Vol.2016-ARC-222 No.14
2016/10/6

LinuxにおけるI/Oプロトコルスタックの改良動向と共有ストレージ装置における性能評価

佐藤 功人[1,a)]　近藤 雄樹[1,b)]　清田雄策[1,c)]

概要：データの保存に用いられる記憶媒体がディスク装置 (HDD) からフラッシュメモリ (SSD) へと移行し，Linux OS におけるストレージ装置のプロトコルスタックも従来の SCSI 規格に加えて SSD に特化した NVMe(Non-Volatile Memory Express) 規格が導入された．NVMe デバイスドライバでは，従来存在したマルチコア CPU で生じるコア間の排他制御競合によるボトルネックが解消されている．本報告では，高性能共有ストレージ装置へのアクセスインターフェースとして NVMe 規格を適用することで，ローカルの SSD と同様にボトルネックが解消することを定量的に示す．また，初期の NVMe デバイスドライバの実装方法では運用上のデメリットが発生するが，運用性を損なわずボトルネックを解消するために後に Linux に導入された，ブロックマルチキューについて述べる．

1. 緒言

1.1 背景

フラッシュメモリの大容量化により，Hard Disk Drive (HDD) に変わる補助記憶媒体として Solid State Drive (SSD) が普及している．SSD は HDD に比べて 3 桁小さい遅延時間と高いバンド幅を特徴としているが，既存の HDD 向けのインターフェースである SATA[*1] 規格や SAS[*2][1] 規格のデータ転送速度では，SSD の能力を最大限発揮することは難しい．この状況を打開するため，**図 1**(a) のようにストレージコントローラチップを経由することなく，直接ホストの PCIe [*3] バスに SSD を接続する形態がとられるようになってきた．この接続形態にすることで SAS や SATA の速度限界に律速されることなく，PCI Express の転送速度でアクセスでき，遅延時間の縮小と帯域幅の向上という両方の効果を得ることができる [2].

PCIe で直結された Flash Memory Controller には，従来の SAS や SATA とは異なる Non-Volatile Memory Express(NVMe)[3] というプロトコルを用いてアクセスする．NVMe は SSD に特化した高性能ストレージ向けのプロトコル規格であり，従来の SAS や SATA に比べて効率の良いインターフェースを採用している．

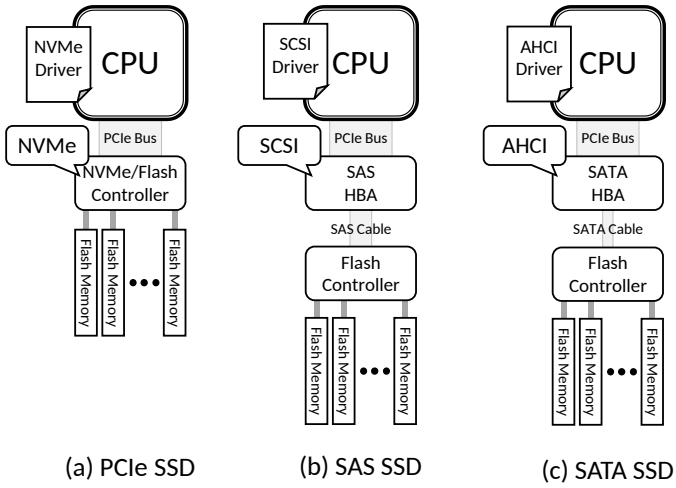

図 1　ストレージ装置への接続インターフェースとプロトコル種別

表 1　データ転送速度比較

Interface	Generation	Speed
SATA	Rev. 3.0	0.6 GB/s
SAS	Rev. 3.0	1.2 GB/s
PCIe	Gen. 3	4.0 GB/s (x4)

1.2 目的

本報告では，Linux の従来の I/O プロトコルスタック上のボトルネックについて示し，NVMe を採用することでこのボトルネックが回避可能であることを述べる．NVMe を共有ストレージ装置との接続インターフェースに採用する

[1] 株式会社 日立製作所
Hitachi, Ltd. Tokyo, Japan.
[a)] katsuto.sato.xa@hitachi.com
[b)] yuki.kondo.fe@hitachi.com
[c)] yusaku.kiyota.hp@hitachi.com
[*1] Serial Advanced Technology Attachment
[*2] Serial Attached Small Computer System Interface
[*3] Peripheral Component Interconnect Express

