Vol.2017-OS-139 No.12
2017/3/2

仮想マシン移送を考慮した リソースレイアウト最適化支援機構

中嶋 将人[1,a)]　山田 浩史[1,b)]

概要：アプリケーションの性能は，使用するリソースのレイアウトによって大きく変動する．これは遅延が大きいメモリ領域へのアクセス多発やリソース競合により発生する現象で，今日提供されているクラウドサービスの運用元でも生じる可能性がある．この性能劣化は広く知られているため，最適なリソースレイアウトを実現するための手法は数多く提案されている．また多くのクラウドサービスは仮想化環境上での運用が行われており，こうした環境では仮想マシン移送が利用できる．しかし，仮想マシン移送の実行によりリソースレイアウトの悪化が生じ，移送される仮想マシン (VM) のみならず，移送先の VM の性能も劣化することがある．これは移送 VM の状況を考慮しないリソース割り当てが原因である．また既存のリソースレイアウト最適化手法は VM 移送の実行を想定しておらず，それらを移送 VM を考慮する形にアドホックに作り込むのはコストが高い．そこで本研究では，移送 VM を考慮してリソースをレイアウトするための機構を提案する．VM 移送による性能劣化を抑制するとともに，VM 移送実行環境下において既存手法のレイアウト最適化ポリシーを容易に利用可能にする．提案機構を Xen 4.4.2 に実装し，複数の既存最適化ポリシーを利用できるようにした．そして VM 移送が実行される環境下において評価実験を行ったところ，リソース使用状況やポリシーに応じたリソースレイアウトの最適化が行われており，VM 移送直後の性能劣化を抑制できていることが確認できた．

1. Introduction

アプリケーションの性能は，使用するリソース (CPU コアやメモリ) のレイアウトによって大きく変動する．これは遅延が大きいメモリ領域へのアクセス多発やリソース競合により発生する現象で，たとえば，NUMA(Non-Uniform Memory Access) 型アーキテクチャにおける Remote Access や，メモリアクセスの集中によるメモリコントローラの競合が原因になる．Amazon EC2 [1] や Google Compute Engine [2] に代表されるクラウドサービスでは，仮想マシン (VM) をユーザに提供するため，1 台の物理マシン上で複数ユーザの VM を稼動させることが一般的であり，CPU インテンシブやメモリインテンシブなアプリケーション用の VM が提供されていることから，このような性能劣化が生じる恐れがある．この性能劣化は広く知られており，最適なリソースレイアウトを実現するためのレイアウト変更手法 [3], [4], [5], [6], [7] やタスクスケジューリング手法 [8], [9], [10], [11] は数多く提案されている．実際に，リソースレイアウトの違いによる性能差を確認する予備実験を行ったところ，既成ベンチマークの性能において最大で 68.3%の差が見られた．

先ほど挙げたものを含め，多くのクラウドサービスは仮想化環境上での運用が行われており，こうした環境では VM 移送が利用できる．また VM 移送の一種にライブマイグレーションという技術があり，VM を稼働させたまま他の物理マシン上に移すことができる [12]．ライブマイグレーションを利用することで VM やその利用者の状態に関係なく移送が行えるため，データセンタ等における負荷分散やメンテナンスが行いやすくなる．この技術は実際に利用されており，たとえば Google のデータセンタでは VM 移送を活用した運用がなされている．

しかし，VM 移送の実行によりリソースレイアウトの悪化が生じ，移送 VM のみならず，移送先の VM の性能も劣化することがある．これは移送 VM の状況を考慮しないリソース割り当てが原因であり，たとえ移送先マシンにおいて VM のメモリや仮想 CPU(vCPU) のレイアウトが適切であったとしても，移送 VM が加わることにより，そのレイアウトが適切でなくなってしまう．たとえば，頻繁にアクセスされるメモリ領域がコアとは異なる socket に配置されることもある．リソースレイアウト最適化のための手法は既に数多く存在するが，それらは移送 VM を考慮して

[1] 東京農工大学
TUAT, Koganei, Tokyo 2-24-16, Japan
[a)] masanakaji@asg.cs.tuat.ac.jp
[b)] hiroshiy@cc.tuat.ac.jp

いない．これらを移送 VM を考慮する形にするには，移送 VM の稼動状況を移送元から取得し，移送 VM のために確保したメモリや vCPU の配置を変化させるよう各手法にアドホックに作り込む必要があり，コストが高い．

そこで本研究では，移送 VM を考慮してリソースをレイアウトするための機構を提案する．VM 移送による性能劣化を抑制するとともに，VM 移送実行環境下において既存手法のレイアウト最適化ポリシーを容易に利用可能にする．提案手法では，移送実行環境への適用に必要な処理を提案機構が肩代わりすることにより，移送 VM であることを隠蔽し，既存ポリシーに対して通常の稼動 VM のように見せる．これにより，移送 VM のリソース使用状況を透過的に加味することが可能となり，移送 VM を含めた形で移送実行中にリソースの再配置を実現する．具体的には，移送開始時に移送 VM の情報を移送先へ伝達し，伝え終わり次第マシン全体のリソースを再配置させる．また，リソース再配置時に既存の最適化ポリシーを利用するためのインタフェースを提供する．このインターフェースにより，仮想化環境におけるパフォーマンスカウンタのモニタリングやリソースレイアウトの変更が可能になる．

