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

金津 穂, 田崎 創, 宇夫 陽次朗, 山田 浩史

mkanatsu@asg.cs.tuat.ac.jp,tazaki@iijlab.jp,yuo@iijlab.jp,hiroshiy@cc.tuat.ac.jp

Abstract

クラウド環境において，ライブラリ OS（LibOS）が再度注目を浴びている．LibOS とは OS の機能を有するライブラリであり，アプリケーションとリンクして動作する．クラウド環境では OS 上で単一のアプリケーションのみ稼動させることが多いため，汎用 OS を稼働させる場合比べて，メモリフットプリントやコードサイズの小さい LibOS が性能面で有利となる．また，これらの特徴は早い起動速度をもたらす可能性がある．起動速度はそのクラウドプラットフォームがもつ負荷分散などの *Elasticty* や障害復旧の速度に直結する．しかしながら，これまで提案されている LibOS を起動速度という尺度で比較した調査は著者らの知る限りは無い．本研究では，現行の LibOS の起動にかかる時間に着目し，その定量的な測定および比較を行なった．ハイパーバイザ型の LibOS を取り上げて比較を行なったところ，起動時間がその種類によって大幅に違うことがわかった．

Keywords: クラウド, ライブラリ OS, 計測, Cloud, Library OS, measurement

1. はじめに

クラウド環境のひとつに Infrastructure as a Service（IaaS）プラットフォームが存在する． IaaS プラットフォームでは，ユーザは自身のアプリケーションの負荷に応じて，アプリケーションをホストする仮想マシン（VM）の数をオンデマンドに増減させることができる．たとえば，負荷が増加した場合には自身の VM（インスタンスと呼ぶ）の起動数を増やし，負荷が減少すればインスタンス数を減少させることで，負荷の増減に追従可能である．実際に，Amazon EC2 [1], Google Compute Engine [2]，Microsoft Azure [3] といった IaaS プラットフォームが実運用されている．

ライブラリ OS（LibOS）とは OS の機能を有するライブラリであり，アプリケーションとリンクして動作する [4–14]．IaaS プラットフォームでアプリケーションを稼動させる場合，ひとつのアプリケーションや言語ランタイムを動作させる場合が一般的であり，汎用 OS の多くの機能が不要となる． たとえば，Web アプリケーションを実行したい場合，ネットワークの機能は必要となるが，ファイルシステムや，複数のアプリケーションに公平になるような複雑なスケジューリングシステムなどは，Web アプリケーションの動作そのものに必要になるわけではない． LibOS では，必要となる OS の機能のライブラリをリンクすればよい．つまり，最低限必要な機能だけをモジュールとして組込める．更に，Linux の VFS のような，あるサブシステムで複数のモジュールを切り替えて使うような仕組みも不要となるため，結果として汎用 OS より軽量となり，メモリフットプリントが小さくなり速度の面でも有利となる．また，構造も簡素になりやすいため，汎用 OS に比べて脆弱性が生じにくい．

LibOS はその軽量さから，早い起動速度を有する可能性がある．アプリケーションにとって必要なモジュールのみを持つことが可能なため，初期化の必要なコンポーネントは少なくなり，全体のメモリフットプリントも小さくなるため，読み込みにかかる時間も短くなるからである．起動速度は資源利用効率やサービスの可用性に直結する要素である．たとえば，起動速度が十分に早ければ，負荷が増加した際にそのタイミングで起動を開始してもサービスの品質は落とさない．もし速度が遅ければ，予め余計にインスタンスを起動する必要があるため，資源使用率を低くする．また，一度に起動できるインスタンス数が多い場合でも素早く起動が可能であれば，障害時において迅速なサービス復旧が可能となる．同時起動時において速度が遅ければ，サービスの復旧にそれだけ時間を要し，サービスの可用性が低下する．これまでクラウド環境を指向する LibOS が数多く提案されているものの，起動速度という面においてこれらを比較した調査や文献は，著者らの知る限り存在しない．

そこで本研究では LibOS の起動速度に関する定量

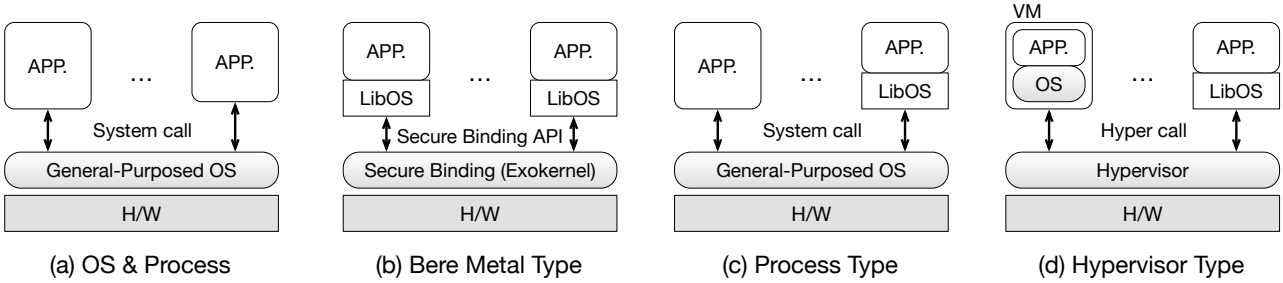

Figure 1: LibOS Types

的な比較をする．LibOS の起動にかかる時間を，クラウドで想定される複数の起動パターンで計測することにより，LibOS の構成法の違いによって起動性能が異なることを明かにし，これを分類する．

