不規則なアクセス負荷集中に適応可能な高性能ストレージの研究

大江, 和一

https://doi.org/10.15017/1654912

不規則なアクセス負荷集中に適応可能な
高性能ストレージの研究

大江　和一

2016 年 2 月

要旨

近年のストレージシステムは，SSD の登場により大幅な性能向上を達成した．しかしながら，SSD は HDD と比較して高価であり，この高価な傾向が少なくとも 2020 年位まで続くと考えられる．そのため，SSD と HDD を組み合わせた階層型ストレージシステムは，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重要な技術である．階層型ストレージシステムでは，高速な SSD と低速な HDD による物理的な階層構造を，一次元のストレージ空間としてアプリケーションに提供することが出来，さらに SSD の容量の比率を少なくしその SSD に IO アクセスの頻度が高い領域を優先的に割り当てることで，コストパフォーマンス向上を図ることが出来る．一方，IO アクセスの頻度が高い領域をどうやって検出するのか，という課題も生じる．

IO アクセスの頻度が高い領域の特徴を把握する目的で，本研究では運用中に採取した複数の共用ファイルサーバワークロード（Samba ワークロードと MSR Cambridge ワークロード）の分析を行った．特に，IO アクセスが集中した領域に注目しその領域の空間的・時間的な偏りを明らかにすることと、その成果を階層型ストレージシステムの制御に反映することが，分析の目的である．その結果，IO アクセスが集中した領域は全領域の高々数%程度であり，この範囲に全 IO アクセスの 58%以上が集まっていた．この IO アクセス集中の 66%以上は任意の領域に発生し，IO アクセス集中の継続時間が平均 60 分であった．さらに本研究では，IO アクセス集中に達しなかった非 IO アクセス集中領域の分析も行い，全領域の 13-99%を占め，この範囲に全 IO アクセスの最大で 40%が集まることを発見した．

階層型ストレージシステムとして従来用いられている Conventional tiering と Caching は，この IO アクセス集中を適切に取り扱うことが出来ない．Conventional tiering は，数時間から１日単位の IO アクセス情報を用いてデータの再配置を行う方式であるため，平均 60 分程度しか継続せず任意の領域に発生する IO アクセス集中が発生する領域を捉えて SSD に配置することが困難である．一方 Caching は，read と write が混在した IO アクセス集中において，write-back 処理が原因となる HDD への大量の random アクセスを引き起こすため，IO アク

セス集中を含んだワークロードを性能向上出来ない.

そこで本研究では，IO アクセス集中及び IO アクセス集中に達しなかった非 IO アクセス集中を含むワークロードの性能向上に取り組んだ．その結果，(1) IO アクセス集中領域を検出して SSD に置換する階層ストレージの提案，(2) 参照局所性と領域移動を考慮して高速に IO アクセス集中領域を検出する階層ストレージ OTF-AST の提案，(3) caching 技術と IO アクセス集中領域検出技術による階層ストレージ OTF-AST with caching の提案，という研究成果を上げた.

(1) では，HDD の過負荷が原因でアプリケーションが求める性能を提供出来なくなった時に HDD の iops と IO busy 率の相関係数を用いて，HDD の過負荷状態を解消するために HDD より IOPS が 2 桁以上大きい SSD へ移動が必要な領域を求め，即座にその領域を移動することで，IO アクセス集中を含むワークロードの性能向上に，成功した．しかしながら，HDD が過負荷状態となりその状態がしばらく続かないと SSD への移動が始まらないため，本提案の効果が得られるまで時間がかかることと，隣接領域に移動しながら IO アクセス集中が継続するパターンで，本提案の効果が薄いことが，課題である.

(2) では，ストレージの部分領域ごとの IO アクセス数の変化を常時監視することで，IO アクセス集中を迅速に把握し，SSD 置換後も IO アクセス集中が継続する領域に絞って SSD に置換することで，(1) で性能向上が難しかった IO アクセス集中を含むワークロードの性能向上に成功した．隣接領域に移動しながら IO アクセス集中が継続するパターンに関して，本提案は，IO アクセス集中の移動速度と IO アクセス集中となるまでの成長速度を用いて，IO アクセスがまだ少ない時に SSD に置換する方法を用いた．この (2) の提案は，IO アクセス集中領域の性能向上問題を解決した.

(3) では，(2) で性能向上出来なかった短時間で終息する IO アクセス集中領域と非 IO アクセス集中領域に対して，Caching と (2) の提案を組み合わせる方法の提案である．本提案を効果的に機能させるには，(2) で用いる SSD と Caching の間の領域置換の高速化が必須である．そこで本提案では，Caching を構成する HDD と SSD から置換に必要な領域を別々に取り出して (2) で用いる SSD に置換することで，HDD のランダムアクセスを回避し，IO アクセス

集中領域と非 IO アクセス領域の両方に発生した IO アクセスの性能向上を達成した．本提案は，同じ条件で動作させた従来方式より優位であることも確認した．

謝辞

本研究を遂行する上で，終始懇切丁寧成る御指導とご鞭撻，格別の御配慮を賜わりました九州大学大学院システム情報科学府情報知能工学専攻　岡村　耕二教授，南里　豪志准教授に深く感謝申し上げます．

本研究をまとめるにあたり，貴重なお時間を割いていただき，丁寧なるご教授を賜わりました九州大学大学院システム情報科学府Ｉ＆Ｅビジョナリー特別部門　井上　弘士教授，九州大学大学院システム情報科学府情報知能工学専攻　天野　浩文准教授に深く感謝申し上げます．

筆者が九州大学大学院システム情報科学府情報知能工学専攻博士後期課程に在学することへの御配慮と援助を賜わりました（株）富士通研究所　コンピュータシステム研究所　データシステムＰＪ　赤星プロジェクトディレクター，塩沢主管研究員，佐藤主任研究員，メディアサーバ PJ 小沢プロジェクトディレクターに感謝申し上げます．

また本研究に関し，日々様々なご討論ご助言を頂くと共に多大なるご支援を頂きました（株）富士通研究所　コンピュータシステム研究所　データシステムＰＪ　岩田　聡研究員,（株）富士通ソフトウエアテクノロジース　本田　岳夫氏に心から御礼申し上げます．ならびに格別なる御指導と御配慮を賜わりました（株）富士通研究所　コンピュータシステム研究所　データシステムＰＪの研究員各位に心から御礼申し上げます．

目 次

表 目 次

図 目 次

第1章　序論

1.1　背景

近年のストレージシステムは，SSD の登場により大幅な性能向上を達成した．しかしながら，SSD は HDD と比較して高価であり，この高価な傾向が少なくとも 2020 年位まで続くと考えられる．そのため，SSD と HDD を組み合わせた階層型ストレージシステムは，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重要な技術である．階層型ストレージシステムでは，高速な SSD と低速な HDD による物理的な階層構造を，一次元のストレージ空間としてアプリケーションに提供することが出来，さらに SSD の容量の比率を少なくしその SSD に IO アクセスの頻度が高い領域を優先的に割り当てることで，コストパフォーマンスの向上を図ることが出来る．一方，IO アクセスの頻度が高い領域をどうやって検出するのか，という課題も生じる．

IO アクセスの頻度が高い領域の特徴を把握する目的で，本研究では運用中に採取した Samba ワークロードと MSR Cambridge ワークロードの分析を行った．特に，IO アクセスが集中した領域に注目しその領域の空間的・時間的な偏りを明らかにすることと，その成果を階層型ストレージシステムの制御に反映することが，本研究の分析の目的である．その結果，IO アクセスが集中した領域は全領域の高々数%であり，この範囲に全 IO アクセスの 58%以上が集まっていた．この IO アクセス集中の 66%は任意の領域に発生し，IO アクセス集中の継続時間が平均 60 分であることも分かった．さらに本研究は，IO アクセス集中とならなかった領域の分析も行い，IO アクセス集中領域より IO アクセス数が少ないがまとまった IO アクセス数が発生する中位 IO アクセス領域と，その中位 IO アクセス領域より一桁以上 IO アクセス数が少ない下位 IO アクセス領域を発見した．中位 IO アクセス領域も全領域の高々数%であり，この範囲

に全 IO アクセスの最大で 22%が集まっていた．下位 IO アクセス領域は，全領域の 13–99%を占め，この範囲に全 IO アクセスの最大で 20%が集まっていた．この分析結果は，IO アクセス集中領域を検出して SSD に集めることがコストパフォーマンスを向上させる上で極めて重要であることを示している．

階層型ストレージシステムとして従来用いられている Conventional tiering と Caching は，この IO アクセス集中を適切に取り扱うことができない．Conventional tiering は，数時間から 1 日単位の IO アクセス情報を用いてデータの再配置を行う方式であるため，平均 60 分程度しか継続せず任意の領域に発生する IO アクセス集中が発生する領域を捉えて SSD に配置することが困難である．一方 Caching は，read と write が混在した IO アクセス集中において，write-back 処理が原因となる HDD への大量の random アクセスを引き起こすため，IO アクセス集中を含んだワークロードを性能向上できない．

そこで本研究の目的は，分析を行ったワークロードで検出した IO アクセス集中が発生した領域を主なターゲットに，効果的な階層ストレージシステムを構築することである．

1.2 主結果

本研究の貢献は次の 3 つから構成される．(1) IO アクセス集中領域を検出して SSD に置換する階層ストレージの提案 [15]，(2) 参照局所性と領域移動を考慮して高速に IO アクセス集中領域を検出する階層ストレージ OTF-AST の提案 [16] [29]，(3) caching 技術と IO アクセス集中領域検出技術による階層ストレージ OTF-AST with caching の提案 [30] [31]，である．

(1) では，HDD の過負荷が原因でアプリケーションが求める性能が提供出来なくなった時に HDD の iops と IO busy 率の相関係数を用いて、HDD の過負荷状態を解消するために HDD より IOPS が 2 桁程度大きい SSD へ置換が必要な領域を求め、即座にその領域を置換することで、IO アクセス集中を含むワークロードの性能向上に，本研究では成功した．しかしながら，HDD が過負荷状態となりその状態がしばらく続かないと SSD への置換が始まらないため効果が得られるまで時間がかかることと，隣接領域に移動しながら IO アクセス集中が継続するパターンで効果が薄いことが，本研究の課題である．

(2) では，ストレージの部分領域ごとの IO アクセス数の変化を常時監視することで，IO アクセス集中を迅速に把握し，SSD 置換後も IO アクセス集中が継続する領域に絞って SSD に置換を行うことで，(1) で性能向上が難しかった IO アクセス集中を含むワークロードの性能向上に本研究では成功した．隣接領域に移動しながら IO アクセス集中が継続するパターンに関しては，本研究において IO アクセス集中の移動速度と IO アクセス集中となるまでの成長速度を用いて IO アクセス集中が移動する領域を予測し，IO アクセスがまだ少ない時に SSD に置換する方法を提案した．

(3) では，(2) で性能向上出来なかった短時間で IO アクセス集中が終息する領域と IO アクセス集中に達しなかった中位 IO アクセス領域と下位 IO アクセス領域に対して，Caching を併用することで (2) のケースより性能向上する提案を行った．本研究で性能向上するには，(2) で用いる SSD と Caching の間の領域置換の高速化が必須である．そこで本研究では，caching の HDD と SSD から置換に必要な領域を別々に取り出すことで HDD のランダムアクセスを回避し，IO アクセス集中と IO アクセス集中に達しなかった領域を含むワークロードの性能向上が可能となった．

1.3 用語の定義

本論文で使用する用語の定義を行う．まず，OS から認識できるストレージの論理的な単位を Logical Unit Number (LUN) と定義する．LUN を 1 GB 前後の適当な大きさで区切った部分を sub-LUN と定義する．sub-LUN は複数の block から構成されており，block の大きさは通常 512 bytes である．この各 block ごとに Logical Block Address (LBA) が割り当てられ，LUN の各部分への IO アクセスはこの LBA を指定することで行われる．また，本論文でのワークロードとは，時間経過ごとに各 block に発生する IO アクセスのことである．

1.4 本論文の構成

次章以降，本論文は次のように構成されている．まず第 2 章でストレージワークロードの分析結果に関して説明を行う．第 3 章は，関連研究に関して説明を行う．次に，第 4 章は，IO アクセス集中領域を検出して SSD に置換する階層ストレージの提案に関して説明を行い，第 5 章は，参照局所性と領域移動を考慮して高速に IO アクセス集中領域を検出する階層ストレージ OTF-AST に関して説明する．そして，第 6 章は，Caching 技術と IO アクセス集中領域検出技術による階層ストレージ OTF-AST with caching の提案に関して説明を行い，最後に第 7 章で結論を述べる．

第2章　ストレージワークロードの分析結果

2.1　背景

近年のストレージシステムは，SSD の登場により大幅な性能向上を達成した．しかしながら，SSD は HDD と比較して高価であり，この高価な傾向が少なくとも 2020 年位まで続くと考えられる [25] [26]．そのため，SSD と HDD を組み合わせた階層型ストレージシステムは，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重要な技術である．階層型ストレージシステムでは，高速な SSD と低速な HDD による物理的な階層構造を，一次元のストレージ空間としてアプリケーションに提供することが出来，さらに SSD の容量の比率を少なくしその SSD に IO アクセスの頻度が高い領域を優先的に割り当てることで，コストパフォーマンス向上を図ることが出来る．一方，IO アクセスの頻度が高い領域をどうやって検出するのか，という課題も生じる．

計算機上で実行されるアプリケーションの大部分は，ファイルシステムを通して IO アクセスを行う．そのため，その IO アクセス性能は，ファイルシステムのストレージ上のフォーマット方法の影響を大きく受ける．一般にファイルシステムのストレージ上のフォーマットは，メタデータと呼ばれる管理領域とユーザデータを格納するデータ領域に分離される．メタデータの領域は予め決められた狭い領域に割り当てられることが多く，同じ領域へ繰り返し IO アクセスが発生する．そのため，メタデータ領域の IO アクセスが増加してもメタデータ領域が割り当てられた狭いストレージ領域を SSD 移動することで IO アクセス性能を向上させることが出来る．一方，データ領域への IO アクセスは，ファイルシステムのデータ領域割り当てアルゴリズムやアプリケーションの IO アクセスのパターンの応じて，IO アクセスが発生するストレージ領域が変動するので，メタデータのように単純にアクセス頻度が高い部分を抽出する

ことが困難である．

そこで本研究では，主にファイルシステムのデータ領域に発生するIOアクセスの特徴を把握する目的で，運用中に採取したストレージワークロードの分析を行うことにした．特に，IOアクセスが集中した領域に注目しその領域の空間的・時間的な偏りを明らかにすることと，その成果を階層ストレージシステムの制御に反映することが，分析の目的である．

2.2 分析に用いたワークロードデータ

本研究は，商用環境で運用したSambaワークロード [5] から採取したIOアクセス情報と公開トレースデータであるMSR Cambridgeワークロード [1][2] を用いて，分析を実施した．

2.2.1 Sambaワークロードの概要

Sambaワークロード [5] は，約3,000 userが常時利用したSambaファイルサーバ 6ヶ月分の分析結果である．ボリュームサイズは約4.4 TBであり，蓄積されている元データは4.4 TBを1 GB sub-LUNで分割し，このsub-LUN単位で1分間隔で採取したIOアクセス情報である．採取したIOアクセス情報には，IO数，RW比，IO size分布，そして応答時間ヒストグラム情報が含まれる．ファイルシステムは，Linux上で動作するVeritas VxFSである．

2.2.2 MSR Cambridgeワークロードの概要

MSR Cambridgeワークロード [1][2] は，ケンブリッジにあるMicrosoft Researchの運用サーバトレースデータである．運用サーバは，ファイルサーバ，プリントサーバ，Web/SQLサーバなどが含まれる．トレースデータは，Event Tracing for Windows (ETW)[23] 形式で36 workload分のデータが約1週間分格納されている．本研究はこの中からproj1, proj2, proj4, src1_0, src1_1, usr1, usr2, web2の8ワークロードを選択した．この8ワークロードはいずれもIOアクセス集中が数時間継続するワークロードであり，ボリュームの大きさは170-821 GBの

範囲である．web2 以外は全てファイルサーバである．なお，ファイルシステムは，Windows Server 2003 SP2 上で動作する NTFS である．

2.3 ストレージワークロードデータの分析方法

本研究の分析の目的は，IO アクセスが集中した領域の空間的・時間的な偏りを明らかにすることである．そこで本研究は，分析を行う Samba と MSR Cambridge トレースデータを 1 GB の sub-LUN に分割し，さらに各 sub-LUN ごとに 1 分間の単位で IO アクセス数を集計し，IO アクセスの偏り分析を行った．この *1-GB-1-min* 単位に分割した理由は，2.2.1 節で説明した Samba ワークロードが *1-GB-1-min* 単位の粒度で情報蓄積が行われたためである．さらに，本研究は *1-GB-1-min* の範囲に発生した IO アクセスを IOPS に変換して分析した．

この分析を行う前に，本研究は Samba ワークロードと MSR Cambridge ワークロードに発生した IO アクセス数の調査を行い，10 IOPS 以上となる sub-LUN に多くの IO アクセスが集まることを把握した．そこで本研究は，この 10 IOPS 以上となった sub-LUN を IO アクセス集中領域と判断し，この領域に関して空間的・時間的な偏りの観点で分析した．さらに本研究は，10〜1 IOPS となった sub-LUN を中位 IO アクセス領域，1 IOPS 未満となった sub-LUN を下位 IO アクセス領域とし，これら領域の分析を行い，IO アクセス集中領域の分析結果と比較した．

2.4 ストレージワークロードデータの分析結果とその考察

2.4.1 IO アクセス集中領域

図 2.1 は，本分析で IO アクセス集中領域として定義した，1 分間の IOPS が 10 IOPS 以上となった sub-LUN の解析結果である．図 2.1 の結果は，平均で 3-6 sub-LUN の範囲に全 IO の 58% 以上が集まることを示している．Samba を例に取ると，全体で 4.4 TB の容量があるので，全容量の 1%にも満たない sub-LUN が全 IO アクセスの半分以上を含むことが分かる．さらに，この sub-LUN の少なくとも 71%は，任意の sub-LUN であることが分かる．MSR Cambridge

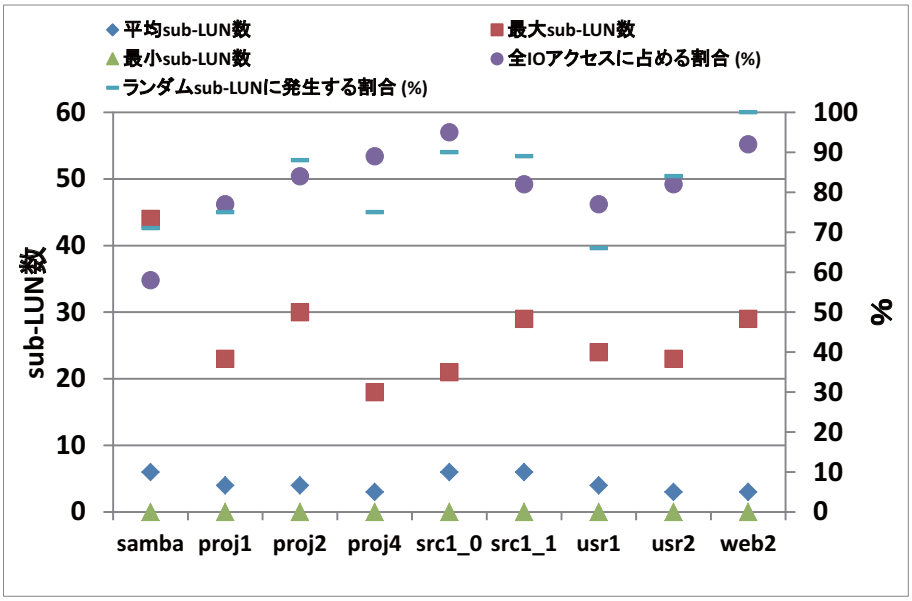

図 2.1: 10 IOPS 以上となった sub-LUN の解析結果.

表 2.1: MSR Cambridge ワークロードの IO アクセス集中継続時間.

ワークロード	proj1	proj2	proj4	src1_0	src1_1	usr1	usr2	web2
42 IOPS 以上の継続時間 (分)	60	54	40	87	66	86	32	58
42 IOPS 以上の平均 IOPS	345	322	325	510	989	238	355	165
42 IOPS 未満の継続時間 (分)	475	294	1057	571	771	211	780	1045
42 IOPS 未満の平均 IOPS	4	3	0	0	4	9	3	0

ワークロードに関しても，10 IOPS 以上となる sub-LUN の割合は全容量の数%である．さらに，この 10 IOPS 以上となった sub-LUN のうち少なくとも 66%は，任意の sub-LUN であることが分かる．なお，本稿での任意の sub-LUN とは，分析に用いた全データで各 sub-LUN ごとに 10 IOPS 以上となった回数を集計し，10 IOPS 以上となった総数の 5%未満となった sub-LUN を指す．

図 2.2 は，本分析で IO アクセス集中領域として定義した，10 IOPS 以上となった sub-LUN が連続して何分継続するのかを解析した結果である．Samba では，10 IOPS 以上となる sub-LUN の 90%が，40 分までの継続時間であることが分かる．一方，MSR Cambridge の各ワークロードでは，10 IOPS 以上となる sub-LUN の継続時間は，3 分前後が支配的である．その理由は，

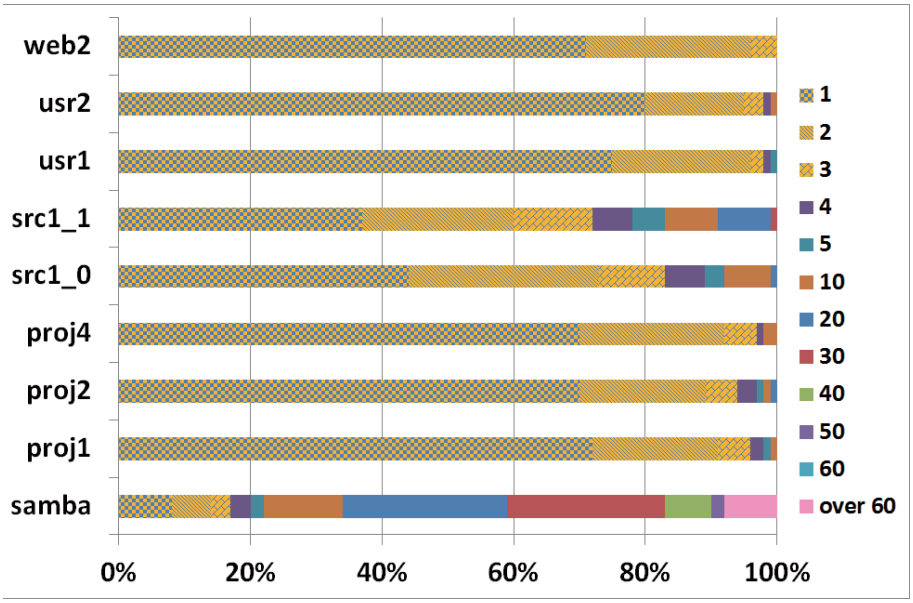

図 2.2: 10 IOPS 以上となった sub-LUN の継続時間分布 (%).

MSR Cambridge の各ワークロードは，1 つの sub-LUN に数分程度 10 IOPS 以上が継続した後，隣接した sub-LUN が 10 IOPS 以上となるためである．表 2.1 は，MSR Cambridge 各ワークロードの LUN 全体が継続して 42 IOPS 以上となった場合と 42 IOPS 未満となった場合を抽出した結果である．LUN 全体で継続して 42 IOPS となる時間は，MSR Cambridge 8 ワークロードを平均すると，60 分である．この結果より，1 つの sub-LUN の IO アクセス集中が数分継続し隣接 sub-LUN に移動しながら 60 分前後継続することが分かる．さらに，LUN 全体が 42 IOPS 未満となる時間が 8 ワークロードの平均で 651 分となることも分かる．なお，MSR Cambridge ワークロードを事前分析したところ，IO アクセス集中は，継続して発生する時間帯と発生しない時間帯に分離していた．そこで本研究は，全ワークロードデータの平均 IOPS である 42 IOPS を境に継続時間を調査した．

次にファイルシステムの観点で解析を行う．ここまでで説明した IO アクセス集中の中で，任意の sub-LUN に発生したケースがファイルシステムのデータ領域に対応し，任意の sub-LUN に発生しないケースがファイルシステムのメタデータ領域に対応すると推定される．データ領域に発生した IO アクセス数は，図 2.1 の結果より全 IO アクセス集中の少なくとも 66%となることも分かる．つまり，このデータ領域に発生する IO アクセス集中の高速化が IO アクセ

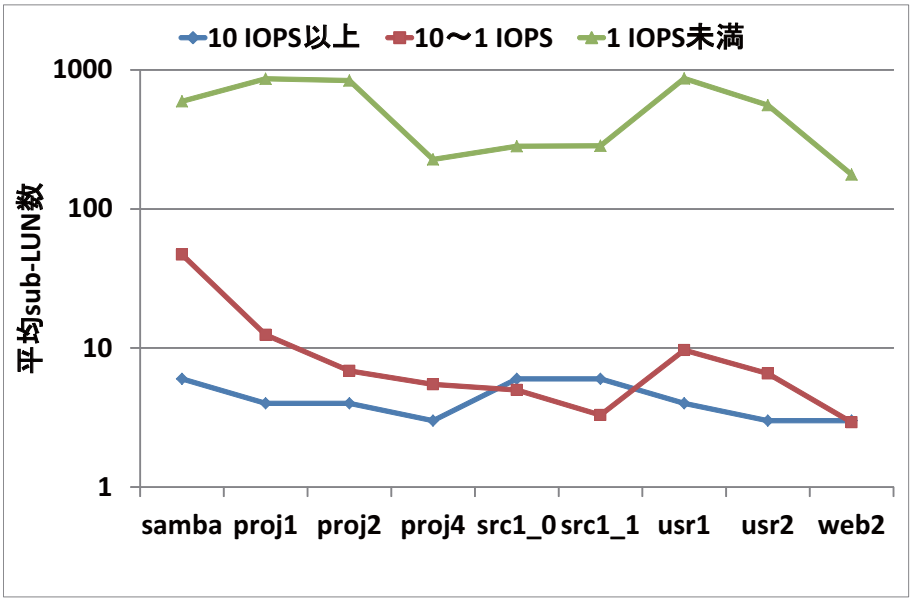

図 2.3: 平均 sub-LUN 数の推移 (10 IOPS 以上，10〜1 IOPS，1 IOPS 未満).

