Vol.2014-OS-128 No.6
2014/3/7

On-The-Fly - Automated Storage Tiering (OTF-AST) の提案

大江 和一[1,a)]　岩田 聡[1]　本田 岳夫[2]　河場 基行[1]

概要：
ストレージシステムのコストパフォーマンスを向上する目的で，SSD と HDD を組み合わせた階層ストレージシステムが提案されており，一定期間にアクセス頻度の高いデータを SSD に置くことで高速化を図る仕組みである．しかし，ファイル共有ストレージへのアクセスパターンにおいては，記憶領域の狭い範囲に数分から数十分間 IO が集中し，時間と共に別の領域に移動する特徴があるため，長い期間の頻度でデータを移動する従来型の階層ストレージシステムが有効ではない．
本論文では，短い時間で IO 集中が発生した領域をその都度捉えて SSD に移動する *On-The-Fly - Automated Storage Tiering (OTF-AST)* の提案を行う．提案手法では，分単位の統計情報を用い，性能向上効果が期待出来る領域を IO の集中度とその継続時間よりその都度 filtering することで SSD に IO を集中させることに成功した．
今回，公開されているストレージアクセス履歴データである MSR Cambridge ワークロードを対象に評価を行い，SSD アクセス率が従来の階層ストレージシステム比で最大 45%向上し，user IO 平均レスポンスが最大 11%向上することを明らかにした．

キーワード：ストレージ，階層，SSD，cache，tiering，ワークロード，リアルタイム

Proposal for On-The-Fly - Automated Storage Tiering (OTF-AST)

Kazuichi Oe[1,a)]　Satoshi Iwata[1]　Takeo Honda[2]　Motoyuki Kawaba[1]

Abstract:
Storage tiering is a technique to utilize both SSDs and HDDs effectively by storing hot data in SSDs and cold data in HDDs. However, it is difficult for conventional methods to follow workload changes immediately, since migration intervals are often set to hours or even a day. This interval is not short enough to follow some real workloads in which hot regions move around every several minutes.

In this paper, we propose *on-the-fly AST (Automated Storage Tiering)* to follow workload changes in a few minutes. On-the-fly AST fulfils the goal by checking the degree of access concentration and the duration of hot status.

By applying on-the-fly AST to some publicly available trace data, SSD access ratio and average response time are improved by up to 45% and 11% respectively compared to a conventional method.

Keywords: storage, hybrid storage system, SSD, cache, tiering, workload, realtime

1. はじめに

近年，SSD の様な高速なデバイスがストレージデバイスとして用いられる様になってきた．SSD は HDD との比

[1] (株) 富士通研究所
FUJITSU LABORATORIES LTD.
[2] (株) 富士通ソフトウエアテクノロジーズ
FUJITSU SOFTWARE TECHNOLOGIES LIMITED.
[a)] ooe.kazuichi@jp.fujitsu.com

較で高速であるが高価である．そこでコストパフォーマンスを向上する目的で，SSD と HDD を組み合わせた階層ストレージシステムが多数提案されている．これらシステムは，アクセス頻度が高いデータを SSD に置くことで高速化を図る仕組みである．この階層ストレージシステムの主な方式としては，tiering 方式と cache 方式が上げられる．製品レベル [3]，[4] で用いられる static tiering は，数時間〜1 日単位で集計した IO 数が多い領域順に SSD 容量を使いきるところまで事前に配置する方式である．最大で SSD 容量のデータ入れ替えが発生するため再配置オーバーヘッドが大きく，深夜など user IO が少ない時間帯にデータの再配置を行うのが一般的である．cache 方式は FACEBOOK FlashCache[1] など多数の実装が存在し，アクセス頻度が多い領域を LRU などのアルゴリズムで SSD にブロック単位[*2]で cache する．

多くのファイル共有ストレージワークロードには，全容量の数%以下の狭い記憶領域の範囲に数分から数十分間 IO が集中し，時間と共に別の領域に移動する特徴がある [6]，[7]．このファイル共有ストレージワークロードを static tiering に適用しても，IO 集中を効果的に SSD へ集めることは出来ない．数分から数十分間 IO 集中した領域が数時間〜1 日単位で集計すると，IO 数が多い領域と一致せず SSD への移動対象とならないためである．cache 方式にこのワークロードを適用すると，数分から数十分間隔でキャッシュ領域の入れ替えが発生し，その入れ替えオーバーヘッドのため十分な性能向上効果が得られないことが容易に想像できる．

本論文では，短い時間で IO 集中が発生した領域をその都度捉えて SSD に移動する *On-The-Fly - Automated Storage Tiering (OTF-AST)* の提案を行う．提案手法では，分単位の統計情報を用い，性能向上効果が期待出来る領域を IO の集中度とその継続時間よりその都度 filtering することで SSD に IO を集中させること，に成功した．

OTF-AST では，運用中に IO 集中が発生した領域の階層移動を行うので，階層移動時に発生する IO が user IO レスポンスに影響を与えない仕組みの実装が必要になる．本論文の評価システムでは，階層移動時の user IO レスポンスを維持する目的で OS 内部に user IO を一時的に蓄えるバッファを準備した．

性能評価は，IO 集中をどれだけ SSD に集めることが出来たのか（SSD アクセス率）と階層移動に伴う遅延まで含めた user IO レスポンスの観点で行った．比較のため，従来の階層ストレージシステムである static tiering と FACEBOOK FlashCache[1] も同一条件で評価を行った．SSD アクセス率は最大で 46%となり，static tiering の 1%や FlashCache の 37%を上回ることを確認した．user IO の平均レスポンスは，階層移動に伴う IO と user IO で HDD が過負荷にならないケースで効果確認が出来た．提案方式の平均 user IO レスポンスが 34.2ms に対して，static tiering が 50.8ms，FlashCache は 36.6ms であった．

以下に本稿の構成を示す．2 章でストレージワークロードの特徴と既存手法適用時の課題について説明する．3 章で提案を行う OTF-AST に関して説明を行い，4 章で OTF-AST の評価を行う．5 章でまとめを行い，6 章で今後の課題を説明する．

