Vol.2016-HPC-156 No.13
2016/9/16

ブロックストレージシステムにおける Latency-aware Inline-dedup Optimization

大辻 弘貴[1]　加藤 純[1]　鈴木 康介[1]　佐藤 充[1]　吉田 英司[1]

概要：コンピュータシステムで取り扱われるデータ量の増大により，ストレージシステムには，データを効率的に格納することが求められている．効率化のための一つの方法として，重複排除によるデータ量の削減が挙げられる．この処理により，同じデータを持つブロックが複数存在する状態を解消することができるため，限られた容量のストレージデバイスを有効活用できることが期待される．一方で，重複排除処理はブロックの同一性を判断するために，一般的にはハッシュ計算を用いており，書き込みと同時にインラインに重複排除を行うと，その分の処理時間が書き込みレイテンシに上乗せされてしまう問題がある．特に，近年採用が進んでいるフラッシュメモリに代表される高価かつ応答の高速なデバイスを用いたシステムの場合，インライン重複除去の価値が高い一方で，重複除去によるレイテンシ悪化の影響が相対的に大きくなる．この問題を解決するために，一時的にデータをキャッシュあるいはストレージに置いたうえで書き込み完了応答を返す手順を用いることも可能であるが，その場合には後で非同期に処理を行わなければならず，最終的な処理コストが高くなり，性能が低下する可能性がある．そこで本稿においては，目標とするシステム I/O 性能の達成と書き込みレイテンシ短縮を両立させるために，二種類の異なる重複排除手順を備えたシステムを用いて，状況に応じて最適な手順を選択する手法を提案および評価する．

1. はじめに

産業や科学など分野を問わず，コンピュータシステムが取り扱わなければならないデータ量は増加し続けており，高速かつ効率的にデータを取り扱うことは不可欠である [1]．最終的なデータ保存を行うシステムとして，ブロック単位で I/O を行うブロックストレージシステムも依然として広く用いられている．しかしながら，データには重複する内容が含まれることも多く [2]，これらの内容を一つの実体にまとめる（重複排除）することは，限られたストレージ容量を有効に活用する上で効果的な手段である．そのような背景から，ブロックストレージシステムにおいても，重複排除技術が用いられている．

従来はバックアップ用途のストレージにおいて重複排除技術が用いられることが多かったが，最近の動向として，ビット単価の高いフラッシュメモリを用いたストレージデバイスの採用が増えてきたことから，効率的に記憶デバイスを使用する必要性が高まり，書き込みと同時に重複排除を行うストレージシステムが登場している．

一方で，重複排除処理を行うためには，同一のブロックが既にストレージシステム内に存在するかどうかを書き込みがあるたびに確認しなければならず [3]，この処理のオーバヘッドが加わる．ブロックが同一であるかの判断はハッシュ値で行われることが一般的であるが，ハッシュ計算は時間のかかる処理であり，この処理時間がシステムとしての書き込みレイテンシを大きくしてしまう．ストレージシステムの書き込みレイテンシは，特にトランザクション処理において影響が大きく [4]，改善することは実システムの性能向上に気よすることが期待される．また，フラッシュメモリのレイテンシは従来のデバイスと比べて短いことから，データの書き込み以外の応答時間が相対的に影響が大きくなる点も，重複排除機構の応答を高速化する意義の一つである．加えて，重複排除機構の応答時間が十分に高速であれば，キャッシュ上においても重複排除を行うこともメモリ領域の効率的な利用につながることから，本稿におけるシステムにおいては原則的にキャッシュ領域も重複排除を行う構造としている．

本稿においては，重複排除処理がシステム応答時間に与える影響を軽減するために，システム性能に余力がある状況を動的に判断した上で，ごく短期的に重複排除を行わずにデータを保持する手順を併用する手法を提案し，その性能評価結果を示す．

[1] 株式会社富士通研究所

2. 関連研究

重複排除技術は，ブロックストレージのみならず，共有ファイルシステムなど，様々なレイヤーのシステムで用いられている．具体的には，Venti [5], HYDRAstor [6], NetApp ONTAP [7], HPE 3PAR StoreServ [8], EMC XtremIO [9], DBLK [10] といった製品やソフトウェアが存在する．[5] や [6] はバックアップ・アーカイブにおける重複排除処理を目的としている．本研究は，プライマリストレージシステムとしても使用可能な，書き込みと同時に重複排除を行うシステムを対象としており，[9][10][7][8] などが近い例として挙げられる．