提案機構を Xen 4.4.2 に実装し，複数の既存最適化ポリシーを利用できるようにした．そして VM 移送が実行される環境下において，それぞれのポリシーを用いた評価実験を行った．実験結果から，リソース使用状況やポリシーに応じたリソースレイアウトの最適化が行われており，VM 移送直後の性能劣化を抑制できていることが確認できた．

本論文では，第 2 章で本研究の背景にあるリソースレイアウトや VM 移送に触れ，問題を提示する．そして第 3 章で問題を解消するための機構を提案する．続く第 4 章で実装について述べ，第 5 章では評価を行う．第 6 章では関連研究を示し，第 7 章でまとめと今後の展望を述べる．

2. Background

2.1 Resource Layout

リソースレイアウトとは，アプリケーションが使用する CPU コアやメモリといったリソースの物理的な配置のことであり，レイアウトの違いにより性能が大きく変動する．遅延が大きいメモリ領域へのアクセスやリソース競合が多いほど性能は劣化し，たとえば，NUMA(Non-Uniform Memory Access) 型アーキテクチャにおける Remote Access や，メモリアクセスの集中によるメモリコントローラの競合により遅延が生じる．

図 1 は NUMA において性能劣化が発生している状況の例である．このような性能劣化はすでに広く知られているため，リソースレイアウトを最適化するための研究は多くなされている．既存手法としては，メモリページの移送と複製を行いメモリレイアウトを最適化する Carrefour [5] や，vCPU のレイアウトを最適化することで VM における高コストなデータアクセスを減少させる Bias Random vCPU Migration [4] などがある．

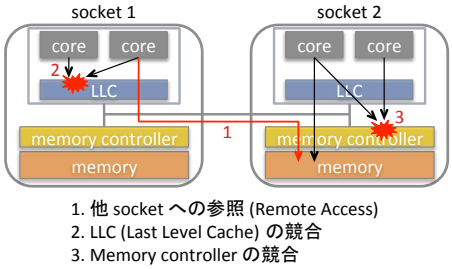

図 1 NUMA における性能劣化

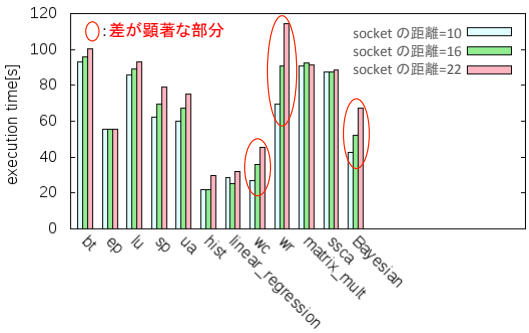

図 2 socket 間の距離に応じた性能差

2.1.1 Preliminary Experiment

リソースレイアウトによる性能差を確認する目的で予備実験を行う．実験は Dell PowerEdge C6145 (4 AMD Opteron Processors 6348, 48 cores, 8 sockets, 128GB memory) 上で行い，ベンチマークとして NAS Parallel Benchmark suite v3.3 [13], Metis MapReduce benchmark suite [14], SSCA2 v2.2 [15], PARSEC benchmark suite v3.0 [16], CSU Face Identification Evaluation System v5.1 [17] を使用する．そして，使用する CPU コアとメモリが含まれる socket 間の距離を三通りに変え，それぞれにおいてベンチマークの実行時間を測定する．

図 2 が結果の一部であり，横軸がベンチマーク名，縦軸が実行完了までに要した時間 [秒] を表している．さらに，まとまっている三本のグラフは同一ベンチマークにおける結果であり，socket 間の距離を 10，16，22 とした時の性能を示している．距離が 10 の時と 22 の時との性能差が最大となったのは wc であり，68.3%という値が得られた．また wr や Bayesian においても 50%以上の差が見られた．CPU コアとメモリの距離が遠いほどメモリアクセスによる遅延が大きくなることを踏まえると，距離による性能への影響が大きいこれらのベンチマークは，メモリインテンシブなアプリケーションだと考えられる．

2.2 Live Migration

ライブマイグレーションは VM 移送の一種 [12] であり，対象 VM を稼働させたまま移送を行う技術である．そのた

め VM 上で動作しているソフトウェアが停止することはない．移送の際は，まず移送開始時点での対象 VM の情報を移送先に転送する．しかし転送中も VM は稼働しているので，メモリの内容は随時更新されてしまう．そこで，一回目の転送が終わった後にメモリブロックの差分を再度転送する．そしてその処理中に更新された内容を再び転送する，といった処理を，移送元と移送先のメモリ内容が同期するまで繰り返す．その後 VM を一瞬停止させ CPU レジスタなどの情報を転送し，移送元の対象 VM を削除することで完了となる．

