複数ホストにまたがるVMのメモリデータ保護の最適化手法

堀尾 周平[1]　高橋 孝汰[1]　光来 健一[1]

概要：仮想マシン（VM）はマイグレーションにより別のホストへ移動させることができるが，大容量メモリを持つ VM のマイグレーション時には十分なメモリを持つ移送先ホストを常に確保できるとは限らない．そこで，VM のメモリを分割して複数の小さなホストに転送する分割マイグレーションが提案されている．分割マイグレーション後には，VM はメモリデータをメインホストとサブホスト間で交換するリモートページングを行いながら動作する．しかし，実行環境によってはネットワーク転送時などに VM のメモリデータを盗聴・改竄される危険性がある．メモリデータは暗号化や整合性検査によって保護することができるが，そのオーバヘッドにより性能低下は避けられない．本稿では，分割マイグレーションとリモートページングにおいてメモリデータの暗号化と整合性検査を最適化する *SEmigrate* を提案する．SEmigrate はサブホストにおいてメモリデータの復号と整合性検査を行わないようにすることでデータ保護のオーバヘッドを削減する．また，必要に応じてメモリデータの選択的な暗号化・整合性検査を行うことでもデータ保護のオーバヘッドを削減する．そのために，SEmigrate は VM のメモリを解析し，OS 内のメモリ属性やプロセス情報，さらにはアプリケーション内の情報を利用する．SEmigrate を KVM に実装し，分割マイグレーション時間を 43%，マイグレーション後の VM 内でのベンチマーク実行時間を 19%削減できた．

1. はじめに

近年，大容量メモリを持つ仮想マシン（VM）が利用されるようになっている．例えば，Amazon EC2 では 24TB のメモリを持つ VM が提供されている．VM はホストのメンテナンス等の際にマイグレーションを行うことによって別のホストへ移動させることができ，サービスを提供し続けることができる．しかし，大容量メモリを持つ VM の場合には，十分なメモリを持つホストを移送先として用意するのはコストの面で負担が大きい．また，そのようなホストが用意できる場合でも他の VM が実行されているとマイグレーションが行えないため，運用の柔軟性が低下する．

そこで，VM の大容量メモリを分割して複数の小さなホストに転送する分割マイグレーション [1][2] が提案されている．分割マイグレーションはメインホストに今後アクセスされそうなメモリデータと仮想 CPU などの VM コアの状態を転送する．サブホストにはそれ以外のメモリを転送する．分割マイグレーション後は VM コアがメインホスト上で動作し，サブホストはその VM にメモリを提供する．VM がサブホスト上のメモリにアクセスした時にはリモートページングを行い，サブホストに存在するメモリデータをメインホストへ転送（ページイン）する．代わりに，メインホスト上の不要なメモリデータをサブホストへ転送（ページアウト）する．

しかし，実行環境によっては分割マイグレーションやリモートページングの際に VM のメモリデータを盗聴されたり改竄されたりする危険性がある．例えば，メモリデータが十分に安全ではないネットワーク経由で転送される場合には攻撃を受ける恐れがある．また，ホスト上にあるメモリデータが攻撃されることも考えられる．このような攻撃を防ぐためにはメモリデータの暗号化および整合性検査を行えばよいが，SSL/TLS 等の暗号通信を用いるとメモリデータを転送するたびに暗号化・復号化と整合性検査の処理が行われるため，性能低下が避けられない．また，暗号化や整合性検査が必要かどうかは考慮されず，すべてのメモリデータに対して一律に保護が行われる．

本稿では，分割マイグレーションとリモートページングにおいてメモリデータの暗号化と整合性検査を最適化する *SEmigrate* を提案する．SEmigrate はサブホストにおいてメモリデータの復号と整合性検査を行わないようにすることでデータ保護のオーバヘッドを削減する．メモリデータを暗号化されたままで保持することで，サブホストにおける情報漏洩を完全に防ぐこともできる．分割マイグレーション時には移送元ホストで暗号化したメモリデータを移送先メインホストでのみ復号して整合性検査を行う．リモートページング時にもメインホストでのみメモリデータの暗号

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

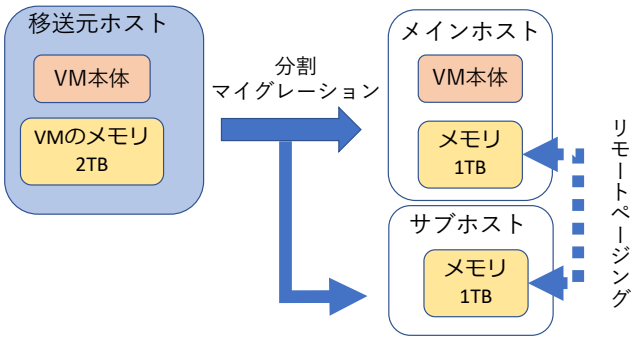

図 1 分割マイグレーションとリモートページング

化・復号化および整合性検査を行う．また，SEmigrate は必要に応じてメモリデータの選択的な暗号化・整合性検査を行うことでもデータ保護のオーバヘッドを削減する．そのために，VM イントロスペクション [3] を用いて VM 内のゲスト OS のメモリを解析し，メモリ属性やプロセス情報を利用する．さらに，VM 内のアプリケーションのメモリも解析して，アプリケーション固有の情報も利用する．

我々は分割マイグレーションおよびリモートページングをサポートした KVM に SEmigrate を実装した．SEmigrate による性能向上を確認するために，VM 内で大量のメモリを使用するアプリケーションを実行して分割マイグレーションとマイグレーション後の VM の性能を調べた．実験の結果，メモリデータの保護を行う場合に分割マイグレーションにかかる時間を 43%削減できた．また，マイグレーション後の VM 内でのベンチマーク実行時間を 19%削減できた．

以下，2 章では VM の分割マイグレーションとリモートページングにおけるメモリデータの保護の問題点について述べる．3 章ではメモリデータの暗号化と整合性検査を最適化する SEmigrate を提案する．4 章では SEmigrate の実装について説明し，5 章では SEmigrate を用いて行っ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. 複数ホストにまたがる VM のデータ保護

