論文 / 著書情報
Article / Book Information

論題(和文)	アクセス時間に基づいた次アクセス予想によるデバイスミックスストレージシステムの制御手法
Title(English)	
著者(和文)	新屋敷裕太, 飯澤健, 小沢 年弘, 荒堀 喜貴, 横田 治夫
Authors(English)	Yuta Shinyashiki, Ken Iizawa, Toshihiro Ozawa, Yoshitaka Arahori, Haruo Yokota
出典(和文)	第8回データ工学と情報マネジメントに関するフォーラム論文集, , , D8-2
Citation(English)	, , , D8-2
発行日 / Pub. date	2016, 3

アクセス時間に基づいた次アクセス予想による デバイスミックスストレージシステムの制御手法

新屋敷裕太† 飯澤 健†† 小沢 年弘†† 荒堀 喜貴† 横田 治夫†

† 東京工業大学 〒 152–8550 東京都目黒区大岡山 2 丁目 12–1
†† 株式会社 富士通研究所 〒 211–8588 神奈川県川崎市中原区上小田中 4–1–1
E-mail: †shinyashiki@de.cs.titech.ac.jp, ††{iizawa.ken,t.ozawa}@jp.fujitsu.com,
†††{arahori,yokota}@cs.titech.ac.jp

あらまし 低コストで高性能なストレージシステム実現に向け，HDD と SSD のデバイスミックスにおける効率的制御が注目されている．デバイスミックス環境は様々な用途で使われることが想定される．公開されているアクセストレースを用いたアクセス間隔分析により，ワークロード毎やストレージ領域毎でアクセスパターンが異なることを検知した．デバイス間でデータの再配置を行う際のタイミングは性能に影響を及ぼす．そのため，アクセス時間間隔を考慮し，ワークロード毎や領域毎に適した制御が求められる．本研究では，領域ごとのアクセスの時間間隔を基に次のアクセスの予測を行うデバイスミックスの制御手法を提案する．アクセス予測に応じて，アクセス対象データのデバイス選択やデータ移動のタイミング制御を行い，その有用性を示す．

キーワード デバイスミックス，ブロックトレース，データ再配置，アクセス予測

1. はじめに

現在，世の中のデータ量は爆発的な増加をしており，データの格納媒体として HDD（ハードディスクドライブ）が広く用いれられてきた．HDD は大容量化により，そのデータ量の増加に対応してきた．しかしながら，HDD 自体の性能は頭打ちとなっている．一方，メモリに半導体素子を用いている SSD（ソリッドステイトドライブ）が存在する．SSD は HDD のような磁気ディスクと比較して，ヘッドがアクセス対象となるデータトラックまで動くシーク時間が存在せずランダムアクセスに優れ，物理的駆動箇所がないため衝撃にも強いという特徴がある．また，プラッタの回転やヘッドのシーク動作など駆動箇所がないため省電力である．しかし，SSD は書き込みの処理の際にチップ内の絶縁膜が劣化する．そのため，動作を保証するため書き込み回数の制限が設けられている．HDD と比べ，容量あたりの単価は高価である．ストレージシステムにおいて，すべての記憶媒体を HDD から SSD に取り換えることは性能向上のために望ましいが，その構築は非常にコストが高くなる．そのため，低コストでストレージシステムの性能を向上させるが必要であり，HDD と SSD を組み合わせたデバイスミックスストレージシステムが注目されている．

SSD などの半導体ストレージを HDD などの磁気ストレージのシステムへ組み込むデバイスミックスストレージシステムの手法として Cache 方式と Tiering 方式が存在する (図 1) [1]．Cache 方式とは全てのデータを HDD に配置し，データのアクセスによって HDD から SSD へとデータのコピーをキャッシュする方式である．SSD へのデータのキャッシュは LRU などのキャッシュアルゴリズムによって制御されている．一方，Tiering 方式ではデータを SSD と HDD に分けて配置をする．

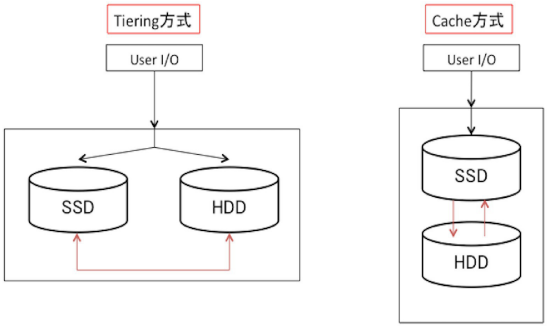

図 1 デバイスミックスストレージシステムイメージ

データのコピーは行わず，SSD と HDD 間でデータの移行を行う．データの移行はデータアクセスの統計情報を基にデバイスの特性を活かすよう，指定されたタイミングで行う．Cahce 方式では一度アクセスが発生したデータを SSD にキャッシュするため，二度目のアクセスから高速化が可能となる．しかし，データのアクセスがランダムなワークロードでは頻繁なキャッシュの入れ替え動作によって，性能の劣化や寿命の短縮が懸念される．また，Tiering 方式では，データのアクセス頻度が多い領域を指定されたタイミングで移行することにより高速化を図っている．しかし，データのアクセスする領域に変化があった場合，移行のタイミングによっては十分な効果が得られない場合がある．よって，SSD と HDD のデバイスミックス環境において効率的な制御が求められている．

