マルチプロセスをサポートする Unikernel-based VM

金津 穂[1,a)]　山田 浩史[1,b)]

概要：IaaS 型のクラウドプラットフォームでは仮想化技術によって計算資源を細かに制御・分配することにより，ユーザがオンデマンドに必要最小限の計算資源を借りることができる反面，仮想マシンを借りることになるためユーザが OS の管理を行なわなければならない．PaaS 型のクラウドプラットフォームではユーザが直接アプリケーションをデプロイできるため容易にエンドユーザへサービスを提供できる反面，IaaS 型ほど細かにユーザが計算資源を制御することは不可能である．また，IaaS 型と PaaS 型のどちらにおいても，既存ソフトウェア資産の活用や普及度の問題から既存の汎用 OS が使用されるものの，特定のサービスを稼動させる目的だけを考えると汎用 OS は巨大であり，不要な機能が多い．そこで，カーネル機能をライブラリとしアプリケーションとリンクし単一のプロセス空間へ配置することで，機構を簡易化しオーバヘッドを大幅に削減することによって軽量化とパフォーマンスの向上を実現した Unikernel が存在する．Unikernel はパフォーマンスや管理の面だけではなく仮想化環境上で直接動作するためカーネルをアプリケーションへ特化させることも容易であるというメリットがある．しかし，単一のプロセス空間で動作するためマルチプロセス機能が存在しないため，POSIX API を実装したものであっても既存のプログラムへの互換性が不十分である．そこで，マルチノードへスケールすることを前提としたマルチプロセス機構を有している Unikernel，*Unikernel fork* を提案する．Unikernel fork は（1）アプリケーションに応じて容易にカーネルを特化させることができる，（2）マルチプロセスに対応する，（3）既存のプログラムを改変せず利用可能である，（4）他ノードへアプリケーション透過的に子プロセスが配置可能である，という四点を特徴とする．本稿では Unikernel fork の実現のため，まずシングルノードで複数 VM を用いてマルチプロセスに対応する Unikernel の機構の設計・実装を示し，また，今後の課題について検討を行なった．

キーワード：クラウド, 仮想化, ライブラリ OS, Unikernel

1. はじめに

クラウドコンピューティングという語が 2006 年に登場してから様々なパブリッククラウドが登場した．Amazon Web Service [1]，Google Cloud Platform [2]，Microsoft Azure [3] など様々な計算基盤を総合的に提供するプラットフォーム事業者も登場し，Web サービスや Web サイトを支えるインフラストラクチャのひとつとなっている．このようなクラウド型のホスティングサービスではエンドユーザへコンテンツを配信する事業者が自身で管理するハードウェアを減らすことが可能である．さらに，従来の専有型レンタルサーバーと異なり，仮想化技術やコンテナ技術を用いて計算資源を細かに制御し分配することによって必要最低限の計算資源を借りられるというメリットが存在する．

クラウドプラットフォームのうち，エンドユーザではなく事業者が利用するものとして主に仮想化技術によりハードウェア資源を提供する Infrastructure as a Service（IaaS）やアプリケーションのデプロイメント環境を提供する Platform as a Service（PaaS）が存在する．PaaS では Windows や Linux といった既存 OS が基盤環境として用いられ，IaaS の場合はハードウェア資源をそのまま活用できるものの既存のソフトウェア資産を活用するために既存の OS をインストールし利用される．例えば Amazon Web Service の提供する IaaS プラットフォームの Amazon EC2 では，Amazon Linux という EC2 に最適化された Linux distribution が用意されている．しかし既存の OS の多くは一台の機器で複数のアプリケーションを動作させることを念頭に開発されており，また，汎用品であるため個々のアプリケーションに特化された OS ではないため，IaaS で特定のサービスを運用するために本来サービスが必要とする以上の資源が必要となる．さらに，サービスの管理とは別に OS のセキュリティアップデートやメンテナンスといった部分でのクラウドプラットフォーム利用者の管理も必要となる．PaaS につ

