分散ストレージシステムによる SMR ハードディスクの直接管理の検討

上西 康太[1,a)]　中野 将生[1,b)]　丸山 泰史[1,c)]

概要：瓦記録方式 (SMR: Shingled Magnetic Recording) のハードディスクは，従来の磁気記録方式 (CMR) より 2 割程度高い記録密度を持つが，host-managed SMR ディスク (HM-SMR) では数百 MB の「ゾーン」に追記するしかなく，任意の位置を書き換えられない．このため，メタデータを小さな単位で頻繁に更新するファイルシステムとは相性が悪い．本研究では，ブロックのゾーン内の位置を含むメタデータを分散ストレージシステムの中央のデータベースに集約し，HDD への書き出しを追記とゾーンのリセットだけに限る設計を提案する．本稿では予備実験として，提案手法におけるディスク I/O の部分を HM-SMR ディスクで評価した．空のディスクに対して 1MiB 以上のブロックサイズで書き込み速度 220〜227 MiB/s が達成された．これは Btrfs の zoned モードの 1.08〜1.12 倍，ノードの索引を NVMe SSD に置くログ構造のストアの 1.04〜1.13 倍であった．

キーワード：HM-SMR，分散ストレージシステム

1. はじめに

ハードディスク (HDD) は，大量のデータを低いコストで保存する媒体として，分散ストレージシステムの容量の大部分を担っている．瓦記録方式 (SMR: Shingled Magnetic Recording) は，隣接するトラックを部分的に重ねて書くことで記録密度を高める技術であり，同世代の従来型磁気記録方式 (CMR: Conventional Magnetic Recording) のディスクより 18〜25 % 大きな容量が得られる [1], [2]．その代わり，あるトラックに書き出すと隣のトラックのデータが壊れるため，ブロック単位で任意の位置を書き換えることができない．とくに host-managed SMR (HM-SMR) ディスクは，記録領域を数百 MB のゾーンに分割し，各ゾーンへの書き出しを，ゾーン内で次に書き出せる位置を示すライトポインタからの追記だけに制限する [3]．ゾーンの領域を再利用するには，ゾーン全体をリセットして空に戻すしかない．

このような制約の下で，単独のディスク上に効率的なファイルシステムを実装することは難しい．ファイルシステムはデータのほかに，ディレクトリ，inode，空き領域の管理情報といったメタデータを持ち，これらを小さな単位で頻繁に書き換えるからである．既存の HM-SMR 向けのファイルシステムやストレージエンジンは，メタデータを同じディスクの別の領域に置くか，SSD などの別のデバイスに置いてきた [4], [5], [6], [7]．前者では，データを追記する途中でヘッドがメタデータの位置へシークすることによりレイテンシが増加する．後者では追加のデバイスが必要になり，索引用のデバイスが 1 台故障すると，索引を作り直すまでノード内のすべての HDD のデータが読めなくなる．

一方，分散ストレージシステムでは，ファイルの名前空間やブロックの配置といったメタデータを，もともとストレージノードの外にある中央のメタデータサービスが管理する設計が多い．我々が提案した分散ストレージシステム [8] (以下，本システム) では，メタデータを RDBMS に格納する．ストレージノード上のディスクサービスはイミュータブルなブロックの保管だけを責務としたシンプルなプロセスである．**図 1** に構成を示す．この構成を利用すれば，ブロックがディスク上のどこにあるかという情報まで中央のデータベースに移し，ストレージノードには索引を一切置かずに済む．HDD に届く書き出しはブロックの追記とゾーンのリセットだけになり，メタデータのためのシークはなくなる．

本稿では，この考え方に基づいて HM-SMR ディスクを分散ストレージシステムから直接管理する方式を提案し，

[1] 株式会社 Preferred Networks
[a)] kota@preferred.jp
[b)] namachan@preferred.jp
[c)] tmaruyama@preferred.jp

ⓒ2026 Information Processing Society of Japan

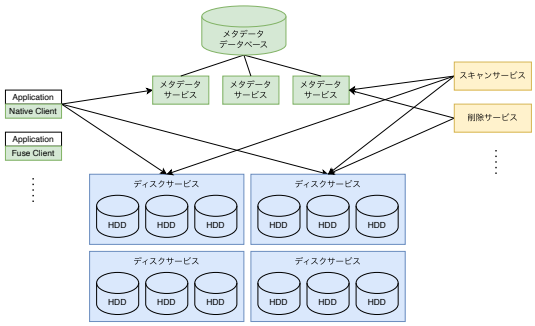

図 1 ディスクを扱うディスクサービスとメタデータを扱うメタデータサービスを別々に用意し，クライアントはそれぞれにデータ I/O とメタデータ I/O を行う．バックグラウンドジョブとしてスキャンサービスと削除サービスがあり，それぞれノードやディスクの状態管理と削除を担う．

