Vol.2014-OS-128 No.7
2014/3/7

マルチテナント環境におけるI/O性能保証方式の提案

北原 友恵　福本 佳史　小田 哲　小西 隆介　湯口 徹

概要：多くの仮想化環境において，一つの筐体のストレージは複数の仮想マシン（VM）に共有されている．そのような環境において，あるVMがストレージのI/Oをバースト的に発生させると，I/O競合が発生し，同じストレージを共有している他のVMの性能低下を引き起こす問題が指摘されている．VM毎にストレージリソースを予め割り当てることによってI/O競合を抑制することが可能であるが，アイドル状態のVMが存在するため，全体のストレージのI/Oリソースの使用率は低下する．本研究では，ストレージのI/Oリソース使用の効率化を行いながら，VMのI/O競合を抑制し，各VMが一定以上のI/O性能を確保できる制御アルゴリズムを提案する．アルゴリズムの有効性の検証を行うと共に，IOPS等の従来のストレージI/O性能保証指標について考察を行う．

キーワード：ストレージ，I/O，性能，仮想マシン

Algorithm for Guaranteeing I/O Performance in Multi Tenant Environment

Tomoe Kitahara　Yoshifumi Fukumoto　Satoshi Oda　Ryusuke Konishi　Toru Yuguchi

Abstract: In most virtualization environments, a chassis of storage is shared from dozens or hundreds of virtual machines. The I/O burst of a virtual machine causes a performance degradation of other virtual machines which are sharing the same storage. This performance degradation can be moderated by pre-allocating storage resource on a per virtual machine basis, however, the total usage of the I/O resource will be ceiled and will not be effectively utilized because resource of idle virtual machines is not reallocated. In this paper, we propose a dynamic control algorithm in which I/O resource competition among virtual machines is moderated and each virtual machine can keep a guaranteed I/O performance adaptively utilizing room of total I/O resources. We verify the validity of the algorithm, and think of indices for such I/O performance guarantee.

Keywords: Storage, I/O, Performance, Virtual Machine

1. はじめに

IaaSをはじめとするクラウドコンピューティングは，新規システム構築の迅速化や拡張容易性，運用管理の簡素化，ITコストの削減が行えることから利用が進んでいる．クラウド環境の多くは，物理リソース（CPU，メモリ，ストレージ，ネットワーク）を一つの基盤としてまとめ，物理リソースを複数の仮想マシン（VM）が共有して利用するマルチテナント環境を採用している．マルチテナント型のクラウドサービスは，CPUやメモリのスペックが選択可能であるのに対し，VMのストレージI/O性能に関して選択できるものが少なく，I/O性能はベストエフォートとなっているものが多い．ベストエフォート型は，同一基盤上のVMは時間帯によりVMの稼働のピークにばらつきがあることや，アイドル状態のものが存在することにより，固定的なI/Oリソースの割り当てではなく，各VMからのI/Oリクエスト量を無条件に受け入れることで，その時々にお

日本電信電話株式会社 ソフトウェアイノベーションセンタ
NTT, Musashino-shi, Tokyo 180-8585, Japan
{kitahara.tomoe, fukumoto.yoshifumi, oda.satoshi, konishi.ryusuke, yuguchi.toru}@lab.ntt.co.jp

ける最大の性能を VM に提供することが可能である．そのため，事業者側では，ストレージ I/O リソースの最大限の活用と効率化，利用者側では，I/O リソースに余裕があれば，VM がより高い I/O 性能を得られることがメリットとなっている．

しかし，このようなベストエフォート型サービスのストレージ I/O 性能では，リソース競合が問題となっている [1][2]．ストレージは，CPU やメモリと比較し，VM が同一の筐体を利用しているため，VM の挙動により，共有する他の VM の性能に影響を与える可能性が高い．特に，リソースの競合では，ある VM が大量の I/O を発生させる（バーストする）ことで，引き起こる VM の性能低下が問題とされている．今後，様々なアプリケーションで広く使われるマルチテナント型クラウドサービスでは，リソースを柔軟に拡充，縮小できることが利点であり，ストレージにおいてもディスク容量だけでなく，I/O 性能も利用者が選択できる環境になることが望ましい．そのため，この I/O 性能低下の問題を解決し，各 VM に安定した I/O 性能を提供することが必要不可欠となる．

性能低下を引き起こさないための最もシンプルな手法としては，ストレージが提供できる性能の範囲内で，固定的に各 VM の I/O リソース使用量の上限値を設定する方法がある．この設定を行うことで，他の VM に影響を及ぼさない範囲でのバーストのみを許可するため，ベストエフォート型で生じる性能低下の影響を抑制することができる．しかし，この手法では，各 VM の I/O を発生させるピーク時間帯のばらつきやアイドル状態の VM の存在が想定されるため，I/O リソースの余剰が発生し，ベストエフォート型サービスで得られていたリソース使用の効率化が得られない．

本研究では，マルチテナント環境における VM の I/O 性能において，「ある VM のバーストによる I/O 性能低下の回避」，「I/O リソースの利用率の最大化」の 2 つを実現するためのアルゴリズムを提案する．まず，性能低下を引き起こす状態を，擬似的に再現する予備実験を行った．その結果，VM 内において，I/O を発生させるプロセス数を増加させることにより，最も他の VM の性能低下を引き起こすことが明らかになった．

