Vol.2021-OS-151 No.8
2021/3/2

Unikernelを用いたコンテナのためのハイパーバイザによる軽量高セキュアな実行基盤の検討

肥沼 健[1]　並木 美太郎[1]

概要：コンテナ型仮想化によってアプリケーションの開発や運用の利便化が進み広く利用されるようになった．ホスト OS のカーネルの機能によってファイルシステムやプロセスを分離した空間でアプリケーションを動かすことができるこの仮想化技術はその扱いやすさ，起動時間が速いことなどからクラウド上でも利用されている．しかしクラウド上で利用する場合にホスト OS のカーネルを共有することからセキュリティの面でコンテナ単位でマルチテナントにすることが難しく，ハードウェアの仮想化支援を利用して仮想マシン (VM) を起動してその上でコンテナを構築することが多い．こうしたことに対してライブラリ OS の利用あるいは Unikernel としてアプリケーションを動作させるコンテナの構築手法が提案されているが，巨大な汎用 OS や仮想化基盤によるものであり基盤自体の数々の脆弱性が報告されている．本研究の目的は Unikernel を利用したコンテナの構築を脆弱性の発生が少ないとされる軽量なハイパーバイザを用いてセキュアに動作させることができる実行基盤を提案することである．現状 Unikernel ならびにハイパーバイザへの少量の追加と修正でソケットプログラミングによるアプリケーションの実行の確認に成功している．

キーワード：仮想化基盤，ハイパーバイザ，Unikernel，コンテナ

A Study on a Lightweight Secure Platform for Container with Unikernel on Hypervisor

1. はじめに

コンテナ型の仮想化によりアプリケーションの開発やデプロイが容易に行えるようになった．ホスト OS のカーネルを共有することで仮想マシンを構築してアプリケーションを起動することよりも軽量で速い起動が可能である．クラウド環境上でのコンテナの起動は，分離性を得るために仮想マシンを起動してその上で複数のコンテナを起動させることが一般的である．

コンテナはホスト OS のカーネルを共有するためホストの権限が不正に獲得され他のコンテナへ攻撃できる可能性があるためセキュリティ対策としてクラウド上ではテナントごとに仮想マシンを起動させその上でコンテナを構築している．セキュリティを確保するために，コンテナを構築するための余裕を持ったリソースで仮想マシンを起動させることが必要であり，さらにはその仮想マシンの管理をしなくてはならない問題が発生する．

このようなことに対して動作させる目的のアプリケーションとライブラリ OS からなる Unikernel を利用して動作させる方法がある．Unikernel の軽量で起動時間の速いといった特徴を利用して仮想マシンとして起動することで，従来の仮想マシンをリソースの割り当てに余裕を持った起動をしてからその上で複数のアプリケーションを動作させることよりもアプリケーションの単位に近い形で起動することができる．しかし目的のアプリケーションを持った Unikernel を動作させる，ということに対して必要以上に多くの機能を持っている仮想化基盤で動作しているため仮想化基盤に脆弱性がある場合には問題になってしまう．巨大な汎用 OS や仮想化基盤ソフトウェアではアプリケーションをセキュアに動かしたいという目的に対して，必要のない異なるところで脆弱性が発生している場合がある．

本研究はこうした巨大な汎用 OS や仮想化基盤ソフトウェアによらず，必要最小限な構成とすることでセキュア

[1] 東京農工大学

ⓒ 2021 Information Processing Society of Japan

な実行基盤を実現する手法を提案する．

2. 従来手法と課題

2.1 仮想マシンによる分離

仮想マシン上でのアプリケーションの動作ではメモリやCPU のリソース設定をしながら仮想マシンをまず起動することで分離された状態を得る．その仮想マシン内でアプリケーションや必要な設定を展開するといったことを行う．つまり幅広い目的に合うような仮想マシンを起動させることができる仮想化基盤を利用してアプリケーションを動作させる．このことは仮想マシンからしたら幅広いデバイスの利用や多様なアプリケーションを動作させることできる一方で，アプリケーション視点では仮想化基盤に不要なデバイスや機能が存在することになる．