ス性能向上に効果的であることが分かる．

最後に，今回評価に用いた Samba ワークロードと MSR Cambridge ワークロードは，異なる OS ファイルシステム上で異なるアプリケーションを運用したファイルサーバのワークロードである．本研究は，これら 2 種類のワークロード分析において上述の様に共通する特徴 (IO アクセス集中) を抽出した．よって，本稿で解析した IO アクセス集中は，他のファイルサーバワークロードでも発生すると推定される．

2.4.2 全 IO アクセス領域

図 2.3 は，IO アクセス集中領域 (10 IOPS 以上)，中位 IO アクセス領域 (10〜1 IOPS)，下位 IO アクセス領域 (1 IOPS 未満) となった各 sub-LUN に関して，平均 sub-LUN 数の比較結果である．なお，下位 IO アクセス領域 (1 IOPS 未満) の sub-LUN は，IO アクセスが全く発生していない sub-LUN を含まない．図 2.3 は，1 IOPS 未満の sub-LUN が 1 IOPS 以上となる sub-LUN と比較して，広範囲に発生することを示している．図 2.4 は，図 2.3 の平均 sub-LUN 数を各 LUN の容量に対する割合に変換した結果である．この結果からも，下位 IO アクセス

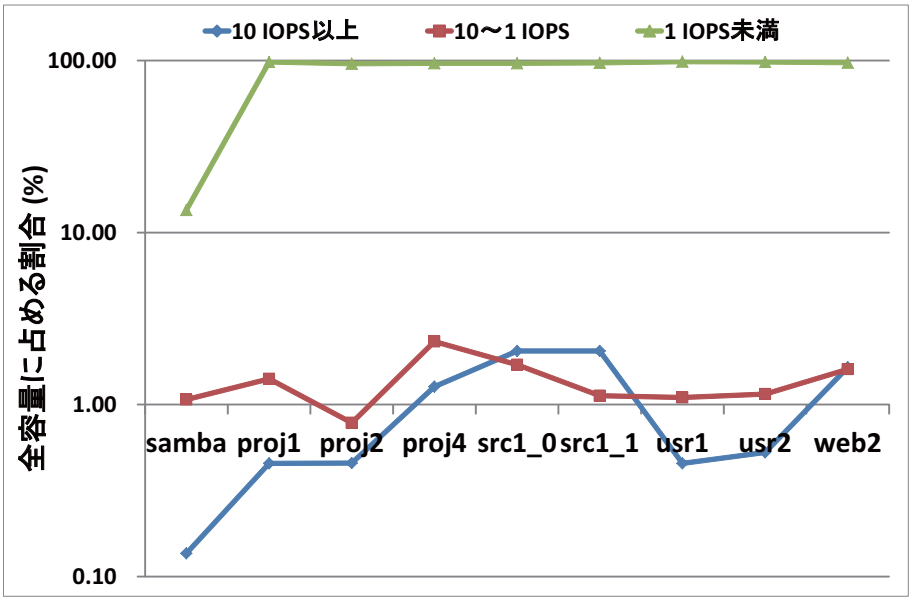

図 2.4: 全容量に占める割合 (10 IOPS 以上，10〜1 IOPS，1 IOPS 未満).

領域が，LUN の広範囲に広がっていることが分かる．

図 2.5 は，IO アクセス集中領域 (10 IOPS 以上)，中位 IO アクセス領域 (10〜1 IOPS)，下位 IO アクセス領域 (1 IOPS 未満) となった各 sub-LUN に関して，全 IO アクセスに占める割合を示している．この結果は，IO アクセス集中領域 (10 IOPS 以上となった sub-LUN) に少なくとも 58%の IO アクセスが集中していることを示している．

図 2.3，図 2.4，及び図 2.5 の結果は，IO アクセス集中領域とならなかった中位 IO アクセス領域と下位 IO アクセス領域の IO アクセス数が少なく且つ多くの sub-LUN に IO アクセスが分散していることを示している．

2.5 階層ストレージシステムを構築するための方針

2.4 節の分析結果は，最大でも全容量の数%相当の少数の sub-LUN に IO アクセス集中が発生し，その IO アクセス集中が数分から数十分間の比較的短時間で別の sub-LUN に移動すること，を示している．さらに，この IO アクセス集中が発生した sub-LUN は，全 IO アクセスの少なくとも 58%を含み，事前の予測が困難である．一方，非 IO アクセス集中は，少なくと

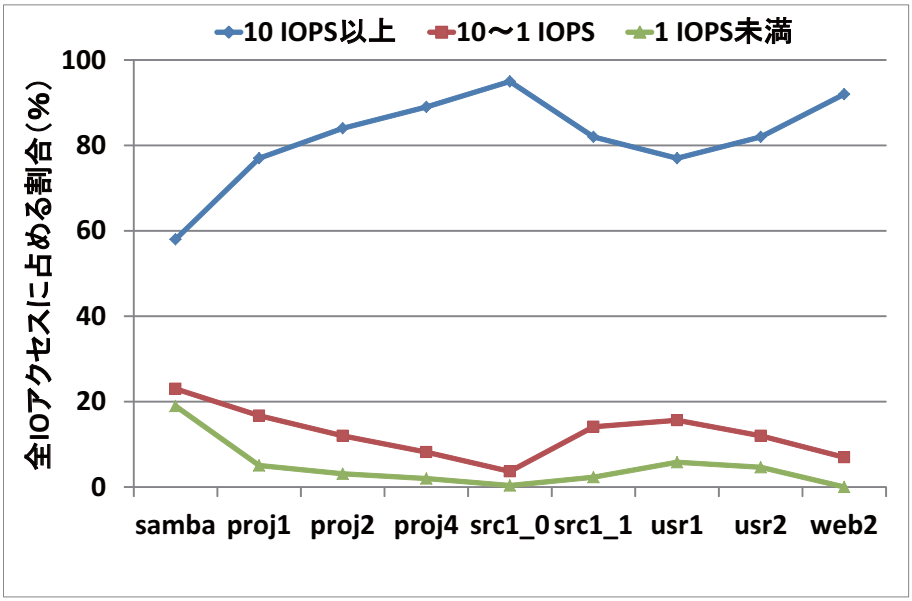

図 2.5: 全 IO アクセスに占める割合 (10 IOPS 以上，10〜1 IOPS，1 IOPS 未満).

も全容量の 13%以上に相当する多くの sub-LUN 上に発生し，最大で全 IO アクセスの 42%を含む.

このワークロードの分析結果を踏まえ，本研究ではまず，IO アクセス集中が発生した sub-LUN を効果的に SSD に置換する方法を追求する．IO アクセス集中が発生した sub-LUN は，事前の予測が困難であり，運用中にこれら sub-LUN の HDD から SSD への置換が必要になる．そこで，IO アクセス性能を低下させずに，これら sub-LUN を SSD に置換する方法の構築が研究の目的である.

次に本研究は，IO アクセス集中の性能向上に非 IO アクセス集中の性能向上を加えることで，さらに性能向上する方法を追求する．非 IO アクセス集中は，IO アクセス集中より広範囲の sub-LUN に発生し，より少ない IO アクセス数となっている．この特徴に効果的であり，IO アクセス集中で用いる方法と組み合わせての動作が可能な方法の構築が研究の目的である.

第3章　関連研究

3.1　ストレージワークロード分析

Bodik[21] らの論文は，インターネットから利用されるサーバの IO アクセス集中により発生する spike の分析とモデル化を行っている．この spike 分析結果は，全容量の 1%程度に全 IO アクセスの 32-88%が集まり，さらにその IO アクセス集中が数時間から数日継続することを示している．

Kavalaekar[22] らの研究は，Windows Server 2008 で運用した 12 台のビジネスサーバのワークロード分析結果を報告している．この中の Exchange server の報告は，IO アクセス集中が 15 分継続することを示している．しかしながら，この研究の分析方法はディスク単位で行われていたため，ディスク内の一部の領域に IO アクセスが偏っていたのかどうかは判断できない．

3.2　階層型ストレージシステム

製品でよく用いられている Conventional tiering [11][12][13] は，sub-LUN 単位で数時間から 1 日分の IO アクセス数の集計を行い，その中から IO アクセス数の多い sub-LUN 順に SSD へ配置する方法である．2 章の分析結果は，IO アクセス集中は少数の sub-LUN に短時間発生して別の sub-LUN 上に移動することを示している．この移動先の sub-LUN に規則性がないため，直前の数時間から 1 日分の IO アクセス数の集計結果で sub-LUN を SSD に置換しても，Conventional tiering は SSD 上で IO アクセス集中を捉えることが出来ない．

製品で用いられるもう 1 つの方法は，Caching [7][8][9][10] である．Caching の主な構成として，Write-through caching と Write-back caching が存在する．Write-through caching は，write 要求は全て HDD に書き込む方式である．read/write が混在する IO アクセス集中が発

生すると，大量の HDD へのランダム write が発生するため，write-through caching は性能向上しない．

一方，Write-back caching の write 要求処理は，caching 領域として割り当てた SSD 側に一旦書き込み，後で HDD 側と同期をとる方法である．この Write-back caching に対して，2 章で説明した IO アクセス集中の中で read/write が混在するケースが発生すると，大量の HDD への write-back が発生する．2 章で説明したように，IO アクセス集中は最大 60 分程度継続した後，今までとは異なった領域に移動する特徴を有する．read/write が混在したケースでは，1 つの IO アクセス集中が終了し新たな IO アクセス集中が生成されると，Write-back caching は新たな IO アクセス集中が発生した領域に対して大量の cache block を割り当てる．一方で，終了した IO アクセス集中に対応する cache block は，IO アクセスが発生しなくなる．write 要求が混在しているため，両方の IO アクセス集中に対応する cache block には相当数の dirty cache block が含まれる．よって，IO アクセス集中の入れ替わりが進む度に dirty cache block の割合が大幅に増加する．この割合があらかじめ設定された閾値を超えてしまうと，Write-back caching は大量の write-back 処理を実行する．Cache block の大きさは一般的に 4–64 KB 位の小さいサイズであるため，大量の write-back 処理は HDD への大量の random access となり，Write-back caching の IO アクセス性能は低下する．

最後に，国際会議で公開された関連研究を紹介する．

Hystor [4] は，ストレージに発生する IO アクセスを監視し，必要な時にデータを SSD に置換する提案を行っている．しかしながら Hystor は，ストレージのボリュームを時間的・空間的局所性の観点で分析を行っておらず，2 章で説明した IO アクセス集中のみを抽出できない．さらに Hystor は，IO アクセスの監視間隔が 15 分以上であるため，短時間で終息する IO アクセス集中を迅速に把握できない．このため，Hystor は 2 章の IO アクセス集中を含むワークロードを性能向上出来ない，

Cost-effective tiering [3] も，運用中に一部のストレージ領域の置換を行うことでシステムの性能向上を行う提案である．この提案システムは，IOPS + size，シーケンシャルアクセス，ランダムアクセス情報を用いて，置換を行う領域を決定する．そのため，提案システムは，ス

トレージボリュームの時間的・空間的局所性情報を用いて抽出する IO アクセス集中領域を，検出して SSD へ置換することができない．

hot random off-loading (HRO) [17] も，運用中に一部のストレージ領域の置換を行うことでシステムの性能向上を行う提案である．HRO は，ファイル単位のアクセスパターン履歴を用いて性能向上効果を動的に見積り，その結果を用いて HDD と SSD の間の置換を行う．アクセスパターン履歴は，特にランダム性と負荷集中度を注視する．しかし HRO はファイル単位で移送を行うため，LUN 内の時間的・空間的局所性分析情報を用いて抽出する IO アクセス集中を認識出来ず，SSD への置換もできない．

Raja らの論文 [14] では，Loris という彼らが開発したシステム上で caching と dynamic storage tiering の比較を行っている．その中で彼らは，Loris-based hot-DST (Dynamic Storage Tiering) システムを提案しており，このシステムはホットファイルを SSD へ移送することが可能である．しかしながら，このシステムは，ストレージボリューム内の時間的・空間的局所性情報を用いた分析を行っておらず，2 章の IO アクセス集中が発生した領域を抽出して SSD へ置換できない．

BASIL [19]，Azor [20]，そして Narayanan [18] らは，tiering や caching を用いた研究成果ではないが，ロードバランシング，メタデータアクセスの最適化，コスト削減を目的にストレージ領域を動的に置換する提案である．BASIL はワークロードをその都度監視することでストレージ領域に発生するホットスポットを均等化する提案である．Azor は metadata の位置を動的に把握して SSD に置換する提案である．Narayanan らの研究は，SSD と HDD を用いた hybrid storage system を前提に price/performance と capacity/performance が最適になるようにデータ配置する提案である．これらの提案は，性能向上のために，ワークロードを監視してデータを動的に移送するものである．しかしこれら提案は，ストレージボリューム内の時間的・空間的局所性情報を用いた分析は行っておらず，2 章の IO アクセス集中を含むワークロードに対して効果的なデータ配置ができない．

第4章　IOアクセス集中領域を検出してSSDに置換する階層ストレージの提案

4.1　提案方法

4.1.1　概要

2.5 節で説明した階層ストレージシステムを構築するための方針に従い，本章では IO アクセス集中が発生した sub-LUN を IO アクセス性能を低下させずに SSD に置換する方法の構築を目指す．本章のアプローチは，HDD の過負荷が原因でアプリケーションが求める性能を提供出来なくなった時に過負荷状態を解消するために必要な sub-LUN を抽出し SSD へ自動的に置換する方法である．2.4 節の分析結果より，IO アクセス集中は全容量の 1%前後に相当する sub-LUN に発生しており，少数の sub-LUN を SSD に置換することで HDD の過負荷状態を解消できる．少数の sub-LUN 移動は，短時間で移動が完了することと IO アクセス性能への影響が小さく済むことを意味する．

本章の提案は，1) IO アクセス数が多い順に sub-LUN を並べ変えておき，2) HDD の過負荷状態を解消するのには何番目の sub-LUN まで SSD に置換すればよいのかを求め，3) 2) で求めた sub-LUN を即座に SSD に置換する，方法である．

SSD に移動する sub-LUN は，HDD の iops と IO busy 率に正の相関関係があることを利用して求めることが出来る．文献 [28] は，HDD の iops と IO busy 率に正の相関関係があることを利用して，複数の LUN が HDD を共有する環境において一部の LUN の IO アクセス性能を保証する方式の提案を行っている．本稿は，文献 [28] の成果を用いて SSD へ置換が必要な

sub-LUN の抽出を行う．文献 [28] は，IO アクセスが発生する LBA が大きく変化すると iops と IO busy 率の相関の係数が変化することを示している．さらに，この変化に追従するために高頻度で iops と IO busy 率の相関係数を更新する必要があることも示している．さらに，2 章で説明した IO アクセス集中は短時間で異なった sub-LUN 上に移動するため，筆者は iops と IO busy 率の相関係数も短時間で変化する可能性が高いと判断した．そこで本章の提案方法は，1 分前後の短い時間間隔で iops と IO busy 率の相関係数を更新し，移動が必要な sub-LUN も更新した相関係数を用いてこの短い時間間隔で再抽出する．

図 4.1 は，本稿の評価で使用した CentOS 5.4 上での実装例である．この図を用いて提案システムの説明を行う．提案システムは，アプリケーションとして動作する分析・構成変更エンジンと OS ドライバとして動作する Tiering driver から構成される．ストレージワークロードの分析に用いるデータとしては，blktrace と iostat の実行結果を用いる．

4.1.2 分析・構成変更エンジン

分析・構成変更エンジンは，Log pool，ワークロード分析，sub-LUN 移動指示，の 3 コンポーネントから構成される．以下の節で各コンポーネントに関して説明する．

Log pool

Log pool では，各 sub-LUN ごとの統計情報と階層制御に使用する各デバイス (SSD と HDD) の統計情報を収集する．これら情報を用いて，ワークロード分析が IO アクセス集中が発生した sub-LUN の抽出を行う．各 sub-LUN ごとに収集する統計情報は，一定時間間隔ごとの IO アクセス数，IO size ごとの割合，read/write の割合，レスポンスのヒストグラム，である．各デバイスごとに収集する統計情報は，一定時間間隔ごとの iops，IO busy 率，などである．

本章の実装システムは，各 sub-LUN ごとの統計情報は blktrace ログの分析結果を用い，各デバイスの統計情報は iostat の実行ログを用いた．さらに，本章の実装システムはこれらのログを 1 分間隔で回収した．

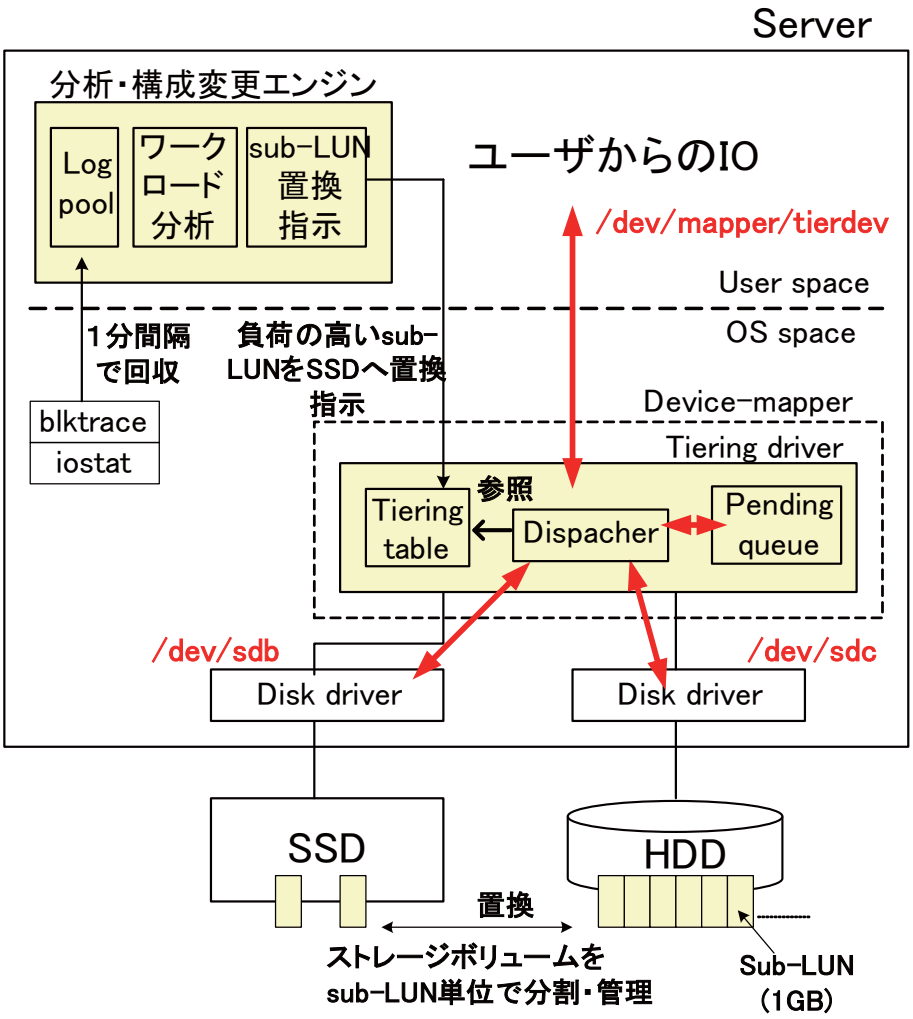

図 4.1: 提案方法の Linux(CentOS 5.4) での実装例

Log pool は，1 分間隔で回収した blktrace ログを用いて，IO アクセス数，IO size ごとの割合，read/write の割合，レスポンスのヒストグラム，を抽出し収集する．blktrace は，Tiering driver を対象に実行する．図 4.1 を例に取ると，/dev/mapper/tierdev に対して行う．この理由は，論理的なストレージ空間の何処に IO アクセス集中が発生しているのかを把握するためである（図 4.2 参照）．

階層制御に使用する各デバイスの統計情報は，1 分間隔で実行した SSD，HDD 各デバイスに対応する iostat -x の実行ログを収集する．これらのデータを用いて，各デバイスの IO busy

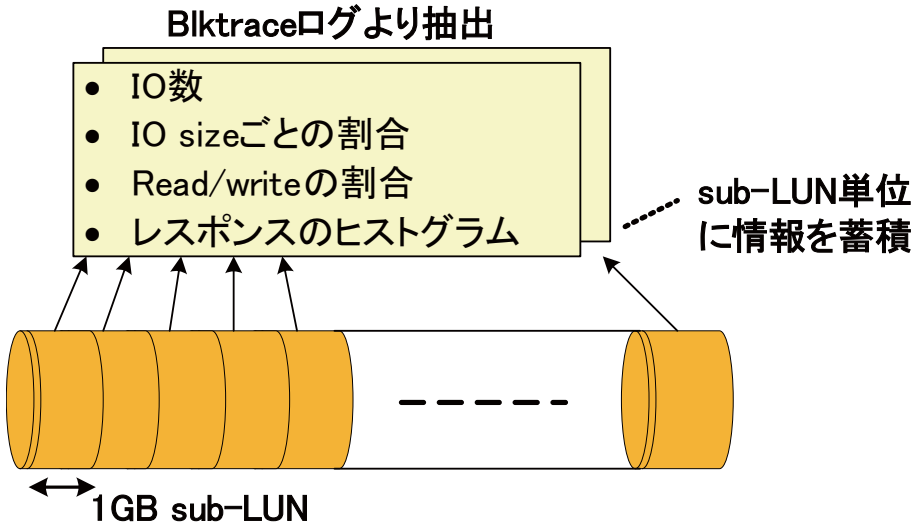

図 4.2: blktrace データの分析方法

率を把握する．図 4.1 を例に取ると，/dev/sdb と/dev/sdc の iostat 実行結果の蓄積を行う．

測定間隔を 1 分間とした理由に関して説明しておく．測定間隔を短くすればするほど構成変更のリアルタイム性を高めることが可能になる．一方，測定時間を短くすると，IO アクセス要求の頻度のばらつきやエレベータアルゴリズムなどの影響も加わり，現象の見え方が大きく変わってしまう悪影響がある．そこで，リアルタイム性を確保しながら最大 60 分程度継続する IO アクセス集中が発生した領域の抽出という目的を可能にするためには，測定間隔を適切に設定する必要がある．本章の実験では事前検証の結果を踏まえ 1 分間を測定間隔とした．

ワークロード 分析

ワークロード分析の目的は，IO アクセス集中が発生した sub-LUN を抽出しその中から SSD に置換する sub-LUN を決めることである．分析は，*Log pool*で収集した複数の blktrace/iostat ログを用いて行う．blktrace/iostat ログは timestamp を付与して管理しており，ワークロード分析は日時を指定したデータや現在時刻からの差分データを取り出すことが可能である．継続して IO アクセス集中が発生した sub-LUN を抽出するために，*Log pool* より現在時刻を起点として直前の n データを取り出しその平均値を求めておく．次に HDD 側の%util があらかじ

め決められた閾値 ($\%util^h_{up}$) を超えた場合に SSD 側に IO アクセス集中が発生した sub-LUN を置換する．なお，%util は，iostat の 1 パラメータで IO busy 率を表す．n はワークロード変化を把握するのに必要なデータ数を設定する．今回は，短時間で急激に IO アクセス数が増加する IO アクセス集中を検出することが目的であるため，n=2 とした．詳細説明は 4.1.4 章で行う．

sub-LUN 置換指示

sub-LUN 移動指示は，ワークロード分析の結果を受け，*Tiering driver* に対して sub-LUN の置換を指示する．今回の実装では，/proc に *Tiering driver* に対して sub-LUN 置換を指示する API を準備した．複数の sub-LUN 置換を行う場合，IO アクセスへの影響を少なくする目的で，HDD 側の負荷に応じて移動間隔を変更する実装を入れた．sub-LUN 置換を同時に複数依頼すると，IO アクセスが長時間ブロックされるためである．

今回の評価に使用した環境 (表 4.1) では，IO アクセスがない状態で 10 秒程度で sub-LUN 置換が可能である．SCSI timeout が 30 秒程度であることを鑑みると，IO アクセスの合間に sub-LUN 置換を行うことは可能である．IO アクセスへの影響が少ない sub-LUN 置換間隔は，負荷するワークロードごとに異なる値となるが，本稿はこの値を求めることを目的としていない．そこで，事前実験でユーザ IO への影響が少なかった 60 秒を用いることにした．具体的には，現在の%util が $\%util^h_{up}$ を上回る場合，*sub-LUN* 置換指示は 60 秒間隔で sub-LUN 置換を実行する様にした．

4.1.3 Tiering driver

Tiering driver は，*Tiering table* の内容に従って SSD もしくは HDD のいずれかに IO 命令の送出を行う．*Tiering table* は，SSD に置換した sub-LUN 情報のみが掲載され，SSD 側の先頭 LBA，HDD 側の先頭 LBA を保持している．HDD の先頭 LBA がボリューム全体の LBA と一致する．登録してある HDD 側の先頭 LBA から sub-LUN の大きさの範囲に収まる IO 命令が来た場合，SSD 側へ IO 命令を出す．*Tiering table* は，sub-LUN 置換指示による SSD〜

HDD 間の sub-LUN 置換後に更新する．置換方法の詳細は 4.1.5 章で説明する．今回の実装では，*Tiering driver* は Linux device-mapper framework 上に実装した．

4.1.4 IO アクセス集中が発生した sub-LUN の抽出方法

制御の基本方針は，HDD 側の%util を$\%util^h_{up}$〜$\%util^h_{lo}$ の範囲に留めることで HDD 側が過負荷にならないようにすることである．さらに$\%util^h_{up}$ と$\%util^h_{lo}$ の中間値として$\%util^h_{md}$ を設ける．%util が$\%util^h_{up}$〜$\%util^h_{lo}$ から出た場合，$\%util^h_{md}$ になるように，提案方式は HDD に発生する IO アクセス数を制御する（図 4.3 参照）．なお，図 4.3 の%utl_hdd_upper は$\%util^h_{up}$ である．%utl_hdd_middle は$\%util^h_{md}$ である．%utl_hdd_lower は$\%util^h_{lo}$ である．

sub-LUN の置換は文献 [28] の方法を参考にした．文献 [28] は，その時々の%util と iops より「iops 当たりの%util」求めておくことで，%util を期待する量増減させるには iops をどの位増減させればよいのかの求めかたを提案している．最初の説明は，HDD 側の%util が$\%util^h_{up}$ を上回るケースである．まず，HDD 側に存在する各 sub-LUN を iops の大きい順にソートする．次に，HDD 側の%util と iops を用いて，iops 当たりの%util を求めておく．そして，ソートした順に各 sub-LUN の iops を iops 当たりの%util に掛け合わせ，得られた結果を加算していく．k 番目の sub-LUN の iops を hdd_iops(k) とすると，以下の計算を行うことになる．

$$\%util\_del = \sum_{k=1}^{n} hdd\_iops(k) * (\%util\_per\_iops)$$

%util_del は n 番目までの sub-LUN を置換した時の%util の変化量となる．

現在の HDD 側の%util から%util_del を引いた値が$\%util^h_{md}$ を下回る n の値を求め，その sub-LUN までを SSD へ置換する．