[1] 東京農工大学
Tokyo University of Agriculture and Technology
[a)] mkanatsu@asg.cs.tuat.ac.jp
[b)] hiroshiy@cc.tuat.ac.jp

いても，ユーザ間の分離のためにクラウドプラットフォーム事業者が仮想化技術やコンテナ技術を用いて複数の既存OS 環境を用意する必要がある．

以上の状況に対して，よりアプリケーションに特化し OS のレイヤを小さくすることで，パフォーマンスの向上と管理コストの削減を目的としたものとして Unikernel [4, 5] が挙げられる．library OS（LibOS） [6] の考えを元にしてアプリケーションと最小限の OS レイヤを一個のバイナリやアプリケーションとして纏めてしまい，Virtual Machine（VM）型のコンテナとして IaaS へデプロイできるようにするものである．コンテナ間とコンテナ・ホスト間の保護は仮想化環境に任せることができるため，OS 機能とアプリケーションの間のメモリ保護が不要となり単一のプロセス空間で動作可能である．このことによって，アプリケーションのシステムコール呼び出しのコストが大幅に削減できるためパフォーマンスの向上が達成でき，また，コンテナの形をとっているため LibOS 部分のアップデートなどがあった際にもアプリケーションごと新たに作り直しデプロイすれば良いため，常に動作する OS をパッケージシステムや設定ファイルによって管理する必要がなくなる．

Unikernel にもデメリットが存在し，POSIX API を移植したもの [5,7] であってもカーネルと同一の単一メモリ空間でアプリケーションが稼動してしまうため，`fork(2)` や `exec(2)` といったマルチプロセスに関する機能が存在しないかあるいは制限されている．このため，既存のソフトウェア資産との互換性に問題があり，また，マルチプロセスを前提とした並列プログラムが動作しないためパフォーマンスが発揮できない場合もある．具体的には PostgreSQL でコネクションごとに isolation し実行する，NginX でイベントループを回し複数のコネクションを受けるといったプログラムがマルチプロセスを利用するが，Unikernel ではそのような動作は不可能となる．加えて Unikernel はシングルノードを前提とする設計であるため，データセンタのようなマルチノードの環境でスケールするプログラムを作成する場合ソフトウェアの書き換えが必要となる．

そこで，Unikernel のデメリットを解消し，既存のソフトウェア資産への互換性の向上およびパフォーマンスの向上を達成するために，Unikernel の新たな機構である *Unikernel fork* を提案する．Unikernel fork は次の特徴を持つ

(1) アプリケーションに応じて容易にカーネルを特化させることができる
(2) マルチプロセスに対応する
(3) 既存のプログラムを改変せず利用可能である
(4) 他ノードへアプリケーション透過的に子プロセスが配置可能である

本稿ではシングルノードで複数 VM を用いてマルチプロセスに対応する Unikernel を設計・実装を行ない，複数 VM をアプリケーション透過的に扱えることを示した．

本論文の構成は次のとおりである．2 章では，研究背景と関連研究について述べる．3 章では，提案システムとその設計・実装を述べる．4 章では，提案システムの今後の予定について述べる．5 章では，本論文のまとめを述べる．

2. 背景

2.1 library OS

library OS（LibOS）とは，OS の機能をライブラリの形で実装し，アプリケーションとリンクして利用する OS アーキテクチャのことである．Engler らの ExoKernel [6] ではハードウェアの抽象化と調停のみをカーネルが提供し，ファイルシステムやネットワークスタックといった機能は LibOS の形で提供する．アプリケーションレベルで計算資源を管理可能なためアプリケーションにとって必要な資源が最大限に活用できかつ LibOS が抽象化レイヤとなるためベアメタルプログラミングの煩雑さを避けることができる．

2.2 Unikernel

ExoKernel と LibOS の関係を Virtual Machine Monitor（VMM）と LibOS に置き換えたクラウド向け軽量 OS として *Unikernel* が存在する [4, 5, 7]．Unikernel は名前空間で分離を行なうコンテナ [8, 9] と異なりコンテナ間，コンテナ・ホスト間を仮想化環境によって分離する，VM を用いたコンテナの一種であるとも言える．

