Vol.2014-ARC-210 No.5
Vol.2014-OS-129 No.5
2014/5/14

クラウドにおけるアプリケーション単位での VM 構成の動的最適化

三宮 浩太[1]　光来 健一[1]

概要：IaaS 型クラウドでは，ユーザは状況に応じて VM を用いたスケールインを行うことでコストを削減することが可能である．しかし，このような VM 構成の最適化では最低性能の VM が 1 台になった時にそれ以上の最適化を行うことができない．一方，さらなる最適化のためにアプリケーションを 1 台の VM に集約すると，集約時にアプリケーションを一旦停止させる必要がある上，アプリケーション間のセキュリティが弱まるという問題がある．本稿では，これらの問題を解決するために，アプリケーション単位での VM 構成の最適化を実現するシステム FlexCapsule を提案する．FlexCapsule はネストした仮想化を用いて VM の中の個々のアプリケーションを軽量な VM の中で動作させ，VM マイグレーションを用いることによりアプリケーションを停止することなく VM 構成の最適化を行う．また，VM による強い隔離によりアプリケーション間のセキュリティを保つ．我々は FlexCapsule を Xen に実装し，アプリケーションを無停止でマイグレーションできることを確認した．

Dynamic Optimization of VM Deployment on a Per-Application Basis in Clouds

Kouta Sannomiya[1]　Kenichi Kourai[1]

Abstract: In IaaS clouds, users can reduce their costs by scale-in using Virtual Machines (VMs) when necessary. However, optimization is not possible when users use a single VM whose performance is minimum. If users consolidate applications into a single VM for further optimization, the applications have to be stopped at consolidation and security between applications weakens. In this paper, to solve these problems, we propose FlexCapsule, which enables dynamic optimization of VM deployment on a per-application basis. FlexCapsule executes each application in a lightweight VM by nested virtualization and optimizes VM deployment without stopping applications by migrating the lightweight VMs. In addition, it maintains security between applications by strong isolation between VMs. We have implemented FlexCapsule in Xen and confirmed that applications could be migrated with no stops.

1. はじめに

近年，ネットワークを介してユーザにサービスを提供するクラウドコンピューティングの利用が広まっている．そのサービスの一つである Infrastructure as a Service (IaaS) 型クラウドではユーザに仮想マシン (VM) を提供する．IaaS 型クラウドの利点の一つとして，利用する VM の構成の変更を柔軟に行えることが挙げられる．自前のマシンで運用を行う場合は，運用開始前にシステムの最大負荷の予測を行ってからハードウェアを設置する必要がある．一方で，IaaS 型クラウドでは必要に応じて VM の台数を増減させることできるため，負荷の変化に迅速かつ容易に対応することができる．そのため，VM の台数が常に必要最低限になるように最適化して無駄な VM を減らし，台数に

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

応じて決まるコストを削減することができる．

しかし，VM の台数の増減では VM が 1 台になった後，台数をそれ以上減らすことができない．さらにコストを削減するには，性能の低い VM に切り替える必要がある．しかし，VM を切り替える際には，アプリケーションを停止させて，低い性能の VM で立ち上げ直す必要があり，サービスのダウンタイムが発生する．また，使用できる VM の性能は，クラウド事業者によって決められていることが多いため，提供されている最低性能の VM を使っていた時には，無駄なリソースがあったとしても，VM の性能を下げることができない．

さらなる最適化を行うためにはアプリケーション単位で最適化を行う必要がある．例えば，2 台の VM で別々の低負荷なアプリケーションが動作している時，それらを 1 つの VM に集約することによって VM の台数をさらに減らすことができる．しかし，VM の性能による最適化の場合と同様に，アプリケーションを集約する際にダウンタイムが発生する．また，同一 VM 上で複数のアプリケーションを動作させることで，アプリケーション間の隔離が弱くなるというセキュリティ上の問題も生じる．

本研究では，クラウドにおいてアプリケーション単位で VM 構成の最適化を行うことが可能なシステム FlexCapsule を提案する．FlexCapsule はネストした仮想化 [2] を用いて VM の中の個々のアプリケーションを軽量 VM の中で動作させる．そして，VM のマイグレーション技術を用いてアプリケーションを停止せずに移動させることを可能とする．アプリケーションが動作する VM 内では軽量なライブラリ OS である FlexCapsule OS を利用し，アプリケーションにリンクして動作させる．これにより，VM のメモリ量を削減し，アプリケーションをマイグレーションする時に発生するダウンタイムを削減することができる．また，VM 間の隔離により複数のアプリケーションを集約することによるセキュリティの低下も防ぐことができる．

我々は Xen 4.2 [1] を用いて FlexCapsule を実装した．FlexCapsule ではクラウド VM 内で準仮想化のアプリケーション VM を動作させ，FlexCapsule OS にサポートを追加することで，アプリケーション VM のマイグレーションを可能にする．FlexCapsule を用いたアプリケーション VM の動作確認を行い，マイグレーションを行うことによって発生するダウンタイムやマイグレーションにかかる時間の計測を行った．さらに，アプリケーション VM を高性能なクラウド VM にマイグレーションすることでアプリケーションの性能を向上させられることを確認した．

