3E8-1

サーバ統合のための組合せ最適化アルゴリズムの提案と評価

A Combinatorial Optimization Algorithm for Server Consolidation

網代育大　　田中淳裕
Yasuhiro Ajiro　Atsuhiro Tanaka

NEC システムプラットフォーム研究所
System Platforms Research Laboratories, NEC Corporation

An objective of server consolidation is to minimize the number of destination servers that accommodate existing servers. This minimization is a variant of the bin packing problem, where items to be packed are existing servers, bins are destination servers, and the sizes of items are resource utilizations. We have developed for the N-dimensional (server) packing problem an improved FFD algorithm based on the classical FFD. The classical FFD packs items in descending order of size and adds a new bin for an item dropped from current bins. In contrast, our improved FFD alters the order for the item dropped to be packed first and retries without adding a new bin. Experiments we conducted with two-dimensional data showed that our algorithm was effective in reducing the number of destination servers when the correlation between the two-dimensional data was weak or negative. The experiments and the adequate numbers of times of the retries are presented.

1. はじめに

サーバ統合は、企業等の組織内に増殖した数十台から数百台のサーバを 1/3 以下の台数の高性能サーバに集約するシステムソリューションである。現在サーバ統合が盛んに行なわれている背景には、サーバの運用管理コストの増大という経営的な側面と、VMware 等の仮想化ソフトウェアによって物理サーバ環境を仮想マシン上に低コストで移行できるようになったという技術的な側面がある。

サーバを統合する際には、サーバをどのように組み合わせて集約するかが重要な問題となる。統合にかかるコストを見積もる上でも、統合に必要な集約先のサーバ台数をあらかじめ見極める必要がある。このとき、まず考慮しなくてはならないのが CPU やディスク、メモリ、ネットワークといった計算資源であり、統合対象のサーバによる資源の必要量が統合先サーバの備える資源容量を超えると、重大な性能劣化を引き起こす怖れがある。メモリ容量の不足によるスラッシング (thrashing) などがその例である。

大きさの異なるアイテムの集合を固定サイズの箱に詰めるとき、必要な箱の個数を求める問題はビンパッキング問題として知られており、アイテムや箱が N 次元の大きさをもつ場合は N 次元ビンパッキング問題と呼ばれる。サーバ統合におけるサーバの組合せを求める問題は、ビンパッキング問題の変形であるベクトルパッキング問題 (vector packing problem) として定式化できる。具体的にいえば、統合先サーバに収容されるサーバの各計算資源使用量 (使用率) の合計値が許容量を超えないという制約の下で、目的関数である統合先サーバの台数が最小になるようなサーバの組合せを求めるのが、サーバ統合における組合せ最適化である。以降、特に区別の必要がない場合は、両者を単にパッキング問題と呼ぶことにする。パッキング問題は、回路設計やスケジューリング、材料の加工といった様々な応用をもつが、組合せ最適化問題の 1 つとして一般に NP 困難である。

本研究の目的は、サーバの組合せ最適化問題を短時間で解くための汎用のアルゴリズムの開発である。このときの要件として、問題のサイズは、アイテムであるサーバが数百台規模、次元数は 4 から 6 前後、計算時間に関しては、エンジニアが現場での設計や提案書の作成に使うことを考えると、数分以内が望ましい。パッキング問題は長い歴史をもち、ヒューリスティックな解法のほか、応用の多い 2 次や 3 次の問題に対して厳密解を求める新しい手法がいくつか提案されている。しかしながら、厳密解を求める手法には解ける問題のクラスに制限があり、我々の要件を満たすのが難しい。そのため今回我々は、一次元ビンパッキング問題に対するヒューリスティックな解法としてよく知られている First-Fit Decreasing (FFD) アルゴリズムをベースにした改良 FFD アルゴリズムを開発した。

