Vol.2010-ARC-189 No.13
Vol.2010-OS-114 No.13
2010/4/22

SSDをディスクキャッシュとして利用するLinuxブロックデバイスドライバ

仁 科 圭 介[†1] 並木 美太郎[†2]

SSD（Solid State Drive）は，HDDよりもランダムアクセス性能に優れ，消費電力も比較的小さいという特徴を持ち，HDDに代わる新しいストレージデバイスとして注目されている．本研究では，比較的低容量のSSDを，ディスクアクセスの高速・省電力化を目標として，ディスクブロックのキャッシュデバイスとして利用するシステムを実現するため，SSDへHDDのブロックをマッピング・管理するLinuxブロックデバイスドライバを実装した．本稿では本デバイスドライバの設計と実装および評価について述べ，本システムのディスクアクセスの省電力・高速化への効果について考察する．

A Linux Block Device Driver Using SSD Storage Device for Disk Cache

Keisuke Nishina[†1] and Mitaro Namiki[†2]

SSD (Solid State Drive) is a new kind of storage device that has advantages such as the random access performance, consumes smaller power than traditional HDD. In this study, a Linux block device driver using small SSD as a disk cache device for the HDD to high speed and the power saving of the disk access was designed and implemented. This paper describes the design, and the evaluation of the device driver that takes account of its effect of the power saving and speed-up of the disk access.

†1 東京農工大学
Tokyo University of Agriculture and Technology
†2 東京農工大学大学院共生科学技術研究院
Graduate school of Engineering, Tokyo University of Agriculture and Technology

1. はじめに

近年，不揮発性記憶素子の一種である，NAND Flashの技術発展により，大容量で，低価格なフラッシュメモリが普及してきている．このNAND Flashを主に用いて，従来のHDDなどのディスク装置と同等の機能を実現する記憶装置を特にFlash SSD（Solid State Drive）と呼び，通常は単にSSDと呼ぶことが多い．このSSDは，HDDよりもランダムアクセス性能に優れ，消費電力も比較的小さいという特徴を持ち，長年に渡り広く使われてきたHDDに代わる新しいストレージデバイスとして注目されている．一般消費者向け，エンタープライズ向けの製品が多くのベンダーから市場に投入され，その消費電力の小ささから，モバイル端末や，省電力サーバなどに利用が広がっている．また，一部の高速な製品は高速な動作を要求されるハイエンドPCに搭載されたり，サーバのボトルネック解消などに利用される例もある．

しかし一方で，SSDはHDDと比べると未だ容量単価が十倍から数十倍程度高く，SSDのみで大容量のストレージを用意するとなると，非常にコストがかかる．そのため大容量のストレージを必要とするシステムでは，容量の比較的小さなSSDと，コスト面で有利なHDDを組み合わせて，SSDの利点である省電力・高速性をより活かせるようにそれぞれのストレージデバイスにデータを分配する構成をとることが一般的である．しかし，ユーザ自身がデータの格納先デバイスを直接管理する場合，データ管理が煩雑化し，ユーザに非生産的な作業を強いることとなり，SSDの効率的な利用が困難となる．したがって，このようなシステムでは容量の限られたSSDを効率的に利用するための工夫が必要となってくる．

SSDを効率的に利用して，システムの高速・省電力化を図る既存の技術として，Intel Turbo Memory[1)]や，Solaris ZFS[2)]ファイルシステムの機能であるZIL（ZFS Intent Log），L2ARC（Level 2 Adaptive Replacement Cache）が挙げられる．

Intel Turbo Memory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，システムの高速化と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，Turbo Memoryに対応したIntelのチップセットと，Windows Vista/7の機能であるReadyBoost，ReadyDriveが必要である．

ZFSファイルシステムで利用可能であるZILは，同期書き込み実行時に書き込みログを格納する領域であり，これをSSDに適用することで同期書き込み実行時の性能低下を抑える．また，L2ARCは，ZFSのメモリキャッシュであるARCの二次キャッシュ領域であり，これをSSDに適用することによりシステムの高速化を図れる．しかし，これらはZFSファ

© 2010 Information Processing Society of Japan

Vol.2010-ARC-189 No.13
Vol.2010-OS-114 No.13
2010/4/22

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．

本稿では，Linux ブロックデバイスドライバにより SSD をディスクキャッシュとして利用し，ディスクアクセスを高速・省電力化するシステムを提案する．デバイスドライバによる管理を実現することにより，他の OS でもドライバの移植によって本方式の利用が可能であり，ブロックレベルでの実装によりファイルシステム独立なシステムが実現できる．本研究ではこのデバイスドライバを設計，実装し，実際に HDD と SSD を接続した Linux サーバ機上で提案システムを動作させて I/O 性能と消費電力についての評価実験を行った．この評価結果より，本提案システムの I/O 性能と電力への効果について考察する．

