G-38

仮想化環境のための ディスクキャッシュの変化に着目した動的メモリ割り当て

石原 吉晃 †　芝 公仁 †

1　はじめに

近年，仮想化環境が普及し，1 台の計算機上で複数のオペレーティングシステム (OS) が動作する環境が増加している．仮想化環境でディスク I/O によるメモリの負荷が動的に変化する場合，静的なメモリ割り当てでは十分に物理メモリを活用することができない．この問題を解決するためには，各仮想計算機 (VM) に割り当てるべき最適なメモリ量を動的に算出し，それぞれに割り当てる必要がある．しかし，動的メモリ割り当ての既存手法では，VM 上のプロセスが使用するメモリ量については考慮されているが，必要なディスクキャッシュについては考慮されていない手法が多い．また，それらのディスクキャッシュについて考慮している手法も適用にはゲストとホストの OS の変更やパラメータの調査が必要となる．他にも，物理メモリ量を考慮せずに割り当てを行うため，ホスト OS でスワップが多発する可能性も考えられる．

本論文では，メモリ量の管理を各 VM が行う自律制御型とホスト OS が行う統合制御型の 2 つの手法について述べる．自律制御型では，各 VM 上で動作する提案機構がディスクキャッシュを考慮した上で VM のメモリを制御する．統合制御型では，ホスト OS 上で動作する提案機構が一定時間ごとに各 VM の情報を取得し，その情報から各 VM に適切なメモリ量を分配する．

本論文では，自律制御型の手法を Linux OS と オープンソースソフトウェアの仮想マシンモニタ (VMM) である QEMU が動作する環境に実装した．VM の動的メモリ割り当てには，QEMU が提供している機能を使用し，各 VM 上で動作する提案機構 Balloon Controller がメモリを割り当てる．実装した環境に対してファイルの読み込みを行うベンチマークプログラムを実行した結果，VM のメモリ量が読み込むファイル量に追従していることを確認できた．

以下，本論文では，2 章で本研究の関連研究について述べる．次に，3 章で自律制御型について，4 章で統合制御型について述べる．5 章で自律制御型の効果を確認する評価実験について述べ，最後に 6 章で本論文のまとめを述べる．

2　関連研究

前章で述べた通り，仮想化環境でメモリを活用するためには，VM のメモリ使用状況に対応して，割り当てるメモリ量を動的に変更する必要がある．仮想化環境では，VM が使用できるメモリを動的に変化させる機能を持っている．しかし，この機能は各 VM の最適なメモリ量を算出することはできず，各 VM に割り当てる最適なメモリ量は設定する必要がある．そこで，各 VM の最適なメモリ量を動的に算出するため，いくつかの手法 [1, 2, 3, 4] が提案されている．

仮想化ソフトウェアの Xen に実装されている xenballoon[1] では，VM 上のプロセスが確保しているメモリ量を最適なメモリ量として扱い，動的メモリ割り当てを行っている．結果，xenballoon ではプロセスが確保したメモリ量に対応した割り当てを実現している．しかしこの手法では，ディスクキャッシュが考慮されていないため I/O 性能が劣化する場合がある．また，無条件にメモリの割り当てを行うため，物理メモリ量を超えた割り当てが行われる可能性がある．

文献 [2] では，xenballoon を改善してディスクキャッシュの確保を行っている．各 VM へのメモリ割り当ては，まず各 VM から一定量のメモリを回収し，そのメモリを再配分用のメモリとする．そして，その再配分用のメモリを各 VM のキャッシュヒット率に応じた量を分配する．すなわち，この手法ではディスクキャッシュの確保だけでなく全 VM の優先度付けを行いメモリの分配を行っている．この優先度付けに従いメモリを与えることで，他の VM と比較して I/O 性能が向上する VM が優先的にメモリを獲得できる．しかしこの手法では，xenballoon とは異なりホストとゲストの OS を改変する必要がある．

