SSDをディスクキャッシュとして用いるLinuxデバイスドライバの実装

仁 科 圭 介† 佐藤 未来子† 並木 美太郎†

Flash SSD（Solid-State Drive, 以下 SSD）は，HDD（Hard Disk Drive）に比べランダムアクセス性能が高く，省電力であるという特徴を持つが，SSD の容量単価は HDD に比べて高く，SSD をコスト効率よく計算機システムの高速化や省電力化に利用する方式が課題である．本研究では，HDD のみで運用されている計算機のディスクアクセスを高速化するために新たに SSD を追加する場合を想定し，SSD を HDD のディスクキャッシュとして用いる方式を提案し，この方式を実現する Linux デバイスドライバの設計・実装を行った．本稿では本デバイスドライバにこれまでに実装した機能について概説するとともに，新たに実装した，SSD 内にキャッシュされた書き込みデータ（ダーティデータ）の追い出しによる性能低下に対処する改善策の設計，実装と評価を示す．評価では，ダーティデータの追い出しを遅延し，ストレージデバイスへの負荷を分散させることでアクセス性能の改善を実現した．

An Implementation of a Linux Device Driver Using SSD as a Disk-cache

Keisuke Nishina,† Mikiko Sato† and Mitaro Namiki†

Solid-State Drive (SSD) takes higher performance at random accesses and consumes lower power than Hard-Disk Drive (HDD). However, SSD is more cost per bytes than HDDs. Therefore, an issue is implementing the cost-effective method to use SSD. This paper proposes a method using SSD as a disk-cache of HDDs by using SSD to improve disk I/O performance of existing computer system with HDD storage. The design and implementation of a Linux device driver for this method is discussed on the paper. This paper reviews feature the device driver in this study, and shows design and implementation of new method newly implemented to improve I/O performance when synchronise cached dirty data to HDD. In the evaluation, improving I/O performance by the newly method is shown.

1. はじめに

近年，NAND 型フラッシュメモリを記憶素子に用いた Flash SSD（Solid State Drive）と呼ばれるストレージデバイスが普及してきている．Flash SSD（以下 SSD）は，これまでストレージデバイスとして広く使われてきた HDD よりも優れたアクセス性能を発揮し，なおかつ消費電力も比較的小さいという特徴を持つ．しかしながら，SSD の容量単価は HDD の十倍以上あり，SSD のみで大容量のストレージを用意すると非常に割高になる．したがって，よりコスト効率の良い SSD の利用法が課題である．

本研究では，HDD をストレージに用いた既存の計算機に，ディスクアクセス性能の向上を目的として SSD を追加する場合を想定し，様々な環境に汎用的に適用可能な方式での SSD の効率的な利用の実現を目指し，SSD を HDD のディスクキャッシュとして利用するシステムの設計と実装を行ってきた．

本稿では，本デバイスドライバにこれまでに実装した機能について概説するとともに，新たに実装した，SSD 内にキャッシュされた書き込みデータ（ダーティデータ）の HDD への追い出しによる性能低下に対処する改善策の設計，実装と評価について述べる．第 2 章で本研究の背景について述べ，第 3 章でこれまでに行ってきた実装の概要を示す．第 4 章では，これまでに適用してきたダーティーデータの追い出し方式の問題点を示し，これを解決するために本研究で新たに提案する追い出し方式の設計を示す．第 5 章で本提案方式の実装と評価を行う．

† 東京農工大学
Tokyo University of Agriculture and Technology

2. 研究の背景

本章では，本研究の背景として，既存のSSD利用方式を紹介し，その課題を整理し，本研究の目標を述べる．また，本研究における本発表の位置づけを示す．

2.1 既存のSSD利用方式

SSDを効率的に利用してディスクアクセスなどのI/Oの高速化や省電力化を図る方式は既に様々なものが存在する．既存の利用可能な方式として Intel Turbo Memory[1]や，Solaris ZFS[2]ファイルシステム，Flachcache[3]が挙げられる．また，SSDをディスクキャッシュとして用いる先行研究にFlaz[4]がある．以下では，これら4つの方式の特徴と課題を示す．

Intel Turbo Memory（ITM）は，PCI-Express接続の専用のSSDデバイスをディスクキャッシュとして用いて，ディスクアクセスの高速化とHDDへのアクセス低減による省電力化を図る技術である．しかし，これを利用するためには，チップセットなどの対応が必要であり，ハードウェア的な制限が大きい．

