Vol.2021-OS-152 No.1
2021/5/27

複数ホストにまたがるVMのデータ暗号化の最適化

堀尾 周平[1]　高橋 孝汰[1]　光来 健一[1]　Lukman Ab. Rahim[2]

概要：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きているが，このようなVMのマイグレーション時に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分割マイグレーションやリモートページングの際にメモリデータを盗聴される危険性がある．メモリデータを暗号化することで情報漏洩を防ぐことができるが，暗号化のオーバヘッドにより性能が大幅に低下する．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暗号化の最適化を行う *SEmigrate* を提案する．SEmigrateはサブホストにおいてメモリデータを復号しないようにすることで暗号化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，機密情報が含まれるメモリのみを選択的に暗号化することで，さらに暗号化のオーバヘッドを削減する．そのために，SEmigrateはVM内のゲストOSやアプリケーションのメモリを解析して，メモリ属性やアプリケーション固有の情報を利用する．SEmigrateをKVMに実装し，実験を行った結果，メモリ暗号化を行う場合に分割マイグレーションとマイグレーション後のVMの性能をそれぞれ最大82%と91%改善できることが分かった．

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きている．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り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メモリデータを盗聴される危険性がある．例えば，メモリデータが安全とは限らないネットワーク経由で転送される場合には盗聴されやすくなる．また，メインホストと管理者が異なるサブホストを利用する場合にも盗聴されるリスクが高まる．情報漏洩を防ぐためにはメモリデータを暗号化すればよいが，SSL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・復号化が行われるため，性能が大きく低下する．また，暗号化が必要な機密情報かどうかは考慮されず，すべてのメモリデータが一律に暗号化される．

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暗号化を最適化する *SEmigrate* を提案する．SEmigrateはサブホストにおいてメモリデータを復号しないようにすることで暗号化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．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，リモートページング時にはメインホストでのみ暗号化・復号化を行

[1] 九州工業大学
Kyushu Institute of Technology
[2] Universiti Teknologi Petronas

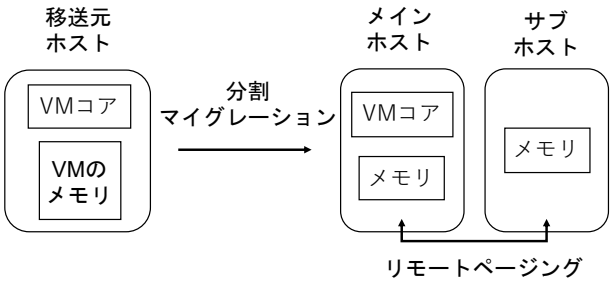

図 1 分割マイグレーションとリモートページング

う．また，SEmigrate は機密情報が含まれるメモリ領域のみを選択的に暗号化することでさらに暗号化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．

具体的には SEmigrate は VM 内のゲスト OS が使用していない空きメモリおよび，OS やアプリケーションのコード領域には機密情報が含まれていないものとしてそのメモリデータを暗号化せずに転送する．また，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せずに転送する．さらに，特定のアプリケーションの特定メモリ領域が指定された場合，SEmigrate はそのメモリ領域に含まれるデータを暗号化せずに転送する．

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，メモリ暗号化を行う場合に分割マイグレーションにかかる時間を最大で 82%削減できることが分かった．また，マイグレーション後の VM の性能を最大で 91%向上させられることも分かった．

2. 複数ホストにまたがる VM の暗号化

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

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メモリデータを盗聴される危険性がある．例えば，分割マイグレーションによって VM のメモリデータの一部がデータセンタ間などの安全とは限らないネットワークを経由して転送される場合には盗聴されやすくなる．同様に，メインホストとサブホストが安全ではないネットワークを経由してリモートページングを行う場合もメモリデータが危険にさらされる．また，VM が動作するホスト（メインホスト）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，サブホストに VM のメモリデータの一部がある場合にはそこから情報漏洩する可能性がある．メインホストが信頼できる場合でもそれとは管理者が異なるサブホストを利用することも考えられる．

分割マイグレーションやリモートページングに伴うこのような情報漏洩は VM のメモリデータを暗号化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 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，まずサブホスト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処理を行う．次に，メインホストの不要なメモリデータをこの暗号通信路を用いて暗号化して転送する．サブホストではそれを復号し，再暗号化して保持する．