以下，2 章ではクラウドにおける VM 構成の最適化手法とそれに伴って発生する問題点について述べ，3 章では提案するシステムについて述べる．4 章では FlexCapsule の実装の詳細について述べ，5 章では FlexCapsule を用いて行った実験について述べる．6 章では関連研究に触れ，7 章でまとめと今後の課題を述べる．

2. VM 構成の最適化手法

IaaS 型クラウドにおける VM 構成の最適化は，VM 内のシステムの CPU 使用率やメモリ使用量などに応じて行われる．最適化手法として，**図 1** のような 3 つの手法が挙げられる．最も一般的な最適化手法は VM の数を増減させるスケールアウト・スケールインである．システムが高負荷になると，スケールアウトによってアプリケーションを動かす VM の台数を増やし，負荷を分散させる．逆に，システムが低負荷になると，スケールインによって VM の台数を減らし，VM のコストを削減する．しかし，VM が 1 台の状態でシステムが低負荷になっても，それ以上 VM の数を減らすことはできない．そのため，アプリケーションがほとんど動いていない時のコストの削減には限界がある．

一方，1 台の VM に対する最適化手法として，VM の性能を上下させるスケールアップ・スケールダウンがある．システムが高負荷になると，スケールアップによって VM に CPU やメモリを追加することで性能を上げる．逆に，システムが低負荷になると，スケールダウンによって VM の性能を下げ，VM のコストを削減する．既存の多くのクラウドでは実行中の VM の性能を変更することはできないため，提供されている何種類かの性能の VM を切り替えて使うことでスケールアップ・スケールダウンを実現することになる．性能変更のためにはアプリケーションを元の VM で停止させてから，データ等を最適な VM に移動し，アプリケーションを起動し直す必要がある．この期間はアプリケーションがサービスを提供できないダウンタイムとなる．また，提供されている最低性能の VM よりも性能を下げてコストを削減することはできない．

さらなる最適化を行うためには，複数のアプリケーションを 1 つの VM に統合する方法が考えられる．例えば，2

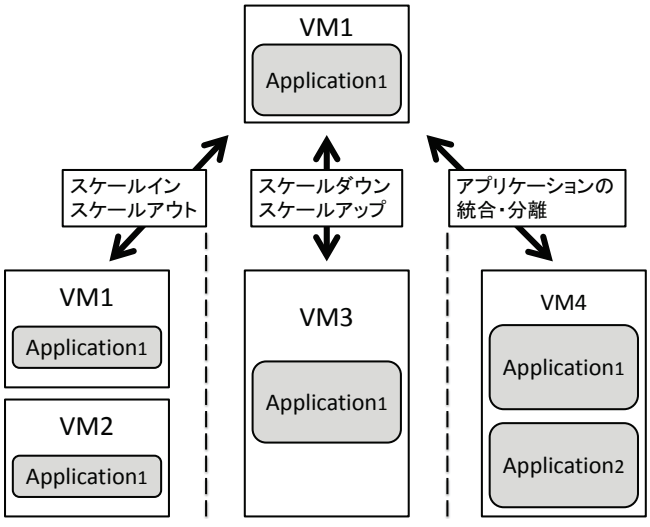

図 1 VM 構成の最適化

台の VM で別々の低負荷なアプリケーションが動作している時，それらを 1 つの VM に集約することによって VM を 1 台に減らすことができる．その後でシステムの負荷が高くなった場合には，アプリケーションを再び分離して，別々の VM で動作させることで負荷を分散させることができる．しかし，スケールアップ・スケールダウンの場合と同様に，アプリケーションを VM 間で移動させる際にダウンタイムが発生する．また，同一 VM 上で複数のアプリケーションを動作させることになるため，統合前よりもアプリケーション間の隔離が弱くなるというセキュリティ上の問題も生じる．

ダウンタイムを発生させずにアプリケーションを移動させるための手段としてプロセスマイグレーションが考えられる．しかし，プロセスは OS への依存度が高く，プロセスの状態を完全に保持したままマイグレーションを行うのは難しい．例えば，オープンされたファイルやネットワーク接続はマイグレーション時に閉じられることが多い [3]．Zap [4] ではプロセスと OS の間に薄い仮想化層を提供することでプロセスをほぼ完全にマイグレーションできる．しかし，プロセス間の隔離は十分ではないため，アプリケーション集約時のセキュリティが低下する．

3. FlexCapsule

本稿では, 個々のアプリケーションを軽量な VM の中で動作させることにより, クラウドにおけるアプリケーション単位での VM 構成の最適化を実現するシステム FlexCapsule を提案する．