ZFSファイルシステムはL2ARCと呼ばれるメモリキャッシュの二次キャッシュ領域や，ZILと呼ばれる書き込みログ領域に任意のボリュームを設定でき，ここにSSDを用いることでファイル入出力の高速化を図れる．しかし，これはZFSファイルシステムの機能であり，他のファイルシステムでは利用できない．

FlashcacheはSSDをHDDの読み書きキャッシュとして利用するもので，LinuxのブロックI/Oレイヤーを扱うデバイスドライバにより実装されている．そのため，任意のファイルシステムや，ブロックデバイスとして仮想化される任意のストレージデバイスで利用可能であるが，SSD上のキャッシュブロックの管理にブロック（4KB～16KBの固定サイズ）一つ当たり24バイト（64bit環境時）の主記憶領域が必要であり，メモリオーバヘッドが大きい．

Flazは，HDDのデータブロックを圧縮してSSDにキャッシュすることで，より効率的にSSDの容量を活用するものである．FlazはFlashcacheと同様にLinuxのブロックI/Oレイヤーでのキャッシュを行うため，任意のファイルシステムやストレージデバイスが利用可能であるが，オンラインでデータブロックの圧縮・解凍を行うため，圧縮による効果と，圧縮・解凍にかかるCPUサイクルのトレードオフとなっている．

2.2 本研究の目標

本研究では，HDDをストレージに利用した既存の計算機にSSDを追加してディスクアクセスの高速化を図ることを想定している．そのため，既存の環境からの変更点が少なく，SSDを利用したシステムへの移行が容易であることが望ましい．したがって，適用可能な環境の制約が少ない方式が求められる．上述した先行システムの例では，次のような制約があった．

・**ITM**　対応チップセットが必須
・**ZFS**　ZFSファイルシステムの利用が必須
・**Flashcache**　メモリオーバーヘッドが大きい
・**Flaz**　圧縮・解凍によるCPUサイクルの消費

本研究では，ここに示したような制約がより少ない方式で，SSDを効率的にディスクアクセスの高速化に利用するシステムを実現することを目標とする．

具体的には，ハードウェアの変更や，HDD上のファイルシステムの変更なしに，利用できるシステムとするため，FlashcacheやFlazと同様に，SSDをHDDのディスクキャッシュとして用いるLinuxデバイスドライバを実装する．また，FlashcacheやFlazの課題である計算機資源の消費を抑える軽量なシステムを目指す．

2.3 本発表の位置づけ

本研究で，これまでに既発表の部分では，次に示す機能を実装している[5]．

- ライトバックキャッシュポリシー
- Readミス時のみ割り当てを行う
  キャッシュポリシー
- キャッシュ永続化情報のSSDへの格納
- 「セット」単位の領域マッピング
- LRUによるキャッシュ置換

これらを実装したシステムを実際に動作させた既発表の評価[5,6]では，キャッシュヒット時とライトバックキャッシュポリシーでのwriteミスによる新規キャッシュ割り当て時に，SSDの基本性能と同等のアクセス性能を実現できることが明らかになった．また，LRUによるキャッシュ置換時にSSDキャッシュ上のダーティデータのHDDへの追い出しが発生すると，アクセス性能が低下することがわかった．

今回，ダーティデータの追い出しによるアクセス性能の低下を防ぐため，新たに追い出しアルゴリズムの改善を行った．本発表での新しい成果は，このアルゴリズムの実装と評価となる．

3. SSDディスクキャッシュシステムの概要

本章では，これまで本研究で実装した既発表のSSDディスクキャッシュシステムの概要を述べる．最初にシステムの全体構成を示す．次に実装したディスクキャッシュドライバの構成を示し，ドライバの構成要素のうち，本研究で作成したdm-ssdモジュールにおけるデー

