iopsあたりのbusy率を用いたIO性能保証方法の提案

大　江　和　一†

ハードディスクから取り出せる最大性能（iops）は，IO size，read/write の割合，IO が発生するボリューム上の位置の分布と各位置ごとの IO 量に影響を受けて変動することが知られている．そのため，ハードディスクからとりだす性能（iops）を保証するには，これらパラメータの変化で変動する最大性能（iops）をどうやって把握するのかが課題になる．そこで取り出せる性能値（iops）の変動を OS が提供する統計情報から計算出来る「1iops あたりの IO busy 率」として常時数値化し，この値を使用して IO 性能保証を行う方法の提案を行う．

検証はファイル共有ワークロードとバックアップワークロードを競合させる 3 ケースで行った．いずれのケースでも提案手法で IO 性能保証可能であることが分かった．

Method of guaranteeing IO performance that uses IO busy rate for each iops.

Kazuichi Oe†

It is well known that maximum performance (iops) of a hard disk is influenced by access position and range on the volume, the distribution of IO size, the ratio of read/write and etc.. In order to guarantee the IO performance for the specified application, it is needed to monitor the current maximum IO performance dynamically. We propose the method of guaranteeing the IO performance by using the ”Rate of IO busy for each iops. The ”Rate of IO busy for each iops” can be calculated from statistical information of the operating system. We have verified the effectiveness of this method by using the real samba workload, backup workload and the mixture of them

1. はじめに

ストレージ装置から取り出せる最大性能はワークロードの違いによって変化することが知られている．ワークロードとは，例えばアプリケーションが実行する IO のボリューム上の位置の分布や read/write の割合の変化を指す．この理由は，seek などハードディスクの機械的なオーバヘッドやストレージ装置に内蔵されているキャッシュの効果が変わるためである．このことは，複数のアプリケーションでストレージ装置を共有した場合，どのようなアプリケーションを一緒に動作させるかで，あるアプリケーションの IO 性能が変わってしまうことを意味する．現在，複数の仮想 OS でストレージ装置を共有することが行われているが，独立であることが望まれる仮想 OS の IO 性能が，お互いに干渉してしまうという問題が引き起こされる．

本研究の目的は，RAID 装置等ハードディスクを使用したストレージ装置を複数のアプリケーションや仮想 OS で共有する場合，特定のアプリケーションや仮想 OS の性能を保証することである．

IO 性能保証方法の設計の準備としてストレージ装置の性能特性を調査した．その結果，ストレージ装置に与える負荷と取り出せる性能に関して，以下の関係の成立が分かった．

- IO busy 率が低い間は，IO busy 率と iops は線形増加関係
- IO busy 率がある値「比例限界 busy 率」を超えると，IO busy 率と iops は非線形増加関係

この知見を活かして，性能保証したいアプリケーション以外のアプリケーション負荷を調整することで，ストレージ装置にかかる全体の負荷を比例限界 busy 率以下に収めて IO 性能保証を行う方式を考案し，その有効性を評価した．

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**でストレージ装置に与える負荷と取り出せる性能に関する考察に関して説明する．**4 章**で提案する IO busy 率を用いた IO 性能保証方法の説明を行う．

† （株）富士通研究所
Fujitsu Laboratories Ltd.

5 章で検証方法とその結果を報告し，**6 章**をまとめとする．

なお，本稿では特に注釈がない限り，性能とは，実際にストレージ装置が送出する 1 秒当たりの IO 数（iops）とする．ストレージ装置は，ハードディスクを使用した装置を想定する．また，PC 側から見えるストレージ装置の領域をボリュームと呼び，PC 上のアプリケーションが実行する一連の IO 処理をワークロードと呼ぶ．

2. 関連研究

ハードウエアレベルで IO 性能保証を行う研究として，PC とストレージ装置の間に専用のコントローラ（Facade controller）を挿入する提案が行われている 1)．システム管理者は，ワークロードごとに SLO（service-level-objective）の登録を Facade controller に対して行う．SLO は request 数ごとの read/write レイテンシのグラフとなっている．Facade controller は，各ワークロードごとにレイテンシと iops の測定を常時行っている．Facade controller による性能保証方法は，この測定値があらかじめ登録された SLO と一致するように各ワークロードごとの IO queue 長を調整することで行われる．この方法では，ワークロードの変化に伴うストレージ装置から取り出せる最大性能の変化を正確にとらえることが出来ない．そのため，本稿の提案のように保証したい iops を指定した性能保証は難しいと考えられる．

VM を含む OS レベルで IO 性能保証を行う研究として，VM が動作する複数の PC 間で 1 つのストレージ装置を共有する場合の性能保証方法の提案が行われている 2)．システム管理者は，複数の PC 間の性能調停を行うために IO share の定義を行う．IO share は各 PC ごとの IO 性能の配分比率を定める．各 PC は read/write レイテンシの測定を常時行い，各 PC 間で行う flow control にこの情報を含めておく．flow control では，各 PC の read/write レイテンシと IO share の定義より各 PC の IO queue 長を決め各 PC に通知する．各 PC はこの IO queue 長に更新することで IO share で定義した比率で IO 性能の共有を行う．PC 内の各 VM 間は SFQ を用いて各 VM ごとの IO queue 長の更新を行う．この方法は，最初に紹介した研究 1) と同じく IO queue 長を調整することで VM 間の性能調停を行うため，本稿の提案のように保証したい iops を指定した性能保証は難しい．

