IoT デバイスにおける軽量実行環境のストレージ性能比較

2016SE074 須崎 大致　2017SE093 外山 大輔
指導教員：宮澤 元

1　はじめに

身の回りものをインターネットに接続する技術 Internet of Things(IoT) が普及している．IoT 技術によってインターネットに接続されるようになった機器 (IoT デバイス) の多くはセンサを備えており，これをデータの収集などに活用している．IoT デバイスの多くでは，データの処理機能は付加的な機能であり，デバイスのデータ処理能力はさほど高くないのが普通である．そこで IoT デバイスは外部にデータ処理を依頼することが多い．

IoT デバイスで生成されたデータはクラウドコンピューティング (クラウド) によって処理されることが多い．クラウドはネットワークを介して様々な計算処理をサービスとして提供するものであり，データセンタに設置された大量の計算リソースを用いて，多数の利用者からの要求を処理することができる．

しかし，IoT のデータ処理をクラウドで行うにはいくつかの問題点がある．まず，多数のデバイスがアクセスすることにより処理負荷が集中し，ネットワーク帯域の利用が増大してしまう問題がある．近年では IoT デバイスの数が増加し続けており，クラウドの処理性能であっても IoT デバイスからのデータを処理しきれない可能性もある．また，アプリケーションによっては IoT デバイスとクラウドとの通信レイテンシの大きさが問題となる．クラウドのデータセンタは一般に IoT デバイスからのネットワーク的な距離が遠く，リアルタイム性の高い IoT アプリケーションにとって，通信レイテンシが大きすぎる．

クラウドの問題点を解決する手段としてエッジコンピューティングが注目されている．エッジコンピューティングは IoT デバイスが生成した大量のデータを IoT デバイスに近い場所に設置されたエッジデバイスを用いて処理する技術である．計算処理やネットワークの負荷がクラウドへ集中してしまう問題の解決策として期待されている．

一方で，エッジコンピューティングにおいてはクラウドとは異なる問題が発生することがありうる．まず，エッジデバイスでの計算処理に関して，エッジデバイスの CPU 性能やメモリ量が一般にクラウドのデータセンタの計算ノードに比べて乏しいことから，アプリケーションの処理性能に影響が出てしまう可能性がある．またこのことで，セキュリティ対策に十分な計算リソースを割くことができないという懸念もある．これに対して，メモリフットプリントが小さくアタックサーフェスも小さいことからセキュリティに優れる Unikernel という軽量仮想実行環境を IoT デバイスで動作させる提案もなされている [1].

次に，エッジデバイスにおけるストレージ入出力処理に関しても考慮すべき点がある．クラウドではデータセンタの計算リソースを生かし，計算ノード上のアプリケーションはストレージノードが提供するストレージシステムを利用して入出力処理を行っている．エッジコンピューティングでは同様のストレージシステムを提供することは計算リソースの制限から困難である．また，エッジデバイスからクラウドのデータセンタへの通信レイテンシの大きさや利用できるネットワーク帯域を考えると，エッジデバイス上のアプリケーションがクラウドが提供するストレージシステムを利用することは難しい．そこで，エッジコンピューティングでは，アプリケーションのストレージ入出力はエッジデバイス自体がローカルに持つストレージを利用して行う必要がある．このとき，エッジデバイスにおいて Unikernel のような仮想実行環境を利用するという提案に対して，仮想化オーバヘッドを含めたエッジデバイスの入出力性能を把握することは重要である．

本研究では，エッジコンピューティングにおいて IoT デバイスの計算リソースを活用する際に，IoT デバイス上で Unikernel を利用することによるストレージ入出力性能への影響について調べる．エッジデバイスにおける複数の実行環境でのストレージ入出力性能を明らかにし，IoT デバイス上で Unikernel を利用することの妥当性を明確にする．具体的には，IoT デバイス上で動作する Unikernel を含む複数のアプリケーション実行環境を用いてファイルアクセス実験を行い，各実行環境におけるファイルアクセス性能を比較する．実行環境には Unikernel 環境の他，Docker 環境，QEMU/KVM 環境，Xen 環境を用意し，各環境での実験に伴う動作時間を測定し比較する．

