南京大学

本 科 毕 业 论 文

院　　系 计算机科学与技术系

专　　业 计算机科学与技术

题　　目 基于模拟退火算法的虚拟机部署技术

年　　级 2008 级　学　　号 081221070

学生姓名 钱行

指导老师 钱柱中　职　　称 副教授

论文提交日期 2012 年 5 月 27 日

学　　号：　　081221070

论文答辩日期：　　____年__月__日

指导教师：　　_____（签字）

Placement Algorithms for Virtual Machines Using Modified Simulated Annealing Algorithm

by

Hang Qian

Directed by

Zhuzhong Qian

Department of Computer Science
Nanjing University

May 27, 2012

Submitted in full fulfilment of the requirements
for the degree of Bachelor of Science in Computer Science.

南京大学本科生毕业论文中文摘要

毕业论文题目：基于模拟退火算法的虚拟机部署技术

计算机科学与技术系 院系 计算机科学与技术 专业 2008 级本科生

姓名：钱行

指导教师（姓名、职称）：钱柱中、副教授

摘 要

虚拟机部署问题是整个云计算的核心。能否高效率高性能的部署虚拟机请求和服务提供商的额成本和服务质量直接相关。虚拟机部署问题本质是一个多维装箱问题，该问题已经被证明属于 NP-Hard 问题。使用确定性算法解决虚拟机部署问题常常需要耗费大量的计算时间。本文描述了一种基于现实情境的部署需求，构造了一个基于模拟退火算法的启发式部署算法。从实验结果表明，本文方法在复杂需求下取得了比传统方法 (FFD) 更好的效果。

关键词： 云计算，虚拟机部署问题，模拟退火算法

南京大学本科生毕业论文英文摘要

THESIS: **Placement Algorithms for Virtual Machines Using Modified Simulated Annealing Algorithm**
DEPARTMENT: **Department of Computer Science**
SPECIALIZATION: **Computer Science**
UNDERGRADUATE: **Hang Qian**
MENTOR: **Zhuzhong Qian**

Abstract

The problem of virtual machine placement is at the core of cloud computing. The VM place problem is a reality extend of Bin Packing Problem, who had been proved to be NP-Hard problem. Using deterministic algorithms would cost vastly. In this paper, we proposed a modified Simulated Annealing alheuristic algorithm to solve a real-based VM placement problem. Experimental results shows that our methods achieves better performance than existing method on certain aspects.

Keywords: Cloud Computing, Virtual Machine Placement,Simulated Annealing

目　　录

插　　图

第一章　引言

1.1　云计算

云计算 (Cloud Computing) 是这两年非 “火热” 一个概念。简单说，云计算是一种基于互联网的计算方式，通过这种方式，包括内存，计算能力 (CPU 资源), 带宽等资源可以分配给有需求的用户。互联网上的云计算服务特征和自然界的云、水循环具有一定的相似性，因此，云是一个相当贴切的比喻。通常云计算服务应该具备以下几条特征:

- 基于虚拟化技术快速部署资源或获得服务
- 实现动态的、可伸缩的扩展
- 按需求提供资源、按使用量付费
- 通过互联网提供、面向海量信息处理
- 用户可以方便地参与
- 形态灵活，聚散自如
- 减少用户终端的处理负担
- 降低了用户对于 IT 专业知识的依赖

云计算环境可以分为 “软件即服务”(SaaS)、“平台即服务”(PaaS)、“架构即服务”(IaaS)。这些服务通过虚拟化技术提供给用户。目前比较有名的相关服务有 Amazon EC2,GoGrid，Windows Azure 和 Rackspace Cloud。

1.2　虚拟资源的部署

在云计算中，用户使用的是部署在一系列实体机之上的虚拟机，通常的实现形式是将虚拟机部署在大规模的计算集群中。在一台实体机 (PM，下同) 上可能存在多个虚拟机。因此，我们希望能够以最优的方式部署计算资源，从而尽可能的减少使用实体机的数量，达到节省成本 (主要是能源成本) 的目的。

在部署虚拟机湿，需要考虑的资源有实体机的内存、带宽、计算能力 (CPU) 等因素。这个问题可以被视为一个多维装箱问题 (multi-dimensional vector bin packing problem)。VM 所需要的资源被视为一个 d 维向量，其中每一维都是一个非负的值 (即 “小球”，一般取 0 – 1 的值)；而每个 PM 所拥有资源也可以被看作一个 d 维向量，其中每一维，和 VM 的资源请求一样，代表一个独立的资源 (即 “箱子”，一般取值为 1)。我们的目标是尽可能的减少 “箱子” 的数量并且能够满足将所有的 VM 请求部署在 PM 资源上且保证每个 PM 的上某一维上的

