Vol.2015-OS-135 No.7
Vol.2015-EMB-39 No.7
2015/11/24

ページ共有率に基づく仮想 CPU スケジューリングによるキャッシュミス削減手法

伊藤 俊輝[1,a)]　河野 健二[1]

概要：近年，仮想化環境によるマシンの集約が広く行われている．仮想マシン上で様々なサービスを動作させることにより物理マシンを集約し，台数を減らすことができる．それぞれのサービスで利用するミドルウェアや OS が同一の場合，ページシェアリングによって効率的にメモリを利用することができる．しかし，共有しているデータを別のプロセッサで読み出す場合，双方にキャッシュされるため，キャッシュを圧迫し，キャッシュヒット率を低下させるという問題が発生する．本研究では仮想 CPU のスケージューリングの際，より多くのメモリを共有している仮想マシン同士を同じプロセッサへ割り当てることにより CPU キャッシュのヒット率を向上させる手法を提案する．本手法の有効性を示すため，提案手法により導出したスケジューリングを実施した場合の web サーバーに関するベンチマークを行い，提案手法に反するスケジューリングの場合と比較した所，処理時間が 15%高速化することを確認した．

キーワード：仮想化，スケジューリング，ページシェアリング，Xen

1. はじめに

近年，仮想化技術は様々なサービスの基盤として広く用いられるようになっている．これは仮想化環境を利用した Platform as a Servie (PaaS) 環境が多く提供されるようになったことに一因がある．

仮想化とは一つの物理マシン上に複数の仮想マシン (VM) を作成することで，本来必要な物理マシンの台数を削減する技術である．PaaS 環境ではこの VM を各ユーザーに独占的に提供するサービスであり，ユーザーはあたかも専用の物理マシンを利用しているかのように VM を利用し，サービスを構築することができる．VM 上で動作する OS やミドルウェアはユーザーが独自に決定できるものではあるが，一般に利用されているものの種類は限られており，同じものが動作している可能性が高い．そこで，重複し冗長となっているメモリを排除する手法としてページシェアリングがある．ページシェアリングは VM のメモリを探索し，内容の同じメモリを検出し共有することによって冗長なメモリを削減する手法である．これによってメモリの利用効率を上げ，VM の集約率を向上させることができる．

マルチプロセッサ環境ではそれぞれのプロセッサが CPU キャッシュを持つ．キャッシュはそれぞれのプロセッサ個別に管理されているため，あるページを参照したプロセスが別のプロセッサから呼ばれ，同一ページを参照する場合，データは新たにキャッシュされることとなり，読み取りに時間がかかる．このため，CPU スケジューリングはアフィニティを考慮して行われる．アフィニティとはプロセスを割り当てるべき CPU の設定情報であり，プロセスを実行した CPU をアフィニティとして保持し，次にスケジューリングされる際に該当 CPU へ優先的に割り当てるためのものである．アフィニティによりプロセスを同じ CPU に割り当てられるように設定することで，切り替えのたびにデータをキャッシュし直すという動作を減らすことことができる．

仮想化環境では VCPU スケジューリングにより仮想 CPU（VCPU）に物理 CPU（PCPU）を割り当てているが，この際もアフィニティーを考慮した割当が行われている．この場合のアフィニティとは，同じ VCPU は同じプロセッサ内の PCPU に割り当てるというものである．プロセススケジューリングではプロセス毎にアフィニティを設定したが，仮想化環境では VM 内の状況を知ることはできないため，VCPU ごとに PCPU を割り当てる．このとき，ページシェアリングによって VM 間でページを共有している場合では，それぞれ VM の VCPU が別のプロセッサに割り当てられてしまうと，共有されていたメモリが別々のキャッシュへそれぞれ読み込まれてしまう．

[1] 慶應義塾大学
Keio University
[a)] ito.t@sslab.ics.keio.ac.jp

本研究では各 VM が利用しているメモリ間のページ共有率を考慮したアフィニティを設定することで，キャッシュミスを削減するスケジューリングの手法を提案する．

提案手法では，共有しているページを多く持つ VM 同士がもつ VCPU を同じプロセッサに割り当てた場合，そのページがキャッシュされた時に，それぞれが同じキャッシュを利用できる点に着目し，アフィニティーを設定する．

