Vol.2012-ARC-200 No.5
Vol.2012-OS-121 No.5
2012/5/7

SSDをディスクキャッシュとして利用するディスクI/Oの高速・省電力化手法の提案

仁科 圭介[1]　坂本 龍一[1]　佐藤 未来子[1]　並木 美太郎[1]

概要：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 SSD を HDD のディスクキャッシュとして用いるための SSD ディスクキャッシュシステムを提案してきた．本論文では，SSD ディスクキャッシュシステムへ新たに追加した HDD の電源制御方式を提案する．これまでに Linux デバイスドライバへ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，HDD の待機時間を検出して自動的に HDD をスピンダウンして省電力化を図る方式の設計，実装を示す．評価により，各種ファイルシステム上の SSD ディスクキャッシュシステムにおいて，HDD のスピンダウンによりエネルギー効率を改善することを確認した．

キーワード：SSD, ディスクキャッシュ, デバイスドライバ, 省電力管理, I/O 管理

Design of the Power-saving Technique for Disk I/O Using an SSD as a Disk Cach

Nishina Keisuke[1]　Sakamoto Ryuichi[1]　Sato Mikiko[1]　Namiki Mitaro[1]

Abstract: 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. In our previous study, the SSD disk-cache system has been proposed which uses SSD as a disk cache of HDD. This paper proposes the power supply control system of HDD newly added to the SSD disk-cache system. The function of this Linux device driver is shown in this study, and the design of a power-saving system and implementation are shown, which detect the standby time of HDD and carry out the spin down of the HDD automatically. In the evaluation, energy efficiency has been improved by the spin down of HDD in the SSD disk-cache system on various file systems.

Keywords: SSD, Disk Cache, Device Driver, Energy Management, I/O Management

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ストレージデバイスとして Flash SSD（Solid State Drive: SDD）が普及している．SSD は，ハードディスクドライブ（Hard Disk Drive: HDD）と比較して優れたアクセス性能を発揮し，消費電力も比較的小さいという特徴を持っているが，容量単価が HDD の十倍以上と高価なデバイスである．そのため，ノート型 PC などの比較的ディスク容量の少ない計算機では，HDD を SSD に置き換えることで，ディスク I/O 性能の向上と省電力化を実現しているが，サーバ系の計算機ですべての HDD を置き換えることは現実的とはいえない．そのため，**図 1** に示すように，主記憶と二次記憶装置との間の新たな記憶階層として SSD を活用する方式が提案されている [2][3][4][5][6].

本研究ではこれまで，SSD をディスクキャッシュとして

[1] 東京農工大学
Tokyo University of Agriculture and Technology

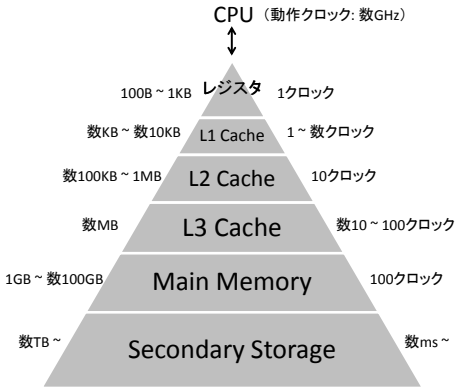

図 1 一般的な記憶階層の例

主記憶と二次記憶装置との間で一時的な格納領域として適用する方式を提案している [7]．既発表では，SSD をディスクキャッシュとして利用することで，ディスクアクセス性能が改善することを示した．また，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を改善したことにより，HDD の待機時間が増加することを確認した．

本発表では，本研究の SSD ディスクキャッシュを適用するシステムにおいて，HDD の待機時間を—OS—が検出して自動的に HDD をスピンダウンする方式を提案する．本論文では HDD スピンダウン方式の設計と実装を述べるとともに，本提案システムと機能面で競合するシステムである Flashcache との比較評価を示し，各種ファイルシステムでの動作確認とスピンダウン実行時の省電力効果について示す．

2. 研究の背景

2.1 既存の SSD を利用した省電力化手法

従来より，SSD をディスクキャッシュとして利用して，ディスク I/O の高速化や省電力化を図る方式が提案されている．以下，既存方式の特徴を示す．

Intel Turbo Memory[2] 専用の SSD デバイスとチップセットを用いて，SSD デバイスをディスクキャッシュとして導入し，ディスクアクセスの高速化と HDD へのアクセス低減とスピンダウンを組み合わせて省電力化を図る技術であるが，ハードウェアの制約がある．

Solaris ZFS ファイルシステム [3] Solaris の ZFS ファイルシステムにおいて，L2ARC（Level 2 Adaptive Replacement Cache）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（ZFS Intent Log）と呼ばれる書き込みログ領域へ設定するボリュームに SSD を用いることで，ファイル入出力の高速化を図る方式である．ZFS ファイルシステム以外では利用できない．

Flashcache[4] SSD を HDD のディスク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，4KB〜16KB の固定サイズのブロック一つ当たり 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[5] Linux のブロック I/O レイヤーで 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，SSD への格納容量の効率化実現する．ファイルシステムやストレージデバイスには依存しないが，オンラインで行うデータブロックの圧縮・解凍にかかる CPU オーバヘッドとのトレードオフが発生する．

SieveStore[6] 過去の一定期間のアクセス履歴を分析して，再利用される可能性の高いデータブロックのみを SSD に割り当てる方式である．SSD へのデータブロック割り当て量を大幅に削減し，同時に SSD のヒット率の向上を実現しているが，アクセス履歴を保持する記憶領域の確保と分析作業が必要となる．

2.2 本研究の目的

前節で述べた既存の SSD ディスクキャッシュ方式では，ディスク I/O の高速化を目的としたものが多い．Intel Turbo Memory のように，専用のハードウェアシステムで構成して省電力化を追及したものはあるが，任意のファイルシステムやストレージデバイスから利用できる SSD ディスクキャッシュの省電力化については提案されていない．

