IoT デバイスにおける軽量実行環境の性能比較

2015SE062 岡崎右希　2015SE086 都築翔太
指導教員：宮澤元

1　はじめに

近年，Internet of Things(IoT) というあらゆるモノをインターネットにつなげる技術が広まっている．インターネットはもともとコンピュータ同士をつなぐためのものだったが，スマートフォンやタブレット，家電製品，自動車などさまざまなモノが IoT によってインターネットにつながれるようになった．IoT によって離れたモノをリモコンなどによって遠隔操作したり，モノにセンサーをつけることでモノの状態を知るといったことができるので，IoT の技術を活用することで身近にあるモノの利便性を向上することができる．

IoT の問題点として計算リソースが少ないこと，これに伴いセキュリティ対策が難しいこと，通信レイテンシが大きくなることなどが挙げられる．普段インターネットにつなぐ必要がないモノをインターネットにつなぐため，CPU 性能は低く，メモリ容量も少なくなるので，IoT デバイスに対する計算リソースは少なくなる．そのため暗号化処理が十分にできず，ウイルス対策ソフトを導入することも現実的ではないので，セキュリティの問題にもつながる．また，IoT アプリケーションには車の自動運転や医療系など高いリアルタイム性を要求するものも多いので，通信レイテンシの大きさも問題になる．

一方，クラウド・コンピューティング（クラウド）において仮想環境での処理を効率化するために Unikernel という技術が提案されている．Unikernel とはライブラリベースの OS を使用し，仮想環境上で動作させるために構築された単一アプリケーション用の軽量実行環境である．クラウドにおける仮想環境ではアプリケーションを実行する際，アプリケーションごとに仮想マシン（VM：Virtual Machine) を起動しなければならず起動時間が長くなるといった問題や，OS とハードウェアの間にハイパーバイザを必要とするのでオーバーヘッドが発生してしまう問題がある．Unikernel では，仮想マシン上で通常の OS を用いる場合と比べてコンテキストスイッチが少ない．また，コードサイズが小さいので起動時間の短縮ができ，セキュリティの改善も期待できる [3,4,6].

Unikernel で解決される問題の多くが IoT の問題と似ていることから，Unikernel を IoT デバイスで動作させる研究が提案されている [1,5]．例えば，Unikernel は実行イメージが小さく，リソースが少ない IoT デバイスでも利用できると考えられる．

しかし，Unikernel を IoT デバイスで動作させることがどの程度有効かはまだ十分検討されているとは言えない．例えば，文献 [1] では Unikernel を動作させた際の具体的な性能の比較がされていない．仮想化支援ハードウェアを備えていない IoT デバイスにおいて Unikernel がどのように利用できるかについても考えなければならない．また，コンテナ仮想化などの技術との比較も行う必要がある [8].

本研究では，IoT デバイスで軽量実行環境を動作させることの実用性について検討する．軽量実行環境として，Unikernel, コンテナ仮想化, Linux/KVM を利用してネットワークを用いたデータ送受信実験を行う．この時，実行時間のほか，複数の性能指標を測定し，それぞれの環境での結果を比較する．

2　研究の背景

この節では，Unikernel，IoT に関する研究について述べた上で研究の課題について示す．

2.1　Unikernel

Unikernel とは，ライブラリ OS を使用して構築された単一アプリケーション用の軽量実行環境である．ライブラリ OS とは OS 機能をライブラリとして実装したものである．これを利用することにより，アプリケーションはユーザ空間とカーネル空間との間の情報のやり取りなしでハードウェアに直接アクセスすることができる．しかし，様々な入出力装置に対応するためには，各装置に対するライブラリ OS 用のドライバを書かなければならないため負担が大きくなる．そこでライブラリ OS を仮想環境上で動かしたものが Unikernel である [7]．ハイパーバイザが提供する仮想デバイス用のデバイスドライバを，ライブラリ OS に実装することで，Unikernel では実際の入出力の制御をハイパーバイザに任せることができる．

