VMが利用可能なCPU数の変化に対応した並列アプリケーション実行の最適化

高山 都旬子　光来 健一

クラウドコンピューティングの普及により，仮想マシン（VM）内で並列アプリケーションを動かすことが増えてきた．VM は別のホストにマイグレーションされることがあるが，CPU のオーバコミットのためにマイグレーション先のホストで物理 CPU が不足すると，VM にはマイグレーション前よりも少ない物理 CPU しか割り当てることができない．このとき，(1) 複数の VM 間で物理 CPU を共有する，(2) VM が利用できる CPU 使用率を制限する，(3) VM に割り当てる物理 CPU を減らす，という三つの対処が考えられる．しかし，我々の実験によると並列アプリケーションの性能は VM への CPU 割り当ての減少分以上に低下することが分かった．そこで本稿では，VM が利用可能な物理 CPU 数を見積もり，その情報を用いて並列アプリケーションの実行を最適化する pCPU-Est を提案する．pCPU-Est は仮想 CPU アフィニティを考慮して，各 VM ごとに利用可能な平均物理 CPU 数を算出する．pCPU-Est が提供する最適化手法は，アプリケーションスレッド数の最適化と仮想 CPU 数の最適化の二つである．物理 CPU の不足時の三つの対処に対してこれらの最適化手法を用いて実験を行ったところ，アプリケーションスレッドの最適化が最も効果的であることが分かった．

1　はじめに

近年，CPU のコア数の増加により，並列アプリケーションの利用が進んでいる．並列アプリケーションは処理をスレッドに分割して CPU の各コアに割り当て，同時に処理を行うことで高速化を図る．また，クラウドコンピューティングの普及により，クラウドが提供する仮想マシン（VM）の中で並列アプリケーションを動作させることも増えてきた．仮想化環境では，CPU のコア（物理 CPU）は仮想化され，複数の仮想的な CPU（仮想 CPU）として VM に提供される．物理 CPU が十分にある場合は，一つの物理 CPU は一つの仮想 CPU に割り当てられる．

しかし，仮想化環境では CPU のオーバサブスクリプションを行うせいで物理 CPU が足りなくなる場合がある．例えば，マイグレーションによって VM が他のサーバに移動され，移動先のサーバで複数の VM が動作する場合などである．十分な物理 CPU を確保できない時には，(1) 複数の VM 間で物理 CPU を共有，(2) VM が利用できる CPU 使用率を制限，(3) VM の物理 CPU 割り当てを削減，という対処が考えられる．しかし，我々の実験によると，物理 CPU の減少分以上にアプリケーションの性能が低下することが分かった．

そこで本稿では，VM が利用可能な物理 CPU 数を見積もり，その情報を用いて並列アプリケーションの実行を最適化する pCPU-Est を提案する．pCPU-Est は仮想 CPU アフィニティを考慮して，各 VM ごとに利用可能な平均物理 CPU 数を算出する．それを基に，pCPU-Est はアプリケーションスレッド数の最適化と仮想 CPU 数の最適化の二つの最適化手法を提供する．

物理 CPU の不足時の三つの対処についてこれらの最適化手法を用いて実験を行い，並列アプリケーションの性能への影響を調べた．その結果，最適なスレッド数は VM が利用可能な物理 CPU 数と同数であったが，最適な仮想 CPU 数はアプリケーションや利用可能な物理 CPU 数に依存することが分かった．また，スレッド数の最適化が最もアプリケーションの性能低下を抑えることができ，仮想 CPU 数の最適化

Tokiko Takayama, Kenichi Kourai, 九州工業大学, Kyushu Institute of Technology.

でもほぼ同様の効果が得られることが分かった．

以下，2 章で物理 CPU の不足時の対処について述べ，並列アプリケーションの性能低下に関する実験の結果を示す．3 章では pCPU-Est の提案を行い，4 章で pCPU-Est を用いて行った実験結果について述べる．5 章で関連研究について触れ，6 章で本稿をまとめる．

2 物理 CPU の不足による VM の性能低下

2.1 CPU 不足時の対処

仮想化環境では物理的な CPU コア（物理 CPU）は仮想化され，仮想的な CPU（仮想 CPU）として VM に提供される．図 1 のように物理 CPU が十分にある場合は，1 つの物理 CPU は 1 つの仮想 CPU にだけ割り当てられる．割り当て先の仮想 CPU はスケジューリングによって変わる可能性があるが，仮想 CPU アフィニティを設定することによって割り当てを固定することもできる．VM 上で並列アプリケーションが動いているとき，アプリケーションのスレッドはプロセススケジューラによって仮想 CPU に割り当てられて動作する．

仮想化環境では CPU のオーバサブスクリプションを行い，物理 CPU より多くの仮想 CPU を動作させることが多い．これはすべての仮想 CPU が物理 CPU を 100%使うとは限らないためである．一般に，サーバの CPU 使用率は 10〜20%程度と言われている．しかし，並列アプリケーションは CPU を占有することが多いため，オーバサブスクリプションを行うと物理 CPU が不足する場合がある．特に，VM がマイグレーションによって他のホストに移動されると，ハードウェア構成や同じホスト上で動作する他の VM のワークロードが大きく変化する．そのため，マイグレーション前は十分な物理 CPU を使えていても，マイグレーション後には物理 CPU が不足するという事態が起こる可能性がある．

