Vol.2009-MPS-76 No.11
Vol.2009-BIO-19 No.11
2009/12/17

仮想マシン再配置問題に対する厳密アルゴリズム

田添　聡士[†1]　　福永　アレックス[†2]

近年, データセンターでは複数のアプリケーションを仮想マシン上で稼働させることで少数の物理サーバに機能を集約して管理を行っている. 各アプリケーションの想定 CPU 使用率は変動するため, 時に物理サーバの CPU リソースをアプリケーションの想定 CPU 使用率の和が越えてしまう場合がある. その場合, 仮想マシンの幾つかを物理サーバ間で移動させることで問題を解決する必要があるが, 無闇に仮想マシンを移動させていては余計なコストやリスクを伴ってしまう. よって, 移動に掛かるコストを極力低く抑えるためのアルゴリズムが必要になる.

本研究ではこの移動コストを最小化することを目的とした組み合わせ最適化問題「仮想マシン再配置問題」に対する厳密アルゴリズムの提案と結果の分析を行った.

An Exact Algorithm for the Min-Cost Virtual Machine Reassignment probrem

Satoshi Tazoe[†1] and Alex S. Fukunaga[†2]

In modern data centers,virtual machines that provide various services are assigned to physical servers. Due to fluctuations in demand of CPU resources, it is sometimes necessary to migrate virtual machines between physical servers in order to eliminate overloading CPU resources. We consider an exact algorithm for finding the minimal number of VMs which need to be moved in order to eliminate overloading.

†1 東京工業大学　数理・計算科学専攻
Dept. of Computer Science, Tokyo Institute of Technology
†2 東京工業大学　グローバルエッジ研究院
Global Edge Institute, Tokyo Institute of Technology

1. はじめに

10 年ほど前まで, 多くの企業ではサーバは各部門が分散して管理すべきと考えられていた. 背景にはサーバの低価格化やネットワークが低速かつ高価であったこと, またセキュリティーの面でも分散管理した方が安全であると考えられていたことなどが挙げられる. しかし, 近年, 分散して配置するために購入した大量のサーバの管理費が IT 投資の 7〜8 割を占めてしまっている事が問題になっている. 一方でサーバのマルチコア化や高性能化に伴い各部門のローカルな使用では有効に CPU のリソースを使いこなせていない場合が多い.

この様な背景から, 高性能なサーバと仮想化技術を用いる事で, 多数のサーバの機能を少数の高性能サーバに集約し, データセンターなどで一元管理をするサーバ統合が盛んに行われ, 現在では多くのサーバが仮想マシン（Virtual Machine, 以下 VM）上で稼働している[1)].

サーバ統合の作業としてまず問題となるのは, 多数の VM をいかに効率よく実際の物理サーバに割り当てるかという導入に関する問題である[3)]. これは各 VM の想定 CPU 使用率をアイテム, サーバのリソースの量をビンと捉えると組み合わせ最適化問題として有名なビン・パッキング問題と捉えることができる（実際には CPU の他にもメモリなど他の指標も考えなければいけないのでベクトル・パッキング問題を解くことになる）.

また, 実際に物理サーバー上で VM を稼働させ運用していく際にも様々な問題がある. 例えば, 定期的な使用率の見直しや, アプリケーションのアップデートなどにより運用している VM のいくつかの想定 CPU 使用率が変化した結果, 物理サーバー全体の想定 CPU 使用率が閾値を越え, 運用に支障をきたす恐れがある. この問題に対し, 新たに物理サーバを購入して VM を移すというのも一つの手ではあるが多大な費用が掛かるため多くの場合には, 運用している他の物理サーバの中で比較的低負荷なサーバに VM の幾つかを移したり, 使用率の低い VM と高い VM とを交換する事で既存の物理サーバ内で問題を処理しようと考える. その時, 生じた VM の移動をコストとして捉え, コストを最小にする VM の移動方法を求めるのが「仮想マシン再配置問題」である[4),5)].