本稿では 4 つの環境について計測を行った．仮想マシンモニタ (VMM) として QEMU+KVM を用いて実験を行い，複雑な操作を行わずにそれぞれの環境について起動速度を計測した．その結果，以下のことがわかった．

- 軽量のインスタンス，LibOS 2 種類，Linux の 3 つの起動時間を比較したところ，それぞれ違いが生じた．
- それぞれの起動時間について，メモリの消費量との相関が見られた．
- LibOS の実装の違いについては起動時間から断定できるほどのデータが見られなかった．

2. LibOS

2.1. *LibOS* の構成

LibOS とは，OS の機能を有するライブラリである．Linux などの汎用 OS は，様々なアプリケーションの要求に答えるために，計算機資源を汎用的に抽象化し，仮想メモリやファイルシステムなど多くの機能を有する．また，OS レイヤを保護するために，アプリケーションはシステムコールを介して資源の確保や解放を行なう．一方，LibOS は単一アプリケーションとリンクして動作するため，アプリケーションに特化した形で作り込むことができるため，計算機資源の効率的な管理，また保持する機能を縮小することができる．また，OS の機能がライブラリとして実現されているため，システムコールなどのモード切り替えが不要となる．これらの特徴を有するため，汎用 OS に比べ，LibOS はメモリフットプリントが小さい，軽快に動作するといったメリットを享受することができる．

LibOS は動作する環境で以下の 3 つに大別する事が可能である．それぞれの概略図を Figure 1 に示す．

ベアメタル型 全アプリケーションが LibOS とリンクして動作することを仮定して，ハードウェアを安全に割り当てるソフトウェアレイヤを提供する．ExoKernel [4] やそれを土台とした Corey [15]，などがあり，Rumprun Kernel [6] もそのように振る舞う事ができる．LibOS を用意してそれとリンクしたアプリケーションしか動作しないものの，ハードウェアを管理するレイヤが小さいので効率性は高い．

プロセス型 汎用 OS 上で動作する LibOS を用意して，それとリンクさせてアプリケーションを稼動させる． Windows 上で Linux バイナリが動作する Windows Subsystem for Linux は Drawbridge を応用したものであり [16]， また，Linux 上で動作する Linux Kernel Library [7] が代表的な例である． LibOS はホスト OS のシステムコールを用いて作られるため，ベアメタル型と比べるとハードウェア管理層が通常の汎用 OS と同等なため効率面では劣る． 一方で，汎用 OS 上で LibOS が動作するため，通常のアプリケーションも同じプラットフォームで動作する．

ハイパーバイザ型 ハイパーバイザ上で動作する LibOS とアプリケーションとがリンクして動作する環境を提供する． LibOS はハイパーコールと呼ばれるハイパーバイザ上のインターフェースを用いて実現されている．たとえば OSv [11] や InlcudeOS [13] などがある．通常の仮想マシンと同様，LibOS とリンクしたアプリケーションはハイパーバイザ上で動作する．ハイパーバイザは汎用 OS のような抽象化を行わないため，プロセス型と比べると軽快に動作する．

2.2. ハイパーバイザベースの *LibOS*

本稿ではハイパーバイザ型の LibOS を対象としていく．ハイパーバイザベースの LibOS には OSv [11] や IncludeOS [13] などがある．OSvは複雑なスケジューリングやメモリ管理機構を廃し，メモリのレイアウトについてもカーネル空間とユーザ空間の分離

をとりやめ，効率化を果たしている．また，当初は既存 Java アプリケーションの動作に特化しており，メモリ管理機構は Java 言語ランタイムに割当てに特化させパフォーマンスを向上させている．現在はその機構を応用し，Java 言語アプリケーションのみならずさまざまな既存の Linux アプリケーションが容易に動作させる事が可能である．単一プロセスしか動作しないものの，マルチスレッド対応をし，スケジューラもスレッドの活用に特化しているのも特徴である．

IncludeOS [13] は全てスタティックリンクなライブラリとして構成された LibOS である [13]． 全て C++ 言語とその標準テンプレートライブラリ（STL）でのみ記述されており，可搬性にも優れている．C++ 言語にはゼロオーバーヘッド原則 [17] があり IncludeOS はそれを遵守しているため，プログラム中で実際に使用されていない機能については一切オーバーヘッドとならず，パフォーマンスを高めている．

3. 関連研究

既存の ライブラリ OS を多数起動した場合の起動速度計測を行った研究として，ClickOS [18] が挙げられる． 彼等の論文では，軽量化された VM がいかに早くユーザのリクエストに応じて起動できるかを示すために，連続的に起動させた時の VM そのものの初期化時間と，その VM で動作するアプリケーションの初期化時間とを分けて計測した．この結果により，ClickOS が Xen のボトルネックを検出した．