ことで，このボトルネックに起因するサーバ側のオーバーヘッドを削減可能であることを定量的に示す．また，このボトルネックを根本的に解消するために進められているLinux カーネルの改良動向について述べる．

本報告は以下のように構成されている．第 2 章では，SCSI と NVMe のプロトコルスタックの実装が Linux OS 内で異なることと，Linux カーネルの従来の I/O プロトコルスタックに内在するボトルネックについて指摘する．第 3 章では，共有ストレージ装置のインターフェースとして NVMe を採用することによって，ボトルネックを回避し，改善される性能を定量的に評価する．第 4 章では，ボトルネック解消のために Linux 3.13 以降で採用されるブロックマルチキューについて述べる．第 5 章では，全体のまとめを述べる．

2. Linux の I/O プロトコルスタック概要

2.1 I/O 処理のながれ

Linux のような OS では，データベース等のユーザプログラムが I/O を行う場合，以下のいずれかの方法を用いる．

(1) POSIX 準拠の API[*4]
(read/write/mmap など)
(2) Linux AIO の非同期 I/O API
(io_submit, io_getevents など)
(3) デバイス毎に固有のカーネルバイパス API
(Intel DPDK[4], SPDK[5] 等)

これらの中で (1)(2) の場合，**図 2** に示す経路を通って I/O 要求がデバイスドライバに対して伝達される．はじめにシステムコールによって伝達された I/O 要求は，カーネル内部の仮想ファイルシステム層に到達し，ファイルシステムとしての処理を経た結果として，ブロック I/O アクセス [*5] へと変換される．このとき，読み書きに関する要求が記述された bio オブジェクトが生成され，次の層に渡される．

ファイルシステム層から発行された bio オブジェクトは，特定の機能を有効化している場合には bio-based デバイスマッパー層に渡される．この層では主にソフトウェア RAID[*6] としての処理を担当しており，1 つの bio オブジェクトに記載された読み書き要求を，RAID の運用状況に合わせて 1 つ以上の bio オブジェクトに変換して次の層に渡す．ソフトウェア RAID のような機能を使用していない場合にはこの層はバイパスされる．

NVMe のデバイスドライバは，この段階で bio オブジェクトを直接受け取り，ストレージ装置へと送信する NVMe

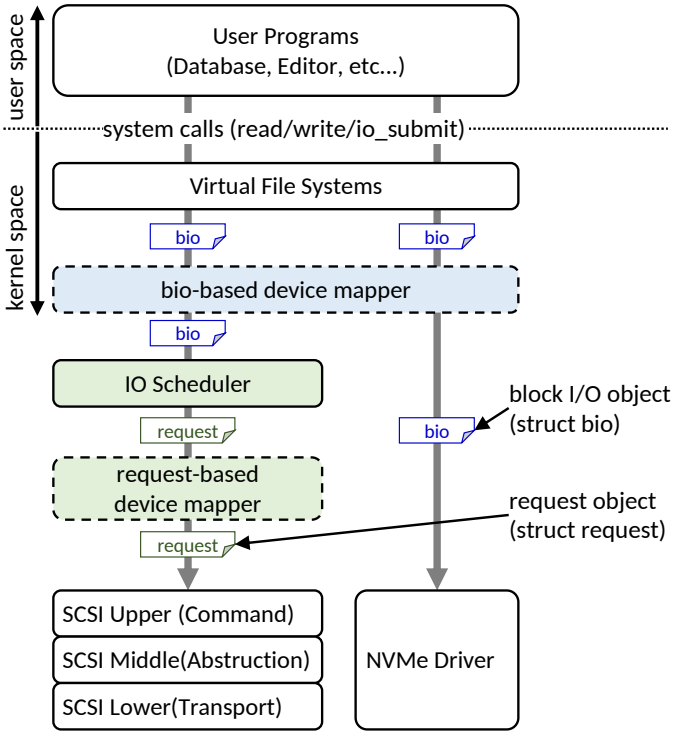

図 2 Linux の I/O プロトコルスタック概要図

規格に従ったコマンドに変換し，ストレージ装置のレジスタに書き込むことで I/O 要求の発行処理を完了する．

SCSI の場合は，I/O スケジューラが bio オブジェクトを受け取る．I/O スケジューラはランダムアクセスに弱いディスク装置に対して，I/O 要求の発行順序を調整する役割を持った層である．**図 3** に I/O スケジューラの概要を示す．I/O スケジューラはいくつかの bio オブジェクトを包含可能な request オブジェクトを生成し，当該オブジェクトをリクエストキューにつなぐ．リクエストキューにつながれた request オブジェクトには，スケジューリングポリシーに従ってリクエスト発行順の並び替えや，隣接するディスクオフセットへアクセスする request オブジェクトとのマージが適用される．I/O スケジューラを通ることで，多数の小規模ランダムアクセスが，数が減少した中規模のシーケンシャルアクセスへと変換される．このようなスケジューリングを経ることで，物理的なヘッドの動作によってランダムアクセス性能が制限されるディスク装置へのアクセスを最適化している．