つまり, 仮想マシン再配置問題は幾つかのサーバにおいて割り当てられている VM の CPU 使用率の和がサーバの閾値をこえてしまっている状況が初期状態として与えられ

(1)　どのサーバも割り当てられた VM の CPU 使用率の和が自身の閾値内に収まっている状況に

(2)　いかに少ない数の VM の移動で移ることができるか

を求める問題である.

Vol.2009-MPS-76 No.11
Vol.2009-BIO-19 No.11
2009/12/17

本研究では仮想マシン再配置問題に対する既存の厳密アルゴリズム[5] で用いられていた探索における下界の計算法方の改善と, ドメイン縮小という問題非依存の手法の提案を行った. 以後,2 章では仮想マシン再配置問題の説明と定式化を行う.3 章では用いた探索空間と探索アルゴリズムについて説明する. そして 4 章では枝刈に用いる下界の計算方法について,5 章ではドメイン縮小について解説をし,6 章で実験結果を示す.

2. 仮想マシン再配置問題

仮想マシン再配置問題は多くの VM を少数の高性能サーバ上で運用する際に, アップデートや VM の CPU 使用率の見直し等によって起こる問題に注目して考えられた. 実際にはそれらの変化によっていくつかの物理サーバにおいて, 割り当てられている VM の想定 CPU 使用率の合計がサーバに設定された使用率の閾値を越えてしまった場合に, どうすれば最も少ない VM の移動でそのような異常を解決できるか求める問題である.

定式化を行うと以下のようになる. 今,n 個の VM と m 個の物理サーバがあるとする.$\mathrm{VM}_i(1 \leq i \leq n)$ の想定 CPU 使用率を c_i とし,$\mathrm{Server}_j(1 \leq j \leq m)$ のリソースを R_j とする. 初期状態 $A = \{a_{ij}\}$ は VM_i が Server_j に割り当てられていた場合は 1 それ以外は 0 となっている. また $X = \{x_{ij}\}$ も割り当てを表し,0,1 の意味合いは A と同じである. 以下の制約を満たすような X に対し $1/2\left(\sum_{i=1}^{n}\sum_{j=1}^{m}|x_{ij} - a_{ij}|\right)$ が割り当て A から X に移る際に移動した VM の数を意味している. よってこれを最小化することが仮想マシン再配置問題の目的関数となる.

$$\text{Minimize}: \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{m}|x_{ij} - a_{ij}|$$

$$\text{Subject to}: \sum_{i=1}^{n} c_i x_{ij} \leq R_j \qquad (1 \leq j \leq m)$$

$$\sum_{j=1}^{m} x_{ij} = 1 \qquad (1 \leq i \leq n)$$

$$x_{ij} \in \{0, 1\}$$

仮想マシン再配置問題にや類似した問題にたいして対してヒューリスティクスや近似アルゴリズムが考えられているが[2),4)], 本研究では厳密解を求めることを目標とした. それにはいくつか理由がある.

(1) 実際に運用されている問題サイズが物理サーバ 5 台から 20 台程度,VM 数 20 個から 100 個程度で, 純粋な組み合わせ最適化問題としては比較的小さい規模で運用されていることが多く, 厳密解を求めることが十分可能であると判断したため.

(2) 扱っているアプリケーションが自社のものではなく他の企業・機関のものをアウトソーシングなどの形で代わりに保守している場合, クライアントとしては無闇に動かしてもらいたくはない. その為, 移動が最小となることが保証される厳密アルゴリズムが有ることがクライアントへの説明のために有効である.

(3) 仮に許容時間内に解が求まらなかったとしても, 問いに対する下界を得ることができるため, その下界を他のヒューリスティクスで求めた解の性能の指標として使うことができる[8].

次章からは, 実際にどのように厳密アルゴリズムを設計したのかについて説明を行う.