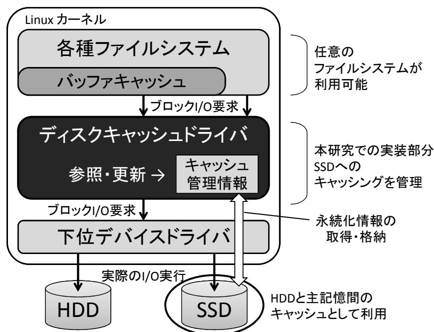

図 1　システムの全体構成

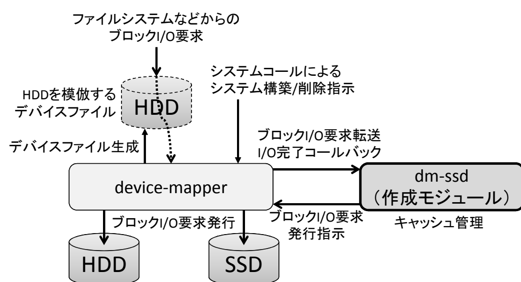

図 2　ディスクキャッシュドライバの全体構成

タ管理機構について述べる．

3.1 全体構成

本システムの全体構成を**図 1** に示す．Linux では，各種ファイルシステムやバッファキャッシュからブロックデバイスに対してブロック I/O 要求が発行される．本システムでは，このブロック I/O 要求をディスクキャッシュドライバが受け取り，SSD を HDD と主記憶の間のキャッシュとして利用しながら要求を処理する．ブロック I/O 要求を扱うレイヤーでの実装により任意のファイルシステムが利用可能である．また，本ドライバでは，キャッシュ管理情報のうち，SSD 内のキャッシュデータの永続化に必要な情報をシステムの終了時に SSD 内に格納し，キャッシュシステムの再構築時に読み込むことで，SSD キャッシュの永続化を可能にしている．

3.2 ディスクキャッシュドライバの構成

本ディスクキャッシュドライバでは，**図 2** に示すとおり，ブロックデバイスの仮想化を支援するデバイスドライバ群である “device-mapper”[7] と，本研究で実装した SSD のキャッシュ管理を行う dm-ssd モジュールで構成される． device-mapper は，SSD ディスクキャッシュ作成/削除時に利用するシステムコールインタフェースの提供，HDD へのブロック I/O 要求を受け取るデバイスファイルの作成，実デバイスへの I/O 発行などを行う．dm-ssd モジュールでは，device-mapper から転送されたブロック I/O 要求を受け取り，device-mapper に実デバイスへの I/O を指示する．

3.3 dm-ssd モジュールのデータ管理機構

dm-ssd 内のキャッシュ管理では，HDD から SSD への領域の対応付け（マッピング）を管理している．この対応付けはキャッシュの置換などを容易に行えるように固定サイズで行い，主記憶上で対応表（マッピングテーブル）により管理する．dm-ssd では，主記憶の消費を抑えるため，マッピングエントリの数を抑え，かつ SSD へのデータ割り当てのオーバヘッドを小さくするため，次に示す 2 種類の管理単位を用いている．

ブロック　SSD へのキャッシュデータ割り当ての単位
セット　HDD 空間を SSD 空間に対応付ける単位

ブロックはセットに含まれ，ブロックは 2 の N 乗セクタ，セットは 2 の M 乗ブロック（N, M は自然数）で指定できる．以下，SSD 内のセット，ブロックのことをそれぞれ「キャッシュセット」,「キャッシュブロック」と呼び，HDD 内のセットのことを「HDD セット」と呼ぶ．対応付けをセット単位にすることで，セット内の個々のブロックについては対応付けの情報が要らず，データの割り当て状態のみを管理すればよい．したがってキャッシュブロック一つ当たりに必要なキャッシュ管理情報（メタデータ）は，ブロックごとにマッピングする場合に比べて小さくなり，メモリオーバヘッドを削減できる．

また，ブロックの大きさには，ファイルシステムなどから受け取るブロック I/O 要求の最小単位を指定することで，要求の大きさと同じ大きさでキャッシュ割り当てが可能になり，割り当てによる余分な I/O を削減できる．

dm-ssd モジュール内では，各キャッシュセットごとにキャッシュセット構造体というメタデータを持ち，これを利用してキャッシュセットの管理を行っている．dm-ssd 内のキャッシュセット管理の模式図を**図 3** に示す． 各キャッシュセットは LRU リスト，空きセットリストのどちらかに連結されている．各リストの意味を次に示す．

LRU リスト:　HDD セットにマッピング済みの全キャッシュセットを LRU ポリシーにより並べたリスト．リストの先頭要素のキャッシュセットは，リスト内のセットの中で，最後に使われてから最も長い時間が経過したキャッシュセットとなる．