ライブマイグレーションを利用すると，VM や利用者の状態に関係なく移送を行うことができるので，物理マシンの状況のみを考慮した効率的なリソース管理が可能になる．たとえば複数のサーバが稼働しているデータセンタ等を考えると，あるサーバに負荷が集中している状況で利用者に意識されることなく負荷分散を行える．またメンテナンス等のために物理マシンを停止させたい場合も，マシン上で動作しているサービスを止めることなく他の物理マシンに移すことができる．

2.3 Performance Degradation

VM 移送の実行によりリソースレイアウトの悪化が生じ，移送 VM のみならず，移送先の VM の性能も劣化することがある．これは移送 VM の状況を考慮しないリソース割り当てが原因であり，たとえ移送先マシンにおいて VM のリソースレイアウトが適切だったとしても，移送 VM が加わることで適切でなくなってしまう．たとえば，頻繁にアクセスされるメモリ領域がコアとは異なる socket に配置されることもある．図 3 の上はそのような状況を表しており，緑色網掛け部のメモリレイアウトが悪くなったことで Remote Access (赤い矢印) が多発するようになっている．この例では移送 VM の性能が下がるが，VM が移送されてくることでリソース競合などの問題発生が促され，移送先マシンで元々稼働していた VM の性能が劣化する可能性もある．

リソースレイアウト最適化のための手法は既に数多く存在するが，それらは VM 移送の実行を想定していないので，VM の移送中には利用できない．また既存手法は観測した状況に応じてリソースレイアウトの最適化を行うため，移送されてきた VM を含めてマシン全体をモニタリングし直すことで，悪くなったレイアウトを改善することはできる．しかしモニタリングの実行によるタイムラグが生じてしまい，VM 移送完了直後の性能劣化を防ぐことはできない．加えて，個々の既存手法を VM 移送実行環境に適用するのは現実的でない．なぜなら移送 VM を考慮できるようにするには，移送 VM の稼働状況を移送元から取得し，移送 VM のために確保したメモリや vCPU の配置を変化させるよう各手法にアドホックに作り込む必要があり，開発コストが高すぎるからである．

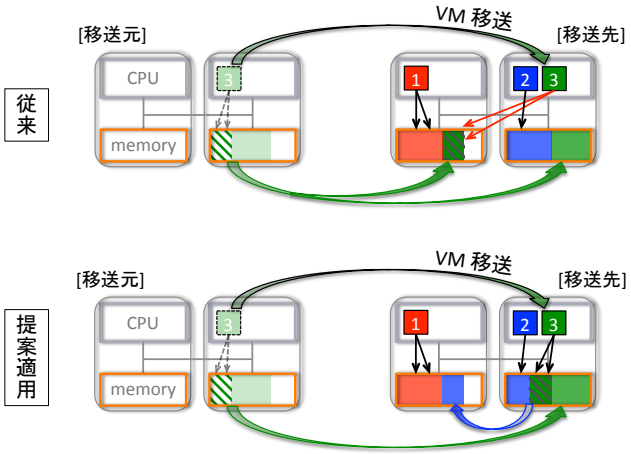

図 3 従来の移送と提案適用時の移送

3. Proposal

VM 移送による性能劣化を抑制するために，移送 VM を考慮したリソースレイアウト機構を提案する．提案機構は，VM 移送中にリソースを再配置することでレイアウトの悪化を防ぐ．また，VM 移送実行環境下において既存手法のレイアウト最適化ポリシーを容易に利用可能にする．VM 移送実行環境への適用に必要な処理を提案機構が肩代わりすることにより，移送 VM であることを隠蔽し，既存ポリシーに対して通常の稼働 VM のように見せる．

図 3 の下は提案適用時の状況を示しており，この時，移送 VM の性能は劣化しない．なぜなら，頻繁にアクセスされる緑色網掛け部のメモリ領域をコアと同じ socket に配置できているからである．リソース使用状況から最適なリソースレイアウトを判断し，移送 VM 用のメモリ領域 (緑色) を確保するために，移送先で元々稼働していた VM 2 のメモリ (青色) の一部を移動する．このレイアウト変更により VM 2 のリソースレイアウトは悪くなっているが，移動したメモリ領域へのアクセスは少ないため性能への影響も小さい．よって，移送先マシン全体を考えると良くなっていると言える．

提案を実現する上で，VM 移送中のリソース再配置をどのように行うかが課題になる．なぜなら，移送実行中に移送 VM の状況を考慮したリソースレイアウトを行うには，移送 VM の稼働状況を把握している必要があるが，移送先マシンはそのような情報を持っていないからからである．加えて，多数ある既存ポリシーをどのように VM 移送実行環境に適用するのかも課題となる．最適化に用いる情報や想定する環境はポリシーによって異なるため，それらすべてを一括して肩代わりしなければならない．

