南京大學

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 26 日

学　　号：　　081221070

论文答辩日期：　　2012 年5 月28 日

指 导 教 师：　　_____（签字）

Solve Virtual Machine Placement Problem Using A Simulated Annealing Based Algorithm

by

Hang Qian

Directed by

Zhuzhong Qian

Department of Computer Science
Nanjing University

May 26,2012

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

近年来，学术界和工业界对云计算给予了特别重要的关注。作为一种新型计算模式，云计算提供了一种更高效的资源利用方式。云计算的成熟及使用很大程度上得益于虚拟化技术的成熟和使用，服务器虚拟化技术将物理服务器（实体机，physical machine, PM）分割为若干逻辑独立的虚拟机（virtual machine, VM），以 VM 为单位进行资源的分配和使用，大大提高了资源利用效率。由于虚拟化技术的引入，虚拟机部署问题成为一个需要首先解决的基本问题，即如何选择 PM 来进行虚拟化并创建 VM 来满足用户的资源需求。

虚拟机部署策略的优劣直接关系到资源的使用效率并影响着所提供服务的服务质量；同时，部署结果也决定了云平台数据中心所需要耗费的运维成本，尤其是能源开销成本。

一般而言，虚拟机部署问题可以归结为经典的装箱问题，该问题已被证明为 NP-Complete 问题。使用确定性算法求解需要耗费大量的时间，不能满足资源调度的实时性需求。

本文基于 PM 的 CPU 资源和内存资源的特点，引入内存资源共享机制，从 CPU 资源和内存资源两个维度考虑虚拟机部署问题。同时，描述了一种综合考虑服务质量和部署成本的部署需求。针对这一新型问题背景，我们采用模拟退火这一启发式算法来解决该部署问题。大量模拟实验结果表明：资源共享机制在保障低冲突率的前提下，可以有效的减少 PM 的使用数量; 可以有效的均衡系统资源的负载，提供更好的服务质量。同时，当问题的需求变动时，算法可以通过修改评价模型来达到重新适应的目的。另外，模拟退火算法也适合并行策略，提高算法的效率。

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: **Solve Virtual Machine Placement Problem Using A Simulated Annealing Based Algorithm**

DEPARTMENT: **Department of Computer Science**

SPECIALIZATION: **Computer Science**

UNDERGRADUATE: **Hang Qian**

MENTOR: **Zhuzhong Qian**

Abstract

Cloud computing is under the spotlight nowadays. As a new type of computing model, cloud computing utilizes resources more effectively. The high pace developing of cloud computing depends on the virtualization technique. A virtualized server is divided into independent virtual machines (VMs). Allocating and using VM means using resources more effectively. A core prolem in virtualization technique is VM placement problem, which means the stretagy of placing VMs into physical machines (PMs).

Therefore, how can we provide high quality service and reduce the cost depends on how well we solve the VM placement problem. Generally, the VM placement problem can be reduced to a classic bin packing problem, which had been proved to a NP-Complete problem. Solving it using a deterministic algorithm may need huge amount of time which usually cannot be afforded to meet the real-time needs.

In this paper, we analyzed the features of CPU resources and RAM resources and introduce a shared-memory scenario. To meet the service quality and cost needs at the same time, we proposed a algorithm based on the simulated annealing algorithm. The experimental result shows that our algorithm diminished the amount of used PMs and achieves a more balanced load which is better than the classic first fit approach generates.

