特別研究報告

題目

軽量仮想化プロセスのバースト起動における総初期化時間の削減のための間隔起動の検討

指導教員
萩原 兼一 教授

報告者
金津　穂

平成 28 年 2 月 16 日

大阪大学 基礎工学部 情報科学科

平成 28 年度 特別研究報告

軽量仮想化プロセスのバースト起動における総初期化時間の削減のための間隔起動の検討

金津　穂

内容梗概

近年仮想化された CPU やメモリ，ストレージのような計算資源を提供する IaaS 型クラウドプラットフォームが増加している．この形のプラットフォームの利点のひとつに，ユーザの要求に応じて計算資源の増減が即座に可能であることが挙げられる．また，クラウドプラットフォームには PaaS 型と呼ばれるアプリケーションを配置するだけでサービスを提供できる環境が存在している．

PaaS 型の利便性と IaaS 型の利便性を共に備える存在として，コンテナ仮想化を利用したクラウドプラットフォームが出現している．しかし，コンテナ仮想化に存在する計算資源管理機構が重複し無駄が発生するという問題点が存在する．これを解決する技術としてライブラリ OS が挙げられる．

ライブラリ OS の登場により，IaaS 型においても大量のアプリケーションが動作する汎用 OS を仮想化するのではなく，アプリケーションごとに仮想化し協調させて動作させるような使い方が増加すると考えられる．しかし，そのようなクラウドプラットフォームはまだ存在しない．

そこで本報告では，ライブラリ OS 同様に軽量な仮想化インスタンスの初期化時間を計測することで，ライブラリ OS を用いるクラウドプラットフォーム上でのインスタンスの初期化・起動の設計について検討する．クラウドサービスプロバイダが使用する，1 台あたり数コア–十数コアの CPU をもつコモディティなサーバの上において，仮想化したインスタンスを起動する場合，特に数秒の間に数百台–数千台を起動するバースト起動について計測した．(1) 起動するインスタンスに割り当てるメモリの上限 (2) 起動するインスタンスの台数 (3) 個々のインスタンスの起動の間隔時間，の 3 つの項について変化させ，それぞれが初期化時間に与える影響を分析した．

計測の結果より，(1) は初期化時間に対してほぼ影響しないことが判明した．(2) では要求台数を 2 回に分け起動した場合と要求台数を起動した場合に，要求台数の初期化に要する時間は後者のほうが小さいことがわかった．(3) について，起動の間隔を大きくする場合においても同じ台数ならば全ての初期化が終了する時刻は大きく変化しなかった．しかし最初に初期化が終わったインスタンスの初期化終了時刻は起動の間隔が大きいほうが 1 秒–10 秒程度早くなるという事実が判明した．

以上の事実からライブラリ OS を前提とするクラウドプラットフォームの設計について，ユーザ要求はある程度まとめて処理し，起動の間隔を起動するプログラムのサイズに応じて大きくするようなスケジューリングが必要であると考えられる．

主な用語

ライブラリ OS　　クラウド　　仮想化　　QEMU　　KVM

目 次

1 はじめに

近年 Web プラットフォームを通して仮想化された計算資源を提供する IaaS 型 [1] クラウドプラットフォームが増加している．計算資源とは CPU やメモリ，ストレージを指し，ユーザの要求に応じて即座な増減が可能である．

現在 IaaS 型クラウドプラットフォームはハードウェアで直接動作する仮想化基盤（Type1 ハイパーバイザ）[2] を用いて計算資源を仮想化している．Type1 ハイパーバイザには VMware 社の VMware ESXi [3] のほか，Windows に統合されている Hyper-V [4]，Linux カーネルに統合されている Xen・KVM [5,6] が存在する．Hyper-V，Xen，KVM の特徴として，仮想化基盤だけではなく汎用の OS（Windows，Linux）も動作してゲスト OS の管理をその汎用の OS（ホスト OS）で行えることが挙げられる．本来 IaaS 型クラウドプラットフォームでは仮想化された計算資源を提供し，ユーザは自分で OS をインストールしたり管理するが，何らかのアプリケーションを動作させるために OS の管理や操作を行う必要があるのは本末転倒である．しかし現在存在するアプリケーションのみを配置（デプロイ）する PaaS 型 [1] クラウドプラットフォームでは IaaS 型のように計算資源を必要に応じて増減させること不可能である．

そこで OS の名前空間の分離の機能により仮想化を行うコンテナ仮想化 [7] を利用したクラウドプラットフォームが登場した．コンテナを利用するクラウドプラットフォームとして Google Container Engine（GKE）が挙げられる [8]．GKE は Google の提供する IaaS 型クラウドプラットフォーム（Google Compute Engine: GCE）[9] の上に Linux OS とコンテナ管理ツールをインストールし提供される．ユーザは Web プラットフォームから GCE の機能により計算資源を自由に増減可能であり，また，Web UI を通してコンテナのみをデプロイすることで簡単にアプリケーションを稼動できる．

コンテナ型仮想化にも，汎用 OS の提供するファイルシステムや名前空間分離機能に依存するため，IaaS 型クラウドプラットフォーム上で用いるために汎用 OS をインストールする必要があるという点で問題がある．Type1 ハイパーバイザとコンテナのを動作させるに汎用 OS の両方で計算資源管理が行われるため大きなオーバーヘッドが存在する．

ライブラリ OS [10–13] と呼ばれる技術がこの問題の解決として挙げられる．ライブラリ OS とは，仮想化を前提にすることで，デバイスドライバをほぼ持たず，メモリ管理やスケジューリングも最低限にした OS だ．ライブラリ OS とアプリケーションは 1 対 1 でリンクしコンテナを作る．コンテナ仮想化と違い IaaS 型クラウドプラットフォームの Type1 ハイパーバイザの上で直接アプリケーションのコンテナが動作することになる．

コンテナ仮想化，そしてライブラリ OS の登場により IaaS 型クラウドプラットフォームの上で巨大な仮想マシンを動作させるのではなく小さなコンテナやアプリケーションが動く

程度のサイズの仮想マシンを大量に動作させるように変化していくと考えられる．例えば Network Functions Virtualization [14] のように機能ごとにブロックになった仮想化された計算資源のインタンスをプラットフォームに複数デプロイし，並列に動作させ高速化を図る場合である．この使い方のためにはユーザの要求に合わせて極短時間に起動する必要がある．このようなユーザ要求に対して僅かな時間で大量の仮想化インスタンスを起動できる（＝バースト起動可能な）プラットフォームはライブラリ OS を前提として設計されたものは存在せず現在のクラウドプラットフォームは妥当ではない．したがってライブラリ OS をクラウドで実用するためにプラットフォームを新たに設計する必要がある．しかし，ライブラリ OS のバースト起動に関しての知見は現在ほぼ存在しない．

