

ライブラリ OS を用いた異種 TEE 間での マイグレーションに向けて

近藤 瑛¹ 光来 健一¹

概要: 近年、クラウドコンピューティングの普及により、クラウド上で機密情報を扱う機会が増えている。このような情報を保護するため、アプリケーションや仮想マシン (VM) を隔離実行環境 (TEE) 上で実行する機密コンピューティング技術が広く利用されている。一方で、クラウド環境ではホストの保守作業や障害対応などを目的として、実行中のアプリケーションや VM を別のホストへ移動させるマイグレーションが必要となる。同種 TEE 間でのマイグレーション手法についてはいくつか提案されているが、異種 TEE 間では TEE ごとに保護対象となる実行状態や実行環境が異なるため、既存の手法をそのまま適用することはできず、マイグレーションを行うことは困難である。本稿では、異種 TEE 間でのマイグレーションを可能にする機構である MigTEE を提案する。MigTEE は共通のライブラリ OS を用いてアプリケーションを実行し、移送元 TEE でアプリケーションの実行状態を保存した後、移送先 TEE でその状態を復元することによって異種 TEE 間でのマイグレーションを実現する。ライブラリ OS として Gramine を用い、MigTEE のマイグレーション機構を実装した。実験により、TEE を用いないアプリケーションについて MigTEE の動作を確認し、保存・復元の性能を調べた。

1. はじめに

近年、クラウドコンピューティングの普及に伴い、クラウド上で機密情報を扱う機会が増加している。一方で、クラウドはインターネットを介してハードウェア資源を提供する形態を取っているため、クラウド管理者などの悪意を持った攻撃者によってサービス内で管理されている機密情報が直接盗まれる危険性がある。このような情報漏洩を防ぐために、近年のクラウドでは CPU の隔離実行環境 (TEE) を用いてユーザのアプリケーションや仮想マシン (VM) を保護する機密コンピューティング技術が活用されている。TEE には、プロセス単位で保護を提供するプロセススペースの TEE と、VM 単位で保護を提供する VM ベースの TEE が存在する。これらを用いることで、信頼できないホスト OS やハイパーバイザからの不正なアクセスを防ぎ、アプリケーションや VM 内の機密情報を保護することができる。

一方、クラウド環境では、ホストの保守作業や障害対応などを目的として、実行中のアプリケーションや VM を別のホストへ移動する必要がある場合がある。このような場合においても、アプリケーションや VM の処理を停止することなく他のホスト上で継続して実行するためには、

マイグレーションを行うための機構が必要となる。同種の TEE 間ではこれまでにマイグレーションを実現する手法がいくつか提案されている。例えば、SGX のエンクレイヴ間のマイグレーションを実現する手法 [1], [2] や、TDX を用いた機密 VM 間のマイグレーションを実現する手法 [3] が提案されている。しかし、異なる種類の TEE 間ではそれぞれの TEE が提供する保護機構や実行環境が異なるため、マイグレーションの実現は困難である。

そこで本稿では、異なる種類の TEE 間でもマイグレーションを可能にする機構である MigTEE を提案する。MigTEE は、TEE 上で共通のライブラリ OS を用いてアプリケーションを実行することにより、TEE の種類に依存しない形でアプリケーションの状態を扱う。ライブラリ OS は汎用 OS を動作させることができないプロセススペースの TEE でも動作させることができ、VM ベースの TEE でも汎用 OS の代わり、または、汎用 OS の上で動作させることができる。移送元 TEE 内でライブラリ OS を介してアプリケーションの実行状態を保存し、その状態を移送先 TEE に転送して復元することによって、異種 TEE 間でのマイグレーションを実現する。

MigTEE では、マイグレーション開始時に移送元 TEE 上で動作するライブラリ OS が、アプリケーションおよび自身が管理する実行状態をチェックポイントとして保存

¹ 九州工業大学
Kyushu Institute of Technology

する。保存対象には、アプリケーションが利用するメモリ領域やプロセスおよびスレッドの実行状態、ファイルシステム情報などが含まれる。この保存処理はアプリケーションが状態保存システムコールを呼び出すことで実行され、取得した状態はファイルに保存する。その後、移送先ホスト上に新たな TEE を作成し、ライブラリ OS が起動時にチェックポイントから保存された状態を復元する。復元時には、保存されたメモリの内容やプロセスおよびスレッドの状態、実行環境に関する情報などを新しい TEE 上に設定する。

ライブラリ OS として Gramine [4] を用い、MigTEE のマイグレーション機構を実装した。MigTEE を用いて実験を行い、TEE を用いないアプリケーションの実行状態をチェックポイントとしてファイルに保存した後、多くのアプリケーションで保存した状態を復元できることを確認した。しかし、復元後にアプリケーションの実行を再開すると、アプリケーションの終了まで正常に動作を継続することはできなかった。また、状態の保存および復元にかかる時間とチェックポイントのファイルサイズを測定した結果、これらの値はアプリケーションによって大きく異なるが、測定値のばらつきは小さいことを確認した。

以下、2 章では TEE 間でのマイグレーションやその問題点について述べる。3 章では異種 TEE 間でのマイグレーションを可能にする MigTEE を提案する。4 章では MigTEE の実装について説明し、5 章では MigTEE を用いた状態保存・復元に関する実験について述べる。6 章で関連研究を示し、7 章で本稿をまとめる。

