Vol.2011-OS-116 No.6
2011/1/25

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

仁科 圭介[†1]　佐藤 未来子[†1]　並木 美太郎[†2]

SSD（Solid-State Drive）は高いアクセス性能と低消費電力を同時に実現できるストレージデバイスであり，この特性を効果的にシステムに活かす利用法が課題となっている．本研究では，SSD を利用してシステムを省電力・高速化することを目標に，SSD をディスクキャッシュとして管理する Linux ブロックデバイスドライバを設計した．特に SSD の省電力性に着目し，書き込みデータをライトバックする方式を採ることで，HDD の無アクセス期間の延長を図り，HDD の省電力機構と組み合わせた省電力化を目指した．本方式を実装したデバイスドライバを作成し，Linux 機上で動作させてシステムの基礎評価を行った結果，I/O 時の消費エネルギーを削減する効果が確認できた．

Implementation of a Linux Block Device Driver Using SSD for Disk Cache for Power Saving and High Performance

Keisuke Nishina,[†1] Mikiko Sato[†1]
and Mitaro Namiki[†2]

SSDs (Solid State Drives) are storage device that can improve storage performance and energy consumption, and finding efficient uses of SSD has become a challenge. This paper describes a design of a Linux block device driver using SSD as a disk cache to achieve high speed and the power saving of the disk access, and evaluate a system applied with the disk cache. As a result of the evaluation, we reduced I/O power consamption.

1. はじめに

近年，NAND Flash などの半導体記憶素子を用いた 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．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ち，次世代のストレージデバイスとして注目されている．しかしながら，SSD の容量単価は 2010 年 12 月現在でも HDD の数十倍〜数百倍ほどもあり，大容量のストレージを必要とするシステムの場合，そのままストレージとして HDD を SSD に置き換えるにはコスト効率が悪い．したがって現状では，よりコスト効率の良い SSD の利用法の確立が課題となっている．

先行研究では，いくつかの SSD の利用法が検討されてきた．SSD を利用して，システムの高速・省電力化を図る既存の利用可能なシステムとして，Intel Turbo Memory[1] や，Solaris ZFS[2] ファイルシステムの機能である，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 が必要であり，制限が大きい．

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，read-only キャッシュとなっている．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．

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[3] がある．これは，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 はブロックレベルでのキャッシュを行うため，ファイルシステム独立であるが，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（SSD キャッシュのヒット率向上による I/O 消費電力の削減）と，圧縮・解凍にかかる CPU サイ

†1 東京農工大学大学院工学府
Graduate school of Engineering, Tokyo University of Agriculture and Technology
†2 東京農工大学大学院工学研究院
Institute of Symbiotic Science and Technology, Tokyo University of Agriculture and Technology

クルのトレードオフとなっている．また，信頼性を確保するため，書き込みは全て HDD へライトスルーする方式をとっており，書き込みが比較的多いワークロードの場合，省電力効果はあまり期待できない．

本研究では，SSD を効果的に利用することで，システムの省電力・高速化を実現することを目標に，SSD を汎用のディスクキャッシュとして利用するための仮想化方式を提案する．この仮想化方式を Linux サーバー機・デスクトップ機上にそれぞれ実装し，SSD をディスクキャッシュとして動作させて I/O 性能と消費電力についての評価実験を行い，HDD の省電力機構と連携した場合にどの程度の電力削減・高速化効果が見込めるか考察した．

2. 目標と設計方針

2.1 目　　標

本研究では，ストレージの省電力化とディスクアクセスの高速化を実現することを目的に，SSD を 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．特に，SSD の I/O 当たりの消費エネルギーの小ささに着目し，この特性を活かしたストレージ全体のエネルギー消費の抑制を重点的な目標とする．また，このディスクキャッシュシステムが多様な計算機環境に柔軟に適用できる形での実装方式を目指す．