3.1 アプリケーション VM

FlexCapsule では，クラウド事業者によって提供される VM（クラウド VM）の中でさらにアプリケーション用の VM（アプリケーション VM）を動作させる．1 つのアプリケーション VM の中では基本的に 1 つのアプリケーションだけを動かす．VM の中で VM を動かすために，FlexCapsule ではネストした仮想化を用いる．この技術により，クラウド VM 内で仮想化システムを動作させて，その上でアプリケーション VM とそれを管理するための管理 VM を動作させる．FlexCapsule のシステム構成を**図 2** に示す．区別のために，クラウドの仮想化システムにおけるハイパーバイザをホストハイパーバイザ，ホストハイパーバイザ上のクラウド VM 内で動作するものをゲストハイパーバイザと呼ぶ．

既存のクラウド環境でアプリケーション VM へのネットワークアクセスを可能にするために，FlexCapsule では Network Address Port Translation（NAPT）を用いる．クラウド VM に割り当てられた IP アドレスは管理 VM が利用し，アプリケーション VM にはプライベート IP アドレスを割り当てる．アプリケーション VM が外部と通信を行えるようにするために，静的 NAPT によりポート番号に基づいてクラウド VM の IP アドレスとアプリケーション VM の IP アドレスの相互変換を行う．例えば，クラウド VM の 80 番ポートにアクセスすると，ウェブサーバを提供するアプリケーション VM の 80 番ポートへのアクセスに変換される．

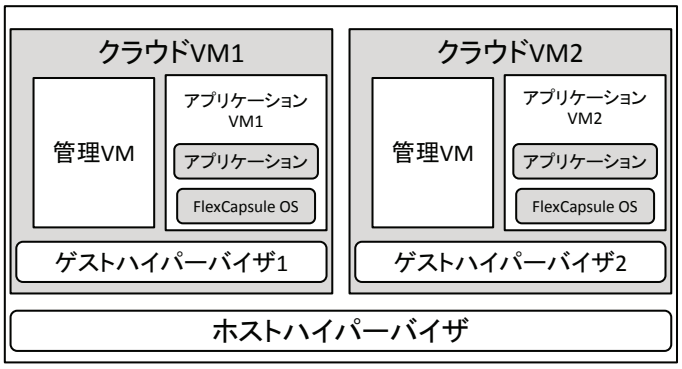

図 2 FlexCapsule のシステム構成

3.2 FlexCapsule OS

FlexCapsule では，アプリケーション VM 用の OS として軽量なライブラリ OS である FlexCapsule OS を提供する．これは，単純に 1 つのアプリケーションを 1 つの VM で動作させ，それぞれで汎用 OS を動かすとリソース消費量が増えてしまうためである．ライブラリ OS は OS の機能をライブラリとして提供し，アプリケーションから利用することを可能とした OS である．アプリケーションの実行に必要な機能しかリンクされないため，従来の汎用 OS を用いた場合よりもメモリ使用量が少なくて済む．このため，クラウド VM 内で複数のアプリケーション VM を動作させても，メモリの消費を抑えることができる．

3.3 アプリケーション VM のマイグレーション

FlexCapsule では VM のマイグレーション技術を利用して，クラウド VM 間でアプリケーションを移動する．この際に，**図 3** のようにアプリケーションだけでなく OS も含んだアプリケーション VM 全体のマイグレーションを行う．これにより，状態を完全に保持したまま，アプリケーションを移動させることができる．プロセスと OS の間の依存度と比べて，アプリケーション VM とハイパーバイザの間の依存度は低いため，プロセスマイグレーションのような問題は生じにくい．

アプリケーション VM のマイグレーションは通常の VM のマイグレーションより高速に行うことができる．マイグレーション時にはアプリケーション VM のメモリをマイグレーション先のホストに転送するため，マイグレーション時間はアプリケーション VM のメモリサイズに依存する．アプリケーション VM のメモリ使用量は少ないため，マイ

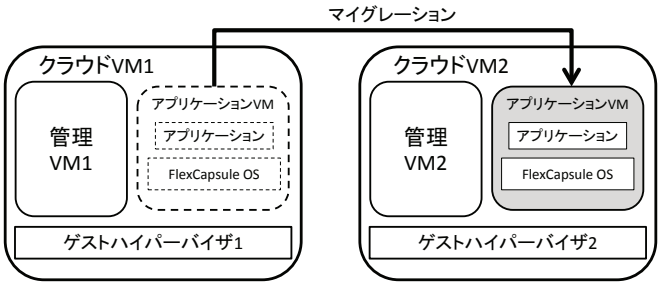

図 3 アプリケーション VM のマイグレーション

グレーション時間も短くできる．また，ライブマイグレーションでは，転送中にアプリケーション VM の状態が変化することにより発生するメモリの差分を繰り返し転送し，最終段階では VM を停止させて残りの差分を転送する．アプリケーション VM のメモリの転送にかかる時間が短いことからこの差分も小さくなると考えられ，マイグレーション時間の削減およびダウンタイムの短縮につながる．

3.4 VM 構成の最適化

クラウド VM のスケールアップ・スケールダウンによる VM 構成の最適化では，FlexCapsule を用いてアプリケーション VM を元の VM から最適な性能の VM に対してマイグレーションを実行する．これにより，アプリケーションを動かしたまま，そのデータとともに移動させることができるため，ダウンタイムをほとんど発生させずに最適化が可能となる．