ディスク I/O に関する部分の実装を実際の HM-SMR ディスクで評価する．本稿の貢献は次の 3 点である．

- ブロックの有効/無効とゾーン内の位置を中央のデータベースだけが持ち，ストレージノードは物理的な状態だけを持つ構成にすることで，メタデータ I/O とデータ I/O を効率的に分離した．
- ヘッダとフッタでブロックを自己記述的にし，データベースを失ってもディスク上の情報だけから，ブロック ID などのブロックに関するメタデータを復元できるようにした．
- 提案手法のディスクサービスを実装し，HM-SMR ディスクで，メタデータを同じディスクに置く Btrfs の zoned モード，ブロック位置の索引を NVMe SSD に置くログ構造のストア，および CMR ディスク上の ext4 にブロックごとのファイルを置く構成と比較した．

2. SMR ハードディスクとメタデータの配置

2.1 SMR ハードディスクとゾーン

SMR ハードディスクには drive-managed (DM-SMR), host-aware (HA-SMR), host-managed (HM-SMR) の 3 種類がある [3]．書き換えをディスクが内部で処理する DM-SMR は，内部のキャッシュを掃き出すたびに長い停止が生じ [9]，HA-SMR も同時に書き出すゾーンが増えると性能が急落する [10]．HM-SMR は，ゾーンの状態とライトポインタを ZBC/ZAC のコマンドでホストに公開し，ゾーンの割り当てと空き領域の回収をホストに任せることで，ホストによる効率的な I/O の制御を可能にする．

本稿で用いる WDC Ultrastar DC HC690 (30 TB) は，256 MiB のゾーンを 111,760 個持つ HM-SMR ディスクである．先頭の 1,118 ゾーン (約 280 GiB) は任意の位置に書き出せる conventional ゾーン (以下 cnv ゾーン) で，残りの 110,642 ゾーンは追記しかできない sequential ゾーンである．Linux の zonefs [11] は，各ゾーンを 1 つのファイルとして見せるファイルシステムである．ファイルの大きさは，そのゾーンに書き出し済みのバイト数，つまりゾーンの先頭からライトポインタまでの長さである．書き出しはファイル末尾への O_DIRECT での追記に限られ，ファイルを長さ 0 に切り詰めるとゾーンがリセットされる．

HC690 を fio で測ると 32 KiB のランダム読み込みは 500 IOPS (15.6 MiB/s) と，シーケンシャル読み込み (193.8 MiB/s) の 12 分の 1 であった．要求を 4 KiB から 32 KiB にしても IOPS はほとんど変わらず (519 から 500)，シークと回転待ちに 1 回あたり約 2 ms かかる．HDD の帯域を引き出すには，シークを減らしてシーケンシャルアクセスを保つことが何より重要である．

2.2 メタデータの配置

HM-SMR の上で通常のファイルシステムを提供するには，メタデータも追記だけで更新しなければならない．Btrfs の zoned モードは，コピーオンライトの B 木をデータとは別のブロックグループに追記する [4]．F2FS はログ構造で書く [12] が，メタデータを cnv ゾーンか別のデバイスに置く必要があり，16 TiB を超える容量には対応しない [13]．zoned XFS はメタデータを cnv ゾーンに置き，データのゾーンをガベージコレクション (GC) で回収する [5]．メタデータを別のデバイスに置かない限り，いずれもメタデータの更新はデータと同じデバイスに対する追加の書き出しとシークになる．

ゾーンを自ら管理するストレージエンジンも，ストレージノードに索引を持つ点では同じである．SMRstore [6] はチャンクの索引をディスクごとにメモリ上に持ち，チェックポイントとジャーナルを同じディスクのメタデータ用ゾーンに書く．SMORE [7] はオブジェクトの索引をフラッシュメモリに置く．Dropbox の Magic Pocket は書き出しを一時的に SSD に置いていたが [14]，のちに SSD を外し，ブロックのメタデータをディスク上に埋め込んで，起動のたびにディスクを走査してメモリ上の索引を作り直す方式に移った [15]．索引を別のデバイスに置けばデータの書き出しは追記だけになるが，1 台の索引用デバイスがノード内の多数の HDD の可用性を左右する．提案手法は，メタデータをストレージノードから完全に取り除いて分散ストレージシステムの中央のデータベースに置き，ノードには索引も起動時の走査も要らないようにする．

3. 提案手法の設計

3.1 本システムの構成

本システムは，メタデータ管理，データ管理，ネイティブクライアント，バックグラウンドジョブの 4 種類のコンポーネントから構成される [8]．メタデータ管理を担うメタデータサービスは，ディレクトリツリー，ファイルメタデータ，ブロック一覧 (各ブロックとその複製の所在を持つ locations 表)，ノード一覧，ディスク一覧を RDBMS

(現在の実装では PostgreSQL) に格納する．**図 2** に示すように，ファイルはパート (Part) から構成され，各パートは複数のブロックの位置 (Location) を持つ．この Location にはどのディスクノードにブロックが保存されているかと、そのディスクノード上のどのゾーンのどこの offset にあるかを示す位置情報（pos）を持つ。データ管理を担うディスクサービスはストレージノードごとに 1 プロセス動作する HTTP サーバで，ノード上のディスクにブロックを保管する．複製はネイティブクライアントが複数のディスクに書き出して作り，失われた複製の修復や削除の後始末は，スキャンサービスや削除サービスといったバックグラウンドジョブが行う．