I/O スケジューラから渡された request オブジェクトは，特定の機能を有効化している場合には request-based デバイスマッパー層に渡される．この層ではマルチパス機能やブロックデバイスの仮想化機能を提供しており，Linux における LVM[*7] 機能はこの層の機能を用いることで実現されている．この層では，1 つの request オブジェクトを受け取り，マルチパスや仮想化の設定に従って 1 つ以上の request オブジェクトを生成する．マルチパス機能や LVM

[*4] Application Programming Interface
[*5] 一定のサイズ毎にまとめて記憶領域の任意の位置に対して読み書きする方式．512bytes または 4096bytes 単位でアクセスすることが多い．
[*6] Redundant Arrays of Independent Disks
[*7] Logical Volume Manager

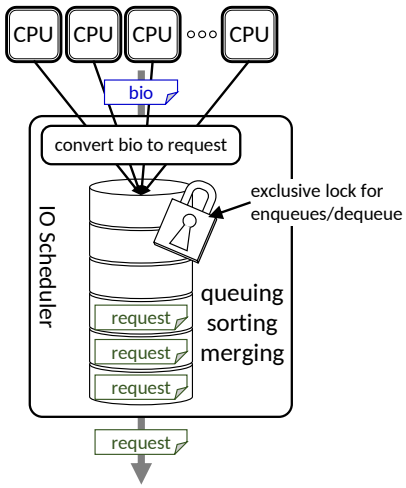

図 3 I/O スケジューラの役割

機能を用いていない場合にはこの層はバイパスされる．

SCSI のデバイスドライバは，request オブジェクトを受け取り，SCSI プロトコルの 3 階層の処理を経てストレージ装置のレジスタに命令を書き込むことで，I/O 要求の発行処理を完了する．

デバイス毎に固有のカーネルバイパス API を用いる (3) の手段は，以上に挙げたパスを通らずに，直接デバイスのレジスタに対してユーザ空間のライブラリからアクセスする手段を提供する．そのため，低遅延時間・低 CPU 負荷でストレージ装置にアクセスできる利点があるが，ストレージ装置そのものの仕様がそのまま見えることになるため，アプリケーションやミドルとウェア側でストレージ装置の構成の違いを吸収しなければならない．このような手段は NAS[*8] 制御ソフトウェアや分散ファイルシステムなどの用途で使われており，一般的なアプリケーションやミドルウェアで使われることはまれである．

2.2 I/O スケジューラにおけるボトルネック

I/O スケジューラは，ディスク装置に対して I/O 要求の発行状況を最適化するために効果的な手段である．しかし，物理的なヘッドの移動を伴わずにランダムアクセス可能な SSD に対しては，I/O 順序の並び替えは効果が薄く，アクセス遅延時間の増大と CPU 時間の消費といったデメリットが見えてくる．

request オブジェクトの並び替えやマージを行うためには，リクエストキューから 2 つ以上のオブジェクトを取り出し，処理を行った後に再びリクエストキューに戻す手順が必要になるが，リクエストキューに対して矛盾無くオブジェクトの追加／削除を行うためには排他制御が必要となる．Linux ではスピンロックを用いてこの排他制御を行っており，CPU コアがリクエストの追加／削除を行う場合には必ずロックを取得してから操作することで矛盾の発生を防止している．

この排他ロックはリクエストキューに対して 1 つ存在し，Linux kernel 3.10 の実装ではリクエストキューは 1 つの記憶領域 [*9] に対して 1 つしか存在しない．すなわち，1 つの記憶領域に対して複数の CPU が同時にアクセスしようとすると，CPU コア間で排他制御に起因する競合が発生することになる．