2. TEE マイグレーション

機密コンピューティングは、CPU の隔離実行環境 (TEE) を用いてアプリケーションや仮想マシン (VM) をクラウドから保護するための技術である。TEE には図 1 のように、プロセス単位で保護を提供するプロセススペースの TEE と、VM 単位で保護を提供する VM ベースの TEE が存在する。プロセススペースの TEE の代表例である Intel SGX は、アプリケーション内部にエンクレイヴと呼ばれる保護領域を作成し、エンクレイヴ内のコードやデータが格納されているメモリを CPU が暗号化し、整合性の検査を行う。これにより、信頼できないホスト OS やハイパーバイザからエンクレイヴ内の機密情報への不正なアクセスを防止し、アプリケーションの機密領域を保護することができる。

VM ベースの TEE の代表例である Intel TDX は、Trust Domain (TD) と呼ばれる機密 VM を提供することで VM 全体を保護する。機密 VM のメモリは CPU によって暗号化され、整合性が保証されるため、信頼できないハイパーバイザなどの特権ソフトウェアからの不正なアクセスを防止することができる。このように、プロセススペースの TEE はアプリケーション全体もしくはその一部のみを保護対象

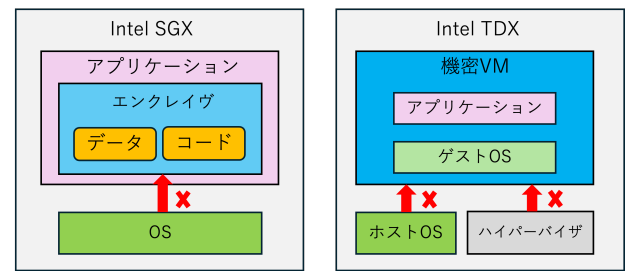


図 1: TEE のシステム構成

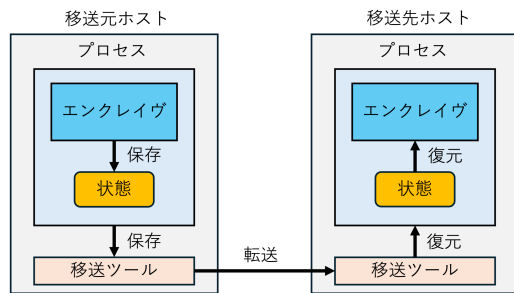
とするのに対し、VM ベースの TEE は OS を含めて VM 全体を保護対象とするというように、両者は保護方式が大きく異なっている。

一方、クラウド環境では、ホストの保守作業や障害対応、負荷分散などを目的として、実行中のアプリケーションや VM を別のホストへ移動させる必要が生じる場合がある。このような場合、アプリケーションや VM を停止して移送先ホストで再起動すると、サービスが中断され可用性が低下する。そのため、アプリケーションや VM の実行状態を移送先ホストへ引き継ぎ、サービスを継続したまま実行環境を移動させるマイグレーション機構が必要となる。マイグレーションを行うことで、ホストの保守作業や障害対応時にサービスを停止する必要がなくなるため、システムを柔軟に運用することが可能となる。

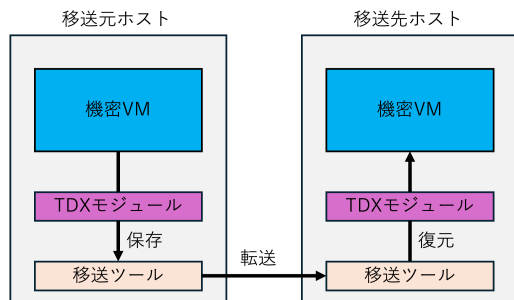
同じ種類の TEE 間では、図 2 に示すように、マイグレーションを実現するための手法が提案されている。例えば、SGX のエンクレイヴ間のマイグレーションを実現する手法 [1], [2] では、移送元のエンクレイヴ自身が CPU に依存しない鍵を用いて自身の状態を暗号化した上で外部のメモリへ書き出すことで状態を保存し、移送先のエンクレイヴで同じ鍵を用いて状態を復号・上書きすることで状態の復元を行う。また、TDX を用いた機密 VM 間のマイグレーションを実現する手法 [3] では、TDX モジュールに登録したマイグレーション鍵を用いて機密 VM の実行状態を暗号化して転送し、移送先で復号して状態を復元する。このように同じ種類の TEE 間では、各 TEE が提供する保護機構に対応した方法で実行状態を保存し、移送先 TEE へ転送して復元することができている。

しかし、必ずしも移送先ホストで移送元ホストと同種の TEE が利用できるとは限らない。例えば、同じ Intel サーバでも CPU が SGX はサポートしているが TDX をサポートしていないということがあり得る。CPU が TDX をサポートしている場合は SGX もサポートされているが、クラウドがサービスとして機密 VM しか提供していないこともあり得る。このような場合には、SGX を利用することはできない。また、移送先ホストとして AMD サーバしか空いておらず、SEV-SNP を用いた機密 VM しか利用できないということも考えられる。