ブロックは UUID で識別され，一度書かれたら変更されない．また，ある複製が有効かどうかは，ブロック一覧 (locations 表) に行があるかどうかだけで決まる．この 2 つの性質は，追記だけで書くログ構造のストアと相性がよい．現在のディスクサービスは 1 ブロックあたり 1 ローカルファイルとして保管しており (以下 posix ストア)，そのままでは HM-SMR ディスクに載せられない．

3.2 設計方針

提案手法では，HM-SMR ディスクを zonefs でマウントし，ディスクサービス自身がゾーンの割り当てとブロックの配置を決める．設計全体は，次の責務の分担に基づく．

メタデータサービス ストレージノードとは別の RDBMS に，ブロック一覧と，ブロックがどのノードに保存されているか、ブロックのゾーン内の位置 (pos) を持つ．

ストレージノード 各ゾーンのライトポインタと容量，空いているゾーンの一覧だけを持つ．いずれも zonefs のファイルに対する stat(2) で読める物理的な状態であり，ノードはマウント時にこれを読んでメモリ上に持つ．提案手法が使うのは sequential ゾーンだけである．

バックグラウンドジョブ スキャンサービスはブロックごとにデータベースの位置とヘッダの UUID を突き合わせ，削除サービスは回収のときにだけノードからゾーンの状態を読む．

3.3 書き出しと読み込み

図 4 に書き出しの流れを示す．ネイティブクライアントは，まずメタデータサービスからブロックの割り当てを受け (AllocatePart)，書き出し先のディスクを得る．次にディスクサービスにブロックを PUT すると，ディスクサービスはメモリ上のゾーン表から書き出し先のゾーンとオフセットを決めてブロックを追記し，その位置を応答のヘッダで返す．応答は O_DIRECT の書き出しが終わった時点で返す．ネイティブクライアントはこの位置を，ブロックの長さや CRC とともにメタデータサービスに報告し (CommitPart)，メタデータサービスはブロックのゾーン内の位置（pos）をデータベースに記録する．

メタデータサービスが一時的に応答しなくても，割り当て済みのブロックのディスクへの書き出しは完了できる．

pos は 64 bit の整数で，**図 3** のように上位から符号 (1 bit，RDBMS の符号付き整数に収めるため常に 0)，予約 (3 bit)，ゾーン番号 (24 bit およびこれらに対する CRC-8 (8 bit) を詰めたものである．読み込みでは，ネイティブクライアントが pos を渡し，ディスクサービスがペイロードの長さとゾーンの容量からディスク上のブロックの範囲を計算する．長さはネイティブクライアントが渡すか，なければ先頭の 4 KiB のヘッダから読む．

3.4 ディスク上のデータ形式

図 5 にゾーンとディスク上のブロックの形式を示す．ブロックは，ヘッダ，ペイロード (ブロックのデータ)，0 埋めの余白，16 B のフッタを並べ，全体を 4 KiB の倍数に切り上げた形でディスクに書く．ヘッダは magic，ブロックの UUID，ペイロードの長さと CRC-32 などを持ち，フッタはこのブロック自身の pos (self_pos) を持つ．フッタは必ずディスク上のブロックの最後の 16 B に置くので，その末尾が分かればフッタを読める．

self_pos は書き出し完了の印を兼ねる．書きかけで途切れたブロックのフッタは 0 か無関係なバイト列になるが，0 を詰めた pos は CRC-8 の検査を通らない．したがって，self_pos が読み手の使った pos と一致すれば，ブロックは完全に書かれている．また，ゾーンの回収でブロックが移り，元の位置に別のブロックが書かれた後に古い pos で読みに来た場合は，ヘッダの UUID が一致しないことで検出できる．

3.4.1 ゾーンへの追記の終了

次のブロックがゾーンの残りに収まらないとき，ディスクサービスはそのゾーンへの追記を終える．このとき，最後のブロックの後ろから容量の 4 KiB 手前までを 0 で埋め，最後の 4 KiB に，そのゾーンに書いたブロックの UUID の Bloom filter [16] を書く．これでゾーンは容量まで書かれ，FULL の状態になる．Bloom filter の位置はゾーンの容量だけで決まるので，データベースがなくても読める．Bloom filter は 32,256 bit で，偽陽性率は 1 ゾーンに 51 個のブロックを入れる既定の構成で 2.3×10^{-26} である．Bloom filter は，データベースとの突き合わせと復旧のときだけに読む．

3.5 障害からの復旧

zonefs ではファイルの大きさがそのゾーンに書き出し済みのバイト数を表すので，ディスクサービスはマウント時に全ゾーンのファイルを stat(2) するだけで，大きさが 0 なら空き，容量に等しければ満杯，その間なら追記中と分