2.1 分割マイグレーション

分割マイグレーション [1][2] は図 1 のように，VM の大容量メモリを分割して複数の小さなホストに転送するマイグレーション手法である．移送先ホストは 1 つのメインホストと 1 つ以上のサブホストからなる．今後アクセスされることが予測されるメモリデータは VM のメモリアクセス履歴に基づいて可能な限りメインホストへ転送する．また，仮想 CPU や仮想デバイスの状態などの VM コアの状態もメインホストに転送する．メインホストに入り切らなかったメモリデータはサブホストのいずれかに転送する．

分割マイグレーション後には，VM はメインホストとサブホストにまたがって動作する．メインホスト上で VM コアが動作し，サブホストはその VM にメモリを提供する．メインホスト上の VM コアがサブホスト上に存在するメモリデータを必要とした時には，リモートページングを行ってメインホストとサブホスト間でメモリデータを交換する．まず，サブホストからメインホストに VM コアが必要としたメモリデータを転送（ページイン）する．同時に，メインホスト上の今後アクセスされないことが予測されるメモリデータをサブホストに転送（ページアウト）する．

2.2 VM のメモリデータの暗号化・整合性検査

実行環境によっては，分割マイグレーションやリモートページングの際に VM のメモリデータを盗聴されたり改竄されたりする危険性がある．例えば，安全であることが保証された専用線以外を使ってメモリデータが転送される場合には，転送中に攻撃を受ける恐れがある．パブリッククラウド内部でメモリデータが転送される場合には，クラウドの管理者などから攻撃を受ける可能性がある．移送元ホストやメインホストのような VM が動作するホストでのメモリ保護については様々な研究が行われてきた [4], [5], [6] が，サブホスト上にある VM のメモリデータは危険にさらされる可能性がある．

このようなメモリデータの盗聴や改竄は，VM のメモリデータを暗号化したり整合性検査を行ったりすることで防ぐことができる．分割マイグレーションの際には，SSL/TLS 等の暗号通信路を用いて移送元ホストでメモリデータを暗号化して転送し，移送先のメインホストとサブホストで復号する．ただし，サブホストでは復号されたメモリデータを別途，暗号化して安全に保持する必要がある．このような再暗号化が必要になるのは，暗号通信路で用いられている暗号鍵を通信完了後に使い続けるのが難しいためである．一方，メインホストでは VM のメモリ保護機構 [4], [5], [6] を用いることができるため，このような再暗号化は不要である．

リモートページングの際には，VM が要求したサブホスト上のメモリデータを復号し，メインホストとの間に確立された暗号通信路を用いて暗号化して転送する．メインホストではそれを復号してページインの処理を行う．同時に，メインホスト上の不要なメモリデータに対してページアウトの処理を行い，暗号通信路を用いて暗号化してサブホストに転送する．サブホストではそれを復号し，再暗号化して保持する．

暗号通信路は転送されるメモリデータの整合性検査も行う．分割マイグレーションの際には，移送元ホストがメモリデータを暗号化する前にメッセージ認証コード（MAC）を計算し，移送先ホストに転送する．移送先ホストでは受信したメモリデータを復号した後で MAC を再計算し，受信した MAC と比較する．MAC が一致しない場合には，転送の途中でメモリデータが改竄されたことを検出するこ

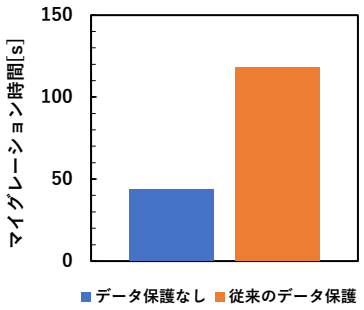

図 2 データ保護によるマイグレーション時間の増加

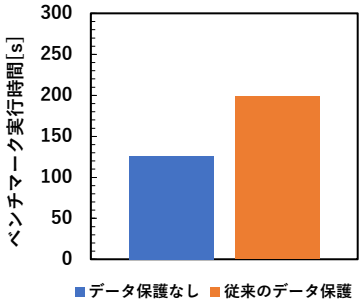

図 3 データ保護によるベンチマーク実行時間の増加

とができる．リモートページングの際には，送信元ホストでメモリデータの MAC を計算して転送し，送信先ホストで MAC を再計算して比較する．

このように，暗号通信路を用いるとメモリデータを転送するたびに暗号化・復号化および整合性検査が行われるためオーバヘッドが大きくなる．データ保護を行った場合に分割マイグレーションにかかる時間を図 2 に，マイグレーション後にリモートページングを多発させるベンチマークを実行するのにかかる時間を図 3 に示す．この測定には 5 章の実験環境を用いた．暗号化と整合性検査により，分割マイグレーションでは 2.7 倍，ベンチマーク実行では 1.6 倍の時間がかかることが分かる．性能の問題に加えて，サブホストではメモリデータが復号された後で再暗号化を行うため，その間に盗聴や改竄が行われる危険性もある．また，サブホストには暗号鍵が置かれているため，暗号鍵を管理者に盗まれた場合には容易にメモリデータを復号されたり改竄されたりしてしまう．

3. SEmigrate

本稿では，分割マイグレーションとリモートページングにおいてメモリデータの暗号化および整合性検査の最適化を行う *SEmigrate* を提案する．SEmigrate はサブホストにおいてメモリデータの復号と整合性検査を行わないようにすることでメモリ保護のオーバヘッドを削減し，サブホストにおける情報漏洩を完全に防げるようにする．また，必要に応じてメモリデータの選択的な暗号化・整合性検査を

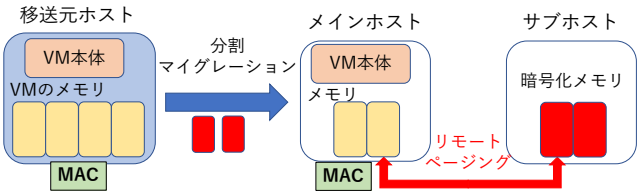

図 4 サブホストにおけるデータ保護の最適化

行うことにより，さらにデータ保護のオーバヘッドを削減する．そのために，SEmigrate は VM イントロスペクション [7] を用いて VM 内のゲスト OS のメモリを解析して，メモリ属性やプロセス情報を利用する．さらに，VM 内のアプリケーションのメモリも解析して，アプリケーション固有の情報も利用する．