重複排除を行うことによる性能とのトレードオフについては，[11][12][13] で述べられている．[11] は重複排除ストレージシステムにおいてレイテンシを短縮する点において目的は同じであるが，重複排除を行うことによってディスク上のデータが断片化することを課題としており，解決しようとする問題が異なる．

本研究が対象としているシステムは，フラッシュメモリをはじめとする応答が高速なデバイスを用いることを前提としており，レイテンシを延長させている原因も異なるため，解決に向けたアプローチも異なっている．

3. 対象とするストレージシステムの構成と動作

3.1 全体の構成

本稿における提案手法は，重複排除を行うストレージシステム全般に通じる課題を解決するものであるが，本章では本稿で前提としているシステム構成を説明する．

本研究が対象としているシステムは，複数のノード上で，スケーラブルに単一のビューを提供するブロックストレージであり，その中でグローバルに重複排除を行うシステムである．**図 1** はこの構成を示したものである．図に示すように，ストレージシステムは記憶装置を備えた複数のノードで構成されており，それぞれはネットワークで接続されている．システムは，任意のサイズを持つ論理ボリュームを通じて，ユーザに対してブロックデバイスとしてのインタフェースを提供する．

本システムにおいては大きく分類すると，データの実体，論理アドレステーブルおよびブロックハッシュテーブル（メタデータ）の 3 種類の情報が存在する．論理ボリュームは複数のノードにまたがる形で作成され，実際のデータやメタデータについても分散して保管・管理される．これにより，ノード数の増加に対してスケールアウトすることを目指している．加えて，本システムの構造上，リモートに存在するメタデータアクセスへのアクセスが頻発することから，原則としてメタデータはすべてオンメモリ処理し，

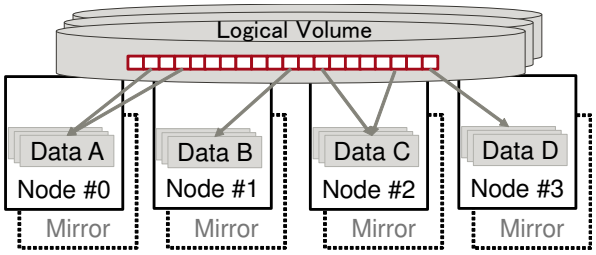

図 1 システム全体の概念図を示している．複数の論理ボリュームが複数のノードにわたって割り当てられており，装置全体のデータに対して重複排除が行われている．

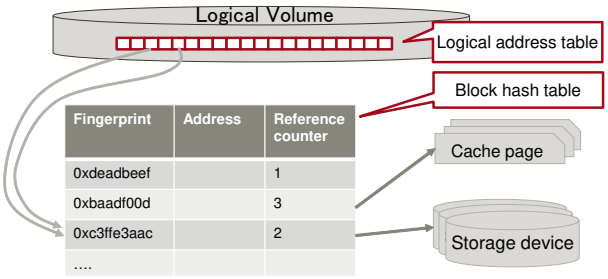

図 2 論理アドレステーブルとブロックハッシュテーブルの関連および指し示すデータを表している

リモートノード上のメタデータに対しては RDMA（遠隔ダイレクトメモリアクセス）を行うことを前提としている．

また，耐障害性を確保することを目的として，各ストレージノードは 2 台 1 組としてミラー構成をとっている．以後この 1 組を，便宜上ノードと呼ぶこととする．何らかのデータ操作が行われる際には，必ずデータおよびメタデータともにミラー部に対する転送が完了してから完了応答を返す．

以降の節においては，これらのメタデータ構造の詳細と，読み書きの際の実際の動作について説明する．

3.2 メタデータ構造

本節においては，本研究が対象とするシステムが用いるメタデータ構造について述べる．

ブロックストレージシステムは，一意に割り当てられたアドレスによって位置を指定することにより，固定長領域のデータに対するアクセス手段を提供するものである．本研究が対象とするシステムにおいては，重複排除を行うこともあり，論理ボリューム上のアドレスを用いる．したがって，論理アドレスと実際のデータを紐づけるためのメタデータとして，論理アドレステーブルが必要になる．この詳細は 3.2.1 にて述べる．