文献 [3] では，xenballoon と同じく VM 上のプロセスが確保しているメモリ量を使用して，動的メモリ割り当てを行っている．さらに，この手法では VM に割り当てられたメモリ量と定数を乗算した値をディスクキャッシュ用のメモリとして確保している．しかし，乗算する定数は VM 上で動作するゲスト OS とプロセスに適した値を事前に調査して設定する必要がある．そのため，VM 上で特定のゲスト OS とプロセスのみが動作するような環境の場合，有効な手法となる．

本論文で述べる手法では，VM 上のプロセスが使用するメモリ量とディスクキャッシュを監視し，各 VM の最適なメモリ量を動的に算出する．自律制御型の手法は，ディスクキャッシュへの考慮をゲスト OS が提供する情報のみを使用するため OS を変更する必要はない．統合制御型の手法は，OS を変更する必要はあるが，ゲスト OS が提供する情報に加えて VM 上でどの程度のファイルの読み書きが行われているかを取得し，これらの情報を使用してディスクキャッシュが必要な VM には優先的にメモリを分配する．また，統合制御型はあらかじめ設定したメモリ量を各 VM に分配するため物理メモリ量を超える割り当ては行われない．

3　自律制御型の動的メモリ割り当て

自律制御型の動的メモリ割り当てでは，各 VM 上で提案機構 Balloon Controller が動作し，ゲスト OS が提供している情報から VM に割り当てるメモリ量の算出と割り当てを行う．VM に割り当てるメモリ量は，プロセスのメモリ使用量とディスクキャッシュを考慮して算出する．

† 龍谷大学, Ryukoku University

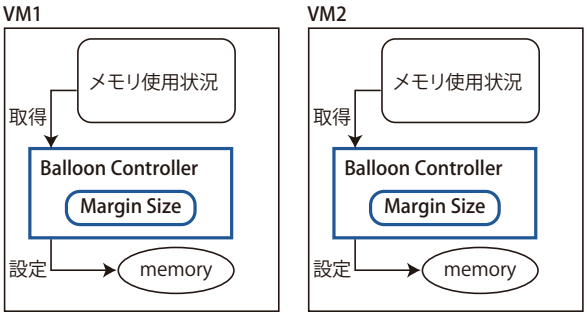

図 1　自律制御型動的メモリ割り当て

以降は，1 節で本手法の構成について述べ，2 節で動作について述べる．

3.1　構成

3.1.1　最適なメモリ量

本手法では，VM 上のプロセスが使用するメモリ量とディスクキャッシュを考慮して各 VM の最適なメモリ量を算出する．最適なメモリ量 W は式 (1) のように VM 上のプロセスが使用しているメモリ量と Balloon Controller が持っている可変量パラメータ Margin Size を合計した値である．

$$W = \text{使用中のメモリ量} + MarginSize \qquad (1)$$

使用中のメモリ量は VM 上のプロセスが確保したメモリの合計である．この値には，物理メモリを未割り当てのメモリも含まれている．本手法では，使用中のメモリ量を考慮して各 VM に割り当てるメモリ量を算出することで，VM 上のプロセスが使用するメモリ量を割り当てる．

ディスクキャッシュへの考慮は可変量のパラメータ Margin Size を使用する．この値は各 VM が十分なディスクキャッシュを確保できるように Balloon Controller が制御する．Balloon Controller は各 VM のキャッシュ量と最近使用したキャッシュ量を考慮して Margin Size を算出する．これらの情報はゲスト OS が提供する情報を使用する．

本手法では，上記のようにプロセスが使用するメモリ量とディスクキャッシュを考慮して各 VM に最適な量のメモリを割り当てることで，物理メモリを活用し，静的メモリ割り当てよりも高い I/O 性能の実現を図る．

3.1.2　Balloon Controller

自律制御型の構成を図 1 示す．本手法では，各 VM 上で提案機構 Balloon Controller が動作しており，VM に割り当てるメモリ量を算出して割り当てを行う．Balloon Controller は前項で述べた Margin Size の制御と VM への動的メモリ割り当てを行う．動的メモリ割り当ての実行は，前項で述べたように使用中のメモリ量と Margin Size を合計した値を最適なメモリ量とし，VM にメモリを割り当てる．

