Vol.2014-ARC-210 No.6
Vol.2014-OS-129 No.6
2014/5/14

仮想化環境における データベース管理システムのメモリ管理手法

福地 開帆[†1,a)]　山田 浩史[†1,b)]

概要：近年データベース管理システム (DBMS) は多くの Web サービスの根幹を担っている．また，仮想化技術の普及により，クラウド環境などの仮想マシン (VM) 上で DBMS が動作する機会も増えている．通常，OS と仮想マシンモニタ (VMM) がメモリなどの資源を管理している一方で，DBMS は OS や VMM とは独立したポリシーに基づいて自身でメモリなどの資源を管理する．そのため，DBMS と VMM とのメモリ管理が競合してしまうことがある．たとえば，VMM は OS のメモリ管理と連動することにより，バルーニングと呼ばれる動作を行い OS のメモリサイズを動的に変更する．その際，OS と DBMS のメモリ管理が連動していないため，OS は DBMS の独自のメモリ管理方法を認識できず，OS は DBMS のメモリをハードディスクにスワップアウトしてしまう．さらに DBMS も自身が管理しているメモリがスワップアウトされていることを知らないため，結果として DBMS は本来の性能を発揮できなくなる．本研究では，VMM のメモリ管理との競合を回避する DBMS のメモリ管理手法を提案する．本研究では，DBMS と VMM のメモリ管理との競合を回避するために，DBMS と OS とのメモリ管理を連動させるアプローチをとる．OS と DBMS のメモリ管理が連動して動作することを可能とするため，1. DBMS のメモリ管理は引き続き DBMS が行うが，OS のメモリ管理処理内から必要に応じて DBMS にメモリの解放・確保の依頼を行う 2. DBMS のメモリ情報を DBMS が OS に共有し，OS が DBMS のメモリを管理する 3. DBMS のデータ構造とメモリ内容を OS・DBMS 間で共有し，OS が DBMS のデータ構造を認識した上で OS が DBMS のメモリを管理する，という 3 通りの方法を提案する．今回は Linux 3.11.5 と MySQL 5.6.14 に実装を行った．実験により，VMM によるバルーニングが行われても DBMS が本来の性能を発揮することを確認した．

1. はじめに

仮想化技術の利用がクラウドなどのデータセンタで広がりを見せている．たとえば，Amazon EC2 や Google Compute Engine などがある．これらクラウド環境では目的やタスクの負荷に応じて容易に複数の仮想マシン (VM) を起動することができる．そのため，ある VM を WEB サーバに，もう一つの VM を DB サーバとして起動し，さらに高負荷時には VM を追加で起動し負荷分散を行うといったことが可能である．仮想化環境では柔軟に VM の追加や削除が行えるため，ワークロードの変化に適応しやすくなっている．

大容量のデータを扱う機会が増えたことなどにより，データベース管理システム (DBMS) がサービスを構成するうえでの重要な要素となっている．DBMS を用いることでデータの一貫性の維持や，大量のデータから目的のものを高速に取得するといったことが可能である．Facebook や Twitter などの WEB サービスは，DBMS を利用することで大量のユーザー情報の管理や投稿されたデータの高速な検索を可能にしている．DBMS はクラウド上でも実行され，Amazon RDS や，Google Cloud SQL などはクラウド環境の上に DBMS を構築している．クラウド上で DBMS を実行することで，クラウドとそのベースである仮想化技術の恩恵を受けることができる．

しかし，DBMS は仮想化環境では十分な性能を発揮しない場合がある．DBMS は VMM や OS とは独立した自身の資源管理ポリシーによって自身で資源を管理している．たとえば，ディスクへのフラッシュを遅延してまとめて行なったり，まとまったメモリ領域を確保しそれを自身で各処理に割り振るなどである．しかし通常，仮想化環境では VMM が資源管理を行い，OS に資源を割り当てている．そのため，VMM と DBMS の資源管理が競合してしまい，

†1 現在，東京農工大学
Presently with Tokyo University of Agriculture and Technology
a) fukuchi@asg.cs.tuat.ac.jp
b) hiroshiy@cc.tuat.ac.jp

© 2014 Information Processing Society of Japan

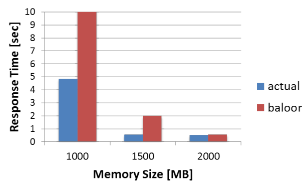

図 1 バルーニングが MySQL の性能に与える影響

DBMS が仮想化環境で本来の性能を発揮することができない場合がある．たとえば，VMM と OS は連動して資源を管理することで，バルーニングと呼ばれる動作をする．これは OS を起動したままで，VM へのメモリ割り当て量を動的に変えるものである．バルーニングにより，高負荷になった VM にはより多くのメモリを割り当て，代わりに低負荷の VM には少ないメモリを割り当てる，といったことを再起動なしで行える．しかしこの際 VMM と DBMS の資源管理が協調して動作しないため，VMM や OS は DBMS にとって不都合な資源管理を行う場合がある．

バルーニングの DBMS への影響を調べるため予備実験を行なった．DBMS を起動したのちバルーニングにより VM のメモリサイズを変更した場合と，元から変更済みのメモリサイズを与えた VM 上で DBMS を起動した場合の性能を比較した．DBMS には MySQL を用いた．MySQL の場合は，VM のメモリサイズを 5GB，バッファプールサイズを 3.5GB で起動したのち，VM のメモリを 1GB，1.5GB，2GB それぞれにバルーニングした場合と，元から VM のメモリを 1GB，1.5GB，2GB，それぞれのバッファプールサイズを 0.7GB，1.05GB，1.4GB で起動した場合の性能を比較した．

結果は図 1 のようになった．同じメモリサイズの VM であっても，バルーニングを行なった場合は DBMS の性能が大幅に低下した．これは，VMM が DBMS の独自のメモリ管理方法を認識できず DBMS にとって不都合なメモリ管理を行ったためと，一方の DBMS 側も VMM と OS のメモリ管理に適応できなかったためである．