Williams 等による Unikernel Monitor [19] においてもゲスト OS が起動してから仮想コンソール，仮想ネットワークデバイスに出力がされるまでの時間を測定した． ゲスト OS が Unikernel のような特定の機能しか利用しないものの場合においてハイパーバイザの処理をどのようなオーバーヘッドがあるかを，最小構成のハイパーバイザー ukvm を実装し比較実験をした．

本研究では，単一のライブラリ OS についてではなく，複数のライブラリ OS と軽量 OS を比較することにより， ライブラリ OS のコア実装における差を起動にかかる時間という尺度で比較できることを明かにした点で，ハイパーバイザのオーバーヘッドを明らかにした両研究と異なる．

4. 起動にかかる時間の計測による分類

ハイパーバイザに特化してドライバを実装することで IaaS プラットフォームでの動作を前提に設計された LibOS は複数存在する [10–13]． このような LibOS は特定の用途に特化しており，使用する際には自身の目的に応じて適切に選択しなければならない．

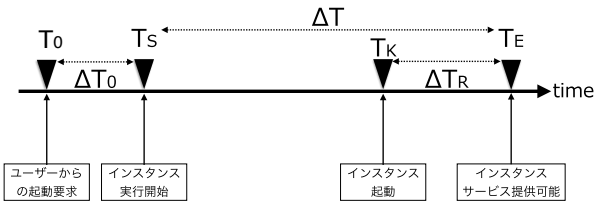

Figure 2: 計測におけるブートシーケンスと用語の定義

しかし現在，LibOS を分類を行なう一般的な方法は知られていない．そこで本研究では，特に LibOS の起動速度に着目して比較を行なう．

起動速度に着目する理由として， IaaS プラットフォームでは，インスタンスの起動速度がサービスの応答速度に直結するためである．もし起動速度の遅い LibOS でサービス品質を損なわずにシステムを運用しようとした場合，インスタンスをホットスタンバイさせなければならず待機インスタンスのリソースに無駄が発生する．また，障害発生時にも，異常終了したサービスが再起動する際，インスタンスの生成・起動の速度は再起動にかかる速度に直結するため，障害復帰にかかる時間に影響する．

LibOS は種類によって構成が異なり，同じハイパーバイザベースのものであっても起動にかかる時間は異なると推測される．本研究では，起動にかかる時間を，複数インスタンス起動した場合に計測を行い，比較することで LibOS の構成の違いによる特徴を明かにする．

これを計測することにより，LibOS を用いた場合の障害復旧性，および負荷の急激な増加を評価することができる．データセンタに停電等の障害が発生した場合，サービスの可用性を確保するために多くの VM が別のデータセンタで一斉起動することになるため，その性能を評価することになる．また，急激に負荷が増加した場合，複数 VM を一度に立ち上げるため，その性能を評価することにもなる．

4.1. 起動時間

OS における起動時間はどの時点をもって起動開始／起動終了とするのか，複数の場合が考えられる．本研究では起動の開始時刻，起動にかかる時間，起動の終了時刻をそれぞれ T_s，ΔT，T_e と置き，その定義を Figure 2 に示す時刻と時間とした．

T_0 計測対象全体の起動開始の時刻を示す．

T_S あるインスタンスの起動開始時刻を表す．計測対象のプログラムの，プロセスが起動する直前の時刻を起動開始とする．

T_E あるインスタンスの起動終了時刻を表す．計測対象のプログラムの標準出力から，起動完了を示すメッセージを計測プログラムが受け取った時刻を起動終了とする．

ΔT あるインスタンスの起動にかかった時間を表す．T_s と T_e の差分を起動にかかった時間とする．

T_K 計測対象のプログラムの，ホスト OS と VMM と QEMU の初期化にかかる時間を除いた起動開始時刻

ΔT_R 計測対象プログラムの，カーネルの起動後，サービスの起動まで終了した時刻

今回の計測では，OS の内部構造を調べ，手を加える必要がある T_K ならびに ΔT_R の計測は行なっていない．ΔT_R について， LibOS は OS がアプリケーションとリンクしているためほぼ 0 になると考えられるが，Linux のような一般的な OS ではカーネルの起動後に init システムやサービスの起動があるため，その場合は 0 にはならない．

5. 評価

本研究では，現状利用が可能な LibOS の実装を，クラウド環境での実運用で想定される大量の VM をシーケンシャルに起動させた場合を想定し，起動に要する時間を測定する．同様の計測を LibOS 以外のいくかの OS でも行い，比較する事により， LibOS の実装の違いによる特徴を明かにする．

5.1. 実験環境

Table 1 に実験に用いた環境を示す．今回は VMM として， Google Compute Engine でも用いられている KVM を用いた．

QEMU には次のオプションを与えた．

```
-nographic -enable-kvm -cpu host,
+x2apic -m 50 -smp 1
```