请求都没有超过“箱子”的容量。因此，资源分配问题就可以视为一个 d 维装箱问题。

1.3　问题定义

我们现在将所述问题进行数学定义:

定义 1.1 ***Vector Bin Packing problem****(VBP)*

给定一个有 n 个元素的 d 维向量的集合 S,$S = \{p_1, p_2, \cdots, p_n \mid p_i \in [0,1]^d\}$, 寻找一个 S 上的划分 $A_1, A_2, \cdots, A_m$ 使得 $\sum_{p \in A_i} p^k \leq 1, \forall i, k$

VBP1.1问题已经被证明是一个 NP-Hard 问题 [7]。

1.4　动机

在很多企业提供的 VPS(Virtual Private Server, 虚拟专用服务器) 服务中，提供给用户的选择很多都是不同的处理器性能和内存的大小。在实际的虚拟机部署中，除了资源分配的算法问题之外，还需要考虑如网络拓扑设计等许多方面的问题。但其中最关键也是最核心的问题还是如何根据用户的请求部署、移动虚拟机。在虚拟机部署考虑的资源中，计算资源和内部存储资源是最关键的，而外部存储资源 (硬盘) 等由于其成本相对较低，使用相对不频繁，因此通常并不做主要因素。

本文主要关注在理想环境下仅考虑计算资源 (处理器核心) 和内存的需求进行虚拟机部署的算法研究。

第二章　相关工作

VBP 问题

一维装箱问题已经被深入研究。Fernandez de la Vega 和 Lueker[12] 给出了首个线形时间逼近策略 (APTAS)。他们的算法随后被 Karmarker 和 Karp[9] 改进，达到了 $(1 + log^2)$-OPT上界。

对于二维装箱问题，Woeginger[13] 证明了不存在 APTAS。对于更高维的情况，Fernandez de la Vega 和 Lueker 拓展了一维时的算法，提出了一种 $(d + \epsilon)$-OPT的算法。Chekuri 和 Khanna[2] 提出了一个 $O(log\ d)$ 近似算法，对于给定的 d, 算法运行在线形时间复杂度内。Bansal 等 [10] 改进了这个结果，提出了一个对于任意 $\epsilon \geq 0$ 的 $(ln\ d + 1 + \epsilon)$ 逼近算法。Karger 等 [4] 提出了一种对于多维随机样例的 VBP 问题使用技术的线形逼近方法。

虚拟机部署问题

虚拟机部署问题是整个云计算的核心问题，有很多研究都显示了合理部署虚拟机的重要性 [6][11]。许多基于 First Fit Decrease(FFD) 的算法被应用在虚拟机部署问题之中。Verma 等 [1] 提出了一个能够通过尽量减少迁移[1]达到最佳部署的方案。Hyser 等 [3] 提出了一个互动式的再部署方案，用以解决动态情境[2]下的问题。Bobroff 等 [8] 提出了一种预测请求并部署的动态规划算法。Shahabuddin 等 [5] 提出了一种简单的启发式算法。

即使现阶段业界纷纷趋向虚拟化技术，虚拟机部署问题仍研究尚浅。由于该问题的本质是一个 NP-Hard 问题，寻求最佳解效率比较低，为了客服这个问题，寻找一种效率较高并仍能获得比较好的结果的方法，我们提出了一个基于模拟退火算法，并能适应不同部署需求的算法。

模拟退火算法

模拟退火算法是一种基于概率的启发式算法，通常用来求解组合优化问题，寻找全局最优解。对于特定的问题，模拟退火的效率通常会优于全局搜索。Bohachevsky 等证明，模拟退火算法依概率收敛到全局最优点。

[1]迁移是指对已经部署的虚拟机进行在其他实体机上再次部署的操作

[2]动态情境指在算法开始前不能完全得知虚拟机请求的信息，需要根据实时信息进行部署的问题

模拟退火算法的名称来源冶金学专有的名词退火。退火是将材料加热后再以特定速率冷却，目的是增大结晶体的体积。我们将热力学的理论模拟至统计学上，将解空间内每一点相像成空气中的分子；分子的能量是它本身的动能，而解空间的每一点也像分子一样带有能量，以表示对命题的合适程度。演算法先以搜寻空间内任意一点作起始，每一步先选择一个 “邻居”，然后再计算从现有位置到达 “邻居” 的概率。