3.2　動作

提案機構 Balloon Controller を Linux カーネルと VMM である QEMU を使用した環境に実装した．動的メモリ割り当ては，QEMU に実装されている Balloon の機能を使用して行う．Balloon は，VM のメモリ上で膨張，収縮してメモリの動的割り当てを実現している．Balloon の機能は，実行中の QEMU を制御することができる QEMU Monitor に対して balloon コマンドを発行して使用する．また，ゲスト OS のメモリ使用状況は /proc/meminfo から取得した．前節で述べた

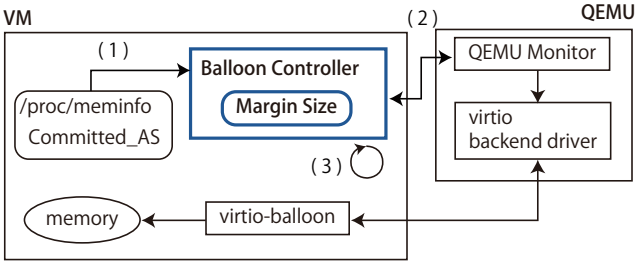

図 2　動的メモリ割り当ての実行手順 (自律制御型動的メモリ割り当て)

プロセスが使用しているメモリ量は Committted_AS，キャッシュ量は Cached，最近使用したキャッシュ量は Active(file) を使用している．

本節では，自律制御型の動的メモリ割り当ての動作について述べる．始めに，Balloon Controller が動的メモリ割り当てを行う手順を述べ，次に Margin Size を制御する手順を述べる．

3.2.1　動的メモリ割り当て

Balloon Controller は QEMU が持つ機能を使用して動的メモリ割り当てを行う．図 2 にその手順を示し，以下に処理の流れを述べる．

(1) Committed_AS の取得
(2) QEMU Monitor に対してコマンドを発行
(3) 1 秒間停止

始めに，/proc/meminfo から Committed_AS を取得する．取得した Committed_AS と Balloon Controller が保持している Margin Size を合計し，最適なメモリ量とする．次に QEMU Monitor に算出したメモリ量を指定して balloon コマンドを発行し，動的メモリ割り当てを要求する．QEMU が動的メモリ割り当てを要求されると QEMU に実装されている virtio backend driver とゲスト OS に実装されている virtio-balloon が virtio[5] を使用して通信を行い，動的メモリ割り当てを実行する．Balloon Controller は balloon コマンドを発行した後，1 秒間停止する．

以降，上記の処理を繰り返す．このように本手法では，1 秒間隔で balloon コマンドを QEMU Monitor に発行し，動的メモリ割り当てを実行する．

3.2.2　Margin Size の制御

図 3 に Balloon Controller が Margin Size を制御する手順を示し，以下に処理の流れを述べる．

(1) /proc/meminfo からメモリ使用状況を取得
(2) 取得した情報から判断して，Margin Size の値を更新
(3) 一定時間停止

まず /proc/meminfo からメモリ使用状況として，キャッシュ量 Cached と最近使用したキャッシュ量 Active(file) を取得する．次に取得した情報から判断して Margin Size を算出し，更新する．Margin Size の更新後は，5 秒間停止する．以降，上記の処理を繰り返す．

Margin Size の制御では，VM に対して UP か DOWN の状態を付与し，その状態に応じて Margin Size の増減を行う．

状態が UP の場合は，キャッシュ量が変動している，または最近使用されたキャッシュが増加している場合に Margin Size を増加させる．そうでない場合は，状態を DOWN に変更する．

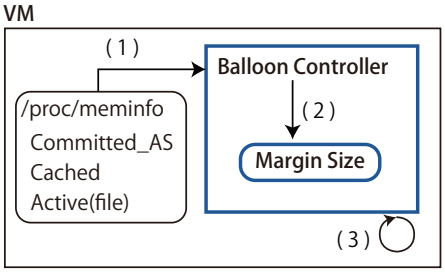

図 3　Margin Size の制御手順 (自律制御型動的メモリ割り当て)

