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)

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

大江 和一[1,a)]　荻原 一隆[1]　野口 泰生[1]　小沢 年弘[1]

受付日 2012年3月28日, 採録日 2012年6月8日

概要：長時間にわたり特定の領域への負荷集中が発生するワークロードを対象に，負荷が集中する領域を動的に抽出し，その領域をリアルタイムに高速ストレージ（SSD）に移動する階層ストレージシステムの提案を行う．Microsoft Research が社内で運用しているサーバワークロード（MSR Cambridge）を公開しており，このワークロードを用いて評価を行ったところ，最大 77%性能向上することを確認した．負荷の分析に使用する blktrace, iostat がユーザからの IO に与える影響が軽微であることも確認した．

キーワード：階層ストレージシステム，SSD，リアルタイム制御，キャッシュ，負荷集中

Proposal for a Hierarchical Storage System that can Move a Spike to High-speed Storage in Real Time

Kazuichi Oe[1,a)]　Kazutaka Ogihara[1]　Yasuo Noguchi[1]　Toshihiro Ozawa[1]

Received: March 28, 2012, Accepted: June 8, 2012

Abstract: For workloads where a spike is generated, we propose a hierarchcal storage system that dynamically extracts the spike area and moves it to high-speed storage (SSD) in real-time. An MSR Cambridge workload was used for evaluation and a maximun of 77% performance improvements were confirmed. It was also confirmed that the influence of blktrace and iostat, which are used to extract the spike, on user IO was negligible.

Keywords: storage tiering, SSD, real-time control, block cache, workload spikes

1. はじめに

ストレージワークロードの分析を行うと，一部のデータ領域に負荷集中するものが少なからず存在する．たとえば，インターネットから利用されるサーバの負荷集中の調査を行った Workload Spikes 論文 [1] によると，全体の1%のデータに 32–88%の負荷が集まり，その負荷が数時間から最大数日継続する．そして，そのデータを支える各ストレージボリュームにおいても同様な負荷集中が発生するという報告が行われている．そこで samba ワークロード [3] や MSR Cambridge ワークロード [2], [4] のようなより規模が小さく性質が異なるワークロードの分析を行ったところ，同様な傾向にあることが分かった．

この任意のデータ領域に一時的に発生する負荷集中をたとえば RAID 装置だけで負荷設計を行うと，最大負荷に対応した構成となり非常に高価なシステムとなってしまう．

そこでこのような負荷集中が発生するワークロードを対象に，ボリューム全体の性能向上とピーク性能の改善を目指して，負荷が集中する領域を動的に抽出し，抽出した領域をリアルタイムに高速ストレージ（SSD）側に移動することを特徴とする HDD との階層ストレージシステムを提案する．提案方式を利用することで，ごく一部の領域を SSD に移動するだけでストレージシステム全体の性能を大幅に向上させ，HDD 単体では耐えられない負荷を処理することができる．さらに，負荷が集中する領域が SSD 側に

[1] 株式会社富士通研究所
Fujitsu Laboratories Ltd., Kawasaki, Kanagawa 211–8588, Japan
[a)] ooe.kazuichi@jp.fujitsu.com

移ることでHDD側のseekなどのオーバヘッドが小さくなり，HDD側から取り出せるIO数が向上する効果も得られる．なお，本稿ではこれ以降負荷が長時間継続して発生する領域をspike領域，短時間で負荷が収束する領域をshort spike領域と呼ぶことにする．

商用の多くの自動階層制御方式は，毎日決まった時間にそこまでに蓄積した統計情報を用いて負荷が高いデータ領域の抽出&移動を行うものであり，少なくとも数日間以上負荷が継続しないと効果が得られない．本提案方式では，負荷が高いデータ領域に変化があった場合リアルタイムに検出&移動を行うものであり，数時間しか継続しない負荷集中に対しても効果が期待できる．

以下に本稿の構成を示す．2章で関連研究を紹介する．3章でspike領域が発生するストレージワークロードに関する考察に関して説明する．4章で提案する階層ストレージシステムの説明を行う．5章で提案方法の評価に関して説明し，6章をまとめとする．

2. 関連研究

SSDなどの高速ストレージとHDDなどの低速ストレージを組み合わせた階層ストレージシステムは，製品レベルまで達したシステムも存在する．EMC FASTでは自動的に階層間移動を行うことが可能になる．文献[6]によると，1 GB sliceでボリュームを分割し，この単位で統計情報を収集し，階層間移動を行う．さらに，毎日決まった時間に階層間移動を行うと説明されている．これらより，少なくとも数日以上負荷が高いデータ領域を，負荷が低い深夜時間帯などを利用してSSDに移動する方式であると考えられる．本稿で提案するリアルタイムにspike領域を抽出し，SSDに移動するような機能の実装はない．

性能だけではなくコストや消費電力などの指標を用いて階層制御を行う提案も行われている．Narayananらの論文[7]では，使用するストレージの価格性能比や容量性能比を用いてデータ配置を決める提案が行われている．