启发式算法适合虚拟机部署问题

对于虚拟机部署问题, 一个解可以被认为是单个 VM 状态的组合，算法的目标是对组合进行优化，寻找全局最优解。解的优劣可以用评价函数来体现。解的维度是虚拟机请求的规模，但是对于大规模的虚拟机请求，解空间将变的巨大，经典的确定性算法将耗费巨大的计算时间。使用启发式算法，特别是基于概率的模拟退货算法，可以有效的在巨大的解空间中快速逼近全局最优点。

本文剩余部分将以下方式组织: 第三部分将对本文所希望解决的问题进行描述和形式化；第四部分将给出算法的设计思路和实现，并且进行模拟实验得出结果；第五部分将对产生的结果进行分析，并给出算法的评价。

第三章　问题描述

3.1　假设

在本文所关注的情境中，如上文(1.4)所说处理器资源 (CPU，下同) 和内存资源 (RAM, 下同) 是我们考量的两个约束条件。

CPU 资源和 RAM 资源在实际需求中具有不同的特点。

3.1.1　CPU 资源的特点

CPU 资源具有原子性，即可以被分配的 CPU 资源一般是以计算核心 (core) 为单位的，而并非连续分配的。因此我们可以认为一台具有 N_{core} 个核心实体机的计算资源

$$C^{PM} = \{ C_1^{PM}, C_2^{PM}, \cdots, C_{N_{core}}^{PM} \} \tag{3.1}$$

在通常情况下，集群中节点的每个计算核心的计算能力都是相同的，因此我们假设

$$C^{PM} = \{ N_{core} \times C_{core}^{PM} \} \tag{3.2}$$

取 $C_{core}^{PM} = 1$，则可以将一台实体机的计算资源 C^{PM} 由一个非负整数 N_{core} 来表示。同样，每个对计算资源的请求可以用一个非负实数 c_{req} 来表示。而由于 CPU 资源的原子性，每个 CPU 请求 c_{req} 应该分配 $\lceil c_{req} \rceil$ 个核心。

3.1.2　RAM 资源的特点

RAM 资源具有可共享的特点，即多个虚拟机可以共享一段内存 (严格的说，应当是分时的使用这一段内存，即在每一时刻如果某一段内存被一 VM 占用，则其他 VM 则不能访问这段内存)。在虚拟机资源的经典情境中，当一个 PM 尚未分配的 RAM 资源小于某一个 VM 的 RAM 请求时，该 VM 就不能被部署在这个 PM 上。而如果我考虑 RAM 资源可以被动态分配的特点，则可以满足 PM 上所有 VM 对 RAM 资源的请求总和大于该 PM 所拥有的 RAM 资源，即

$$\sum_i r_i^{req} \geq R^{PM} \tag{3.3}$$

我们相信，这样的模型假设虽然简单，但是是合理的。比如在现代计算机系统中存在的虚拟内存就是内存可动态分配的一种体现。我们作出这样假设的前提是 VM 并不总是在使用它所请求的**所有**资源。所以，不可避免的，会出

现一台 PM 上所有 VM**实际使用的** RAM 总和大于其所拥有的 RAM 总和, 即 $\sum_i r_i^{util} \geq R^{PM}$, 我们称这样的情况为 “冲突”。当冲突发生时，由于需要进行换页等操作来满足用户的需求，相当于牺牲了服务质量，可以认为付出了一定的代价。所有我们希望算法能够将此代价控制在一个阈值内。

所以，问题的输入为一个由资源请求组成的集合 V，其每个元素由三个值组成 $v_i =< c_i, r_i, \mu_i >$, 分别表示对 CPU 资源和 RAM 资源的请求以及 RAM 资源的平均利用率。另有一个由 PM 资源组成的集合 P, 其每个元素为一个值对 $p_j =< C, R >$，分别表示所拥有的 CPU 资源和 RAM 资源，且 C,R 均为常数; 问题的输出是 V 上的一个划分，并且每一个划分都满足一定的条件。

3.1.3　评价函数

在使用启发式算法求解问题时，需要对解空间内的每个解进行评价 (Value)，并根据评价来确定解的优劣。在经典的装箱问题中，求解的目标是尽可能的减少使用实体机的数量，因此计算一个解中使用 PM 的数量 (即产生划分的个数) 可以作为评价函数。

经典的装箱问题中，希望尽可能减少 “箱子” 数量的原因是为了降低成本。但是我们注意到，减少使用的 PM 数量所减少的成本通常是能源成本。而一个计算集群的成本还与服务质量有关，比如通信的质量，故障率等因素。在本文所描述的情境中，我们希望能同时关注成本与性能的问题。

我们定义了以下三方面的问题:

- **降低成本**。与经典的装箱问题相同，我们希望部署的结果在保证质量的前提下能尽可能的降低成本，即减少 PM 的使用。一台 PM 产生的成本，取决于是否有虚拟机被部署于其上，而可以认为与其上的负载没有关系 (即一旦启动则认为成本一定)，所以我们仍旧采用计算 PM 的数量来作为衡量解优劣的指标。
- **负载均衡**。对于需要相互通信的计算集群 (比如 Hadoop 集群), 如果单个节点上的负载过高, 则有可能会成为整个系统的性能瓶颈。我们希望部属的结果能尽可能的保证每个节点上的负载相对均衡。
- **减少冲突**。由于我们采用了可以共享内存的模型，所以会出现一台 PM 上部署的 VM 的 RAM 请求之和大于 PM 自身拥有的 RAM 总和，因此有一定概率出现 “冲突” 的情况。我们希望部属的结果能尽可能的保证整个系统的冲突概率尽可能的减少。

由于这三方面的问题都和具体的应用情境相关，因为我们无法给出之间的具体权重。我们将会单独对三方面问题进行讨论，然后再对综合情况的结果加

以探讨，以期给出一个相对开放的结论。

3.2　虚拟机部署问题

下面对虚拟机部署问题基于上述假设描述的情境进行形式化。

变量表

表 3.1　变量表

变量名	说 明
x_{ij}	VM_i 是否装入 PM_j
N_v	虚拟机的数量
N_p	实体机的数量
VM_i	虚拟机请求 i
PM_j	实体机 j
C_i^{req}	VM_i 的 CPU 请求
R_i^{req}	VM_i 的 RAM 请求
ER_i	VM_i 的 RAM 平均使用率
C_j^{used}	PM_j 已被分配的 CPU
R_j^{used}	PM_j 已被分配的 RAM
C^T	PM_j 所拥有的 CPU
R^T	PM_j 所拥有的 RAM

* x_{ij} 为一位二进制变量
** N_p 理想情况下应为无穷大

约束条件

上述变量(3.1)应当满足如下的约束条件:

$$N_v \leq N_p \tag{3.4}$$

$$x_{ij} = \{1, 0\} \tag{3.5}$$

$$\sum_j x_{ij} = 1 \tag{3.6}$$

$$\sum_i C_i^{req} x_{ij} \leq C^T, \forall j \tag{3.7}$$

$$\sum_i R_i^{req}\, ER_i\, x_{ij} \leq R^T, \forall j \tag{3.8}$$

这些约束条件的涵义如下:

- 公式 (3.4)说明系统可以提供的 PM 的数量总多于 VM 的请求数[1]
- 公式 (3.5)表示一个 VM 请求的两种状态:1 表示被部署在 PM_j 中，0 表示没有被部署在 PM_j 中
- 公式 (3.6)说明一个 VM 能且只能被部署在一个 PM 上。
- 公式 (3.7)和公式 (3.8)说明每一个 PM 上所有 VM 的资源请求不能超过自身所拥有资源[2]

目标

算法的目标是找到这样的一个解: 在满足上述约束条件的限制下，评价尽可能 “低”[3]。由于我们将会讨论多个评价函数下算法的表现，因此用 $value(s)$ 表示对解 s 的评价。

[1]在实际情境中，虚拟机数量一般大于最终使用的实体机数量。实际上，我们应当定义实体机数量趋近无穷 ($0 \leq N_p \to +\infty$)，即不对实体机数量作约束。而在算法实现中，我们需要保证算法能够产生至少一个有效解，因此约定 $N_v \leq N_p$, 并且在后文的实现中取不同的数据规模进行分析。

[2]注意到公式 (3.8)限制每个 PM 上的所有 VM 的**平均** RAM 资源请求，是因为 RAM 的可以共享性质。所以实际解中是会出现 $\sum_i R_i^{req} x_{ij} \geq R^T$ 的情况的。

[3]这里使用 “低” 是指采用的评价函数的值尽可能的小。由于本文采用的是模拟退火算法求解，评价函数的值越低表明系统的能量越小，即更靠近最优解。

第四章　算法的设计与实现

4.1　设计

模拟退火算法是一个可以被高度抽象的算法。它的设计并不受制于所需要求解的问题 (但是参数的设置需要根据问题的性质来设定)。我们设计的算法主要框架如下:

Algorithm 1: Simulated Annealing

INPUT : List si which is a permutation of ids of PMs
OUTPUT: List so which is another permutation of ids of PMs
begin
    cur_state $\leftarrow$ si
    cur_value $\leftarrow$ `value`(si)
    temperature $\leftarrow$ $InitTemperature$
    term $\leftarrow$ 0
    while term $<$ $maxTerm$ & temperature $>$ $minTemperature$ **do**
        neighbor $\leftarrow$ `get_neighbor`(cur_state)
        new_value $\leftarrow$ `value`(neighbor)
        if `accept`(cur_value,new_value,temperature) **then**
            cur_state $\leftarrow$ neighbor
            cur_value $\leftarrow$ new_value
        term $\leftarrow$ term $+$ 1
        temperature $\leftarrow$ `cooldown`(temperature,term)
    return cur_state

算法的输入是一个 “状态”S，表示形式为一个列表，表中的元素 S_i 表示 VM_i 被部署到 PM_{S_i}。一个状态即代表问题的一个解。

算法的输出同样是一个状态，这是经过模拟退火算法迭代过后的新状态，它收敛到一个局部最优解。

4.1.1 初始化

由于算法对初始状态并不敏感，因此我们采用了 `random_fit`的方法生成初始状态，其算法描述如下:

Algorithm 2: Random Fit

INPUT : List V which is the VMsŕequests infomation
OUTPUT: List s which is a state(solution)
begin
  state ← [ ]
  plist ← $EmptyPmInfoList$
  for $everyvm_i \in V$ **do**
    pi ← `randomInt(`*0*`,length(plist))`
    p ← plist [pi]
    repeat
      pi ← `randomInt(`*0*`,length(plist))`
      p ← plist [pi]
    until $available(vm_i$, p)
    `assign(`vm_i`,`p`)`
    state [i] ← pi
  return state

算法2随机的在提供的 PM 序列里寻找可以 (满足约束条件) 部署的节点。直到所有 VM 均被成功部署后返回解状态。

4.1.2　找寻邻解

算法1中使用的 `get_neighbor(state)`过程用来从当前状态生成一个相邻的状态，找寻邻解的算法描述如下:

Algorithm 3: Get Neighbor

INPUT : Current state s

OUTPUT: Neighbor state $neighbor$

begin

neighbor ← state

for 0 **to** max **do**

src ← `randomInt(`0,N_v`)`

des ← `randomInt(`0,N_p`)`

v ← VM_{src}

p ← PM_{des}

repeat

des ← `randomInt(`0,N_p`)`

p ← PM_{des}

until `available(v,p)`

neighbor [src] ← des

return neighbor

在算法3中，我们对当前状态中的 max 个值进行了重新随机部署，max 的取值决定了生成的邻解和当前解的距离。在模拟退火算法中，我们希望邻解有较高的随机性但是与当前解的距离不能太远，因此在实现中通常选 $\frac{1}{10}$ 比例的值。

4.1.3　接受条件

算法1中使用的 `accept(cur,new,t)`用来判断是否接受当前邻点。根据 Metropolis 准则，金属粒子在温度 T 时趋于平衡的概率为 $e^{-\frac{\triangle E}{KT}}$, 其中 K 是 Boltzmann 常数。本文所使用的策略即来自这一准则，算法的描述如下:

Algorithm 4: Accept

INPUT : Current value cur, neighbor value new, current temperature t

OUTPUT: Boolean value indicates accept neighbor or not

begin

```
delta ← new − cur
if delta < 0 then
    return true
else
    if e^(−delta/t) > random(0,1) then
        return true
return false
```

在上述算法4中，`random(a,b)`函数随机返回一 a 到 b 之间的实数。

4.2 实现

我们使用了 Python 编程语言对算法进行了实现。下面介绍一些值得注意的部分。

4.2.1 参数的设置

模拟退火算法对参数设置是非常敏感的，包括初始温度的设置，降温的速度，评价函数的设计都会对最后收敛的速度和结果有比较大的影响。经过反复试验，我们确定了以下的参数设置:

- **初始温度**。由于在判断是否接受时计算了 $\Delta value/t$ 的值，因此我们希望能将温度和评价归一化。初始温度被设置为初始评价的 K 倍，$K = N_v N_p$。
- **降温速度**。降温速度太快会导致算法很快收敛，得不到比较理想的结果，而降温速度太慢则会导致收敛过程很长，并且在计算初期接受过于不理想的结果，影响效率。经过试验，使用了 $t_{n+1} = 0.89t_n$ 这样的线形降温速度，收到了良好的效果。
- **评价函数**。在实现中采用了四种因素综合评价的方式。我们考虑了 PM 的数量，RAM 的负载均衡，CPU 的负载均衡和平均冲突率，并以这四个值的乘积作为评价函数的返回值。