そこで本報告では，ライブラリ OS 同様に軽量な仮想化インスタンス対象にバースト起動の初期化時間を実験と分析を行う．具体的には，1 台あたり数コア–十数コアの CPU をもつコモディティなサーバの上で，一定時間に何台初期化が終了するか，要求された台数のインスタンスを全て初期化するのにかかる時間は何秒か計測し，インスタンスの割り当てメモリ量や台数，起動の間隔時間との関連を分析する．

以降，2 章で仮想化技術，3 章で関連研究の紹介をし，4 章で実験で想定しているユースケースと環境について説明する．5 章ではまず実験の主眼を説明する．のちに，計測を行うにあたり妥当な計測方法と計測の対象について検討し，実験の結果とその考察を示す．最後に 6 章で本報告をまとめる．

2 仮想化技術

2.1 ハイパーバイザの分類

計算機を抽象化した複数のプロセスを制御および管理する OS をスーパーバイザと呼ぶ．一方，ハードウェアの仮想化により，複数の OS を制御および管理する仮想化基盤をハイパーバイザと呼ぶ．Popek と Goldberg は，ハイパーバイザに対して 2 種類の分類を提唱した [2].

Type1 ハイパーバイザ ハードウェア上で直接ハイパーバイザが動作する．例として，VMware 社の VMware ESXi [3] が挙げられる．ベアメタル・ハイパーバイザとも呼ばれる．

Type2 ハイパーバイザ OS 上の単一アプリケーションとして動作する．つまり，計算機のエミュレータアプリケーションである．ただし，狭義においては，ハイパーバイザに含めない．

2.2 QEMU

QEMU とは，オープンソースの汎用マシンエミュレータである．QEMU は，ユーザモードエミュレーションとシステムモードエミュレーションからなる 2 つの動作モードを持つ [15].

ユーザモードエミュレーション あるアーキテクチャのプロセッサを搭載したマシン上で，アーキテクチャの異なるプロセッサをエミュレーションする動作モードである．この動作モードでは，プロセッサの命令を動的に変換し，システム自体はホストのものを使用する．

システムモードエミュレーション 周辺機器を含めてコンピュータシステム全体をエミュレーションする動作モードである．この動作モードでは，ホストマシン上で複数のゲストマシンを動作できることから，OS 仮想化に用いられる．

どちらの動作モードにおいても，QEMU そのものは通常 Type2 ハイパーバイザとして動作する．しかし，後述の KVM の管理ソフトウェアとして使用することにより，Type1 ハイパーバイザのインターフェイスにすることが可能である．

2.3 KVM

KVM は，Linux カーネルに内蔵された Type1 ハイパーバイザである [6]．ハードウェア上で独立して動作するのではなく，Linux カーネルの内部機構として動作する．Linux カーネルは，仮想化インスタンスを Linux OS のプロセスとして提供する．このため，ユーザは使い慣れた OS のプロセスとして仮想化インスタンスを管理でき，都合が良い．ユーザの観点からは Type2 ハイパーバイザにも分類される．

KVM そのものは Linux の疑似ファイル `/dev/kvm` を通して仮想化 API を提供するが，ネットワークや VGA といったハードウェアの仮想化は一切提供しない．したがって，一般的に，前述の QEMU は周辺機器の仮想化にのみ使用する．

同様に，汎用 OS に組み込まれた Type1 ハイパーバイザとして，Linux の Xen [5] および Windows の Hyper-V [4] がある．

2.4 コンテナ仮想化

コンテナ仮想化は OS 仮想化ではないが，ファイルシステムの一部分にシステム全体を再現するディレクトリを作成し，そのディレクトリをホスト OS から隔離し仮想化されたシステムに見せかける仮想化を行う．

UNIX の 4.2BSD から存在するシステムコールである `chroot()` はコンテナ仮想化の原点である．しかし，このシステムコールで実現したコンテナは名前空間や計算資源へのアクセス権限が分離されないため仮想化として不適切であり，コンテナの外部から内部，内部から外部への干渉が容易である．そこで FreeBSD jail [16] や，Linux の namespace，cgroup [7] といった技術が完全なコンテナ化のために用いられる．

2.5 ライブラリ OS

マイクロカーネルの，必要最小限だけをカーネルが提供し残りの機能はユーザ空間で実装するという考えをさらに推進した Exokernel が 1995 年に発表されている [17]．Exokernel ではカーネル部分はハードウェアの抽象化しか提供しない．残りのファイルシステムやネットワークスタックといった OS に最低限必要な機能は Exokernel のカーネル上で動作するライブラリとして提供され，アプリケーションはこのライブラリをリンクし実行する．この Exokernel が実現する OS のアーキテクチャをライブラリ OS と言う．

本来のライブラリ OS では，カーネルによりハードウェア抽象が提供されていた．ここで，さらに OS のレイヤを小さくするために，Type1 ハイパーバイザがハードウェア抽象を提供することを前提にするライブラリ OS が登場した．Exokernel 以上に効率良くハードウェアを使用する事が可能になる．また，ハイパーバイザによる仮想化はライブラリ OS にリンクされている個々のアプリケーションを完全に分離する．つまり，ライブラリ OS とそれにリンクされているアプリケーションの一対をコンテナ仮想化としてみなせる (図 1).

この考えにもとづいて，Windows に実装が進められているライブラリ OS に DrawBridge [10] や KVM の開発者が開発した OS^{v} [11]，Xen の開発者が開発する Unikernel [12] がある．

この技術によって，仮想化基盤の上にライブラリ OS をリンクしたアプリケーションコンテナを直接デプロイし計算資源の無駄な使用を抑えられる．また，仮想化による権限の分離

やコンテナ技術のアプリケーションデプロイの容易さといったメリットを受けられる．(図 1)