複数のアプリケーションを 1 つの VM に統合する最適化では，スケールアップ・スケールダウンと同様に統合時にアプリケーション VM のマイグレーションを行うことでダウンタイムの発生を抑えることができる．さらに，個々のアプリケーション VM の間は隔離が強く，他の VM のリソースにアクセスすることはできない．そのため，同一クラウド VM 内で複数のアプリケーション VM を動かしてもセキュリティ上のリスクは小さい．

3.5 アプリケーション VM の実行環境

FlexCapsule ではアプリケーション VM を従来のプロセスと同様に扱うことのできる実行環境を提供する．これにより，ユーザはアプリケーションを VM 内で動かしていること意識せずに管理できるようにする．この実行環境はクラウド VM 内の管理 VM 上に作られ，従来のプロセスの管理に用いるコマンドでアプリケーション VM の管理を行うことを可能にする．例えば，実行中のアプリケーションの情報を表示する場合，プロセスの場合には ps コマンドが用いられるが，アプリケーション VM の場合には VM の一覧を表示するコマンドを用いる必要がある．FlexCapsule の提供する実行環境では，アプリケーション VM に対しても ps コマンドによるアプリケーション情報の取得を可能とする．

4. 実装

我々は FlexCapsule を Xen 4.2 を用いて実装した．

4.1 Xen におけるネストした仮想化

ネストした仮想化環境を構築するために，**図 4** のようにクラウド VM を動作させるためのホストハイパーバイザとして Xen を動作させ，クラウド VM 内でも同様に Xen を動作させた．ホストハイパーバイザの上でクラウド VM としてドメイン U を完全仮想化で動作させ，ゲストハイパーバイザの上でアプリケーション VM としてドメイン U を準仮想化で動作させた．準仮想化を用いたのはネストした仮想化のオーバヘッドを削減するためである．クラウド VM 内の管理 VM にはドメイン 0 を用いた．

4.2 FlexCapsule OS

FlexCapsule OS は Xen でサポートされている軽量 OS である Mini-OS をベースに実装を行った．Mini-OS は従来の Xen ではドメイン 0 で動いているコンポーネントを個々の独立した VM として動作させることを目的として利用されることが多い．Mini-OS はノンプリエンプティブな OS であり，Mini-OS 内のカーネルレベルで C や OCaml で書かれたアプリケーションを動作させることができる．

Mini-OS では，コンソールやネットワークなどのデバイスドライバが提供されており，ドメイン 0 のバックエンドドライバと通信して I/O を行うことができる．例えば，ドメイン 0 からのキーボードの入力情報はドライバ間で情報を受け渡すためのコンソールリングと呼ばれるバッファに書き込まれる．その後，バックエンドドライバはイベントチャネルを使って Mini-OS に入力があることを通知する．これにより，Mini-OS はコンソールリングよりキーボードからの入力情報を取得する．ネットワークに関しては，Mini-OS は軽量な TCP/IP プロトコルスタックの LwIP を用いる．IP アドレスと MAC アドレスは VM の設定ファイルに記述して割り当てる．Mini-OS では起動時にネットワークフロントエンドが専用のスレッドを作成し，ドメイ

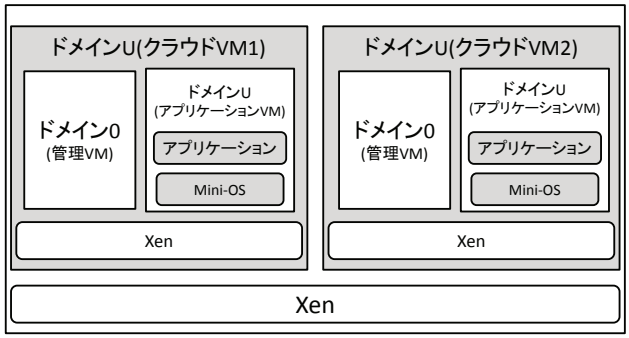

図 4 Xen を用いた FlexCapsule の構成

ン 0 のバックエンドドライバからパケットを受け取る．

Mini-OS はアプリケーションに対して標準 C ライブラリのサブセットを提供する．しかし，マルチプロセスをサポートしていないために fork などプロセスに関する関数はサポートしていない．また，Xen 4.1 以降の Mini-OS では安定性の問題よりファイルシステムドライバが削除されているため，ファイル関連の関数を使用することができない．

4.3 マイグレーションのサポート

FlexCapsule OS では Mini-OS をベースとしてマイグレーションのサポートを行った．

4.3.1 マイグレーションの流れ

マイグレーション元の管理 VM はマイグレーションを開始すると，アプリケーション VM のメモリの内容をマイグレーション先に転送する．ライブマイグレーションでは転送中に書き換えられたメモリの差分を再度，転送する．この差分が十分小さくなったら，管理 VM はアプリケーション VM に対してサスペンド要求を送り，アプリケーション VM をサスペンド状態とする．その状態で，メモリの差分と CPU の状態をマイグレーション先へ転送する．マイグレーション先の管理 VM はメモリの差分と CPU の状態を反映させてからアプリケーション VM のレジュームを行い，マイグレーションを完了する．