4.2.2　评价函数

在所有参数中，评价函数的确定是对算法影响最大的，因为它直接决定了算法的走向。比如我们选取使用 PM 的数量作为评价的标准，那么在迭代的过程中接受那些使用更少 PM 的解。如前文**??**所说，我们希望综合考虑更重因素，所以我们采用了如下的评价函数:

- **PM 的数量**。我们用 `count_active(plist)`函数来评价。函数接受一个根据状态生成的 PM 信息列表，计算其中被 “激活”(有 VM 部属在其上) 的 PM 的个数 (为了不同规模数据间的比较，实际返回的是 PM 的个数与 N_v 的比例。
- **RAM 的均衡负载**。我们用 `ram_stdev(plist)`函数来评价。函数接受一个根据状态生成的 PM 信息列表，计算 RAM 利用率的均方差 (Standard Deviation)。如果 RAM 利用率的均方差的足够小则说明每台 PM 上的 RAM 负载比较均衡。
- **CPU 的均衡负载**。和 RAM 的均衡负载类似，我们用 `cpu_stdev(plist)`函数来评价。函数接受一个根据状态生成的 PM 信息列表，计算 CPU 利用率的均方差。如果 RAM 利用率的均方差的足够小则说明每台 PM 上的 CPU 负载比较均衡。
- **平均冲突率**。我们定义平均冲突率 $\mu \geq 1$, 使用 `count_conficts(plist)`函数来计算。函数接受一个根据状态生成的 PM 信息列表，结算其中 RAM 利用率超过 1 的 PM 的平均 RAM 利用率。如果不存在利用率超过 1 的 PM, 则取值 1.

4.2.3　测试数据的选取

通常，云计算服务提供商提供的用户的选择都是有限的，比如提供给用户选择 “双核 500MB 内存”，通过限制虚拟机请求的数据可以针对性的优化部属算法。在这种情境下，`First-Fit`算法通常能收到良好的效果。

在本文所描述的情境中，为了更好的模拟不同情境下算法的效果，我们没有对请求的数据进行限制 (但是为了避免无解的情况，对数据的上下限进行了规定)。我们尝试了多种测试数据生成的方法，最终确定以下的测试数据:

- 每组测试设计都是**随机**生成的。
- 将 C^T 设置为 8，即认为每个实体机都具有 8 个计算核心；将 R_T 设为 1。
- C_i^{req} 将取 $0.1 - 8.0$ 之间的随机数；R_i^{req} 取 $0.2 - 0.8$ 之间的随机数；ER_i 取 $0.2 - 0.8$ 之间的随机数。生成的数据可以认为即包含运算需求高的请求也包含存储需求高的请求。

- 测试数据的数量取 $100 - 1000$ 间隔 100 的整数，用以模拟不同规模下的请求。

在随机的数据和不同的数据规模上的得到相似测试结果可以从某种程度上证明算法的稳定性和可行性。

第五章　实验结果与结论

通过不同数据规模和多次试验，我们得出了相应的结论，并且与 `First-Fit`算法在同样的数据上的结果进行了比对。

5.1　有效性

下图5.1是算法在 $N_v = 200, N_p = 200$ 时的迭代情况，我们看到在算法最终收敛在一个比较好的解上。在所有测试中算法均能成功收敛在一个解上。