OS レベルで IO 性能保証を行う別の研究として，Linux Block IO Controller3) が存在する．Linux Block IO Controller では，複数の IO queue を CFQ でスケジューリングしている．この中で優先度を高く設定した IO queue を優先度が高いタスクや VM に割り当てることで IO 性能保証を実現している．この方法は，既に紹介した研究事例 2)，1) と同じく IO queue を用いた性能保証方法であるが，ワークロードの変化に応じて IO queue 長などを変更する仕組みが存在しない．このため，ワークロードが変化し，ストレージ装置から取り出せる最大性能が変化すると，性能保証出来ないケースがあると考えられる．

ファイルシステムレベルでは，あらかじめ必要な性能（スループット）の予約を行えるシステムの研究が存在する 4)．製品レベルでも Linux XFS 及びその元となった SGI IRIX が IO 性能保証をサポートしている 8)．両者とも，ファイルシステム使用者が保証したい性能値（スループット）をファイルシステム側にあらかじめ登録することで性能の予約を行う．特に XFS は，登録時に保証可能な性能値を動的に計算し，保証の可否の判断を行っている．この方法はストレージ装置のボリュームをファイルシステムで占有する場合は性能保証可能であるが，ボリュームを複数に分割しその一部を使用するケースでは性能保証出来ないケースがでる．

アプリケーションレベルでは，アプリケーションが出す IO 性能をアプリケーションプロセス内で監視しておく．IO 性能があらかじめ決められた閾値を超えると，閾値内に収まる様に一定時間そのプロセスからの IO を止める方法の提案が行われている 5)，6)．この方法は 1 アプリケーションの性能を一定に保つ提案であり，本稿が課題としている最大性能が変動する環境で性能を保証する仕組みの提案はない．

3. ストレージ装置に与える負荷と取り出せる性能に関する考察

ストレージ装置から取り出すことが出来る最大性能はワークロードごとに異なる．具体的には，ワークロードを構成する以下のパラメータの変化に依存して最大性能も変動する．

- IO size 内訳☆と read/write の割合
- IO が発生するボリューム上の位置の分布
- IO が発生するボリューム上の各位置ごとの IO 量

これらのパラメータを OS 等が提供する統計情報のみで把握することは困難である．そのため，より単純なパラメータを求めて実験を行い，以下の知見を得た．

☆ 複数の IO size が混在実行されるケースで size 毎の割合

- IO busy 率と iops の間に正の相関関係が成立
  - IO busy 率が低い間は，IO busy 率と iops は線形増加関係
  - IO busy 率がある値「比例限界 busy 率」を超えると，IO busy 率と iops は非線形増加関係
- IO busy 率と性能が線形増加となる範囲はワークロードによって変化．
- IO busy 率=100%のときの性能が最大性能

この知見の妥当性を，実験を交えながら説明する．実験では，実行するアプリケーションがアクセスするストレージ装置上の位置を変えて，得られる IO 性能を測定している．この結果より，ワークロードの違いにより取り出せる IO 性能がどのように変化するのかを明らかにする．

この特徴はディスク単体だけではなく，RAID 装置にも当てはまる．RAID 装置では，64KB 前後の stripe size で分割されたデータ片が stripe size * ディスク数間隔で格納される．このデータ配置のため，ディスク単体に IO を行った時のボリューム上の位置の分布や各位置ごとの IO 量が，RAID を構成する各ディスクにも 1/(ディスク数) の領域に凝縮されて発生する．

実行するアプリケーションとしては，以下の２種類を用いた．

- ファイル共有ワークロード
  Samba を使用したファイル共有システムを構築し，1000 ユーザ前後にファイル共有サービスの提供を行った場合のワークロード．ファイルシステムは，Veritas VxFS．ボリュームサイズは 500GB である 7)．
- バックアップワークロード
  NetVault ☆ のワークロード分析結果を参考に作成したワークロード．

それぞれの詳細な特性パラメータを（**表 2**）に示す．

実験では，2TB ボリュームの先頭 500GB を A，最後 500GB を B とし，それぞれのアプリケーションにおいて，A，B 別々に負荷を与えた場合と A，B 同時に負荷を与えた場合 (A+B) で，iops，%util，svctm の関係を測定した．%util は Linux での IO busy 率であり，svctm は 1IO の平均実行時間である．両者とも iostat コマンドから取得できる．svctm と iops との関係は，次式で与えられる．

