Vol.2017-OS-141 No.24
2017/7/27

IoT向け大容量小型コンピュータOliveにおけるOpen Channel SSDの実践

石川 広男[1,a)]　追川 修一[1,b)]

概要：本稿では、Linux の Open Channel SSD 実装である LightNVM を使って、IoT 向け大容量小型コンピュータ Olive に搭載された SSD を管理する実装を試作し、その効果や課題を示す。Open Channel SSD は、SSD の構成方法の一つである。その特徴は、従来であれば SSD のコントローラ内部に実装されていた Flash Translation Layer (FTL) によって隠蔽されている NAND フラッシュメモリのチャンネルへのアクセスを、ホストの OS に公開するというところにある。これによって NAND のチャネルに対するホストアクセスの並列度を向上し、ホストのアプリケーションのワークロードに応じた処理を可能にする。Olive を使った実験では、デバイスの並列度を変更することによって入出力性能も変化することが確かめられた。

A Study on Open Channel SSD with Olive

Ishikawa Hiroo[1,a)]　Oikawa Shuichi[1,b)]

Abstract: Open Channel SSD is a methodology for organizing a system with SSDs (Solid State Drives). In contrast to traditional SSDs, a Open Channel SSD allows a host OS to access physical page address spaces of its NAND flash chips, and to implement an application-oriented FTL on the host OS side. LightNVM is a framework running inside the Linux kernel to use Open Channel SSDs. This paper reports our study on the Light NVM framework with Olive. Our results show that the parallelism of buses and chips influences the I/O performance under the framework.

1. はじめに

Solid State Drive (SSD) は NAND フラッシュメモリを記憶媒体とする記憶装置である。半導体であるため、省電力性や耐衝撃性あるいは機構部品を持たないといった特徴を持つことや、ランダムアクセスの性能が良いという特徴を持つ。このことから、スマートフォン、タブレットやノートパソコンなどのモバイル製品全般で広く用いられている。近年では All Flash Array と呼ばれる、ストレージに SSD のみを用いた製品が製造されるにいたり、サーバー分野でも普及が進んでいる。

SSD は一般に、NAND フラッシュメモリそのものと、ホスト PC とのインタフェースを実装するコントローラとから構成される。コントローラは内部に Flash Translation Layer (FTL) を持つ。この FTL が LBA と言われる論理アドレスと NAND 上の物理アドレスとの変換や、ウェアレベリングおよびガーベジコレクションのような媒体の状態管理機能を提供している。

NAND フラッシュメモリは、新しい情報を記憶させるためには必ず古い情報を消去しなければならない上に、消去回数が増えるほどデータ保持の特性が物理的に劣化するデバイスである。NAND フラッシュメモリを用いた周辺機器は一般に、この特性をウェアレベリングやガーベジコレクションといったソフトウェアのロジックによって補填している。ところが、ウェアレベリングなどのロジックはバックグラウンドで動作するため、ホストからのデータの読み書き性能に影響を与えることがある。[1] の関連研究では特に、高レスポンスの要求される Web キャッシュサー

[1] 株式会社フィックスターズ
Fixstars Corporation, Gate City Ohsaki West Tower 18F, 1-11-1, Ohsaki, Shinagawa-ku, Tokyo 141-0032, Japan
[a)] hiroo.ishikawa@fixstars.com
[b)] shui@fixstars.com

表 1　Olive の仕様

Chassis	2.5 inch HDD Form Factor
FPGA	Xilinx Zynq-7030
CPU	ARM Cortex A9 Dual Core
RAM	DDR2 512MB
NIC	1Gb Ethernet
Storage	eMMC NAND Flash Memory (512GB∼13TB)
Power Consumption	6.5W

バにおいて SSD の読み出し性能が予測不能となる点を問題視している。

Open Channel SSD (OC-SSD) は Bjørling らによって提案されている SSD の構成方法およびそれを使ったシステムである。OC-SSD は、NAND フラッシュメモリのインタフェース (チャネル) をホスト側に開放する。これによって従来の FTL の処理をホスト OS で実装できるようにする。この仕組みによって SSD へのアクセスの予測性や、ワークロードに特化した管理ロジックを組むことの容易性が高まることが期待できる [1]。以上のように、OC-SSD はホスト OS での FTL の実装を前提とするデバイスである。Linux カーネルでは、OC-SSD 管理のソフトウェアフレームワークとして、LightNVM が導入されている。