Unikernel には次のようなメリットが存在する．(1) アプリケーションごとに特化させた OS を実装しやすい，(2) 既存の IaaS 環境にそのままデプロイ可能かつ高速に起動する軽量な OS である，そして (3) 汎用 OS に比べコードベースが格段に小さく Trusted Computing Base（TCB）を小さくできるという三点である．

Unikernel はカーネル・アプリケーション間のメモリ保護が不要なため非常に簡潔な設計が可能となり，コードベースが小さくなる．この構造によってカーネル・アプリケーション間のオーバーヘッドを削減を可能とし，よりカーネルがアプケーションに密接となった特化型の OS の実装が容易となる．また，特化型の OS のためカーネルをアプリケーションごとにフルスクラッチする必要も無くなる．Unikernel をアプリケーション特化型の OS として用いる研究では，Network on Chips に特化した DlibOS [10]，Software Defined Network へ特化させた ClickOS [11] が挙げられる．

Unikernel は既存の汎用 OS に比べ起動にかかる時間が非常に短いため [12]，IaaS のオンデマンド性を活用し，必要最小限の資源でサービスが運用することでインフラを利用するコストが削減しつつサービスの負荷が向上した際には資源をすぐに借りサービスを展開・起動する運用が可能となる．また，データセンタでのトラブルが発生した場合，いち速くサービスを復旧させる際にもサービスの起動する

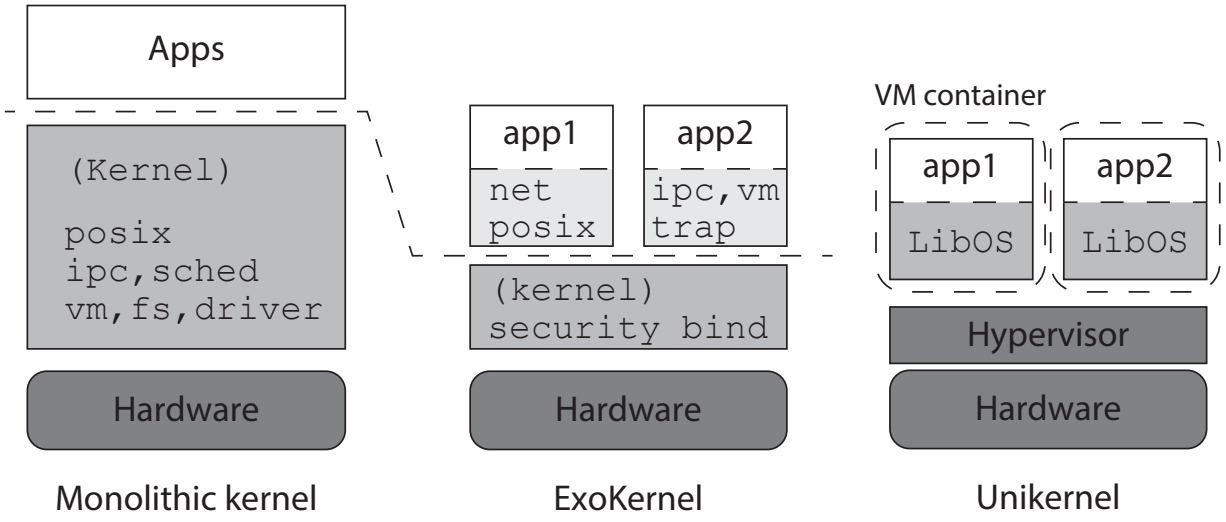

図 1 カーネルアーキテクチャの概念図

速度が重要となる．Manco らの研究では Xen [13] 上での VM 起動にかかるボトルネックを洗い出し高速化することによって，既存の OS のみならずコンテナやプロセスと比較しても遜色のない起動速度を Unikernel で達成した [14].