3.1 サブホストにおけるデータ保護の最適化

SEmigrate は図 4 のようにサブホストにおいて VM のメモリデータを復号しないようにする．分割マイグレーション時には，移送元ホストからメモリデータを暗号化して転送し，移送先メインホストでのみ復号する．移送先サブホストではメモリデータを復号せずに暗号化された状態のまま保持する．このメモリデータを後で復号できるようにするために，マイグレーション専用の暗号通信路は用いず，VM のライフサイクルを通して利用可能な暗号鍵を用いる．これにより，サブホストにおいて暗号通信路によって一旦，復号されたデータを再暗号化して情報漏洩を防ぐ必要がなくなる．また，復号されてから再暗号化するまでの間に情報が漏洩する恐れもない．さらに，サブホストは復号のための鍵を保持せずに済むようになるため，サブホストの管理者であっても保持されている VM のメモリデータを盗聴することはできない．

リモートページング時には，メインホストでのみ VM のメモリデータの暗号化・復号化を行う．ページイン時には転送元のサブホストではメモリデータを復号しないため，暗号通信路を用いることなくメモリデータを安全に転送することができる．転送先のメインホストではそのメモリデータを復号して VM のメモリに格納してアクセスする．一方，ページアウト時には転送元のメインホストで暗号通信路を用いずにメモリデータを暗号化し，安全に転送する．転送先のサブホストではそのメモリデータを復号せずにそのまま保持する．このメインホストでの暗号化・復号化には分割マイグレーション時に移送元ホストとの間で共有した暗号鍵を用いる．

また，サブホストでは VM のメモリデータの整合性検査も行わないようにする．分割マイグレーション時には，移送元ホストでサブホストに転送するメモリデータの MAC を計算し，それを移送先メインホストに転送する．これはサブホストで整合性検査を行わないようにすることで，

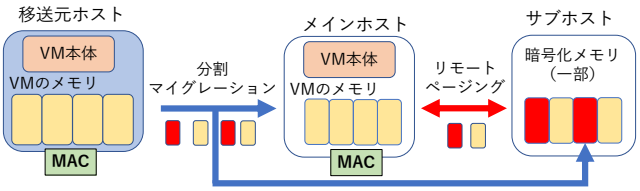

図 5 選択的なデータ保護

MAC をサブホストに転送する必要がなくなるためである．そして，ページイン時にメインホストがサブホストからのメモリデータを受信すると，それを復号した後で MAC を再計算し，保持していた MAC との比較を行うことで整合性検査を行う．ページアウト時には転送するメモリデータの MAC をメインホストで計算して保持しておく．このように，SEmigrate はメモリデータの MAC をサブホストとメインホストの間で転送しない．

サブホストで整合性検査を行わないようにすることのデメリットとして，メモリデータの改竄の検知が遅れることが挙げられる．分割マイグレーション時にサブホストに転送されるメモリデータが改竄された場合，マイグレーション完了後にページインした時に整合性検査を行って初めてメモリデータの整合性が取れていないことに気づくことになる．分割マイグレーション中に改竄を検知できればマイグレーションをキャンセルできるが，マイグレーションが完了していると VM を実行し続けることができなくなる．そのため，整合性検査の最適化を行う際には性能向上と信頼性のトレードオフを考える必要がある．この問題への対処として，サブホストに転送したデータは移送元ホストなどのディスク上に保持しておくことが考えられる．最初のページイン時に改竄が検知された場合には，移送元からメインホストに改竄前のメモリデータを再転送することで VM の実行を継続することができる．

3.2 VM 内情報に基づく選択的なデータ保護

SEmigrate は移送元ホストおよびメインホストでメモリデータを選択的に暗号化することでオーバヘッドを削減する．分割マイグレーション時には，移送元ホストは図 5 のように機密情報を含むメモリだけを暗号化し，それ以外のメモリは暗号化せずに移送先ホストに転送する．移送先メインホストでは暗号化されている場合だけメモリデータを復号する．移送先サブホストでは暗号化されているかどうかに関わらず，受信したメモリデータをそのまま保持する．

ページイン時には，サブホストは保持しているメモリデータが暗号化されているかどうかに関わらずそのままメインホストに転送する．メインホストはメモリデータが暗号化されている場合だけ復号する．ページアウト時には，メインホストは機密情報を含まないメモリは暗号化せずにサブホストに転送する．サブホストは暗号化の有無に関わらずそのままメモリデータを保持する．

VM 内のゲスト OS が使用していないメモリ領域には機密情報は格納されないため，SEmigrate はゲスト OS の空きメモリを暗号化しない．メモリデータを転送する際にはまず，ゲスト OS からそのメモリの属性を取得し，空きメモリかどうかを調べる．また，OS やアプリケーションのコード領域には一般に機密情報は格納されていないため，SEmigrate は VM 内のゲスト OS やプロセスのコード領域を暗号化しない．メモリデータを転送する際にはまず，ゲスト OS のメモリ管理情報からそのメモリデータが含まれるメモリ領域の属性を調べ，実行可能であればコード領域と判定する．特定のアプリケーションのコードを秘匿する必要がある場合には，そのコード領域だけを暗号化することもできる．

機密情報を扱わないアプリケーションが指定された場合，SEmigrate は VM 内でそのアプリケーションを実行するプロセスのメモリを暗号化しない．例えば，暗号化されたデータしか扱わないインメモリ・データベースのメモリは暗号化する必要がない．SGX アプリケーションの場合も機密情報がエンクレイヴ内でのみ扱われる場合にはメモリを暗号化する必要はない．メモリデータを転送する際にはまず，そのメモリデータが含まれるメモリ領域をゲスト OS のメモリ管理情報から見つけ，そのメモリ領域を所有するプロセスを特定する．そして，そのプロセスの名前や ID が指定したものと一致すれば暗号化を行わない．

