单位代码：10293　密　级：公开

南京邮电大学

硕士学位论文

论文题目：面向绿色云计算的资源配置及任务调度研究

学　　号	1011041029
姓　　名	曹玲玲
导　　师	徐小龙
学科专业	计算机软件与理论
研究方向	基于网络的计算机软件应用技术
申请学位类别	工学硕士
论文提交日期	二零一四年二月

南京邮电大学学位论文原创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得南京邮电大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

本人学位论文及涉及相关资料若有不实，愿意承担一切相关的法律责任。

研究生签名：黄培培　日期：2014.4.2

南京邮电大学学位论文使用授权声明

本人授权南京邮电大学可以保留并向国家有关部门或机构送交论文的复印件和电子文档；允许论文被查阅和借阅；可以将学位论文的全部或部分内容编入有关数据库进行检索；可以采用影印、缩印或扫描等复制手段保存、汇编本学位论文。本文电子文档的内容和纸质论文的内容相一致。论文的公布（包括刊登）授权南京邮电大学研究生院办理。

涉密学位论文在解密后适用本授权书。

研究生签名：黄培培　导师签名：[illegible]　日期：2014.4.2

Research on Resource Allocation and Task Scheduling for Green Cloud Computing

Thesis Submitted to Nanjing University of Posts and Telecommunications for the Degree of

Master of Engineering

By

Lingling Cao

Supervisor: Prof. Xiaolong Xu

February 2014

摘要

随着云计算技术的迅速发展，使得云数据中心服务器的规模每年都在不断的扩大，产生巨大的能源开销。不合理的调度策略同样导致能源浪费严重，使得云数据中心的运营成本不断增加。因此，绿色云计算的概念应运而生。针对目前云数据中心的高能耗、低效率问题，本文围绕着绿色云计算系统模型的构建、资源配置和任务调度算法的设计及验证等工作展开了一系列的研究与探索，主要包括：

首先，从实现绿色云计算的内部和外部因素出发，设计了一种高效可靠的绿色云计算系统模型。该模型重点着手于系统内部的资源配置模块和任务调度模块，针对这两个子模块的实际需求，给出了具体的实施方案。资源配置模块使得系统资源的分配更加合理，提高了系统资源的利用率，降低了云计算系统整体的能源消耗；任务调度模块能够保证云计算系统具有合理的任务调度机制。

其次，从全局角度出发，抽象任务调度问题为资源配置问题。通过预测用户请求的负载大小，并结合当前系统状态和资源分布，采取保守控制策略，计算下一个周期内任务对系统资源的需求量。然后，建立满足多目标约束的能耗模型，提出基于概率匹配的资源配置算法，实现低能耗资源配置。在此基础上，提出基于改进型模拟退火的资源配置算法，进一步降低系统能耗。实验结果表明，采取的预测与控制策略能够有效避免资源配置滞后于用户请求的问题，提高系统的响应比和稳定性；提出的资源配置算法能够激活更少主机，实现激活主机集合之间更好的负载均衡和资源的最大化利用，降低云计算系统能耗。

最后，为了设计合理的任务调度机制，提出一种面向云计算平台任务调度的多级负载评估方法。该方法为绿色云计算系统的任务调度提供一种合理的负载评估方法。在此基础上，提出一种基于动态负载调节的自适应云计算任务调度策略，任务节点能够自适应负载的变化，按照计算能力获取任务，实现各个节点自调节，同时避免因采用复杂的调度算法，使得管理节点承载巨大的系统开销，成为系统性能瓶颈。实验结果表明，基于动态负载调节的自适应云计算任务调度策略高效可靠，适用于异构云计算集群的计算环境，性能优于现有的任务调度策略。

关键词： 绿色计算，云计算，节能，资源配置，任务调度

Abstract

With the rapid development of cloud computing, the servers' scale of cloud data center is constantly expanding every year, which causes huge power consumption. Furthermore unreasonable scheduling policies lead to energy waste, making the cloud data center operating costs continuely increase. To address the issue of high energy consumption and low efficiency of cloud computing, this thesis is conducted a series of research and exploration, mainly includes:

Firstly, a highly efficient and reliable green cloud computing system model is designed. This thesis focuses on resource allocation module and task scheduling module. Aimed at the actual needs of the two modules, implementation scheme is given. Resource allocation module makes system resource allocation more reasonable, improves the utilization rate of system resources and reduces the overall energy consumption of cloud computing system. And task scheduling module is to ensure a reasonable task scheduling mechanism for cloud computing system.

Secondly, abstract the task scheduling problem to the virtual machine deployment issue with the virtualization technology. The demand of resources for the next period of task requests is calculated by combining with a conservative control strategy. Then, a multi-objective constrained optimization model of power consumption is established, and a low-energy resource allocation algorithm based on probabilistic matching is proposed. Based on this algorithm, another low-energy resource allocation algorithm is designed with the improved simulated annealing algorithm, in order to reduce power consumption further. Experiment results show that the prediction and conservative control strategy make resource pre-allocation catch up with demands, while improving real-time response ratio and stability of the system. Two algorithms can both activate fewer hosts, achieve better load balance among the set of high applicable hosts and maximum utilization of resources, and greatly reduce power consumption of cloud computing systems.

Finally, a method called multi-level load assessment method for cloud computing task scheduling is proposed for green cloud computing task scheduling. On this basis, a novel strategy is presented, called adaptive task scheduling strategy based on dynamic workload adjustment. With this strategy, task nodes can adapt to the changes of load at runtime, and obtain tasks in accordance with the computing ability of each node that realizes self-regulation, while avoiding the complexity of algorithm, which is the prime reason making the master node be the system performance bottleneck. Experiments show that the strategy is a highly efficient and reliable algorithm, which

makes the heterogeneous cloud clusters more stable, faster and load balancing. Furthermore, its performance is superior to current task scheduling strategy.

Key words: Green Computing, Cloud Computing, Energy Saving, Resource Allocation, Task Scheduling

目录

专用术语注释表

符号说明：

S_i	传感器节点
R_i	节点感知半径
$C_{s_i}(t_j)$	目标点 t_j 相对于传感器节点 S_i 的覆盖率
$d(s_i,s_p)$	目标点 S_i 相对于传感器节点 S_p 的距离
V(T)	目标区域 T 的面积
p_{ij}	节点 S_i 对监测区域内目标 j 的感知概率
μ	覆盖影响因子
$\varphi_s(w)$	曲线上的点 w 被传感器节点集合 S 覆盖
$C_s(L)$	路径曲线 L 被传感器节点集合 S 覆盖的覆盖率
$Ef(S_i,w)$	节点 S_i 对监测区域内目标 w 的衡量函数（有效函数）

缩略词说明：

QoS	Quality of Service	服务质量
PSO	Particle Swarm Optimization	粒子群优化算法
TIVFA	Target Involved Virtual Force Alogorithm	涉及目标的虚拟力算法
PFCCEA	Potential Field and Coverage-factor based Coverage-enhancing Algorithm	基于虚拟势场和覆盖因子算法
PRMCA	Probability model based Rotate or Move along fixed direction Coverage-enhancing Algorithm	基于概率模型可转动或定向移动的覆盖增强算法
PMCCA	Coverage Control Algorithm Based on Probability Model	基于概率模型的传感器节点覆盖控制算法
MTPCA	Moving Target based Path Coverage-enhancing Algorithm	基于移动目标的路径覆盖算法

第一章 绪论

1.1 课题背景和研究意义

云计算（Cloud Computing）是目前计算机领域研究的热点，它将计算任务调度到由大量计算和存储资源节点构成的资源池上，使用户能按需获取计算力、存储空间和信息服务[1-2]。云计算本身是节能的，例如通过虚拟化技术，有效地整合资源，提高资源利用率；通过关闭/休眠技术，降低空闲能耗，实现能耗的降低。随着云计算技术的迅速发展，使得数据中心发生了巨大的变革，产生了新一代的数据中心，称之为云数据中心。云数据中心包含大量服务器，并且这些服务器的数量每年都在不断的增加，即便云计算是节能的，云数据中心每天都在消耗着巨大的能量。据美国国家环境保护局（Environmental Protection Agency，EPA）的数据表明，数据中心的能耗超过了美国能源消耗总量的 3%，占全球能源消耗的 1.5%-2%，并且每年在以 12%的速度增长[3]。《数据中心能效测评指南》显示，我国数据中心总耗电量在 2011 年已达 700 亿 kWh，占当年全国电力消耗总量的 1.5%，相当于 2011 年天津市全年的总用电量[4]。

目前国内外针对绿色计算的研究与应用已经取得了一定的成果，包括能耗模型及评价、低功耗硬件、体系结构、关闭/休眠技术、动态电压调整技术、网络通信等方面；针对云计算的节能技术也有一系列的研究成果。但是仍然存在着问题，例如当前针对数据中心节能的研究存在局限性，如未考虑云数据中心的规模、调节的粒度太粗等。而云数据中心的高能耗，低效率问题已刻不容缓，本文在绿色云计算系统模型、资源配置及任务调度等方面进行了一系列的研究与探索。本文的研究从实现绿色云计算的内部和外部因素出发，设计了一种高效而可靠的绿色云计算系统模型，针对各个子模块的实际需求，给出了具体的实施方案。本文的研究成果能够保证云计算系统具有合理的任务调度机制，使得系统的资源配置更加合理，同时提高了系统资源的利用率，降低了云计算系统整体的能源消耗。

1.2 本人工作

本人所做工作如下：

（1）从实现绿色云计算的内部和外部因素出发，设计了一种高效可靠的绿色云计算系统模型。该模型重点着手于系统内部的资源配置模块和任务调度模块，针对这两个子模块的实际需求，给出了具体的实施方案。资源配置模块使得系统资源的分配更加合理，提高了系统

资源的利用率，降低了云计算系统整体的能源消耗；任务调度模块能够保证云计算系统具有合理的任务调度机制。

（2）从全局角度出发，抽象任务调度问题为资源配置问题。通过预测用户请求，结合当前系统状态和资源分布，采取保守控制策略，计算下一个周期内任务对系统资源的需求量。然后，建立满足多目标约束的能耗模型，提出基于概率匹配的资源配置算法，实现低能耗资源配置。在该算法的基础上，提出基于改进型模拟退火的资源配置算法，进一步降低系统能耗。实验结果表明，预测算法和保守控制策略能够有效避免资源配置滞后于用户请求的问题，提高系统的响应比和稳定性；提出的资源配置算法能够激活更少主机，实现激活主机集合之间更好的负载均衡和资源的最大化利用，降低云计算系统能耗。