そこで，本研究では，任意のブロックデバイスやファイルシステムへの適用が可能で，ディスク I/O の高速化とともに省電力を実現する SSD ディスクキャッシュ管理方式を提案することを目的とする．本研究ではこれまで，HDD 上のファイルシステムの変更なしに SSD をディスクキャッシュとして利用できる，Linux デバイスドライバによる SSD ディスクキャッシュ方式を提案している [7]．従来方式よりも SSD の管理に要する計算機資源を抑えつつ，ディスクアクセス性能を改善できることを示した．また，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の改善により，HDD の待機時間が増加するということを確認し，このタイミングで HDD のスピンダウンを行えば，省電力化が期待できる見込みを得た．そこで本発表では，Linux へ設けたデバイスキャッシュドライバにおいて，HDD が待機状態であることを検出する機構を追加し，状況に応じた HDD のスピンダウンを自動的に実施する手法を提案する．以下，本論文では，この HDD スピンダウン手法の設計と実装を述べるとともに，本提案システムと機能面で競合するシステムである Flashcache との比較評価，各種ファイルシステムでの動作確認とスピンダウン実行時の省電力効果について述べる．

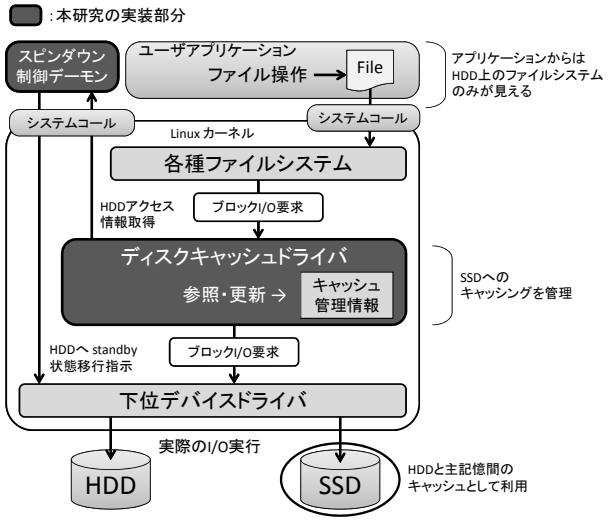

図 2 システムの全体構成

3. SSD ディスクキャッシュシステムの設計

本章では，これまで本研究で実現した SSD ディスクキャッシュシステムの設計を示し，本研究で新たに提案する SSD ディスクキャッシュシステムにおける HDD の電源制御の設計について述べる．

3.1 ディスクキャッシュシステムの全体構成

本システムの全体構成を**図 2** 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，HDD と主記憶の間のディスクキャッシュとして SSD を利用しながら I/O 要求を処理する．本システムではディスクキャッシュドライバを，ブロック I/O 要求を扱うレイヤーで実装したことにより，ファイルシステムに依存せずに利用できる．また，本ディスクキャッシュドライバでは，HDD のスピンダウン実施のタイミングを見極めるのに必要な各種統計情報を保持する．本研究では，本統計情報に基づいて HDD のスピンダウンを指示する「スピンダウン制御デーモン」を新たに設けた．スピンダウン制御手法の詳細設計については 3.4 節で述べる．

3.2 ディスクキャッシュドライバ

ディスクキャッシュドライバは，HDD と SSD の組を単一のブロックデバイスとして仮想化し，ファイルシステムやユーザアプリケーションへ提供する．以下，本研究ではこの仮想ブロックデバイスのことを「SSD キャッシュ適用ブロックデバイス」と呼ぶ．本ディスクキャッシュドライバは，SSD に対するディスクブロック単位でのキャッシングを管理する役割を担い，以下の機能を備える．

- SSD キャッシュ適用ブロックデバイスの作成／削除
- SSD キャッシュ適用ブロックデバイスへの I/O 要求の処理
- SSD のキャッシュ管理

SSD キャッシュ適用ブロックデバイスは，HDD の内容や容量をそのまま引き継ぐものとする．すなわち，HDD に既存のファイルシステムが構築されている環境へ SSD のディスクキャッシュを適用した場合，SSD キャッシュ適用ブロックデバイスを通してシームレスに HDD へアクセスできる．これにより，本ディスクキャッシュシステムは，任意のファイルシステムで適用可能である．

3.3 キャッシュ管理方式

本論文では特に断らない限り，「キャッシュ」とは SSD 内に本ドライバが管理する，SSD キャッシュ適用ブロックデバイスに対して発行されるブロック I/O 要求の対象データをキャッシュする領域のことを指す．キャッシュをどのように HDD 領域と対応付け，管理するかはキャッシュのヒット率に関わる重要な要因となる．本研究と同じ，Linux デバイスドライバによるディスクキャッシュ管理方式を用いている Flashcache では，HDD のセクタ番号からハッシュ関数を用いて，SSD の割り当て先のキャッシュ領域を求める方式を採用している．しかしこの方式は，ハッシュの衝突によりキャッシュの入れ替えが発生しやすく，個々のキャッシュされたデータがキャッシュに存在する期間が小さくなり，ヒット率が低下すると考えられる．そこで本ディスクキャッシュドライバでは，新たな HDD 領域とキャッシュの対応付けを行うときに，まだ対応付けられていない任意のキャッシュ領域を対応付けられるようにして衝突を防ぎ，ヒット率の向上を図る．

しかし，ハッシュ関数を使用しないため，そのままではキャッシュの検索時に全キャッシュを対象に割り当てを走査する必要があり，計算量が増大する．そこで，HDD の領域がどのキャッシュ領域に対応付けられているかを管理する対応表を用意し，これを利用することでキャッシュ検索を $O(1)$ で行えるようにする．