加えて，本ストレージシステムは，重複排除処理を行うために，ブロックの情報もメタデータとして保持する必要がある．このメタデータについては，3.2.2 にて詳細を述べる．また，これらのメタデータの関連を**図 2** に示す．

3.2.1 論理アドレステーブル

重複排除処理を行う場合，一つの実体データに対して複数のアドレスが紐づくことから，ストレージデバイスの物理アドレスとは切り離された論理アドレスを用いる必要がある．このアドレスの管理を行っているのが論理アドレステーブルであり，スケールアウト性能のため，論理アドレス空間をストライピングする形で各ノードに分散管理される．分散方法については 3.4 で述べる．

また，図 2 に示したように，論理アドレステーブルは，論理ボリュームの論理アドレスと，データの実体を結びつける働きをしている．実際には，本システムが重複排除処理を行うストレージシステムであることから，重複排除処理が行われたデータに関しては，次の節 3.2.2 に記すブロックハッシュテーブルのエントリを指すことになる．例外的に，ブロックの一部分のみに対する書き込み (partial write) が行われた場合には，直接論理アドレステーブルから断片的なデータが含まれるキャッシュページを対応付ける．

3.2.2 ブロックハッシュテーブル

本研究が対象としているストレージシステムは，重複排除処理を行うために，すでに記録されているブロックに関する情報を管理する必要があり，ブロックハッシュテーブルがこの役割を果たしている．加えて，ブロックアドレステーブルには，論理アドレステーブルのエントリとキャッシュページのアドレスあるいはデータの実体が配置された物理アドレスの橋渡しをする役割もある．

ブロックハッシュテーブルは図 2 に示すとおりである．図に示されるように，ブロックハッシュテーブルのエントリには，ハッシュ値をキーとして，データの実体に関する情報と，ブロックの参照数が含まれている．ブロックの参照数は，あるハッシュ値を持つブロックを参照する論理アドレスの数を表しており，あるブロックの重複排除処理が行われた場合にインクリメントされる値である．なお，本システムにおいては，ハッシュ関数として SHA-1 [14] を用いており，そのビット長は 160 である．SHA-1 のハッシュ空間は，少なくとも本稿が対象としているストレージ容量（最大数十〜数百 TB）に対しては，衝突困難性の観点からも十分である [15][16].

ブロックハッシュテーブルはストレージシステムに書き込まれた全データについてのハッシュ値を保持しており，そのエントリ数は非常に多くなることが想定される．そのため，ハッシュ値の先頭ビットの一部を用いてハッシュリストを構成し，検索の高速化を行っている．

3.3 キャッシュページと物理アドレス

本節では，データの実体であるキャッシュページと物理アドレスの取り扱いについて説明する．前節において述べたように，本システムは 2 種類のメタデータを使用して論理アドレスの管理と重複排除処理を行っている．データがキャッシュに残っている場合には，実体データはキャッシュページへのポインタによって指し示すことができる．また，データがキャッシュにない場合には，記憶装置上の物理アドレスが実体データに対応する．

したがって，図 2 に示したように，論理アドレスからデータの実体をたどるために，論理アドレステーブル上のエントリがブロックハッシュテーブルのエントリのポインタを保持する．ブラックハッシュテーブルのエントリは，当該データがキャッシュに乗っている場合にはそのメモリアドレスを指し，乗っていない場合にはステージングに必要な物理アドレスを指し示す．

3.4 データ・メタデータの分散方法

高いスケーラビリティを実現するためには，適切にデータ (キャッシュページ）や，メタデータを分散させる必要がある．一方で，一貫性を保つために，それぞれの情報に関する分散方法が一意に定められなければならない．