ホストの物理 CPU が不足した場合，以下の三つの対処が考えられる．

- **複数の VM 間で物理 CPU を共有**
  物理 CPU が不足する場合には，図 2 のように 1 つの物理 CPU を複数の VM の仮想 CPU に割り当てるのが最も一般的である．各仮想 CPU が利用できる物理 CPU の割合は，VM に設定された重みによって決まる．例えば，2 つの VM の重みが 1:2 の場合には，33%と 67%の CPU 時間がそれぞれの仮想 CPU に配分される．ただし，一部の仮想 CPU が割り当てられた物理 CPU を使い切らない場合には，その分を他の仮想 CPU が利用する．
- **VM が利用できる CPU 使用率を制限**
  VM に CPU 使用率の上限を明示的に設定することによっても，複数の VM 間で物理 CPU を分け合うことができる．初期状態では VM は割り当てられた物理 CPU 数× 100%の CPU を使用できる．例えば，図 1 の場合では 400%が使用可能である．このときに CPU 使用率の上限を 200%に設定すると，図 3 のように CPU 使用率が合計で 200%を超えないように各物理 CPU を使用する．この例ではすべての物理 CPU を使用

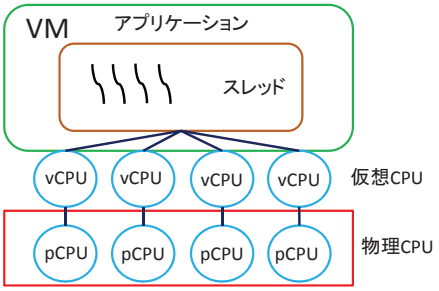


図 1　VM を用いた並列アプリケーション実行

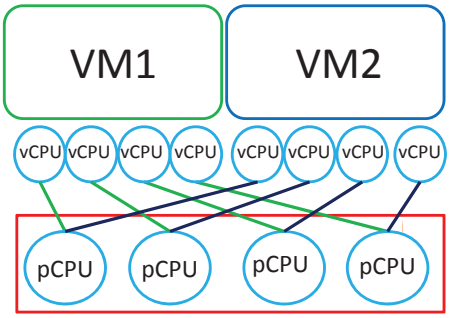


図 2　VM 間での物理 CPU の共有

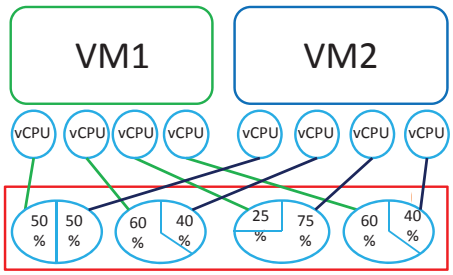


図 3　VM の CPU 使用率の制限

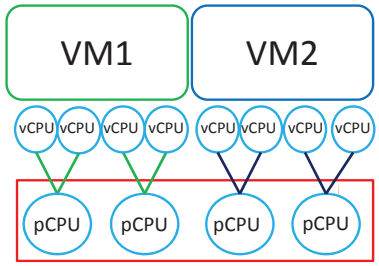


図 4　VM への物理 CPU 割り当ての削減

しているが，何個の物理 CPU を何%ずつ使用するかは仮想 CPU スケジューラによって決まる．

- **VM への物理 CPU 割り当てを削減**
  各 VM が割り当てられた物理 CPU を占有できるように，仮想 CPU アフィニティを利用して仮想 CPU 数より少ない物理 CPU を割り当てることもできる．この場合，VM の仮想 CPU 数は変わらないため，1 つの物理 CPU が 1 つの VM の複数の仮想 CPU に割り当てられる．図 1 の状態から割り当てる物理 CPU を半分に減らすと，図 4 のように 2 つの仮想 CPU で 1 つの物理 CPU を分け合うことになる．

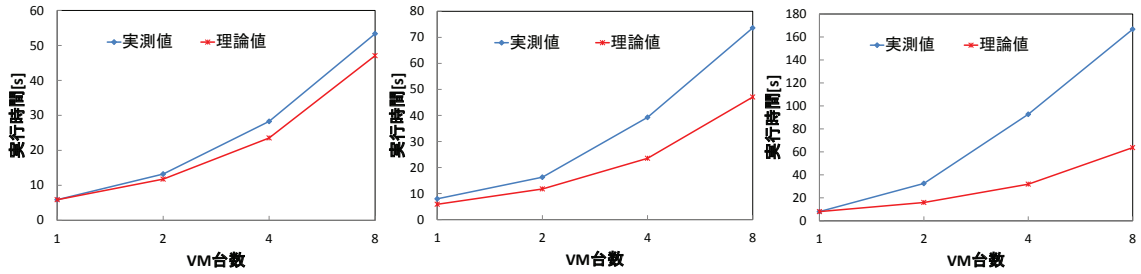
2.2　並列アプリケーションの性能低下

物理 CPU が不足した場合の三つの対処を行ったときに，VM 上で動く並列アプリケーションの性能の変化について調査した．この実験には，2 基の AMD Opteron 6376（16 コア），320GB のメモリを搭載したマシンを用いた．仮想化ソフトウェアには Xen 4.4.0 を使用し，16 個の仮想 CPU と 4GB のメモリを持つ VM を用いた．対象とした並列アプリケーションは，Tascell [2] を用いて並列化されたフィボナッチ数計算 (fib) と NAS Parallel Benchmarks [5] の EP と BT である．