また，「セット」と「ブロック」という単位を用意し，領域の対応付けと，データの割り当ての単位を切り離して管理することで，データ割り当てにかかるオーバヘッドを最小に維持しつつ，対応表の管理コストを調節可能にする．各単位は次の通りに定義される．

ブロック SSD への実データ割り当ての最小単位．2^N セクタで指定（N は自然数）．

セット HDD 領域と SSD キャッシュ領域の対応付けの単位．2^M ブロックで指定（M は自然数）．

2 のべき乗の大きさにそれぞれすることで，セット番号とセット内のブロック番号が，セクタ番号からそれぞれシフト演算とビット演算により求められる．

本ディスクキャッシュドライバにおけるキャッシュ管理

の概念図を図 3 に示す．ディスクキャッシュドライバでは，HDD の領域をセット単位に区切り，セット番号により管理する．この HDD 上のセットのことを「HDD セット」と呼ぶ．SSD 上には HDD セットに対応付けるセットを管理し，これを「キャッシュセット」と呼ぶ．キャッシュセットは，同時に一つの HDD セットに対応付けられ，複数の HDD セットと対応付けられることはない．HDD セットとキャッシュセットの対応付けは対応表によって管理することで，任意のセットの対応付けを行える．

キャッシュセット内のブロックを「キャッシュブロック」と呼ぶ．領域の対応付けはセット単位で行うが，セット内のデータの実際の割り当ては，ブロック単位で行う．

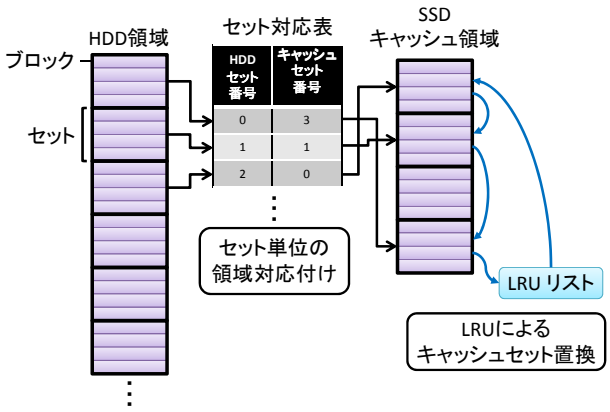

図 3 SSD キャッシュ管理の概要

LRU リストには，HDD セットに対応付けられているすべてのキャッシュセットが一列のリストとして LRU (Least Recently Used) 順にソートされて管理される．本研究では，キャッシュの置換が必要になったときは LRU リストの先頭にあるキャッシュセットを置換対象にすることで，より頻繁にアクセスされるキャッシュセットを SSD に残し，SSD の利用率の向上を目指す．

3.4 SSD ディスクキャッシュシステムにおける HDD 電源制御手法

一般的な HDD は通常のデータの読み書きが行えるアクティブ状態の他に，ホストシステムからの指示によって，スタンバイ（standby）またはスリープ（sleep）と呼ばれる省電力状態に移行する機能を持つ．これらの状態のときは，一般的に HDD 内部のディスクの回転速度を低下させるか，完全に停止させ消費電力を抑える．これをスピンダウンと呼ぶ．スピンダウン中はデータの読み書きができないため，再びデータの読み書きを行う場合は，ディスクの回転を通常の状態に戻す必要がある．これをスピンアップまたはウェイクアップと呼ぶ．ウェイクアップ時は，ディスクを加速させるために大きな電力を消費する他，回転速度がアクセス可能な状態に安定するまで待ち時間が発生する．

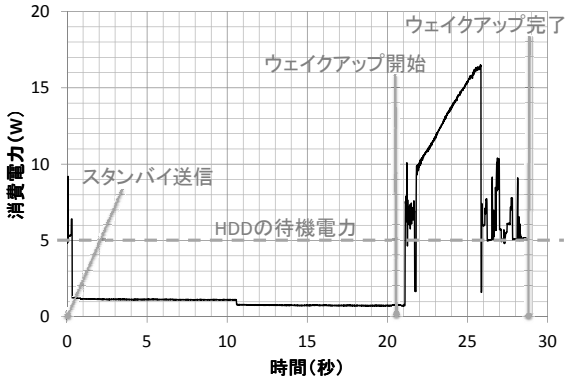

図 4 評価用 HDD のスピンダウンとウェイクアップ時の消費電力の推移

HDD のスピンダウンによる省電力化手法は Intel Turbo Memory や参考文献 [8] などでも用いられているが，スピンアップ時に大きなエネルギーを消費するため，いかにスピンダウンした状態を長く維持してこのエネルギーを償却するかが課題である．実際に本論文の評価で用いた HDD をスピンダウン・スピンアップさせたときの HDD の消費電力の推移を**図 4** に示す．この HDD では，ウェイクアップ時に消費する分のエネルギーを削減するために，約 18.6 秒のスピンダウン期間が必要であることが測定結果から判明した．本ディスクキャッシュシステムでは，書き込み I/O については，SSD に空きがある限り，一旦 SSD に書き込むことで，HDD への書き込み I/O を遅延させ，これにより HDD のスピンダウン期間の延長を図る．

HDD のスピンダウンは通常，HDD に割り当てられたデバイスファイルに対して，ユーザアプリケーションから ioctl システムコールにより HDD を standby または sleep 状態に移行するコマンドを発行することで実現される．そこで本研究でもこのシステムコールインタフェースによる HDD 制御を行う．図 2 で示したスピンダウン制御デーモンプログラムが，ディスクキャッシュドライバからシステムコールにより HDD へのアクセスの統計情報を逐次取得し，HDD の待機状態を検出，HDD のデバイスファイルに対して standby コマンドをシステムコールにより発行する．

4. 実装

4.1 ディスクキャッシュドライバの全体構成

ディスクキャッシュドライバは，ブロックデバイスの仮想化を device-mapper[9] 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．