このような場合、異なる種類の TEE 間では既存のマイグ



(a) エンクレイヴ間のマイグレーション



(b) 機密 VM 間のマイグレーション

図 2: 同種 TEE 間のマイグレーション

レーション手法を適用することはできない。例えば、SGX アプリケーションを機密 VM にマイグレーションして、機密 VM 内でそのまま動作させることはできない。通常の VM では仮想 SGX を利用して SGX アプリケーションを動作させることができるが、TDX を用いた機密 VM 内では仮想 SGX を利用することはできない。SGX アプリケーションを機密 VM 内で動かせたとしても、TEE をネストさせることになりオーバーヘッドが大きくなる。

逆に、機密 VM 内で実行されている一般のアプリケーションをマイグレーションして SGX エンクレイヴ内で動作させるのも容易ではない。SGX アプリケーションは専用の SGX SDK を用いて作成する必要があるためである。アプリケーションが動作している機密 VM 内のゲスト OS の状態を移送先のホスト OS に反映させるのも容易ではない。また、同じ機密 VM 間であっても、TDX を用いた機密 VM と SEV-SNP を用いた機密 VM ではマイグレーション手法が異なる。このように、TEE ごとに保護対象となる実行状態や実行環境が異なるため、異なる種類の TEE 間でマイグレーションを行うことは困難である。

3. MigTEE

本稿では、異なる種類の TEE 間でのマイグレーションを可能にする機構である MigTEE を提案する。MigTEE は、TEE 上で共通のライブラリ OS を用いてアプリケーションを実行し、ライブラリ OS が管理するアプリケーションの状態を保存・復元することでマイグレーションを実現する。ライブラリ OS は、OS の機能をライブラリとしてアプリケーションに直接提供する形態であるため、汎用 OS を

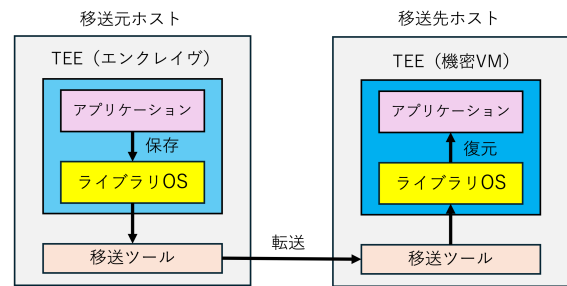


図 3: MigTEE のシステム構成

動作させることができないプロセススペースの TEE と、汎用 OS を動作させることができる VM ベースの TEE の両方で利用することが可能である。ライブラリ OS を利用することで、TEE の種類が異なる場合でも共通のライブラリ OS を介してアプリケーションの状態を扱うことができる。

MigTEE のシステム構成を図 3 に示す。MigTEE では、移送元 TEE 内で動作するライブラリ OS が、実行中のアプリケーションの状態およびライブラリ OS が管理する状態のみを取得し、取得した状態をチェックポイントとして保存する。保存されたチェックポイントは移送先ホストへ転送される。その後、移送先ホスト上に新たな TEE を作成し、その内部でライブラリ OS を起動する。ライブラリ OS は、移送元 TEE で保存されたチェックポイントを読み込み、保存されている状態を移送先 TEE 上に復元する。TEE に依存しないようにするために、アプリケーションとライブラリ OS 以外の TEE ごとに特有の状態は保存・復元を行わない。

例えば、プロセス単位で保護を行う TEE である Intel SGX では、エンクレイヴ内でライブラリ OS およびアプリケーションを動作させる。マイグレーションの際には、移送先ホスト上に新たなエンクレイヴを作成し、その内部でライブラリ OS を起動することで状態を復元する。一方、VM 単位で保護を行う TEE である Intel TDX では、機密 VM 内でライブラリ OS およびアプリケーションを動作させる。マイグレーションの際には、移送先ホスト上に新たな機密 VM を作成し、その内部でライブラリ OS を起動することで状態を復元する。また、移送先ホストにおいて機密 VM 内のゲスト OS 上にプロセスを作成してライブラリ OS を起動し、チェックポイントを読み込むことで状態を復元することも可能である。

移送元 TEE では、マイグレーション開始時にアプリケーションが状態保存システムコールを実行することで、状態を保存する。システムコールが実行されると、ライブラリ OS は実行中のアプリケーションが使用しているメモリ領域やプロセスの実行時情報、ファイルシステム情報、システム情報などを取得し、チェックポイントとしてファイルに保存する。メモリ情報としては、各メモリ領域の情報に加えて、実際に格納されているメモリデータを保存する。

また、プロセス実行時情報としてプロセス ID や実行環境に関する情報、ファイルシステム情報としてルートディレクトリやマウント情報、システム情報として CPU およびメモリに関する情報や DNS 設定などを保存する。

移送先 TEE では、移送元 TEE で保存された状態をもとにアプリケーションとライブラリ OS の状態を復元する。まず、移送先ホスト上に新たな TEE を作成し、移送元 TEE で保存されたチェックポイントファイルを読み込むことで保存されている状態を取得する。次に、移送元 TEE と同じメモリ領域を移送先 TEE 上に確保し、確保した領域内にメモリデータを書き込むことでアプリケーションのメモリ状態を復元する。さらに、保存されていたプロセス実行時情報、ファイルシステム情報、システム情報などをプロセスに設定することで、アプリケーションのすべての実行状態を復元する。