Keywords: Cloud Computing, Virtual Machine Placement,Simulated Annealing

目　　录

插　　图

表　格

第一章　引言

1.1　研究背景

随着信息技术的广泛应用和快速发展，云计算作为一种新兴的商业计算模型得到了人们的广泛关注。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

云计算可以分为 “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。

虚拟化技术的引入为实现云计算的弹性资源供给、按需服务的等特性提供了保障。在云计算中使用的虚拟化技术通常被称为平台虚拟化技术，通过使用控制程序，隐藏特定计算平台的实际物理特性，为用户提供抽象的、统一的、模拟的虚拟环境 (称为虚拟机)。在云计算服务提供商提供给用户的服务是以部署在实体机 (Physical Machine,PM) 之上的虚拟机 (Virtual Machine，VM) 实现的，通常的实现形式是将 VM 部署在大规模的计算集群中。在一台 PM 上可能存在多个 VM。因此，我们希望能够以最优的方式调度计算资源，从而尽可能的减少使用 PM 的数量，达到节省成本 (主要是能源成本)、提高服务质量的目的。

如何高效的部署 VM, 并尽可能利用已有资源是一个重要的问题。在部署 VM 时，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向量装箱问题[1](multi-dimensional vector bin packing problem，已被证明为一个 NP-Complete 问题 [7])。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

[1]多维装箱问题的定义是: 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 $\sum_{p \in A_i} p^k \leq 1, \forall i, k$

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。

1.2　研究动机及本文工作

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 VM 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 VM。在 VM 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。

PM 资源利用率直接关系到云平台的工作效率以及运维成本，本文以提高 PM 资源有效利用率为目标，采用资源复用的思想，使用模拟退火算法解决因复用而产生的冲突问题，并通过实验表明了复用机制的有效性。具体而言，本文工作包括几下几个方面:

1. **分析资源使用特点，引入内存资源复用机制。**
   根据 CPU 使用的原子性以及内存资源的可共享性，我们对提出的 VM 请求进行分别处理：独占式的满足 CPU 请求，以概率共享的方式满足内存资源请求。内存共享机制可以减少内存资源的空置率，从而提高整体资源利用率。
2. **通过概率模式刻画使用冲突并采取模拟退火算法规避冲突。**
   由于概率共享方式的引入，在多 VM 共同使用内存时必然会引起 VM 间对内存使用的冲突。我们采用概率的方式来刻画冲突的强弱程度，并在部署时使得冲突的产生控制在合理的概率范围内。我们所使用的基于模拟退火的算法能够很好的做到这一点。
3. **进行大量实验，给出了资源复用机制在提高资源利用率方面的有效性**
   针对该问题背景，我们提出了考虑所使用的 PM 数量、负载均衡、冲突概率等多因素在内的效果评价体系。通过大量的模拟实验，我们发现：在牺牲少量服务质量（由冲突引起）的前提下，引入内存共享机制可以有效的提高整体资源利用率。另一方面，实验结果也表明了模拟退火算法在控制冲突方面有着优异的表现。

1.3　本文的组织结构

本文的剩余部分将按以下方式组织: 第二部分将对相关工作进行介绍；第三部分将对本文所希望解决的问题进行分析以及形式化描述并介绍算法的设计；第四部分将对设计的算法进行实现，模拟实验得出结果并对产生的结果进行分析，并给出算法的评价; 最后一部分进行总结并展望未来工作。

第二章　相关工作

2.1　虚拟机部署问题

VM 部署问题是整个云计算的一个基本问题，有很多研究都显示了合理部署 VM 的重要性 [6][11]。许多基于 First Fit Decrease(FFD) 的算法被应用在 VM 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，VM 部署问题仍研究尚浅。由于该问题的本质是一个 NP-Complete 问题，寻求最佳解效率比较低，为了克服这个问题，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。

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，将解空间内每一点想像成空气中的分子；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。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。

模拟退火算法在解决虚拟机部署问题时的优势

对于 VM 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 VM 请求，解空间将变的巨

[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

大，经典的确定性算法将耗费巨大的计算时间和空间。使用启发式算法，特别是基于概率模型的模拟退火算法，可以有效的在巨大的解空间中快速逼近全局最优点。

2.2　装箱问题

VM 部署问题的本质是装箱问题。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(Vector Bin Packing，向量装箱) 问题使用技术的线形逼近方法。