本稿では、LightNVM フレームワークを用いて Olive に搭載されているフラッシュメモリを管理する実装を試作し、LightNVM の効果や課題を議論する。Olive とは株式会社フィックスターズが開発した 2.5 インチ・フォームファクタのストレージシステムである。大容量を実現するために複数の eMMC フラッシュメモリを搭載する。これらのフラッシュメモリは FPGA を経由してブロックデバイスドライバが管理する構成となっている。LightNVM フレームワークは本来 OC-SSD を対象とするフレームワークであり、eMMC の SSD を管理することは本来の目的とは異なる部分があるが、Olive のハードウェアが持つ非対称性を OC-SSD により制御することによる高性能化について、その可能性を実験する。

2. Olive

Olive は、株式会社フィックスターズが開発した 2.5 インチ・フォームファクタのサーバーである [2]。Olive の概略仕様を表 1 に示す。デュアルコアの ARM プロセッサ、FPGA、512MB の RAM、および 1 Gigabit Ethernet を搭載している。ストレージとして eMMC NAND フラッシュメモリを搭載し、最大 13TB の容量を実現する。NAND フラッシュメモリは eMMC コントローラを実装した FPGA を経由して ARM プロセッサと接続されている (図 1)。FPGA は複数ある NAND フラッシュメモリを統合して管理するためのインタフェースをプロセッサ側に提供している。ARM プロセッサでは Linux カーネルが稼動し、そ

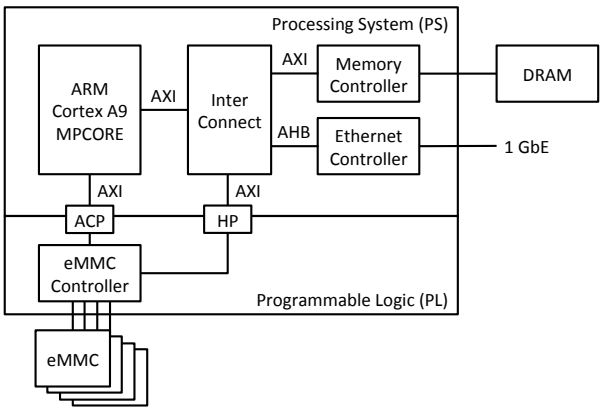

図 1　Olive の内部構成

のカーネルには FPGA や NAND フラッシュメモリを駆動するためのブロックデバイスドライバが実装されている。ユーザーランドは Yocto Project をベースにカスタマイズしたものを使用している。

3. 設計と実装

本節では、LightNVM フレームワーク向けのブロックデバイスドライバの設計と実装について述べる。

3.1 LightNVM フレームワーク

LightNVM フレームワークは、Linux カーネルに標準で実装されている OC-SSD 向けのフレームワークである。LightNVM フレームワークそのものは大きく 3 つの部品からなる (図 2)。一つ目の NVMe ドライバはフレームワークの最下層にあたり、NVMe デバイスの制御およびデータの転送を処理する。フラッシュメモリの PPA（物理ページアドレス）空間を上位層に提供する。二つ目は、LightNVM サブシステムである。この部品は、NVMe ドライバを統合し、上位のメディアマネージャやユーザ空間のアプリケーションへのインタフェースを提供する。三つ目はメディアマネージャである。この部品は、ウェアレベリングやガーベッジコレクションなどのメディア管理のアルゴリズムや書き込みバッファを実装する。カーネル空間あるいはユーザ空間に実装される。今回の実装では、NVMe デバイスド

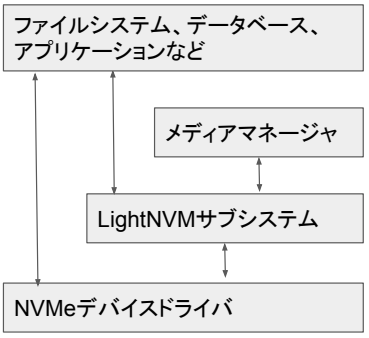

図 2　LightNVM フレームワーク

ライバにあたる部分を実装し、メディアマネージャに関しては既存の実装 (gennvm および rrpc) を使用した。

3.2 Olive における実装

元々の Olive のブロックデバイスドライバは、blk-mq と呼ばれるブロックデバイスのフレームワークを使って実装されている。blk-mq は、SSD のような高い IOPS を持つディスクでの利用を想定し、複数 CPU および複数のハードウェアキューに対して効率よく入出力を処理するためのフレームワークである [3]。LightNVM サブシステムは、ブロック I/O をデバイスドライバに対して渡してくる。そのため、blk-mq を使った実装を流用することが可能である。今回のわれわれの実装では Olive のディスクデバイスドライバを元に、LightNVM フレームワークへの登録に必要なデータ構造 (`struct nvm_dev_ops`) を追加で実装することで、LightNVM フレームワークへ組み込むことができた。