提案手法はオープンソースの仮想マシンモニタである Xen 上に実装する．ページスケジューリングの機構の追加とクレジットスケジューラの制御により行った．

本手法の有用性を示すために，ベンチマークを実行中に上記スケジューリングを実施し，提案手法の割当と相違する割当をした場合と比べて処理速度がどう変化したかを比較した．その結果，提案手法を実行した場合，処理速度が 15 %向上した．

本論文の構成を以下に示す．2 章ではスケジューリングの動作とページシェアリングの動作との連携について分析する．3 章では提案手法について説明する．4 章では提案手法の実装について説明する．5 章ではベンチマーク下における提案手法の有用性の調査とその結果について説明する．6 章ではスケジューリングとキャッシュの活用に関する関連研究を紹介する．7 章ではまとめと今後の課題を述べる．

2. 背景

2.1 VCPU スケジューリング

仮想化環境では各 VM に対して PCPU を割り当てるために VCPU スケジューリングが行われている．Xen ではクレジットスケジューラにより VCPU スケジューリングが行われている．これはマルチプロセッシング環境において効率的な CPU 配置ができるよう設計されている．各 VM に対して CPU 使用リクエストが多いものに対して CPU の持ち時間を与え，あまり利用しないものに対しては最低限の持ち時間とすることで無駄なくスケジューリングする仕組みである．

プロセスを別のプロセッサへ割り当て直した場合，キャッシュや TLB の再構築が必要となり，頻繁に行えばオーバーヘッドが大きくなる．クレジットスケジューラにおいても，仮想 CPU と物理 CPU の対応は基本的には固定され，CPU の使用バランスが大きく崩れた場合のみ割当を行い直すという対応をしている．

2.2 ページシェアリング

仮想化環境におけるリソースの割当に関して，各 VM の CPU 使用率は一般的に 5-10%のため，仮想化による多重化により余剰リソースを有効に利用できるようになるが，メモリーは各 VM ごとに情報を保持し続ける必要があるため，ボトルネックになりやすいという指摘が有る [1]．そこでこの問題を解決し，集約率を向上させるためにページシェアリングという手法 [2], [3], [6] が有る．ページシェアリングは仮想化環境において，それぞれの VM が保持しているメモリの中から重複するものを共有し除去する手法である．これによりメモリの利用効率が向上するため，VM の集約数を増やすことができる．

2.3 共有ページをキャッシュする場合

先の指摘は CPU リソースは使用率が低いため多重化に向いているというものであるが，キャッシュに関してはこの指摘が当たらない．プロセッサには L1 から LLC（L2 または L3 の場合が多い）まで数段のキャッシュがあり，下段のキャッシュになればなるほどそのプロセッサ上の多くのコアで共有している．CPU はデータを読み込む場合，これらキャッシュを順にたどるが，LLC までみても無い場合，メインメモリから読み込むこととなる．この場合 LLC にある場合のおよそ 6 倍もの CPU サイクルが必要となる [4]．そのためできるだけ多くのデータをキャッシュすることが望ましいが，プロセッサの大きさや距離等の物理的制約によりキャッシュサイズを増やすことは難しいとされている．またキャッシュはメモリと違い VM ごとに利用可能量を制限することができず，かつ LLC などの共有されているキャッシュはどの CPU からも利用されるため管理が困難である．[5].