3.1 Resource Layout during Live Migration

VM 移送中のリソース再配置を可能にすることで，移送

によるレイアウトの悪化を防ぐ．再配置の際には，移送先で元々稼働していた VM に加えて移送 VM の状況も考慮する．具体的には，移送開始時に移送 VM の情報を移送先マシンへ伝達し，伝え終わり次第マシン全体のリソースを再配置させる．移送先マシンは受け取った情報から移送 VM のリソース使用状況を把握し，それを加味して新しいレイアウトを決定する．そして，VM 移送に要する時間を利用してレイアウトの変更を行う．リソース使用状況を判断するための情報としては，キャッシュミス率やメモリコントローラへの要求数などを利用する．VM 移送実行時にすべての VM の状況を考慮できるように，移送元マシンと移送先マシンそれぞれにおいて定期的にモニタリングを行っておく．

3.2 Applying Existing Resource Layout Policies

VM 移送実行環境でのリソースレイアウト変更時に既存の最適化ポリシーを利用できるように，複数のインタフェースを提供する．それらを利用することで，VM 移送の実行を想定していない多数の既存ポリシーを個別に適用させる必要がなくなる．つまり環境の違いにより生じるギャップを一括して埋めることができる．たとえば，既存ポリシーの中には仮想化環境を想定していないものもあり，それらが使用する情報を仮想化環境でも同様に取得できるとは限らない．また仮想化環境ではリソースの管理法が複雑になるため，リソースレイアウトを変更するために必要な処理も純粋な OS 上とは異なる．しかし本研究で提供するインタフェースを利用すると，環境の違いを意識することなく，パフォーマンスカウンタのモニタリングやリソースレイアウトの変更を実行できる．

4. Implementation

提案機構は三つの要素で構成される．それぞれの要素は 1) *Monitor* :VM 情報の収集，2) *Transferer* :移送 VM 情報の伝達，3) *Optimizer* :リソースレイアウトの最適化を担当する．図 4 は提案機構の概略である．*Monitor* は移送元マシンと移送先マシンそれぞれにおいて動作し，マシン上で稼働している VM のモニタリングを定期的に行う．そして VM 移送実行時には，*Transferer* が移送 VM の情報を移送先マシンに伝える．その情報を移送先マシン内の *Optimizer* が受け取り，元々保持していた VM 情報と合わせてすべてを考慮したリソースレイアウトの最適化を行う．

本研究では，提案機構を Xen 4.4.2 上に実装した．また改良は施していないが，Xen 上で起動する VM の OS には Linux 4.0 を利用した．

4.1 *Monitor*

VM のリソース使用状況を把握するためにモニタリングを行う．既存の最適化ポリシーの多くが，LLC ミス数

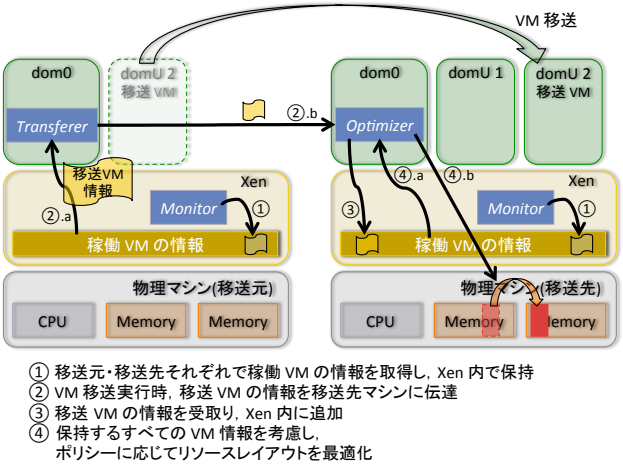

図 4 提案機構の概略

などのハードウェアパフォーマンスカウンタの情報を利用しているので，*Monitor* ではカウンタ値を収集する．カウンタ値を取得する際は，hypervisor 層から MSRs(Model Specific Registers) を直接操作する．観測したいイベントに対応する値を特定のレジスタに書き込み，その後に特定のレジスタを読み込むことで，指定したイベントのカウンタ値を得ることができる．取得した情報は Xen 内で保持する．

4.2 *Transferer*

VM 移送実行時に，移送 VM に関するモニタリング結果を移送先マシンに伝達する．情報伝達をすることで，移送先でモニタリングをやり直すことなく移送 VM のリソース使用状況を把握できるようになる．この操作により VM 移送中のリソースレイアウトの最適化が可能になる．伝達する情報は *Monitor* が得たものであり，必要に応じて Xen から取り出す．

4.3 *Optimizer*

移送 VM を考慮したリソースレイアウトの最適化を行う．そのために *Transferer* から送られてくる情報を受け取り，Xen 内に追加する．そして Xen 内で保持するすべての情報を踏まえて，全 VM のリソース使用状況を考慮したリソース再配置を行う．最適なレイアウトは指定したポリシーに応じて決定し，その後実際にレイアウトを変更する．レイアウト変更の対象にするリソースはメモリと vCPU とする．

リソース再配置時に利用する最適化ポリシーとして，オリジナルなものも含めた複数を組み込んだ．表 1 はそれらをまとめたものである．

4.4 Interfaces

VM 移送実行環境において既存の最適化ポリシーを利用可能にするためには，移送 VM であることを意識しないモ

表 1 リソースレイアウト最適化ポリシー

名称	概要
Simple (オリジナル)	LLC ミス率が高い VM ほど優先して良いレイアウトとする
DINO [3]	LLC ミス率が高い VM を別 socket に分散させ，リソース競合の発生を抑制する
AsymSched [7]	メモリ使用帯域が大きい VM の高コストなアクセス (remote access) を減らす