次の説明は，HDD 側の%util が$\%util^h_{lo}$ を下回るケースである．このケースでは，負荷の軽い SSD 側の sub-LUN を HDD 側に置換することになる．まず，SSD 側に存在する各 sub-LUN を iops の小さい順にソートしておく．次に，HDD 側の%util と iops より iops 当たりの%util を求めておく．そして，ソートした順に各 sub-LUN の iops を iops 当たりの%util に掛け合わせ，得られた結果を加算していく．k 番目の sub-LUN の iops を ssd_iops(k) とすると，以下の

計算を行うことになる．

$$\%util\_add = \sum_{k=1}^{n} ssd\_iops(k) * (\%util\_per\_iops)$$

現在の HDD 側の%util から%util_add を加算した値が$\%util^h_{md}$ を上回る n の値を求め，その sub-LUN までを HDD へ置換する．

最後に$\%util^h_{up}$，$\%util^h_{md}$，$\%util^h_{lo}$ の値の決め方指針と今回の評価での設定値に関して説明する．$\%util^h_{up}$ は HDD が過負荷かどうかを判定するパラメータであり，この値を上回ると SSD への置換を開始する．この値を大きく取れば，その分 HDD の性能をぎりぎりまで使うことが出来るが，SSD への置換を判断するタイミングが遅れることになる．筆者は，HDD 側の%util が 100%に到達する前に SSD への置換が始まるかどうかの観点で事前評価を行い，今回評価に使用したワークロードでは 70%程度が望ましいと判断した．$\%util^h_{md}$ は，sub-LUN 置換後の HDD 側%util の僅かな変動で再び置換判定とならないようにするために，$\%util^h_{up}$〜$\%util^h_{lo}$ の中間程度の値にする必要がある．今回の評価だと 40 前後に設定する必要があり 50 に設定して評価を行った．$\%util^h_{md}$ は，中間値程度に設定しておけば実験結果への影響はほとんどなく，実験当初に設定した値をそのまま使い続けた．$\%util^h_{lo}$ は HDD の負荷が軽いかどうかを判定するパラメータであり，この値を下回ると SSD へ置換済み sub-LUN の一部を HDD へ移動する．この値を大きく取ればその分 HDD の稼働率を上げることが出来るが，ワークロードによっては HDD 側の負荷が一時的に下がったタイミングで SSD へ置換した sub-LUN を HDD に戻してしまい，しばらくして HDD 側の負荷が上がったタイミングで再び SSD へ置換する動作となる．筆者は，事前評価でこの様な動作とならない値を求め，10%に設定することにした．

4.1.5 SSD への移動方法

本節は，4.1.4 節で求めた sub-LUN を実際に SSD へ置換する方法を説明する．図 4.4 は，本章の評価に使用した CentOS 5.4 における *Tiering driver* の詳細ブロック図であり，筆者は IO アクセスと sub-LUN 置換を同時に行える設計を行った．

最初の説明は，IO アクセス命令の処理方法である．IO アクセス命令が *Tiering driver* に入っ

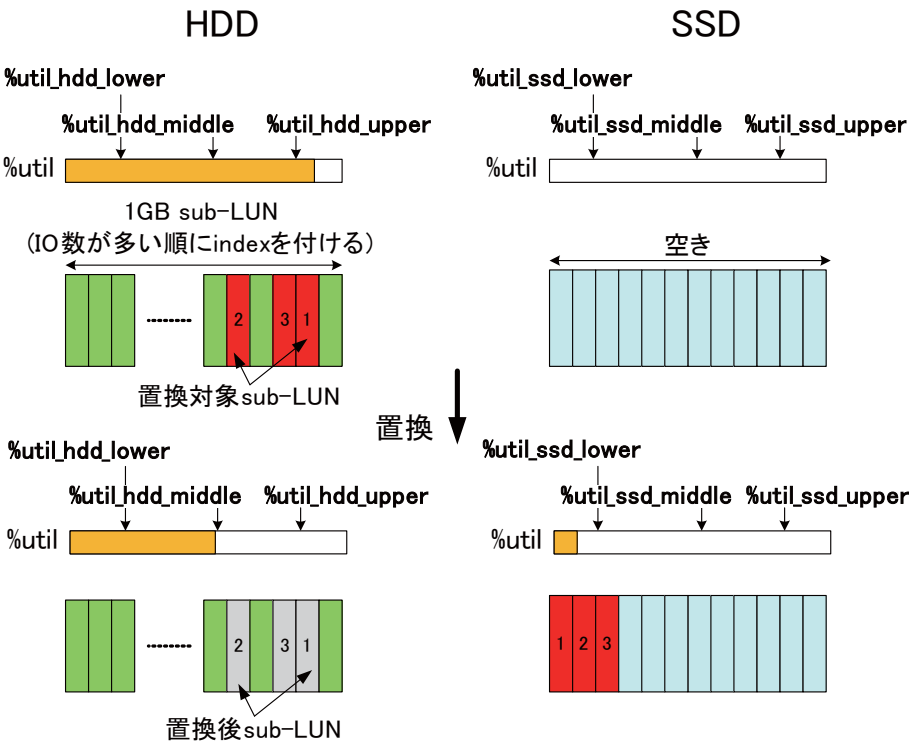

図 4.3: HDD から SSD に sub-LUN を置換する方法

てくると，*Tiering_map()* で受け取った IO アクセス命令の LBA と *Tiering table* の比較を行う．比較の結果，*Tiering table* に載っていれば SSD へ，そうでなければ HDD へ IO アクセス命令を送出する．

次の説明は，sub-LUN の SSD〜HDD 間置換方法である．図 4.5 は，sub-LUN 置換シーケンス図である．分析・構成変更エンジンからの置換指示は，/proc/tiering_table を介して *Tiering driver* に渡される．*Tiering driver* は，置換指示を受け取ると device-mapper 内共通モジュールである kcopyd を用いて sub-LUN 置換を行う．データ置換中の sub-LUN への IO アクセス命令は Pending Queue に蓄積しておき，sub-LUN 置換完了を契機に Pending Queue に蓄えられた IO を実行する．

4.2 提案方法の評価と考察

4.2.1 概要

評価は以下の 4 つの観点で行った．

- HDD 単体との比較
- long IO アクセス集中領域と short IO アクセス集中領域の構成比変更実験
- cache 方式との比較
- blktrace,iostat オーバーヘッド調査

表 4.1 は，評価に使用した環境である．評価に使用したワークロードは，2 章で分析を行った MSR Cambridge ワークロードの一部である．本章の評価は，このワークロードの中から IO アクセス集中が長時間継続していた表 4.2 の 4 ワークロードを用いた．表 4.3 は，これらワークロードの IO アクセスの偏りを分析した結果である．いずれも全ボリュームの 10%に半分以上の IO アクセス (iops) が発生している．

ワークロード再現を行うプログラムは，blktrace で取得したデータの再生を行う btreplay である．MSR Cambridge ワークロードを構成するトレースデータは blktrace とは異なる ETW[23] 形式であるので，本章の評価は btreplay が解釈可能な形式に変換する filter を通してワークロードを再現した．評価の目的は，HDD 単体で負荷出来ないワークロードに対して，僅かな SSD 容量に本章の提案方式を組み合わせることで負荷出来ることを示すことにある．そこで，そのままでは HDD 側を過負荷に出来ないワークロードに関して，btreplay を倍速実行することで過負荷状態をつくり出した．btreplay の倍速実行方法は，-x num オプションを用いることで何倍速にするのかを指定できる．なお，本章で過負荷とは，iostat %util が継続してほぼ 100%となる状態を指す．

表 4.1: ストレージ装置性能調査環境

PC	FUJITSU PRIMERGY RX200S6 Xeon E5640 2.67 GHz,Mem=32GB,CentOS 5.4
HBA	LSI SAS 9200-8e
SSD	intel X25-E(SSDSA2SH064G1GC) (64GB)
HDD	SEAGATE ST9600204SS (600GB)

表 4.2: 評価に使用したワークロード

ワークロード	処理内容	全容量 (GB)	peak iops	負荷集中が発生した時間帯 (分)*
usr1	ユーザのホームディレクトリ	600	237	2090-2340
web2	web と SQL サーバ	170	231	5870-5930
proj2	プロジェクトディレクトリ	600	1070	6760-6880
proj4	プロジェクトディレクトリ	236	794	7390-7590

*: トレース先頭からの経過時間

4.2.2 HDD 単体との比較

HDD から SSD への sub-LUN 置換オーバーヘッドまで含めた提案方法の効果を確認する目的で，HDD のみで表 4.2 のワークロードを実行した場合との比較を行った．評価は，ワークロード起動から 1 時間までの平均 iops を比較することである．HDD を過負荷状態にするために，本章の評価では btreplay を usr1, web2 は 8 倍速で，proj2，proj4 は 2 倍速で，実行した．表 4.4 が実行結果である．表 4.4 は，usr1，web2，proj2 で性能向上が見られ最大 77%向上するが，proj4 ではほとんど効果がないこと，を示している．

表 4.5 は提案方法における SSD への IO アクセス比，及び提案方法と HDD 単体における HDD 側の svctm である．IO アクセス比は (SSD への IO 数*100)/(全 IO 数) にて求めた．svctm は iostat の 1 パラメータであり，1IO の平均実行時間を意味する．usr1 は，SSD 6GB の範囲

表 4.3: 評価ワークロードの IO アクセス (iops) 集中度

全ボリュームに占める割合	usr1	web2	proj2	proj4
1%	95%	8%	17%	43%
10%	100%	48%	75%	70%

に 89%の IO アクセスが発生しており，提案方法により IO アクセス集中が発生した sub-LUN を SSD に置換でき，その結果として HDD 単体と比較して 77%性能向上した．全体の 1%の sub-LUN(6GB) を SSD に置換するだけで性能向上出来たことが分かる．

web2 は，SSD に置換した 27GB の sub-LUN に 55%の IO アクセスがが発生している．usr1 より効果が小さい主な理由は，HDD と SSD の間の sub-LUN 置換数が多いためである．4.1.2 節にて説明したように複数の sub-LUN 置換を同時に依頼すると，提案システムは sub-LUN 置換を 60 秒間隔で実行する．これは，web2 の SSD への sub-LUN 置換に 27 分必要とすることを意味し，提案システムによる十分な効果が得られるまで時間がかかるためである．

proj2 は，SSD に置換した 23GB の sub-LUN に 11%の IO アクセスしか発生していないが 45%性能向上している．表 4.5 の提案方法と HDD 単体における HDD 側の svctm を比較すると，提案方法の HDD 側 svctm が 0.39ms 向上している．svctm はその時に取り出すことが可能な最大 iops の逆数と解釈出来るので，svctm が小さくなるということは，HDD 側から取り出すことが出来る性能が大きくなったことである．svctm が小さくなった理由は，SSD へ IO アクセス集中が発生した sub-LUN を置換することで HDD 側の seek などのオーバーヘッドが小さくなったため，と推定している．提案方式と HDD 単体の%util の比較を行ったところ，両者とも 1 時間の平均値がほぼ 100%であった．このことより，svctm の差がそのまま性能差となったと言える．

proj4 は，SSD に置換した 23GB の sub-LUN に 1%の IO アクセスしか発生せず向上率も 1%，である．図 4.7 は，ベンチマーク終了時には提案方法の HDD 側の%util は 50%程度でそれほど負荷がかかっていないが HDD 単体は常に 100%であること，を示している．図 4.6 は，提案方法と HDD 単体における 1 分間隔の iops の推移である．提案方法は，3000iops 超の負

荷が 3 回発生し，一度に高負荷状態を解消し，その後負荷が下がる．一方，HDD 単体は提案方法の負荷が下がり始めた 31 分後も 1000 iops 前後の性能を維持したため，最終的に HDD 単体と提案方式との差がなくなった．proj4 は，平均 iops 向上には効果がなかったが，peak iops 向上には効果があった．SSD へ 23GB sub-LUN 置換したのに平均 IOPS の向上率が 1%に留まった理由は，SSD へ置換した sub-LUN への IO アクセス集中の継続時間が短かったためと考えられる．

まとめると，usr1 の様に IO アクセス集中が発生する sub-LUN 数が少ないワークロードは，提案方法によりすぐに性能向上する．web2 の様に IO アクセス集中が発生する sub-LUN 数が多いワークロードは，sub-LUN 移動に時間が必要なため，十分な効果が出るまで時間が必要となる．proj2 の様に SSD への IO アクセス数が余り多くないワークロードでも，IO アクセス集中が発生した一部の sub-LUN が SSD に置換されることで HDD 側オーバーヘッドが小さくなり，SSD から得られる性能以上に性能向上する．proj4 の様に IO アクセス集中が発生する sub-LUN の IO アクセス数が極めて小さいワークロードでも，peak iops 向上に効果がある．

最後の議論は，sub-LUN の大きさに関してである．表 4.6 は，評価に用いた各ワークロードの IO アクセスが発生した領域の大きさである．usr1，proj2 は平均値が 0.3-0.4GB であるので 1GB sub-LUN にすると無駄な領域が生じるが，proj4 は 1GB が適しており，web2 は sub-LUN size を 12GB 程度にするのが適している．一方，sub-LUN size を大きくすると HDD から SSD への sub-LUN 転送時間が増加するため，sub-LUN 転送による負荷が IO アクセス性能の低下させる可能性が高まる．本章の検証環境は，1GB 転送するのに 10 秒程度かかった．SCSI timeout が 20-30 秒であることを鑑みると，本章の検証環境では sub-LUN size を最大 2GB 程度に留めるべきであることが分かる．まとめると，本方式を適用する環境ごとに，HDD から SSD への転送時間による上限 sub-LUN size を設定した上で，ワークロードに応じて sub-LUN の大きさを変更出来る様に実装するのが SSD 領域をより有効に利用できる．

表 4.4: 1 時間経過後の平均 iops(HDD 単体との比較)

ワークロード	平均 iops (提案方法)	平均 iops (HDD 単体)	SSD 使用量 (GB)	向上率 (%)
usr1	1483	836	6	77
web2	1439	1174	27	22
proj2	635	436	23	45
proj4	865	854	23	1

表 4.5: SSD rw の割合と svctm(HDD 単体との比較)

ワークロード	SSD への IO アクセス比 (%)	HDD の svctm (提案方法) (ms)	HDD の svctm (HDD 単体) (ms)
usr1	89	0.39	1.20
web2	55	0.60	0.86
proj2	11	1.93	2.32
proj4	1	0.90	1.07

4.2.3 long IO アクセス集中領域と short IO アクセス集中領域の構成比変更実験

本実験の目的は，同じ sub-LUN に長時間継続する IO アクセス集中（long IO アクセス集中領域）と短時間で IO アクセス集中が別 sub-LUN 上に移動する IO アクセス集中（short IO アクセス集中領域）に関して，其々の領域の IO アクセス数の割合を変更して実験することで，提案方式が効果的なワークロードを明確にすることである．

最初の議論は，本実験における short IO アクセス集中領域の IO アクセスの継続時間に関し

表 4.6: IO アクセス集中領域の大きさ (単位: 1GB)

	usr1	web2	proj2	proj4
最小値	0.1	0.1	0.1	0.1
平均値	0.4	11.9	0.3	1.0
最大値	1.0	34.3	1.0	4.8

表 4.7: long IO アクセス集中領域と short IO アクセス集中領域の構成比変更実験結果

	平均 iops	SSD 使用量 (GB)	増加 iops (実測値)	増加 iops (期待値)
IO accessx4	1498	2		
short IO access x1	261			
i x4 + si x1	1676	2	178	261
i x4 + si x2	1967	6	469	523
i x4 + si x3	2262	8	765	784
i x4 + si x4	1699	10	201	1046
i x4 + si x5	1284		-213	1307

i: long IO access, si: short IO access

表 4.8: blktrace,iostat オーバーヘッド調査結果

	分析・構成変更エンジン有り	分析・構成変更エンジン無し
iops	1481	1466
CPU %idle	92.5	92.5
w/s*	8.1	0.5

*: システムディスクに対する write sector per sec.

てである．short IO アクセス集中領域を本章の実装に照らしあわせると，少なくとも IO アクセス集中が発生した sub-LUN を検知し，その sub-LUN を SSD へ置換が終わったときには，IO アクセス集中が終息している sub-LUN，となる．提案システムは直前の 2 つの *Log pool* に蓄えられたデータを用いて SSD に置換する sub-LUN の判定を行っており，さらに sub-LUN 置換は少なくとも sub-LUN 当たり 10 秒必要とする．この事実より，本実験環境における short IO アクセス集中領域の IO アクセスの継続時間は，少なく見積もっても 3 分程度になる．なお，long IO アクセス集中領域は，short IO アクセス集中領域より継続時間が長い領域とする．

実験は，表 4.2 の proj4 を用いた．proj4 は，4 GB の long IO アクセス集中領域と 187 GB の short IO アクセス集中領域から構成されている．そこで，long IO アクセス集中領域と short IO アクセス集中領域の IO アクセス数の割合を変更できるようにする目的で，本実験は proj4 を long IO アクセス集中領域に対応するトレースデータと short IO アクセス集中領域に対応

するトレースデータに分離し，各トレースデータを用いて同時に btreplay を実行した．long IO アクセス集中領域の btreplay 倍速度は 4 で固定し，short IO アクセス集中領域の倍速度を 1 から 5 に変化させた．btreplay の実行時間は 20 分とした．事前の調査で short IO アクセス集中領域の btreplay 倍速度を 4 倍以上に設定すると HDD 側%util がほぼ 100%となることが分かった．そこで，HDD が過負荷となる場合とならない場合で提案方式の効果を明確にするために，本実験では上記の様な設定を行った．

表 4.7 が実験結果である．SSD 使用量は，SSD へ移動が完了していない sub-LUN は含まない．増加 iops(期待値) は short IO アクセス領域 x1 実測値が倍速加算された場合の値である．実験結果より，ピーク性能が得られるのは short IO アクセス領域に対応するトレースデータを 3 倍速で実行した場合である．これは short IO アクセス集中領域の IO アクセスが HDD 性能の範囲に収まっている範囲では性能向上することを意味する．

short IO アクセス集中領域に対応するトレースデータを 4 倍速以上で実行すると，short IO アクセス集中領域に対応する sub-LUN の SSD への移動オーバーヘッドが大きくなり，SSD による性能向上効果が薄れてしまう．

この実験結果より，提案方法は short IO アクセス集中領域に発生する IO アクセスが HDD 側の性能を大幅に上回らないワークロードで効果的であることが分かる．

4.2.4 cache 方式との比較

4.2.3 節の実験結果は，ワークロード構成要素の 1 つである long IO アクセス集中領域と short IO アクセス集中領域の割合が提案方式の効果に影響することを示した．また，SSD の使いかたとして，本章で提案した階層ストレージ方式と並んで cache 方式がよく用いられる．そこで，cache 方式より提案方式を適用した方がより性能を引き出せるワークロードの条件を明確にする目的で，本実験は両方式の比較を行う．比較する cache 方式は，オープンソースとなっている Facebook Flashcache [7] である．比較実験を行う PC の仕様は，表 4.1 である．

事前の調査は，Flashcache が IO 命令の LBA が 4KB 境界と一致した場合のみ cache する実装であり，表 4.2 の全てワークロードの大部分が cache unhit となること，を明らかにした．

そこで本実験では，表 4.2 のワークロードの LBA をもよりの 4KB 境界に書き換える filter を準備し，実験に使用したワークロードは全てこの filter を通して LBA を変更した．なお，この filter における LBA の変更方法は，new_LBA=(LBA/4096)*4096 である．さらに，本章の提案方式と cache 方式は，SSD へ置換する時刻と置換単位が異なるだけであり，read 中心のワークロードではほとんど性能差がでない，と予想される．そこで，本実験では，read の一部を write に変更する filter を作成し，この filter を通して read/write 比を変更したワークロードを一部の実験で用いた．

本節の実験は，cache 方式の実装である flashcache を動かすために人為的に変更したワークロードを用いた評価となる．そのため，cache 方式を実環境に適用した場合，実験結果ほどの効果が得られない可能性がある．しかしながら，提案方式の効果は LBA 変更 filter の影響を受けないため，本実験で得られた提案方式の優位点は実環境でも継続すると考えられる．

評価に用いるワークロードは，表 4.2 の中から long IO アクセス集中領域と short IO アクセス集中領域の構成比が異なる usr1 と proj2 である．flashcache が使用する SSD 容量は，表 4.4 と一致するように設定した．btreplay の倍速は usr1 が 8 or 10 倍，proj2 が 2 倍に設定した．さらに，4.1.4 節で説明を行った IO アクセス集中が発生した sub-LUN 抽出ロジックに関して，本実験では HDD 側が過負荷状態に陥っていなくても IO アクセスが継続的に発生する sub-LUN を積極的に SSD へ置換するロジックを追加した．この理由は，cache 方式に対して優位性を持たせるためには，負荷が余り大きくなくても継続的に IO が発生する領域を SSD に置換する必要があることが事前実験で判明したため，である．具体的には，HDD 側の%util が $\%util^h_{up}$〜$\%util^h_{lo}$ の範囲に収まっていても，新たなロジックは同じ sub-LUN に連続して m 回 IO アクセスが継続する sub-LUN を SSD へ置換する．m はワークロードに依存するパラメータであり，m 回 IO アクセスが継続した sub-LUN は，さらに長期間 IO アクセスが継続することを見込める．m はワークロードに応じて最適値が異なると考えられるが，今回は m=8 で事前検証を行い継続的に IO が発生する sub-LUN の抽出が出来たため，本実験ではそのままこの値を用いた．

表 4.9 が実験結果である．usr1 は，read の割合が高いと Flashcache との性能差がないこ

表 4.9: 1 時間経過後の平均 iops(flashcache との比較)

ワークロード	r:w	平均 iops (提案方法)	平均 iops (cache 方式)	向上率 (%)
usr1	91:9	1466	1466	0
usr1*	46:54	1407	1462	-4
usr1$	46:54	1587	1474	8
usr1(x10)*	46:54	1531	1609	-5
usr1(x10)$	46:54	1805	1610	12
proj2	88:12	788(#)	1478	-47
proj2*	44:56	460	683	-33

*: read 要求を 2 回に 1 回 write に変更 (サイクリックに変更)
$: 提案方法における sub-LUN 移動が終わった後から比較
#: ワークロードを変更したため表 4.4 と値が異なる

とが分かる．usr1 の write の割合を大きくすると，1 時間を通して比較すると若干提案方法の性能が劣るが，SSD への sub-LUN 置換が終了した後の期間で比較すると提案方法が最大 12%性能向上することが分かる．図 4.8 は usr1(x10)$の 1 分間隔の iops の推移である．最初の 3GB sub-LUN の置換は，4.1.4 節で説明をおこなったロジックによるものである．最後の 3GB sub-LUN の置換は，本実験用に追加した積極的に SSD への置換を行うロジックによるものである．両者の移動に 10 分近い差がある理由は，最初の 3GB sub-LUN 置換完了後から 8 データ蓄積する必要があるためである．write の割合を増やすと提案方法の性能が上回る理由は，flashcache では write の割合を増やすと HDD への write back 帯域が増加するためである．表 4.9 に関して，btreplay を 8 倍速から 10 倍速に変更すると向上率が上昇するのはこのためである．usr1 の様に long IO アクセス集中領域の割合が大きいワークロードでは，write の比率が高いと flashcache より提案方法の方がより効果的である．ただし，IO アクセス集中が発生した sub-LUN 抽出ロジックは，4.1.4 節で説明したロジックに加えて，ワークロードの特徴に応じて積極的に SSD への置換を行うロジックを加えると，より効果的である．

proj2 に関しては，常に Flashcache の性能が提案方法を上回った．図 4.9 は，proj2*の 1 分間隔の iops の推移である．usr1 より short IO アクセス集中領域の割合が大きくなる proj2 で

は，flashcacheが効果的である．

この実験結果より，数分から数十分の範囲で負荷が収束するshort IOアクセス集中領域が支配的なワークロードに関してはcache方式が適しており，数時間負荷が継続するlong IOアクセス集中領域が支配的でwrite比率が高いワークロードに関しては提案方式が適している，と言える，EMC FAST[11]が採用する自動階層制御方式は1日単位の階層移動となるため，本稿で取り上げた数時間程度継続するIOアクセス集中領域の性能向上へは効果的ではない．

4.2.5 blktrace,iostat オーバーヘッド 調査

4.1.2節で説明したように，提案方法は，blktraceとiostatを常に実行することで，ワークロードの短期間の分析を行う．そこで本節は，blktraceとiostatを常に実行することによるIOアクセスへの影響調査結果を報告する．調査方法について説明する．最初のステップは，提案システムを動かした状態でusr1の再現を行い，6 GB sub-LUNのSSD移動まで進めた．次のステップは，usr1を10分間づつ再実行し，分析・構成変更エンジンの有無でiopsやcpu使用率がどのように変化するのかを調査した．

表4.8は，実験結果である．実験結果は，ほぼ同じiopsが得られている状態でCPU %idleがほぼ同じ値となることを確認し，システムディスク側のwrite負荷が若干増加するがユーザ性能に影響を与えるほどではないことを示した．今回実験に使用した環境では，blktrace,iostatのIOアクセスへの影響はほとんどないと判断できる．

4.3 まとめ

本章では，特定の領域へのIOアクセス集中が数時間発生するワークロードを対象に，IOPSの平均値と最大値の向上を目指して，IOアクセスが集中する領域を短時間に抽出し，その領域を即座に高速ストレージ (SSD) に置換する階層ストレージシステムの提案，を行った．本章の提案方式は，最大でも全ボリューム容量の10%のSSD容量を追加するだけでIOPSの平均値と最大値を大幅に向上することに成功した．さらに提案方式は，IOアクセス集中領域が狭

いとすぐに大きな性能向上が得られること，広い場合は効果が出るまで時間がかかるため IO アクセス集中の長時間の継続が必要なこと，を示した．本章の評価は，IO アクセス領域が広い場合で性能向上効果を得るには SSD への sub-LUN 置換方法の高速化が必要であることも示した．