そこで，本研究では仮想化環境において，ページシェアリングを行っている場合を対象に，VCPU スケジューリングを改善することによってキャッシュの利用効率を上げキャッシュミスによる性能低下を防ぐことを目標とする．

ページシェアリングにより共有されているページをそれぞれの VM の VCPU が同じプロセッサから読み出す場合，共有されたままのページがキャッシュに乗ることとなる，そのデータは両 VM ともに利用可能であるので，ページシェアリングによるメモリの利用効率向上と同様に，キャッシュの利用効率を上げる事ができる．

予備実験としてまず OS が共通している VM としていない VM で共有率はそれぞれどの程度のになるかを調べた．結果 Fedora20 を実行している VM を 2 台，Debian7.5 を実行している VM を 2 台準備し，起動直後のページ共有率を測定した．Fedora 同士で 52%，Debian 同士では 71%が共有できたのに対して，Debian と Fedora 間では 1%を下回る共有率であった（表 1）．次に，OS は共通の状態で，利用しているライブラリやフレームワーク等のデータが異なる場合と共通する場合で共有率がどの程度になるかを調査するための実験を行った．Debian7.5 を実行している VM を 4 台の起動させ，18MB のデータ A と B を用意し，2 台にはデータ A，残りにはデータ B を読み込ませた後ページシェアリングを行った．この結果，同一データを読み込んだ VM 同士は，異なるデータを読み込んだ VM 同士より

も 10%ほど高い共有率となった（表 2）．いずれの実験も 8 コア，2.7GHz の Intel Xeon CPU E5-2680 を 2 個搭載した物理マシン上で行い，各 VM は 512MB のメモリを割り当てた．この結果より，同一 OS で動作している場合であれば，ライブラリや基本データ等共通するデータを読み込んでいる VM はページシェアリングの結果を利用することで判別することができることが確かめられた．もちろん異なる OS が動作している場合でも共有率に従ってグルーピングすればキャッシュヒット率に有利に働くと考えられる．

表 1 OS の違いによるページ共有率

	VM1 (Debian)	VM2 (Debian)	VM3 (Fedora)	VM4 (Fedora)
VM1	-	71.17%	0.07%	0.05%
VM2	71.43%	-	0.06%	0.05%
VM3	0.06%	0.05%	-	52.18%
VM4	0.04%	0.05%	52.38%	-

表 2 読み込みデータの違いによるページ共有率

	VM1	VM2	VM3	VM4
VM1	-	64.9%	55.3%	53.6%
VM2	64.8%	-	53.3%	54.2%
VM3	55.5%	54.4%	-	67.4%
VM4	53.5%	53.4%	67.9%	-

3. 提案手法

3.1 概要

本研究ではページシェアリングによる VM 間のページ共有率を利用することで，キャッシュミスによる遅延を削減するスケジューリング手法を提案する．先の実験で示したとおり，同じデータを扱うプログラムが動作している VM 同士はページ共有率が高くなる．そこでページ共有率の高い VM をできるだけ同じプロセッサに割り当てることでキャッシュヒット率を向上させる．現在のスケジューリングは VCPU をできるだけ同じ PCPU に当てることしか考慮されていないため，キャッシュミスの削減には不十分である．提案手法を用いることで，キャッシュを有効に利用するためのより好ましい PCPU 割当を導出することが可能となる．

3.2 アプローチ

PaaS 環境では各 VM 上で動作するプログラムやライブラリには同一のものが多く存在すると考えられる．これらの同じデータの読み出しが行われる VM の VCPU を同じプロセッサに割り当てることで，他の VM によってキャッシュされたデータをそのまま利用することができ，キャッシュミスとならずに済み，また重複してデータをキャッシュすることを防ぐ事ができるため，残りのキャッシュ領域を有効に利用することができる．

ページシェアリングをメモリ全体に対して実行したタイミングで VCPU スケジューラに設定されている PCPU の割り当てを変更することで，VM のワークロードの変化に追随することができる．また，ページシェアリングの実行間隔は VMware ESX Server[3] や KSM[6] では分のオーダーであるので，割当変更の頻度は少なく，オーバーヘッドを抑えられると考えられる．

ページシェアリングによって VM 間で共有しているページは，キャッシュに乗った時に両 VM から利用可能である．従って VM 間のページ共有率が高い VM をできるだけ同じプロセッサに割り当てる．