並列アプリケーションは仮想 CPU 数と同数の 16 スレッドで実行した．複数台の VM で物理 CPU を共有する場合，VM の台数を 1 台から 8 台まで増加させて実行時間を計測した．各 VM に設定する重みは同じとし，VM 内では同じ並列アプリケーションを動作させた．CPU 使用率を制限する場合は，Xen の Credit スケジューラの cap の値を変更することにより CPU 使用率を 1600%から 100%まで減少させた．物理 CPU の割り当てを削減する場合には，VM に割り当てる物理 CPU を 16 個から 1 個まで減らした．後者の二つの対処を行う場合には VM は 1 つだけ動作させた．物理 CPU を 16 個占有した状態を初期状態として，物理 CPU の減少分に比例して低下することを仮定したアプリケーションの性能を理論値とした．

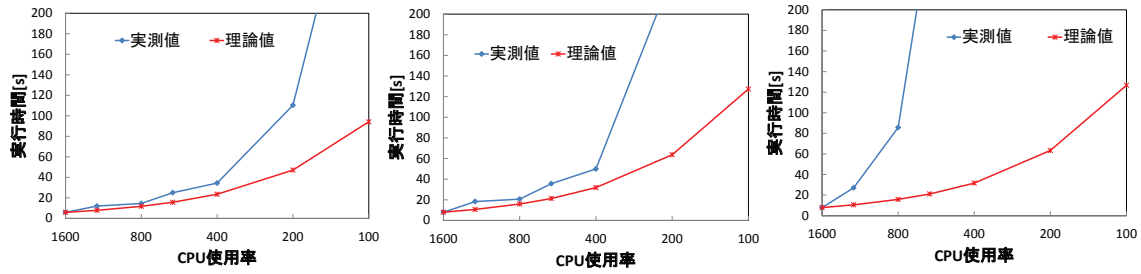
実験に用いたすべてのアプリケーションにおいて，どの対処を行った場合でも理論値より性能は低下した．物理 CPU を共有した場合の結果を図 5 に示す．fib，EP，BT の順に理論値と実測値の差が大きくなっており，fib では最大で 20%，EP では 23%，BT では 191%の性能低下が起きた．物理 CPU を多くの VM で共有するほど性能低下が大きくなった．図 6 に示すように，CPU 使用率を制限した場合ではその傾向がより顕著に現れた．fib では 23%から 328%，EP では 30%から 373%，BT では 157%から 4138%の性能低下が起きた．一方，物理 CPU の割り当てを削減した場合の結果は図 7 のようになり，fib では最大で 25%，EP では 27%，BT では 78%と性能低下が小さかった．

理論値よりも性能が低下する原因として，仮想 CPU スケジューリングのオーバヘッドが考えられる．物理 CPU が不足するとスケジューリングによって仮想 CPU を切り替える必要があるため，物理 CPU の減少分以上の性能低下が起きた可能性がある．また，ロックホルダ・プリエンプション [8] や vCPU スタッキング [7] の発生も原因の一つと考えられる．ロック



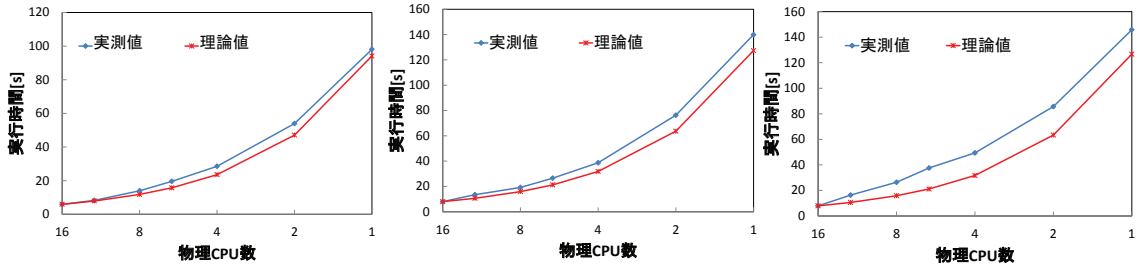
(a) fib　　(b) EP　　(c) BT

図 5　物理 CPU 共有時の性能低下



(a) fib　　(b) EP　　(c) BT

図 6　CPU 使用率の制限時の性能低下



(a) fib　　(b) EP　　(c) BT

図 7　物理 CPU の割り当て削減時の性能低下

ホルダ・プリエンプションは，ロックを保持している仮想 CPU に割り当てられた物理 CPU が他の仮想 CPU に奪われることで処理が遅延する現象である．vCPU スタッキングは，ロックを待っている仮想 CPU がロックを保持している仮想 CPU よりも先に物理 CPU を割り当てられたために処理が進められない現象である．