分散ストレージシステムを対象にワークロードに応じて動的にストレージ構成を変更する提案として文献[10]があげられる．ここでは10–100 MBで分割したbinという単位でrequest latencyの監視を行い，この値があらかじめ定められたService-Level Objectives（SLOs）を満たせない場合にあらかじめ定められたperformance modelに従ってbinの再配置を行うという提案である．

SSDに代表される高速ストレージをキャッシュとして利用する研究としてfacebook flashcache[5]やDirectcache[11]などが存在する．これらはすべてOSSとしてソースが公開されていたり，製品として販売されていたりする．このうちflashcacheはIO offsetが4 KB境界のIOのみしかキャッシュできないので，IO offsetが4 KB境界とならないIOの割合が多いワークロードでは効果が期待できない．また，キャッシュ方式で実現した場合，SSDからHDDへのライトバックがユーザIOとは別に発生する．本稿の提案では，キャッシュ方式でライトバックに使用する帯域をユーザIOに用いることができるため，より性能を引き出すことが可能になる．

ストレージワークロードの分析を行った研究紹介も行っておく．文献[1]はインターネットから利用されるサーバの負荷集中により発生するspikeの分析とモデル化を行っている．本稿でもこの分析結果を参考に，より小さなワークロードにおけるspikeの分析を行った．文献[9]はWindows Server 2008上に構築した12の業務サーバに関するETW[*1]分析結果がまとめられている．この分析結果においてもExchange serverなどでspike発生の報告が行われている．

3. spike領域が発生するストレージワークロードに関する考察

インターネットから利用されるサーバの負荷集中の調査を行ったWorkload Spikes論文[1]では，data spikeとvolume spikeの存在が明記されており，data spikeに関して以下のような報告が行われている．

- Top 1%データへの負荷：全アクセスの32–88%
- Top 10%データへの負荷：全アクセスの67–99%
- peak到達まで40–130分．継続時間は数時間から数日．

さらに，企業内で運用されるsambaワークロードを1カ月間分析した報告[3]においても，spike領域[*2]が発生していることが分かる．この領域は全データ領域の0.9%にしかすぎず，さらに日中の時間帯に最大480分程度負荷が継続する．

そこで文献[1]のようなインターネットから利用されるサーバの負荷集中だけではなく，より小さいワークロードにおいても同等な結果が得られるのではないかと考え，一般に公開されているMSR Cambridgeワークロード[2]における負荷集中の分析を行うことにした．MSR Cambridgeはケンブリッジにあるマイクロソフトリサーチサーバに関するブロックI/Oトレースデータである．このトレースデータを用いた論文[4]のTable 1にトレースデータの概要がまとめられている．このトレースデータは13 servers，36 volumes，179 disksから収集しており，User home directoriesやWeb/SQL serverなど様々なサーバのIOのトレースとなっている．この中から最大50 iops以上の負荷が発生していた13 volumes[*3]を対象にspike領域発生に関する分析を行い，以下の結果が得られた．

- Top 1%ボリュームへの負荷：全アクセスの8–95%

[*1] Event Tracing for Windows
[*2] 1 iops以上の負荷が発生する領域
[*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

- Top 10%ボリュームへの負荷：全アクセスの 48–100%
- peak 到達まで 10–60 分．継続時間は 10–200 分．

文献 [1], [3] に掲載された調査結果，および一般に公開されている MSR Cambridge ワークロードに対する分析結果より，インターネットで利用されるサーバのワークロードだけではなく，企業内の samba や MSR Cambridge のような規模の小さなワークロードにおいても全領域の 10%程度に spike 領域が発生していることが分かった．さらに，spike の継続時間は数時間で終わっているものも多く，これらの短時間で終わる spike に対して効果的に性能改善する方法が必要なことが判明した．

4. 提案方法の説明

4.1 概要

本稿では，spike 領域を動的にとらえ，その領域をリアルタイムに高速ストレージに移動することを特徴とする SSD と HDD との階層ストレージシステムの提案を行う．提案方式を利用することで，ごく一部の領域を SSD に移動するだけで spike 領域で発生する IO を受け止めることが可能になり，安価で高性能なストレージシステムを構築することができる．必要とする SSD 容量は，全ボリューム容量の最大 10%程度を目安とし，数時間程度 spike 領域が継続するワークロードを主なターゲットとする．移動にともなうコストを性能向上効果が上回れば，トータルで性能向上を期待できる．

spike 領域を動的にとらえ，SSD へ移動する方法に関して述べる．ストレージボリュームを segment という固定の大きさに分割して扱う．この segment 単位に負荷の大小の判断を行い，負荷の高い segment を SSD へ移動する．segment の大きさに関しては，大きくとると spike 領域以外も含んでしまう確率が上昇するが spike 領域の揺らぎには強くなる．一方小さくとると，spike 以外の領域を減らすことができるが，spike 領域の揺らぎにより SSD へ移動した領域から外れる確率が上昇する．segment の大きさに関して定量的な指針は確立していないため，本稿では 1 GB として評価を行いその妥当性に関して評価時に議論することにした．