VM の I/O はゲスト OS の I/O キュー，ホストのキューを経由してストレージのキューに送られ順次処理されるが，プロセス数の多い VM のエンキュー量が多いため，最終的にストレージのキュー内の占有率が高まり，他の VM の I/O 処理の遅延につながって性能が低下している．そのため，エンキュー量を抑えるために，一般的な VMM（Virtual Machine Monitor）や Linux に標準的に実装されている帯域と IOPS の上限値を動的に設定することにより，エンキュー量を抑え，ほかの VM の性能低下を回避する．本論文で提案する方式は，各 VM の帯域・IOPS を定期的に監視しながら，リソースを占有する VM を検知し，帯域および IOPS において一時的な上限値を設定する．この方式の有効性を検証するために，ベンチマークツール fio を用いて競合状態を再現し，アルゴリズムの適用前後で各 VM の性能がどの程度保てるのかを実験を行った．アルゴリズムの適用前では，高負荷な VM 以外の VM の性能低下がある一方で，アルゴリズム適用後は，各 VM のスループット・IOPS の低下を抑制し，応答時間に関しても，改善することができた．

2. 既存技術

従来，ストレージの I/O を制御する技術として使用されている方式が，優先制御と I/O 帯域制御である．本章ではこの二つの制御方法について述べる．

2.1 優先制御

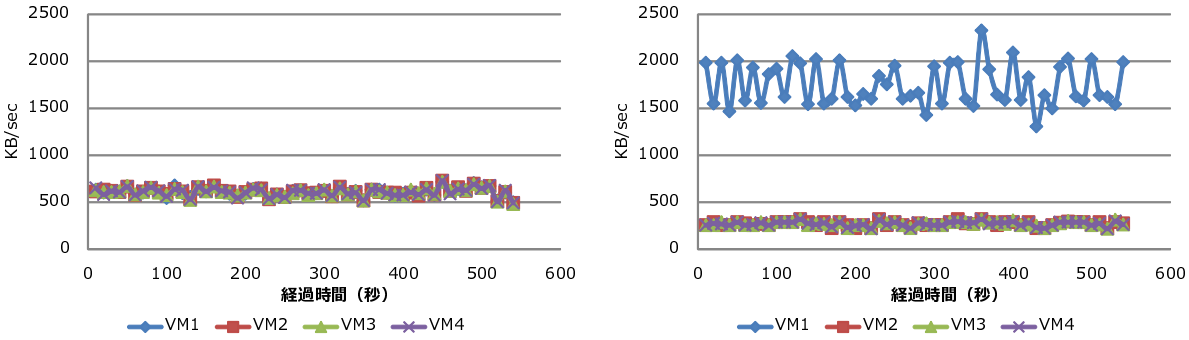
優先制御は，VM からの I/O 毎や VM 毎に優先度付を行い，その優先度に応じて I/O の処理順を変更する方法である．この処理順を決定する I/O スケジューラに関する研究が広く行われている．

Wang らは，VMM 内で各 I/O に優先度を設け，制御を行う Nested QoS[3] を提案した．Nested QoS では，保証したいレイテンシのレベルを定義づけるクラスを用意する．各 I/O は決められたレイテンシ以内に処理が行われるようにクラスに分類され，処理順が決定する．また，Ajay らの提案した PARDA[6] は，実際に VMware で，Storage I/O Control の I/O スケジューリングとして実装されている．PARDA では，予め決められた優先度に基づいてストレージ I/O リソース利用のシェア率を決定し，それに応じた I/O スケジューリングを行っている．

これら，I/O スケジューリングを用いた優先制御では，優先度の高い I/O に関しては，設定された性能値を維持させようとする．しかし，優先制御では，優先度の高い VM の I/O 量に応じて優先度の低い VM の性能が低くなる場合がある．優先制御では，決められた割合に基づいて I/O の処理が行われるため，優先度の低い VM の I/O 性能は，優先度の高い VM の I/O リクエスト量に依存するため，優先度の高い VM の I/O リクエスト量が少ない場合には，ほかの VM が I/O リクエストを出しても処理待ち状態になり，性能が低くなることがある．また，優先度やリソースシェア率に応じて利用できる割合が決定するため，ストレージ I/O リソースに余剰が発生する．そのため，I/O リソースの有効活用を達成できていない．

2.2 I/O 性能の制御

Kerlson らは，スループットとレイテンシの情報をもとに，保証値に達していない VM を検知し，周りの VM のスループットの制御を行う Triage を提案している [7]．ま