2.3 Unikernel におけるマルチプロセス

Unikernel は単一プロセス空間でアプリケーションとカーネルが動作する設計にすることで機構を簡潔にしパフォーマンスを向上している反面，マルチプロセスをサポートしない．たとえば，OSv [5] や RumpRun [7,15] は POSIX API を実装しているが，`fork(2)` や `pipe(2)`，`exec(2)` といった API はサポートしない[*1]．このため，既存ソフトウェアに対して互換性に難がありマルチプロセスを用いた並列プログラムが実行できないため，パフォーマンスに難がある場合もある．

Tsai らの研究では LibOS の一種である picoprocess にマルチプロセスをサポートする機構を備えることにより，マルチプロセスアプリケーションのサンドボックス化と監視を可能にした [16]．しかし，picoprocess は Unikernel 型の LibOS ではなく，既存のソフトウェアとの互換性は高いものの IaaS 環境へ直接デプロイし実行することは不可能である．また，縣らの研究では Linux カーネルに単一プロセス空間で擬似的にマルチプロセス動作をする機構を提案・実装した [17]．この手法はハイパーバイザ上へのデプロイ・既存のマルチプロセスアプリケーションの実行を可能にしたが，カーネルのコードベースが巨大であるためアプリケーションへ特化させる実装を行なうのは容易ではない．この両者の研究ではそれぞれ，Tsai らの研究ではホスト OS の存在を前提とし縣らの研究では単一プロセス空間での疑似マルチプロセスであるため，複数の VM や複数のノード上でアプリケーションを展開・実行することは不可能である．だが，IaaS は複数の VM を使用するアーキテクチャであり，また，コンテナのデプロイサービスの流行によりマイクロサービスアーキテクチャのアプリケーションやフレームワークが増えつつある．このような環境下では，シングルノードでのマルチプロセスのみならずマルチノードでの分散マルチプロセスが互換性・パフォーマンス上で必要である．

VM を他のノードへ複製する既存研究としては Lagar-Cavilla らの SnowFlock [18] と Bryant らの Kaleidoscope [19] がある．これらの研究では特定の VM を親とし，これを Copy-on-Write を活用し高速に複製・動作させる VM fork を実現している．しかし，VM fork では VM そのものは高速に複製されるもののアプリケーションから `fork(2)` として透過的に使用することはできない．ただし，親と子には ID が設定され ID と IP アドレスを紐付けることにより，パッケットフィルタリングを用いたグループ化を行なって VM 間の協調が容易となる仕組みをもつ．

他の VM 複製技術に Wang らの VM substrate [20] が存在する．これはテンプレートとなる VM について vCPU を 1 つにする，キャッシュの解放・ネットワークインターフェース取り外し，メモリ割り当て最小化といったことを行ない，予め複製しやすい状態となった VM のスナップショットをメモリ上に保存しておくことで，高速に VM をマルチノードへデプロイ可能にした仕組みである．同一かつ複数の VM を一度に高速に起動できるが，VM 間での協調機能は一切追加されていない．

[*1] OSv はマルチスレッド機構と名前空間の分離によって疑似的な `exec(2)` である `osv_execve()` を実装しているものの完全に `exec(2)` を代替できない．

3. Unikernel fork

3.1 提案と設計

2 章より，既存研究が対応する機能を表 1 に示す．カーネル機能を容易にアプリケーションごとに特化させやすく軽量であるという Unikernel の利点を保ちつつ，マルチプロセス機能によってより既存のソフトウェアの互換性を向上させ，また，マルチプロセスとして透過的にマルチノードへスケール可能なシステムを実現することによって，既存研究よりさらにクラウド環境に特化したシステムを目指したい．そこで，次を設計の目標とするシステム，Unikernel fork を提案する．

(1) アプリケーションに応じて容易にカーネルを特化させることができる
(2) マルチプロセスに対応する
(3) 既存のプログラムを改変せず利用可能である
(4) 他ノードへアプリケーション透過的に子プロセスが配置可能である