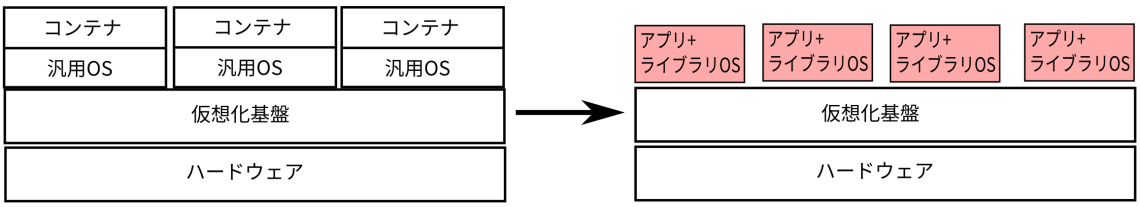

図 1: コンテナとライブラリ OS におけるスタック図

3 関連研究

3.1 複数仮想化インスタンスの起動

現在，Type1 ハイパーバイザを利用するライブラリ OS の大量起動をクラウドで利用することについて，Joao らの研究 [13] が存在する．

NFV（Network Functions Virtualisations）[14] の実現手段として，ClickOS と呼ばれる Xen [5] 上に実装したライブラリ OS とそれを用いたミドルボックスの開発を行い，数マイクロ秒–数十マイクロ秒の間に数万台起動する試みである．

しかし，この研究では NFV のみに主眼を置いている．これはコンテナ仮想化によるクラウドアプリケーションプラットフォームの延長線上として，ライブラリ OS を使用するクラウドプラットフォームについて考え，その実現のために計測・分析を行うという本報告の目的と方向性が異なる．

3.2 ライブラリ OS のクラウドプラットフォームでの使用

三宮らの研究 [18] はライブマイグレーションについて，Type1 ハイパーバイザによる仮想化のためライブラリ OS が，コンテナ仮想化より容易であることに注目している．

この研究は Avi らの OS^{v} [11] を用いて，クラウドプラットフォーム内の実行中のコンテナを動的に移動することで複数のコンテナを同一の計算ノードに集約し，計算資源の効率化を図るものである．

クラウドプラットフォームのインスタンスの起動時に着目する本報告とは，既に実行状態にあるインスタンスに着目する点で異なっている．

4 仮想化インスタンスのバースト起動

4.1 バースト起動の特徴と問題点

ライブラリ OS によるコンテナ化の登場により，IaaS 型クラウドプラットフォームの上で直接コンテナを扱うという考えが生まれている．そのひとつの例として挙げられるのは NFV のライブラリ OS による実装である [13,14]．IaaS 型クラウドプラットフォームでは従来，汎用の OS を仮想化しその上で様々なアプリケーションを動作させていた．しかし，プロセスレベルで実現されるような小さな単機能の部品をそれぞれライブラリ OS の仮想化インスタンスとして実現し大量に動作させるように変化している．

ライブラリ OS を仮想化インスタンスとして用いる IaaS 型クラウドプラットフォームでは，ユーザの要求に対して，仮想化インスタンスを大量に起動する必要が生じる．また，ユーザの要求に対してレスポンスを高速に行うために，即座の起動が必要である．

本報告では，このようなユーザの要求に対する大量の仮想化インスタンスの即座の起動をバースト起動と言う．具体的には，サービスプロバイダが用いる数コア–数十コアの CPU を持つコモディティなサーバ上において，数秒のうちに数百台–数千台の仮想化インスタンスを起動することをさす．

4.2 間隔起動

仮想化インスタンスのバースト起動について，各インスタンスを同時に全て起動すると仮想化ソフトウェアの起動処理に負荷がかかり初期化の時間が遅くなる．そこで，バースト起動で各インスタンスの起動間隔を空けることが重要になる．ただし，起動間隔が広すぎる場合バースト起動ではなくただの起動になってしまい，ユーザ要求を即座に満たせなくなってしまう．

本報告では，仮想化インスタンスのバースト起動について，間隔起動を考慮し実験する．

5 実験

5.1 実験概要

5.1.1 計測方法

今回はライブラリ OS ではなくとても小さなアセンブリプログラムを仮想化基盤の上でロード・実行し，その初期化時間を測り仮想化インスタンスそのものの初期化の挙動を分析した．実行したプログラムは次のとおりである．

- 実行環境である x86 のアセンブリによって記述
- 最低限のレジスタやスタックの初期化の後に文字列出力を行なう
- 文字列出力は I/O ポートへ値を mov し，out 命令によりシリアルから文字を出力する

仮想化インスタンスは QEMU のプロセスとして起動するため，初期化時間を知るためにはその初期化終了を何らかの手段で親プロセスに通知する必要がある．そこで，今回はシリアルから文字出力を行なうように I/O ポートへ値を書き込み out 命令を用いるという手段を採用した．QEMU はシリアルの出力を標準出力に出力できるため，親プロセスは個々の仮想化インスタンスの標準出力の出力を監視すれば良い．

初期化時間の計測は次の方法で行なった．

- 仮想化インスタンスは fork+exec を用いて起動する．
- 仮想化インスタンス 1 つにつき 1 つのスレッドを起動し，各仮想化インスタンスの標準出力を監視する
- 仮想化インスタンスのプロセス起動時点を初期化開始時刻とする
- 標準出力からの文字列をスレッドが受けとった時点で初期化終了時刻とする．

計測の目的として，バースト起動の際に全体ではなく個々の仮想化インスタンスの初期化にかかる時間を知りたかったために，計測は 1 インスタンスにつき 1 スレッドを起動しインスタンスの監視を行なった．

5.2 実験の目的

ライブラリ OS を前提とするクラウドプラットフォームは，ユーザ要求に対してバーストな起動が発生すると考えられる．ライブラリ OS のためのプラットフォームの設計のために，ユーザ要求に対するレスポンスを知る必要がある．そこで，ライブラリ OS はバースト起動の初期化に要する時間を計測する．また，起動するインスタンス数，インスタンスの起動間隔および割り当てメモリ上限といったバースト起動におけるパラメータの変化と初期化に要

する時間の影響を分析した．さらに，クラウドプラットフォームの設計にはなにを考慮すべきか，何が必要なのかといった要素を検討する．

本報告では，QEMU+KVM 上で動作する極小の文字列出力プログラムで実現した，ライブラリ OS 同様に軽量な仮想化プロセスを用いて計測を行う．

5.3 実験結果

5.3.1 計測環境

次の環境で実験した．

- CPU：Xeon E3-1230 V2
- メモリ：DDR3 32 GB
- ストレージ：SATA SSD
- OS：Fedora 23

また，次のように値を変化させ，計測実験を行なった．

- インスタンスメモリ割り当て上限：16 MB 刻みで 16 MB–256 MB
- 台数：100 台刻みで 100–1000 台
- 間隔：2 ミリ秒刻みで 1 ミリ秒–19 ミリ秒

5.3.2 メモリの変更

表 1 に割り当てメモリ上限を変化させた場合の最初の初期化時刻を，表 2 に割り当てメモリ上限を変化させた場合の全ての初期化終了時刻を示す．行った実験のうち，16 MB，128 MB，256 MB の場合を示す．