図 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．

4.1.1 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，segment 移動指示，の 3 コンポーネントから構成される．以下，各コンポーネントに関して説明する．

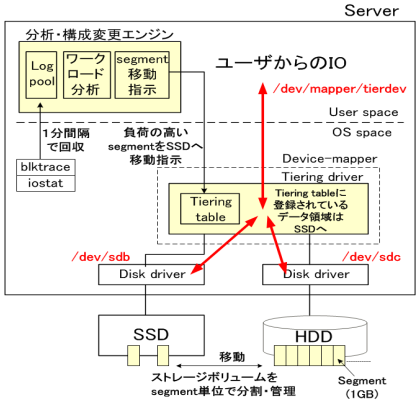

図 1　提案方法の Linux（CentOS 5.4）での実装例
Fig. 1　Mounting example in Linux (CentOS 5.4) of proposed method.

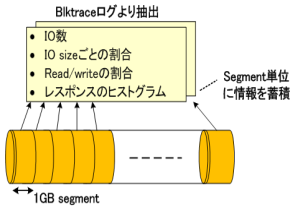

図 2　blktrace データの分析方法
Fig. 2　Methods of analysis of blktrace data.

4.1.1.1 Log pool

Log pool では，各 segment ごとの統計情報と階層制御に使用する各デバイスの統計情報を収集する．これら情報を用いて，高負荷 segment の抽出を行う．各 segment ごとに収集する統計情報は，一定時間間隔ごとの IO 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．

実装では，各 segment ごとの統計情報は blktrace を用いて抽出する．各デバイスの統計情報は iostat を用いる．実行間隔は 1 分間である．

1 分間隔で実行した blktrace より，IO 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 ではボリューム全体に発生する IO のトレースを，階層制御で IO を SSD と HDD に分離する前の段階である Tiering driver のレベルで取得する．図 1 を例にとると，/dev/mapper/tierdev に対して行う（**図 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 の実行結果を収集する．これらのデータを用いて，各デバイスの IO busy 率を把握する．図 1 を例にとると，/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 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という悪影響がある．そこで，リアルタイム性を確保しながら数時間前後継続する spike 領域の範囲と継続時間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．本稿の実験では事前検証の結果をふまえ 1 分間を測定間隔とした．

4.1.1.2　ワークロード分析

ワークロード分析は，負荷の高い segment を抽出しその中から SSD に移動する segment を決めることがその目的である．分析には，Log pool に収集した複数の blktrace/iostat データを用いて行う．blktrace/iostat データは timestamp を用いたインデックスを付与して管理しており，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．継続して負荷が高い segment 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[*4]があらかじめ決められた閾値（$\%util^h_{up}$）を超えた場合に SSD 側に高負荷 segment を移動する．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 数が増加する spike 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 4.2 節で行う．

4.1.1.3　segment 移動指示

segment 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 segment の移動を指示する．今回の実装では，/proc に Tiering driver に対して segment 移動を行う API を準備した．複数の segment 移動を行う場合，ユーザからの IO 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．segment 移動を同時に複数依頼すると，ユーザからの IO 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 1**）では，ユーザからの IO がない状態で 10 秒程度で segment 移動が可能である．SCSI timeout が 30 秒程度であることを考えれば，ユーザ IO の合間に segment 移動を行うことは可能である．ユーザ IO への影響が少ない segment 移動間隔は，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 $\%util^h_{up}$ を上回る場合は 60 秒間隔で実行するようにした．

表 1　ストレージ装置性能調査環境
Table 1　Machine environment.

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

[*4] iostat の 1 パラメータで IO busy 率を表す．

4.1.2　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 の送出を行う．**Tiering table** には，SSD に移動した segment 情報のみが掲載され，SSD 側の先頭 offset，HDD 側の先頭 offset を保持している．HDD の先頭 offset がボリューム全体の offset と一致する．登録してある HDD 側の先頭 offset から segment の大きさの範囲に収まる IO 要求が来た場合，SSD 側へ IO 要求を出す．Tiering table は segment 移動指示による SSD～HDD 間のデータ移動後に更新する．移動方法の詳細は 4.3 節で行う．今回の実装では，Tiering driver は Linux device-mapper framework 上に実装した．

4.2　高負荷 segment の抽出方法

制御の基本方針は，HDD 側の%util を $\%util^h_{up}$～$\%util^h_{lo}$ の範囲にとどめることで 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，$\%util^h_{md}$ になるように制御する（**図 3** 参照）．なお，図 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．

segment の移動は文献 [8] の方法を参考にした．文献 [8] では，そのときどきの%util と iops より「iops あたりの%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に関する提案を行っている．最初に HDD 側の%util が $\%util^h_{up}$ を上回るケースの説明を行う．まず，HDD 側に存在する各 segment を iops の大きい順にソートしておく．次に，HDD 側の%util と iops より iops あたりの%util を求めておく．そして，ソートした順に各 segment の iops を iops あたりの%util に掛け合わせ，得られた結果を加算していく．k 番