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・復号化が行われるためオーバヘッドが大きくなる．このオーバヘッドはネットワークが高速になるほど顕著になる．図 2 にギガビットイーサネット（GbE）

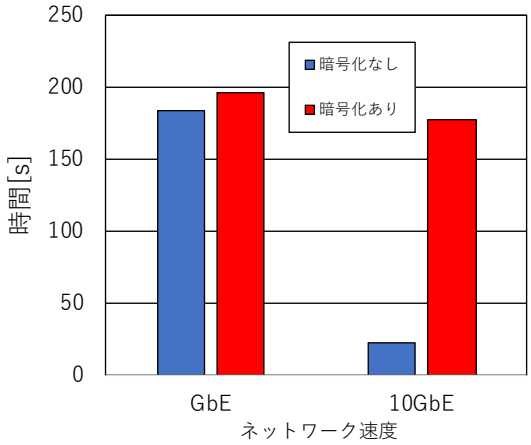

図 2 分割マイグレーションにおける暗号化のオーバヘッド

を用いた場合と 10GbE を用いた場合について，20GB のメモリを持つ VM の分割マイグレーションにかかる時間を示す．この測定には 5 章の実験環境を用いた．GbE の場合には転送するメモリデータを暗号化してもマイグレーション時間は少ししか増加しないが，10GbE の場合には約 8 倍になることが分かる．また，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，その間に盗聴が行われる危険性もある．サブホストの管理者に暗号鍵を盗まれた場合には容易にメモリデータを復号されてしまう．加えて，暗号化が必要な機密情報かどうかは考慮されず，すべてのメモリデータが一律に暗号化されることもオーバヘッドが大きくなる原因である．

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暗号化の最適化を行う *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を行わないようにすることで暗号化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，機密情報が含まれるメモリだけを選択的に暗号化することにより，さらに暗号化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暗号化の最適化

SEmigrate は図 3 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．

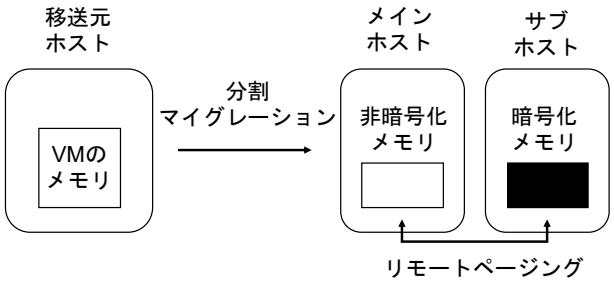

図 3 サブホストにおける暗号化の最適化

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．

3.2 VM 内情報に基づく選択的暗号化

SEmigrate は移送元ホストおよびメインホストでメモリを選択的に暗号化することで暗号化の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 4 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

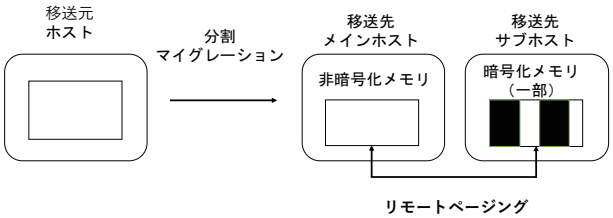

図 4 選択的なメモリ暗号化

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．空きメモリであっても機密情報が格納されたままになっている場合もあるため，SEmigrate はすべてのデータを 0 に置き換えて転送する．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．

ユーザによって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．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．一方，復号鍵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，図 5 のようにゲスト 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．

4. 実装

我々は分割マイグレーションとリモートページングをサポートした QEMU-KVM 2.11.2 に SEmigrate を実装した．選択的暗号化のために用いる 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 を用いた．メ