2.2 Unikernel を利用した軽量な仮想マシン

Unikernel は目的のアプリケーションに対して必要なライブラリ OS を含めてビルドすることで単一のアドレス空間で動作する [1]．メモリフットプリントが小さく起動が速く，必要なコードしか含まれていないため汎用 OS よりも攻撃対象領域が狭いためセキュアであるといった特徴がある．Unikernel を利用することで軽量な仮想マシンとして動作させることができ，分離性を得ることができる．しかし Unikernel を動作させる場合には従来の仮想マシンを構築するような仮想化基盤上で動作させるため，これまでの仮想マシンとは違った特徴を持つ Unikernel にとっては高速で多くのインスタンスを起動したいといったことやリソースの効率面で不十分であるといった問題も指摘され研究されている [2]．

2.3 従来手法

Unikernel を利用した起動には，Xen などのハイパーバイザや Linux 上の QEMU と KVM を用いることが多いが，仮想化基盤自体がが必要最小限なインターフェースだけ備えることで単純化し， Unikernel にとって最適になるような手法が提案されている [3]．文献の手法ではアプリケーションにとっては不要であったデバイスのエミュレーションに関する QEMU の脆弱性をとりあげている．

またハイパーバイザである Xen のカーネルを拡張した基盤上で Linux を Xen の準仮想化を用いたマルチプロセスを可能とするライブラリ OS へと変貌させ VM として起動させこれをコンテナとする手法がある [4]．いずれも目的のアプリケーションの動作にとって速い起動とマルチテナントに対する分離性を得ることができる．しかしそれらを実現する仮想化基盤が Linux などの巨大な汎用 OS やハイパーバイザの上で成り立っている．巨大であるがゆえに脆弱性が発生する可能性が高い．

Unikernel を Linux などの汎用 OS 上の 1 プロセスとして動作させる手法も提案されている [5]．アプリケーションに対して必要なライブラリ OS を組み合わせ提案されているプラットフォームに依存するバインディングを含めてビルドすることで動作させる手法である．システムコールの発行制限をする seccomp を利用した手法であるが，Linux などの巨大な汎用 OS の上で成り立っているため，仮想化基盤自体のセキュリティに関して課題である．

以上のようなことから仮想化基盤自体の脆弱性が抑えられている状態が必要である．

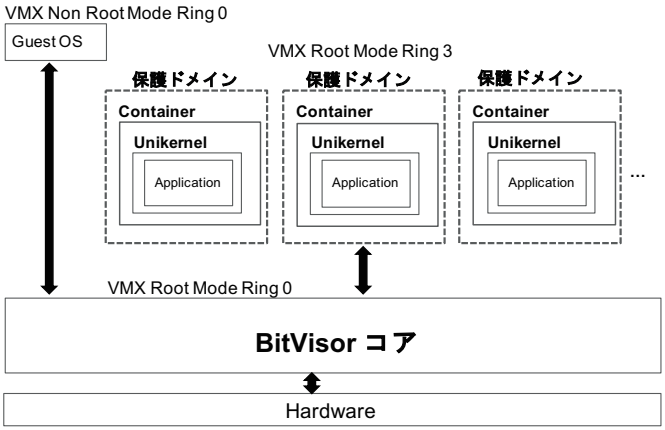

図 1 提案手法の全体構成図

2.4 目標

本研究は Linux などの巨大な汎用 OS やハイパーバイザを利用しなくても実現できるということを目標とする．巨大な汎用 OS や仮想化ソフトウェアによらない小規模なソフトウェアで Unikernel を利用したコンテナの実行基盤を提案する．

3. 提案

本研究の提案は BitVisor の Root Mode の Ring3 の実行である保護ドメインを利用して Unikernel を動作させることでコンテナを構築する手法である．扱うデバイスをネットワークデバイスとブロックデバイスと限定して BitVisor が専用のインターフェースを提供し，Unikernel がこのインターフェースを利用して保護ドメインで動作する．本研究ではこれをコンテナとし，複数のコンテナが動作する．提案手法の全体構成図を示すと図 1 のようになる．図中の VMX とは CPU による仮想化を支援する機構である Intel VT-x による機能を利用していることを指す．