（3）为了设计合理的任务调度机制，提出一种面向云计算平台任务调度的多级负载评估方法。该方法为绿色云计算系统的任务调度提供一种合理的负载评估方法。在此基础上，提出一种基于动态负载调节的自适应云计算任务调度策略。任务节点在运行的过程中及时地自适应负载的变化，按照计算能力获取任务，实现各个节点自调节，同时避免因采用复杂的调度算法，使得管理节点承载巨大的系统开销，成为系统性能瓶颈。实验结果表明，该策略高效可靠，适用于异构云计算集群计算环境，性能优于现有的任务调度策略。

1.3 本文组织

本文的内容组织如下：

第一章 绪论。介绍了课题背景和研究意义、本人的主要工作和本文的组织结构；

第二章 相关研究工作。首先介绍了云计算技术的相关概念和技术。随后就绿色计算的定义和发展现状进行了分析和总结。最后介绍了云计算中的节能技术，从虚拟化技术、资源配置、任务调度等方面进行了详细的阐述；

第三章 绿色云计算系统模型。提出了一种绿色云计算模型，并从系统模型、资源配置模型、任务调度模型三个方面进行了重点阐述；

第四章 面向低能耗云计算任务调度的资源预配置算法。提出了一套完整的绿色云计算数据中心的节能机制，首先介绍了负载预测与控制方法、然后建立满足多目标约束的能耗模型，提出了两种面向低能耗云计算任务调度的资源预配置算法，并通过仿真实验证明了这两种算法在降低云系统能耗方面所具有的优势；

第五章 基于动态负载调节的自适应云计算任务调度算法。针对现有任务调度算法存在的问题，首先提出一种面向云计算平台任务调度的多级负载评估方法，接着在此基础上，提出

一种基于动态负载调节的自适应云计算任务调度策略，并通过具体的实验说明了该算法的有效性；

第六章 总结与展望。总结了全文所做的研究工作，并对本课题的进一步研究进行了展望。

1.4 本章小结

本章首先介绍了论文的课题背景和研究意义，然后从三个方面概括了本人的主要工作，最后说明了本文的章节安排，并简要介绍了每章的内容。

第二章 相关研究工作

本章首先介绍了云计算技术的相关概念和技术。随后就绿色计算的定义和发展现状进行了分析和总结，主要包括能耗模型及评价、低功耗硬件、体系结构、关闭/休眠技术、动态电压调整技术、网络通信等。最后介绍了面向云计算系统的节能机制，从虚拟化技术、资源配置、任务调度、数据存储与部署、网络结构和温控设施六个方面进行了详细的阐述。

2.1 云计算技术

“云计算”一词在 2006 年 8 月首次被提出。云计算（Cloud Computing）作为一种新技术，在短短数年间就得到了飞速的发展。各大 IT 巨头们极大力地推动云计算技术和产品的推广和普及，学术界也对其高度认同，学术研究活动如火如荼地展开。本节对云计算技术的定义、体系架构、关键技术、典型的云平台和仿真技术进行了简要概述。

2.1.1 云计算的定义

云计算在近几年得到了快速的发展，但是对于什么是云计算，如何定义云计算，商业界和学术界没有形成统一的定论。在这里，本文给出了一个概括性的定义：云计算通过虚拟化技术将各种软硬件资源有机地整合成一个大规模、可扩展的资源池，以网络为载体提供基础设施、平台、软件服务。用户根据自己的需求，通过网络接入以低廉的价格获取服务和资源。

2.1.2 云计算体系架构

随着云计算技术的不断发展和成熟，各大 IT 厂商根据自身的业务需求和发展，分别提出了各自的云计算解决方案，然而目前仍然未有统一的适用于云计算的体系架构，这在一定程度上阻碍了我们对云计算的理解。美国国家标准与技术研究院（National Institute of Standards and Technology，NIST）提出的云计算参考体系架构[5]如图 2.1 所示。

该体系架构定义了 5 个主要的角色：云用户（Cloud Consumer）、云服务提供者（Cloud Provider）、云载体（Cloud Carrier）、云审计（Cloud Auditor）和云代理（Cloud Broker）。Cloud Provider 负责为用户提供可用的服务，包括云服务部署、云服务编排、云服务管理、安全、隐私模块。Cloud Auditor 可以独立评估云服务、信息系统操作及云实现的性能和安全性。Cloud

Broker 是一个管理云服务的使用、性能和交付的实体，同时协调 Cloud Provider 和 Cloud Consumer 之间的关系。Cloud Carrier 是一个中间层，为云服务提供了连接和传输。

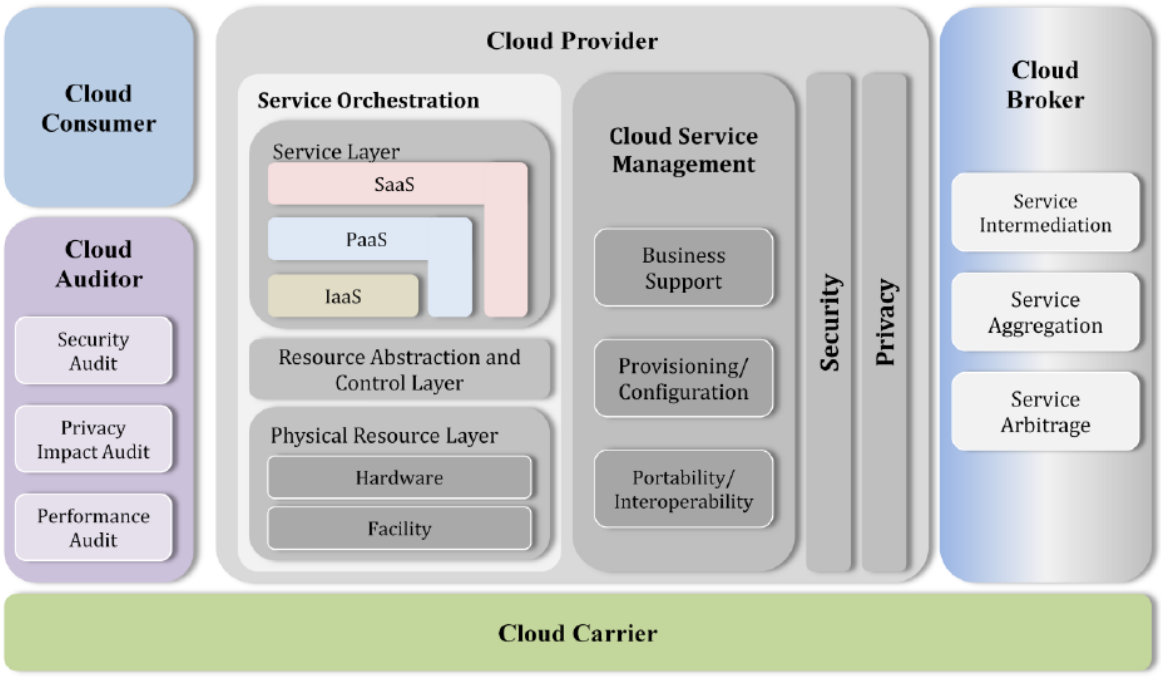


图2.1 云计算体系架构

2.1.3 关键技术

云计算有很多新技术，其中最为关键的技术为编程模型、海量数据存储技术、海量数据管理技术和虚拟化技术。

（1）编程模型

云计算提供了分布式编程模型，最典型的代表是 Google 公司的 MapReduce 模型，它是一种思想简洁的分布式并行编程模型和高效的任务调度模型，主要用于大规模数据集的并行运算和并行任务调度[6]。MapReduce 的主要思想是将要处理的任务分解成映射（Map）和化简（Reduce）任务，先用 Map 函数将大数据集分块，调度到计算节点来处理，进行分布式计算，再用 Reduce 函数汇总中间数据，并输出最终结果。

（2）海量数据存储技术

云系统采用分布式存储技术存储海量数据，采用多副本技术保证数据的可靠性。Google 的文件系统 GFS（Google File System）[7]和基于 GFS 思想开发的 Apache 开源文件系统 HDFS（Hadoop Distributed File System）目前得到了广泛使用。HDFS 是一个分布式文件系统，为存储超大文件而设计，在商用硬件（普通、廉价的硬件）上运行，提供容错功能。HDFS 文件默认被切分为 64MB 的块存储，且每个块在集群中保存 3 个以上的备份进行冗余存储。在

典型的 HDFS 集群中，通常有一个管理节点（NameNode）和多个数据节点（DataNode）。NameNode 管理文件系统的命名空间（NameSpace），它维护着文件系统树以及这棵树上所有的文件和目录，记录存在于 HDFS 中的文件云数据以及各个文件是如何被分块的。DataNode 把数据块存储在本地文件系统中。NameNode 定期通过心跳包与 DataNode 通信，给 DataNode 传递指令并收集它们的状态。

（3）海量数据管理技术

云计算采用高效的数据管理技术管理分布的、海量的数据。较典型的有 Google 的 BigTable[8]和 Apache 的开源数据结构化管理组件 HBase。Hbase 是一个分布式存储系统，可在商用硬件上搭建大规模结构化的存储集群。与关系型数据库不同，Hbase 用于非结构化数据存储。由于它是基于列存储的，所以适合读写大数据。HBase 提供了配套的 TableInputFormat 和 TableOutputFormat API，可将 Hbase Table 作为 Hadoop MapReduce 的数据源，对于 MapReduce 应用的开发人员来说，基本不需要关注 HBase 系统自身的细节。

（4）虚拟化技术

虚拟化技术是指在物理主机上虚拟出若干台虚拟机，通过这种方式能够扩大硬件的容量。通过虚拟化技术可以隔离高层应用与底层硬件，它包括分解模式和聚合模式，分别对应将单个资源划分成多个虚拟资源和将多个资源整合成一个虚拟资源。在云计算实现中，虚拟化建立在云服务与云应用的基础之上，主要用在操作系统、CPU、服务器等方面，利用它能够提高资源利用率、降低能源消耗。

2.1.4 典型的云计算平台

随着云计算技术的成熟，各个 IT 厂商推出了自己的云计算平台。例如，Google 推出了互联网应用服务引擎（Google App Engine，GAE），IBM 推出了蓝云计算平台，Amazon 推出了弹性计算云（Elastic Compute Cloud，EC2），Microsoft 推出了 Windows Azure 云计算平台。Hadoop[9]是 Apache 的一个开源分布式架构系统，适用于结构化和非结构化的数据搜索、分析和挖掘等任务，是一个高效而可靠的云计算平台。Hadoop 部署在基于大规模服务器集群的云计算数据中心上，提供可信赖的计算能力和存储能力。Hadoop 用户只需根据需求来开发应用程序，不需要关注分布式系统底层的细节。

（1）Hadoop 集群架构

Hadoop 由两个核心部分组成：分布式文件系统 HDFS 和分布式计算引擎 MapReduce[10]。HDFS 用于管理 Hadoop 集群中所有存储节点中的数据文件。MapReduce 则是一种高效的编程

