# 省電力・高速性を考慮した SSDによるファイルシステムデバイスの試作

仁科 圭介† 並木 美太郎‡
東京農工大学工学部 †/東京農工大学大学院共生科学技術研究院 ‡

## 1 はじめに

近年，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．なかでも，SSD（Solid State Drive）は，従来のHDDなどと同じフォームファクタとインタフェースを持つフラッシュメモリであり，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．

しかし一方で，SSDはHDDと比べると未だ容量単価が高く，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．そのため現状では，容量の比較的小さなSSDと，コスト面で有利なHDDを組み合わせて利用する構成をとることが一般的であり，ユーザ自身がファイルの格納デバイスを管理していることが多く，SSDの効率的な利用を難しいものとしている．

本研究では，このようなSSDを部分的に利用するシステムにおいて，SSDと他のストレージデバイスを一元的に管理する論理ファイルシステムデバイスを構築し，SSDの利点である省電力・高速性をより活かせるようにストレージデバイス間のデータの分配を自動的に行うシステムを設計，試作を行った．

## 2 SSDの概要

SSDはNAND Flashチップ，キャッシュメモリ（主にDRAM），コントローラなどの半導体回路で構成されている．HDDのような機械的な駆動部は存在しないので，動作が静粛であるのはもちろん，発熱も少なく，衝撃にも強い．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，ランダムアクセス（特に読み出し）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出しよりも書き込みの方が処理に時間がかかる[1]．

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，遥かにHDDよりも省エネルギーであるといえる．

## 3 目標

本研究では，SSDと他のストレージデバイスが混在するシステムにおいて，容量にとらわれない柔軟なSSDの利用を実現すると同時に，SSDの利点である高速性・省電力性を最大限活かすディスクI/Oの分配を実現することを目標とする．このためにLinux上でディスクI/OをSSDと他のデバイスに分配・管理する仮想的なファイルシステムデバイスを設計．既存のOS機能を用いた実装を行う．

## 4 設計

### 4.1 全体構成

本研究では，HDDのみで運用してきたLinux PCまたはサーバに対して，ディスクI/Oの高速化・省電力化を目的としてSSDを追加するというケースを仮定し，HDDとSSDにI/Oを分配するための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出すことでマッピングの整合性を保つ．

全体の構成を次の図1に示す．

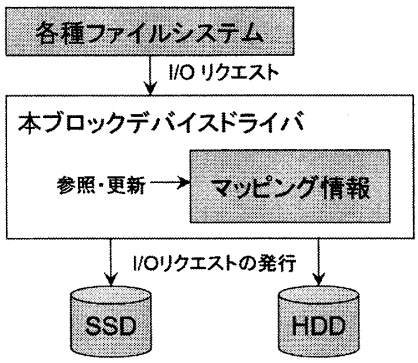


図1: システムの全体構成

### 4.2 ブロックマッピングの方針

本研究では，SSDの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，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を採る．

消費電力を抑えるためには，消費電力が小さいSSDへのアクセスを促進し，消費電力が大きいHDDへのアクセスをなるべく抑える必要がある．このためには，SSDに利用頻度の高いデータブロックを置くことが必要になる．

そこで本デバイスドライバでは，ファイルシステムから読み書きを要求されたHDDのデータブロックをSSDにマッピングしていき，以降同じデータブロックへのアクセスはSSDに対して行われるようにする．SSD内がマッピングしたブロックで埋まってしまい，SSD内にないデータブロックへの読み書き要求が発生した場合は，LRUポリシーにより最近アクセスされていないSSD内のブロックを新しいブロックに置き換える．

---

Prototype of a File System Device Driver for SSD Storage Device

† Keisuke NISHINA
Department of Computer and Information Sciences, Tokyo University of Agriculture and Technology
‡ Mitaro NAMIKI
Graduate school of Engineering, Tokyo University of Agriculture and Technology

