Vol.2015-OS-132 No.8
2015/2/26

コンテナ型仮想化の機能を応用したページング方式

中川 岳[1,a)]　追川修一[2]

概要：コンテナ型仮想化は，コンテナと呼ばれるプロセスグループごとに独立した実行環境を提供する，オペレーティングシステム（OS）レベルの仮想化である．それぞれのプロセスはコンテナと呼ばれるグループに分類され，異なるコンテナに属するプロセスは互いに影響を受けない．互いに関連のあるプロセスを同じコンテナに所属させ，他のプロセスから隔離することで，ホスト OS 上で複数の独立したサービスを提供することができる．このコンテナ型仮想化では，関連があるプロセスが同じコンテナに所属するため，通常のプロセス実行よりも詳細にプロセス間の関連性を把握することができる．この情報を利用することで，あるプロセスが実行待機状態になった場合に，関連するプロセスがページアウトされているならば，スワップ領域からの先読みが可能である．つまり，コンテナの情報を利用することでページングのコストを削減することが可能である．本発表では，コンテナ型仮想化環境で得られる情報を応用したページング方式について議論する．

A Paging Method utilizing Container-Based Virtualization

1. はじめに

コンテナ型仮想化は単一のホストオペレーティングシステム（OS）上で，ユーザープロセスがそれぞれ独立した環境で動作する OS レベルの仮想化である [1, 2]．それぞれのユーザプロセスはコンテナと呼ばれるプロセスグループに分類される．コンテナ毎にプロセス ID 空間，ファイルシステム空間などは独立しており，異なるコンテナに属するプロセスは互いに影響を受けない．また使用可能な主記憶の大きさや，プロセッサ時間を制限することも可能である．実用的な OS への実装としては，FreeBSD Jail, Solaris Container, OpenVZ, LXC などがある．このコンテナ型仮想化は Web サービスの提供基盤として，急速に利用が拡大している．

単一のホスト OS で，複数のコンテナを動作させる場合には，それぞれのコンテナが使用可能な最大主記憶サイズを，実メモリより多く割り当てることが行われる．これをオーバーコミットと呼ぶ．例えば，全てのコンテナに対して，最大で 1GB の主記憶が割り当て可能なホスト OS があるとする．この状態で，最大主記憶サイズを 256MB にしたコンテナを 5 個ホストすると，コンテナ群に対する主記憶の割り当ては最大で 1.25GB となる．この状態がオーバーコミットである．それぞれのコンテナの主記憶使用量が最大主記憶サイズよりも十分に小さければ，実際に使用可能な物理メモリを超過することはない．もし超過した場合には，通常の OS でのプロセスと同様に，コンテナに所属するプロセスもページアウトの対象となる．

これまでのコンテナ型仮想化環境におけるページングでは，プロセスがどのコンテナに所属するか考慮されていなかった．しかしながら，プロセスのコンテナへの所属情報を利用することで，ページングのコストを削減することが可能である．あるプロセスが実行待機状態になったとき，同じコンテナに含まれる他のプロセスが続いて実行される可能性は高い．例えば，1 つのコンテナで，ある Web サービスを提供する HTTP サーバとデータベースサーバ（DB サーバ）がホストされているとする．HTTP サーバと DB サーバはそれぞれ待機状態で，ページアウトされているとする．この状況で外部からのリクエストにより，待機中だった HTTP サーバプロセスが実行された場合には，続いて DB サーバへのリクエストが発生し，DB サーバも実行される可能性は高い．通常ならば，HTTP サーバ，

[1] 筑波大学大学院 システム情報工学研究科 コンピュータサイエンス専攻
Department of Computer Science, University of Tsukuba, Ibaraki, 305-8573, Japan
[2] 筑波大学 システム情報系情報工学域
University of Tsukuba, Ibaraki, 305-8573, Japan
[a)] gnakagaw@cs.tsukuba.ac.jp

DB サーバそれぞれにページイン処理が発生する．もし，HTTP サーバをページインする時点で，DB サーバが含まれるページもページインすれば，それぞれに必要であったページインのコストを削減することが可能である．この例のように，あるプロセスについてページインする際に，同じコンテナに属する他プロセスのページを先読みしておくことで，ページングのコストを下げることが可能である．