空きセットリスト:　HDD セットに未マッピングの全キャッシュセットのリスト．

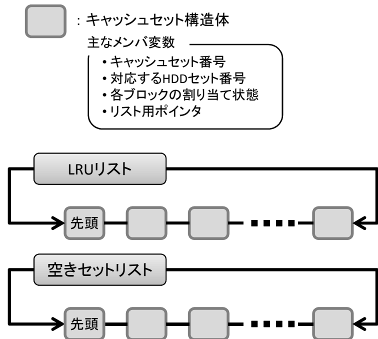

図 3 dm-ssd でのキャッシュセット管理

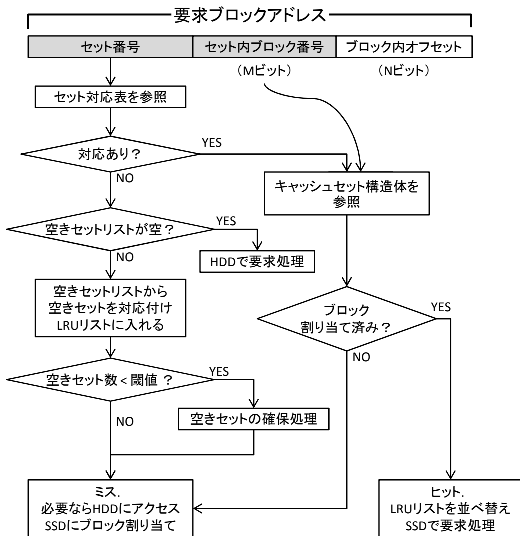

図 4 I/O 要求処理アルゴリズム

各リストは次に説明する I/O 要求処理において利用される．

図 4 に本デバイスドライバの要求処理アルゴリズムを示す． HDD セットに対応付けたキャッシュセット番号を管理するセット対応表と，キャッシュセット内の各ブロックの状態を管理するキャッシュセット構造体により，SSD 内のキャッシュの状態を参照する．SSD に要求ブロックが割り当てられていない，すなわちキャッシュミスならば新たに割り当てを行う．この際，要求ブロックが含まれた HDD セットのキャッシュセットへの対応づけがされていない場合には，新たにセットの対応付けを行う．要求ブロックが SSD に割り当てられていた場合，すなわちキャッシュヒットの場合は，SSD で I/O を行う．

また，空きセット数がシステムで予め設定された閾値より少なくなった場合は，空きセットの確保処理（図 4 下側中央）を行う．ここでは，LRU リストの先頭にあるキャッシュセットを空きセットにする処理を行うが，このキャッシュセットがダーティデータを含んでいた場合，ダーティデータを HDD に追い出す処理を行う．

4. SSD ディスクキャッシュの追い出し方式

本 SSD ディスクキャッシュシステムでは，SSD 内の残りの空きセット数が事前に設定された閾値よりも小さく，かつ，割り当て済みセットの LRU リストの先頭にあるセットがダーティデータを含む場合に SSD から HDD への追い出し処理を行う．本章では，これまでに適用してきた追い出し方式の問題点を示し，本研究で新たに提案する追い出し方式の設計について述べる．

4.1 ダーティデータの追い出し方式の問題点

これまでに適用していた SSD ディスクキャッシュにおけるダーティデータの追い出し方式の処理の流れを **図 5** に示す． この処理は，図 4 の図中の「空きセットの確保処理」の部分で行われる．

図 4 で示すように，ドライバに到着した I/O 要求がアクセスを要求するブロックを含んだセットがまだキャッシュセットに対応付けされていない場合，空きセットリストから空きセットを 1 つ取り出して新たに対応付けを行う．空きセットが消費されるのはこの場合のみであるから，このときに空きセット数が予め設定された閾値より少なくなったかを調べ，少ない場合は，空きセットを増やすために LRU リストの先頭にあるセットを取り出す．このセットがダーティデータを含んでいなかった場合は，そのままメタデータ上のみでこのセットを無効化し，空きセットリストに加える．LRU リストの先頭から取り出したセットがダーティデータを含んでいた場合，このセット内のダーティデータの追い出しを実施する．

これまでのディスクキャッシュシステムでは，追い

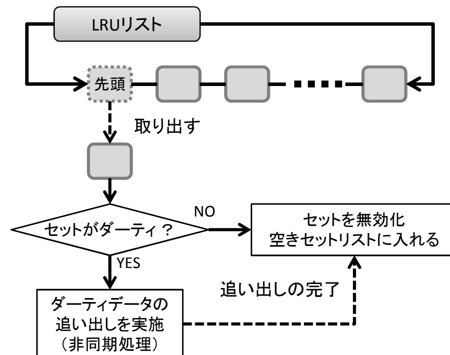

図 5 ダーティデータの追い出し処理の流れ

出しの際の SSD，HDD へのアクセスは非同期で行っていたが，必ず I/O 要求処理の途中で行われていたため，I/O 要求処理のためのブロックデバイスへのアクセスと追い出しのためのアクセスが重なり，I/O 要求処理の性能が悪化することが問題となっていた．