2　研究の背景

本節では，本研究の背景として IoT，クラウド/エッジコンピューティング，Unikernel および先行研究について述べる．

2.1　IoT

Internet of Things(IoT) は身の回りのあらゆるものをインターネットに接続する技術のことを指す．家電や自動車など様々なものをインターネットに接続できるようになり，人の手を介することなく自動で情報を伝達し合えるようになる．また，多くの IoT デバイスはセンサを備えており，これを利用したデータ収集に広く活用されている．IoT 技術が進むにつれてこれまでインターネットに接続できなかった機器をインターネットに接続することが可能になってきている．このような IoT デバイスの多くはさほど高いデータ処理能力を持たないので，デバイス外部にデータ処理を依頼することが多い．

2.2 クラウドコンピューティング

IoT デバイスで生成されたデータの多くはクラウドコンピューティング（クラウド）に送られ，クラウド内のデータセンタに配置された膨大な計算リソースで処理される．しかし，IoT デバイス数は増加し続けているので，データ処理要求の集中によりクラウドの負荷が増加し，処理しきれなくなる可能性がある．また IoT デバイスとクラウド間の通信レイテンシが問題となるアプリケーションには，データ処理をクラウドに依頼する方法では対応できない．

2.3 エッジコンピューティング

IoT デバイスから生成されるデータの処理を IoT デバイスの近くに配置した計算リソースを用いて行うようなコンピュータの利用形態をエッジコンピューティングと呼ぶ．IoT デバイスからのデータはエッジデバイスに送信され，そこで処理されたのちにクラウドに送られる．これによりクラウドに負荷が集中してしまうという問題を解決できる．また，近年では比較的多くの計算リソースを備えた IoT デバイスも登場してきており IoT デバイスそのもののエッジコンピューティングでの活用も進んでいる．一方で，クラウドで利用されるデバイスに比べるとエッジデバイスの計算リソースは限られており，処理能力も低い．このことに起因したエッジデバイスでの処理レイテンシの増大や不十分なセキュリティ対策といったさまざまな問題が生じる可能性がある．

2.4 Unikernel

Unikernel とはライブラリ OS を用いて構成された単一アプリケーション用の実行環境である [2] ．ライブラリ OS とは，単一アドレス空間で OS をライブラリとして実装し，ハードウェアの排他制御を最低限に抑える技術である．これにより，ユーザ空間とカーネル空間の間のコンテキスト切替が不要になり，アプリケーションがハードウェアに直接アクセスできることで性能を向上できる．

2.5 先行研究

岡崎らは，IoT デバイスに Unikernel を用いることの有効性を実際に他の環境と比較して検討している [3]．IoT デバイスに見立てた Raspberry Pi 3 に Unikernel の他，Docker 環境，QEMU/KVM 環境を構築し実験を行った．この実験では，IP 通信を用いたクライアント・サーバ間のデータ送受信時間や，それに伴うメモリ使用量および CPU 使用率を各実験環境ごとに測定し比較している．測定結果から IoT デバイス上での Unikernel は有望であると結論されているが，入出力性能についてはネットワーク性能に関する評価を行っただけであり，ストレージ性能に関する評価は行われていない．

3 軽量実行環境におけるストレージアクセス

エッジコンピューティングにおいて IoT デバイスそのものを活用することを考慮すると，そのストレージ性能を把握することは重要である．特にセキュリティを考慮して Unikernel のような軽量実行環境を IoT デバイスで利用する場合，ストレージ入出力に伴うオーバーヘッドがどの程度の性能低下を引き起こすのか確認する必要がある．