2. 関連研究

ストレージ間のデータ配置の最適化を図ることを目的として

様々な研究が行われてきた．ファイルベースでデータの移行を行う研究として次のようなものがある. [2] ではファイル間の相関関係を発見し，HDD から SSD へのプリフェッチを行っており, [3] では時間的に共起しやすいファイルを同一ディスクに配置する手法を提案している．ブロックベースで移行を行う研究として, [4] [5] などの研究がある．これらはブロックの相関を検出し，HDD からメモリへのプリフェッチを行う．データ移行のポリシーを動的に変更する研究として, [6] ではキャッシュのヒット率などに応じてキャッシングやプリフェッチに割り当てるメモリの配分を変更している. [7] ではシステム全体の I/O レイテンシを最小化するようにポリシーを変更している．また，ストレージシステム上を流れるワークロードに注目した研究として, [8] では昼夜でワークロードの特性が大きく切り替わるようなシステムにおいて，特性が切り替わる前に次のワークロードを先読みし，あらかじめデータを SSD へと移行している. [9] では全 I/O の多くが連続した領域に集中し，分オーダーで継続した後，別領域に I/O が移動していくようなワークロードを想定している．I/O 集中フィルタリングを行い，性能向上が期待できる領域を SSD へ移行することで SSD への I/O を集中させている．

また，SSD はフラッシュメモリを使用しているものが一般的であるが，次世代の不揮発性のメモリとして PCM（相変化メモリ）や ReRAM（抵抗変化形メモリ），NRAM（カーボンナノチューブを使用したメモリ），MRAM（磁気抵抗メモリ）等が注目され，ストレージへの需要も高まっている．これらの次世代メモリはフラッシュメモリよりも I/O 性能に優れ，書き込み回数の制限が緩和されているという利点と製品として開発段階であるものも多く，価格が高いという欠点がある．次世代メモリはメインメモリとしての用途に関して，様々な研究が行われている. [10] では，PCM と従来の DRAM を組み合わせたハイブリッドメモリにおいて PCM への Write を緩和するページ置換アルゴリズムの提案を行い，メモリへの Write を予測する方法として 2 つの事象を観測している．1 つ目は時間の周期的なアクセスよりもアクセス頻度に応じた予測が優れていることであり，2 つ目は Read/Write を合わせたアクセス履歴よりも Write のみに注目した履歴を使用する方が予測に優れていることである．デバイスミックスストレージシステムにおいても SSD と HDD では使用の寿命が異なるため，SSD への Write を緩和することでシステム全体の寿命を伸ばすことが可能であると考えられる．

また，デバイスミックスストレージシステムは実際に製品化され，運用されているものも存在する，EMC VNX シリーズ製品における FAST VP という機能 [11] では，1GB あたりの sub-LUN の粒度で I/O の統計情報を集計し，その情報を元に各階層へのデータの再配置を行っている．

3. 本研究の目的

1. で述べたように Cache 方式では頻繁なキャッシュの入れ替え動作による性能の劣化と寿命の短縮が問題となる．また，今後 SSD の大容量化により，使用可能な領域が増大した際，単純にキャッシュによってデータのコピーを配置することは領域を効率的に使用できない．これらの問題を回避するため本研究では Tiering 方式を採用する．2. でのデバイスミックス環境における取り組みではデータのアクセス頻度に注目し，データ再配置における最適化や特定のワークロードにおいてアクセス集中領域を予測する研究は多く行われている．しかしながら，データの再配置の周期に注目して行われている解析は少ない．Tiering 方式では予め決められた時間によって再配置を行うことが多く，次のデータ移行の時間に達するまでアクセスの高速化を行えないという欠点がある．そのため，最適な時間間隔での再配置が必要である．また，デバイスミックス環境は様々な用途で使われることが想定される．上記で述べたように特定のワークロードに対して特徴を捉える研究は行われているが，全てのワークロードに対して最適な制御を行われているものは少ない．さらに Tiering 方式ではアクセスの統計情報を基に決められた領域（extent）単位でデバイス間のデータの移行を行う．Cache 方式のようなブロック単位でのデータ移行は，統計情報を管理するコストを大きくしてしまうからである．そのため，ワークロード毎やアクセスが発生する extent 単位でのアクセス傾向を捉える必要がある．アクセスパターンを知ることにより，最適な時間間隔でデータの再配置を行うことが可能である．アクセス時間間隔を考慮したデータ配置を行い，その管理コストを軽量化することにより従来の Tiering 方式の問題点を解決できると考える．本研究ではデバイスミックス環境の高寿命かつ高効率運用を目的として，データのアクセス時間間隔を考慮したデータ配置の最適化の手法を提案する．本手法の有用性を示すため，以降の分析と実験を行う．

4. アクセス間隔分析

上記で述べたように全てのワークロードに対して最適な制御を行われているものは少ない．そのため，ワークロード毎やアクセスが発生する決められた領域（extent）単位でのアクセス傾向を捉えるため必要がある．アクセスパターンを捉えるため，以下の分析を MicroSoft Research が一般に公開しているサーバーワークロードのブロックトレースを用いて行った [13]．より詳しい分析の内容と考察については [13] を参照されたい．

4.1 MSR Cambridge

MSR Cambridge は MicroSoft Rearch サーバに関する I/O のブロックトレースである [12]．Usr home directories や Project directories，Web/SQL server 等，様々な用途で使用されたワークロードのトレースを一般に公開している．このブロックトレースのデータは 13 servers，36 volumes，179 disks から約 1 週間分の I/O が採取されている．トレースのデータは csv ファイルで公開されており，Timestamp，Hostname，DiskNumber，R/W Type，Starting offset，Responsetime から成る．

4.2 同一 extent へのアクセス間隔分析

同一 extent へのアクセス間隔分析は，ワークロード毎のアクセス特徴を捉えるため行う．ストレージ領域を extent 毎に区切り，extent 内に発生した I/O の Timestamp の差分を計算