図 1 に提案手法の設計概念図を示す．ページ共有の実行によってページを共有するごとに，そのページを参照する VM 間の共有数として加算し記録する．ページ共有が全てのページに対して完了した段階で，記録した VM 間ページ共有数をその VM が利用中のページ数で除して共有率を求める．この共有率の高い VM のペアから順に割り当てるべきプロセッサを決める．その後，VCPU スケジューラに対して導出したプロセッサに割当たるようにアフィニティを設定する．この流れをページシェアリングを実施する都度実行することで，本提案手法は実現する．

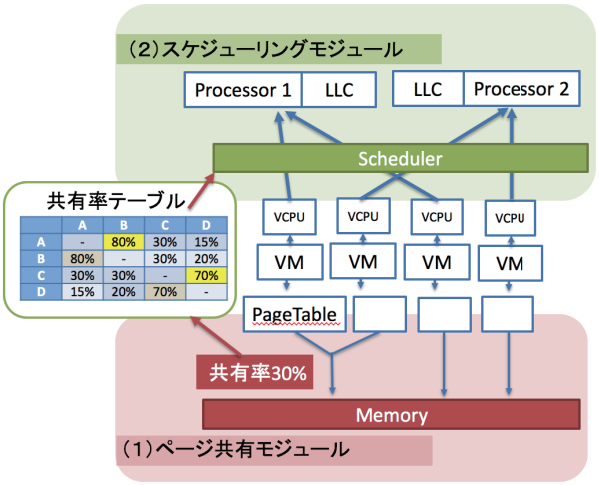

図 1 設計概念

4. 実装

提案手法を，オープンソースの仮想マシンモニタである Xen[7] 上に実装した．Xen は 4.3.3 を，ホスト OS とゲスト OS には共に Linux3.2.0 を使用した．

4.1 ページ共有率の取得

Xen には完全なページシェアリングの機構が無いため，提供されている API を組み合わせて，VMware ESX server などで実現している content-based ページシェアリング [8]

を実装する Xen ではページシェアリングのための機構として通常の OS でプロセスフォークに利用されるような CoW を用いたページ共有を実現するための API コードが含まれているため利用する．利用する API は nominate と share，そして unshare である．nominate はドメインの ID とゲストフレーム番号（GFN）を受け取りそのページを読み取り専用にする．share は nominate したページを 2 つ渡すと一方の参照するマシンフレーム番号 (MFN) を他方のものへ変更する．unshare は共有しているページへの書き込みが起こった場合にページのコピーを作成し割当なおすというものである．なお share はページ参照を変更する際，内容の比較はしないため，ページ内容を全バイト確認する機構を追加した．また，共有できるページを探索するため，VM ごとにページの内容でハッシュテーブルを作成し一致するものを全バイト比較することとした．実行フローを図 2 に示す．これらの結果，ある VM と共有に至ったページ数を全ページ数で除した数をその VM 間の共有率として保持する．

4.2 VCPU スケジューリング

Xen のクレジットスケジューラの加工を考える．まず，クレジットスケジューラでは VM 起動時にそれぞれの仮想 CPU のアフィニティに特定のプロセッサが指定され，以降そのプロセッサ上の物理 CPU が割り当てられる．ページシェアリングによる共有率に従ってプロセッサ割当を変更するためにそれぞれの VCPU に指定されたアフィニティーを変更し，共有率の高い VM を同じプロセッサに集める．(図 3)．