さらに，VM 内の特定のアプリケーションの特定のメモリ領域が指定された場合，SEmigrate はそのメモリ領域を暗号化しない．例えば，暗号データとその復号鍵をメモリ上に保持するインメモリ・データベースの場合，暗号データについては転送時に暗号化する必要はない．SGX アプリケーションは VM マイグレーションの際にエンクレイヴのメモリが暗号化されて保存される [8] が，保存先のメモリ領域は暗号化する必要がない．一方，エンクレイヴ外でも機密情報を扱う場合にはそのメモリ領域は暗号化する必要がある．メモリデータを転送する際にはまず，ゲスト OS の対象プロセスのメモリを解析してアプリケーションのデータを取得し，そのプロセス上にある暗号化を除外するメモリ領域を特定する．そして，転送しようとしているメモリデータがそのメモリ領域に含まれていれば暗号化を行わない．

同様にして，SEmigrate はメモリデータの選択的な整合性検査を行うことでもオーバヘッドを削減する．分割マイグレーション時には，移送元ホストは整合性検査が必要なメモリに対してのみ MAC を計算して移送先メインホストに転送し，それ以外のメモリに対しては MAC の計算を行わない．ページイン時には，メインホストが MAC を保持していう場合だけ，受信したメモリデータの MAC を計算して比較を行う．ページアウト時には，整合性検査が必要なメモリに対してのみ MAC を計算して保持する．常に

整合性検査が必要ないメモリ領域としては，情報が格納されていない空きメモリが考えられる．また，アプリケーションが整合性検査を行っているデータ領域については SEmigrate が整合性検査を行う必要はない．さらに，メモリデータが暗号化されている場合には攻撃者が意図したように改竄することはできないため，整合性検査は必須ではない．

4. 実装

分割マイグレーションとリモートページングを QEMU-KVM 7.1.0 に移植し，SEmigrate を実装した．選択的なデータ保護のために用いる VM イントロスペクションの対象となるゲスト OS として Linux 4.18.17 を用いた．メモリデータの暗号化には CPU の AES-NI 命令セットをサポートした OpenSSL の AES-XTS を用い，鍵長は 256 ビットとした．メモリデータの整合性検査には CPU の SHA 拡張をサポートした OpenSSL の SHA-256 を用いた．

4.1 空きメモリの判定

SEmigrate は VM 内のゲスト OS によって使われていないメモリを保護しないようにするために，VM のメモリを解析することで転送しようとしているページが空きメモリであるかどうかを判定する．Linux では物理メモリは Buddy システムを用いて連続する 2^n ページ単位で割り当てられ，空きメモリは 2^n ページからなる空きメモリ領域として管理されている．空きメモリ領域の先頭ページを管理するページ構造体には Buddy ページフラグが設定され，空きメモリ領域に含まれるページ数（正確には 2^n の n の値）が格納される．しかし，これらの情報は先頭ページのページ構造体にしか格納されないため，それ以外のページについては空きメモリかどうかの判定を行うのは容易ではない．

そこで，SEmigrate は分割マイグレーションの際にはページが基本的にページ番号の小さい順に転送されることを利用して，効率よく空きメモリかどうかを判定する．ページを転送する際にはそのページ番号をインデックスとして，Linux が管理しているページ構造体の配列から対応するページ構造体を取得する．空きメモリ領域の先頭ページを転送する際にはそのページ番号と空きメモリ領域に含まれるページ数を記録しておく．連続するページを送信する際には，そのページ番号が記録しておいたページ範囲に入っているかどうかを調べることで，即座に空きメモリと判定することができる．

分割マイグレーションにおいて更新されたメモリの再送を行う場合や，リモートページングで必要とされたページの転送を行う場合にはページがページ番号順に転送されないことが多い．その場合には，空きメモリ領域に含まれるページ数が $2^n (n = 0, 1, \cdots, 10)$ であることを利用して空きメモリかどうかを調べる．まず，図 6 のように転送しようとしているページが含まれる可能性のある空きメモリ領域の先頭ページの候補を調べる．先頭ページの候補は転送しようとしているページのページ番号の下位 n ビットを 0 にマスクすることで容易に見つけることができる．見つけたページが先頭ページであり，かつ，転送しようとしているページが見つけたページを先頭とする空きメモリ領域に含まれる場合，転送するページは空きメモリと判定する．

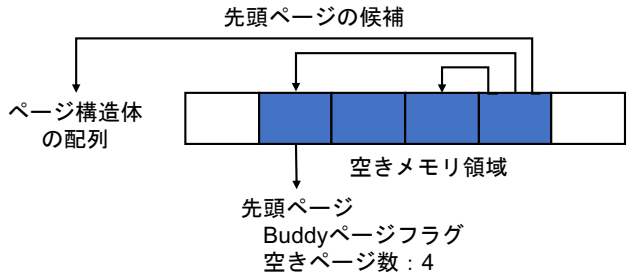

図 6 任意のページに対する空きメモリ判定

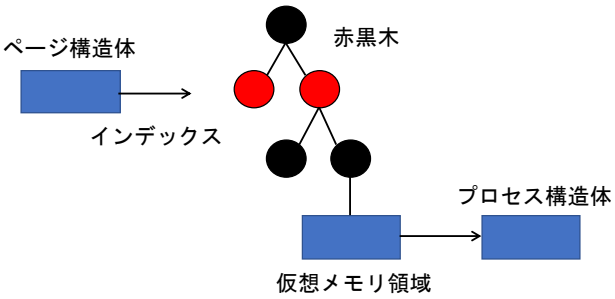

図 7 Linux の仮想メモリ領域の探索

4.2 コード領域の判定

SEmigrate はコード領域を保護しないようにするために，転送しようとしているページが VM 内のゲスト OS またはプロセスのコード領域に含まれるかどうかを判定する．そのために，図 7 のようにゲスト OS が管理しているデータ構造の中から当該ページが含まれる仮想メモリ領域を探す．まず，当該ページのページ番号に対応するページ構造体を見つけ，仮想メモリ領域を管理するために用いられている赤黒木におけるインデックスを取得する．そのインデックスを基に赤黒木を探索して目的の仮想メモリ領域を見つける．その仮想メモリ領域の属性を調べ，メモリが実行可能であればコード領域のページであると判定する．