そこで，本研究では，VMM のメモリ管理との競合を回避する DBMS のメモリ管理手法を提案する．本研究では，DBMS と VMM とのメモリ管理の競合を回避するために，DBMS と OS とのメモリ管理を連動させるアプローチをとる．DBMS と OS とのメモリ管理を連動する方法として，本研究では 3 つの案を示す．1 つめは，今までどおり DBMS がメモリを管理するが，必要に応じて OS が DBMS にバルーニングを依頼する方法である．2 つめは，DBMS が自身が管理していたメモリの用途などを OS に通知し，OS がそれに基づいて管理を行う方法である．3 つめは，OS が DBMS のデータ構造やポリシーといったメモリ管理方法を認識し，それに基づいて OS 側で管理する方法である．3 つの方法では，最終的に OS と DBMS のどちらでメモリを管理するかと，お互いのメモリ管理ポリシーを相手にどれだけ共有するかが異なる．

今回は MySQL 5.6.11 と Linux 3.11.5 に対して実装を行った．これらは一般的な DBMS と OS カーネルである．実験により，提案手法によりバルーニング後も DBMS が本来の性能を発揮することを確認した．さらに，複数種類の TPC-H クエリを用いた実験により，クエリの種類やバルーニングの条件が異なっても提案手法が有効であることを確認した．さらに，オーバーヘッドもごく僅かであることを確認した．

2. 背景

2.1 バルーニング

VMM は VM のメモリサイズを動的に変更するバルーニング [1] を行う．バルーニングにより，高負荷状態の VM により多くのメモリを割り当て，その分低負荷の VM には少量のメモリを割り当てる，といったことを VM の再起動無しで行える．

Xen ではゲストカーネル内で動作するバルーンドライバが利用される．バルーンドライバの使用するメモリ量が変化することで，OS 全体の使用可能メモリ量を変化させている．VM のメモリサイズを減らす場合，バルーンドライバ自身がメモリを確保して OS から奪ったのち VMM に返却し，マッピングの解除を行っている．VM のメモリサイズを増やす場合，バルーンドライバがメモリを OS へ返すか，あるいは VMM に新たなメモリ割り当てを要求する．

バルーンドライバは，OS の処理に基づいて通常のプロセスと同様の手段でメモリを確保する．そのため，OS 自身のメモリ管理ポリシーに則ったバルーニングが可能となっている．OS 本来のメモリ管理ポリシーが用いられるため，バルーニングのために OS やアプリケーションを修正する必要はない．

2.2 仮想化環境でのデータベース管理システム

自身で資源を管理する DBMS は，仮想化環境において十分な性能を発揮できない場合がある．VMM と OS は協調してメモリなどの資源の管理を行うが，VMM と DBMS は協調して動作しない．DBMS 自身が資源を管理している場合，その資源の用途や管理ポリシーを VMM が認識することは難しい．

2.2.1 バルーニング時の DBMS の性能劣化

バルーニングによって，DBMS の性能が著しく低下することがある．OS，DBMS それぞれが自身のポリシーに基づいてメモリの管理を行う．DBMS が OS のメモリ管理とは独立して自身でメモリを管理していることにより，OS が DBMS のメモリ管理ポリシーを認識できず，OS は DBMS にとって不都合なメモリ管理を行なっている場合がある．

たとえば，図 2 のようなことが発生する場合がある．DBMS は自身の管理するバッファプール内のページの多

© 2014 Information Processing Society of Japan

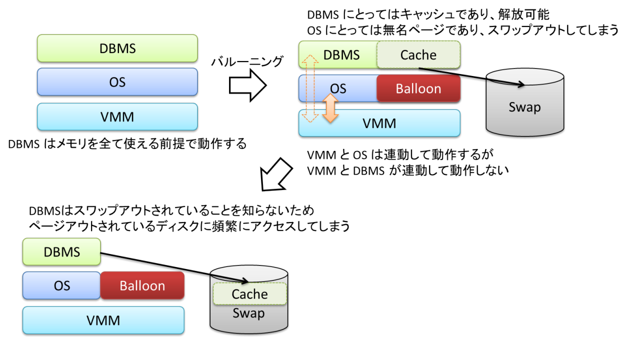

図 2　バルーニング時の DBMS の性能劣化

くをキャッシュとして用いているが，OS 側からはそれら領域は単なる無名ページとしてしか認識できない．そのため，OS のページ回収処理時には，本来は DBMS にとってはキャッシュであり解放しても動作に問題がない領域であるのに，OS は無名ページとしてディスクにスワップアウトしてしまう．その結果，DBMS はバッファプールがメモリ上に存在しているという認識の元でバッファプールを利用するが，実際はディスクにスワップアウトされているため性能が劣化してしまう．バルーニング時に DBMS の性能が劣化することは 1 章の図 1 にて既に述べた通りである．

2.2.2　OS のメモリ管理と DBMS のメモリ管理の競合

DBMS が自身でメモリを管理せず，OS にメモリ管理を任せた際にどうなるかを見るために予備実験を行なった．DBMS のメモリ管理を OS が担えば，バルーニング時の性能劣化を回避できるためである．その結果，単に OS にメモリ管理を任せるだけでは性能が劣化することが分かった．DBMS のメモリは DBMS のポリシーに基づいて管理しないと性能が劣化する．

予備実験は次の方法で行った．実メモリに占める，OS のページキャッシュと MySQL が管理するバッファプールの割合を変えつつ 3 種類のベンチマークを実行した．実験は Xen 4.2.1 上の VM で行い，dom0・domU 共に Linux 3.8.6 とした．VM にはメモリを 4GB 割り当てた．ベンチマークは TPC-C [2]， TPC-E [3]， TPC-H [4] を使用した．