4.2 追い出し方式のアルゴリズムの設計

前節で述べたこれまでの追い出しアルゴリズムの問題点を解決する新しい追い出し方式のアルゴリズムを示す．

4.2.1 設 計 方 針

追い出し処理による SSD，HDD 双方へのアクセス負荷を分散させ，I/O 要求処理への影響を軽減する．これを実現するために，従来方式のように空きセット数が閾値より小さくなればすぐに追い出しを行うのではなく，I/O 要求の負荷（単位時間当たりに I/O 要求をストレージが処理する時間の割合）が小さくなってから HDD への追い出しを実施する方針とする．

4.2.2 I/O 要求の負荷の監視方法

本設計では，I/O 要求の負荷を監視し，負荷が低いことを判断することが必要となる．I/O 要求の負荷が小さくなったことを判断する条件として次の 2 通りの方法が考えられる．

(1) I/O 要求が一定時間以上到着しない場合
(2) 過去一定時間の I/O 要求量が一定量以下の場合

方法（1）は最後の I/O 要求到着から，追い出し実行までの「待ち時間」を予めシステムで設定し，その待ち時間の間に次の I/O 要求到着がない場合，I/O 要求による負荷が小さいと判定して追い出しを実行する方法である．この方式は最近の I/O 要求の到着時刻を管理することで実現が可能である．予め設定された待ち時間よりも I/O 要求の到着間隔が短い場合は，追い出しを実行しない．したがって，1 秒間隔程度の頻度で繰り返し少量のアクセスをするようなアクセスパターンが存在すると，それ自体の I/O 量は決して多くないにもかかわらず，設定された待ち時間がその間隔よりも長いと，追い出しを実施できないことになるため，待ち時間の設定には注意が必要である．

方法（2）は現在から一定期間遡った間にあった I/O 要求の量が一定量以下ならば，I/O 要求による負荷が小さいと判定する方法である．I/O 要求の量が条件に加わるため，周期的アクセスがある場合でも，I/O 要求の負荷の低下を検出できると考えられる．しかし，方法（1）よりも実装がやや複雑になる．また，どれほどの量なら負荷が低いと言えるのかは，使用するストレージデバイスの性能にもよると考えられ，適切な閾値設定が困難である．そこで今回は方法（1）の方式を採用する．

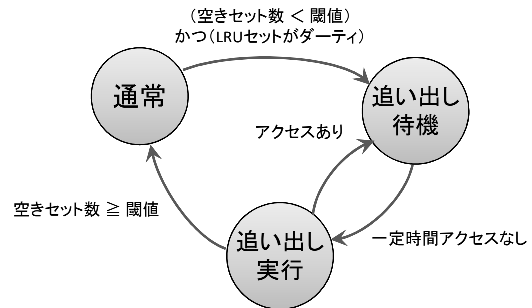

図 6 追い出しアルゴリズムの状態遷移図

4.2.3 追い出しアルゴリズムの詳細

提案する追い出し方式では，新たに「追い出しを待機する」状態を設け，追い出しが必要になる度に HDD への書き出しを行うのではなく，I/O 要求の頻度に応じて追い出し処理を実行するように改変した．**図 6** に，本提案方式の追い出しアルゴリズムにおける状態遷移図を示す．

通常状態は，残り空きセット数が閾値以上の状態であり，空きセットのマッピングが新たに起こっても閾値以上の空きセットが残っていれば何もしない．残り空きセット数が閾値より小さくなり，なおかつ LRU リストの先頭にあるセットがダーティセットだった場合，追い出し待機状態に移行する．

追い出し待機状態では，まだ LRU リストの先頭にあるセットの追い出しを開始しない．最後の I/O 要求がドライバに到着してから一定時間アクセスがない場合に限り追い出し実行状態へ遷移する．この最後の I/O 要求から追い出し実行までの待ち時間を「追い出し待ち時間」と定義する．

追い出し実行状態では，LRU リストの先頭にあるセットの追い出しを開始し，追い出しが完了したセットを空きセットにしていく．この処理は残り空きセット数が閾値以上になるまで繰り返されるが，途中で I/O 要求がドライバに到着すると一旦処理を中断し，再び追い出し待機状態へ遷移する．

5. 追い出しアルゴリズムの実装と評価

4 章で示した追い出しアルゴリズムを Linux オペレーティングシステムにおいて実装し，SSD ディスクキャッシュシステムの性能を評価した．本章では，実装および評価について述べる．