ディスクキャッシュドライバでは，SSD キャッシュ適用

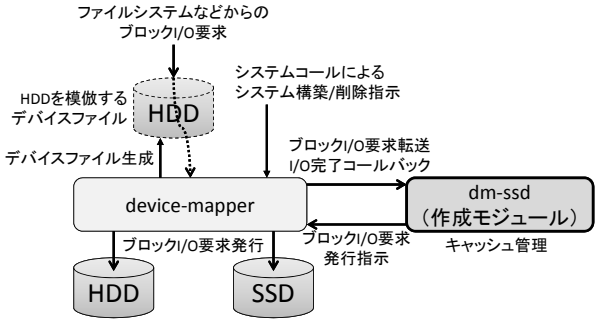

図 5 ディスクキャッシュドライバの全体構成

ブロックデバイスの作成／削除，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，キャッシュ管理や，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．ディスクキャッシュドライバの実装の全体構成の概略を次の図 5 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，キャッシュ管理情報を持ち，device-mapper から受け取った I/O 要求の内容とキャッシュ管理情報から，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．

4.2 device-mapper の概要

device-mapper の基本的な機能は，仮想的なブロックデバイスを作成し，その任意のセクタ範囲を，カーネル内の「ターゲット」と呼ばれるモジュールにマッピングすることである．マッピングされた範囲へのブロック I/O 要求は，マッピング先のターゲットに転送され，ターゲットが I/O 要求に様々な変換を施したり I/O の振り分けを行ったりすることが可能である．また，device-mapper を介することで，仮想ブロックデバイスにターゲット独自の ioctl コマンドを実装できる．本研究のディスクキャッシュドライバは，device-mapper に dm-ssd モジュールをターゲットとして登録し，仮想ブロックデバイスを作成してその全体を dm-ssd にマッピングすることにより実現する．

device-mapper の制御用インタフェースは`/dev/mapper/`配下の `control` キャラクタデバイスファイルによるシステムコールインタフェースでアプリケーションプログラムに提供される．このインタフェースを用いて，device-mapper に仮想ブロックデバイス作成，ターゲットへのマッピングの設定，仮想ブロックデバイスの削除などを指示するアプリケーションプログラム dmsetup が提供されている．

```
standby = 0;   //standby状態のフラグ
cur_access;    //最近1秒間のHDDへのアクセス数
idle_time = 0; //最近のHDDの連続無アクセス時間

while (1) {
    cur_access = get_cur_access();
    if (!standby) {
        if (cur_access > 0)
            idle_time = 0;
        else
            idle_time++;
        if (idle_time >= 10) {
            do_standby();
            standby = 1;
            idle_time = 0;
        }
    }
    else if (cur_access > 0) {
        standby = 0;
        idle_time = 0;
    }
    sleep(1);
}
```

図 6 スピンダウンアルゴリズムの擬似コード

4.3 HDD 自動スピンダウン機構の実装

HDD 自動スピンダウン機構は，device-mapper を介した ioctl コマンドインタフェースを用いて，ディスクキャッシュドライバから HDD へのアクセス数の情報を取得し，HDD のスピンダウンを行うユーザアプリケーションにより実装する．このアプリケーションの動作アルゴリズムの擬似コードを**図 6** に示す．

本アプリケーションは，1 秒ごとにディスクキャッシュドライバから HDD への合計アクセス数を取得し，1 秒前からアクセス数が増加しているか否かで，HDD へのアクセスがその間にあったかどうかを判定する．10 秒以上アクセスが無かった場合，HDD へ standby コマンドを発行し，HDD をスピンダウンさせる．

今回，スピンダウンを行う待機時間を 10 秒間に設定した理由は，評価に用いる HDD のスピンアップに要する時間が約 6〜8 秒であり，その間は HDD へのアクセスが不可でアクセス数が増加しないため，それ以上の待機時間を設定しなければスピンアップが完了しない間にスピンダウンを実行する可能性があり，それを防ぐためである．

5. 評価

本研究のディスクキャッシュドライバを実装した実機上で，SSD を HDD のディスクキャッシュとして用いたシステムを構築し，実際にディスク I/O を実行するベンチマークを行うことにより，本提案システムの I/O 性能と電力特性について評価を行う．また，ファイルシステムへの変更なしに利用可能であることを検証する．

表 1 評価環境の諸元

項目	内容
CPU	Xeon X5550 2.66GHz
主記憶	4GB
SATA インタフェース	3Gbps
オペレーティングシステム	Linux 2.6.35 (x86_64)
システム用 HDD	USB 接続ポータブル HDD
評価用 HDD	SATA 3.5 インチ 7200rpm 1TB
評価用 SSD	SATA 2.5 インチ MLC 100GB

5.1 評価環境

本評価に使用する計算機とストレージデバイスの諸元を**表 1** に示す．

SSD キャッシュの設定と評価結果のラベルの対応を**表 2** に定める．また，各設定の意味を次に説明する．

設定 1

R/W キャッシュ方式で最良の性能を発揮できると予測される設定である．100GB の SSD の全体を R/W キャッシュとして使うので，空きセットの枯渇が起こりにくい．

設定 2

キャッシュ容量を 2GB と小さくすることで，R/W キャッシュ方式で空きセットが無くなる場合の評価を行うための設定である．

設定 3

RO キャッシュ方式の評価を行うための設定である．本評価では，SSD 全体をキャッシュとして利用する場合のみを評価する．

本研究で提案した SSD ディスクキャッシュシステムとの比較対象のシステムとして, 本研究の提案方式と同様, Linux デバイスドライバでブロック I/O 要求を管理して SSD をディスクキャッシュとして利用する方式である Flashcache を用いる．本評価実験での Flashcache のキャッシュ設定の概要を，**表 3** に示す．キャッシュ方式の違い以外の条件の公平性を確保するため，Flashcache のキャッシュ設定では，本研究で提案した SSD ディスクキャッシュシステムと同一のキャッシュ容量と，ブロックの大きさを設定した．