模型，由管理节点（JobTracker）和任务节点（TaskTracker）组成。Hadoop 通过 JobTracker 把大计算的作业（Job）分割成多个称为 Map 和 Reduce 的任务，然后分配给多个 TaskTracker 来并行执行以实现集群系统的高效性，如图 2.2 所示：JobTracker 完成作业的调度和任务的分派，在 TaskTracker 上执行具体的任务；任务在执行过程中受到 TaskTracker 的监控，并由它们周期性地向 JobTracker 汇报相关信息[6]。

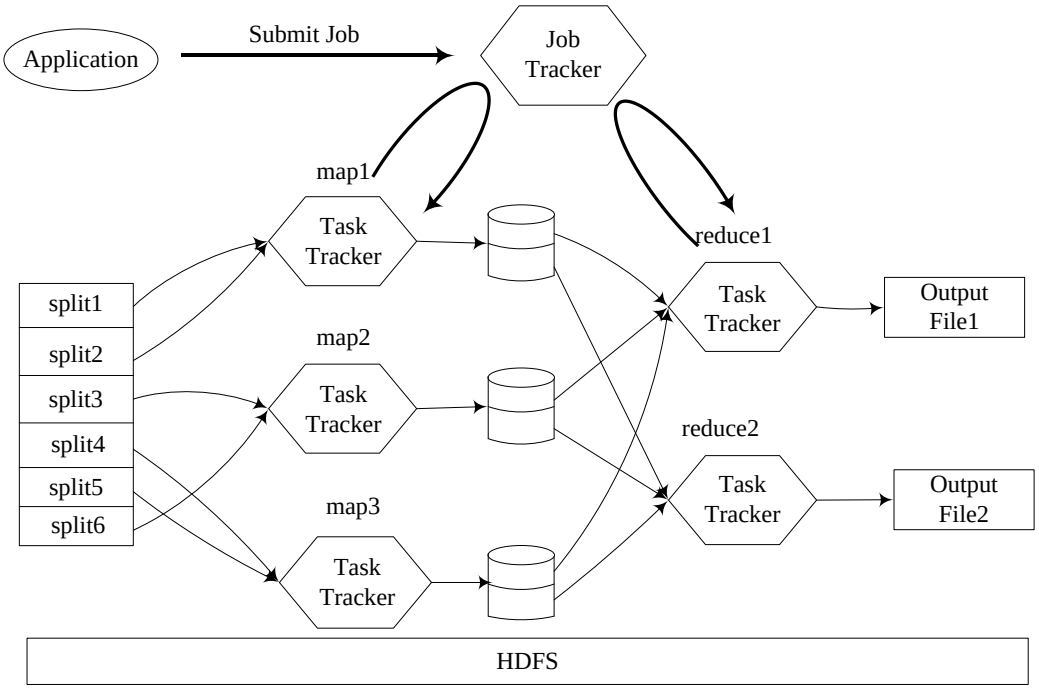


图2.2 JobTracker和TaskTracker任务调度过程

（2）Hadoop 任务调度模式

在 Hadoop 调度算法中，TaskTracker 端任务的获取流程如图 2.3 所示。

（a）TaskTracker 首先统计本地当前正在执行任务数（RunningTasks）。

（b）TaskTracker 判断正在执行任务数是否小于节点启动前预先设定的任务槽数（A fixed number of slots for map tasks and reduce tasks，标记为 FixedTasksCapacity）。

（c）如果 RunningTasks 小于 FixedTasksCapacity，则表示还可以接受新的任务，置可申请获取新任务的标志位 AskForNewTask 标志为 True，否则为 False。

TaskTracker 周期性地向 JobTracker 发送心跳信息（HeartBeat），汇报节点的状态信息及是否要索取新任务。当 JobTracker 接收到 TaskTracker 的心跳信息后，主要工作包括：

（a）JobTracker 首先检查上次心跳响应是否完成，如果完成，检查 TaskTracker 通过心跳发送过来的 AskForNewTask 是否为 True。

（b）若 AskForNewTask 为 True，则 JobTracker 会使用任务调度器 TaskScheduler 来组装任务到一个任务列表中，该调度过程在 TaskScheduler 的 assignTasks()方法中实现。具体的方

法是：分配的任务数=固定任务槽数-正在执行的任务数，即一次填满空闲的任务槽，我们将这种方式成为“一次心跳，全额分配”；若 AskForNewTask 为 False，JobTracker 将停止向该 TaskTracker 继续派发任务。

（c）JobTracker 得到任务列表数据后，将任务封装起来后，发给 TaskTracker 执行。

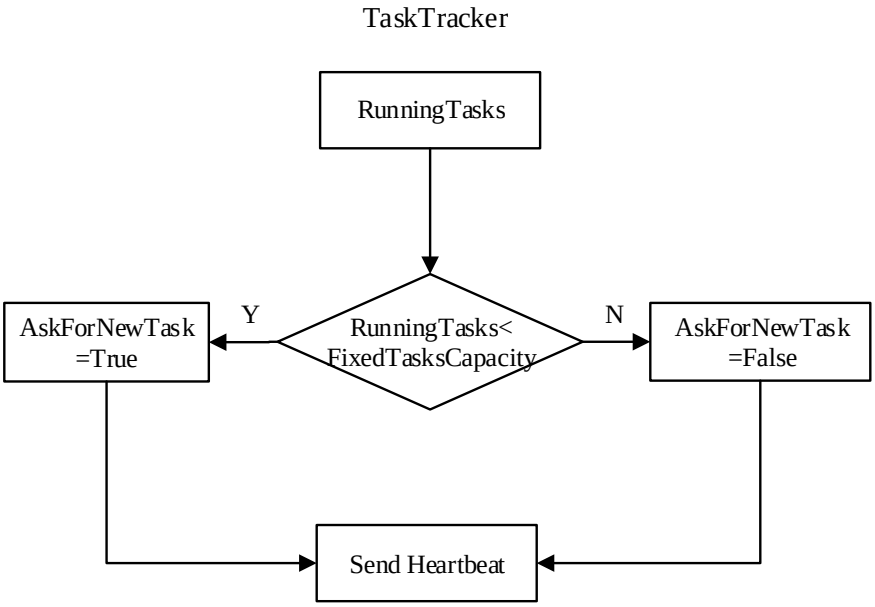


图2.3 Hadoop的任务获取流程

（3）Hadoop 任务调度相关研究

当 Hadoop 系统同时运行多个任务时，保证系统在整个集群内高效地进行任务的调度与执行，是实现集群计算资源充分利用的前提。因此，面向 Hadoop 的任务调度策略不断被提出。文献[11]提出了一个面向资源感知的调度器（Towards a Resource Aware Scheduler，TRAS），任务节点采集自身的资源信息，然后将该信息汇报给管理节点，由管理节点实现资源的调度。TRAS 策略存在以下几个问题：（1）实时性差，不能及时地根据任务节点的资源及负载状况进行自适应调节；（2）大量任务节点不断地向管理节点汇报资源及负载信息，使得管理节点承载巨大的系统开销，容易成为系统瓶颈。文献[12]提出了推测执行策略：当用户提交的作业被划分为多个任务并行执行时，如果大部分的任务已经完成，少量由硬件性能较差的节点执行的任务可能会剩下，成为拖尾任务。为了减少这些任务拖累整个作业的完成，JobTracker 会再启动一个拖尾任务的副本，和该任务一起并行执行，这两个任务中只要有其中一个完成，这个任务就完成了。推测执行策略是针对出现少量拖尾任务所采取的改进措施。如果当集群系统中出现大量拖尾任务时，该策略会产生批量拖尾任务的副本，可能导致集群系统性能急剧恶化。文献[13]提出了一个基于 Deadline 约束的任务调度算法：允许用户指定一个作业的 Deadline，并试图在这个期限前完成工作。任务按照将异构集群中的节点按照计算能力分类，

节点按照计算能力来获取任务。该算法仅仅从任务槽需求的角度来决定任务该调度到哪个节点，但忽视了节点的实时负载状况。另外，文献[14]介绍并分析了一系列现有任务调度策略，但是这些调度策略都没有对任务节点负载状况进行合理的评估，忽视了其在任务调度中的重要地位。

2.1.5 云计算仿真平台

CloudSim 由澳大利亚墨尔本大学的网格实验室和 Gridbus 项目共同推出的开源云计算仿真平台[15]。它是基于 JAVA 语言开发的，可实现跨平台运行。CloudSim 是在 GridSim、SimGrid、OptorSim 和 GangSim 的基础上开发和改进的。CloudSim 平台有助于加快面向云计算的算法设计与测试的速度，降低了开发的成本。用户可以通过 CloudSim 提供的众多核心类来进行大规模的云计算基础设施的建模与仿真，包括云数据中心的创建、云任务的提交，服务代理人的模拟、调度策略的仿真设计等。CloudSim 核心类主要包括以下几种：

(a)Cloudlet 类：云任务，用于构建云端用户提交的任务；

(b)DataCenter 类：数据中心，提供虚拟化的资源服务；

(c)DataCenterBroker 类：服务代理人，用于隐藏虚拟机的管理等；

(d)Host 类：主机资源，一台主机可以对应多个已创建的虚拟机，该类扩展了主机对虚拟机的众多参数分配策略，如带宽、存储空间、内存等；

(e)VirtualMachine 类：虚拟机类，运行在 Host 上，与其它虚拟机共享资源，每台虚拟机由一个拥有者所有，可提交任务，并由 VMScheduler 类制定该虚拟机的调度策略；

(f)VMScheduler 类：虚拟机的调度策略，制定相应类型的虚拟机调度策略；

(g)VMAllocationPolicy 类：虚拟机资源配置策略类，定义创建的虚拟机在主机资源上的调度策略。

（1）CloudSim 共享策略

CloudSim 提供的虚拟化引擎可以帮助用户建立和管理数据中心节点。因为（1）虚拟机对主机资源存在竞争；（2）各个任务单元对虚拟机资源同样存在竞争。所以，CloudSim 对创建的虚拟机和提交的任务单元提供了灵活的时间和空间共享策略[16]。用户可以根据不同的共享策略组合来开发相应的调度算法，从而实现调度算法的模拟。