準仮想化ゲストであるアプリケーション VM をマイグレーションするにはゲスト OS のサポートが必要である．これは，準仮想化ではゲスト OS とハイパーバイザが密接に連携して動作しているためである．FlexCapsule OS が管理 VM からのサスペンド要求を受け取ると，シャットダウンハンドラを呼び出してサスペンド処理を行う．マイグレーション先でアプリケーション VM が再開されると，FlexCapsule OS はレジューム処理を行う．

4.3.2 シャットダウンハンドラ

シャットダウンハンドラは管理 VM からシャットダウンやリブート，サスペンド要求を受け取った時にゲスト OS によって呼び出されるイベントハンドラである．管理 VM は VM ごとに XenStore に control/shutdown ノードを作成し，そのノードに要求を書き込む．ゲスト OS は要求を受け取るとシャットダウンハンドラを呼び出し，要求に応じた処理を行う．

Mini-OS にはこのような機構が実装されていなかったため，FlexCapsule OS に電源処理要求を受け取るための機構とシャットダウンハンドラを実装した．FlexCapsule OS は起動時に専用スレッドを立ち上げ，XenBus 経由で XenStore の control/shutdown ノードを監視する．マイグレーション時に管理 VM から control/shutdown ノードに対して要求の書き込みが発生した場合，XenBus からのイベント通知でそれを検知したアプリケーション VM はシャットダウンハンドラを呼び出してサスペンド処理を行う．アプリケーション VM を管理 VM から制御できるようにするために，サスペンド要求だけでなく，シャットダウン要求にも対応した．

4.3.3 サスペンド処理

FlexCapsule OS ではコンソール，ネットワーク，XenStore，タイマ，グラントテーブル，P2M テーブルに関してサスペンド処理を行う．FlexCapsule OS では管理 VM 上のバックエンドドライバとやりとりするのにイベントチャネルを使用する．そのため，アプリケーション VM と管理 VM の間ではネットワークやコンソールで利用するイベントチャネルが確立されている．マイグレーション後はアプリケーション VM が別のクラウド VM 上の管理 VM との間で新たにイベントチャネルを確立する必要があるので，サスペンド時にはイベントチャネルの切断を行う．

ネットワークについては，マイグレーション後にマイグレーション前と同じ IP アドレスや MAC アドレスを使って立ち上げるために，サスペンド時にはネットワークの機能を停止させる．この時，ネットワークの専用スレッドは停止させずに実行可能状態にしておくことで，マイグレーション前の状態を引き継いでレジューム処理を行えるようにする．

FlexCapsule OS は XenBus を用いて XenStore との通信を行っている．XenBus への書き込み通知や XenBus からのイベント通知にはイベントチャネルが用いられているため，サスペンド時にはこのイベントチャネルの切断を行う．また，XenBus スレッドを待機状態にする．

タイマハンドラは一定時間ごとに発生する仮想タイマ割り込みによって呼び出される．仮想タイマ割り込みはハイパーバイザから VM に送られるため，FlexCapsule OS はハイパーバイザとの間でイベントチャネルを確立することで割り込みを受信する．マイグレーション先ではイベントチャネルを再確立する必要があるため，サスペンド時にはイベントチャネルを切断する．

FlexCapsule OS はアプリケーション VM 内のメモリを読み書きするために管理 VM に与えた権限をグラントテーブルを用いて管理している．マイグレーション元のグラントテーブルはマイグレーション先では使用することができないため，サスペンド時にグラントテーブルの内容を削除する．

FlexCapsule OS は VM 内の疑似物理メモリをマシンメモリに変換するための P2M テーブルを持っている．P2M テーブルは**図 5** のように 3 つのレベルを持つ木構造で構成されており，各ノードはホストごとに固有のマシンフレーム番号 (mfn) でリンクされている．しかし，マイグレーション先のホストではアプリケーション VM に割り当てられるマシンメモリが変わるため，P2M テーブルを引くことができなくなる．そこで，FlexCapsule OS ではサスペンド時に L3_list，L2_list，L1_list の仮想アドレスを保存し

Vol.2014-ARC-210 No.5
Vol.2014-OS-129 No.5
2014/5/14

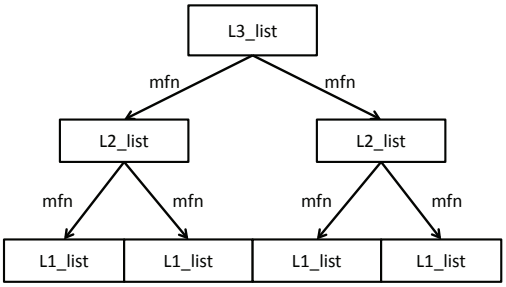

図 5 P2M テーブルの構造

ておく．これは仮想アドレスであればマイグレーション先でも変わらないためである．

これらの処理が完了したら FlexCapsule OS はアプリケーション VM のサスペンドを行うためのハイパーコールを発行する．このハイパーコールは引数としてアプリケーション VM 内にある start_info 構造体の mfn を必要とする．start_info にはコンソールや XenStore の通信に使われる共有メモリなどの情報が格納されている．この情報はマイグレーション先でレジュームを行う際に使用される．