また， *minkernel* と Rumprun については，QEMU のカーネルダイレクトロードを用いた．

5.2. 計測方法

QEMU+KVM [20, 21] で構成された LibOS インスタンスの起動にかかる時間を，C/C++ のプログラムを用いて計測した．

計測環境のホスト OS およびエミュレータ，VMM への変更を行わずに計測を行うため，次のような構成にした

- 計測プログラムは，複数スレッドを生成し，各スレッドが `fork()`，`execp()` を行う事によってインスタンスを起動する．
- 生成されたスレッドは，起動したインスタンスの標準出力を監視し，起動メッセージの出力を観測する事によって起動の終了時刻とし，これを記録する．

この計測プログラムについて，疑似コードを 1 に示す．

Algorithm 1 計測に用いたプログラム

$T_0 \leftarrow$ 現時刻
500 μsec 毎のシステム全体のメモリ消費量記録開始
for 計測対象の起動するインスタンス数 N **do**
    $T_s \leftarrow$ 現時刻
    計測対象プログラムの起動
    計測対象プログラムの標準出力に何か出力されるまで待機
    $T_e \leftarrow$ 現時刻
end for
全インスタンスの起動終了待機
500 μsec 毎のシステム全体のメモリ消費量記録終了

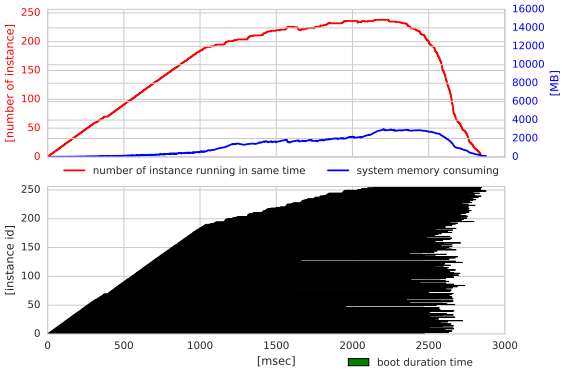

Figure 3: minkernel を 256 台起動した際の起動時間，同時起動数，消費メモリ量

5.3. ベースラインの評価

プラットフォームの性能を計測 (本計測環境のベースライン) するために，軽量のインスタンスを用意して起動時間を計測した．これらのデータは，起動時間のボトルネックが LibOS にあるか否かを判断する材料として用いる．軽量インスタンスとして，x86_64 アセンブリ言語で記述した *minkernel* を用いた．

minkernel は QEMU 上での動作を前提としており，起動時に以下の 3 処理のみを行う．

- スタック初期化
- I/O ポート（0x3F8）に対する値の出力
- halt

minkernel のブートは QEMU のカーネルダイレクトブートを用いたため，起動にかかるオーバーヘッドは QEMU 上で最小である．

このプログラムは QEMU+KVM 上で “Hello” という文字列を出力するのみの，14 命令のプログラムであり，つまりこのプログラムの起動にかかる時間は，QEMU+KVM そのものの初期化やブートストラップ

Table 1: 実験環境

ベンダー	QCT D51U
CPU	Intel®Xeon®CPU E5-2640 v3 @ 2.60GHz （8 コア 2way）
主記憶	DD4 256GB
ストレージ	Intel S3610 SSD （128GB）
ホスト OS	CentOS 7 （Linux kernel 3.10.0-327.el7.x86_64）
VMM	QEMU（version 2.0.0） + KVM

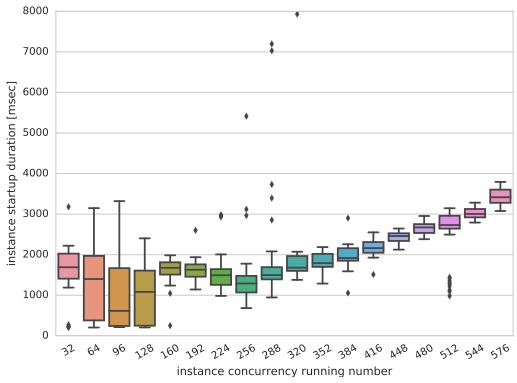

Figure 4: minkernel の同時起動数 N における起動にかかる時間についての最小／最大／中央値の様子

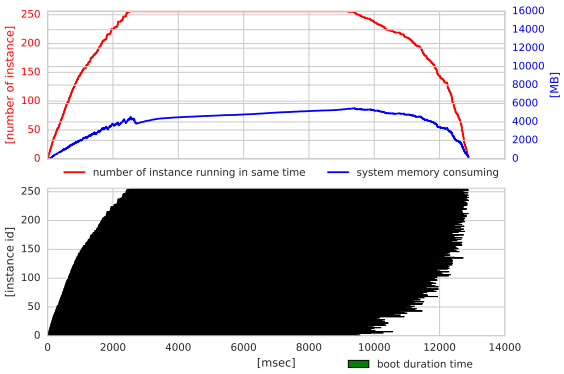

Figure 5: OSᵛを 256 台起動した際の起動時間，同時起動数，消費メモリ量

の起動といった動作にかかる時間に近似できる．この環境での測定値は，他の OS での測定値を考える際のベースラインとして機能する．

minkernel について，256 台のインスタンスを 5 ミリ秒ごとにに起動し，計測した結果を Figure 3 に示す．

Figure 3 において，青線はその瞬間でのシステムが使用しているメモリ総量，赤線は同時 OS 起動インスタンス数，緑線の左側は起動開始指示を出した時刻，緑線の右側は起動終了時刻を示している．以降の Figure 5 においても，同様の凡例を使用する．