（2）CloudSim 仿真步骤

仿真过程主要包括以下步骤：

(a)初始化 CloudSim 的工具包；

(b)创建数据中心 DataCenter；

(c)创建代理 DataCenterBroker；

(d)创建虚拟机列表 VmList，然后将虚拟机列表提交到数据中心代理；

(e)创建云任务列表 CloudletList，然后将云任务列表提交给数据中心代理；

(f)开始模拟 StartSimulation；

(g)结束模拟 StopSimulation，输出结果。

（3）CloudSim 资源调度相关研究

CloudSim 的资源调度是指通过选择满足条件（内存、CPU（MIPS），带宽）的 host 创建执行任务所需求的虚拟机的过程，这个过程由创建的数据中心（Datacenter）负责。抽象类 VmAllocationPolicy 代表资源调度的过程，可以通过继承该类实现自己的调度策略[15]。在 CloudSim 中，实现了一种简单的调度策略：VmAllocationPolicySimple，它实现了从数据中创建的主机列表中选择一台合适的主机，并在其上创建需求的虚拟机。主要实现过程的描述如下：

(a) 从所有主机中选出可用 CPU 核数最多的一台主机，并在其上创建虚拟机。

(b) 如果(a)失败且还有主机没有被试过，就排除当前选定的这台主机，重做(a)。

(c) 如果最后该虚拟机创建成功，返回 TRUE，否则返回 FALSE。

2.2 绿色计算

绿色计算顺应低碳社会建设的需求，是推动社会可持续发展和科技进步的一个重要方面。关于绿色计算的内涵和定义，已经有很多学者提出了自己的看法[17-19]。本文的理解是，绿色计算是一种环境友好型的计算模式，结合包括云计算在内的其他计算模式，通过构建低能耗的计算环境，合理的资源配置环境和高效的任务执行环境，来保证计算机系统的高效性和可靠性，以达到节能、环保的目的。作为一种新型的计算模式，绿色计算受到了工业界与学术界的广泛关注并取得了一定的成果。本节对目前绿色计算国内外的研究成果进行了介绍。

2.2.1 能耗模型及评价

能耗及其评价问题是实现绿色计算的核心问题之一，通过研究能效及其影响因素间的关系，建立能耗模型与评测机制，是实现绿色计算的前提。能耗问题涉及到计算系统的各个方面，包括硬件芯片、存储部件、体系结构、编译器、操作系统、通信网络、应用软件等，这些方面依赖关系复杂，研究难度大[18]。林闯等人将能量作为一种系统资源，将能耗管理和控

制问题归结为资源管理和配置问题，在对绿色网络的机制和策略详尽调研分析的基础上，提出基于随机模型的绿色评价框架，为绿色网络的评价体系创造了前提[20]。文献[21]通过对操作系统的能耗实验数据进行分析，得到服务例程级的宏模型，即能耗与通信量、软件的算法复杂度和路径基本块关联信息等高层度量特征之间的关系。文献[22]提出了一种对科学计算应用能耗和功耗特性建模的方法，该方法应用于分布式计算系统。文献[23]提出了一种基于组件的并行科学计算应用功耗和性能模型。

2.2.2 低功耗硬件

处理器的多核技术是提高处理器计算能力的一种重要手段。然而，多核技术使得处理器的功率不断增加，功耗问题更为突出。如何实现高能效比是这些处理器设计的一个重要指标。采用最新的架构，最新的工艺和最新的节能技术，可以在很大程度上实现高性能、低能耗。各大硬件制造商，尤其是 CPU 芯片制造商（如 Intel、AMD）不断改进工艺，降低 CPU 能耗。文献[24]深入的研究了处理器能耗优化的问题，提出了基于处理速度的能量消耗函数，采用速度缩放策略，将能耗优化问题抽象为最优任务调度问题。

固态硬盘取代机械硬盘可以很大程度上降低硬件的功耗[25]。机械硬盘从待机状态切换到工作状态，需要进行电机加速。移动磁头臂需要的瞬时电流达到硬盘正常工作电流的两倍以上。而固态硬盘的启动电流几乎和工作电流一样，因此电源无需进行额外的功率设计。再者，固态硬盘只需极短时间就能从待机状态切换到工作状态，所以不断将固态硬盘切换到待机状态，不会增加额外的电力消耗，反而能减少电能。

除了处理器和硬盘外，还有很多的因素决定了服务器硬件层面上的功耗。文献[26]在设计数据存储系统时采用超低功耗存储节点，硬件电路板的耗电量仅为 5 瓦，从硬件层面上降低了系统功耗，同时为系统的高度集成打下了基础。Google 对服务器主板进行修改，并利用蓄电池来提高能源的利用率。因此，对于不同类型的服务器应该结合能耗和性能加以平衡，从而实现真正意义上的高效能绿色计算。

2.2.3 体系结构

为了降低系统的能耗，工业界和学术界都在不断地研究和改进计算机的体系结构。例如，以通用处理器为主，专用处理器为辅的异构计算机，通过两种处理器的协作可以得到良好的能效比。文献[27]提出了一种新颖的集群体系结构，用于处理大规模数据密集型任务，该结构具有性能优异和系统功耗低的优点。Gordon 是针对高度并发的数据型任务提出的一个基于

Flash 存储器的的集群系统，该系统处理速度快，但功耗较低[28]。然而，目前计算机体系结构的低功耗研究主要集中于通用型体系结构，而针对我国自主研发的处理器的低功耗研究很少，因此迫切需要开发拥有我国自主知识产权处理器的高性能、低功耗计算机系统。

2.2.4 关闭/休眠技术

关闭/休眠技术也是常用的实现分布式系统节能的技术，该技术通过关闭或休眠空闲节点的方式来降低空闲能耗。文献[29]采用休眠负载较轻的节点，减少系统的能耗。这种方法把研究的焦点创新性地从动态节点转移到空闲节点，即通过休眠空闲的节点来减少能耗。该策略假定休眠后节点的能耗为 0，且不考虑休眠节点存储的副本，但是在实际应用中必须考虑这两个问题。关闭/休眠技术的缺点是当需要的节点不满足需求时，重启节点需要很长时间，这会导致系统的响应时间变长，影响用户体验。该技术一般提前设定或预测需要关闭/休眠主机或关键部件的时机。所以，对于拥有大量计算资源的云系统而言，关闭/休眠技术需要解决的难题是在已知单位时间任务的到达量的前提下，确定需要关闭/休眠多少主机，以及关闭哪些主机等问题。

2.2.5 动态电压调整技术

根据系统实时负载的大小调节系统部件功耗的大小，在降低能耗的同时保证性能，典型的代表是动态电压调节（Dynamic Voltage Scaling，DVS）技术。文献[30]提出一种基于能耗感知的启发式任务调度算法，动态调整同一任务执行时的电压；算法采用能耗梯度作为任务调度的评价指标，同时又考虑任务调度的顺序。文献[31]提出了一种基于时间片的 DVS 能耗优化算法，预测任务的执行时间，针对预测不准确的问题：预测偏长时降低电压来降低能耗；预测偏短时将未完成部分作为一个新的任务重新调度。文献[32]根据当前包任务的执行情况进行预测，保证在用户定义的 Deadline 前完成任务，动态调整处理器的运行电压，降低系统的能耗。文献[33]提出了一种基于 DVS 技术的启发式调度算法，用来降低任务的执行能耗。但是 DVS 应用于云计算系统时，会遇到以下问题：(a) 任务到达系统的时间是不确定的，所以到达任务的类型很难预测；(b) 就算能够预测任务的类型，适合该任务的处理器电压也很难确定；(c) DVS 主要用来降低主机处理器的能耗，但用来优化整个主机或整个云计算系统的能耗就比较局限。

2.2.6 网络通信

网络通信的节能研究一般通过减少网络中的无用能耗，提高资源的利用率来达到节能的目的。文献[34]针对当前网络能效算法研究的局部性，从全局角度出发研究网络的能耗问题，提出了优化的节能路由算法。但是该算法不能直接用于基于分布式路由器的网络。网络环境的节能研究一般通过优化网络的拓扑结构来节约能耗，其中绿色代理技术与休眠机制的结合的研究最为广泛。Nordman 引入绿色代理技术，通过代理端在需要节点工作时唤醒节点，减少了能耗，但并不降低网络的性能[35]。

2.3 面向云计算系统的节能机制

如何减少云数据中心的能源消耗，实现高效能绿色云计算成为影响可持续发展和低碳节能的一个重要方面。目前针对云计算系统的能耗问题及节能减排措施也已经有了一系列的研究成果。文献[36]提出了基于动态定价策略的数据中心能耗成本优化方案，首先对服务价格和能耗成本建立统一的模型，同时优化服务价格与能耗成本之间的关系，使数据中心的收益最优；数据中心采用基于重载近似的大规模排队模型来建模，根据不同数据中心的服务请求量和电价的差异，设计一种多数据中心间的负载路由机制，来减少数据中心的整体能耗成本；同时针对单个数据中心，定义双阈值策略，动态调整节点的状态，进一步优化数据中心能耗成本。文献[37]定义了能效的测量方法及其计算公式，并推导出产生最大能效的条件；改进了 CPU 工作状态与计算机功率之间关系的数学表达式，以方便能效的计算；为了简化能效的测量，通过 CPU 使用率和频率来计算能效；同时分别针对云环境中的 CPU 密集型、I/O 密集型及交互型运算的能效进行了评估。

本节余下部分针对目前云计算的节能优化措施进行了详细的研究与分析，具体内容如下。

2.3.1 虚拟化技术

虚拟化技术在一台主机上虚拟出多个虚拟机，将若干个任务分配到这些虚拟机上运行，通过提高主机资源的利用率，来减少所需主机的数量，从而降低能源消耗。文献[38]提出了一个能源模块化管理模型，包括两个组件：主机级子系统和虚拟机级子系统。主机级子系统负责调控整个系统的能耗，根据应用请求合理地分配所有的硬件资源，每个虚拟机的能耗不能超过相应阈值，使得系统有能力针对特定应用进行细粒度的能源管理。虚拟机级子系统重新分配虚拟机的硬件资源，确保每个任务消耗的能源不超过相应阀值。然而该模型只考虑磁

盘活动与空闲两种模式，节能效果不佳。文献[39]融合了虚拟化技术与功耗管理机制、策略，提出了一种在线功耗管理方法，部署于大规模数据中心。该方法可以在每一个虚拟机上独立，同时能够从全局来控制和协调同一个虚拟化平台上的不同虚拟机之间、不同虚拟化平台之间的功耗管理策略，实现功耗的全局优化管理。文献[40]提出了一项关于基于负载感知的虚拟机在线调度器的研究成果，探索了多虚拟机共存于同一节点导致的性能干扰以及对 CPU 温度变化的影响，还提供了一种云环境下的分布式能耗管理机制。文献[41]提出了一个由动态虚拟机部署管理器和基于效用的动态虚拟机供给管理器组成的资源管理框架，通过建模约束模型，实现全局效用最大化，同时满足 SLA，使得云计算基础设施能量相关的操作开销最小化。