第三章　基于内存共享的虚拟机部署技术

3.1　资源特点分析

在本文所关注的情境中，如上文所说处理器资源 (CPU) 和内存资源是我们考量的两个约束条件。CPU 资源和内存资源在实际需求中具有不同的特点。

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。

内存资源的特点

内存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 尚未分配的内存资源小于某一个 VM 的内存请求时，该 VM 就不能被部署在这个 PM 上。而如果我考虑内存资源可以被动态分配的特点，则可以满足 PM 上所有 VM 对内存资源的请求总和大于该 PM 所拥有的内存资源，即

$$\sum_i R_i^{req} x_{ij} \geq R^T, \forall j \tag{3.3}$$

其中 R_i^{req} 是某个虚拟机的内存请求，x_{ij} 表示虚拟机 VM_i 是否被部署在实体机 PM_j 上，R^T 是一台 PM 所拥有的内存。

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**实际需要使用的**内存总和大于其所拥有的内存总和, 即, 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。

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，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指标:

- **减少 PM 的使用**。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 的内存请求之和大于 PM 自身拥有的内存总和，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　问题描述

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。表3.1给出了问题描述中所使用到的变量。

表 3.1　变量表

变量名	说 明
x_{ij}	VM_i 是否装入 PM_j
N_v	VM 的数量
N_p	PM 的数量
VM_i	VM 请求 i
PM_j	PM 资源 j
C_i^{req}	VM_i 的 CPU 请求
R_i^{req}	VM_i 的内存请求
γ_i	VM_i 的内存平均使用率
C_j^{used}	PM_j 已被分配的 CPU
R_j^{used}	PM_j 已被分配的内存
C^T	PM_j 所拥有的 CPU
R^T	PM_j 所拥有的内存

* 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:

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

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

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

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

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

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, 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 ($0 \leq N_p \rightarrow +\infty$)，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。
- 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。注意到公式 (3.8)限制每个 PM 上的所有 VM 的**平均**内存资源请求，是因为内存的可以共享的性质。所以实际解中是会出现 $\sum_i R_i^{req} x_{ij} \geq R^T$ 的情况的。

问题描述

问题的输入是 N_v 个 VM 请求组成的集合 V，其中每个请求 VM_i 拥有三个参数 C_i^{req},'R_i^{req} 和 γ_i；问题的输出是找到满足约束条件(3.4)的 V 上的一个划分。我们把一个合法的划分成为问题的一个解 (solution, 简称 s)。算法的目标是找到一个评价尽可能 “低”。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。的解。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。

3.3　算法设计

模拟退火算法被证明是解决组合优化问题的一个高效算法。在本文描述的问题中，算法每次迭代后的结果可以被认为是 VM 状态的组合，而算法的目标

是将这个组合进行优化。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 ← si
    cur_value ← value(si)
    temperature ← InitTemperature
    term ← 0
    while term < maxTerm & temperature > minTemperature do
        neighbor ← get_neighbor(cur_state)
        new_value ← value(neighbor)
        if accept(cur_value,new_value,temperature) then
            cur_state ← neighbor
            cur_value ← new_value
        term ← term + 1
        temperature ← 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。

算法是一个迭代过程，迭代终止的条件是达到最大迭代次数，或系统温度低于最低温度。每次迭代过程都根据当前解生成一个邻居解，并根据二者的能量差和当前温度以一定概率接受这个解。算法启动时，由于不存在 “当前” 状态，我们需要一个初始化的过程来生成一个初始解。

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`的随机方法生成初始状态，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 *every* $vm_i \in V$ **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试探性的随机尝试部署某个虚拟机，一旦成功 (满足约束条件) 便将部署信息写入状态序列，直到所有请求都被成功部署。在算法的迭代过程中，我们需要找寻当前解的相邻解。相邻解的选取应当是随机的，这样算法才能在迭代初期给跳出局部最优解。同时相邻解应当和当前解相隔不远，因为一旦相邻解和当前解距离过大，则丧失了当前解带来的好处，最后很难收敛道较好的解。