従来の FFD は、アイテムを大きい順にソートしてから箱詰めし、既存の箱に詰められないときはすぐに新しい箱を追加するという単純な方法を用いているため、最適性に関しては劣るものの計算コストが非常に低い。また、多次元の問題に対しては、ある次元、たとえば高さと幅をもつアイテムの高さのみに着目して FFD を適用する FFDH アルゴリズムなどが存在し、多次元の問題にも拡張しやすいという特徴をもつ。しかしながら N 次元の問題に対しては、着目していない次元の大きなアイテムのせいで箱の数が無駄に増えるという問題があった。我々の改良アルゴリズムは、箱詰めに失敗してもすぐには箱を追加せず、失敗したアイテムを優先的に詰めるようにアイテムの順序を変更したのち、箱詰めを最初からやり直す。従来の FFD の手法は、大きいものを先に詰めてから小さいものをその隙間に詰める方が少ない箱で済むというヒューリスティクスに基づいているが、改良 FFD は、箱詰めに失敗したアイテムを着目していない次元の大きなアイテムとみなし、これを優先的に詰めることで最終的な箱の数を削減する。

改良 FFD アルゴリズムが箱を追加するタイミングは、箱詰めをやり直す回数 (リトライ数) が所定の回数を超えたときである。我々は重要度の高い CPU 使用率とディスク使用率の 2 つに次元を限定した 2 次元パッキング問題を対象に実験を行ない、アルゴリズムの有効性を確認した。実験は各次元の大きさに相関のあるデータと相関のないデータをランダムに生成して行ない、アルゴリズムの算出した統合台数とリトライ数との関係を調べた。その結果、相関が低い、あるいは負の相関が高い場合は従来の FFD による統合効率が低下するものの、改良 FFD によって効率の低下を改善できることがわかった。正の

連絡先: 網代育大，日本電気 (株) 中央研究所
〒 211-8666 神奈川県川崎市中原区下沼部 1753
y-ajiro@cd.jp.nec.com

相関がある場合は、着目する次元によらずアイテムのソート結果がほぼ等しくなるため、従来の FFD であっても充分な効率が得られる。本稿ではアルゴリズムと実験の詳細を示すとともに、適切なリトライ数について考察を加える。

2. サーバ統合における組合せ最適化

サーバの組合せ最適化においてアイテムの次元に相当する項目は、CPU やディスクの使用率、ディスクやネットワークの転送量、メモリ使用量といった、計算資源に関する単位時間当たりの使用率や使用量である。どの資源を考慮するかは OS の提供する計測手段や統合するサーバの性質に依存する。後述のアルゴリズムは 3 以上の次元 (資源使用率/量) に対応したものとなっているが、今回は特に、CPU 使用率とディスク使用率の 2 つを想定した二次元のパッキング問題を実験の対象とした。これら 2 項目の値はサーバによる差異が大きく、統合結果に与える影響が大きい。これに対し、メモリ使用量やネットワーク転送量等は比較的変動が小さく、統合結果に与える影響も小さいことから、省略しても実験結果の一般性を大きく損なうことはないと考えた。

複数台の統合対象サーバを 1 台の統合先サーバ上で稼働させたとき、統合先サーバの CPU 使用率は、一般に統合対象サーバの CPU 使用率の和として見積もることができる。ディスク使用率や他の資源の使用量に関しても同様である。たとえば、CPU 使用率とディスク使用率の組がそれぞれ (15%, 10%) であるサーバ A と、(20%, 12%) であるサーバ B を統合した場合、統合先サーバの資源使用率は、これらの組をベクトルとみなしたときのベクトル和 (35%, 22%) と見積もれる。統合対象サーバの資源使用率/量としては、運用時に一定期間計測した値の最大値等が用いられる。また、資源使用率/量が一定の閾値を超えると重大な性能劣化が生じるため、使用率/量が閾値を超えないように統合計画を立案するのが一般的である。

統合対象のサーバ s_i における CPU とディスク使用率をそれぞれ ρ_{c_i}, ρ_{d_i}、閾値を R_c, R_d、n 台の統合先サーバ s'_j $(j = 1, \cdots, n)$ に統合されるサーバの集合を X_j としたとき、サーバの組合せ最適化問題は次のようなベクトルパッキング問題として定式化できる。

$$\begin{array}{ll} min & n \\ s.t. & \sum_{s_i \in X_j} \rho_{c_i} \leq R_c\,, \\ & \sum_{s_i \in X_j} \rho_{d_i} \leq R_d \quad (j = 1, \cdots, n)\,. \end{array}$$