(a) 全 VM：同一並列数（1 プロセス）　(b) VM1：10 プロセス，VM2〜VM3：1 プロセス

図 1　I/O 並列数による性能の変化

た，多くのストレージサーバや VMM の機能として IOPS やスループットの制御機能が備わっている．I/O 性能の制御には IOPS とスループットの性能指標が多く用いられている．この制御では，IOPS とスループットに関して，上限値と下限値の 2 種類の設定方法があり，各 VM の利用可能な I/O リソース量（IOPS，帯域といった性能値）を設定する．ストレージの最大性能の範囲で IOPS（または帯域）の上限を設定した場合，各 VM で利用できるストレージの I/O リソースの上限が設定されるため，バーストが発生してもほかの VM の I/O 性能を低下させずに，継続的な利用が可能である．しかし，IaaS などのパブリッククラウドにおいて様々なアプリケーションが実行される環境で，稼働している VM 内でのアプリケーションによって夜間や日中などのピーク時間の異なりやアイドル状態の VM の存在が想定され，固定的に上限値を割り当てた際に，I/O リソースの余剰が発生することがある．そのため，I/O リソースは最大限に活用されない．また，一般的な下限値の設定は，目標値となっている場合が多く，周りの VM の負荷状況に影響を受けるため，各 VM の最低性能を厳密に保証することはできていない．

本論文では，各 VM に最低ラインの性能（帯域や IOPS）を提供しつつ，ストレージリソースの最大限に活用するための最低性能保証型ベストエフォート（最低保証をしながら，空きがあれば，割り当て以上の性能を出せる）を実現するためのアルゴリズムを提案する．

3. 提案するアルゴリズム

提案するアルゴリズムは，同一ストレージを共有する VM のバーストによる性能低下を抑制し，かつ I/O リソースを最大限利用するための手法である．性能低下の抑制と I/O リソースの利用最大化について述べ，アルゴリズムの詳細を示す．

3.1 性能低下の抑制

最低性能を保証しつつ，ストレージの I/O リソースを有効活用するためには，ある VM のバーストによる同一のリソースを共有する VM の性能低下を抑制しなければならない．ほかの VM の性能低下を引き起こす VM の挙動はどのようなものかを確かめるための，予備実験を行い，I/O の並列度の増加によって，他の VM の性能低下を引き起こすことが明らかになった．その結果を図 1 に示す．図 1(a) では，プロセス数 (I/O の並列度) がすべての VM で同じ条件となっており，各 VM が約 600KB/s のスループットが得られている．図 1(b) では，1 台の VM のみ I/O の並列度を 10 倍に増やしたものである．バーストしている VM は，平均で約 1,750KB/s であるのに対し，そのほかの VM が 270KB/s とすべての VM が同じとき（図 1(a)）の状態と比較し約 50%程性能が低下している．

このように，ある VM における I/O の並列数が高くなると，ほかの VM の性能低下を引き起こす．ある VM 内において，I/O の並列度が高くなり，同じストレージを共有する VM に性能低下を引き起こしているとき，I/O の並列度の高い VM はストレージにおけるキューにおいてもその VM の I/O の割合が多くなっている．ストレージへの特定の VM からのエンキュー量が多くなるため，ほかの VM の I/O 処理が遅くなり，結果として性能低下につながる．したがって，高い I/O 並列度の VM が出現した時には，その VM のストレージにおけるキュー割合を下げることで，性能低下の影響を回避できると考えられる．ストレージへのエンキュー量を減らすには，スループットや IOPS を用いて制御を実行することで，VM からストレージへのリクエスト量を減らすことができる．ただし，VM の稼働するホストサーバ内または VMM 内で制御する必要がある．また，ストレージの I/O リソースの有効活用の観点から，VM に対して制御を実行する場合は，I/O リソースに余剰がなく，最低性能値をすべての VM に対し保証できない場合に実行しなければならない．

3.2 I/O リソースの利用最大化

I/O リソースを何も制御しない環境では，各 VM の利用

要求に応じて，最大限の性能を提供することである．しかし，最低性能値を保証するためには，最大限の I/O リソースの提供する上で，最低性能に到達していない VM の性能値に配慮する必要がある．商用サービスとして提供されている IaaS では，インフラを利用者に提供するサービスであるため，基本的に利用者がどのように VM を使用するのかを把握しない．I/O 性能に関しても，すべての VM に対して，要求する I/O 量を管理側で把握するのは困難である．そのため，最低性能値に到達していない VM に対して，要求する I/O があるにもかかわらず処理が実行できないのか（性能低下の状態），要求するリクエストの量が保証値よりも下回っているため，低い性能であるのかの判断が難しい．特に，VM 側で一つの I/O が終わった後に次の I/O を発生させる（同期 I/O）ときには，本来 VM の要求している I/O 量の把握は極めて困難となる．本手法では，VM のリソースの余剰状態を把握するために，ストレージの I/O リソースの利用状態が最大値に近づいてきたときに，もっとも I/O リソースを利用している VM に対し制御を行い，性能を向上するための I/O リソースを利用できる環境を用意することで，性能低下の影響を排除する．