表 2 Interfaces

Register_vcpu_id(int domid, int vcpu_id, int set)
ReadWrite_msr(int read, int event, unsigned long *val)
Move_pages_socket(int ALL, unsigned long num, int domid, unsigned int dst_socket)

ニタリングとリソース再配置を行えるようにしなければならない．なぜなら既存手法では VM 移送の実行を考慮しておらず，新たな VM が追加される環境下での適切な状況判断とレイアウト変更が行えないからである．また手法によっては仮想化環境自体も想定していないため，情報取得やレイアウト変更を正しく行えないこともある．そこで本研究では，環境の違いを透過にしたモニタリングとリソースレイアウト変更を実行するインタフェースを提供する．表 2 は今回追加したインタフェースをまとめたものである．

Register_vcpu_id() はモニタリング対象を決定する際に利用できる．domid で指定した VM が使用する vCPU の内，vcpu_id で指定したものと対応付けられている物理 CPU のパフォーマンスカウンタを観測対象とする．set を 1 として実行した時にモニタリングを開始し，0 で実行した時点で終了する．ReadWrite_msr() は MSRs への読み書きを行うインタフェースで，read の値に応じてカウンタ値取得に必要な処理を実行する．event により観測対象のイベントを指定でき，取得したカウンタ値は val に格納する．

Move_pages_socket() は VM のメモリページを移動するためのインタフェースであり，メモリレイアウト変更時に利用できる．domid で指定した VM のページを dst_socket で指定した socket に移動するのだが，移動するページは ALL の値に応じて決定する．ALL=1 の際は対象 VM が使用するページを num の数だけ適当に選び，一気に移動させる．一方，ALL=0 の際は num を MFN(Machine Frame Number) として捉え，対応する 1 ページを他 socket に移す．また vCPU のレイアウト変更には，Xen に用意されている xl コマンドの一つである vcpu-pin を利用する．

5. Case Studies

5.1 目的

本研究で実装した提案機構を実際に動作させ，VM 移送実行環境下における VM の性能への影響を評価する．また指定したポリシーによらず，ポリシーに応じた最適化が行えているかどうかを確認する．

5.2 実験環境

実験マシンとして Dell PowerEdge C6145(4 AMD Opteron Processors 6348, 48 cores, 8 sockets, 128GB memory) を二台用意し，一方を移送元，もう一方を移送先として使用する．それぞれのマシン上で Xen 4.4.2 を用いた仮想化環境を作り出し，その上で Linux 4.0 を用いた VM を起動する．各 VM には 1 vCPU と 5GB memory を割り当てる．

VM の性能評価には，二種類の自作ベンチマーク (malloc_access，cubic_calc) を用いる．malloc_access は確保したメモリ領域に対してひたすらアクセスし続けるもので，メモリインテンシブなアプリケーションとして扱う．一方 cubic_calc は，三次方程式の計算を主処理とする CPU インテンシブなアプリケーションである．また，これらには 2GB のメモリを使用させる．

5.3 Case Study 1：Simple

5.3.1 実験方法

VM 上で動作するベンチマークの性能 (スループット) を測定する．リソースレイアウト最適化ポリシーは Simple を利用する．実験開始から 20 秒経過した時点で VM の移送を開始し，移送が完了した後もしばらく測定を続ける．このようにして移送元マシン上，移送中，移送先マシン上での性能をそれぞれ記録する．その際，移送先マシンでは二つの VM を稼働させておき，その上で動作するベンチマークの性能も測る．また本実験は，1)*proposal* :提案機構を適用，2)*existing method* :移送完了後にモニタリングを始めて最適化を図る，3)*default* :リソースレイアウトの変更を行わない，の三つの状況下で行い，それらの結果を比較する．さらに移送先マシンにおける各 socket のメモリ領域を圧迫させ，移送 VM を考慮しない移送では確実にレイアウトが悪化する状態を作り出しておく．

上記の実験を，VM 上で実行するベンチマークの種類を変えて二回を行う．Case 1-1 では移送 VM 上で malloc_access(LLC ミス率 高) を，移送先の VM 上で cubic_calc(LLC ミス率 低) を実行する．一方 Case 1-2 では移送 VM 上で cubic_calc を，移送先の VM 上で malloc_access を実行する．

5.3.2 実験結果

図 5 と図 6 が Case 1-1 の結果である．縦軸がベンチマークのスループット，横軸が経過時間を表している．図 5 より，状況に応じて移送 VM の性能に差が出るのは VM 移送完了 (実験開始後 100 秒) 以降であることが分かる．*default* では移送によるリソースレイアウトの悪化で 40.4%の性