VM 上で一定量の I/O が発生している場合，その I/O に対応した量のキャッシュが確保されると，キャッシュ量は変化しなくなる．本手法では，キャッシュが変化しなくなるまで Margin Size を増加させることで VM 上の I/O 量に適したメモリの割り当てを行っている．キャッシュ量が変化しなくなった場合，現在のキャッシュ量が最適な量である場合と現在のキャッシュ量よりも VM 上で発生している I/O 量の方が低い場合が考えられる．そのため，VM の状態が UP でかつキャッシュ量が変化しなくなった場合，VM の状態を DOWN に変更して最適なキャッシュ量を確認するため一度キャッシュ量を減少させる．

また，Margin Size は式 (2) のように UP 状態が続くにつれて増加する．上昇量の上限値は 200MB とする．

$$\text{上昇量} = 25 \times UP \text{ の連続回数 } [MB] \tag{2}$$

DOWN の場合，キャッシュ量が増加していない，または最近使用したキャッシュ量が減少していない場合に Margin Size を減少させる．メモリ量を減少する際は物理メモリを回収する必要があり，処理が完了するまで時間がかかる．そのため，Margin Size を減少させる際は，動的メモリ割り当てが完了するまで待機する．また，Margin Size の下限値は 100 MB とする．上記の Margin Size を減少させる条件に当てはまらない場合は，状態を UP に変更する．

Margin Size を減少させ動的メモリ割り当てが行われると，キャッシュ用のメモリが減少する．その時，キャッシュ用のメモリが不足した場合は，キャッシュの上書きによって本来有効に活用できるはずのキャッシュが削除されてしまう．そのため，Margin Size の減少により，有効なキャッシュが減少した場合はキャッシュ用のメモリが不足していると判断し，状態を UP に変更する．また，キャッシュ量が増加した場合も新たな I/O が発生したとして状態を UP に変更する．

Margin Size は，式 (3) のように DOWN 状態が続くにつれて減少する．減少量の上限値は UP の場合と同じく 200MB とする．

$$\text{減少量} = 50 \times DOWN \text{ の連続回数 } [MB] \tag{3}$$

また，Margin Size を減少する場合は，前回の状態が UP であり，Margin Size がキャッシュ量よりも多いならばキャッシュ量を Margin Size に設定する．

以上の処理は状態変化の有無に関わらず 5 秒間隔で行う．上記のように Margin Size を制御し，各 VM にメモリを割り当てることで各 VM の I/O 量に適したディスクキャッシュが確保される．

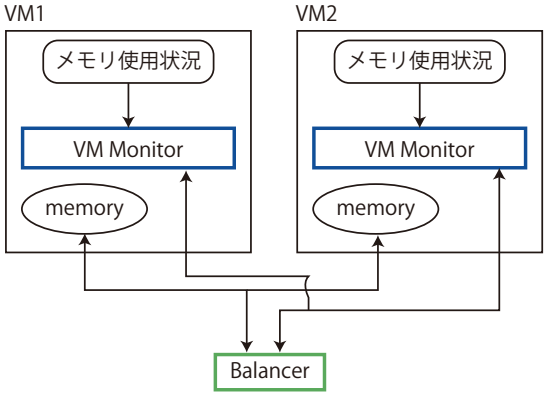

図 4　統合制御型動的メモリ割り当て

4　統合制御型動的メモリ割り当て

統合制御型動的メモリ割り当てでは，ホスト OS で各 VM のメモリを制御する Balancer が動作しており，各 VM 上では Balancer からのリクエストに応じて VM のメモリ使用状況を取得する VM Monitor が動作している．Balancer は各 VM からメモリ使用状況を取得し，VM のメモリが不足していると判断した場合，各 VM のメモリ配分を調整する．

以降は，1 節で本手法の構成について述べ，2 節で動作について述べる．

4.1　構成

統合制御型動的メモリ割り当ての構成を図 4 に示す．本手法では図 4 の Balancer はホスト OS 上で動作し，VM Monitor は各 VM 上で動作する．