2. SSD の概要

SSD は NAND Flash チップ，DRAM などを用いたキャッシュメモリ，I/O を制御するコントローラなどの半導体回路で構成されており，従来の HDD のような駆動部は存在しない．そのため，動作が静粛であるのはもちろん，発熱が少なく，衝撃に強い．HDD では必要であったシーク動作，回転待ちといったレイテンシが発生しないため，その分ランダムアクセスが高速に行える．

SSD に用いられている NAND Flash はその記録方式の特性上，すでに何かが書き込まれている部分に直接データを上書きすることはできず，一旦書き込まれる部分のデータを消去してから書き込み動作を行う必要がある．そのため一般的に，同じ量のデータでは読み込みよりも書き込みの方が処理に時間がかかる[3]．

SSD の消費電力は 3.5 インチ HDD と比べて小さく，2.5 インチ HDD と比べても同程度かそれ以下であり，I/O も比較的高速なので，I/O 当たりの消費エネルギーで見ると，HDD よりも省エネルギーであると考えられる．

3. 目標と設計方針

SSD の利用における問題点として，SSD をそのままファイルストレージとして HDD と併用する場合，SSD を効率的に利用するためにはデータ管理が複雑化してしまうこと，また，SSD をシステムの高速化・省電力化に有効利用する Intel Turbo memory や ZFS の ZIL，L2ARC などの先行技術は，特定の OS やファイルシステムでのみ利用でき，適用できるシステムが限定されることが挙げられる．これらの問題を解決する手法を提案，実現することは，有用性が高いと考える．

そこで本研究では，容量で不足する SSD を，ディスクアクセスの高速化・省電力化を目的として HDD のディスクキャッシュとして利用するシステムを実現することを目標とする．また，このシステムが多様な計算機環境・OS に容易に適用可能な形での実装方式を目指す．

本試作システムでは，サーバから組み込み用途まで，幅広い計算機環境に利用されている Linux をターゲットとし，ブロックデバイスドライバによる実装とする．ブロックレベルでのキャッシュ管理を行うことで，どのようなファイルシステムにも適用可能となると同時に，デバイスドライバでの実装とすることで，本方式を他の OS に適用することも可能である．

本研究では，SSD のディスク I/O 当たりの消費エネルギーの小ささに着目し，ディスク I/O に関わるストレージデバイス全体の電力消費の抑制を優先的な目標とした．その中で，ディスク I/O 全体の高速性にも配慮する設計方針とした．消費電力の抑制のためには，SSD のヒット率を向上させ，I/O 当たりの消費エネルギーが大きい HDD へのアクセスを抑制する必要がある．そこで本デバイスドライバでは，SSD 内により利用頻度の高いデータブロックをマッピングすることにより，ディスクキャッシュのヒット率の向上を目指す．

SSD は不揮発性の記憶装置であるため，これを活かし，システムの再起動後もこの内容を再利用できるようにする．これにより再起動直後のディスク I/O 性能の向上が期待できる．これは，データブロックのマッピング情報を SSD 内に格納することで可能になる．

4. 設　　計

4.1 全体構成

SSD にファイルのデータを格納するための Linux で動作するブロックデバイスドライバを実装する．このデバイスドライバは，ファイルシステムからのディスク I/O 要求を受け取り，独自に管理するデータブロックのマッピング情報を参照して I/O 要求を実デバイスへ発行する．マッピング情報は SSD 内にも格納され，システムの再起動後もこれを読み込むことでマッピングの整合性を保つ．全体の構成を図 1 に示す．

本デバイスドライバでは，SSD と，SSD によるディスクキャッシュを適用したいブロックデバイスとを統合した仮想ブロックデバイスを提供する．図 2 にその概要を示す． システム構築者が指定した HDD のパーティション（例えば/dev/sdb1）と，SSD（全体でもパーティションでも可）を用いて新たなブロックデバイスを構築する．SSD はディスク I/O のキャッシュデバイスとなり，プログラマや一般利用者は意識せずに利用できる．本方式による仮想ブロックデバイス（仮想ディスク）は，SSD と統合する前の HDD パーティションと同容量の記憶領域と内容を持つデバイスとして定義される．このパーティションに ext3 などのファイルシステムが構築されていた場合，仮想ブロックデバイスにはこの内容がそのまま

