Vol.2015-OS-135 No.1
Vol.2015-EMB-39 No.1
2015/11/24

クラウドにおけるライブラリOSを用いたインスタンス構成の動的最適化

三宮 浩太[1]　光来 健一[1]

概要：IaaS 型クラウドでは，ユーザはアプリケーションの利用率が低い時にインスタンスのスケールインやスケールダウ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，このようなインスタンス構成の最適化では利用率の低いアプリケーションを動かすためであっても最低性能のインスタンスが最低 1 台必要となり，それ以上のコスト削減を行うことができない．さらなる最適化のために複数のアプリケーションを 1 台のインスタンスに統合することも考えられるが，統合時にアプリケーションを一旦停止させる必要がある上，統合後のアプリケーション間の隔離が弱くな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，ライブラリ OS を用いてインスタンス構成の動的な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とライブラリ OS を用いてインスタンスの中の個々のアプリケーションを軽量な VM の中で動作させる．そして，VM マイグレーションの技術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インスタンス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 の Mini-OS および OSv を用いて実装し，FlexCapsule におけるいくつかの機能の性能を調べた．

1. はじめに

Infrastructure as a Service (IaaS) 型クラウドではユーザにインスタンスと呼ばれる仮想マシン (VM) を提供する．IaaS 型クラウドでは必要に応じてインスタンスの構成を柔軟に変更できるため，負荷の変化に迅速かつ容易に対応することができる．例えば，サービス開始時は最小構成で運用を行い，アプリケーションの負荷が高い場合にはインスタンスの台数を増やして負荷に対応できる．逆に，負荷が低くなった場合はインスタンスの台数を減らしてコストを抑えることも容易である．そのため，IaaS 型クラウドではインスタンスの構成が常に必要最低限になるように最適化することが求められる．

しかし，インスタンスの台数の増減では利用率の低いアプリケーションを動かすためであってもインスタンスが最低 1 台必要となり，それ以上台数を減らしてコスト削減を行うことができない．さらにコスト削減を行うためには，より性能の低いインスタンスに切り替える必要があるが，使用できるインスタンスの性能はクラウド事業者によって固定されていることが多い．そのため，すでに最低性能のインスタンスを使っている時には，アプリケーションがそのインスタンスの性能を必要としていなかったとしても，それ以上コスト削減を行うことができない．

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 台のインスタンス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 つのインスタンスに統合することによってインスタンスの台数をさらに減らすことができる．しかし，統合の際にアプリケーションを停止させて統合先のインスタンスで起動し直す必要があるため，サービスのダウンタイムが発生する．また，同一インスタンス上で複数のアプリケーションを動作させなければならないため，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．

本稿では，ライブラリ OS を用いてアプリケーション単位でのインスタンス構成の動的な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 [1] を用いてインスタンスの中の個々のアプリケーションを**アプリケーション** *VM* と呼ばれる VM 内で動作させる．また，各アプリケーションに *FlexCapsule OS* と呼ばれるライブラリ OS を提供することでアプリケーション VM のメモリ量を削減し，軽量化を図る．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インスタンス間で移動させることを可能にする．また，VM 間の強い隔離により複数のアプリケーショ

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

ンを 1 台のインスタンスに統合する際のセキュリティの低下も防ぐことができる．

我々は Xen の Mini-OS および OSv [2] を用いて FlexCapsule を実装した．アプリケーション VM のマイグレーションを可能とするために，FlexCapsule OS にサスペンド・レジュームのサポートを追加した．また，アプリケーション VM の管理やアプリケーション VM 内では実現できない機能を提供するために，OS サーバを開発した。実験として，FlexCapsule を用いてアプリケーション VM のマイグレーション時に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 内で動くアプリケーションと従来の汎用 OS 上で動くアプリケーションの性能比較や，インスタンス性能の変更によるアプリケーション性能の変化を調べた．

以下，2 章ではクラウドにおけるインスタンス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. インスタンス構成の最適化手法

IaaS 型クラウドにおけるインスタンス構成の最適化は，インスタンス内のシステムの CPU 使用率やメモリ使用量などに応じて行われる．最も一般的な最適化手法はインスタンスの数を増減させるスケールイン・スケールアウトである（**図 1** 左）．システムが高負荷になった場合はスケールアウトを行い，アプリケーションを動かすインスタンスの台数を増やして負荷を分散させる．逆に，システムが低負荷になった場合はスケールインを行い，インスタンスの台数を減らしてインスタンスの利用にかかるコストを削減する．しかし，インスタンスが 1 台の状態でシステムが低負荷になっても，それ以上インスタンスの数を減らすことはできない．例えば，Web サーバへのリクエストがほとんどない状態ではシステムの負荷は限りなくゼロに近いが，Web サーバを停止することができない場合は 1 台のインスタンスが必要になる．そのため，アプリケーションによる負荷がほとんど発生していない時のコストの削減には限界がある．