2.2 設 計 方 針

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割り当てることにより，ディスクキャッシュのヒット率の向上を目指す．また，I/O 時の消費電力だけでなく，待機時の消費電力も削減しなければ，待機電力は HDD と SSD の電力の和となり，待機時間が多ければ多いほど，消費エネルギーは増加する．待機時の電力を削減するには，HDD の省電力機構（スピンドルの回転数を下げるなどして待機時の消費電力を抑える機能）を用いるほかないが，一度省電力状態（スリープまたはスタンバイ）に切り替わると，通常状態への復帰に電力と数秒の時間がかかる．したがって，一度省電力状態に移行したらなるべく長い期間その状態を持続できるような管理が必要である．そこで，なるべく HDD への無アクセス期間が長くなるように，HDD への書き込みを SSD で遅延するライトバックポリシーを導入した．また，システム再起動後の SSD ヒット率を維持するため，SSD 内のキャッシュデータはシステムを再起動しても利用できるようにする．

具体的な実装は，Linux 上で動作するブロックデバイスドライバで実現する．ブロックレベルでのキャッシュ管理を行うことで，その上位で実装されるファイルシステムや，下位のブロック型デバイスの種類に関係なく利用できる．

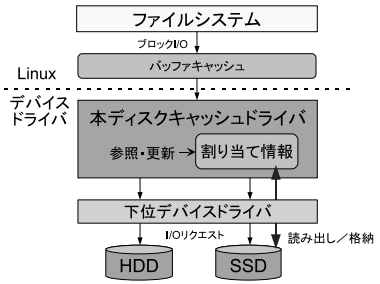

図 1　本ディスクキャッシュドライバの構成

3. 設　　計

3.1 全 体 構 成

本研究では，図 1 に示すように，ブロック I/O の制御を行う Linux のブロックデバイスドライバにおいて，SSD へのブロックの割り当てを管理する．ブロック I/O とは，各ファイルシステムやバッファキャッシュとブロック型デバイスの間で行われる，単一の連続したデータ（これを一般にブロックと呼ぶ）の I/O である． 本ドライバは，各ファイルシステムやバッファキャッシュから，HDD に対するブロック I/O 要求を受け取る．この要求内容と主記憶上で管理するブロックの割り当て情報を利用して，実デバイスへのブロック I/O 要求を作成し，下位のデバイスドライバに発行していく．こうしてブロック I/O 要求を処理しながら，SSD に HDD のブロックをキャッシュして利用する．ブロック I/O 層での実装にすることで，特定のローカルファイルシステムや特定のブロック型デバイスに依存しないディスクキャッシュシステムが実現できる．また，システムの電源を切っても再起動後に再び SSD 内のキャッシュデータを利用するために，データブロックの割り当て情報は，システム終了時に SSD 内に格納し，再び同じ構成でディスクキャッシュを適用するときに読み出して利用する．

3.2 ブロック割り当て情報の管理

本デバイスドライバは，SSD ディスクキャッシュのブロック割り当てを管理・制御するために，SSD 内のどのブロックが割り当て済みで，どのブロックが空きなのか，また，割り

当て済みなら HDD のどのブロックに対応するのか，それは今ダーティブロックなのか，といった情報を保持する．ここでは，この割り当て情報をどのように管理していくかを述べる．

3.2.1 ブロック割り当て管理構造

本デバイスドライバでは，固定長単位でブロック割り当てを行う．このブロックの大きさは，SSD ディスクキャッシュ適用時にシステム構築者が 2^N セクタで指定する．一般的な Linux のファイルシステムも同様に固定長単位でディスク上のファイル格納を管理しているものが多く，Linux の主記憶管理の単位であるページに合わせて大きさを設定することが多い．一般的なページの大きさは 4KB なのでブロック割り当ての単位も 4KB に設定するのが適切であると考えられる．これより大きいサイズで割り当てを行うと，割り当ての元となったファイルシステムからの I/O 要求のデータサイズに対して SSD に割り当てるブロックの大きさのほうが大きくなる場合があり，ブロック割り当てのために余分なデータを HDD から読み込まなければならず，性能が悪化する．