以上を背景として，本発表ではコンテナ型仮想化環境における，コンテナによって示されるプロセスの関係を考慮したページング方式について議論する．コンテナ型仮想化環境としては，Linux 上で動作する LXC を対象とする．

2. 研究の背景

この節では研究の背景にあるコンテナ型仮想化，コンテナ型仮想の基盤の 1 つである Linux Container，コンテナ型仮想化におけるメモリオーバーコミットについて述べる．

2.1 コンテナ型仮想化

コンテナ型仮想化は，コンテナと呼ばれるプロセスグループごとに独立した実行環境を提供する，オペレーティングシステム（OS）レベルの仮想化である．それぞれのプロセスはコンテナと呼ばれるグループに分類され，異なるコンテナに属するプロセスは互いに影響を受けない．互いに関連のあるプロセスを同じコンテナに所属させ，他のプロセスから隔離することで，ホスト OS 上で複数の独立したサービスを提供することができる．

1 つの物理ハードウェア上で複数のサービスを隔離して提供する手段としては，ハイパーバイザを用いたハードウェア仮想化がある．コンテナ型仮想化は，よりコストを小さくサービスの隔離を実現できる．ハードウェア仮想化では，隔離する単位ごとに OS を動作させる必要がある．これに対してコンテナ型仮想化では，その必要はあない．全てのコンテナは，単一の OS カーネルを共有する．そのため，ハードウェア仮想化による隔離よりも，仮想化のオーバーヘッドを小さく抑えることができる．

2.2 Linux Container

Linux Container (LXC) とは，Linux の機能を用いたコンテナ仮想化機構である．Linux カーネルが備えている機構を利用して，プロセス ID 空間．プロセス間通信空間，ファイルシステム空間，ネットワーク空間などをコンテナごとに隔離する．

この隔離は，主に 2 つの Linux カーネルの機能を利用して実現されている．1 つは Name Space，もう一つは Controle Groups（cgroups）である．Name Space は，ユーザ ID やプロセス ID といった，システム上でユニークな管理情報を複数の異なるオブジェクトに対して重複して使うことを可能にする機能である．コンテナ型仮想化の場合はこの機能により，プロセス ID 空間，プロセス間通信のディスクリプタなどが分離されている．

cgroups は，プロセスグループごとに CPU 時間，メモリ，ディスクの使用量といったリソース使用を制限する機能である．cgroups では，すべてのプロセスはいずれかのプロセスグループに所属し，そのプロセスに設定されているリソース制約を受ける．LXC ではこの機能をコンテナごとのリソース制限のために用いている．

2.3 メモリオーバーコミット

コンテナ型仮想化におけるメモリオーバーコミットとは，システムに実際に搭載されているメモリ容量より多くのメモリ領域の確保をコンテナに許すことである．理想的には，コンテナが利用可能な最大メモリサイズは，物理メモリの範囲内で決定されるのが望ましい．しかしながら，プロセスは常にその最大メモリ要求をするわけではない．メモリの要求量はプロセスの処理内容に応じてプロセスの実行を通して変化する．そのためコンテナ内のプロセスが要求するメモリサイズが小さい場合には，物理メモリの利用が非行率になる．そこで，オーバーコミットをすることで，効率的なメモリ利用が可能になり，1 つのシステムで管理できるコンテナの数を増やすことができる．

オーバーコミットが行われているシステムでは，メモリの使用状況に応じて，2 次記憶へのページングが行われる．プロセスの実行状況によっては，プロセスからのメモリ要求が物理メモリのサイズに逼迫，あるいは超過することがある．この場合には，使用していないページを 2 次記憶に移すことで，空き物理メモリを確保する．この処理をページアウト処理と呼ぶ．ページアウトされたメモリ内容にアクセスが起こったときには，反対に 2 次記憶からメモリへのページ内容の移動が行われる．この処理をページイン処理と呼ぶ．

このページング処理には，大きなコストがかかる，ページイン，ページアウト処理には 2 次記憶への読み書きが伴う．2 次記憶は一般的にメモリへのアクセスに比べて低速である．そのため，これらの処理が起こると，データへのアクセスは低速になる．また，ページアウトされたメモリ空間へアクセスが発生した場合は，ページフォールト例外が発生し，ページイン処理を行う．これにも大きなコストがかかる．