図 3・図 4・図 5 はそれぞれ，バッファプールサイズを変化させた際の，TPC-C，TPC-E，TPC-H ベンチマークの実行結果である．バッファプールサイズが大きくなるに連れて性能が向上していることが分かる．バッファプールサイズが小さい場合，DBMS の管理するメモリ領域が減るため，より多くのメモリを OS が管理することになる．しかし OS にとっては DBMS は単なるアプリケーションであり，特殊なメモリ管理ポリシーを持っていることを認識できないため，DBMS にとって不都合なメモリ管理を行なってしまう．特に書き込みの場合，InnoDB のメモリ管理ポリシーを用いない場合遅延的な書き込みができず，ランダムアクセスの頻発により性能が劣化している．

よって，バッファプールサイズを小さくし単に OS にメモリ管理を任せるだけでは性能が劣化すると言える．DBMS のメモリは DBMS のポリシーに基づいて管理する必要がある．

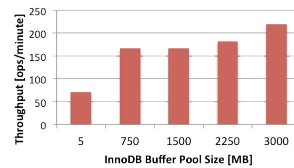

図 3　バッファプールサイズごとの TPC-C の結果

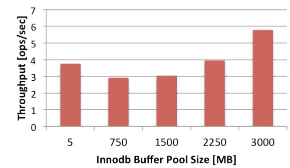

図 4　バッファプールサイズごとの TPC-E の結果

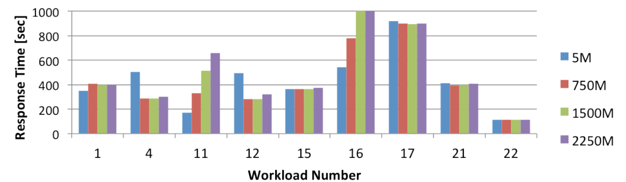

図 5　バッファプールサイズを変えたときの TPC-H の結果

3.　提案

本研究では，VMM のメモリ管理との競合を回避する DBMS のメモリ管理手法を提案する．本研究では，DBMS と VMM のメモリ管理との競合を回避するために，DBMS と OS とのメモリ管理を連動させるアプローチをとる．連動させる方法として，以下の 3 種類の実現案を提案する．

3 種類それぞれの案では，DBMS と OS 間での情報の共有の度合いと，最終的に DBMS と OS のどちらがメモリを管理するかが異なる．お互いの内部でのメモリ管理方法全てを共有することは困難なため，OS と DBMS のどちらで最終的にメモリを管理するかによりどちらのポリシーを最大限に活かせるかが変わる．表 3 はそれぞれの案を適用する場合に DBMS，OS 側で必要となる修正量と特徴である．

1. DBMS のメモリ管理は引き続き DBMS が行うが，OS のページ回収・確保処理時に必要に応じて OS が DBMS にメモリ解放・確保の依頼を行う
2. DBMS のメモリ情報を DBMS が OS に共有し，OS が DBMS のメモリを管理する
3. DBMS のデータ構造とメモリ内容を OS・DBMS 間で共有し，OS が DBMS のデータ構造を認識した上でメモリを管理する

DBMS のバッファプール上には，削除しても動作には問題がないキャッシュ領域が存在する．それらを OS のページキャッシュと同様にスワップアウトなしで回収することで，バルーニング時の性能低下を防ぐ．3 種類の方法いずれの場合においても，OS のページ回収処理時はまずは従来通りページキャッシュを解放し，それでも足りなければ DBMS の管理するキャッシュを，それでも足りない場合は従来通り無名ページや dirty ページを回収するようにする．

	DBMS 側の修正量	OS 側の修正量	OS の都合に合わせたメモリ管理の可否
案 1	少ない	少ない	場合による (DBMS に依存)
案 2	中程度	中程度	可能
案 3	少ない	多い	可能

表 1 それぞれの実現案の比較

4. 設計

4.1 案 1:DBMS が自身のメモリを管理し, OS が DBMS にメモリの解放・確保を依頼

4.1.1 概要

案 1 では，DBMS のメモリ管理は引き続き DBMS が行う．そして必要に応じて OS が DBMS にメモリの解放・確保を依頼する．バルーニングが起こると，OS はまずはページキャッシュやクリーンなページを解放する．ページキャッシュやクリーンなページが少なくなると，OS は DBMS に使用メモリの削減を依頼し，DBMS 自身に使用メモリ量を減らしてもらう．OS が DBMS の使用するべきメモリ量を求め DBMS に通知し，DBMS がそれに基づいて自身でバルーニングを行うことで，DBMS 自身に使用メモリ量を減らしてもらう．

そのために，DBMS 側に，DBMS が使用するべきメモリ量の通知を OS から受け取るためのインターフェースを追加する．このインターフェースを通じて，バルーニングの際は OS が DBMS に DBMS の使うべきメモリ量を通知し，DBMS はその使用メモリ量の目標を達成するべく自身でメモリの解放や確保を行う．

DBMS は OS からの通知に従って自身の使用メモリ量を動的に変更する．通常の処理では，DBMS が最初に一括して大きなメモリを確保し，その後はその領域を自身で管理して適宜必要な処理に割り当てていた．処理に使われたメモリはキャッシュとして残したり，あるいは新しい処理のために用いるなど，確保したメモリを解放して OS に返すことはせずに DBMS 自身で管理し続けていた．しかしインターフェースの追加により，DBMS は OS からの通知に従って自身の管理しているメモリ領域の中で不要な領域を解放し，OS に返還するようになる．

4.1.2 DBMS 側の設計

DBMS が使用するべき実メモリ量を OS から受け取るためのインターフェースを追加する．OS が，メモリ管理処理時に DBMS の使用するべき実メモリ量を求め，このインターフェースを通じて OS が DBMS に通知する．DBMS は OS から通知を受け取ると，そのメモリ量に向かって自身の管理するメモリの解放や確保を行う．解放や確保は，DBMS が管理するメモリ中でも特に free リストや LRU リストなどのキャッシュ領域のページに対して行う．