4.3 プロセスメモリの判定

SEmigrate は特定のプロセスのメモリを保護しないようにするために，転送しようとしているページが指定されたプロセスのメモリに含まれるかどうかを判定する．そのために，当該ページを所有しているプロセスを探す．まず，4.2 節と同様にして赤黒木を探索し，当該ページが含まれ

る仮想メモリ領域を見つける．次に，その仮想メモリ領域を所有しているプロセスを見つけ，プロセス構造体からプロセスの名前や ID を取得する．それらが指定したものと一致すれば対象プロセスのページであると判定する．

4.4 プロセスの特定メモリ領域の判定

SEmigrate は特定のアプリケーションの中の特定のメモリ領域を保護しないようにするために，転送しようとしているページが指定されたプロセス内の指定された仮想メモリアドレス範囲に含まれるかどうかを判定する．まず，4.3 節と同様にして当該ページを所有するプロセスの名前が指定されたものと一致するかどうかを調べる．一致する場合には，当該ページが含まれる仮想メモリ領域のアドレス範囲と赤黒木のインデックスから，当該ページに割り当てられている仮想アドレスを計算する．この仮想アドレスがプロセス内の指定された仮想メモリアドレス範囲に入っていれば，対象メモリ領域のページであると判定する．

プロセス内のメモリ領域はプロセスのメモリを解析することによって特定する．現在のところ，当該メモリ領域の先頭を指すポインタとそのサイズがアプリケーションの大域変数に格納されている場合についてサポートしている．大域変数のアドレスはアプリケーションのバイナリファイルから取得することができ，その実行時の仮想アドレスは取得したアドレスに固定のオフセット（0x555555554000）を加えたものとなる．Linux のアドレス空間配置のランダム化（ASLR）が有効になっている場合には仮想アドレスに乱数も加えられてしまうため，現在の実装では ASLR は無効にしている．上記のようにして見つけた，当該ページを所有するプロセスについてプロセス構造体からページテーブルを取得し，アプリケーションの大域変数の仮想アドレス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 のメモリにアクセスすることで大域変数に格納されている値を取得する．

4.5 VM 内情報へのアクセス

QEMU-KVM から VM 内の情報を取得するために，明示的に VM 内のゲスト OS と通信するのは望ましくない．VM 内のゲスト OS を改変するのは避けたいためである．そこで，SEmigrate は LLView[3] を QEMU-KVM に移植したものを用いて，VM のメモリ上の OS データを解析することで VM 内の情報を透過的に取得する．LLView は OS のソースコードを用いて VM の外から OS データを取得するためのプログラムを記述することを可能にする．そのために，コンパイル時に自動的にプログラム変換を行い，OS データにアクセスする箇所で仮想アドレスを物理アドレスに変換して VM のメモリにアクセスするようにする．

SEmigrate は VM の仮想ハードウェアによるメモリ再マップを考慮して VM の物理メモリにアクセスする．VM 内では 3〜4GB の物理アドレスは PCI のメモリマップト領域として使用されるため，3GB を超える物理メモリは 4GB 以降の物理アドレスに再マップされる．一方，QEMU-KVM が VM に割り当てるメモリは連続しており，メモリの先頭から 3GB 以降の部分については 1GB 分のずれが生じる．そのため，SEmigrate は 4GB より大きい物理アドレスを持つ VM 内の OS データにアクセスする場合には，その物理アドレスから 1GB を減じたオフセットを用いて VM に割り当てたメモリにアクセスする．

4.6 選択的に保護されるデータの指定

空きメモリやコード領域についてはメモリ属性から自動的に判別されるため，ユーザが指定する必要はない．プロセスを指定する場合は設定ファイルにプロセス名を記述する．この設定ファイルは QEMU-KVM が起動時に読み込む．プロセスの特定メモリ領域を指定する場合は，設定ファイルにプロセス名およびメモリ領域の先頭アドレスとサイズを記述する．一方，VM の実行時に動的に割り当てられるプロセス ID を用いてプロセスを指定する場合には，VM ソケット（Vsock）を用いて VM 内から QEMU-KVM に通知する．Vsock は VM 内外で通信を行うための機構であり，ネットワークソケットと同様に利用することができる．この指定方法は同じ名前を持つプロセスが複数あり，その一部だけを指定したい場合に用いる．

5. 実験

SEmigrate を用いてデータ保護を最適化することにより，分割マイグレーションとマイグレーション後の VM の性能をどの程度向上させられるかについて調べる実験を行った．移送元ホスト，移送先メインホスト，移送先サブホストには Intel Core i7-12700 の CPU，128GB のメモリを搭載したマシンを用い，OS には Linux 5.15.60 を用いた．NIC として，移送元ホストでは ConnectX-5 VPI アダプタカードを用い，メインホストとサブホストでは Connect X-4 VPI アダプタカードを用いた．移送元ホストとメインホスト，移送元ホストとサブホストはそれぞれ 100 ギガビットイーサネットで直結し，メインホストとサブホストは移送元ホストをネットワークブリッジとして用いて接続した．iperf を用いて測定した 1 並列でのネットワーク性能は，移送元ホストとメインホスト間で 27Gbps，メインホストとサブホスト間で 19Gbps であった．VM には 1 個の仮想 CPU と 96GB のメモリを割り当て，ゲスト OS には Linux 4.18.17 を用いた．分割マイグレーション時にはメモリを半分ずつに分割した．