これらの図から以下のことが読み取れる．

- *minkernel* の実行にかかる時間は極めて小さく，全てのインスタンスの起動が 3 秒以内に終了している
- *minkernel* の実行に必要なメモリ量は少ないため，同時実行数が増えた場合でも計測環境のメモリ使用量に対する影響がない

このように *minkernel* を用いることにより，本計測環境（Table 1）のベースライン値を次のように推定できる．

- QEMU + KVM を用いた場合のインスタンスの起動にかかる時間は最短で 500 ミリ秒前後
- QEMU + KVM のインスタンス 1 つあたり最低でも 10 MB 以上のメモリを消費する

また，64，128，256，512，1024，1280 台のそれぞれの場合で間隔を開けずに起動した場合，同時起動数が 0–32，33–64，… 545–576 台の時の起動にかかった時間の分布を，Figure 4 に示す．

5.4. *LibOS*

ベースラインの評価を踏まえ，現行の LibOS の起動時間を測定する．クラウド環境を指向する，ハイパーバイザベースの LibOS は複数存在するが，今回は計測対象として OSᵛと IncludeOS を選択した．これらを選択した理由は次の通りである．

- 既存の汎用 OS のコードを用いる Rump kernel [6] と異なり，一から設計されている
- クラウド環境のハイパーバイザ上で高速に動作させる事を期待して設計された．
- 計測プログラムが前提とする QEMU + KVM の環境で動作する

OSᵛと IncludeOS はどちらも C++ によって記述され，単一のメモリ空間しか持たない．また，両者ともにユーザプログラムのために標準 C ライブラリ

Table 2: OSᵛと IncldeOS における違い

	OSᵛ	IncludeOS
バイナリ方式	ダイナミックリンク	スタティックリンク
ファイルシステム	ZFS	RamFS
使用ライブラリ	STL + Boost	STL
提供する標準 C ライブラリ	musl libc [22]	newlib [23]
OS の中のユーティリティ	含む	含まない
実行アプリの違いによる，アプリ起動までの実行経路の違い	ある	ない

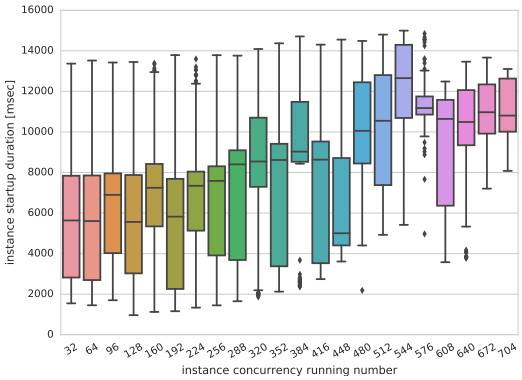

Figure 6: OSᵛの同時起動数 N における起動にかかる時間についての最小／最大／中央値の様子

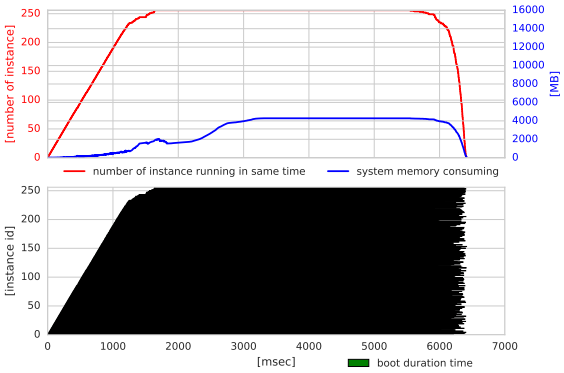

Figure 7: IncludeOS を 256 台起動した際の起動時間，同時起動数，消費メモリ量

を提供する．しかし，ダイナミックリンクを活用し，ZFS のようなファイルシステムを持つ OSᵛと異なり，IncludeOS の場合は RamFS を使用しており，また，バイナリを全てスタティックリンクにする（Table 2）．

OSᵛについて，256 台のインスタンスを 5 ミリ秒ごとにに起動し，計測した結果を Figure 5 に，Include について，256 台のインスタンスを 5 ミリ秒ごとにに起動し，計測した結果を Figure 7 に示す．これらの図から以下の事実がわかる

- *minkernel* と比較しインスタンス起動にかかる時間が 6–10 秒と大きい

また，64，128，256，512，1024，1280 台のそれぞれの場合で間隔を開けずに起動した場合，同時起動数が 0–32，33–64，… 673–704 台の時の起動にかかった時間の分布を，それぞれ Figure 6 と Figure 8 に示す．