2. ストレージワークロードの特徴と既存手法適用時の課題

2.1 ストレージワークロードの特徴

ストレージワークロード分析の記述がある文献 [6]，[7] よりその特徴を整理すると，全容量の数%以下の領域に大部分の IO が集中し，その集中した IO が数分以上継続し，今までとは異なった領域に IO の集中が移動することが分かる．

文献 [7] は，商用環境で採取した Samba ワークロード半年分に関して，IO の集中度やその継続時間などを定量的に分析した結果を掲載している．この報告より，全容量の 1%の領域に全 IO の 81%が集中し，その集中した IO の 83%は 3 分以上継続する．10 分以上継続するケースも 50%前後あることが分かる．さらに，IO 集中が発生した 1%の領域に再び IO 集中が起きる割合は 30%程度しかなく，残りの 70%は任意の領域に分散することも分かる．この調査結果より，数時間〜1 日単位で IO 数の集計を行うと，約 70% の領域は IO 集中が発生した領域として捉えられないことが分かる．IO 集中が発生する領域の大きさは平均 6GB であり，全体の 90%が 12GB までであることも分かる．

文献 [6] には，MSR Cambridge ワークロード [8] の中から最大 50iops 以上が発生した 13 volumes [*3] の分析結果が掲載されている．ここで，全容量の 1%の領域に全 IO の 8-95%が集中し，全容量の 10%の容量だと全 IO の 48-100%集中することが分かる．IO 集中の継続時間も 10 分以上である．

MSR Cambridge ワークロードに関しては，文献 [6] を元に追加調査を行い，IO 集中する領域の範囲が数分単位で隣接する今まで負荷が低かった領域への移動を繰り返しながら，十数分程度継続することも分かった．

2.2 既存手法適用時の課題

SSD と HDD を組み合わせた階層ストレージシステムの主な方式として，tiering 方式と cache 方式が上げられる．2.1 節にて説明したワークロードを両方式に適用した場合

[*2] 0.5KB,4KB などのサイズ

[*3] mds_1, proj_1, proj_2, proj_4, prxy_1, src1_0, src1_1, src2_1, stg_1, usr_1, usr_2, web_0, web_2

の課題に関して本節で議論する．

2.2.1 tiering 方式

EMC FAST[3] や FUJITSU ETERNUS-AST[4] など製品レベルで用いられている static tiering は，数時間〜1 日単位で集計した IO 数が多い領域順に SSD 容量を使いきるところまで事前配置する方式である．2.1 節で説明したワークロードでは，IO 集中する領域が数分から数十分の単位で移動すること．さらに，IO 集中の中の 70%は異なった領域に発生することが分かる．そのため，数時間〜1 日単位で IO 数を集計すると，この短時間で移動する IO 集中領域を捉えることが出来ず，static tiering では十分な性能向上効果が得られないことが分かる．

Hystor[5] は，研究レベルのシステムであるが，IO 数の変化を常時モニタリングし IO 数が多い領域を SSD へ移動する提案が行われている．しかし，モニタリングの単位は 15 分以上であり，数分単位で IO 集中が移動するワークロードへの適用は困難である．

文献 [6] では，HDD の busy 率を 1 分間隔で監視し，HDD が過負荷になると過負荷を解消のに必要な領域を SSD に移動することで性能向上を行う提案が行われている．HDD が過負荷かどうかを判断条件としているため，目的とする IO 集中を的確に捉えることが出来ない．

Jorge らの論文 [9] では，本稿の様に性能向上を目的とするのではなく，各デバイスの価格性能比や容量性能比を用いて，コストや消費電力が最小になるようにデータ配置を行う提案が行われている．

Raja らの論文 [10] では，彼らが開発した Loris 上での Cacheing と dynamic storage tiering の比較評価が行われている．この中で，Loris-based Hot-DST(Dynamic Storage Tiering) system は，IO 集中となった hot files を dynamic に SSD に移動する提案が行われている．しかし，本論文で提案する IO の集中度とその継続時間による filtering は行われていない．

Xiaojian らの論文 [11] では，SSD と HDD を用いた hybrid 構成のストレージに関して，SSD と HDD でレスポンスが均等になる様に IO 量を配分することで，SSD と HDD の性能を使いきる提案が行われている．

2.2.2 cache 方式

cache 方式の代表例として，FACEBOOK FlashCache[1] や FUSION IO DIRECTCACHE[2] などが上げられる．cache 方式は，IO 数が多い領域がストレージボリュームの広範囲に分散し，その領域が一度に大幅に変わらないワークロードで特に効果的である．IO 数が多い領域が頻繁に入れ替わるワークロードでは，キャッシュブロック入れ替えに伴う IO が大量に発生し性能遅延を引き起こす可能性がある (文献 [7] 3 章)．

Yiying らの論文 [12] では，キャッシュの大容量化が進みキャッシュ効果が得られるまで時間がかかる問題に取り組んでいる．解決方法は，最近アクセスしたブロック (Last-K) を監視し，Logging Volume に Warmup Data として蓄えておく．システムを再起動した場合は，この Warmup data をキャッシュに展開することで cold start 直後の性能向上を果たしている．本論文で取り上げる課題や提案方式との関連は薄いが，cache 方式における性能向上研究として紹介した．

3. On-The-Fly - Automated Storage Tiering(OTF-AST) の提案

3.1 概要

SSD と HDD を用いた階層ストレージシステムを前提に，2.1 節で述べたワークロードに効果的に処理する目的で，我々は 1 分前後の間隔で IO 数の変化をモニタリングし，IO 数が多い領域をその都度 SSD へ up し，SSD up 済領域の IO 数が収束したらその都度 HDD へ down することが可能な *On-The-Fly - Automated Storage Tiering (OTF-AST)* の提案を行う．

1 分前後の短い間隔で SSD への up/down を行う場合，SSD 移動に必要な時間以上に IO 集中が継続する領域を抽出し SSD へ移動することと SSD up/down 時に user IO のレスポンス悪化が起きないようにすることが重要になる．