本システムにおいては，論理アドレステーブルについて，あるサイズにおいてストライプ分割し，管理を担当するノードを分散させている．ブロックハッシュテーブルおよびデータについては，ハッシュ値から一意に定まる方法により，担当するノードを決定し，そこで情報を管理する．これは，同一ブロックが並列に書き込みされた場合などにおいて，一貫性を保つことを容易にするためである．

3.5 システムの動作

本節では，本稿において用いるシステムの動作およびデータ処理手順について説明する．本稿において用いるシステムには，二つの書き込み手順が存在する．一つ目は dedup-through モードで，書き込みと同時にブロックのハッシュ値計算を行い，その結果に基づいてデータを配置する．二つ目は dedup-back モードで，一旦論理アドレスベースでキャッシュページを置き，書き込み完了応答を返す．それぞれの特徴としては，dedup-through モードは総処理コストが低いことから IOPS は高いがレイテンシが長い一方，dedup-back モードはレイテンシは短い一方で総処理コストが増えるため IOPS 性能の面で不利である．具体的な手順については次節以降に記す．

3.5.1 書き込み手順 (dedup-through モード)

本節においては，重複排除を同期的に行う dedup-through モードについて説明する．

dedup-through モードにおいては，同期的に重複排除処理を行うため，最初にブロックのハッシュ値計算処理が行われる．次に，計算されたハッシュ値に基づいて，キャッシュページの保存先とブロックハッシュテーブルの作成先を決定し，データの転送およびブロックハッシュテーブルのエントリ作成を行う．加えて，今回追加されたブロック

ハッシュテーブルのエントリへのアドレスを，書き込みの行われたブロックの論理アドレスに対応する論理アドレステーブルのエントリに反映する．最後に，書き込み完了応答を返すことにより処理が終了する．

この書き込み手順においては，メタデータおよびデータを配置するノードを選択するためには，まずブロックのハッシュ値を計算しなければならないうえ，ハッシュ値計算が終わるまで次の処理に入ることができない．従って，書き込み応答までの時間に直列にハッシュ計算の時間が上乗せされることから，レイテンシ面では不利である．一方で，書き込まれたデータ（ブロック）の配置はハッシュ値によって決定されていることから，再び移動することはなく，論理アドレステーブルを今後更新することはない．

3.5.2 書き込み手順 (dedup-back モード)

本節においては，一時的に論理アドレスベースでキャッシュページを配置し，書き込み完了応答を返した後にハッシュ値の計算およびデータの再配置を行う，dedup-back モードについて説明する．

dedup-back モードにおいては，書き込みが行われたブロックのデータの配置を，一旦論理アドレスに基づいて決めて，仮置きを行う．dedup-through モードにおいてブロックの配置に用いるハッシュ値と異なり，論理アドレスは書き込みリクエストを受け取った段階で分かるため，すぐにデータの転送およびキャッシュページの作成を行うことができる．次に，論理アドレステーブル上のエントリを，今回の書き込みデータが記録されたキャッシュページを指し示すように変更する．この段階で書き込み完了応答を返すことができるため，ハッシュ計算時間がレイテンシに入らず，応答時間の短縮が期待される．

しかしながら，重複排除を行うストレージであることから，仮置きされたデータを本来のハッシュ値に基づくレイアウトに変更しなければならない．この処理は，ストレージデバイスへの書き出し（デステージ）時またはデータがキャッシュ上にある任意のタイミングに行うことができる．この処理は図中後半に示されており，dedup-through モードにおいて行われる処理と同様，仮置きされたキャッシュページのハッシュ値を計算することにより担当ノードを決定し，データの転送及びブロックハッシュテーブルの作成を行う．最後に，論理アドレステーブルのエントリを更新し，作成されたブロックハッシュテーブルのエントリを指し示すようにする．

dedup-back モードは，ハッシュ値の計算を後で行うことから，システムに対する書き込み応答時間を短縮できることが期待される．一方で，一旦論理アドレスに基づいてデータの配置を行っており，改めてハッシュ値の計算を行った後に本来の配置に移動し，論理アドレステーブルの更新処理を行わなければならない．データの再配置に関連した処理は，dedup-back モードにのみ存在し，通信メッセージ数を増大させる．従って，レイテンシの短縮を目的としてすべてのリクエストを dedup-back モードで使用することは，IOPS 性能の低下を招くことにつながるため，状況に応じて dedup-through モードを併用する必要がある．