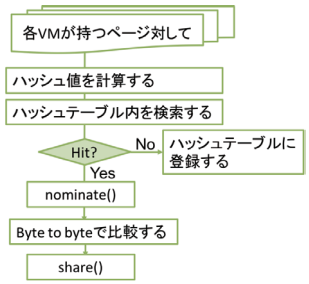

図 2 ページシェアリングと共有率の取得

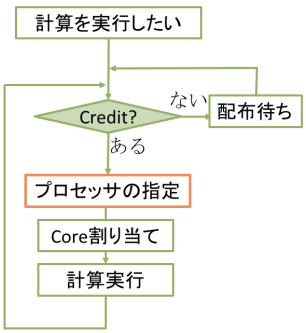

図 3 スケジューリング

ハッシュテーブルの作成から全ての share を実行し，共有率に基づいてアフィニティを設定するまでを 1 サイクルとし，これをタイミングで実行できるようにした．

5. 実験

5.1 マイクロベンチマーク

はじめに本提案手法が最大限活かされる環境を意図的に作成し，ページ共有率の高い VM を同じプロセッサへ割り当てることでどの程度の性能向上が起こるかを検証する．

ワークロードとしては，固有のデータをメモリ上にロードし，そのメモリからデータを一定回数ランダムリードを行うものを作成した．実験環境として，LLC20MB，8 コア，2.7GHz の Intel Xeon CPU E5-2680 を 2 個搭載した物理マシン上に 4 台の VM を稼働させた．各 VM には仮想 CPU を 1 個，メモリを 512MB 割り当てた．2 台の VM では Debian7.5 を実行し，15MB のランダムデータ A を使用する．残り 2 台の VM では Fedora20 を実行し，15MB のランダムデータ B を使用する．

まずこのワークロードを何度か実行した後ページシェアリングを行った結果が表 3 である．

表 3 特定データを読み込んだ場合のページ共有率

	VM1 (Debian ・dataB)	VM2 (Debian ・dataB)	VM3 (Fedora ・dataB)	VM4 (Fedora ・dataB)
VM1	-	66.20%	0.05%	0.05%
VM2	66.49%	-	0.08%	0.05%
VM3	0.03%	0.05%	-	45.42%
VM4	0.03%	0.03%	45.46%	-

この結果より同じデータ OS，データ共に同じ VM のページ共有率は高く，異なる場合はほとんど共有できないことがわかった．次に (A):ページ共有をしない場合，(B):ページ共有を行い，提案手法通りの PCPU 配置をした場合，(C):ページ共有を行い，提案手法と相違する PCPU 割当を行った場合の 3 通りで実行時間を測定した..

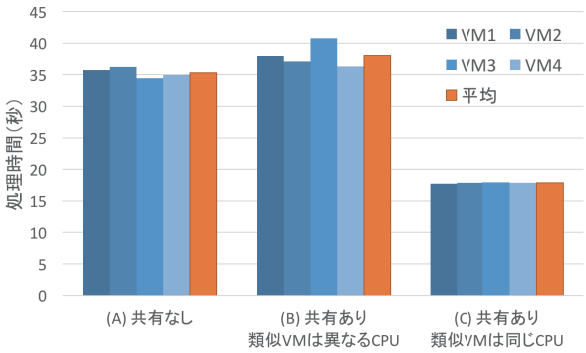

図 4 VCPU 割当による処理時間の違い

結果は図 4 のとおりである．(C) は (B) の 2.1 倍高速と

なり，キャッシュを最も有効に利用した場合の最適環境での実行時間に明らかな差があることがわかった．

5.2 Web サーバー環境での性能評価

提案手法により実際の運用でもキャッシュミスによるオーバーヘッドが削減され，処理速度が向上することを示すために，Web サーバー環境を作成し，その応答時間を提案手法によりアフィニティを変更した場合と，提案手法と反するアフィニティを設定をした場合で比較した．実験環境として，8 コア，2.7GHz の Intel Xeon CPU E5-2680 を 2 個搭載した物理マシン上に 4 台の VM を稼働させた．ホスト，ゲスト共に Debian7.5 が動作している．各 VM には仮想 CPU を 1 個，メモリを 512MB 割り当てた．各 VM はウェブサーバーとして Apache を動作させ，２つの VM にはデータ A を，残りの VM にはデータ B を読み込む php のプログラムを配置する．各 VM はローカルより http のリクエストを発行し，600 回のリクエストを行いデータ転送準備完了までの時間を比較する．