Balancer は，どの VM を管理するのかを登録し，登録された VM に対して一定時間ごとに情報取得を行う．そして，VM のメモリが不足していると判断した場合，他の VM の余っているメモリを不足している VM に割り当てる．Balancer は，あらかじめ設定されているメモリ量を各 VM に分配する．そのため，物理メモリ量よりも多い値を設定しなければ，物理メモリ量を超えた割り当ては行われない．

VM Monitor は，動作する VM が起動した際に Balancer に通知し，自身が動作する VM を Balancer に登録する．以降は，Balancer からの要求に応じて，メモリの使用状況として，メモリの総容量，空き容量，キャッシュ量，ファイルを読み書きした回数を取得する．ファイルを読み書きした回数はゲスト OS を変更して取得する．

4.2　動作

Balancer が各 VM からメモリ使用状況を取得する手順を図 5 に，動的メモリ割り当てを行う手順を図 6 に示す．本節では，ホスト OS 上で動作する Balancer と各 VM 上で動作する VM Monitor がどのような動作するのか，例を挙げながら説明する．ここでは，1 台の物理計算機上で 2 台の VM が動作し，分配するメモリ量は 4GB と設定した場合の動作を述べる．また，本手法を実装した環境は前章で述べた自律制御型動的メモリ割り当てと同じ環境とする．

まず，1 台目の VM が起動した際に VM Monitor が Balancer に通知を送り，Balancer のメモリ管理下に登録する．Balancer は一定時間ごとに登録された VM の情報取得を行う．そのため，登録された 1 台目の VM に対して情報取得を行う．

現時点では，Balancer はどの VM にも割り当てを行っていないため，設定されているメモリ量の 4GB を 1 台目の VM に割り当てる．また，Balancer が VM を一台も管理していない状態で複数の VM が追加されていた場合は，設定されているメモリ量を均等に分配する．

次に 2 台目の VM を起動して，VM Monitor が Balancer に登録を行う．登録後に Balancer が情報取得を行うと，新たな VM が登録されているため，この追加された 2 台目の VM に対してメモリを割り当てる．このようにすでに Balancer が他の VM にメモリが割り当てている場合は，一定量のメモリを他の VM から均等に回収して，回収したメモリを追加された VM に割り当てる．そのため今回の場合は，一定量を 1 台目の VM から回収し，2 台目の VM に割り当てる．

Balancer に追加された直後の VM へのメモリ割り当ては上記のようにして行う．また，各 VM へのメモリの割り当ては前章の自律制御型動的メモリ割り当ての Balloon Controller と同じく Balancer がメモリを割り当てる VM の QEMU Monitor に対して balloon コマンドを発行してメモリ割り当てを実行する．

2 台の VM が Balancer に登録された後は，各 VM のメモリ使用状況の取得を一定時間ごとに繰り返す．この際に取得する情報は前節で述べた通り，その VM のメモリの総容量，空き容量，キャッシュ量，ファイルを読み書きした回数とする．Balancer は，情報取得をした際にまず取得した各 VM のメモリの総容量と空き容量から使用中のメモリ量の割合を求める．そして，その割合が 95%を超えている VM はメモリが不足している状態の VM とする．もし，メモリが不足している VM が存在する場合は，他の VM からメモリを回収して，不足している VM に回収したメモリを均等に分配する．メモリの回収では，まずメモリが不足していない VM の空き容量を回収する．さらにメモリが不足しておらずファイルの読み書きの回数が一定数を下回っている VM から，キャッシュ量を回収する．ここで，回収を行った VM に対して回収後のメモリ量を定数倍した値を余裕分として回収したメモリ量から引いてその VM に加算する．

今回の例では，もし情報取得時に 1 台目の VM のみメモリが不足しており，2 台目の VM でファイルの読み書きが一定数を下回っている場合は，2 台目の VM からメモリの空き容量とキャッシュ量を余裕分を持たせた上で回収し，1 台目の VM に割り当てる．もし，2 台の VM のメモリが共に不足していた場合は，何も行わない．