表 2 SSD キャッシュの設定

設定項目	設定 1	設定 2	設定 3
キャッシュ容量	約 100GB	約 2GB	約 100GB
ブロックサイズ	4KB		
セットサイズ	256 ブロック		
割り当て方式	R/W キャッシュ		RO キャッシュ
空きセット数の閾値	1000		-

表 3 Flashcache の設定

設定項目	内容
キャッシュ容量	約 100GB
ブロックサイズ	4KB
セットサイズ	512
割り当て方式	R/W キャッシュ
キャッシュ置換ポリシー	LRU

Flashcache のキャッシュ管理方式を簡単に説明する．Flashcache は，次のハッシュ関数により HDD のブロックを割り当てるキャッシュセット番号（CSN）を得る．

$$CSN = (LBN/CS_{size}) mod (CS_{total})$$

なお，LBN は HDD の論理ブロック番号，CS_{size} はキャッシュセットの大きさ，CS_{total} は全キャッシュセット数とする．キャッシュセットは，複数のブロックで構成される．割り当て先キャッシュセットが衝突しても，セット内のブロック数分のブロックはキャッシュセット内に格納できるが，キャッシュセット内のブロックの検索は線形探索となる．キャッシュセット内のブロックの置換ポリシーは，FIFO と LRU が選択できる．本評価では本研究の提案方式と似た方式である LRU を設定する．

5.2 評価方法

実際にファイルシステムを構築した SSD キャッシュ適用ブロックデバイスをマウントし，そのファイルシステム上で，複数のスレッドから同時並行的に各種ファイル操作を実行する場合の処理性能と消費電力を測定する．このベンチマークプログラムには，ファイルシステムやストレージの性能評価などに用いられるベンチマークツールである Filebench[10] を使用する．本評価で使用した Filebench のワークロードは次の 4 種類である．

fileserver

複数のユーザのホームディレクトリを格納するファイルサーバを想定したワークロードを実行する．各ユーザからのファイル操作を模倣するため，各スレッドは，そのスレッド ID に基づいたファイル集合を扱う．各スレッドは，ファイルの作成，削除，追記，読み込み，書き込み，ファイル情報取得を順番に実行する．

varmail

電子メールサーバのワークロードを模倣したものである．同様にメールサーバのワークロードを模倣したベンチマークプログラムに PostMark[11] があるが，これはシングルスレッドで動作する．一方，varmail はマルチスレッド化されたワークロードである．varmail では，各スレッドがメールの読み込み，メールの作成，メールの削除を行う．

webserver

ウェブサーバを想定したワークロードである．このワークロードでは，多数のスレッドがそれぞれファイルをシーケンシャルに読み込む．すべてのスレッドは共通の単一のログファイルに 16KB のログを追記するため，このログファイルに対してデータとメタデータの更新が集中する．

oltp

オンライン・トランザクション処理（OLTP: OnLine Transaction Processing）を行うデータベースサーバを模倣するワークロードである．一つ当たり数百 MB のファイルの集合に対して，200 の読み込み用スレッドがランダムな場所に同期読み出しを行い，10 の書き出し用スレッドがランダムな場所に書き出しを非同期に行う．また，単一のスレッドがログファイルへ比較的まとまった（256KB）単位で同期書き出しを行う．

これらワークロードのパラメータ設定を**表 4** に示す．この設定は，サーバ用計算機におけるファイルシステムの性能とエネルギー効率について評価した文献 [12] を参考にしたものである．

本評価では，次の 2 種類の評価を行った．次の 5.3 節でこの評価結果と考察を示す．

Flashcache との比較

デフォルト設定の ext3 ファイルシステムを構築した評価用 HDD に，評価用 SSD によるディスクキャッシュを適用したストレージ（設定 1，設定 2，設定 3，Flashcache）を作成し，その上で Filebench によるベンチマークを実行し，処理速度，消費電力特性について比較する．

HDD スピンダウンの効果の検証

HDD 自動スピンダウン機構を使用することで，実際にどれだけの電力削減が実現できるかや，どの程度処理性能に影響が出るかを，スピンダウンを行う場合とそうでない場合の評価結果を比較することで検証する．

5.3 評価結果と考察

5.3.1 Flashcache との比較

各ワークロードのブロック I/O の R/W の割合の特徴を示すため，設定 1 の SSD キャッシュ適用ブロックデバイスで各ワークロード実行時に SSD キャッシュ適用ブロックデバイスへ発行された総ブロック I/O 要求量を**図 7** に，read 要求量と write 要求量に分けて示す．

総 I/O 要求量が多い順にワークロードを並べると，fileserver，varmail，oltp，webserver となる．webserver，varmail はほとんどの read データがメモリキャッシュに載っていると考えられ，これにより read 要求の割合が少ないと考えられる．read 要求量よりも write 要求量が少ない結果となったのは oltp のみで，他のワークロードは write 要求量のほうが多い．

処理速度の測定結果と考察

Filebench の各ワークロードを 10 分間実行する試行を 5 回連続して行い，1 秒あたりに処理したファイル操作の数（ops/s），すなわち処理速度の平均をそれぞれ求めた．fileserver，varmail，webserver，oltp の各ブロックデバイスにおける処理速度を**図 8** 示す．