バルーニング時に DBMS が回収したページは，DBMS 内に新たに追加するバルーンリストで管理する．DBMS はバルーンリストを回収されたページを格納するリストとして用い，バルーンリストに属するページは通常の処理には使用しない．バルーニング時に使用メモリサイズを減らす場合には，まず free リストや LRU リストからページを抜き出し，そのページが指すデータ領域のメモリを解放して OS に返還したのち，そのページをバルーンリストに移す．使用するメモリサイズを増やす場合には，バルーンリストからページを抜き出し，データ領域用に新たに OS からメモリを確保した後そのページがそのデータ領域を指すようにし，ページを free リストに戻す．

ただし，OS のデマントページングにより，DBMS のページが指すデータ領域に実際に実メモリが割り当てられているとは限らない．そこで，DBMS 側でもページが指すデータ領域が一度でも処理に使われたかどうかを管理する．ページ回収とメモリ解放の際に，そのページがまだ一度も処理に使われていなければ，そのページはスキップして 1 度でも処理に使われたことのあるページを探す．

4.1.3 OS 側の設計

スワップアウト処理の発生をできるだけ抑止するために，ページ回収の優先順位を，1. ページキャッシュとクリーンなページ 2.DBMS の管理するページ 3. 無名ページとする．無名ページはスワップアウト処理が必要であり，DBMS の管理するメモリ領域に割り当てられたページも OS にとっては無名ページある．しかし DBMS に解放依頼を行い DBMS 自身に解放してもらえばスワップアウトの必要がない．OS は，DBMS の管理するページも含む無名ページ全般が回収対象となったとき，DBMS のインターフェースを通じてメモリ使用量の削減を依頼する．これによりできる限り DBMS 側で DBMS の使用するメモリの解放を行ってもらうようにする．DBMS のメモリ使用量と OS 全体のメモリ使用量を減らすことで，無名ページがスワップアウトされることも極力防ぐ．

案 1 ではカーネル内のデータ構造は変更しない．カーネル内のページ回収処理時に適宜 DBMS のインターフェースを用いて DBMS が使用するべきメモリ量の通知を行うのみである．

ただし，DBMS の管理するメモリを DBMS 側で解放するためには，一度 DBMS にプロセススケジューリングを行う必要がある．そのため，カーネルがインターフェースを通じて DBMS の使用するべきメモリ量を DBMS に通知しメモリの解放を依頼しても，即座に解放が行われるとは限らない．DBMS による解放が間に合わず，DBMS 自身

ではなく OS が DBMS の管理する領域の回収を試みるとスワップアウトが発生してしまう．

そこで，DBMS による解放ができるだけ間に合うようにするために，DBMS のページがカーネル内の inactive リストに移されようとした場合にも DBMS に通知を行い，予めメモリ使用量の削減を依頼しておく．そして inactive リストに移されようとしていた DBMS のページを再び active リストに戻す．inactive リストにページが移されたということは近いうちに回収対象のページとなりスワップアウトされてしまうおそれがあるため，予め DBMS 側で解放してもらうようにする．

4.2 案 2 : DBMS がメモリ情報を OS に共有し，OS が DBMS のメモリを管理

4.2.1 概要

案 2 では，DBMS の管理するメモリ領域であっても，OS 側で解放や確保処理を行う．DBMS は，自身の管理するメモリ領域の中でのキャッシュ領域の位置やその領域の解放可否など，DBMS のみが持つ情報を予め OS に伝えておく．その情報を元に OS 側が DBMS のメモリを管理する．どのメモリ領域が解放可能かは DBMS の処理中に頻繁に変化するが，その都度 DBMS はシステムコールで OS に解放可否を通知する．OS は，DBMS の管理するメモリ領域のメモリを解放や確保を行なった際にそのことを DBMS に通知し，DBMS はその通知に基づいて解放されたページは処理に使わないようにするといった事後処理を行う．

バルーニングが起こると，OS はまずはページキャッシュやクリーンなページを解放する．それでもメモリが足りなければ，DBMS から通知されている情報を元に，DBMS が管理するメモリに割り当てられたページの中で解放可能なものを探す．解放可能なページを見つけると，OS はそのページを回収する．そして最後に回収したことを DBMS に通知し，DBMS 内部での整合性をとってもらう．

案 2 では，OS・DBMS 双方に連携用のインターフェースを追加する．また，OS 内に DBMS のメモリ情報を管理するためのデータ構造を追加する．OS 側のシステムコールでは，DBMS から DBMS が管理するキャッシュ領域の開始アドレスと長さ，解放の可否を受け取る．DBMS 側のインターフェースでは，OS からメモリの解放や確保の通知を受け取る．

DBMS は，通常の動作でページを用いていた処理部分で，OS が用意したシステムコールを利用してページの状態の変化を OS に通知する．ページが処理に使われ始めるときには解放不可であることを通知し，処理が終わり free リストに戻ってきた際には解放可能であることを OS に通知する．そのほか，OS によって解放されたメモリの情報を DBMS が OS から受け取ると，DBMS はそのページをキャッシュ用のリストから取り除き使用不可ページ用のリストに移動する．

4.2.2 DBMS 側の設計

DBMS 側には 1 つのインターフェースを追加する．インターフェースでは，OS によって解放または再確保された領域の先頭アドレスを受け取る．バルーニング時，OS が DBMS の管理している領域のメモリを解放または再確保すると，その領域の先頭仮想アドレスを OS がこのインターフェースを利用して DBMS に通知する．DBMS は仮想アドレスを受け取ると，そのアドレスをデータ領域として指しているページをキャッシュ領域から見つけ出し，キャッシュ用のページリストから取り除く．そして使用不可のページ用のリストに移動する．同様に，OS によりメモリが再確保され，その仮想アドレスを受け取った場合は，使用不可のページ用のリストから対応するページを見つけ出し，再び使用可能にする．