5.3.3 台数の変更

表 3 に起動台数を変化させた場合の最初の初期化時刻を，表 4 に起動台数を変化させた場合の全ての初期化終了時刻を示す．ただし，いずれも割り当てメモリ上限は 16 MB の場合である．

5.3.4 間隔の変更

表 5 に起動間隔を変化させた場合の最初の初期化時刻を，表 6 に起動間隔を変化させた場合の全ての初期化終了時刻を示す．ただし，いずれも割り当てメモリ上限は 16 MB の場合である．

表 1: 割り当てメモリ上限を変化させたときの最初の初期化時刻

(a) 起動台数 100 台

起動間隔 [ms]	最初の初期化時刻 [s]		
	16 MB	128 MB	256 MB
1	4.04	4.00	3.81
3	3.90	3.85	3.89
5	3.75	3.96	3.86
7	3.94	3.87	4.19
9	3.96	4.08	3.65
11	4.14	4.18	4.02
13	3.65	4.13	3.99
15	4.06	4.27	3.94
17	4.15	3.97	3.87
19	4.06	3.87	4.21

(b) 起動台数 200 台

起動間隔 [ms]	最初の初期化時刻 [s]		
	16 MB	128 MB	256 MB
1	5.62	5.39	5.23
3	5.44	5.68	5.51
5	5.45	5.54	5.55
7	5.19	5.64	5.42
9	4.95	5.56	5.61
11	5.44	5.17	5.67
13	4.97	5.37	5.52
15	7.53	5.44	5.53
17	4.88	4.71	5.10
19	4.99	5.17	5.37

(c) 起動台数 300 台

起動間隔 [ms]	最初の初期化時刻 [s]		
	16 MB	128 MB	256 MB
1	7.03	7.04	7.29
3	7.53	6.67	6.67
5	7.10	6.51	7.06
7	7.25	6.82	6.55
9	7.03	6.92	7.31
11	6.29	6.28	6.76
13	6.52	6.32	6.96
15	5.00	5.73	5.93
17	5.11	5.43	5.41
19	5.28	5.58	5.33

(d) 起動台数 400 台

起動間隔 [ms]	最初の初期化時刻 [s]		
	16 MB	128 MB	256 MB
1	8.73	9.11	8.49
3	8.43	8.03	8.70
5	7.68	7.87	7.08
7	7.56	7.50	7.80
9	7.28	7.94	7.99
11	7.23	7.21	7.13
13	6.64	6.60	6.17
15	5.76	6.18	5.82
17	5.52	5.68	6.03
19	4.99	4.55	5.78

(e) 起動台数 500 台

起動間隔 [ms]	最初の初期化時刻 [s]		
	16 MB	128 MB	256 MB
1	6.27	9.28	10.17
3	10.04	9.31	8.15
5	8.81	9.81	9.93
7	8.05	8.26	4.66
9	7.51	7.49	8.12
11	6.94	7.02	7.42
13	6.97	6.57	6.42
15	5.86	6.55	5.55
17	4.92	5.79	5.10
19	4.91	4.53	5.45

(f) 起動台数 600 台

起動間隔 [ms]	最初の初期化時刻 [s]		
	16 MB	128 MB	256 MB
1	9.12	9.12	9.00
3	5.07	6.76	11.81
5	8.87	9.62	6.43
7	9.39	7.40	9.08
9	8.71	8.39	7.13
11	6.81	8.05	5.85
13	6.64	6.59	6.40
15	6.40	5.85	6.27
17	4.81	5.58	4.86
19	5.46	4.79	5.02

(g) 起動台数 700 台

起動間隔 [ms]	最初の初期化時刻 [s]		
	16 MB	128 MB	256 MB
1	7.05	12.96	12.20
3	13.42	12.41	12.62
5	8.96	8.45	10.33
7	9.68	9.99	8.44
9	6.72	7.79	7.71
11	7.89	6.49	6.31
13	6.04	6.11	5.68
15	6.32	5.30	4.92
17	4.96	5.43	5.43
19	5.22	5.23	5.03

(h) 起動台数 800 台

起動間隔 [ms]	最初の初期化時刻 [s]		
	16 MB	128 MB	256 MB
1	10.88	10.23	13.26
3	12.98	10.08	13.55
5	11.61	11.31	10.51
7	9.46	9.13	6.73
9	4.41	7.92	8.53
11	7.38	6.63	7.21
13	5.57	5.47	6.07
15	6.40	5.88	5.75
17	5.55	5.09	5.37
19	4.58	4.99	5.13

(i) 起動台数 900 台

起動間隔 [ms]	最初の初期化時刻 [s]		
	16 MB	128 MB	256 MB
1	13.03	9.99	9.89
3	12.84	14.02	11.32
5	7.58	9.34	5.04
7	8.63	7.54	9.80
9	7.00	7.08	8.62
11	6.21	6.89	6.74
13	6.46	5.99	5.92
15	5.75	5.84	5.87
17	5.08	5.49	4.52
19	4.68	5.12	5.42

(j) 起動台数 1000 台

起動間隔 [ms]	最初の初期化時刻 [s]		
	16 MB	128 MB	256 MB
1	14.38	9.46	16.47
3	15.34	8.88	7.66
5	10.06	10.22	11.67
7	9.06	9.22	8.80
9	4.53	8.04	6.95
11	7.36	7.70	6.04
13	6.37	5.59	5.92
15	5.69	6.18	5.83
17	5.47	5.42	5.37
19	4.87	5.46	4.92

表 2: 割り当てメモリ上限を変化させたときの全ての初期化終了時刻

(a) 起動台数 100 台

起動間隔 [ms]	全ての初期化終了時刻 [s]		
	16 MB	128 MB	256 MB
1	4.56	4.36	4.03
3	4.13	4.22	4.20
5	4.13	4.30	4.24
7	4.41	4.25	4.56
9	4.31	4.26	4.29
11	4.35	4.42	4.42
13	4.36	4.50	4.25
15	4.36	4.46	4.34
17	4.52	4.43	4.44
19	4.46	4.31	4.37

(b) 起動台数 200 台

起動間隔 [ms]	全ての初期化終了時刻 [s]		
	16 MB	128 MB	256 MB
1	6.40	6.44	6.48
3	6.39	6.34	6.38
5	6.36	6.33	6.39
7	6.30	6.27	6.33
9	6.42	6.40	6.33
11	6.33	6.46	6.39
13	6.52	6.52	6.48
15	8.17	6.52	6.46
17	6.54	6.36	6.52
19	6.42	6.48	6.59