ⓒ 2010 Information Processing Society of Japan

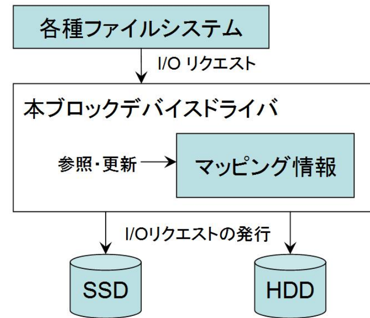

図 1　システムの全体構成

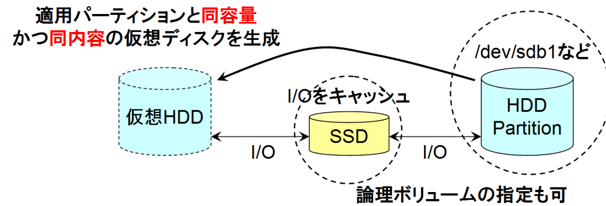

図 2　ブロックデバイス仮想化の概要

反映され，ユーザはこの仮想ブロックデバイスをマウントしてそのまま元々HDD パーティションに構築されていたファイルシステムを利用することができる．また，SSD 側，HDD 側双方に論理ボリュームを指定することも可能とした．これにより，システムの規模によって柔軟に仮想ブロックデバイスとディスクキャッシュの構成を変更できる．

本デバイスドライバは基本的には，仮想ブロックデバイスへのディスク I/O リクエストの到着によって，図 3 のフローチャートに示す処理を行う．ライトバック/ライトスルーの書き込みポリシーは，仮想ブロックデバイス作成時にシステム構築者が指定する．ライトバックポリシーを用いることで，HDD への書き込みの抑制が期待でき，省電力に効果があると考えられる．しかし，データの変更が非同期に HDD に書き込まれるため，システムが突然クラッシュした場合，変更データが失われてしまう．ライトスルーを指定すると，書き込みは HDD のみに行われるため信頼性は向上するが，HDD への書き込み量がライトバックよりも多くなり電力消費量が増加すると考えられる．しかし，書き込み性能がそれほど高くない SSD を使用した場合，ライトスルーとしたほうが，かえって書き込み性能が向上するとも考えられる．

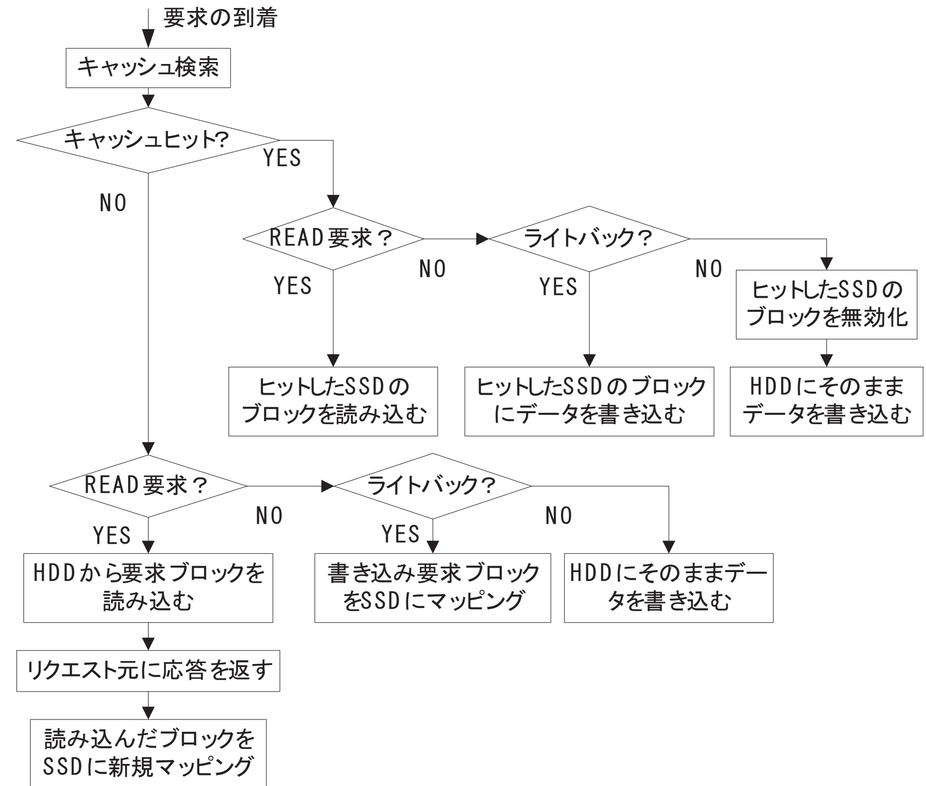

図 3　リクエスト処理の流れ

4.2　ブロックマッピング管理構造

本デバイスドライバでは，ディスクキャッシュとして用いられる SSD への HDD のデータのマッピングを一定サイズのブロック単位で行う．このブロックの大きさは，仮想デバイスの生成時に，1 セクタの 2 の冪乗の大きさを設定可能とする．