また本章の評価は，IO アクセス集中領域以外の IO アクセスに関して，IO アクセス集中領域が SSD に移ることで HDD のオーバーヘッドが小さくなり，SSD から得られる性能以上にボリューム全体の IO アクセス性能が向上するケースを確認し，IO アクセスの分析に使用する blktrace,iostat の実行が IO アクセスにほとんど影響しないことも，確認した．

cache 方式との比較結果は，IO アクセス領域の IO アクセス数が全 IO アクセス数に対して高い割合となり，さらに write の比率が高いワークロードで提案方式が優位であることを示した．この理由は，cache 方式で発生する write back が提案方式では発生しないためである．

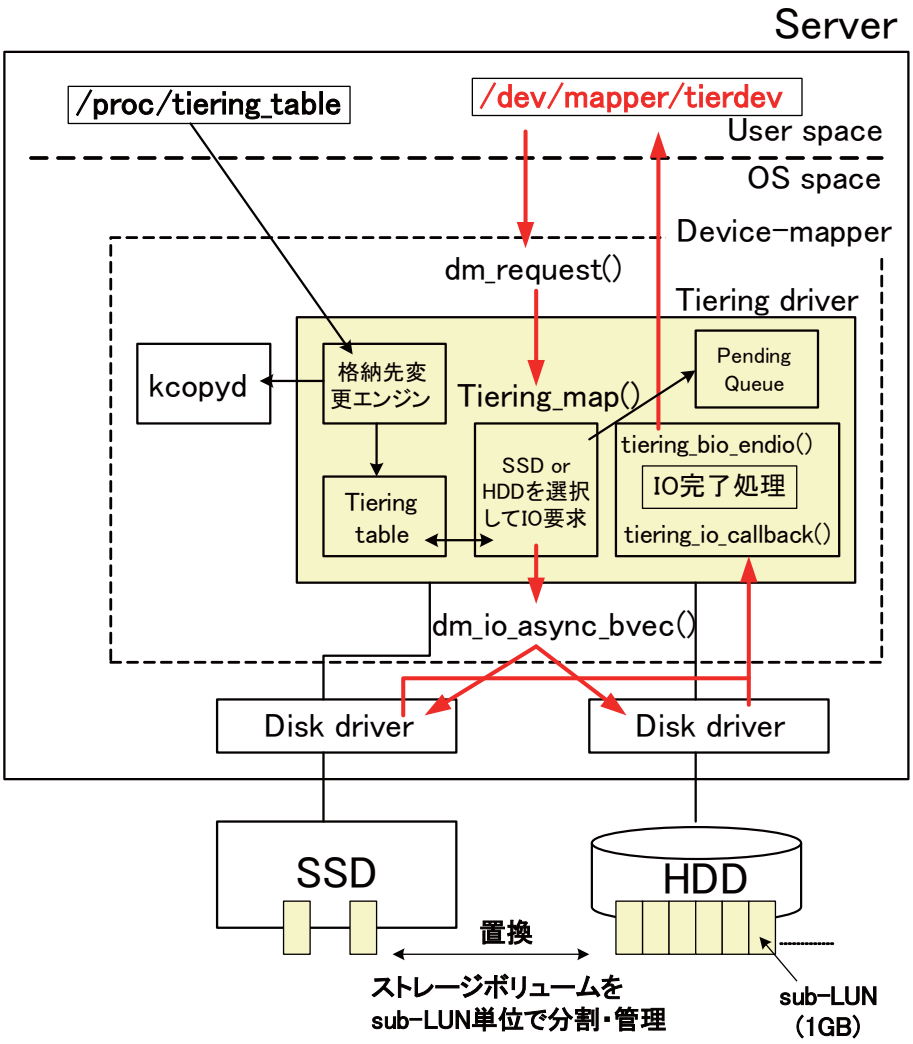

図 4.4: tiering driver の Linux(CentOS 5.4) での実装例

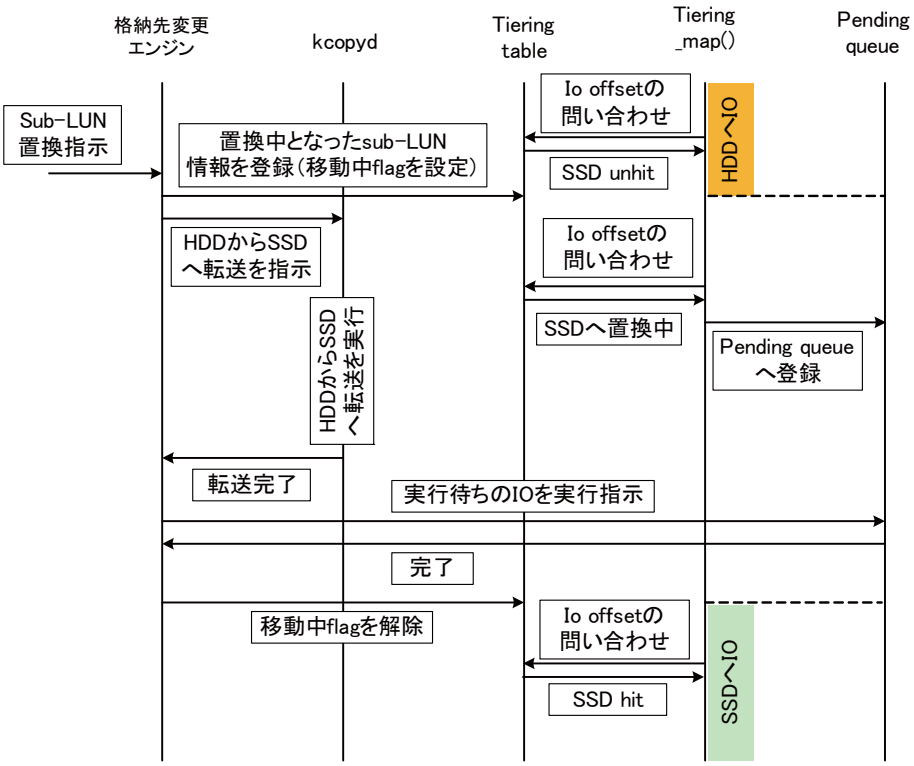

図 4.5: HDD から SSD にデータ置換を行うシーケンス図

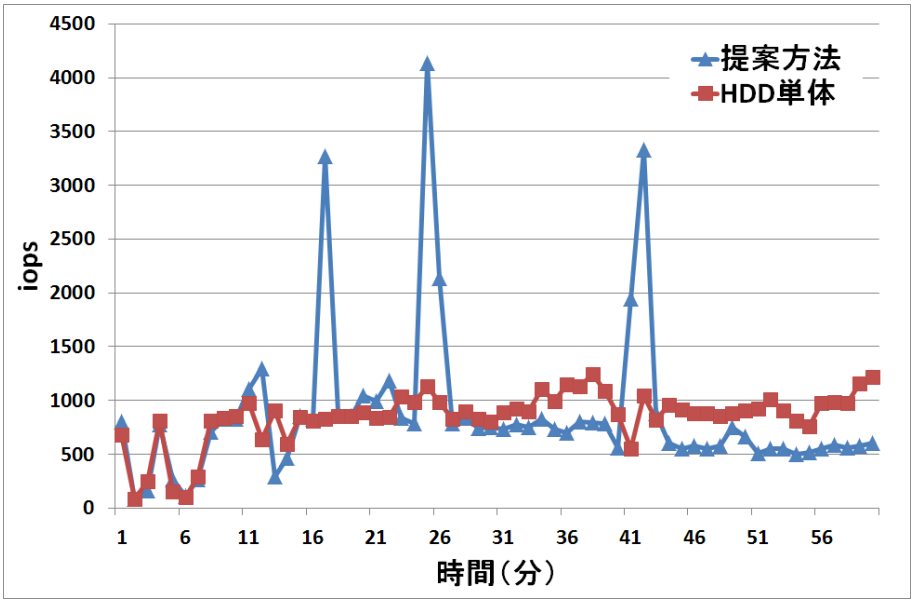

図 4.6: proj4 の 1 分間隔の iops(HDD 単体との比較)

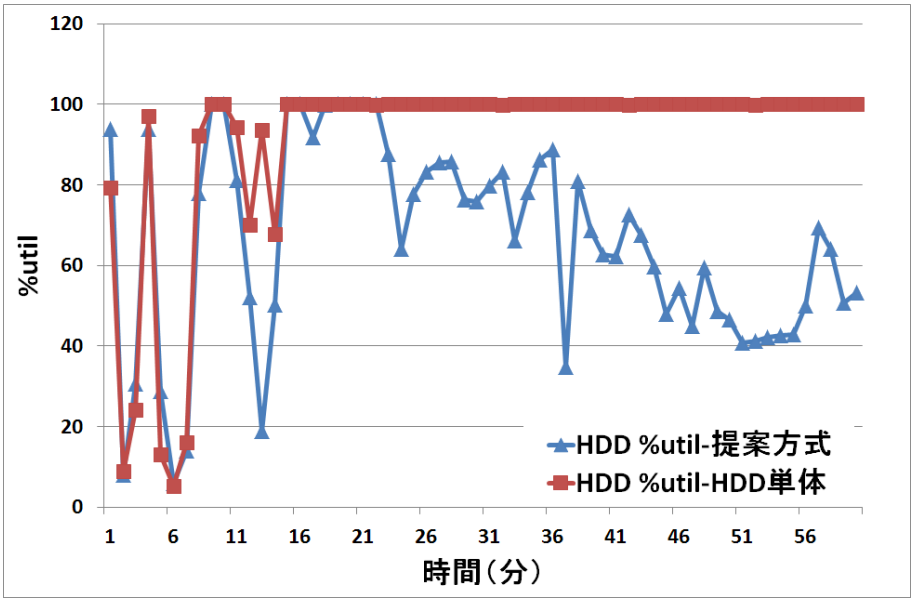

図 4.7: proj4 の 1 分間隔の%util(HDD 単体との比較)

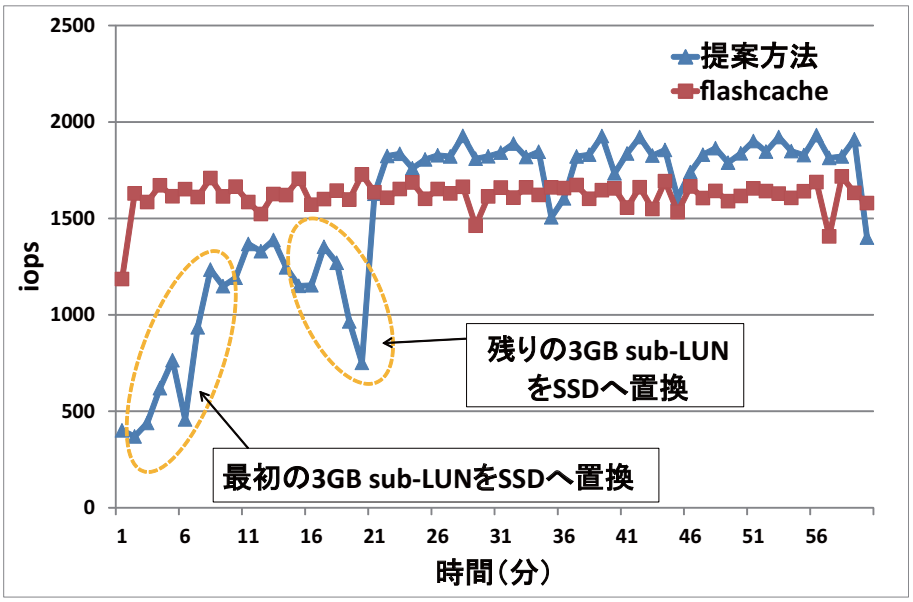

図 4.8: usr1(x10)$の 1 分間隔の iops(flashcache との比較)

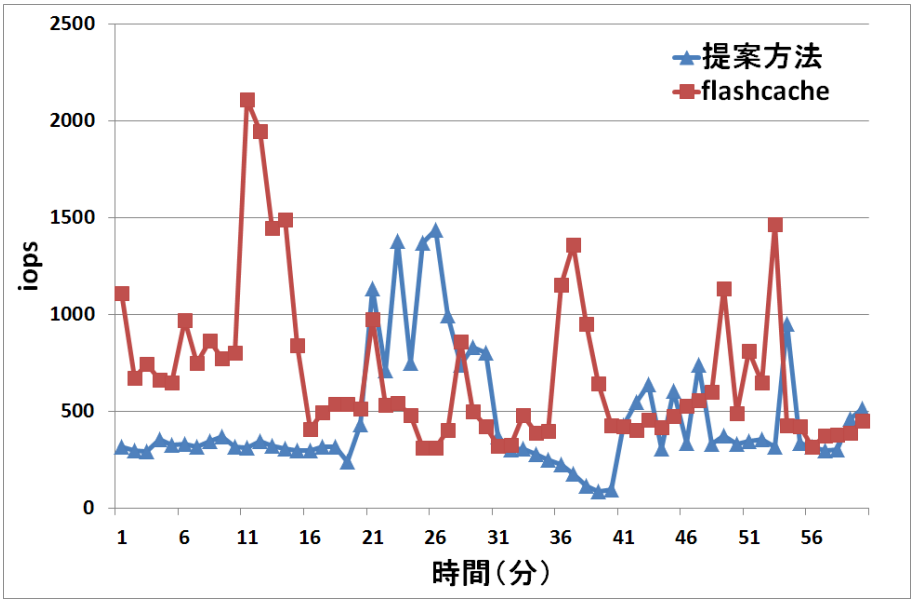

図 4.9: proj2*の 1 分間隔の iops(flashcache との比較)

第5章　参照局所性と領域移動を考慮して高速にIOアクセス集中領域を検出する階層ストレージ OTF-ASTの提案

5.1 On-the-fly automated storage tiering (OTF-AST)の提案

5.1.1 概要

4 章で説明した *IO busy* 率を用いた階層制御方式は，2 章で説明を行った IO アクセス集中の中の一部のパターンの性能向上に成功した．一部のパターンとは，同一 sub-LUN 上に長時間 IO アクセス集中が継続するケースである．一方，同一 sub-LUN 上に短時間 IO アクセス集中が継続した後に隣接 sub-LUN 上へ移動するパターンは，4 章の提案方法では IO アクセス集中が発生した sub-LUN を迅速に置換できないため，IO アクセスの性能向上が難しかった．IO アクセス集中が発生した sub-LUN を迅速に置換できない原因は，提案方式が HDD の過負荷状態を検知して sub-LUN 置換を行う方法を用いたためである．

そこで本章では，IO アクセス集中が発生した領域を空間的局所性と時間的局所性の観点で分析し，SSD 置換によって性能向上が期待できる程度の IO アクセス数と継続時間が発生している領域を抽出し，即座に SSD に置換する on-the-fly automated storage tiering (OTF-AST) の提案を行う．さらに本章では，IO アクセス集中領域の移動速度と成長速度を常時計測し，これら情報を用いて IO アクセス集中がごく近い将来発生する sub-LUN を事前に SSD へ置換す

る方法も合わせて提案する．

OTF-AST は，運用中に HDD と SSD の間で sub-LUN 置換を行うため，sub-LUN 置換と並行して実行される IO アクセスは遅延する．そこで，OTF-AST は，性能向上を期待できる程度の継続時間がありまとまった IO アクセス数が期待できる IO アクセス集中領域のみを短時間で検出して SSD に置換することで，置換中に生じた IO アクセス遅延を補い，IO アクセス集中を含むワークロード実行時の IO アクセス性能の向上を行う．OTF-AST は，SSD への置換が必要な IO アクセス集中が発生した sub-LUN を選択するアルゴリズム，前述のアルゴリズムで選択した HDD 上の sub-LUN を短時間で SSD 上へ置換するためのシステム構成，さらに HDD 上の sub-LUN を SSD に置換する際に生じる IO アクセスの遅延を抑えるための技術，から成る．

3.2 節や 3.2 節で説明した様に，従来方式は，IO アクセス集中が発生する領域を効果的に SSD に配置出来なかったり，HDD への同期アクセスが発生する，ため IO アクセス性能が向上しない．一方 OTF-AST は，性能向上を期待できる程度の継続時間がありまとまった IO アクセス数を期待できる IO アクセス集中領域を短時間で検出して SSD に置換することで，IO アクセス性能の向上が可能となった．さらに OTF-AST は，SSD 置換後は HDD へのアクセスは一切発生しないため，Caching の様に HDD への同期アクセスがボトルネックとならない．

なお，IO アクセス集中を含まないワークロードを適用すると，OTF-AST の IO アクセス性能は，SSD への置換が発生しないため，HDD のみで実行した性能と同一となる．

5.1.2 OTF-AST が SSD に移動する IO アクセス集中

前節で説明したように，OTF-AST は性能向上を期待出来る程度の継続時間がありまとまった IO アクセス数が期待できる IO アクセス集中が発生した sub-LUN を短時間で選択する必要がある．2 章の分析結果を例に取ると，IO アクセス集中の継続時間は数分から 60 分前後となっており，SSD に置換する IO アクセス集中領域の短時間抽出は，性能向上に有効である．そこで本提案は，短時間で IO アクセス集中が終息する sub-LUN と IO アクセスの全 IO アクセス数に対する割合が低い sub-LUN を排除し，残った sub-LUN を HDD から SSD に置換す

る方針を用いた．さらに，実際に SSD に置換を行う sub-LUN を Walking IO Concentration Area (WIOCA) と命名した．

WIOCA の抽出は，IO アクセス集中が発生した sub-LUN に対して，全 IO に対する集中度と継続時間の 2 つの尺度でフィルタリングすることで行う．OTF-AST は，1 つの各 sub-LUN ごとの IO アクセス数を集計した情報を用いて，全 IO に対する集中度の観点でフィルタリングを行い，WIOCA の候補となる Candidate WIOCA (C-WIOCA) の抽出を行う (5.1.4 節)．さらに，この C-WIOCA が予め決められた閾値以上継続すると，OTF-AST はこの C-WIOCA を WIOCA と判断し，WIOCA に対応する sub-LUN を即座に SSD に置換する (5.1.4 節)．

5.1.3 OTF-AST パラメータの設定方法

表 5.1 は，SSD に置換する IO アクセス集中領域 (WIOCA) を選択するために用いるパラメータ群である．これらパラメータの初期値は，OTF-AST を適用するワークロードの事前分析結果と OTF-AST を動かす PC 性能より決めることが出来る．2 章で分析を行った Samba ワークロードと MSR Cambridge ワークロードは，いずれも分析データ全体を通して同じ特徴のログが残っている．よって，OTF-AST を適用する直前のワークロードの特徴が，OTF-AST 運用直後も継続すると考えられる．ここでは，2 章の分析結果と表 5.2 にまとめた本章の実装システムを例に，設定方法を説明する．まず t は，2 章の分析が 60 秒単位で分割したデータを用いていることより，60 と設定する．次に c の設定方法を説明する．図 2.2 の samba は，3 分までの継続時間を取り除くと残りの IO アクセス集中の約 94% が 10 分以上の継続となるので，3 分と設定する．i は LUN 全体の IOPS が低く SSD に置換を行っても性能向上効果を期待できないケースを排除するための閾値である．本章の評価では 50 IOPS に設定した．n は t と OTF-AST を動作させる PC の HDD から SSD への置換速度から求める．本章の実装システムの移動速度は，表 5.2 より HDD から SSD への置換速度が 2-3 秒/sub-LUN であることが分かる．よって，3 秒/sub-LUN を採用して 20 と設定する．m は IO アクセスの集中度を参考に決めることが出来る．2 章の分析結果より，平均 3–6 sub-LUN の範囲に 58% 以上の IO アクセス集中が発生していることが分かる．よって，m の設定値として 50-60 前後が候補となり，

表 5.1: OTF-AST のパラメータ

パラメータ	意味
c	連続してこの回数 WIOCA となった sub-LUN を SSD へ置換
i	LUN 全体の IOPS 閾値．この値以上となったデータに対して WIOCA 判定を行う
m	WIOCB 候補にする sub-LUN group を cut off する値
n	1 度に置換可能な最大 sub-LUN 数
o	連続してこの回数 WIOCA から外れた sub-LUN を HDD へ置換
t	各 sub-LUN ごとの IO 数を集計する間隔 (秒)

本章の評価では 60 を採用した．o は SSD 置換済み sub-LUN を HDD に戻すときに用いる閾値である．2 章で紹介したワークロードに関して，WIOCA が収束した後もしばらくの間短時間の IO アクセス集中が発生するケースがあることが分かった．この短時間の IO アクセス集中を SSD で捉えられる様に o を設定する必要があり，本章の評価では 10 に設定した．

5.1.4 SSD に置換する IO アクセス集中領域 (WIOCA) を選択するアルゴリズム

C-WIOCA の検出

図 5.1 は，LUN の大きさが 300 GB，sub-LUN の大きさが 1 GB での WIOCA の候補となる Candidate WIOCA (C-WIOCA) を検出するアルゴリズムである．C-WIOCA の検出は 4 つの Step で行われる．まず，LUN 全体の IOPS があらかじめ決めておいた閾値 i 以上かどうかを確認する．i を超えていない場合，OTF-AST はこのデータにおける C-WIOCA 検出を中止する (Step 1).

次に OTF-AST は，sub-LUN ごとに発生した IO アクセス数が多い順に並び替え，その中から上位 n までの sub-LUN を抽出する (Step 2).

次に OTF-AST は，IO アクセス数が多い順に上位 n の sub-LUN 内で隣接する sub-LUN

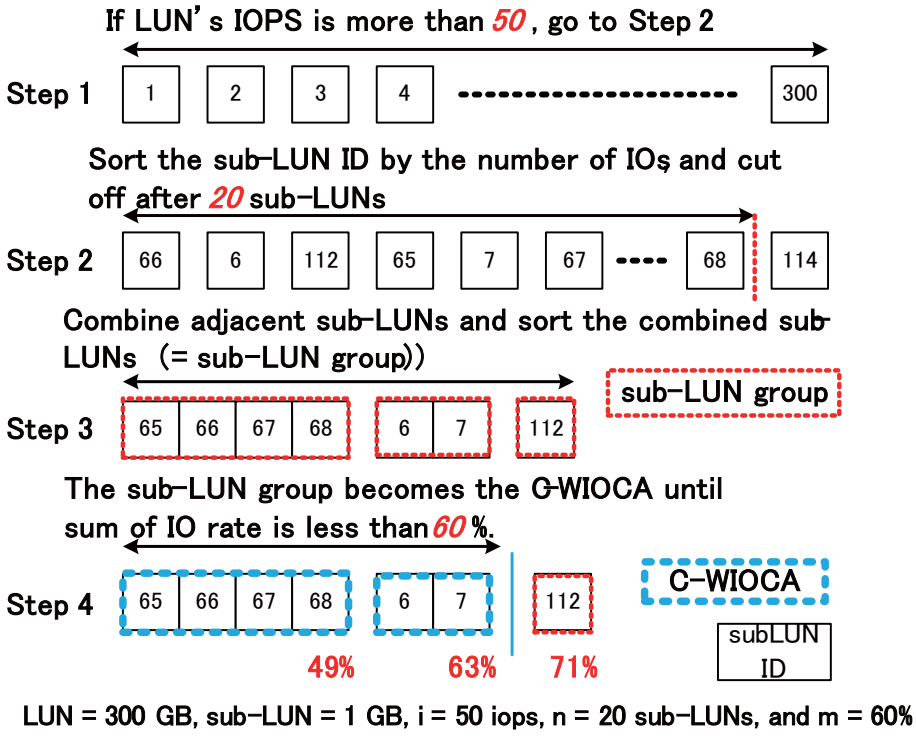

図 5.1: SSD に置換する WIOCA の選択

どうしを結合し，1 つの sub-LUN group として扱う．さらに，OTF-AST は，この sub-LUN group で IO アクセス数を合計し，IO アクセス数が多い順に並び替える (Step 3).

最後に，OTF-AST は，この sub-LUN group の IO アクセス数を累積していき，全 IO の m%を超えたところまでに入った sub-LUN group を C-WIOCA とする (Step 4).