能劣化が生じた．しかし *proposal* ではこの性能劣化を抑制し，移送完了直後から移送前と同等の性能を示している．これは移送 VM の LLC ミス率を考慮し，移送実行中に良いレイアウトを実現できているからである．*existing method* でも最終的には良い性能になっているが，性能を取り戻すまでに 22 秒かかってしまった．移送先で稼働する VM 上の性能は図 6 で確認でき，レイアウト最適化に伴うメモリ移動時 (*proposal* における移送開始時付近と *existing method* における移送完了時付近) に一時的な劣化が生じているだけである．最適化によりこの VM のリソースレイアウトは悪くなったが，LLC ミス率が低いため性能への影響はほとんどなかった．つまり，ポリシー Simple による最適化は正しく働いていると言える．

Case 1-2 の結果は図 7 と図 8 が示しているが，これらにおいては，三つの状況下での結果に大きな差は生じていない．なぜなら移送 VM における LLC ミス率が低いため優先されることがなく，提案適用により差が出るようなリソースレイアウトの変更が行われなかったからである．このことから，すべての VM の稼働状況を考慮してリソースをレイアウトできていることが確認できた．

5.4 Case Study 2：DINO

5.4.1 実験方法

基本な方法は 5.3.1 と同様である．ただしリソースレイアウト最適化ポリシーには DINO [3] を利用する．また移送先で稼働しておく VM のうちの一方で頻繁なメモリアクセスを発生させ，LLC ミス率が高くなるようにする．そして，その VM と移送 VM が使用するリソースが同じ socket に配置されるような状況を想定する．

5.4.2 実験結果

本実験の結果を図 9 と図 10 に示す．結果の傾向は 5.3.2 と似ている．移送 VM 上の性能としては，*default* において生じた 30.8%の性能劣化を *proposal* で抑制できており，*existing method* では性能が上がるまでに 17 秒かかった．移送先で稼働する VM 上の性能は *default* が最良になっているが，これはリソースレイアウトの変更により移送 VM と同一 socket 内のリソースを使うようになったからである．つまりリソース競合による遅延が発生するようになった．しかしこの VM における性能劣化は 10.0%であり，移送 VM の性能向上率 30.8%と比べると小さいので，移送先マシン全体の性能を考えると良くなっていると言える．

また，移送 VM 上で cubic_calc を実行した際はリソースレイアウトの変更が起こらず，三つの状況下での結果に差は見られなかった．

5.5 Case Study 3：AsymSched

5.5.1 実験方法

基本な方法は 5.3.1 と同様である．ただしリソースレイアウト最適化ポリシーには AsymSched [7] を利用する．また本実験においては，malloc_access をメモリ使用帯域が大きいアプリケーション，cubic_calc をメモリ使用帯域が小さいアプリケーションとして扱う．

5.5.2 実験結果

本実験の結果を図 11 と図 12 に示すが，傾向は 5.3.2 と同様である．移送 VM 上の性能としては，*default* において生じた 39.0%の性能劣化を *proposal* で抑制できており，*existing method* では性能が上がるまでに 12 秒かかった．移送先で稼働する VM 上の性能に状況に応じた大きな差は見られず，リソースレイアウト変更時に一時的な差が出るだけであった．

また，移送 VM 上で cubic_calc を実行した際は性能に差が出るようなリソースレイアウトの変更は行われず，三つの状況下での結果に差は見られなかった．

6. Related Work

Carrefour [5] は，NUMA における性能劣化を解消するためのメモリレイアウト最適化手法である．メモリバスやメモリコントローラの使用率を均等にすることで競合の発生を抑制するとともに，Remote Access を減少させる．そのためにメモリアクセスの状況を観測し，状況に応じてメモリページの移動と複製を行う．しかし対応していない仮想化環境下では，状況判断のために利用する情報を十分に取得できない．また仮想化環境に適用させたとしても，レイアウト最適化の前にはモニタリングが必要なので，VM が移送されてきた直後の性能劣化を防ぐことはできない．

Bias Random vCPU Migration [4] は仮想化環境に対応しており，vCPU のレイアウトを最適化することで VM の性能を上げる．vCPU のレイアウトとは，各 vCPU がどの物理 CPU と対応付けられているかを意味し，この対応関係を変えることにより高コストなアクセスの発生を抑える．だが，この手法でも状況判断のための情報収集は必要になるので，VM 移送実行時は移送 VM の情報を得るまで最適化を行えない．

VM 移送を利用した負荷分散手法に A-DRM [18] と呼ばれるものがある．A-DRM は複数の物理マシン間で負荷を分散する際に利用され，各マシン内におけるリソース使用率が均等になるように VM を移送する．仮想化環境で利用されることになるが，VM からは見えないメモリバスや LLC の状況も考慮したバランシングを行う．しかし，VM 移送によりリソースレイアウトが悪化することがある以上，負荷分散のために行う処理で逆に性能が下がってしまう可能性がある．

今回提案した機構では，VM 移送実行時にその VM の状況を判断するための情報を伝達することで，移送先における即座なリソースレイアウトの変更を可能にしている．すなわち移送による性能劣化を抑制できるので，VM 移送を