この競合は同時に同じ記憶領域にアクセスしようとする CPU コアが多いほどオーバーヘッドが大きくなるため，計算機システムに搭載されている CPU コア数が多いほど，または利用可能な記憶領域の数が少ないほど競合オーバーヘッドが多くなる性質がある．また，CPU ソケットをまたいだ CPU コア間で排他ロック競合が発生すると，CPU 間インターコネクト (QPI[*10] 等) を経由したアクセスとなるためオーバーヘッドはさらに大きくなる [6]．

このような単一キューに対する複数 CPU コアによるアクセス競合の問題は，プロセススケジューラでは以前から問題になっており [7]，改良が図られてきた．しかし，I/O スケジューラの対応は，HDD の性能の低さと I/O 要求並び替え効果の利点が大きかったために SSD の登場までは大きな問題にはなっていなかった．

NVMe デバイスドライバはこの競合の発生を，I/O スケジューラを経由せずに I/O 要求をストレージ装置に伝える経路を用いることで回避している．I/O スケジューラをスキップする効果を定量的に評価するためには，同一の性能を持つストレージ装置に対して，I/O スケジューラを経由する経路を通る SCSI プロトコルと，I/O スケジューラを経由しない経路を通る NVMe プロトコルの性能を比較することが有効である．

次節では，同じストレージ装置に対して SCSI と NVMe の 2 種類のプロトコルでアクセス可能な環境を構築し，本節で述べたプロトコルの実装による差を定量的に評価する．

3. 性能評価

本節では，高性能共有ストレージ装置へのアクセスインターフェースとして NVMe を用いることで，SCSI を用いた場合に存在するボトルネックを回避し，サーバの CPU 使用率を削減できることを定量的に評価する．

3.1 評価環境

本評価では，**図 4** に示す構成の装置を用いて行う．

同一ストレージ装置に SCSI と NVMe の両方でアクセスできるようにするため，両方のプロトコルインターフェースを持ち，ストレージ装置と PCIe バスで直結可能なフロントエンドコントローラを用いた．

ストレージ装置は，フロントエンド側にサーバ直結用の

[*8] Network Attached Storage
[*9] SCSI における LU，NVMe の Namespace に相当
[*10] Quick Path Interconnect

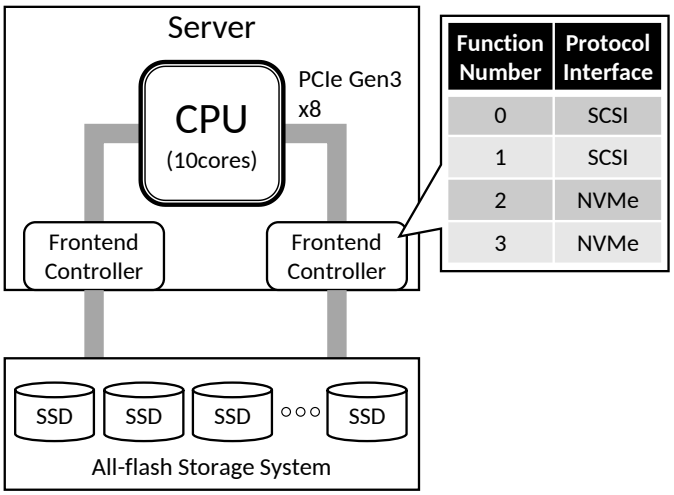

図 4 評価用実験機のハードウェア構成図

表 2 評価用実験機の仕様一覧

項目	仕様
CPU	Intel Xeon E5-2690 v3 (2.60GHz, 1socket, 10cores, SMT)
メモリ	16GB (8GB DDR4-2133MHz × 2)
OS	RHEL 7.1.1503 (Linux kernel 3.10.0-229)
記憶領域数	192
ストレージ装置 (メモリ) (記憶媒体)	 256GB DRAM Cache, SAS SSD × 48
接続経路	PCIe Gen3 x8 2 組

コントローラを接続できるようになっており，バックエンド側にボトルネックにならないだけの十分な台数の SSD が SAS で接続されている．

このほか，サーバ側のハードウェア・ソフトウェア仕様を**表 2** に示す．

本実験は一定のコマンドスループット (IOPS[*11]) でストレージ装置に対して負荷をかけることができるツールを用いて行った．接続経路の最大データスループットは 12.8GB/s であり，I/O ブロックサイズはデータベースミドルウェアで一般的に想定されるアクセスサイズである 8KiB としたため，理論最大コマンドスループットは

$$\frac{12.8\ \mathtt{GB/s}}{8\ \mathtt{KiB/IO}} = 1600\ \mathtt{kIO/s}$$

である．今回の測定では最大で約 1300kIO/s(接続経路の帯域使用率約 80%) までの設定で計測しており，接続経路がボトルネックとなる条件下で計測している．ストレージ装置の性能限界よりも低い条件を設定しているため，ストレージ装置性能によらない結果を得ることができる．