最初に SSD へ移動する領域の抽出方法について説明する．2.1 節より，IO 集中が発生する 1 つの領域の平均サイズは 6GB であり，全体の 90%が 12GB までであることが分かる．継続時間は 3-10 分前後が支配的であることが分かる．この中で，出来るだけ狭い領域に全 IO の大部分が集まり，その領域への IO 集中が出来るだけ長時間継続するケースを抽出して SSD へ移動することが最も効果的と考えている．これは，SSD への移動が短時間で収束し，長時間の SSD hits を期待出来るためである．この方針を具体化するために IO がどの程度狭い領域に集中しているのか，とその狭い領域に集中した IO がどの程度継続するのか，という 2 つの観点で filtering を行う．IO の集中度に関しては，2.1 節の分析結果を参考に，全容量の数%程度の領域に全 IO の少なくとも半分程度が集中するケースの抽出を行う．さらに，IO 集中が発生した周囲の領域に関して，あらかじめ決められた値以上の領域も IO 集中が発生した領域の一部として抽出する．この周囲の領域は，近い将来 IO 集中が起きる可脳性が高いことが MSR Cambridge ワークロード [8] などの調査で分っている．IO の継続時間に関しては，1-2 分で収束する IO 集中を SSD への移動領域から取り除くのが filtering の目的となる．事前に MSR Cambridge ワークロード [8] などの調査を行ったところ，3 分前後継続した IO 集中がさらに 3 分以上継続する可能性が高いことが分かった．そこで，IO 集中が発生した領域のうち，3 分以上継続した領域のみを SSD への移動対象と

することにした．3.2 節にて，詳細なアルゴリズムの説明を行う．

次に SSD up/down 実行時の user IO レスポンスを維持する方法について説明する．本提案では，SSD up/down を行っている領域への user IO を一旦一時バッファに蓄積し，移動が終ったら一時バッファに書き込んだデータを移動先の SSD or HDD に反映することで，SSD up/down 実行時でも user IO レスポンスを出来るだけ維持出来る設計を行った．3.3.2 節にて，より詳細な設計内容について説明する．

3.2 提案するアルゴリズム

3.2.1 準備

最初にアルゴリズム説明に必要な用語の定義を行う．これより，サーバから認識される 1 つのストレージボリュームのことを LUN[*4] と呼ぶ．さらに，LUN をあらかじめ決められたサイズで分割した 1 単位を subLUN とよぶ．subLUN には LBA が小さい順に subLUN ID が採番される．例えば，300GB の容量がある LUN を 1GB 単位で分割すると，300 個の subLUN を持つことになり，1subLUN の大きさは 1GB である．subLUN ID は 1 から始まって 300 で終了する．ここでは，2.1 節の分析結果を参考に subLUN=1GB で議論を行う．

次にこのアルゴリズムを動かす上での入力情報について説明する．このアルゴリズムを動かすには，一定時間間隔 (t) ごとに subLUN 単位で IO 数を集計したデータが必要になる．一定時間 (t) は移動判定を行う時間間隔に対応するので，例えば 1 分間隔で移動判定を行うのなら，この一定時間間隔は 1 分に設定する．

同時に階層移動を行う最大 subLUN 数 (n) の設定も必要になる．これは一定時間 (t) 内に移動可能な subLUN 数を目安に設定する．例えば，t=1 分で 1subLUN の移動に 3 秒必要なら，最大 subLUN 数 (n) は 20 前後に設定する．

SSD 移動可否を判断する IO 割合 (m) も設定しておく．この IO 割合 (m) とは，全 IO に占める割合のことを指し，IO 数が多い順に最大 subLUN 数 (n) までの subLUN の合計 IO 数が IO 割合 (m) を超えると，その最大 subLUN 数 (n) までに入った subLUN 群を SSD への移動候補として扱う．IO 割合 (m) は，値を大きく設定すると subLUN 移動後の性能向上効果は大きくなるが subLUN 移動が置きにくくなり，逆に値を小さくすると subLUN 移動後の性能向上効果は小さくなるが subLUN 移動頻度は上昇する．MSR Cambridge ワークロード [8] を用いた事前調査では，60%前後の設定が効果的であった．

IO 集中継続数 (c) について説明する．このパラメータは IO 割合 (m) で抽出した subLUN の IO 集中が何分継続するのか，を判断するのに用いる．例えば，3 分以上 IO 集中が継続した subLUN を SSD への移動対象とするのなら，C=3 に設定し同一 subLUN が連続して 3 回 IO 割合 (m) の条件を満たすのかを監視し，条件を満たしたら対象 subLUN を SSD に移動する．

SSD から落とす条件を判断する timeout(o) について説明する．SSD に移動した subLUN に関して，IO 割合 (m) を用いて抽出する SSD 移動候補に o 回連続して入らないと SSD から落とす判定を行う．例えば，o=10 に設定すると，連続して 10 回 SSD 移動候補から外れると SSD から追い出される．

最後に提案アルゴリズムを cut off する iops 閾値 (i) の説明を行う．LUN 全体に発生する負荷が低い状態では，提案アルゴリズムを適用して階層移動を行っても，大きな効果は望めない．そこで LUN 全体の iops が i を下回ったら，アルゴリズムを適用しないことにした．例えば i=50 とすると，LUN 全体の iops が 50 未満の場合は次のデータを受け取るまで何もしない．i の決め方は，直前の数日間の平均 iops 値などを用いる．

3.2.2 移動判定アルゴリズム

3.2.2.1 SSD への移動候補抽出

まず，subLUN 単位の IO 数が入力された 1 つのデータを用いて SSD への移動候補を抽出する．図 1 が SSD への移動候補抽出の説明図である．抽出方法は，アルゴリズムを動作させる環境上で 1-2 分で移動可能な範囲に入った IO 数が多い subLUN を抽出し，隣接 subLUN でグルーピングした上で，あらかじめ決めておいた IO 集中 (図では 60%) 以上となると SSD への移動候補とする，アルゴリズムである．隣接 subLUN でグルーピングする目的は，IO 集中が発生した subLUN の周囲に近い将来負荷が移動する可能性が高いことが MSR Cambridge ワークロード [8] 調査で分かったためである．

このアルゴリズムは，SSD 移動による効果を厳密に判断するのではなく，移動によって効果が出る可能性が高いケースを抽出して投機的に動かす方式である．3.2.1 節で説明した各パラメータは，このアルゴリズムを適用する LUN の直近の数日程度をワークロード分析し，アルゴリズムが効果的に機能出来るよう設定する．