4. 実装

ライブラリ OS の Gramine [4] を拡張することで MigTEE を実装した。現在の実装はまだ TEE に対応しておらず、Linux 上のプロセスとしてライブラリ OS とアプリケーションを実行する。

4.1 Gramine

Gramine はライブラリ OS と Platform Adaptation Layer (PAL) から成る。ライブラリ OS はアプリケーションに対して Linux のシステムコールインタフェースを提供し、プロセス、スレッド、仮想メモリ、ファイルシステムなどの状態を管理する役割を持つ。アプリケーションから実行されるシステムコールはライブラリ OS によって処理され、処理内容に応じて PAL を経由して実行環境の機能呼び出す。

PAL はライブラリ OS から利用される低レベルな機能を実行環境に応じて提供する役割を持つ。例えば、プロセスの生成、メモリ管理、ファイルやストリームの操作など、ライブラリ OS がホスト OS の機能を間接的に利用するために必要となる処理を PAL が行っている。PAL により、実行環境の違いを意識することなくライブラリ OS を動作させることができ、様々な TEE に容易に対応することができる。現在のところ、Gramine は SGX と TDX に対応している [5], [6]。

4.2 状態の保存

状態の保存処理は、Gramine の fork システムコールにおいて親プロセスの状態を取得する処理を利用して実装した。

4.2.1 状態保存の流れ

MigTEE における、アプリケーションの状態保存の流れを図 4 に示す。MigTEE では、アプリケーションが save システムコールを実行することで、ライブラリ OS 内部の

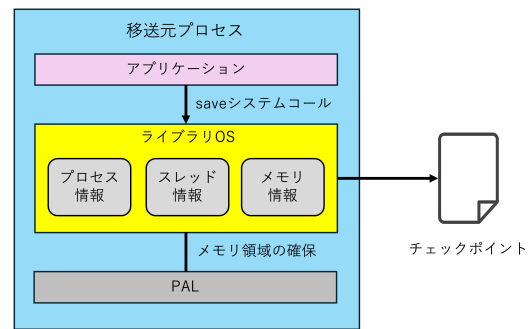


図 4: MigTEE における保存処理

状態保存処理を呼び出す。例えば、マイグレーション時にアプリケーションにシグナルを送信して save システムコールを呼び出させる方法が考えられる。ライブラリ OS はまず、チェックポイントデータを一時的に保持するための連続したメモリ領域を PAL を介して確保する。次に、確保したメモリ領域上にアプリケーションの状態を書き込むことで、チェックポイントデータを作成する。保存する状態としては、プロセス情報、スレッド情報、メモリ情報に加えて、ルートディレクトリやマウント情報などのファイルシステム情報、DNS 設定などのシステム情報、オープン中のファイルを扱うためのハンドル情報などがある。

その後、open システムコールの実装を用いてチェックポイントファイルを作成する。まず、作成したファイルにチェックポイントデータの先頭アドレスやサイズなどの管理情報をヘッダとして書き込む。次に、チェックポイントデータ本体を書き込み、最後にメモリデータを書き込む。fork システムコールでは、チェックポイントは親プロセスと子プロセスの間で共有される Unix ドメインソケットに書き込まれる。しかし、一般に移送元ホストと移送先ホストが異なるマイグレーションでは Unix ドメインソケットは利用することができない。移送元プロセスと移送先プロセスの間でネットワーク接続を確立し、ネットワークソケットに書き込むことが考えられるが、その実装は今後の課題である。

4.2.2 プロセス・スレッド情報の保存

プロセスの状態として、プロセス ID、親プロセス ID、プロセスグループ ID、セッション ID を保存する。また、ルートディレクトリおよびカレントディレクトリからなるディレクトリ情報、ファイルのアクセス権に関する設定値、実行プログラムに関する情報についても保存する。これらの情報は、移送先で新たに作成されるプロセスに対して、移送元と同一のプロセス識別情報および実行環境を再現するために必要となる。

スレッドの状態として、スレッド ID、ユーザおよびグループに関する情報、ファイルディスクリプタ情報、シグナル管理情報、スタック情報、CPU アフィニティ、参照カウントやロックなどのスレッド管理情報、スレッドロー

カルストレージのベースアドレス、スケジューリング情報を保存する。プロセスは複数のスレッドから構成される場合があり、各スレッドは独立した実行状態を持つ。そのため、各スレッドが持つ状態を個別に保存する必要があるが、現在の実装では fork システムコールの場合と同様に、save システムコールを呼び出したスレッドの状態のみが保存できている。

4.2.3 プロセスメモリの保存

プロセスのメモリ情報として、プロセスが使用している各メモリ領域について、先頭アドレスおよびサイズを保存するとともに、実際に割り当てられているメモリデータを取得して保存する。このとき、読み取りが許可されていないメモリ領域については、データの取得に先立って PAL を介して一時的に読み取りを許可する設定に変更し、データ取得後に元の設定へ戻す。一部のメモリ領域については、メモリデータを保存せずにメモリ領域に関する情報のみを記録する。