しかし，この 4KB のブロックごとに割り当て管理情報（メタデータ）を持つとすると，このメタデータのエントリ数は膨大な量となり，メタデータの総量が非常に大きくなる．メタデータが大きくなると，その分 SSD にブロックを格納する領域が減少するほか，主記憶上でメタデータを管理するために必要な領域が増加する．例えば 4KB のブロックごとに，対応する HDD と SSD のブロック番号の対（8-16B），ステート情報（1B），そして LRU/空きブロック管理用リンクドリスト（8-16B）を保持する場合，合計 17-33B となる．ブロック 1GB（4KB × 2^{18}）当たりでは，17B × 2^{18}=4.25MB から，33B × 2^{18}=8.25MB となる．

そこで，ある程度まとまったブロックの集合単位で HDD のブロックとの対応付けを行えば，集合のエントリ数は小さく抑えられると考え，ブロックに加えて,「セット」と呼ぶ管理単位を導入した．セットは隣接するブロックの集合で，大きさは SSD ディスクキャッシュ適用時にシステム構築者が 2 のべき乗のブロック数で指定する．

以下，SSD のセットを「キャッシュセット」と呼び，ブロックを「キャッシュブロック」と呼ぶことにする．領域の割り当てはセット単位で行い，実際にデータブロックの格納状態を管理するのはキャッシュブロック単位で行う．さらに，キャッシュブロック単位の状態管理情報にはビットマップを用いて無駄をなくすことにより，1 ブロックの状態管理情報を 3 ビットで表現できる．これにより，キャッシュデータの置換操作はセットごとになるが，実データの格納単位はブロック単位となる．これにより，4KB のページ単位でのキャッシュブロックの割り当てをしつつも，セットを大きく設定することによりメタデータ量を抑えることができる．

例えば図 2 に示すように，1 ブロックを 4KB，1 セットを 256 ブロックとすると，1 セットは 4KB × 256=1MB の大きさとなる．セットごとに，キャッシュセット番号と対応する HDD

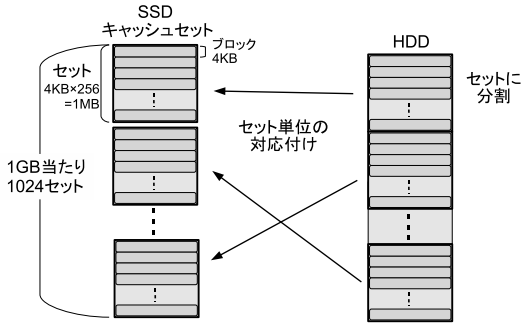

図 2 セットによる HDD と SSD の対応の例

のセット番号の対（8-16B），256 ブロック分のステート情報（3bit × 256=96B），そして LRU/空きセット管理用リンクドリスト (8-16B）を保持する場合，1 セット当たり 112-128B のメタデータとなり，キャッシュセット 1GB 当たりでは，1K 倍して，112K-128KB のメタデータで済む．これは前述の 4KB ブロックごとに割り当てを管理した例に比べても非常に小さいことがわかる．

ただし，セットを大きく設定すると，セット内のほんの一部のブロックにしか I/O 要求が来なければ，その他のブロックは領域だけ確保されても利用されずに無駄になってしまう．したがって，セットの大きさについては，ある一定の大きさが適切というわけではなく，適用する計算機システムでどれだけの主記憶消費が許容されるかや、用いる SSD の容量などによって異なるため，セットの大きさは，前述の通りディスクキャッシュ適用時に設定できるようにした．

3.2.2 主記憶上でのブロック割り当て情報の管理

全体構成でも触れたように，本デバイスドライバは動作中，主記憶上で SSD へのブロック割り当て情報を保持する．基本的には，図 3 に示すように，SSD 全体のキャッシュセットの管理情報を各セットごとにまとめ，それらを各キャッシュセットのインデックスに対応したインデックスを持つ配列で保持する．配列のメタデータ構造体の各メンバの意味は次の通り．