3.3 アルゴリズムを適用するための構成

提案するアルゴリズムは，リソースを占有する VM を検知して，その VM に対して制御を行う．制御方法は，ストレージのキューにリクエストが格納される前のホストサーバや VMM 上であればよい．KVM には，IOPS，スループットの制御機能が実装されており，今回の実装では，この機能を利用した．本実験では KVM を利用しているが，各 VM に対する I/O 帯域制御は Xen や Hyper-V などの他の VMM においても実装されている機能であり，VMM に限定はされない．また，VMM の制御機能を利用しなくても Linux の cgroups などの利用も可能である．図 2 に本アルゴリズムを適用するための構成を示す．VM を稼働させるホストサーバ・制御実行判断を行う管理サーバ・ストレージの 3 つから構成される．本アルゴリズムは，大きく 3 つの処理フェーズに分かれている．まず，管理サーバにおいて，I/O 性能の制御の有無・対象や制御値を決定するための情報を収集する．情報収集には，iostat，qemu-monitor の blockstats を使用し，各 VM の使用する LUN（Logical Unit Number）に対する IOPS，スループット，の情報を取得する．次に，受信した性能情報を基に，管理サーバは制御の実行判断，制御対象・制御値の決定を行う．最後に，管理サーバは制御対象となる VM が決定した後，VMM に対して VM の制御の実行命令を送り，VMM は制御と緩和を管理サーバからの情報をもとに制御を実行する．次節で，図 2 における制御情報の決定と制御の実行方法の詳細を述べる．

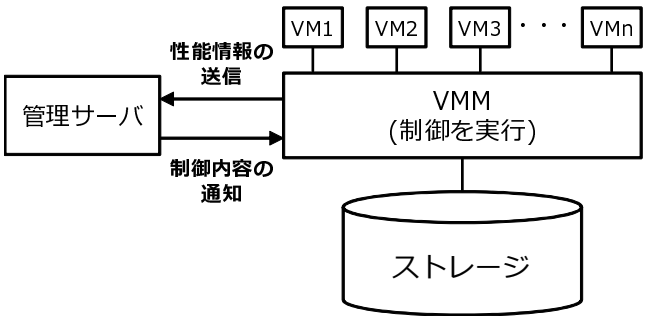


図 2 アルゴリズムを適用する構成

3.4 実行判断・制御対象の決定

アルゴリズムに使用するパラメータを表 1，管理サーバにおける処理の流れをアルゴリズム 1 に示す．本アルゴリズムでは，スループットと，IOPS を制御の対象とする．I/O 性能については，I/O サイズが小さい程 IOPS が高く，大きいほどスループットが大きくなる．IOPS とスループットの両方を制御の対象とすることで，I/O サイズがどの様な状態であっても適切な制御が可能である．IOPS とスループットの制御のアルゴリズムは同様のフローで行う．以下では，スループットの制御アルゴリズムを説明する．

ひとつのストレージを共有する m 台の VM が存在するマルチテナント環境において，まず，各 VM のスループットのデータを取得する (3 行目)．取得するデータは，現在時刻 t の直前の $t-1$ 時点のデータを用いる．次に，ストレージにおける総処理量を求めるために，各 VM のスループットの総和 b_{sum} を計算する (5 行目)．スループットの上限設定は，判定基準値である b_{border} よりも b_{sum} の値が大きい場合に制御を実行する (7 行目)．b_{border} の方が大きい場合には，リソースに余裕がある状態と判断し，制御を行わない．実行判断基準を少なく見積もると本来制限をかけなくても良い VM に対しても制限がかかってしまう．そのため，ストレージの最大性能値に実行性能値が限りなく近づいた時に制御が実行されることが望ましく，b_{border} はストレージの最大性能に近い値を設定する．次に，実際に制御を実行するために，制御対象の VM と，制御値を決定する．制御の対象となる VM は，取得した情報の中で，最も高い性能を出していた 1 台の VM である．8 行目において，スループットが最も高い VM とその VM が出したスループット値を取得する．複数台の高負荷な VM がいる環境では，次の判定で他の高負荷 VM を制御する．

制御をかける値は，制御の対象となる VM のスループット値に対して $rate$ の値をかけたものである (9 行目)．$rate$ は，対象となる VM に対し，性能を減らす割合であり，$0 < rate < 1$ の範囲で決定する．最後に，VM に対し，制御を実行する (10 行目)．ここでのパラメータは対象となる VM(n_{target})，制御値 (b_{cap_val})，制御をかけ続ける時間 (cap_time) の 3 つをパラメータとして実行する．もし，次の判断時にも，制御実行中の VM が制御の対象となった時

は，これまで実行していた制御プロセスを終了させ，新たな制御値で最初から制御をやり直す．少数の VM がリソースの大部分を占めている状況では，平均値を基にした 1 回の制御では，他の VM との性能差が完全に埋まりきらない状況が起こる．そのため，短い間隔で制御判断を行い，制御をかける値を常に更新する必要がある．また，制御を実行する時間は，複数台の高負荷 VM が存在する状況を考慮したときに，判断間隔より，長い時間制御を実行しなければ，高負荷 VM が代わる代わる出現し，各 VM の公平にならない状況が続く事が推測される．そのため，制御の実行時間は判断間隔よりも長く設定する必要がある．すなわち，$cap\_time > cap\_interval$ でなければならない．

表 1 主な記号の定義

記号	定義
N	入力データとなるノードリスト
I	入力データとなる IOPS のリスト
B	入力データとなるスループットのリスト
m	VM 数
i_m	VM n_m の IOPS の値
b_m	VM n_m のスループットの値
b_{border}	スループットにおける制御実行基準
n_{target}	制御の対象 VM
b_{target}	制御の対象 VM のスループット値
b_{cap_val}	スループットにおけるリソース利用の上限値
$rate$	制御値を減少させる割合
cap_time	制御継続する時間
$cap\_interval$	制御の実行間隔