3. 探索空間・探索アルゴリズム

本研究のアルゴリズムでは探索には木探索アルゴリズムを用いた. 厳密解アルゴリズムには他にも整数計画法や虱潰しに探す等様々な方法が考えられる. 特に仮想マシン再配置問題については少し工夫すれば整数計画問題のかたちに直すことが可能で IP ソルバーを使って解くことができる. しかし, 前章で定式化したものをそのまま使って解くことは簡単な実験の結果から非常に困難であるとみなし本研究では扱っていない.

木探索を効果的に行う上で大切になることは

(1) 探索空間の構築方法

(2) 探索の方針 (アルゴリズム)

(3) 枝刈りに用いる下界の性能

の 3 点である. 本章では (1) 探索空間 Commitment Space 探索木と (2) 探索アルゴリズム IDA*について説明を行う. そして 4 章以降で (3) に関する研究成果と,(1)(2) にまたがるドメイン縮小という手法について解説する.

探索空間の説明に入る前にノードをどのように表現するか説明する. ノード n は $n = (5, 10, \underline{12}^{*})_{25}(10, \underline{9})_{25}$ のように括弧で区切られいて, 中には下線やアスタリスク（＊）が打たれた数が並んでいる. 括弧は物理サーバを表しており, 添字がそのサーバの CPU リソースの量となる. 中に並んでいる数字はそれぞれ一つの VM を表しており, 数字の大きさが VM に割り当てられるべきリソースの量である. また下線が引かれている VM は「コミット」されており, 木探索においてこのノードの下に現れるノードでは下線の引かれた VM は

Vol.2009-MPS-76 No.11
Vol.2009-BIO-19 No.11
2009/12/17

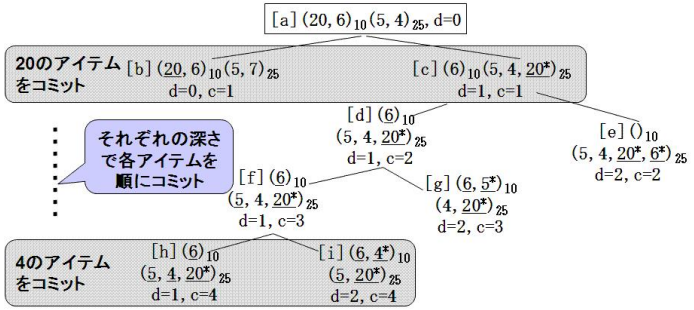

図 1 Committed Space Search Tree

常に同じサーバに割り当てられていることを意味している. またアスタリスク（∗）が打たれた VM は初期状態と異なるサーバにコミットされてたことを意味し, すべてがコミットされた状態ではこのアスタリスクの総数がその VM の割り当て方に対するコストと一致する. 上の例では大きさが 12 と 9 の VM が現状コミットされており, 特に 12 の VM は初期状態とは別のサーバにコミットされコストとしてカウントされると理解することができる.

3.1 探索空間：Commitment Space 探索木

Commitment Space 探索木はコミットされている VM の数が深さの基準となる探索木である. ルートとなるのはすべての VM がコミットされていない状態である. 次に深さ 1 の地点には, 一つ VM を選びその VM がそれぞれのサーバにコミットされた状態が子ノードとして並ぶ. 以後同様に深さが深くなるごとにコミットされている VM の数が多くなり深さ d の地点では d 個の VM がコミットされていることとなる. 図 1 の場合, 先ず初めに大きさ 20 の VM が各サーバにコミットされその次に大きさ 6 の VM がコミットされ, 最終的に大きさ 4 の VM がコミットされて, すべてが範囲内に収まった答え (i) を一つ得る事が出来た. この探索空間が他の探索空間に対して優位である事が実証されている[5]. そのため, 実験は Commitment Space 探索木のみを用いて行った.

