Vol.2011-HPC-130 No.31
2011/7/28

SSDを用いたオブジェクトベース・ストレージデバイスのI/O性能制御

谷村勇輔[†1]　鯉江英隆[†1,†2]
工藤知宏[†1]　小島功[†1]

HPCクラスタの各ノードで共有されているストレージシステムへのアクセスの集中を防ぐため，我々は性能予約のコンセプトを提案し，予約情報に従ってI/O経路上の資源を割り当て，必要なI/O性能を個々のアプリケーションに対して提供するストレージシステムの研究を進めている．本研究では，提案システムの構成要素であるオブジェクトベース・ストレージデバイスにおいて，バックエンドにSolid State Drive（SSD）の利用を前提としたI/O性能制御手法を実装し，同時アクセスがあった際の個々のアクセスに対するスループットの保証レベルを調査した．そして，性能の分割パターン，計測粒度，同時アクセス数を変えた実験により，同時アクセスとI/O制御に因るオーバーヘッドの見込みに対するスループットの保証レベルを明らかにした．

Performance Control on An Object-based Storage Device Backed by A Solid State Drive

Yusuke Tanimura,[†1] Hidetaka Koie,[†1,†2]
Tomohiro Kudoh[†1] and Isao Kojima[†1]

In order to prevent performance degradation caused by data access congestion from multiple applications simultaneously running on a same HPC cluster, we propose an advance reservation concept in use of the shared storage system. In such the storage system, all components on the I/O path are assigned to the reserved access to satisfy its performance requirement, and actual I/O operations are controlled properly during the reserved time. In this study, we focused on the Object-based Storage Device (OSD) which is an abstraction of the storage device and implemented an I/O scheduling method with performance control capability. Through the experiments of concurrent accesses to our OSD implementation using a Solid State Drive (SSD) as a backend, we investigated the throughput guarantee level with several throughput sharing patterns, measurement granularities, and the numbers of concurrent accesses.

1. はじめに

HPC（High Performance Computing）クラスタの利用においては，全ノードを用いた大規模計算の実行に限らず，各計算ノードを分割して同時に複数のプログラムを実行する利用形態も珍しくない．しかし，そのような場合には，クラスタ全体で共有されている並列ファイルシステムなどのストレージにアクセスが集中し，各アプリケーションの実行性能に大きな影響を及ぼすことがある[1)]．それを防ぐ1つの方法として，並列ファイルシステムにEnd-to-Endでの性能制御機構を組み込み[2)]，アプリケーション毎にアクセス時のQoSを設定することが考えられ，我々はユーザの明示的な予約に基づいてアクセス性能の保証を行うコンセプトを提案している[3),4)]．これはクライアントプログラム，ストレージネットワーク，ストレージデバイス，各サーバ上のバッファスペースなど，クライアントとストレージデバイス間のI/O経路上にある全ての構成要素の制御を必要とする．このうち，ストレージデバイスに関しては，近年の並列ファイルシステム[5)–7)]のアーキテクチャではファイルデータをオブジェクトとして格納できるオブジェクトベース・ストレージデバイス（Object-based Storage Device：OSD）[8)]として抽象化して扱うことから，OSD内部にて実際のデバイスを制御するのが妥当である．このような背景に基づき，我々はこれまで開発を進めてきたOSD[9)]を改良して，OSD内部にスループット（MBytes/sec）に基づくI/O性能制御手法を実装し，その評価を行った．

開発したOSDはOSDの基本機能をソフトウェアで実装し，バックエンドとしてNAND Flashメモリを用いたSolid State Drive（SSD）を利用することを前提としている．これはI/O負荷の特性とSeek時間に影響を受ける従来のハードディスクドライブ（HDD）とは異なり，SSDではSeek時間がほぼ0であり[⋆1]，Readにおいて安定した性能を期待できるためである．しかし，WriteにおいてはSSD内部のGarbage CollectionやWear Levelingの実装に依存した性能のばらつきがあり，さらにSSDの製造会社がそれらの詳細を一般に公開していないことから性能予測も困難である．また，OSDからバックエンドのSSDに対

†1 産業技術総合研究所
National Institute of Advanced Industrial Science and Technology (AIST)
†2 数理技研
SURIGIKEN Co., Ltd.
⋆1 OCZ VERTEXではSeek時間は0.1[msec]である[10)]．