Algorithm 1 制御対象・制御値の決定方法

Input: $N = \{n_1, n_2, ..., n_m\}, b_{border}, i_{border}$, $rate, cap\_time, cap\_interval$

1: **while** 1 **do**
2: t = current_time()
3: B = collect_bandwidth_data(N, $t-1$)
4: I = collect_iops_data(N, $t-1$)
5: $b_{sum} = \sum_{k=1}^{m} b_k (b_k \in B)$
6: $i_{sum} = \sum_{k=1}^{m} i_k (i_k \in I)$
7: **if** $b_{sum} > b_{border}$ **then**
8: $\{n_{target}, b_{target}\}$ = get_top_node(N, B)
9: $b_{cap\_val} = rate * b_{target}$
10: do_throttle(n_{target}, b_{cap_val}, cap_time)
11: **end if**
12: **if** $i_{sum} > i_{border}$ **then**
13: $\{n_{target}, i_{target}\}$ = get_top_node(N, I)
14: $i_{cap\_val} = rate * i_{target}$
15: do_throttle(n_{target}, i_{cap_val}, cap_time)
16: **end if**
17: sleep $cap\_interval$
18: **end while**

4. 実験

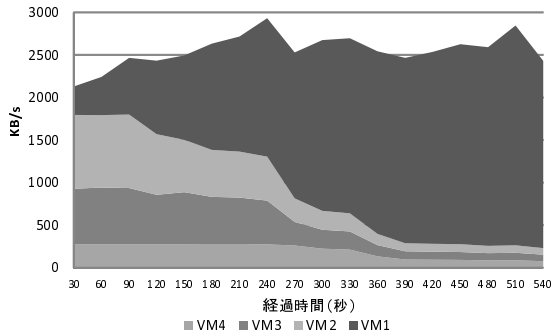
4.1 実験環境

本検証には，CPU が Intel Xeon 1.80GHz，メモリが 32GB の RedHat サーバ 6 台と，コントローラが 2 つ，メモリが 32GB，2TB の SAS ディスク 12 本の RAID6 構成，インターフェースが iSCSI である共有ストレージを利用した．VM は VMM に Qemu-KVM を使用し，vCPU が 1 つ，メモリ 1GB，OS が CentOS6.4 となっている．また，今回の検証では，アルゴリズムの有効性を確認するため，VM においてホストマシンのページキャッシュ，ディスクキャッシュは利用しないものとする．VM の利用するディスクは，ローデバイスマッピングを用いて共有ストレージから切り出した LUN を直接利用する．

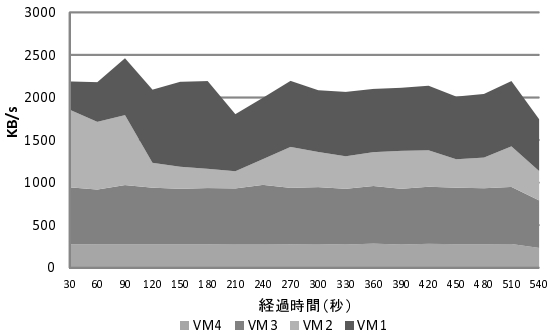
一般的にディスクへの I/O 処理では，VM に割り当てられた仮想ディスク上に作成されたファイルシステムを経由して I/O 処理が行われる．I/O は一度バッファに蓄積され，I/O が一定量たまった後，ファイルシステムが I/O をまとめてストレージへリクエストを渡す．そのため，ストレージ内で処理される I/O はほぼ同サイズものであると想定される．本実験においては，ベンチマークツール fio を用いて I/O 負荷をかけた．負荷の条件については，全ての VM の I/O サイズおよびワークロードが同一となるように設定した．実施したワークロードは各 VM で同じものを使用する．使用したワークロードは，I/O サイズが 64KB，100%ランダムライトである．実験で使用する VM は，VM の性能低下を引き起こす原因となる高負荷 VM と，高負荷 VM の影響を受けるその他 VM の 2 種類に分けられる．高負荷 VM では，プロセス数を最大 10 プロセスとしてベンチマークを行い，ほかの VM の性能低下を引き起こす状態を再現した．高負荷 VM の負荷量は，1 分毎にプロセス数を 1 ずつ追加し，段階的に負荷を増加させた．その他 VM については，常に 1 プロセスのみを実行する．VM 数については，4VM と 20VM の 2 パターンを行った．4VM のパターンでは高負荷 VM を 1 台配置する．20VM では，高負荷 VM を 5 台配置し，複数台の高負荷な VM が存在するときの有効性を検証する．