Figure 8 に比べて，Figure 6 のほうが起動にかかる時間のバラつきが大きく，同時起動数に対しての変化があまり見られない．

5.5. 既存の *OS* を利用する *LibOS*

Rumprun について，256 台のインスタンスを 5 ミリ秒ごとにに起動し，計測した結果を Figure 11 に示す．

Rumprun とは，NetBSD のソースコードを使用する LibOS である Rump Kernel [6] の派生である [12]．オリジナルの Rump Kernel は既存の汎用 OS のアプリケーションの一つとして動作するが，Rumprun は追加でブートストラップと仮想化環境用ドライバを実装することによって，VMM 上で直接動作する LibOS として機能する．

この実験では，Rump Kernel 用ツールキットを用いてコンパイルしたソースコードを，`rumprun-bake` コマンドを用いる事によって，Rumprun のイメージにした．rumprun-bake では `hw_generic` コンフィグを使用した．

また，64，128，256，512，1024，1280 台のそれぞれの場合で間隔を開けずに起動した場合，同時起動数が 0–32，33–64，… 673–704 台の時の起動にかかった時間の分布を，Figure 10 に示す．

5.6. *Linux*

Alpine Linux は Linux ではあるものの，設定によってカーネルサイズが小さくされており，ユーザーランドにも musl libc と BusyBox [24] を用いることにより軽量化を行ったディストリビューションである．

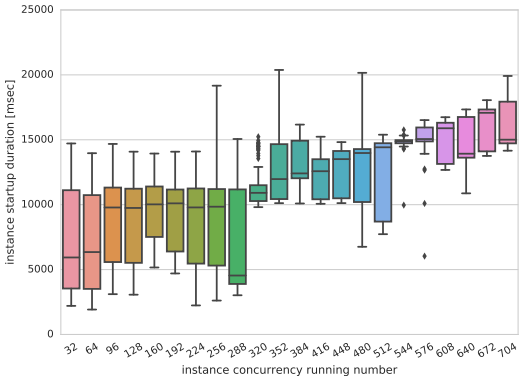

Figure 8: IncludeOS の同時起動数 N における起動にかかる時間についての最小／最大／中央値の様子

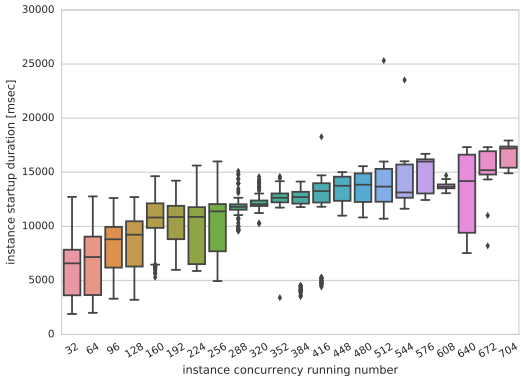

Figure 10: Rumprun の同時起動数 N における起動にかかる時間についての最小／最大／中央値の様子

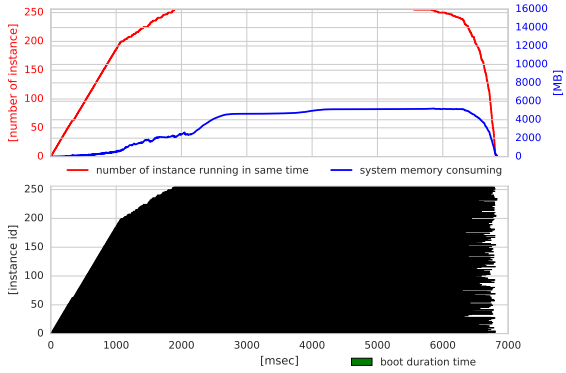

Figure 9: Rumprun を 256 台起動した際の起動時間，同時起動数，消費メモリ量

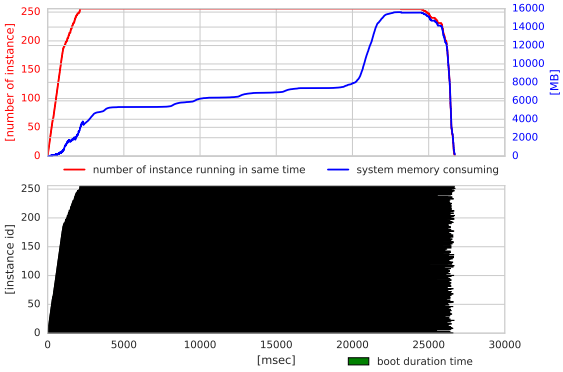

Figure 11: Linux を 256 台起動した際の起動時間，同時起動数，消費メモリ量

例えば，Docker のベースイメージとして 5 MB 程度のイメージが配布されている [25].