Linux では，主記憶を 4KB 単位で管理することが多く，ディスク I/O もこの単位で行われることが多いと考えられる．また多くのファイルシステムは，ファイルを 4KB 単位で管

© 2010 Information Processing Society of Japan

```
struct cacheblock {
    sector_t block; // マッピングされた HDD のブロック番号を格納
    unsigned short state; // ブロックの状態フラグ群を格納
    spinlock_t lock; // ブロック I/O リスト処理の排他制御用スピンロック
    struct bio_list bios; // 処理中のブロック I/O のリストを格納
};
```

図 4 cacheblock 構造体

理している．したがってこの場合，4KB が SSD のブロックの管理単位として適切であると推測される．

しかし，利用されるアプリケーションのアクセスパターンによっては，より大きなブロックにした方が効果的な場合もある．例えば，比較的大きいシーケンシャルなデータアクセスが多い場合，大きなブロックサイズを設定することで，先読みが効果的に働く．

この SSD 上のブロックごとに，マッピングされた HDD のブロック番号やブロックの現在の状態（割り当て，有効，ダーティ，書き戻し中など）を表すフラグなどのマッピング管理情報（メタデータ）が必要である．このメタデータは，SSD 内にも格納されるが，システム動作時は主記憶上で管理され，メタデータの変更は仮想デバイス削除時に SSD に書き出される．各ブロックごとのメタデータは，C 言語の構造体を用いて次のように定義される．

本デバイスドライバでは，この構造体の配列を用いて，SSD の全ブロックの管理を行う．つまり，この配列の要素数は SSD 内のブロック数と一致し，配列のインデックスは SSD のブロック番号と対応する．このメタデータのうち仮想ブロックデバイス削除時に SSD 内に格納されるメタデータは，block と state の組となる．これも SSD のデータブロック番号をインデックスとする配列として格納される．

マッピング情報の管理構造は，ブロック検索時や LRU による置き換えブロックの選択時の性能に大きく影響する．本デバイスドライバでは，マッピング情報の管理に，ハッシュ表と LRU リストを用いることで，ブロック検索やブロックの置き換えにおける時間計算量をほぼ O(1) で行えるようにした．図 5 にマッピングの管理構造を示す．

I/O 要求発生時，このハッシュ表によりマッピング先の SSD のブロックを検索する．また，メタデータは，双方向リスト構造による LRU リストでも管理され，ミス時のブロック置き換えに利用する．SSD 内のキャッシュブロックの管理に用いるデータ構造をまとめると C 言語様の擬似コードで図 6 のとおりとなる． ハッシュ表の大きさについては，小さければその分同じハッシュ値となる論理ブロック番号が確率的に増え，チェインが長くなりやすくなり，検索にかかる時間計算量も増加する．そのため，できるだけ大きいほうがよい．本試作では，エントリー数を 2M 個とした．ハッシュ表のチェインや，LRU リストは全て配列を用いて双方向連結リストを実現する．これによりポインタを使用するよりも主記憶の使用量を抑えられる．以下，図 6 で示した変数名を用いてアルゴリズムを説明する．

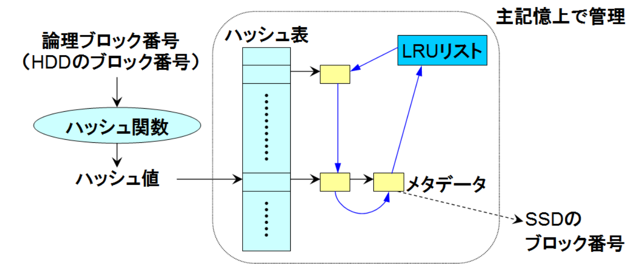

図 5 ハッシュ表と LRU リストによるマッピング管理

```
hash_size = ハッシュ表の大きさ（2 の冪乗）;
size = SSD 内に確保できる総キャッシュブロック数;
struct cache_c {
    struct cacheblock cache[size]; /* メタデータ */
    int hash_table[hash_size]; /* ハッシュ表 */
    int chain_prev[size]; /* チェインで前にあるブロックのインデックス */
    int chain_next[size]; /* チェインで次にあるブロックのインデックス */
    int lru_prev[size]; /* LRU リストで前にあるブロックのインデックス */
    int lru_next[size]; /* LRU リストで次にあるブロックのインデックス */
    int lru_head; /* 最後の利用時間が最も古いブロックのインデックス */
    int lru_tail; /* 最後の利用時間が最も新しいブロックのインデックス */
    unsigned int index; /* マッピング時に利用 */
};
```