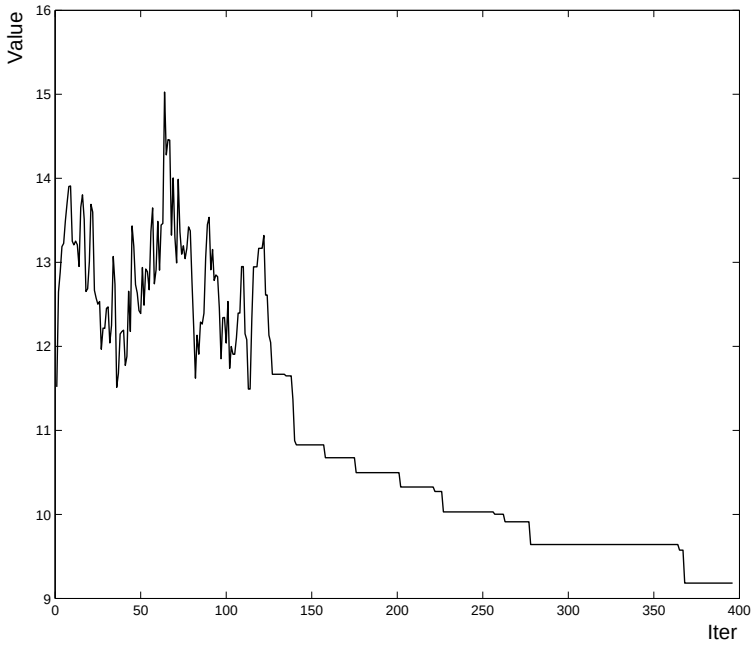

图 5.1　算法收敛

5.2　稳定性

我们已经看到算法可以成功收敛，但是如果每次收敛的结果差距过大则说明算法并不稳定，即不能保证每次都能得到一个较好的解。从下表5.1我们可以看到，当取 $N_v = 100, N_p = 100$ 时，十组不同数据的测试最终收敛的结果非常相近，并且都是较好的解。说明算法具有一定的稳定性，对初值并不敏感。

表 5.1　算法稳定性

	评价	PM 使用率	RAM 均衡负载	CPU 均衡负载	平均冲突率
1	4.019	0.760	0.184	0.270	1.061
2	3.692	0.770	0.241	0.180	1.110
3	4.643	0.810	0.210	0.256	1.064
4	4.052	0.740	0.271	0.174	1.165
5	4.126	0.730	0.254	0.214	1.039
6	4.360	0.740	0.245	0.222	1.081
7	3.609	0.810	0.165	0.266	1.012
8	3.145	0.790	0.166	0.240	1.000
9	4.433	0.770	0.220	0.251	1.045
10	3.442	0.770	0.194	0.230	1.000

[1] $N_v = 100, N_p = 100$
[2] Using Simulated Annealing Algorithm 1

5.3　结果

我们选取了 100 到 1000 步长为 100 的十组数据规模，分别使用模拟退火算法1和 `First-Fit`算法对同一组数据进行计算，并取 10 次不同数据的平均值得到了下表5.2:

5.4　结论

从实验获得的数据5.2中我们可以得到以下结论。

模拟退火算法是有效的

First-Fit 算法已经被证明是解决背包问题的一个高效的算法，从得到数据中我们可以看到，我们设计的基于模拟退火算法得到的效果接近于 FF 得到的效果。甚至在某些指标上比如平均冲突率得到比 FF 得到的更好的效果。

First-Fit 算法在减少 PM 的使用上有更好的效果

从 FF 算法和 SA 算法在 PM 使用率这一指标的对比来看，FF 算法有更好的效果，其平均使用率要低 20% 所有，即每 100 个随机虚拟机请求，FF 算法要比本文所使用的算法少使用 20 个左右的实体机。

产生的这样结果有两方面的因素:

表 5.2　测试数据

数据规模	方法	评价	PM 使用率	RAM 均衡负载	CPU 均衡负载	平均冲突率
$N_v = 100$	SA	3.815	0.780	0.207	0.221	1.072
	FF	3.636	0.600	0.321	0.157	1.199
$N_v = 200$	SA	4.138	0.770	0.230	0.220	1.061
	FF	2.943	0.635	0.331	0.120	1.165
$N_v = 300$	SA	4.370	0.773	0.245	0.212	1.087
	FF	2.374	0.587	0.344	0.100	1.182
$N_v = 400$	SA	5.061	0.785	0.239	0.249	1.082
	FF	1.627	0.585	0.293	0.082	1.159
$N_v = 500$	SA	4.768	0.792	0.236	0.234	1.090
	FF	2.022	0.588	0.330	0.089	1.177
$N_v = 600$	SA	4.824	0.797	0.236	0.235	1.095
	FF	2.218	0.607	0.313	0.098	1.189
$N_v = 700$	SA	5.673	0.801	0.249	0.259	1.101
	FF	1.917	0.593	0.320	0.087	1.168
$N_v = 800$	SA	5.411	0.814	0.239	0.253	1.097
	FF	1.935	0.598	0.312	0.089	1.163
$N_v = 900$	SA	5.392	0.793	0.246	0.251	1.101
	FF	1.684	0.573	0.305	0.082	1.170
$N_v = 1000$	SA	5.433	0.796	0.248	0.248	1.109
	FF	1.713	0.584	0.318	0.078	1.180

1. FF 算法可以认为是以使用 PM 的数量作为评价的贪心算法，因为算法总是试图将请求部署在已经使用的 PM 上，只有所有已使用的 PM 都无法满足请求，才会申请新的 PM.
2. 本文所使用的算法的评价函数是综合考量四种不同因素的结果 (四者的乘积), 因此算法总趋向于综合更优的解。而 PM 使用率仅是其中的一项因素。