5.1 サブホストにおける最適化の効果

SEmigrate を用いてサブホストにおけるデータ保護の最適化を行った際の性能向上について調べた．分割マイグ

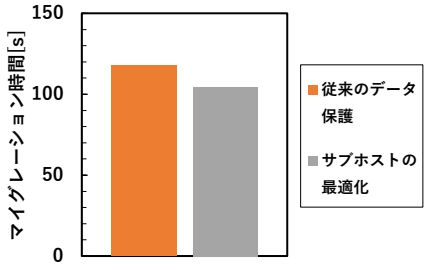

図 8 分割マイグレーション時間（サブホスト最適化）

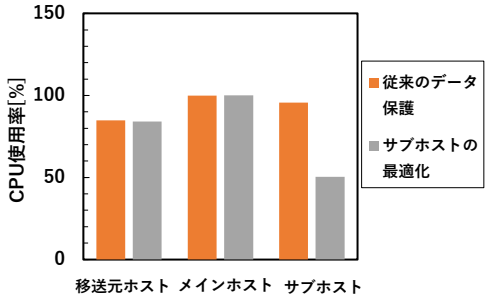

図 9 マイグレーション時の CPU 使用率（サブホスト最適化）

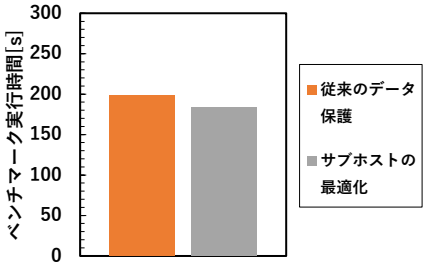

図 10 ベンチマーク実行時間（サブホスト最適化）

レーションにかかった時間を図 8 に示す．サブホストで受信したメモリデータの復号と整合性検査を行わないようにすることで，マイグレーション時間が 11%短縮された．この時の移送元ホスト，移送先メインホスト，移送先サブホストの CPU 使用率の平均を図 9 に示す．サブホストでは暗号化・復号化および MAC の計算を行わないことにより CPU 使用率が 45 ポイント低下した．

次に，分割マイグレーション後に VM 内で 50GB のメモリにアクセスするベンチマークを実行した．このベンチマークの実行時間を図 10 に示す．リモートページングの際にサブホストで暗号化・復号化および MAC の計算・転送を行わないようにすることで，アプリケーションの実行時間が 7%削減できた．この時の CPU 使用率の平均を図 11 に示す．暗号化と整合性検査を行う場合と比べて，サブホストでは CPU 使用率が 9 ポイント低下した．

5.2 選択的なデータ保護の効果

SEmigrate を用いて選択的なデータ保護を行った際の性能向上について調べた．

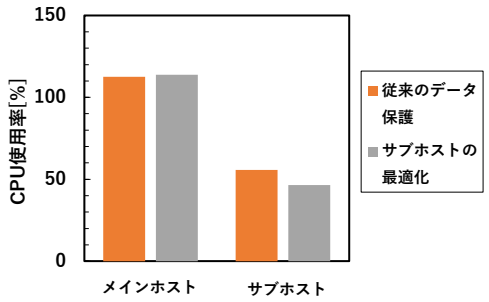

図 11 ベンチマーク実行時の CPU 使用率（サブホスト最適化）

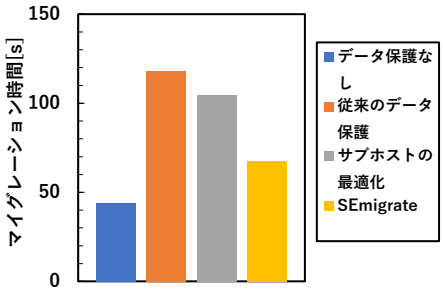

図 12 分割マイグレーション時間

5.2.1 分割マイグレーション性能

VM の分割マイグレーションにかかった時間を図 12 に示す．この実験では，VM 内で 50GB のメモリを使用するアプリケーションを実行し，このプロセスのメモリを暗号化しないように設定した．SEmigrate は選択的なデータ保護を行うことにより，一律にデータ保護を行う従来手法と比べてマイグレーション時間を 43%削減することができた．サブホストにおける最適化のみを行った場合と比較しても 35%短縮できている．また，データ保護を行わない場合と比べても 54%の増加となった．

この時の CPU 使用率の平均を図 13 に示す．一律にデータ保護を行う場合と比べて，SEmigrate はサブホストでのみ CPU 使用率を 43 ポイント低下させることができた．これはサブホストにおける最適化によるものである．一方，移送元ホストでは 92GB のメモリデータは暗号化されず，42GB のメモリデータについては MAC の計算が行われなかったにも関わらず，CPU 使用率はほとんど低下しなかった．これはデータ保護の処理が減った分，より高速にメモリデータの転送処理を行えるようになり，CPU 負荷が上昇したためと考えられる．同様のことが，データ保護を行わなかった場合についても言える．メインホストでも復号と整合性検査を行うメモリデータは減ったが，転送が高速になったために CPU 使用率はほとんど低下しなかったと考えられる．

選択的なデータ保護による性能向上の内訳を調べるために，1 種類のメモリ領域についてのみ最適化を行った場合

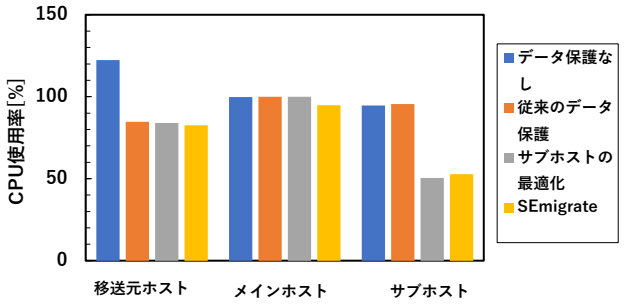

図 13 マイグレーション時の CPU 使用率

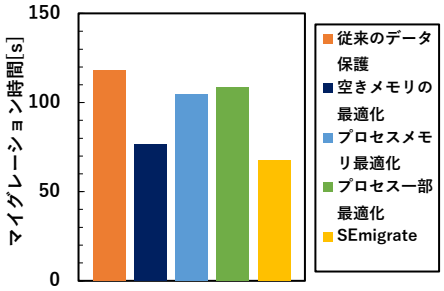

図 14 分割マイグレーションの性能向上の内訳

のマイグレーション性能を調べた．測定結果を図 14 に示す．空きメモリの暗号化・整合性検査を行わないようにしただけでマイグレーション時間が 35%短縮されたことから，42GB の空きメモリのデータを保護しないことによる性能向上が大きいことが分かる．一方，プロセスメモリの暗号化を行わないようにしただけでは 11%しか削減することができなかった．このことから整合性検査を行わないようにすることの効果が大きいと考えられる．アプリケーションが確保したメモリ領域だけを暗号化しないようにした場合には，プロセス単位で最適化を行った場合より 4%遅くなった．