具体的な手順を説明する．Step 1 では，ある時間 T の subLUN 単位の IO 数情報を獲得したら，LUN 全体の IO 数を計算し，その値を t で割ることで LUN 全体の iops を求める．ここでこの値が i を下回っていたら，このデータでの処理を終了し，次データ待ちになる．Step 2 は，獲得したデータに関して IO 数が多い順に subLUN を sort し，最大 subLUN 数 (n) までの subLUN で cut off する．Step 3 は，残った subLUN 群の中から隣接 subLUN でグルーピングする．さらに，各グループの合計 IO 数順に sort する．Step 4 は，subLUN グループの IO 数を加算していき，全

[*4] Logical Unit Number

IO の m% を超えた subLUN グループで cuf off する．ここまでが SSD への移動候補である．

図 1 の例では，Step 1 で i=50iops 以上の負荷が発生していたため Step 2 に進み，Step 2 では subLUN ID=68 までの 20subLUN で cut off する．Step 3 では，Step 2 で cut off した上位 20subLUN を隣接 subLUN でグルーピングし，グループの合計 IO 数で sort しておく．最後の Step 4 で，IO 数の多い順に subLUN グループの IO 数を加算していき，m=60%を超えたところで cut off する．

3.2.2.2 SSD への移動判定

3.2.2.1 節で説明した SSD への移動候補となった subLUN グループが c 回連続して移動候補となれば，実際に SSD への移動を行う．図 2 を例に取ると，ある時刻 T で SSD 移動候補に入った subLUN ID=65,66,67,68 が，次のデータが得られる T+60，さらにその次の T+120 でも移動候補に入っており，時刻 T+120 の時点で c=3 回連続となり，SSD への移動が行われる．T,T+60,T+120 で subLUN ID は必ずしも一致しないが，重複した subLUN ID が存在すれば同一 subLUN グループへの負荷と見なす．さらに，SSD へ移動判定が行われた subLUN グループは，その後のデータで新たな subLUN が subLUN グループに加わると即座に SSD への移動を行う．図 2 では時刻 T+120 で subLUN ID=65-68 が SSD への移動判定となるが，時刻 T+180 で subLUN ID=69 が新たに subLUN グループに加わり，即座に SSD への移動が実行される．

3.2.2.3 HDD への移動判定

HDD への移動は，subLUN グループへの負荷が完全に収束したタイミングで行う必要がある．文献 [7] によると，一旦 IO 集中が収束しても数分以内に再び IO 集中が発生するケースがあり，移動コストを鑑みると負荷が完全に収束した後に HDD への移動を行う必要がある．timeout(o) は，直前のワークロード分析結果より，完全に IO が収束する値を抽出して設定する．

SSD 移動済みになった subLUN グループに対しても，新しい subLUN 単位の IO 数データを得るたびに，IO 割合 (m) までの subLUN に含まれるかどうかを監視する．もし，timeout(o) 回連続して IO 割合 (m) に入らなかった subLUN は HDD へ移動する．図 3 は，SSD 移動となった subLUN グループの一部の subLUN を HDD へ移動する例である．図 4 は，SSD 移動となった subLUN グループの全 subLUN を HDD へ移動する例である．

3.3 Linux 上での実装方法

3.3.1 システム構成

図 5 は，OTF-AST の Linux 上での実装概要図である．3.2 節で説明したアルゴリズムが動作する，**分析・構成変更エンジン**と user IO の振り分けや SSD〜HDD 間の subLUN 移動を制御する *tiering driver* から構成される．

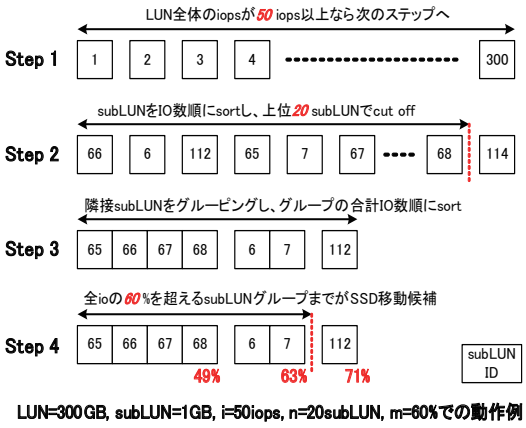

図 1 OTF-AST のアルゴリズム（SSD 移動候補 subLUN 抽出）

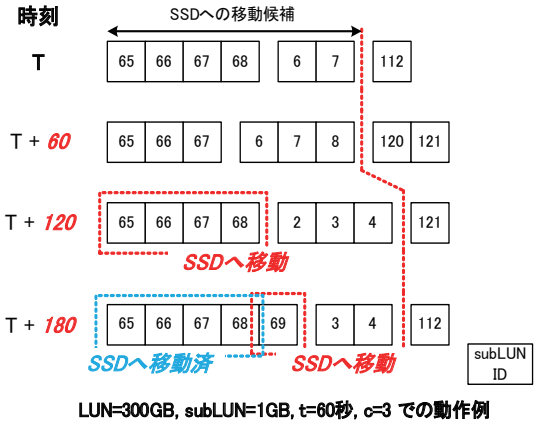

図 2 OTF-AST のアルゴリズム（SSD 移動 subLUN 抽出）

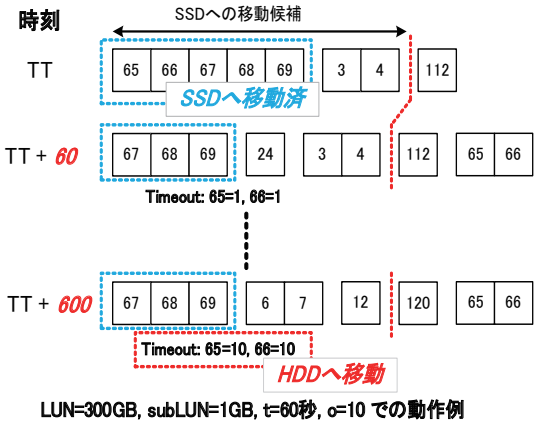

図 3 OTF-AST のアルゴリズム（HDD 移動（一部の subLUN））

分析・構成変更エンジンは，Log pool，ワークロード分析，subLUN 移動指示，の 3 コンポーネントから構成される．Log pool は blktrace を 1 分間隔で実行し，実行結果より subLUN 単位の IO 数を抽出し，timestamp 情報と共に保存する．ワークロード分析は，Log pool に新しい subLUN 単位の IO 数情報が登録されると，その情報を取り出して 3.2 節で説明したアルゴリズムで SSD/HDD 移動判定を行う．SSD/HDD に移動が必要な subLUN を抽出すると，subLUN 移動指示にその subLUN 情報を渡す．subLUN 移動指示は，指示に従って tiering driver に