WIOCA の検出と SSD への置換

OTF-AST は，前節で検出した C-WIOCA の継続時間を評価し，あらかじめ決められた閾値を超えて継続する C-WIOCA を WIOCA と判定し，その WIOCA を構成する sub-LUN を SSD へ置換する．具体的には，OTF-AST は $t \times c$ 秒継続した C-WIOCA のみを SSD へ置換する．tは sub-LUN ごとの IO アクセス数を集計する間隔である．cは SSD 置換判断に用いる C-WIOCA を検出したデータ数である．

図 5.2 は，c=3 のケースにおける動作例である．C-WIOCA-1 は，C-WIOCA に属する sub-

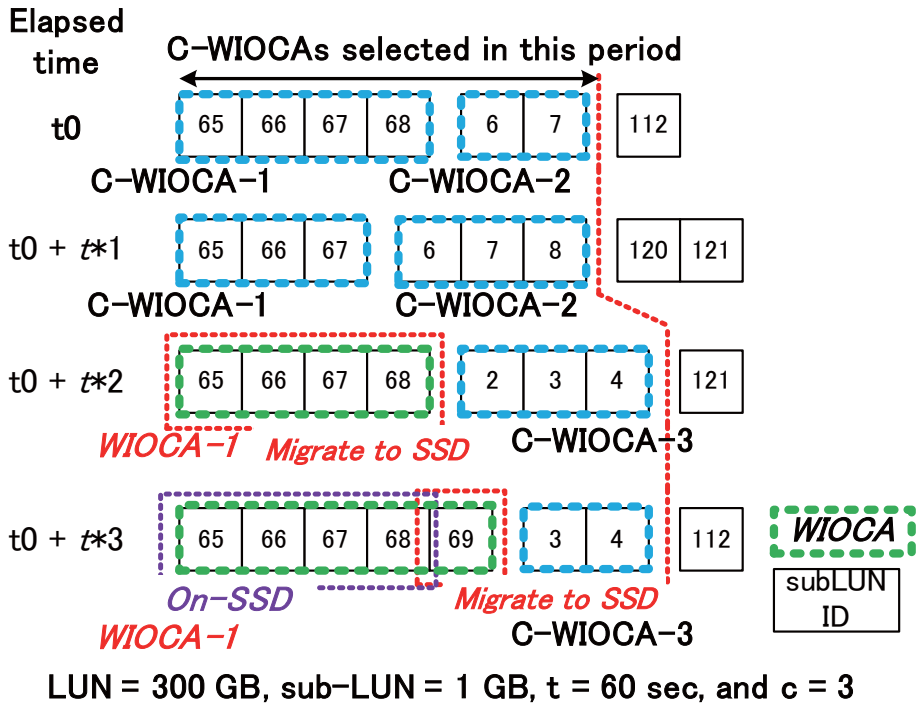

図 5.2: WIOCA の HDD から SSD への置換

LUN ID が時間経過と共に一部入れ替わるが継続して C-WIOCA となるため，$Elapsedtime = t0 + t \times 2$ のタイミングで WIOCA と判定され，SSD へ置換となる．さらに，$Elapsedtime = t0 + t \times 3$ のタイミングで，新たに WIOCA(=sub-LUN group) に加わった sub-LUN ID=69 が SSD 置換となる．

一方，C-WIOCA-2 は，2 データしか継続しないため，SSD へ置換しない．

WIOCA の終息と HDD への置換

図 5.3 は，WIOCA と判定されなくなった sub-LUN を HDD へ戻す方法説明であり，HDD 置換判定に用いるデータ数 o=*10* における動作例である．WIOCA-1 は，$Elapsedtime = t0 + t \times 1$ のタイミングで sub-LUN ID=65,66 が WIOCA の範囲から外れてしまう．そこで，OTF-AST は，Timeout と書いてある HDD への置換候補リストにこれら sub-LUN を入れる．次に OTF-AST は，WIOCA の継続判定を行う度に，HDD への置換候補リストに入った sub-LUN に関して，WIOCA に再び入っているかどうかを確認する．OTF-AST は，HDD への置換リスト

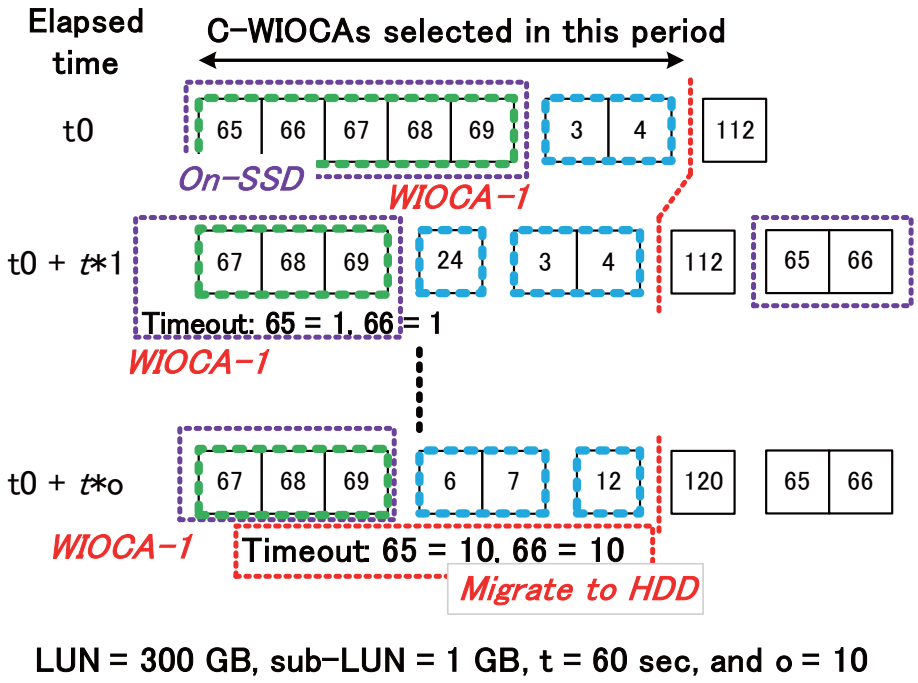

図 5.3: WIOCA の SSD から HDD への置換

に入った sub-LUN が WIOCA に再び入れば HDD への置換候補リストから削除し，そうでなければ HDD への置換候補リストに連続して入っている回数を更新する．そして，この連続数が o に達したところで，OTF-AST は対象となる sub-LUN を HDD へ置換する．図 5.3 では，$Elapsedtime = t0 + t \times o$ のタイミングで，sub-LUN=65, 66 が HDD への置換対象となる．

実際の HDD への置換は，前パラグラフで抽出した sub-LUN 情報 (sub-LUN ID=65,66) を 5.1.6 節で詳細説明する Observational migration 機能に提供し，Observational migration が HDD の負荷が低い適切なタイミングを見計らって置換を実行する．

5.1.5 WIOCA を速やかに SSD へ置換するためのシステム構成

図 5.4 が OTF-AST の構成図である．OTF-AST は Tiering manager と Tiering driver から構成され，ワークロード分析と分析結果による sub-LUN 入替えを 1 分前後の短時間で繰り返し実行できる構成となっている．Tiering manager はさらに，Data collector，Workload analyzer，sub-LUN mover，Proactive and observational migration から構成される．最初の

説明は，Tiering manager における sub-LUN 入替え方法である．まず，HDD から SSD への sub-LUN 置換を説明する．Workload analyzer は，5.1.4 節で説明した手順で SSD に置換する sub-LUN を決め，その情報を sub-LUN mover と Proactive and observational migration に通知する．sub-LUN mover は通知を受けた sub-LUN 情報を即座に Tiering driver 側に伝え，HDD から SSD への sub-LUN 置換が即座に始まる．一方 ，Proactive and observational migration は，受け取った sub-LUN 情報を一旦格納して終了し，Workload analyzer からの sub-LUN 置換指示がないタイミング時に 5.1.6 節で説明する手順を用いて HDD から SSD への置換指示を sub-LUN mover に出す.

次に SSD から HDD への sub-LUN 置換を説明する．Workload analyzer は，5.1.4 節で説明した手順で HDD に置換する sub=LUN を決め，その情報を Proactive and observational migration に通知する．Proactive and observational migration は，5.1.6 節で説明する手順を用いて SSD から HDD への置換指示を sub-LUN mover に出す.

Tiering driver は，Tiering table，Dispatcher，Kernel memory buffer，Small data sending から構成され，Workload analyzer や Proactive and observational migration で SSD への置換判定となった sub-LUN group の移動を Tiering table が行う．Dispatcher は user からの IO 要求を SSD，HDD，又は Kernel memory buffer へ振り分ける．user からの IO 要求を受け取ると，Tiering table のアクティブとなったエントリを検索し，user IO の LBA と一致している sub-LUN があれば SSD 側，そうでなければ HDD 側，移動中なら Kernel memory buffer に IO 要求を送出する．

最後に，本章の実装で用いた機材の仕様を表 5.2 にまとめた.

5.1.6 sub-LUN 置換に伴う IO アクセス性能の低下を抑える技術

Kernel memory buffer

Kernel memory buffer は，sub-LUN の HDD と SSD の間の置換時にその sub-LUN に対して発生した IO アクセス性能の低下を出来るだけ小さくするための仕組みである．置換中 sub-LUN への write 命令は，全てこの buffer に書き込んで応答し，read 命令は buffer に書き

表 5.2: 実装システムの概要

PC	Fujitsu PRIMERGY TX300S7 intel Xeon E5-2650L x2, mem= 32 GB Cent-OS 5.4 (64 bits)
HDD	HBA 4-disk RAID0 (SAS, 10,000rpm, 450GB) x4
SSD	Intel 520(240 GB, MLC)
Migration time(sec)	from HDD to SSD: 2–3 from SSD to HDD: 6–7

込みがあれば書き込まれたデータを返すことで性能向上を行う．sub-LUN 置換が終了すると，OTF-AST は Kernel memory buffer の内容を sub-LUN 置換先の HDD 又は SSD に書き込む．

図 5.5 は，sub-LUN 置換中の IO アクセスの処理方法を示している．置換中 sub-LUN に対する write 命令は，全て kernel memory buffer に書き込まれる．置換中 sub-LUN に対する read 命令は，まず kernel memory buffer に同一領域の書き込みがあると kernel memory buffer の内容が返される．kernel memory buffer に書き込みがない場合は，read 命令は sub-LUN 置換元デバイスの内容が返される．

図 5.6，5.7 は，sub-LUN の置換が完了したが，kernel memory buffer に書き込まれたデータを sub-LUN 置換先デバイスに書き込む処理が終わっていないときの，IO アクセス処理方法を示している．図 5.6 は read 命令の処理方法である．read 命令は，kernel memory buffer に対象領域の書き込みがあれば kernel memory buffer の内容が返され，そうでない場合は sub-LUN 置換先デバイスの内容が返される．図 5.7 は write 命令の処理方法である．write 命令は，対象領域が kernel memory buffer から置換先デバイスに書き込みが行われてない場合は，kernel memory buffer に書き込む．そうでない場合は，write 命令は置換先デバイスへの書き込みを待って実行される．

Proactive migration

Proactive migration は，現在 WIOCA が発生している sub-LUN group がごく近い将来どの sub-LUN に移動するのかを予測し，WIOCA が移動する前に対象となる sub-LUN を HDD か

ら SSD に置換する機能である．IO アクセスが発生する前に sub-LUN の SSD への置換が完了するため，sub-LUN 置換に伴う IO アクセス性能の低下はほとんど発生せず，さらに事前置換する sub-LUN における IO アクセス発生時から SSD ヒットとなるため，Proactive migration は大幅な性能向上が期待できる．

移動先となる sub-LUN の予測は，WIOCA が発生した sub-LUN group の移動速度と IO アクセスがほとんど無い状態から集中状態となるまでの経過時間を用いて行う．図 5.1 で説明したように，Workload analyzer は WIOCA に対応する各 sub-LUN の IO アクセス数を一定時間間隔で計算している．Workload analyzer は，この WIOCA を構成する sub-LUN group の中で最も IO アクセス数が多い sub-LUN のみを抽出し，経過時間と sub-LUN の移動距離の情報より移動速度 (*pm_speed*) を求める．さらに，この最も IO アクセス数が多い sub-LUN に関して，過去の各 sub-LUN ごとの IO アクセス数の情報を調査することで，Workload analyzer は図 5.1 の Step 2 の範囲に最初に入ってから IO アクセス数が WIOCA を構成する sub-LUN group の中で最も多くなるまでの経過時間 (*pm_growth*) を求める．Workload analyzer は，この移動速度 (*pm_speed*) と経過時間 (*pm_growth*) を WIOCA 検出に併せて更新していく．

図 5.8 は，移動速度 (*pm_speed*) と経過時間 (*pm_growth*) の算出例である．前パラグラフで説明した様に，Workload analyzer は，一定時間間隔ごとに収集する sub-LUN 単位の IO アクセス数の情報の中から，最も IO アクセス数が多い sub-LUN を抽出し，毎回その sub-LUN を記録する．そして，各回で最も IO アクセス数が多い sub-LUN の推移と経過時間を用いて，Workload analyzer は移動速度 (*pm_speed*) を求める．経過時間 (*pm_growth*) は，毎回最も IO アクセス数が多い sub-LUN に対して，過去の sub-LUN ごとの IO アクセス情報を逆引きすることで求めることが出来る．

次に sub-LUN の予測方法を説明する．図 5.2 の動作で，Workload analyzer は SSD に置換する sub-LUN ID を抽出し，sub-LUN mover に抽出した sub-LUN ID の SSD 置換を依頼する．Proactive and observational migration は，この sub-LUN ID 情報を受け取り，*timestamp* を添付して一旦保存する．その後，Workload analyzer が SSD への sub-LUN 置換を指示しないタイミングで，Proactive and observational migration は一旦保存した sub-LUN ID 情報を

取り出し，以下の計算を行うことで Proactive migration で置換する sub-LUN ID を求める．

*New sub-LUN ID = sub-LUN ID + ((current time - timestamp) + pm_growth)*pm_speed + A*

Proactive and observational migration は，sub-LUN ID を保存した時刻と現在時刻の差分に pm_growth を加えることで IO 集中が発生する前の sub-LUN ID を求める．A は補正値であり，本稿の評価では 2 に設定した．図 5.1 の Step 2 は，IO アクセス数が多い上位 n sub-LUN で cut_off している．このため，*pm_growth* は多少の IO アクセスが発生している状態からの経過時間となっており，さらに *A=2* で補正したほうが OTF-AST の性能が向上した．この新たに求めた sub-LUN ID の SSD 置換を，Proactive and observational migration が sub-LUN mover に依頼することで，Proactive migration が実現する．なお，Proactive migration による sub-LUN 置換と Workload analyzer による sub-LUN 置換が重ならないようにしている理由は，一度に大量の sub-LUN 置換を実行すると IO アクセス性能が大きく悪化するためである．

図 5.9 は，Proactive migration の実施例である．*1. Find WIOCA* は，Workload analyzer が WIOCA を認識した時刻を表している．*2. OTF-AST migration* は，*1. Find WIOCA* で検出した WIOCA が一定時間継続したため，このタイミングで WIOCA に対応する sub-LUN が SSD へ置換する．*3. proactive migration* は，このタイミングで *2. OTF-AST migration* が発生しないため，前パラグラフで説明した方法で SSD に置換する sub-LUN を抽出し，即座に SSD へ置換することを示している．

Observational migration

Observational migration は，WIOCA が終息した sub-LUN を HDD に置換する場合に，SSD から HDD へのデータ転送負荷が IO アクセス性能に影響を及ぼさないタイミングを見計らって HDD への sub-LUN 置換を行う機能である．

Workload analyzer が 5.1.4 節で説明した手順で HDD に置換する sub-LUN を抽出すると，その sub-LUN 情報を全て Proactive and Observational migration に通知する．Proactive and Observational migration は，HDD へ置換する sub-LUN 情報を受け取るとその情報を一旦内部

に設けたFIFO queueに挿入する．そして，Proactive and Observational migrationは，Tiering driverのSmall data sending機能を起動し，HDDとSSDの空き領域を利用してsub-LUNの大きさより十分に小さい大きさのSSDからHDDへのデータ転送を実行する．さらに，Proactive and Observational migrationは，データ転送と並行して，IOアクセスの応答時間を監視する．監視期間中に予め決めておいた応答時間よりIOアクセス時間が遅くなっていると，Proactive and Observational migrationはsub-LUNのHDDへの置換は中止し，遅くならない場合はFIFO queueから順番にHDDへ置換するsub-LUN情報を取り出しHDDへの置換を実行する．sub-LUNのHDDへの置換を中止した場合は，Proactive and Observational migrationは次にWIOCA判定を行うタイミングで再度試行する．

図5.10は，SSDからHDDへデータ転送を行い，その後のIOアクセス応答時間を監視する手順を示している．前パラグラフで説明した様に，Proactive and Observational migrationは，最初にsub-LUNよりも小さなサイズのデータ転送をSSDからHDDに対して行い，その後の一定期間IOアクセス応答時間の監視を行う．

本章の評価では，空き領域にデータ転送する大きさは50 MBとし，IOアクセス応答時間の閾値は80 msとした．IOアクセスの応答時間を監視する機能はTiering driverに実装し，閾値を超えたIOアクセス数をカウントするようにした．さらに，FIFO queueにHDDへの置換待ちのsub-LUNが存在する時は，Proactive and Observational migrationはWorkload analyzerのWIOCA判定タイミングと同期して再試行した．

5.2 提案方法の評価と考察

5.2.1 評価方法

本章では，提案を行ったOTF-ASTの評価方法とその結果に関して説明する．評価は，IOアクセスの平均応答時間が従来方式であるcaching方式とconventional tiering方式と比較して優位性があるのかの観点で行った．ベンチマークは，2章で分析を行ったSambaワークロードとMSR Cambridgeワークロードを再現することで行った．

表 5.3: 評価に用いたワークロード

Workload (r:w)	GB	Comment
src1_0 (56:44)	293	先頭から 1920〜2020 分を抽出
src1_1 (95:5)	293	先頭から 480〜600 分を抽出
src1_0+ (56:44)	293	6 sub-LUN で WIOCA を構成し，M 分継続 (M = 5, 10, 15, 20, 90)
src1_1+ (95:5)	293	6 sub-LUN で WIOCA を構成し，M 分継続 (M = 10, 20, 90)

評価用ワークロード の生成方法

最初に Samba ワークロードに関して説明する．2.2.1 節で説明したように，Samba ワークロードを構成するデータは，1 GB の各 sub-LUN ごとに 1 分間隔で集計した，1-GB-1-min 単位の統計情報の集合体となっている．そのため，このデータで評価を行うと 1-GB-1-min 単位でしか IO アクセスを再現出来ず，caching との比較が出来ない．そこで本章の評価では，MSR Cambridge workload [1][2] の中からファイルサーバのトレースデータであり，IO アクセス集中時の平均 IOPS が大きい src1_0 と src1_1 のデータを用いて図 2.1，2.2 の Samba ワークロードの特徴となる src1_0+，src1_1+ワークロードを生成して評価した．src1_0 と src1_1 ワークロードは，2.4.1 節で説明したように IO アクセス集中が 60-90 分連続して発生した後 IO がほとんど発生しない時間が 600 分前後続き，再び IO アクセス集中が発生する，特徴を有する．本章の評価では，連続した 1 つの IO アクセス集中を含むように src1_0 のデータの先頭から 1920–2020 分の区間と src1_1 の 480-600 分の区間を取り出し，この取り出したデータを用いて src1_0+，src1_1+ワークロードを生成した．

図 5.11 は，src1_0+を例にワークロード生成方法を説明した図である．src1_0 の IO アクセス集中は，2-3 分継続して隣接 sub-LUN に移動する．よって，本章の評価ではこの隣接 sub-LUN に移動する IO アクセス集中の継続時間を長くする変更を行った．図 5.12 が具体的な生成手順である．ます，1 分単位で sub-LUN ごとの IO アクセス数を集計し，IO アクセス数が多い順に 6 sub-LUN を選択する (1)．次に (1) で選択した sub-LUN をあらかじめ決めておいたストレージ領域に順番に置く (2)．M 分単位であらかじめ決めておいたストレージ領域を隣接する

領域に移動する (3)．なお，M の値は 5–90 分の間で数ポイント設定した．

MSR Cambridge ワークロードは，Samba ワークロード生成に用いた src1_0 と src1_1 ワークロードをそのまま用いた．表 5.3 が，評価に使用したワークロードである．なお，表 5.3 に掲載した read/write 比は，対象となるワークロードのトレースデータより read 命令数と write 命令数の比と求め，掲載したものである．

評価用ワークロードの実行方法

表 5.3 の再現は Linux btreplay command を用いて行った．MSR Cambridge ワークロードを構成するトレースデータは，Event Tracing for Windows (ETW) 形式で保存されている．そこで本章の評価では，表 5.3 のワークロードを Linux btreplay コマンドが読み込める形式に変換し，-X 1 -W 1 option 付きで実行した．

評価方法は，同一条件で実行した Caching と Conventional tiering との間で平均応答時間を比較することである．本章の評価に用いた Caching は，OSS の Caching 実装として有名な FACEBOOK FlashCache[7] である．Caching に用いる SSD 容量は，ワークロードごとに OTF-AST が消費した最大の SSD 使用量にあわせた．

Conventional tiering は，2 つの方法で比較した．最初の方法は，IO アクセス数の集計間隔を 1 分間にし，その間の IO アクセス数で毎分 sub-LUN 入替えを行う．2 章で示した IO アクセス集中の継続時間が数分から 1 時間程度であるため，この IO アクセス集中を捉える目的で，本章の評価は集計間隔を 1 分間とした．この方法で評価を行うために，本章の評価では proactive and observational migration 機能を無効にし，さらに 1 分単位で集計した sub-LUN ごとの IO アクセス数に基づいて sub-LUN 入替えを行う OTF-AST を用いた (Tiering1)．

もう 1 つは，3.2 節で説明した Conventional tiering の sub-LUN 入替え方法，である．本章の評価では，表 5.3 で示したトレースデータを用いる．そこで，この評価では再現対象となるトレースデータの直前の 120 分間を取り出し，sub-LUN 単位の合計 IO アクセス数順に 24 sub-LUN までを事前に SSD に配置した．Conventional tiering の SSD 層には全ストレージ容量の 10%前後を割り当てられることが多く，本評価の SSD 層は全ストレージ容量の約 8%に

表 5.4: 比較システムの実行条件

OTF-AST	$n = 20$, $t = 60$, $m = 60$, $c = 3$,and $o = 10$ パラメータで実行
Caching	Facebook FlashCache[7] を使用し，4-KB block size で初期化した．置換アルゴリズムは default の FIFO を用いた．SSD 容量は OTF-AST の最大使用容量にあわせた．
Tiering1	Proactive and observational migration 機能を無効にし，$n = 20$, $t = 60$, $m = 60$, $c = 1$,and $o = 1$ パラメータで OTF-AST を実行
Tiering2	replay する trace data の直前のアクセス頻度で SSD に移動する sub-LUN を決め，replay 中はその sub-LUN を SSD 上に置く．SSD 容量は 24 GB

相当する 24 sub-LUN とした．なお，src1_0+，src1_1+ワークロードは直前のトレースデータが存在しないため，本評価では src1_0，src1_1 ワークロードの直前の 120 分のトレースデータに対して前節で説明したフィルタリングを行ったトレースデータを用いて事前に sub-LUN 入替えを行った．このフィルタリングの影響で，直前のトレースデータも評価に用いるトレースデータと同一 sub-LUN へ IO アクセスを集めてしまうため，src1_0+，src1_1+の実行結果は Conventional tiering で得られる最高値に近い性能値となる．実際の運用では，事前に SSD に配置した sub-LUN に IO アクセスが集中する保証はなく，もっと低い性能値となると推定している (Tiering2)．

Proactive and observational migration 機能の効果を明確にする目的で，proactive and observational migration 機能のみを無効にした OTF-AST の評価と HDD のみを用いた場合の評価も合わせて行った．表 5.4 は各方式の詳細実行方法である．

5.2.2 評価結果

Samba で発生した WIOCA の評価

表 5.5 は，OTF-AST，Proactive and Observational migration 機能を無効にした OTF-AST (no migration)，FlashCache，Conventional Tiering (Tiering1, Tiering2 と表記)，HDD の平

表 5.5: Samba ワークロードの評価結果 (ms)

*src1_0+*の評価結果						
M	OTF-AST	OTF-AST (no migration)	FlashCache	Tiering1	Tiering2	HDD
5	26 (93 GB)	53	26	69	44	55
10	21 (50 GB)	34	29	69	39	57
15	20 (35 GB)	29	30	65	32	56
20	19 (29 GB)	25	30	66	20	57
90	14 (14 GB)	13	36	61	14	57
*src1_1+*の評価結果						
10	9.0 (72 GB)	28.8	7.5	74.0	7.7	9.4
20	6.5 (39 GB)	17.0	6.5	75.6	7.2	9.5
90	5.0 (33 GB)	6.2	5.2	71.4	2.1	9.9

均応答時間比較結果である．OTF-AST の括弧内の容量は，OTF-AST が消費した最大の SSD 容量である．OTF-AST はワークロードに含まれる WIOCA に依存して SSD の消費容量が異なるため，平均応答時間と併せて掲載した．なお，FlashCache の SSD 容量は，前節で説明したように OTF-AST が消費した最大の SSD 容量にあわせた．

src1_0+の評価結果　表 5.5 の src1_0+の各項目の結果は，OTF-AST の平均応答時間が全てのケースで他方式を上回ることを示しており，src1_0+の様なリードとライトがほぼ半々のワークロードにおける提案方法の有用性が確認できた．OTF-AST (no migration) と比較すると，M の値が小さい (=WIOCA の移動頻度が大きい) ほど効果が大きくなることが分かり，実験結果は Proactive and Observational migration の有効性を示した．一方，SSD の消費量は，OTF-AST が一時的であるが OTF-AST (no migration) を大きく上回る．図 5.13 は M=5 の時の OTF-AST と OTF-AST (no migration) の SSD 消費量の推移である．OTF-AST の SSD 消費量は最大で 93 GB に達する．これは，ワークロードでの WIOCA の発生が終息するまで Observational migration が機能して HDD への置換が進まなかったためである．しかしながら，WIOCA が終息すると，OTF-AST は 20 分程度で SSD 領域を回収する．

OTF-AST と FlashCache の評価結果を比較すると，M=5 のケースのみほぼ同じ平均応答時

間であるが，M の値が大きくなるにつれて性能差が大きくなることが分かる．これは，3.2 節で説明したように write-back 負荷の影響で FlashCache 側の性能が伸びなくなることに加えて，使用できる SSD 容量が小さくなるためである．M=5 の時は 93 GB で動作させたが，M=90 では 14 GB で動かした．なお，M=90 のケースで，OTF-AST は FlashCache と比較して約 257%平均応答時間が向上したことになる．

次は，Conventional Tiering との比較である．最初に 1 分間隔で sub-LUN 入替えを行う Tiering1 に関して議論する．表 5.5 の結果より，Tiering1 の平均応答時間は OTF-AST だけではなく OTF-AST (no migration) や HDD よりも大きくなることが分かる．この原因は，1 分単位で sub-LUN 入替えを行うため，短時間で終息する IO アクセス集中まで SSD にあげてしまい，置換に必要なコストが置換後の性能向上効果を上回ってしまうためである．図 5.14 は，OTF-AST，OTF-AST (no migration)，Tiering1，そして HDD の応答時間を Cumulative Distribution Function (CDF) にした結果である．Tiering1 は，10%tile 以降全てのポイントで OTF-AST と OTF-AST (no migration) の結果を下回っており，頻繁な sub-LUN 入替えを行うため IO アクセスを効果的に SSD に集められていないことが分かる．さらに Tiering1 は，85%tile 付近で HDD の結果を下回っており，sub-LUN 入替えに伴う IO アクセス性能の悪化が大きいことが分かる．

次に直前の 120 分のトレースデータより sub-LUN 単位の IO アクセス数を求め．IO アクセス数が多い順に 24 sub-LUN まで SSD に事前配置した Tiering2 について議論する．Tiering2 は，Tiering1 の様に HDD の平均応答時間を下回ることはないが，M の値が小さいほど平均応答時間が大きくなることが分かる．これは WIOCA の移動頻度が大きくなるほど，事前に SSD に配置した sub-LUN へ IO アクセスが集まらなくなるためである．さらに，Tiering2 は，OTF-AST の平均応答時間を下回ることはないが，それほど大きな差が付いていないことも把握できる．これは，5.2.1 節で説明したように，Conventional tiering で得られる最高値に近い性能となっているためであり，実運用でここまでの性能値となるのは稀だと判断してる．

src1_1+の評価結果　最初に OTF-AST と FlashCache の評価結果を比較する．M=10 のみ FlashCache の平均応答時間が最も低くなるが，後は OTF-AST と FlashCache の平均応答時