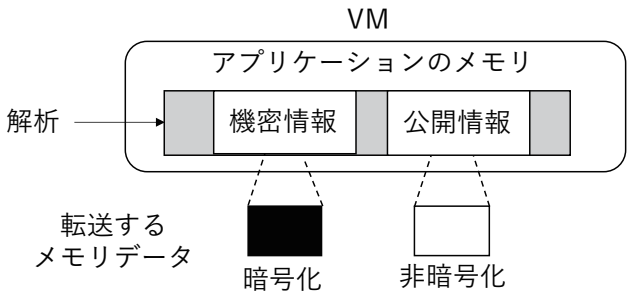

図 5 アプリケーション内のメモリ領域の選択的暗号化

モリデータの暗号化には OpenSSL の AES-ECB を用い，鍵長は 128 ビットとした．

4.1 暗号化ビットマップ

SEmigrate は暗号化ビットマップと呼ばれるビットマップを用いて，VM のメモリが暗号化されているかどうかを 4KB のページ単位で管理する．暗号化ビットマップは VM に割り当てられているメモリのページ数分のビットからなり，ページが暗号化されていれば対応するビットが 1 になり，暗号されていなければ 0 になる．

分割マイグレーション時に選択的暗号化によってメモリページを暗号化せずに転送する場合，SEmigrate は同時に移送先メインホストに非暗号化フラグを送信する．メインホストではこのフラグを基に転送されたページが暗号化されているかどうかを判別し，暗号化ビットマップを作成する．メインホストに転送されたページについては，非暗号化フラグを受信しなかった場合にのみ復号を行う．その結果，ページは暗号化されていない状態になるため，暗号化ビットマップの対応するビットは常に 0 にする．一方，サブホストに転送されたページについては，非暗号化フラグを受信した場合にはページが暗号化されていないため，対応するビットを 0 にする．フラグを受信しなかった場合はページが暗号化されているため，対応するビットを 1 にする．

ページイン時にはサブホストからページのデータを受信した時にメインホストで暗号化ビットマップを確認し，ビットが 1 の場合のみ復号する．ページアウト時にはメインホストで再度，選択的暗号化のための判定を行い，ページを暗号化した場合には暗号化ビットマップのビットを 1 にし，暗号化しなかった場合には 0 にする．現在の実装では，ページアウト時には選択的暗号化のための再判定は行わず，分割マイグレーション時に作成された暗号化ビットマップに基づいてページの暗号化を行っている．

4.2 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暗号化しないようにするために，VM のメモリ

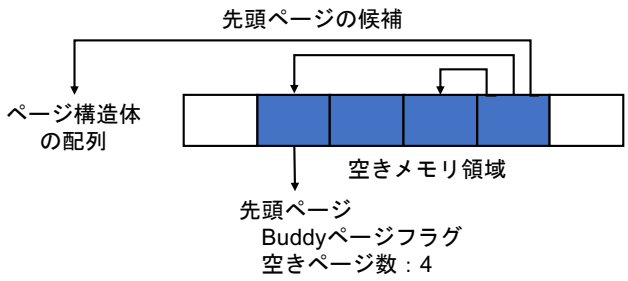

図 6 Linux における空きメモリの管理

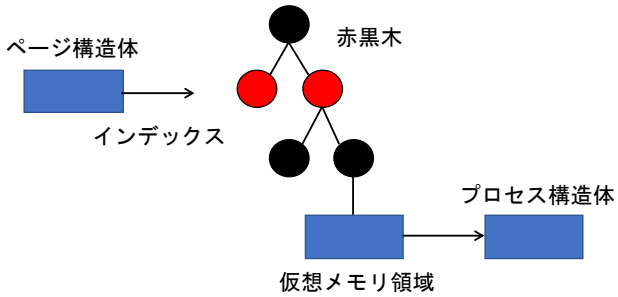

図 7 Linux の仮想メモリ領域の探索

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図 6 のように 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，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．

4.3 コード領域の判定

SEmigrate はコード領域を暗号化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．

4.4 プロセスメモリの判定

SEmigrate は機密情報を扱わないプロセスのメモリを暗号化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，4.3 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名を取得する．プロセス名が指定したものと一致すれば暗号化を除外するプロセスのページであると判定する．

4.5 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機密情報が含まれないメモリ領域を暗号化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，4.4 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，暗号化を除外するメモリ領域のページであると判定する．