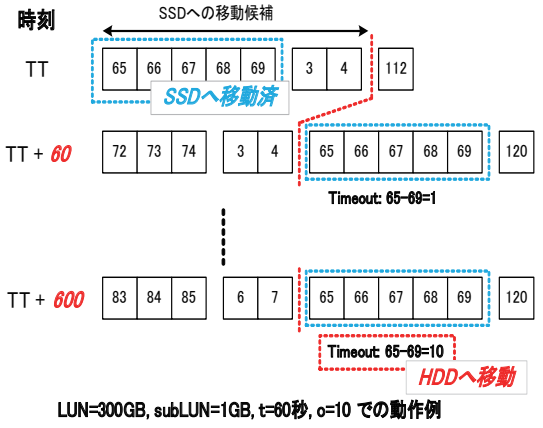

図 4　OTF-AST のアルゴリズム（HDD 移動（全 subLUN））

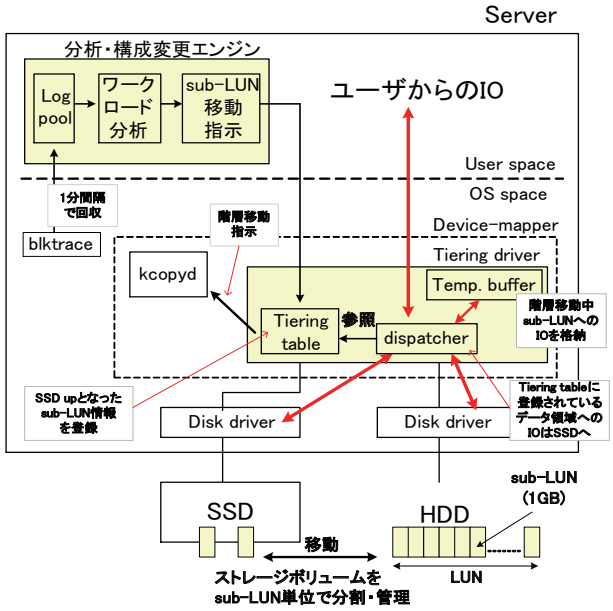

図 5　OTF-AST の Linux 上での実装

subLUN 移動を指示する．

tiering driver は，tiering table，dispatcher，temporary buffer から構成される．tiering table は，SSD 移動済，SSD/HDD への移動中 subLUN 情報の管理を行っており，分析・構成変更エンジンから subLUN 移動指示が来ると，kcopyd [*5] を用いて subLUN 移動を行う．dispatcher は，user IO が来ると tiering table を参照し，SSD 移動済み subLUN への IO なら SSD に IO を転送する．SSD/HDD 移動中 subLUN への IO なら，write は temporary buffer へ書き込む．read は temporary buffer にデータがあれば temporary buffer から読み出す．それ以外の IO は全て HDD に IO を転送する．temporary buffer に書き込まれたデータは，subLUN 移動が終わった後に移動先の SSD or HDD に反映する．temporary buffer に関しては，3.3.2 節で詳細に説明する．

[*5] Linux device-mapper に準備されているモジュール

3.3.2　階層移動処理に用いる temporary buffer

OTF-AST では，システム運用時に IO 集中が発生する subLUN を短時間で検出し，SSD への移動を行う．そのため，subLUN 移動に伴う負荷が出来るだけ user IO 性能に影響しない対策が必要になる．我々はこの問題を解決するために，subLUN と同一サイズの temporary buffer を tiering driver 内に準備し，階層移動を行っている subLUN への user IO write を temporary buffer に書き込むことでレスポンス低下が起きない実装を加えた．

temporary buffer の管理は bitmap で行う．512bytes 単位[*6] に 1bit 割り当て，データの書き込みが行われると対応する bit を on にする実装とした．

階層移動中 subLUN への write は全て temporary buffer に行う．read は temporary buffer に書き込みがあればそこから read し，そうでないケースは移動元 SSD or HDD に転送する．

最後に subLUN 移動完了後の temporary buffer と移動先 SSD or HDD との同期について説明する．temporary buffer 管理 bitmap の検索を address 順に行い，bitmap で on になっている領域を移動先 SSD or HDD に書き込む．この際，同期に伴う IO が user IO のレスポンス低下を起こさない様にするため，同期 IO のフロー制御を組み込んだ．今回の実装では，500iops 以上は同期 IO が出ない設定とした．移動先 SSD or HDD と同期を行っている時の user IO は，user IO の書き込み先が同期が終わった領域だと移動先の SSD or HDD に転送し，同期が終わってない領域なら temporary buffer に書き込む．

4.　評価

4.1　評価方法

3.3.1 節で説明を行ったシステム構成を表 1 で示した PC 上に実装し，評価を行った．評価は，MSR Cambridge ワークロード [8] の中から IO 集中が発生した表 2 の区間を btreplay command [*7] で再現することで行った．btreplay は 1 倍速で実行し，write を行う-W option も付加した．

この評価環境上で，OTF-AST，static tiering，FACEBOOK Flashcache の 3 方式の比較を行った．各方式の実行条件は表 3 に整理した．static tiering では表 2 の直前のトレースデータを分析し，IO 数順に 24subLUN までを SSD に事前配置した．例えば src1_1 ならば，360-480 分の区間を分析し，IO 数が多い 24subLUN を SSD に事前配置する．24subLUN は，OTF-AST を実行したところ，最大 24subLUN までが同時に SSD 配置となったためである．図 6 は SSD up となった subLUN 数の推移であり，表 2 の両ワークロード共に最大 24subLUN が SSD up となったこ