次に，アプリケーションが使用するメモリ量を変えてマイグレーション時間を測定した．図 15 に示すように，アプリケーションがメモリを使わないほど，つまり，空きメモリが増えるほど，性能が向上していることが分かる．例えば，空きメモリが 92GB の場合にはマイグレーション時間は 63%短くなった．一方，アプリケーションが 90GB のメモリを使う場合にはマイグレーション時間は 15%しか短縮されなかった．

5.2.2 マイグレーション後のベンチマーク性能

分割マイグレーション後に VM 内で 5.1 節のベンチマークを実行するのにかかる時間を測定した．図 16 に示すように，SEmigrate はリモートページングの際に選択的なデータ保護を行うことにより，従来手法と比べてベンチマーク実行時間を 19%削減することができた．サブホストにおける最適化のみを行った場合と比べても 13%短縮できている．データ保護を行わない場合と比べても 28%の実行

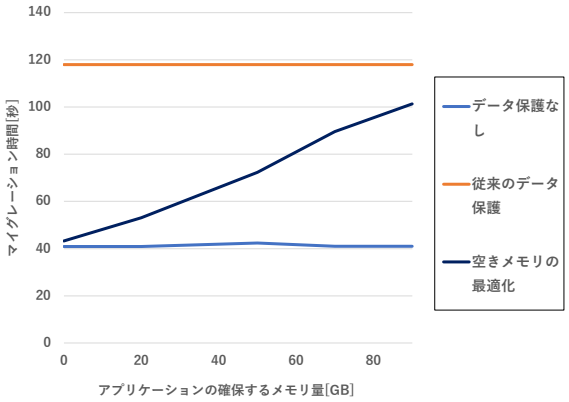

図 15 プロセスのメモリ量とマイグレーション時間の関係

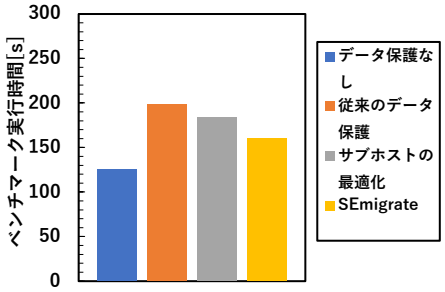

図 16 ベンチマーク実行時間

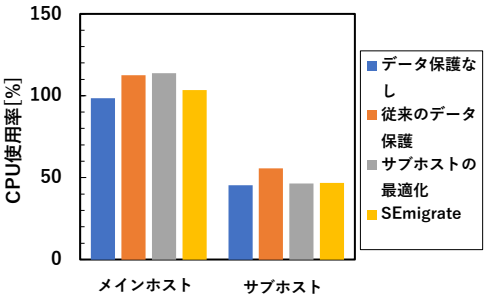

図 17 ベンチマーク実行時の CPU 使用率

時間の増加となった．この時の CPU 使用率の平均を図 17 に示す．従来手法と比べて，メインホストでは CPU 使用率を 9 ポイント，サブホストでも 9 ポイント低下させることができた．

図 18 に示すように，空きメモリの暗号化・整合性検査だけを行わないようにした場合にはベンチマークの実行時間は 15%削減された．また，プロセスのメモリの暗号化だけを行わなかった場合でも 15%短縮された．アプリケーション内のメモリ領域だけを暗号化しなかった場合には実行時間の削減率は 12%となった．これは分割マイグレーションの場合とは異なる傾向であるが，リモートページングの特性によるものであると考えられる．

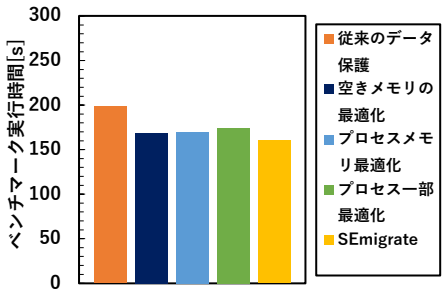

図 18 ベンチマークの性能向上の内訳

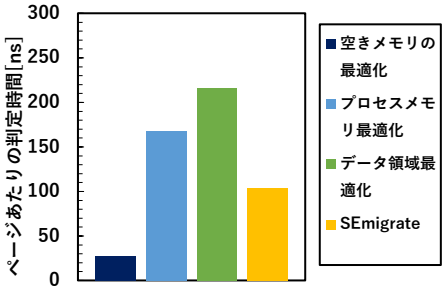

図 19 選択的に保護するデータの判定時間

5.3 選択的に保護するデータの判定時間

SEmigrate において選択的なデータ保護を行うために必要になる VM のメモリ解析のオーバヘッドを調べた．この実験では 5.2 節のアプリケーションを実行し，そのメモリを暗号化しないようにした．図 19 に 1 ページあたりの解析時間の平均を示す．空きメモリの判定にかかる時間は 27 ns であり高速に行うことができるが，プロセスメモリの判定にはその 6.2 倍の時間がかかる．アプリケーション内に確保したメモリ領域だけを暗号化しないようにした場合，プロセス内の特定メモリ領域の判定にはさらに 48 ns かかることが分かった．これらの選択的なデータ保護をすべて適用した場合には判定に 1 ページあたり平均 104 ns かかり，96GB のメモリに対しては 2.6 秒かかった．