(c) 起動台数 300 台

起動間隔 [ms]	全ての初期化終了時刻 [s]		
	16 MB	128 MB	256 MB
1	8.27	8.38	8.50
3	8.35	8.33	8.23
5	8.19	8.49	8.50
7	8.33	8.42	8.41
9	8.48	8.43	8.53
11	8.42	8.46	8.58
13	8.50	8.59	8.69
15	8.41	8.52	8.48
17	8.39	8.66	8.56
19	8.55	8.65	8.52

(d) 起動台数 400 台

起動間隔 [ms]	全ての初期化終了時刻 [s]		
	16 MB	128 MB	256 MB
1	10.30	10.57	10.44
3	10.50	10.34	10.53
5	10.44	10.27	10.51
7	10.41	10.41	10.41
9	10.48	10.62	10.47
11	10.50	10.61	10.57
13	10.52	10.55	10.65
15	10.59	10.56	10.76
17	10.55	10.65	10.79
19	10.83	10.66	10.85

(e) 起動台数 500 台

起動間隔 [ms]	全ての初期化終了時刻 [s]		
	16 MB	128 MB	256 MB
1	12.17	12.39	12.53
3	12.43	12.44	12.45
5	12.43	12.39	12.55
7	12.64	12.61	12.77
9	12.59	12.55	12.65
11	12.54	12.66	12.72
13	12.61	12.70	12.60
15	12.75	12.83	12.79
17	12.62	12.86	12.73
19	12.72	12.74	12.83

(f) 起動台数 600 台

起動間隔 [ms]	全ての初期化終了時刻 [s]		
	16 MB	128 MB	256 MB
1	14.31	14.46	14.50
3	14.38	14.23	14.64
5	14.42	14.56	14.71
7	14.69	14.76	14.84
9	14.82	14.71	14.75
11	14.76	14.99	14.89
13	14.71	14.85	15.03
15	14.80	14.99	14.85
17	14.80	14.85	14.96
19	15.01	14.87	15.14

(g) 起動台数 700 台

起動間隔 [ms]	全ての初期化終了時刻 [s]		
	16 MB	128 MB	256 MB
1	16.21	16.59	16.70
3	16.52	16.59	16.67
5	16.54	16.69	16.67
7	16.87	16.86	16.96
9	16.84	16.90	17.01
11	16.96	16.91	17.01
13	16.93	16.99	17.11
15	17.03	16.94	16.98
17	17.13	17.10	17.16
19	17.27	17.28	17.29

(h) 起動台数 800 台

起動間隔 [ms]	全ての初期化終了時刻 [s]		
	16 MB	128 MB	256 MB
1	18.30	18.37	18.60
3	18.74	18.53	18.83
5	18.88	18.91	18.96
7	18.96	19.05	19.15
9	18.92	18.95	19.17
11	19.15	19.15	19.14
13	19.11	19.23	19.35
15	19.21	19.46	19.45
17	19.31	19.51	19.52
19	19.46	19.71	19.60

(i) 起動台数 900 台

起動間隔 [ms]	全ての初期化終了時刻 [s]		
	16 MB	128 MB	256 MB
1	20.43	20.65	20.98
3	20.79	21.00	20.99
5	21.06	21.15	21.18
7	21.13	21.22	21.39
9	21.22	21.22	21.45
11	21.47	21.61	21.60
13	21.46	21.55	21.54
15	21.51	21.67	21.72
17	21.70	22.17	21.99
19	21.85	22.16	22.33

(j) 起動台数 1000 台

起動間隔 [ms]	全ての初期化終了時刻 [s]		
	16 MB	128 MB	256 MB
1	22.94	23.13	23.11
3	23.03	23.18	23.20
5	23.13	23.39	23.48
7	23.33	23.68	23.62
9	23.52	23.80	23.66
11	23.68	23.84	23.82
13	23.91	23.77	23.88
15	24.08	24.06	24.08
17	24.14	24.29	24.27
19	24.55	24.66	24.56

表 3: 起動台数を変化させたときの最初の初期化時刻

起動間隔 [ms]	各起動台数での最初の初期化時刻 [s]									
	100	200	300	400	500	600	700	800	900	100
1	4.04	5.62	7.03	8.73	6.27	9.12	7.05	10.88	13.03	14.38
3	3.90	5.44	7.53	8.43	10.04	5.07	13.42	12.98	12.84	15.34
5	3.75	5.45	7.10	7.68	8.81	8.87	8.96	11.61	7.58	10.06
7	3.94	5.19	7.25	7.56	8.05	9.39	9.68	9.46	8.63	9.06
9	3.96	4.95	7.03	7.28	7.51	8.71	6.72	4.41	7.00	4.53
11	4.14	5.44	6.29	7.23	6.94	6.81	7.89	7.38	6.21	7.36
13	3.65	4.97	6.52	6.64	6.97	6.64	6.04	5.57	6.46	6.37
15	4.06	7.53	5.00	5.76	5.86	6.40	6.32	6.40	5.75	5.69
17	4.15	4.88	5.11	5.52	4.92	4.81	4.96	5.55	5.08	5.47
19	4.06	4.99	5.28	4.99	4.91	5.46	5.22	4.58	4.68	4.87

表 4: 起動台数を変化させたときの全ての初期化終了時刻

起動間隔 [ms]	各起動台数での全ての初期化終了時刻 [s]									
	100	200	300	400	500	600	700	800	900	100
1	4.56	6.40	8.27	10.30	12.17	14.31	16.21	18.30	20.43	22.94
3	4.13	6.39	8.35	10.50	12.43	14.38	16.52	18.74	20.79	23.03
5	4.13	6.36	8.19	10.44	12.43	14.42	16.54	18.88	21.06	23.13
7	4.41	6.30	8.33	10.41	12.64	14.69	16.87	18.96	21.13	23.33
9	4.31	6.42	8.48	10.48	12.59	14.82	16.84	18.92	21.22	23.52
11	4.35	6.33	8.42	10.50	12.54	14.76	16.96	19.15	21.47	23.68
13	4.36	6.52	8.50	10.52	12.61	14.71	16.93	19.11	21.46	23.91
15	4.36	8.17	8.41	10.59	12.75	14.80	17.03	19.21	21.51	24.08
17	4.52	6.54	8.39	10.55	12.62	14.80	17.13	19.31	21.70	24.14
19	4.46	6.42	8.55	10.83	12.72	15.01	17.27	19.46	21.85	24.55