虚拟化技术是实现云计算节能的一种方式，通过将物理资源抽象为虚拟资源的方式，提高了资源的利用率。但是，虚拟化本身要付出较高的能效代价，且虚拟化的层次越深代价越高。因为虚拟化技术是一层一层进行虚拟化的（从最低层的硬件到最高层的应用），每一层的虚拟都要付出能效的代价；仅采用现有的虚拟化技术，在云计算系统性能和能效方面的优化效果是有限的；现有的虚拟机管理器不能与其上层支撑的多操作系统相互传递能耗特征，也不能感知上层应用的负载和运行状况，导致在实现任务调度时能效比不能令人满意。

2.3.2 资源配置

采取合理的资源配置策略，调整资源的需求，不仅能够保证云计算系统具有足够的计算/存储资源来完成用户提交的各种不同类型的任务，还能提高系统资源的利用率，降低能耗。一种有效的方法是根据动态变化的负载请求、资源利用状况等对集群系统中的资源进行动态配置，在保证用户服务质量的情况下提高集群的资源利用率，实现绿色计算。文献[42]提出的算法仅适用于小规模集群，当集群达到一定规模，算法导致系统时间开销巨大，不能满足实时性任务的需求。文献[43]主要将虚拟数据中心环境下虚拟机的放置问题建模成多目标优化问题，采用分组遗传算法和模糊多目标评价的方法来达到高效使用多维资源等设计目标，但没有考虑工作负载的类型，同时算法复杂度较高。文献[44]采用关闭/休眠技术，尽量将虚拟机分配给一些激活的主机，将剩余的主机置于节电状态，以减少数据中心的能量消耗。针对应用场景，建立数学模型，将虚拟机放置问题抽象为一个布尔整数线性规划问题，提出一种启发式算法来解决这个问题。文献[45]提出了一种云计算环境下基于最小相关系数的节能虚拟机放置方法，该方法使用模糊层次分析法尝试在服务水平协议和低能耗之间进行权衡。文献[46]通过设计虚拟机的调整机制来实现绿色计算，达到节能目的。[44-46]都只局限于具体的应用场景，并没有考虑到负载的变化；文献[47]直接利用二次指数平滑算法实现负载的

预测，忽略了较大预测误差的存在，并同样采用分组遗传算法实现资源的配置，算法复杂度高。文献[48]提出了一种面向能耗最小化的虚拟机部署策略，首先根据服务器的能耗模型，以贪婪方式计算数据节点的目标利用率，反复选择单位容量增长能耗最少的数据节点作为虚拟机部署地点，直至完成所有虚拟机的部署地点的确定；然后基于最先适配递减算法来完成虚拟机的部署任务，通过将虚拟机部署到预定的节点集合上，同时满足每个数据节点的目标利用率，使得所有数据节点的能耗最小化。文献[49]设计了一种启发式算法来实现虚拟机的迁移，针对虚拟机的部署设置数据节点 CPU 利用率的单阈值上限，用 CPU 资源预留的方法，保证 SLA 的实施，并采用最优适配递减算法来重新分配虚拟机资源。文献[50]针对异构云计算环境，采用调度优先级约束的平行双目标混合遗传算法来解决资源分配问题，以达到完工时间最小和能耗最低的目标。文献[51]提出一种云环境下的节能调度算法，采用神经网络技术来预测负载，将多种资源融合以实现资源最小化分配，把资源配置问题抽象为多维装箱问题，即最小化能耗问题转化为求解最小化箱子数量问题。文献[52]指出在一个共享的虚拟计算环境中，负载的动态变化以及应用程序的不同质量需求，使得他们产生动态的资源需求，现有的静态资源配置导致低资源利用率及低用户满意度，因此提出一个两层的资源按需分配机制，即局部和全局资源分配与反馈机制。该机制在资源竞争时根据实时计算力需求，优先保证关键应用程序的性能，从而提高资源利用率，降低系统能耗。在云计算中，低利用率的资源仍然消耗大量能耗，因此文献[53]提出了两个资源感知的启发式任务整合算法，旨在最大化资源利用率，同时考虑激活能耗和空闲能耗。

综上所述，大多数资源配置算法都没有考虑到如下四点：

（1）云数据中心的规模，当数据中心的规模达到一定程度时，算法的执行时间会成指数级增长，不能满足用户对响应时间的要求，因此不适用于大规模云数据中心；

（2）任务的多样性，算法仅针对一种类型的任务（如 WEB 型任务），而未考虑云数据中心可能接受任务的多样性。适用于特定类型任务的算法，可能不适用于其他类型的任务，因此不能处理多种类型任务的情形，这显然是不合理的；

（3）调整的时机，有些研究工作只根据当前时刻的任务到达量来进行调整，没有考虑调整算法及节点调整的时间开销，导致调节落后于负载请求的变化；

（4）预测算法的准确性，不论采用何种预测算法，预测值都有可能存在或多或少的误差波动，导致出现预测值小于实时负载值的情况，系统实时性要求得不到保障，除此之外，预测值的频繁波动，导致系统稳定性极差，不能满足实际需求。

2.3.3 任务调度

合理的任务调度算法可以提高任务的执行效率和系统资源的利用率，减少整个集群的总能耗。文献[54]针对云系统在运行期间由于节点空转造成的能源浪费，及由于任务调度的不匹配而产生的能源浪费问题，提出一种基于任务调度的能耗优化管理方法，分别针对上述两个能耗提出优化策略。在此基础上，进一步提出满足性能约束的最小期望执行能耗调度算法。由于算法是基于任务达到随机性建立任务模型，没有考虑真实的云计算系统任务达到的特性。文献[55]提出的任务调度优化模型，通过将任务调度到最低数量的服务器上来降低云数据中心的能耗，同时又保证任务的响应时间。采用贪婪算法计算需要激活服务器的数量，遵循优先将任务调度到最高效的服务器上的原则。同样的，仿真实验时任务到达用指数分布来描述，未考虑实际的云数据中心任务到达场景。文献[56]针对采用复制并行调度可以减少后续任务的等待时间并减少任务的总执行时间，但会产生附加能耗的问题，提出了一种并行任务节能调度优化方法。该方法通过将负载较轻的处理器上的任务调度到其它处理器上执行的方式，减少系统所用处理器的总数，来降低系统的能耗。该方法在理论上可以达到预期的效果，但是在实际应用中很难确定每个任务的结束时间。文献[57]提出的面向大规模云数据中心的低能耗任务调度算法引入胜者树作为模型将任务节点作为该树的叶结点，以低能耗为目的两两比较并选出最终的节能者。同时将不同的任务赋予不同的优先级，处理多个任务同时调度的问题。该算法假设每个到达任务对资源的需求是已知的，但是实际中这是很难确定的。文献[58]提出了基于能耗感知的启发式任务整合算法，通过检测每个资源，把能效最高的资源分配给任务。文献[59]提出一种基于任务复制策略的任务调度算法，减少整个任务集的执行时间，同时消除部分任务的通信能耗；针对算法导致的任务执行能耗培加问题，进一步引入遗传算法和蚁群算法，提高算法的进化速度，减少算法的时间开销，通过牺牲较小的调度时间来降低任务执行的能耗。文献[60]设计了一种启发式任务调度算法，采用动态电压调整技术来降低并行任务的执行能耗，搜索非关键任务的空闲时间，在保证不增加作业整体执行时间的情况下降低能耗。文献[61]致力于在保证最大完工时间的同时降低能耗，算法识别出效率低下的处理器，关闭它们以减少能耗。然后，任务被重新调度以使用更少的处理器，进一步降低能耗。然而，算法只限定于特定类型的任务，适用性不是很高。

2.3.4 数据存储与部署

目前，国内外的不少研究人员已经开展了云计算中的节能型数据存储和部署的研究工作。

文献[62]对绿色节能数据存储展开了研究，提出了一种基于布局的虚拟磁盘节能调度方法，在将用户请求调度到目标磁盘时考虑了响应时间及动态优化等因素，将磁盘阵列分为工作区与就绪区，用户请求调度到工作区，并根据实时负载动态优化虚拟磁盘布局。文献[63]在对磁盘的节能管理进行了详细的定量分析后，指出应该在磁盘休眠两秒后将其切换为休眠模式，以使能耗最小化。文献[64]重构虚拟机的读写操作（特别是写操作），以及提前释放虚拟化层上的休眠前缓冲区，来使磁盘的休眠时间最大化，从而节约能源。然而（1）该方法是基于虚拟机与磁盘的，属于系统盘的调度范畴，所以与独立于虚拟机的数据盘的调度不同；（2）物理内存不支持低功耗模式，因此节能的空间不大；（3）虚拟机自带的磁盘较小，且与虚拟机是紧耦合的，因此用户一般不会选择系统盘来存数据，而是选择一块独立的虚拟磁盘。文献[65]基于多数据中心的云计算环境，提出一种三阶段数据布局的策略，对跨数据中心数据传输、全局负载均衡和数据依赖关系这三个目标对数据布局方案进行设计和优化。

2.3.5 网络结构

文献[66]构建了一种面向云数据中心底层网络的体系结构，旨在提高数据中心网络的吞吐量以及网络带宽分配的灵活性。文献[67]针对目前设计数据中心网络拓扑时未考虑云计算自身特点的问题，提出了一种基于BCube 和Fat-tree 结构的网络拓扑结构，该结构具有直径小、连通性高、可拓展性强等优点。文献[68]指出可以通过提高服务器与交换机数量比的方式降低能耗，在深入研究新型数据中心的网络结构后，提出了一种新型雪花结构，提高数据中心的扩展性、减少网络开销。文献[69]针对不同节点上的虚拟机之间的通信需求，考虑数据中心网络带宽、延迟等因素，提出一种基于数据传输时间最小化的虚拟机部署策略，实现网络层面上的系统优化。

2.3.6 温控设施

云系统的配套温控设施耗电成本极大。为了节能和降低开销，Google云数据中心采用自然冷却策略，将数据中心建设于气候寒冷的地域，来降低温控设备的能耗；同时采取多项节能措施（如蒸发冷却塔），并在环境温度超标时关闭数据中心，将任务请求迁到其他低气温地域的数据中心中，但这种机制不适用于基于单一数据中心的云计算系统。Microsoft的“集装箱”式数据中心在一个集装箱中放置2200台机器，这种方式增强了部署的灵活性和机动性，但随之产生了狭小空间的散热问题。Microsoft采取了一系列节能措施来应对这个问题，节约了25%的能耗。文献[70]提出一种基于温度感知的数据中心任务放置策略，包括基于区域的离