3　pCPU-Est

本稿では，VM が利用可能な物理 CPU 数を見積もり，それに基づいて仮想 CPU の切り替えを減らすための最適化を行う pCPU-Est を提案する．これにより，仮想 CPU の切り替えオーバヘッドの削減やロックホルダ・プリエンプションや vCPU スタッキング

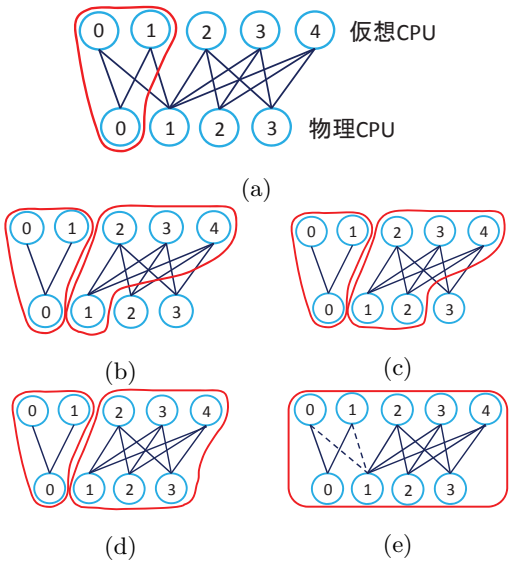


図 8　物理 CPU と仮想 CPU の分割例

の解消もしくは緩和が可能となり，物理 CPU の不足時に並列アプリケーションを最適に実行することができる．

3.1　物理 CPU 数の見積もり

pCPU-Est は物理 CPU と仮想 CPU をいくつかのグループに分け，各仮想 CPU が利用できる物理 CPU の割合を算出する．そして，それを基に各 VM が利用できる平均物理 CPU 数を見積もる．このとき，同じグループ内のすべての仮想 CPU は，VM に設定された重みが同じであれば同じ割合の物理 CPU を利用することができるようにする．物理 CPU の割り当てに制約がなければ物理 CPU と仮想 CPU は全体で 1 つのグループを形成するが，仮想 CPU アフィニティが設定されていると複数のグループに分かれる可能性がある．明示的にアフィニティを設定していない場合でも，仮想化システムによって暗黙のうちに NUMA アフィニティが設定されることもある．NUMA アフィニティは NUMA を考慮した CPU 割り当てである．

pCPU-Est は図 8 のような物理 CPU と仮想 CPU のグラフを用いてグループを作成する．物理 CPU と仮想 CPU は頂点となり，物理 CPU が仮想 CPU に割り当てられている場合にそれらは辺でつながれる．いくつかの物理 CPU からいくつかの仮想 CPU へのアフィニティが設定されている場合，各物理 CPU はそれらの仮想 CPU すべてにつながれる．図 8(a) は，物理 CPU 0, 1 から仮想 CPU 0, 1，および，物理 CPU 1, 2, 3 から仮想 CPU 2, 3, 4 へのアフィニティが設定された場合の例である．

pCPU-Est のグラフ分割アルゴリズムは以下のようになる．まず，グラフ上でつながっている仮想 CPU 数が最小の物理 CPU を選択し，その物理 CPU とそれからつながっている仮想 CPU からなるグループを作成する．これは，グループの $\frac{\text{物理 CPU 数}}{\text{仮想 CPU 数}}$（物理 CPU の割合）がなるべく大きくなるようにするためである．次に，そのグループからつながっている物理 CPU のうち，つながっている仮想 CPU 数が最小の物理 CPU を選択し，その物理 CPU とそれからつながっている仮想 CPU を含めたより大きなグループを作成する．このグループの物理 CPU の割合が元のグループより大きいか等しければ新しいグループを採用する．これを繰り返してグループをできるだけ大きくする．新しいグループの物理 CPU の割合のほうが小さくなった場合は，グループからその外側につながっている辺を削除する．

グラフの残りの部分に対してこの作業を繰り返し，グラフ全体がいくつかのグループに分割されたら，作成されたグループをできるだけマージする．これは，局所的にグループ分けを行うだけでは正しいグラフ分割ができないためである．まず，削除された辺でつながれていた 2 つのグループを選び，その辺の仮想 CPU 側をグループ 1，物理 CPU 側をグループ 2 とする．グループ 2 の物理 CPU の割合のほうが大きいか等しければ，2 つのグループをマージする．これを削除された辺でつながれていたすべてのグループについて繰り返す．