Alpine Linux について，256 台のインスタンスを 5 ミリ秒ごとにに起動し，計測した結果を Figure 11 に示す．Linux の場合 multi user mode と single user mode の 2 つの起動の種類があるが，今回は multi user mode の場合で計測を行った．

グラフより，次の事がわかる

- Linux は顕著に遅く，また，メモリの消費がはげしい．

また，64，128，256，512 台のそれぞれの場合で間隔を開けずに起動した場合，同時起動数が 0–16，17–32，… 176–192 台の時の起動にかかった時間の分布を，Figure 12 に示す．

5.7. 計測結果における考察

どの OS の場合でも，起動を開始したインスタンスの順番が若いものは起動にかかる時間が多く，加えてメモリの消費量と同時起動数も増加していき，ある時点でメモリ消費量と同時起動数が減りはじめ，起動にかかる時間も一定のものになっていくという状況は同じだった．しかし，*minkernel*，OSv と IncludeOS の LibOS，Alpine Linux と，この 3 つで大きく起動にかかる時間が異なっている事が数値として明らかになった．

それぞれのメモリ消費の最大値について，*minkernel* が 3020 MB，OSv が 5486 MB，IncludeOS が 4281 MB，Linux が 15620 MB Rumprun が 5220 MB となっており，起動にかかる時間が長いものほどメモリの消費量も大きくなっている． OS の起動にかかる時間のボトルネックとして，一般に外部との I/O とデバイスの初期化が挙げられるが，ハイパーバイザベースの LibOS の場合は特定の VMM での 動作に限定し設計することで，外部との I/O 及びデバイス初期化がほぼ存在しない．そのため，メモリの消費量と起動処理の時間が比例する傾向が高い． また，

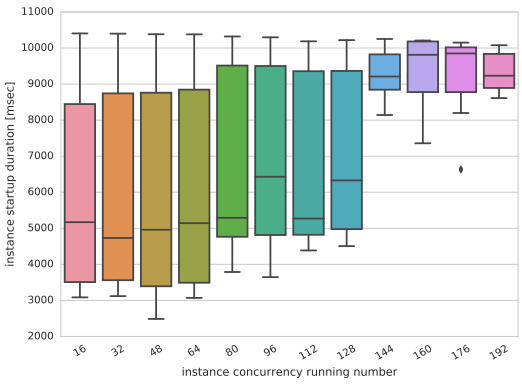

Figure 12: Linux の同時起動数 N における起動にかかる時間についての最小／最大／中央値の様子

Linux についても，今回は仮想環境上で KVM を用いて動作させたため，同様の傾向を示したと考えられる．

そして，起動にかかる時間について，同時起動数に対しての分布を見てみると，起動にかかる時間そのものは似かよっている IncludeOS や Rumprun，OSv についてもグラフの形が大きく異なっていることがわかった．

6. まとめ

本稿では，LibOS を想定用途別の起動方法で起動し，その起動時間を計測することで，LibOS の分類を行った．

仮想環境上で複数の OS の起動にかかる時間を観測することにより，*minkernel* と LibOS と Linux については，それぞれ起動の時間に大きな差が見られたが，LibOS の中でも OSv と IncludeOS の差については，現在の計測結果からコードパスの違いを断定する事ができなかった．

今後の課題として，システムコールの発行量やかかる時間などの実行プロファイルを，それぞれの OS において調査することで，各 LibOS の実装の差がその結果から分類できないかを調査する．加えて，その結果を本研究の手法にフィードバックし，簡易な時間計測のみでより精度の高い分類を行いたい．

7. 参考文献

[1] Amazon Elastic Compute Cloud, Accessed 2016-08-12.
URL https://aws.amazon.com/ec2

[2] Google Compute Engine, Accessed 2016-08-12.
URL https://cloud.google.com/compute

[3] Microsoft Azure, Accessed 2016-08-12.
URL https://azure.microsoft.com

[4] D. R. Engler, M. F. Kaaspoek, J. O'Toole Jr, ExoKernel: An Operating System Architecture for Application-Level Resource Management, in: 15th ACM Symposium on Operating Systems Principles (SOSP '95), ACM, Copper Mountain, CO, USA, 1995, pp. 251–266. doi:10.1145/224056.224076.
URL https://pdos.csail.mit.edu/6.828/2008/readings/engler95exokernel.pdf

[5] G. Ammons, J. Appavoo, M. Butrico, D. D. Silva, D. Grove, K. Kawachiya, O. Krieger, B. Rosenburg, E. V. Hensbergen, R. W. Wisniewski, Libra: A Library Operating System for a JVM in a Virtualized Execution Environment, in: 3rd ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '07), ACM, San Diego, CA, USA, 2007, pp. 44–54. doi:10.1145/1254810.1254817.
URL http://portal.acm.org/citation.cfm?id=1254810.1254817

[6] A. Kantee, Flexible Operating System Internals: The Design and Implementation of the Anykernel and Rump Kernels, Phd thesis, Aalto University (2012).
URL http://urn.fi/URN:ISBN:978-952-60-4917-5

[7] O. Purdila, L. A. Grijincu, N. Tapus, LKL: The Linux Kernel Library, in: 9th Roedunet International Conference (RoEduNet '10), Sibiu, Romania, 2010, pp. 328–333.
URL http://ieeexplore.ieee.org/document/5541547/

[8] D. E. Porter, S. Boyd-Wickizer, J. Howell, R. Olinsky, G. C. Hunt, Rethinking the Library OS from the Top Down, in: 16th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '11), ACM, Newport Beach, CA, USA, 2011, pp. 291–304.
URL http://research.microsoft.com/apps/pubs/default.aspx?id=141071