[*6] SSD/HDD の最小アクセス単位
[*7] MSR Cambridge は event tracing for Windows 形式で記録されているので，blktrace への変換を行った

表 1　評価システムの概要

PC	FUJITSU PRIMERGY TX300S7 - intel Xeon E5-2650L x2 - 32GB memory
HDD	HBA 4disk RAID0 (SAS,10,000rpm, 450GB) x4
SSD	intel 520(MLC, 240GB)
OS	CentOS 5.4(64bits)
subLUN 移動時間 (user IO なし)	HDD から SSD: 2-3 秒 SSD から HDD: 6-7 秒

表 2　評価に用いた MSR ワークロード

ワークロード	IO 集中の発生箇所（分） (トレース先頭からの経過時間)
src1_0(293GB)	1920-2020
src1_1(293GB)	480-600

表 3　各方式の実行条件

方式	実行条件
OTF-AST	t=60,i=50,n=20,m=60,c=3,o=10
static tiering	btreplay 区間直前の 100 or 120 分間から IO 数が多い順に 24 subLUN を SSD に事前配置
FlashCache	block size=4KB,cache size=24GB, LRU

とが分かる．24subLUN は全容量 (293GB) の約 8%に相当する容量である．事前配置の方法は，図 5 の tiering table に直接書き込むことで行った．FACEBOOK FlashCache も同様な理由で cache size=24GB で初期化し，アルゴリズムは default の LRU を選択した．FlashCache は連続して 2 回実行し，両方のデータを掲載する．これは cold start 時と cache block が満たされた時の結果を比較するためである．他方式との比較は，実際の利用状況に近い 2 回目の結果を中心に行う．

評価の観点は，各方式の SSD アクセス率と user IO レスポンスを比較することで行う．SSD アクセス率比較に必要なデータに関して，OTF-AST 及び static tiering は tiering driver 内アクセス統計情報を用いた．FACEBOOK FlashCache は，FlashCache 内アクセス統計情報を用いた．user IO レスポンスは，tiering driver 内の 10us-1second の範囲でレスポンスヒストグラムを取得する機能を用いて測定した．FACEBOOK FlashCache は，tiering driver の下に組み込むことで，tiering driver の機能を利用して測定を行った．

4.2　SSD アクセス率

表 4 が SSD アクセス率の実験結果である．まず，MSR src1_0 の評価結果に関して議論する．OTF-AST の SSD アクセス率は 24%である．さらに，3.3.2 節で説明した temporary buffer のアクセス率 5%である．temporary buffer へのアクセスは，temporary buffer に書き込むと user 側に

表 4　SSD アクセス率 (%)

	src1_0	src1_1
OTF-AST (temp. buf hits)	24 5	46 0
static tiering	1	1
FlashCache(1 回目) (dirty write hits)	7 0	37 0
FlashCache(2 回目) (dirty write hits)	19 5	37 0

書き込み完了を返しており，SSD アクセスより高速である．この temporary buffer へのアクセス率まで含めると，29%の IO が SSD アクセス相当となったと言える．static tiering の SSD アクセス率は 1%であり，直前のワークロード分析結果を用いて SSD への事前配置を行っても，IO 集中が起きる subLUN を的確に捉えられないことが分かる．FlashCache に関しては，1 回目 7%，2 回目 19%といずれも OTF-AST の結果を下回った．特に 2 回目は dirty write hits の割合が 5%に上昇しており，writeback がいずれ発生することが分かる．MSR src1_0 の read 比は約 11%であり，write 中心のワークロードである．

次に MSR src1_1 の評価結果に関して議論する．OTF-AST の SSD アクセス率は 46%となったが，temporary buffer のアクセス率は 0%であった．static tiering の SSD アクセス率は MSR src1_0 と同じく 1%であった．FlashCache の SSD アクセス率は，1 回目，2 回目共に 37%であり，いずれも OTF-AST の結果を下回った．MSR src1_0 とは異なり，dirty write hits はほとんど発生しなかった．MSR src1_1 の read 比は約 99%であり，ほぼ read only のワークロードである．

図 6 は，OTF-AST の SSD に up した subLUN 数の推移である．src1_1 は 20-80 分の間で 15subLUN 以上が使われており，46%の SSD アクセス率に繋がった．一方 src1_0 は，40-70 分の subLUN 数が伸びず，src1_1 ほど SSD アクセス率が伸びなかったと考えられる．24subLUN は全容量の約 8%相当である．これは，3.2.2.3 節で説明した HDD に書き戻す timeout 値を大きめの 10(表 3) としたことも影響し，全容量の数%より若干大きくなった．timeout 値を大きくとると，subLUN への IO 集中が終わった後に続く IO まで SSD で捉える効果がある．

MSR src1_0/src1_1 の両方のワークロードで OTF-AST の SSD アクセス率が最もよい結果となり，3.2 節で説明したアルゴリズムの有効性を示すことが出来た．また，MSR src1_0 では，temporary buffer のアクセス率が 5%となり，3.3.2 節で説明した階層移動時のオーバーヘッド削減のための実装が機能していることを確認した．

4.3　user IO レスポンス

表 5 に user IO の平均レスポンスをまとめた．

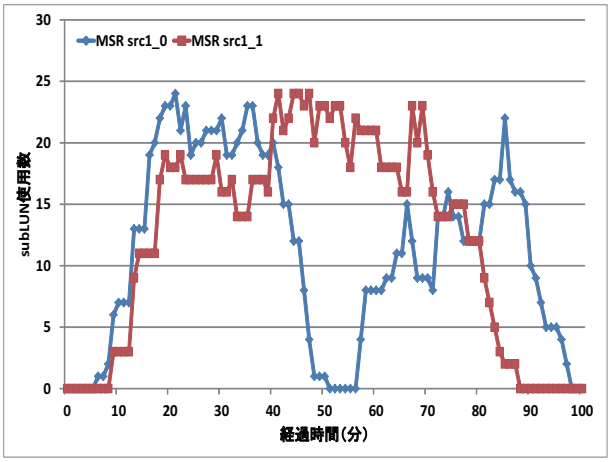

図 6　SSD up した subLUN 数の推移

表 5　user IO の平均レスポンス (ms)

	src1_0	src1_1	src1_1 (-100ms まで)
OTF-AST	45.4	56.8	9.9
OTF-AST (HDD 移動無し)	38.2	15.3	3.7
OTF-AST (0-20 分)	34.2	—	—
static tiering	50.8	7.1	6.6
FlashCache(1 回目)	33.6	6.9	6.4
FlashCache(2 回目)	36.6	6.9	6.4