1 台のインスタンスに対する最適化手法として，インスタンスの性能を変更するスケールアップ・スケールダウンがある（**図 1** 中央）．システムが高負荷になるとスケールアップを行い，インスタンスに CPU やメモリを追加することで性能を向上させる．逆に，システムが低負荷になるとスケールダウンを行い，インスタンスの性能を下げてリソース使用にかかるコストを削減する．しかし，既存の多くのクラウドではインスタンスを実行したまま動的に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インスタンスを切り替えることでスケールアップ・スケールダウンを実現する．そのため，提供されている最低性能のインスタンスよりも性能を下げてコストを削減することはできない．また，性能を切り替える際にはアプリケーションを一度停止させてから，データなどを切り替え先のインスタンスに移動して，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．

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 台のインスタンスに統合する方法が考えられる（**図 1** 右）．例えば，2 台のインスタンス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 台のインスタンスに統合することによってインスタンスの数を減らし，コストを削減することができる．その後，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インスタンス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インスタンス間で移動させる際にダウンタイムが発生する．また，同一インスタンス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容易ではない．Zap [3] 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ようにしている．しかし，プロセス間の隔離は十分ではないため，アプリケーション統合時のセキュリティが低下する．また，マイグレーションの単位であるプロセス群にグローバル IP アドレスを割り当てる必要がある．多くの IaaS 型クラウドではグローバル IP アドレスの

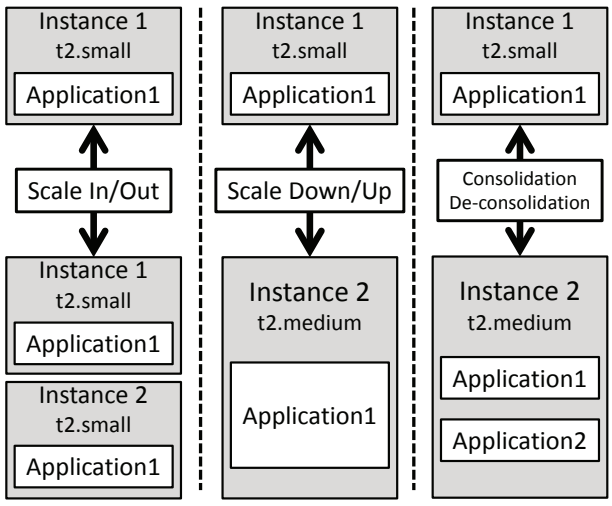

図 1 インスタンス構成の最適化

追加は有料オプションになっているため，プロセス群ごとにグローバル IP アドレスを割り当てることはコストの問題により避けるべきである。

3. FlexCapsule

本稿では，クラウドにおいてアプリケーション単位でのインスタンス構成の動的最適化を実現するシステム FlexCapsule を提案する．FlexCapsule ではインスタンスの中の個々のアプリケーションを VM 内で動作させ，ライブラリ OS を用いて軽量化を実現する．FlexCapsule では VM のマイグレーション技術を利用して，インスタンス間でアプリケーションを移動する．これにより，ダウンタイムの発生を抑えて，OS の状態も含めてアプリケーションを移動することが可能となる．さらに，それぞれのアプリケーションの間には VM 間の強い隔離が働いているため，アプリケーション統合時のセキュリティの低下も防ぐことができる．

3.1 FlexCapsule のシステム構成

FlexCapsule のシステム構成を**図 2** に示す．FlexCapsule では，ネストした仮想化を用いてインスタンス (VM) の中でハイパーバイザを動作させる．そのハイパーバイザの上でアプリケーション VM と呼ばれる VM を動作させる．アプリケーション VM の中ではアプリケーションのプロセスを一つだけ動作させ，ライブラリ OS を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から OS の機能を利用可能とする．アプリケーション VM の管理を行ったり，アプリケーション VM の中では実現できない機能を提供するために，各インスタンス内で OS サーバを動作させる．

FlexCapsule ではグローバル IP アドレスはインスタンスに対して割り当てる．各アプリケーション VM にはそれぞれプライベート IP アドレスが割り当てられるため，グローバル IP アドレスの割り当てによるコスト増加を抑えることができる．Network Address Port Translation (NAPT) を用いることにより，インスタンスに割り当てられたグローバル IP アドレスを用いてアプリケーション VM のサービスを提供する．さらに，拠点間 VPN を用いることによって，複数のインスタンス間で同一セグメントのネットワークを構築する．それぞれのアプリケーション VM を同一セグメント内で動作させることによって，マイグレーションにより別のインスタンス内に移動した場合も移動前のプライベート IP アドレスを使い続けることが可能となる．外部からのパケットは一旦，指定されたグローバル IP アドレスを持つインスタンスに送られるが，VPN によってアプリケーション VM が動作している適切なインスタンスに転送される．