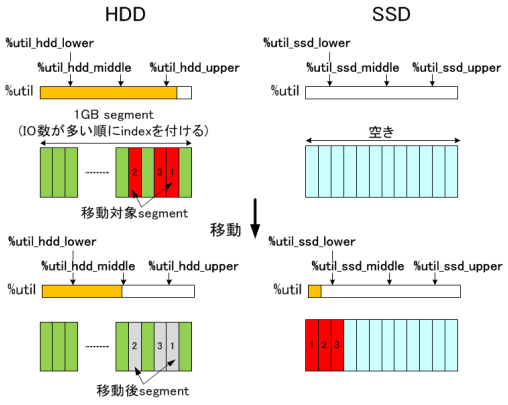

図 3　HDD から SSD に segment を移動する方法
Fig. 3　Method of moving segment from HDD to SSD.

目の segment 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 segment を移動したときの%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 $\%util_{md}^{h}$ を下回る n の値を求め，その segment までを SSD へ移動する．

次に HDD 側の%util が $\%util_{lo}^{h}$ を下回るケースの説明を行う．このケースでは，負荷の軽い SSD 側の segment を HDD 側に移動することになる．まず，SSD 側に存在する各 segment 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 あたりの%util を求めておく．そして，ソートした順に各 segment の iops を iops あたりの%util に掛け合わせ，得られた結果を加算していく．k 番目の segment 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 $\%util_{md}^{h}$ を上回る n の値を求め，その segment までを HDD へ移動する．

最後に $\%util_{up}^{h}$，$\%util_{md}^{h}$，$\%util_{lo}^{h}$ 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．$\%util_{up}^{h}$ 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 への移動を開始する．この値を大きくとれば，その分 HDD の性能をぎりぎりまで使うことができるが，SSD への移動を判断するタイミングが遅れることになる．HDD 側の%util が 100%に到達する前に SSD への移動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．$\%util_{md}^{h}$ は，segment 移動後の HDD 側%util のわずかな変動で再び移動判定とならないようにするために，$\%util_{up}^{h}$〜$\%util_{lo}^{h}$ 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．$\%util_{md}^{h}$ 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いことが分かっている．$\%util_{lo}^{h}$ 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 へ移動済み segment の一部を HDD へ移動する．この値を大きくとればその分 HDD の稼働率を上げることができるが，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 へ移動した segment 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 へ移動する動作となる．事前評価でこのような動作とならない値を求め，10%に設定することにした．

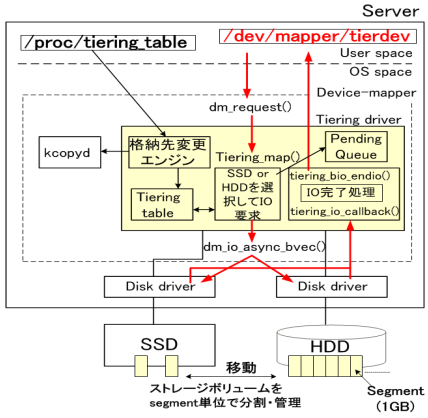

図 4　tiering driver の Linux（CentOS 5.4）での実装例
Fig. 4　Mounting example in Linux (CentOS 5.4) of tiering driver.

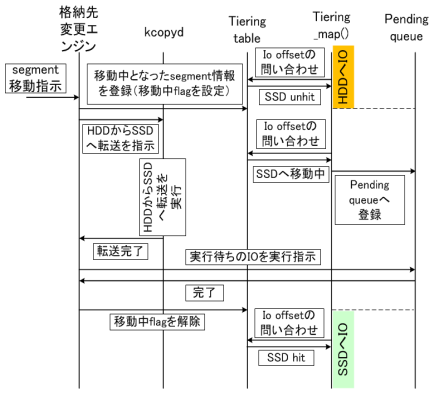

図 5　HDD から SSD にデータ移動を行うシーケンス図
Fig. 5　Sequence diagram where data moves from HDD to SSD.

4.3　SSD への移動方法

4.2 節で求めた segment を実際に移動する方法に関して説明する．**図 4** は，本稿の評価に使用した CentOS 5.4 における Tiering driver の詳細ブロック図である．ユーザからの IO を行ったまま segment の移動を実行できる設計とした．

最初にユーザからの IO 実行方法に関して簡単に説明する．ユーザからの IO が Tiering driver に入ってくると，IO 処理を行う Tiering_map() で受け取った IO 要求の offset 値と 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 要求を送出する．

次に segment の SSD〜HDD 間移動方法に関して説明する．segment 移動シーケンスを**図 5** にまとめた．分析・構成変更エンジンからの移動指示は/proc/tiering_table を介して Tiering driver に渡される．Tiering driver は，移動指示を受け取ると device-mapper 内共通モジュールである kcopyd を用いて segment 移動を行う．データ移動中の segment へのユーザ IO は Pending Queue に蓄積しておき，segment 移動環境を契機に Pending Queue に蓄えられた IO を実行する．