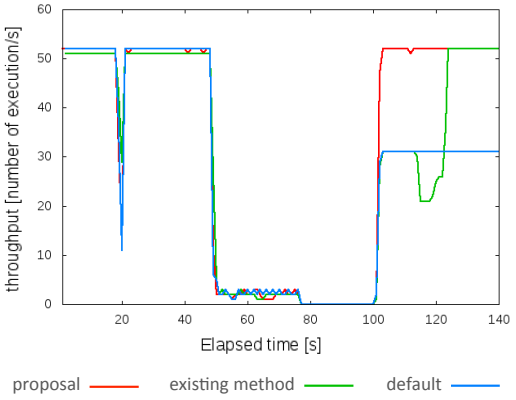

図 5 Case 1-1：移送 VM 上の malloc_access

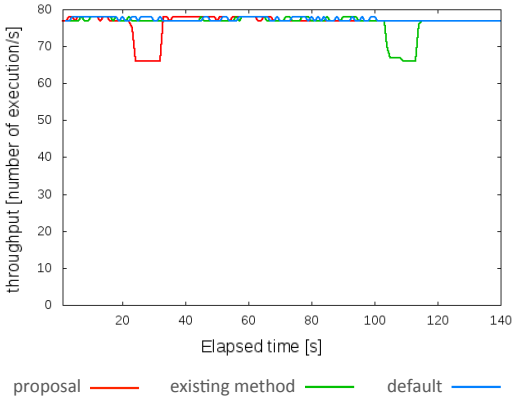

図 6 Case 1-1：移送先で稼働する VM 上の cubic_calc

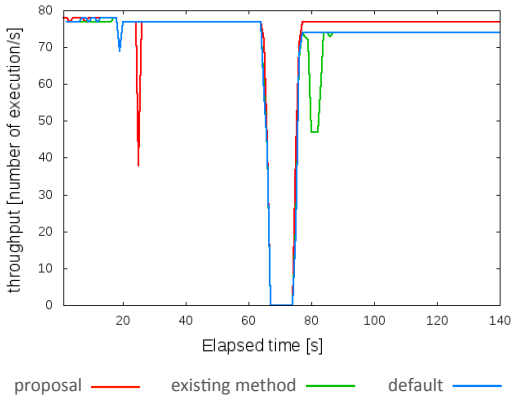

図 7 Case 1-2：移送 VM 上の cubic_calc

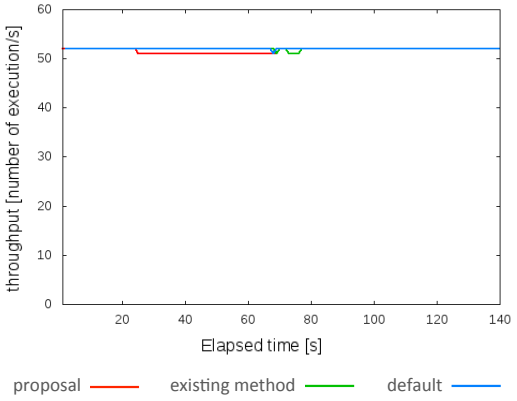

図 8 Case 1-2：移送先で稼働する VM 上の malloc_access

利用する A-DRM などの既存手法による恩恵を大きくすることもできる．

7. Conclusion

本研究では，VM 移送による性能劣化を抑制するためのリソースレイアウト機構を提案した．提案機構を Xen 4.4.2 上に実装し，最適化ポリシーやリソース使用状況を変えて評価実験を行ったところ，ポリシーや状況に応じた最適化により移送直後の性能劣化が抑制されていることを確認できた．今後は提案機構の汎用性を上げるために，複数 VM を同時に移送した場合などのより複雑な環境下への適用を目指す．また機構に組み込む最適化ポリシーの種類を増やす．

謝辞

本研究の一部は総務省 SCOPE(152103004) の助成を受けたものである．

参考文献

[1] Amazon: amazon web services, `https://aws.amazon.com/jp/?nc2=h_lg`.
[2] Google: Google Cloud Platform, `https://cloud.google.com`.
[3] Blagodurov, S., Zhuravlev, S., Dashti, M. and Fedorova, A.: A Case for NUMA-aware Contention Management on Multicore Systems, *2011 USENIX Annual Technical Conference (ATC)* (2011).
[4] Rao, J., Wang, K., Zhou, X. and Xu, C.-Z.: Optimizing Virtual Machine Scheduling in NUMA Multicore Systems, *High Performance Computer Architecture (HPCA)*, pp. 306–317 (2013).
[5] Dashti, M., Fedorova, A., Funston, J., Gaud, F., Lachaize, R., Lepers, B., Quema, V. and Roth, M.: *Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 381–394 (2013).
[6] Gaud, F., Lepers, B., Decouchant, J., Funston, J., Fedorova, A. and Quema, V.: Large Pages May Be Harmful on NUMA Systems, *2014 USENIX Annual Technical Conference (ATC)*, pp. 231–242 (2014).
[7] Lepers, B., Quema, V. and Fedorova, A.: Thread and Memory Placement on NUMA Systems: Asymmetry Matters, *2015 USENIX Annual Technical Conference (ATC)*, pp. 277–289 (2015).
[8] Zhuravlev, S., Blagodurov, S. and Fedorova, A.: Addressing Shared Resource Contention in Multicore Processors via Scheduling, *Fifteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 129–142 (2010).
[9] Merkel, A., Stoess, J. and Bellosa, F.: Resource-conscious scheduling for energy efficiency on multicore processors, *5th European Conference on Computer Systems (EuroSys)*, pp. 153–166 (2010).
[10] Srikanthan, S., Dwarkadas, S. and Shen, K.: Data Sharing or Resource Contention: Toward Performance Trans-