このアルゴリズムを図 8 に適用すると以下のようになる．

(1)　図 8(a) では，物理 CPU 0 につながっている仮想 CPU 数が最小の 2 であるので，これらでグループを作成する．

(2)　(1) で作成されたグループの物理 CPU の割合が $\frac{1}{2}$ であるのに対し，物理 CPU 1 と仮想 CPU 2, 3, 4 を追加した新しいグループの物理 CPU の

割合は $\frac{2}{5}$ と小さくとなるため，物理 CPU 1 と仮想 CPU 0, 1 をつなぐ辺を削除する．

(3) 残った物理 CPU の中で，つながっている仮想 CPU 数が最小であるのは物理 CPU 1 なので，物理 CPU 1 と仮想 CPU 2, 3, 4 で図 8(b) のようなグループを作成する．このグループからつながっている物理 CPU の中で，つながっている仮想 CPU 数が最小であるのは物理 CPU 2 なので，図 8(c) のように物理 CPU 2 を追加したグループを作成する．元のグループの物理 CPU の割合が $\frac{1}{3}$ であるのに対し，新しいグループは $\frac{2}{3}$ と大きくなるため，新しいグループを採用する．

(4) 同様にして，図 8(d) のように物理 CPU 3 を追加したグループを作成すると物理 CPU の割合が 1 となるので，新しいグループを採用する．

(5) 作成された 2 つのグループは削除された辺（図 8(e) の点線）でつながれていたため，マージできないかをチェックする．最初に作成されたグループは削除された辺の仮想 CPU 側であるためグループ 1 となり，次に作成されたグループは物理 CPU 側であるためグループ 2 となる．グループ 1 の物理 CPU の割合は $\frac{1}{2}$ であるのに対し，グループ 2 は 1 と大きいため，これらのグループをマージして 1 つのグループを作成する．

グラフ分割アルゴリズムによって作成されたグループ内で，VM に設定された重みに応じて物理 CPU を仮想 CPU に均等に分配する．図 8 の場合，VM の重みが同じであれば各仮想 CPU に分配される物理 CPU の割合は $\frac{4}{5}$ となる．次に，それぞれの VM の持つ仮想 CPU に分配される物理 CPU をすべて足し合わせる．この値が小数となる場合は小数点以下を切り上げ，その値を VM が利用可能な物理 CPU 数とする．例えば，図 8 の仮想 CPU 0, 1 が 1 つの VM に属する場合，その VM が利用可能な物理 CPU 数は 2 となる．

3.2 最適化手法

pCPU-Est が提供する並列アプリケーションの実行最適化は以下の二つである．

- **アプリケーションのスレッド数の最適化**
  並列アプリケーションが用いるスレッド数を VM が利用可能な物理 CPU 数に応じて調整することで，並列度をなるべく高く保ちつつ，仮想 CPU の切り替えを減らす．スレッド数はユーザが容易に調整することが出来るが，実行中に変更可能かどうかはアプリケーション依存である．
- **VM の仮想 CPU 数の最適化**
  VM の持つ仮想 CPU 数を利用可能な物理 CPU 数に応じて調整にすることで仮想 CPU の切り替えを減らす．VM の仮想 CPU 数を変更することはシステムの管理者にのみ可能であるため，VM のユーザがこの最適化を行うことはできない．

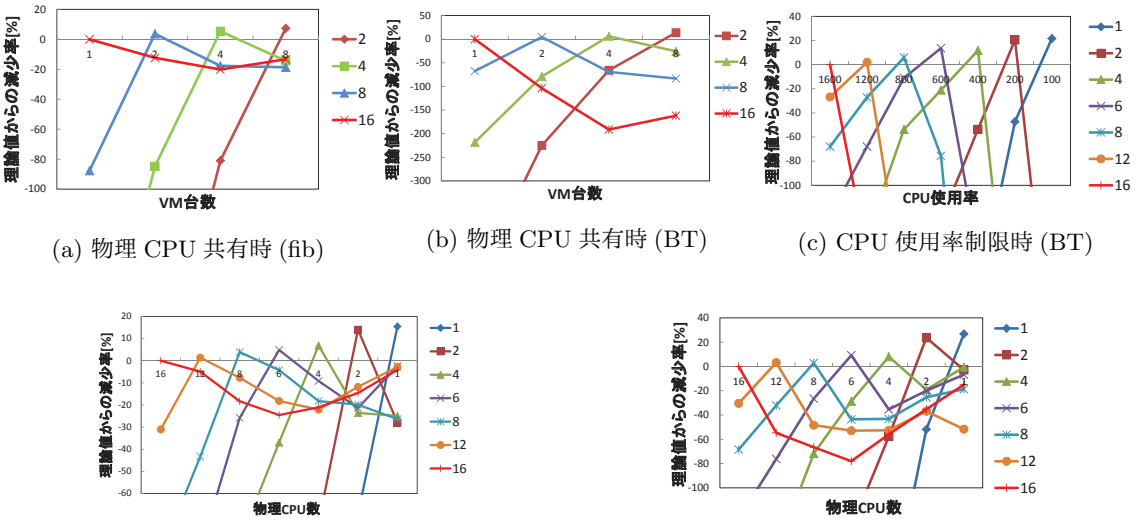
4 実験

pCPU-Est の最適化手法における最適なスレッド数および最適な仮想 CPU 数を調査し，CPU が不足した場合の三つの対処について最適化の効果を調べた．この実験では，2.2 節と同じ実験環境，同じ並列アプリケーションを用いた．