5. 提案方法の評価

5.1 概要

評価は以下の 4 つの観点で行った．

- HDD 単体との比較
- spike 領域と short spike 領域の構成比変更実験
- cache 方式との比較
- blktrace, iostat オーバヘッド調査

評価に使用した環境を表 1 にまとめた．評価に使用したワークロードは，3 章の調査に使用した MSR Cambridge ワークロードである．このデータの中から負荷集中が長時間継続していた**表 2** に示す 4 ワークロードを選択した．いずれも 1 時間以上継続する高負荷状態が繰り返し発生するものである．**表 3** はこれらワークロードの負荷の偏りを整理したものである．いずれも全ボリュームの 10%に半分以上の負荷（iops）が発生している．

ワークロードの再現は blktrace で取得したデータの再生を行う btreplay を使用した．btreplay が解釈可能な形式に変換する filter を通して実行した．評価の目的は，HDD 単体で負荷できないワークロードに対して，わずかな SSD 容量に本稿の提案方式を組み合わせることで負荷できることを示すことにある．そこで，そのままでは HDD 側を過負荷にできないワークロードに関して，btreplay を倍速実行[*5]することで過負荷状態をつくり出した．なお，本稿で過負荷とは，iostat %util が継続してほぼ 100%となってしまう状態を指す．

表 2　評価に使用したワークロード
Table 2　Workload used for evaluation.

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

表 3　評価ワークロードの負荷（iops）集中度
Table 3　Load (iops) concentration level of evaluation workload.

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%

*5　-x num で何倍速にするのかを指定する．

5.2 HDD 単体との比較

HDD から SSD への segment 移動オーバ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 2 のワークロードを実行した場合との比較を行った．比較方法はワークロード起動から 1 時間までの平均 iops の比較である．btreplay は，usr1，web2 は 8 倍速で実行し，proj2，proj4 は 2 倍速で実行した．**表 4** が実行結果である．usr1，web2，proj2 で性能向上が見られ，最大 77%向上しているが，proj4 ではほとんど効果が得られない．

表 5 は提案方法における SSD rw の割合[*6]，および提案方法と HDD 単体における HDD 側の svctm[*7]である．usr1 は SSD 6 GB の範囲に 89%の IO が発生しており，提案方法により spike 領域を SSD に移動でき，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%の領域（SSD 6 GB）を移動するだけで性能向上を達成できたことが分かる．

web2 は SSD に移動した 27 GB の領域に 55%の IO が発生している．usr1 より効果が小さい主な理由は移動領域が多いことである．今回の実装では，4.1.1.3 で説明したように複数の segment 移動を同時に行う場合は 60 秒間隔で行う．27 GB だと移動だけでも 27 分必要としてしまい，segment 移動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 に移動した 23 GB の領域に 11%の IO しか発生していないが 45%性能向上していることが分かる．表 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 0.39 ms 向上していることが分かる．svctm はそのときに取り出すことが

表 4　1 時間経過後の平均 iops（HDD 単体との比較）
Table 4　Average iops after one hour passes (comparison with HDD unit).

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 5　SSD rw の割合と svctm（HDD 単体との比較）
Table 5　RW ratio and svctm of SSD (comparison with HDD unit).

ワークロード	SSD rw の割合（%）	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

*6　=(SSD への IO 数*100)/(全 IO 数)
*7　iostat の 1 パラメータ．1IO の平均実行時間

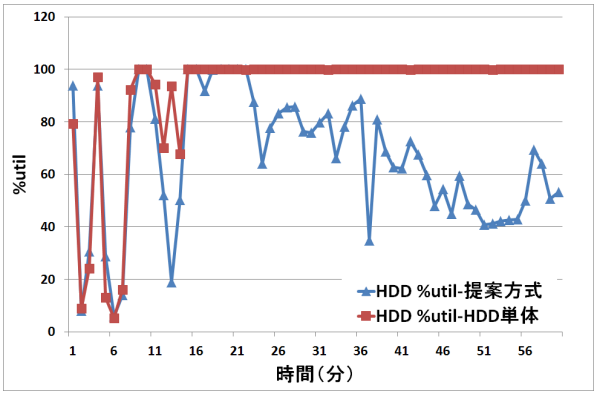

図 6　proj4 の 1 分間隔の%util（HDD 単体との比較）
Fig. 6　%util of proj4 every one minute (comparison with HDD unit).