4.3.4 レジューム処理

マイグレーションが完了してアプリケーション VM がレジュームされた後，FlexCapsule OS はレジューム処理を行う．まず，FlexCapsule OS はハイパーバイザと共有している shared_info 構造体に必要な情報を格納する．shared_info 構造体には時刻や仮想 CPU に関する情報が保存されている．これらを復元することにより，スケジューリングやイベントチャネルによるイベントの送受信がマイグレーション後も正常に行われるようになる．

次にサスペンド状態にある各種機能のレジュームを行う．コンソール，ネットワーク，XenBus，タイマに関してはイベントチャネルの再確立を行い，XenStore にはデバイス情報の書き込みを行う．また，XenStore のノードの監視が解除されているため XenBus に監視の再登録を行う．ネットワークについては，VM の設定ファイルを基に IP アドレスや MAC アドレスの再設定を行い，マイグレーション前のネットワーク処理を再び実行できるようにする．さらに，移動先の管理 VM に対するグラントテーブルの作成を行う．

また，FlexCapsule OS は P2M テーブルの再構築を行う．まず，サスペンド処理で保存しておいた L2_list の仮想アドレスを新しいマシンフレーム番号に変換し，L3_list から再リンクする．L3_list は保存しておいた仮想アドレスを用いてアクセスすることができる．次に，L1_list の仮想アドレスを新しいマシンフレーム番号に変換し，L2_list から再リンクする．FlexCapsule OS は L1_list を連続領域に確保し，先頭の仮想アドレスを保持しているため，すべての L1_list を容易に見つけられる．L1_list の中のマシンフレームはレジューム前に管理 VM によって再構築済みである．

5. 実験

FlexCapsule を用いてアプリケーション VM 内でアプリケーションが正常に動作することを確認する実験を行った．また，アプリケーション VM のマイグレーションを行い，マイグレーション後に正常にアプリケーション VM が動作することを確認する実験を行った．同時に，マイグレーションにかかる時間とダウンタイムの計測を行った．さらに，アプリケーション単位での VM 構成の最適化の効果を確かめるために，アプリケーション VM のマイグレーションを用いたクラウド VM のスケールアップを行った．実験には Intel Xeon E3-1290 3.70 GHz の CPU，8 GB のメモリを搭載したマシンを使用した．システム全体の仮想化とクラウド VM 内の仮想化のためにどちらも Xen 4.2.2 を用い，クラウド VM を管理するための管理 VM では Linux 3.5.0 を動かした．クラウド VM には 2 CPU と 1 GB のメモリを割り当て，クラウド VM 内の管理 VM では Linux 3.8.0 を動かした．今回の実験では，アプリケーション VM に対する NAPT 変換は行わなかった．

5.1 アプリケーション VM の動作確認

アプリケーション VM 内で正しくアプリケーションが動作することを確認するために，まず，一定周期でコンソール出力を行うアプリケーションを動作させた．アプリケーション VM のコンソールから確認したところ，正常に出力が行われていた．次に，アプリケーション VM 内で echo サーバを動かした．外部のクライアントからのアクセスを行ったところ，正常にコネクションを確立して通信を行うことができた．この結果より，標準 C ライブラリのいくつかの関数を利用したアプリケーションがアプリケーション VM 内で正常に動作することが確認できた．しかし，子プロセスを生成するために fork 関数を実行したところ，FlexCapsule OS のベースである Mini-OS によってサポートされていないため警告が表示された．他に FlexCapsule OS で使用できない関数を調べたところ，ファイルシステムやプロセス関係の関数の他に，動的ライブラリやシグナルに関するものがサポートされていないことが確認できた．

5.2 アプリケーション VM のマイグレーション

アプリケーション VM のマイグレーションを行い，正常にマイグレーションが完了することの確認を行った．アプリケーション VM 内で echo サーバを立ち上げ，マイグレーション前に外部とのコネクションを確立した．このアプリケーション VM をマイグレーションした後でクライアントから文字列を送信したところ，正常にアプリケーション VM 内のサーバから返答が返ってきた．この結果より，アプリケーション VM でサービスを提供中にマイグレーションを行っても移動先で正常に再開し，サービスに影響

を及ぼさないことが確認できた．

次に，アプリケーション VM のマイグレーションにかかる時間と発生するダウンタイムの計測を行った．マイグレーション時間はユーザがマイグレーション要求を出してから，マイグレーションが完了するまでの時間であり，この間にサービスが停止している時間をダウンタイムとした．アプリケーション VM のメモリサイズを変化させて，それぞれのメモリサイズで 10 回ずつ計測したときのマイグレーション時間とダウンタイムの平均値をそれぞれ**図 6**，**図 7** に示す．この結果より，マイグレーション時間はメモリサイズと比例関係にあるが，ダウンタイムは常に 0.2 秒ほどになっており，メモリサイズの影響を受けないことが確認できた．