ⓒ2026 Information Processing Society of Japan

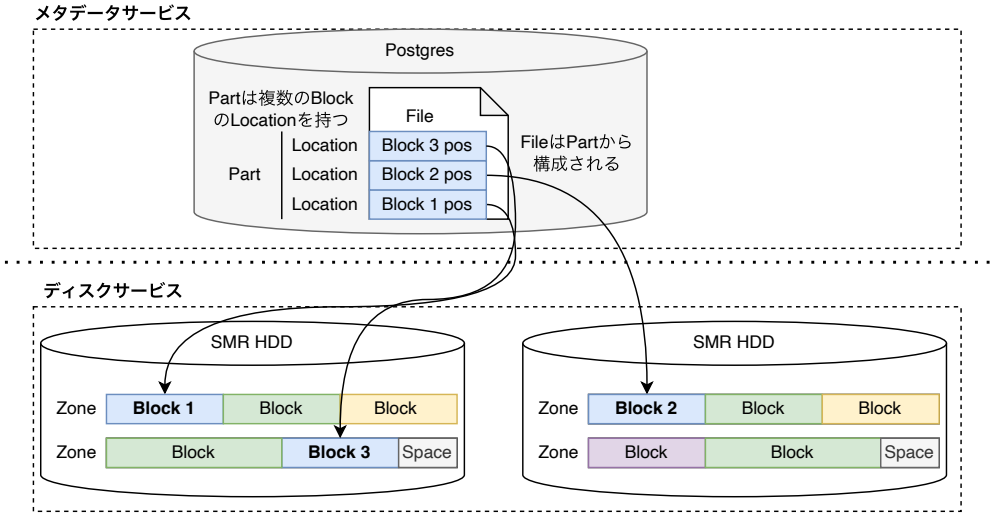

図 2 ファイル，パート，ブロックの関係．メタデータサービスは各ブロックの位置 (pos) を PostgreSQL に持ち，ディスクサービスはその位置にあたる HM-SMR ディスクのゾーン内にブロックを保管する．

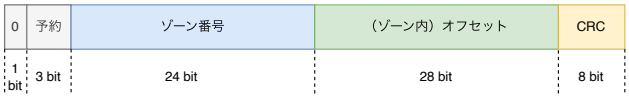

図 3 pos のビット構造．ゾーン内のオフセットは 4 KiB 単位で，CRC はそれより上位の 56 bit に対する CRC-8 である．

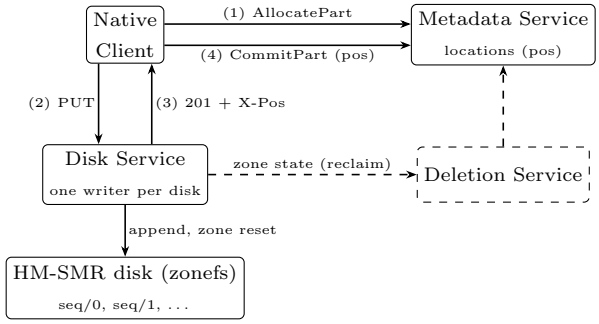

図 4 提案手法の書き出しの流れ．書き出し先はディスクサービス (図中 Disk Service) が決めて応答で返し，ネイティブクライアント (図中 Native Client) がメタデータサービス (図中 Metadata Service) に報告する．破線は，回収のときだけ削除サービス (図中 Deletion Service) が使う経路である．

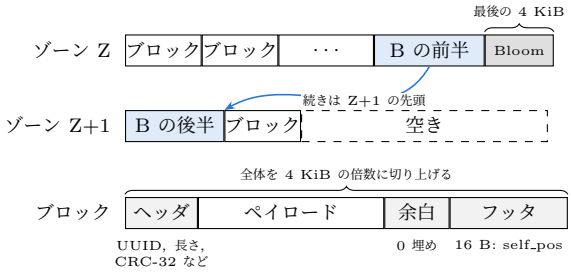

図 5 ゾーンとディスク上のブロックの形式．収まらないブロック B は次のゾーンの先頭にだけ続けてよい．追記を終えたゾーンの最後の 4 KiB には，そのゾーンのブロックの UUID の Bloom filter を置く．

類できる．追記中のゾーンについてだけ最後のページを読んでフッタの self_pos を検査し，最後のブロックが途切れていれば，0 埋めと Bloom filter の書き出しによってそのゾーンへの追記を終える．

データベースを失っても，ディスク上のブロックが自己記述的なので，ディスク上のデータだけで復元することが出来る．あるディスクにあるブロックが書かれたことがあるかどうかは，全ゾーンの Bloom filter を読めば分かる．これは約 11 万回の 4 KiB のランダム読み込みで，500 IOPS ならば約 3.7 分で終わる．位置まで復元するには，全ゾーンを順に読み，ブロックの先頭にあるヘッダと，それを self_pos で指すフッタの組を探せばよい．またいだブロックの続きは次のゾーンの先頭だけを調べればよい．30 TB のディスクの全ゾーンを順に読むには約 40 時間かかる見込みであり，Bloom filter のスキャンによって時間を短縮出来る．

4. 実装

ディスクサービスは前稿 [8] から引き続き Rust で実装した．本稿では次の 3 つの保管方式（ストア）を比較する．

posix ストア 前稿と同じく，通常の Linux ファイルシステム上に 1 ブロックあたり 1 ローカルファイルとして保管する．本稿ではファイルシステムは ext4 と Btrfs を用いた。ページキャッシュは無効化しておらず，書き出しの後に fsync もしない．

ログストア 我々が先行して試作したログ構造のストアである．zonefs 上で 256 MiB のゾーン 1 本を 1 セグメントとして追記し，ブロックの位置の索引を組み込みデータベース (redb) に持つ．1 MiB 未満の小さいブロックとそれ以外を別のセグメントに書き，提案手法