4.3 状態の復元

状態の復元処理は、Gramine の fork システムコールにおいて子プロセスに親プロセスの状態を設定する処理を利用して実装した。Gramine は PAL を起動する際に実行形態を指定し、一から実行されるプロセスか、子プロセスとして実行されるプロセスかによって異なる起動処理を行う。MigTEE では、PAL を起動する際に移送先で復元されるプロセスかも指定できるようにして、これらの実行形態とは異なる起動処理を行う。

4.3.1 状態復元の流れ

MigTEE における、アプリケーションの状態復元の流れを図 5 に示す。まず、PAL が指定されたチェックポイントファイルをオープンし、ライブラリ OS を起動する。ライブラリ OS は PAL を介してチェックポイントファイルからチェックポイントデータとメモリデータを読み込む。ライブラリ OS は読み込んだチェックポイントデータを基に、プロセス実行時情報、ファイルシステム情報、システム情報などを復元する。また、ライブラリ OS は PAL を介して状態保存時と同じメモリ領域を確保し、確保されたメモリ領域に読み込んだメモリデータを書き込む。最後に、状態保存時に呼び出された save システムコールから復帰することによってアプリケーションの実行を再開する。

4.3.2 チェックポイントの読み込み

fork システムコールでは、子プロセスは親プロセスから Unix ドメインソケット経由でチェックポイントヘッダおよびチェックポイントデータを受信する。MigTEE でもネットワークソケット経由で受信することは可能であると考えられるが、現在の実装ではチェックポイントファイルから読み込むことによって受信している。チェックポイントファイルをオープンした後は、ソケットからの読み込み

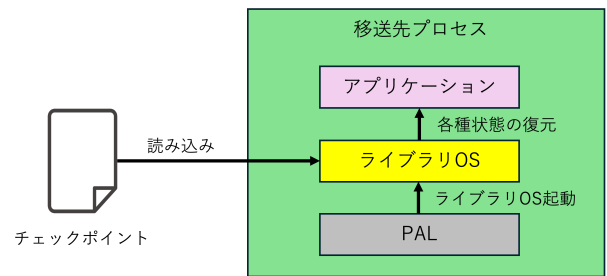


図 5: MigTEE における復元処理

と共通の処理を用いることができる。

ライブラリ OS はチェックポイントファイルを読み込むために直接、PAL のストリーム API を呼び出す。状態保存時とは異なり、状態復元時にはライブラリ OS のファイル API をまだ利用することができないためである。しかし、既存の読み込み処理はソケットからの読み込みを想定して常にオフセットに 0 が指定されているため、ファイルからの読み込みの場合には常に先頭から読み込まれてしまう。そこで、読み込み対象がシーク可能であるというフラグを設定することで、オフセットを指定せずにファイルから順番に読み込まれるようにする。

4.3.3 プロセス・スレッドの復元

チェックポイントに保存されたプロセス情報およびスレッド情報は、移送先プロセス内のライブラリ OS に設定する。プロセス情報としては、プロセス ID、親プロセス ID、ディレクトリ情報などを復元する。スレッド情報としては、ユーザおよび権限情報、ファイルディスクリプタ情報、スタック情報などを復元する。移送先ホストの CPU トポロジ情報は移送元ホストとは一般に異なるため、移送先ホストで proc ファイルシステムから再取得する。

4.3.4 プロセスメモリの復元

チェックポイントファイルに保存されているメモリ領域には、メモリデータが保存されている領域と、メモリデータは保存されておらず、メモリ領域が存在するという情報のみが記録されている領域がある。前者のメモリ領域については、ライブラリ OS は PAL を介してメモリ領域を確保し、チェックポイントファイルから読み出したメモリデータを書き込む。書き込み先のメモリ領域が読み込み専用である場合には、ライブラリ OS はデータの書き戻しに先立って、PAL を介して一時的に書き込みを許可する設定を行い、書き込み後に元の設定へ戻す。後者のメモリデータが保存されていないメモリ領域については、実際のメモリ確保は行わず、対応するアドレス範囲をアドレス空間上に予約する。

4.3.5 ハンドルの復元

ライブラリ OS はハンドルを用いてオープン中のファイルやコンソールなどを管理している。ハンドルの実体は PAL によって生成される PAL ハンドルであり、PAL ハンドルの実体はファイルディスクリプタである。fork システ

表 1: 実験環境

ホスト	
CPU	Intel Xeon Gold 6526Y
メモリ	128 GB
OS	Linux 6.8.0-rc5+
ライブラリ OS	Gramine 1.9

ムコールでは、親プロセスのファイルディスクリプタを子プロセスがそのまま利用することができるため、特別なハンドルの復元処理を行う必要がない。しかし、マイグレーションの場合には、移送元プロセスと移送先プロセスの間に親子関係がないため、ファイルディスクリプタを引き継ぐことはできない。また、一般的にはホストも異なるため、Unix ドメインソケットなどを用いてファイルディスクリプタパッシングを行い、ファイルディスクリプタを送信することもできない。

そのため、MigTEE はハンドルを再生成することによって復元する。チェックポイントに保存されたハンドル情報に含まれるハンドルの種類、アクセスモード、URI を用いて対象を再オープンし、移送先プロセスで利用できる PAL ハンドルを生成する。コンソールを扱うためのハンドルについては、標準入出力をファイルディスクリプタとしてもつ PAL ハンドルを再生成する。ファイルがマッピングされているメモリ領域を復元する際には、再生成した PAL ハンドルを用いてファイルを再マッピングする。