図 6 SSD キャッシュブロック管理データ構造

本デバイスドライバは，仮想デバイスにブロック I/O 要求が到着したとき，I/O 要求の開始セクタ番号から，要求部分を含んだ論理ブロック番号を計算．そのハッシュ値によりハッシュ表を検索し，チェインをたどりながら論理ブロック番号の一致するブロックを検索する．チェインが長くならなければ，このアルゴリズムによりほぼ O(1) の時間計算量で

ⓒ 2010 Information Processing Society of Japan

```
size = 総キャッシュブロック数;
index = 現在マッピング済みのブロック数;
if (index < size) { // まだ一度もマッピングされていないキャッシュブロッ
クが存在
    cache_block = index;
    index++;
} else { // 全てのブロックが一度以上マッピングされた
    this = lru_head;
    while (this > -1) {
        if(is_state(cache[this].state, WRITEBACK ||
            is_state(cache[this].state, RESERVED)) {
            // 現在ライトバック中またはマッピング中のブロックは選ばない
            this = lru_n[this];
        } else if (is_state(cache[this].state, DIRTY)) {
            cache[this] のブロックをライトバック;
            return 2; // マッピングは行わない
        } else break; // this がマッピング先
    }
    if (this < 0) {
        // 全てのブロックが現在処理中のためマッピングはしない
        return -1;
    }
    cache_block = this;
}
return 1; // cache_block にマッピング
```

図 7 ブロックマッピングアルゴリズム

キャッシュ検索が可能である．

次に，SSD に新たに HDD のブロックをマッピングするときのアルゴリズムを述べる．次の図 7 にこのマッピング先決定アルゴリズムの擬似コードを示す． 検索がヒットしない場合は，SSD 上の未割り当てブロックを新たに割り当てる．このとき，最もブロック番号の若い未割り当てブロックを割り当てる．これにより，アプリケーションのアクセス順序に従って SSD 上にブロックがマッピングされやすくなり，以降の読み込み時に SSD 内コントローラによる先読みが有効に働きやすくなると考えられる．

SSD のブロックを上から順にマッピングしていくと，遂には SSD の全てのブロックがマッピング済みとなる．この後に I/O 要求がミスした場合，すでに HDD の他のブロックがマッピングされている SSD のブロックを，I/O 要求部分を含んだ新しいブロックをマッピングしなおす必要がある．この置き換えブロックは，LRU リストの先頭にあるブロックとなる．ただし，このブロックがダーティ（HDD にデータの変更が同期されていない）ブロックだった場合，HDD への内容の書き出し（ライトバック）が必要となる．この場合，このダーティブロックのライトバックのみを行う．ディスク I/O 処理が集中し性能が低下するため，新しいブロックのマッピングは行わなず，I/O 要求をそのまま HDD へ発行する．

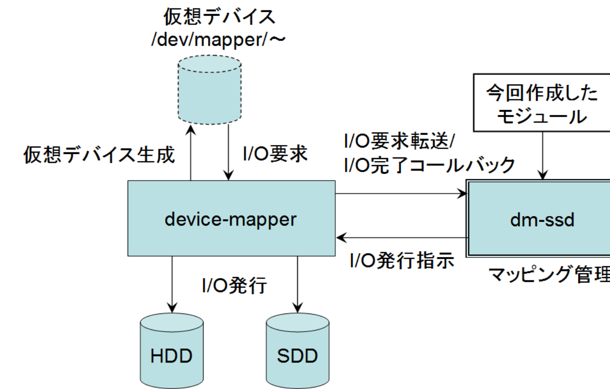

図 8 本デバイスドライバの全体構成

5. 実　　装

本デバイスドライバは，ブロックデバイスの仮想化を device-mapper[4] のフレームワークを活用して実現する．device-mapper は，仮想ブロックデバイスの作成，管理に利用できる，ブロックデバイスドライバおよびそれをサポートするライブラリ群であり，Linux 2.6 系で利用できる．本デバイスドライバでは，仮想デバイスの生成，I/O 要求の受け付け，I/O の実デバイスへの発行などの機能を，この device-mapper により実現し，マッピングの管理や，device-mapper へ I/O 発行を指示する部分をカーネルモジュール「dm-ssd」として新しく作成した．本デバイスドライバの実装の全体構成の概略を次の図 8 に示す．

device-mapper は，生成した仮想デバイスへの I/O 要求をファイルシステムから受け取ると，それを dm-ssd に転送し，dm-ssd の指示で，実デバイスへの I/O 発行を行う．また，I/O 完了後，dm-ssd から指定されたコールバック関数に処理を戻す．

dm-ssd は，マッピング情報を管理し，device-mapper から受け取った I/O 要求の内容と

© 2010 Information Processing Society of Japan

マッピング情報から，必要な実デバイスへの I/O 要求を構築し，device-mapper に指示する．また，I/O 完了後に呼び出されるコールバック関数により，ファイルシステムへ I/O 完了の応答なども行う．