して行われる Write にはオブジェクトとして格納するデータだけでなく，オブジェクトの属性やチェックサム，インデックスなどのメタデータも含まれる．このメタデータの更新のタイミングがアプリケーションレベルで計測される Write 性能に影響を与えてしまう．本研究では，このような性能の不安定要因の影響を調査し，規則的にメタデータの更新と同期処理を行うことでそれらの影響を緩和し，かつトークンを用いた優先 I/O 制御を行う手法を検討する．そして，複数のアクセスが同時に行われる状況下において，どの程度の性能制御，そしてアプリケーションに対する性能保証が可能であるかを明らかにする．

2. SSD を用いた OSD の I/O 性能制御のための課題と要件

並列ファイルシステムなどの分散ストレージにおいてオブジェクトベース・ストレージデバイス（OSD）を利用する利点は，ディスクスペースとデータブロック（例えば，複数のオブジェクトからなるファイル断片）の管理層と，ファイルシステムのようなデータの論理構造をアプリケーションに提示する上位層とが分離され，より柔軟なファイル共有やセキュリティの実装が可能になることである．しかし，上位層から性能制御を行うことを考えると，OSD の機能を拡張し，ディスクスペースの管理だけでなく I/O スケジューリングについても責任を持てる仕組みが望ましい．QoS 機構を備えた OSD の研究は Q-EBOFS[11]，iSCSI OSD[12]，Façade[13] などいくつか存在し，YFQ[14] のようなスケジューリングアルゴリズムを OSD 内部の I/O スケジューリングに適用している．ただし，これらは HDD を対象としており，QoS を達成しつつ，Seek コストを最小化することで合計スループットを高く保つように動作する．SSD をバックエンドに用いた場合は，Seek コストの最小化が不要である一方，SSD の I/O 特性を考慮した QoS の実現方法を考えなければならない．

一般に SSD の Read 性能は 2∼8KB のページ単位のアクセスにおいて数百 μsec であり，Write 性能はその 4∼5 倍である．ただし，Write においては，Flash 上に残っている不要なデータを削除した上でデータを書き込んだり，既存のデータを別の場所にコピー（Read & Write）して新しいデータと合わせて書き込んだりするといった動作が SSD 内部にて行われる．これは Flash の特徴として，Erase 操作がブロック単位（64∼256 ページ）でしか行うことができず，Erase を実行できる回数に制限があること，また Erase 操作 1 回にかかるコストが数 $msec$ と大きいことに起因する．先にも述べたように，SSD 内部の挙動は製造会社や製造モデルに依存し，詳細が不明であるため，SSD の Write 性能の予測は困難である．

SSD の I/O 特性に加えて，OSD の内部処理がさらに性能をばらつかせる可能性があり，特にメタデータの更新については注意が必要である．OSD のメタデータとしては，未割り当てのエクステント（ディスク上の連続した格納領域）やオブジェクト参照表などの B+tree 構造で格納された情報と，使用するエクステントのリスト，オブジェクトのサイズ，チェックサムデータなどオブジェクト毎の情報とがある．OSD はオブジェクトへの Write 操作と合わせてこれらのメタデータを更新する必要があり，QoS を考慮した I/O スケジューリングを行う場合は，OSD はメタデータの更新コストを把握して，それに基づいた制御を行わなければならない．メタデータの更新がジャーナリングによって高速化されている場合においても，OSD はジャーナリングデバイスの性能を把握し，同様の制御を行うことが求められる．

さらに，オブジェクトが増えたことによるメタデータアクセスの遅延，SSD の空き容量の減少など様々な理由により，OSD が提供できる性能が運用中に低下する事態も考えられる．OSD 自身が現在の提供可能な性能を把握し，それをスケジューリングに反映できる仕組みも必要である．

3. 設計と実装

3.1 概　　要

我々が提案する予約に基づいてアクセス性能を保証するストレージシステム全体のアーキテクチャの概要とその中での OSD の位置づけを**図 1** に示す．このストレージシステムでは 1）に示すように，ユーザが明示的に性能を予約できるインタフェースを備える．ストレージマネージャは予約を受け付ける際に資源を割り当て，予約時間帯においては要求された性能を提供できるよう，OSD を含む各コンポーネントを制御する．OSD に対する制御命令は 2）アクセス開始時に Capability という形でストレージマネージャからクライアントに渡され，その後，3）クライアントがストレージサーバに対してファイル断片の ‘open’ 要求を行う際にストレージサーバに渡される．ストレージサーバは受け取った Capability の情報をもとに，そのセッションにおいて 4）引き続き行われる Read または Write において要求性能が満たされるように OSD を制御する．なお，Capability を用いた伝達の仕組みは，パーミッションなど OSD のセキュリティモデルにおいて既に使われており，ストレージマネージャが Capability に署名を行い，ストレージサーバはその署名を検証することで Capability の情報が正しいことを確認できる．