5.1 実　　装

本提案方式のアルゴリズムの実装には，Linux のカーネルスレッド実装であるワークキュー（workqueue）

```
state: 状態(normal, wait, writeback)
access: アクセスフラグ
free_floor: 閾値
nr_free: 空きセット数
nr_wbset: 追い出し中のセット数

workqueue() // 追い出し処理ワークキュー
{
    if (state == wait) {
        do {
            access = 0;
            sleep(wait_time);
        } while ( access );
        state = writeback;
    }

    while ((nr_wbset < 4) AND
          (nr_free + nr_wbset < free_floor)) {
        set = LRUリストの先頭セット;
        if (setがダーティ) {
            nr_wbset = nr_wbset + 1;
            writeback(set); // 追い出し
        } else {
            setを無効化;
            空きセットリストにsetを入れる;
            nr_free = nr_free + 1;
        }
    }

    if (nr_free + nr_wbset >= free_floor) {
        state = normal;
    }
}

callback(set) // 追い出し完了コールバック
{
    setを無効化;
    空きセットリストにsetを入れる;
    nr_wbset = nr_wbset - 1;
    nr_free = nr_free + 1;
    if ( nr_wbset <= 0 ) {
        if (state == writeback AND access) {
            state = wait;
        }
        workqueueを呼ぶ;
    }
}
```

図 7 提案追い出しアルゴリズムの擬似コード

を用いる．このワークキューは dm-ssd モジュール内に組み込まれ，図 6 の 3 つの状態のうち，追い出し待ち状態と追い出し実行状態の 2 つの状態において動作する．このアルゴリズムの擬似コードを**図 7** に示す．

通常状態から追い出し待ちに遷移するときに要求処理ルーチンからワークキューが呼び出される．

追い出し待ち状態のときは，予め設定された追い出し待ち時間の間スリープし，スリープ解除後にスリープ中にアクセスがあったか，フラグを確認し，アクセスがあった場合はまた追い出し待ち時間分スリープする．これをスリープ中にアクセスがなくなるまで続け，追い出し実行状態に遷移し，LRU リストの先頭要素内のダーティデータを HDD に追い出していく．

ダーティデータの追い出しは一度に 4 セットまでとし，追い出し完了の通知をコールバック関数で受け取り，追い出し処理の間にアクセスがあったか，フラグを確認し，アクセスがあった場合は追い出し待ち状態に戻る．アクセスがなければ，残り空きセット数が閾値以上になるまで LRU リストの先頭に来たセットのダーティデータの追い出しを繰り返す．

5.2 評価方法

本評価では，最初に，これまでの追い出し方式と，本研究で提案した新たな追い出し方式の性能の比較を行うため，追い出しが起こりやすい条件下での読み書き性能を各方式でそれぞれ測定する．その次に，新しい追い出し方式における閾値の調整がどのような効果をもたらすかを検証するため，新しい追い出し方式を実装したシステムにおいて閾値設定を変更して読み書き性能の測定を行う．

今回評価に用いた環境の諸元を**表 1** に示す．

5.2.1 評価に使う I/O 評価プラグラム

本評価に使う I/O 評価プログラムは次の 4 種類である．

w3g: 3GB のシーケンシャル書き込み

r3g: 3GB のシーケンシャル読み込み

wrand: 4KB ブロック × 3000 回のランダム書き出し

rrand: 4KB ブロック × 3000 回のランダム読み込み

本評価ではこれらの I/O を直接 I/O インタフェースを用い，バッファキャッシュを介さずに実行する．

5.2.2 条件設定

本評価の測定条件を**表 2** に示す．SSD キャッシュの設定は，追い出しが発生するように，あえて小さなキャッシュ領域で設定する． キャッシュ容量が 2GB で，セットサイズが 1MB なので，約 2000 個のキャッシュセットが用意される．

w3g, r3g では 2GB のキャッシュ容量に対してそれ

表 1 評価環境の諸元

CPU	Xeon X5550 2.66GHz
主記憶	4GB
オペレーティングシステム	Linux 2.6.35 (x86_64)
評価用 HDD	3.5 インチ 7200rpm 1TB
評価用 SSD	2.5 インチ MLC 100GB
システム用 HDD	USB 接続ポータブル HDD

表 2 SSD ディスクキャッシュの設定

キャッシュ容量	2GB
ブロックサイズ	4KB (8 セクタ)
セットサイズ	256 ブロック (1MB)
キャッシュポリシー	ライトバック
空きセット数の閾値	1000