プロセス内の暗号化を除外する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．例えば，当該メ

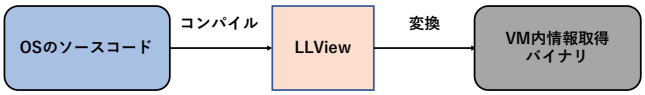

図 8 LLView による透過的な VM 内情報の取得

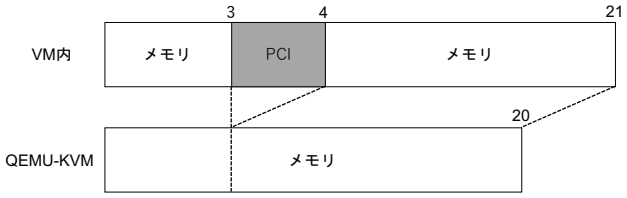

図 9 VM 内の物理メモリの再マップ

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考える．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．

4.6 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，図 8 のよう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図 9 のように 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．

4.7 メモリデータの整合性検査

SEmigrate では，分割マイグレーションとリモートページングの際に転送されるメモリデータが改ざんされていないことをハッシュ値を用いて検査する．分割マイグレーション時には，暗号化を行う前に SHA-256 を用いてメモリデータのハッシュ値を算出し，メモリデータとともに移送先ホストへ転送する．メインホストでは受信したメモリデータを必要に応じて復号した後でハッシュ値を算出し，受信したハッシュ値と一致するかどうかを確認する．サブホストではメモリデータが暗号化されていても復号しないため，受信したハッシュ値の確認は行わず，そのまま保持しておく．

そして，ページインの際にサブホストからメインホストへメモリデータとともにハッシュ値を転送し，メインホストで必要に応じてメモリデータを復号した後でハッシュ値の確認を行う．ページアウト時には，メインホストで転送するメモリデータのハッシュ値を算出してメモリデータとともにサブホストに転送する．メモリデータを暗号化しない場合でもハッシュ値を意図通りに改ざんできないようにするためには，メモリデータを暗号化するために使う鍵もハッシュ値の計算に含める必要がある．その実装については今後の課題である．

5. 実験

SEmigrate を用いて，転送するメモリデータを暗号化する際の分割マイグレーションとマイグレーション後の VM の性能向上を調べる実験を行った．比較として，常に暗号化・復号化を行う場合（常暗号），サブホストで復号を行わないようにしただけの場合（復号略），それに加えてそれぞれの選択的暗号化も行う場合，暗号化を行わない場合（非暗号）を用いた．選択的暗号化には，空きメモリ領域の非暗号化（未使用），コード領域の非暗号化（コード），特定プロセスの非暗号化（アプリ），特定プロセスの特定メモリ領域の非暗号化（データ）を用いた．

移送元ホストと移送先メインホストには Intel Core i7-8700 の CPU，64GB のメモリを搭載したマシンを用い，OS には Linux 4.18.17 を用いた．移送先サブホストには Intel Xeon E3-1226 v3 を搭載した CPU，32GB のメモリを搭載したマシンを用い，OS には Linux 4.18.20 を用いた．これらのマシンは 10 ギガビットイーサネットで接続した．VM には 1 個の仮想 CPU と 20GB の VM 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．この実験では，メモリ暗号化の最適化の効果について調べ

るため，メモリデータの整合性検査は行わなかった．

5.1 分割マイグレーション時間

SEmigrate を用いた際の分割マイグレーションの性能を調べるために，VM 内で大量のメモリを使用するアプリケーションを動かして分割マイグレーションを行った．その際に，アプリケーションが使用するメモリを 1〜18GB に変化させてマイグレーション時間を測定した．また，転送時に暗号化されなかったメモリ量についても測定した．特定プロセスの非暗号化を適用する際には，このアプリケーションを暗号化しないように指定した．特定プロセスの特定メモリ領域の非暗号化を適用する際には，このアプリケーション内で確保されたメモリ領域を暗号化しないようにした．分割マイグレーションにかかった時間を図 10 に，転送する際に暗号化されなかったメモリ量を図 11 に示す．