通常の処理において，DBMS はシステムコールを利用して，キャッシュとして利用しているメモリ領域の先頭アドレス，その領域の長さ，その領域は OS により任意に解放可能かどうか，を OS に通知する．キャッシュ領域として利用している範囲や解放可能かどうかは頻繁に変わるため，その都度システムコールを利用して OS に通知する．

4.2.3 OS 側の設計

1 つのシステムコールを追加する．システムコールを通じて，OS は，DBMS からキャッシュ領域の情報としてキャッシュ領域の先頭アドレス・その領域の長さ・解放の可否フラグを DBMS から受け取る．OS はそれらの情報を保存しておき，バルーニング時にその情報を利用して解放可能な領域を探し出す．

バルーニング時，OS はクリーンなページやページキャッシュが少なくなると，保存しておいた DBMS のキャッシュ領域の情報を元に解放可能である領域を探し出し，メモリを解放する．そして解放した領域の先頭アドレスを DBMS に通知し，DBMS 内部での整合性をとってもらう．もしも解放可能である領域が見つからなければ，OS は DBMS のインターフェースを利用してより多くの解放可能な領域を準備してもらう．そしてその領域を OS が解放する．

ただし，OS がメモリの解放を行なったあと，DBMS がその通知を受取るまえにその領域を使用しようとしてしまうとセグメンテーション違反を起こしてしまう．そこで，ロックの機構もシステムコール内に含める．OS 内で管理する DBMS のキャッシュ領域の情報を更新する前後ではロックを取ることでデータの一貫性を保つ．もしもシステムコールにより解放不可がセットされようとしたときに，すでにその領域が解放ずみであれば OS はエラーを返す．DBMS は処理に使うためのページを free リストから取り出す直前にシステムコールを用いて解放不可であることを通知しようとするため，もしもその際に OS からエラーが返ってきた場合，DBMS はそのページがすでに解放済みと

© 2014 Information Processing Society of Japan

判断でき，別のページを使用するようにできる．

4.3 案 3 : DBMS のデータ構造を OS が認識し，OS が DBMS のメモリを管理

4.3.1 概要

案 3 では，DBMS のデータ構造とメモリ内容を OS・DBMS 間で共有し，OS が DBMS のデータ構造やメモリ管理方法を認識した上で，OS が DBMS のメモリの解放・確保を行う．DBMS の扱うデータ構造や，メモリ管理方法をあらかじめ OS 側に登録しておく．バルーニング時，OS はページキャッシュやクリーンなページが少なくなると，DBMS の管理するページを解放する．OS は予め登録しておいた DBMS のデータ構造やメモリ管理方法を元に，DBMS にとって不要なメモリ領域を探し出し，その領域の解放を行う．また，十分な空きメモリがある場合，OS は DBMS に再びメモリを割り当てる．

DBMS が扱うデータ構造やメモリ管理方法を OS に登録できるようにするために，DBMS 個々のデータ構造・メモリ管理方法に基づいたメモリ管理処理を登録可能なハンドラを OS 内に定義する．OS はメモリ管理時にそれらハンドラに登録された DBMS 固有のメモリ管理処理を呼び出す．

さらに，DBMS は自身のバッファプールのメモリ領域を OS と共有するようにする．通常 DBMS 側のユーザー空間で確保されるメモリを，OS カーネル内で確保するようにする．DBMS はその領域をマッピングすることでゼロコピーでアクセス可能であり，一方のカーネルもカーネル内に確保されたメモリのためゼロコピーでアクセス可能である．メモリ領域そのものを共有することで，DBMS からも OS からもバッファプールのメモリ領域にゼロコピーでアクセスすることができる．ゼロコピーでアクセスできることにより，OS が DBMS のメモリ領域を管理することを容易にする．

案 3 では，DBMS 側の変更はほとんど必要としない．必要な変更は，DBMS がメモリを確保する際にバッファプールのメモリを明示的に OS と共有するようにするのみである．

4.3.2 DBMS 側の設計

メモリ確保方法の変更を行う．バッファプールのメモリを確保する際，明示的に OS カーネルとメモリを共有し，バッファプールの先頭仮想アドレスをカーネルに通知する．

バルーニング時，DBMS 側は通常通りの処理を続ける．メモリの解放や内部の整合性の維持は OS カーネルによって行われる．

4.3.3 OS 側の設計

DBMS 固有のメモリ解放や確保の処理を OS に登録できるようにし，OS はそれらの処理を自身のメモリ管理処理の中から適宜呼び出すようにする．OS 側で，DBMS 固有のメモリ解放や確保を行う処理のハンドラを定義する．そして，その定義に従って DBMS 固有の処理を定義し，そのハンドラに登録する．

定義するするハンドラの処理内容は，メモリの解放と確保である．まず，バルーニング前に DBMS 固有のメモリ管理方法に基づいたメモリの管理処理を定義し，そのハンドラに登録する．バルーニング時，OS はページキャッシュやクリーンなページがなくなると，ハンドラに登録されている DBMS 固有のメモリ解放処理を呼び出す．DBMS 固有のメモリ解放処理では，DBMS のデータ構造を認識した上でのメモリ解放が行われる．DBMS 固有のメモリ解放処理を呼び出せるようになることで，ページキャッシュ，DBMS のキャッシュ領域，その他無名ページなどのスワップアウトが必要なページ，の順でのページ解放が可能となる．

5. 実装

OS 側の実装は Linux 3.11.5 に，DBMS 側の実装は MySQL 5.6.14 に対して行なった．いずれの実装でも，OS から MySQL への通知には netlink ソケットを利用した．