散化策略和热量再流通最小化策略，从降低温控设备能耗的角度来节约温控开销，同时提高硬件的稳定性。文献[71]提出了基于功耗感知的高性能计算程序动态部署机制。

2.4 本章小结

本章概述了与研究课题密切相关的技术及概念。首先概述了云计算技术的相关概念、体系架构、关键技术、典型的云平台和仿真技术。在此基础上，从能耗模型及评价、低功耗硬件、体系结构、关闭/休眠技术、动态电压调整技术、网络通信等方面对绿色计算的研究成果进行了详尽分析和总结。最后从虚拟化技术、资源配置、任务调度、数据存储与部署、网络结构和温控设施等角度出发，对云计算中的节能技术进行了详细阐述和分析。

第三章 绿色云计算系统模型

本章针对目前现有的云计算系统存在能耗大，效率低下等问题，建立一种高效能、低能耗的绿色云计算系统模型。该模型通过资源配置和任务调度子模块实现系统内部计算与存储设备的配置和调度；通过能耗监测和温控子模块实现外部监控系统的控制和调节。重点从绿色云计算系统内部考虑，设计一种具有低能耗、高效率的资源配置和任务调度子模块。其中，资源配置子模块保证云计算系统具有足够的计算/存储资源来完成用户提交的各种不同类型的任务；任务调度子模块实现云计算系统对不同类型任务的实时调度。

3.1 系统模型

为了达到绿色云计算节能、高效的目标，设计一种合理的云计算系统模型是必须的。本章从实现绿色云计算的内部和外部因素出发，设计了如图 3.1 所示的绿色云计算系统模型。

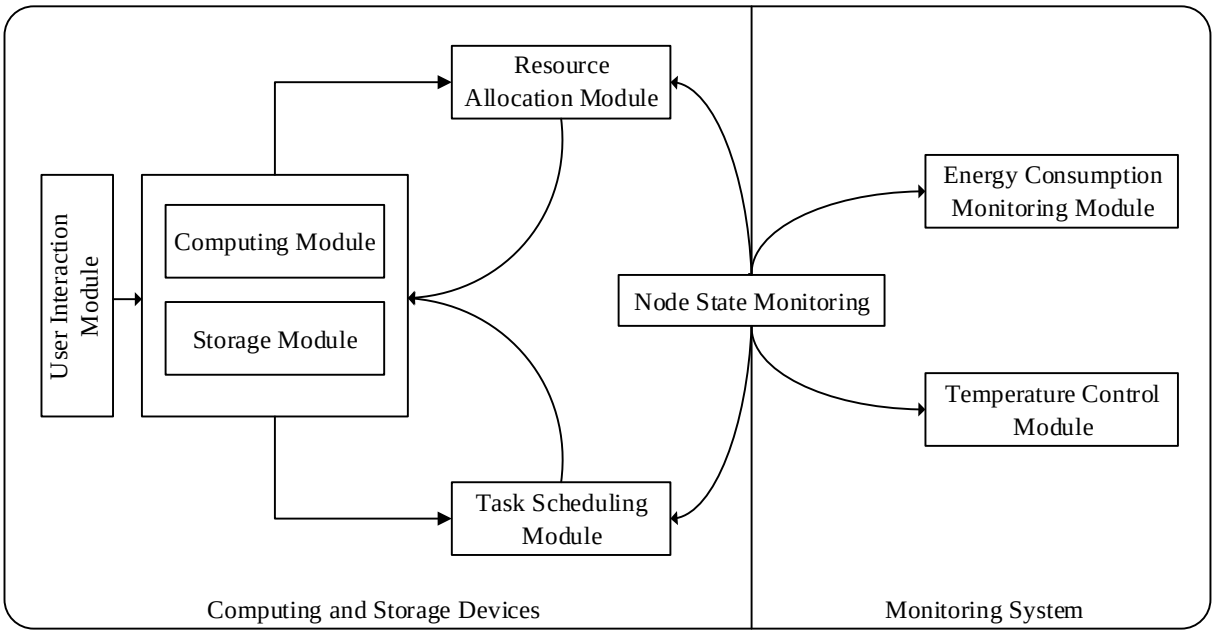


图3.1 绿色云计算系统模型

为了更加具体地描述该模型，我们先从功能上将云计算系统的集群节点进行两类划分：管理节点（Master Node，MN）和任务节点（Task Node，TN）。当作业到达云系统后，管理节点按照一定规则将其切分成若干个任务，并部署到合适的任务节点上。任务在任务节点上运行，得到结果，返回给管理节点。

（1）用户交互模块（User Interaction Module，UIM）：用户和云端资源交互的接口，该模块帮助用户简化任务的提交过程，帮助用户实时高效的获取任务的响应。

（2）计算模块（Computing Module，CM）与存储模块（Storage Module，SM）：这两个模块由管理节点和任务节点组成，负责具体任务的执行和存储工作，实现用户与云端计算/存储资源的交互。

（3）资源配置模块（Resource Allocation Module，RAM）：对于整体的云计算系统而言，对某种类型的任务节点需求量表现为对某种计算与存储资源的需求量，而资源配置模块负责资源的总体配置，统筹全局的资源分配。为了实现高效可靠的资源配置，资源配置模块（RAM）需要从全局出发，设计一种合理的资源配置模型，其中包括预测子模块，资源调度器子模块等，采取合理的资源配置策略，调整资源的需求，保证云计算系统具有足够的计算/存储资源来完成用户提交的各种不同类型的任务。

（4）任务调度模块（Task Scheduling Module，TSM）：负责各个不同类型的任务调度，从局部（即任务节点）的角度提高不同类型任务的执行效率，降低管理节点的负担。为了实现高效可靠的任务管理，任务调度模块（TSM）需要设计一种合理的任务调度模型，配合高效的任务调度算法，提高系统资源的利用率，减少整个云计算系统的总能耗。

（5）节点状态监测（Node State Monitoring，NSM）：负责监测管理节点和任务节点的各种状态，包括任务的执行状态、节点的温度和资源利用率等各类信息，将这些信息汇总并分类汇报给相应的模块，帮助整个云计算系统实现绿色计算。

（6）能耗监测模块（Energy Consumption Monitoring Module，ECMM）：负责监测整个云计算系统的能耗。

（7）温控模块（Temperature Control Module，TCM）：负责监测外界环境温度，根据收集的实时系统温度进行合理性调节，降低系统能耗。

从上述的功能描述可知：资源配置和任务调度子模块是实现绿色云计算系统内部的核心，也是本文研究的重点，必须设计一种合理的资源配置模型和任务调度模型。

3.2 资源配置模型

本节结合资源配置模块（RAM）对于云计算系统的全局需求，提出了一个新型的绿色云计算资源配置模型。该模型的架构如图3.2所示。该模型分为三个模块，包括预处理模块（Preprocessing Module），调度模块（Dispatching Module）和执行模块（Executing Module）。

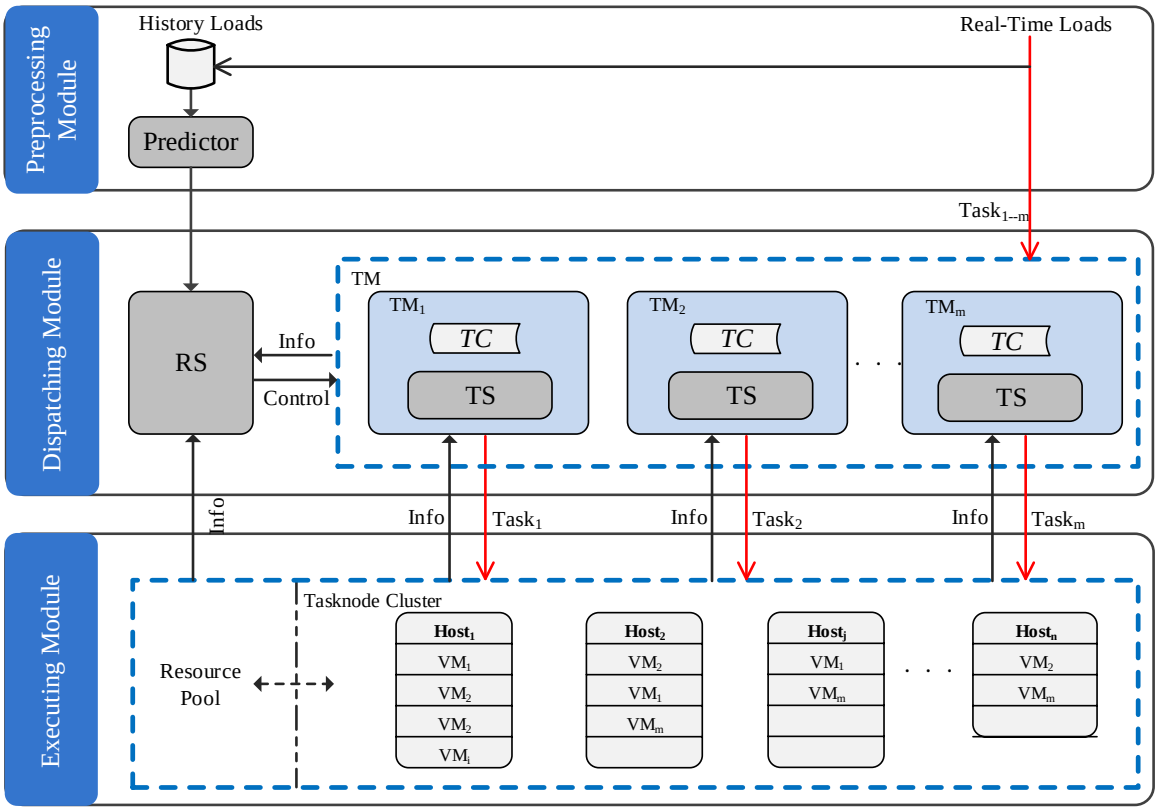


图3.2 资源配置模型

（1）预处理模块，负责作业的预处理工作，主要功能为周期性地预测系统的作业请求量，为资源调度提供数据，从而能够有效地避免资源配置滞后于用户请求的问题。系统首先读取存储在磁盘上的历史负载数据。根据读取的历史数据，结合当前系统的实时负载量，负载预测器（Predictor）采用合适的预测算法对云数据中心下一个周期内的作业请求量进行预测，并将预测结果汇报给系统的资源调度器（Resource Scheduler，RS）。

（2）调度模块，包括资源调度器（RS）和任务管理器（Task Manager，TM），其实际的工作由资源调度器（RS）和任务管理器（TM）完成。根据预测负载量的大小和当前任务集群（Tasknode Cluster）的负载状况，资源调度器（RS） 采取适当的控制策略，对下一个周期内的任务请求所需的资源（包括虚拟机和物理主机资源）进行预配置，调整执行模块中开启的主机节点（Host）和运行的相应类型虚拟机（VM）的数量。

（3）执行模块，实际负责任务的执行，物理主机被虚拟化为一个资源池（Resource Pool），任务在相应类型的虚拟机上运行。假定资源池（Resource Pool）能满足用户请求，即对于用户来说，资源是无限的，在满足用户计算需求的前提下提高任务集群（Tasknode Cluster）的资源利用率，减少整个集群系统的能耗。