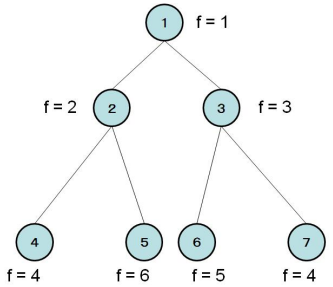

図 2 IDA*

3.2 探索アルゴリズム：IDA*

IDA*は Korf[6] に提案されたアルゴリズムで, 分枝限定法の一種である.IDA*の特徴は, 通常の深さ優先探索で分枝限定法を用いる際には一つ目の解が見つかるまで上界は設定されないのに対し,IDA*ではあらかじめ上界を定め探索範囲を限定して探索を行うことと, コスト関数 f を用いるという点である. コスト関数 f とは確定済コスト g と, 現状から更に掛かる可能性のある不確定コストの下界 h の和で表される.

IDA*ではまず Upper Bound(以下 UB) を 0 に設定し, 答えが見つかるまで以下を繰り返す.$f(n) \leq$ UB の範囲で深さ優先探索を行う. そしてもし探索に失敗した場合は UB を探索中にコスト関数が UB 以上になってしまったノードの中でもっとも小さかったコスト関数の値を新たな UB に設定し, 再度深さ優先探索を行う. 答えが見つかった場合はその時点で探索を終了する. 各ノードのコスト関数が図 2 のようになっていた場合, 先ず初めは UB が 0 であるためノード 1 で探索を止めて UB を 1 に変更する. 次はノード 1,2,3 まで探索して同じく終了する.UB を 2 に変更してノード 1,2,4,5,3 と探索を行い, そして UB を 3 に更新してノード 1,2,4,5,3,6,7 を探索, と答えが見つかるまで UB を変化させ探索を繰り返し行う.

IDA*は通常の深さ優先探索と違い, 徐々に探索範囲を広げていくため初めに見つかった答えが問題の最適解であることが保証されている. また深さ優先探索のメモリ効率の良さもかね備えている.

Vol.2009-MPS-76 No.11
Vol.2009-BIO-19 No.11
2009/12/17

4. 下界の計算方法

本章では前章で挙げた木探索におけるポイントの 3 つ目である下界の計算方法についての説明を行う.

4.1 既存の手法 (Fukunaga[5])

既存の計算方法はまず各サーバについて下界の計算を行いその結果を足し合わせてノード全体の下界としている.

実際に例を用いて説明すると, ノード $n = (5, 10, \underline{12}^*)_{25}(10, \underline{9})_{25}$ であった場合, 一つ目のサーバ s_1 は割り当てられた VM の使用率の和がサーバのリソース量を越えてしまっている. よって幾つか VM を取り除かなくてはいけない. この場合は 5 か 10 のいずれかを取り除けば良いので $\mathrm{lb}(s_1) = 1$ となる次に二つ目のサーバ s_2 を見ると既に $10 + 9 \leq 25$ となっており一つも取り除く必要が無いため $\mathrm{lb}(s_2) = 0$ となりノード全体の下界は $h(n) = 1 + 0 = 1$ となる. また $g(n) = 1$（*の数）なのでコスト関数は $f(n) = g(n) + h(n) = 2$ となる. この時 UB の値が 2 よりも大きかった場合は子ノードを順次計算していくこととなる.

4.2 Max Free Space,Min Excluded Items

本論文で提案する計算方法を説明するにあたり Max Free Space（以後 MFS）と Min Excluded Item（以後 MEI）について説明を行う. ともにサーバに対して定義される関数である.

まず MFS は, サーバから $\mathrm{lb}(s)$ 個のコミットされていないアイテムを取り出してサーバに作ることができる最大の空き容量である.