データセンタの計算資源をより効率的に用いるためには，汎用 OS ではなく，アプリケーションやワークロードに応じて特化されたカーネルを用いたほうが無駄なく資源を活用可能である．また，カーネルを特化させるだけではなくマルチノードにスケールすることによって効率良くパフォーマンスが向上する．しかし，ユーザにとっては既存のプログラムとの互換性のないシステムは非常に使い辛いものとなってしまう．そこで，アプリケーションに対しカーネルを特化させやすい Unikernel に対し，マルチプロセス機能を追加し，さらにこの機能によって他ノードへアプリケーション透過にマルチノードへスケールすることによって，互換性とパフォーマンスの向上を目指す．

Unikernel fork の実現のため，第一段階としてシングルホストでの Unikernel の `fork` を実現するシステムを設計する．概要図を図 2 に示す．図は Unikernel fork システム上で VM が起動してから複製を作成するまでの流れを左から順に示したものである．(1) の時点で起動された Unikernel を用いた VM は起動時にまず VM の構成を保存する (2)．この VM が (3) で `fork(2)` を実行した際，Unikernel はハイパーバイザへパススルーをし，(4) でメモリ領域や task 構造体の複製を行ない，最後に (5) で保存された VM の構成を元に再構成をする，という形になっている．

Unikernel で `fork(2)` システムコールをハイパーコール呼び出しとして実装し，このハイパーコールをハイパーバイザでハンドリングする形で Unikernel fork を実現する．また，新たなシステムコールを実装し，ハイパーバイザ起動時に VM の構成をカーネルに保存する．ハイパーバイザでは，VM のメモリを複製しその後 vCPU と仮想メモリといった VMCS への設定を別の VM として作りなおし，後に保存した VM の構成を利用して新たな仮想デバイスを作成する．

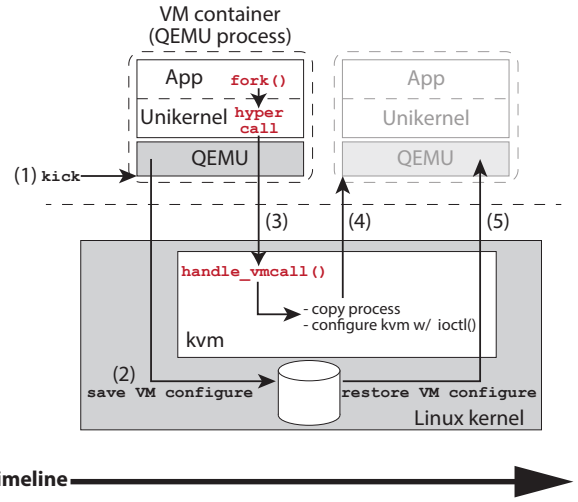

図 2 Unikernel fork の概要図

3.2 実装

プロトタイプとして，OSv と QEMU/kvm [21, 22] に拡張する形で実装を行なった．既に POSIX API を実装する OSv は既存のソフトウェアへの互換性が高く，また，追加のシステムコールの実装が容易である．また，QEMU/kvm は CPU の仮想化支援機構を用いつつもホスト OS である Linux のプロセスとしても振る舞うため，VM の制御にホスト OS の機能を用いやすいという面で `fork` の制御が容易である．

Unikernel でのハイパーコールの kvm でのハンドリングは次のように行なう．

(1) 起動中の QEMU/kvm のプロセスについて task 構造体のコピーを行ない，新たなプロセスとして設定する．
(2) 新しく VMCS を作成する
(3) 起動中の VM の仮想メモリとして設定されている領域を新しい VMCS へ設定する
(4) 保存されている起動中の VM の構成を利用し新たなデバイスモデルを作成する
(5) 新しい VM をスケジューリングする

`fork` によって生まれた新しい VM は既存の VM と同じメモリ領域を使用するため，kvm へ仮想メモリの Copy-on-Write 機能を追加し実現する．

VM の構成のカーネルへの保存は `saveopt(2)` というシステムコールの実装によって実現した．これは，QEMU に与えられた引数をカーネル内にそのまま保存する簡易なシステムコールである．