4.1 最適なアプリケーションスレッド数

VM が利用できる物理 CPU 数と最適なアプリケーションスレッド数の関係を調べる実験を行った．物理 CPU の不足時の三つの対処を行った際に様々なスレッド数でアプリケーションを実行し，その実行時間を計測した．実測値の理論値からの減少率を図 9 に示す．物理 CPU を共有した場合には fib，BT ともに $\frac{16}{\text{VM の台数}}$ が最適なスレッド数となった．CPU 使用率を制限した時には CPU 使用率に対応する物理 CPU 数と同じスレッド数にしたときが最適になり，物理 CPU を削減した場合にも物理 CPU 数と同数のスレッド数が最適という結果になった．このことから，最適なスレッド数は VM が利用可能な物理 CPU 数と同数であることが分かった．

また，最適なスレッド数で実行したアプリケーション性能は理論値よりもよくなることが分かった。さらに、VM に割り当てる物理 CPU 数が少なくなるほど、理論値からの減少率が大きくなることも分かった。これはスレッド数を減らすことによるスケジューリングオーバヘッドの削減が原因ではないかと考えら



(a) 物理 CPU 共有時 (fib)　(b) 物理 CPU 共有時 (BT)　(c) CPU 使用率制限時 (BT)

(d) 物理 CPU 削減時 (fib)　(e) 物理 CPU 削減時 (BT)

図 9　スレッド数がアプリケーション性能に及ぼす影響

れる。

4.2　最適な仮想 CPU 数

VM が利用可能な物理 CPU 数と最適な仮想 CPU 数の関係を調べる実験を行った．図 10 にアプリケーションの実行時間の理論値からの減少率を示す．fib では三つの対処のいずれの場合でも，VM が利用できる物理 CPU 数と同数の仮想 CPU 数が最適となった．一方，BT ではある程度その傾向が見られたものの，異なる仮想 CPU 数が最適となる場合もあった．例えば，CPU 使用率を 1200%〜600%に制限した場合の最適な仮想 CPU 数は対応する 12〜6 ではなく，それよりも大きな値になった．また，物理 CPU 数を 12〜6 に削減した場合も，最適な仮想 CPU 数はそれよりも大きな値となった．この結果より，最適な仮想 CPU 数は VM が利用可能な物理 CPU 数と同数であることが多いが，アプリケーションや物理 CPU の不足量にも依存することが分かった．

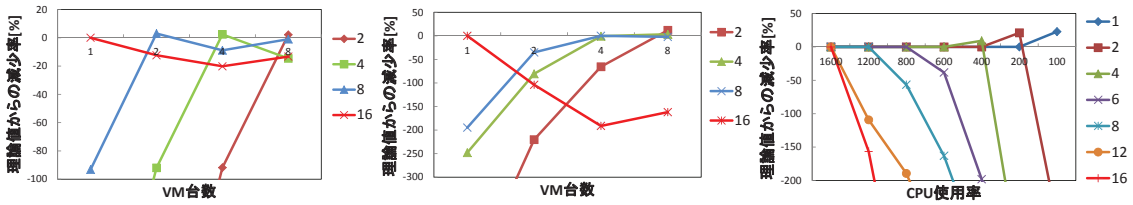
4.3　最適化の効果

pCPU-Est の二つの最適化手法を物理 CPU の不足時の三つの対処に適用し，どちらがアプリケーションの性能をより改善することができるかを調べた．この実験では，最適なスレッド数と最適な仮想 CPU 数は物理 CPU 数と同数と仮定した．

物理 CPU を共有した場合，最も性能低下を抑えることができたのは図 11 よりスレッド数の最適化であることが分かる．仮想 CPU の最適化でもほぼ同等の効果が見られたが，VM が 8 台の時の fib や 2 台の時の BT でスレッド数の最適化ほどの性能改善は見られなかった．ただし，いずれの場合でも最適化を行わない場合よりは大幅に性能を改善できた．

CPU 使用率を制限した場合でも，図 12 のようにスレッド数の最適化と仮想 CPU の最適化は似た傾向を示した．CPU 使用率を 100%に制限した時の fib と 1200%〜600%に制限した時の BT ではスレッド数の最適化のほうが効果があった．後者については，VM が利用可能な物理 CPU 数が最適な仮想 CPU 数ではないためだと考えられる．