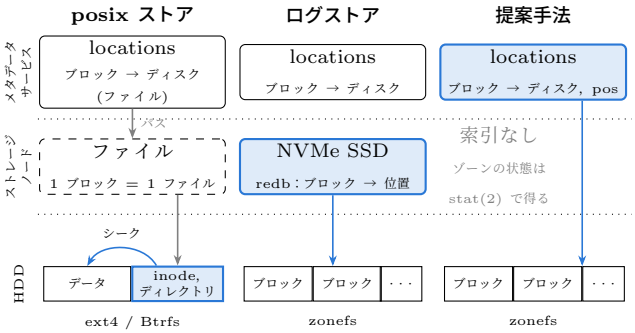

図 6 構成ごとのブロックの位置の持ち方．色付きの箱がブロックのディスク上の位置を持つ場所である．posix ストアはブロックごとのファイルを介して同じ HDD 上のファイルシステムのメタデータを，ログストアはノードの NVMe SSD 上の索引をたどるが，提案手法は中央のデータベースの pos から直接読む．

と同じくゾーンへの追記を終えたときとライタの終了時にだけ同期する．

提案手法　ディスクごとに 1 本のライタが，バッファに組み立てたブロック全体を，1 回の pwritev で O_DIRECT で追記する．ネットワークからの受信とバッファの組み立ては次の書き出しと重ねて行い (深さ 2 のパイプライン)，読み込みはライタを通さず並行に行う．fdatasync を，ゾーンへの追記を終えたときとライタの終了時にだけ行う．

図 6 に，各ストアがブロックのディスク上の位置をどこに持つかを示す．

本稿で評価したのは，提案手法のうちディスクサービスの部分である．この評価ではベンチマークプログラムがディスクサービスと直接 HTTP で通信し，応答で得た位置を自身のメモリに保持してメタデータサービスの代わりとした．

5. 性能評価

5.1 評価環境とワークロード

評価には，Xeon Gold 6426Y を 2 基 (計 32 コア，SMT 無効) と DDR5-4800 のメモリ 512 GB を持ち，Ubuntu 24.04 このホストに，CMR の Toshiba MG07ACA14TE (14 TB，以下 Toshiba CMR) と WDC Ultrastar DC HC570 (22 TB，以下 WD CMR)，HM-SMR の HC690 (30 TB) 2 台 (SMR1，SMR2) をオンボードの SATA (AHCI，6 Gb/s) で接続し，ドライブごとに独立したディスクサービスを動かした．どの測定も，ほかのドライブの測定と並行して行った．ログストアの索引は NVMe SSD (KIOXIA KBG50ZNS512G) に置いた．**表 1** に比較した構成を示す．構成によって，メタデータの配置，書き出しのパターンが異なり，同じディスクでも性能特性が変わる．

ベンチマークプログラムはドライブごとに 1 つのディスクサービスに HTTP でリクエストを送り，スループットとレイテンシを測る．実験には次のワークロード (以下 put-get) を用いる．

表 1 比較した構成．永続化はデータを同期する単位で，「ゾーン」はゾーンへの追記を終えたときだけ同期する

構成 (ドライブ)	メタデータ
ext4 + posix (Toshiba CMR)	同じディスク
Btrfs + posix (WD CMR)	同じディスク
Btrfs-zoned + posix (SMR1)	同じディスク
zonefs + ログストア (SMR2)	NVMe 上の索引
zonefs + 提案手法 (SMR2)	中央の DB（模倣）

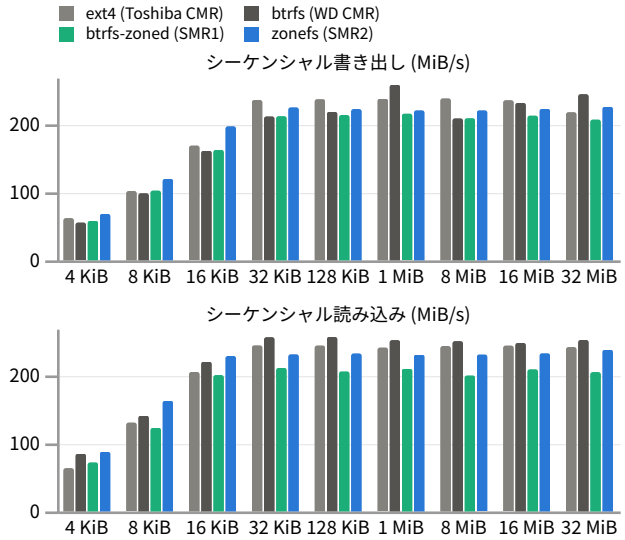

図 7 fio のシーケンシャル書き出しと読み込み (各 15 秒，conventional 領域の 4 GiB)

put-get　空のファイルシステムに，ブロックサイズ (110 KiB，1 MiB，4 MiB，8 MiB，16 MiB，32 MiB) ごとに並列数 128 で 24 GiB キャッシュを捨ててから 120 秒間ランダムに GET し，最後にすべてを DELETE する．