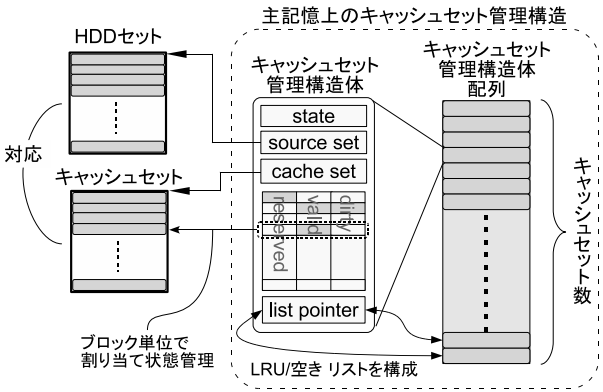

図 3 主記憶上のキャッシュセット管理構造

- **status** キャッシュセットの状態（有効・無効・ダーティ・ライトバック中）
- **source set** 対応づけられている HDD のセット番号
- **cache set** このメタデータが管理するキャッシュセットのインデックス
- **valid** 有効データが格納されているブロックを示すビットマップ
- **reserved** 実データ書き込み処理中のブロックを示すビットマップ
- **dirty** HDD に内容が同期されていないブロックを示すビットマップ
- **LRU/free list pointer** 他の構造体と接続して LRU リストや空きリストを構築するポインタ

本デバイスドライバは，この主記憶上のメタデータを参照して，要求ブロックが SSD に割り当てられているか否かを判定し，割り当てられていれば SSD のどこに割り当てられているかを特定する必要があるが，このメタデータ配列をそのまま線形探索するのは非常に効率が悪い．そこで，HDD のセット番号から対応する SSD のセット番号を検索できる対応表（図 4）を導入した．

この対応表は，HDD のセット番号をインデックスにし，割り当てられたキャッシュセット番号が格納される．この対応表も，HDD のセット数と同じ数のエントリを持つので，HDD のセット数が多くなるとメモリ空間を圧迫する．その場合は，代わりにハッシュ表を用いることで，エントリの増加を抑えることができる．

3.2.3 SSD 内のデータ配置

SSD 内のメタデータとデータブロックの配置を図 5 に示す． SSD 内には各メタデータとデータブロックを種類ごとにまとめて配置した．これにより，セクタ境界による隙間ができるだけ少なくなり，無駄な空間を極力なくすことができると考えられる．最初の部分には，キャッシュセット番号をインデックスとして各キャッシュセットに対応する HDD のセット番号が格納される．その次には各セット内ブロックの状態ビットマップが同様に格納される．このビットマップは，ブロックが有効であることを示す Valid ビットマップが格納される．ビットマップの次にキャッシュセットが格納される．最後の 1 セクタには，このキャッシュをシステムの再起動後に再び用いるときに必要なブロックサイズなどの初期設定パラメータが格納される．

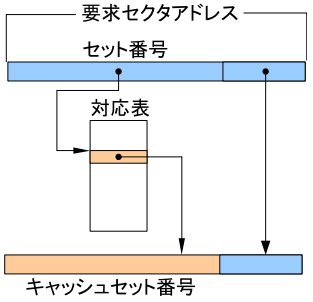

図 4 キャッシュセット検索構造

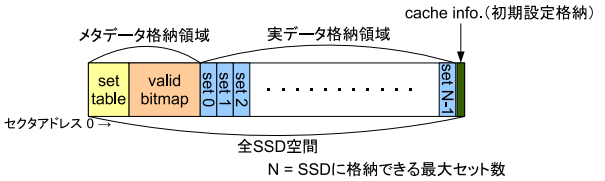

図 5 SSD 内データ配置

3.3 ブロック割り当て方式

本ドライバでは，バッファキャッシュから HDD へ出された I/O 要求に応じて，HDD のブロックを SSD に割り当てていく．読み込み要求については，要求ブロックが SSD に割り当てられている場合（ヒット）は，SSD に割り当てられたブロックを読み，要求ブロックが SSD に割り当てられていない場合（ミス）は，HDD から要求ブロックを読んできて，SSD に新しく割り当てる．書き込み要求については，HDD の無アクセス期間を延ばすため，