今回開発した OSD は SSD をバックエンドに用い，EBOFS（Extent B-tree based Object Storage）[15] をもとにストレージ容量の予約および I/O 制御機能を拡張実装したものである．ストレージ容量の予約は以前に実装しており[9]，ディスク領域を使用中，予約済み，空

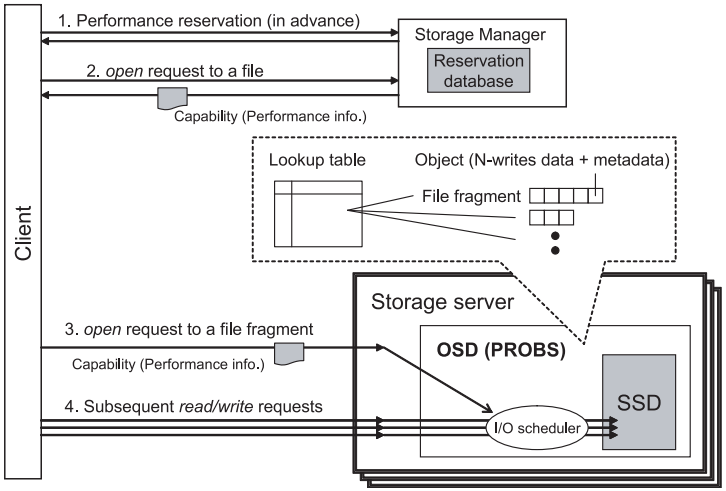


図 1 アクセス性能を保証するストレージシステムと QoS を提供する OSD の位置づけ

きの 3 状態で管理し，固定長のブロック単位での予約を実現している．今回は I/O スケジューリングについて改造を行い，個々のアクセスが指定した性能を満たすように動作する優先 I/O 制御手法（後述）を実装した．本研究報告では以降，この拡張した EBOFS を PROBS と記す．EBOFS 同様，PROBS 自体は ‘open’ に相当する API がなく，オフセットと長さを指定してオブジェクトにアクセスするインタフェースを提供する．そこで，新たに allocate_{read/write}_throughput() の 2 つの API を用意し，それらの返り値である性能識別子を read() や write() の引数として渡せるようにした．図 1 のストレージサーバはクライアントからの ‘open’ 要求の際に allocate_{read/write}_throughput() を呼び出し，個々のアクセスの性能要求を PROBS に伝えることになる．なお，EBOFS/PROBS はユーザ空間で動作し，O_DIRECT でストレージデバイスを open するため，カーネル内のバッファを経由せずに I/O を行う．

3.2 前提条件と性能指標

性能予約・保証を行うにあたっては最初に前提条件とそれに基づく性能指標を定義した．そもそも本研究ではデータステージングやマルチメディアストリーミングなど，一定時間をかけて数 GB 以上の大量のデータの入出力を行うアプリケーションを想定している．すなわち，各アクセスは Read または Write のいずれかであり，ファイル断片の ‘open’ から ‘close’ までの 1 つのアクセスにおいて Read と Write の混在がない．Write においては既存データの書き換えを伴わない逐次アクセスのみを考える．さらに，図 1 のクライアントからストレージサーバおよび OSD へのアクセスは，固定長のストライプサイズでのアクセスとなるため，OSD へのアクセスはより単純化される．最適なストライプサイズはアプリケーションやハードウェアに依存するが，PROBS ではデータ割り当てサイズ（Allocation Unit：AU，デフォルトは 1MB）[9] の倍数に制限する．このような OSD へのアクセスに対して，どれくらいのデータ量を Read あるいは Write できたかというスループット（MBytes/sec）を性能指標に用い，PROBS ではそれを満たすように I/O 制御を行う．