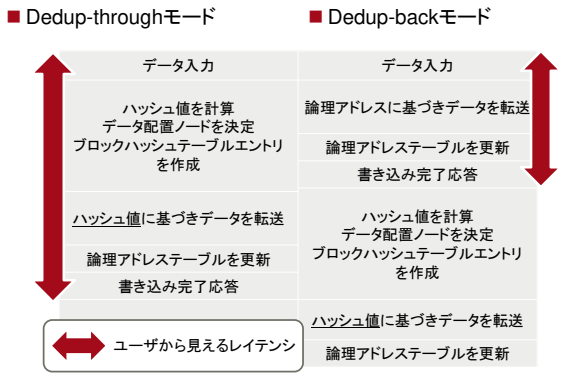

図 3 Dedup-through モードと dedup-back モードの処理内容およびレイテンシ比較

3.5.3 書き込み手順の比較

Dedup-through モードと dedup-back モードでは，それぞれ書き込み完了応答を返す段階が異なる．本節では，書き込み要求を受け取ってから応答を返すまでの時間（レイテンシ）の違いを，書き込み手順を時系列に並べることにより示して比較する．**図 3** は，2 つのモードを比較した図である．これまでに述べたように，dedup-thrhgou モードは書き込み完了応答を返すまでの時間は長いが，処理の総量は少ない．一方で，dedup-back モードは，書き込み完了応答を返すまでの時間は短いが，全体の処理は長くなることがわかる．

3.5.4 読み込み手順

本稿は，重複排除ストレージシステムに対する書き込みレイテンシを対象としているが，前提としているシステムの理解のため，読み込み手順も本節に示す．

読み込みリクエストは，論理アドレスで位置を指定される．従って，最初に論理アドレステーブルから対応するエントリを読み出し，その中に格納された情報からキャッシュページのアドレス（ブロックハッシュテーブルを経由する場合もある）または物理アドレスを得る．キャッシュページのアドレスであった場合，（遠隔）メモリアクセスを行い，物理アドレスであった場合にはストレージデバイスからステージング処理を行い，データの読み込みが完了する．

4. 課題と提案手法

4.1 重複排除処理と性能のトレードオフ

3.5.1 にて述べたように，重複排除処理を同時に行う dedup-through モードにおいては，データをキャッシュに配置する前の段階でハッシュ計算を行わなければならない．当然ながら，書き込み完了応答はキャッシュにデータを配

置した後でなければ返すことができないため，ハッシュ計算の時間はレイテンシに上乗せされる．

しかしながら，ハッシュ計算はその他の通信などと比較しても時間のかかる処理であり，dedup-through モードのみを用いた場合には，必ずこの分の時間レイテンシが伸びる．書き込み手順の説明でも述べたように，dedup-back モードを利用した場合，ハッシュ計算の時間を書き込み応答時間から除外することができるが，一方でハッシュ値に基づくデータの再配置処理などにより，1 ブロックを書き込むために要する処理量および通信量が増加する．その結果，IOPS 性能が低下する可能性がある．

4.2 書き込み手段の選択手法

3.5.2 で述べたように，dedup-back モードには，書き込みレイテンシを短縮する効果がある．一方で，前節で述べたように，dedup-through と dedup-back には IOPS とレイテンシにおいてトレードオフがあることから，これらを使い分けることが必要である．本節では，提案手法である，異なる I/O 手順の選択手法について説明する．

本提案手法は，目標とする IOPS を満たしつつ，できる限り重複排除処理に伴うレイテンシ増大を抑えることを目指している．したがって，目標 IOPS を下回らないように I/O 手順を選択する必要がある．異なる I/O 手順でデータを書き込む場合，キャッシュ上には，論理アドレスに基づいて配置されたもの (dedup-back) と，ブロックのハッシュ値に基づいて配置されたもの (dedup-through) が混在した状態になる．この混在比率によって，キャッシュをすべてストレージデバイスに書き出す（デステージ）する場合の IOPS が変化する．そこで，任意の時点において，キャッシュ全体をデステージする際に目標となる IOPS を下回ることのないように，キャッシュ上のデータ比率を制御する．