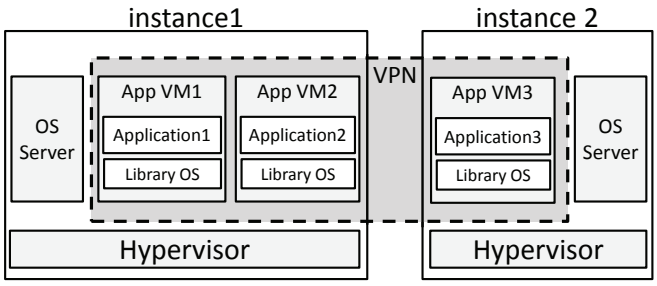

図 2 FlexCapsule のシステム構成

3.2 FlexCapsule を用いたインスタンス構成の最適化

スケールアップ・スケールダウンによるインスタンス構成の最適化では，目的の性能をもったインスタンスを新たに用意し，そのインスタンスにアプリケーション VM をマイグレーションする．マイグレーション後はインスタンスが使用していたグローバル IP アドレスを移動先のインスタンスに割り当て直す．この処理は Amazon Web Services の Elastic IP アドレスのような IP アドレスの動的な付け替えにより行う．このように，アプリケーションを透過的に移動させることができるため，移動先のインスタンスですぐにサービスを再開させることが可能である．

アプリケーション統合による最適化では，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．統合元のインスタンスを停止する場合には，そのグローバル IP アドレスを統合先のインスタンスに追加することでマイグレーション前の IP アドレスを用いたネットワークアクセスを可能にする．個々のアプリケーション VM の間の隔離により，同一インスタンス内の他の VM に干渉することはできない．そのため，同一インスタンス内で複数のアプリケーション VM を動かした場合でもセキュリティ上のリスクは小さい．

3.3 FlexCapsule OS

FlexCapsule OS は，アプリケーション VM 内で動作する軽量なライブラリ OS 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コンパイル時にアプリケーションの実行に必要な OS の機能のみをリンクするため，従来の汎用 OS を用いた場合よりもメモリの使用量が少なくすることができる．これにより，汎用 OS を用いた場合と違い，一つのアプリケーションに対して一つの VM を動作させてもリソース消費量を抑えることができる．さらに，割り当てるメモリを少なくすることができるため，マイグレーション時のメモリの転送にかかる時間が短くなり，マイグレーションを高速化することができる．

FlexCapsule OS は準仮想化を用いて仮想化のオーバーヘッドの削減を行う．準仮想化では OS がハイパーバイザ

と密接に連携することにより，VM の性能を向上させることができる．その一方で，マイグレーション時には OS とハイパーバイザの連携を一時的に解除する必要があるため，FlexCapsule OS はマイグレーションをサポートする機能を提供する．

3.4 OS サーバ

OS サーバ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管理インタフェースを提供する．このインタフェースにより，ユーザはアプリケーションが VM 内で動いていることを意識せずに管理できるようになる．例えば，実行中のアプリケーションの情報を表示するために，従来の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使用する必要がある．OS サーバでは，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．

また，OS サーバは FlexCapsule OS 単体で実現できない，複数のプロセスが関連する機能を実現する．FlexCapsule ではアプリケーションの一つのプロセスだけが VM として動作するため，複数のプロセスが関係する処理を行うことができないためである．例えば，プロセスの複製を作成する際には，OS サーバがアプリケーション VM の複製を行う．また，プロセス間通信を行うには，アプリケーション VM が送信したメッセージを適切なアプリケーション VM が受信できるように OS サーバが中継する．

さらに，OS サーバはアプリケーション VM にパケットを転送するための NAPT ルールの管理も行う．アプリケーション VM は NAPT を用いて外部との通信を行うため，アプリケーションの待ち受けポートに応じた NAPT ルールを登録する必要がある．例えば，Web サーバが動いている場合には，インスタンスのグローバル IP アドレスと 80 番ポートの組をアプリケーション VM のプライベート IP アドレスと 80 番ポートの組に変換するルールを登録する．

4. 実装

FlexCapsule を Xen 4.2 を用いて実装した．DomU をインスタンスとし，DomU の中でも Xen 4.2 を用いて仮想化システムを構築した．その中の管理 VM（Dom0）で OS サーバを動作させ，DomU をアプリケーション VM とした。

4.1 FlexCapsule OS

FlexCapsule OS は Xen でサポートされているライブラリ OS である Mini-OS をベースとしたものと仮想環境向けに設計されたライブラリ OS である OSv [2] をベースとしたものの 2 種類を実装した．