4.2 パラメータ設定

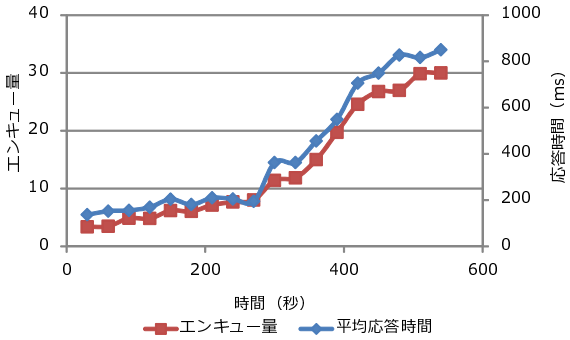
制御実行基準値，判定間隔，制御時間の設定について述べる．今回，予めベンチマークを行い取得したワークロードに対して性能の最大値を取得し，制御実行基準値 b_{border} を 90%，継続時間 (cap_time) を 60 秒，制御の実行判断する間隔 ($cap\_interval$) は 5 秒に設定した．実行判断の間隔は，今回作成したプログラムにおいて計算から実行命令を行うまでに 20 台の仮想マシン環境で約 2 秒を必要とした．そのため，判断間隔は 2 秒以上とる必要があり，重複して判断の計算を行わないように，本実験では 5 秒と設定し



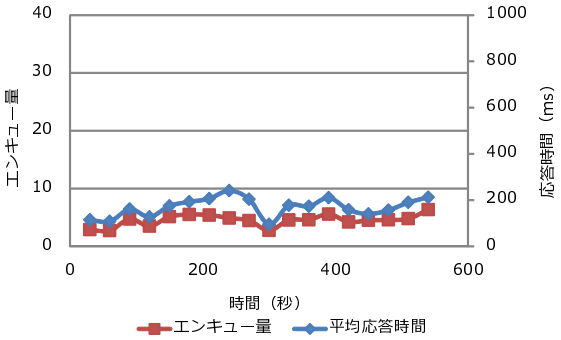
(a) 実行前スループットの推移



(b) 実行時スループットの推移



(c) 適用前：エンキュー量と応答時間の推移



(d) 適用後：エンキュー量と応答時間の推移

図 3 4VM ベンチマーク

た．そして，多くの仮想マシンが稼働する環境で一台の仮想マシンの制御を実行した際に，各仮想マシンに余剰分のリソースが利用できるまでに，約 60 秒を必要とすることが，予備実験の中で判明したため，今回の実験においては制御時間を 60 秒とした．

4.3 実験結果

4.3.1 VM 数：4 台

スループットの推移の積み上げグラフ，キューサイズと平均応答時間の推移を図 3 に示す．また，4 台の VM のスループットの最大値，最小値，平均値を表 2 に示す．図 3(a) において，VM1 のプロセス数の増加に応じて，ほかの VM のスループットが減少している．各 VM の要求スループットは，VM2 が 800KB/s，VM3 が 600KB/s，VM4 が 300KB/s と設定されている．その要求量が多い VM から性能低下の影響を受け，390 秒過ぎからは VM2〜4 がそれぞれ 85KB/s まで落ち込んでいる．アルゴリズム適用後の結果である図 3(b) では，アルゴリズムの適用によって，図 3(a) において性能低下を起こしていた VM の性能の低下幅が減少している．特に VM3，VM4 においては性能低下をほとんど起こしていないことがわかる．しかし，VM2 のように性能低下を起こしている VM の存在も確認した．VM2 は常に最大 800KB/s の I/O スループットを要求している．しかし，ベンチマーク中の平均スループットは，440KB/s であり，VM の要求しているスループットに到達していない．

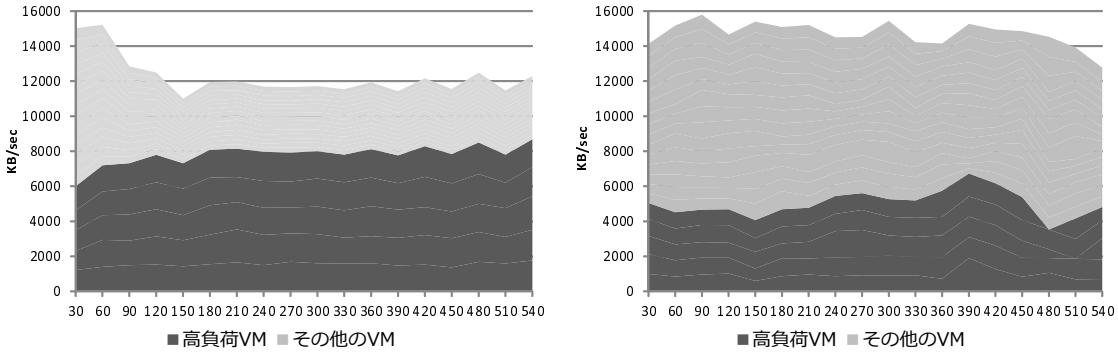
図 3(c)，3(d) では，それぞれアルゴリズム適用前と適用後の各 VM のディスクキューのサイズの総和と応答時間の推移を示している．図 3(c) では，VM1 の I/O の並列度に応じて，キューサイズが増加している．一方でアルゴリズム適用後では，帯域と IOPS 制御が行われたことにより，キューサイズが増加せずに，キューサイズが 5 程度で推移している．また，平均応答時間に関して，アルゴリズムの適用前は，キューサイズの増加に比例して各 VM で増加しているが（最大 900ms），アルゴリズム適用後は，増減が少なく推移している（最大 400ms）．アルゴリズムを適用しキューサイズを減少させることで，性能低下の影響を抑制するとともに各 VM に対しての I/O の平均応答時間の増加の抑制にも寄与していることが分かった．