上記のように Balancer への登録，各 VM からの情報取得，メモリの割り当てを行うことで，プロセスが使用するメモリとディスクキャッシュが不足している VM に対して，余裕がある VM から回収したメモリを分配することでメモリを必要としている VM に対して優先的にメモリを割り当て，物理メモリを静的メモリ割り当てよりも活用する．また，前章の自律制御型動的メモリ割り当てでは最適なキャッシュ量であるか確認するために常に VM のメモリ量を変動させる必要があった．そのため，最適なメモリ量を長時間維持するとことができない．もし静的メモリ割り当てで十分対応できるような I/O 量しか発生しない場合，常に必要なキャッシュ量を確保できる静的メモリ割り当てと比較すると自律制御型動的メモリ割り当ての性能が劣化する可能性が考えられる．統合制御型動的メモリ割り当てでは，閾値を超えた場合のみ VM のメモリを変動させる

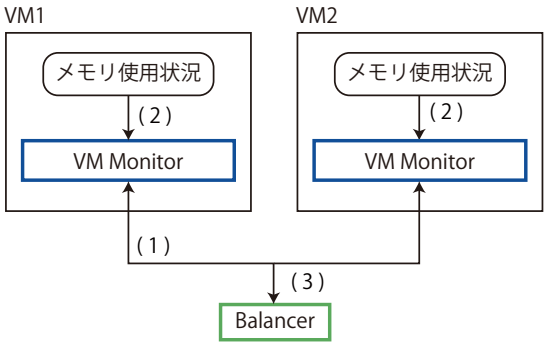

図 5　各 VM のメモリ情報取得手順 (統合制御型動的メモリ割り当て)

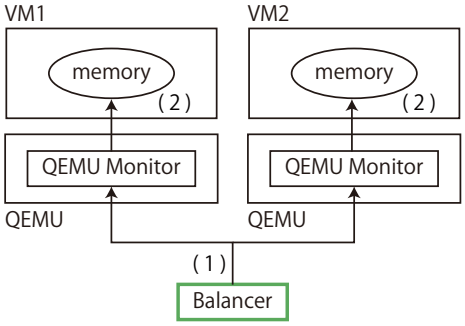

図 6　動的メモリ割り当ての実行手順 (統合制御型動的メモリ割り当て)

ため，自律制御型動的メモリ割り当てで考えられるような劣化を解決できると考えられる．さらに，あらかじめ定められたメモリ量を分配するため，これまでの既存手法や自律制御型動的メモリ割り当てのように物理メモリ量を超えた割り当てを防ぐことができる．

5　評価

本章では，自律制御型動的メモリ割り当ての手法が I/O 量に対応して適切に Margin Size を制御し，静的メモリ割り当てと比較して I/O 性能が向上しているか確認する．1 節で，評価実験について述べ，2 節でその実験結果を示す．そして，最後に 3 節で考察を述べる．

5.1　評価方法

実験では，1 台の物理計算機上で VM を 2 台同時に稼働させ，各 VM 上でファイル読み込みを行う計測プログラムを動作させる．計測プログラムは 30 分間 ファイルの読み出しを行う．読み出すファイルは 1 万個のファイルから一定数のファイルを選択する．ファイルは一様分布乱数を使用して選択する．各ファイルの容量は 1MB で，読み出し対象となるファイルの数は 10 分ごとに変化させる．読み出すファイルの数は各 VM で 500，0，500 と 0，500，0 のように一定数のファイル読み出しを交互に行う．また，読み出すファイルの数は 500 から 7000 個まで 500 刻みで変化させる．以上の実験を自律制御型動的メモリ割り当てを適用した環境と均等にメモリを分配した静的メモリ割り当ての環境で行い，結果を比較する．

表 1，2 に実験で使用する環境を示す．実験環境では，物理メモリ量が 4096 MB で，本手法を適用した環境の各 VM は 3072 MB 以下のメモリが動的に割り当てられる．均等にメモ