実験で計測したページ共有率を表 4 に，処理時間の変化を図 5 に示す．提案手法の場合の応答時間の分布のうち，データ A に関するもの（dataA-shared），データ B に関するもの（dataB-shared），提案手法と反する配置の場合のデータ A に関するもの（dataA-unshared），データ B に関するもの（dataB-unshared）を示している．データ A を読み込んでいる VM の応答時間は平均 5.1ms，データ B を読み込んでいる VM は 5.4ms となった．また，提案手法の配置ではない場合，データ A は 6.2ms，データ B は 6.1ms の応答時間となった．これにより提案手法を用いた場合は用いない場合よりも応答速度が 15%高速化することがわかった．また提案手法は応答時間がまとまっているのに対して，提案手法の配置ではない場合はある程度ばらつきがある．これは後者の場合，キャッシュミスが発生し，データ取得までの遅延が頻発したためと考えられる．

表 4 読み込みデータの違いによるページ共有率

	VM1 (dataA)	VM2 (dataA)	VM3 (dataB)	VM4 (dataB)
VM1	-	30.5%	24.8%	25.1%
VM2	30.5%	-	24.7%	24.8%
VM3	24.7%	24.7%	-	30.4%
VM4	25.0%	24.6%	30.5%	-

6. 関連研究

Lee らが提案する Region scheduling[9] はキャッシュの効率的な利用に着目したスケジューリング手法である．実行中のプログラムが確保しているのメモリ領域を検知し，これをキャッシュしている場所へ VM がアクセスできるよう割り当てを行うこれによってキャッシュの有効利用と，整合性検査が不要になり，高速化が果たせる．しかしながら，全メモリを対象としているため，読み出しがほとんど行われないキャッシュに対しての動作がオーバーヘッドになりやすいという問題がある．

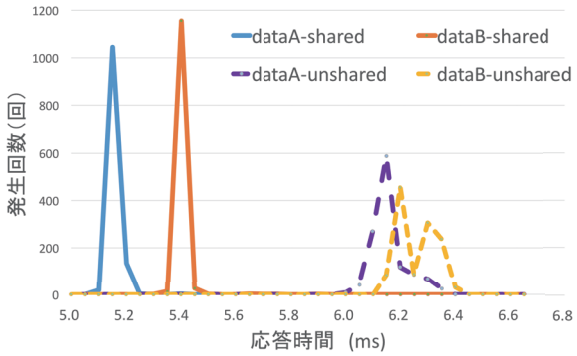

図 5 アクセス時間の比較

Seyed らの研究 [10] では，メモリバンド幅やネットワークスループットなど異なるリソースの割り当て状況を加味した割り当てを行うことで，需要が集中するリソースを有効に割り当てられるというものである．しかし個々の VM の割り当てを理想形にしたものの，全体としての性能が向上につながるわけではない．

Sisu らはリアルタイムスケジューリングのための手法として RT-xen を提案している [11]．これは，複数の PCPU に対して一つのランキューを共有し Dealline の近いスケジューリングを先に処理する手法というものである．この手法によりクレジットスケジューラよりも多くのプロセスを deadline の内に処理することが可能となった．しかしながらキャッシュインテンシブなワークロードを実行した場合，処理できる量は約 16%程減少した．これはキューを全体で共有しているために，PCPU の割当変更が頻繁に発生し，キャッシュミスが増加したためである．

Diwaker らは，CPU やディスクに比べメモリが仮想化のボトルネックになりやすい点を指摘し，ページの共有・圧縮・差分保存のための，DifferenceEngine を提案した [1]．ESX のページシェアリングよりも 1.5 倍のメモリ使用を削減でき VM の集約数を増やすことにも成功しているが，キャッシュに対しては考慮されておらず，オーバーヘッドとして 7%程度のの処理時間増加が確認されている．

7. まとめと今後の課題

本論文ではページシェアリングにより，メモリページを共有している VM を共有率が高いものから同じプロセッサに割り当てる事によって，キャッシュヒット率を向上させる手法を提案した．

提案手法ではページシェアリングによって VM 間で共有できたページの量を元に，VCPU スケジューリングのアフィニティを変更し，プロセッサを割り当てることで実現できる．これを Xen4.3.3 上に提案手法を実装した．まず既