5. 実験

MigTEE を用いて、実行中のアプリケーションの状態が正しく保存・復元できるかどうかを確認する実験を行った。次に、状態の保存および復元にかかる時間を測定した。また、チェックポイントとして状態が保存されるファイルのサイズについても測定した。本実験は、表 1 に示すホストを用いて行った。ライブラリ OS には、Intel SGX および Intel TDX の両方に対応している Gramine を用いた。ただし、MigTEE は TEE にまだ対応できていないため、本実験では TEE を使用せず、通常の Linux 上で Gramine を用いてアプリケーションを動作させた。

本実験では、helloworld, lighttpd [7], nginx [8], memcached [9] の 4 種類のアプリケーションを用いた。helloworld は文字列を表示するだけのシンプルなアプリケーションである。それぞれのアプリケーションは初期化完了後に save システムコールを実行してチェックポイントを作成するようにした。

5.1 状態の保存・復元

まず、アプリケーションに save システムコールを呼び出させることで状態保存処理を実行し、実行中のアプリケーションの状態をチェックポイントとしてファイルに保存

した。ライブラリ OS のログを確認した結果、アプリケーションのメモリ領域やプロセス情報などの状態がチェックポイントとしてファイルに保存されていることが確認できた。

次に、状態復元処理を実行し、保存したチェックポイントからアプリケーションの状態を復元した。その結果、チェックポイントファイルのヘッダおよびチェックポイントデータ本体の読み込み、保存されたメモリ領域へのデータの書き戻し、プロセスやスレッド情報などの各種状態の復元が順に行われ、その後は PAL ハンドルを除いたハンドル情報の復元が行われることを確認した。復元処理に必要なとなる PAL ハンドルについては、チェックポイントに保存された URI やアクセスモードなどのハンドル情報を用いて、それに対応する PAL ハンドルが移送先環境で新たに生成されることを確認した。

helloworld については、これらの復元処理の完了後、アプリケーションの実行を再開できることを確認した。しかし、実行を終了する段階でエラーが発生した。具体的には、プロセス終了の通知に失敗したことを示すエラーや、プロセス ID の管理に関する不整合を示すエラーが発生し、最終的に Gramine 内部で不正な命令が検出され、プロセスが終了した。

lighttpd については、復元処理の完了後、アプリケーションの実行再開に先立つソケット処理においてエラーが発生した。具体的には、ソケットの受け付け処理において必要な PAL ハンドルが正しく設定されておらず、NULL の PAL ハンドルが PAL のソケット受け付け処理に渡されたことでメモリアクセスエラーが発生し、プロセスが終了した。

nginx については、復元処理の完了後、アプリケーションの実行を再開できることを確認した。しかし、アプリケーションの実行再開後に処理が停止し、正常に動作を継続できなかった。

memcached については、チェックポイントに保存された状態の復元処理を進めることができたが、アプリケーションの実行再開には至らなかった。具体的には、ファイルディスクリプタやファイルシステムなどのハンドル情報を復元した後、パイプの復元処理内でエラーが発生し、復元処理が中断された。

5.2 状態保存性能

MigTEE を用いて、実行中のアプリケーションの状態保存にかかる時間の測定を行った。save システムコールの実行前後の時刻の差分を保存時間とした。4 種類のアプリケーションに対して 10 回ずつ保存時間を測定した。保存時間の平均値と標準偏差を図 6 に示す。保存時間はアプリケーションによって大きく異なることが確認できた。特に、memcached は状態の保存にかかる時間が最も長くなって

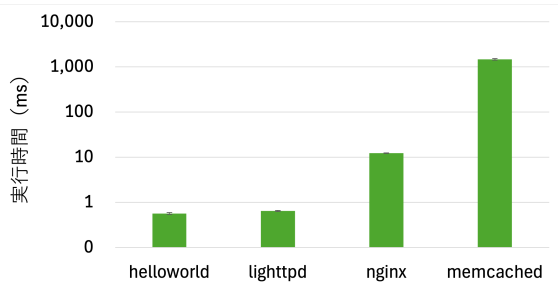


図 6: 状態保存にかかる時間

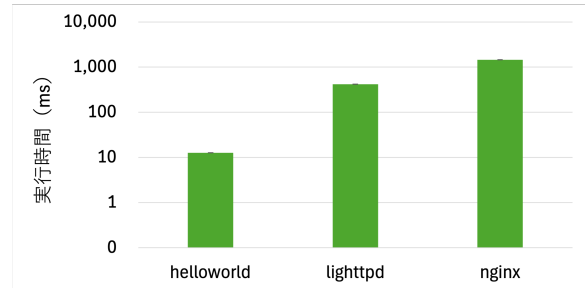


図 8: 状態復元にかかる時間

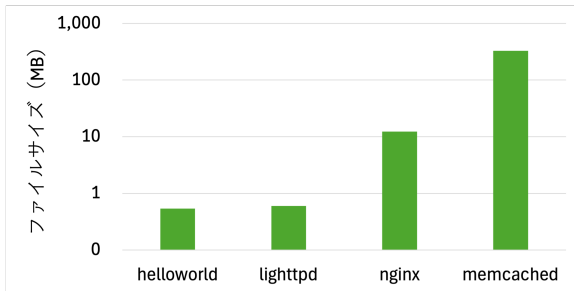


図 7: チェックポイントファイルのサイズ

おり、helloworld の 2,580 倍に達した。これは memcached が大量のメモリデータを保持しており、保存するメモリ領域が大きいためであると考えられる。保存処理ではメモリの読み出しやアクセス権限の変更などを行うため、対象となるメモリ領域が大きいアプリケーションほど、保存にかかる時間が長くなると考えられる。また、平均値に対する標準偏差の割合（変動係数）はいずれのアプリケーションも 5%以内に収まっており、保存時間のばらつきは小さいことが確認できた。