模拟退火算法在处理 RAM 负载均衡效果较好，而在 CPU 负载均衡效果不如 FF 算法

从获得数据5.2中我们看到，SA 算法得到的解总是有较低的 RAM 负载均衡但 CPU 的均衡负载不如 FF 算法。

产生在这样结果原因主要是由于:

1. FF 算法在判断当前 VM 能否部署在某个 PM 上时是依据约束条件(3.7), 即是否有足够的 CPU 核心剩余可以被分配，这就使得 FF 算法得到的解由 CPU“饱和” 的 PM 构成。如果 C_j^{used}/C^T 均趋近于 1 则反而被认为更加“均衡”。
2. 同样的，本文所使用的算法受到评价函数中其他因素的影响，因此不会趋近于某一个因素更优的解。

评价函数的重要性

从得到的数据和上面的讨论可以看出，评价函数对本文所使用的算法的“优劣” 有很大影响。但是，我们靠什么去评价一个解的优劣呢? 显然不能以评价函数来评价 —我们已经知道在评价函数下是好的了。因此，使用启发式算法来求解实际问题，应当是提出分析问题根据需求给出评价函数的模型，通过算法求解后代回问题观察是否在实际问题中理论上优的解真的有效，根据观察的结果不断的修正评价函数的模型才能获得理想的结果。

5.5　结语

本文提出了一种基于模拟退火算法求解数据中心中虚拟机资源配置问题的方法。实验结果表明，本文所述方法取得了良好的效果，并能根据所需条件的不同调整算法参数达到适应的目的。在本文中，对背景问题做了若干假设。如何在实际应用中通过修改模型取得良好的效果，达到更好的性能和效率，是接下来值得研究的内容。

致　　谢

衷心感谢南京大学计算机科学与技术系钱柱中老师对我的悉心指导和帮助，他为我的选题和思路提供了许多宝贵的意见；衷心感谢南京大学软件新技术国家重点实验室李鑫对本文提出的宝贵的修改意见和建议。

感谢我的父母在对我的支持和鼓励；感谢我的朋友们对我的帮助。

参考文献

[1] P. Ahuja A.Verma and A. Neogi. pMapper. Power and migration cost aware application placement in virtualized systems. 2008.

[2] Sanjeev Khanna Chandra Chekuri. On multi-dimensional packing problems. pages 185–194, Baltimore, Maryland, USA, 1999. Society for Industrial and Applied Mathematics,.

[3] Rob Gardner Chris Hyser, Bret Mckee and Brian J. Watson. Autonomic virtual machine placement in the data center. *HP Labs tech- nical report*, HPL-2007-189, 2007.

[4] Krzysztof Onak David Karger. Polynomial approximation schemes for smoothed and random instances of multidimensional packing prob- lems. pages 1207–1216, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[5] Abhay Chrungoo Johara Shahabuddin, Vishu Gupta, Sandeep Juneja, Sanjiv Kapoor, and Arun Kumar. Stream-packing: Resource allocation in web server farms with a qos guarantee. pages 182–191, Springer Berlin / Heidelberg, 2001.

[6] A.Yumerefendi L.Grit, D.Irwin and J.Chase. Virtual machine hosting for networked clusters: Building the foundations for autonomic orchestration. Washington, DC, USA, 2006. IEEE Computer Society.

[7] R.L. Graham M.R. Garey and D.S. Johnson. Resource constrained scheduling as generalized bin packing. *J. Combinatorical Theory*, 21(3):257–298, 1976.

[8] A. Kochut N. Bobroff and K. Beaty. Dynamic placement of virtual machines for managing sla violations. Washington, DC, USA, 2007. IEEE Computer Society.

[9] Richard M. Karp Narendra Karmarkar. An efficient approxima- tion scheme for the one-dimensional bin-packing problem. *SFCS’ 82: Proceedings of the 23rd Annual Symposium on Foundations of Com- puter Science*, pages 312–320, 1982.

[10] Maxim Sviridenko Nikhil Bansal, Alberto Caprara. Improved ap- proximation algorithms for multidimensional bin packing problems. pages 697–708, WashingtonDC, USA, 2006. IEEE Computer Society.

[11] R.Rajamony R.Bianchini. Power and energy management for server systems. *IEEE Computer*, 37, 2004.

[12] George S. Lueker Wenceslas Fernandez de la Vega. Bin packing can be solved within 1+epsilon in linear time. *Combinatorica*, 1(4):349–355, 1981.

[13] Gerhard J. Woeginger. There is no asymptotic ptas for two- dimensional vector packing. *Information Processing Letters*, 64(6):293–297, 1997.