間はほぼ同等となる．FlashCache は，src1_1+がほぼ read 中心であるため，write-back がほとんど発生せず，src1_0+と比較して大きく平均応答時間が向上する．OTF-AST も，M が大きくなると sub-LUN 入替えに伴う IO アクセス応答時間が低下する割合が減り，FlashCache と同等の平均応答時間となった．図 5.15 は，OTF-AST と FlashCache の CDF である．M=10 では 97%tile で OTF-AST と FlashCache が逆転し，平均応答時間で比較すると FlashCache が OTF-AST より低い値となった．しかし，M=20 以降では，OTF-AST の sub-LUN 入替えが減るため，98%tile 以降での逆転となり，ほぼ同じ平均応答時間となった．

次に Conventional Tiering との比較を行う．Tiering1 に関しては，OTF-AST，OTF-AST (no migration)，HDD よりも平均応答時間が大きくなり，その理由は前節で説明した通りである．Tiering2 に関して，M=90 のみ OTF-AST や FlashCache の平均応答時間を下回ることが分かる．これは，WIOCA が隣接 sub-LUN に移動しないため，大部分の IO アクセスが SSD で処理されたためである．

MSR Cambridge で発生した WIOCA の評価

src1_0 の評価結果　表 5.7 は，OTF-AST，OTF-AST (no migration)，FlashCache，Conventional Tiering (Tiering1，Tiering2 と表記)，HDD の平均応答時間比較結果である．最初に OTF-AST と FlashCache の評価結果に関して議論する．表 5.7 より，OTF-AST が約 42%FlashCache の平均応答時間を下回ることが分かる．これは 3.2 節で議論したように，FlashCache は write-back 負荷の影響で平均応答時間が下がらないためである．

次に OTF-AST と Conventional Tiering の評価結果に関して議論する．Tiering1 は，OTF-AST だけではなく HDD よりも平均応答時間が大きくなることが分かる．この理由は，性能向上を期待できない数分前後の短時間で終息する IO アクセス集中が発生した sub-LUN まで置換を行ってしまうため，sub-LUN 置換に伴う遅延のみが増加し，結果として HDD よりも平均応答時間が増加したためである．Tiering2 は，HDD より若干平均応答時間が向上するが，OTF-AST の方がが約 202%平均応答時間が小さくなることが分かる．これは，評価に用いた表 5.3 のトレースデータとその直前の 120 分間のトレースデータに関して，IO アクセス数が

表 5.6: Observational migration 閾値と平均応答時間との関係

SSD 消費量 *(GB)*	106	99	90	75	66
IO アクセス応答時間閾値 *(ms)*	80	100	150	200	250
OTF-AST (ms)	16.8	18.3	21.5	25.8	29.5
FlashCache (ms)	23.8	26.7	25.3	26.5	25.1

多い sub-LUN が余り一致しなかったためである．

最後に，5.1.6 節で説明した Observational migration に関して，migration を中止する IO アクセス応答時間閾値を 80 ms から大きくした場合の議論を行う．表 5.6 が実験結果である．参考として，同じ SSD 容量の FlashCache のデータも添付した．表より，IO アクセス応答時間閾値を大きくすると平均応答時間は悪化するが，代わりに SSD 消費量が減少することも分かる．IO アクセス応答時間を 200 ms に設定したケースでほぼ FlashCache と同じ性能値となることも分かる．図 5.16 は経過時間と SSD 消費量との関係を表しており，IO アクセス応答時間閾値を小さくすると，運用中に IO アクセス集中が終了した sub-LUN の回収が進まないことが分かる．

src1_1 の評価結果　まず，5.1.6 節の src1_1+の平均応答時間と比較して，実験結果は src1_1 の平均応答時間が 9 ms 以上大きくなっていることを示している．これは src1_1+と比較して，src1_1 は，sub-LUN 入替えの頻度が多く src1_1+ ほど性能向上しなかったためである．図 5.17 は，OTF-AST の src1_1+ (M=10) と src1_1 の平均応答時間比較結果であり，ほぼ全てのポイントで src1_1+ (M=10) が上回ることがわかる．

FlashCache と平均応答時間を比較すると，OTF-AST が 11.2 ms ほど大きくなることがわかる．この理由は，src1_1+と比較して，src1_1 は WIOCA が移動する頻度が高くその影響で IO アクセス性能が低下するケースが多くなることと，src1_1 が read 中心であり write-back が発生しないため FlashCache 側の平均応答時間が大きく向上するためである．Conventional Tiering の結果も同様の解釈で説明できる．

表 5.7: MSR Cambridge ワークロードの評価結果 (ms)

	OTF-AST	OTF-AST (no migration)	FlashCache	Tiering1	Tiering2	HDD
src1_0	16.8 (106 GB)	45.4	23.8	61.2	50.8	53.3
src1_1	18.4 (118 GB)	56.8	7.2	85.4	7.1	8.5

評価結果の考察

5.2.2，5.2.2 節の評価結果は，read/write 混在の WIOCA が含まれる src1_0 と src1_0+ワークロードに関して，OTF-AST の優位性を示した．一方，read 中心の WIOCA が含まれる src1_1 と src1_1+ワークロードに関して，src1_1+では従来方式と差がなく src1_1 では従来方式を下回る結果となった．この理由は，read 中心で write-back が発生しないため従来方式の性能が向上することに加え src1_1 では WIOCA の移動頻度が高く OTF-AST のオーバーヘッドが大きくなるためである．以上より，少なくとも read/write が混在する WIOCA を含むワークロードにおいて，OTF-AST が従来方式を上回る効果が得られる．

次に，2 章で示した，コストパフォーマンスに関して議論する．表 5.5，5.7 の様に，OTF-AST は一時的ではあるが LUN の大きさの約 36%に達する 106 GB の容量を使ってしまう．一方，表 2.1 の様に MSR Cambridge ワークロードにおいて，IOPS が低い状態が平均 651 分発生することも分かる．これは IOPS が高い状態の約 10 倍の長さである．そこで，WIOCA が発生する時間帯が異なるワークロードを組み合わせて SSD を共有して OTF-AST を運用すれば，今回の評価より大幅にコストパフォーマンスを向上できる．例えば，9 ワークロードで運用できれば，対応する LUN の 4%程度の SSD 容量を準備すればよいことになる．

5.2.3 OTF-AST パラメータ設定に関する今後の課題

OTF-AST アルゴリズム運用に必要なパラメータは，表 5.1 に示す通りである．本章の評価では，5.1.3 節で説明した方法で設定を行い，前節までで説明したように，従来手法を上回るアクセス性能を達成することに成功した．

一方，パラメータ設定に失敗すると，OTF-AST は却ってアクセス性能を悪化させてしま

う場合もある．表 5.4 の Tiering1 は，OTF-AST でパラメータ設定を失敗した極端な例と解釈することも出来，表 5.5 や表 5.7 の結果より HDD の性能を下回ることが分かる．よって，OTF-AST を有効に機能させるには，適用するワークロードや運用に用いる PC 性能に応じて適切なパラメータを設定することが重要であることが分かる．

実用化に向けて今後取り組むべき課題は，5.1.3 節で説明した方法を突き詰めることで，OTF-AST 導入時のパラメータ値決定方法を体系化することと運用後のパラメータ値更新方法を確立することである．OTF-AST の運用を開始した後，適用するワークロード変化に併せて，OTF-AST 側のパラメータ更新が必要である．特に表 5.1 の c，m，o に関しては，5.1.3 節で決定し運用に用いる値以外に複数の候補を準備しておき，運用時にこれら候補となった複数のパラメータで運用を行った場合のシミュレーションを OTF-AST が並行して行っておく．そして，OTF-AST が定期的に最も成績がよいパラメータに変更することで，ワークロードの変化に自律的に対応出来る．

5.2.4 OTF-AST 適用が可能なワークロード

2 章の分析に用いた Samba ワークロードと MSR Cambridge ワークロードは，いずれも共有ファイルサーバのワークロードである．異なった環境で取得されたワークロードから共通の特徴が抽出されたことより，少なくとも共有ファイルサーバに関しては OTF-AST 適用が可能なワークロードが存在すると考えられる．

次に既発表の storage workload 分析論文の中から，OTF-AST 適用が可能かどうかの考察を行う．Bodik [21] らの論文は，workload spikes の調査を行っており，インターネットからアクセスされるサーバの IO アクセス集中を分析し spikes 負荷のモデル化を提示した．さらに，この spikes の分析結果として，全容量の 1%程度に全 IO アクセスの 32-88%が集まっており，数時間から数日継続することが示されている．この結果を 2 章の分析結果と比較すると，継続時間が長いこと以外は分析結果を満たすことが分かる．read 中心の web server のアクセスログであることも考慮すると，Bodik らが分析したワークロードは，WIOCA を含み，OTF-AST もしくは caching 適用が性能向上に有効である．

Kavalaekar [22] らの研究では，Windows Server 2008 で運用された 12 台のビジネスサーバの workload の分析結果が示されており，exchange server の IO アクセス集中が 15 分継続する報告が行われている．しかしながら，分析が disk 単位となっており，2 章の分析結果と比較することが出来ない．よって，Kavalaekar らの分析結果は，OTF-AST の適用可否を判断できない．

5.2.5 SSD のみで LUN を構成する場合との比較

HDD 業界動向 [25] によると，2020 年時点でも HDD の需要が存在することが述べられており，さらに Blue3 コラム [26] における 2020 年時点での HDD と SSD の GB 単価予測では，現在より差が小さくなるが SSD が HDD の GB 単価を下回ることはないと推定している．一方，世界のストレージ市場調査 [27] では，2011 年に供給したストレージデバイス供給容量の実績を元に 2017 年に必要となるストレージデバイスの必要容量の推定を行っている．この調査報告によると，必要容量は約 6 倍になると推定しており，このストレージデバイス供給容量の大幅な増加は今後も継続すると筆者は考えている．OTF-AST を用いると，2 章，5.2.2 章の分析結果より，理想的には必要容量の数%の SSD を準備することで性能向上を図ることが出来，コストパフォーマンスを向上できる．今後ユーザが使用するデータ量が増大しつづけることを鑑みると OTF-AST の様な階層型ストレージシステムは重要だと判断している．

また，SSD にランダムライトを実行し続けると，SSD から取り出せる性能が極端に悪化することが一般に知られている．Dongchul [24] らの研究によると，SSD の記憶域である cell block の erase が約 2 ms と桁違いに大きいため性能低下することが分かる．trim 命令を実行すると SSD の性能悪化は回復するが，内部のデータを全て消去してしまうため，SSD のみで LUN を構成した場合は別の LUN へのデータの退避が必要になる．一方，OTF-AST は常に SSD 領域を使っているわけではないので，SSD への sub-LUN 割り当てがないタイミングで trim を実行することで，SSD の性能悪化を起こすことなく運用を続けることが出来る．

5.3 まとめ

2 章において，主に共用ファイルサーバのデータ領域で発生する特徴的な IO アクセス集中の存在を示した．この IO アクセス集中は，非常に狭い領域に IO アクセスの集中が発生し，しかもその集中が数分から数十分間の比較的短時間で別の領域に移動する，という特徴を有する．

そこで本章では，性能向上を期待できる程度継続しまとまった IO アクセス数が発生する IO アクセス集中領域を短時間に抽出して SSD に置換を行う On-the-fly Automated Storage Tiering (OTF-AST) を提案した．OTF-AST は，IO アクセス集中領域から非常に短い時間で終息する領域と IO 数が比較的少ない領域を排除し，残った領域を SSD へ置換することで，SSD 置換時に発生する IO アクセスの遅延を上回る性能を得た．さらに，SSD 置換時に発生する IO アクセス遅延を抑える proactive and observational migration が，従来方式と比較して大幅な性能向上を実現した．

OTF-AST の評価は，実ワークロードを再現した上で，Caching 方式および Conventional Tiering 方式との間で平均応答時間を比較する方法で行った．その結果，OTF-AST は，Caching 方式に比べて 42%，Conventional Tiering 方式に比べて 202%応答時間を短縮し，OTF-AST の優位性が示された．

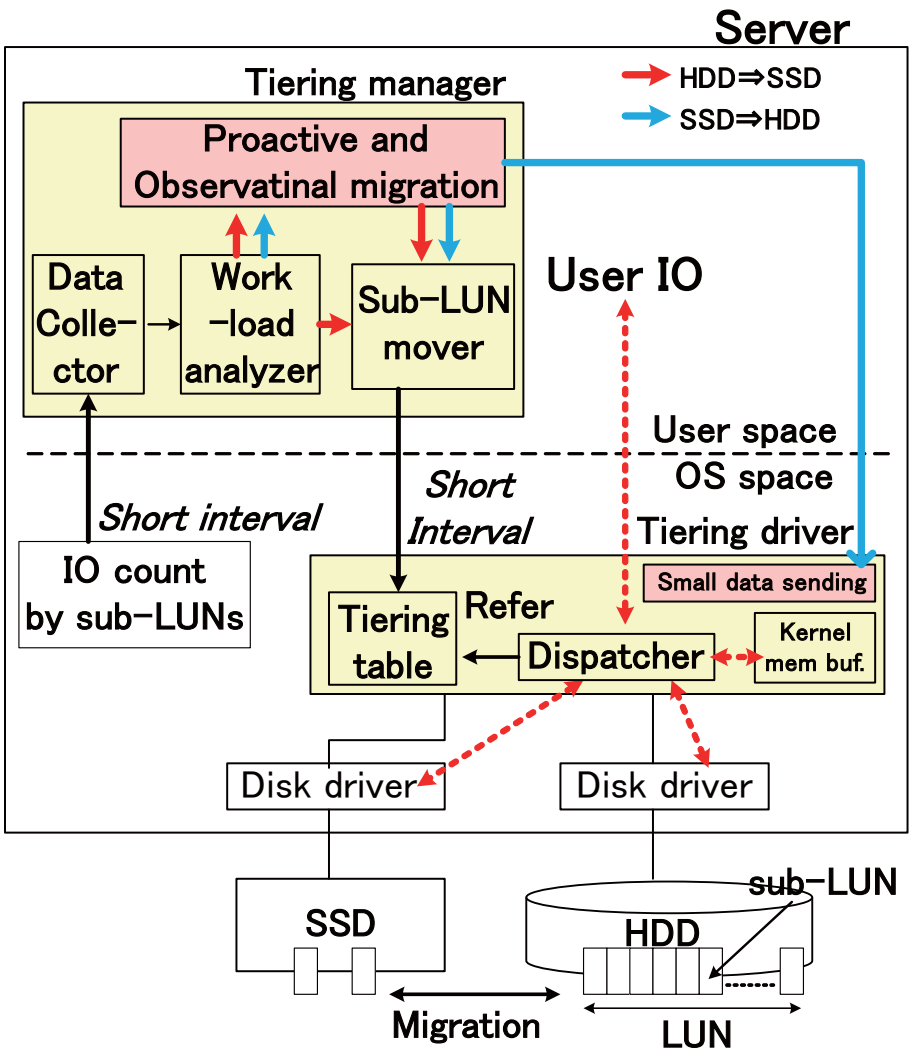

図 5.4: OTF-AST のシステム構成

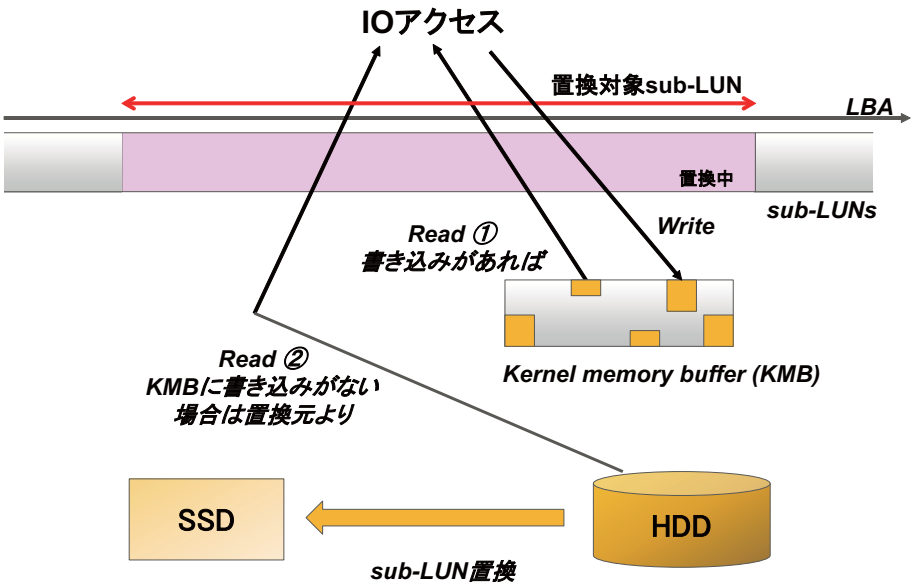

図 5.5: Kernel memory buffer の制御方法 (sub-LUN 置換中)

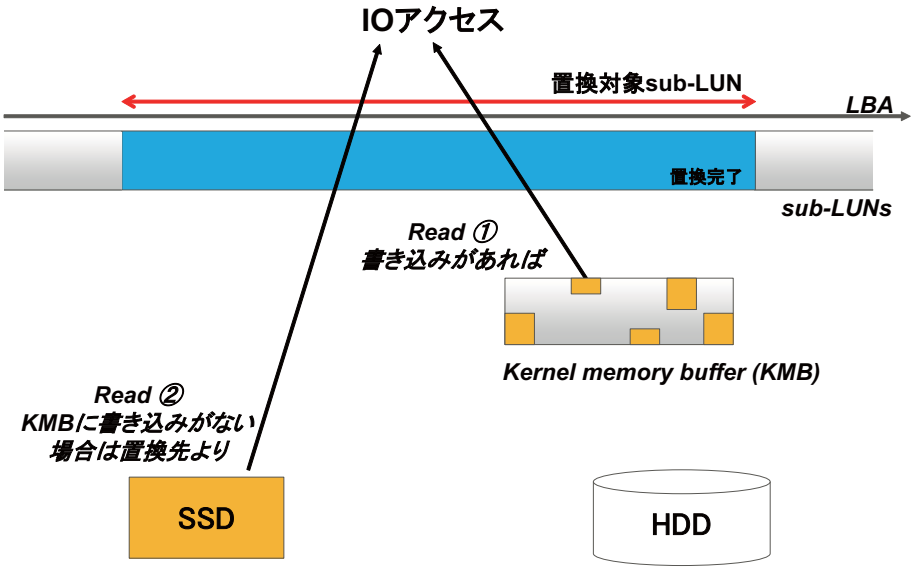

図 5.6: Kernel memory buffer の制御方法 (sub-LUN 置換完了,read)

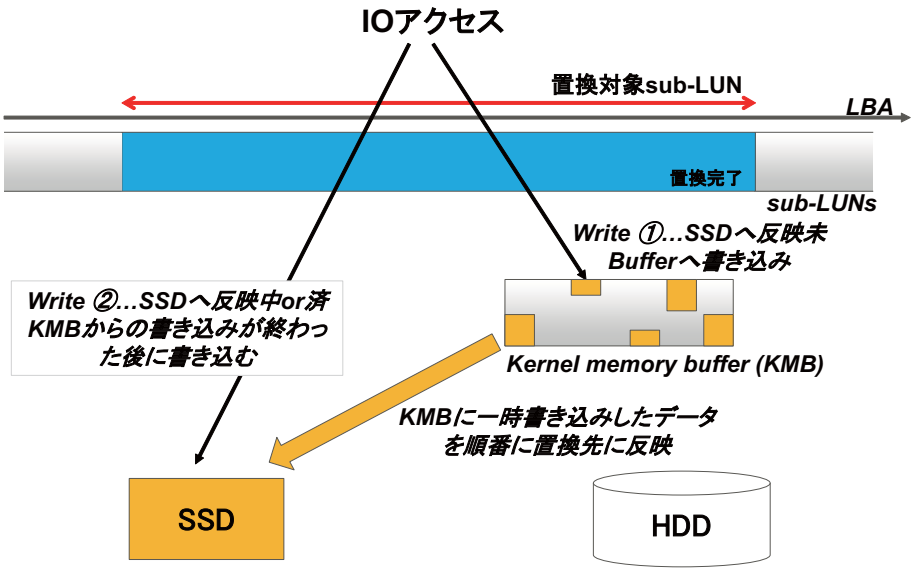

図 5.7: Kernel memory buffer の制御方法 (sub-LUN 置換完了,write)