Mini-OS は Xen の管理 VM 内で動いている個々のコンポーネントを独立した VM として動作させ，セキュリティや拡張性を高めるためによく用いられている．アプリケーションは Mini-OS とリンクされ，カーネル空間で動作する．Mini-OS では C 言語のランタイムの他に Ocaml ランタイムや Haskell プログラムのための GHC ランタイムなどが提供されている．Mini-OS は準仮想化 OS であるため，仮想化のオーバーヘッドは小さい．しかし，Mini-OS の問題として対応するアプリケーションの少なさが挙げられる．

一方，OSv は仮想化システムに対して最適化された OS である．OSv では Java 仮想マシンを動作させることも可能であり，いくつかの既存のアプリケーションの稼働実績がある．OSv は完全仮想化 OS であるが，I/O のオーバヘッドを最小限に抑えるために，準仮想化デバイスドライバを用いている．しかし，完全仮想化 OS であるため Mini-OS と比べてマイグレーションの性能が低い．

4.2 マイグレーションのサポート

FlexCapsule OS のベースとなった OS に対してそれぞれマイグレーションのサポートを追加した．

4.2.1 マイグレーションの流れ

準仮想化におけるマイグレーション処理の流れを**図 3** に示す．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 をサスペンド状態にする．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．

マイグレーション時にサスペンド要求を受けとった OS はシャットダウンハンドラを呼び出して，サスペンド処理を実行する．サスペンド処理が完了すると OS はサスペンドが完了したことをハイパーバイザに通知するハイパーコールを発行する．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，ハイパーコールが完了した状態から OS の処理が再開され，直ちにレジューム処理を行う．

4.2.2 サスペンド要求の受信機構

FlexCapsule OS に管理 VM からのサスペンド要求を受けとるための機構の実装を行った．Xen では VM 間で情報を共有するためのストレージとして XenStore が提供されている．管理 VM は電源管理の要求を XenStore の control/shutdown ノードに書き込むため，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．ノードに対して書き込みが発生した場合，内部で割り込みが発生して電源処理を行うハンドラであるシャットダウンハンド

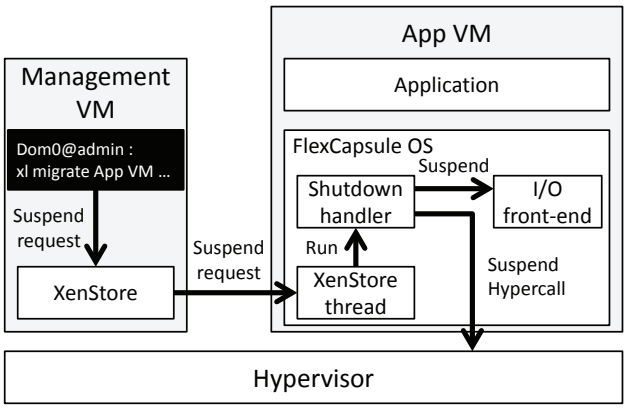

図 3 アプリケーション VM のサスペンド処理

ラが呼び出される．

4.2.3 サスペンド・レジューム処理

シャットダウンハンドラでは XenStore のノードに書き込まれた情報を基に，システムのサスペンドを行う．Xen では，アプリケーション VM 上のフロントエンドドライバが管理 VM 上のバックエンドドライバと通信を行うため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 の間では各種ドライバで利用するイベントチャネルのコネクションが確立されている．マイグレーション後はアプリケーション VM が他のインスタンス上の管理 VM と新たにイベントチャネルを確立するため，サスペンド時にはイベントチャネルの切断を行う．この時，コンソールリングなど内部状態が変わらないものに関してはイベントが発生した時に呼び出されるスレッドは終了させず，イベントチャネルのマスクだけを行う．これによって，レジューム時にはマスクを解除するだけで再利用可能となる．ネットワークの IP アドレスなど，レジューム時に変化する可能性がある情報を持つドライバに関してはサスペンド時にフロントエンドドライバを終了させ，レジューム時に新たな情報を基にフロントエンドドライバを立ち上げ直す．

準仮想化 OS である Mini-OS ベースの FlexCapsule OS は，デバイスドライバのみ準仮想化されたものを用いる OSv ベースの FlexCapsule OS と比べてより密接にハイパーバイザと連携する．そのため Mini-OS ベースの FlexCapsule OS では，サスペンド時とレジューム時にこれらの連携を維持するための処理も行う必要がある．例えば，準仮想化 OS ではメモリ共有を管理するグラントテーブルやアプリケーション 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 テーブルがインスタンスの情報を用いて作成されている．マイグレーション後は別のインスタンス内で動作するようになるため，Mini-OS ベースの FlexCapsule OS ではこれらについてもサスペンド処理とレジューム処理を行う．これらはマイグレーション後にそのまま使うことはできないため，サスペンド時に破棄してレジューム時に新たに作り直す．