表 2 に各インタフェースの実装状況を示す。今回の実装では読み書き性能の計測を第一の目的とし、ブロック消去の実装は省略した。論物変換テーブルや不良ブロックテーブルは eMMC デバイスの内部に実装され、eMMC インタフェースに隠蔽されている。したがって、ホスト OS の機能として実現できないため、これらのテーブルの実装も省略している。I/O 要求の登録の実装は、LightNVM フレームワークから渡される要求を blk-mq に変換する処理となっている。

Olive に組み込まれた Linux カーネルのバージョンは 4.4 である。これは LightNVM フレームワークがメインストリームに取り込まれたバージョンである。LightNVM フレームワークとしてはカーネルバージョン 4.12 で pblk と呼ばれるメディアマネージャが新たに導入されている [1]。今回の評価は環境構築の手間を省くためカーネルバージョン 4.4 を採用している。バージョン 4.12 以降の評価については機会を改めたい。

3.3 アプリケーションとのインタフェースおよびアドレス空間

ユーザー空間へのデバイスの見え方は、ブロックインタフェースであっても LightNVM インタフェースであっても変化はなく、単一のデバイスファイルとしてアクセスできる。LightNVM インタフェースの場合には、ひとつのディスクに対して LightNVM デバイスとしてアクセスすることも NVMe デバイスとしてアクセスすることも可能である。LightNVM デバイスとしてアクセスする場合には、LightNVM サブシステムおよびメディアマネージャの処理を経由して NVMe デバイスを制御する。

OC-SSD を NVMe デバイスとしてアクセスするときには注意が必要である。従来のブロックインタフェースは LBA という仮想化されたアドレス空間がデバイスによって提供されていることが前提となっている。従来の SSD では、コントローラに内蔵された FTL がアドレス空間の変換を処理していた。これに対して、OC-SSD の仕組みでは、ホスト OS が論理アドレスと物理アドレス空間の変換を行うため、デバイスドライバは物理アドレス空間に対して処理を行う。これは、データの読み出し、書き込みおよび消去に関するフラッシュメモリの非対称性をホスト OS が考慮しなければならないことを意味している。NAND フラッシュメモリは、データを読み出す単位、書き込む単位、消去する単位それぞれが異なる。

各チップのブロック数が切りの良い数字とはならないため、物理ブロックがアドレス空間に連続して配置されるとは限らないことも注意が必要である。NAND フラッシュメモリのチャネルへ直接アクセスできることで、入出力処理の並列度が向上することが OC-SSD のメリットの一つだが、このような NAND フラッシュメモリの特性をホスト側で考慮する必要がある。

Olive の実装では、元々のブロックデバイスドライバが blk-mq インタフェースを実装していたこともあり、LightNVM に対して PPA 空間ではなく LBA 空間を提供している。そのため、本来想定される OC-SSD の特性は持たない。フラッシュメモリの非対称性はデバイスドライバや eMMC のインタフェースに隠蔽されている。

4. 評価実験

fio[4] を使って LightNVM の性能を評価するための実験を行った。実験環境としては Olive を用いた。まず、LightNVM 化することの性能への影響を計測した。次に、LightNVM の特徴であるハードウェアの構造を考慮したアクセスを模擬した実験を行った。ここでは、並列度を高めたアクセスを模擬した実験、およびアクセスに使用する入出力ポートの変更を模擬した実験を行った。