具体流程描述为：当作业到达时，首先被预处理模块切分归类，定位为$Task_i$。将任务$Task_i$分发到相应类型的任务管理器（TM_i）中。由其中相应的任务调度器（Task Scheduler，TS）以尽可能最优的方式将到达的任务调度到对应类型的虚拟机上执行。同时，比较实时负载和

预测结果，将超出预测的负载存储在任务缓存（Task Cache，TC）中，反馈给资源调度器（RS），做出相应的资源配置优化，解决预测结果和实际负载相偏差带来的问题。

3.3 任务调度模型

本节根据系统模型任务调度模块（TSM）的需求，提出的一种高效可靠的任务调度模型，优化现有的任务调度机制。该模型如图 3.3 所示，主要由监控模块（Monitoring Module）、任务获取模块（Task Capturing Module）、任务执行模块（Task Running Module）和通信模块（Communication Module，CM）组成。

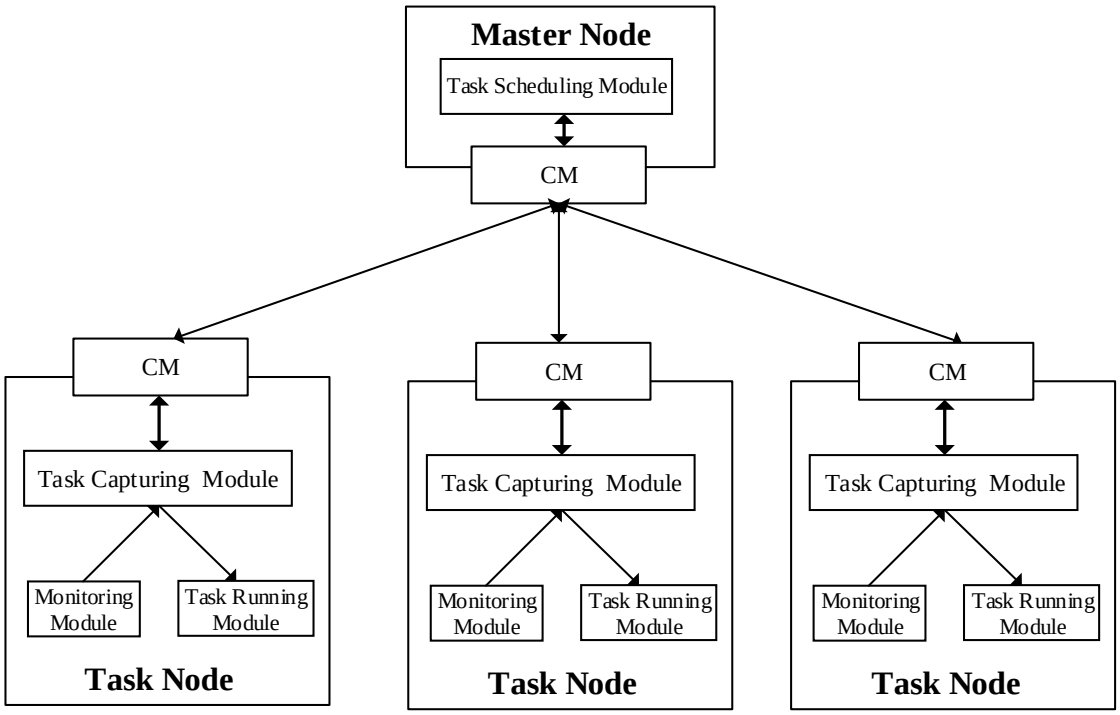


图3.3 任务调度模型

（1）监控模块，部署于任务节点端，负责检测节点的负载状况（资源使用情况），将信息汇报给任务获取模块（Task Capturing Module）。

（2）任务获取模块，部署于任务节点端，负责采用合适的任务获取机制。节点根据监控模块检测的结果判断自身状态，依此进行调节，改变任务获取的方式，适应自身的负载变化，实现集群节点的自调节。

（3）任务执行模块，实现任务的具体执行。

（4）通信模块，负责节点之间的通信。

节点监测和任务获取机制都是在 Task Node 端进行的，因此，能够在保证集群响应实时性的同时，避免 Master Node 采用复杂调度算法而产生巨大的系统开销。

3.4 本章小结

本章从实现绿色云计算的内部和外部因素出发，设计了一种高效而可靠的绿色云计算系统模型。重点着手于系统内部的资源配置模块和任务调度模块，针对这两个子模块的实际需求，给出了具体的实施方案。设计的资源配置模型使得系统资源配置更加合理，提高了系统资源的利用率，降低了云计算系统整体的能源消耗；任务调度模型能够保证云计算系统具有合理的任务调度机制。

第四章 面向低能耗云计算任务调度的资源预配置算法

针对云计算系统中存在大量能耗浪费的问题，本章采取“关闭冗余，开启需求”的资源预配置策略。首先，根据提出的资源配置模型，采用虚拟化技术，抽象任务调度问题为虚拟机部署问题。其次，预测用户请求的负载大小，结合当前系统状态和资源分布，采取保守控制策略，计算下一个周期内任务对系统资源的需求量。然后，建立满足多目标约束的能耗模型，提出基于概率匹配的资源配置算法，实现低能耗资源配置。在该算法的基础上，提出基于改进型模拟退火的资源配置算法，进一步降低系统能耗。预测算法和保守控制策略能够有效避免资源配置滞后于用户请求的问题，提高系统的响应比和稳定性；提出的资源配置算法能够激活更少主机，实现激活主机集合之间更好的负载均衡和资源的最大化利用，降低云计算系统能耗。

4.1 问题建模

不同类型的任务对计算资源的需求是不同的，所以将任务分类为数据密集型任务、计算密集型任务、通信密集型任务、和 I/O 密集型任务等不同类型[54]。假设有 M 种类型的任务，在某一时刻系统需要处理的每种类型任务的总数为 M_i。根据不同类型的任务对计算资源的需求，将不同类型的任务调度到匹配类型的虚拟机上，完成任务的执行。即第 $i(1\le i\le M)$种类型的任务 $task_i$ 调度到 VM_i 上执行：$<task_i>\rightarrow<VM_i>,\ i\in(1,M)$。

假设相应类型的虚拟机在某一个时刻同时只能运行一个任务，当前正在执行的任务完成后可继续执行相应类型的任务。对于用户提交的各种类型的任务量，需要多个相应类型的虚拟机来执行。为了分析问题的方便：现将任务的调度问题抽象为“以尽可能最优的方式，将能够满足用户需求的多个虚拟机调度到集群规模为 N 的主机（Host）上，实现最高能效比”，即：$<VM_i>\rightarrow<host_j>,\ i\in(1,M),\ j\in(1,N)$。

4.1.1 问题分析

将任务调度到相应类型的虚拟机上，需要考虑到虚拟机的分布和当前状态。在第三章提出的资源配置模型中，首先通过合适的预测算法和控制策略获得各类任务的预测值及控制值，结合当前相应类型的虚拟机分布及状态，做出资源的合理预配置。为了简化问题分析，做如

下状态行为定义：

定义 4.1：对于所有物理主机集合：$<host_j>=(host_1,host_2,...,host_j,...,host_N)$和虚拟机类型集合：$<VM_i>=(VM_1,VM_2,...,VM_i,...,VM_M)$，定义 M×N 状态统计矩阵 $A_{M\times N}$。其中 $A_{M\times N}$ 矩阵中的元素 a_{ij} 定义为：VM_i 类型的虚拟机在 $host_j$ 上启动的总个数。VM 状态统计矩阵表示如下：

$$A_{M\times N}=\begin{pmatrix} a_{11} & \cdots & a_{1N} \\ \vdots & a_{ij} & \vdots \\ a_{M1} & \cdots & a_{MN} \end{pmatrix}，其中：\ i\in\{1,...,M\},\ j\in\{1,...,N\} \tag{4.1}$$

定义 4.2：对于所有的物理主机集合：$<host_j>$和任务类型集合：$<task_i>$，定义 M×N 的平均速率矩阵$U_{M\times N}$。矩阵中的元素 u_{ij} 定义为：$task_i$ 类型的任务在 $host_j$ 上执行的平均速率（平均执行时间为$1/u_{ij}$）。平均速率矩阵表示如下：

$$U_{M\times N}=\begin{pmatrix} u_{11} & \cdots & u_{1N} \\ \vdots & u_{ij} & \vdots \\ u_{M1} & \cdots & u_{MN} \end{pmatrix}，其中：\ i\in\{1,...,M\},\ j\in\{1,...,N\} \tag{4.2}$$

定义 4.3：集群中的物理主机的总数为 N。对于启动的 n（$n\le N$）个物理主机，每个物理主机($host_j$)上启动的各类虚拟机副本总数为 q_j，负责执行相应类型的任务，虚拟机处于运行/空闲状态。定义状态量 R_{ik} 表示在该物理主机上的第 k 个虚拟机(类型为VM_i)，是否正在执行相应类型的任务，即表示为：

$$R_{ik}=\begin{cases}1,\text{处于运行状态}\\0,\text{处于空闲状态}\end{cases}，其中：\ i\in\{1,...,M\},k\in\{1,...,q_j\} \tag{4.3}$$

通过分析集群和各个相应类型的虚拟机状态及分布，结合合适的预测算法，可以提前做出相应的控制行为，在提高云计算平台的资源利用率，降低能耗的同时，可以提高系统的实时响应比和稳定性。现做如下控制行为定义：

定义 4.4：对于所有物理主机集合：$<host_j>$和虚拟机类型集合：$<VM_i>$，定义 M×N 的开启控制矩阵 $Start_{M\times N}$。矩阵中的元素 $Start_{ij}$ 定义为：在 $host_j$ 上启动 $Start_{ij}$ 个 VM_i 类型的虚拟机。开启控制矩阵 $Start_{M\times N}$ 表示如下：

$$Start_{M\times N}=\begin{pmatrix} Start_{11} & \cdots & Start_{1N} \\ \vdots & Start_{ij} & \vdots \\ Start_{M1} & \cdots & Start_{MN} \end{pmatrix}$$，其中：$Start_{ij}=\begin{cases} 非0, & 在host_j上启动Start_{ij}个VM_i类型的虚拟机 \\ 0, & 在host_j上不需要启动VM_i类型的虚拟机 \end{cases}$　（4.4）