3. コンテナ仮想化の機能を利用したページングの最適化

プロセスの中には，協調して動作するものがある．例えば，あるシステム上で，HTTP サーバのプロセスと，データベースサーバ（DB サーバ）のプロセスが動作している状況を考える．HTTP サーバでは，DB サーバに格納された情報を利用する Web サービスが提供されているとする．

ユーザからのリクエストに基づいて，HTTP サーバは DB サーバに問い合わせを行い，サービスを提供する．このように複数のプロセスを組み合わせて 1 つのサービスを提供する機会は多い．本論文では，このように協調して動作するプロセスを互いに，関連性のあるプロセス，反対に協調して動作しないプロセスを関連性のないプロセスと表す．

コンテナ型仮想化環境においては，サービスや実行環境ごとにプロセスを独立したコンテナに格納する．コンテナ型仮想化環境においては，それぞれのコンテナには互いに関連性の高いプロセスが格納される可能性が高い．例えば，前述した HTTP サーバと DB サーバの場合は 2 つのプロセスは同じコンテナに格納され，Web サービスの提供に不要なプロセス，つまり 2 つのプロセスと関連性のないプロセスはそのコンテナには格納されない．つまりコンテナへの格納状況を調べることで，プロセス間に関連性があるのか，ないのかを推測することができる．

このコンテナから得られる情報を利用することで，メモリオーバーコミットされたコンテナ型仮想化環境でのページングの最適化が可能である．具体的には，コンテナへの格納状況に基づいた，ページインの先読みが可能になる．同じコンテナに格納されたプロセス同士は関連があり，連続して実行される可能性が高い．そのため，あるプロセスのメモリ空間についてページフォルトが発生し，ページインが発生する際に，同じコンテナに含まれている他のプロセスがページアウトされているかを調べ，もしそうならば，先読みしておくことでページフォルト例外の発生を防ぐことができる．

4. 予備実験

第 3 節では，プロセスの関連性に基づいたページインの先読みによるページング処理の最適化について議論した．本節ではその可能性を議論するための予備実験について述べる．この実験では，ある 1 つのサービスを提供するための，関連性のあるプロセス群のページング処理を分析する．この分析を通して，第 3 節で言及した，プロセスの関連性と，ページング処理の対応関係について議論と考察を行う．本実験の段階では，コンテナ型仮想化は利用していない．しかしながら，コンテナ型仮想化環境で動作するプロセスは OS カーネルの層からは，通常のプロセスと同等と見なすことができ，この実験を通して得られた知見は，本論文が主題とするコンテナ型仮想化環境の応用にも適用可能である．

4.1 実験環境と実験内容

この実験は，qemu を用いて構築した仮想マシンで動作する OS 環境を基盤とする．この実験で用いたホスト計算機環境を表 1 に，仮想マシン環境を表 2 に示す．

この OS 環境では，2 つの独立したサービス A, B が提

表 1 ホスト計算機環境

プロセッサ	Intel Core i7-4960X
主記憶容量	32GB
ホスト OS	Linux 3.12.21
仮想化ソフトウェア	qemu 2.1.2

表 2 仮想マシン環境

プロセッサ	Intel Xeon E312 Series
主記憶容量	1GB
ルート領域容量	10GB
スワップサイズ	1GB
ゲスト OS	Linux 3.14.0