MEI はサーバに空き容量を作ることができる $lb(s)$ 個のコミットされていないアイテムの組み合わせの中に現れるアイテムの内, もっとも使用率の小さいアイテムを指している. 以後例を用いて説明する. サーバ $s = (5, 12, 9, 1, \underline{4})_{16}$ に対して MFS と MEI を計算する. まず, このサーバの $\mathrm{lb}(s)$ は 2 となる. MFS は 2 個のコミットされていない VM を取り除いて作ることができる最大の空き容量の大きさである. よって

$$\begin{aligned} MFS(s) &= 16 - ((5 + 12 + 9 + 1 + 4) - 12 - 9) \\ &= 16 - (31 - 21) = 6 \end{aligned}$$

となる. 次に MEI は, 一番小さな 1 について考えてみるとこの VM と後一つの VM を取り除くだけではサーバに空き容量を作ることはできない. 次に小さい 4 はコミットされているので考えない. そしてその次に小さいのは 5 であるが 5 と 12 を取り除くと $16 - (31 - 5 - 12) \geq 0$ となり空き容量を作ることができるよってこのサーバの MEI は 5 となる.

4.3 新たな計算方法

既存の下界 $h(n)$ と MFS,MEI を用いて新たな下界 $h'(n)$ を以下のように定義する.

$$h'(n) = \begin{cases} h(s) + 1 & \text{(case 1)} \\ h(s) & \text{(o.w)} \end{cases}$$

case 1 : $Items = \{MEI(s_1), MEI(s_2), \cdots, MEI(s_m)\}$, $Bins = \{MFS(s_1), MFS(s_2), \cdots, MFS(s_m)\}$ としたビン・パッキング問題が答えを持たない.（すべてのアイテムをアサインすることができない）

実際に $h(s)$ 個の VM の移動で問題を解決できるかを考えると,case 1 を満たした場合は解ける可能性がない. なぜなら各サーバからは最低 $\mathrm{lb}(s)$ 個の VM を取り出さなくてはいけないのだが, その時に作り出すことができる空き容量は MFS 以下である. 一方でどんなに小さめのアイテムを取り出そうとしても,MEI より大きなアイテムは取り出してどこかに移さなくてはいけない. 新たにできる空き容量を多めに見積り, 移動させるアイテムの大きさを小さめに見積もった状況で, そのアイテムを割り当て直そうと試みた結果割り当てることができなかったという事は, 実際に $h(n)$ 個の移動で問題解決をすることは出来ないということとなる. よって少なくとも $h(n) + 1$ 個の移動が必要となることが解るため case 1 を満たす場合はそのノードの下界を既存のものより更に 1 大きな値にすることができる.

5. ドメイン縮小

前章で扱った下界の計算や改善の方法は仮想マシン再配置問題に限った「問題依存」の話であったのに対し, 本章で扱うドメイン縮小はすべての最適化問題にというわけではないが仮想マシン再配置問題と構造が似た多くの問題に対して有効である可能性のある「問題非依存」の考えである. 本章ではドメイン縮小のアルゴリズムと意味合いについて説明を行う.

そもそもドメインとは何かについて説明を行う. 本問題では, 各 VM がそれぞれドメインを持ち, そのドメインは「アサインされる可能性の有るサーバの集合」である. 一般の最適化問題に関して言うとそれぞれの変数が取りうる値の集合がドメインである. 現状の探索アルゴリズムではドメイン縮小を行っていないため,VM はコミットされない限りすべてのサーバをドメインとして持つ. ここで探索を行いつつコミットされていない VM のドメインを少なくしていくことが出来れば探索空間が小さくなり, 結果として探索時間を短縮することが

Vol.2009-MPS-76 No.11
Vol.2009-BIO-19 No.11
2009/12/17

出きるのではないかと考えた. これがドメイン縮小の大本となる考え方である.

ドメイン縮小のアルゴリズムは以下のようになっている. なおアルゴリズム内に出てくるDomain 関数はその地点でのサーバのドメインを表す.