ハードウェアの上で BitVisor を起動したのち BitVisor の通常の起動のとおり，ゲスト OS は起動される．つまり提案手法によってゲスト OS が利用できない状態とはならない．ゲスト OS は Non Root Mode の Ring 0 で動作するが，保護ドメインが動作するコンテナは Root Mode の Ring 3 であり，BitVisor は Root Mode の Ring 0 の動作

である．近年の仮想化は CPU の仮想化支援機構を利用して Non Root Mode でゲスト OS を動作させることで I/O などのセンシティブな命令をトラップすることで安全処理へと制御をすることが多いが，提案手法ではこの方法ではなく，Ring 3 での動作であることが特徴である．

3.1 本研究の特徴

高度な新しいセキュリティ対策を提案して実装するということをせず，コンテナを動作させる基盤としてセキュリティが高いと言えるソフトウェア上で必要最低限な実装をすることでコンテナを動作させるといったことが本研究の特徴である．利用するソフトウェアもセキュリティが高いことを維持するために大幅な変更や追加を行わず必要最小限とする．

本研究の特徴をまとめると以下のようになる．

(1) コンテナを軽量なソフトウェア上で動作させること
(2) 大規模な拡張や修正を行わず，必要最低限とすること
  (a) 物理デバイスドライバを新たに実装せず，既存のものを使うようにすること
(3) 実行基盤として扱えるデバイスはネットワークとブロックのみでよい

こうすることで実行基盤はソースコードの修正や追加を最小限におさえることができ，大幅に変更することなくセキュアな環境でコンテナを実行するということが実現できる．また，巨大なソフトウェア基盤とは対照的に小規模で少ないデバイスという制限でコンテナの実行基盤を実現できる．

3.2 ハイパーバイザ

小規模なソフトウェア基盤としてハイパーバイザである BitViosr を利用する．セキュリティに特化したハイパーバイザであり，それを実現しているコードの規模が少ないことも特徴である [6]．機能拡張を目的とした保護ドメインを利用してコンテナを構築することで，ハイパーバイザのコアの機能を壊すことなくコンテナを動かすことができる．保護ドメインはメモリ空間が分離していること，コアの機能とはメッセージインターフェースを介してだけやりとりできることなどが特徴でもある．また BitVisor のコア機能はもともと機能が少なくシステムコールも少ない，結果的に巨大な汎用 OS や他のハイパーバイザとは対照的に小規模であり，提案手法の特徴であるネットワークデバイスとブロックデバイスのドライバを備えていることや特定のデバイスに対してモジュールを割り当てられる仕組みを利用できることからも BitVisor を選択する理由である．

3.3 コンテナ

保護ドメインで動作する Unikernel を本研究でのコンテナとする．Unikernel は汎用 OS とアプリケーションの関係と違って CPU の動作モードを複数必要としない単一のアドレス空間で動作するということから保護ドメインで動作させることができる．コンテナからは BitVisor に対してハイパーバイザコールのみでやりとりし，Unikernel は提案手法の基盤が提供するインターフェースを利用するドライバを実装することでネットワークとブロックデバイスが利用できる．また本研究で利用する Unikernel には研究分野でも実績のある IncludeOS を利用する [7].

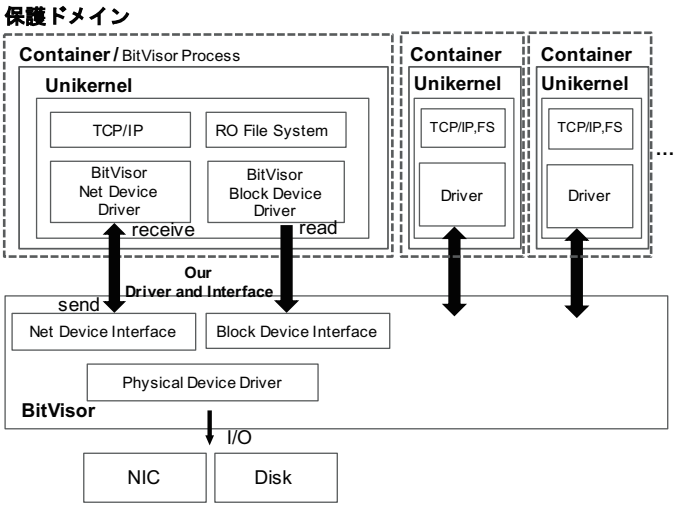

図 2 全体設計

4. 設計と実装

4.1 全体設計

本研究では BitVisor 上の保護ドメインから利用できるインターフェースを定義して，Unikernel ではそのインターフェースを利用するドライバを実装する．アプリケーションとライブラリ OS が一体となったその Unikernel を保護ドメインで複数動作させる．全体設計を図 2 に示す．