常にメモリデータの暗号化・復号化を行う場合（常暗号）と比較して，SEmigrate はマイグレーション時間を 82%以上削減することができた．この実験では，実行したアプリケーションが使用したメモリとゲスト OS の空きメモリの合計がほぼ一定となり，その両方を暗号化しなかったため，マイグレーション時間はアプリケーションが使用するメモリ量にほぼ依存しなかった．一方，暗号化を行わない場合（非暗号）と比べるとマイグレーション時間は 56%長くなった．これは暗号化されたメモリデータが 2.3GB あったこと，および，選択的暗号化の判定のオーバヘッドが原因である．

暗号化のそれぞれの最適化の効果を調べると，常に暗号化・復号化を行う場合と比べて，サブホストでの復号を行わなかった場合（復号略）は 24%高速化した．この場合，VM のメモリの半分の 10GB が復号されなかった．この場合と比べて，さらに空きメモリを暗号化しないようにした場合（未使用），アプリケーションが 1GB のメモリを使用した時は 69%高速化できたが，18GB 使用した時はまったく高速化できなかった．これは，アプリケーションが使用するメモリが増えたことにより暗号化が行われない空きメモリが減ったためである．

コード領域を暗号化しないようにした場合（コード）は，選択的暗号化を行わなかった場合とほぼ変わらなかった．むしろ，この選択的暗号化を行った場合のほうが遅くなることもあった．これはコード領域が相対的に小さく，すべてのページについて実行可能かどうかを調べるオーバヘッドがメモリデータを暗号化するオーバヘッドよりも大きくなったためである．特定プロセスのメモリを暗号化しないようにした場合（アプリ），対象アプリケーションの使用メモリが増えるほど暗号化されるメモリ量が減少し，マイグレーション時間が短くなった．アプリケーション内の暗号化しないメモリ領域を指定した場合（データ）も同様であった．

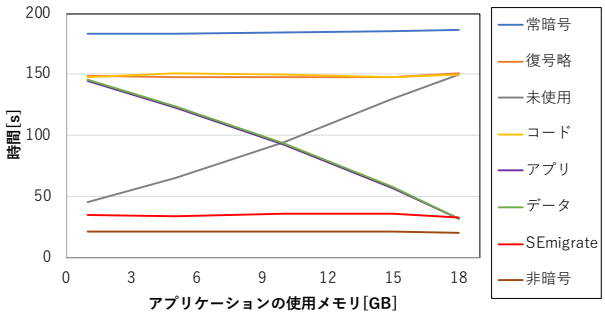

図 10 マイグレーション時間

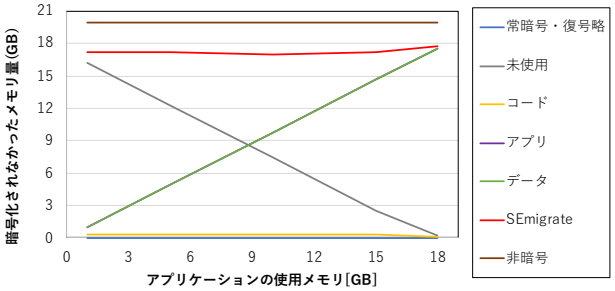

図 11 マイグレーション時に暗号化されなかったメモリ量

5.2 CPU 使用率

SEmigrate によりホストの負荷がどのような影響を受けるかについて調べるために，移送元ホストおよび移送先のメインホストとサブホストで分割マイグレーション中の CPU 使用率を測定した．図 12 に移送元ホストでの平均 CPU 使用率を示す．移送元ホストでは，SEmigrate を用いるとアプリケーションが使用するメモリ量が減るにつれて CPU 使用率が徐々に低くなった．アプリケーションが 18GB のメモリを使用した時には常に暗号化・復号化を行う場合（常暗号）と同程度であったのに対し，1GB のメモリを使用した時にはメモリデータの暗号化を行わなかった場合（非暗号）と同程度になった．SEmigrate において暗号化されなかったメモリ量は図 11 の結果よりほぼ一定であったため，空きメモリが多い場合にはページが空きメモリかどうかを判定するオーバヘッドが小さいということである．それ以外の選択的暗号化の負荷はアプリケーションが使用するメモリ量に関わらずほぼ一定であった．