表 5: 起動間隔を変化させたときの最初の初期化時刻

起動台数	各起動間隔 [ms] での最初の初期化時刻 [s]									
	1	3	5	7	9	11	13	15	17	19
100	4.04	3.90	3.75	3.94	3.96	4.14	3.65	4.06	4.15	4.06
200	5.62	5.44	5.45	5.19	4.95	5.44	4.97	7.53	4.88	4.99
300	7.03	7.53	7.10	7.25	7.03	6.29	6.52	5.00	5.11	5.28
400	8.73	8.43	7.68	7.56	7.28	7.23	6.64	5.76	5.52	4.99
500	6.27	10.04	8.81	8.05	7.51	6.94	6.97	5.86	4.92	4.91
600	9.12	5.07	8.87	9.39	8.71	6.81	6.64	6.40	4.81	5.46
700	7.05	13.42	8.96	9.68	6.72	7.89	6.04	6.32	4.96	5.22
800	10.88	12.98	11.61	9.46	4.41	7.38	5.57	6.40	5.55	4.58
900	13.03	12.84	7.58	8.63	7.00	6.21	6.46	5.75	5.08	4.68
1000	14.38	15.34	10.06	9.06	4.53	7.36	6.37	5.69	5.47	4.87

表 6: 起動間隔を変化させたときの全ての初期化終了時刻

起動台数	各起動間隔 [ms] での全ての初期化終了時刻 [s]									
	1	3	5	7	9	11	13	15	17	19
100	4.56	4.13	4.13	4.41	4.31	4.35	4.36	4.36	4.52	4.46
200	6.40	6.39	6.36	6.30	6.42	6.33	6.52	8.17	6.54	6.42
300	8.27	8.35	8.19	8.33	8.48	8.42	8.50	8.41	8.39	8.55
400	10.30	10.50	10.44	10.41	10.48	10.50	10.52	10.59	10.55	10.83
500	12.17	12.43	12.43	12.64	12.59	12.54	12.61	12.75	12.62	12.72
600	14.31	14.38	14.42	14.69	14.82	14.76	14.71	14.80	14.80	15.01
700	16.21	16.52	16.54	16.87	16.84	16.96	16.93	17.03	17.13	17.27
800	18.30	18.74	18.88	18.96	18.92	19.15	19.11	19.21	19.31	19.46
900	20.43	20.79	21.06	21.13	21.22	21.47	21.46	21.51	21.70	21.85
1000	22.94	23.03	23.13	23.33	23.52	23.68	23.91	24.08	24.14	24.55

5.4 考察

5.4.1 メモリの変更

まずはインスタンスへの割り当てメモリの上限について，これは，変化させた範囲では初期化にかかる時間に大きな変化は見られなかった（図 2）.

5.4.2 台数の変更

次は起動の台数の違いについて，13 ミリ秒間隔で 400 台起動した場合と 800 台起動した場合を示す（図 3）．400 台起動の場合，全インスタンスの初期化は 10.52 秒で終了しており，800 台起動の場合は 19.11 秒で終了している．800 台起動した場合の全初期化時間よりも 400 台起動の全初期化時間を 2 倍した時間のほうが 0.93 秒大きい.

5.4.3 間隔の変更

最後に起動間隔の違いについて，800 台を 7 ミリ秒間隔で起動した場合と 13 ミリ秒間隔で起動した場合を示す（図 4）．全インスタンスの初期化終了時刻は 7 ミリ秒間隔の場合は 18.96 秒，13 ミリ秒間隔の場合 19.11 秒となっており，その差は 0.15 秒である．しかし，1 台目のインスタンスの初期化終了時刻をみてみると，7 ミリ秒間隔の場合は 9.46 秒，13 ミリ秒間隔の場合 5.57 秒であり，その差は 3.89 秒である．全初期化終了時刻の差に対して個々のインスタンスの初期化終了時刻の差がとても大きく，これは，ユーザ要求に対してのレスポンスに間隔がある程度大きいほうが非常に有利であると言える.

5.4.4 総括

実際のクラウドプラットフォームでは，ユーザ要求によって割り当てメモリの上限と起動の台数が指定され，プラットフォーム側で変更できない．しかし，本計測の結果より，起動する台数の要求が 1 ノードに対しとても大きい場合は複数のノードに割り当てるように分割し，起動する台数の要求が 1 ノードに対しとても小さい場合は他の要求と一緒にまとめて起動するといったスケジューリングが有効であると考えられる．また，起動の間隔についても，スケジューリングの結果最終的に起動する台数が小さければ大きめに間隔を取り，台数が大きければ小さめに間隔を取るといった対策によりユーザへのレスポンス速度を高められる.

計測の結果から台数や間隔とは違う事実も判明した．それは初期化の終了時刻について，最初の数秒間はどのインスタンスも初期化が終了せず，その数秒後は逐次初期化が終了していくという挙動を見せている点である．これは QEMU のスレッド処理やカーネル内部のス