6. 実験

6.1 実験環境

実験は，16GB のメモリ， Intel E3-1240v2(3.40 GHz，8MB キャシュ， 4 コア/8 スレッド) プロセッサ，500GB 7200RPM HDD のマシン上で行なった．Xen 4.2.1 を使用し，ホスト OS，ゲスト OS 共にカーネルは Linux 3.11.5 の 64bit 版を用いた．domU には 8 コアを割り当てた．実験 3 のみ，domU には 1 コアを割り当てた．

ベンチマークは SysBench[5] と TPC-H を使用した．SysBench は継続的に負荷をかけ続け，毎秒のスループットを算出可能なものである．TPC-H は 1 つの高負荷のクエリを実行するオンライントランザクション処理を模したベンチマークである．DBMS には MySQL 5.6.14 を用いた．

6.2 実験方法と目的

バルーニング後でも MySQL が本来の性能を発揮することを確認するため，様々なクエリでも提案手法が有効であることと MySQL・Linux への変更によるオーバーヘッドを確認するため，そして各案でのバルーニングにかかる時間の比較とその内訳を調べるため 3 つの実験を行った．

6.2.1 実験 1:ワークロードサイズの変化とバルーニング

OS によるバルーニングが行われた後でも，OS と DBMS の連動したメモリ管理により DBMS が本来の性能を発揮することを確認するため，実験を行った．SysBench を実行し続け，途中でワークロードサイズの変更とそれに伴うバルーニングを行なった際の毎秒のスループットの変化を計測した．

まず，VM のメモリ量を 10GB，MySQL のバッファプールサイズを 8GB として VM と MySQL を起動し，ワークロードサイズ 8GB で SysBench を実行した．3000 秒待機し，十分に MySQL のバッファプールのキャッシュを温めた．次に，ワークロードサイズを 5GB に変更し，再び Sysbench を実行した．500 秒後，バルーニングにより VM のメモリサイズを 5GB に減らした．そして 2000 秒待機した．その後，再びワークロードサイズを 8GB に戻し，VM のメモリサイズもバルーニングにより 10GB に戻した．そして 2000 秒待機した．この 7500 秒間の毎秒のスループットを計測した．

SysBench のアクセスの分布は均一分布，スレッド数は 16，アクセス方法は読み込みのみとした．

6.2.2 実験 2:バルーニング前後の性能の変化とオーバーヘッド

実験 2 は異なるワークロードにおいても提案手法によりバルーニング後に本来の性能が発揮できることを確認するためのものである．バルーニングによってメモリサイズを変更した場合と，元からそのメモリサイズで VM と MySQL を起動していた場合での処理時間の比較を行った．

6.2.2.1 バルーニングによってメモリサイズを変更した場合

まず，VM のメモリを 10GB，MySQL のバッファプールを 9GB として VM を起動した．TPC-H の同一のクエリを 3 回実行し十分にキャッシュを温めた．その後，バルーニングにより VM のメモリを 8GB または 4GB に変更し，再度同一のクエリを実行し，処理時間を計測した．

6.2.2.2 元から同一のメモリサイズで起動した場合

VM のメモリサイズを 4GB または 8GB に，MySQL のバッファプールの大きさを 2.8GB または 5.6GB にした状態で VM を起動し，TPC-H の同一のクエリを 3 回実行して十分にキャッシュを温めたのち，再度同じクエリを実行して処理時間を計測した．バッファプールの大きさは，MySQL のドキュメントの指示どおり OS のメモリサイズの 70%の値としている．

6.2.3 実験 3:バルーニングに要する時間とその内訳

実験 3 は各アプローチによるバルーニング時間の差と，その内訳を調べるためのものである．メモリを 10GB から 5GB にバルーニングした際の，バルーニングに要した時間を計測した．実験 3 のみ，時間を測りやすくするため domU には 1 コアを割り当てた．まず，実験 1 と同様の設定で SysBench を実行し，バッファプールのキャッシュを温めた．次に，バルーニングにより VM のメモリサイズを 10GB から 5GB に変更した．その際の，バルーニングに要した時間とその内訳を計測した．バルーニングにかかる時間はゲストカーネルのバルーンドライバ内にて，OS のブートからの実時間をナノ秒精度で返す get_monotonic_boottime 関数を用いて計測した．時間の内訳は，各手法のコード中にて get_monotonic_boottime を用いて同様に計測した．

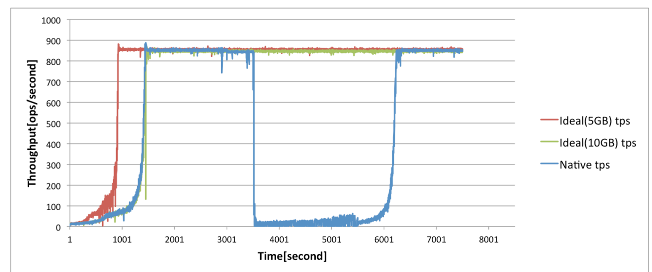

図 6 ワークロードサイズの変更後にバルーニングを行なった場合の本来のスループットの変化

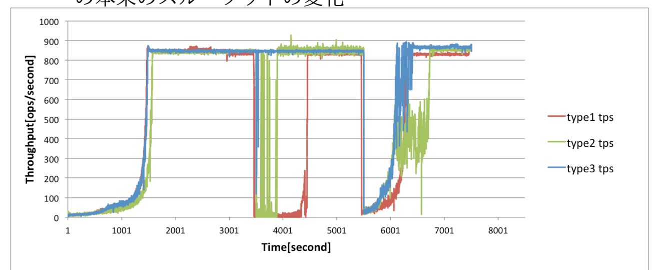

図 7 ワークロードサイズの変更後にバルーニングを行なった場合のスループットの変化

6.3 実験結果と考察

6.3.1 実験 1:ワークロードサイズの変化とバルーニング

図 7 は SysBench 実行開始から 3000 秒後にワークロードサイズを 8GB から 4GB に変更し，その 500 秒後に VM のメモリサイズを 10GB から 5GB に，その 2000 秒後に再びワークロードサイズを 8GB に，VM のメモリサイズを 10GB に変更した際の，スループットの推移である．Ideall(10GB) は VM のメモリサイズを 10GB にし，ワークロードサイズを 8GB のままでバルーニングを行わなかった場合の本来の性能の目安である．同様に Ideal(5GB) は VM のメモリサイズを 5GB，ワークロードサイズを 4GB にした場合の本来の性能である．Native は提案手法なし，Type1～3 は提案手法の案 1～3 に対応する．