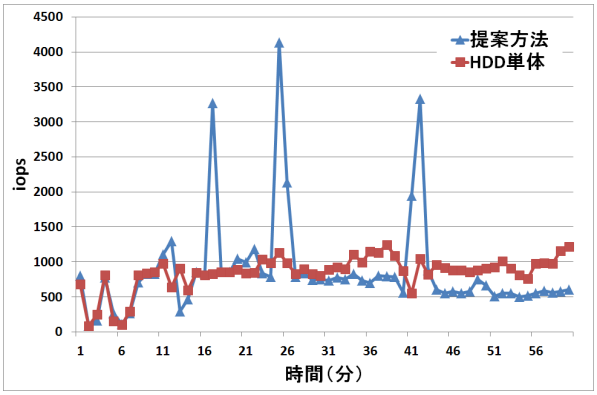

図 7　proj4 の 1 分間隔の iops（HDD 単体との比較）
Fig. 7　Iops of proj4 every one minute (comparison with HDD unit).

可能な最大 iops の逆数と解釈できるので，svctm が小さくなるということは HDD 側から取り出すことができる性能が大きくなったことを示している．SSD へ spike 領域を移動することで HDD 側の seek などのオーバヘッドが小さくなり，その結果として svctm が小さくなったと推定している．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いえる．

proj4 は SSD に移動した 23 GB の領域に 1%の IO しか発生せず，向上率も 1%であることが分かる．**図 6** より，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 単体はつねに 100%であることが分かる．**図 7** 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．提案方法では，3000 iops 超の負荷が 3 回発生し，1 度に高負荷状態を解消し，その後負荷が下がっていることが分かる．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，結果として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 23 GB 移動したのにアクセス量が 1%にとどまった理由は，移動した segment への負荷集中の継続時間が短かったためと考えられる．

表 6　spike 領域の大きさ（単位：1 GB）
Table 6　Spike area size (one spike: 1 GB).

	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

まとめると，usr1 のように spike 領域が狭いワークロードでは，提案方法によりすぐに性能向上する．web2 のように spike 領域が広いワークロードでは，segment 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 のように short spike 領域の負荷が大きいワークロードでは，spike 領域が SSD に移動することで HDD 側オーバヘッドが小さくなり，SSD から得られる性能以上に向上する場合がある．proj4 のように spike 領域の負荷がきわめて小さいワークロードでも，peak iops 向上に効果がある場合がある．

最後に segment の大きさに関する議論を行っておく．**表 6** は評価に用いた各ワークロードの spike 領域の大きさである．usr1，proj2 は平均値が 0.3–0.4 GB であるので 1 GB segment にすると無駄な領域が生じるが，proj4 は 1 GB が適しており，web2 は segment size を 12 GB 程度にするのが適している．一方，segment size を大きくすると HDD から SSD への転送時間が増加してしまい，ユーザ IO に影響を与える可能性が出る．本稿の検証環境では 1 GB 転送するのに 10 秒程度かかっていた．SCSI timeout が 20–30 秒であることを考えれば，本稿の検証環境では segment size を最大 2 GB 程度にとど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 segment size を設定したうえで，ワークロードに応じて segment の大きさを変更できるように実装するのが SSD 領域をより有効に利用できる．

5.3　spike 領域と short spike 領域の構成比変更実験

本実験の目的は，同じ領域に長時間継続して負荷が発生する spike 領域と負荷が短時間に収束&移動する short spike 領域に発生する負荷の割合を変更した実験を行うことで，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．

はじめに本実験環境における short spike 領域の負荷継続時間に関して議論する．short spike 領域を本稿の実装に照らしあわせると，少なくとも spike が発生した segment を検知し，その segment を SSD へ移動が終わったときには，spike が収束している領域，となる．提案システムは直前の 2 data を用いて判定を行っており，segment 移動は

表 7　spike 領域と short spike 領域の構成比変更実験結果
Table 7　Composition ratio change experiment result of spike area and short spike area.

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

s：spike，ss：short spike

少なくとも segment あたり 10 秒必要とする．この事実より，本実験環境における short spike 領域の負荷継続時間は，少なく見積もっても 3 分程度になる．

実験には表 2 の proj4 を用いた．proj4 を調査したところ 4 GB の spike 領域と 187 GB の short spike 領域から構成されることが分かった．そこで spike 領域と short spike 領域の負荷割合を変更できるようにするために，proj4 を spike 領域と short spike 領域に分離し，それぞれを重複して btreplay の実行を可能にした．spike 領域の btreplay 倍速度は 4 で固定し，short spike 領域の倍速度を 1 から 5 に変化させることで実験を行った．btreplay の実行時間は 20 分とした．事前の調査で short spike 領域の負荷を 4 倍速以上で実行すると HDD 側%util がほぼ 100%となることも把握した．

表 7 が実験結果である．SSD 使用量には，SSD へ移動が完了していない segment は含まない．増加 iops（期待値）は short spike 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 spike 領域を 3 倍速で実行した場合である．これは short spike 領域の負荷が 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 spike 領域を 4 倍速以上で実行すると，short spike 領域に関する SSD 移動オーバ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 spike 領域への負荷が 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．

5.4　cache 方式との比較

5.3 節の実験により，ワークロード構成要素の 1 つである spike 領域と short spike 領域の割合が提案方式の効果に影響することが分かった．また，SSD の使いかたとして，本稿で提案したような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比較を行うことにした．比較する cache 方式はオープンソースとなっている flashcache（[5]）とし，表 1 の環境を用いて比較実験を行う．