まず，src1_0 に関して議論を行う．最初に static tiering との比較と行う．static tiering の平均レスポンス 50.8ms に対して OTF-AST は 45.4ms となっており，static tiering 比で 5.4 ms 平均レスポンスを向上出来たことが分かる．

次に FlashCache(2 回目) との比較を行う．FlashCache(2 回目) の平均レスポンスは 36.6ms となっており，OTF-AST より 8.8ms よいことが分かる．図 7 は取得したデータを CDF [*8]に整理した結果である．この結果を見ると，80%タイル値付近で Flash Cache に逆転されることが分かる．そこでベンチマーク実行時に採取した iostat log [*9]を分析すると，HDD がベンチマーク起動 20 分後から過負荷[*10]となるタイミングが度々起きることが分かった．そこで起動から 20 分間までのデータを CDF 化してみることにした．さらに表 1 より，SSD から HDD へ戻すコストが SSD に上げるコストの倍以上であることが分かり，SSD から HDD へ戻す処理を skip した評価も行うことにした．表 5 より，最初の 20 分間の平均レスポンス 34.2ms となり，FlashCache(2 回目) を上回ることが分かる．SSD から HDD へ戻す処理を skip した OTF-AST の平均レスポンスは 38.2ms となり，1.6ms ほど FlashCache(2 回目) の結果を上回るが，オリジナルの OTF-AST より大幅に改善したことが分かる．実験結果を CDF 化した結果が図 8 である．この結果より，最初の 20 分間の CDF は FlashCache(2 回目) を若干上回っていることが分かる．SSD から HDD へ戻す処理を skip した OTF-AST の CDF もほぼ FlashCache(2 回目) と一致することが分かる．この実験結果より，user IO と階層移動に伴う IO をあわせても HDD が過負荷にならない範囲で運用出来れば FlashCache(2 回目) より平均レスポンスを向上出来ることが分かる．しかし，HDD が過負荷となってしまうと，3.3.2 節で説明した階層移動時のオーバーヘッド削減のための実装だけではまだ不十分であることも分かった．user IO の増加で HDD が過負荷となりそうな時は，OTF-AST による subLUN 移動を一時的に絞るなどの対策が考えられ，今後の課題としたい．

次に src1_1 関して議論を行う．表 5 より，static tiering，FlashCache(2 回目) の平均レスポンスが 7ms 前後であるのに対して，OTF-AST は 56.8ms と大幅に悪化することが分かる．～100ms までのサンプリング値で平均レスポンスを求めてみると，OTF-AST のレスポンスは 9.9ms へと大幅に短縮となることも分かる．src1_1 は read 比 99%であり，SSD から HDD への書き戻しはキャンセル可能である．そこで SSD から HDD へ戻す処理を skip した OTF-AST で評価を行ったところ，15.3ms となり OTF-AST オリジナル比で 41.5ms 削減となった．～100ms までに限れば，3.7ms となり，static tiering や FlashCache(2 回目) の平均レスポンスを大きく上回ることも分かる．実験結果を CDF 化した結果が図 9 である．この結果より，SSD から HDD へ戻す処理を skip した OTF-AST では，95%タイル値前後までは FlashCache(2 回目) と比較しても提案方式が優位であることが分かる．OTF-AST の subLUN size は 1GB であり，FlashCache は 4KB である．OTF-AST で SSD から HDD へ戻す処理を skip してしまうと，実質的に cache 方式と変わらなくなる．そのため，1GB のほぼ全てに user IO が発生するケースでは SSD へのデータ移動が速く完了する OTF-AST が有利となる．ただし，SSD up となった 1GB の subLUN の一部にしか user IO が発生しないケースも考えられ，その場合は FlashCache の方が効果的なのかもしれない．これが 95%タイル値以降に FlashCache に逆転されてしまう原因となっている可能性もある．ここまでの議論により，データ更新がない subLUN の HDD への書き戻しを skip する機能を実装し，95%タイル値以降のオーバーヘッド対策が出来れば FlashCache(2 回目) の性能を上回れることが分かった．オーバーヘッド対策は，95%タイル値以降となる 20ms 以上の user IO が発生する原因分析を行った上で検討を行っていきたい．

5. まとめ

多くのファイル共有ストレージワークロードには，全容量の数%以下の領域に数分から数十分間 IO が集中し，時間と共に別の領域に移動する特徴がある．この様なワーク

*8　cummulative distribution function, 累積分布関数
*9　20 秒間隔でベンチマークが終了するまで採取
*10　%util が 100%に達する

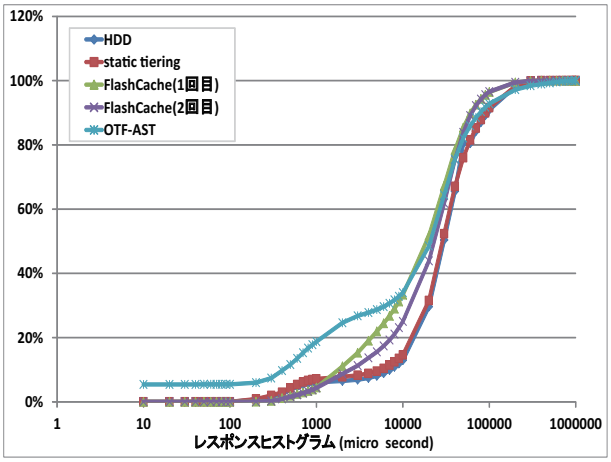

図 7　MSR src1_0 のレスポンス (1)