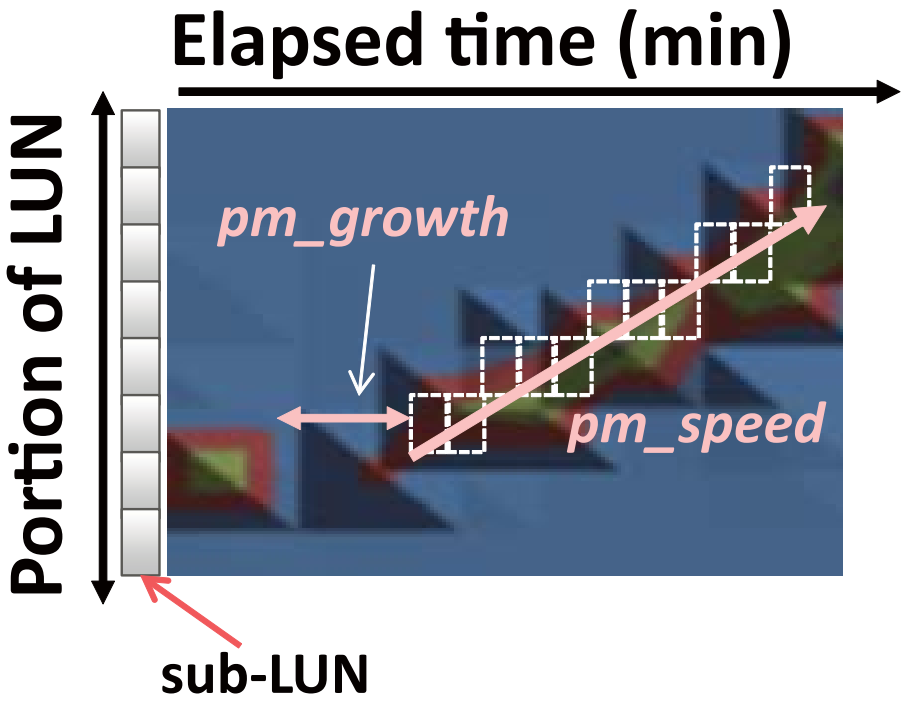

図 5.8: pm_speed, pm_growth の求めかた

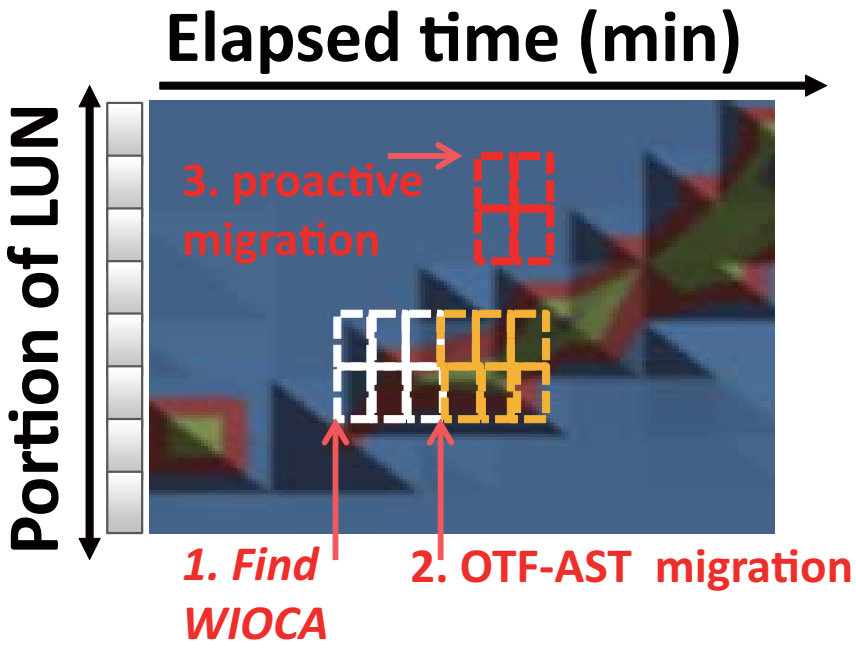

図 5.9: Proactive migration の実現例

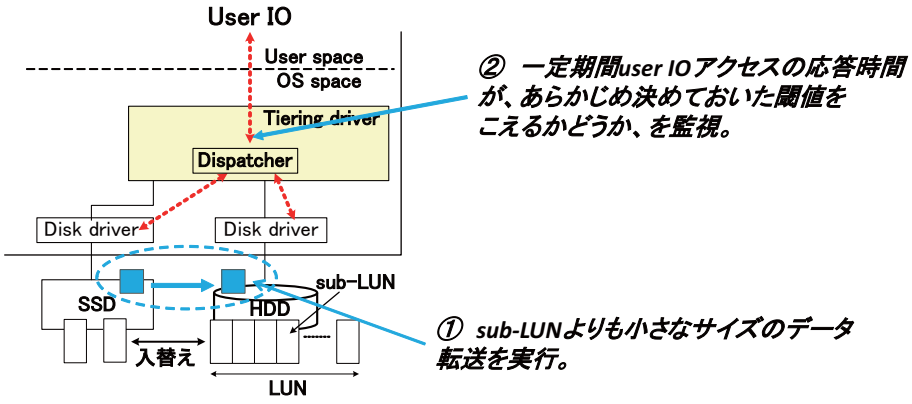

図 5.10: Observational migration の実現例

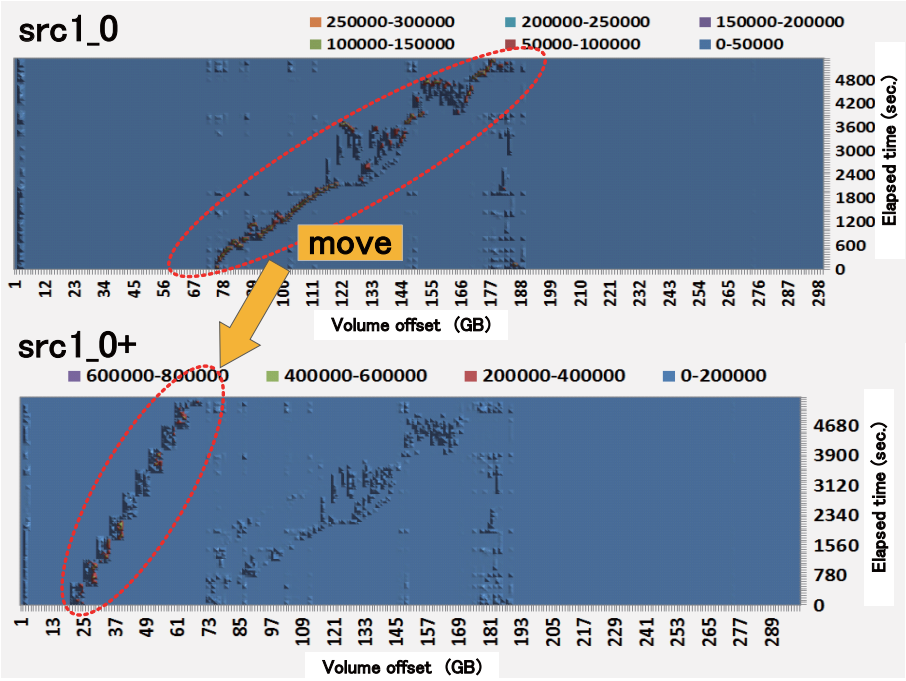

図 5.11: src1_0+ workload

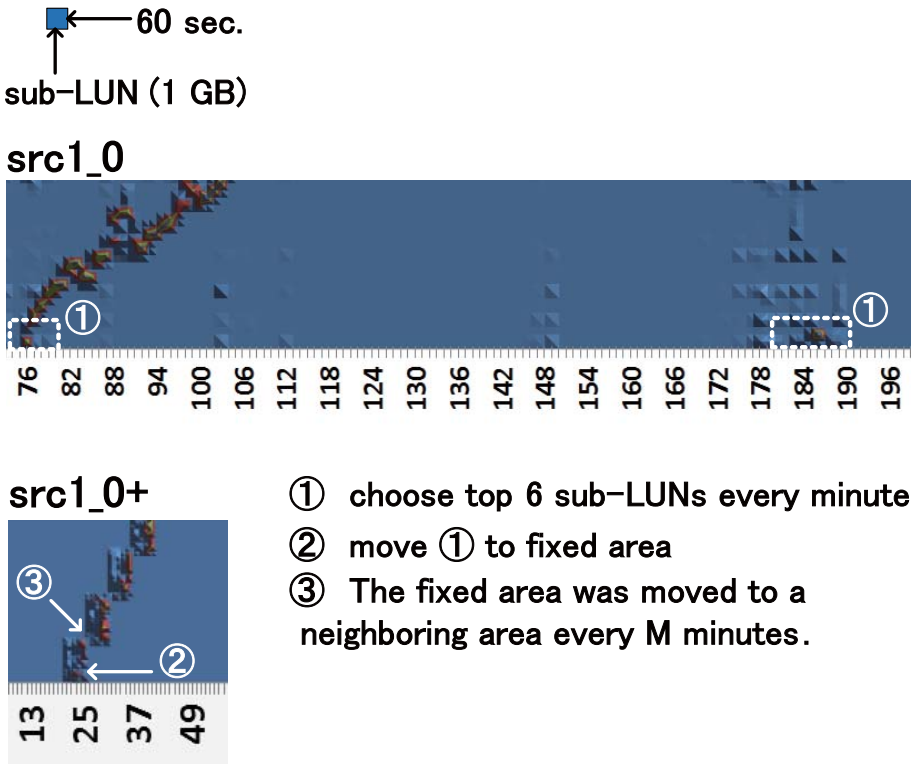

図 5.12: src1_0+ workload の生成

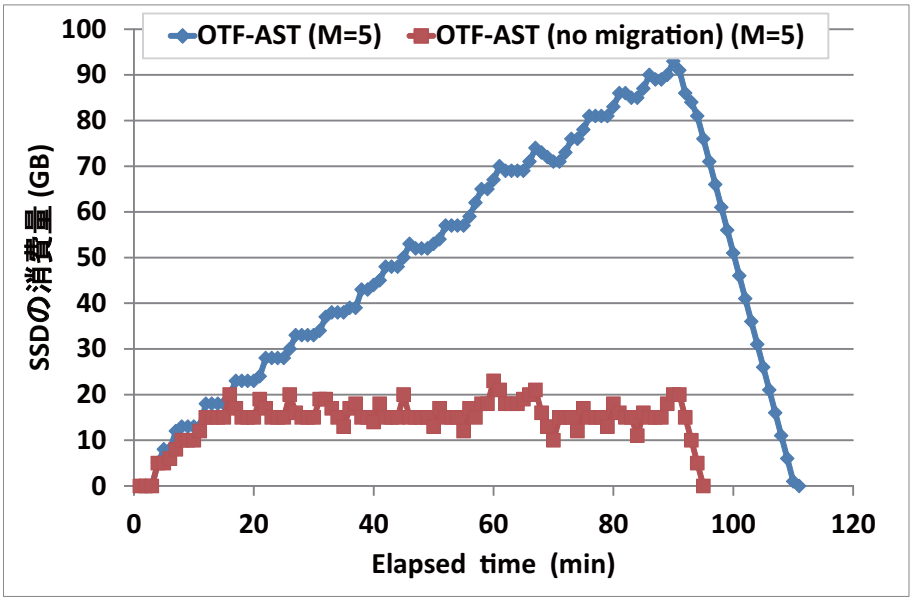

図 5.13: M=5 の時の SSD 消費量 (GB) の推移

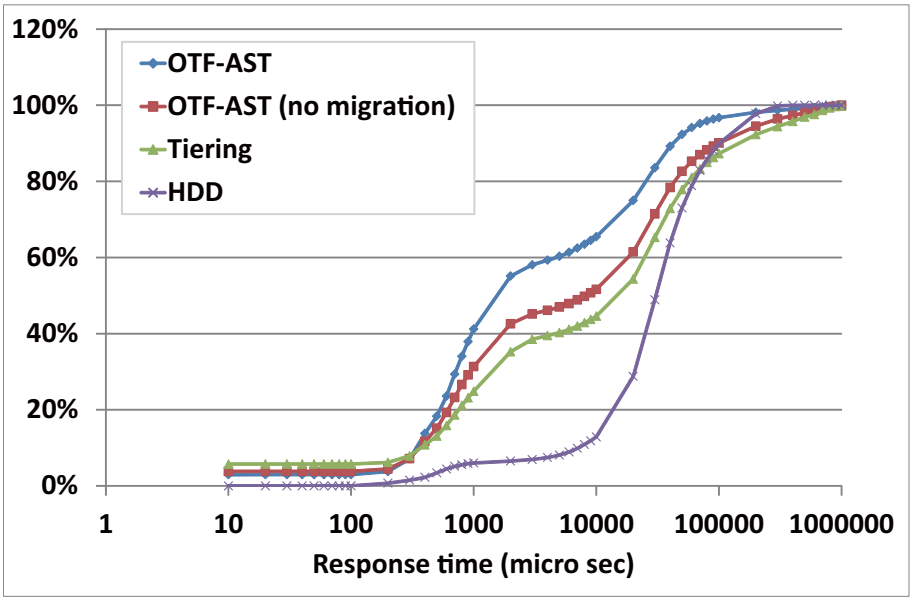

図 5.14: M=5 の時の CDF 比較 (src1_0+)

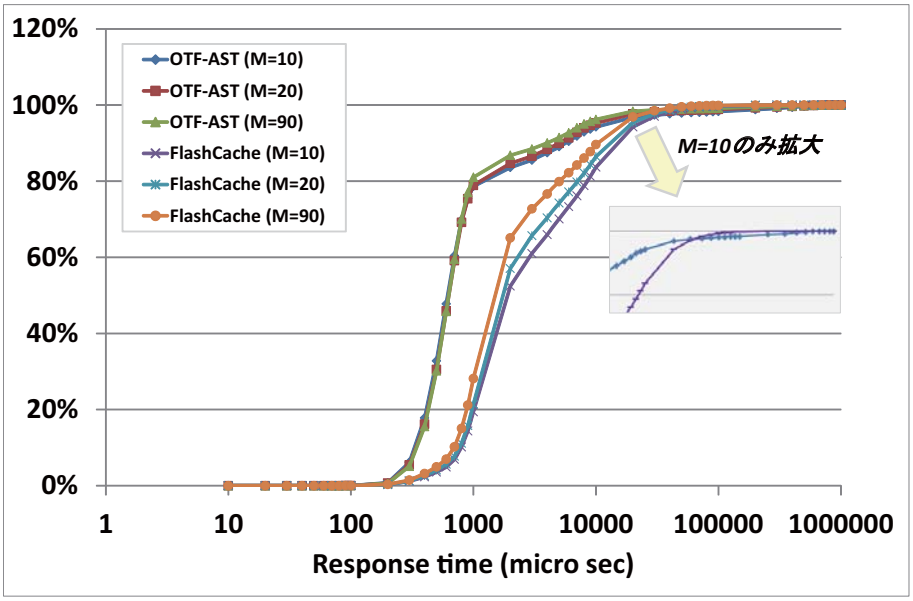

図 5.15: M=10 の時の CDF 比較 (src1_1+)

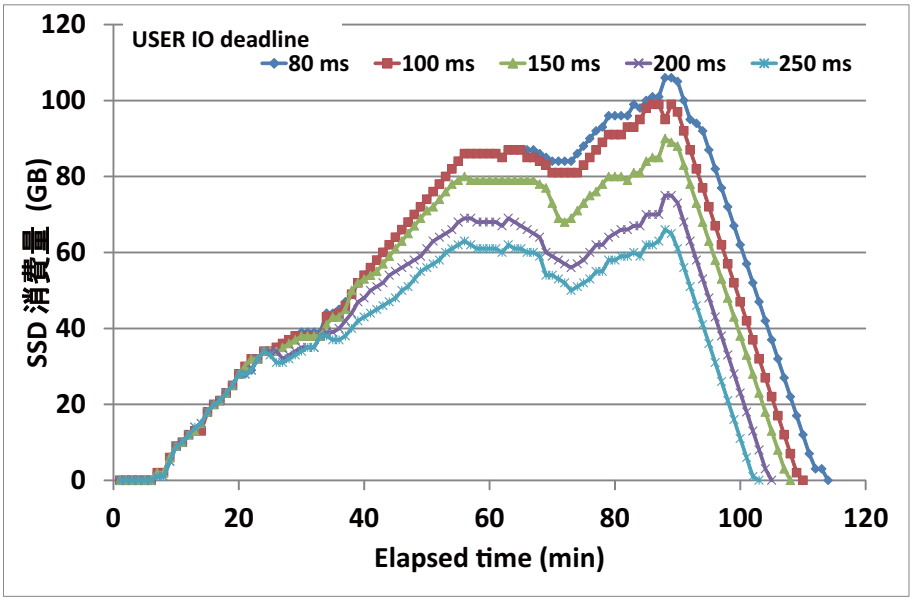

図 5.16: OTF-AST の SSD 消費量 (src1_0) (GB)

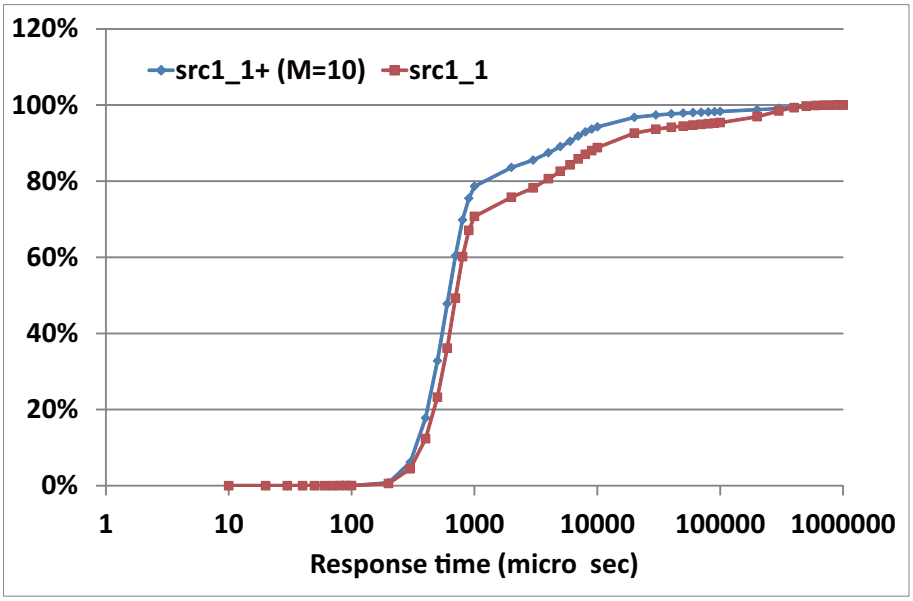

図 5.17: src1_1+(M=10) と src1_1 の OTF-AST CDF 比較

第6章　cachingの技術とIOアクセス集中領域検出技術による階層ストレージOTF-AST with cachingの提案

6.1　OTF-AST with cachingの提案

6.1.1　概要

2章のストレージワークロードの分析結果は，ストレージのごく少数のsub-LUNにIOアクセス集中が発生すること (2.4.1節) とストレージの多くのsub-LUNにIOアクセス（非IOアクセス集中）が発生すること (2.4.2節) を示している．5章で説明したOTF-ASTの成果は，SSDに置換することで性能向上が見込めるIOアクセス集中が発生したsub-LUNを選択して即座にSSDに置換することで，IOアクセス性能が向上することである．性能向上が見込めるIOアクセス集中が発生したsub-LUNは，SSD置換後のIOアクセス性能向上がSSD移動時のIOアクセス低下を上回るケースである．OTF-ASTは，sub-LUNに発生するIOアクセス集中の継続時間と全IOアクセス数に対する割合を用いて，sub-LUNの置換を判断する．しかしながらOTF-ASTは，継続時間が短くIOアクセス数が余り多くないIOアクセス集中を性能向上できない．さらにOTF-ASTは，2.4.2節で説明したIOアクセスに対しても，IOアクセスが発生したsub-LUNをSSDへ置換しないため，性能向上できない．この理由は，図2.3の結果よりIOアクセスが発生したsub-LUN数 (10〜1 IOPS, 1 IOPS未満) がIOアクセス集中が発生したsub-LUN数 (10 IOPS以上) と比較して1桁から2桁多いため，SSDへの置換を行ってもSSD置換時のIOアクセス性能低下を上回る効果を期待できないためである．

そこで本章では，OTF-ASTで性能向上出来なかったIOアクセスや継続時間が短くIOアク

表 6.1: 実装システムの概要

PC	Fujitsu PRIMERGY TX300S7, Intel Xeon E5-2650L x2, mem= 32 GB, Cent-OS 5.4 (64 bits)
HDD	HBA 4-HDD RAID0 (SAS, 10,000 rpm, 450GB) x4
SSD	Intel 520(240 GB, MLC)
Migration time (sec)	from HDD to SSD: 2–3, from SSD to HDD: 6–7

セス数が余り多くない IO アクセス集中を caching を用いて性能向上させ，さらに OTF-AST と組み合わせることで 5 章で説明した IO アクセス集中も性能向上出来る，On-the-fly automated storage tiering with caching (OTF-AST with caching) の提案を行う．OTF-AST with caching は，まず，全ての IO アクセスを caching で処理し，その中から 5 章で説明した置換条件を満たした sub-LUN を OTF-AST が制御する SSD に置換することで，全ての IO アクセスの性能向上を行うストレージシステムである．

OTF-AST with caching を効果的に動作させるための課題は，OTF-AST が管理する SSD と caching 間の sub-LUN 置換の高速化である．caching は，4 KB 前後の小さなサイズで領域管理を行っている．OTF-AST が caching に対して 1 GB 前後の大きさとなる sub-LUN 単位の IO アクセスを実行すると，caching は 4 KB 単位で SSD もしくは HDD へ IO アクセスを実行する．そのため，caching への sub-LUN 単位の IO アクセスは，HDD へのランダムアクセスを誘発し，sub-LUN 置換時間が大きく増加する．そこで，本章では，caching を構成する SSD と HDD から其々sub-LUN 置換に必要なデータを取り出し，OTF-AST が管理する SSD 上で合成する方法を提案し，sub-LUN 置換高速化を実現した．この技術は，6.1.3 節で説明する．

6.1.2 システム構成

図 6.1 は，OTF-AST with caching のシステム構成である．図 5.4 の OTF-AST の構成図との違いは，OTF-AST の HDD の位置に caching を組み込んだことである．SSD は，OTF-AST が制御する Tiering SSD と caching が制御する Cache SSD が存在する．OTF-AST の各コンポーネントの役割は，5.1.5 節の説明と同じである．表 6.1 は，OTF-AST with caching の実装に用いたシステムの概要である．

6.1.3 sub-LUN 置換高速化のための技術

概要

6.1.1 節で説明した様に，OTF-AST with caching を効果的に機能させる鍵は，sub-LUN 置換の高速化である．そこで本章では，HDD へのランダムアクセスを抑える目的で，FlashCache を構成する HDD と SSD から別々に sub-LUN 置換に必要なデータを取り出す方法を提案する．図 6.2 は，OTF-AST with caching で用いた sub-LUN 置換方法の概要である．最初に OTF-AST with caching は，置換する sub-LUN に対応する HDD 領域を Tiering SSD 領域に置換する．この置換は，sub-LUN 単位で行うため HDD へはシーケンシャルアクセスとなり，短時間で終了する．次に OTF-AST with caching は，caching SSD を管理するメタデータの検索を行い，置換する sub-LUN の LBA 範囲内で且つ dirty 状態となっている cache block を抽出して SSD へ置換する．最後に OTF-AST with caching は，kernel memory buffer に一時的に書き込んだ block を，tiering SSD に書き込む．

sub-LUN の範囲の cache block を caching SSD から tiering SSD に置換する方法

sub-LUN 置換高速化は，FACEBOOK FlashCache[7] に必要な機能を追加することで実現した．追加した機能は，図 6.2 の手順 2 に示した dirty 状態の cache block を異なった SSD にコピーする機能である．本章では，この拡張を行った FlashCache を *FlashCache+*と呼ぶ．以下のパラグラフは，*FlashCache+*の実装方法を説明する．

OTF-AST は，*FlashCache+*に対して，置換を行う sub-LUN の先頭 LBA と置換先の Tiering SSD の先頭 LBA の情報を付加して dirty 状態の cache block の置換を依頼する．FlashCache は，デフォルトで 512 cache block をひとまとめにしたセットという単位で内部の管理を行っている．そこで *FlashCache+*は，このセット単位で置換を行う sub-LUN の範囲に入り dirty 状態の cache block を検索し，該当する cache block が見つかると一旦蓄積する．そしてセット単位の検索が終了すると，*FlashCache+*は蓄積した cache block に対してまとめて Tiering SSD への置換を実行し，その後次のセットの検索を行う．

今回の *FlashCache+*の実装は，sub-LUN 置換を行う度に *FlashCache+*のメタデータを全て検索する方法である．そのため，*FlashCache+*のキャッシュ容量を大きくすると，メタデータ検索に必要な時間が増加する．今回の評価に用いた数十 GB のキャッシュ容量では，*FlashCache+*のメタデータ検索時間は，HDD の sub-LUN 置換時間と比較して十分小さく，数 ms から数十 ms のオーダーであった．しかしながら，キャッシュ容量を今回の評価より大きくした場合は，*FlashCache+*のメタデータ検索時間も増加し sub-LUN 置換時間を大きくする要因となる可能性もある．その様な場合，*FlashCache+*のメタデータ検索時間を短縮する実装の追加が必要になる．具体的な実現方法は，dirty となった cache block のみをたどれる link を加える，もしくは置換予定の sub-LUN を予め登録しておきその sub-LUN の範囲を管理する cache block をたどれる link を加える，なとが考えられる．

6.2 提案方法の評価と考察

6.2.1 評価方法

評価は，5.2.1 節と同じく，Samba ワークロードと MSR Cambridge ワークロードを構成するトレースデータを Linux btreplay コマンドで再現することで行った．表 6.2 は，本章の評価に用いたワークロードである．本章では，5.2.1 節と同じく，src1_0, src1_1 のトレースデータの中から IO アクセス集中が継続する箇所が含まれる様に，データを取り出した．さらに，Samba ワークロードは，10 分継続のケースのみ評価した．

提案方式の評価は，ワークロード再現時の平均応答時間を比較することである．比較対象は，FlashCache(Caching) と 5 章で紹介した OTF-AST である．

6.2.2 評価結果

MSR Cambridge ワークロード

表 6.3 は，5.1.6 節で説明した Observational migration の閾値を 80 ms に設定したときの実験結果である．IO アクセスの応答時間が 80 ms を上回ると，OTF-AST with caching は HDD へ

表 6.2: 評価に用いたワークロード

Workload (r:w)	GB	Comment
src1_0 (56:44)	293	先頭から 1920〜2020 分を抽出
src1_1 (95:5)	293	先頭から 480〜600 分を抽出
src1_0+ (56:44)	293	6 sub-LUN で WIOCA を構成し，10 分継続
src1_1+ (95:5)	293	6 sub-LUN で WIOCA を構成し，10 分継続

表 6.3: MSR Cambridge の平均応答時間 (ms)

Workload	OTF-AST with caching	OTF-AST	FlashCache	HDD
src1_0	11.2	16.8	23.8	53.3
src1_1	18.4	18.4	7.3	8.5

の sub-LUN 置換を中止する．最初に src1_0 の結果に関して議論する．OTF-AST with caching は従来手法より応答時間が小さくなっており，FlashCache の 42%，OTF-AST の 59%の平均応答時間である．図 6.3 は，3 方式の応答時間を CDF(Cumulative Distrbution Function) にした結果であり，OTF-AST の応答時間が常に小さい．

次に src1_1 の結果に関して議論する．FlashCache の平均応答時間が最も小さくなっており，さらに OTF-AST with caching と OTF-AST の平均応答時間が同じ値である．OTF-AST with caching と OTF-AST は，SSD への sub-LUN 置換に伴う IO アクセス性能低下が発生し，FlashCache より平均応答時間が大きくなった．図 6.4 は，3 方式の応答時間を CDF(Cumulative Distrbution Function) にした結果であり，90%tile 以降で FlashCacheg の応答時間が OTF-AST with caching を上回る．

次に，Observational migration の閾値を 80 ms より大きくした時の実験結果に関して説明する．表 6.4 が，実験結果である．*USER IO deadline* は，5.1.6 節で説明した Observational migration の閾値である．*SSD* 消費量は，OTF-AST with caching における最大の SSD 消費量である．OTF-AST with caching の caching の容量は，常に 16 GB である．しかし，OTF-AST はワークロード 再現中に発生する WIOCA の領域に従って SSD 容量を割り当てるため，OTF-AST with caching の最大の SSD 消費量は実行条件ごとに異なった値となる．FlashCache

表 6.4: User IO deadline を変化させた時の平均応答時間 (ms)

workload=src1_0					
User IO deadline (ms)	80	100	150	200	250
SSD 消費量 (GB)	118	116	98	87	82
OTF-AST with caching (ms)	11.2	13.0	18.0	16.5	20.8
OTF-AST (ms)	16.8	18.3	21.5	25.8	29.5
FACEBOOK FlashCache (ms)	22.4	21.6	23.3	22.0	22.5
workload=src1_1					
User IO deadline (ms)	80	100	150	200	250
SSD 消費量 (GB)	136	129	119	107	98
OTF-AST with caching (ms)	18.4	18.4	21.0	25.8	27.7
OTF-AST (ms)	18.4	19.8	21.0	22.0	28.8
FACEBOOK FlashCache (ms)	7.3	7.4	6.9	7.0	7.0

の SSD 容量は，この OTF-AST with caching の最大の SSD 消費量に合わせて設定した．

表 6.4 の上半分は，src1_0 の実験結果である．OTF-AST with caching の平均応答時間は，常に FlashCache を下回る．*USER IO deadline* が 80 ms の時に，OTF-AST with caching の平均応答時間は FlashCache より 113%高速である．さらに，OTF-AST と比較しても，OTF-AST with caching は 50%高速である．一方，FlashCache の平均応答時間は，SSD 容量が増減しても，ほとんど変化しない．この理由は，3.2 節で説明した様に，write-back 処理がボトルネックとなるためである．

図 6.5 は，応答時間の cumulative distribution function (CDF) である．*SSD* 消費量が 118 GB の時，OTF-AST with caching は FlashCache より常に応答時間が短い．しかし，*SSD* 消費量が 82 GB の時，sub-LUN 移動に伴う遅延が OTF-AST with caching の遅延をもたらし，90%タイル付近以降は FlashCache の応答時間が短くなる．その結果，OTF-AST with caching と FlashCache の平均応答時間は，ほぼ同じ値となった．

図 6.6 は，src1_0 の時間経過ごとの OTF-AST with caching *SSD* 消費量の推移であり，*USER IO deadline* を 80 ms から 250 ms まで変化させて実験を行った．この *SSD* 消費量は caching に割り当てた 16 GB を除いた値である．実験結果は，*USER IO deadline* を大きくすると *SSD*

表 6.5: Samba の平均応答時間 (ms)

Workload	OTF-AST with caching	OTF-AST	FlashCache	HDD
src1_0+	10.4	21.2	29.1	56.9
src1_1+	7.0	9.0	7.5	9.4

消費量が減少することを示している．一方で，表 6.4 より *SSD* 消費量が減ると IO アクセスの平均応答時間は増加し，両者はトレードオフの関係にあることが分かる．さらに，IO アクセス集中終了後，*SSD* 消費量は約 20 分で 0 になることが分かる．この事実は，IO アクセス集中終了後に数十分程度 IO アクセスが少ない時間帯が存在すると，次の IO アクセス集中に備えることが可能であることを意味する．表 2.1 の結果より，分析を行った MSR Cambridge ワークロードは，平均で IO アクセス集中が 60 分継続した後，IO アクセスが低い状態が 651 分継続する．よって，MSR Cambrdge ワークロードを前提にすると，OTF-AST with caching は，IO アクセス集中が発生する時間帯が重ならないワークロードを選択できれば，Tiering SSD を複数のワークロードで共有でき，より少ない SSD 容量での運用が可能になる．

表 6.4 の下半分は，src1_1 の実験結果である．FlashCache の平均応答時間は，常に OTF-AST with caching より小さい．この理由は，src1_1 ワークロードが read 中心であり，FlashCache 上で src1_1 を再現しても，write-back がほどんと発生しないためである．さらに，OTF-AST with caching と OTF-AST の平均応答時間がほとんど同じであることもわかる．この理由は，両者とも sub-LUN 移動に伴うオーバーヘッドがボトルネックとなっているためである．