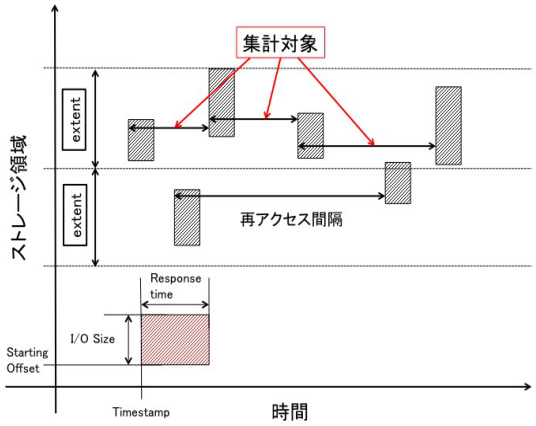

図 2　同一 extent へのアクセス間隔分析

ファイル名	Read 数	Write 数	Read size	Write Size
usr_0	904,493	1,333,406	約 35.05GB	約 13.08GB
mds_0	143,973	1,067,061	約 3.26GB	約 7.37GB
wdev_0	229,529	913,732	約 2.75GB	約 7.15GB

表 1　分析対象トレース

し，extent 毎の再アクセス間隔を集計する（図 2）．どの extent へアクセスが発生しているかは I/O の Starting offset がどの extent 領域内に含まれているかによって判定する．時間間隔として，1ms 以下，1ms から 10ms 以下，10ms から 100ms 以下，100ms から 1s 以下，1s から 60s 以下，60s から 10min 以下，10min から 20min 以下，20min から 1h 以下，1h から 2h 以下，2h から 6h 以下，6h から 12h 以下，12h から 1day 以下，1day から 2days のパターンに分けてアクセスカウントを採取している．この時間間隔の分け方は人間の感覚として分かりやすい時間間隔に設定している．実際の製品におけるデータの再配置なども 3 時間毎，2 日毎など人の手によって設定されている場合が多い．ある extentA のアクセスに対し，次の extentA へのアクセスが 40min 後だったとすると，20min から 1h 以下のアクセスをインクリメントする．ここでは User home directories（usr_0），Media server（mds_0），Test web server（wdev_0）のワークロードについて分析した結果を示す（図 3 から図 5）．この時の extent のサイズは 1GB としている．縦軸が extent へのアクセス回数，横軸がアクセス間隔である．それぞれのワークロードの詳細は表 1 である．

ワークロードごとに extent へのアクセス回数の偏りやアクセスの時間間隔に違いがあることが分かる．usr_0（図 3）はアクセス間隔が 1ms 以下や 10ms から 100ms 以下が多くを占める．また 1s から 60s の間隔のアクセスも比較的多く存在する．mds_0（図 4）では usr_0 と比較して 10ms から 100ms 以下のアクセス間隔のパターンが一番多く，続いて 1ms から 10ms 以下のアクセス間隔，1ms 以下のアクセス間隔が多くを占めるワークロードとなる．wdev_0（図 5）では 1ms 以下と 1s から 60s 以下のアクセス間隔が大半を占めるワークロードとなる．これらのアクセス分析からストレージのアクセスパターンはワークロード毎で異なることが分かる．また，60s から 10min

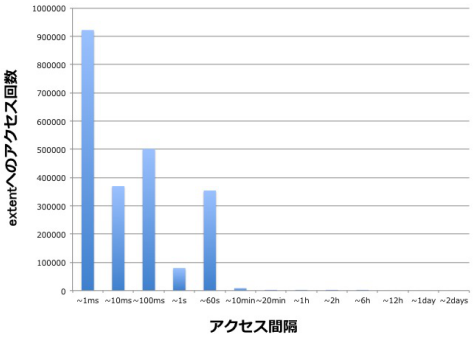

図 3　usr_0

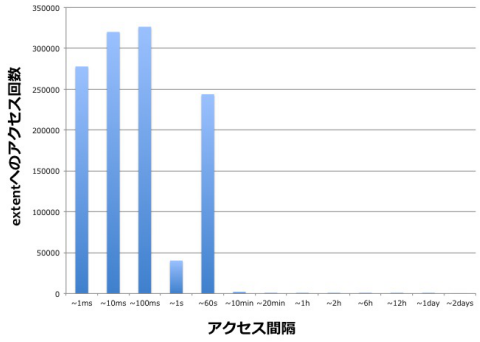

図 4　mds_0

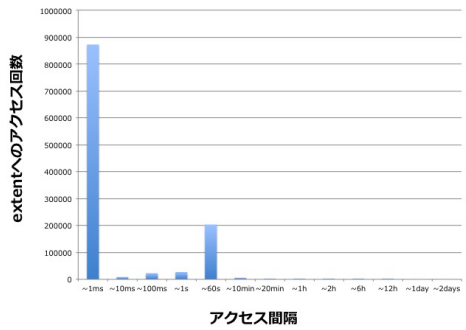

図 5　wdev_0

以下のアクセス間隔やそれより長いアクセス間隔のパターンも少なくはあるが検知することができた．

4.3　extent 毎のアクセスの持続性に関する分析

4.2 からワークロード毎にアクセスパターンが異なることが分かった．この分析ではワークロード全体としての傾向を得ることができたが，どの extent にアクセスがあったのかという情報はない．そのため，直接データ配置に活かすことが難しい．extent 毎の傾向のアクセスの傾向を知る必要があり，extent 毎のアクセスの持続性に関する分析を行う．extent 毎の特徴を捉えることにより，Tiering に向いた移行対象の extent の発見ができると考える．この分析で使用するワークロードは usr_0 を用いる．1 週間のブロックトレースのうち，I/O 数が最も多い