4.1 LightNVM 化の性能への影響

LightNVM 化することの性能への影響を計測した結果を、表 3, 4 および図 3, 4 に示す。読み出し性能に関しては、シーケンシャルアクセスにおいて LightNVM が若干スケールした結果となり、アクセスサイズ 128KB の時 5.7%性能が向上している。ランダムアクセスにおいてはほぼ変わりない結果となった。一方、書き込み性能はシーケンシャルアクセス、ランダムアクセスともに、LightNVM 化により大幅に性能が向上した。シーケンシャルアクセス時には、アクセスサイズ 4KB の時に 2.7 倍、64KB, 128KB の時も 2 倍の性能となった。ランダムアクセス時には、アクセスサイズ 128KB の時に 1.8 倍、4KB, 64KB の時に、それぞれ 1.6 倍，1.5 倍の性能となった。

表 2　nvm_dev_ops インタフェース

名前	型	概要	Olive での実装
identity	nvm_id_fn	デバイスの設定値取得	✓
get_l2p_tbl	nvm_get_l2p_tbl_fn	論物変換テーブルの取得	-
get_bb_tbl	nvm_op_get bb_tbl_fn	不良ブロックテーブルの取得	-
set_bb_tbl	nvm_op_set_bb_tbl_fn	不良ブロックテーブルの設定	-
submit_io	nvm_submit_io_fn	I/O 要求の登録	✓
erase_block	nvm_erase_blk_fn	指定ブロックの消去	-
create_dma_pool	nvm_create_dma_pool_fn	データ転送用メモリプールの確保	✓
destroy_dma_pool	nvm_destroy_dma_pool_fn	データ転送用メモリプールの開放	✓
dev_dma_alloc	nvm_dev_dma_alloc_fn	メモリプールからの動的確保	✓
dev_dma_free	nvm_dev_dma_free_fn	メモリプールへの返却	✓
max_phys_sect	unsigned int	1 回あたりの最大不揮発化セクタ数	✓

表 3　標準のブロックデバイス経由の場合と LightNVM 化した場合の性能比較 (Read)

	4KB	64KB	128KB
Blk Seq. Read (MB/s)	42.620	165.325	128.632
Lnvm Seq. Read (MB/s)	42.776	165.234	210.009
Blk Rand. Read (KIOPS)	9.833	4.584	2.760
Lnvm Rand. Read (KIOPS)	10.154	4.367	2.741

表 4　標準のブロックデバイス経由の場合と LightNVM 化した場合の性能比較 (Write)

	4KB	64KB	128KB
Blk Seq. Write (MB/s)	5.9816	15.719	24.854
Lnvm Seq. Write (MB/s)	16.205	32.212	51.138
Blk Rand. Write (KIOPS)	2.839	0.553	0.449
Lnvm Rand. Write (KIOPS)	4.516	0.813	0.825

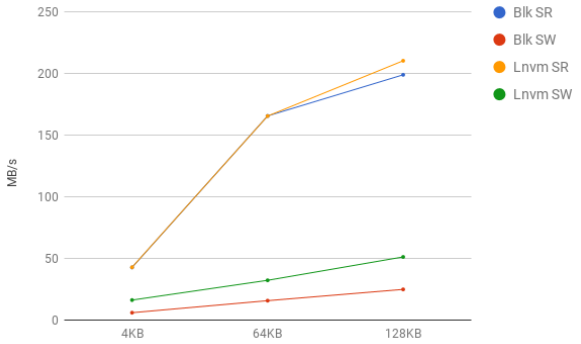

図 3　標準のブロックデバイス経由の場合と LightNVM 化した場合のシーケンシャル性能の比較

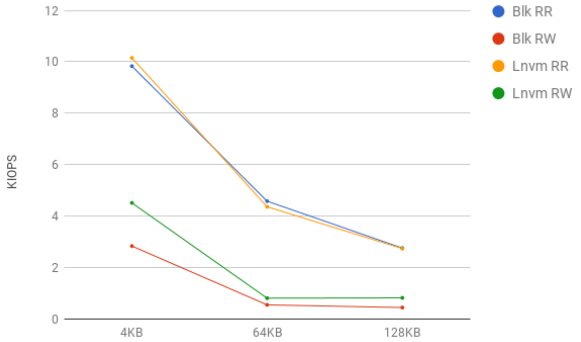

図 4　標準のブロックデバイス経由の場合と LightNVM 化した場合のランダム性能の比較

4.2　LightNVM における並列アクセスを模擬した評価

LightNVM の特徴は、ハードウェアの構造を考慮したアクセスにある。Olive は eMMC コントローラ数よりも eMMC が接続されたバスの方が多く、また複数の eMMC がバスには接続されている。そのため、eMMC コントローラの不足および 1 つの eMMC バスへのアクセスの衝突が起こりうる。そこで、理想的な条件を模擬するため、eMMC コントローラ数と eMMC が接続されたバスを等しくし、また各 eMMC バスには 1 つだけ eMMC が接続された状態を初期設定として実験を行った。ここで、eMMC へのアクセスの並列度を変化させるために、インタリーブサイズを 4KB から 128KB まで変更して、シーケンシャルにアクセスする実験を行った。実験結果を、図 5, 6 に示す。デフォルトのインタリーブサイズは 2MB であったため、その結果も合わせて図中に示した。