事前の調査で，flashcache は IO 要求の offset が 4 KB 境界と一致した IO のみ cache する実装となっており，表 2 のすべてワークロードに関してほとんど cache unhit となることが分かった．そこで，表 2 のワークロードの offset をもよりの 4 KB 境界に書き換える filter[*8] を準備し，実験に使用したワークロードはすべてこの filter を通して offset の変更を行った．さらに，本稿の提案方式と cache 方式では，SSD へ移動する時期とサイズ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考えられる．そこで，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 方式に関して，実環境に適用した場合に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，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る．

ワークロードは表 2 の中から spike 領域と short spike 領域の構成比が異なる usr1 と proj2 を選択した．flashcache が使用する SSD 容量は表 4 と一致するようにした．btreplay の倍速は usr1 が 8 or 10 倍，proj2 が 2 倍とした．さらに，4.2 節で説明を行った高負荷 segment 抽出ロジックに関して，HDD 側が過負荷状態に陥っていなくても IO が継続的に発生する segment を積極的に SSD へ移動するロジックを追加した．cache 方式に対して優位性を持たせるためには，負荷があまり大きくなくても継続的に IO が発生する領域を SSD に移す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，同じ segment に連続して m 回 IO が継続する segment を SSD へ移動する．m はワークロードに依存するパラメータであり，m 回 IO が継続した segment はさらに長期間 IO が継続することを見込んで SSD に移動を行う．m はワークロードに応じて最適値が異なるが，今回は m=8 で事前検証を行い継続的に IO が発生する segment の抽出ができたため，そのままこの値を使用した．

表 8 が実験結果である．usr1 に関して，read の割合が高いと性能差がないことが分かる．write の割合を高くすると，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 segment 移動が終了した後に限って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 8** は usr1(x10)$ の 1 分間隔の iops の推移である．最初の 3 GB

[*8] new_offset=(offset/4096)*4096

表 8　1 時間経過後の平均 iops（flashcache との比較）
Table 8　Average iops after one hour passes (comparison with flashcache).

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 segment 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 と値が異なる

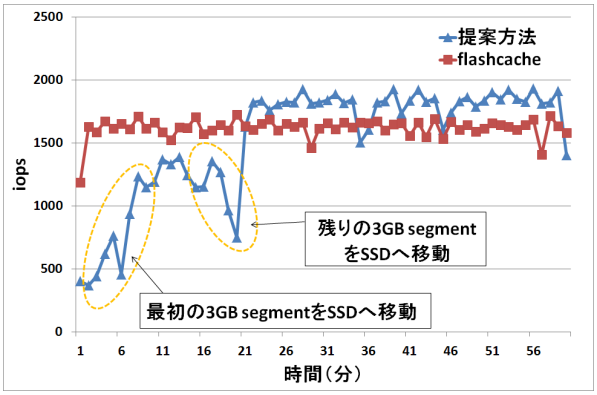

図 8　usr1(x10)$ の 1 分間隔の iops（flashcache との比較）
Fig. 8　Iops of usr1(x10)$ every one minute (comparison with flashcache).

segment の移動は，4.2 節で説明を行ったロジックによるものである．最後の 3 GB segment の移動は，本実験用に追加した積極的に SSD への移動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 3 GB segment 移動完了後から 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 8 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 のように spike 領域の割合が大きいワークロードに関して，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ことが分かる．ただし，高負荷 segment 抽出ロジックに関して，4.2 節だけではなく，ワークロードの特徴に応じて積極的に SSD への移動を行うロジックも必要であることが分かる．

proj2 に関しては，つねに flashcache の性能が提案方法を上回った．**図 9** は proj2*の 1 分間隔の iops の推移である．usr1 より short spike 領域の割合が大きくなる proj2 では，flashcache の方が効果的であることが分かる．

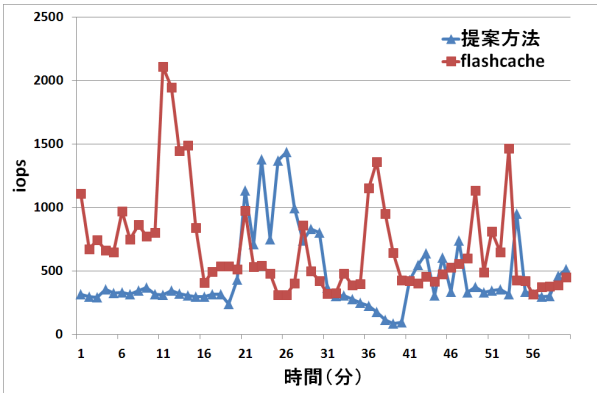

図 9　proj2*の 1 分間隔の iops（flashcache との比較）
Fig. 9　Iops of proj2* every one minute (comparison with flashcache).

表 9　blktrace, iostat オーバヘッド調査結果
Table 9　Blktrace and iostat overhead investigation result.