$$iops = 1/svctm$$

使用したハードウエアの仕様は，**表 1** の通りである．

☆ http://www.bakbone.co.jp/products/netvault.html

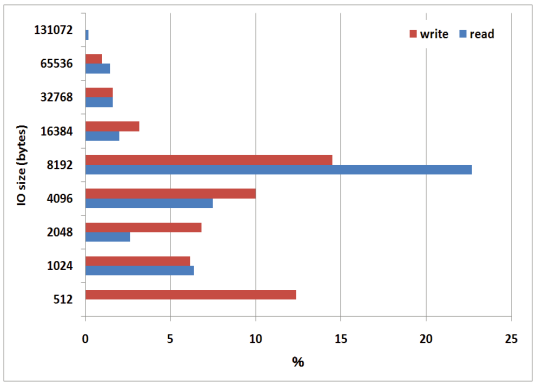

図 1　実験に使用した IO size 分布
Fig. 1　IO size distribution used to experiment

図 3 にファイル共有ワークロードの場合を，**図 4** にバックアップワークロードの場合を示す．いずれも A のみに負荷を与えた場合が最もとりだせる iops が高く，次に B のみに負荷を与えた場合（A のみに比べて，それぞれ約 90%，約 89%の性能），A と B 同時に負荷を与えた場合が最も取り出せる iops が低くなる（A のみに比べて，それぞれ約 70%，69%の性能）．A のみと B のみの性能差は，ディスクの内外周差が原因である．A+B でさらに性能が低下するのはアクセス範囲が広がったことによるシークなどのオーバヘッド増のためと判断できる．このように，ストレージ装置にアクセスする位置が変わると，30%もの性能差が出ることが分かる．また，**図 3** と**図 4** を比較すると，例えば A のみの性能最高値がそれぞれ 239iops，65iops であり，ワークロードの違いにより約 3.7 倍の性能差が出ている．

実験で得られた結果をソース上から裏付ける目的で

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

PC	FUJITSU PRIMERGY RX100S4 Celeron 3.06 GHz,Mem=2GB,CentOS 5.3
disk	Newtech Evolution II 400GB モデル (SATA 2U) 6disk/2TB のパーティションを RAID5 にフォーマット (stripe size=64KB)

表 2　実験に使用したワークロード
Table 2　Workload used to experiment

	ファイル共有	バックアップ
IO size	**図 1**	128KB
read:write	**図 1**	50:50
offset	**図 2**	0-30GB, 300-330GB

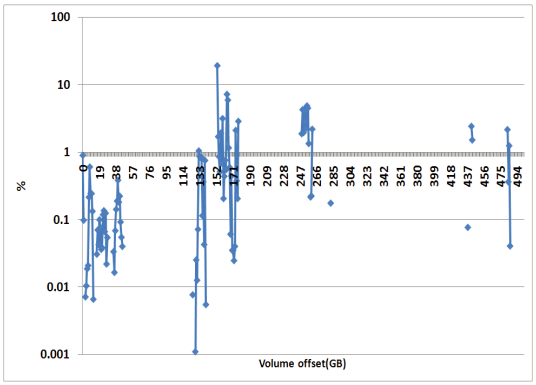

図 2　実験に使用した IO offset 分布
Fig. 2　IO offset distribution used to experiment

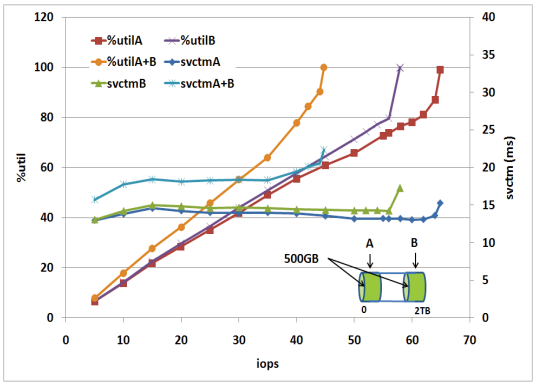

図 4　IO offset と svctm との関係（バックアップ）
Fig. 4　Relation between IO offset and svctm (backup)

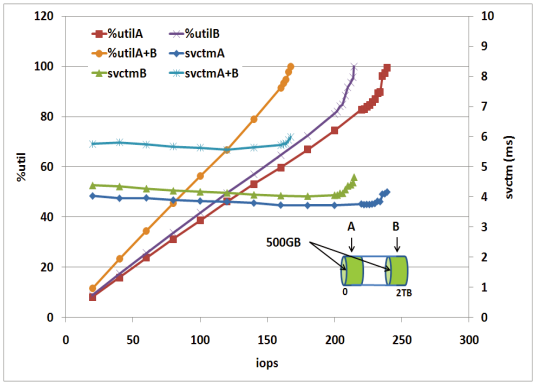

図 3　IO offset と svctm との関係（ファイル共有）
Fig. 3　Relation between IO offset and svctm (file sharing)

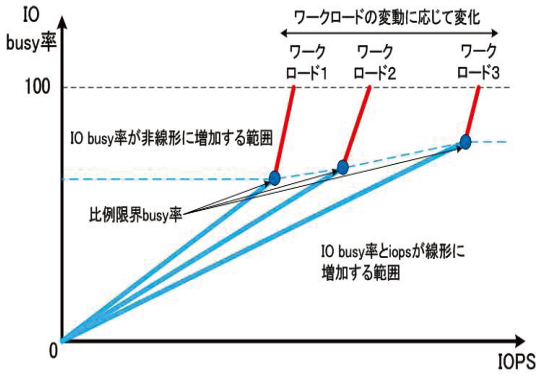

図 5　ワークロード分析結果のまとめ
Fig. 5　Summary of workload analysis result

iostat の実装調査☆を行った．その結果，kernel block device ☆☆ IO queue に実行中の IO が存在した CPU 時間比を%util としていることが分かった．これは，%util=100 では常に IO を実行し続けていることになり，これ以上性能向上しないことになる．

%util と iops との関係をみると，いずれの場合においても，ある%util 値までは取り出せる iops は%util に線形に増加する．その%util を超えると iops の伸びは鈍化し，非線形な関係になっている．%util が 100%で最高性能値に達している．本稿では，iops が線形増加する上限%util 値を「比例限界 busy 率」と呼ぶ（図 5 参照）．比例限界 busy 率はワークロードによって異なり，ファイル共有ワークロードでは A のみ，B のみにおいては約 90%，A+B では約 95%である．バックアップワークロードでは B のみにおいては約 80%，A のみ，A+B では約 85%であった．