記憶領域数は 192 とした．前章で述べた効果を見るためには，記憶領域数をできる限り少なくすることが望ましいが，極端な環境を想定して実環境とかけ離れた状態を観測しても実用上の意味は薄い．そこで，この規模のストレージ装置で I/O 性能を評価する場合に一般的である記憶領域数を選択する．

OS には RHEL[*12] 7.1 を使用し，非仮想化環境にて測定する．

[*11] Input/Output command per second

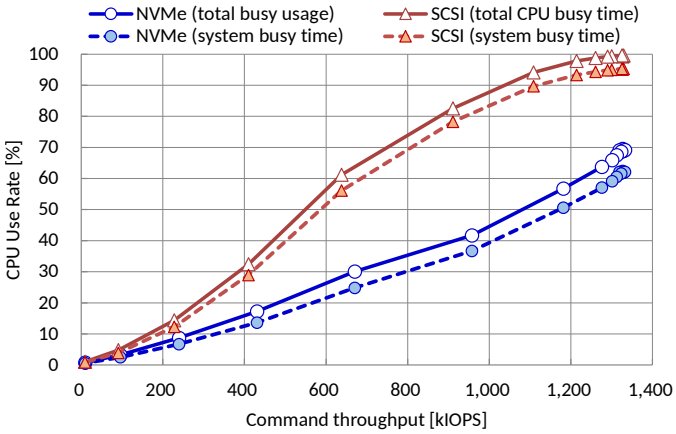

図 5 CPU 使用率の比較

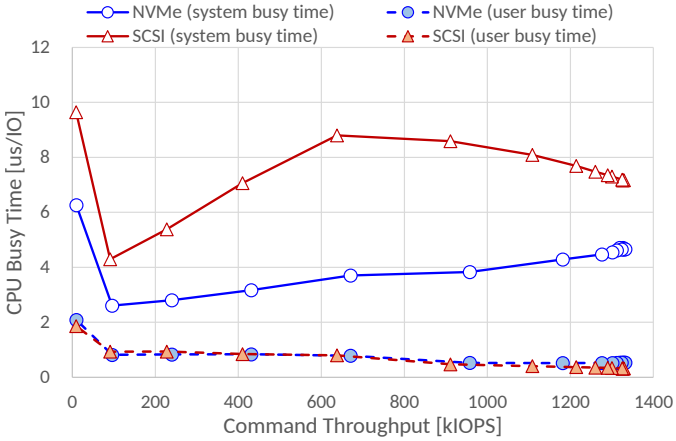

図 6 I/O あたりの CPU ビジー時間の比較

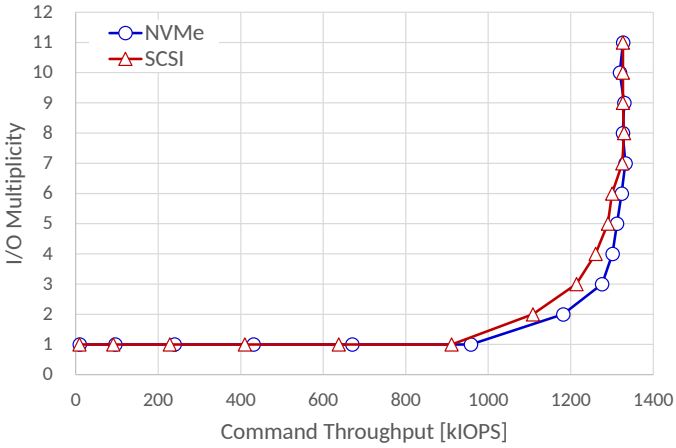

図 7 I/O 多重度と IOPS 性能の関係

3.2 評価結果

図 5 は，SCSI と NVMe を用いて I/O を行った場合の CPU 使用率を示している．実線は CPU のアイドル時間を除く全ビジー時間の割合を示し，破線は Linux カーネルが消費した時間を示している．SCSI は同じ IOPS 性能を出すために必要とする CPU 使用率が NVMe よりも約 2 倍高い結果となった．