こうして，SSD にはより利用頻度の高いデータブロックが配置されていく．言い換えれば，SSD は HDD への I/O のキャッシュデバイスとなる．

### 4.3 ブロックマッピング管理

本デバイスドライバでは，SSD へのデータのマッピングを一定サイズのブロック単位で管理する．このブロックの大きさは，仮想デバイスの生成時に設定可能とする．Linux カーネルは，メモリを 4KiB 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．また多くのファイルシステムは，ファイルを 4KiB 単位で管理している．したがってこの場合，4KiB が SSD のブロックの管理単位として適切そうである．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，もっと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，ブロックを大きく設定しておくことで，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）が必要となる．本デバイスドライバは I/O 要求発生時にメモリ上で管理されるメタデータを要求された HDD のブロック番号により検索し，ヒットすれば，対応する SSD のブロックへ I/O を発行する．ヒットしない場合は，I/O を要求された部分を含んだブロックを HDD から読み出し，SSD へ新しくマッピングする．SSD にすでにマッピングされたブロックとの置き換えが必要なときは，LRU ポリシーによる置き換えを行う．

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．次の図 2 にマッピングの管理構造を示す．

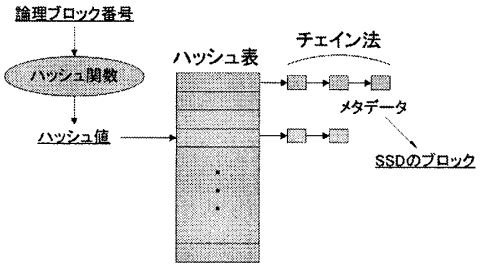


図 2: ハッシュ表によるマッピング管理

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．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，ハッシュ表にメタデータを追加する．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とで，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，以降の読み出し時に 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．

全てのブロックがすでに割り当てられているときは，LRU リストから置き換えブロックを選択（後述），ハッシュ表から置き換え前のメタデータを削除し，新たなメタデータを追加する．これらのメタデータの更新処理の後，HDD から必要なブロックを読み出して要求された処理を行い，新しくマッピングされた SSD のブロック番地へ書き出す．

メタデータはハッシュ表とは別に LRU リストでも管理され，SSD の全てのブロックが割り当てられているときに，SSD にないブロックへの I/O 要求があった場合の置き換えブロックの選択に利用する．LRU リストには，双方向リスト構造を用いることで，リストの並び替え，置き換えブロックの選択の処理を O(1) で行える．次の図 3 に LRU リストの管理構造を示す．

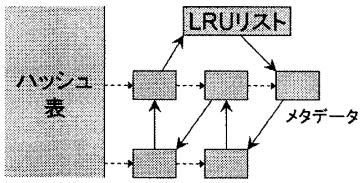


図 3: LRU リストの管理構造

この LRU リストは，SSD の全てのマッピングされたブロックを 1 つのリストで管理する．そしてブロックへのアクセス毎にそのブロックをリストの最後尾に並び替える．

リストから置き換えブロックを選択するときは，先頭要素から見ていき，ダーティブロック（HDD にデータが同期されていないブロック）でない最初のブロックを置き換えブロックとする．LRU リストの先頭に来たダーティブロックは，このときライトバック処理される．こうして HDD へのアクセスを時間的に集中させることで，HDD の無アクセス期間を延ばし，HDD のスピンダウンなどの省電力機能を効率的に利用できると考えられる．

## 5 実装

本デバイスドライバの機能のうち，ブロックデバイスとしての基本機能は，Linux の仮想ブロックデバイス作成モジュールである Device-mapper[2] によって実現できる．この Device-mapper 上でマッピングを管理し，実デバイスへの I/O 発行を指示するモジュールを現在新たに作成中である．

## 6 おわりに

本論文では，SSD の省電力・高速性を有効活用する論理ファイルシステムデバイスの設計と，Linux 上での実装について述べた．今後，本システムの実装と評価を行って，その有効性を検証することが課題となる．

### 参考文献

[1] Jim Cooke: Flash memory technology direction, Windows Hardware Engineering Conference (2007).
[2] Device-mapper. http://sourceware.org/dm.
[3] Eric Van Hensbergen, Ming Zhao: Dynamic policy disk caching for storage networking, IBM Research Report RC24123.
[4] Matthews J., Trika S., Hensgen D., Coulson R., Grimsrud K.: Intel TurboMemory: Nonvolatile disk caches in the storage hierarchy of mainstream computer systems, ACMTrans, Storage4, 2, Article4(May2008), 24pages.