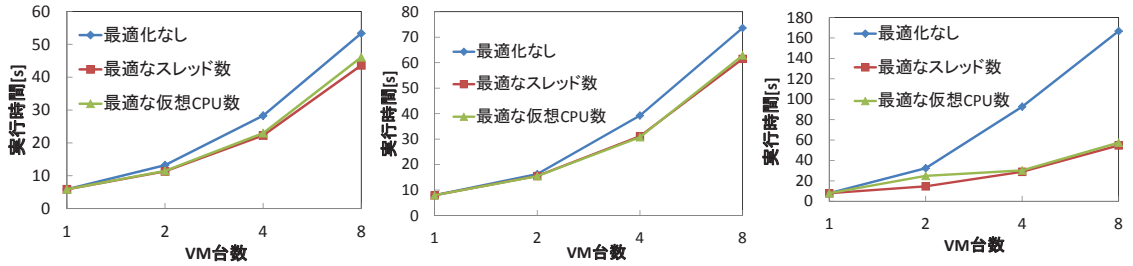
物理 CPU を削減した場合でも，図 13 に示すように同様の結果になった．物理 CPU 数が 12 の場合の BT では，仮想 CPU 数の最適化を行わなかった場合より性能が悪化した．これも仮想 CPU 数が最適になっていないためと考えられる．



(a) 物理 CPU 共有時 (fib)　(b) 物理 CPU 共有時 (BT)　(c) CPU 使用率制限時 (BT)

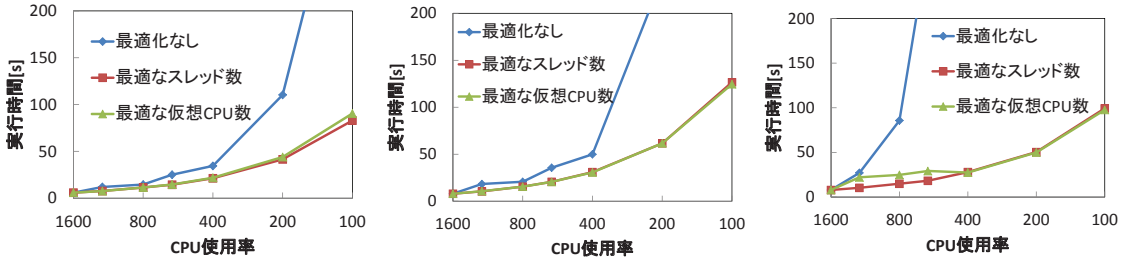
(d) 物理 CPU 削減時 (fib)　(e) 物理 CPU 削減時 (BT)

図 10　仮想 CPU 数がアプリケーション性能に及ぼす影響



(a) fib　(b) EP　(c) BT

図 11　物理 CPU の共有時の最適化

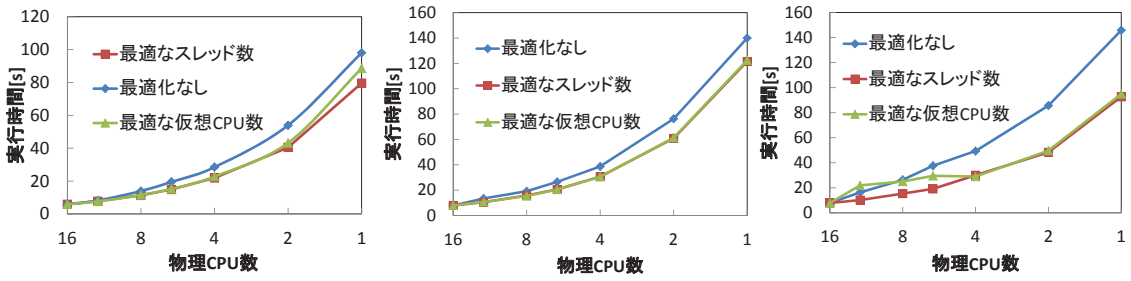


(a) fib　(b) EP　(c) BT

図 12　CPU 使用率の制限時の最適化

5　関連研究

VCPU-Bal [6] は VM の仮想 CPU 数を動的に増減させる vCPU バルーニングを用いて，ロックホルダ・プリエンプション [8] や vCPU スタッキング [7] を防ぐ．これらの問題は，仮想 CPU とアプリケーションスレッドの二重のスケジューリングによって生じるため，VCPU-Bal では仮想 CPU が物理 CPU に一対一に割り当てられるように VM の仮想 CPU 数を調整する．各 VM の仮想 CPU 数は VM の重みに応じて



(a) fib　(b) EP　(c) BT

図 13　物理 CPU の削減時の最適化

決定される．一方，pCPU-Est では仮想 CPU アフィニティも考慮して VM の仮想 CPU 数を決定することができる．FlexCore [4] は VCPU-Bal を実装したシステムであり，仮想 CPU の競合の検出，VM とハイパーバイザ間の効率のよい通信，仮想 CPU のホットプラグを実現している．

vScale [1] は VM の仮想 CPU 数を VCPU-Bal よりもきめ細かく調整することを可能にしている．VCPU-Bal と同様に，vScale も基本的には VM の重みに応じて仮想 CPU 数を決定する．しかし，VM が割り当てられた物理 CPU を使い切らなかった場合，その VM の仮想 CPU 数を減らし，他の VM の仮想 CPU 数を増やす．このような頻繁な仮想 CPU 数の変更を可能にするために，vScale では OS が迅速に仮想 CPU を再構成する機構を提供する．VCPU-Bal や vScale では複数の VM が物理 CPU を共有することを想定しているが，pCPU-Est では CPU 使用率を制限したり物理 CPU 割り当てを削減したりすることも想定している．