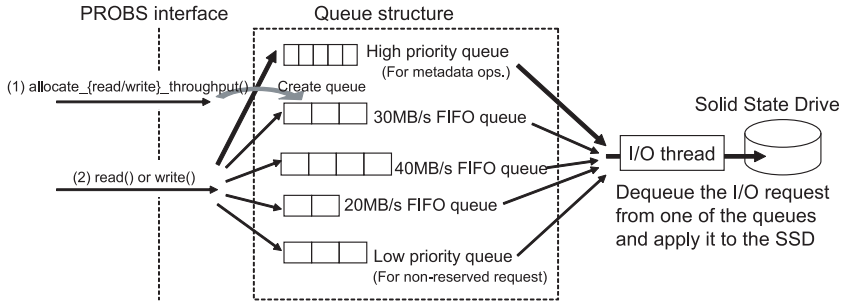


図 2 アクセス性能別に用意されたキューの構成例

3.3 優先度付け I/O スケジューリング

各アクセスが指定したスループットを満たすように I/O 制御を行うため，トークンを用いた Weighted Round Robin（WRR）に基づく I/O スケジューリング手法を PROBS に実装した．具体的には**図 2** に示すように，PROBS は allocate_{read/write}_throughput() が呼ばれるたびに FIFO キューを動的に作成する．つまり，同時に N 個の独立したアクセスがある場合は N 個の FIFO キューが存在する．スケジューリングは指定した I/O（I/O サイズは 1AU）の回数を 1 ラウンドとし，ラウンド毎に行う．まず，各ラウンドの最初に I/O 回数分のトークンを用意し，個々の FIFO キューに対して要求性能に比例したトークンを割り当てる．次に，I/O スレッドが最も多くトークンを保持する FIFO キューから I/O 要求を取り出し，処理を実行する．I/O 要求が処理されるたびにその FIFO キューは I/O サイズ分のトークンを失うため，トークンが他のキューよりも小さくなると I/O スレッド

は別の FIFO キューにある I/O 要求を処理するようになる．現在の実装では，ラウンドの実行中に新しい FIFO キューが作成された場合は，次のラウンドからその FIFO キューに要求性能に見合う数のトークンが割り当てられることになる．

一方，FIFO キューとは別に High Priority キューと Low Priority キューを 1 つずつ用意する．High Priority キューはメタデータ操作のためのキューであり，I/O スレッドは上記のトークンの仕組みとは無関係に最優先でこのキューにある I/O 要求を処理する．この処理はラウンド毎の I/O 回数にも含めない．また，FIFO キューとは異なり，High Priority キューでは細粒度の I/O を効率的に処理するため，複数の I/O 要求の順序の入れ替えや統合を許している．メタデータ操作に伴う I/O は，AU の倍数であるデータアクセスの I/O に比べれば小さいが，PROBS が提供可能な性能を見積もる際にはこのメタデータ操作分のオーバーヘッドを見込まなくてはいけない．

各ラウンドの最後には PROBS はコミット処理を実行し，FIFO キュー由来の I/O に関する未処理のメタデータがあればそれを全て処理する．そして，そのラウンドでの各 FIFO キューのスループットや SSD に対する平均スループットを算出する．現在はベンチマーク用にこれらの情報を記録しているが，将来的には上位にフィードバックするインタフェースを用意し，新規の性能予約や ‘open’ を受け付けられるかどうかなどの判定に利用できるようにすることを検討している．

上記のスケジューリングの実装においては，キュー内で待機している I/O 要求のためにバッファが必要である．バッファは SSD から読まれるデータ，SSD に書き込まれる前のデータを保持しておくために用いられる．PROBS では個々のアクセスの要求性能が異なるため，確保したバッファを要求性能に応じた比率で割り当てを行うように実装している．

4. 評価実験

今回開発した PROBS を評価するため，初めに単一アクセスにおけるスループットとその安定性を検証した上で，同時に複数のアクセスを行った場合の優先 I/O 制御手法による性能の保証レベルを調査した．以降の実験では，PROBS にアクセスする際の I/O サイズは全て 1MB とし，4 回分の I/O に相当する 4MB をオブジェクトサイズとした．そして，スケジューリングの 1 ラウンドにおける I/O 回数（以降，SROUND と記述）には 4 の倍数を用いた．スループットは PROBS の API 呼び出しの前後，PROBS 内部の両方で測定しているが，ここでは断りのない限り前者の測定結果を示す．なお，実験は全て SSD として OCZ VERTEX[10] を搭載した**表 1** のマシンを用いて行った．