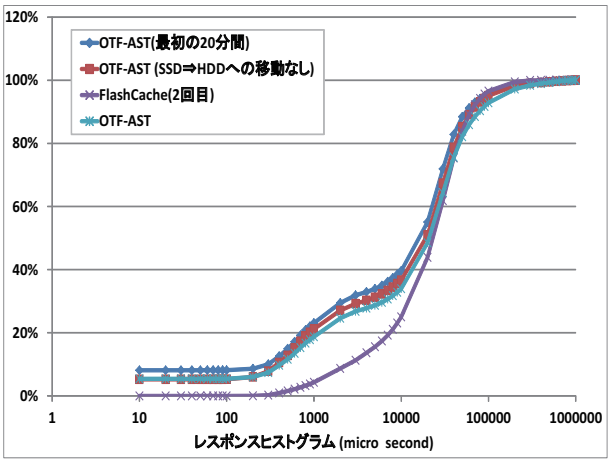

図 8　MSR src1_0 のレスポンス (2)

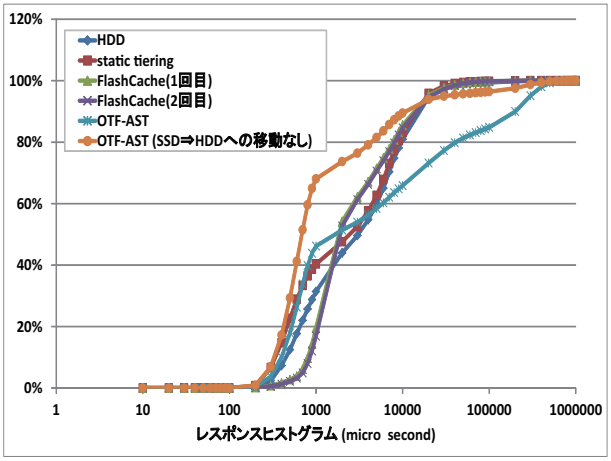

図 9　MSR src1_1 のレスポンス

ロードを SSD と HDD を用いた static tiering や cache 方式などの従来の階層ストレージシステムに適用しても，IO 集中を効果的に SSD に集めることは難しい．

そこで我々は，短い時間での IO 集中が発生した領域をその都度捉えて SSD に移動する *On-The-Fly - Automated Storage Tiering (OTF-AST)* の提案を行った．提案手法では，分単位の統計情報を用い，性能向上効果が期待出来る領域を IO の集中度とその継続時間よりその都度 filtering することで SSD に IO を集中させること，に成功した．

性能評価は，IO 集中をどれだけ SSD に集めることが出来たのか (SSD アクセス率) と階層移動に伴う遅延まで含めた user IO レスポンスの観点で行った．比較のため，従来の階層ストレージシステムである static tiering と FACEBOOK FlashCache[1] も同一条件で評価を行った．ベンチマークは，MSR Cambridge ワークロード [8] の中から，IO 集中が発生したケースを選んで replay した．SSD アクセス率は最大で 46%となり，static tiering の 1%や FlashCache の 37%を上回ることを確認した．

user IO のレスポンスは，write 中心ワークロードと read 中心ワークロードで結果が分かれた．write 中心ワークロードの user IO レスポンスは，階層移動に伴う IO と user IO で HDD が過負荷にならないケースで効果確認が出来た．提案方式の平均 user IO レスポンスが 34.2ms に対して，static tiering が 50.8ms，FlashCache は 36.6ms であった．HDD 過負荷時の対策としては，HDD 過負荷時のみ階層移動を絞るなどが考えられるが，詳細は今後の課題としたい．read 中心ワークロードは，現状の実装のままでは static tiering，FlashCache からの優位性は主張出来ず，更新のない領域の HDD への書き戻しをキャンセルする実装を追加し，20ms 以上となる全 IO の 5%の IO の遅延を小さくできれば，static tiering 及び FlashCache より優位な性能を得られることが分かった．

6. 今後の予定

- write 中心ワークロードにおける HDD 過負荷時のフロー制御
- SSD up 時に書き込みがなかった領域の HDD への書き戻しをキャンセルする機能の実装と評価
- 本稿の評価に用いなかったワークロードでの実験

参考文献

[1] https://github.com/facebook/flashcache
[2] http://www.fusionio.com/data-sheets/directcache/
[3] EMC White Paper, EMC FAST VP for Unified Storage Systems A Detailed Review, March 2011
[4] FUJITSU Automated Storage Tiering: available from (http://storage-system.fujitsu.com/jp/products/diskarray/feature/i03/).
[5] F.Chen, D.A. Koufaty, and X. Zhang, 'Hystor: Making the Best Use of Solid State Drivers in High Performance Storage Systems,' In *Proc. of International Conference on Supercomputing* (2011), ACM.
[6] 大江和一，荻原一隆，野口泰生，小沢年弘，spike 領域をリアルタイムに高速ストレージに移動することが可能な階層ストレージシステムの提案．情報処理学会論文誌コンピューティングシステム　Vol.5 No.5 118-127 (Oct. 2012)
[7] 大江和一，本田岳夫，河場基行．階層ストレージ方式検討に向けた商用 Samba ワークロード分析と考察．2012 年

度 OS12 月研究会.

[8] Dushyanth Narayanan, Austin Donnelly, and Antony Rowstron, Write Off-Loading: Practical Power Management for Enterprise Storage. In *Proc. 6th USENIX Conference on File and Storage Technologies* (FAST 2008), USENIX.

[9] Jorge Guerra, Himabindu Pucha, Joseph Glider, Wendy Belluomini, and Raju Rangawami, Cost Effective Storage using Extent Based Dynamic Tiering. In *Proc. 9th USENIX Conference on File and Storage Technologies* (FAST 2011), USENIX.

[10] Raja Appuswamy, David C.van Moolenbroek, and Andrew S. Tanenbaum, Integrating Flash-based SSDs into the Storage Stack. In *Proc. 28th IEEE Storage Conference on Massive Data Storage* (MSST 2012), IEEE.

[11] Xiaojian Wu, A.L. Narasimha Reddy, Exploiting concurrency to improve latency and throughput in a hybrid storage system. In *Proc. 18th Annual Meeting of the IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS2010), IEEE.

[12] Yiying Zhang, Gokul Soundararajan, Mark W. Storer, Lakshmi N. Bairavasundaram, Sethuraman Subbiah, Andrea C.Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, Warming up Storage-Level Caches with Bonfire. In *Proc. 11th USENIX Conference on File and Storage Technologies* (FAST 2013), USENIX.