コスケジューリング [9] は VM のすべての仮想 CPU を同時に物理 CPU にスケジュールすることにより，ロックホルダ・プリエンプションなどの問題を解決する．しかし，必要な数の物理 CPU がそろうまでスケジューリングを行うことができず，高い優先度を持った仮想 CPU が低い優先度の仮想 CPU より後にスケジュールされる可能性もある．バランススケジューリング [7] は VM の各仮想 CPU を異なる物理 CPU にスケジュールすることで vCPU スタッキングを防ぐが、ロックホルダ・プリエンプションを防ぐことはできない．

VM の I/O 性能を改善するために，仮想 CPU スケジューリングのタイムスライスを調整する手法が提案されている．vSlicer [11] は I/O レイテンシに敏感な VM に短いタイムスライスを割り当てることにより，仮想 CPU をより速く横取りできるようにして I/O 処理を高速化する．しかし，VM は一般に様々な処理を行うため，VM 単位での割り当ては難しい．この問題を解決するために，MDPVS [3] は実行時の I/O 情報を用いて，I/O 処理にだけ短いタイムスライスを割り当てた物理 CPU を用いる．vTurbo [10] では各 VM に I/O 処理専用の仮想 CPU を用意し，そのタイムスライスだけを短くする．しかし，タイムスライスを短くするとスケジューリングのオーバヘッドが増加する．

6　まとめ

本稿では，物理 CPU が不足した時に並列アプリケーションの実行を最適化する pCPU-Est を提案した．仮想化環境では物理 CPU が不足した場合に様々な対処が行われるが，物理 CPU の減少分以上に並列アプリケーションの性能が低下することが分かった．pCPU-Est は仮想 CPU アフィニティを考慮して VM が利用可能な物理 CPU 数を見積もり，アプリケーションスレッド数の最適化および仮想 CPU 数の最適化の二つの最適化手法を提供する．実験の結果，スレッド数の最適化が最も効果的であることが分かった．

今後の課題は，pCPU-Est を用いて見積もった物理 CPU 数でどこまで最適な実行が可能かを調べるこ

とである．別の最適化手法として，Linux の cgroups を用いてアプリケーションが利用する CPU を制限する手法についても検討する予定である．また，1 つの VM の仮想 CPU 数を最適化するとシステム全体に影響が及ぶため，それに合わせて他の VM の仮想 CPU 数も最適化できるようにする必要がある．さらに，VM の仮想 CPU 数を減らしても利用可能な物理 CPU 数が減らないようにするスケジューリング機構も必要である．

参 考 文 献

[1] Cheng, L., Rao, J., and Lau, F.: vScale: Automatic and Efficient Processor Scaling for SMP Virtual Machines, *Proceedings of the 11th European Conference on Computer Systems*, 2016.

[2] Hiraishi, T., Yasugi, M., Umatani, S., and Yuasa, T.: Backtracking-based Load Balancing, *Proceedings of the 14th Symposium on Principles and Practice of Parallel Programming*, 2009, pp. 55–64.

[3] Hu, Y., Long, X., Zhang, J., He, J., and Xia, L.: I/O Scheduling Model of Virtual Machine Based on Multi-core Dynamic Partitioning, *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, 2010, pp. 142–154.

[4] Miao, T. and Chen, H.: FlexCore: Dynamic Virtual Machine Scheduling Using VCPU Ballooning, *Tsinghua Science and Technology*, Vol. 20, No. 1(2015), pp. 7–16.

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

[6] Song, X., Shi, J., Chen, H., and Zang, B.: Schedule Processes, not VCPUs, *Proceedings of the 4th Asia-Pacific Workshop on Systems*, 2013.

[7] Sukwong, O. and Kim, H. S.: Is Co-scheduling Too Expensive for SMP VMs?, *Proceedings of European Conference on Computer Systems*, 20011, pp. 257–272.

[8] Uhlig, V., LeVasseur, J., Skoglund, E., and Dannowski, U.: Towards Scalable Multiprocessor Virtual Machines., *Proceedings of the 3rd Virtual Machine Research and Technology Symposium*, 2004, pp. 43–56.

[9] VMware, Inc.: Co-scheduling SMP VMs in VMware ESX Server, 2008.

[10] Xu, C., Gamage, S., Lu, H., Kompella, R., and Xu, D.: vTurbo: Accelerating Virtual Machine I/O Processing Using Designated Turbo-sliced Core, *Proceedings of the 2013 USENIX Annual Technical Conference*, 2013, pp. 243–254.

[11] Xu, C., Gamage, S., Rao, P. N., Kangarlou, A., Kompella, R. R., and Xu, D.: vSlicer: Latency-aware Virtual Machine Scheduling via Differentiated-frequency CPU Slicing, *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing*, 2012, pp. 3–14.