6. 評　価

本デバイスドライバを実装した Linux を用いて実機上での評価を行った．本章ではその評価方法と結果について述べる．

6.1 評価方法

実装した計算機の諸元は次のとおりである．

- **プロセッサ**　Intel Xeon E5450 3GHz
- **主記憶**　DDR2-677 16GB
- **インタフェース**　3Gb/s SATA
- **OS**　Linux 2.6.31 x86_64

この計算机上で，本デバイスドライバによって作成した仮想ブロックデバイスと通常の HDD，そして通常の SSD のそれぞれに同じ入出力処理を行い，その入出力時間を計測した．それと同時にその間のストレージデバイスの平均消費電力を測定した．なお消費電力は，各ストレージデバイスに接続した電源ケーブルを流れる電流の電流値を非接触直流電流計で計測した値と，各ケーブルの定格電圧を積算して求めた．

今回の評価に用いたストレージデバイスを表 1 に示す．また，今回の評価のために行った I/O 処理を次に示す．

- **Read 1G**　1GB のファイルの内容を読み込む
- **Reread 1G**　Read 1G の後に再び同じファイルを読み込む
- **Write 1G**　1GB の一つのファイルを書き出す
- **Rewrite 1G**　Write 1G の後に同じファイルを 1GB 上書きする
- **Read at random**　10GB の範囲内でランダムに選んだ 1000 箇所で 1KB 読み込む
- **Reread at random**　Read at random と同じアクセスパターンで再度読み込む

表 1 評価に用いたストレージデバイス

デバイス	概要
2.5 インチ SSD	32GB SLC
2.5 インチ HDD	160GB 5400rpm
3.5 インチ HDD	250GB 7200rpm

最初の四つはシーケンシャルな比較的大きな単位のディスク I/O についての実験であり，後の二つはランダム読み込みについての実験となる．Reread（再読み込み），Rewrite（再書き込み）は，SSD によるディスクキャッシュの効果を調べるために行った．

また，入出力を行うファイルシステムはすべての場合において EXT3 とし，マウントオプションには sync（同期書き込み）を指定した．sync オプションを指定しない場合，ファイルの書き込みはまず主記憶上にキャッシュされ，遅延されて実デバイスに書き込まれるため，入出力時間の測定が難しい．sync オプションを指定することで書き込みデータはその書き込みの実行時にすべて実デバイスに同期される．

マッピングの設定は，最初の四つの実験では 1GB という大きなサイズのファイルの読み書きを行うため，先読みが有効に働くようにブロックの大きさを 128KB に設定した．後の二つの実験は細かいランダムリードを行うため，ブロックの大きさを 4KB に設定した．また，書き込みポリシーはすべての場合でライトバックとした．

6.2 実験結果

実験によって得られた入出力時間と平均消費電力の測定結果を示す．

6.2.1 1GB ファイル読み込み時の入出力時間と消費電力

まず，ファイル書き込み時の入出力時間と消費電力のそれぞれの測定結果を次の図 9，10 に示す．2.5HDD，3.5HDD，SSD というラベルが付いているものは，そのデバイスそのものをマウントして 1GB のファイル読み込みを行った結果を示している．また，2.5HDD & SSD とラベルが付いているデータは，その HDD に SSD によるディスクキャッシュを適用した仮想デバイスをマウントし，1GB のファイル読み込みを行った結果である．仮想デバイスの消費電力については，これに適用された SSD と HDD の二つのデバイスの平均消費電力を足した値を示している．

入出力時間については，初めてファイルを読み込みたときは，HDD をそのまま利用した場合に比べより時間がかかっているが，二回目では，それが逆転し，仮想デバイスの方が HDD をそのまま利用するよりも高速に読み込みが行えた．また，SSD をそのまま利用した場合と比較しても，読み込みにかかった時間はほぼ同じであり，SSD によるディスクキャッシュがうまく働いていることを示している．結果より，2.5 インチ HDD をそのまま利用するよりも SSD によるディスクキャッシュを適用することでおよそ 36%読み込み時間が削減された．

消費電力については，本試作システムによる仮想デバイスは，HDD をそのまま利用した場合に比べ，一回目の読み込みでは，HDD からの読み込み SSD への書き込みがあるため