表 1 実験環境

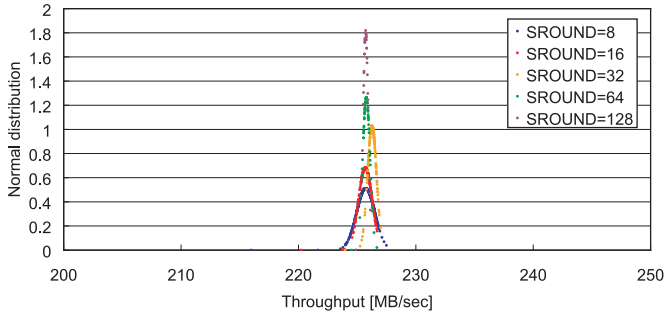
CPU	AMD Opteron 8Core 6128 2.0 GHz
Memory	8GB memory
Disk	OCZ VERTEX (AHCI & write cache: enabled)
OS	CentOS 5 (Kernel version 2.6.18)

表 2 直接アクセスと PROBS 経由のアクセスの性能比較

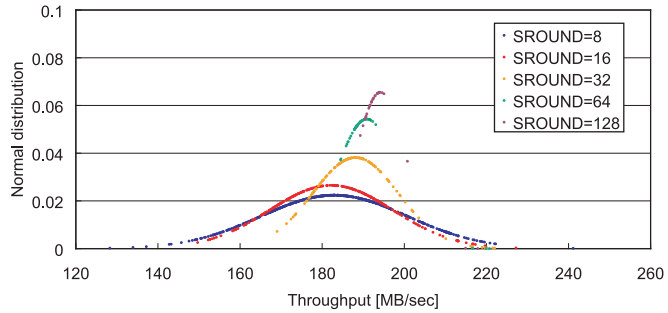
		Average [MB/sec]	Standard Deviation
Read	Direct	256	1.97
	PROBS	236	1.24
	PROBS-CSUM	196	1.06
Write	Direct-INIT	221	18.7
	Direct	203	34.7
	PROBS-INIT	190	21.3
	PROBS	172	26.5
	PROBS-CSUM	166	18.0

4.1 単一アクセス時の性能の安定性

単一アクセス時の性能測定では，まず 1MB の逐次アクセスを SSD に対して直接行った場合（Direct）と同様のアクセスを PROBS を通して行った場合（PROBS）のスループットを測定した．Direct は 1MB の I/O 単位で測定し，PROBS は SROUND を 4 に設定してオブジェクトサイズ単位，すなわち 4MB 単位で測定した．**表 2** は SSD の全容量の 80% までデータを Write した後，それを先頭から Read した場合のそれぞれの平均スループットと標準偏差を示している．表中の Write の INIT は SSD の全領域を空にした初期状態での測定，PROBS の CSUM はチェックサムの計算を有効にした場合の測定結果である．Read の性能はいずれも安定しており，PROBS のオーバーヘッド，さらにチェックサム計算のオーバーヘッドが加わると性能のばらつきはさらに少なくなる．それに対して，Write の性能は大きくばらついており，SSD が初期状態であっても PROBS のオーバーヘッド等が加わってもばらつきは大きい．表 2 は全アクセスについての平均と標準偏差を計算しており，オブジェクト数の増大やディスク使用量の増加に伴う性能低下の影響が考えられるため，Write の PROBS についてディスクの使用量が 1)0∼20%，2)20∼40%，3)40∼60%，4)60∼80% である区間別の解析も行った．結果は平均スループットは 1) から 4) の間で約 4% 低下したが，区間別の標準偏差はほぼ一定で平均 26.3 であった．つまり，性能低下の影響が小さく