表 3 各ストレージデバイスの入出力時間（秒）

	HDD	SSD	OLD	NEW
w3g	27.3	11.6	23.6	17.1
wrand	9.94	0.130	17.3	0.223
r3g	27.1	13.8	30.5	30.7
rrand	31.0	0.43	45.1	45.8

表 4 各空きセット数閾値設定下での I/O 処理の入出力時間（秒）

	10	100	1000
w3g	17.1	17.1	17.1
wrand	0.162	0.198	0.226

を超える 3GB の I/O を行うため，従来の方式では追い出し処理が必ず発生する．本発表での提案方式においても，キャッシュセットをすべて使い切り，HDD への直接 I/O に切り替わると予想される．wrand, rrand では，3000 回のランダムアクセスのほとんどがセットの大きさである 1MB 以上のオフセットでアクセスされるので，約 2000 個のキャッシュセットは途中ですべて割り当てられると予想される．

5.3 評 価 結 果

最初に，HDD，SSD，HDD に SSD の 2GB 分の領域を従来方式の SSD キャッシュとして適用したストレージ（OLD），HDD に SSD の 2GB 分の領域を本発表の提案方式の SSD キャッシュとして適用したストレージ（NEW）の 4 つに対して，4 つの I/O 評価プログラムを実行したときの I/O 完了までの時間を測定した結果を**表 3** に示す． また，特に w3g，wrand を OLD，NEW に行ったときのスループットの推移を**図 8**，**図 9** に示す．

次に，空きセット数の閾値を 10，100，1000 とした場合の各書き出し I/O 時の入出力時間を測定した結果を**表 4** に示す．なお読み込み I/O の入出力時間については閾値変更による変化はないため割愛する．

5.4 考 察

5.4.1 提案追い出し方式の有効性

表 3 の w3g, wrand の入出力時間の結果から，提案

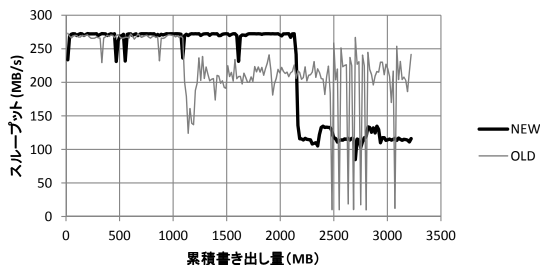

図 8 2GB の SSD キャッシュを適用した HDD に 3GB 書き出し（w3g）を行ったときのスループットの推移

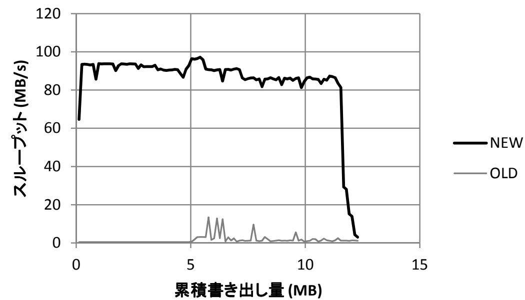

図 9 2GB の SSD キャッシュを適用した HDD に 4KB ブロックランダム書き出し（wrand）を行ったときのスループットの推移

方式を実装した NEW は，従来方式の OLD よりも短い時間で書き出しが行えていることがわかる．w3g では約 6.5 秒，wrrand では約 17 秒の I/O 時間の短縮になった．

図 8 は w3g を OLD と NEW で実行したときのスループットの推移を示している．これを見ると，OLD が 1GB ほどのデータを書き出してから，空きセット数が 1000 を下回り，追い出し処理により，スループットが低下しているのに対し，NEW は書き出し中に追い出しが実行されないため，2GB の SSD キャッシュが全部埋まるまで SSD のスループットで書き出しが行えているのがわかる．そして後半では，OLD の SSD キャッシュが埋まり，HDD へ直接 I/O を行い始めるとさらにスループットが急激に低下しているのに対して，NEW は SSD キャッシュが埋まった後も追い出しを実行せずにいるため，HDD への直接 I/O も安定したスループットで行われた．これらの結果から，提案した追い出し方式は従来の追い出し方式による I/O 要求処理の性能悪化を改善したと結論付けられる．

OLD の 2500MB 以降のスループットの急激な落ち込みは，空きセットの消費に追い出しが間に合わず，空きセットがなくなったことによるものだと考えられる．空きセットがなくなると，HDD へ直接書き出しを行うが，HDD は既に SSD からの追い出しデータの書き出しで負荷が高い状態にあり，そのため，スループットの急激な低下を招いたと推測される．