この違いは IO あたりの CPU ビジー時間を示す**図 6** を見るとさらに顕著であり，SCSI では負荷が上がるにつれて CPU ビジータイムが単位時間あたりの IO 数に応じて上昇するが，NVMe ではビジータイムの増加が小さいという状況が見られる．

SCSI のビジータイムの増加は，SCSI の論理記憶域 (Logical Unit) にアクセスする際に，I/O 要求をリクエストキューにつなぐための排他ロックを複数 CPU コアで取り合う競合が発生することによる．NVMe はリクエストキューを経由せずに I/O 要求が発行されるため，ロック競合によるビジー時間の増加が少ないことがこの差につながっている．また，I/O 要求を出しているユーザプログラムが消費している CPU 時間はプロトコルによらずほぼ一定であり，性能の差が出る要因に関与していないことがわかる．

図 6 では，1000kIOPS を境界として，より高い IOPS 性能を得るために，**図 7** に示す様に 1 回のシステムコールで発行する IO 要求数 (以降，I/O 多重度と呼ぶ) を 2 以上に段階的に増加させている．そのため，SCSI の I/O あたりの CPU ビジー時間は 600〜900kIOPS をピークとして，CPU ビジータイムが低下しているように見える．このように，多重度を上げることでボトルネックの影響を軽減することができるが，影響を無視できるほどにするためには高い I/O 多重度が必要となる．しかし，I/O 多重度はアプリケーションやミドルウェアの処理の仕方によって決まるため，ストレージプロトコルスタック側の事情によって決められるものではない．I/O 多重度が 2〜4 の範囲であれば，NVMe は最大で約 80 kIOPS 高い性能が得られるため，I/O 多重度が低い，逐次処理が多いソフトウェアでは当該ボトルネックを回避する利点が大きいと言える．

3.3 考察

評価結果から示されたとおり，I/O スケジューラ層に存在するリクエストキューの排他ロックを複数 CPU コア間で取得するために発生する競合関係は CPU ビジー時間に無視できない影響を及ぼす．この競合が発生した場合，Linux の排他ロック機構 (spinlock) はビジーループで待つように実装されているため，排他ロックの解放を待つためだけに CPU 時間を消費することになる．

この影響はリクエストキューを取り合うコア数が多くなるほど顕著になるため，システムで利用可能な CPU コア数とリクエストキューの本数の比で決定されることになる．I/O スケジューラ層に存在するリクエストキューは SCSI における LU 毎に 1 本存在するため，マルチコア CPU を搭載した計算機システムにおいて排他競合確率を下げるためには，多数の LU をストレージ装置側に用意し，それらの LU 群に対してインターリーブアクセスをする必要がある．このような LU 数を増加させる最適化手法は，高性能ストレージ装置向けのストレージアクセス性能を向上させる最適化技法として一般的に用いられている．

一方で NVMe のドライバスタックは I/O スケジューラ層を経由することなく，I/O 要求元のユーザプログラムからデバイスにコマンドが発行されるまでの間に，他 CPU コアと共有している資源が存在せず，排他ロックが介在しない．このため，CPU コア数が増加し，ストレージ装置の性能が上がって I/O 負荷が増大したとしても，CPU のビジー時間が増加しにくい設計になっている．すなわち，計算機システムに搭載されている CPU のコア数が多く，高性能なストレージ装置と接続されるエンタープライズ用途向けのサーバには適した性質であるといえる．また，高いアクセス性能を持つ PCIe 接続の SSD に対しても好ましい性質であるといえる．

4. Linux における I/O プロトコルスタックの改良動向

4.1 NVMe 方式実装の利点・欠点

前章で示したように，従来の I/O スケジューラの特性は HDD に対しては適している一方で，マルチコア CPU には不適である．NVMe は I/O スケジューラをスキップすることでオーバーヘッドの発生を回避しているが，この方式では request-based デバイスマッパーが提供する機能を用いることができなくなり，以下に示す利点・欠点が発生することになる．

利点
- CPU コア数の増加に対する I/O 発行性能のスケーラビリティが良い
- 記憶領域数を増加させる最適化が不要 (性能最適化の手間がかからない)

欠点
- Logical Volume Manager と連携できない
- OS 標準のマルチパス機能が使えない
- OS 標準のシン・プロビジョニングが使えない
- SSD に最適化された I/O スケジューラオプションを利用できない
- そのほか，request-based デバイスマッパーに将来実装される機能と連携できない