4.2 詳細設計

4.2.1 複数のコンテナ

全体設計で示したように複数のコンテナが動作する，これらのコンテナは互いにデータを共有することをしない．例えば，あるコンテナと別のコンテナの間でファイルを共有するといったことはしない．コンテナ間で同一のデータに対して読み書きができる利便性よりも，分離することでセキュアであることを重視する設計とする．

4.2.2 実行基盤のインターフェース

ハイパーバイザコールを定義してコンテナへ提供する．このハイパーバイザコールは BitVisor にすでに存在するゲスト OS から呼び出されるものとは別である．インターフェースはネットワークデバイス，ブロックデバイスのインターフェースだけであり，コンテナ側からは仮想デバイスとして見ることになる．そして BitVisor では BitVisor が制御するデバイスである NIC，ディスクを扱い，実装さ

表 1　BitVisor ハイパーバイザコール

ハイパーバイザコール	機能
bv_yield	BitVisor へ処理を明け渡す
bv_net_send	BitVisor へパケットを送る
bv_net_receive	BitVisor からパケットを受け取る
bv_block_write	BitVisor へブロックデータを書き込む
bv_block_read	BitVisor からブロックデータを読み込む
bv_get_time	BitVisor から時間を取得する

表 2　Unikernel の BitVisor 向けデバイスドライバの機能

名称	デバイス	機能
create_packet	ネットワーク	パケットを生成する
transmit	〃	パケットを送る
receive_packet	〃	パケットを受け取る
block_size	ブロック	ブロックの大きさを返す
write	〃	ブロックデータを書き込む
read	〃	ブロックデータを読み込む

れているデバイスドライバを使うことになる．

利用する BitVisor には 11 のシステムコールが実装されている．Unikernel からは実質的には保護ドメインからのシステムコールと同じであり，実装されているシステムコールに倣った設計とすることでセキュアであることを達成する．この実装されている 11 のシステムコールに加えて BitVisor ハイパーバイザコールとして 6 つ追加する．機能を表 1 に示す．

4.3　Unikernel の BitVisor 向けデバイスドライバ

本研究で利用する Unikernel には BitVisor をプラットフォームとするドライバが必要になる．ドライバに必要な機能を表 2 に示す．Unikernel はビルド時にこのドライバを含めることで BitVisor をプラットフォームとして動作する設計である．

4.4　実装

本研究ではネットワークデバイスが利用できることを中心に実装した．コンテナで動作するアプリケーションはネットワークの機能が使えるだけで Web サーバの動作が可能であることからネットワーク機能が使えることを優先として実現することで評価を行う方針とした．

4.4.1　技術的課題

保護ドメインで Unikerne を動作させるにあたりいくつかの技術的な課題が発生した．一つは保護ドメインとして追加する拡張機能は単体の処理を行うものであり，あるメッセージに対して登録しておいた処理を行い終了するといった構造になっている．したがって保護ドメインとして追加する各機能には処理が呼び出されたら終了することが期待されている．BitVisor はプリエンプティブな CPU のコンテキストではないため，イベントを待ち続けるプログラムの場合，CPU 時間を占有してしまう．そのため本研究では二つのアプローチを検討した．一つ目は CPU のコンテキストをプリエンプティブになるようにコアの機能を修正すること，二つ目は保護ドメインで動作するプログラム側で自ら処理を明け渡すようにすること．本研究ではコアの機能の修正を積極的に行うアプローチを取っていないこと，また高度なコンテキストの管理やスケジュールについては本研究にとって重要性が低く，保護ドメインでの Unikernel の動作を示すことが優先であることから後者の処理を明け渡す方式を選択して実装した．