サスペンド処理が完了したら，VM のサスペンドを行うハイパーコールを発行する．この時に，VM の開始情報が格納されている start_info 構造体をハイパーコールの引数としてハイパーバイザに渡す．start_info 構造体には時刻や仮想 CPU 状態の情報を持つ shared_info 構造体などの VM の内部情報が含まれているため，マイグレーション先のインスタンスで VM を起動させる際に必要となる．しかし，完全仮想化 OS では，これらの情報は VM 内部で閉じた状態で管理されているため，OSv ベースの FlexCapsule OS では start_info 構造体の受け渡しは行わない．

4.3 OS サーバ

OS サーバは各インスタンス内の管理 VM 上で動作し，アプリケーション VM やユーザからの要求の処理を行う．OS サーバの構成を**図 4** に示す．

4.3.1 アプリケーション VM の管理

管理者は提供されるシェルを介して OS サーバの管理インタフェースにアクセスし，アプリケーション VM の管理を行う．このシェルではユーザにアプリケーションの起動，終了 (kill)，情報取得 (ps) コマンドなどを提供する．ユーザがシェル上でアプリケーション名を入力すると，OS サーバはアプリケーション名に対応する VM イメージの起動を行う．アプリケーションを終了させるには，従来のアプリケーションと同様に FlexCapsule OS の提供するインタフェースで終了処理を行うか，シェル上でプロセス ID またはアプリケーション名を指定して kill コマンドを実行する．ユーザが ps コマンドを実行した場合は，OS サーバは Xen からアプリケーション VM の情報を取得してユーザへと返す．これらの処理は Xen が提供するツールキットである xenlight (libxl) を用いて実装した．アプリケーション VM の情報の取得は libxl により取得できる情報の他に XenStore からも取得する．

4.3.2 アプリケーション VM の fork

FlexCapsule では VM の複製によってプロセスの複製を行う．プロセスを複製する fork 関数がアプリケーションによって実行された時に，FlexCapsule OS は XenStore を介して OS サーバに fork 要求を送る．要求を受けとった OS サーバはアプリケーション VM のメモリのスナップショッ

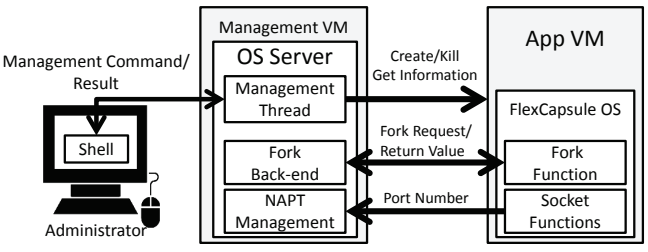

図 4 OS サーバの構成

トを保存し，それを用いて子プロセスに相当する子アプリケーション VM のレジュームを行う．その際に，子アプリケーション VM の IP アドレスを付け替え，子アプリケーション VM に対する NAPT ルールの追加を行う．子アプリケーション VM を起動した後，XenStore 経由で親アプリケーション VM には子アプリケーション VM の ID を返し，子アプリケーション VM には 0 を返す．

4.3.3 NAPT ルールの管理

アプリケーションがネットワークの接続要求を待つために listen 関数を実行した時に，FlexCapsule OS は通常の listen 処理と同時に指定された待ち受けポート番号を XenStore を介して OS サーバに送信する．OS サーバは受け取ったポート番号を用いて，インスタンスの IP アドレスの指定されたポート番号に届いたパケットをアプリケーション VM へと転送する NAPT ルールを iptables に追加する．ルールの追加には netfilter を操作するライブラリである libiptc [4] を使用した．アプリケーションが listen 状態のソケットに対して close 関数を実行した場合にも，XenStore を介して待ち受けポート番号を OS サーバに送信する．転送終了要求を受け取った OS サーバは使用していた NAPT ルールの削除を行う．

5. 実験

アプリケーション VM のマイグレーション時間とダウンタイムの計測を行い，Linux をインストールした VM をマイグレーションした場合との比較を行った．また，インスタンスの性能を変化させることによるアプリケーションの性能の変化を調べた．さらに，アプリケーションで fork 関数および listen 関数を実行し，従来環境との性能比較を行った．

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．ハイパーバイザには Xen 4.2.2 を用い，ハイパーバイザ上の管理 VM では Linux 3.13.0 を動かした．インスタンスには 2 個の仮想 CPU と 1 GB のメモリを割り当てた．インスタンスの中で動作させるハイパーバイザおよび管理 VM の OS にも同じものを用いた．アプリケーション VM には 1 個の仮想 CPU と様々なサイズのメモリを割り当てた．