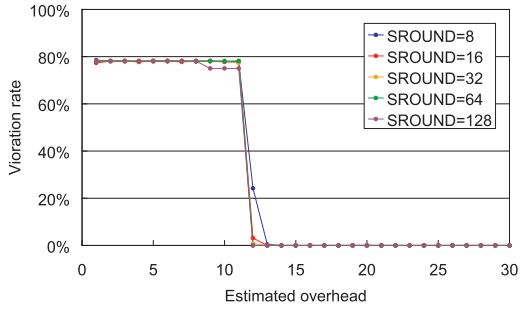


a) Read

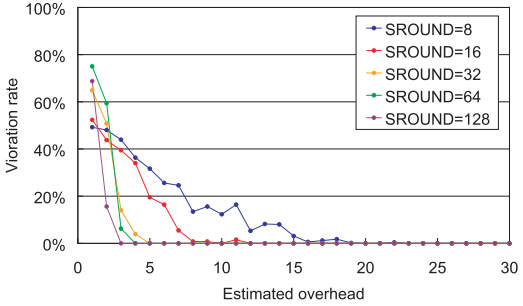


b) Write

図 3 PROBS のスループットのばらつき（SROUND 別）



a) Read



b) Write

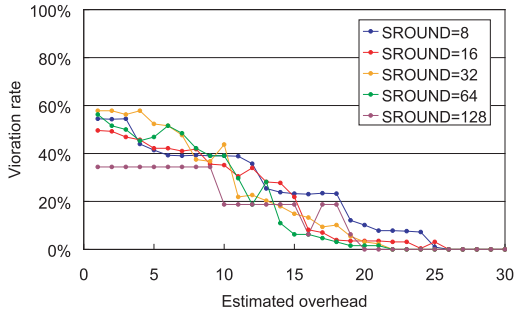
図 4 同時に 2 つのアクセスを処理した場合のスループットの保証レベル（等分割）

ても性能のばらつきは一定程度ある結果となった．

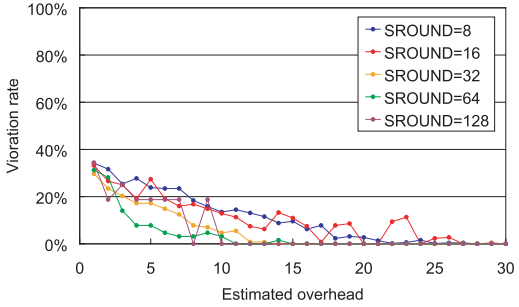
次に，PROBS の SROUND を 8 から 128 まで増やしてスループットを測定し，そのばらつきを調べた．**図 3** は 4GB のデータを逐次的に Read または Write し，SROUND 別に各計測値の分布曲線を描いてそのばらつきを比較したものである．図より Read，Write ともに SROUND を増やすことで見かけ上の性能のばらつきが減っているのが分かる．もちろん，個々の I/O の性能はばらついたままであり，どの程度の SROUND での性能の安定性を許容するかはアプリケーションが選ぶことになる．また，性能保証の観点では，ばらついた時の最低値が保証できる性能値になるため，ばらつきが大きい場合にはより余裕をもってスケジューリングを行わないといけない．

4.2 同時アクセス時の性能保証

同時アクセス時の性能測定では，実装した優先 I/O 制御手法による性能の保証レベルを評価する実験を行った．最初の実験では，PROBS に対して同時に 2 つの Read アクセス，あるいは 2 つの Write アクセスが行われる状況において，各アクセスが全体のスループットを等分割した場合と 2:1 の割合で比例分割した場合のスループットを測定した．分割する「全体のスループット」は前節の図 3 に示した単一アクセス時の平均スループット（以降，基準スループットと記述）から同時アクセスにおけるオーバーヘッドを 1~30% 見込んで，その分を差し引いた値とした．そして，本研究では最終的に性能保証を目的としているため，測定



a) Read



b) Write

図 5 同時に 2 つのアクセスを処理した場合のスループットの保証レベル（2:1 の分割）

したスループット値のばらつきではなく，分割したスループット値 [*1] を下回る割合を要求性能の「違反率」として定義し，その割合を求めた．例えば基準スループットが 200[MB/sec] であり，それに 10% のオーバーヘッドを見込んだ場合，等分割では 85[MB/sec] のスループットを下回る測定値の検出回数を数え，全体の測定回数に対する割合を調べた．なお，各アクセスは 2GB のデータの Read または Write を行い，1 つ目のアクセスの開始から 2 秒後に次のアクセスを開始した．アクセスが競合する時間は実験ケースにより異なるが，いずれも 10 秒以上継続することを確認している．

*1 先に述べたように，この値は allocate_{read/write}_throughput() により PROBS に通知される．

図 4 が 1:1，**図 5** が 2:1 の割合で分割した場合の実験結果である．グラフの縦軸の違反率（Vioration rate）は個々のアクセスの違反率を平均した値である．図 4 の結果は，Read アクセスの制御には 15% 以上のオーバーヘッドを見込む必要があり，10% 以内のオーバーヘッドでおおよそ制御できている Write よりも大きな見込みが必要であることを示している．図 5 においても同様に，必要なオーバーヘッドの見込みは Write よりも Read の方が大きい．この原因は，Write の場合には若干のバッファキャッシュが作用し，同時アクセス時の SSD への合計スループットが単一アクセス時とほぼ同等であるのに対し，Read の場合には SSD への合計スループットは基準スループットの 90~92% に落ち込んでいることにある．分割方式の比較では，比例分割においては片方のアクセスの要求性能が低く，違反率が小さいため，両アクセスの違反率が同程度である等分割に比べて，平均違反率は低めとなる．しかし，違反率を 0 に近づけるのに必要なオーバーヘッドは比例分割の方が大きく，比例分割の方が性能保証が困難であるといえる．