表 1　LibOS の対応する機能

手法	特定用途へのカーネルの特化	既存プログラムへの互換性	マルチプロセス対応	マルチノード
MirageOS [4]	✓	✗	✗	✗
OSv [5]	✓	partial	N/A	✗
Graphene [16]	✗	✓	✓	✗
縣らの研究 [17]	partial	✓	partial	✗
Unikernel fork	✓	✓	✓	✓

4. 今後の予定

4.1 Unikernel プロセス間通信

本稿では Unikernel の複製機構を示したが，複製元と複製先での VM 同士の通信方法について未検討である．単純な VM 間での通信はオーバーヘッドが大きくなるため，効率的に VM 間で通信する方法としてたとえば親子関係にある VM のみがアクセスできる共有メモリ領域を用意し，名前付き共有パイプとして Unikernel fork へ実装するなどが考えられる．また，シェル上で複数のコマンドを `pipe(2)` によって接続しフィルタとして使用するといったプロセスの利用方法も考えられる．このような方式では，`fork(2)` を実行しているものの，実際には新しいコマンドを実行することが目的であり実質的に `fork(2)` は不要である．以上のようにプロセス間通信の目的によって `fork(2)` の利用目的も異なるため，今後は複数の場合に応じて実際には複製を行なわない Unikernel fork の機構も含めて検討する必要がある．LibOS をシェル上のコマンド・チェインへ応用する既存研究としては，LibOS のひとつである Linux Kernel Library（LKL）[23] へ UNIX pipe を実装したものに Utsumi らの `/dev/stdpkt` [24] が存在する．

4.2 仮想ディスクへの書き込みの調停

現状では仮想ディスクへの書き込みに対し調停を行なっていないため，複製先 VM と複製元の VM が同時に書き込みを行なった場合に問題が発生する可能性がある．そこで，VM fork [18, 19] を参考に，Copy-on-Write を利用することによって複数の VM からの仮想ディスクへの書き込みの調停機構を実装する．

4.3 ネットワークインターフェース

本稿で示した機構ではネットワークインターフェースをそのまま複製してしまうため，MAC アドレスの重複が発生し通信が不可能になってしまう問題が存在する．また，VM が通信中であった場合，ただメモリ空間を複製しただけでは通信中のシーケンスが再現できないため，通信が中断する問題も存在する．そこで，通信を媒介し代理として応答する機構を設計し，ハイパーバイザへ実装する．Unikernel が起動する間に通信の応答を代理し VM が実際に起動し終える時間を隠蔽した研究として，Madhavapeddy らの Jitsu [25] が存在する．

4.4 マルチノードへの拡張

最終的には，Unikernel の複製，Unikernel プロセス同士での通信，仮想ディスクへの書き込みの調停などをマルチノードで透過的に実行できるように拡張する．そのために，現在ホスト OS 内に保持されている VM の構成情報，仮想メモリ，ホスト OS 上に存在する Unikernel のディスクイメージといったものをネットワークを介しマルチノードで分散・共有する仕組みを検討しなければならない．また，マルチノード上でスケールする Unikernel について，実アプリケーションを用いて既存システムとの比較を行なうため，その計測方法を確立する必要がある．

5. まとめ

本稿ではまずクラウド環境を取り巻く現状を指摘し，汎用 OS に代わるシステム，Unikernel を紹介した．また，Unikernel に存在する問題点を洗い出しすことで，クラウド環境へ特化したシステムの設計目標として（1）アプリケーションに応じて容易にカーネルを特化させることができる，（2）マルチプロセスに対応する，（3）既存のプログラムを改変せず利用可能である，（4）他ノードへアプリケーション透過的に子プロセスが配置可能である, という四点定め，これを実現する機構 Unikernel fork の提案を行なった．Unikernel fork の実現の第一段階として本稿ではシングルノードで複数 VM へとスケールする設計と実装を示し，今後の課題と予定明かにした．