存 API を加工し，ページシェアリングを実装し，それぞれの VM 間で共有できたページ量を返す機構を追加した．この値を元にクレジットスケジューラのアフィニティー設定により，VCPU を割り当てるべきプロセッサを指定する．これによってページ共有率の高い VM が持つ VCPU は同じプロセッサで動作するようにできる．提案手法を用いることで，現状のクレジットスケジューラによるスケジューリングの場合に比べ，処理速度がおよそ 15%向上することが確かめられた．これは共有しているページを持つ VM の VCPU が，共通のキャッシュを使用しているため，冗長なキャッシュを持つ必要がなくなったことで，キャッシュミスを削減できたためと言える．本手法では，メモリーの書き換えが頻発するワークロードの場合，ページシェアリングの効果があまり出ず，従ってキャッシュも有効に利用できないため，VCPU の割当変更によるキャッシュミスがオーバーヘッドになりえる．またどページシェアリングの頻度と CPU の割当の変更のタイミングを調整する仕組みや，プロセッサごとの使用率の偏りを修正する機構などは，今後の検討課題である．

参考文献

[1] Gupta, D., Lee, S., Vrable, M., Savage, S., Snoeren, A. C., Varghese, G., Voelker, G. M. and Vahdat, A.: Difference Engine: Harnessing Memory Redundancy in Virtual Machines, *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, OSDI'08, Berkeley, CA, USA, USENIX Association, pp. 309–322 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1855741.1855763⟩ (2008).

[2] Bugnion, E., Devine, S. and Rosenblum, M.: Disco: Running Commodity Operating Systems on Scalable Multiprocessors, *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, SOSP '97, New York, NY, USA, ACM, pp. 143–156 (online), DOI: 10.1145/268998.266672 (1997).

[3] Waldspurger, C. A.: Memory Resource Management in VMware ESX Server, *Proceedings of the 5th Symposium on Operating Systems Design and implementationCopyright Restrictions Prevent ACM from Being Able to Make the PDFs for This Conference Available for Downloading*, OSDI '02, New York, NY, USA, ACM, pp. 181–194 (online), DOI: 10.1145/1060289.1060307 (2002).

[4] Intel Corporation: *Intel® 64 and IA-32 Architectures Software Developer's Manual*, http://www.intel.com/products/processor/manuals/ (2008).

[5] Guan, N., Stigge, M., Yi, W. and Yu, G.: Cache-aware Scheduling and Analysis for Multicores, *Proceedings of the Seventh ACM International Conference on Embedded Software*, EMSOFT '09, New York, NY, USA, ACM, pp. 245–254 (online), DOI: 10.1145/1629335.1629369 (2009).

[6] Arcangeli, A., E. I. . and Wright, C.: Increasing memory density by using KSM, *Linux Symposium*, pp. 19–28 (2009).

[7] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and the Art of Virtualization, *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, New York, NY, USA, ACM, pp. 164–177 (online), DOI: 10.1145/945445.945462 (2003).

[8] 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).

[9] Lee, M. and Schwan, K.: Region Scheduling: Efficiently Using the Cache Architectures via Page-level Affinity, *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVII, New York, NY, USA, ACM, pp. 451–462 (online), DOI: 10.1145/2150976.2151023 (2012).

[10] Zahedi, S. M. and Lee, B. C.: REF: Resource Elasticity Fairness with Sharing Incentives for Multiprocessors, *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, New York, NY, USA, ACM, pp. 145–160 (online), DOI: 10.1145/2541940.2541962 (2014).

[11] Xi, S., Xu, M., Lu, C., Phan, L. T. X., Gill, C., Sokolsky, O. and Lee, I.: Real-time Multi-core Virtual Machine Scheduling in Xen, *Proceedings of the 14th International Conference on Embedded Software*, EMSOFT '14, New York, NY, USA, ACM, pp. 27:1–27:10 (online), DOI: 10.1145/2656045.2656066 (2014).