次に、チェックポイントとして状態が保存されたファイルのサイズを測定した。チェックポイントファイルのサイズを図 7 に示す。状態の保存時間と同様に、ファイルサイズはアプリケーションによって大きく異なることが確認できた。保存時間が長いアプリケーションほど、チェックポイントとして保存されるファイルサイズも大きくなる傾向が見られた。これは、保存するメモリデータの量が多いほど、そのデータをチェックポイントとして保存するために必要な時間とファイルサイズがともに増加するためであると考えられる。

5.3 状態復元性能

MigTEE を用いて、アプリケーションの状態復元にかかる時間の測定を行った。復元処理の開始時刻と復元処理が完了してアプリケーションの実行が再開される直前の時刻の差分を復元時間とした。これは save システムコールから復帰するための最終的な処理で lighttpd と nginx の復元が失敗するためである。復元の途中で失敗する memcached を除く、3 種類のアプリケーションに対して 10 回ずつ復元時間を測定した。復元時間の平均値と標準偏差を図 8 に

示す。

保存時間と同様に、復元時間もアプリケーションによって大きく異なることが確認できた。特に、nginx は復元時間が最も長くなっており、helloworld の 115 倍に達した。これは、復元対象となるメモリデータの量が多いアプリケーションほど、ライブラリ OS のメモリ領域への書き戻しにかかる時間が長くなるためであると考えられる。平均値に対する標準偏差の割合はいずれのアプリケーションも 1%以内に収まっており、復元時間のばらつきは小さいことが確認できた。

保存時間と復元時間を比較すると、いずれのアプリケーションにおいても復元時間の方が保存時間を上回っていることが確認できた。これは、復元処理がチェックポイントファイルの読み込み、メモリ領域の割り当て、各種情報の設定など、保存処理よりも多くの処理で構成されているためであると考えられる。

6. 関連研究

FUTEE [10] は、プロセススペースの TEE と VM ベースの TEE を横断的に統一して扱うことを目的としたフレームワークである。TEE の利用を容易にするフレームワークはこれまでも提案されているが、対応するアーキテクチャが限定されていたり重要な機能が不足しているといった問題点がある。これに対し FUTEE は、アテストーション、メモリ管理、通信といった各 TEE に共通する基本機能を抽象化することで、ハードウェアに依存しない共通インタフェースを提供する。FUTEE は、形式検証済みの Rust 製カーネルである Asterinas 上に構築される。FUTEE のユースケースとしては、異種ハードウェア環境における統一的なアプリケーション開発や、ベンダに依存しない機密 VM のマイグレーションなどが挙げられている。MigTEE とは異なり、機密 VM 内ではライブラリ OS ではなく、FUTEE カーネルモジュールを組み込んだ汎用 OS が動作する。

Gu らは信頼できないクラウド環境において、SGX エンクレイブを含む VM の安全なライブマイグレーション機構を提案している [1]。この機構では、各エンクレイブ内に制御スレッドを配置し、エンクレイブ内部から実行状態

を安全に保存できるようにしている。移送元エンクレイヴは実行状態を暗号化して外部メモリに保存してネットワーク経由で移送先へ転送する。移送先では新たなエンクレイヴを作成した後、受信した状態を復元することで実行を再開する。このとき、状態の保存および復元はすべてエンクレイヴ内部で実行されるため、機密性を維持したままマイグレーションを行うことができる。VMのマイグレーションを対象としているが、TEEとしてはSGXエンクレイヴのみに対応しているため、機密VMとの間でのマイグレーションは行えない。

MigSGX [2] は、SGXエンクレイヴを含むアプリケーションが動作するコンテナをマイグレーション可能にする機構である。この機構もGuらの機構と同様に、移送元のエンクレイヴがCPUに依存しない鍵を用いて自身の状態を暗号化し、暗号化された状態を外部メモリへ書き出す。その後、移送先において同一の鍵を用いて状態を復号し、移送先エンクレイヴのメモリへ復元することでマイグレーションを実現する。この機構はコンテナのマイグレーションを対象にしているが、TEEとしてはSGXエンクレイヴのみに対応している。