3.1 実ストレージデバイスへのアクセス

図 1 は通常の Linux 環境において実ストレージデバイスへ書込みを行う操作の手順を表している．通常の Linux 環境では，書込みシステムコールである write を使用しても書き込まれるデータは直接ストレージデバイスには書き込まれず，一度 OS 内でメモリ上にバッファされる．バッファされたデータはシステムコールの fsync を使うとフラッシュされ，ストレージデバイスに書込まれる．

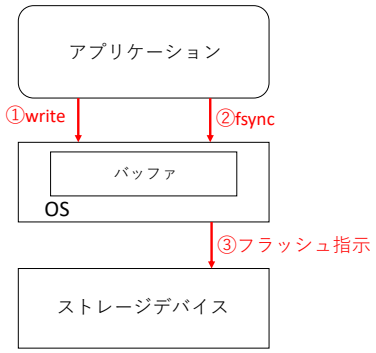

図 1 Linux 環境での実ストレージデバイスへの書込

一方仮想環境ではゲスト OS 上のアプリケーションが fsync を使うと，バッファは実ストレージデバイスではなく仮想ストレージデバイスへ書込まれる．その後に実ストレージデバイスへも書込ませるためには各環境の仮想マシンの実装に修正を加える必要がある．図 2 は Unikernel 環境の一種である hvt/solo5 環境において実ストレージデバイスへ書き込みを行う操作の手順を表している．solo5 Unikernel の書込み関数 solo5_block_write を用いてホスト上のファイルとして実現された仮想ストレージデバイスに書込みを行うと，ホスト OS 上のバッファに書込みが行われる．その後，仮想化支援ソフトウェアの hvt が fsync を用いると，バッファがフラッシュされ実ストレージデバイスに書き込まれる．

4 実験

IoT デバイス上で動作する複数の軽量実行環境において，環境ごとにストレージシステムの実装に違いがある中でのファイルアクセス性能を比較するために実験を行った．実験環境として Linux 環境，コンテナ実行環境である Docker 環境，ハイパーバイザ型仮想環境である QEMU/KVM 環境，hvt/solo5 環境，Xen 環境を利用し

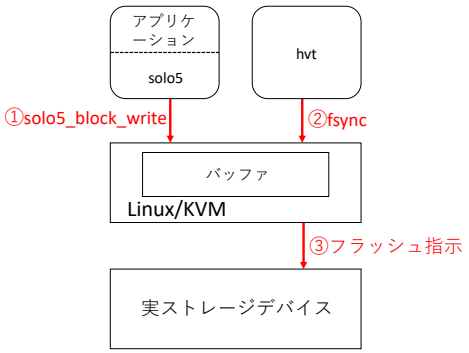

図 2　hvt/solo5 環境での実ストレージデバイスへの書込

て，2 通りのサイズのファイルをそれぞれ読み書きする処理時間を計測する．

4.1　実験環境

実験では IoT デバイスに見立てた Raspberry Pi4(Model B) を使用する．比較対象の各実行環境の仕様を表 1 にまとめた．

4.2　計測実験

ファイル (hvt/solo5 環境ではブロックデバイス) へアクセスし，ファイルを読み書きする処理時間を測定する．ファイルアクセスを開始した時刻と，デバイスへの操作が終了した時刻を取得し，その差し引きでファイルの操作にかかった時間を計測する．ファイルの操作は読み込み，書き込みを別々のプログラムで行う．ファイルアクセスするプログラムは C 言語で作成し，hvt/solo5 環境では，同様の挙動をする Unikernel アプリケーションを作成した．書込み操作では fsync を用いて実ストレージデバイスへの書込みを行った．読込み操作ではファイルを単純に読込んだだけでファイルキャッシュは無効化していない．