書き込み量は，どの構成でもブロックサイズごとに 24 GiB，合計 144 GiB である．

5.2 デバイス単体の性能

図 7 に fio で計測した結果を示す．128 KiB 以上のシーケンシャル書き出しは，zonefs 上の HC690 が 221〜227 MiB/s で，ext4 上の Toshiba CMR より 128 KiB〜16 MiB では 5〜7 % 低く，32 MiB では 4 % 高かった．Btrfs では，WD CMR が 210〜259 MiB/s，HC690 (Btrfs-zoned) が 208〜216 MiB/s であった．シーケンシャル読み込みは，どのドライブも 32 KiB 以上でほぼ一定になり，HC690 が zonefs で 231〜239 MiB/s，Btrfs-zoned で 201〜212 MiB/s，CMR が 242〜257 MiB/s であった．

5.3 put-get：空のディスクへの書き出しと読み込み

図 8 に put-get の結果を示す．1 MiB 以上のブロック

ⓒ2026 Information Processing Society of Japan

では，提案手法は 220〜227 MiB/s で書き出し，同じドライブで取った fio のシーケンシャル書き出し (221〜227 MiB/s) と同じ水準に達した．これは同じドライブのログストア (201〜216 MiB/s) の 1.04〜1.13 倍，Btrfs-zoned + posix (203〜209 MiB/s) の 1.08〜1.12 倍である．ext4 + posix は Toshiba CMR で 84〜141 MiB/s にとどまり，8 MiB の PUT の 99 パーセンタイルのレイテンシは 34 秒に達した．ブロックごとのファイル，ディレクトリ，inode の書き出しのたびにシークが発生するためと考えられる．

110 KiB のブロックでは，提案手法の PUT は 180 MiB/s で，Btrfs + posix (WD CMR) の 183 MiB/s とほぼ同じであり，ログストアの 123 MiB/s の 1.46 倍であった．8 MiB の PUT の 99 パーセンタイルのレイテンシは提案手法が 4.8 秒と最も短く，ログストアは 5.8 秒，Btrfs-zoned + posix は 10.2 秒であった．

110 KiB の GET は，ログストアが 39.0 MiB/s，提案手法が 29.3 MiB/s で，ext4 + posix の 10.7 MiB/s の 2.7〜3.7 倍であった．ext4 では GET のたびにディレクトリと inode を読むシークが加わる．なお，大きなブロックで Btrfs-zoned の構成の GET のスループットが高く出るのは，ページキャッシュの影響と考えられる．posix ストアはページキャッシュを通して読み，GET は 120 秒の間に同じブロックを重複して選ぶので，2 回目以降の読み込みはキャッシュに当たり得る．たとえば Btrfs-zoned + posix の GET は，8 MiB で 305 MiB/s と，同じドライブで取った fio のシーケンシャル読み込み (最大 212 MiB/s) を上回った．

6. 関連システム

2 章で述べたもののほかに，LSM 木のキーバリューストアである GearDB [17] も HM-SMR を直接扱い，zonefs の上にも実装されている [18] が，索引はストレージノードに持つ．SMORE [7] はゾーンの組を閉じるときにレイアウト情報の要約を書いて復旧時の走査を 1 回の I/O で済ませ，Venti [19] は追記の単位の末尾側にブロックの一覧を置いて索引を再構築できるようにしている．本稿の Bloom filter はこれらに近いが，突き合わせのための確率的な要約で，位置がゾーンの容量だけで決まる．

書き出した位置を上位に返す設計としては，ZEA [20] が SMR ディスクへの追記の結果として LBA を呼び出し元に返し，索引の管理を呼び出し元に任せる．ZNS の Zone Append [21]，GFS の record append [22]，SSD の nameless write [23] も，書き出し先をデバイスやサーバが決めて呼び出し元に返す．一方，HDD に追記する BLOB ストアの Haystack [24] は，データの位置の索引をストレージノードに持つ．Windows Azure Storage [25] はエクステントとオフセットを上位のパーティション層の索引に持つが，ストレージノードにもエクステントごとの索引を持つ．Apple の ACOS [26] も，オブジェクトを格納したコンテナとその中のオフセットを中央のメタデータに持つが，コンテナはノード上のファイルシステムのファイルである．Tectonic [27] はメタデータを ZippyDB に置くが，ブロックを構成するチャンクはノード上の XFS のファイルとして保存する．我々の調べた範囲では，ゾーン内の位置を中央のデータベースだけに持ち，ストレージノードに索引を一切持たないストアは見当たらない．

zoned XFS [5] は Linux 6.15 以降を要するので比較に含めていない．

7. 今後の課題

今後の課題としては，ブロックの削除によって不要になった領域を回収し，ゾーンを再編成することで容量を効率よく利用する GC の実装が必要である．

直感的には Copy GC が有効だと考えられるが，Copy GC の場合はディスクに対するアクセスが一つの HDD デバイスに集中する．そのため，読み込みと書き込みの帯域が同じ HDD ヘッドに載り効率が悪いことが予想される．

GC を行う場合では回収と書き込み・読み込みで帯域を食い合うため、一定の読み込み帯域を GC に明け渡す仕組みが要る．設計では回収の帯域を前景の書き出しの実績から需要に応じて割り当てるが，その効果は未評価である．