比例限界 busy 率はワークロードによって変化するが，それ以下では取り出せる iops が%util に比例するという性質を利用すれば，ワークロードの細かなパラメータを測定することなく，iops を指定した IO 性能保証を行うことが出来る．この方式を次に述べる．

4. IO busy 率を用いた IO 性能保証方法の提案

本稿での IO 性能保証の方針は，IO busy 率が比例限界 busy 率を超えないように制御すること，である．比例限界 busy 率内では，IO busy 率と iops が線形増加関係となる．測定で得た IO busy 率と iops を「1iops 当たりの busy 率」の形にするとワークロードが変化しない限り「1iops 当たりの IO busy 率」はほぼ同じ値を維持することになる．つまり，比例計算で IO 性能保証が可能になる．そこでアプリケーション実行による IO busy 率を比例限界 busy 率を超えない

☆ 2.6.9-67 のソースコードを調査
☆☆ ll_rw_blk.c

ある IO busy 率に誘導することで IO 性能保証を行う．本稿ではこの IO busy 率のことを目標 IO busy 率とする．

運用で得られるデータは例えば図 **6** の様に複数のワークロードを跨いだ点として観測出来る．そのため，何時別のワークロードに移動したのかや何時比例限界 busy 率が変化したのかを把握できない．ワークロードの移動が発生すると「1iops 当たりの IO busy 率」が変化する可能性がある．「1iops 当たりの IO busy 率」が変化したら，IO 性能保証のための比例計算をやり直す必要がある．そこで，iops と IO busy 率を常時モニタし，その値を「1iops 当たりの busy 率」にその都度変換する．「1iops 当たりの busy 率」が変化したら，比例計算をやり直す．

目標 IO busy 率の初期値☆は代表的なワークロードの測定結果を用いる．目標 IO busy 率の更新は，目標 IO busy 率が比例限界 busy 率を超えた場合と目標 IO busy 率と比例限界 busy 率の差が大きくなりすぎた場合に行う．しかし，比例限界 busy 率を直接求めることが出来ないため，OS 統計情報などよりこれら条件を満たしたと判断出来るケースに更新を行う．最初に目標 IO busy 率が比例限界 busy 率を超えた場合の説明を行う．図 **6** などの様な方法で観測される IO busy 率が 100%に達してしまう場合に目標 IO busy 率が比例限界 busy 率を超えたと判断する．目標 IO busy 率が比例限界 busy 率を超えると，IO busy 率と iops は非線形増加の関係になり，些細な負荷変動で IO busy 率が 100%に達してしまう．図 **4** を例にとると，比例限界 busy 率が 80-85%の間に存在し，そこから 2-3iops で IO busy 率が 100% に達してしまう．よって，目標 IO busy 率が比例限界 busy 率を超えていると，大部分は IO busy 率=100%として観測される．

次に目標 IO busy 率と比例限界 busy 率の差が大きくなりすぎた場合の説明を行う．可能性があるのは以下の 2 つのケースである．これらのケースでも目標 IO busy 率の更新を行う．

- 観測された IO busy 率が目標 IO busy 率を大きく下回った場合
- 現在の「1iops 当たりの busy 率」が直前の「1iops 当たりの busy 率」との比較で大きく変化した場合

本稿での IO 性能保証の制御方針について説明しておく．本稿の提案手法は，ストレージ装置の性能を使いきった中での IO 性能保証を目指している．そのため，決められた性能保証値を一時的に守れなくなることは容認する．

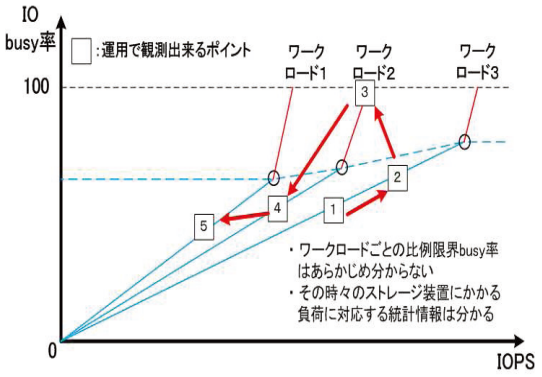

図 **6** 運用時に得られる情報
Fig. 6 Workload information obtained when operating it

具体的な提案手法を説明する．IO 性能保証を行うアプリケーションを A，性能保証する性能値を a iops，性能保証を行わないアプリケーションを B，A と B の合計 IO busy 率の目標値を X%とする．X は目標 IO busy 率のことであり，比例限界 busy 率以下に設定する必要がある．

提案手法はまず性能保証を行いたいボリュームの iops や IO busy 率を常時モニターしておき，以下の計算方法で最新の「1iops あたりの IO busy 率」の把握を行う．

busy_per_iops = IO busy 率/iops

Linux で実装する場合は，iostat -x コマンドを常時モニターしておき，以下の計算を行う．

busy_per_iops = %util/(r/s + w/s)

「1iops あたりの IO busy 率」が変動した場合は，以下の計算を行うことで B 側が使うことが出来る性能（iops）を求め，B 側の性能を絞ることで A 側の性能保証（iops）を行う．