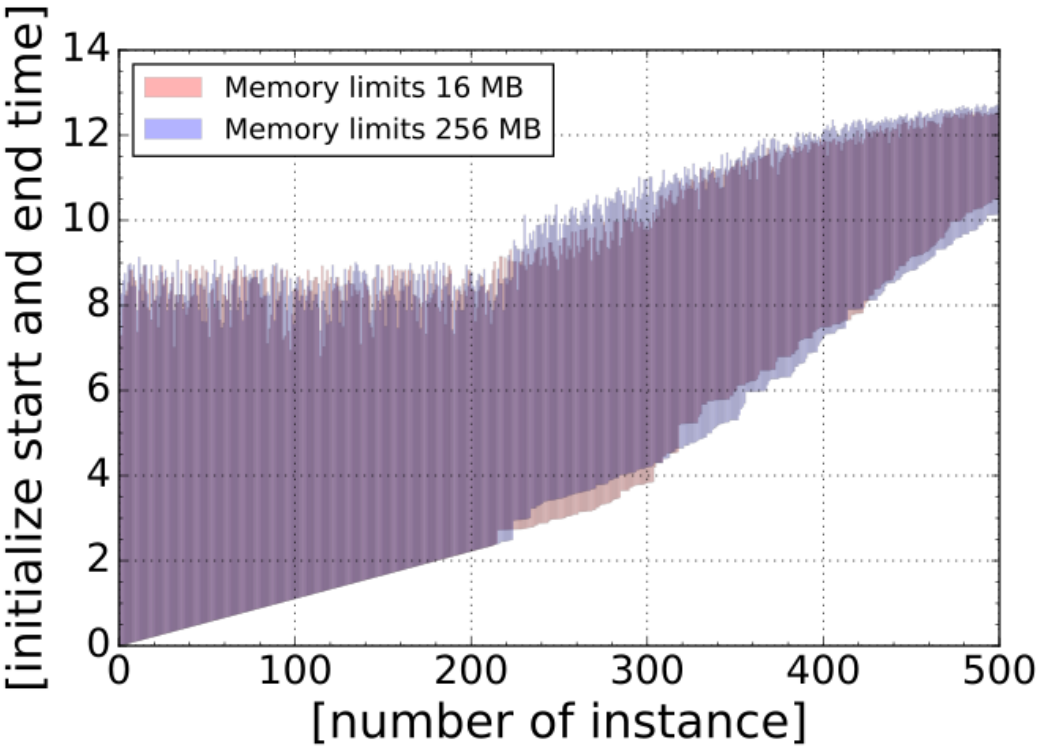

図 2: 11 ミリ秒間隔で 500 台を起動する時割り当てメモリ上限を 16 MB と 256 MB で変化させた場合の初期化時間比較

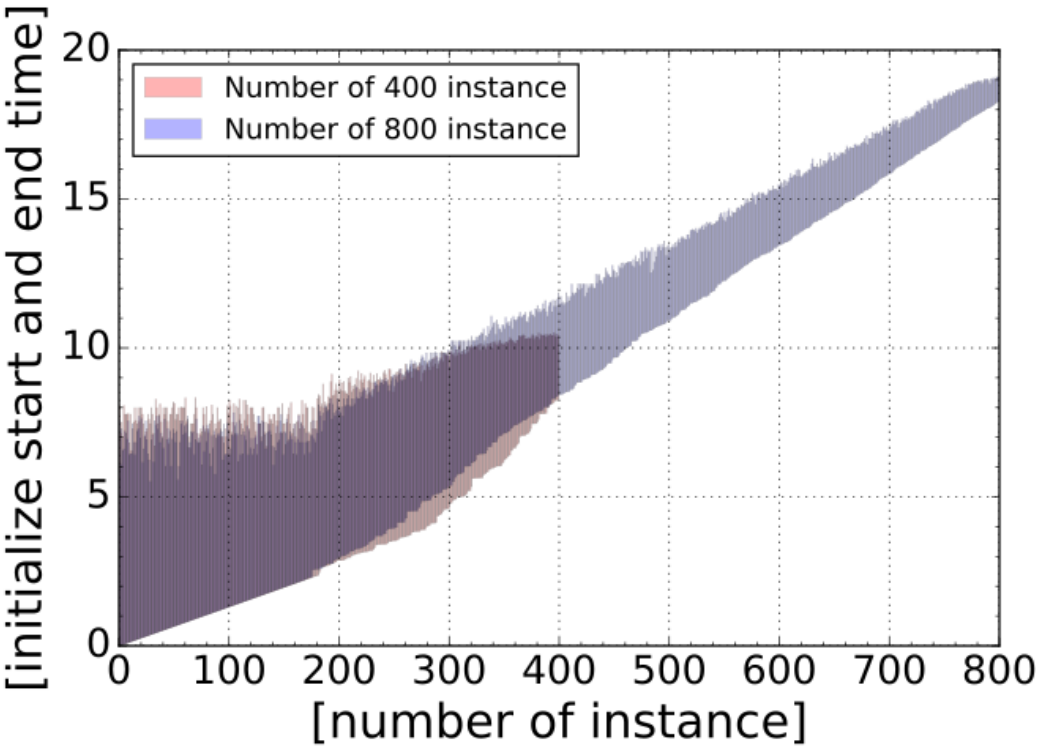

図 3: 13 ミリ秒の起動間隔で 400 台のインスタンスを起動した場合と 800 台のインスタンスを起動した場合の初期化時間比較

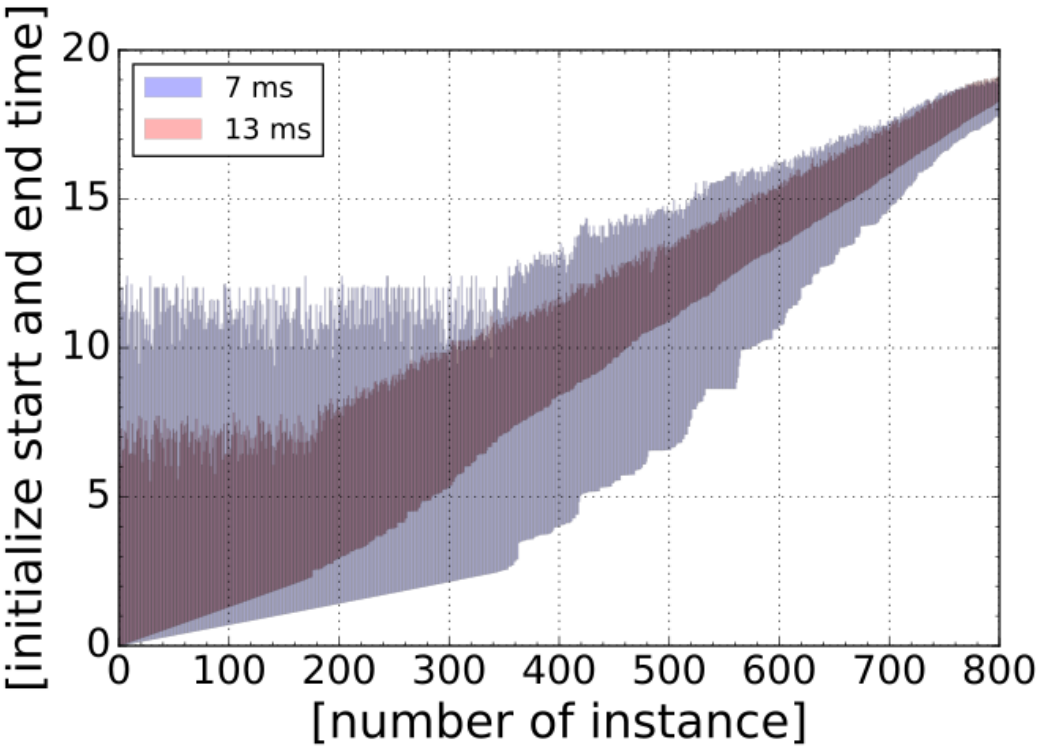

図 4: 800 台のインスタンスの起動間隔を 7 ミリ秒にした場合と 13 ミリ秒にした場合の初期化時間比較

ケジューリングといった部分にも影響されている可能性があるため，更なる調査をしなければならない.

6 まとめ

本報告では，軽量な仮想化インスタンスの初期化時間計測を通して，軽量な仮想マシンを用いたコンテナであるライブラリ OS を前提としたクラウドプラットフォームのユーザ要求に対し，起動のスケジューリングを検討した.