実際の統合計画においては、統合対象のサーバと統合先サーバとの性能差による使用率の換算や仮想マシンの同時稼働数に基づくオーバヘッド [1] を考慮する必要があるが、スコープ外としてここでは考えないことにする。

一方のビンパッキング問題はベクトルパッキング問題よりもやや難しく、二次元上の長方形の箱にアイテムの長方形を重なりのないように配置しなければならない (図 1)。この図では、(1.0, 1.0) の大きさの左側の箱に 3 つのアイテムが収まっているが、アイテムの大きさが CPU とディスク使用率の組を表していたとすると、アイテムを収容した統合先サーバの使用率は (1.9, 1.3) となって (1.0, 1.0) に収まらない。このような問題の性質の違いによって、ビンパッキング問題のための手法が我々の対象とするサーバの組合せ最適化には適用できない場合も多く存在する。

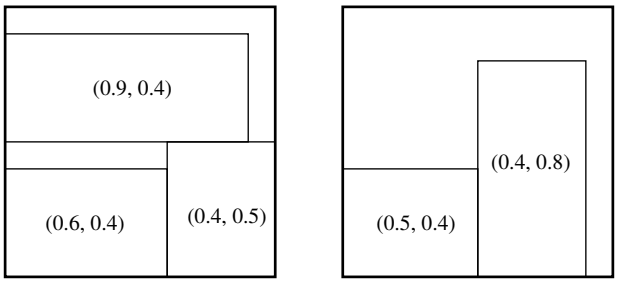

図 1: 二次元ビンパッキング問題において、5 つのアイテムを 2 つの箱に詰めた例。括弧内の数字はそれぞれ x, y 軸方向の大きさを示し、箱の大きさは (1.0, 1.0) である。

```
n ← 1;  X_1 ← {};
for i ← 1 to m do
  for  j ← 1 to n do
    if packable(X_j, s_i) then
      X_j ← X_j ∪ {s_i};
      break fi
  end for;
  if  j = n + 1 then /* 詰めるのに失敗した場合 */
    n ← n + 1;   /* サーバを 1 台追加 */
    X_n ← {s_i} fi
end for
```

図 2: サーバの組合せを算出する FFD アルゴリズム

3. アルゴリズム

3.1 従来の FFD アルゴリズム

サーバの組合せを算出するための従来の FFD アルゴリズムを図 2 に示す。このアルゴリズムは、ある資源の使用率 (たとえば CPU 使用率) について降順にソートされた m 台のサーバ $s_1, \cdots, s_m$ の資源使用率を入力とし、統合台数 n と組合せ X_j $(j = 1, \cdots, n)$ を算出する。関数 $packable(X_j, s_i)$ は、統合先サーバ s'_j 上にサーバ s_i を追加しても資源使用率が閾値を超えない (つまり制約条件を満たす) 場合に真、超える場合に偽を返す。前提として、各サーバ s_i の資源使用率は、統合先サーバの閾値以下であるとする。従来の FFD アルゴリズムは、統合対象であるサーバ $s_1, \cdots, s_m$ が統合先サーバ $s'_1, \cdots, s'_n$ (の収容するサーバの集合 $X_1, \cdots, X_n$) に詰められるかどうかを順に調べ、詰められるサーバに詰めるという動作を繰り返す。サーバが既存の n 台の統合先サーバに詰められなかった場合は、$n+1$ 台目の統合先サーバを追加して詰める。統合先サーバの台数 n は統合対象サーバの台数 m にほぼ比例することから、このときの計算量は $O(m^2)$ となる。