ドメイン縮小アルゴリズム

for all *item* ∈Uncommitted Items **do**
  for all *server* ∈ Domains(*item*) **do**
    Commit *item* to *server*,and let this assignment be n'
    Calculate h(n')
    if h(n') > UB **then**
      Remove *server* from Domains(*item*)
    end if
  end for
end for

現在コミットされていないすべての VM に対して, 自分のドメインの各サーバそれぞれに移動させてみて再度下界を計算し, もしそれが UB 以上であるならば VM のドメインからそのサーバを取り除いている.

この操作によって現在よりも深い部分で大量に発生する可能性の有る,VM を今回取り除かれたサーバにアサインし下界を計算するという作業がドメイン縮小によって取り除かれたということになる.

ドメイン縮小を行うドレードオフは各ノードでの計算量が今までに比べてとても大きくなるということである. 現状は一度下界を計算すれば良かったのに対し, ドメイン縮小を行うとコミットされていない VM それぞれが自身のドメインの回数だけ下界を計算することになる. よってドメイン縮小を有効に活用するためには, アイテム数とドメイン数の比や, 下界の計算量等, 様々な指標からドメイン縮小を行うか否か, またするならどの程度の深さまで行うのかを決定する場合分けの方法や, パラメータの調整が重要になってくると考えられる.

6. 実験結果

実験としては以下の 2 点を行った

(1) Fukunaga[5] の下界を用いた場合と, 本論文んで提案した下界を用いた場合, ドメイン縮小を用いた場合, その両方を行った場合での実行時間の変化

(2) ドメイン縮小を行う深さに制限をかけ, その変化によって生じる変化の分析

なお, 実験で用いた問題インスタンスはサーバ数に関わらず,$\sum |VM| / \sum |Server| = 95\%, \forall |Server| = 1000, 200 \leq |VM| \leq 400$ とした. これは実問題では一台の物理サーバ上で 3,4 台の VM を走らせるのが一般的であるということと組み合わせ問題として困難な問題が多くなるという点からである. 実行環境は 2GHz の Intel Core2 プロッセッサーで, インスタンスあたりの制限時間は 600 秒とした.

6.1 下界の違い・ドメイン縮小の有無による性能の違い

下界の計算方法に Fukunaga[5] で用いられた方法を使った場合と本研究で提案したものを使用した場合, また, それぞれでドメイン縮小を行った場合と行わなかった場合によって実行時間にどのような変化が起こるかを実験した（**表 1**）. 各表の値は 30 インスタンスの平均であり括弧中の数字は制限時間 600 秒以内に答えを見つけることが出来なかったインスタンス数を指している. なお, 平均をとる際に 600 秒以内に答えを見つけることができなかったインスタンスは一律 600 秒として扱った. 結果は, すべてのサーバ数の場合で本研究の下界の計算方法を使用し, 尚且つドメイン縮小を行った場合が早いという結果になった. また一見するとサーバ数が多くなるごとに既存の手法と本研究の手法を使った場合の実行時間比が小さくなっているようにも見えるがこれは, 制限時間内に終わらず打ちきったインスタンス数に大きな差があるためと考えられる. 実際に図 3 のように各インスタンスについて実行時間比をプロットしてみるとサーバ数が多い場合でも本研究の計算方法を用いた場合の方が総じて 10 倍以上の早さで最適解を求められていることがわかる.

6.2 ドメイン縮小を行う深さに応じた性能の違い

ドメイン縮小を行う深さに制限を掛け, その変化に応じてどの様に実行時間が変化するのか実験を行った. 表 2 はサーバ数=16 の 2 例に対して深さに応じた探索時間, ノード数の変化に加えて関数 lb の計算回数の変化をまとめた物である（lb の計算回数はノードでの計算に加えてドメイン縮小内での計算回数を含む）. インスタンス 1 はドメイン縮小を行う深さ