Samba ワークロード

表 6.5 は，5.1.6 節で説明した Observational migration の閾値を 80 ms に設定したときの実験結果である．最初に src1_0+の結果に関して議論する．OTF-AST with caching は従来手法より応答時間が小さくなっており，FlashCache の 36%，OTF-AST の 49%の平均応答時間である．

次に src1_1+の結果に関して議論する．OTF-AST with caching は従来手法より応答時間が

小さくなっており，FlashCache の 93%，OTF-AST の 77%の平均応答時間である．同じ read 中心の src1_1 と比較して src1_1+で OTF-AST with caching が効果的となる理由は，IO アクセス集中の継続時間の違いである．src1_1+の継続時間は 10 分であり，sub-LUN 移動回数が src1_1 の 1/3 以下に減少する．このため，sub-LUN 移動に伴う IO アクセス性能が低下する割合も減り，OTF-AST with caching は従来方式より平均応答時間が下回った．

6.3 まとめ

本章では，OTF-AST で性能向上が可能な IO アクセス集中に加えて，OTF-AST では性能向上できない継続時間が短い IO アクセス集中 (2.4.1 節) や IO アクセス集中に届かなかった IO アクセス (2.4.2 節) の両方の性能向上が可能な，On-the-fly automated storage tiering with caching (OTF-AST with caching) の提案を行った．

OTF-AST と caching を協調して動作させる上で重要な事項は，OTF-AST が管理する Tiering SSD と caching の間の sub-LUN 置換を高速に行うことである．そこで本章では，caching を構成する SSD と HDD から別々に置換が必要なデータを抽出して置換を行う提案を行い，実装した．

評価は，MSR Cambridge ワークロードを再現したときの平均応答時間を比較することで行った．比較対象は，FACEBOOK FlashCache と OTF-AST である．その結果，read と write が混在するワークロードにおいて，OTF-AST with caching は FlashCache より 113%平均応答時間が短くなることが分かった．OTF-AST と比較したケースでも，OTF-AST with caching は 50%平均応答時間が短くなることが分かった．read 中心のワークロードでも，IO アクセス集中の継続時間が 10 分以上のケースで，OTF-AST with caching が従来方式の平均応答時間を下回った．

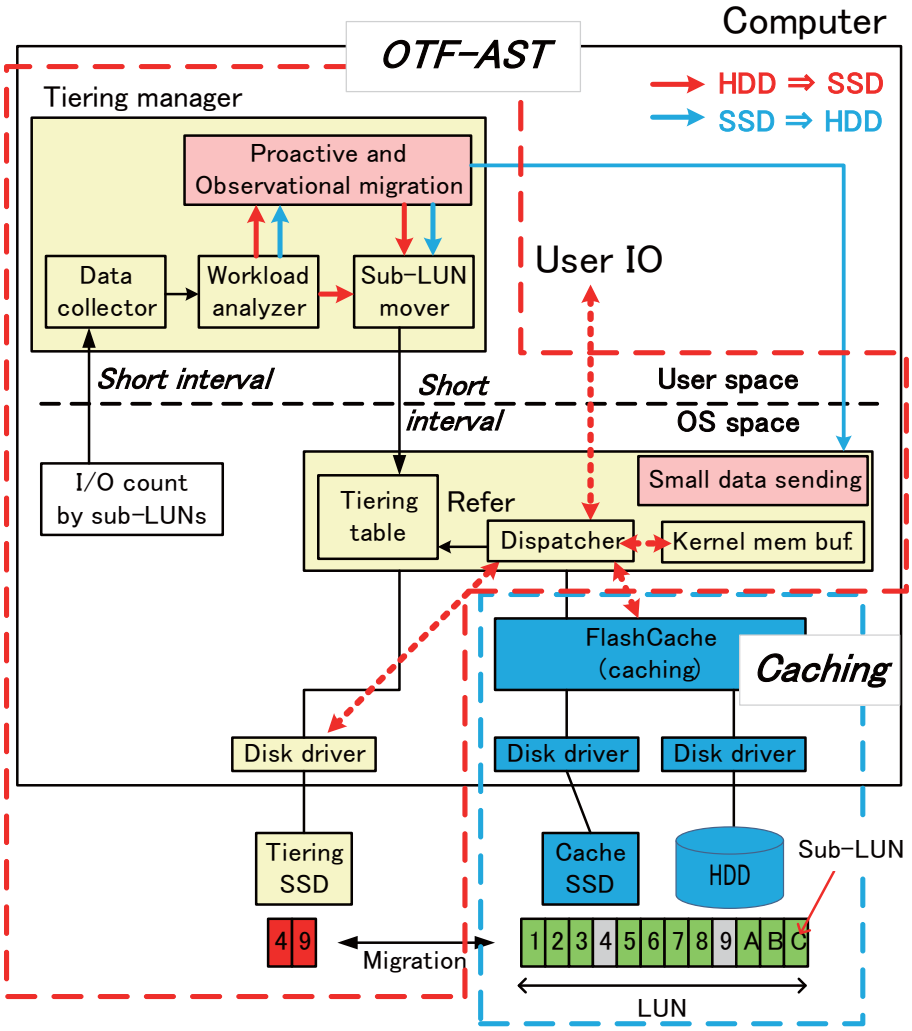

図 6.1: OTF-AST with caching のシステム構成

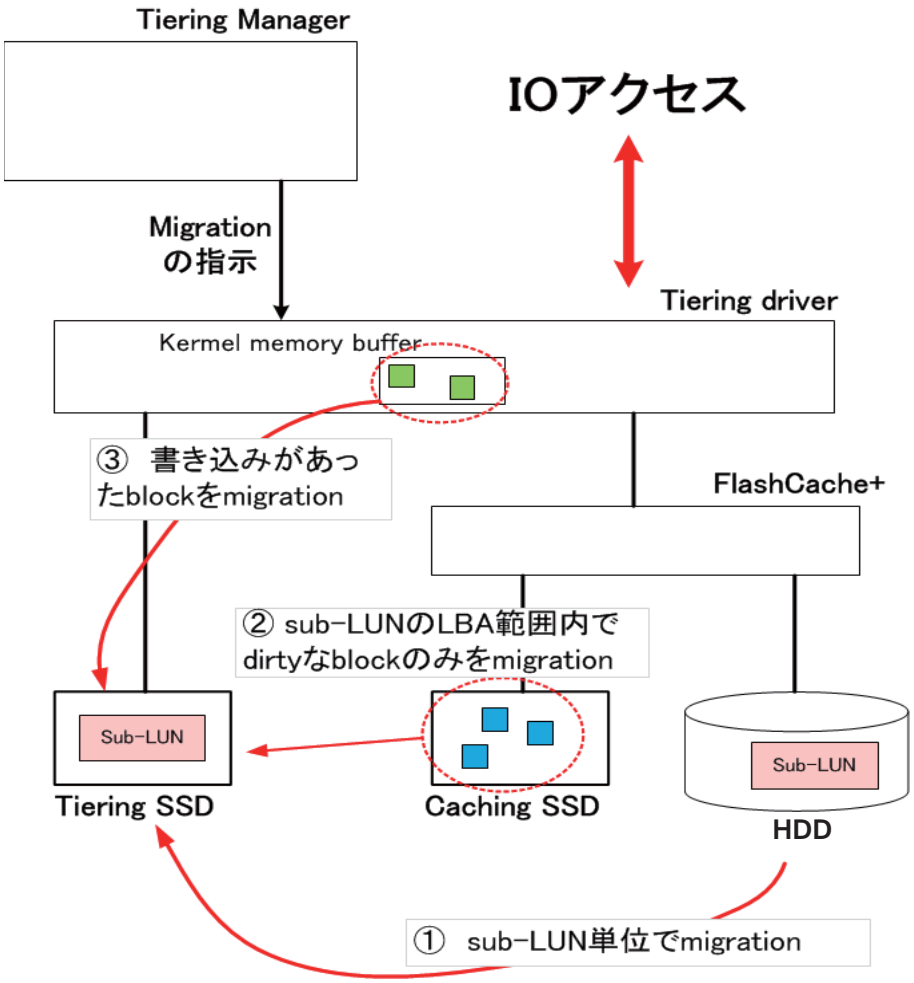

図 6.2: sub-LUN 置換方法の概要

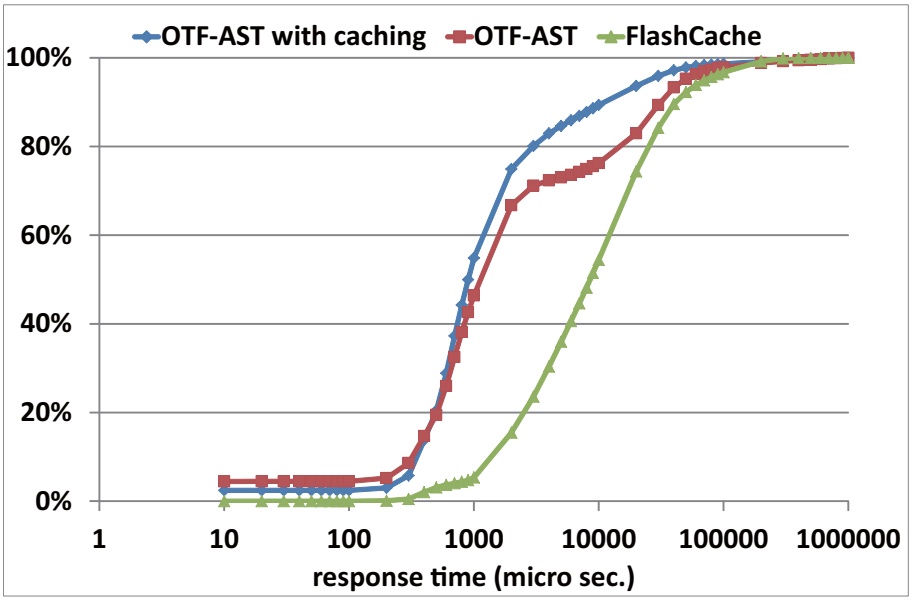

図 6.3: CDF of response time (src1_0) (1)

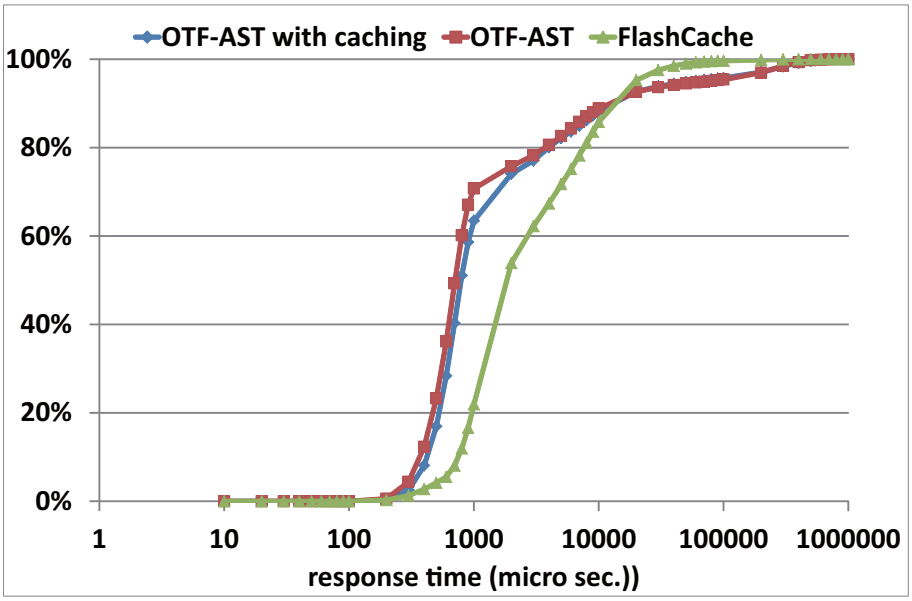

図 6.4: CDF of response time (src1_1)

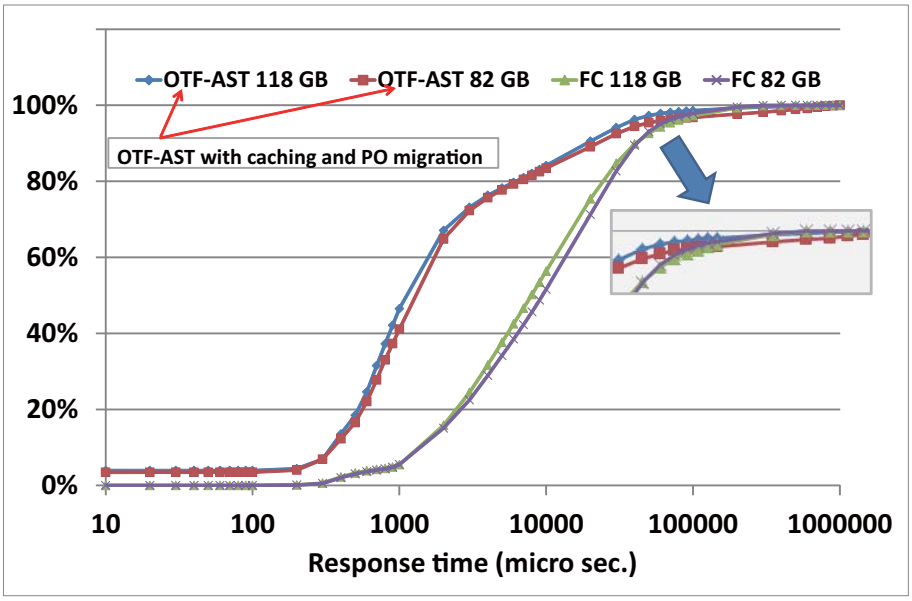

図 6.5: CDF of response time (src1_0) (2)

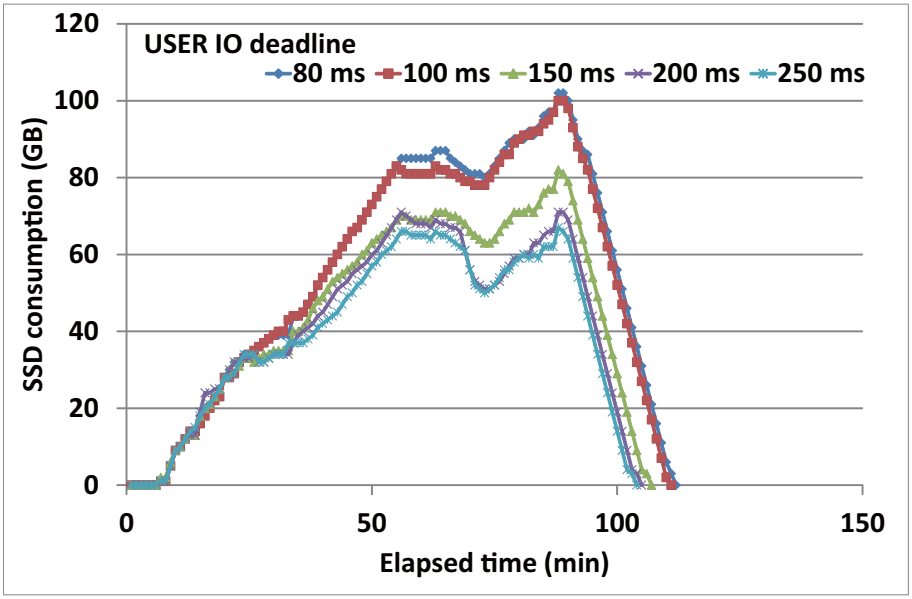

図 6.6: OTF-AST with caching の SSD 消費量の推移 (src1_0)

第7章　結論

7.1　本研究のまとめ

本研究では，まず，共有ファイルサーバとして運用されている複数のストレージシステムから採取したワークロードデータの分析を行い，IO アクセス集中が発生する領域の特徴抽出に成功した．IO アクセス集中が発生する領域は，全領域の高々数%程度であり，この範囲に全 IO アクセスの 58%以上が集まる．この領域の 66%以上は任意の領域であり，IO アクセス集中の継続時間が平均 60 分である．さらに本研究では，IO アクセス集中に達しなかった非 IO アクセス集中領域の特徴抽出も行った．非 IO アクセス集中領域は，全領域の 13-99%を占め，この範囲に全 IO アクセスの最大で 40%が集まる．

この分析結果より，本研究で最初に取り組む課題は，IO アクセス集中が発生する領域の IO アクセス性能の向上である．IO アクセス集中はごく狭い領域に発生するので，本研究では IO アクセス集中が発生した領域を抽出して即座に SSD へ移動する方法構築に取り組んだ．次の課題は，非 IO アクセス集中領域の IO アクセス性能の向上である．非 IO アクセス集中領域は，IO アクセス集中領域より範囲が広く，少ない IO アクセス数となっている．本研究では，この特徴に効果的であり，IO アクセス集中で用いる方法と組み合わせての動作が可能な方法構築に取り組んだ．

最初の提案は，HDD 過負荷が原因でアプリケーションが求める性能を提供出来なくなった時に HDD の iops と IO busy 率の相関係数を用いて，HDD の過負荷状態を解消するために HDD より IOPS が 2 桁程度大きい SSD へ置換が必要な領域を求め，即座にその領域を置換する方法である．しかしながら，HDD が過負荷状態となりその状態がしばらく続かないと SSD への置換が始まらないため効果が得られるまで時間がかかることと，隣接領域に置換しながら

IO アクセス集中が継続するパターンで効果が薄いことが，本提案の課題である．

そこで，2 番目の提案は，ストレージの部分領域ごとの IO アクセス数の変化を常時監視することで，IO アクセス集中を迅速に把握し，SSD 移動後も IO アクセス集中が継続する領域に絞って SSD に置換を行う方法である．この提案は，IO アクセス集中を短時間で抽出して SSD に置換を行うため，短時間で性能向上する．さらに，隣接領域に置換しながら IO アクセス集中が継続するパターンに関して，本提案は，IO アクセス集中の移動速度と IO アクセス集中になるまでの成長速度を用いて，IO アクセス数がまだ少ない時に IO アクセス集中となる領域を SSD へ置換することで，大幅な性能向上を達成した．IO アクセス集中領域を抽出して即座に SSD に置換する課題は，2 番目の方法を用いることで，解決することが出来た．

3 番目は，2 番目の提案の成果に非 IO アクセス集中領域の IO アクセス性能を向上させる方法を追加し，さらに性能向上を行う提案である．非 IO アクセス集中領域は，IO アクセス集中領域と比較して，IO アクセスが発生する領域の範囲が広く IO アクセス数が小さい．そこで本研究では，非 IO アクセス集中領域を SSD へ置換するのではなく，Caching を用いることにした．さらに，この Caching に 2 番目の提案を組み合わせることが 3 番目の提案である．この提案において，Caching と 2 番目の提案で用いる SSD 間の領域置換の高速化が，性能向上の鍵となる．そこで本提案では，Caching を構成する HDD と SSD から領域置換に必要なデータを個別に取り出して 2 番目の提案で用いる SSD へ転送する方法を用いた．この方法は，Caching の HDD におけるランダムアクセス発生を防ぐ効果があり，置換領域の高速化を実現した．本提案は，2 番目の提案よりさらに性能向上し，同じ条件で動作させた従来方式より優位であることを確認した．

本研究は，ストレージのワークロード分析から特徴抽出した IO アクセス集中領域と非 IO アクセス集中領域を対象にし，SSD と HDD を組み合わせた階層型ストレージシステムを用いて，コストパフォーマンスを向上する問題に取り組んだ．その結果，3 番目の提案方法は，IO アクセス集中領域と非 IO アクセス集中領域の両方に発生した IO アクセスの性能向上が可能となり，従来方式と比較してコストパフォーマンスを向上出来ることを確認した．本研究で明確にした IO アクセス集中領域の特徴は，ある程度汎用的な現象である，と筆者は考えてい

る．その根拠は，今回分析を行った共用ファイルサーバ以外に，web サーバのアクセス集中を分析した論文で同様な特徴が見出せることが上げられる．さらに，現在筆者らが分析を進めている進めている VDI(Virtual Desk Infrastructure) やメールサーバで用いるストレージのワークロードも，同様な特徴を有している．

7.2 今後の課題

2 番目の提案は，SSD 置換後も IO アクセス集中の継続が期待できる領域の抽出を行う．この抽出を行う課程で，ワークロードの特徴に依存するパラメータ設定が必要になる．パラメータの初期値は，運用を開始する直前のワークロードを分析することで求めることが出来る．しかし，時間経過と共にワークロードの特徴が変化する可能性があり，この変化に応じてパラメータの更新も必要になる．このパラメータの更新方法の確立が，今後の課題である．

関連図書

[1] Dushyanth Narayanan, Austin Donnelly, and Antony Rowstron, 'Write Off-Loading: Practical Power Management for Enterprise Storage' in Proc. of 6th USENIX Conf. on File and Storage Tech., 2008.

[2] http://iotta.snia.org/traces/388

[3] Jorge Guerra,Himabinde Pucha, Joseph Glider, Wendy Belluomini, Raju Rangaswami, 'Cost Effective Storage using Extent Based Dynamic Tiering,'in Proc. of the 9th USENIX Conf. on File and Storage Tech., 2011.

[4] 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 the 25th ACM Conf. on International Conf. on Supercomputing (ICS), 2011

[5] Kazuichi Oe, Takeo Honda, and Motoyuki Kawaba, 'Samba workload analysis and consideration for hybrid storage system' in IPSJ SIGOS, Tokyo, Japan, Dec 2012 (in Japanese).

[6] Satoshi Iwata, Kazuichi Oe, Takeo Honda, and Motoyuki Kawaba, 'On-the-fly Automated Storage Tiering (poster)' in Proc. of the 12th USENIX Conf. on File and Storage Tech, 2014.

[7] Facebook FLACHCACHE

https://github.com/facebook/flashcache

[8] Fusion IO Directcache

http://www.fusionio.com/data-sheets/directcache/

[9] Virident EnhanceIO

http://www.virident.com/products/enhanceio-ssd-cache-software/

[10] DATAPLEX

http://nvelo.com/products.html

[11] EMC White Paper, EMC FAST VP for Unified Storage Systems A Detailed Review, March 2011

[12] IBM Easy Tier

http://www-03.ibm.com/systems/data/flash/hk/storwize/challenges/riddle/IT_Riddle1.pdf

[13] FUJITSU Automated Storage Tiering: available from

http://storage-system.fujitsu.com/jp/products/diskarray/feature/i03/

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

[15] Kazuichi Oe, Kazutaka Ogihara, Yasuo Noguchi and Toshihiro Ozawa, 'Proposal for a hierarchical storage system that can move a spike to high-speed storage in real time', IPSJ Transactions on Advanced Computing Systems (No.40), Oct. 2012 (in Japanese).

[16] Kazuichi Oe, Satoshi Iwata, Takeo Honda, Motoyuki Kawaba, and Koji Okamura, 'On-The-Fly Automated Storage Tiering (OTF-AST)', in Proc. of The 3th Asian Conference on Information Systems – Special Session on Information Storage (ACIS-IS 2014), Nha Trang, Viet Nam, Dec 2014.

[17] Lin Lin, Yifeng Zhu, Jianhui Yue, Zhao Cai and Bruce Segee, 'Hot Random Off-loading: A Hybrid Storage System With Dynamic Data Migration', in Proc. of the 19th IEEE International Symposium on Modeling Analysis and Simulation of Computer and Telecommunication Systems, Raffles Hotel, Singapore, July 2011.

[18] Dushyanth Narayanan, Eno Thereska, Austin Donnelly, Sameh Elnikety, and Antony Rowstron, Microsoft Research Cambridge, UK. Migrating Server Storage to SSDs: Analysis of Tradeoffs. the 4th ACM European conference on Computer systems, April 2009.

[19] Ajay Gulati, Chethan Kumar, and Irfan Ahmad, VMware, Inc.; Karan Kumar, Carnegie Mellon University, 'BASIL: Automated IO Load Balancing Across Storage Devices', 8th USENIX conference on File and Storage Technorogies (FAST2010), Feb 2010.

[20] Y.Klonatos, T.Makatos, M.Marazakis, M.D Flouris, and A. Bilas, 'Azor: Using Two-level Block Secection to Improve SSD-based I/O caches', in Proc. of the Sixth Intl. Conf. on Networking, Arch., and Storage, 2011.

[21] Peter Bodik, Armando Fox, Michael J.Franklin, Michael l.Jordan, and David A.Patterson, 'Characterizing, Modeling, and Generating Workload Spikes for Stateful Services'. ACM Symposium on Cloud Computing (SOCC 2010), June 2010.

[22] Swaroop Kavalanekar, Bruce Worthington, Qi Zhang, and Vishal Sharda, 'Characterization of Storage Workload Traces from Production Windows Servers', the 7th International Smantic Web Conference(ISWC2008), Octorber,2008

[23] Microsoft. Event tracing. http://msdn.microsoft.com/library/, 2002, Platform SDK: Performance Monitoring, Event Tracing.

[24] Dongchul Park, Biplob Debnath, and David H.C Du, 'A Workload-Aware Adaptive Hybrid Flash Translation Layer with an Efficient Caching Strategy', in Proc. of the 19th IEEE International Symposium on Modeling Analysis and Simulation of Computer and Telecommunication Systems, Raffles Hotel, Singapore, July 2011.

[25] HDD 業界動向 - SNIA Japan (2013/11/25) http://snia-j.org/docs/education/20131125-1.pdf

[26] 第一回「フラッシュ・ストレージ出現の背景」NTT データ先端技術株式会社 http://www.intellilink.co.jp/article/column/b301.html

[27] 世界のストレージ市場調査 http://www.group.fuji-keizai.co.jp/press/pdf/120402_12032.pdf

[28] 大江和一: iops 当たりの busy 率を用いた IO 性能保証方法の提案，第 23 回コンピュータシステムシンポジウム Comsys2011, Dec., 2011.

[29] 大江和一，岩田聡，南里豪志，岡村耕二，性能向上を期待できる継続時間と IO アクセス数を満たした IO アクセス集中領域を自動抽出して SSD に移動することで性能向上を図る階層型ストレージシステムの提案と評価，情報処理学会 ACS 論文誌第 53 号，2015 年 12 月.

[30] Kazuichi Oe, and Koji Okamura, 'A hybrid storage system composed of on-the-fly automated storage tiering (OTF-AST) and caching', in Proc. of 2nd International Workshop on Computer Systems and Architectures (CSA'14), Mt. Fuji, Shizuoka, Japan, Dec 2014.

[31] Kazuichi Oe, Takeshi Nanri, and Koji Okamura, 'On-The-Fly Automated Storage Tiering with Caching and both Proactive and Observational Migration', in Proc. of 3rd International Workshop on Computer Systems and Architectures (CSA'15), Sapporo, Japan, Dec 2015.