TEEndor [11] は、ハードウェアセキュリティモジュール(HSM)を利用することで、SGXエンクレイヴを安全にマイグレーションするための機構である。HSMは、暗号鍵の保管および暗号化処理を行う物理デバイスである。この機構では、移送元エンクレイヴの状態データをHSMへ転送して暗号化し、暗号化されたデータをネットワーク経由で移送先へ送信する。その後、移送先では受信したデータをHSMで復号した後、復号した情報を用いてエンクレイヴの状態を復元することでマイグレーションを実現する。

eMotion [12] は、実行中のSGXエンクレイヴを安全にマイグレーションするために、SGXへ追加命令およびマイグレーション支援機能を導入した拡張機能である。この機構では、異なる物理ホスト間でエンクレイヴのマイグレーションに用いる鍵(MMK)を安全に確立することで、ホスト上のVMMがその鍵を利用して実行中のエンクレイヴを移送できるようにしている。これを実現するため、MMKをSGX対応プロセッサに保存する命令(EPUTKEY)、エンクレイヴページを安全に退避する命令(ESE)、移送先でエンクレイヴページを安全に復元する命令(ESL)の3種類のSGX命令を新たに追加している。

MigTD [3] は、Intel TDXにおいて機密VM内で実行されるアプリケーションをマイグレーションするための機構である。この機構では、移送元および移送先の両方でMigTDと呼ばれるTDを起動し、相互認証を行うことで信頼できるホストであることを検証する。その後、MigTDはマイグレーション鍵を生成してTDXモジュールへ登録する。移送元では、機密VMの実行状態を事前にTDXモジュールに登録したマイグレーション鍵を用いて暗号化

し、これを移送先へ転送する。移送先では、受信したデータを復号・復元することで、TDX間でのマイグレーションを実現する。

AMD SEV環境におけるマイグレーション [13] は、VM内のメモリを暗号化したまま安全に移送先ホストへ移送するための仕組みである。SEVおよびSEV-ESでは、VMのマイグレーションはゲスト所有者が設定したポリシーに基づいて制御される。一方、SEV-SNPではMigration Agent(MA)を導入することで、より柔軟なマイグレーション管理を実現している。MAは、対象VMと同一の物理ホスト上で、SEV-SNPによって保護されたVMとして動作する。また、移送元と移送先のVMを同時に実行する必要はなく、一度VMを停止して実行状態をMAへ退避・保存し、保存された状態を用いて同じホストまたは別のホスト上でVMを再構築して実行を再開することができる。

7. まとめ

本稿では、TEE内で共通のライブラリOSを用いることで、異なる種類のTEE間においてもマイグレーションを可能にするMigTEEを提案した。MigTEEは、移送元TEE内でライブラリOSがアプリケーションの状態を保存し、その状態を移送先TEEに転送して復元することによって、実行中のアプリケーションを中断することなく異なる種類のTEE間で移動させることを可能にする。ライブラリOSとしてGramineを用い、MigTEEのマイグレーション機構を実装した。実験により、MigTEEを用いてアプリケーションの実行状態を保存・復元できることを確認したが、一部の状態は正常に復元されなかった。状態の保存および復元にかかる時間とチェックポイントのファイルサイズはアプリケーションによって大きく異なるが、測定値のばらつきは小さいことを確認した。

今後の課題として、移送元TEEで保存したアプリケーションの状態をもとに、移送先TEEで様々なアプリケーションの復元ができるようにする必要がある。そのために、PALハンドルの情報が移送先で正常に復元できるようにする。また、SGXやTDXといったTEE上でもMigTEEを動作させ、アプリケーションの状態の保存・復元を行えるようにする予定である。

謝辞 本研究は、JST 経済安全保障重要技術育成プログラム【JPMJKP24U4】の支援を受けたものである。

参考文献

- [1] Gu, J., Hua, Z., Xia, Y., Chen, H., Zang, B., Guan, H. and Li, J.: Secure Live Migration of SGX Enclaves on Untrusted Cloud, *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (2017).
- [2] Nakashima, K. and Kourai, K.: MigSGX: A Migration Mechanism for Containers Including SGX Applications,

- Proceedings of the 14th IEEE International Conference on Cloud Computing (UCC)* (2021).
- [3] Intel Inc.: Intel TDX Migration TD Design Guide (2023).
 - [4] Gramine Project: Gramine, <https://gramineproject.io>.
 - [5] Tsai, C.-C., Porter, D. E. and Vij, M.: Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX, *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pp. 645–658 (2017).
 - [6] Kuvaiskii, D., Stavrakakis, D., Qin, K., Xing, C., Bhatotia, P. and Vij, M.: Gramine-TDX: A Lightweight OS Kernel for Confidential VMs, *CCS 2024 - Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security*, pp. 4598–4612 (2024).
 - [7] lighttpd Project: lighttpd, <https://www.lighttpd.net>.
 - [8] NGINX: nginx, <https://nginx.org>.
 - [9] Memcached Project: Memcached, <https://memcached.org>.
 - [10] Prevost, T., Aublin, P.-L., Tazaki, H. and Suzaki, K.: A Case for Unifying Trusted Execution Environments , *2026 56th Annual IEEE International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pp. 180–183 (2026).
 - [11] Guerreiro, J., Moura, R. and Silva, J. N.: TEnder: SGX enclave migration using HSMs, *Computers & Security* (2020).
 - [12] Park, J., Park, S., Kang, B. B. and Kim, K.: eMotion: An SGX extension for migrating enclaves, *Computers & Security* (2019).
 - [13] AMD Inc. : AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More (2020).