二つ目の課題は割り込みである．BitVisor は割り込みは禁止された状態で動作している．したがって保護ドメインで動作する Unikernel が割り込みによって処理を行うことが難しい．設定の変更やコアの機能への修正による検討をしたが，最終的に BitVisor 自体が特定のデバイスを占有する機能を利用することにした．こうすることで特定のデバイスに対してモジュールを割り当てることができ，コアの修正を大きくせずに済む．この機能によってネットワークモジュールとして割り込みではなく定期的に問い合わせてパケットを受け取る仕組みが実現されていることが分かっていたためこれを利用して Unikernel でもネットワークパケットが受け取れるよう定期的な処理をするよう実装した．また Unikernel 内部でも定期的な問い合わせによってパケットを受け取るように実装した．

4.4.2　コンテナ向けモジュール

コンテナを動作させるにあたり，コンテナを起動することやコンテナからのハイパーバイザコールによってコアの処理へと移った後に最終的にデバイスドライバを使うといったことが必要である．コンテナの起動を行うモジュールを実装して，さらに特定デバイスへとモジュールを割り当てられる機能を利用してコアの機能を修正することなく追加のモジュールとしてハイパーバイザコールからデバイスドライバの利用を受け持つモジュールを実装した．ハイパーバイザとしてコアの機能を大きく変更せず追加の拡張モジュールとすることでセキュアであることを重視した．

5.　評価

実装した実行基盤によって評価を行った．それぞれの評価ではアプリケーションを実装してビルドした Unikernel を，実装したコンテナ基盤上で起動することで評価を行った．評価の環境は表 3 のとおりである．

表 3　評価環境

項目	内容
CPU	Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz
RAM	8GB
NIC	Intel 82574L Gigabit Netowork Connection
ゲスト OS	Ubuntu 20.04

ⓒ 2021 Information Processing Society of Japan

5.1 評価方法

評価ではコンテナの起動とネットワーク機能に関する評価を行う．各評価では Linux 上で動作する QEMU/KVM を用いて Unikernel を動作させた場合と比較する．これは 2 章で述べた従来の Unikernel の起動方法として一般的であり，比較をすることで提案手法の有効性について考察ができるためである．

測定の対象になるアプリケーションは Unikernel のアプリケーションとして TCP，UDP 通信を行うアプリケーション，Web サーバを実装して評価を行う．このアプリケーションは本研究の実行基盤で動作するコンテナのアプリケーションと Linux 上の QEMU/KVM を用いた仮想化基盤で動作する Unikernel のアプリケーションと同一のものである．動作させる Unikernel としても提案手法とほとんど同じものを動作せることができるといったことからも QEMU/KVM を用いた場合と比較をする理由にもなっている．両者の違いを述べるならばプラットフォームに依存するカーネルの初期化処理やドライバが異なるが，これらの違いも含めて評価の対象である．

5.2 起動時間

提案手法のコンテナの起動時間について評価を行った．従来手法でも起動が速いことは特徴の一つであり，本研究の実行基盤によるコンテナでも評価の対象とすることで従来との比較によって考察するためである．評価を行うアプリケーションは Unikernel のアプリケーションとしては最も簡単なコンソールへの出力をするプログラムである．コンテナの構築，起動開始から Unikernel のカーネルの初期化処理が終わりアプリケーションの処理が開始した時点を測定の対象とする．なおコンテナの起動は BitVisor 自体の起動のタイミングであり，デバイスドライバの登録などが終わりゲスト OS へと処理を移す直前で起動する．このタイミングで起動させる理由は BitVisor がこのタイミングで拡張機能などのモジュールを初期化するタイミングであり，実装したコンテナを扱うモジュールもこの配置となっているためである．また通常の動作であるゲスト OS を起動したことによる干渉がない状態で計測できるためである．

5.2.1 起動時間の評価

実施内容はコンテナを 1 から 10 まで起動する場合の起動時間のそれぞれの最小，平均，最大時間を計測した．結果は図 3 である．いずれの起動数においても 5ms 程度での起動となった．

また Linux 上の QEMU/KVM を利用した Unikernel を起動する時間も計測して比較を行った．なお Unikernel に含めたアプリケーションは本研究のコンテナのアプリケーションと同一のものである．結果は表 4 となった．提案手法によるコンテナの起動時間のほうが速い結果となり，Linux 上の QEMU/KVM による起動よりも速く起動できる結果となった．