*12 Redhat Enterprise Linux

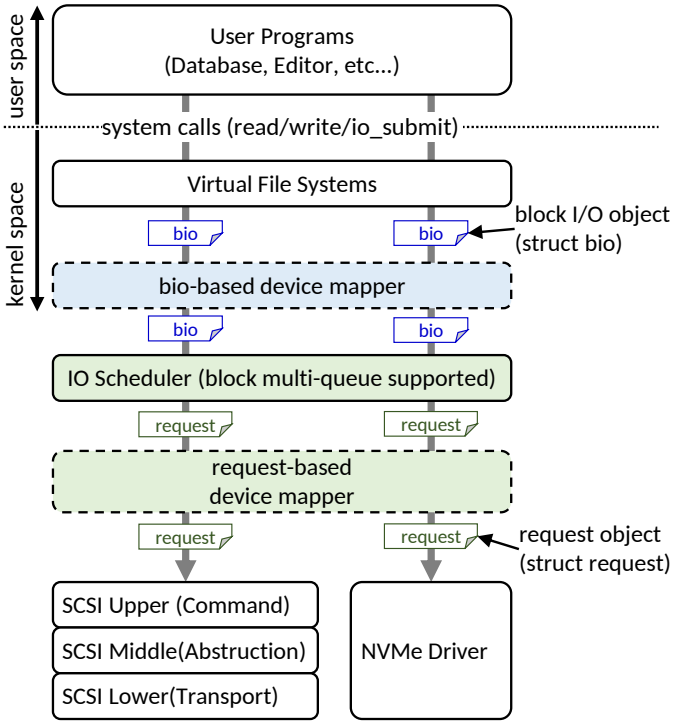

図 8 Linux の I/O プロトコルスタック概要図
(ブロックマルチキュー対応)

NVMe が DAS[*13] のインターフェースとして使用され，かつキャッシュ領域として利用されている場合には欠点はあまり問題にならない．しかし，データ領域のインターフェースとして利用する場合には，LVM やシン・プロビジョニングと連携できないことが運用上の問題を招くことになる．また，共有ストレージ装置とのインターフェースとして利用する場合には，OS 標準のマルチパス機能と連携できないことが高可用が求められるシステムでは無視できない欠点となる．さらに，SSD 内部の並列性に着目して I/O コマンドのスケジューリングを行う手法 [8] も提案されているが，当該手法と組み合わせることもできない．

4.2 ブロックマルチキューの導入

前述のような性能と利便性のトレードオフを解決するために，Linux kernel 3.13 からブロックマルチキュー [6] が導入された．このブロックマルチキューの導入に伴い，**図 8** に示す様に Linux kernel 3.19 から NVMe デバイスドライバも request オブジェクトを受け取るように修正されている．なお，SCSI デバイスドライバも Linux kernel 3.17 からブロックマルチキューに対応している [9].

ブロックマルチキューでは，**図 9** に示す様に I/O スケジューラに存在していたリクエストキューを，CPU のコア数分生成されるソフトウェアキューと，ハードウェアが並列に扱えるコンテキスト数分生成されるハードウェアキューに分離している．ソフトウェア側のキューは CPU

[*13] Direct Attached Storage

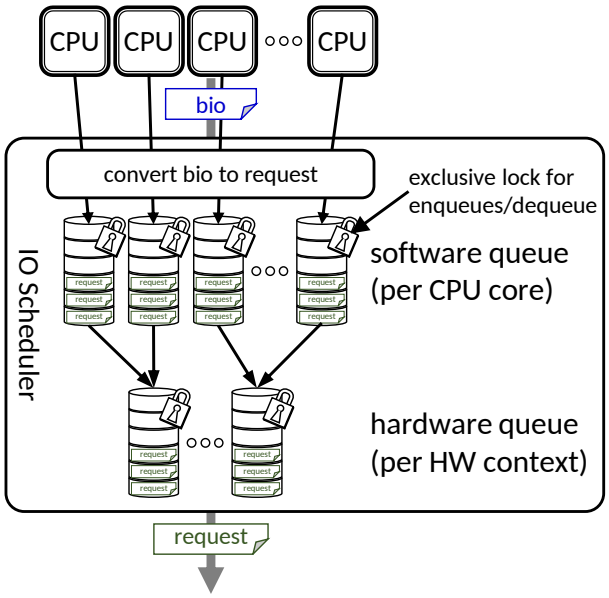

図 9 I/O スケジューラの役割

コア数分用意されているため，原理的に CPU コア間の排他ロック競合が発生せず，前述までのスケーラビリティが改善される．ハードウェアキューが十分な数存在する場合には，ソフトウェアキューと 1 対 1 で対応づけられるため，こちらも排他ロック競合が発生しない．ハードウェアキューがソフトウェアキューよりも少ない場合には複数 CPU コア間での排他ロック競合が発生するが，この排他競合は本質的なものであり，ストレージ装置の並列処理能力の限界に応じて必要となる排他ロックの競合オーバーヘッドであるため，ソフトウェアアーキテクチャで解決する範囲外の課題である．