読み出し性能に関しては、アクセスサイズ 4KB ではインタリーブサイズの影響はほぼ無い結果となった。アクセスサイズ 64KB, 128KB では、インタリーブサイズが 4KB, 8KB の時よりも、16 128KB での性能の方が高くなっている。それぞれインタリーブサイズが 16KB, 32KB の時に最も高い性能を示している。デフォルトのインタリーブサイズ 2MB では、非常に低い性能となっている。これは、並列度が低いアクセスが行われた結果であると考えられる。

書き込み性能に関しては、アクセスサイズ 4KB では、インタリーブサイズが 8KB の時に最も性能が高く、インタリーブサイズが大きくなるにつれ性能が低下している。一方、アクセスサイズ 64KB, 128KB では、インタリーブサイズが 128KB の時に最も性能が高い結果となった。特に、アクセスサイズ 128KB でのインタリーブサイズが 128KB の時の性能は突出した結果となった。

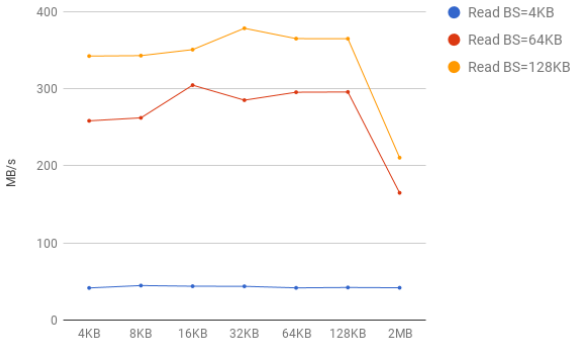

図 5 LightNVM における異なる並列度での読み出し性能の比較

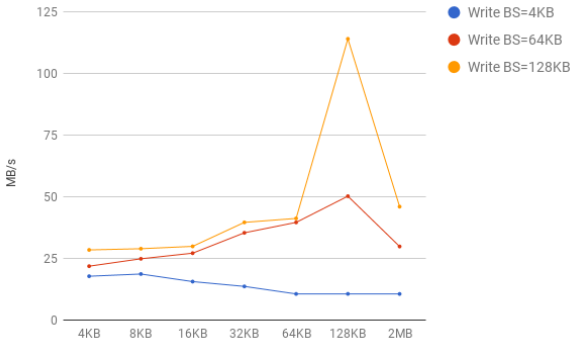

図 6 LightNVM における異なる並列度での読み出し性能の比較

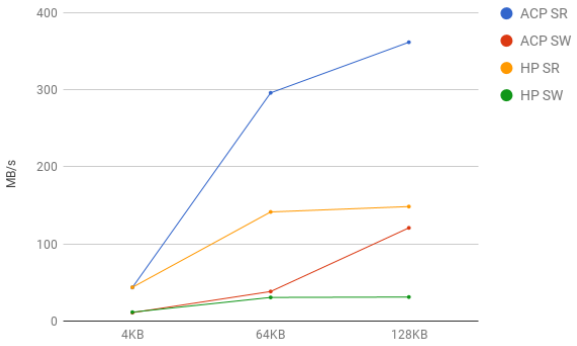

図 7 LightNVM における異なる性能の入出力ポートでの性能の比較

4.3 LightNVM における異なる性能の入出力ポートの使用を模擬した評価

Olive の eMMC コントローラが接続されている Zynq の入出力ポートには、ACP と HP の 2 種類がある (図 1)。ACP はキャッシュコヒーレントなポートであるため、アクセス時にキャッシュフラッシュを行う必要が無いため、より高性能なアクセスが可能である。そこで、必要に応じて最適なポートを選択することを想定し、入出力ポートの違いによる性能への影響を計測した。実験は、ACP を 4 つのみを使用した場合、および ACP を 1 つと HP を 4 つ使用した場合で行った。eMMC が接続されたバスは 4 つ、インタリーブサイズは 128KB とした。計測結果を図 7 に示す。