Vol.2009-MPS-76 No.11
Vol.2009-BIO-19 No.11
2009/12/17

表 1 下界の計算法の違い, ドメイン縮小の有無による結果の変化

	F09	F09+reduce-domain	TF09	TF09+reduce-domain
Server=6	0.04(0)	0.02(0)	0.005(0)	**0.004(0)**
Server=8	3.26(0)	2.14(0)	0.28(0)	**0.17(0)**
Server=10	64.5(3)	55.9(2)	9.3(0)	**5.6(0)**
Server=12	202.8(5)	117.4(3)	30.6(1)	**26.4(1)**
Server=14	241.9(9)	182.7(6)	52.7(2)	**48.0(2)**
Server=16	332.4(15)	313.9(14)	109.4(3)	**90.4(3)**

30 インスタンスの実行時間の平均. 括弧内は 600 秒以内に最適解が
見つからなかったなかったインスタンスの数.
最適解が見つからなかった場合は 600 秒として平均は計算

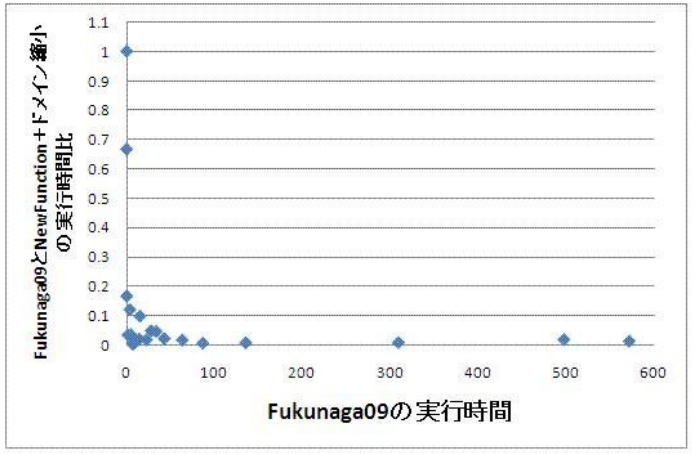

図 3 Server 数=14 の各インスタンスの Fukunaga CP09 と本研究との実行時間比

が深くなるにしたがって実行時間が短くなって行っているのに対し, インスタンス 2 は深さ 9 のところで実行時間が最小となり以後, 微小ではあるが探索時間は増加に転じている. これはインスタンス 2 が比較的簡単な問題であったため, 深い部分ではドメイン縮小をしながら計算を行うよりも単純に探索してしまった方が早く計算が済んでしまうからだと考えられる. また, 探索時間と lb の計算回数の大小は必ずしも一致していない. インスタンス 1 では深さ 10 の地点, インスタンス 2 では深さ 5 の地点で計算回数は最小となっている. これはドメイン縮小をより深くまで行うことで探索するノード数自体は減るため, 探索のための様々な処理（終了条件のチェックや, 次にどのアイテムをどこにコミットするかを決める作業等）が減り結果として実行時間はより深くまでドメイン縮小を行った方が短くなっていると考えられる. よって, それらの作業が限りなく短い時間で出来る場合, プログラム全体の実行時間は lb の計算回数に依ってくると考えられる. 計算回数を性能の基準とすると今回のようにインスタンスによってドメイン縮小を行う最適な深さが大きく異なってくる可能性があるため制限の掛け方が重要になってくる.