[9] A. Baumann, D. Lee, P. Fonseca, L. Glendenning, J. R. Lorch, B. Bond, R. Olinsky, G. C. Hunt, Composing OS Extensions Safely and Efficiently with Bascule, in: 8th ACM SIGOPS/EuroSys European Conference on Computer Systems (EuroSys '13), ACM, Prague, Czech Republic, 2013, pp. 239–252. doi:10.1145/2465351.2465375.
URL http://dl.acm.org/citation.cfm?doid=2465351.2465375

[10] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, J. Crowcroft, Unikernels: Library Operating Systems for the Cloud, in: 18th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '13), ACM, Houston, TX, USA, 2013, pp. 461–472. doi:10.1145/2451116.2451167.
URL http://dl.acm.org/citation.cfm?doid=2451116.2451167

[11] A. Kivity, D. Laor, G. Costa, P. Enberg, N. Har'El, D. Marti, V. Zolotarov, OSv—Optimizing the Operating System for Virtual Machines, in: 2014 USENIX Annual Technical Conference (ATC '14), USENIX Assocation, Philadelphia, CA, USA, 2014, pp. 61–72.
URL https://www.usenix.org/system/files/conference/atc14/atc14-paper-kivity.pdf

[12] A. Kantee, J. Cormack, Rump Kernels: No OS? No Problem!, ;login: 39 (5) (2014) 11–17. doi:10.1007/BF02896304.
URL http://rumpkernel.org/misc/usenix-login-2014/login_1410_03_kantee.pdf

[13] A. Bratterud, A.-A. Walla, H. Haugerud, P. E. Engelstad, K. Begnum, IncludeOS: A Minimal, Resource Efficient Unikernel for Cloud Services, in: 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom 2015), IEEE, Vancouver, BC, Canada, 2015, pp. 250–257. doi:10.1109/CloudCom.2015.89.
URL http://dx.doi.org/10.1109/CloudCom.2015.89

[14] S. Lankes, S. Pickartz, J. Breitbart, HermitCore—A Unikernel for Extreme Scale Computing, in: 6th International Workshop

on Runtime and Operating Systems for Supercomputers (ROSS '16), ACM, Kyoto, Japan, 2016, pp. 4:1–4:8. doi:10.1145/2931088.2931093.
URL http://doi.acm.org/10.1145/2931088.2931093

[15] S. Boyd-Wickizer, H. Chen, R. Chen, Y. Mao, F. Kaashoek, R. Morris, A. Pesterev, L. Stein, M. Wu, Y. Dai, Y. Zhang, Z. Zhang, Corey : An Operating System for Many Cores, in: 8th USENIX Conference on Operating Systems Design and Implementation (OSDI '08), USENIX Assocation, San Diego, CA, USA, 2008, pp. 43–57.
URL http://www.usenix.org/events/osdi08/tech/full{_}papers/boyd-wickizer/boyd-wickizer{_}html/

[16] J. Howell, B. Parno, J. R. Douceur, How to Run POSIX Apps in a Minimal Picoprocess, in: 2013 USENIX Annual Technical Conference (ATC '13), USENIX Assocation, San Jose, CA, USA, 2013, pp. 321–332.
URL https://www.usenix.org/conference/atc13/technical-sessions/papers/howell

[17] B. Stroustrup, Evolving a language in and for the real world: C++ 1991-2006, in: 3rd ACM SIGPLAN Conference on History of Programming Languages (HOPL-III), ACM, San Diego, CA, USA, 2006, pp. 4:1–4:59. doi:10.1145/1238844.1238848.
URL http://www.stroustrup.com/hopl-almost-final.pdf

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

[19] D. Williams, R. Koller, Unikernel Monitors: Extending Minimalism Outside of the Box, in: 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16), USENIX Assocation, Denver, CO, USA, 2016.
URL https://www.usenix.org/conference/hotcloud16/workshop-program/presentation/williams

[20] F. Bellard, QEMU, a Fast and Portable Dynamic Translator, in: Annual Conference on USENIX Annual Technical Conference (ATEC '05), USENIX Assocation, Anaheim, CA, USA, 2005, pp. 41–46.
URL https://www.usenix.org/legacy/publications/library/proceedings/usenix05/tech/freenix/bellard.html

[21] A. Kivity, Y. Kamay, D. Laor, U. Lublin, A. Liguori, kvm: the Linux Virtual Machine Monitor, in: Ottawa Linux Symposium, The Linux Foundation, Ottawa, ON, Canada, 2007, pp. 225–230. doi:10.1186/gb-2008-9-1-r8.
URL https://www.kernel.org/doc/mirror/ols2007v1.pdf#page=225

[22] musl libc, Accessed 2016-08-19.
URL https://www.musl-libc.org

[23] The Newlib Homepage, Accessed 2016-08-20.
URL http://www.sourceware.org/newlib/

[24] BusyBox, Accessed 2016-08-19.
URL https://busybox.net

[25] Docker Hub - alpine, Accessed 2016-08-20.
URL https://hub.docker.com/r/library/alpine/