図 13 に移送先メインホストでの平均 CPU 使用率を示す．移送先メインホストでは，SEmigrate を用いた場合でも CPU 使用率はほぼ一定であり，常に暗号化・復号化を行う場合の約半分に抑えることができた．サブホストで復号を行わないだけの場合（復号略）と比べると 30 ポイント増加したが，これは暗号化を行うメモリが減ったことによりメモリデータの転送レートが高くなったためであると考えられる．暗号化を行わない場合（非暗号）と比べると 10 ポイント低下したが，これは逆に暗号化を行うメモリが増えたことによりメモリデータの転送レートが低くなったためと考えられる．空きメモリのみを暗号化しない場合

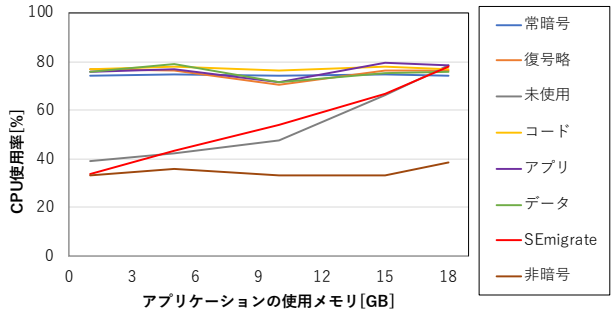

図 12 移送元ホストの CPU 使用率

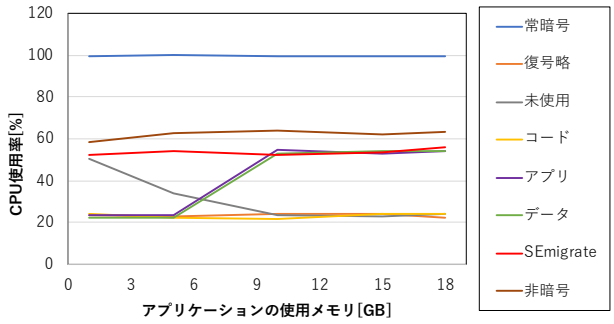

図 13 移送先メインホストの CPU 使用率

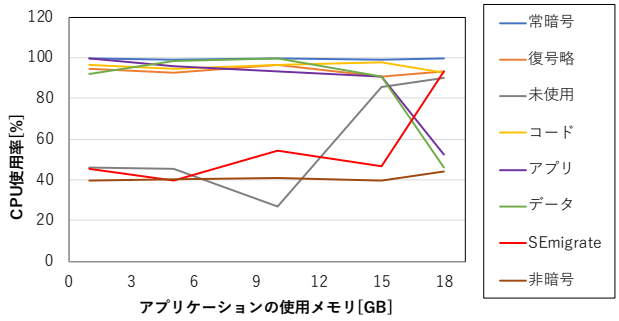

図 14 移送先サブホストの CPU 使用率

(未使用) には，アプリケーションのメモリ使用量が 10GB までは CPU 使用率が減少し，それ以上の場合には一定となった．逆に，特定プロセスのみを暗号化しない場合（アプリ，データ）には，アプリケーションのメモリ使用量が 10GB を超えると CPU 使用率が増加した．これらはメインホストに転送された 10GB のメモリデータだけしか復号されるかどうかの影響を受けないためである．

図 14 に移送先サブホストでの平均 CPU 使用率を示す．移送先サブホストでは，SEmigrate を用いると暗号化を行わない場合（非暗号）よりも少しだけ CPU 使用率が上昇した．ただし，アプリケーションが 18GB のメモリを使用した場合には選択的暗号化を行わない場合（復号略）と同程度まで上昇した．移送先サブホストにおいてこのような CPU 使用率になる原因は現在，調査中である．