(1) a iops を取り出すのに必要な IO busy 率を求める

a_busy = a * busy_per_iops

(2) 次に B 側が使用することが出来る IO busy 率を求める

b_busy = X - a_busy

(3) 次に b_util から B 側が使うことが出来る性能値（b iops）を求める

b = b_busy / busy_per_iops

(4) B 側のディスクドライバ等に b iops までの帯域制御をかける

☆ 本稿では **3** 章の実験結果より 80%-95%から選ぶことが可能．実験は 95%で行った．

(4) で行う帯域制御方法に関して，**5 章**の検証に使用したプログラムの実装方法を説明する．IO 実行を行うエンジン部分☆1で実行した IO 回数のカウントを行う．IO 実行回数が b を超えてしまったら，1 秒経過するまでの残りの時間の IO をブロックする．1 秒を経過すると IO 実行回数がリセットされ，再び IO 実行を再開する．**5 章**の検証では，全てユーザレベルで実装したベンチマークプログラムを使用した．そのため，提案方式も全てユーザレベルでの実装となった．帯域制御の機能に関しては，本来 kernel driver などに実装する機能と判断している．

次に目標 IO busy 率の更新方法に関して説明する．前述の更新タイミングで提示した条件のいずれかの成立で更新を行う．

A と B の合計 IO busy 率が 100%に達してしまった場合の目標 IO busy 率の更新方法を説明する．はじめに更新を始める条件を説明する．目標 IO busy 率を 100%に近い値に設定すると，ワークロードの変動がなくても些細な負荷変動で一時的に 100%に届いてしまう場合がある．この様なケースを除外するため，今回の評価では IO busy 率が過去 10 回の計測のうち 3 回以上☆2100%となる場合に目標 IO busy 率の見直しを行うことにした．

次に具体的な更新方法を説明する．新しい目標 IO busy 率を New X とすると X ＞ New X の関係が成立する．New X と X の差分を x とすると以下の式で New X を求めることが出来る．

New X = X - x

図 7 のポイント 12 を現在の比例限界 busy 率とすると，New X がポイント 12 以下となる x を求めればよいことになる．x を求めるために，まず以下のデータ取得を行う．

- IO busy 率=100 のときの iops(=C)
- IO busy 率がポイント 12 を下回ったときの iops と IO busy 率☆3

IO busy 率がポイント 12 を下回ったかどうかは，ポイント 1 とポイント 1 以外の測定点の「1iops 当たりの IO busy 率」差分計算結果を使用する．差分が大きいポイントをポイント 12 を下回った（=ポイント 2 相当）とする．ポイント 2 のデータを得るために，IO busy 率が 100%に到達したら性能保証を行わない B 側の目標性能値を提案した計算方法より一時的に減らすことを行う．IO busy 率が 100%に到達すると，実行できなかった IO が kernel 内に蓄積され，A，B とは独立して蓄積された IO の実行が行われる．そのため，B 側の性能をさらに絞ることで kernel 内に蓄積された IO の実行を促進させ，IO busy 率を引き下げる．今回の評価では，IO busy 率=100%の時の A，B の合計 iops の 20 %を本来の B 側の計算値より差し引く実装とした．IO busy 率が 100%を下回るまで継続する．

このデータを用いて，**図 7** のポイント 1 のときの「1iops 当たりの IO busy 率」を求める（=c）．さらにポイント 2 のときの「1iops 当たりの IO busy 率」を求める（=d）．d はポイント 12 の「1iops 当たりの IO busy 率」相当である．ポイント 10 は d を Ciops まで延長したポイントである．ここで x はポイント 1 とポイント 10 の IO busy 率の差（=(c-d)*C）と線形増加関係にあるので以下の式で求めることが出来る．

x = (c-d)*C*y　(y=0 ～ 1)

この結果より，New X は以下の式で求めることが出来る．

New X = X - (c-d)*C*y　(y=0 ～ 1)

一度の更新で正しい New X が求められない場合は，この計算を何回か繰り返すことで New X を求めていく．y の値は使用するストレージ装置や負荷するワークロードの範囲で変化する．y の値が本来の値☆4より大きいと比例限界 busy 率より低い値を New X とすることになり，逆に小さいと New X が比例限界 busy 率より大きくなり再計算が必要になる．y の値は想定されるワークロードを用いて New X の再計算回数を少なくする値を求めておく必要がある．**5 章**の評価では y=0.5 とした．事前検証で 0.5 前後までは一度の更新で New X が求まるケースが多かったためである．

目標 IO busy 率と比例限界 busy 率の差が大きくなりすぎた場合の説明を行う．可能性があるのは，1）観測された IO busy 率が目標 IO busy 率を大きく下回った場合，2）現在の「1iops 当たりの busy 率」が直前の「1iops 当たりの busy 率」との比較で大きく変化した場合，の 2 ケースである．この情報より，比例限界 busy 率が変化した可能性があるは分かる．しかし，目標 IO busy 率との差が広がったのかどうかまでは判断できない．ここでは初期値に戻し，前節の New X を求める方法で目標 IO busy 率を求め直す制御とした．

今回の評価では IO busy 率が目標 IO busy 率を大

☆1 **図 8** の負荷 thread のこと
☆2 20 秒間隔で計測するので最短 60 秒で検出出来ることになる
☆3 **5 章**では，B 側の帯域を絞ることで IO busy 率が 100%から下がった後のデータを利用
☆4 目標 IO busy 率=比例限界 busy 率となる値