	最大		最小		平均	
制御の有無	無	有	無	有	無	有
VM1	2580	1030	330	330	1600	735
VM2	870	910	85	200	400	420
VM3	665	690	85	640	360	665
VM4	280	280	85	270	205	275

表 2 図 3 のスループット結果 (単位：KB/s)

4.3.2 VM 数：20 台

図 4 は，高負荷 VM 5 台，その他 VM15 台のそれぞれの



(a) 実行前スループットの推移　(b) 実行時スループットの推移

図 4　20VM ベンチマーク

スループットの総和の積み上げグラフである．4VM での実験と同様に，高負荷 VM のプロセス数の増加とともにその他の VM のスループットが減少している（4(a) : 150 秒まで）．また負荷が高くなった状態（図 4(a) : 150 秒〜）では，ストレージにおけるスループットの内，約 7 割が高負荷 VM の I/O 処理となっている．アルゴリズムを適用することにより，段階的に高負荷な VM に対し制御が実行され，ストレージに対する高負荷 VM のスループットの割合は，3 割に低減している．それに伴い，その他の VM はアルゴリズム適用前に平均で 250KB/s のスループットに対し，適用後は，平均で 600KB/s と改善が見られた．

5. 考察

5.1 性能低下の抑制とリソースの有効活用

IOPS や帯域を指標として，最も高い性能を出している VM に対し，制御を行い，I/O リソースを共有する VM の性能低下の抑制が行えた．段階的に負荷を増加させている VM1 において，図 3(b) では 90 秒を境に，性能低下がみられている．一方で，アルゴリズム適用後の図 3(b) においては負荷が増加しているが，VM に与える影響を抑制できていることがわかる．しかし，要求するスループットが 800KB/s である VM2 のみ性能が低下している．これは，VM1 の負荷を増加する過程において，VM2 がすべての VM の中で最も性能を出しているポイントがあったため，制御が実行され，性能を落としていた．また VM2 は，図 3(b) の 120 秒から 240 秒の間のスループットの減少については，KVM の I/O Throttle では，設定値を超えた分を次の単位時間においてバランスを取ろうとするため，急激な回復は行われない．このように，今回実装したアルゴリズムでは，単純に性能の最も高い VM を対象として制御を実行するため，本来制御を必要としない VM に対しても制御が実行される可能性がある．そのため，周りの VM の性能値を考慮しながら，制御値を柔軟に変更しなければならないと考えられる．また，制御の解除に関しても，周りの VM の性能値を考慮しながら段階的に制御を緩めていくなどの方式を採用することで，バーストする VM の発生を抑制することができると考えられる．図 3(c)，3(d) では，キューサイズのアルゴリズムの適用により，ストレージに対するキューの深さはほぼ一定になっている．ストレージの I/O の応答時間は，キューの深さと密接に関係しており，キューの深さが一定以上になると急激に，応答時間が増加する．そのため，アルゴリズムを適用し，バーストする VM のエンキュー量を制御したため，ストレージに対するエンキュー量も相対的に減少し，平均応答時間の改善することができたと考えられる．

5.2 制御パラメータについて

本アルゴリズムでは，各 VM の IOPS とスループットを用いて I/O リソースを占有する VM の特定を行った．ファイルシステムは I/O を一定のサイズにまとめてストレージに対してリクエストを渡すが，Linux の ext4 では 512KB，Windows の NTFS では 256KB の I/O サイズにまとめられる場合が多く，すなわちファイルシステムによって I/O 毎の負荷が異なる．また，アプリケーションによって I/O サイズも異なると想定される．そのため，様々な OS で運用するクラウド環境においては，I/O サイズを意識した制御方法にしなければならない．

実際に，ストレージに対して，I/O サイズが異なるワークロードを用いて負荷を発生させた場合，周りの VM に与える性能低下の影響に差が生じる．図 5 では，バーストがない状態，高負荷な VM の I/O サイズが 64KB である場合と 4KB である場合を示している．高負荷 VM の I/O サイズが 64KB のものに比べ，高負荷 VM の I/O サイズが 4KB の方が VM2〜4 の得られるスループット値が大きい．4KB での I/O サイズのバーストの際には，VM2〜VM4 のスループットが向上している．今回提案したアルゴリズムを適用した場合，この 4KB の I/O サイズでバーストする VM1 は IOPS の観点で，ほかの VM よりも高い値を出しているので制御の対象となる．しかし，実際には，ほかの VM に与える性能低下の影響はなく，平均応答時間は悪

くなるものの，スループットが向上する．このように，スループットで比較した場合，4KB などの I/O サイズが小さい I/O の方がほかの VM に与える影響は少ない．しかし，ストレージのディスクに対する負荷は，I/O サイズが 64KB のものより 4KB のものの方が大きい．これは，細かな I/O が頻繁に発生するため，ディスクのシーク時間が余計にかかるためである．そのため，スループットの観点では，負荷が低く見えても，ストレージやディスクに対する負荷の観点からは負荷が大きくなる．このように，利用者に公平に I/O リソースを利用させるためには，負荷の大きさを単純にスループットや IOPS の観点のみで判断することは困難である．そのため，公平に VM へ I/O リソースを提供するためには，ディスクシーク時間などのストレージ内部の負荷に応じたパラメータを設定する必要がある．IOPS やスループットだけでなく，ディスク負荷なども考慮した制御パラメータの設定手法が今後の課題である．