3.2 改良 FFD アルゴリズム

サーバが複数の資源をもつ場合、従来の FFD アルゴリズムでは、ソートの際に着目していない資源を多く使うサーバが原因となって最終的な統合台数が増加する恐れがある。我々の提案する改良 FFD アルゴリズム (図 3) は、そのようなサーバを検出して優先的に詰めることでこの問題の解決を図る。アルゴリズムは統合台数 n に関する反復深化探索 (iterative-deepening search) を行なっており、n 台への統合可能性を最大 $MAXR$

```
n ← LB({s_1, ··· , s_m});
while true do
  T' ← {};   T ← {s_1, ··· , s_m};
  for r ← 1 to MAXR do
    for  j ← 1 to n do
      X_j ← {} /* 初期化 */
    end for;
    s ← pack(T');
    if s ≠ ∅ then break fi;
    s ← pack(T);
    if s ≠ ∅ then
      T' ← T' ∪ {s};
      continue
    else /* パッキングに成功 */
      return n fi
  end for;
  n ← n + 1
end while

function pack(S) begin
  for each s ∈ S do
    for  j ← 1 to n do
      if packable(X_j, s) then
        X_j ← X_j ∪ {s};
        break fi
    end for;
    if  j = n + 1 then /* 詰めるのに失敗した場合 */
      S ← S \ {s};
      return s fi
  end for;
  return ∅
end
```

図 3: サーバの組合せ最適化のための改良 FFD アルゴリズム

回だけ探索する。この $MAXR$ はユーザの与える定数で、サーバを詰める順番を変更する回数 (リトライ数) の最大値に対応する。関数 $LB(\{s_1, \cdots, s_m\})$ は統合台数に関する理論上の下限値を算出して返す。このときの下限値は、統合対象サーバの資源使用率の合計を統合先サーバの閾値で割った値を下回らない整数で、CPU 使用率に関しては $\lceil \sum_{i=1}^{m} \rho_{c_i}/R_c \rceil$ と書ける。関数 LB はこの CPU 使用率に関する下限値とディスク使用率に関する下限値 $\lceil \sum_{i=1}^{m} \rho_{d_i}/R_d \rceil$ の大きい方を返す [2]。

下請けの関数 $pack(S)$ の動作は、図 2 で示した FFD の動作とほぼ同様であるが、サーバの集合 S を既存の n 台の統合先サーバに詰められなかったときに、統合先サーバを 1 台追加するのではなく、詰められなかった当のサーバ s を S から削除して返す。すべてのサーバを詰めるのに成功した場合は、ヌルに相当する $\emptyset$ を返す。メイン部には、優先的に詰めるサーバの集合 T' とその他のサーバの集合 T があり、すべての統合対象サーバは最初 T に属する。関数 $pack(S)$ を使って T' のサーバ群を詰め、次に T を詰める。両方のサーバ群をすべて詰めることができた場合は、統合台数 n を返す。集合 T' を詰めるのに失敗した場合や、リトライ数が $MAXR$ に達した場合は、統合サーバを 1 台追加する。集合 T を詰めるのに失敗したときは、統合サーバをすぐには追加せず、$pack(S)$ が返すサーバ s を T' に移動して再探索する。集合 T' は最大 $MAXR$ 個のサーバをもつ。つまり、ソートの際に着目していない資源の使用率が大きく、統合台数の増加要因となるサーバを $MAXR$ 個まで検出できる。

改良 FFD アルゴリズムの計算量は、下限値と最終的な統合台数との差を d としたとき、$O(d \cdot MAXR \cdot m^2)$ となる。今回は特に $MAXR$ の適切な値を得るため、実験を行なってリトライ数と統合台数との関係について調べた。次節でその結果を示す。

4. 評価

今回の実験では、200 台のサーバの CPU 使用率とディスク使用率のデータをランダムに生成した二次元パッキング問題のインスタンスに対して従来の FFD ならびに改良 FFD を適用し、得られた統合台数を比較した。我々は CPU 使用率とディスク使用率の相関関係を考慮し、相関係数の異なる複数のインスタンスを生成して実験を行なった。各資源使用率が強い相関をもつ場合、たとえば CPU 使用率とディスク使用率の値が等しいと、このときの問題は一次元の問題に帰着するため、従来の FFD がうまく働く。これに対し、各資源使用率に相関がないか、負の相関が高い場合は FFD がうまく働かず、改良 FFD による改善が期待できる。