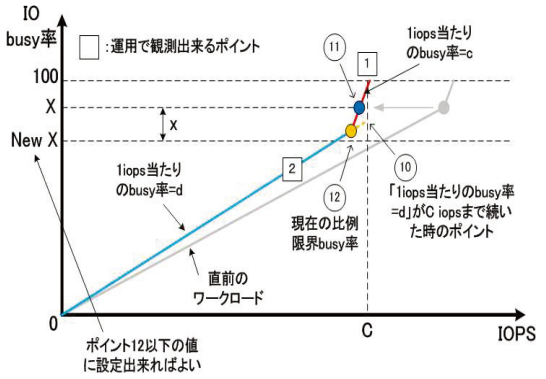

図 7　比例限界 busy 率の更新方法
Fig. 7　Method of updating proportion limit busy rate

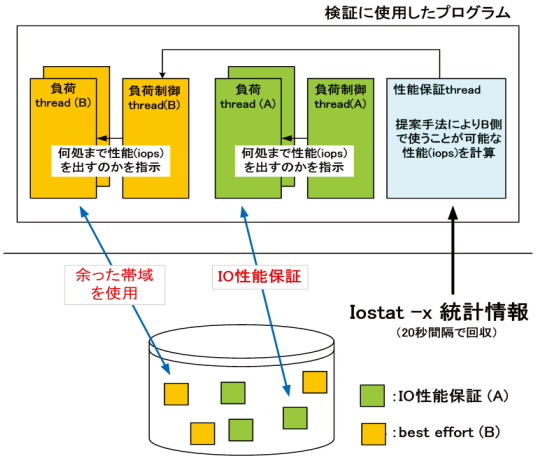

図 8　検証システムの概要
Fig. 8　Outline of verification system

きく下回った場合のみの実装とし，20%を下回る場合に初期値に戻すことにした．

提案方式での IO 性能保証レベルに関して説明する．**3 章**で示したように，比例限界 busy 率の範囲であれば，性能（iops）と IO busy 率は線形増加する．負荷が増減しても比例限界 busy 率の範囲に留まっていれば，「1iops 当たりの IO busy 率」はほぼ一定であるので指定した iops 値への誘導が可能である．負荷の増減に伴い 1IO の平均実行時間が変動する場合は，その変動率がそのまま誤差として表れる．**図 3** の svctmA+B を例に説明する．iops=80 のときの svctm=5.68ms である．iops=160 は svctm=5.72 である．この時の svctm の変動率は 0.78%ととなる．この値は iops が 80 から 160 まで増加したときの提案方式で IO 性能保証する場合の誤差と一致する．提案方式では「1iops 当たりの IO busy 率」を常時モニタしその都度誘導値の見直しを行うので，誘導値近辺で「1iops 当たりの IO busy 率」が一定であれば指定した iops への誘導が可能である．

5. 提案方法の評価

提案方法の評価を行うプラットフォームとしては，**3 章**でも使用した表 1 の環境を使用した．評価に使用したベンチマークプログラムは，**図 8** の構成のプログラムを専用に作成した．このプログラムは全てユーザレベルで動作するものであり，以下の特徴を備える．

- A 側：IO 性能保証を行う thread 群
  - 負荷 thread：
    10thread で並行して IO を実行．負荷するワークロードは**表 3** に従う．後半の 500GB 領域に IO 命令を送出 (**図 9** 参照)
  - 負荷制御 thread：
    表 4，**表 5**，又は**表 6** の値に従って上限性能を負荷 thread に通知
- B 側：IO 性能保証を行わない thread 群
  - 負荷 thread：
    10thread で並行して IO を実行．負荷するワークロードは表 3 に従う．前半の 500GB 領域に IO 命令を送出 (**図 9** 参照)
  - 負荷制御 thread：
    表 4，表 5，又は表 6 の値と性能保証制御 thread の値を比較し，より小さい値を上限性能として負荷 thread に通知
- 性能保証制御 thread
  - 提案手法のロジックが動作し，B 側に使うことが可能な性能値を通知☆
  - 目標 IO busy 率の初期値=95%

A 側，B 側に与えるワークロードは，ファイル共有とバックアップ相当のワークロードを組み合わせて与えた（表 3 参照）．両者とも **3 章**の実験で用いたものを使用した．

評価 1，評価 2，評価 3 で与える負荷は其々表 4，表 5，表 6 とした．

表 3　thread A,B に与えるワークロード
Table 3　Workload given to thread A and B

	A 側	B 側
評価 1	ファイル共有	バックアップ
評価 2	ファイル共有	ファイル共有
評価 3	バックアップ	バックアップ

☆ 20 秒間隔で出力される iostat -x コマンドログを用いて提案手法の計算を行う

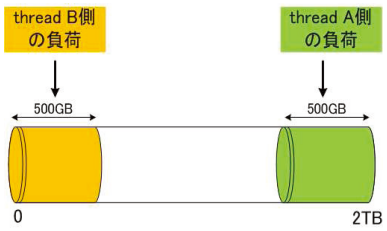

図 9　負荷の与え方
Fig. 9　How to give load

まず評価 1 に関して議論する．図 10 は表 4 の負荷を与えた結果である．図 11 はこの時の%util=100 となった累積回数である．A 側の目標値を 50iops に引き上げた直後は挙動が不安定になることが分かる．A 側と B 側の iops が交互に振動する理由は，**図 7** のポイント 2 相当のデータを獲得するために%util=100 に到達した時の B 側の iops を提案手法による計算値よりさらに削減する仕組みが働くためである．図 11 より%util=100 のポイントが 3 回観測された 500 秒後に目標 IO busy 率☆が提案手法で 63%に引き下げられることが確認できる．しかし挙動は安定せず，さらに%util=100 のポイントが 3 回観測された 700 秒後の引き下げで thread A の性能が 50iops で安定することが分かる．評価 1 では新しい目標 IO busy 率を求めるのに 2 回遷移が必要だったことが分かる．