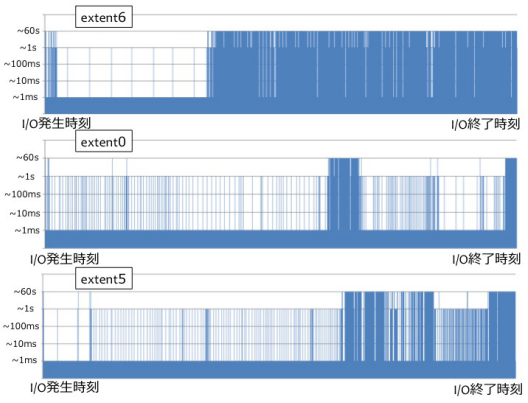

図 6　I/O 数上位の extent に対するアクセスの持続性

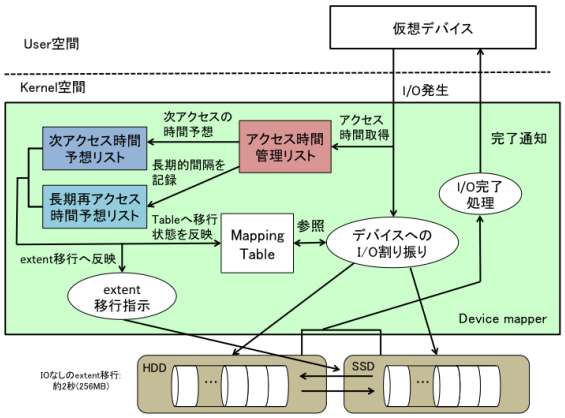

図 7　システム概要図

日の 3 時間を抽出し，その I/O 数上位の extent に対し，4.2 の分析における時間間隔がどの程度持続するかについて分析する（図 6）．この時の extent のサイズは 512MB としている．I/O 数上位となる extent の番号は上位の番号から 6，0，5 である．縦の軸は，アクセス間隔を示しており，1ms 以下，1ms から 10ms 以下，10ms から 100ms 以下，100ms から 1s 以下，1s から 60s 以下のアクセス間隔が発生したことを示す．横軸はワークロードにおいて extent への初めて I/O が発生してからの時間の流れである．縦の値が 1ms 以下になっている時刻では前のアクセスから次のアクセスまでの時間間隔が 1ms 以下であったことを示している．図 6 が示すように，ワークロード内の extent 毎でアクセスの持続性は異なる．最も I/O 数が多い extent である extent6 ではワークロードの始まりは 100ms から 1s 以下や 1s から 60s 以下の比較的長いアクセス間隔が少しの間継続し，1s 以下のアクセスパターンが連続する．その後は 1s から 60s 以下のアクセス間隔が多くを占める傾向がある．extent0 では 1s 以下のアクセス間隔が継続した後，中盤に 1s から 60s 以下のアクセスが頻出し，再び短いアクセス間隔が継続する傾向がある．extent5 についても extent6 と extent0 とは違う傾向がある．このため，あるワークロード内のそれぞれの extent についてもアクセスパターンが異なることが分かる．

5.　次アクセス予想制御

4. の分析結果より，ワークロード毎にアクセス間隔の分布は異なり，extent 毎でもアクセスの持続性が異なることが分かった．そのため，ワークロード毎，extent 毎で適切なデータ移行の管理を行う必要がある．extent 毎のアクセスの持続性の分析で見られたアクセスパターンの変化をうまく捉えるためには個々の extent でアクセスの時間情報を管理することが望ましい．Tiering 方式において，アクセス予測に用いる情報の集計・管理・解析にはコストがかかる．特に extent のアクセス間隔分析のように 1s 以下のアクセス間隔が多く存在する場合，アクセス情報の解析に時間をかけると短いアクセス間隔の活用は難しい．本稿では，アクセス単位で extent 毎のアクセス時間情報を管理するリストを用いて，アクセス情報の管理を軽量化し，アクセス時間間隔から次のアクセスを予想するデータ移行を提案する．

5.1　システムの概要

提案システムは OS ドライバとして動作する．Linux Kernel が提供する device-mapper と呼ばれるフレームワークを利用し実装を行った．device-mapper はブロック型デバイスドライバをサポートするライブラリ群であり，複数のブロック型の実デバイスを管理し，仮想デバイスの作成や管理に利用できる．そのため，ユーザは仮想デバイスにアクセスすることで通常通りに使用することができる．HDD から SSD に移行した情報はドライバ自身が Mapping Table を保持しており，ユーザからの I/O はこの Mapping Table を参照することにより，実デバイスへと振り分けられる．Write の I/O が移行中の extent へ発行された場合は，I/O 待ちキューに追加する．キュー内の I/O は移行終了後、対象の extent へ I/O を発行される．実際のデータ移行は本システムが保持する次アクセス予想リストを基にデータの移行を決定する．また，次アクセス予想リストは同様に保持しているアクセス時間管理リストによって作成される．本システムの概要図を図 7 に示す．

5.2　アクセス時間管理リスト

アクセス時間管理リスト（図 8）では，extent へアクセスが発生した時刻をリング状のリストで管理する．従来の Tiering 手法では時間単位でアクセス履歴を保持し，データ移行へと活用していたが本システムではアクセス単位によりアクセス履歴を発生時刻と共に保持する．一定回数のアクセス（Window Size）分の時刻を各 extent 毎が記憶している．一定回数のアクセス間隔を集計し，平均値または最頻値を算出する．算出した値と最新のアクセスが発生した時刻から次のアクセスが発生する時刻を予想する．リングリストのポインタを進めて打刻することにより，古いアクセス間隔は破棄され，新しいアクセス間隔がアクセス毎に逐次更新されていく．新たなアクセスが発生した場合，アクセス時間管理リストは以下のような動作を行う．