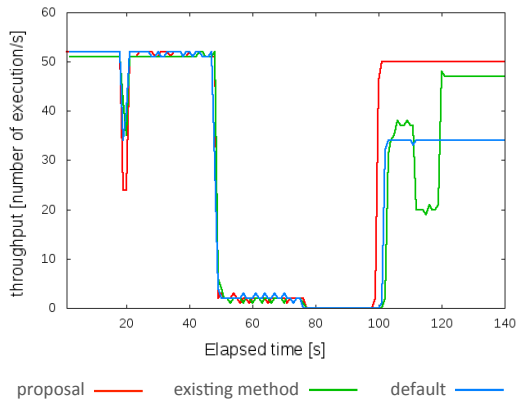

図 9 Case 2：移送 VM 上の malloc_access

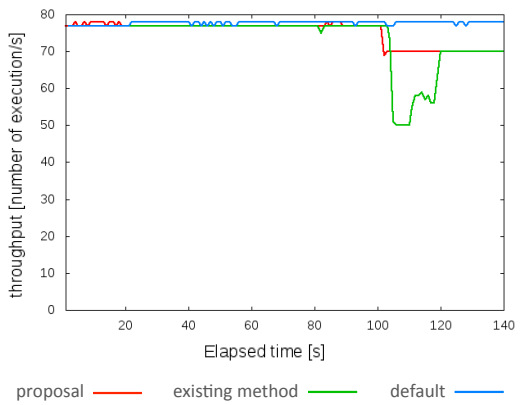

図 10 Case 2：移送先で稼働する VM 上の cubic_calc

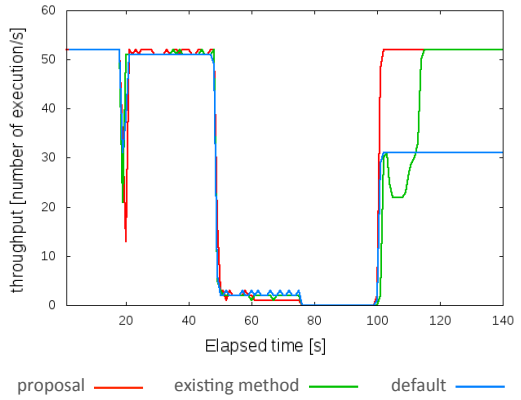

図 11 Case 3：移送 VM 上の malloc_access

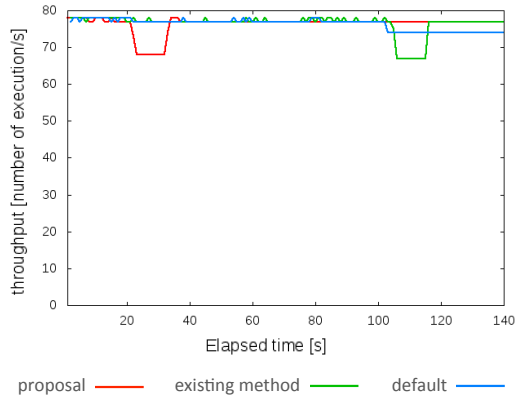

図 12 Case 3：移送先で稼働する VM 上の cubic_calc

parency on Multicore Systems, *2015 USENIX Annual Technical Conference (ATC)*, pp. 529–540 (2015).

[11] Cheng, L., Rao, J. and Lau, F. C.: vScale: Automatic and Efficient Processor Scaling for SMP Virtual Machines, *Eleventh European Conference on Computer Systems (EuroSys)*, pp. 2:1–2:14 (2016).

[12] Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I. and Warfield, A.: Live Migration of Virtual Machines, *2nd Conference on Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 273–286 (2005).

[13] NASA Advanced Supercomputing Division: NAS Parallel Benchmarks, `https://www.nas.nasa.gov/publications/npb.html`.

[14] Mao, Y., Morris, R. and Kaashoek, F.: Metis MapReduce Library, `https://pdos.csail.mit.edu/archive/metis/`.

[15] GraphAnalysis.org Team: HPC Graph Analysis, `http://www.graphanalysis.org/benchmark/`.

[16] PARSEC team: PARSEC benchmark suite, `http://parsec.cs.princeton.edu`.

[17] Beveridge, R., Bolme, D., Teixeira, M. and Draper, B.: CSU Face Identification Evaluation System, `https://www.cs.colostate.edu/cstop/`.

[18] Wanga, H., Isci, C., Subramanian, L., Choi, J., Qian, D. and Mutlu, O.: A-DRM: Architecture-aware Distributed Resource Management of Virtualized Clusters, *2011 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE)*, pp. 93–106 (2015).