表 3 サービス A を構成するソフトウェア群

httpd	Apache 2.4.6
スクリプト言語処理系	ruby 2.0.0 p353
RDBMS	MariaDB

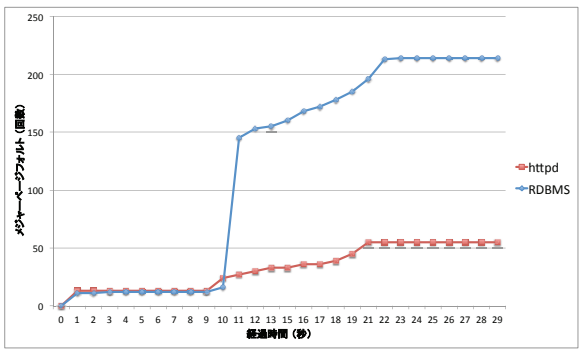

図 1 メジャーページフォルトの回数

供されているとする．サービス A は HTTP サーバソフトウェア（httpd）とリレーショナルデータベース管理システムソフトウェア（RDBMS）から構成される Web サービスである．サービス B は任意のサービスとする．サービス A，B それぞれは，主記憶を最大で 819MB 利用できるように設計されているものとする．これは，512M Bを平均的に使用するサイズとし，その 1.6 倍までの割り当てを許容する，という設計意図である．仮想マシン環境では 1GB の主記憶が割り当てられているので，この環境では最大で 614MB のオーバーコミットを行うことを想定している．このようなシステム上で，サービス B のメモリ要求が高まったときのサービス A についてのページングの状況を観測，分析する．

サービス A は，httpd，RDBMS，両者のインタフェースとなるスクリプト言語処理系，の 3 つの要素で構成されている Web システムである．ユーザは httpd 経由でスクリプト言語によって実装されたインタフェースにアクセスし，RDBMS にクエリを要求し．検索結果を受け取る．サービス A を構成するソフトウェアについての詳細を表 3 に示す．RDBMS に登録するデータセットとしては，Sample database with test suite [3] のバージョン 1.0.6 を用いた．RDBMS のメモリ利用の傾向は，パラメータの設定によって大きく変化する．本実験で使用した RDBMS には様々なパラメータを持つが，本実験において特に重要であるのは，innodb_buffer_pool_size である．これは RDBMS がストレージ上の情報をどの程度メモリにバッファリングさせておくか指定するものである．本実験では，655M とした．これは，サービス A の最大使用量である 819MB の 8 割を割り当てることを意図している．

サービス A は企業の社員名簿を検索するシステムである．RDBMS には従業員の氏名，役職，給与，部署などの情報を含んだテーブルが含まれている．この RDBMS が持つ情報には，ruby で実装されたインタフェースを介してアクセスする．このインタフェースプログラムを query.cgi とする．query.cgi は httpd から呼び出し可能なプログラムとする．今回の実験では．以下のようにサービス A が提供される．Step3. で要求するクエリは別々のテーブルに保存されている社員の個人名と役職のテーブル結合し，その中から，役職が Engineer であるものを抽出するものである．

Step1. httpd 経由で query.cgi に対する実行要求が起きる．
Step2. query.cgi は RDMBS へ接続する．
Step3. SQL クエリを RDBMS に要求する．
Step4. 受け取った結果をユーザへ送信する

サービス B については，サービス B の最大割り当てサイズ（819MB）を使用する任意のワークロードとする．この実験では，以下の動作をするプログラム α を作成して，サービス B によるメモリ要求の圧迫を再現した．

Step1. サービス B の最大割り当てサイズ（819MB）のメモリを確保する．
Step2. 確保した領域の先頭から，システムのページサイズごとにダミーデータの書き込みを行う．
Step3. 確保したメモリ領域をそのまま保持する．

Linux のページング方式はデマンドページングであり，メモリの割り当て要求をした時点では実メモリの割り当ては行われない．確保したページに対して書き込みが発生して，実際のメモリ確保が行われる．Step 2. でページサイズごとに書き込みを行うのは，実際にメモリを確保することを強制するためである．今回の実験では，ページサイズは 4096KB であるので，確保した領域の先頭から，4096KB ごとに 4byte のダミーデータを書き込んだ．

この実験では，サービス A を構成する httpd, RDBMS についてページフォルトの回数を一定周期ごとに計測した．ページフォルトには，ブロックデバイスへの I/O が伴うメジャーページフォルト，ブロックデバイスへの I/O が伴わないマイナーページフォルトがある．今回の実験では，性能上の問題となる，メジャーページフォルトの回数を対象に実験を行う．Linux カーネルでは，プロセスごとのメジャーページフォルトの回数を proc ファイルシステムを経由することで取得可能である（/proc/[プロセス番号]/stats にページフォルトに関する情報が含まれる）．今回の実験では，この値を 1 秒ごとに記録して，メジャーページフォルトの回数の変化を記録した．計測を開始時点では，既にページフォルトが起こっており，得られる回数は 0 ではない．そのため，1 番目に記録した値を取得値から減ずることによって正規化した．