一旦書き込みデータを SSD に貯めてすぐには HDD へ書き込まず，空きキャッシュセットが少なくなってくるか，或いは読み込みミスにより HDD へのアクセスが起こったときに，HDD に書き出すライトバック方式とした．

しかし，SSD は書き換え回数に上限があり，全ての書き込みを一旦 SSD に書くと，SSD の故障を早める恐れがある．また，本デバイスドライバでは，SSD へのブロックの割り当て情報をシステム終了時にだけ SSD に書き込むので，システムが突然クラッシュすると，HDD へ同期されていない書き込みデータが消失する．そこで，ライトバック方式に加えて，書き込み要求は SSD に書き込まずに HDD に直接書き込む方式もディスクキャッシュ適用時に選択可能とした．

また，ライトバックポリシーを選択した場合は，空きキャッシュセット数の閾値を設定できる．本ドライバは空きキャッシュセット数がこの閾値を下回ると，割り当て済みブロックの中から LRU ポリシーによって選択したブロックを無効化し，空きブロックを増やす．このときダーティブロックがあった場合は，そのデータは HDD に同期される．HDD に直接書き込む方式を選んだ場合，当然ダーティブロックは SSD に存在しないので，空きブロックがない状態で新たに HDD のブロックの割り当てが必要になった場合は，LRU によって選択した SSD のブロックに新しいデータで上書きし，新しい割り当てを設定するだけでよい．

4. 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構成の概略を次の図 6 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，ブロックの割り当て情報を管理し，device-mapper から受け取った I/O 要求の内容とブロックの割り当て情報から，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．

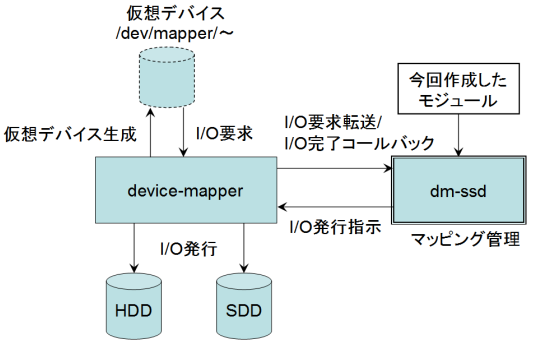

図 6　本デバイスドライバの構成

5. 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．

5.1 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 2.6.35.22 x86_64

この計算機上で，本デバイスドライバによって SSD をディスクキャッシュとして適用した HDD 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の概要と，その待機電力，スリープ電力，ウェイクアップ電力とウェイクアップにかかる時間をまとめたものを表 1 に示す．SSD には，スリープなどの省電力機構が元々ついていない．

今回の評価のために行った I/O 処理を次に示す．

表 1 評価に用いたストレージデバイス

デバイス	概要	待機電力	スリープ電力	ウェイクアップ電力×時間
2.5 インチ SSD	32GB SLC	1.0W	—	—
2.5 インチ HDD	160GB 5400rpm	1.8W	0.6W	2.4W × 1.8s
3.5 インチ HDD	250GB 7200rpm	5.0W	0.7W	11.4W × 3.2s

- **W1G：** 1GB（2^{30}B）ファイル書き込み
- **R1G：** 1GB（2^{30}B）ファイル読み込み
- **R4K random：** 4KB（2^{12}B）× 1000 ランダム読み込み

いずれの処理も，バッファキャッシュ等の影響を排除するため，sync オプションを付加してマウントした ext3 ファイルシステム上のファイル入出力で実現した．ランダム読み込みについては，10GB のファイルを 1 つ用意し，その中からランダムに 1000 箇所選んで 4KB ずつ読み込むという処理により実現した．また，各入出力処理を行う前には，バッファキャッシュ等の解放を行った．ディスクキャッシュの設定は，ブロックの大きさを 4KB，セットの大きさを 256 ブロック（4KB × 256=1MB）とし，書き込みポリシーはライトバックを指定した．