6. 関連研究

VM マイグレーションについては VM イントロスペクションを用いた様々な最適化手法が提案されている．MiG[9] は，VM 内のゲスト OS からメモリ属性を取得し，それに基づいてメモリの圧縮方法を最適化する．例えば，空きメモリについてはページ番号だけを転送し，ヒープ領域は冗長性が高いため gzip で圧縮する，などである．これにより，マイグレーション中に転送されるバイト数が平均で 51〜65 %削減でき，マイグレーション時間を半分にすることができている．しかし，VM の状態を一括保存してから圧縮しており，VM のライブマイグレーションには対応できていない．

IntroMigrate[10] は VM 内のゲスト OS の空きメモリを特定し，そのメモリデータを転送しないようにすることで VM マイグレーションを高速化する．ただし，マイグレーション開始時にすべてのページについて空きメモリ情報を取得するため，マイグレーション中に使用中になったり空きメモリになったりしたページについては古い情報に基づいて最適化を行ってしまう．マイグレーション中に使用中になったページの転送については再送に頼っている．また，類似研究 [11] では空きメモリに加えてゲスト OS のページキャッシュについても転送を行わないようにし，移送先でホストでページキャッシュを破棄する．

FCtrans[12] は分割マイグレーションやリモートページングの際に VM の未使用メモリを転送しないことで最適化を行っている．VM の未使用メモリは一旦アクセスされると使用中になるため，定期的に VM 内のゲスト OS の空きメモリを特定して VM のメモリを未使用状態に戻す．SEmigrate では空きメモリは暗号化せずに転送しているが，データを転送しないようにすることでさらなる高速化が可能である．

VM のメモリの盗聴を防ぐために VMCrypt[5] が提案されている．VMCrypt は VM には暗号化されていないメモリデータを提供しつつ，管理 VM には暗号化された VM のメモリデータを提供する．VM マイグレーションの際などには VM のメモリデータを転送する必要があるが，ハイパーバイザによって暗号化されたメモリデータを転送することで機密情報の漏洩を防ぐ．SEmigrate でも移送元ホストと移送先メインホストに VMCrypt を適用することで，ハイパーバイザが攻撃されない限りは盗聴を防ぐことができる．

7. まとめ

本稿では，分割マイグレーションおよびリモートページングにおいてメモリデータの暗号化と整合性検査を最適化する *SEmigrate* を提案した．SEmigrate はサブホストにおいてメモリデータの復号と整合性検査を行わないようにすることでデータ保護のオーバヘッドを削減し，サブホストにおける情報漏洩を完全に防ぐ．また，必要に応じて選択的な暗号化・整合性検査を行うことで，さらにデータ保護のオーバヘッドを削減する．そのために，VM イントロスペクションを用いて VM 内のゲスト OS やアプリケーションのメモリを解析し，空きメモリやコード領域，指定されたアプリケーションやそのメモリ領域の一部を特定する．SEmigrate を KVM に実装し，分割マイグレーションおよびマイグレーション後の VM の性能を調べる実験を行った．その結果，メモリデータを暗号化する場合に分割マイグレーションにかかる時間を 43%削減できた．また，マイグレーション後の VM 内のベンチマーク実行時間を 19%削減できた．

今後の課題は，様々な実アプリケーションに SEmigrate

を適用し，選択的なデータ保護を行えるようにすることである．そのために，ユーザやアプリケーション開発者が保護すべきデータを指定しやすくする仕組みが必要である．

参考文献

[1] Suetake, M., Kizu, H. and Kourai, K.: Split Migration of Large Memory Virtual Machines, *Proc. ACM SIGOPS Asia-Pacific Workshop of Systems* (2016).
[2] Suetake, M., Kashiwagi, T., Kizu, H. and Kourai, K.: S-memV: Split Migration of Large-memory Virtual Machines in IaaS Clouds, *Proc. IEEE Int. Conf. Cloud Computing*, pp. 285–293 (2018).
[3] Ozaki, Y., Kanamoto, S., Yamamoto, H. and Kourai, K.: Detecting System Failures with GPUs and LLVM, *Proc. ACM SIGOPS Asia-Pasific Workshop*, pp. 47–53 (2019).
[4] Li, C., Raghunathan, A. and Jha, N. K.: Secure Virtual Machine Execution under an Untrusted Management OS, *Proc. Int. Conf. Cloud Computing*, pp. 172–179 (2010).
[5] Tadokoro, H., Kourai, K. and Chiba, S.: Preventing Information Leakage from Virtual Machines' Memory in IaaS Clouds, Trans, *ACS* (2012).
[6] Zhang, F., Chen, J., Chen, H. and Zang, B.: CloudVisor: Retrofitting Protection of Virtual Machines in Multi-tenant Cloud with Nested Virtualization, *Proc. Symp. Operating Systems Principles*, pp. 203–216 (2011).
[7] Garfinkel, T. and Rosenblum, M.: A Virtual Machine Introspection Based Architecture for Intrusion Detection, *Proc. Network and Distributed Systems Security Symp.*, pp. 191–206 (2003).
[8] Gu, J., Hua, Z., Xia, Y., Chen, H., Zang, B., Guan, H. and Li, J.: Secure Live Migration of SGX Enclaves on Untrusted Cloud, *Proc. Int. Conf. on Dependable Systems and Networks*, pp. 225–236 (2017).
[9] Rai, A., Ramjee, R., Anand, A., Padmanabhan, V. and Varghese, G.: MiG: Efficient Migration of Desktop VMs using Semantic Compression, *Proc. USENIX Annual Technical Conf.*, pp. 25–36 (2013).
[10] Chiang, J., Li, H. and Chiueh, T.: Live virtual machine migration with adaptive memory compression, *Proc. Virtual Execution Environment*, pp. 51–61 (2013).
[11] Wang, C., Hao, Z., Cui, L., Zhang, X. and Yun, X.: Introspection-based Memory Pruning for Live VM Migration, *Int. J. Parallel Program*, Vol. 45, No. 6, pp. 1298–1309 (2017).
[12] Tauchi, S., Kourai, K. and Rahim, L. A.: Optimizing VMs across Multiple Hosts with Transparent and Consistent Tracking of Unused Memory, *Proc. Int. Conf. Cloud Computing*, pp. 467–477 (2021).