5.1 マイグレーション時間とダウンタイム

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その際に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 VM が停止している時間をダウンタイムとした．比較として，準仮想化（PV）と完全仮想化（HVM）の Linux 3.13.0 が動作する VM のマイグレーションについても計測を行った．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 5**，**図 6** に示す．

この結果より，Mini-OS をベースとした FlexCapsule OS を用いたアプリケーション VM では，ダウンタイムに関しては準仮想化 Linux を用いた VM とほぼ同じか，最大で 0.01 秒短いことがわかる．また，Mini-OS ベースのアプリケーション VM は最小で 4MB のメモリで動作させることができており，Linux を用いた VM よりマイグレーション時間を短くできていることがわかる．しかし，同じメモリサイズで比較すると，Linux を用いた VM のほうが約 1 秒速かった．ダウンタイムに大きな差はないため，FlexCapsule OS 内でのサスペンド・レジューム処理ではなく，ハイパーバイザや管理 VM 側での処理に遅延が生じていると考えられる．

OSv をベースとした FlexCapsule OS を用いたアプリケーション VM では，完全仮想化 Linux を用いた VM と比べて，ダウンタイムは 0.15 秒短くなっていることがわかる．また，OSv ベースのアプリケーション VM は Linux を用いた VM の半分のメモリで動作可能であるため，マイグレーション時間を短くすることができている．同じメモ

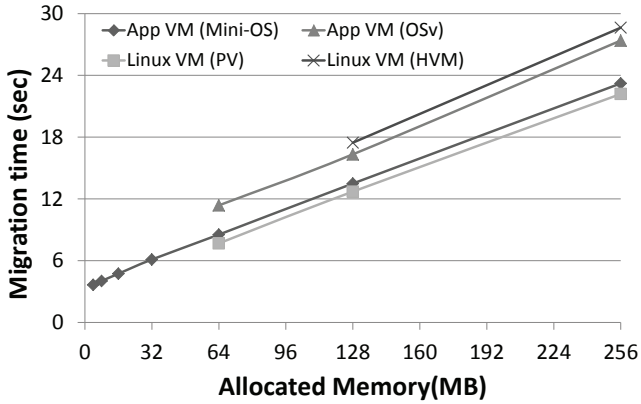

図 5 マイグレーション時間の計測結果

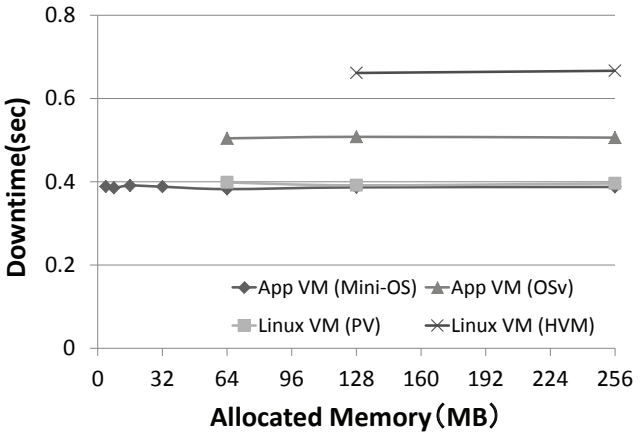

図 6 ダウンタイムの計測結果

リサイズで比較した場合でも，マイグレーション時間を 1 秒短縮できている．ただし，準仮想化 OS を用いた場合と比べると，マイグレーション時間，ダウンタイムともに長くなった．これは，準仮想化のほうがマイグレーション時のデバイスの処理にかかる時間が短いためと考えられる．

5.2 アプリケーションの性能

アプリケーション VM が動作するインスタンスの性能を変化させて，アプリケーションの性能にどのような影響が生じるかを調べた．インスタンスの性能を変化させるために，Xen の credit スケジューラを用いて，VM の CPU 使用率の上限を設定した．今回の実験では 60%から制限のない 100%まで 10%刻みで CPU 使用率の制限を変更した．アプリケーション VM 内で Dhrystone ベンチマークを実行し，メインループを 5 千万回実行するのにかかる時間を測定した．実験対象は Mini-OS および OSv をベースとした FlexCapsule OS を用いたアプリケーション VM と準仮想化および完全仮想化の Linux 3.13.0 を用いた VM とした．

計測結果を**図 7** に示す．この結果より，アプリケーションの性能はインスタンスの性能によって変化することが分かる．つまり，アプリケーション VM を用いてインスタンスのスケールアップ・スケールダウンが可能であることが示されている．さらにそれぞれの結果を比較すると，Mini-OS ベースの FlexCapsule OS を用いた場合が最も高い性能を示していることが分かる．この原因を調べるため，それぞれの OS で整数四則演算を 1 万回ずつ実行し，その処理時間を測定した．その結果を**図 8** に示す．この結果から Mini-OS は除算の処理時間が他の OS と比べて著しく速いことが分かった．その原因については現在，調査中である．