参考文献

[1] Amazon: Amazon Elastic Compute Cloud, https://aws.amazon.com/ec2, last accessed on 2016-08-12, (2006).
[2] Google: Google Compute Engine, https://cloud.google.com/compute, last accessed on 2016-08-12, (2012).
[3] Microsoft: Microsoft Azure, https://azure.microsoft.com, last accessed on 2016-08-12, (2010).
[4] 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, *18th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, Houston, TX, USA, ACM, pp. 461–472 (online), DOI: 10.1145/2451116.2451167 (2013).
[5] Kivity, A., Laor, D., Costa, G., Enberg, P., Har'El, N.,

Marti, D. and Zolotarov, V.: OS^{v}—Optimizing the Operating System for Virtual Machines，*2014 USENIX Annual Technical Conference*, USENIX ATC '14, Philadelphia, CA, USA, USENIX Assocation, pp. 61–72（オンライン），入手先 ⟨https://www.usenix.org/system/files/conference/atc14/atc14-paper-kivity.pdf⟩ (2014).

[6] Engler, D. R., Kaashoek, M. F. and O'Toole Jr, J.: Exokernel: An Operating System Architecture for Application-Level Resource Management, *15th ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '95, Vol. 1, No. 212, Copper Mountain, CO, USA, ACM, pp. 251–266 (online), DOI: 10.1145/224056.224076 (1995).

[7] Kantee, A. and Cormack, J.: Rump Kernels: No OS? No Problem!, *;login:*, Vol. 39, No. 5, pp. 11–17 (online), DOI: 10.1007/BF02896304 (2014).

[8] Kamp, P.-H. and Watson, R. N. M.: Jails : Confining the omnipotent root, *2nd Internatinal SANE Conference*, SANE 2000, Maastricht, The Netherlands, System Administration and Network Engineering, (online), available from ⟨http://www.sane.nl/events/sane2000/papers/kamp.pdf⟩ (2000).

[9] Docker: Docker, https://www.docker.com, last accessed on 2018-01-12, (2013).

[10] Mallon, S., Gramoli, V. and Jourjon, G.: DLibOS: Performance and Protection with a Network-on-Chip, *23rd ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '18, Williamsburg, VA, USA, ACM, pp. 737–750 (online), DOI: 10.1145/3173162.3173209 (2018).

[11] Martins, J., Ahmed, M., Raiciu, C., Olteanu, V., Honda, M., Bifulco, R. and Huici, F.: ClickOS and the Art of Network Function Virtualization, *11th USENIX Symposium on Networked Systems Design and Implementation*, NSDI '14, Seattle, WA, USA, USENIX Assocation, pp. 459–473 (online), available from ⟨https://www.usenix.org/system/files/conference/nsdi14/nsdi14-paper-martins.pdf⟩ (2014).

[12] 金津 穂，田崎 創，宇夫 陽次郎，山田浩史：クラウド環境を指向するライブラリ OS の分類のための起動にかかる時間の計測，*Internet Conference 2016*, IC2016, Tokyo, Japan,（オンライン），入手先 ⟨https://orum.in/ic2016-mkanatsu.pdf⟩ (2016).

[13] 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, *19th ACM SIGOPS Symposium on Operating System Principles*, SOSP '03, Bolton Landing, NY, USA, ACM, pp. 164–177 (online), DOI: 10.1145/1165389.945462 (2003).

[14] Manco, F., Lupu, C., Schmidt, F., Mendes, J., Kuenzer, S., Sati, S., Yasukata, K., Raiciu, C. and Huici, F.: My VM is Lighter (and Safer) than your Container, *26th ACM Symposium on Operating Systems Principles*, SOSP '17, Shanghai, China, ACM, pp. 218–233 (online), DOI: 10.1145/3132747.3132763 (2017).

[15] Kantee, A.: Flexible Operating System Internals: The Design and Implementation of the Anykernel and Rump Kernels, Phd, Aalto University (2012).