また，図 9 に示されたランダム書き出しにおいても NEW は高い性能を示すことができた．書き出しの最後で急激にスループットが低下しているのはランダムアクセス性能が低い HDD への直接アクセスが始まったためと考えられる．ランダムアクセスはセット単位の領域マッピングを行っている本ドライバにとっては無駄なマッピング領域を増加させる要因となり，空きセットをすぐに消費してしまうという問題点があるが，

空きセット数閾値を高めに設定し，空きセットを多く確保しておくことで一時的なランダムアクセスに対処することは可能である．

r3g，rrand はすべて読み込み I/O なので，SSD キャッシュ上にダーティデータは書き込まれず，追い出しの必要がない．OLD と NEW の違いは追い出しアルゴリズムのみであるため，r3g，rrand の I/O ではほぼ同じ結果が得られた．ただし，どちらも HDD で直接 I/O を行うよりも入出力時間が大きく，SSD へのデータのキャッシングによるオーバヘッドが依然として大きいことがわかる．このオーバヘッドへの対処は今後の課題である．

5.4.2 空きセット数の閾値の大きさによる性能への影響

表 4 では，空きセット数の閾値の大きさごとに I/O 処理を行った結果を示しているが，大きな違いはない．本提案方式では，閾値が大きいかどうかにかかわらず，I/O 処理を最優先に行い，追い出しは完全に後回しになるので，全ての場合で，まず空きセットを全て消費し，それから HDD へ直接 I/O を行うという同じ処理をしている．したがって性能にあまり差が出ない．

空きセット数の閾値が大きく性能に影響するのは SSD キャッシュ内の空きセット数が閾値に到達した後になる．この後は最大でも閾値までの数の空きセットしか確保されないので，新しいセットの割り当てが連続して起きた場合，閾値が小さいとすぐに空きセットが枯渇し，HDD へ直接アクセスすることになる．

表 4 のランダムアクセスでは閾値が小さいほうが性能が僅かによいことがわかる．これは閾値を空きセット数が下回り，追い出しのワークキューが動作を始めるタイミングが早いか遅いかでオーバヘッドの差が生じていると予想される．

6. おわりに

本発表では，SSD を HDD のディスクキャッシュとして利用してディスクアクセス性能を向上させることを目的に，本研究でこれまで設計，実装を行ってきた Linux デバイスドライバの機能について概説した．その上で，これまでの SSD ディスクキャッシュ上のダーティデータの追い出し方式では，追い出し処理によって I/O 要求処理の性能が低下する問題があることを示し，これを解決する新しい追い出し方式のアルゴリズムの設計と実装を行った．

この新たな追い出し方式を実装したシステムの評価では，追い出し処理の実行の遅延により，SSD，HDD への I/O が分散されることで I/O 要求処理性能が向上し，本方式は従来方式の I/O 性能を改善することが実証された．

また，評価では，読み込みのキャッシュミス時のオーバヘッドが依然として高いことも確認され，今後の課題として，このオーバヘッドの削減や，アクセスの統計情報に基づく選択的な割り当てアルゴリズムの適用が挙げられる．

謝辞 本研究は科学研究費補助金 基盤研究（B）「ユーザコンテキストに応じた電力管理による省電力コンピューティング環境の研究」の支援によるものである．

参 考 文 献

1) Matthews, J. et al.: Intel Turbo Memory: Non-volatile disk caches in the storage hierarchy of mainstream computer systems, *ACM Transactions on Storage*, Vol. 4, No. 2, pp. 1–24 (2008).
2) Sun Microsystems, Inc.: *Solaris ZFS Administration Guide, Part No: 819-5461* (2009).
3) Srinivasan, M.: Facebook / Flashcache, https://github.com/facebook/flashcache.
4) Makatos, T., Klonatos, Y., Marazakis, M., Flouris, M. D. and Bilas, A.: Using Transparent Compression to Improve SSD-based I/O Caches, *Proceedings of the 5th European conference on Computer systems* (2010).
5) 仁科圭介, 佐藤未来子, 並木美太郎: 省電力・高速化を目的とした SSD を用いたディスクキャッシュシステムのブロックデバイスドライバによる実装, 情報処理学会「システムソフトウェアとオペレーティング・システム」第 116 回研究報告, Vol. Vol.2011-OS-116, No. 6, pp. 1–8 (2011).
6) 仁科圭介: SSD ディスクキャッシュシステムの評価, *JSASS2011* (2011).
7) : Device-mapper Resource Page, http://sourceware.org/dm.