5.3 クラウド VM のスケールアップの効果

アプリケーション VM を用いた際のクラウド VM のスケールアップの効果を調べるために，クラウド VM のスケールアップ時のアプリケーション VM の性能の変化を調べた．クラウドにおけるスケールアップを行うために，Xen の credit スケジューラを用いて性能差を生じさせた 2 つのクラウド VM を用意した．credit スケジューラは各 VM の CPU 使用率に上限を設定することができる．スケールアップ前のクラウド VM1 では CPU 使用率を 70 %に制限し，スケールアップ後のクラウド VM2 では制限を行わなかった．アプリケーション VM には 1 つの CPU，メモリ 4 MB を割り当てた．アプリケーションとして，整数や浮動小数点などの四則演算，ベクトル演算などを行うアプリケーションを実行し，処理時間をそれぞれのクラウド VM で計測した．

計測結果を**図 8** に示す．この結果より，クラウド VM の性能を約 1.4 倍向上させることでアプリケーション VM の性能は約 1.5 倍に向上していることが確認できた．これにより，クラウド VM をスケールアップさせてアプリケーション VM をマイグレーションさせることでアプリケーションの性能も向上させることができることが確認できた．

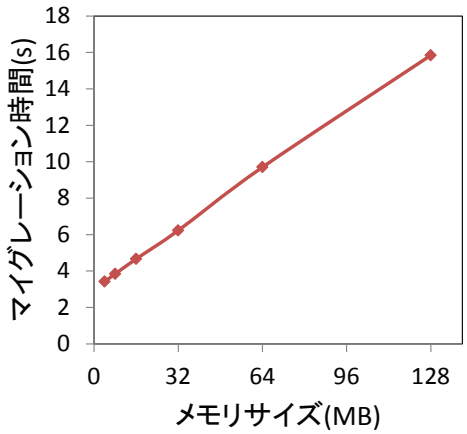

図 6 マイグレーション時間の計測結果

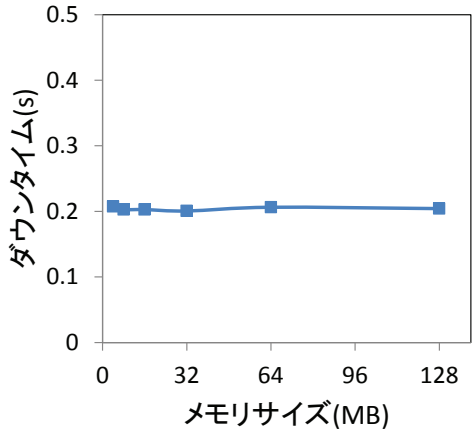

図 7 ダウンタイムの計測結果

6. 関連研究

ライブラリ OS [7] では，OS が持っていた機能の大部分をライブラリとしてアプリケーションにリンクすることで，アプリケーションが独自のリソース管理を行うことを可能とする．各アプリケーションの特性を考慮してライブラリ OS をカスタマイズすることにより，アプリケーションの性能を高めることができる．しかし，カーネルとライブラリ OS は密接に結びついており，マイグレーションを行うのは難しい．

DrawBridge [9] は Windows の一部の機能をライブラリ OS としてアプリケーションに提供しており，既存の Windows アプリケーションを動作させることができている．DrawBridge を用いることにより，アプリケーションのマイグレーションが容易になり，スナップショットのサイズも小さくすることができる．また，それぞれのアプリケーションの独立性が高くなり，特定のアプリケーションへの攻撃がホスト OS や他のアプリケーションに与える影響を小さくできる．

Libra [5] や GUK [6] はライブラリ OS を用いて Java VM を Xen のハイパーバイザの上で動作させ，Java アプリケー

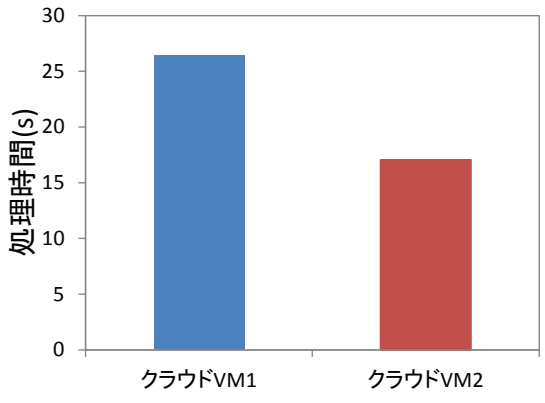

図 8 性能の異なるクラウド VM におけるアプリケーションの性能

ションの高速化を行っている．Libra では Java VM の性能に影響を及ぼす機能だけをライブラリ OS が提供し，その他のファイルシステムやネットワークなどの機能は同じハイパーバイザ上で動く管理 VM が提供する．これにより，OS の機能をすべてライブラリ OS に実装することなく，Java VM に特化したライブラリ OS の提供を可能とする．GUK は Mini-OS を拡張して Java VM を動作させている．Java VM を動作させるために Mini-OS のメモリ管理が改良されており，複数 CPU のサポートやメモリのバルーニングなどが追加されている．さらに，VM のサスペンド，レジュームも可能となっている．GUK は Xen 3.4 での Mini-OS をベースとしており，Xen 4.2 では動作させることができなかった．FlexCapsule OS の機能の一部は GUK を参考にして実装した．