アクセスサイズ 4KB では差がないものの、64KB, 128KB では大きく性能差がある。アクセスサイズ 128KB では、ACP は HP よりも、読み出しで 2.4 倍、書き込みで 3.9 倍の性能となった。この結果から、ACP を用いた時の性能が高いことがわかる。従って、そのアクセスがファイルアクセス性能全体に影響を及ぼす、メタデータやジャーナリングのアクセスには ACP を優先して割り当てることで、ファイルアクセス性能を向上できる可能性がある。

5. 考察

今回の取り組みでは、LightNVM の本来の想定からは外れた使い方となっている部分がある。本来の LightNVM は OC-SSD を管理対象としており、NAND チップのチャネルへ直接アクセスできることが想定されている。Olive の場合、フラッシュメモリの物理的な要素は容量をのぞいて eMMC の FTL によって LBA 空間に隠蔽されている。したがって、LightNVM が最高の性能を出せる実装とはなっていないと思われる。一方で、入出力の並列化は LightNVM の特徴の一つである。Olive のハードウェアを調整して eMMC アクセスの並列度を変化させることで入出力の性能が変化する実験結果を得られており、LightNVM が並列度を活かすことを目的とした実装となっていることを確かめられた。

Olive のように大量のディスクを使用するのであれば、device-mapper で管理するのがより正しいアプローチと言える。しかしながら、device-mapper は HDD を前提とした設計となっており、SSD の制御には最適とは言えない。

device-mapper は元来論理ボリュームに対する I/O を物理デバイスに対する I/O にマッピングする機能を提供する。開発者はターゲットと呼ばれるマッピングアルゴリズムを実装することで、論理ボリュームと物理ボリュームのマッピング方法をカスタマイズすることが可能である。この仕組みを利用することで、ディスク I/O の並列性に関しては LightNVM と同等の性能が得られると考える。しかしながら、device-mapper 自体は仮想的ブロックデバイスであるため、その入出力要求は、ユーザープロセスから device-mapper へブロック I/O を使って到達した後、device-mapper からブロックデバイスドライバへ同じ仕組みを使って配送される。HDD ではこの配送オーバーヘッドはディスクのシークタイムに隠蔽されていたが、SSD の短いレイテンシでは隠蔽されず SSD の性能を低下させる要因となる可能性がある。

eMMC では FTL が動作しているとは言っても、媒体としてはフラッシュメモリであり、長期的な運用の視点で考えれば信頼性の低下の問題はつきまとう。device-mapper のようなフレームワークではなく LightNVM フレームワークを用いることにより、グローバルなガーベッジコレクションやウェアレベリングの既存のアルゴリズムを使用できることは魅力であると考える。

6. おわりに

本稿では、Olive に搭載されている SSD を LightNVM フレームワークで管理する方法について調査した。調査のために従来のブロックデバイスドライバを元に、LightNVM に対応したデバイスドライバを試作した。LightNVM は OC-SSD 向けのフレームワークではあるが、blk-mq を実装したブロックデバイスドライバであれば、容易に LightNVM 化することができるという知見を得た。今回の実験では、LightNVM 化した場合は、オリジナルのブロックデバイスドライバに対して書き込み性能が向上するという結果を得られた。また、インタリーブサイズやバスを調整することで性能を改善できることを示した。

参考文献

[1] Bjørling, M., Gonzalez, J. and Bonnet, P.: LightNVM: The Linux Open-Channel SSD Subsystem, *Proceedings of the 15 the USENIX Conference on File and Storage Technologies (FAST '17)*, (2017).
[2] 追川修一, 中村孝史, 飯塚拓郎, 三木聡: Solid State Server "Olive" における eMMC および NIC 高速化, 2016 年並列／分散／協調処理に関する『松本』サマー・ワークショップ (SWoPP2016), (2016)
[3] Bjørling, M., Axboe, J., Nellans, D. and Bonnet, P.: Linux Block IO: Introducing Multi-queue SSD Access on Multi-core Systems, *Proceedings of the 6th International Systems and Storage Conference (SYSTOR '13)*, (2013).
[4] Axboe, J.: Flexible I/O tester, 入手先 ⟨https://github.com/axboe/fio⟩.