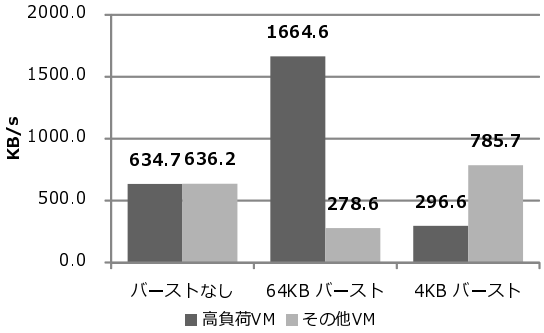


図 5 高負荷 VM の I/O サイズの変化による平均スループットの変化

6. まとめ

本論文では，マルチテナント環境における VM の I/O 性能において，最低 I/O 性能保証を保証と I/O リソースの有効活用を同時に実現するための方式を提案した．ベンチマークツールを用いて，VM の性能低下を引き起こす状態を疑似的に再現し，アルゴリズムの有効性を検証した．アルゴリズムを適用することで，VM の性能低下の影響を低減することができた．また，周りの VM の性能低下を引き起こす VM のエンキュー量を抑えることで，ストレージに対するエンキュー量も減少し，平均応答時間が改善した．しかし，すべての VM に対して，性能を維持できておらず，アルゴリズムの改良が必要であると考えられる．また，本アルゴリズムでは，IOPS とスループットの値を制御パラメータとして用いたが，様々な大きさの I/O が発生する環境において，I/O サイズによりストレージに対する負荷量が異なる．I/O サイズが小さい場合には，シークタイムが多く発生し，物理ディスクに対する負荷が高まる．一方で，大きなサイズの I/O では，シーク時間が短くなるが，ネットワーク帯域についての負荷が高まる．そのため，ストレージ筐体や HDD や SSD といったディスクも考慮した制御方法が必要になる．そのために，ユーザに決められた性能を提供する上で，ストレージの I/O 性能をどのように捉え，利用者に提供していくのかを明確にし，IOPS やスループットだけでなく，ディスク負荷なども考慮した性能保証手法が今後の課題である．

参考文献

[1] Jason Lango, ”Toward software-defined SLAs”, Communications of the ACM, Volume 57 Issue 1, pp.54-60, (2014)
[2] David Shue, Michael J. Freedman, Anees Shaikh, ”Performance isolation and fairness for multi-tenant cloud storage”, OSDI’12 Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation, pp.349-362, (2012).
[3] Hui Wang, Peter Varman, ”Nested QoS: Providing Flexible Performance in Shared IO Environment”, WIOV’11 Proceedings of the 3rd conference on I/O virtualization, pp.5-5 ,(2011).
[4] Ajay Gulati, Arif Merchant, Perter J. Varman, ”pClock: An Arrival Curve Based Approach For QoS Guarantees In Shared Storage System” , ACM SIGMETRICS international conference on Measurement and modeling of computer systems, pp.13-24, (2007).
[5] Sarala Arunagiri, Yipkei Kwok, Patricia J. Teller, Ricardo Portillo, Seetharami R. Seelam, ”FAIRIO: An Algorithm for Differentiated I/O Performance”, 2011 23rd International Symposium on Computer Architecture and High Performance Computing, pp.88-95, (2011).
[6] Ajay Gulati, Irfan Ahmad, Carl A. Waldspurger, ”PARDA: proportional allocation of resources for distributed storage access”, FAST ’09 Proccedings of the 7th conference on File and storage technologies, pp.85-98, (2009).
[7] Magnus Karlsson, Christos Karamanolis, Xiaoyun Zhu, ”Triage: Performance Differentiation for Storage Systems Using Adaptive Control”, ACM Transactions on Storage (TOS) , Volume 1 Issue 4, pp.457-480, (2005)
[8] Ajay Gulati, Ganesha Shanmuganathan, Irfan Ahmad, Carl Waldspurger, Mustafa Uysal ”Pesto: Online Storage Performance Management in Virtualized Datacenters” The 2nd ACM Symposium on Cloud Computing (SOCC), Article No. 19, (2011)
[9] Seetharami R. Seelam, Patricia J. Teller; “Virtual I/O Scheduler: A Scheduler of Schedulers for Performance Virtualization”; The 3rd international conference on Virtual execution environments (VEE); pp.105-115, (2007)
[10] Jainyong Zhang, Alma Riska, Anand Sivasubramanaim, Qian Wang, Erik Riedlel, ”Storage Performance Virtualization via Throughput and Latency Control”, Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS),pp.135-142, (2005)
[11] Xing Lin, Yun Mao, Feifei Li, Robert Ricci; “Towards Fair Sharing of Block Storage in a Multi-tenant Cloud”; HotCloud’12 Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing; pp. 15-15 (2012)
[12] Yubin Wu, Bowen Jia, Zhengwei Qi; “IO QoS: A New Disk I/O Scheduler Module with QoS Gurarantee for Cloud Platform”; Information Science and Engineering (ISISE), 2012 International Symposium; pp.441-444, (2012)