© 2010 Information Processing Society of Japan

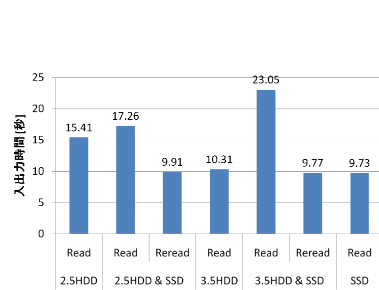

図 9 1GB のファイル読み込み時の入出力時間

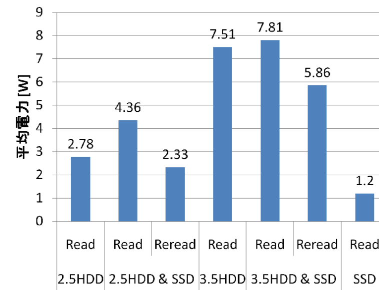

図 10 1GB のファイル読み込み時の平均消費電力

電力が大きくなっている．しかし，再読み込みでは，HDD そのままよりも平均電力が低くなっている．

これらの結果から，この 1GB のファイル読み込み全体の入出力処理に関係するストレージデバイスで消費された総エネルギーを平均電力×入出力時間（単位：ジュール）で求められる．この結果を，図 11 に示す．消費エネルギーのグラフより，本試作デバイスドライバによる仮想ブロックデバイスは，最初のデータ読み込み時は，HDD をそのまま利用するよりも多くのエネルギーを消費するが，同じデータを再度読み込みたときには消費エネルギーが逆に小さくなることがわかる．特に，2.5 インチ HDD に本システムを適用した場合は，HDD をそのまま利用する場合に比べておよそ 46%の消費エネルギーが削減できた．

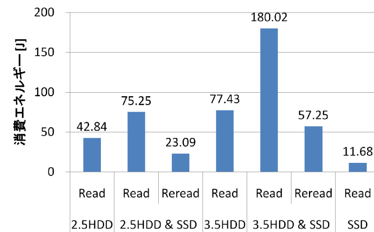

図 11 1GB のファイル読み込み時のストレージデバイス全体の消費エネルギー

6.2.2 1GB ファイル書き込み時の入出力時間と消費電力

次に，1GB のファイルの書き込み時の消費電力と入出力時間について測定を行った．入出力時間の結果を図 12 に，消費電力についての結果を図 13 にそれぞれ示す．また，これらの結果より，計算される総消費エネルギーを図 14 に示す．

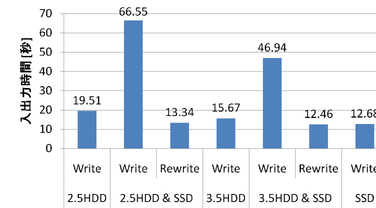

図 12 1GB のファイル書き込み時の入出力時間

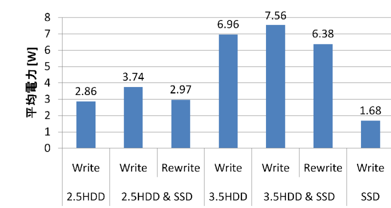

図 13 1GB のファイル書き込み時の平均消費電力

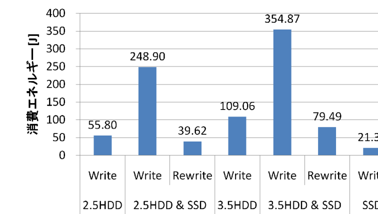

図 14 1GB のファイル書き込み時のストレージデバイス全体の消費エネルギー

入出力時間については，新規書き込み時の仮想デバイスの入出力時間が非常に長い．この理由について考えられるのは，キャッシュミス時の HDD からのブロックの読み込み待ちである．キャッシュミス時，本デバイスドライバでは，新しく書き込み要求のあったブロックを SSD に割り当てるため，HDD から該当ブロックの必要な部分を読み込む．これが読み込み終わって初めて割り当てられたキャッシュブロックへ要求データを書き出すことができる．したがって，この間，そこに書き込まれるべきデータは HDD からブロックが読み込まれてくるのを待たなければならず，処理に時間がかかると推察される．上書きでは，どの仮想デバイスも SSD をそのまま利用した場合とほぼ同等の性能を示しており，新規書き込みの速度が遅いのは SSD 側の問題ではないと思われる．

次に平均消費電力について見ると，仮想デバイスの平均電力は，読み込み時と同じように，最初の書き込み時に HDD をそのまま利用した場合に比べて大きくなっている．次の上書き時には，3.5 インチ HDD を使った仮想デバイスは 3.5 インチ HDD そのものを利用した場合に比べるとわずかに低くなったが，2.5 インチ HDD を使った仮想デバイスは，HDD