表 2 ドメイン縮小を行う深さに制限を加えた場合の実行時間の変化

	Instance1			Instance2		
	Time	Nodes	Call lb func	Time	Nodes	Call lb func
depth = 0	258.62	2.51E+07	2.51E+07	18.45	1.86E+06	1.86E+06
depth = 1	258.94	2.51E+07	2.51E+07	18.22	1.85E+06	1.86E+06
depth = 2	252.33	2.46E+07	2.46E+07	17.92	1.77E+06	1.81E+06
depth = 3	248.57	2.39E+07	2.40E+07	17.55	1.67E+06	1.82E+06
depth = 4	243.02	2.33E+07	2.34E+07	15.39	1.33E+06	1.61E+06
depth = 5	235.69	2.24E+07	2.27E+07	13.23	9.17E+05	**1.47E+06**
depth = 6	223.01	2.05E+07	2.13E+07	13.46	7.98E+05	1.62E+06
depth = 7	203.85	1.81E+07	1.96E+07	13.04	6.50E+05	1.72E+06
depth = 8	190.8	1.63E+07	1.87E+07	12.65	5.10E+05	1.82E+06
depth = 9	158.49	1.28E+07	1.60E+07	**12.22**	3.66E+05	1.91E+06
depth = 10	139.7	1.04E+07	**1.45E+07**	12.46	3.00E+05	2.02E+06
depth = 15	130.72	8.22E+06	1.52E+07	12.45	3.52E+04	2.26E+06
depth = 20	102.16	3.99E+06	1.48E+07	12.47	**1.86E+04**	2.30E+06
depth = 25	95.92	2.20E+06	1.61E+07	—	—	—
depth = 30	93.52	1.05E+06	1.71E+07	—	—	—
depth = 35	**92.27**	3.84E+05	1.78E+07	—	—	—
depth = 40	92.91	**3.07E+05**	1.80E+07	—	—	—

Server 数=16 のインスタンスから実行時間に差がある 2 例を用いて実験を行った.
インスタンスによって実行時間と lb 関数の計算回数ともに最適な深さが異なっている.

Vol.2009-MPS-76 No.11
Vol.2009-BIO-19 No.11
2009/12/17

7. 終わりに

本研究では, 仮想マシン再配置問題に対する厳密アルゴリズムの改善を行った. まず一つ目として既存の下界の計算方法の改善を行い 10 倍〜20 倍のスピードアップに成功した. また, ドメイン縮小という問題非依存の手法の提案し仮想マシン再配置問題においてはある程度有効であることを示すことが出来た. 今後の課題として, 本論文ではコストを移動した VM の数としていたがそれ以外にも移動した VM の大きさの和など他のコストの定義が考えられる. 本研究での提案手法はこれらの場合に対応できない. よって他のコストの定義に対しても効果的な下界の計算方法を定義出来るかどうかを考えていく事が一点目である.

二点目として, 組み合わせ最適化問題の中には, 初期状態から, いかにコストを抑えて目的状態に移ることができるかを求める問題がある[7]. そのような問に対してドメイン縮小の適用可能性を検討していきたい.

参 考 文 献

1) Vogels, W.: Beyond server consolidation. ACM Queue 6(1) (2008)
2) Aggarwal, G., Motwani, R., Zhu, A.: The load rebalancing problem. In: Proc. 15th ACM Symp. on parallel algorithms and architectures, pp. 258265 (2003)
3) Ajiro, Y., Atsuhiro T.: A Combinatorial Optimization Algorithm for Server Consolidation. In: Proc. The 21th Annual Conference of the Japanese Society for Artificial Intelligence (2007)
4) Ajiro, Y.: Recombining virtual machines to autonomically adapt to load changes, in Proc. 22nd Annual Conference of the Japanese Society for Arti ¯ cial Intelligence (2008)
5) Fukunaga, A.: Search algorithms for minimal cost repair problems. In: Proc. CP/ICAPS 2008 Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (2008)
6) Korf, R.: Depth-first iterative-deepening: an optimal admissible tree search. Artificial Intelligence 27(1), 97109 (1985)
7) Barbulescu, L.; Whitley, D.; and Howe, A.: Leap before you look: An effective strategy in an oversubscribed scheduling problem. In Proceedings of AAAI.(2004)
8) Fukunaga, A., Ajiro, Y.: Exact and Hybrid Algorithms for Virtual Machine Reassignment : The 23rd Annual Conference of the Japanese Society for Artificial Intelligence, (2009)