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。

在找寻邻解3和初始化2的过程中，我们都用到了函数 `available(v,p)`。这个函数的作用是判断将 VM 请求 `v`部署在 PM 资源 `p`上是否满足约束条件3.7和约束条件3.8。

引入约束条件的必要性

对于经典的模拟退火算法来说，根据评价函数的不同，算法有可能停止在解空间内的任意一点，并不需要对解本身做约束。然而在本文描述的情境中，我们关注的是多方面的因素，因此有可能出现一些算法认为很好，但实际上不合理的解的情况。针对这个问题，一个方法是修改评价函数的模型，引入更复杂、分情况的讨论；另一种方法对多个因素中的某些因素进行阈值的设定，达

到缩小解空间 (剪枝) 的效果。所以引入约束条件后，相当于在较小的解空间进行启发式搜索的，有利于提高算法的效率。

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 $e^{-\frac{\Delta E}{KT}}$, 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 $\leftarrow$ new $-$ cur
  if delta < 0 **then**
    return true
  else
    if $e^{-\frac{\text{delta}}{t}} >$ `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。从中我们可以看到，算法进行的初期，t相对较大，则 $e^{-\frac{\Delta E}{KT}}$ 趋近于 1，这时算法会接受一些比较差的相邻解；随着算法的进行，t逐渐冷却，$e^{-\frac{\Delta E}{KT}}$ 趋近于 0，这时算法慢慢接受比当前解更差的相邻解。所以，算法的迭代过程是一个对相邻解逐渐挑剔的过程，当接受概率等于 0 时，算法退化成一般的 “爬山法”，只接受随机产生的比自己更好的解了。

第四章　实验结果与分析

4.1　算法实现

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

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。

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 部署在其上) 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.

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；E_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。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。

4.2　有效性

图4.1是算法在 $N_v = 200, N_p = 200$ 时的迭代情况，我们看到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。

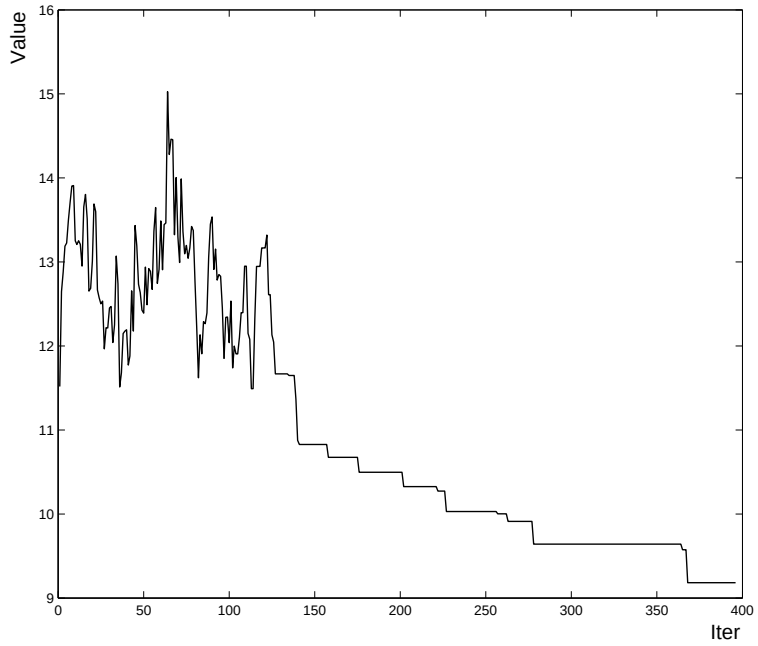

图 4.1　算法收敛

4.3　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。从表4.1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。

4.4　实验结果

我们选取了 100 到 1000 步长为 100 的十组数据规模，分别使用 SA 算法1和 FF 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表4.2:

从实验获得的数据4.2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，从得到的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要好于 FF 得到的结果。