© 2010 Information Processing Society of Japan

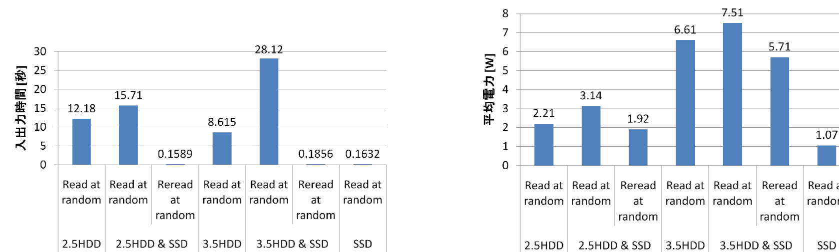

図 15　1KB × 1000 のランダム読み込み時の入出力時間

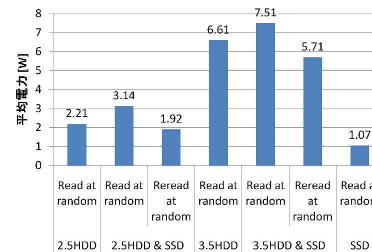

図 16　1KB × 1000 のランダム読み込み時の平均消費電力

をそのまま利用する場合よりも逆に平均消費電力が大きくなった．これは SSD の書き込み動作時の平均消費電力が読み込み動作時よりも高いためである．

消費エネルギーについては，やはり新規書き込み時における仮想デバイスの消費量が大きい．しかし，上書きにおいては，SSD の書き込み性能が他より優れているため HDD をそのまま利用するよりも若干少ないエネルギー消費となった．

6.2.3　ランダム読み込み時の入出力時間と消費電力

10GB 分の領域からランダムに 1000 箇所選んでそれぞれ 1KB を読み込むプログラムを実行し，その間の平均消費電力と入出力時間についてそれぞれのデバイス上で測定した．仮想デバイスに対しては，再び同じアクセスパターンの読み込みを行い，SSD によるディスクキャッシュの効果を測定した．図 15 に入出力時間の測定結果を，図 16 に平均消費電力の測定結果を，図 17 にこれらの結果から計算した消費エネルギーをそれぞれ示す．　入出力時間では，仮想デバイスは最初の読み込みに時間がかかる傾向にある．特に 3.5 インチ HDD を用いた仮想デバイスでは，HDD をそのまま利用する場合よりも 3 倍以上の時間がかかった．しかし，再読み込みでは，仮想デバイスは 2 種類とも 0.2 秒以下という SSD ならではのランダムアクセスの高速性を発揮した．

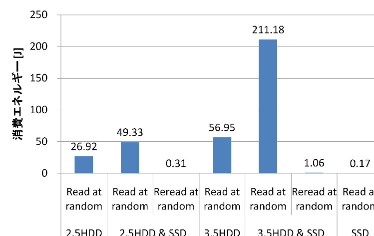

図 17　1KB × 1000 のランダム読み込み時のストレージデバイス全体の消費エネルギー

平均消費電力については，すべての項目が 1GB 読み込みの測定結果より約 4%から約 28%小さい値となった程度で，特に大きな変化は見られなかった．

消費エネルギーでは，再読み込み時の消費エネルギーが HDD と比べて圧倒的に小さく，このグラフだけでもランダム読み込みの多いシステムへの SSD の適用が消費電力削減に有効であることを示しているといえる．

7. おわりに

本稿では，SSD をディスクキャッシュとして用いる Linux ブロックデバイスドライバの設計と実装について述べ，これを実装した実機上で消費電力の削減と I/O 性能の向上への効果について評価実験を行った．その結果，キャッシュヒット時には入出力時間，消費エネルギーともに HDD のみのシステムよりも削減でき，特にランダム読み込みにおいて顕著な入出力時間の短縮と消費エネルギーの削減効果が確認できた．しかし，キャッシュミス時は逆に入出力時間，消費エネルギーともに大きく増加した．特に入出力時間の増加はアルゴリズムに問題があると考えられ，改善の余地があると言える．また，入出力処理時の消費エネルギーを削減できても，待機時の電力は SSD と HDD の待機電力の和となるため，待機時間の割合が高ければ全体的な省電力効果は薄まることになる．したがって，HDD のスピンダウンなどの省電力機構との連携によるさらなる消費電力削減が今後の課題として挙げられる．

参 考 文 献

1) J., M., S., T., D., H., R., C. and K., G.: Intel TurboMemory: Nonvolatile 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, I.: *Solaris ZFS Administration Guide, Part No: 819-5461* (2009).
3) Cooke, J.: Flash Memory Technology Direction, Technical report, Windows Hardware Engineering Conference (2007).
4) : Device-mapper. http://sourceware.org/dm.

© 2010 Information Processing Society of Japan