図 3 と図 4 は実験にあたり作成したプログラムを擬似コードで表したものである．図 3 は hvt/solo5 環境以外で動作させたプログラムである．実際に読込みや書込みの処理をしている時間を計測するために，ファイルを開いたり閉じたりする処理は繰り返しの外に置いた．図 4 は hvt/solo5 環境で動作させた Unikernel アプリケーションであり， hvt/solo5 の仕様で 1 ブロックずつ読み書きを行っている．

読み書きを行うファイルサイズは以下の通りである．

- 4KB(8 セクタ分) のファイル
- 1GB(2,097,152 セクタ分) のファイル

実験を行うファイルサイズは，エッジコンピューティングにおいて IoT デバイスそのものの計算リソースが使われることを想定して決めた．4KB 分のファイルアクセスでは，100 行程度のプログラムや 4000 文字程度のデータが格納されたファイルを処理することを想定した．1GB 分のファイルアクセスでは，1000 枚程度の画像や短めの動画を処理することを想定した．

```
ファイルを開く
start = 時刻取得
for ( int  i = 0;  i < 繰返し回数;  i++){
    書き込みまたは読み込み
    （fsync処理）
}
end = 時刻取得
ファイルを閉じる
実行時間を表示（（start - end）/　繰返し回数）
```

図 3　実験用擬似プログラム

```
ブロック情報取得
start = 時刻取得
for (solo5_off_t  i = 0;  i < 繰返し回数;  i++){
    for(solo5_off_t  offset = 0; offset < ファイルサイズ;
                                    offset += ブロックサイズ){
        書き込みまたは読み込み(1ブロック)
    }
}
end = 時刻取得
実行時間を表示（（start - end）/　繰返し回数）
```

図 4　実験用擬似プログラム (hvt/solo5)

4KB のファイルアクセスでは 100000 回，1GB では 10 回の読み書き処理を繰り返し，その平均値を結果とした

4.3　実験結果

各環境での実験結果を表 2，表 3 に示す．全体の傾向としては Linux 環境が一番速く， Docker 環境は少し劣るが同等の結果だった．これらに次ぐ結果となったのが仮想環境である Xen 環境と QEMU/KVM 環境である．条件によって変動はあるものの両者の間に大きな差はなかった．hvt/solo5 環境 は最も遅かった． fsync 関数利用の有無で比較すると，利用する方が時間がかかるが実行時間の順位に大きな差は出なかった．

4.4　考察

Docker 環境が Linux 環境とほぼ同等の結果だったのは，コンテナがホスト OS のカーネルを利用するオーバーヘッドが非常に小さいからであると考える．QEMU/KVM 環境の結果は Linux 環境よりもオーバーヘッドが大きいことを考慮すると概ね妥当であるが， 1 GB の読み込みでは Linux 環境より 15 倍以上の時間がかかった．これは仮想マシンに割り当てているメモリ量が影響してキャッシュが効いてないことが原因だと考える．同じ仮想環境である Xen 環境でも QEMU/KVM 環境と同様の傾向が見られた．

Unikernel である hvt/solo5 環境でのストレージアクセスに時間がかかるのは，以下の 3 項目が原因であると考える．

項目 1　ストレージアクセス要求がハイパーコールを使って kvm 経由で hvt に送られている

表 1 実験に用いた実行環境の仕様

項目	Linux 環境	Docker 環境	QEMU/KVM	hvt/solo5 環境	Xen 環境
OS	Ubuntu 20.04 LTS				
Kernel ver	5.4.0-1026				5.10.0-rc3
ハードウェア	Raspberry Pi4				
割り当てメモリ量	4GB	4GB	2GB	512MB	1GB

表 2 計測結果 (4KB)

	Linux	Docker	QEMU	hvt/solo5	Xen
write(sync)	2.35ms	2.61ms	10.86ms	20.14ms	4.83ms
write	13.58 μ s	15.68 μ s	26.31 μ s	131.89 μ s	26.65 μ s
read	1.61 μ s	4.77 μ s	3.15 μ s	96.36 μ s	4.52 μ s

表 3 計測結果 (1GB)