（1）　アクセス時刻を取得

（2）　リングリストのポインタが指し示す時刻（Window Size 内の最も古いアクセス）と 2 番目に古いアクセスの発生時刻のアクセス間隔を集計値より減算

（3）　最新のアクセス時刻をリングリストに記録し，1 つ前のアクセス時刻とのアクセス間隔を集計値へ加算

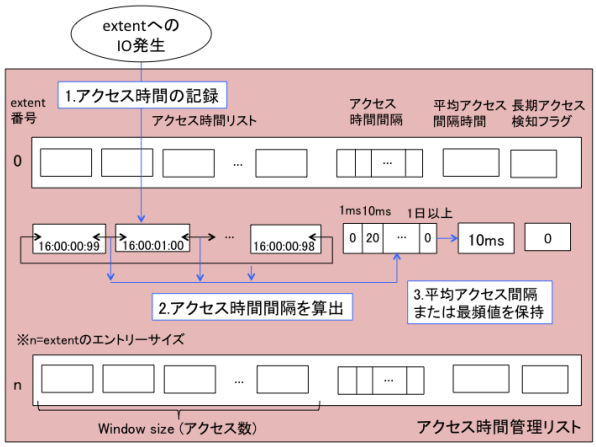

図 8　アクセス時間管理リスト

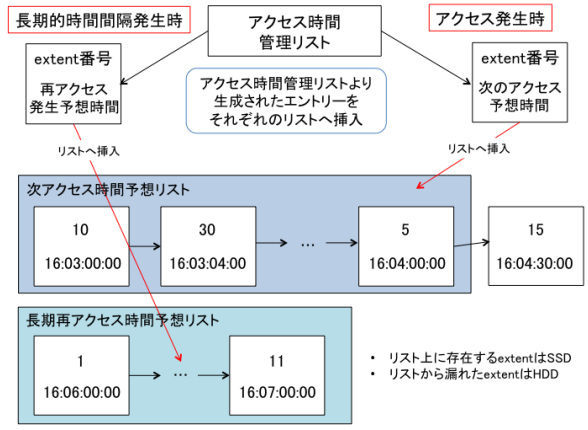

図 9　次アクセス予想リスト

（4）　アクセス間隔の平均値または最頻値の更新

（5）　次アクセス予想時刻の更新

このようなリスト操作をアクセス毎に行うことにより，アクセス間隔をそれ以前のアクセスパターンから更新し，アクセス履歴の解析時間の短縮を行うことが可能である．

長期アクセス検知フラグを用意しており，長期的なアクセス間隔を検知していた場合，フラグを立て予め長期アクセス用に割り当てておいた領域へプリフェッチを行う．

5.3　アクセス予想リスト

アクセス予想リスト（図 9）では，アクセス時間管理リストによって算出した次アクセス予想時刻をリスト上で管理する．このリストを利用して，SSD から HDD または HDD から SSD のデータの移行を行う．アクセス予想リストは次アクセス時間予想リストと長期的再アクセス時間予想リストの 2 本のリストを持つ．これらのリスト上に存在する extent が SSD に存在することとなる．2 本のリストの長さの合計は使用可能な SSD の extent 分のサイズである．

5.3.1　次アクセス時間予想リスト

次アクセス時間予想リストはアクセス時間管理リストの次アクセス予想時刻に基づいて更新される．アクセス時間管理リストにより，次アクセス予想時刻は各 extent のアクセス毎に更新される．その際に，リスト挿入用に extent 番号と次にアクセスが発生する予想時刻を保持したエントリーを作成する．リストは時間順に並んでおり，新しく作成したエントリーのアクセス時間が時間順に並ぶよう挿入する．挿入する時以下の 3 通りの場合が考えられ，それぞれの場合の処理を示す．

（1）　リストに同一の extent が存在する場合

リスト上の同一 extent エントリーを削除し，時間順に挿入する．この場合，リストのサイズは変わらないので移行に影響はない．

（2）　リストに同一 extent が存在せず，挿入箇所がリストの末尾の場合

挿入された新たなエントリーを削除する．最もアクセス時間が遅いエントリーのため，SSD に移行する必要がなく，移行は行われない．

（3）　リストに同一 extent が存在せず，挿入箇所が末尾以外の場合

末尾のエントリーの extent を HDD へ戻し，新たに挿入されたエントリーの extent を SSD へと移行させる．この時，extent の移行が発生する．

上記のリスト操作により，リスト上のエントリーの extent は SSD へ移行され，リストから漏れたエントリーの extent は HDD へと移行される．また，リストに存在していながら以降アクセスがなかった extent に対しては予想時間が更新されず，リストに残り続けてしまう問題点がある．その問題を回避すべく，現時刻とリスト内の最も早い予想時間を比較し，現時刻が予想時間を超えていた場合，リストから削除し，extent を HDD へ移行する処理を行っている．リストで管理するアクセス予想時間はアクセス時間管理リストにおける平均値または最頻値によって導出される．

また，次アクセス時間予想リストはアクセス毎に更新が発生する．リストの更新によるデバイス間の extent 移行が頻繁に発生すると，I/O 待ちが多くなり性能へ影響を及ぼす．そのため，以下の二つの機能により，性能を劣化させるような extent の移行を抑制する．

- Time restraint

extent の移行が次のアクセス予想時間に間に合わない場合，extent 移行を行わない．

- Queue-length based restraint

I/O 待ちキューの長さによって，システムの負荷を判断し，負荷が大きい場合，extent 移行を行わない．

4.2 で述べたように比較的長いアクセス間隔が発生する extent も存在する．こうした extent で長いアクセス間隔時間が記録された場合，平均値によるアクセス予想では次時間アクセス予想リストに反映されにくいという欠点がある．そのため，次の長期的再アクセス時間予想リストを用いて別の移行を行うことで，次時間アクセス予想リストでは漏れてしまう extent へのアクセスを捉えることができる．

5.3.2　長期的再アクセス時間予想リスト

長期的再アクセス時間予想リストは 5.2 で述べた長期アクセス検知フラグが検出された際に更新される．次アクセス時間予

CPU	Xeon X5460 3.16GHz
MEMORY	48GB
OS	Linux 2.6.32（x86_64）
SSD	SAS 3.0 MLC 400GB
HDD	SATA 3.0 7200rpm 1TB

表 2　実 験 環 境

ファイル名	Read 数	Write 数	Read size	Write Size
usr_0	39,537	10,138	約 1.12GB	約 0.12GB
mds_0	6,169	7.993	約 25.54MB	約 77.41MB
wdev_0	35,732	135,482	約 0.43GB	約 1.09GB