提案手法なしの場合，バルーニング開始直後から性能が著しく低下し，その後もバルーニング中は性能が回復していない．これは多量のスワップアウトが行われたためである．ワークロードサイズの変更により DBMS が必要とするメモリ量が減ったにも関わらず，OS がそれを認識できずスワップアウトしてしまっている．

提案手法ありの場合，いずれの案でもバルーニング後，最終的に本来の性能に近い性能を発揮している．ワークロードサイズの変更に伴って生じた DBMS 内の不要なメモリを OS がスワップアウトなしで回収出来たためである．

案 1 がバルーニング直後に性能が低下しているのは，500MB ほどスワップアウトが発生していたためである．案 1 の場合，MySQL が管理するメモリを解放するめには一度 MySQL にコンテキストスイッチする必要がある．MySQL

が別の処理を行っている場合，即座にメモリの解放を行うことができない．今回の実装では OS と MySQL が同期をとりつつメモリの解放を行うわけではないため，MySQL の解放が間に合わない場合，OS は MySQL のメモリをスワップアウトする．

一方で案 2 と案 3 の場合，カーネルがメモリを管理するため，カーネルの都合により即座にメモリの解放が可能である．

提案手法を適用した場合，バルーニング後も本来に近い性能を発揮することが確認された．提案手法により OS と DBMS とのメモリ管理を連動させることで，VMM のメモリ管理と DBMS のメモリ管理の競合を回避できていると言える．

6.3.2 実験 2:バルーニング前後の性能の変化とオーバーヘッド

図 8 と図 9 は予めメモリサイズを変更しておいた場合と，バルーニングによって後から変更した場合での TPC-H の結果である．実行に 30 分以上掛かるクエリは除外した．図 8 の 4GB へのバルーニングはバッファプール上にデータが乗りきらない場合を，図 9 の 8GB へのバルーニングはバッファプール上にデータが乗りきる場合を想定している．クエリ 20 を除き，いずれの場合も提案手法の性能は本来のものとほぼ同じとなった．

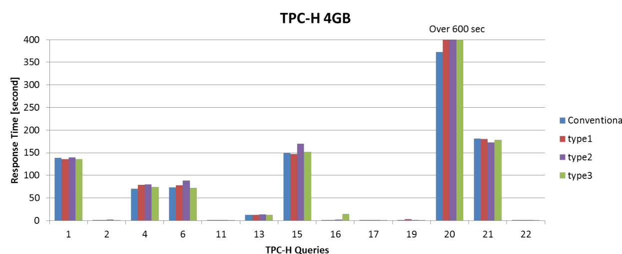

図 8　10GB から 4GB にバルーニングした際の TPC-H の結果

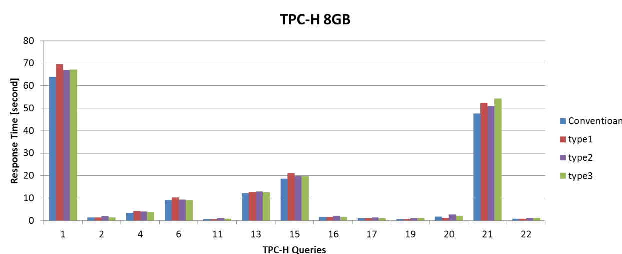

図 9　10GB から 8GB にバルーニングした際の TPC-H の結果

この実験の結果より，バッファプール上にデータが乗り切るかどうかや，クエリの種類によらず提案手法が有効であると言える．

6.3.3 実験 3:バルーニングに要する時間とその内訳

図 10 はバルーニングにかかった時間とその内訳である．type1〜3 が案 1〜3 に対応する．

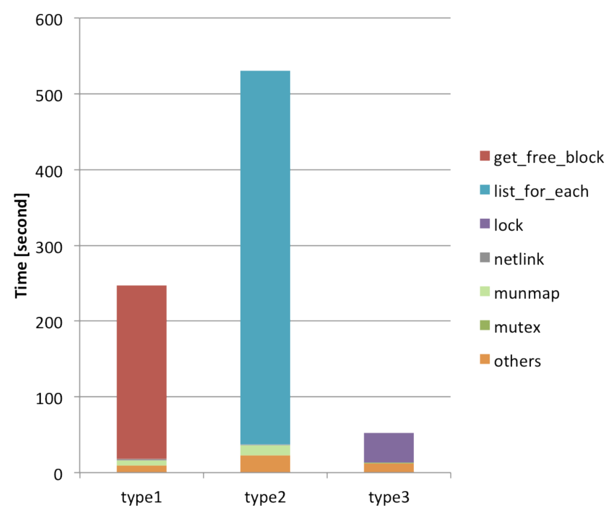

図 10　バルーニングに要した時間とその内訳

案 1 では，LRU リストからの解放可能な要素の探索に最も時間を要している．この処理では，lru リストの要素を一度 free リストに戻す際にメモリ上のデータ領域への書き込みを行っている．その際に，そのページがスワップアウトされている場合はスワップインの処理が必要となる．そのため，時間を要したと考えられる．案 1 ではバルーンドライバによる処理は 7 秒程度で終わっているが，ユーザー空間での MySQL での処理にはおよそ 245 秒かかった．バルーンドライバの処理が終わったということは何らかの方法で実メモリが回収されたということであり，MySQL の解放が間に合わなかった場合はスワップアウトが発生している．