表 1　物理計算機の構成

OS	Debian 9.2
Kernel	Linux-4.11.2
CPU	Core i5-6600 3.30GHz
CPU Core	4
Memory	4GB
HDD	2TB

表 2　仮想計算機の構成

OS	Ubuntu 17.10
Kernel	Linux-4.13.0
Virtual CPU Core	1
Virtual Memory	計測によって変動

リを分配した静的メモリ割り当ての環境は各 VM に 1536 MB のメモリを割り当てる．

5.2　実験結果

図 7，8 に 2000 MB のファイルを 2 つの VM で交互に読み出した時の各 VM のメモリ量の変化を示す．縦軸は VM が読み出すファイルサイズと VM に割り当てられたメモリ量，横軸は計測プログラム開始からの経過時間である．図中の opti_size は VM 上で動作しているプロセスが読み込むファイルの容量で，MemTotal は VM に割り当てられたメモリ量である．図 7 から，読み出すファイル量が変化する約 600 秒で，遅れてはいるが不要なメモリが回収されて，VM のメモリ量が減少している．また読み出すファイルが増加して必要なメモリ量が増加する約 1200 秒でも，そのファイル量に対応して VM に割り当てられたメモリ量が増加している．図 8 も図 7 と同じように読み出すファイル量が変化する約 600 秒と約 1200 秒で，メモリ量が読み出すファイル量に対応して増減している．以上のことから，I/O 量の増減に対応して Balloon Controller が適切に Margin Size を制御していることが確認できた．

図 9 に，2 台の VM で行ったファイル読み出しの速度を示す．縦軸は 1 秒間で読み出したファイルサイズ，横軸は VM 上で読み出すファイルサイズである．自律制御型動的メモリ割り当てを適用した環境で，読み出すファイル量が 1500 MB 以上の場合，静的メモリ割り当てと比較して読み出し速度が最大約 9.5 倍に向上していた．しかし，約 1500 MB 以下の場合，読み出し速度が約 14 % 劣化していた．また，読み出すファイル量が 3000 MB 以上になると，自律制御型動的メモリ割り当てを適用した環境でも読み出し性能が劣化していった．そして，読み出すファイル量が 6000 MB 以上では自律制御型動的メモリ割り当てを適用した環境も静的メモリ割り当てと変わらない性能となった．

5.3　考察

図 7，8 から，Margin Size がうまく制御され，ファイル読み出しに対応して，VM のメモリ量が割り当てられていることを確認した．

図 7 で約 600 秒から約 1200 秒の間，メモリ量が変化しなかったのは，Margin Size の下限値が 100 MB に設定されているからであると考えられる．Balloon Controller が Margin Size を制御する際，VM の状態が DOWN で 100 MB 以下の値が Margin Size として算出された場合，Margin Size を