5.3 fork 関数の性能

FlexCapsule OS の fork 関数の性能を調べるために，アプリケーションが fork 関数を実行してアプリケーション VM の複製を行い，子アプリケーション VM で fork 関数が完了するまでにかかる時間を測定した．結果を**図 9** に示す．現在の実装では，アプリケーション VM の複製を作成するために VM のメモリ全体をコピーする必要があるため，fork 関数の実行時間は VM のメモリ割当量に比例する結果となった．完全仮想化 Linux 上で fork 関数を実行するのにかかる時間は 9 ミリ秒であるため，現在の fork 関数の実装ではアプリケーションの著しい性能低下を引き起こすことが分かった．VM の複製にコピーオンライトを用いる VM fork [5] のような技術を用いることで，fork 関数の高速化を行うことが可能である．

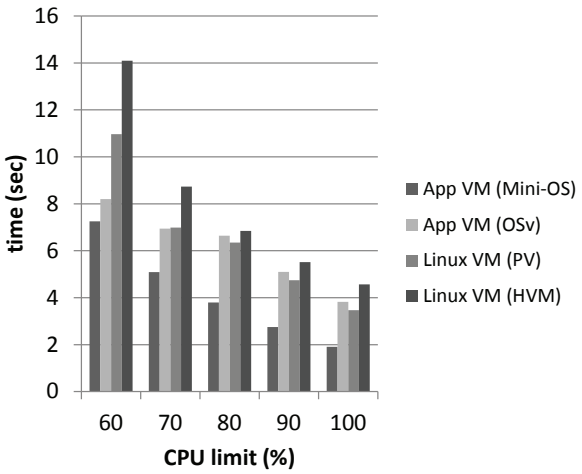

図 7 インスタンス性能とアプリケーション性能の関係

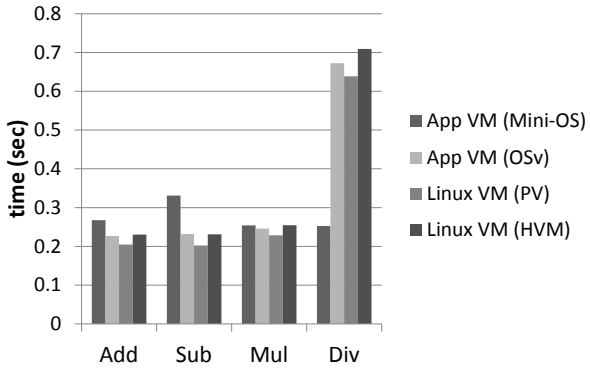

図 8 整数四則演算の処理時間

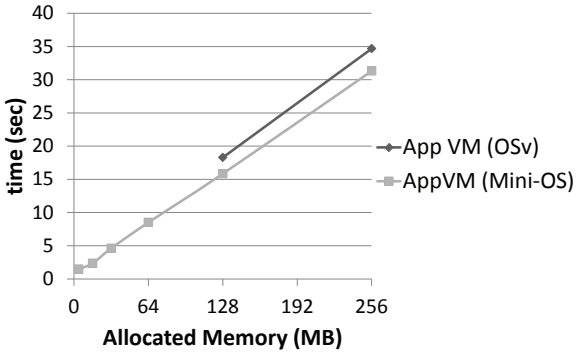

図 9 fork 実行時間

5.4 listen 関数の性能

FlexCapsule OS の listen 関数の性能を調べるために，アプリケーションが listen 関数を実行して OS サーバと通信し，listen 関数を完了するのにかかる時間を測定した．比較のために，OSv のデフォルトの listen 関数の実行時間および，完全仮想化 Linux の listen 関数の実行時間も測定した．結果を**表 1** に示す．Linux 上で実行した場合と比べて

表 1 listen 実行時間

	time (msec)
AppVM (OSv)	22.1
AppVM (OSv，デフォルト)	17.9
Linux (HVM)	2.0

OSv ベースの FlexCapsule OS では実行時間が 20 ミリ秒増加した．しかし，OSv の listen 関数のデフォルトと比べると実行時間の増加は 4 ミリ秒であるため，NAPT ルールの追加処理による性能の低下は小さい．

6. 関連研究

ライブラリ OS [6] 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．Xok/ExOS [7] は Exokernel とライブラリ OS を用いて既存の Unix アプリケーションを改変することなく動作させることを可能にしている．ExOS はプロセスの fork やシグナルなどの IPC も提供している．このようなマルチプロセスのサポートを実現するために共有メモリを用いている．しかし，Exokernel ではアプリケーションのマイグレーションを行うことは考えられていなかった．