また，本稿の評価は HDD 1 台に対するディスクサービス単体のベンチマークである．GC を含めた評価に加えて，メタデータサービスとネイティブクライアントを含めたストレージシステム全体を，ストレージのベンチマークで評価する必要がある．その際には，CommitPart での pos の書き込みや，GC での回収のための範囲問い合わせがメタデータサービスにかける負荷の調査を行う．ベンチマークとしては機械学習の学習データを供給する性能を測る MLPerf Storage [28] を検討している．

このほか，容量の大部分を埋めた状態や削除を伴う定常的なワークロードでの評価，flush を待たない応答のもとで複製を持つ複数のノードが同時に電源を失った場合の影響，zoned XFS との比較が課題である．

8. おわりに

本稿では，分散ストレージシステムが HM-SMR ディスクを直接管理する方式を提案した．メタデータをゾーン内の位置まで含めて中央のデータベースに集約し，ストレージノードは索引を持たない．HDD への書き出しは追記とゾーンのリセットだけになり，データベースを失っても自己記述的なブロックからその位置を再構成できるようにした．

30 TB の HM-SMR ディスクでの評価では，ゾーンへの追記を終えたときだけ flush する構成で，空のディスクに 1 MiB 以上のブロックを fio のシーケンシャル書き出しと

ⓒ2026 Information Processing Society of Japan

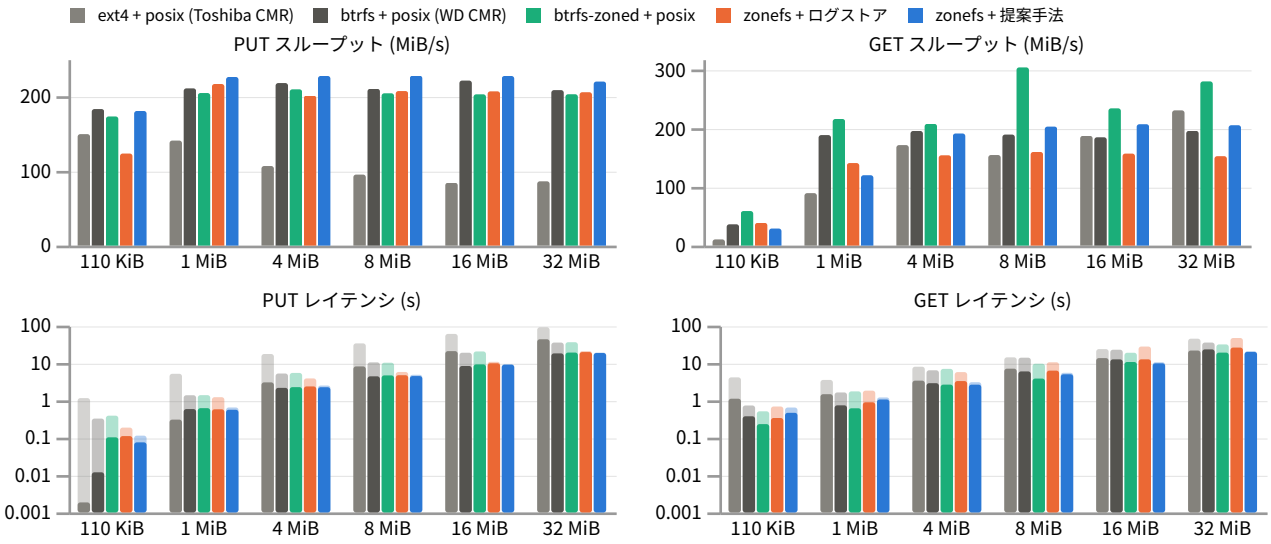

図 8 put-get の結果をブロックサイズごとに示す．レイテンシは対数軸で，濃い棒が p50，薄い棒が p99 であり，短いほど良い

同じ水準の 220〜227 MiB/s で書き出し，Btrfs の zoned モードの 1.08〜1.12 倍となった．8 MiB の PUT の 99 パーセンタイルのレイテンシも 4.8 秒と，比較した構成の中で最も短かった．その代わりに増えるメタデータサービスの負荷の評価は今後の課題である．

謝辞 本システムの実験環境を構築する上で，株式会社 Preferred Networks のクラスタ構築・運用チームのみなさんに多くの協力をいただいた．ここに感謝の意を表する．

参考文献

[1] Western Digital: Tech Brief: UltraSMR Technology, Western Digital (online), available from ⟨https://documents.westerndigital.com/content/dam/doc-library/en_us/assets/public/western-digital/collateral/tech-brief/tech-brief-ultrasmr-technology.pdf⟩ (accessed 2026-10-24).

[2] Holmberg, H. and Hellwig, C.: Unlocking SMR Capacity in EOS Using Zoned XFS, EOS 2026 Workshop, CERN (online), available from ⟨https://indico.cern.ch/event/1622471/contributions/6902527/⟩ (accessed 2026-10-24).

[3] Western Digital: Getting Started with SMR Hard Disks, Zoned Storage (online), available from ⟨https://zonedstorage.io/docs/getting-started/smr-disk⟩ (accessed 2026-10-24).

[4] BTRFS: Zoned mode, BTRFS documentation (online), available from ⟨https://btrfs.readthedocs.io/en/latest/Zoned-mode.html⟩ (accessed 2026-10-24).