具体的には，キャッシュ上の各 I/O 手順ごとのキャッシュ比率目標を設定し，現在のキャッシュページ数の比率と比較したうえで，目標値に近づくよう I/O 手順を選択する．

4.2.1 キャッシュ上のデータ比率と IOPS

あるサイズ（N）のキャッシュを持つシステムにおいて，dedup-back によって配置されたキャッシュ数比率を F，dedup-through によって配置されたキャッシュ数比率を $1-F$ とする．また，合計のキャッシュページ数を c とする．さらに，dedup-back で配置されたページをハッシュ計算及び再配置したのち，ストレージデバイスに書き出すのに要する時間を L，同様に dedup-thgouh で配置されたページをストレージデバイスに書き出す所要時間を H とする．満たさなければならない IOPS 性能を S とする．

この場合には，以下の式を満たす F を選択しなければならない．

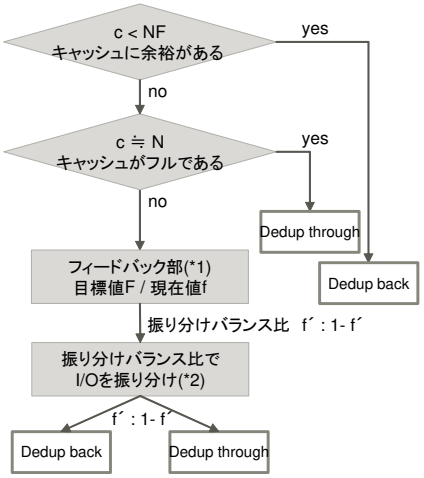

図 4 I/O 手順の選択手法

$$\frac{1}{FL+(1-F)H} >= S \tag{1}$$

この中で，最大の F をとることがレイテンシ短縮につながるため，目標とする dedup-back により配置されたブロックの比率 F は以下の通りになる．

$$F = \frac{(1-SH)}{S(L-H)} \tag{2}$$

キャッシュ上において，dedup-back により配置されたキャッシュページ数の比率が F になるよう，I/O 手順の選択を行う．

4.2.2 I/O 手順の選択手法

本節では，前節で求めた F を利用して，I/O 手順の選択を行う手法を説明する．**図 4** は，この手順を示したフローチャートである．キャッシュに余裕がある状態では，システムの余力も大きいため，dedup-back モードを選択することができる．一方でキャッシュがほぼ埋まっている状態は，持続的な書き込みが発生し，常にストレージデバイスに対する書き出しが行われている状態のため，処理コストの低い dedup-through モードを利用する必要がある．

キャッシュの状態が上記いずれの状態でもない場合には，dedup-back モードによって配置されたキャッシュページ数の比率が F になるよう，振り分け処理を行う．現在の dedup-back モードによるキャッシュページ比率を f とするとき，目標値 f' は以下の通りである．

$$f' = F + (F - f) \tag{3}$$

（ただし，値は 0〜1 の範囲でキャップされる）これにより，現在との差分を用いて，目標値に近づけることを目指した f' を求めることができる．$f' : 1-f'$ の比率で，I/O 手順を選択する．

5. 性能評価

5.1 評価環境および条件

性能評価に用いた計算機の仕様は，**表 1** の通りである．

表 1　評価環境

CPU	Intel(R) Xeon(R) CPU E5-2609 v2 2.50GHz 4 cores x2
RAM	96GB
OS	CentOS 7.2
ベンチマークツール	fio-2.2.8
InfiniBand HCA	Mellanox 4x FDR (56Gbps)