実験に際して、指定した相関係数をもつ乱数列になるように各サーバの資源使用率を生成する必要があったが、我々の知る限りでは、標準的な生成手法が存在しない。そこで、アドホックな方法ではあるが、CPU 使用率とディスク使用率がともに平均値以上あるいは以下になる確率 P を導入し、以下に示すアルゴリズムを用いて使用率データを生成した。

```
for i ← 1 to m do
  ρ_{c_i} ← rand(2ρ̄_c);   ρ_{d_i} ← rand(ρ̄_d);
  r ← rand(1.0);
  if  (r < P ∧ ρ_{c_i} ≥ ρ̄_c) ∨ (r ≥ P ∧ ρ_{c_i} < ρ̄_c) then
    ρ_{d_i} ← ρ_{d_i} + ρ̄_d  fi
end for
```

ここで、$rand(a)$ は範囲 $[0, a)$ の `double` 型の一様乱数を返す関数、$\overline{\rho_c}$, $\overline{\rho_d}$ はそれぞれ CPU とディスクの平均使用率を表し、確率 P を変化させることで相関をある程度調節できる。今回の実験では、$P = 0.0, 0.25, 0.50, 0.75, 1.0$ の 5 つの場合について、それぞれ 100 組ずつ、合計 500 組のインスタンスを生成した。また、実際のサーバ統合では 1 台の統合先サーバに 4 台前後のサーバを集約する例が多いことから、CPU とディスクの平均使用率をともに $\overline{\rho_c} = \overline{\rho_d} = 15\%$、統合先サーバにおける閾値を $R_c = R_d = 80\%$ に設定した。

各インスタンスに対するアルゴリズムの適用結果を表 1 に示す。各表の値はそれぞれ (P を除いて) すべて 100 組のインスタンスに対する平均値を算出したもので、LB は第 3.2 節で述べた下限値を表す。列 Algorithm における FFD, FFDR は、それぞれ従来の FFD と我々の改良 FFD に対応する。列 $MAXR$ は $MAXR$ の設定値を表しており、今回の実験では統合台数の (平均) 削減数が 0.5 より小さくなるまで $MAXR$ の値を 20 ずつ増やしながら統合台数の変化を観察した。削減率は、FFD による統合台数を n_F、FFDR による統合台数を n_R としたときの $(n_F - n_R)/n_F$ である。一番右の列は、アルゴリズムの実行にかかった秒数を示している。アルゴリズムは Linux

表 1: ランダムに生成したサンプルデータに対して FFD および改良 FFD (FFDR) を適用して得られた統合台数の比較

Algorithm	$MAXR$	統合台数	削減率 (%)	時間 (sec)
相関係数 -0.749 ($P = 0.0$), LB 39.1				
FFD	—	54.8	—	0.132
FFDR	20	50.5	7.85	13.9
FFDR	40	47.5	13.3	19.2
FFDR	60	45.2	17.5	20.9
FFDR	80	43.9	19.9	22.1
FFDR	100	43.6	20.4	25.1
相関係数 -0.383 ($P = 0.25$), LB 38.9				
FFD	—	50.6	—	0.111
FFDR	20	46.9	7.31	9.80
FFDR	40	45.3	10.5	15.0
FFDR	60	44.3	12.5	18.7
FFDR	80	44.0	13.0	22.5
相関係数 -0.00332 ($P = 0.50$), LB 38.8				
FFD	—	46.8	—	0.125
FFDR	20	44.1	5.77	6.88
FFDR	40	43.4	7.26	11.2
FFDR	60	43.1	7.91	14.9
相関係数 0.373 ($P = 0.75$), LB 38.7				
FFD	—	43.7	—	0.102
FFDR	20	42.2	3.43	4.60
FFDR	40	41.9	4.12	7.82
相関係数 0.751 ($P = 1.0$), LB 38.5				
FFD	—	41.2	—	0.106
FFDR	20	40.7	1.21	2.75
FFDR	40	40.7	1.21	4.83

2.4.33 上の Java (JDK 1.5) を用いて実装し、Pentium4 2.0 GHz CPU と 768 MB メモリを搭載した PC/AT 互換機上で実行した。