代表的な Unikernel に Xen project が開発した MirageOS がある．MirageOS はクラウドなどのプラットフォームにおける高性能なネットワークアプリケーションのための Unikernel を構築するライブラリ OS である．これは，ハードウェアの上でハイパーバイザが動作し，その上でアプリケーション本体とそれが必要とするランタイムのみをパッキングした Unikernel が動作するという構造である．これらをハイパーバイザのような仮想環境上で動かせるようにコードがコンパイルされる．そのためユーザとカーネル間のやり取りがなくなりハイパーバイザに直接アクセスすることができるので仮想マシン上で従来の OS を利用するよりも性能が向上している．

2.1.1　hvt

hvt[*1]は Unikernel とハイパーバイザ間のインタフェースを特殊化し，ホスト OS 上で直接動作する Unikernel で

[*1] ukvm という名前で開発が行われていたが 2018 年 9 月 13 日に hvt という名前に変更になった．

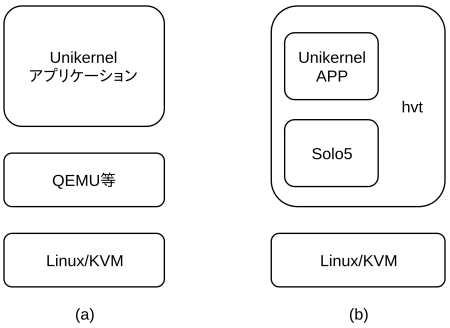

図 1 従来の Unikernel と hvt の構造比較

ある [2]．hvt は従来の Unikernel を更に改良したものだと言える．従来の Unikernel が QEMU や XEN のような一般化された仮想環境の上で動作させるものであったのに対し，hvt はホスト OS とのインタフェースの特殊化及び仮想化を担う Solo5 を利用して動作する．

hvt として動くアプリケーションは予めインタフェースなどを含めてビルドされている．hvt は Solo5 を用いて Linux/KVM 上で動作する．

図 1 は従来の Unikernel と hvt の構造を比較した図である．図 1 において，(a) は従来の Unikernel の構造を表しており，Linux/KVM の上で QEMU のような仮想化ソフトウェアが動作し，その上で Unikernel のような軽量化されたアプリケーションが実行される．それに対し，(b) では hvt の構造を表しており，Linux/KVM 上でアプリケーションと Solo5 がひとまとめになった hvt が実行される．

2.2 IoT と Unikernel

IoT デバイスのリアルタイム性やセキュリティといった問題を解決するためにも Unikernel によって性能がどの程度向上しているかを示すことが重要である．

しかし，Unikernel アプリケーションは通常，仮想環境上で動作するように作られているので，IoT デバイスなどのリソースが少ない環境では実装が非常に困難である．文献 [5,6] では，クラウドでのセキュリティとプライバシーを向上させるために，Unikernel を使用することが提案されている．Unikernel を使用することによって，物理ホストごとにいくつもの小さな VM を実行することが可能になる．最小限の VM を使用することで機密性が高くなり，セキュリティ性が向上する．文献 [1] では，この提案を IoT デバイスにも組み込むことで少ないリソースで従来のセキュリティツールを使用する問題を改善できるか，セキュリティとプライバシーの観点から Unikernel をどのように使用するかを検討している．IoT に Unikernel の技術を組み込む際の課題，問題点または制限については述べられているが，従来の OS ではなく Unikernel をリソースの少ないデバイスに組み込むことでどの程度性能が向上しているか，もしくは本当に Unikernel を組み込む必要があるのか具体的に示されてはいない．

3 軽量実行環境の性能比較

本研究では IoT デバイス上で軽量実行環境を動作させることの実用性を調べるための比較実験を行なう．そのために，以下のような実行環境を IoT デバイスで動作させる．まず，仮想化を用いない時の指標を測るため，通常の Linux 環境を用意する．次に，仮想化環境をいくつか用意する．1 つ目は QEMU などを用いる仮想化環境，2 つ目はコンテナを用いて仮想化を行う仮想化環境，3 つ目は Unikernel を用いる Unikernel 環境．これらの環境で比較実験を行い，通信時間，メモリ使用量，CPU 使用率の各性能指標を比較する．IoT デバイスはインターネットに繋げる必要があり，データをより速く送受信することが求められるので，通信にかかる時間は重要な要素の一つである．また，低リソースな IoT デバイスでは限られたメモリしか使用できず，CPU の使用率についても少ないほうが良い．