表 3　実験データ

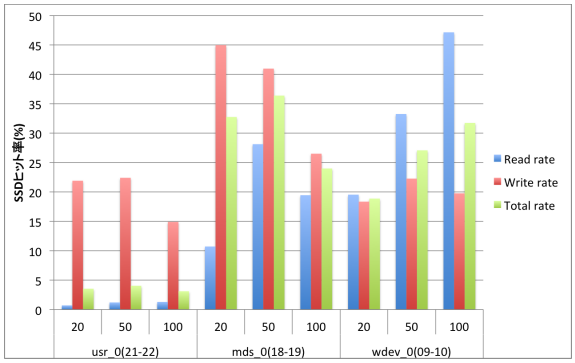

図 10　Window Size を変更した SSD ヒット率

想リストと同様，extent 番号と再アクセスが発生する予想時刻と保持したエントリーを作成し，リストへと挿入する．リストに挿入された場合，SSD へのプリフェッチを行う．長期的なアクセス時間は 4.2 の分析で検知できた 60s から 10min 以下の時間間隔を用いたい．

5.4　特　　徴

本システムではアクセスの履歴管理を全てリスト操作のみにより行っている．一定アクセス数内におけるアクセス情報の逐次更新により，前までのアクセスパターンを考慮しながら次のアクセス予想を行うことを可能とした．そのため，従来の一定時間内での集計・解析・予想と比較し，解析時間を短縮及びデータを配置する適切なデバイス選択が可能となったのではないかと考える．また，アクセス毎に管理情報を更新することにより，柔軟なアクセスのパターンの変化も捉えることが出来るようになると考える．Tiering 方式の問題点であった再配置の周期によって，十分な効果が得られない点をアクセスパターンの変化を即座に捉えることで改善できると考える．

6.　実　　験

提案システムの有用性を示すため，デバイスミックス環境における SSD のヒット率と平均応答時間についての実験を Time restraint で移行を抑制した提案手法，Queue-length restraint で移行を抑制した提案手法，Cache 方式である FlashCache と従来 Tiering 手法に対して行う．FlashCache [14] は，Facebook が開発した SSD を HDD のキャッシュとして利用可能なデバイスドライバである．device-mapper のフレームワークを用いて実装されている．また従来 Tiering 方式では extent に発生した I/O 数をカウントし，一定時間毎に I/O 数上位の extent を HDD から SSD へ移行する．実験環境は表 2 に示す．

6.1　実験データ

本実験では表 3 に示すトレースファイルを対象とし，I/O トレースのリプレイを各方式に行った．表 3 は表 1 で示した 1 週間分の I/O トレースの内，I/O 数の最も多い日付の 1 時間を抽出したトレースデータである．リプレイには btreplay（トレースのリプレイツール）[15] を利用した．

6.2　Window Size による効果

まず提案手法における Window Size を決定するため，Window Size を 20，50，100 と変化させた場合のヒット率を示す．図 10 で示したグラフが usr_0，mds_0，wdev_0 で Widow Size を変更し，実行した結果である．

usr_0 では Window Size を変更しても，Total のヒット率に大きな差はない．しかしながら 100 の場合，Write のヒット率が低下が見られる．SSD は Write 操作によって性能の劣化や寿命の短縮が懸念されるため，Write のアクセスパターンによる影響も見られたのではないかと考える．mds_0 では 50 の場合にヒット率が最も良い．その際，Read のヒット率も向上しているため，SSD に移行される extent として適切なものが次アクセス時間予想リストで管理されていると考えられる．wdev_0 では Window Size を大きくするごとにヒット率が向上している．Total のヒット率に伴い，Read のヒット率も向上しているため， mds_0 と同様に適切な extent が移行されている．

6.2.1　Window Size の選択について

Window Size を大きくする場合，アクセスの時間履歴をより多く保存することになる．このため，アクセスの少ない extent は次アクセス予想リストに反映されにくくなり，高頻度でアクセスのある extent が優先的にリストへ反映できる利点がある．しかしながら，ワークロード内でアクセスパターンが変化した時，Window Size 分のアクセス履歴が埋まるまでは過去のアクセス間隔が反映され，アクセスパターンの捉えにくくなる欠点も存在する．3 つのワークロードによって Window Size を変更した際のヒット率の変化は異なることが分かった．そのため，ワークロードによってこれらの利点反映されやすい Window Size とある Window Size からは欠点が反映されやすくなる場合がある．ワークロード毎によって，適切な Window Size の決定が必要であると考える．

6.3　従来手法との比較

比較対象である FlashCache は Block Size を 4KB，従来 Tiering の I/O 集計を行う一定の時間は 10 分間と設定している．また，従来 Tiering と提案手法共に移行を行う extent のサイズは 256MB である．提案手法の Window Size は usr_0 が 50，mds_0 が 50，wdev_0 が 100 である．図 11 から図 16 が実行結果である．使用する SSD の領域は 2GB 分であり，その内 512MB 分を長期的アクセス用に割り当てている．長期的なアクセス時間間隔は 5 分と設定した．

6.3.1　FlashCache との比較

提案手法と FlashCache では SSD ヒット率において、usr_0

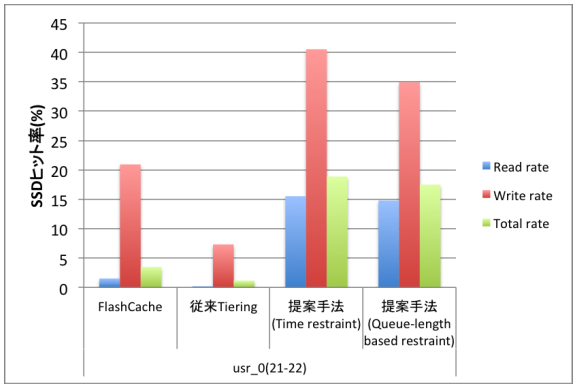

図 11　SSD ヒット率 (usr_0)