2 つ目の実験では SROUND を 64 に固定し，同時アクセス数を 2 から 8 まで増やした時の個々のアクセスに対する性能の保証レベルを調べた．これまでと同様，同時に実行されるアクセスは Read または Write のいずれかとし，アクセス数を N，基準スループットを $BaseThput[MB/sec]$，基準スループットに対するオーバーヘッドの見込みを $Overhead[\%]$ とした時，i（$1 \leq i \leq N$）番目のアクセスが要求するスループット $Thput_i$ を次のように計算して実験を行った．

$$Thput_i = BaseThput \times \frac{(100 - Overhead)}{100} \times \frac{2}{N(N+1)} \times i \qquad (1)$$

図 6 では Read と Write のそれぞれにおいて，性能の違反率を 10% 未満に抑えることのできるオーバーヘッドの割合（Estimated overhead）と，その時に PROBS 内部で計測した全アクセスの合計スループットの基準スループットに対する性能低下の割合（Throughput degradation）の 2 種類のグラフを示している．前者のグラフからは，同時アクセス数が 6 までであれば見込みオーバーヘッドの割合は 15% 程度で済み，同時アクセス数が 7 を超えると見込みオーバーヘッドの割合が急激に高くなることが分かる．つまり，7 アクセス以上の競合は効率性の観点から避けた方が良い．後者のグラフでは，Read の合計スループットはアクセス数に依らずほぼ一定であり，Write についてはアクセス数が増えるにつれて低下しているものの，基準スループットに対する割合は Read より高く 90% 以上である．そこで，同時アクセス数が 7 以上の結果についてさらに調べた結果，Read については合計スループットは足りており，個々のアクセス性能にばらつきがある状態であるため，SROUND を大き

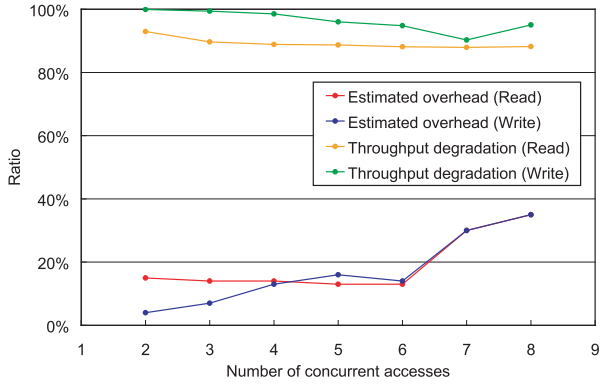


図 6 アクセス数に対する性能保証レベル

くすることで性能の違反率を減らせる可能性があることが分かった．一方，Write については同時アクセス数の増加により合計スループット自体が大きくばらついており，SROUND を大きくすることによる改善は望めないことが分かった．

4.3 考　　察

前節までに述べた実験により，性能の保証レベルを個々のアクセスが求めた性能の違反率という形で調査し，同時アクセス時のオーバーヘッドを適度に見積もることで一定レベルの優先 I/O 制御ができることを示した．しかし，オーバーヘッドの見積もりと性能の保証レベルの関係は SSD のモデル，要求される性能のパターン、同時アクセス数などによって変化する．SSD の性能モデルが開示されれば，それをもとにモデル式を作成して導きだすことも可能であるが，そうでない場合には今回のような実験をベンチマークとしてまとめ，自動的に実験を行うような仕組みが必要になるであろう．その際に SSD をブラックボックスとし，機械学習アルゴリズムなどを用いて性能モデルを構築する方法も考えられる．

また，性能の違反率が 0~10% となる実験結果が比較的多く見られ，優先 I/O 制御手法についてもいくつか改善が必要だと考えている．特に，新たなアクセスが開始される際，そのアクセスに関連づけられた FIFO キューは次の SROUND において初めてトークンが割り当てられるため，アクセス開始直後に性能違反を起こし，結果として違反率が 0% にならないことがある．これについては，実行中のスケジューリング・ラウンドにおいてトークンを再割り当てするような仕組みを導入するなどの解決策がとれるであろう．