DrawBridge [8] 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，VM 単位で行う場合と比べて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することができる．しかし，ライブマイグレーションは実現されておらず，マルチプロセスもサポートされていない．

Graphene [9] は Linux 互換のライブラリ OS を用いたアプリケーションにおいてマルチプロセスを実現したシステムである．このシステムではプロセス毎に個別のライブラリ OS が用いられており，それぞれのプロセス間では RPC を用いて通信を行う．例えば，プロセスが別プロセスに対してシグナルを送ると，RPC 経由で対象プロセスの適切なスレッドを呼び出す．また，プロセスのチェックポイントを用いることで fork も実現している．しかし，DrawBridge と同様に，それぞれのプロセスは同一ホスト OS 上で動作するため，ホスト OS 上の脆弱性が各プロセスに大きな影響を与えるという問題がある．FlexCapsule ではアプリケーションはホスト OS よりは脆弱性の少ないハイパーバイザ上で動作する．

Libra [10] や GUK [11] 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．

Mirage [12] はライブラリ OS を OCaml アプリケーションに特化させた Unikernel を生成し，既存の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択することができる．これにより，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．Mirage は Mini-OS をベースとしており，FlexCapsule OS として用いることも可能だと考えられる．

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 は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 間の隔離は弱い．

7. まとめ

本稿では，ライブラリ OS を用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，クラウドにおけるアプリケーション単位でのインスタンス構成の最適化を実現するシステム FlexCapsule を提案した．FlexCapsule はアプリケーション VM のマイグレーション，アプリケーションの fork，クラウド環境でのネットワーク利用，アプリケーション VM の管理などを提供する．FlexCapsule を用いた実験により，アプリケーション VM の様々な性能を調べた．

今後の課題は，FlexCapsule OS と OS サーバへの機能追加によって FlexCapsule で幅広いアプリケーションをサポートすることである．例えば、ネットワークサービスを提供するアプリケーションを fork することによるプロセスプールの実現が挙げられる．そのためには，複数のアプリケーション VM が同じポートで待ち受けることを可能にし，接続リクエストをいずれかのアプリケーション VM が処理できるようにする必要がある．

参考文献

[1] Ben-Yehuda, M., Day, M. D., Dubitzky, Z., Factor, M., Har'El, N., Gordon, A., Liguori, A., Wasserman, O. and Yassour, B.-A.: The Turtles Project: Design and Implementation of Nested Virtualization, *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, pp. 423–436 (2010).

[2] Kivity, A., Laor, D., Costa, G., Enberg, P., Har'El, N., Marti, D. and Zolotarov, V.: OSv –Optimizing the Operating System for Virtual Machines, *2014 USENIX Annual Technical Conference* (2014).

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

[4] Balliache, L.: Querying libiptc HOWTO, http://www.tldp.org/HOWTO/Querying-libiptc-HOWTO/ (2002).

[5] Lagar-Cavilla, H. A., Whitney, J. A., Scannell, A. M., Patchin, P., Rumble, S. M., de Lara, E., Brudno, M. and Satyanarayanan, M.: SnowFlock: Rapid Virtual Machine Cloning for Cloud Computing, *Proceedings of the 4th ACM European Conference on Computer Systems* (2009).

[6] Engler, D. R., Kaashoek, M. F. and O'Toole, Jr., J.: Exokernel: An Operating System Architecture for Application-level Resource Management, *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pp. 251–266.

[7] Kaashoek, M. F., Engler, D. R., Ganger, G. R., Briceño, H. M., Hunt, R., Mazières, D., Pinckney, T., Grimm, R., Jannotti, J. and Mackenzie, K.: Application Performance and Flexibility on Exokernel Systems, *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles* (1997).

[8] Porter, D. E., Boyd-Wickizer, S., Howell, J., Olinsky, R. and Hunt, G. C.: Rethinking the Library OS from the Top Down, *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 291–304.

[9] Tsai, C.-C., Arora, K. S., Bandi, N., Jain, B., Jannen, W., John, J., Kalodner, H. A., Kulkarni, V., Oliveira, D. and Porter, D. E.: Cooperation and Security Isolation of Library OSes for Multi-process Applications, *Proceedings of the Ninth European Conference on Computer Systems* (2014).

[10] Ammons, G., Silva, D. D., Krieger, O., Grove, D., Rosenburg, B., Wisniewski, R. W., Butrico, M., Kawachiya, K. and Hensbergen, E. V.: Libra: A Library Operating System for a JVM in a Virtualized Execution Environment, *Proceedings of International Conference on Virtual Execution Environments*, pp. 44–54 (2007).

[11] Jordan, M. and Roeck, H.: Guest VM Microkernel, GUK, https://kenai.com/projects/guestvm (2009).

[12] 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).