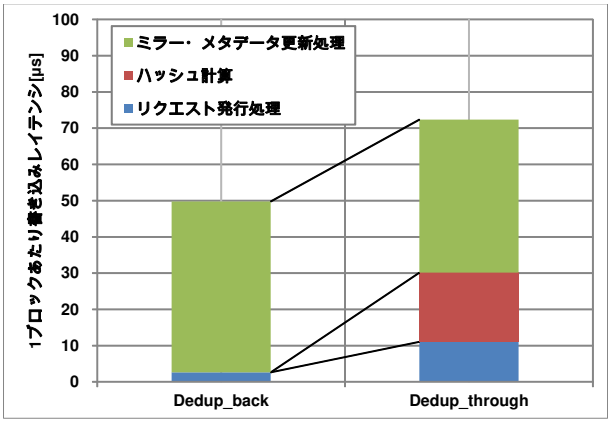

図 5　Dedup through モードと Dedup back モードの処理段階別レイテンシ比較

今回の評価では，この計算機を 2 台用いて，ミラーリングを行うノードを構成した．現時点における評価環境は，最終的なデータの記録先であるフラッシュメモリデバイスを備えていないため，キャッシュからの書き出し（デステージ）部が含まれていない．そのため，図 4 に示す条件判断のうち，キャッシュに余裕があるかフルであるかについての判断はスキップしている．

また，本章において示す評価結果は，8KB ブロックの書き込みを 5 秒間行い，3 回測定した結果の平均値である．

5.2　予備評価

5.2.1　SHA-1 ハッシュ値計算時間

上記評価環境における 8KB ブロックの SHA-1 ハッシュ計算時間は，19.54 μ秒であった．重複排除のための SHA-1 計算は 8KB ブロック単位で行う必要があるため，これを並列で行うことは困難であり，これ以上時間を短縮することは難しい．

5.2.2　各ステップごとの処理時間

図 5 は，dedup-through モードと dedup-back モードにおいて，内部の各処理に要している時間の内訳を示したものである．尚，dedup-back モードにおいては，再配置の伴う処理は書き込み応答時間には含まれないため省略している．図から，ハッシュ計算時間が dedup-through モードの応答時間のうち，3 割近くを占めていることがわかる．dedup-back モードではこの分を応答時間から除外できるため，書き込みレイテンシを短縮することができる．

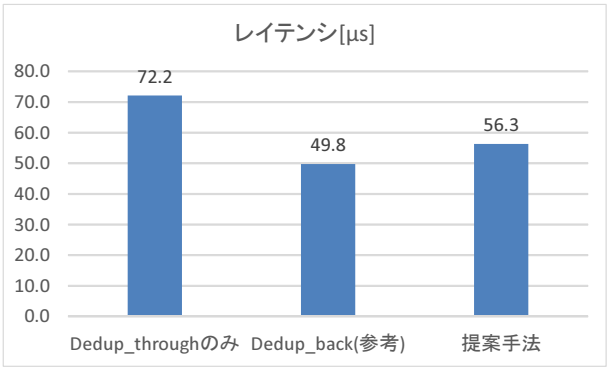

図 6　Dedup-through のみ (ナイーブな重複排除ストレージ) と，提案手法の書き込みレイテンシ比較

5.3　書き込みレイテンシと考察

図 6 は，インライン重複排除をナイーブに実装した場合 (dedup-through モードのみ) のレイテンシと，提案手法および参考として dedup-through モードのみの書き込みレイテンシを比較したグラフである．

提案手法の利用により，ナイーブな重複排除実装に比べ，16 μ秒のレイテンシ短縮を実現することができており，適切に I/O 手順を選択できていることがわかる．

6.　まとめと今後の課題

6.1　まとめ

本稿においては，一般的には書き込みレイテンシを増大させる重複排除処理について，ハッシュ値計算のタイミングが異なる二つの書き込み手順を用意したシステム上において，それらの書き込み手順を動的に選択することにより，レイテンシ短縮を実現する手法を提案した．また，ベンチマークにより，ナイーブに重複排除を行う場合と比較して，実際に書き込み応答時間が短縮されることを示した．

本提案手法は，性能要件の範囲内でレイテンシを短縮する手段を提供するものであることから，本稿と同様の書き込み手段を取り得る重複排除ストレージであれば，応用が可能である．

6.2　今後の課題

本稿における提案は，I/O 手順の選択比率を求めるものであり，今回の実装においては確率的な振り分けを行っている．しかしながら，そのようにして得られたストレージシステムのレイテンシ短縮効果が，実際にアプリケーションの応答時間を短縮できるかどうかについては検討の余地がある．加えて，本稿において評価に用いたシステムは，性能評価の章でも述べたように，キャッシュからデバイスにデステージを行う部分が含まれていない．最終的なストレージデバイスにはフラッシュメモリをはじめとするデバイスが考えられるが，データの書き込みパターンによっては必ずしも性能が一定ではないため，持続的な書き込み性