5. ま と め

本研究では予約に基づいて性能保証を行うストレージシステムの実現に向けて，その構成要素の 1 つである OSD の優先 I/O 制御を検討し，スケジューリング・ラウンド毎にトークンを用いた WRR により，個々のアクセスが要求した性能に応じて I/O 処理を行う手法を実装した．また，Write アクセスでは OSD のメタデータ更新処理をスケジューリング・ラウンド単位で行うようにし，性能のばらつきを軽減した．評価実験では単一アクセスにおける性能の安定性を検証した後，同時アクセスにおける性能の保証レベルを性能の分割パターンと計測粒度を変えて調べ，さらに同時アクセス数の増加についても同様の調査を行った．その結果，Write の同時アクセスを含む多くのケースにおいて，15~20% のオーバーヘッドを見積もることで性能の違反率を 10% 以下に抑制できる I/O 制御が行えること，同時アクセス数を 6 程度まで増やせることを確認した．

今後は優先 I/O 制御手法をさらに改善して，より細かい計測粒度で性能を保証できることを目指すとともに，今回調査したような性能の保証レベルを自動的に見つける方法の検討などを行っていく予定である．

謝辞　本研究の一部は日本学術振興会科学研究費補助金（23680004）の助成，および文部科学省イノベーションシステム整備事業によるものである．

参 考 文 献

1) Lofstead, J., Zheng, F., Liu, Q., Klasky, S., Oldfield, R., Kordenbrock, T., Schwan, K. and Wolf, M.: Managing Variability in the IO Performance of Petascale Storage Systems, *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pp.1–12 (2010).
2) Bigelow, D.O., Iyer, S., Kaldewey, T., Pineiro, R.C., Povzner, A., Brandt, S.A., Golding, R.A., Wong, T.M. and Maltzahn, C.: End-to-end Performance Management for Scalable Distributed Storage, *Proceedings of the Petascale Data Storage Workshop* (2007).
3) 谷村勇輔，鯉江英隆，工藤知宏，小島　功，田中良夫：アクセス性能を保証する並列ファイルシステムの提案とストレージサーバの設計，情報処理学会研究報告，Vol.2009–HPC–121, No.33, pp.1–8 (2009).
4) Tanimura, Y., Koie, H., Kudoh, T., Kojima, I. and Tanaka, Y.: A Distributed Storage System Allowing Application Users to Reserve I/O Performance in Advance for Achieving SLA, *Proceedings of the 11th ACM/IEEE International Conference on Grid Computing*, pp.193–200 (2010).

5) Lustre: http://www.lustre.org/.
6) Welch, B., Unangst, M., Abbasi, Z. and Gibson, G.: Scalable Performance of the Panasas Parallel File System, *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, pp.17–33 (2008).
7) Weil, S.A., Brandt, S.A., Miller, E.L., Long, D. D.E. and Maltzahn, C.: Ceph: A Scalable, High-Performance Distributed File System, *Proceedings of the 7th Conference Operating Systems Design and Implementation (OSDI'06)*, pp.307–320 (2006).
8) Mesnier, M., Ganger, G.R. and Riedel, E.: Object-Based Storage, *IEEE Communications Magazine* (2003).
9) 谷村勇輔，鯉江英隆，工藤知宏，小島　功，田中良夫：予約利用可能なオブジェクトベース・ストレージの設計，情報処理学会研究報告，2009–HPC–119，Vol.2009, No.14, pp.79–84 (2009).
10) OCZ Technology Group Inc.: OCZ Vertex Product Sheet, http://www.ocztechnology.com/ocz-vertex-series-sata-ii-2-5-ssd.html.
11) Wu, J.C. and Brandt, S.A.: Providing Quality of Service Support in Object-Based File System, *Proceedings of the 24th IEEE Conference on Mass Storage Systems and Technologies (MSST 2007)* (2007).
12) Lu, Y., Du, D. H.C. and Ruwart, T.: QoS Provisioning Framework for an OSD-based Storage System, *Proceeding of the 22nd IEEE / 13th NASA Goddard Conference on Mass Storage Systems and Technologies*, pp.28–35 (2005).
13) Lumb, C.R., Merchant, A. and Alvarez, G.A.: Façade: virtual storage devices with performance guarantees, *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, pp.131–144 (2003).
14) Bruno, J., Brustoloni, J., Gabber, E., Mcshea, M., Özden, B. and Silberschatz, A.: Disk Scheduling with Quality of Service Guarantee, *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, Vol.2, pp.400–405 (1999).
15) Weil, S.A.: Leveraging Intra-object Locality with EBOFS, Technical Report UCSC CMPS-290S (2004).