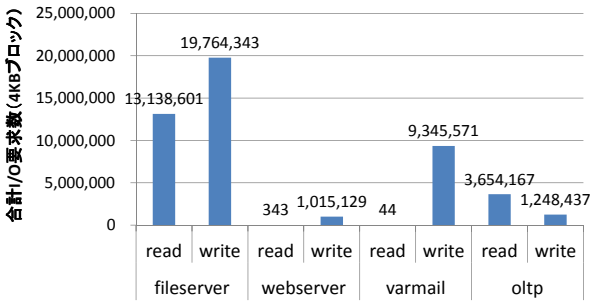

図 7 設定 1 において各ワークロードを実行した時の合計ブロック I/O 要求量

fileserver では，すべてのキャッシュ設定において，HDD 単体よりも良い性能を実現した．特に設定 1 では，Flashcache と比べて約 41%性能が向上した．

varmail では，すべてのキャッシュ設定において，HDD よりも良い性能を実現した．また，設定 1 は Flashcache と比べて約 48%性能が向上した．設定 2 は，Flashcache より処理速度が低い fileserver の結果と異なり，varmail では，Flashcache より約 18%良い性能となった．varmail は，総 I/O 要求量が少なく，ほとんどが writeI/O 要求であるため，キャッシュ容量の少ない設定 2 でも SSD の書き出し性能を有効に活用できたと考えられる．

webserver では，どのブロックデバイスでも同程度の処理速度となった．ベンチマークの処理の内，ブロックデバイスへのアクセスの割合が少なく，ほとんどのファイル操作がメモリキャッシュ上で行われていると考えられる．

oltp では設定 1 がほぼ Flashcache と同程度の処理速度を実現した．しかし，他の設定では，HDD よりも処理速度が低下した．oltp は read 要求の割合が多く，小さいサイズのランダム読み込みが多いため，read 要求のキャッシュミスによるキャッシュ割り当てのオーバヘッドが大きく，性能が向上しにくいと考えられる．

エネルギー効率の考察

各ワークロードを 10 分実行したときの実行中の HDD，SSD の平均消費電力（W = J/s）と，前述の平均処理速度（ops/s）から，fileserver，webserver，varmail，oltp の各ブロックデバイスにおける消費したエネルギー 1J 当たりに処理できたファイル操作の数（ops/J）を算出した．その結果を**図 9** に示す．

各結果より，設定 1 はすべての場合で Flashcache よりも高いエネルギー効率を示した．fileserver，varmail の場合，Flashcache に比べてそれぞれ約 53%と約 61%エネルギー効率が向上した．また，書き出し要求がほとんどの割合を占める webserver や，varmail では，設定 2 も Flashcache より高いエネルギー効率を示した．ただし，webserver においては，HDD そのままのほうがエネルギー効率が高い．これは，HDD 単体とワークロードの処理速度に違いがなく，HDD に SSD を加えたシステムの消費電力のほうが

表 4　各ワークロードのパラメータ設定

ワークロード	平均ファイルサイズ	平均ディレクトリ深度	ファイル数	I/Oサイズ		スレッド数	R/W比率（回数）
				Read	Write		
webserver	32KB	3.3	20,000	1MB	16KB	100	10:1
fileserver	256KB	3.6	50,000	1MB	16KB	100	1:2
varmail	16KB	0.8	50,000	1MB	16KB	100	1:1
oltp	0.8GB	0.3	10	2KB	2KB	200 + 10	20:1

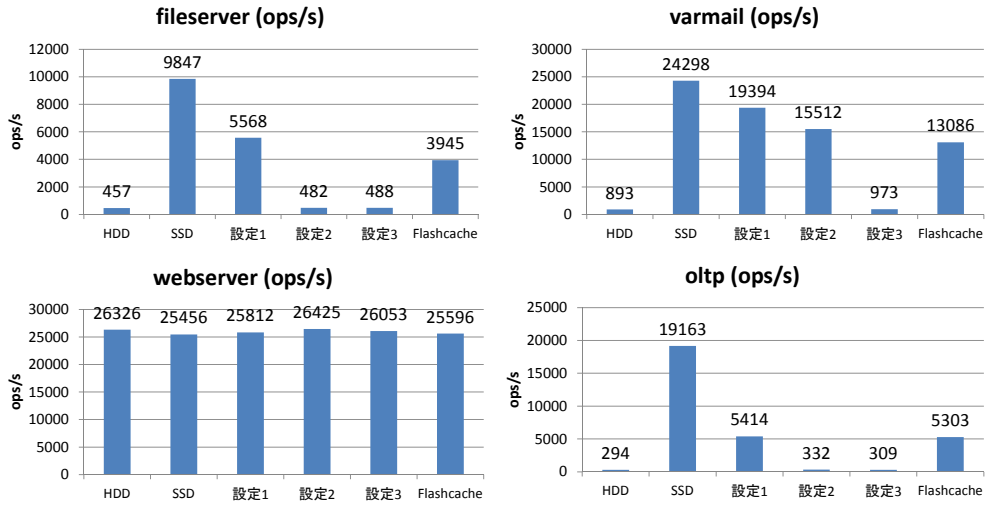

図 8　HDD, SSD, 提案方式，Flashcache での各ベンチマーク実行時の平均処理速度

HDD 単体の消費電力よりも単純に大きいためである．

5.3.2　HDD スピンダウンの効果の検証

次に，各種ファイルシステム上でのディスクキャッシュシステムの動作確認と，自動スピンダウン機構を利用した場合の省電力効果や処理性能への影響について検証するため，ext3，XFS，NILFS の 3 種類のファイルシステム上で各ベンチマークを実行し，平均処理速度と平均消費電力をそれぞれ測定した．その結果，3 種類すべてのファイルシステムにおいて，ファイルシステムの整合性を保ちつつ正常に動作することを確認した．