能について正確に知るためには，デステージを含めた検討が必要であり，この点は今後の課題である．

参考文献

[1] Hey, T., Tansley, S. and Tolle, K.(eds.): *The Fourth Paradigm: Data-Intensive Scientific Discovery*, Microsoft Research, Redmond, Washington (2009).
[2] Meyer, D. T. and Bolosky, W. J.: A Study of Practical Deduplication, *Proceedings of the 9th USENIX Conference on File and Stroage Technologies*, FAST'11, USENIX Association, pp. 1–1 (2011).
[3] Mandagere, N., Zhou, P., Smith, M. A. and Uttamchandani, S.: Demystifying Data Deduplication, *Proceedings of the ACM/IFIP/USENIX Middleware '08 Conference Companion*, Companion '08, pp. 12–17 (2008).
[4] Rahm, E.: Performance Evaluation of Extended Storage Architectures for Transaction Processing, *SIGMOD Rec.*, Vol. 21, No. 2, pp. 308–317 (1992).
[5] Quinlan, S. and Dorward, S.: Venti: A New Approach to Archival Storage, *Proceedings of the Conference on File and Storage Technologies*, FAST '02, USENIX Association, pp. 89–101 (2002).
[6] Dubnicki, C., Gryz, L., Heldt, L., Kaczmarczyk, M., Kilian, W., Strzelczak, P., Szczepkowski, J., Ungureanu, C. and Welnicki, M.: HYDRAstor: A Scalable Secondary Storage, *Proccedings of the 7th Conference on File and Storage Technologies*, FAST '09, pp. 197–210 (2009).
[7] NetApp ONTAP 9: (online), available from ⟨http://www.netapp.com/jp/products/platform-os/ontap/⟩.
[8] HPE 3PAR StoreServ: (online), available from ⟨https://www.hpe.com/jp/ja/storage/3par.html⟩.
[9] Introduction to the EMC XtremIO storage array (Ver 4.0): (online), available from ⟨https://www.emc.com/collateral/white-papers/h11752-intro-to-XtremIO-array-wp.pdf⟩.
[10] Tsuchiya, Y. and Watanabe, T.: DBLK: Deduplication for Primary Block Storage, *Proceedings of the 2011 IEEE 27th Symposium on Mass Storage Systems and Technologies*, MSST '11, pp. 1–5 (2011).
[11] Srinivasan, K., Bisson, T., Goodson, G. and Voruganti, K.: iDedup: Latency-aware, Inline Data Deduplication for Primary Storage, *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, FAST'12, USENIX Association, pp. 24–24 (2012).
[12] Costa, L. B., Al-Kiswany, S., Lopes, R. V. and Ripeanu, M.: Assessing data deduplication trade-offs from an energy and performance perspective, *Green Computing Conference and Workshops (IGCC), 2011 International*, pp. 1–6 (online), DOI: 10.1109/IGCC.2011.6008567 (2011).
[13] Fu, M., Feng, D., Hua, Y., He, X., Chen, Z., Xia, W., Zhang, Y. and Tan, Y.: Design Tradeoffs for Data Deduplication Performance in Backup Workloads, *13th USENIX Conference on File and Storage Technologies (FAST 15)*, Santa Clara, CA, USENIX Association, pp. 331–344 (online), available from ⟨https://www.usenix.org/conference/fast15/technical-sessions/presentation/fu⟩ (2015).
[14] NIST: Secure hash standard, *Federal Information Processing Standard, FIPS-180- 1* (1995).
[15] De Cannière, C. and Rechberger, C.: *Finding SHA-1 Characteristics: General Results and Applications*, pp. 1–20 (online), DOI: 10.1007/11935230_1, Springer Berlin Heidelberg (2006).
[16] Wang, X., Yin, Y. L. and Yu, H.: *Finding Collisions in the Full SHA-1*, pp. 17–36 (online), DOI: 10.1007/11535218_2, Springer Berlin Heidelberg (2005).