4 実験

3 節で示した環境を実際に用意し，それぞれの環境でネットワーク通信を行うアプリケーションを動作させた．

4.1 実験内容

IoT デバイスをクライアント，PC をサーバとし，クライアントからサーバに対しデータを IP 通信によって送信する実験を行った．実験ではクライアントが起動してからサーバからの返信を受け取るまでを 1 回とし，これを複数回ループさせる．クライアントはまず，センサーからの信号とみなした 0 から 99 の乱数を発生させる．次に発生させた値をあらかじめ動作しているサーバに対して，Internet Control Message Protocol(ICMP) を用いて送信する．その後，クライアントはサーバから受信完了コードを受け取るとプログラムを終了する．一方，サーバはクライアントから乱数を受け取ると受信完了コードを返信する．サーバはクライアントに返信を行うとまた，通信を待ち受ける状態に戻る．

また，実験中の各項目について計測を行い，比較する．比較する項目は以下のようである．

- CPU 使用率
- メモリ使用率
- 実験にかかった合計時間

CPU 使用率はクライアントが起動してからプログラムが終了するまでの間の状態を計測する．計測には proc ファイルシステムを用いて実験中の CPU の状態を計測する．メモリ使用量は仮想環境でクライアントを動かした時の物理メモリ量を計測する．実験にかかった合計時間は値をサーバに送信してから返信を受け取るまでの間にかかった時間を計測する．一回行うごとにこれらを計測し，記録する．

表 1 比較環境の仕様

項目	Linux 環境 (PC)	Linux 環境 (RPi3)	Docker 環境	QEMU/KVM 環境	hvt 環境
利用する仮想化環境	無	無	Docker	Linux/KVM	Unikernel
仮想化の有無	無	無	有		
OS	Ubuntu14.04LTS	OpenSuse Leap			
ハードウェア	PC	Raspberry Pi3			
CPU	Intel Core i5-4210M	ARM Cortex-A53			
メモリ	4GB	1GB			

4.2 実験環境

実験では IoT デバイスとして RaspberryPi3(RPi3) を用いる．性能比較を行う環境を表 1 に示した．表 1 は環境ごとの利用する仮想環境，仮想化の有無，OS，ハードウェア，CPU，メモリをまとめたものである．Linux 環境 (PC) では Ubuntu14.04LTS を用いた．Linux 環境 (RPi3) では，64bit linux として動作する OpenSuse Leap を用いた．Docker 環境ではホスト OS として OpenSuse Leap を動作させ，その上で Docker を用いてアプリケーションを動作させた．QEMU/KVM 環境では，ホスト OS として，OpenSuse Leap を動作させ，QEMU/KVM を用いて仮想マシンを動作させた，仮想マシンには debian を利用する．hvt 環境ではホスト OS として OpenSuse Leap を動作させ，その上で Solo5 と hvt を用いて仮想化を行い，Unikernel アプリケーションを実行させ実験を行なった．

4.3 実験結果

表 2 は各環境で計測した CPU 使用率の平均値とメモリ使用量を表した表である．CPU 使用率は RPi3 を用いた 4 つの環境で計測した．結果は Linux 環境，Docker 環境，QEMU/KVM 環境，hvt 環境の順に使用率が低くなった．仮想環境の中では Docker 環境が最も優れていた．hvt 環境は QEMU/KVM 環境と比較してわずかに使用率が高くなったが大きく劣ることはなかった．

メモリ使用量は仮想環境を用いた 3 つの環境で計測した．結果は Docker 環境，hvt 環境，QEMU/KVM 環境の順に少なくなった．CPU 使用率と同様に Docker 環境が最も優れていたが，hvt 環境もほとんど同等の結果となった．