表 4.1　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

共享内存有效减少了 PM 的使用

从表4.2中的数据我们可以看到，采用了内存可共享模型的 SA 算法可以有效减少 PM 的使用，比没有采用内存共享模型的 FF 平均要少 12% 左右。同时，由于我们把平均冲突率作为因素之一引入了评价函数，算法可以将平均冲突率控制在 1.1 左右，即如果一台 PM 上的内存使用发生了冲突，冲突内存平均只占 PM 总内存的 10%，这个代价通常是可以被接受的。

SA 算法能够获得负载均衡的解

从表4.2中的数据我们可以看到，我们提出的算法可以有效控制系统的负载均衡，取得了比 FF 算法更好的结果。在实际问题中，更均衡的负载意味着更少的系统瓶颈，尤其是对网络通信任务比较繁重的系统来说，更均衡的负载可以明显提高服务的质量，减少延迟。

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。

表 4.2　测试数据

数据规模	方法	PM 使用率	RAM 均衡负载	CPU 均衡负载	平均冲突率
$N_v = 100$	SA	0.780	0.207	0.221	1.072
	FF	0.920	0.247	0.297	N/A
$N_v = 200$	SA	0.770	0.230	0.220	1.061
	FF	0.895	0.236	0.293	N/A
$N_v = 300$	SA	0.773	0.245	0.212	1.087
	FF	0.903	0.248	0.277	N/A
$N_v = 400$	SA	0.785	0.239	0.249	1.082
	FF	0.860	0.250	0.285	N/A
$N_v = 500$	SA	0.792	0.236	0.234	1.090
	FF	0.880	0.244	0.287	N/A
$N_v = 600$	SA	0.797	0.236	0.235	1.095
	FF	0.873	0.248	0.287	N/A
$N_v = 700$	SA	0.801	0.249	0.259	1.101
	FF	0.867	0.266	0.293	N/A
$N_v = 800$	SA	0.814	0.239	0.253	1.097
	FF	0.874	0.247	0.297	N/A
$N_v = 900$	SA	0.793	0.246	0.251	1.101
	FF	0.864	0.264	0.293	N/A
$N_v = 1000$	SA	0.796	0.248	0.248	1.109
	FF	0.871	0.271	0.291	N/A

第五章　总结与展望

以提高资源利用率为指导思想，本文进行了以下几方面的研究。首先，通过分析物理资源（CPU，内存）资源的使用特点，我们在传统的虚拟机部署问题背景下引入内存资源共享机制。通过多 VM 进行内存资源的分时复用，可以有效的减少资源的闲置，提高资源的有效利用率。其次，针对共享内存可能引起的冲突问题，通过资源的使用概率来对可能产生的冲突进行描述，而所采用的模拟退火算法能够使冲突的产生降至较小的水平，而通过参数的设定可以杜绝高冲突情况的产生。最后，通过充分的实验验证，和经典的 First-Fit 算法进行对比，我们所提出的基于模拟退火算法在负载均衡和使用的 PM 数量上都有较大优势。

对资源使用的冲突问题是解决该部署问题的关键所在，对冲突的刻画和避免仍有很大的改进空间。首先，相比于目前采用的单一概率模式，资源的使用情况通过使用量的概率分布来描述更加合理。这也带来了多了 VM 使用资源的联合概率分布的描述等问题，需要进一步的后续研究。其次，冲突无可避免的会发生，如何妥善处理冲突发生时各 VM 资源的分配，也是一个需要进一步研究的问题。

提高资源利用率是本文的核心思想，云计算环境下，尤其是大数据中心背景下，存在着比较严重的资源浪费现象。开展该项研究以各种突进提高资源利用率进而减少能源等开销是有意义的，也将继续成为一个重要的研究方向。

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。

感谢我的父母对我一直以来的支持和鼓励；感谢我的朋友们对我的帮助，你们是我大学生活最大的收获。

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.