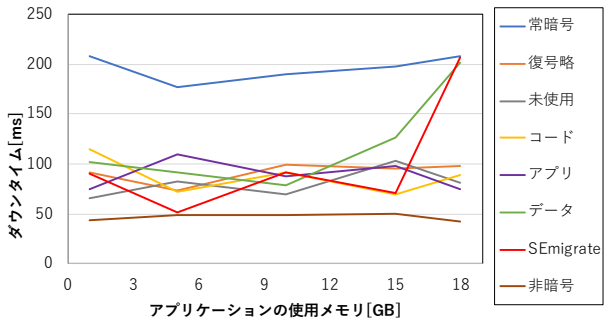

図 15 ダウンタイム

5.3 ダウンタイム

分割マイグレーションを行う際に VM が一時停止する時間（ダウンタイム）を図 15 に示す．SEmigrate を用いると，サブホストで復号を行わないようにしただけの場合（復号略）よりもダウンタイムが少し短くなった．これは VM が一時停止している間にメモリデータの一部を暗号化せずに転送できたためである．ただし，アプリケーションが 18GB のメモリを使用した場合にはダウンタイムが大幅に増加した．この原因は現在，調査中である．一方，暗号化しない場合（非暗号）と比べるとダウンタイムは少し長くなった．これはメモリデータの一部は依然として暗号化する必要があったためである．

5.4 マイグレーション後の VM 性能

SEmigrate を用いて分割マイグレーションを行った後の VM の性能を調べるために，VM 内で大量にリモートページングを発生させるアプリケーションの性能を測定した．このアプリケーションは 10GB のメモリを割り当て，そのメモリを 1 ページあたり 1 バイトずつ書き換える処理を繰り返した．この処理にかかった時間を図 16 に示す．常に暗号化・復号化を行う場合（常暗号）と比べて，SEmigrate は実行時間を 91%短くすることができた．これはアプリケーションが必要としたメモリをページインする際に，メモリデータの暗号化・復号化がほとんど行われなかったためである．一方，暗号化を行わなかった場合（非暗号）と比べると実行時間は 53%長くなった．これはページアウト時に一部のメモリが暗号化されたためであると考えられる．

暗号化のそれぞれの最適化の効果を調べると，常に暗号化・復号化を行う場合と比べて，サブホストで復号を行わない場合（復号略）には実行時間が 57%短縮された．しかし，空きメモリやコード領域を暗号化しないようにする最適化（未使用，コード）は効果がなかった．空きメモリやコード領域はほとんどリモートページングの対象にならなかったためである．選択的暗号化の中で効果があったのは，対象アプリケーションのメモリを暗号化しないようにする最適化（アプリ，データ）であった．これはアプリケーションのメモリに対してページインを行う際に復号化

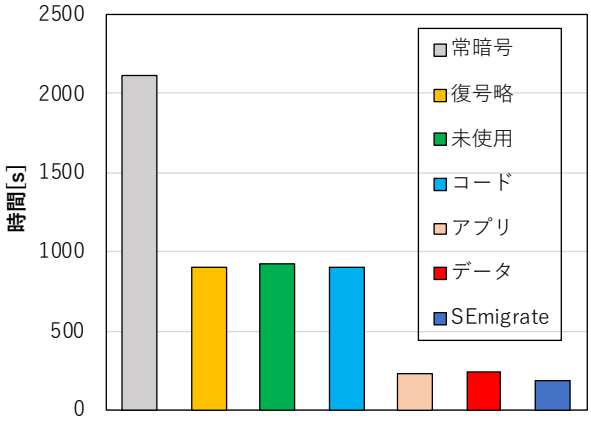

図 16 マイグレーション後のアプリケーション実行時間

のオーバヘッドが削減できたためである．

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[8] 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[9] 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 [10] 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[11] 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，FCtrans と組み合わせることでさらなる最適化が期待できる．

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暗号化を最適化する *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を行わないようにすることで暗号化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，機密情報が含まれるメモリのみを選択的に暗号化することで，さらに暗号化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暗号化しないようにする．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最大で 82%削減できることが分かった．また，マイグレーション後の VM の性能を最大 91%向上させられることも分かった．

今後の課題は，実際に利用されているアプリケーションの内部情報を用いて選択的なメモリ暗号化が行えるかどうかを調べ，性能を測定することである．また，メモリデータ内の機密情報の有無を調べた後，転送するまでの間にメモリの使用状況が変わった場合でも情報漏洩が発生しないようにする必要がある．

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

formation 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] 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).

[9] Chiang, J., Li, H. and Chiueh, T.: Live virtual machine migration with adaptive memory compression, *Proc. Virtual Execution Environment*, pp. 51–61 (2013).

[10] 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).

[11] 田内聡一朗, 光来健一: 複数ホストにまたがる大容量メモリ VM の未使用メモリに着目した高速化, *ComSys 2020*, pp. 9–17 (2020).