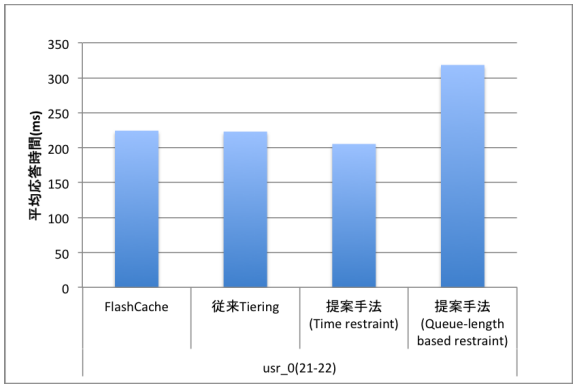

図 12　平均応答時間 (usr_0)

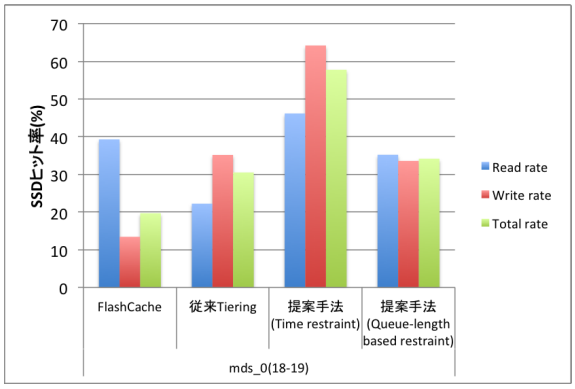

図 13　SSD ヒット率 (mds_0)

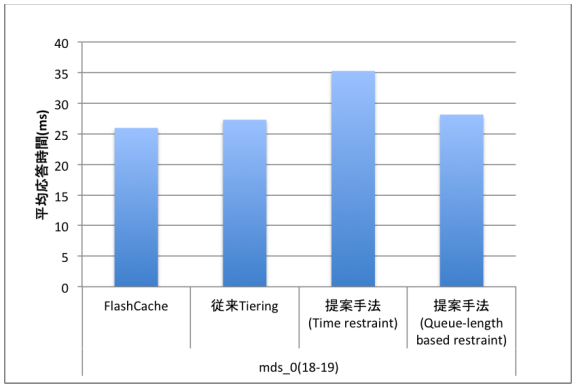

図 14　平均応答時間 (mds_0)

では約 5 倍から約 5.4 倍，mds_0 では約 1.7 倍から 2.9 倍，提案手法が優れていおり，wdev_0 では FlashCache が約 1.2 倍優れている結果となった．また，平均応答時間は usr_0 のワークロードにおいて，Time restraint 付与の提案手法が優れていることが分かった．ワークロードによって，提案手法と FlashCache とでヒット率が大幅に変わってしまう．そのため，Cache 方式に適したワークロードと Tiering 方式に適したワークロードがそれぞれある．Tiering 方式に Cache 方式を組み合わせることにより，それぞれに方式に適したワークロードにも柔軟に対応することが可能であると考える．

6.3.2　従来 Tiering との比較

提案手法と従来 Tiering では SSD ヒット率において，usr_0 では約 15.1 倍から約 16.3 倍，mds では約 1.1 倍から約 1.8 倍，wdev_0 では約 4.1 倍から約 4.6 倍の向上が見られた．平均応答時間は，usr_0 と wdev_0 のワークロードにおいて Time restraint 付与の提案手法が優れており，mds_0 のワークロードにおいて Queue-length based restraint 付与の提案手法と同等であることが分かった．時間毎のアクセスを集計し，ソートする従来 Tiering と比べて，アクセス毎に時間情報を更新する提案手法では移行のため解析のコストが低いため有用であると考えられる．

7.　考　　察

MSR のブロックトレースを用いて，アクセス間隔・持続性の分析を行い，アクセスの時間間隔を用いたデータ配置の制御

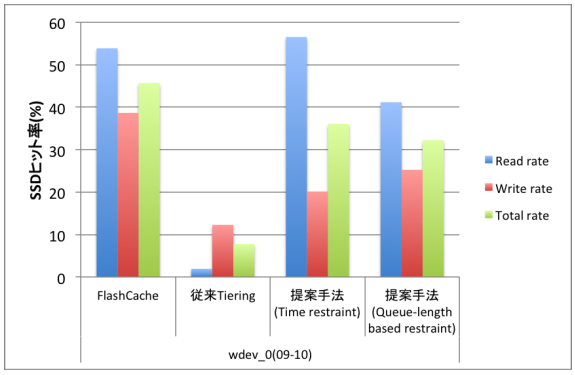

図 15　SSD ヒット率 (wdev_0)

手法の提案を行った．同一 extent へのアクセス時間間隔の分析から各ワークロードについてアクセスパターンが異なることが分かった．extent のアクセスの持続性の分析により，同一のワークロード内においても extent 毎にアクセスパターンは異なり，時間の流れで変化することが分かった．アクセス履歴を管理する幅である Window Size を変更したヒット率の実験においては，ワークロード毎の適切な Windows Size を決定する必要があると分かった．提案手法と FlashCache との実験において，ワークロードによってヒット率と平均応答時間に大幅な差があることが分かった．そのため，Cache 方式に適したワークロードと Tiering 方式に適したワークロードがあると考えられ，Tiering 方式に対し Cache 方式を組み合わせることによって，それぞれに適したワークロードへの対応も可能であると考

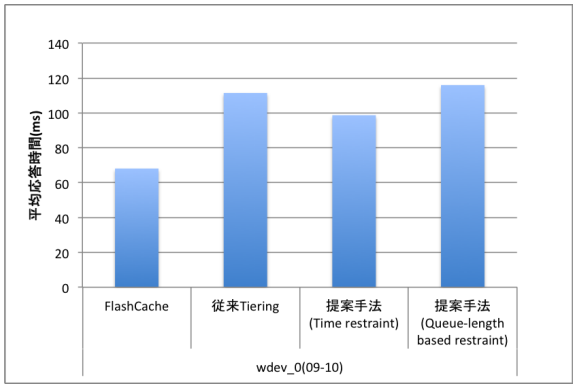

図 16　平均応答時間 (wdev_0)