検討の結果，次の 3 つの事実が判明した.

- 割り当てメモリ容量の上限は初期化時間にほぼ影響しない.
- 起動要求に対し，要求台数を 2 回に分け起動した場合と要求台数を起動した場合では，要求台数の初期化にかかる時間は後者のほうが小さい.
- 起動の間隔を 20 ミリ秒ほど空けても全部の初期化の終了の時間は 0.5 秒以下しか違いがない．一方，1 台目の初期化の終了時刻は，数秒から十数秒小さくなる.

以上より，ユーザの起動要求に対し要求通りに起動するのではなく，ある程度のリクエストをキューイングしてまとめて処理する，また，要求の台数や起動するプログラムによって，起動の間隔を 10 ミリ秒以上空けるといったスケジューリングが，ライブラリ OS を用いるクラウドプラットフォームに必要だと考えられる．ただし，計測の結果，最初の数百台の初期化が数秒経過するまで終了せず，ある時刻で一斉に初期化が終了し，その後残りの初期化が逐次終了するという挙動が見られた.

今後の課題として，まず数秒経過するまで最初の数百台の初期化が終了しないという挙動について，Linux におけるプロセススケジューリング，Linux カーネルおよび QEMU の実装といった細部まで調査し原因を追求する．次に，今回得られた知見が KVM ではなく Xen や Hyper-V といったハイパーバイザでも同様か検証し，ライブラリ OS を用いるクラウドプラットフォームの設計に取り組む.

謝辞

本研究の全過程を通じて日頃からの御督励，御指導を賜りました萩原兼一教授に深く感謝致します.

本研究を纏めるにあたって貴重な御意見，御指導を賜りました伊野文彦准教授に深く感謝いたします.

本研究の全過程を通じて丁寧かつ熱心な御指導，適切な御助言を賜りました置田真生助教に深く感謝致します.

本研究を進めるにあたって数多くの御指導，御助言を頂きました萩原研究室博士前期課程2年高木研太朗氏に感謝致します.

本研究を進めるにあたって数多くの御指導，御助言を頂きました萩原研究室博士前期課程1年三木脩弘氏に感謝致します.

本論文を執筆するにあたって御助言，御助力を頂きました萩原研究室の皆様に感謝致します.

参考文献

[1] Peter M. Mell and Timothy Grance. SP 800-145. The NIST Definition of Cloud Computing. Technical report, National Institute of Standards & Technology, Gaithersburg, MD, USA, 2011.

[2] Gerald J. Popek and Robert P. Goldberg. Formal Requirements for Virtualizable Third Generation Architectures. *Communications of the ACM*, Vol. 17, No. 7, pp. 412–421, 1974.

[3] Charu Chaubal. The Architecture of VMware ESXi. *VMware White Paper*, 2008.

[4] Robert Larson, Janique Carbone, and Microsoft Windows Virtualization Team. *Windows Server 2008 Hyper-V Resource Kit*. Microsoft Press, 1st edition, 2009.

[5] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the Art of Virtualization. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pp. 164–177, New York, NY, USA, 2003.

[6] Avi Kivity, Uri Lublin, Anthony Liguori, Yaniv Kamay, and Dor Laor. kvm: the linux virtual machine monitor. In *Proceedings of the Linux Symposium*, Vol. 1, pp. 225–230, Ottawa, Ontario, Canada, 2007.

[7] Stephen Soltesz, Herbert Pötzl, Marc E. Fiuczynski, Andy Bavier, and Larry Peterson. Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors. *SIGOPS Operating System Review*, Vol. 41, No. 3, pp. 275–287, 2007.

[8] Google Container Engine. `https://cloud.google.com/container-engine/`. Accessed 2016-2-16.

[9] Google Compute Engine. `https://cloud.google.com/compute/`. Accessed 2016-2-16.

[10] Donald E. Porter, Silas Boyd-Wickizer, Jon Howell, Reuben Olinsky, and Galen C. Hunt. Rethinking the Library OS from the Top Down. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVI, pp. 291–304, New York, NY, USA, 2011.

[11] Avi Kivity, Dor Laor, Glauber Costa, Pekka Enberg, Nadav Har'El, Don Marti, and Vlad Zolotarov. OSv: Optimizing the Operating System for Virtual Machines. In *Proceedings*

of the 2014 USENIX Annual Technical Conference, USENIX ATC'14, pp. 61–72, Berkeley, CA, USA, 2014.

[12] Anil Madhavapeddy, Richard Mortier, Charalampos Rotsos, David Scott, Balraj Singh, Thomas Gazagnaire, Steven Smith, Steven Hand, and Jon Crowcroft. Unikernels: Library Operating Systems for the Cloud. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, pp. 461–472, New York, NY, USA, 2013.

[13] Joao Martins, Mohamed Ahmed, Costin Raiciu, Vladimir Olteanu, Michio Honda, Roberto Bifulco, and Felipe Huici. ClickOS and the Art of Network Function Virtualization. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pp. 459–473, Berkeley, CA, USA, 2014.

[14] Margaret Chiosi, Don Clarke, Peter Willis, Andy Reid, James Feger, Michael Bugenhagen, Waqar Khan, Michael Fargano, Dr. Chunfeng Cui, Dr. Hui Deng, Javier Benitez, Uwe Michel, Herbert Damker, Kenichi Ogaki, Tetsuro Matsuzaki, Masaki Fukui, Katsuhiro Shimano, Dominique Delisle, Quentin Loudier, Christos Kolias, Ivano Guardini, Elena Demaria, Roberto Minerva, Antonio Manzalini, Diengo López, Francisco Javier Ramón Salguero, Frank Ruhl, and Prodip Sen. Network functions virtualisation: An introduction, benefits, enablers, challenges and call for action. Technical report, SDN and OpenFlow World Congress, 2012.

[15] Fabrice Bellard. QEMU, a Fast and Portable Dynamic Translator. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '05, pp. 41–41, Berkeley, CA, USA, 2005.

[16] Poul-Henning Kamp and Robert NM Watson. Jails: Confining the omnipotent root. In *Proceedings of the 2nd International SANE Conference*, Maastricht, The Nertherlands, 2000.

[17] Dawson R. Engler, M. Frans Kaashoek, and James William O'Toole, Jr. Exokernel: An Operating System Architecture for Application-level Resource Management. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, SOSP '95, pp. 251–266, New York, NY, USA, 1995.

[18] 三宮 浩太, 光来 健一. 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. 一般社団法人 情報処理学会 研究報告 IPSJ SIG Techinical Report, Vol. 2015-OS-135, No. 1, pp. 1–9, 2015.