次に，平均処理速度の測定結果を**図 10** に，平均消費電力の測定結果を**図 11** に，それぞれ示す．各棒グラフの凡例の意味は次の通りである．

HDD　HDD そのままの結果

HDD+SSD

設定 1 の SSD キャッシュ設定を適用し，HDD のスピンダウンは実行していない結果

+spindown

設定 1 の SSD キャッシュ設定を適用し，さらに HDD 自動スピンダウンアプリケーションを実行した場合の結果

まず HDD+SSD については，NILFS 上で varmail を実行した場合を除く，すべての結果における処理速度が HDD で実行した場合とほぼ同程度か，増加した．varmail は図 7 に示した通り，ほとんどが書き込み I/O である．ログ構造化ファイルシステムである NILFS ではデータの更新が，更新された順にシーケンシャルにデバイスに書き出されていく．そのため，データの更新頻度が高い varmail においては，シーケンシャルな書き込み I/O の割合が他と比べて大きく，シーケンシャルアクセスが高速に行える HDD での処理性能が他のファイルシステムで行った場合よりも高速になったと考えられる．SSD ディスクキャッシュを利用した場合に性能が低下した原因については，varmail の場合，HDD への I/O はしておらず，新規書き込みブロックの SSD キャッシュへの割り当てがほとんどを占めており，この処理オーバヘッドが大きいのではないかと推測される．

webserver で，HDD と HDD+SSD との性能に差があまり生じない理由は，Flashcache との比較評価でも述べたように，ほとんどのファイル操作がメモリキャッシュ上で完結していたからであると推測される．

次に，スピンダウンを行うことによる，処理性能への影響について考察する．HDD+SSD と+spindown で大きく結果が異なったケースは ext3 と webserver，oltp の組み合わせである．webserver では+spindown の方が処理速度が低下した．この原因は ext3 でのみ発生した定期的な read ミスである．ext3 上で webserver を実行すると，約 20 秒に 1 度の頻度で SSD キャッシュの read ミスが起こり，HDD のスピンアップが発生した．そのため頻繁な HDD のスピンアップ・スピンダウンが発生し，主にスピンアップによるオーバヘッド時間が増加したことによって性能が低下した．

ext3 と oltp の組み合わせでは，webserver とは逆にスピンダウンした方が処理性能が向上する結果となった．この原因は特定に至っていないが，ログファイルへの同期書き

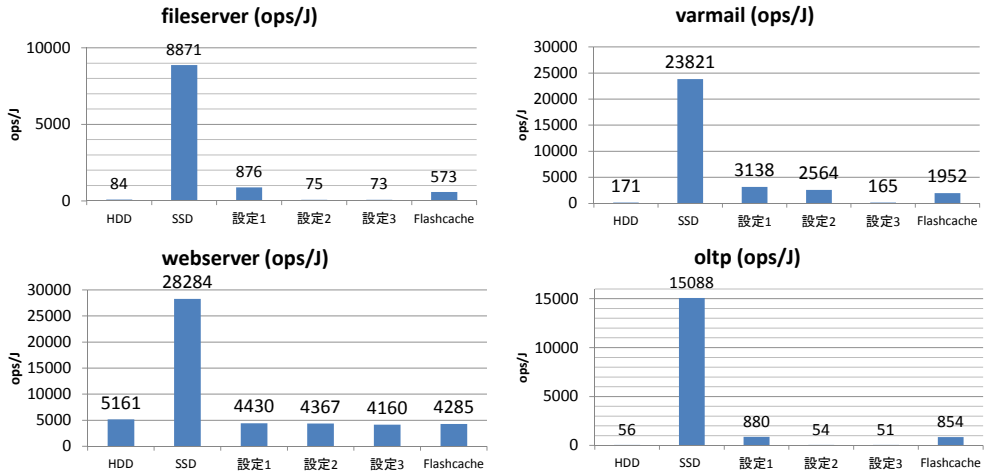

図 9 HDD, SSD, 提案方式，Flashcache での各ベンチマーク実行時のエネルギー効率

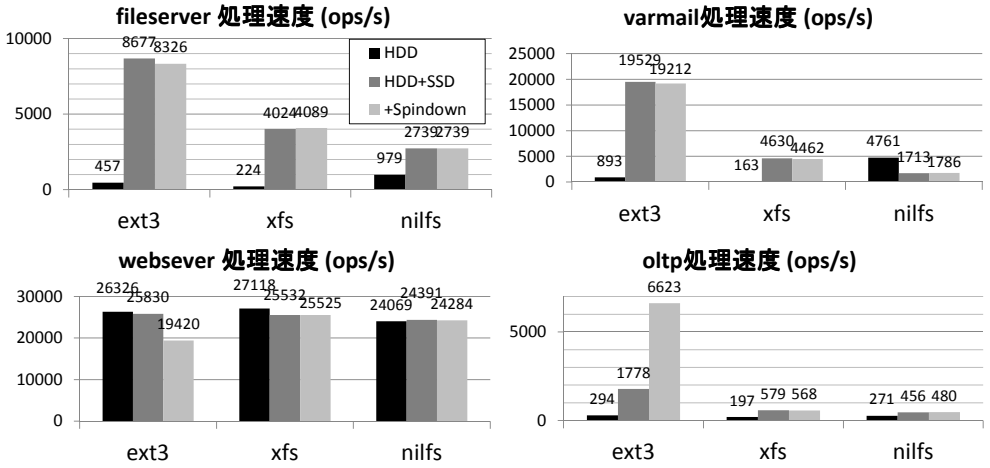

図 10 各ファイルシステムにおける各ベンチマーク実行時の平均処理速度

込みの所要時間のばらつきが大きく，それが全体の処理速度に大きく影響しているため，ファイルシステムの内部処理の影響である可能性が高い．

平均消費電力については図 11 に示す通り，ext3 と webserver，xfs と oltp の組み合わせを除くすべての場合で，スピンダウンを行うことにより，HDD よりも削減することができた．xfs と oltp の組み合わせでは，スピンダウンを行えるような HDD の待機期間がほとんどなかったと推測される．