えられる．また同等の性能である場合，単純にデータのコピーを配置する FlashCache と比較すると領域利用率の面から見ると提案手法は有用であると考えられる．従来 Tiering との比較では mds_0 以外の 2 つのワークロードについてヒット率と平均応答時間において優位性を示せた．統計情報を基にした移行より時間間隔情報を基にした移行の有用性を示せた．アクセス本研究の分析と実験から従来 Tiering と比較してアクセス単位の時間情報管理が有用であると考えられ，Cache 方式との比較ではワークロードによって方式を複合し対応させることができると考えられる．

また，実際のシステムではいくつかのワークロードが複合して実行されることがある．複合ワークロードの場合であっても，本手法ではアクセスパターンを即座に捉えることができるため，有用に働くのではないかと考える．

8. まとめと今後の課題

本研究ではデバイスミックス環境における高寿命かつ高効率なデータ移行の制御手法として，アクセス単位で extent 毎のアクセス時間情報をリストで管理し，アクセス時間間隔から次のアクセスを予想するデータ移行を提案した．本手法では Cache 方式における頻繁なキャッシュの入れ替え動作に性能の劣化やデバイスの寿命短縮の問題を回避するため Tiering 方式を採用した．従来の Tiering 方式において，データ再配置のタイミングが性能に影響を及ぼすため，アクセス時間間隔を考慮したデータ配置により高効率化を図った．また，アクセス単位のリスト更新により，アクセス情報管理の軽量化も行った．本手法の有用性を示すため SSD のヒット率と平均応答時間に関する実験を行った．Window Size を変更した実験においてはワークロードによって適切な Window Size の決定が必要であることが分かった．3 つのワークロードに対し，提案手法と Cache 方式，従来の Tiering 方式の比較を行った結果，Cache 方式とはワークロードによって優位性が異なること，従来の Tiering 方式に対しては優位性があることが分かった．

また，SSD の特徴の観点から Write のヒット率は低下させることが望ましい．そのため，アクセス情報を Read の I/O のみに限って記録するなど Write の SSD へのアクセスを避ける手法についても検討したい．また，今回の実験で得られたワークロードによる適切な方式の選択や Window Size の決定が必要であるという結果からこれらを効率的に導き出す必要性がある考えられる．また，本研究では HDD と SSD を用いたが，次世代メモリの研究も進んでおり，次世代メモリストレージを利用した最適な制御手法についても検討が必要である．

文　献

[1] 林 真一，薦田 憲久，“階層ストレージにおけるボリューム階層化方式と SSD キャッシュ方式の評価，” 電学論C（電子・情報・システム部門），vol.134，No.3，pp.459-465，2014.

[2] T.M. Kroeger, D.D.E. Long, “Design and implementation of a predictive file prefetching algorithm,” Proceedings of the 2001 USENIX Annual Technical Conference, pp.105-118, Jun, 2001.

[3] M. Iritani, H. Yokota, “Effects on Performance and Energy Reduction by File Relocation based on File-access Correlations,” Proceedings of the 1st Workshop on Enegy Data Management, pp.79-86, Mar, 2012

[4] Z. Li, Z. Chen, S.M. Srinivasan, and Y. Zhou, “C-Miner: Mining Block Correlations in Storage System,” Proceedings of the 3rd USENIX Symposium on File and Storage Technologies, pp.173-186, Mar, 2004

[5] X. Ding, S. Jiang, F. Chen, K. Davis, and X. Zhang, “DiskSeen:exploiting disk layout and access history to enhance I/O prefetch,” Proceedings of the 2007 USENIX Annual Technical Conference, pp.17-22, Jun, 2007

[6] R.H. Patterson, G.A.Gibson, E. Ginting, D. Stodolsky, and J. Zelenlka, “Informed prefetching and caching,” Proceedings of the 15th ACM symposium on Operating systems principles, pp.79-95, Dec,1995

[7] D. Vengerov, “A reinforcement learning framework for online data migration in hierarchical storage systems,” The Journal of Supercomputing Vol.43, Issue 1, pp.1-19, Jan, 2008

[8] G. Zhang, L. Chiu, C. Dickey, L. Liu, P. Muench, and S. Seshadri, “Automated lookahead data migration in SSD-enabled multi-tiered storage systems,” Proceeding of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies, pp.1-6, May, 2010

[9] 大江 和一，岩田 聡，本田 岳夫，河場 基行，“On-The-Fly - Automated Storage Tiering（OTF-AST）の提案”，情処学システムソフトウェアとオペレーティング・システム研究会，pp.1-10，Feb，2014

[10] S. Lee, H. Bahn, S.H. Noh, “CLOCK-DWF: A Write-History-Aware Page Replacement Algorithm for Hybrid PCM and DRAM Memory Architectures,” IEEE Transactions on Computers, vol.63, no.9, pp.2187-2200, Sept, 2014

[11] EMC White Paper, EMC VNX FAST VP A Detailed Review, 2013

[12] D. Narayanan, A. Donnelly, and A. Rowstron, “Write Off-Loading:Practical Power Management for Enterprise Storage,” ACM Transactions on Storage, vol.4 Issue 3, no.10, Nov, 2008

[13] 新屋敷 裕太，飯澤 健，小沢 年弘，荒堀 喜貴，横田治夫，“階層ストレージシステムに向けたワークロードのブロックアクセス分析に関する考察，” 信学技報，vol .115，no.177, DE2015-12, pp.1-6，Aug，2015.

[14] M. Srinivasan, https://github.com/facebook/flashcache

[15] A.D. Brunelle, “btrecord and btreplay User Guide ,” http://www.cse.unsw.edu.au/ aaronc/iosched/doc/btreplay.html