[5] Hellwig, C., Holmberg, H. and Le Moal, D.: Staying in the Zone - Retrofitting Zoned Storage into a Scalable Enterprise File System, *Proc. ACM APSys '25*, pp. 30–37 (2025).

[6] Zhou, S., Xu, E., Wu, H. et al.: SMRSTORE: A Storage Engine for Cloud Object Storage on HM-SMR Drives, *Proc. USENIX FAST '23*, pp. 395–408 (2023).

[7] Macko, P., Ge, X., Haskins, Jr., J. et al.: SMORE: A Cold Data Object Store for SMR Drives, *Proc. MSST '17* (2017).

[8] 上西康太，杉原航平，中野将生，田村敢太：商用機械学習基盤の継続的開発に向けたストレージシステムの検討，情報処理学会研究報告， Vol. 2025-OS-168, No. 8, pp. 1–9 (2025).

[9] Aghayev, A. and Desnoyers, P.: Skylight—A Window on Shingled Disk Operation, *Proc. USENIX FAST '15*, pp. 135–149 (2015).

[10] Wu, F., Yang, M.-C., Fan, Z. et al.: Evaluating Host Aware SMR Drives, *Proc. USENIX HotStorage '16* (2016).

[11] The kernel development community: ZoneFS - Zone filesystem for Zoned block devices, The Linux Kernel documentation (online), available from ⟨https://docs.kernel.org/filesystems/zonefs.html⟩ (accessed 2026-10-24).

[12] Lee, C., Sim, D., Hwang, J.-Y. and Cho, S.: F2FS: A New File System for Flash Storage, *Proc. USENIX FAST '15*, pp. 273–286 (2015).

[13] Western Digital: F2FS File System, Zoned Storage (online), available from ⟨https://zonedstorage.io/docs/filesystems/f2fs⟩ (accessed 2026-10-27).

[14] Magic Pocket Hardware Engineering Teams: Extending Magic Pocket Innovation with the first petabyte scale SMR drive deployment, Dropbox (online), available from ⟨https://dropbox.tech/infrastructure/extending-magic-pocket-innovation-with-the-first-petabyte-scale-smr-drive-deployment⟩ (accessed 2026-10-24).

[15] Kulshrestha, A., Goel, R. and Ummadi, S.: Increasing Magic Pocket write throughput by removing our SSD cache disks, Dropbox (online), available from ⟨https://dropbox.tech/infrastructure/increasing-magic-pocket-write-throughput-by-removing-our-ssd-cache-disks⟩ (accessed 2026-10-24).

[16] Bloom, B. H.: Space/Time Trade-offs in Hash Coding with Allowable Errors, *Commun. ACM*, Vol. 13, No. 7, pp. 422–426 (1970).

[17] Yao, T., Wan, J., Huang, P. et al.: GearDB: A GC-free Key-Value Store on HM-SMR Drives with Gear Compaction, *Proc. USENIX FAST '19*, pp. 159–171 (2019).

[18] Zhang, Y., Yao, T., Wan, J. and Xie, C.: Building GC-free Key-value Store on HM-SMR Drives with ZoneFS,

ⓒ2026 Information Processing Society of Japan

ACM Trans. Storage, Vol. 18, No. 3, pp. 23:1–23:23 (2022).

[19] Quinlan, S. and Dorward, S.: Venti: A New Approach to Archival Storage, *Proc. USENIX FAST '02* (2002).

[20] Manzanares, A., Watkins, N., Guyot, C. et al.: ZEA, A Data Management Approach for SMR, *Proc. USENIX HotStorage '16* (2016).

[21] NVM Express: NVM Express Zoned Namespace Command Set Specification, Revision 1.5, NVM Express (online), available from ⟨https://nvmexpress.org/specification/nvme-zoned-namespaces-zns-command-set-specification/⟩ (accessed 2026-10-26).

[22] Ghemawat, S., Gobioff, H. and Leung, S.-T.: The Google File System, *Proc. ACM SOSP '03*, pp. 29–43 (2003).

[23] Zhang, Y., Arulraj, L. P., Arpaci-Dusseau, A. C. et al.: De-indirection for Flash-based SSDs with Nameless Writes, *Proc. USENIX FAST '12* (2012).

[24] Beaver, D., Kumar, S., Li, H. C. et al.: Finding a Needle in Haystack: Facebook's Photo Storage, *Proc. USENIX OSDI '10*, pp. 47–60 (2010).

[25] Calder, B., Wang, J., Ogus, A. et al.: Windows Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency, *Proc. ACM SOSP '11*, pp. 143–157 (2011).

[26] Baron, B., Bousquet, A., Metens, E. et al.: ACOS: Apple's Geo-Distributed Object Store at Exabyte Scale, *Proc. USENIX FAST '26*, pp. 53–66 (2026).

[27] Pan, S., Stavrinos, T., Zhang, Y. et al.: Facebook's Tectonic Filesystem: Efficiency from Exascale, *Proc. USENIX FAST '21*, pp. 217–231 (2021).

[28] MLCommons: MLPerf Storage, MLCommons (online), available from ⟨https://mlcommons.org/benchmarks/storage/⟩ (accessed 2026-10-24).

ⓒ2026 Information Processing Society of Japan