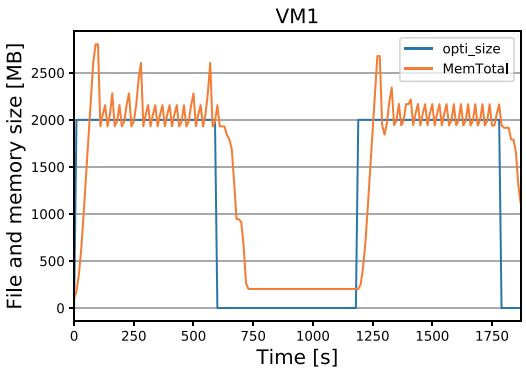

図 7　読み出すファイル量が 10 分ごとに 2000，0，2000 と変化する VM のメモリ量

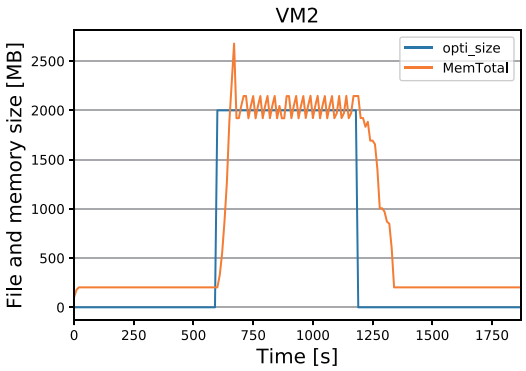

図 8　読み出すファイル量が 10 分ごとに 0，2000，0 と変化する VM のメモリ量

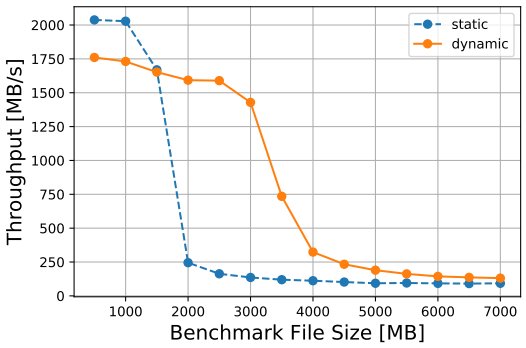

図 9　提案手法と静的メモリ割り当てのファイル読み出し速度

100 MB に設定する．また，DOWN から UP の状態に遷移するためには，キャッシュ量が増加するか最近使用したキャッシュ量が大きく減少する必要がある．ファイル読み出しが発生しない場合はこの条件には当てはまらない．そのため，ファイル読み出しが停止した VM は，100 MB まで Margin Size が減少し，UP に状態が変更されることもなく DOWN の状態が続き，VM のメモリ量が変化していなかったと考えられる．

図 9 から，読み出すファイルのサイズが一定量を超えると提案手法を適用した環境の方が静的メモリ割り当てよりも読み出し速度が向上した．これは，静的メモリ割り当てよりも，提案手法を適用した環境の方が使用できるメモリ量が多いから

であると考えられる．静的メモリ割り当ての場合，均等にメモリを分配し，1536 MB のメモリしか使用できない．一方，提案手法を適用した環境では，最大 3072 MB のメモリを使用することができる．そのため，確保できるキャッシュ量に差ができ，静的メモリ割り当てと比較してファイル読み出し速度が向上した．そして，読み出すファイルサイズが 3000 MB を超えると確保できるキャッシュ量以上の読み出しが発生し，自律制御型動的メモリ割り当てを適用した環境もファイルの読み出し速度が劣化していった．

また，1500 MB までは，静的メモリ割り当ての方が提案手法よりもファイルの読み出し速度が高かった．これは，Balloon Controller が常にメモリ量を変化させているため，最適なメモリ量を長時間維持できないことが原因として考えられる．

6 おわりに

本論文では，自律制御型と統合制御型の動的メモリ割り当てについて述べた．これらの手法では，プロセスが使用するメモリ量とディスクキャッシュを考慮して VM への動的メモリ割り当てを行っている．自律制御型動的メモリ割り当てを実装して実験を行った結果，読み出すファイルサイズが一定量を超えると，静的メモリ割り当てよりも本手法の方が読み出し速度が向上した．

現在，統合制御型ではディスクキャッシュを回収する際に VM 上でファイルの読み書きが行われていなければ全ディスクキャッシュを回収するような極端な制御となっている．そのため今後は，VM 上でファイルの読み書きが行われている場合でもディスクキャッシュ量が過剰に与えられていないかを考慮できるような手法を検討していきたい．

参考文献

[1] Stephen Spector．MemoryOvercommit. https://blog.xenproject.org/2008/08/27/xen-33-feature-memory-overcommit/，2008.

[2] 日名川幸矢，竹内洸祐，山口実靖．仮想化環境におけるキャッシュヒット率を考慮した VM メモリ割り当て．研究報告マルチメディア通信と分散処理（DPS） 2013-DPS-154(17)，2013-03-07.

[3] Anna Melekhove，Larisa Markeeva．Estimating Working Set Size by Guest OS Performance Counters Means．The Sixth International Conference on Cloud Computing, GRIDs, and Virtualization，2015.

[4] Automatic Ballooning. https://www.linux-kvm.org/page/Projects/auto-ballooning，2013.

[5] Rusty Russell．virtio: Towards a De-Facto Standard For Virtual I/O Devices，2008.