960 秒後に A 側の目標値が 5iops に減少した後，B 側の負荷が最初の状態に戻らないことも確認できる．

表 4　評価 1 の目標 iops
Table 4　Target iops of evaluation 1

A の目標 iops	B の iops	継続時間 (分)
5	50	5
50	50	10
5	50	10

表 5　評価 2 の目標 iops
Table 5　Target iops of evaluation 2

A の目標 iops	B の iops	継続時間 (分)
10	200	5
100	200	10
50	200	10

表 6　評価 3 の目標 iops
Table 6　Target iops of evaluation 3

A の目標 iops	B の iops	継続時間 (分)
5	30	5
30	30	10
5	30	10

☆ **4 章**の X 相当

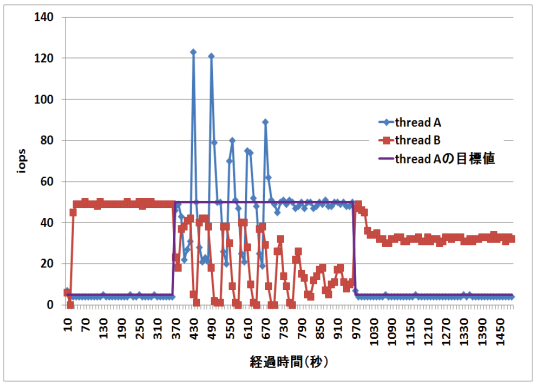

図 10　評価 1 の実験結果
Fig. 10　Outcome of an experiment of evaluation 1

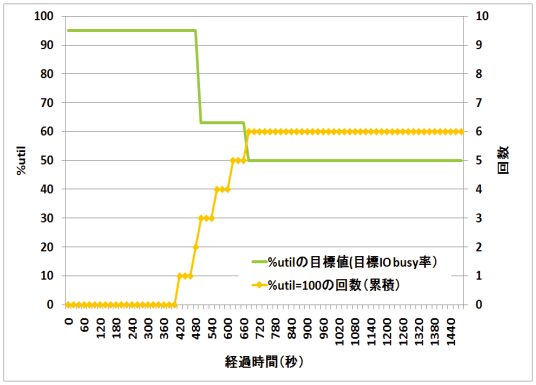

図 11　評価 1 の実験結果（%util=100 が観測された回数）
Fig. 11　Outcome of an experiment of evaluation 1
(frequency to which %util=100 is observed)

A 側の負荷が下がった後も B 側の負荷上昇で%util が 50%前後に達してしまい，目標 IO busy 率が初期値に戻らないためである．一方，%util/iops は 5iops に減少する前後で 0.8 から 1.3 に変化していた．この情報を使用することで目標 IO busy 率を元に戻すことが出来る．今回実装しなかった機能を実装することで解決出来ることが確認できた．

次に評価 2 に関して議論する．図 12 は表 5 の負荷を与えた結果である．図 13 はこの時の%util=100 となった累積回数である．A 側の目標値を 100iops に引き上げた直後は A 側の iops がすぐに目標値に到達しない．これは B 側の負荷が十分に下がりきれず性能干渉が起きたためである．その後目標 IO busy 率が 87%に引き下げられると，A 側の性能が 100iops で安定することが分かる．1 回で新しい目標 IO busy 率を求めることが出来た．