5.2 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．

5.2.1 入出力時間

3.5”HDD，2.5”HDD，SSD，3.5”HDD+SSD，2.5”HDD+SSD のそれぞれについて三種類の入出力処理を行ったときの入出力時間（ファイルを開いてから I/O 完了後に閉じるまでの時間）を次の図 7 に示す．なお，ディスクキャッシュ適用デバイスの入出力時間は，ディスクキャッシュミス時とヒット時の二つの場合についてそれぞれ測定した．

HDD や SSD 単体での I/O の比較では，書き込みについては SSD が最も遅く，読み込みについては SSD が最も速い結果となった．SSD をキャッシュとして利用した場合を見ると，ライトバックポリシーであるため，W1G では，キャッシュミス，ヒットともに全て SSD への書き込みとなり，そのまま SSD の書き込み時間となった．R1G では，キャッシュミス時は，単純に HDD と同じ入出力時間となり，ヒット時は SSD と同じ入出力時間となっている．

R4K random では，キャッシュヒット時は SSD の入出力時間とほぼ同じだが，3.5”HDD+SSD のミス時は，HDD で直接行うよりも入出力時間が増加している．これは，SSD の 4KB ランダム書き込み速度が，3.5”HDD の 4KB ランダム読み出し速度よりも低速であるためだと考えられる．2.5”HDD の場合は，ミス時でも HDD と同じ入出力時間となっている．これは 2.5”HDD からのデータの読み出しよりも SSD へのミスデータの書き込みのほうが高速なためだと考えられる．

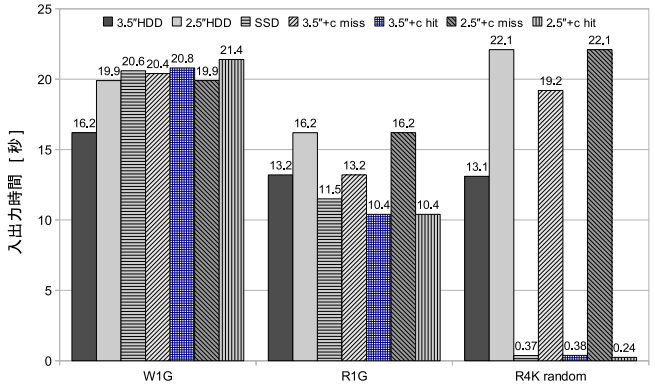

図 7 入出力時間の比較

5.2.2 消費電力

それぞれのデバイスに対して W1G，R1G，R4K random を行っているときの平均消費電力の測定結果を次の図 8，9，10 にそれぞれ示す．

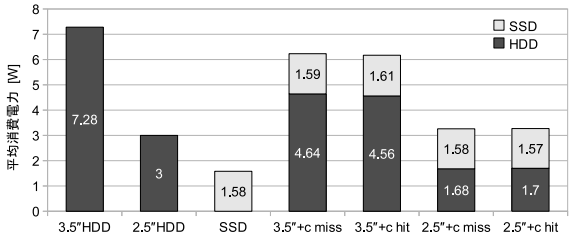

図 8 W1G 実行時の平均消費電力

各ストレージデバイスの I/O 時の消費電力は，3.5”HDD ＞ 2.5”HDD ＞ SSD となった．ディスクキャッシュを適用した結果では，W1G は，全て SSD で書き込みをしており，HDD

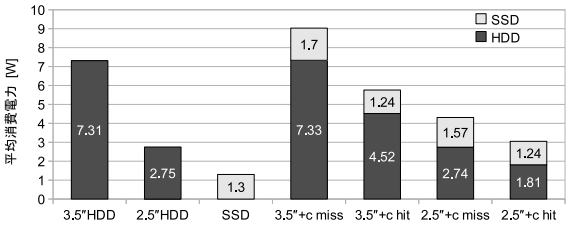

図 9 R1G 実行時の平均消費電力

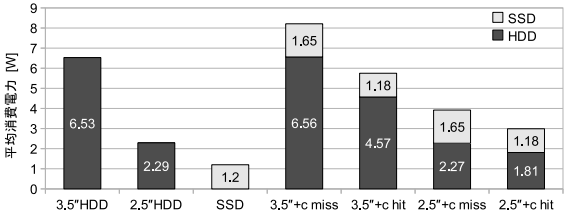

図 10 R4K random 実行時の平均消費電力

には一切アクセスしていないので，HDD は待機電力となっている．他の 2 つの読み込み I/O は，ミス時は HDD からの読み出しと，SSD への書き込みが起こるので，消費電力は高くなる．

