.3　様々な負荷が復元性能に与える影響

VM 内で stress-ng を用いて様々な負荷をかけた場合に，コンテナを復元するのにかかる時間を測定した．コンテナ内の memcached には 5GB のデータを格納した．負荷の種類ごとにコンテナの復元にかかった時間を図 10 に示す．実験結果より，OVrestorer はディスクにかかる負荷の影響のみを受けることが分かった．従来ツールは I/O の負荷の影響も大きく受けたが，OVrestorer は影響を受けず，従来ツールよりも 93%高速に復元できることが分かった．これは，従来ツールの Docker によるイメージファイルのコピーが大きな影響を受けたためと考えられる．一方，OVrestorer と従来ツールはともに，CPU 負荷，パイプに対する負荷，メモリ負荷の影響をあまり受けないことも分かった．

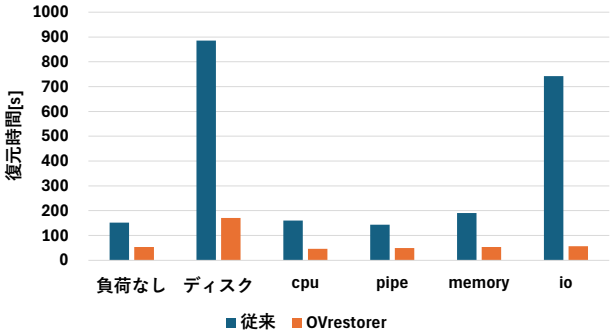


図 10: 様々な負荷をかけた時のコンテナ復元時間

6.　関連研究

コンテナマイグレーションはコンテナの状態の保存・復元と状態の転送を組み合わせたものであるが，その負荷を軽減する手法が提案されている．

OVmigrate [5] は移送元 VM の外でコンテナの状態を保存することを可能にしている．VM 外で動作するマイグレーション機構が VM イントロスペクションを用いて VM のメモリに格納されている状態を取得し，VM 外のネットワークを使って移送先 VM に転送する．これにより，VM 内での仮想化オーバヘッドや負荷の影響を受けずにコンテナの状態の保存・転送を行うことができる．しかし，OVmigrate はプロセスの状態の保存にしか対応できておらず，プロセスマイグレーションの最適化にとどまっている．また，移送先 VM での状態の復元は従来通り，VM 内で行う必要がある．

Portkey [7] は VM 内でコンテナの状態を転送するオーバヘッドを削減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ができる．OVrestorer は VM 外で保存されたコンテナの状態を VM 外で復元するため，VM 内で状態を転送する必要はない．

mWarp [13] 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 をチェックポイント・リストアに用いる際にはこの最適化を用いることはできないが，コンテナマイグレーションに応用する場合には利用可能であると考えられる．

攻撃からの復旧を行うためにコンテナのチェックポイント・リストアを用いるのではなく，攻撃の影響を VM 外からピンポイントに取り除く手法も提案されている．

EXTERIOR [14] は復旧対象 VM と同一の OS カーネルを動かす別の VM を用意し，その VM 内で実行したコマンドによるメモリ更新を対象 VM に反映する．この機構を用いて，`kill` コマンドを実行して不正なプロセスを強制終了させたり，`rmmod` コマンドを実行して不正なカーネルモジュールをアンロードしたりすることができる．しかし，復旧に用いる VM を追加で動かす必要があるため，システムのリソースへの影響が大きい．

VMMfas [15] はホストから VM のメモリを書き換えることによって OS データを変更し，攻撃からの復旧を行う．例えば，VM 外からシグナル関連のデータを書き換えることで不正なプロセスに `KILL` シグナルを送信して強制終了させることができる．その際に，一時停止中のプロセスであっても即座に終了させることができるように，VM 外から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する．OVrestorer でもこの手法を用いることにより，状態の復元が完了したプロセスに VM 外から `CONT` シグナルを送信して再開させ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 はコンテナ復元機構を VM 外で動作させ，VM 内にあるコンテナの状態を VM 外から復元することで，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処理に与える影響を最小化することができる．OVrestorer は VM のメモリ上にある OS のデータを解析して VM 内のプロセスの状態を書き換える．VM 外のコンテナ復元機構が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 内のコンテナ復元機構と連携してコンテナの復元を行う．実験結果より，OVrestorer は VM 内のコンテナ復元機構と協調実行することにより，コンテナを保存時の状態に復元できることが確認できた．また，OVrestorer は VM の負荷が高い時でも VM のディスクの帯域制限を少し行うだけで，既存ツールより高速にコンテナを復元できることが分かった．

今後の課題はコンテナのより多くの状態を VM 外から復元できるようにすることである．まだ VM 内で復元しているコンテナの状態として，名前空間やプロセス ID などの情報が挙げられる．

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] Asakura, Y. and Kourai, K.: An Efficient State-Saving Mechanism for Out-of-band Container Migration, *Proceedings of the 15th IEEE International Conference on Cloud Computing Technology and Science*, pp. 63–70 (2024).
[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] The CRIU Team: CRIU, `https://criu.org/Main_Page`.
[9] Google LLC: Protocol buffers, `https://protobuf.dev/`.
[10] 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).
[11] 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).
[12] Dormando: memcached - a distributed memory object caching system, `https://memcached.org/`.
[13] 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).
[14] Fu, Y. and Lin, Z.: EXTERIOR: Using a Dual-VM Based External Shell for Guest-OS Introspection, Configuration, and Recovery, *Proceedings of the 9th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, p. 97–110 (2013).
[15] Kimura, K. and Kourai, K.: Xfas: Fault Recovery by Externally Controlling OS Behavior, *Proceedings of the 16th IEEE/ACM International Conference on Utility and Cloud Computing*, No. 9 (2023).