次に評価 3 に関して議論する．図 14 は表 6 の負荷を与えた結果である．図 15 はこの時の%util=100 と

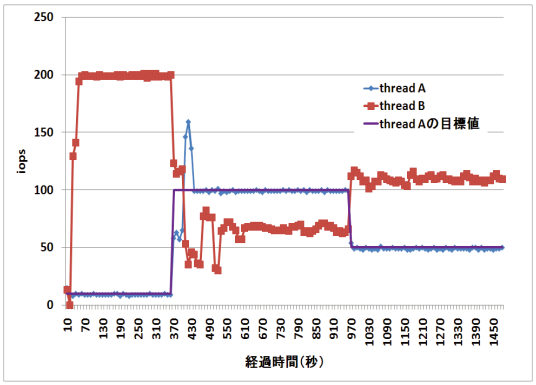

図 12　評価 2 の実験結果
Fig. 12　Outcome of an experiment of evaluation 2

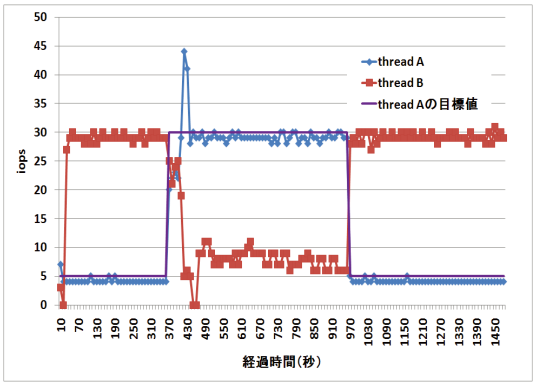

図 14　評価 3 の実験結果
Fig. 14　Outcome of an experiment of evaluation 3

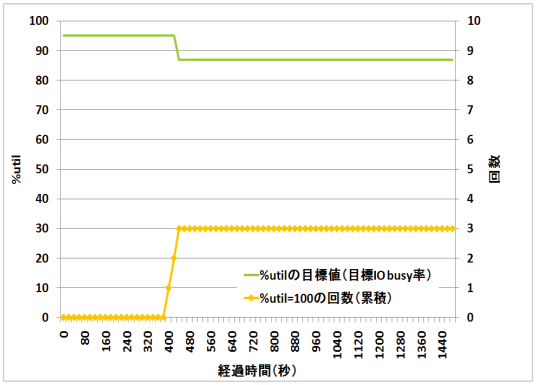

図 13　評価 2 の実験結果（%util=100 が観測された回数）
Fig. 13　Outcome of an experiment of evaluation 2
(frequency to which %util=100 is observed)

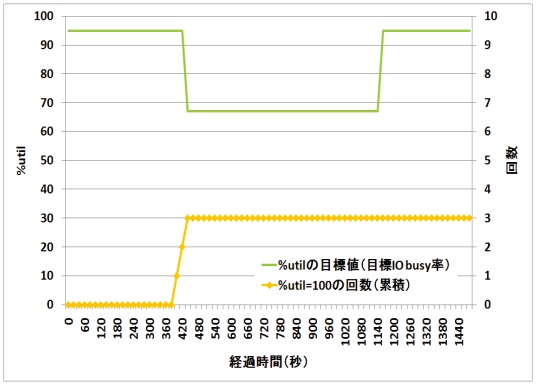

図 15　評価 3 の実験結果（%util=100 が観測された回数）
Fig. 15　Outcome of an experiment of evaluation 3
(frequency to which %util=100 is observed)

なった累積回数である．A 側の目標値を 100iops に引き上げた時の挙動は評価 2 と同じ経緯をたどることが分かる．評価 3 も 1 回で新しい目標 IO busy 率を求めることが出来た．

最後に評価のまとめを行う．評価 1，評価 2，評価 3 共に提案方式で A 側 thread の IO 性能保証が実現できることを示した．ファイル共有とバックアップを競合させた評価 1 に関しては，今回の目標 IO busy 率の計算方法では 2 回の計算が必要であった．また，目標 IO busy 率を初期値に戻す条件として，IO busy 率が目標 IO busy 率を大きく下回ったとき，だけでは不十分であることが分かった．初期値に戻せなかったケースでは，1iops 当たりの IO busy 率が大きく変化していたことも分かっている．今回実装を行わなかった 1iops 当たりの IO busy 率の変化量を条件に加えると評価 1 でも初期値に戻すことが出来ると判断している．

6. ま と め

ストレージワークロードの IO busy 率と iops の調査を行ったところ，ワークロードごとに比例限界 busy 率が存在し，比例限界 busy 率の範囲内では IO busy 率と iops は線形増加関係となることが分かった．そこで，ワークロード全体の IO busy 率が目標 IO busy 率と一致する様に制御し，「1iops あたりの IO busy 率」を用いた比例計算で性能保証する方法の提案を行った．目標 IO busy 率が比例限界 busy 率を下回るように制御することでワークロードの変化に対応する．

評価はアプリケーションとして実装した専用プログラムで行い，ファイル共有ワークロードとバックアップワークロードを組み合わせた 3 パターンのいずれでも IO 性能保証出来ることを示した．

ファイル共有ワークロードとバックアップワークロードが競合するパターンでは，提案した目標 IO busy 率の変更方法では 2 回の変更が必要である．あらゆる

ワークロードに関して，1 回の計算で正しい目標 IO busy 率を求める方法の確立が今後の課題である．

今回は，決められた性能保証値を一時的に守れなくなることを容認した上でストレージ装置の性能を使いきることを目指した提案である．今回提案を行わなかった決められた性能保証値を確実に守る方式検討も今後の課題である．

7. 謝　　辞

本稿執筆にあたり，適切な助言を頂いた（株）富士通研究所 IT システム研究所　小沢年弘主席研究員に感謝いたします．

参 考 文 献

1) Christopher R.Lumb, Arif Merchant, and Guillermo A.Alvarez. Facade: virtual storage devices with performance guarantees. Proceedings of FAST'03, March 2003.
2) Ajay Gulati, Irfan Ahmad, and Carl A. Waldspurger. PARDA: Proportional Allocation of Resources for Distributed Storage Access. Proceedings of FAST'09, February 2009.
3) http://sourceforge.jp/projects/sfnet_ioband/
4) 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，HOKKE-2009
5) 山田　浩史，河野　健二: ユーザレベルでのディスク帯域制御機構，第 8 回　プログラミングおよび応用のシステムに関するワークショップ（SPA 2005）
6) Kyung D.Ryu, Jeffery K. Hollingsworth, and Peter J. Keleher. Efficient Network and I/O Throttling for Fine-Grain Cycle Steamling. In Proceedings of the ACM/IEEE SC2001 Conference (SC '01), November 2001
7) 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.
8) SGI IRIX Admin: Disks and Filesystems(日本語版), 007-2825-009JP