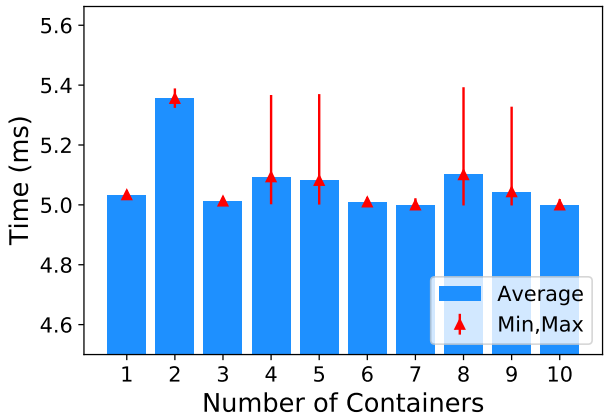

図 3 コンテナ数 1 から 10 までを起動したときの起動時間の結果

表 4 起動時間の比較

	QEMU/KVM	提案手法
起動時間 (ms)	3233	5

両者の差には，従来では QEMU などの仮想化ソフトウェアによる初期化や Unikernel のカーネルの初期化処理に時間がかかることや，提案手法では利用するデバイスドライバが固定であるため速く起動できるといったことが考えられる．

5.3 ネットワーク機能の評価

ネットワーク機能の評価として目的のコンテナが動作するマシンに対して別のマシンからネットワークを経由して機能を使う評価を行った．

この評価ではの二つの項目で実施した．まず TCP，UDP 通信を利用したアプリケーションで性能の計測を行う．TCP で受け付けるサーバーを実装してクライアントに対してデータの受信と送信が可能なアプリケーションを実装して評価を行う．UDP でも同様にデータの送信と受信をが可能なアプリケーションによって評価を行う．もう一つは Web サーバによる評価を行う．簡易的な Web サーバを Unikernel のアプリケーションとして実装することで評価を行う．コンテナを実際にアプリケーション視点で考えたとき，HTTP による Web サーバーやアプリケーションの利用が想定されるためである．実装された Web サーバはあらかじめ作成した HTML のページをディスクイメージに含めておいて Unikernel が読み込み，このファイルをサーバーからクライアントへ返却する．

さらに CPU の仮想化支援機構の機能の一つである Preemption Timer を有効化した場合の評価も行った．BitVisor は通常，CPU の仮想化支援機構を用いてゲスト OS を動作させている．BitVisor の動作上，状況によってはゲスト OS の処理から BitVisor のコアの処理に移らず目的の

表 5　UDP 受信と送信時間の結果

測定対象 (μ s)	QEMU/KVM	提案手法
送信	14	18
受信	556	2237

表 6　TCP 受信と送信時間の結果

測定対象 (μ s)	QEMU/KVM	提案手法
送信	7945	76537
受信	3140	267245

表 7　HTTP レスポンスタイムの計測結果

測定対象 (ms)	QEMU/KVM	提案手法	Timer 有効化
レスポンスタイム	13	163	52

コンテナの処理に CPU 時間が与えられない場合がある．Non Root Mode で動作するゲスト OS の動作から強制的に Root Mode の動作へと切り替えることができるタイマー機能である Preemption Timer 機能を有効化する．こうすることで提案手法で利用するハイパーバイザの処理の都合によってどの程度の影響が出ているかも考察するために評価に加えた．

5.3.1　TCP と UDP 通信の計測結果

目的のコンテナのアプリケーションとして TCP の性能の評価を行うためにサーバーとして 32KB のファイルの受信と送信をするもの，UDP でも 32KB のファイルの受信と送信をするプログラムを実装して評価を行った．またそれぞれの結果には QEMU/KVM で同様の計測を行った結果も含めることで比較できるようにしている．

UDP では表 5 の結果となった．提案手法は Linux 上の QEMU/KVM による Unikernel の動作と比べて送信，受信において両者とも性能が劣化する結果となった．