[16] 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, *9th ACM SIGOPS/EuroSys European Conference on Computer Systems*, EuroSys '14, Amsterdam, The Netherlands, ACM, pp. 9:1–9:14 (online), DOI: 10.1145/2592798.2592812 (2014).

[17] 縣 直道，大須賀 敦俊，窪田 貴文，河野健二：仮想化環境に特化した単一アドレス空間 OS における疑似マルチプロセス実行，*Summer United Workshops on Parallel, Distributed and Cooperative Processing 2017*, SWoPP 2017, Vol. 2017-OS-141, No. 9, Akita, Japan,（オンライン），入手先 ⟨http://id.nii.ac.jp/1001/00182762/⟩ (2017).

[18] Lagar-Cavilla, H. A., Whitney, J. A., Scannell, A., Patchin, P., Rumble, S. M., de Lara, E., Brudno, M. and Satyanarayanan, M.: SnowFlock: Rapid Virtual Machine Cloning for Cloud Computing, *4th ACM SIGOPS/EuroSys European Conference on Computer Systems*, EuroSys '09, Nurenberg, Germany, ACM, pp. 1–12 (online), DOI: 10.1145/1519065.1519067 (2009).

[19] Bryant, R., Tumanov, A., Irzak, O., Scannell, A., Joshi, K., Hiltunen, M., Lagar-Cavilla, H. A. and de Lara, E.: Kaleidoscope: Cloud Micro-Elasticity via VM State Coloring, *6th ACM SIGOPS/EuroSys European Conference on Computer Systems*, EuroSys '11, Salzburg, Austria, ACM, pp. 273–286 (online), DOI: 10.1145/1966445.1966471 (2011).

[20] Wang, K., Rao, J. and Xu, C.-Z.: Rethink the Virtual Machine Template, *7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '11, Newport Beach, CA, USA, ACM, pp. 39–50 (online), DOI: 10.1145/2007477.1952690 (2011).

[21] Bellard, F.: QEMU, a Fast and Portable Dynamic Translator, *2005 USENIX Annual Technical Conference*, USENIX ATC '05, Anaheim, CA, USA, USENIX Assocation, pp. 41–46 (online), available from ⟨https://www.usenix.org/legacy/publications/library/proceedings/usenix05/tech/freenix/bellard.html⟩ (2005).

[22] Kivity, A., Kamay, Y., Laor, D., Lublin, U. and Liguori, A.: kvm: the Linux Virtual Machine Monitor, *Ottawa Linux Symposium*, OLS '07, Vol. 1, Ottawa, ON, Canada, The Linux Foundation, pp. 225–230 (online), available from ⟨https://www.kernel.org/doc/ols/2007/ols2007v1-pages-225-230.pdf⟩ (2007).

[23] Purdila, O., Grijincu, L. A. and Tapus, N.: LKL: The Linux Kernel Library, *9th RoEduNet International Conference: Networking in Education and Research*, RoEduNet '10, Sibiu, Romania, IEEE, pp. 328–333 (online), available from ⟨http://ieeexplore.ieee.org/document/5541547/⟩ (2010).

[24] Utsumi, M., Tazaki, H. and Esaki, H.: /dev/stdpkt: A Service Chaining Architecture with Pipelined Operating System Instances in a Unix Shell, *Asian Internet Engineering Conference*, AINTEC '17, Bangkok, Thailand, ACM, pp. 8–15 (online), DOI: 10.1145/3154970.3154972 (2017).

[25] Madhavapeddy, A., Leonard, T., Skjegstad, M., Gazagnaire, T., Sheets, D., Scott, D., Mortier, R., Chaudhry, A., Singh, B., Ludlam, J., Crowcroft, J. and Leslie, I.: Jitsu: Just-In-Time Summoning of Unikernels, *12th USENIX Symposium on Networked Systems Design and Implementation*, NSDI '15, Oakland, CA, USA, USENIX Assocation, pp. 559–573 (online), available from ⟨https://www.usenix.org/system/files/conference/nsdi15/nsdi15-paper-madhavapeddy.pdf⟩ (2015).