	Linux	Docker	QEMU	hvt/solo5	Xen
write(sync)	36.05s	37.88s	31.06s	5388s	54.47s
write	28.35s	22.97s	26.37s	49.74s	28.85s
read	0.97s	0.98s	15.51s	26.43s	23.58s

項目 2 hvt が solo5 のストレージアクセス要求を Linux のファイルシステムに対するアクセスとして処理している

項目 3 solo5 のストレージアクセス要求が 1 ブロック単位でしか行われない

項目 3 の 1 ブロックのアクセスのたびに項目 1 と項目 2 のオーバーヘッドが発生するので，他の環境と比べて hvt/solo5 環境でのストレージアクセスに大幅に時間がかかる．

項目 1 は hvt/solo5 環境のみならず仮想マシンを実装する環境であれば発生するオーバーヘッドである．これは，この場合は 仮想マシンに対してブロックデバイスのインターフェースを提供する BitVisor などのハイパーバイザを用いることで軽減できる．この機能は準パススルーと呼ばれ，仮想マシン上のデバイスドライバに対し，実ハードウェアの多くをそのまま見せることができる [4].

項目 2 は仮想マシンの入出力を hvt や QEMU などの制御ドメインの OS 上で動くユーザプロセスに行わせていることによる制限であり，仮想マシン内の OS や Unikernel が準仮想化ドライバを実装することで対処できる．準仮想化が実現できる Xen などのハイパーバイザと virtio などの準仮想化ドライバを使って実現する．これにより完全仮想化に伴うハイパーバイザがハードウェアをエミュレートするためのオーバーヘッドが軽減でき，ゲスト OS からハードウェアを操作する性能が向上する．

項目 3 は solo5 のディスクアクセスを複数ブロックで行えるようにすることで改善が見込まれる．現在の solo5 では実装上の制約でディスクアクセスが単一ブロック単位に制限されており，本実験でもディスクアクセスを単一ブロック単位で行った．例えば 4KB の読み書きで hvt/solo5 環境は他の仮想環境と比べディスクアクセス関数を 8 倍多くの回数呼び出していることになる．この問題点のみでも解決できれば hvt/solo5 環境におけるストレージアクセス時間を大幅に短縮できると考える．

5 まとめ

IoT デバイス上で動作する軽量実行環境におけるファイルへのアクセス性能を調べ比較することを目的として，実行環境の構築とプログラムの作成，および実験を行った．実験結果から，Raspberry Pi 4 においては仮想化のオーバヘッドが大きく，Linux 環境や Docker 環境と比べ，他の仮想環境のストレージアクセス性能が低いことがわかった．特に Unikernel 環境である hvt/solo5 環境では，実装上の制限もありストレージ入出力性能が低く，IoT デバイスで用いるのに現状では適切ではないという結果になった．

エッジコンピューティングの計算ノードとして IoT デバイスが用いられていく中で，メモリフットプリントが小さくセキュアであることや起動時間が素早くオンデマンドな処理が可能であるという Unikernel の特徴を活かして IoT デバイスの問題点を解決するのは重要だと考える．今後の課題は前節で述べた hvt/solo5 環境の解決策を実施した後，再度実験を行いその有効性をより厳密に検証することである．また，今回用意することができなかった Xen の Domain U を使った実験も行う．

参考文献

[1] B.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] A.Madhavapeddy ,et al., “Unikernels: Rise of the Virtual Library Operating System, “ in *Communications of the ACM*, Vol. 57, No 1, pp.61—69, Jan. 2014.

[3] 岡崎右希ほか:”IoT デバイスにおける軽量実行環境の性能比較”，南山大学理工学部卒業論文 (2019)

[4] 品川 高廣ほか，” 準パススルー型仮想マシンモニタ BitVisor の設計と実装”，情報処理学会研究報告システムソフトウェアとオペレーティング・システム（OS），Vol. 2008, No 77, pp.69—76, Jul. 2008.