実験にかかった合計時間については図 2 のようになった．図 2 は通信の結果から各環境ごとに箱ひげ図を作成しまとめたものである．縦軸は実験にかかった合計時間を表しており，単位はマイクロ秒である．ひげの上端下端はそれぞれ，最大値，最小値を表している．箱の上端下端は 75 %値, 25 %値, 中央の白線は 50 %値を表している．＊は平均値を表している．通信は各環境ごとに 100000 回行なった．PC と IoT デバイスとの性能差を図るために用意したそれぞれの Linux 環境では PC のほうが 200 マイクロ秒ほど早かった．次に仮想環境での結果をみると，Docker がもっとも良いということが分かる．仮想化を用いない Linux 環境とも 100 マイクロ秒ほどしか差がなく全体的に安定もしている．先述の通り，単一プロセスのみで動作していることが一番の要因ではないかと考えられる．次に hvt 環境である．この環境では Docker 環境と 200 マイクロ秒ほど差がでた．

hvt 環境も Docker 環境と同様に単一プロセスとして動作しているがこの差がでた要因として考えられるのは，hvt 環境では Solo5 というソフトウェアが Linux/KVM を用いて仮想化支援を行うことである．hvt 環境では Linux/KVM というホスト OS 型とハイパーバイザ型の中間のような仮想化を行い，独自のアプリケーションを動かすので，Docker 環境との差がでたと考えられる．最も遅かったのは Linux/KVM 環境である．この環境は hvt 環境と 200 マイクロ秒ほどの差がでた．この環境では仮想化に Linux/KVM を用い，更に QEMU というソフトウェアでユーザーとのやりとりを行う．また，仮想化の際，単一プロセスではなく Linux kernel ごと仮想化を行っているので，これほどの差が出たと考えられる．

これらの結果及び考察により, 現時点では Docker を IoT デバイスで動作させることがもっとも実用的であると考えられる．しかし，Unikernel を用いた場合についてもそれほど劣る結果にはならなかった．まず, CPU 使用率の面では Docker 環境ほどの安定性は無いものの，QEMU/KVM 環境のように起動時に大きく CPU 使用率を上昇させることなく動作させることができる．また，メモリ使用量の面においても Docker 環境とほとんど同じ程のメモリしか使用せず，IoT デバイスのような低リソースな環境ではとても有望であると考えられる．さらに，通信実験にかかった合計時間の面でも安定した通信を行うことができ，QEMU/KVM 環境よりも早く通信を行うことができるので，Unikernel を IoT デバイスで動作させることも有望であると考えられる．

5 おわりに

IoT デバイスで軽量実行環境を動作させることの実用性を検討するために，軽量実行環境である Unikernel, Docker，Linux/KVM を IoT デバイス上で動作させ比較実験を行った．実験を行なうための性能指標として，CPU 使用率，メモリ使用量，通信にかかった時間を計測した．実験結果から，Docker を IoT デバイスに用いることが