結果から、確率 P が小さい、すなわち CPU 使用率とディスク使用率の正の相関が低いか、あるいは負の相関が高いほど、下限値 LB に比べて FFD による統合台数が増大している。しかしながら、負の相関が高いときは FFDR による削減効果が高く、$MAXR$ の値が大きいほど削減率も上昇している。相関係数 -0.749 のインスタンスに対しては、$MAXR = 100$ のときの削減率が 20.4% となった。正の相関が高い場合は、FFDR による削減率が低いが、これは従来の FFD によってすでに下限値に近い統合台数が得られているためである。

我々の経験によれば、サーバの用途やアプリケーションによって、CPU を多く使うものやディスクを多く使うものがあり、両方をバランスよく使うサーバは少数派である。また、あるアプリケーションの障害がほかのアプリケーションに影響するのを避けるため、1 サーバ上では 1 種類のアプリケーションを稼働させる例が多い。これらの理由から、多くの場合、CPU とディスク使用率には相関関係がないか、あるいは正負の弱い相関がある程度と予想される。そこで、表を参照しながら $MAXR$ の妥当な値を考えると、負の弱い相関がある (相関係数 -0.383 の) 場合に 60、正の弱い相関がある (相関係数 0.373 の) 場合には 20 で削減率がほぼ飽和していることから、統合対象サーバ数 (200 台) の 1–3 割程度が $MAXR$ の目安といえそうである。また、改良 FFD の処理時間に関しては、FFD の $d \cdot MAXR/2$ 倍程度という結果が得られた。

5. 関連研究

二次元パッキング問題の厳密解を求める手法は、文献 [2, 3] 等で提案されており、アイテム数 100 以下の問題の多くを数十分以内に解くことができる。これらの手法は、分割統治法による探索を行ないながら、下限や上限を動的に更新し、効率よく枝刈りを行なうところに特徴がある。我々の改良 FFD は最適性よりも計算コストを優先しているが、一定時間内に厳密解が見つからない場合に改良 FFD を適用するといった連携が可能である。

6. まとめと今後の課題

サーバ統合時における統合台数の最小化をパッキング問題として定式化し、この準最適解を短時間で算出するための改良 FFD アルゴリズムを提案した。実験を行なってアルゴリズムの有効性を確認するとともに、200 台程度のサーバ統合に対しては充分な性能が得られることを示した。また、アルゴリズムにおけるリトライ数や計算量について考察した。今回の実験は CPU 使用率とディスク使用率の 2 つを想定した二次元のデータに対して行なったが、アルゴリズム自身は N 次元の問題にも適用可能である。

一般のパッキング問題とは異なり、サーバログ等の実データを用いる場合は、相関関係のような入力データの傾向を利用することでアルゴリズムをさらに改良できる可能性がある。今回は実験結果を示していないが、500 台のサーバに対する計算時間は 400–800 秒に増大し、やや許容範囲を超える結果が得られた。改良 FFD の計算量が $O(d \cdot MAXR \cdot m^2)$ であることから、相関係数をもとに最適解が下限値 LB の何%増かを見積もることができれば、統合台数の初期値を増やすことで d を削減し、計算量を削減できる。また、事前に算出した相関係数に応じて適切な $MAXR$ の値を決定できる可能性や、改良 FFD の算出結果を厳密解を探索する際の上限値として利用できる可能性がある。これらの検討が今後の課題である。

参考文献

[1] 網代育大, 田中淳裕. 仮想計算機環境における資源管理オーバヘッドの評価. 情処システム評価研究会研究報告 (EVA-17), pp. 17–22, 2006.

[2] A. Caprara and P. Toth. Lower bounds and algorithms for the 2-dimensional vector packing problem. *Discrete Applied Mathematics*, Vol. 111, No. 3, pp. 231–262, 2001.

[3] J. Carlier, F. Clautiaux, and A. Moukrim. New reduction procedures and lower bounds for the two-dimensional bin packing problem with fixed orientation. *Computers and Operations Research*, Vol. 34, No. 8, pp. 2223–2250, 2007.