次に，エネルギー効率について考察する．図 10，図 11 より，各ファイルシステムにおける各ベンチマーク実行時のエネルギー効率（ops/J）を算出した結果を**図 12** に示す．ext3 と webserver の組み合わせを除いたすべての場合で+spindown は HDD よりも高いエネルギー効率を示した．したがって，SSD ディスクキャッシュと HDD のスピンダウンの併用は，ストレージデバイス全体の省電力化に一定の効果があると言える．ただし，ext3 上での webserver 実行のような頻度の HDD へのアクセスにどのように対応するかが今後の課題である．

6. 結論

本研究の提案方式は，Filebench による ext3 ファイルシステム上でのベンチマークの結果，すべてのワークロードにおいて，比較対象とした従来方式である Flashcache よりも処理速度が向上し，最大で約 48%の性能向上を実現できた．また，同時に Flashcache よりも高いエネルギー効率を実現し，最大で約 61%のエネルギー効率の向上を実現した．

HDD スピンダウン機構を実装したシステムでの評価では，HDD のスピンダウンを組み合わせることでさらにエネルギー効率を改善することができたが，HDD へのアクセス頻度によっては，頻繁なスピンアップが発生し，処理性能とエネルギー効率の低下を招くことがわかった．今後の課題として，このようなエネルギー的に損になるスピンダウンを行わないようにより精度の高いスピンダウンタイミングの判定アルゴリズムの開発があげられる．

謝辞 本研究は科研費基盤研究 (B) 22300015 「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

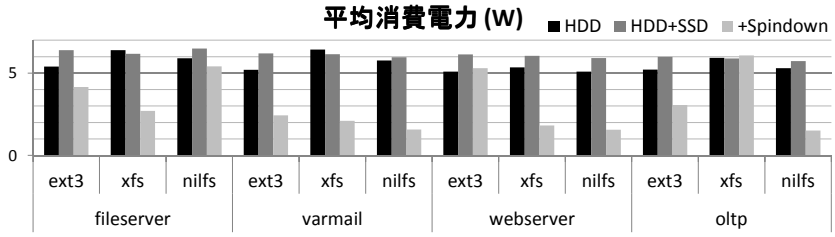

図 11 各ファイルシステムにおける各ベンチマーク実行時の平均消費電力

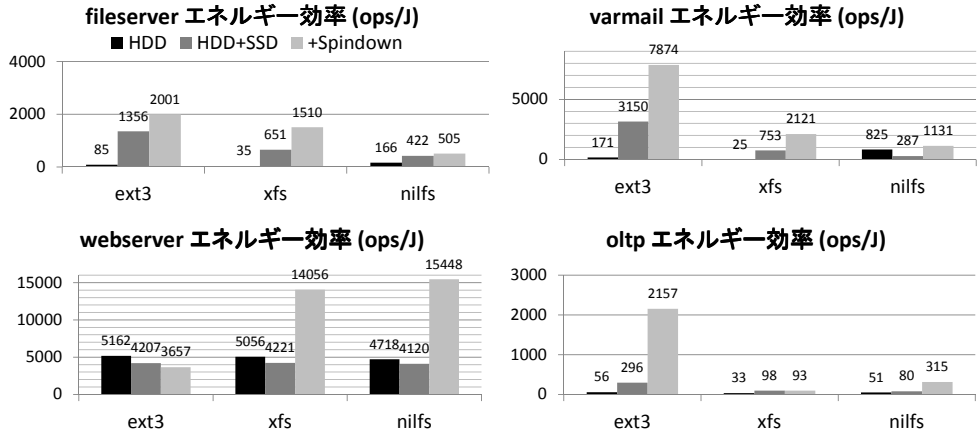

図 12 各ファイルシステムにおける各ベンチマーク実行時のエネルギー効率

ティング環境の研究」の助成を受けたものである.

参考文献

[1] David A. Patterson（原著）, John L. Hennessy（原著）, 成田 光彰 （翻訳）: コンピュータの構成と設計〜ハードウエアとソフトウエアのインタフェース 第 3 版（下）, 日経 BP 社 (2006).
[2] Matthews, J. et al.: Intel Turbo Memory: Nonvolatile disk caches in the storage hierarchy of mainstream computer systems, *ACM Transactions on Storage*, Vol. 4, No. 2, pp. 1-24 (2008).
[3] Sun Microsystems, Inc.: *Solaris ZFS Administration Guide, Part No: 819-5461* (2009).
[4] Mohan Srinivasan: Facebook / Flashcache, https://github.com/facebook/flashcache.
[5] Thanos Makatos, Yannis Klonatos, Manolis Marazakis, Michail D. Flouris and Angelos Bilas: Using Transparent Compression to Improve SSD-based I/O Caches, *Proceedings of the 5th European conference on Computer systems* (2010).
[6] Pritchett, Timothy and Thottethodi, Mithuna: SieveStore: a highly-selective, ensemble-level disk cache for cost-performance, *Proceedings of the 37th annual international symposium on Computer architecture* (ISCA '10), pp. 163-174 (2010).
[7] 仁科圭介, 佐藤未来子, 並木美太郎: 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, プログラミング・シンポジウム予稿集, Vol. 53, pp. 29-36 (2012).
[8] Lei Ye, Gen Lu, Sushanth Kumar, Chris Gniady and John H. Hartman: Energy-efficient storage in virtual machine environments, *Proceedings of the 6th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments* (VEE '10), pp. 75-84, (2010)
[9] Device-mapper Resource Page, http://sourceware.org/dm.
[10] Filebench Web site, http://filebench.sourceforge.net.
[11] Katcher, J.: PostMark: A new filesystem benchmark, Technical report, Network Appliance (1997).
[12] Sehgal, P., Tarasov, V. and Zadok, E.: Optimizing energy and performance for server-class file system workloads, *ACM Transactions on Storage*, Vol. 6, pp. 10:1-10:31 (2010).