5.2.3 消費エネルギー

ここまで測定結果から，各デバイスでの入出力処理に費やされた総エネルギーを平均消費電力×入出力時間で求めた結果を次の図 11，12，13 にそれぞれ示す．

キャッシュ適用デバイスの W1G 実行時の消費エネルギーは，キャッシュ適用前に比べて大きくなっているが，HDD へは一切書き込みをしていないため，HDD の待機エネルギーが無駄に消費されている．書き込み処理では，当然 SSD キャッシュミス時の消費エネルギーはキャッシュ適用前の HDD と比べて大きくなっているが，ヒット時の消費エネルギーは逆に減少する結果となった．

これらの結果より，本ディスクキャッシュシステムの消費エネルギー削減の可能性について考察する．まず，書き込みについては，HDD をスリープせずに SSD に書き込むと，HDD の待機消費エネルギーによってわずかにディスクキャッシュ適用前の HDD より消費エネルギーが増加する．しかしこのとき，HDD をスリープしていれば，消費エネルギーはディスクキャッシュ適用前の HDD よりも小さくなる．読み込みについては，ミス時の消費エネルギーは損失になるが，その後一回でもそのブロックに読み込みヒットが起これば，その損失分を上回る消費エネルギーの削減ができることが図 12，13 から読み取れる．

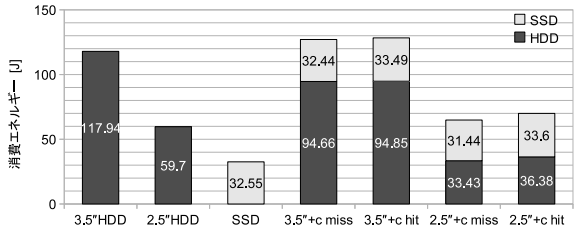

図 11 W1G 実行時の消費エネルギー

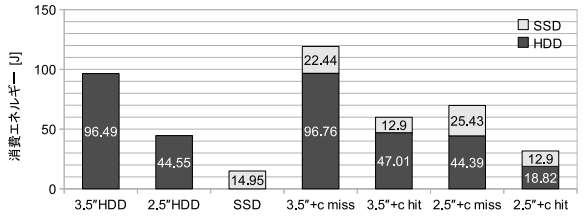

図 12 R1G 実行時の消費エネルギー

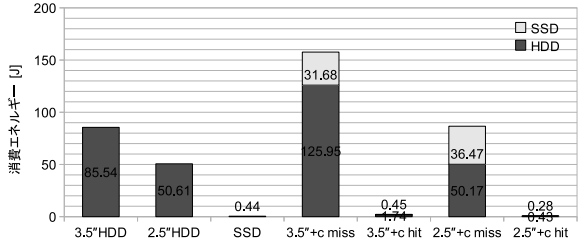

図 13 R4K random 実行時の消費エネルギー

Vol.2011-OS-116 No.6
2011/1/25

6. 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基礎的な評価実験を行った．その結果，読み込み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，読み込みミス時の消費エネルギー損失も上回る消費エネルギーの削減効果があることがわかった．また，キャッシュミス時も，シーケンシャル読み込み時の入出力時間は，HDD のみのシステムと変わらず，SSD へのブロック割り当てによる時間的オーバーヘッドはほとんどなかった．書き込み時の消費エネルギーについては，HDD のスピンダウンなどの省電力機構との連携による消費電力削減によって省エネルギー化できることがわかった．HDD の省電力機構との連携は今後の課題である．

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) 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).
4) : Device-mapper.