	分析・構成変更エンジンあり	分析・構成変更エンジンなし
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.

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 spike 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 spike 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ことが分かる．EMC FAST [6] 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 spike 領域の性能向上へは効果的ではない．

5.5　blktrace, iostat オーバヘッド調査

4.1 節で説明したように，提案方法では blktrace と iostat をつねに実行することで，ワークロードのリアルタイム分析を実現した．そこで blktrace と iostat をつねに実行することによるユーザ IO への影響調査を行った．調査は usr1 を起動し，6 GB segment の SSD への移動をまず行っておく．その後，usr1 を 10 分間ずつ再実行し，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 9 は実験結果である．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が確認できた．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が確認できた．今回実験に使用した環境では，blktrace, iostat のユーザ IO への影響はほとんどないと判断できる．

6. まとめ

特定の領域への負荷集中が数時間発生するワークロードを対象に，ボリューム全体の iops 性能向上と，ピーク iops 性能の改善を目指して，負荷が集中する領域を動的に抽出し，その領域をリアルタイムに高速ストレージ（SSD）に移動する階層ストレージシステムの提案を行った．提案方式を利用することで，最大でも全ボリューム容量の 10%未満の SSD 容量を追加するだけでストレージシステム全体と，ピーク性能を大幅に向上することができる．

spike 領域が狭い場合は，すぐに大きな性能向上が得られること，広い場合には効果が出るまで時間がかかるので spike の長時間の継続が必要なことが分かった．この点を改善するためには，SSD への segment 移動方法を改良する必要がある．また，spike 以外の領域の負荷についても，spike 領域が SSD に移ることで HDD のオーバヘッドが小さくなり，SSD から得られる性能以上に向上するケースがあることを確認した．負荷の分析に使用する blktrace, iostat がユーザからの IO に与える影響が軽微であることも確認した．

cache 方式との比較に関しては，spike 領域の負荷が支配的かつ write の比率が高いワークロードでは提案方式が優位であることが分かる．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．

今後の課題は以下である．

- ユーザ IO への影響を最小にしつつ，迅速に SSD への segment 移動する方法の追求
- ワークロードの特徴に応じた高負荷 segment 抽出方法
- spike 領域のみを抽出する方法
- 負荷が下がってきたときに HDD へ segment を戻す方法の検証

参考文献

[1] Bodik, P., Fox, A., Franklin, M.J., Jordan, M.L. and Patterson, D.A.: Characterizing, Modeling, and Generating Workload Spikes for Stateful Services, *ACM Symposium on Cloud Computing* (*SOCC 2010*) (June 2010).

[2] MSR Cambridge Traces: available from ⟨http://iotta.snia.org/traces/388⟩.

[3] 大江和一，熊野達夫，野口泰生：iops 当たりの IO Busy 率を用いた IO 性能設計方法の提案，第 8 回先進的計算基盤システムシンポジウム SACSIS2010 (May 2010).

[4] Narayanan, D., Donnelly, A., Rowstron, A. and Microsoft Research Ltd.: Write Off-Loading: Practical Power Manangement for Enterprise Storage, *Proc. 6th USENIX Conference on File and Storage Technologies* (*FAST 08*) (2008).

[5] facebook flashcache: available from ⟨https://github.com/facebook/flashcache⟩.

[6] EMC White Paper, EMC FAST VP for Unified Storage Systems A Detailed Review (Mar. 2011).

[7] Narayanan, D., Thereska, E., Donnelly, A., Elnikety, S., Rowstron, A. and Microsoft Research Cambridge, UK.: Migrating Server Storage to SSDs: Analysis of Tradeoffs, *The 4th ACM European conference on Computer systems* (Apr. 2009).

[8] 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).

[9] Kavalanekar, S., Worthington, B., Zhang, Q. and Sharda, V.: Characterization of Storage Workload Traces from Production Windows Servers, *The 7th International Smantic Web Conference* (*ISWC2008*) (Oct. 2008).

[10] Trushkowsky, B., Bodik, P., Fox, A., Franklin, M.J., Jordan, M.I. and Patterson, D.A.: The SCADS Director: Scaling a Distributed Storage System Under Stringent Performance Requirements, *9th USENIX Conference on File and Storage Technologies* (Feb. 2011).

[11] Fusion-io directCache: available from ⟨http://www.fusionio.com/data-sheets/directcache/⟩.

大江 和一

1988 年九州大学工学部情報工学科卒業．同年富士通株式会社に入社．現在，株式会社富士通研究所にてストレージシステムの研究開発に従事．

荻原 一隆

1993 年株式会社富士通研究所に入社．現在，ストレージシステムの研究開発に従事．

野口 泰生　（正会員）

1988 年東京大学大学院薬学系研究科中退．同年株式会社富士通研究所に入社．現在，同社にてストレージシステムの研究開発に従事．

小沢 年弘　（正会員）

1984 年株式会社富士通研究所に入社．並列計算機，コンパイラおよびストレージシステムの研究開発に従事．