案 2 では，単純にリストの線形探索に時間を要している．これは十分に改善の余地がある．

案 3 では内部処理での排他処理のためのロック取得待ちに最も時間を要している．同じく実装上の問題であり，十分に改善の余地がある．

また，実装上必要不可欠である netlink による通知と munmap によるメモリ解放にかかる時間の合計は，いずれの場合も 15 秒未満であった．

3 種類の案で比較した場合，案 3 が最も速くバルーニングを終えている．案 3 では，MySQL のメモリ管理ポリシーを OS が認識しているうえ，MySQL と OS でメモリ領域を共有しているため，整合性の維持などを除いたメモリ解放部分はカーネル空間内で完結する．一方で，案 1 の場合は DBMS 側にメモリの解放を依頼するため，一度 DBMS にプロセススケジューリングされる必要がある．さらに，今の実装では OS が MySQL のメモリ解放を待つことはしていないため，必ず DBMS の解放が間に合うとは限らず，スワップアウトが発生するおそれがある．

この実験の結果より，いずれの案もまだ十分に改善の余地があると言える．また，必要不可欠な処理に要する時間はいずれも 15 秒未満であり，改善を行えば提案手法は現

© 2014 Information Processing Society of Japan

実的であると言える．

7. 関連研究

Application-Level Ballooning[6] はアプリケーション自身にバルーニングを行わせ，バルーンドライバと連携することで自身でメモリを管理するアプリケーションが仮想化環境上で十分な性能を発揮しない問題に対処している．しかしこれは OS のメモリ管理とは連動していない．バルーンドライバと DBMS の連携にとどまっているため，バルーニング処理のときにのみ VMM と DBMS が協調して動作する．一方本研究で提案する手法では，OS と DBMS のメモリ管理を連動させることで，VMM と DBMS の協調した動作を可能にする．VMM に頼らず OS と DBMS が連動することで，VMM，OS，DBMS の 3 レイヤでのより柔軟な協調したメモリ管理が可能となる．また，DBMS と OS のメモリ管理ポリシーについては特に考慮がされていない．本研究では，DBMS のメモリ管理ポリシーと OS のメモリ管理ポリシーを活かした上での統合を行い，DBMS 本来の性能が得られるようにする．

COD では，特に CPU の資源管理に着目し，DBMS もマルチコア環境用の CSCS を用いている．COD ではメモリに着目していないが，本研究ではメモリ資源に着目する．また，COD と同様に OS と DBMS 両サイドの資源管理ポリシーを用いて資源管理を行う．

Kairos は DBMS 自体を統合するため，セキュリティ上統合できる DBMS は同じ所有者のものに限られる．一方で，本研究では，OS と DBMS のメモリ管理を連動させるため，VM ベースで統合することができる．また，本研究では行なっていないが，Kairos のワーキングセット見積もりモデルを本研究に適応することで，VM ベースでの効率的な統合が可能になると考えられる．

CRAMM では，OS とアプリケーションの資源管理の競合に着目している．また，自身で資源を管理するアプリケーションとして，ガベージコレクションをもつ言語ランタイムに着目している．本研究では自身でメモリを管理するアプリケーションとして DBMS を選択したが，ガベージコレクションを持つ言語ランタイムにも同様の手法が適応可能だと考えられる．

8. 結言

8.1 まとめ

本研究では，VMM のメモリ管理との競合を回避する DBMS のメモリ管理手法を提案した．DBMS と VMM のメモリ管理との競合を回避するために，DBMS と OS のメモリ管理を連動させるアプローチを取った．そのための方法として 3 つの実現案を提案した．

8.2 今後の課題

今後の課題として，バルーニングに要する時間の短縮が挙げられる．これは実装上の問題であるため，十分に解決可能である．そのほか，Phase-based Reboot[7] での実験のように，バルーニング以外でのメモリ管理ポリシー競合の実験を行うことが挙げられる．Phase-based Reboot は OS のバッファキャッシュを破棄することでスナップショットの復元を行うものであるが，DBMS が大量のメモリを使用した場合，DBMS にとってはキャッシュで破棄可能なものであっても，OS はそれをキャッシュとして認識しないため，スナップショット復元の性能が劣化すると考えられる．

参考文献

[1] Waldspurger, C. A.: Memory Resource Management in VMware ESX Server, *SIGOPS Oper. Syst. Rev.*, Vol. 36, No. SI, pp. 181–194 (online), DOI: 10.1145/844128.844146 (2002).

[2] Council, T. P. P.: TPC-C, Transaction Processing Performance Council (online), available from ⟨http://www.tpc.org/tpcc/⟩ (accessed 2013-08-13).

[3] Council, T. P. P.: TPC-E, Transaction Processing Performance Council (online), available from ⟨http://www.tpc.org/tpce/⟩ (accessed 2013-08-13).

[4] Council, T. P. P.: TPC-H, Transaction Processing Performance Council (online), available from ⟨http://www.tpc.org/tpch/⟩ (accessed 2013-08-13).

[5] Kopytov, A.: Sysbench, MySQL AB (online), available from ⟨http://sysbench.sourceforge.net/⟩ (accessed 2013-12-09).

[6] Salomie, T.-I., Alonso, G., Roscoe, T. and Elphinstone, K.: Application Level Ballooning for Efficient Server Consolidation, *Proceedings of the 8th ACM European Conference on Computer Systems*, EuroSys ’13, New York, NY, USA, ACM, pp. 337–350 (online), DOI: 10.1145/2465351.2465384 (2013).

[7] Yamakita, K., Yamada, H. and Kono, K.: Phase-based Reboot: Reusing Operating System Execution Phases for Cheap Reboot-based Recovery, *Proceedings of the 41st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN ’11)*, pp. 169–180 (2011).

© 2014 Information Processing Society of Japan