4.2 実験結果

前項で述べた実験環境において，以下の実験手順を実行し，データを収集した．

Step 1. サービス A の httpd を起動する
Step 2. サービス A の RDBMS を起動する
Step 3. 5 秒待機する
Step 4. プログラム α を起動する
Step 5. 1 秒待機する
Step 6. サービス A の httpd, RDBMS について，ページフォルト状況の記録を開始する
Step 7. 10 秒待機する
Step 8. httpd に，query.cgi に対する要求を行う
Step 9. query.cgi が実行され，結果が送信される
Step 10. 10 秒待機する
Step 11. 計測を停止し，実験を終了する

実験結果を 1 に示す．このグラフはサービス A の httpd, RDBMS についてメジャーページフォルトの回数を時系列でプロットしたものである．赤の線が httpd，青の線が RDBMS についての回数である．縦軸がメジャーページ

フォルトの回数，横軸が経過時間を示す．今回の実験ではメジャーページフォルトの回数を計測し，ページフォルトに伴うブロックデバイスへの入出力の量は計測していない．

5. 議論

この節では 4 節で述べた実験とその結果についての議論を行う．実験結果からは，計測を開始して，10 秒後にページフォルト回数について変化が起こっていることがわかる．これは，サービス A にリクエストが送信されるタイミングと一致している．httpd についてのデータに着目すると，計測開始後の 9 から 10 秒の間に，メジャーページフォルトが起こっている．これは，実験手順 Step. 8 のリクエストにより，httpd が実行状態に移行し，ページアウトされていた httpd のデータについてのページフォルトが起こったことが理由であると考えられる．RDBMS についてのデータに着目すると，計測開始後の 10 から 11 秒の間に，メジャーページフォルトの回数が急激に伸びている．これは，httpd によるリクエストにより RDBMS の実行がアクティブになり，ページアウトされていた RDBMS のデータについてのページフォルトが起こったことが原因であると考えられる．

実験結果より，httpd のページインの直後にそれから呼び出される RDBMS についても多量のページインが起こっていることがわかった．仮に，httpd がページインするタイミングで，RDBMS についてのページインを先行して実行できるとすると，RDBMS についてのページフォルトを大きく減らすことが可能である．もちろん，ページの先読みを行えば，それだけブロックデバイスに対する入出力のコストが発生する．しかしながら，ページフォルトのコストが削減できることを考慮すると，この先読みは有用である可能性がある．ページフォルトが起こると，ユーザモードからカーネルモードへのコンテキストスイッチが発生する．このコストは一般的に大きい．これが削減できることは大きなメリットである．

この実験では，関連性のある 2 つのプロセスについて，その関連性について着目したページングの先読みが有効である可能性を示した．このプロセスの関連性については，システムの設計者が明示的に指定することも可能であるが，第 3 節で示したように，コンテナ仮想化の機能を利用することで，汎用的に利用可能である．

6. まとめ

本論文では，コンテナ型仮想化の機能を利用したページングの最適化についての提案と議論を行った．提案手法はコンテナ仮想化により得られる，プロセスの関連性を利用することで，ページインの先読み効果を高めるものである．この手法の効果の議論のために，関連性のあるプロセスについてのページング処理を分析する予備実験を行った．実験の結果，プロセスの関連性を利用することで，ページインの先読み効果を向上できる可能性が明らかになった．

本論文では，提案手法の根幹的なアイデアであるプロセスの関連性とページイン処理についての関連性を示した．しかしながら，提案手法の有効性を検討するためには，さまざまな課題がある．その中でも重要なものの 1 つとしては，今回行った予備実験をコンテナ型仮想化環境で追試することが挙げられる．

参考文献

[1] Banga, G., Druschel, P. and Mogul, J. C.: Resource Containers: A New Facility for Resource Management in Server Systems, *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, Berkeley, CA, USA, USENIX Association, pp. 45–58 (online), available from ⟨http://dl.acm.org/citation.cfm?id=296806.296810⟩ (1999).

[2] Soltesz, S., Pötzl, H., Fiuczynski, M. E., Bavier, A. and Peterson, L.: Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors, *SIGOPS Oper. Syst. Rev.*, Vol. 41, No. 3, pp. 275–287 (online), DOI: 10.1145/1272998.1273025 (2007).

[3] : Sample database with test suite, `https://launchpad.net/test-db/`.