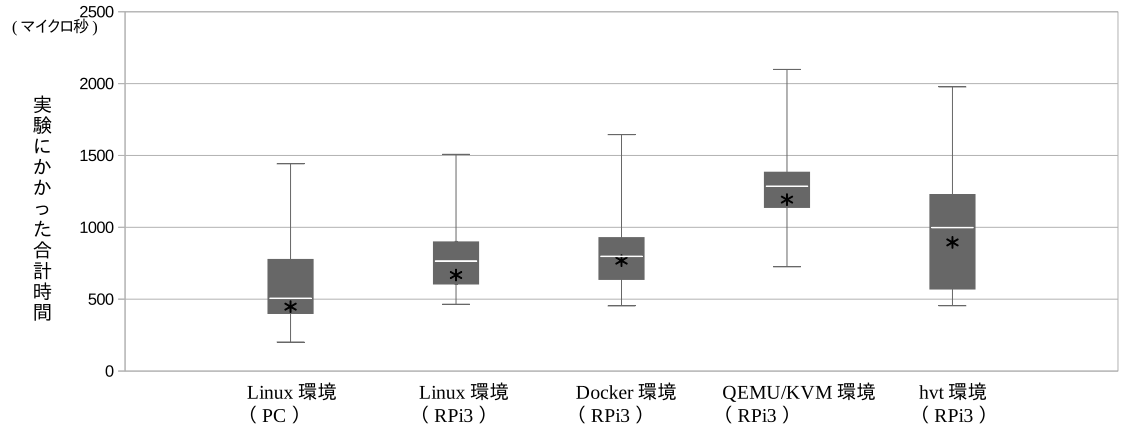

図 2 実験にかかった合計時間

表 2 CPU 使用率・メモリ使用量の計測結果

項目	Linux 環境 (RPi3)	Docker 環境	QEMU/KVM 環境	hvt 環境
CPU 使用率 (平均)	0.4%	0.5%	6.6%	7.1%
メモリ使用量	—	7MB	238MB	10MB

もっとも実用的であると考えられる．しかし，hvt 環境は Docker 環境と同等もしくはわずかに劣る結果であったが，QEMU/KVM 環境と比較すると優れているので，IoT デバイスに Unikernel を用いることは有望であると考えられる．

今後さらに IoT デバイスを効率的に動かせる手法として，ハイパーバイザを使わず Unikernel を直接ベアメタル環境で動作させることが考えられる．ベアメタルとは，基盤となる OS またはハイパーバイザのないコンピュータシステムである．Unikernel と比較すると仮想化を行なうホスト OS またはハイパーバイザがなく，ハードウェア上で直接動作する．オーバヘッドの削減や起動時間の短縮といった観点からベアメタル上で Unikernel を動かす手法もある．ホスト OS がないので仮想環境を用いる通常の Unikernel 環境よりもさらに効率よくハードウェアにアクセスすることができる．これを実現するためには，OS やハイパーバイザのように共通の仮想レイヤがないので，デバイスごとにプログラムを書き換える必要がある．

参考文献

[1] Bob Duncan, et. al, *“Enterprise IoT Security and Scalability: How Unikernels can Improve the Status Quo,”* in Proceedings of 2016 IEEE/ACM 9th International Conference on Utility and Cloud Computing, pp.292–297, Dec., 2016.

[2] Dan Williams and Ricardo Koller, *“Unikernel monitors: extending minimalism outside of the box,”* In 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16), USENIX Association, 2016.

[3] Madhavapeddy Anil, Leonard Thomas, Skjegstad Magnus, Gazagnaire Thomas, Sheets David, Scott David, Mortier Richard, Chaudhry Amir, Singh Balraj, Ludlam Jon, Crowcroft Jon and Leslie Ian, *“ Jitsu: Just-In-Time Summoning of Unikernels,”* the 12th USENIX Conference on Networked Systems Design and Implementation (NSDI),2015.

[4] Madhavapeddy Anil, Mortier Richard, Charalampos Rotsos, Scott David, Singh Balraj, Gazagnaire Thomas, Smith Steven, Hand Steven and Crowcroft Jon, *“Unikernels: Library Operating Systems for the Cloud,”* SIGPLAN Notices, 48 (4),2013.

[5] B. Duncan and M. Whittington, *“Enhancing Cloud Security and Privacy: The Power and the Weakness of the Audit Trail.”* In Cloud Comput. 2016,pp.125 130, Rome, 2016.

[6] Maxime Compasti, Rmi Badonnel, Olivier Festor, Ruan He and Mohamed Kassi-Lahlou *“Unikernel-based Approach for Software-Defined Security in Cloud Infrastructures,”* in NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium,23-27 April 2018.

[7] *“Unikernels:Rise of the Virtual Library Operating System,”* `http://queue.acm.org/detail.cfm?id=2566628`（2018 年 9 月 23 日アクセス）.

[8] Roberto Morabito, et. al, *“Consolidate IoT Edge Computing with Lightweight Virtuaization,”* in IEEE Network, Vol.32, Issue 1, pp.102–111, Feb 2018.