Mirage [8] はコンパイル時にライブラリ OS を OCaml アプリケーションに特化させた Unikernel を生成し，ハイパーバイザ上で動作させる．Unikernel を用いるとコンパイル時にアプリケーションに必要な機能だけを選ぶことができる．それにより，バイナリサイズを小さくでき，クラウドで実行するコストを削減することができる．さらに，コンパイル時の特化や型安全性，ランタイムやコンパイラのバグからアプリケーションを守る仕組みによりセキュリティを高めている．

Zap は OS のプロセスの透過的なマイグレーションを可能とするシステムである．Zap では pod と呼ばれる薄い仮想化層の上でアプリケーションのプロセスグループを動作させる．pod が依存関係のあるプロセスをひとまとまりにし，OS のリソースを仮想化する．これによりプロセスと OS の間の直接的な依存性を除去することができ，プロセスのほとんどすべての状態を保持したままマイグレーションが可能となる．ただし，VM 間の隔離と比べると，pod 間の隔離はそれほど強くない．

GMO クラウド Public は GMO インターネットが提供する IaaS 型クラウドサービスである．既存の多くの商用クラウドでは実行中の VM を停止させずにその性能を変更することはできない．これに対して GMO クラウド Public では VM を停止することなく VM のメモリや CPU コア数の増設といった性能変更を行うことができる．しかしながら，GMO クラウド Public で提供されるのは VM 単位での最適化であるため，無停止で複数のアプリケーションを一つの VM に統合するといったアプリケーション単位での最適化は行えない．

7. まとめ

本研究では，クラウドにおいて個々のアプリケーションを軽量な VM の中で動作させることにより，アプリケーション単位での VM 構成の最適化を実現するシステム FlexCapsule を提案した．FlexCapsule はライブラリ OS を用いたアプリケーション VM のマイグレーション，および，アプリケーション VM を管理するための実行環境を提供する．我々は FlexCapsule を用いてアプリケーション VM のマイグレーションを行い，クラウド VM のスケールアップによる最適化を行えることを示した．

今後の課題は，アプリケーション VM の実行環境の開発を行うことである．また，FlexCapsule OS で幅広いアプリケーションをサポートするために，従来のアプリケーションで使用されているプロセス制御，ファイル関連の機能を使用できるようにする．さらに，CPU コア数の増加やメモリの増設による VM のスケールアップに対応するために複数の CPU やメモリバルーニングのサポートを行う．

参考文献

[1] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and the Art of Virtualization, *In Proceedings of the 19th Symposium on Operating Systems Principles*, pp. 164–177 (2003).

[2] Ben-Yehuda, M., Day, M. D., Dubitzky, Z., Factor, M., NadavHar'El, Gordon, A., Liguori, A., Wasserman, O., Yassour, B.-A. : The Turtles Project: Design and Implementation of Nested Virtualization, *In USENIX Symposium on Opearting Systems Design & Implementation(OSDI)*, pp. 423–436 (2010).

[3] Plank, J. S., Beck, M., Kingsley, G. and Li, K.: Libckpt: Transparent Checkpointing under Unix, *Usenix Winter Technical Conference*, pp. 213–223 (1995).

[4] Osman, S., Subhraveti, D., Su, G. and Nieh, J.: The Design and Implementation of Zap: A System for Migrating Computing Environments, *In Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, pp. 361–376 (2002).

[5] Ammons, G., Appavoo, J., Butrico, M., Silva, D. D., Grove, D., Kawachiya, K., Krieger, O., Rosenburg, B., Hensbergen, E. V. and Wisniewski, R. W.: Libra: A Library Operating System for A JVM, *In Proceedings VEE '07 Proceedings of The 3rd International Conference on Virtual Execution Environments*, pp. 44–54 (1995).

[6] Jordan, M. and Roeck, H.: Guest VM Microkernel, GUK (2009).

[7] R.Engler, D., Kaashoek, M. and J.O'Toole, L.: Exokernel: an Operating System Architecture for Application-Level Resource Management, *In Proceedings of the 15th ACM Symposium on Operating System Principles (SOSP)* (1995).

[8] Madhavapeddy, A., Mortier, R., Rotsos, C., Scott, D., Singh, B., Gazagnaire, T., Smith, S., Hand, S. and Crowcroft, J.: Unikernels: Library Operating Systems for the Cloud, *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 461–472 (2013).

[9] Baumann, A., Bond, B., Howell, J., Hunt, G. and Meyers, B.: Drawbridge: A New Form of Virtualization for Application Sandboxing (2011).