TCP では表 6 の結果となった．提案手法は QEMU/KVM と比べて UDP の結果と同様に受信，送信において両者とも性能が劣化する結果となった．

UDP では差が大きくないものの TCP では大きな差が出る結果となりいずれも提案手法のほうが性能が劣化した結果となった．

5.3.2　HTTP レスポンスタイムの計測結果

Web サーバによるレスポンスタイムの計測の結果は表 7 のようになり Linux 上の QEMU/KVM と比較して 15 倍程度の性能劣化となった．性能は劣化するもののハイパーバイザの処理の都合を減らす Preemption Timer を有効化した場合，性能の劣化は 4 倍までに縮められる結果となった．この結果から利用しているハイパーバイザによる動作の都合が大きく影響していることが分かった．いずれにしても性能は劣化するものの本研究は性能の改善が目的ではなく機能的には問題ない．

6.　おわりに

本研究の提案手法により小規模なハイパーバイザを用いて最終的には実装した Web サーバをコンテナとして動作させることができた．性能の観点では従来の仮想化基盤には劣るものの機能的には問題なく目標としていた実行基盤を実現して有効性を示すことができたといえる．またコンテナの代表的な特徴である起動時間が速いことも従来の仮想化基盤と比較することで示された．

ブロックデバイスを扱うための実装には至らなかったが，ネットワーク機能の評価によって提案手法の妥当性を示すことができた．ブロックデバイスはネットワークデバイスと同様な実装を進めていくだけである．

今後の課題には各評価における性能劣化の原因を究明して改善することが挙げられる．現段階では利用しているハイパーバイザの処理の都合が大きく影響していると考えており，例えば Unikernel が動作するコンテキストがプリエンプティブではなくスケジュールされることを待ち，順番がきたところで処理が進むためである．このことに加えてネットワークパケットの受信でいえば，割り込み起点ではなく定期的な受け取り処理によって成り立っていることも影響していると考えている．本研究ではハイパーバイザに BitVisor を利用しているが，改善するためには別のハイパーバイザあるいは OS を利用していくことも視野に入れてコンテナにとっての最適な実行基盤を検討していく必要がある．

また本研究によるコンテナの実行の実現は機能拡張を行う保護ドメインで実現されており，保護ドメインに対してプログラムを書くことで機能を拡張するのではなく，コンテナの単位で機能を拡張することができる．したがってハイパーバイザのコアの機能との協調によってゲスト OS からプロキシや特定用途などのファイルシステムなどとして連携をすることが考えられる．

参考文献

[1] 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*, ASPLOS '13, New York, NY, USA, Association for Computing Machinery, p. 461–472 (online), DOI: 10.1145/2451116.2451167 (2013).

[2] 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, *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, New York, NY, USA, Association for Computing Machinery, p. 218–233 (online), DOI: 10.1145/3132747.3132763 (2017).

[3] Williams, D. and Koller, R.: Unikernel Monitors: Extend-

ⓒ 2021 Information Processing Society of Japan

ing Minimalism Outside of the Box, *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)*, Denver, CO, USENIX Association (2016).

[4] Shen, Z., Sun, Z., Sela, G.-E., Bagdasaryan, E., Delimitrou, C., Van Renesse, R. and Weatherspoon, H.: X-Containers: Breaking Down Barriers to Improve Performance and Isolation of Cloud-Native Containers, *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, p. 121–135 (2019).

[5] Williams, D., Koller, R., Lucina, M. and Prakash, N.: Unikernels as Processes, *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '18, p. 199–211 (2018).

[6] Shinagawa, T., Eiraku, H., Tanimoto, K., Omote, K., Hasegawa, S., Horie, T., Hirano, M., Kourai, K., Oyama, Y., Kawai, E., Kono, K., Chiba, S., Shinjo, Y. and Kato, K.: BitVisor: A Thin Hypervisor for Enforcing i/o Device Security, *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '09, New York, NY, USA, Association for Computing Machinery, p. 121–130 (online), DOI: 10.1145/1508293.1508311 (2009).

[7] Bratterud, A., Walla, A., Haugerud, H., Engelstad, P. E. and Begnum, K.: IncludeOS: A Minimal, Resource Efficient Unikernel for Cloud Services, *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 250–257 (online), DOI: 10.1109/CloudCom.2015.89 (2015).

ⓒ 2021 Information Processing Society of Japan