この構造を採用することで，従来の request-based デバイスマッパーの機能を使いつつ，リクエストキューの排他ロック競合でスケーラビリティが制限される問題が解決される．性能傾向は本報告の NVMe の測定結果に近いものとなり，CPU コア数の増加によらず IO 処理にかかる CPU 時間の，IOPS 性能に対する増加は抑制されたものになることが予想される．ただし，bio オブジェクトを直接受け取っていた場合に比べるとデバイスドライバに達するまでのダイナミックステップ数が増加するため，性能は若干低下するものと考えられる．

5. 結言

本報告では，Linux OS に新しく導入された NVMe デバイスドライバの実装においてストレージプロトコルスタックのボトルネックを回避していることに着目し，高性能共有ストレージ装置へのアクセスプロトコルとして NVMe を使うことによって当該ボトルネックに起因するオーバーヘッドを削減可能であることを示した．評価結果では，同じ I/O 性能を達成するための CPU 使用率が最大で 50%低

下することを示し，マルチコア CPU を搭載した計算機システムに適した特性を備えていることを述べた．また，報告したボトルネックを解消するために Linux に導入されたブロックマルチキューの効果について議論した．

今後の取り組みとして，本報告と同様の評価を，ブロックマルチキューが導入された Linux OS で実行し，ブロックマルチキュー対応の SCSI プロトコルの性能傾向が改善されていることを検証したいと考えている．今回用いた Linux ディストリビューションの一つである RHEL 7.1 では従来の I/O プロトコルスタックが用いられているが，RHEL 7.2 からはブロックマルチキューに対応したデバイスドライバが導入されているため，第 4 章で述べた効果の定量的な比較評価を行うことが可能である．

参考文献

[1] SCSI Trade Association: STA SCSI Trade Association. http://www.scsita.org/.
[2] Xu, Q., Siyamwala, H., Ghosh, M., Suri, T., Awasthi, M., Guz, Z., Shayesteh, A. and Balakrishnan, V.: Performance Analysis of NVMe SSDs and Their Implication on Real World Databases, *Proceedings of the 8th ACM International Systems and Storage Conference*, SYSTOR '15, New York, NY, USA, ACM, pp. 6:1–6:11 (online), DOI: 10.1145/2757667.2757684 (2015).
[3] NVM Express, Inc.: NVM EXPRESS. http://www.nvmexpress.org.
[4] Intel: Data Plane Development Kit (2014). http://dpdk.org.
[5] Intel: Storage Performance Development Kit (2014). http://01.org/spdk.
[6] 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, New York, NY, USA, ACM, pp. 22:1–22:10 (online), DOI: 10.1145/2485732.2485740 (2013).
[7] 谷口宏樹，石川　裕，平木　敬：SMP 環境における Linux スケジューラの評価，システムソフトウェアとオペレーティング・システム（OS），No. 60(2002-OS-090), 情報処理学会 (2002).
[8] 奥村開里，小林諒平，吉瀬謙二：SSD の並列性を引き出す I/O スケジューラ，システムソフトウェアとオペレーティング・システム（OS），No. 14, 情報処理学会 (2015).
[9] Fischer, W.: Linux Multi-Queue Block IO Queueing Mechanism (blk-mq). https://www.thomas-krenn.com/en/wiki/Linux_Multi-Queue_Block_IO_Queueing_Mechanism_(blk-mq).

他社商標

- Linux は Linus Torvalds 氏の米国および他国における商標である．
- NVMe および NVM Express は NVM Expressm Inc. の商標である．
- PCIe および PCI Express は PCI-SIG の商標である．
- Xeon は米国および他国における Intel Corporation の商標である．
- Red Hat，Red Hat Enterprise Linux は米国および他国における Red Hat, Inc. の登録商標である．