定义 4.5：对于所有物理主机集合：$<host_j>$和虚拟机类型集合：$<VM_i>$，定义 M×N 的关闭控制矩阵 $Shut_{M\times N}$。矩阵中的元素 $Shut_{ij}$ 定义为：在 $host_j$ 上关闭 $Shut_{ij}$ 个 VM_i 类型的虚拟机。关闭控制矩阵 $Shut_{M\times N}$ 表示如下：

$$Shut_{M\times N}=\begin{pmatrix} Shut_{11} & \cdots & Shut_{1N} \\ \vdots & Shut_{ij} & \vdots \\ Shut_{M1} & \cdots & Shut_{MN} \end{pmatrix}$$，其中：$Shut_{ij}=\begin{cases} 非0, & 在host_j上关闭Shut_{ij}个VM_i类型的虚拟机 \\ 0, & 在host_j上不需要关闭VM_i类型的虚拟机 \end{cases}$　（4.5）

由定义 4.4 和 4.5 可知：本章提出的云计算系统模型中，对于所有的物理主机集合：$<host_j>$，VM_i 类型的虚拟机在某一次资源预配置的过程中需要控制开启或关闭的总数分别定义为 $y'_{i\,start}$，$y'_{i\,shut}$：

$$\begin{cases} y'_{i\,start}=\sum_{j=1}^{N} Start_{ij} \\ y'_{i\,shut}=\sum_{j=1}^{N} Shut_{ij} \end{cases} \quad (4.6)$$

4.1.2 多目标约束优化模型

通过上面的讨论，我们可以得出这样的结论：云计算平台产生的能耗主要取决于开启的主机数和虚拟机数，频繁的开关机同样会带来巨大的额外能耗。因此，云计算平台能耗可以表示为：

$$E=E(n,a_{ij})+E(\Delta V,\Delta H) \quad (4.7)$$

其中 $E(n,a_{ij})$ 表示开启的主机和虚拟机产生的稳定能耗，$E(\Delta V,\Delta H)$ 表示开关虚拟机和物理主机所带来的额外控制能耗，ΔV 代表虚拟机控制值，ΔH 代表主机控制值。使需求的资源配置在尽可能少的主机上可以提高能效，降低能源消耗。本章采取“关闭冗余，开启需求”类型的虚拟机，提高集群整体的能效比。

本章把每个物理主机的可用资源抽象为一个二维向量 $host_j:(MIPS_j^{remain},Mem_j^{remain})$，$MIPS_j^{remain}$ 代表 $host_j$ 的可用 CPU 资源，Mem_j^{remain} 代表 $host_j$ 可用的内存空间。向量空间分析

如下：

（1）主机内存空间（Memory，Mem）的大小决定了该主机能够同时运行虚拟机的数量，也就是说，没有足够的剩余内存无法启动更多的虚拟机。分配给所有虚拟机的 Mem 总和不得超过物理主机 Mem 上限。

（2）主机 CPU 内核单元是所有虚拟机共享的，本章采用虚拟机时间共享策略[16]。内核为每个虚拟机分配时间片。所有虚拟机运行时，CPU 峰值不得超过主机的承受能力。本章使用 MIPS（Million Instructions Per Second，每秒百万级机器语言指令数速率）来衡量 CPU 性能，即分配给所有虚拟机的 MIPS 总和不得超过物理主机 CPU 的 MIPS 上限。

所以，选取二维向量：$host_j:(MIPS_j^{remain},Mem_j^{remain})$ 作为物理主机的资源空间。每个虚拟机亦选取二维向量：$VM_i:(mips_i,mem_i)$ 作为虚拟机的资源需求，即：$host_j:(MIPS_j^{remain},Mem_j^{remain}),VM_i:(mips_i,mem_i)$。

综上所述，将所要研究的问题建立为多目标约束优化模型，其数学形式如下所示：

$Min:$

$$E(\mathrm{n},a_{ij})=\sum_{\mathrm{j}=1}^{n}[p_j^{host}+q_j\times p_j^{vm}]\times t,\ q_j=\sum_{\mathrm{i}=1}^{M}a_{ij} \tag{4.8}$$

$$\begin{aligned}E(\Delta V,\Delta H)=&\sum_{\mathrm{j}=1}^{N}\sum_{i=1}^{M}Start_{ij}\times\Delta p_j^{vmStart}\times\Delta t_j^{vmStart}\\&+\sum_{\mathrm{j}=1}^{N}\sum_{i=1}^{M}Shut_{ij}\times\Delta p_j^{vmShut}\times\Delta t_j^{vmShut}\\&+\sum_{j}[\Delta p_j^{hostStart}\times\Delta t_j^{hostStart}]\\&+\sum_{j}[\Delta p_j^{hostShut}\times\Delta t_j^{hostShut}]\end{aligned} \tag{4.9}$$

$s.t:$

$$\sum_{j=1}^{N}Start_{ij}=y'_{i\,start} \tag{4.10}$$

$$\sum_{\mathrm{j}=1}^{N}Shut_{ij}=y'_{i\,shut} \tag{4.11}$$

$$\sum_{i=1}^{M}[Start_{ij}\times mem_i]\le Mem_j^{remain} \tag{4.12}$$

$$\sum_{i=1}^{M}[Start_{ij}\times mips_i]\le MIPS_j^{remain} \tag{4.13}$$

式（4.8）中 $E(\mathrm{n}, a_{ij})$ 表示开启的主机和虚拟机产生的稳定能耗，正比于集群开启的主机总数 n 和各个主机上开启的各类虚拟机总和 q_j，p_j^{host}，p_j^{vm} 分别表示主机 $host_j$ 的功耗和开启每个虚拟机需要增加的功耗；式（4.9）中 $E(\Delta V, \Delta H)$ 表示开关虚拟机和物理主机所带来的额外控制能耗，其中 $\sum_{j=1}^{N}\sum_{i=1}^{M} Start_{ij} \times \Delta p_j^{vmStart} \times \Delta t_j^{vmStart}$ 等于开启虚拟机过程产生的额外控制能耗，$\sum_{j=1}^{N}\sum_{i=1}^{M} Shut_{ij} \times \Delta p_j^{vmShut} \times \Delta t_j^{vmShut}$ 等于关闭虚拟机过程产生的额外控制能耗，$\sum_{j}[\Delta p_j^{hostStart} \times \Delta t_j^{hostStart}]$ 等于开启主机过程产生的额外控制能耗，$\sum_{j}[\Delta p_j^{hostShut} \times \Delta t_j^{hostShut}]$ 等于关闭主机过程产生的额外控制能耗。$\Delta p_j^{vmStart}, \Delta t_j^{vmStart}, \Delta p_j^{vmShut}, \Delta t_j^{vmShut}$ 分别表示在主机 $host_j$ 上开启虚拟机的瞬时功率、开启虚拟机的时间、关闭虚拟机的瞬时功率和关闭虚拟机的时间，$\Delta p_j^{hostStart}, \Delta t_j^{hostStart}, \Delta p_j^{hostShut}, \Delta t_j^{hostShut}$ 分别表示开启主机 $host_j$ 的瞬时功率、开启时间、关闭主机 $host_j$ 的瞬时功率和关闭时间；在约束式（4.10）中，$\sum_{j=1}^{N} Start_{ij} = y'_{i\,start}$ 表示开启的相应类型的虚拟机总数等于测算需要开启的虚拟机个数，约束式（4.11）$\sum_{j=1}^{N} Shut_{ij} = y'_{i\,shut}$ 表示关闭的相应类型的虚拟机总数等于测算需要关闭的虚拟机个数；约束式（4.12）和（4.13）分别指明集群中物理主机的可用 CPU 和 Mem 资源对虚拟机的约束。

4.2 资源的预配置

为了提高云计算平台的系统资源利用率，降低系统能耗，需要对集群中各个物理主机资源进行配置，完成虚拟机的调度。在此过程中，通过查看物理主机和虚拟机实时信息的方法，只能够得到当前时刻物理主机和虚拟机的负载状况，资源配置过程始终滞后于用户请求。因此，本章依托合适的负载预测模型，采取保守控制策略，对云计算系统内不同类型任务的到达量进行周期性预测和控制，从而实现合理的资源预配置，避免资源配置滞后于用户请求的问题。在预测算法的基础上提出了两种低能耗的资源配置算法：基于概率匹配的资源配置算法（Resource Allocation Algorithm based on Probabilistic Matching，RA-PM）和基于改进型模拟退火的资源配置算法（Resource Allocation Algorithm based on Improved Simulated Annealing，RA-ISA）。RA-PM 算法本质上属于启发式算法，以减少系统能耗为目的，通过适用性主机集

群的划分和供需资源之间的匹配，寻找优化的可行解；RA-ISA 算法依托多目标约束优化模型(公式（4.8）-（4.13）)，搜索全局近似最优解，完成资源的配置。

4.2.1 预测与控制

为了使资源配置能够持续满足各类任务对资源的需求，避免资源配置滞后于用户请求的问题，需对下一周期内各类任务的到达量进行预测。针对不同的任务类型、预测周期和应用场景，需要选取不同的预测算法，如指数平滑法、周期分析法、趋势外推法和马尔科夫预测模型等[72-74]。本章利用三次指数平滑算法预测相应类型负载的大小[73-75]。预测周期的大小应当取决于任务的执行时间、算法耗时、物理主机和虚拟机的开关机耗时等因素。如果选取的预测周期过短，对系统的稳定性会产生较大的影响，同时会增大系统控制能耗的开销。云计算服务提供商可以根据不同的情况合理选择预测周期的大小。

假设系统当前处于第k个周期，第$k+1$个周期$task_i$类型的任务预测值$x_i^{'}(k+1)$表示为：

$$x_i^{'}(k+1)=a_i^{'}(k)+b_i^{'}(k)+c_i^{'}(k) \tag{4.14}$$

其中的参数 $a_i^{'}(k)$、$b_i^{'}(k)$、$c_i^{'}(k)$ 分别为：

$$a_i^{'}(k)=3p_i^1(k)-3p_i^2(k)+p_i^3(k) \tag{4.15}$$

$$b_i^{'}(k)=\frac{\alpha}{2(1-\alpha)^2}[(6-5\alpha)p_i^1(k)-2(5-4\alpha)p_i^2(k)+(4-3\alpha)p_i^3(k)] \tag{4.16}$$

$$c_i^{'}(k)=\frac{\alpha^2}{2(1-\alpha)^2}[p_i^1(k)-2p_i^2(k)+p_i^3(k)] \tag{4.17}$$

式中 $p_i^1(k)$ 为一次平滑值， $p_i^2(k)$ 为二次平滑值， $p_i^3(k)$ 为三次平滑值，计算公式如下：

$$p_i^1(k)=\alpha x_i(k)+(1-\alpha)p_i^1(k-1) \tag{4.18}$$

$$p_i^2(k)=\alpha p_i^1(k)+(1-a)p_i^2(k-1) \tag{4.19}$$

$$p_i^3(k)=\alpha p_i^2(k)+(1-a)p_i^3(k-1) \tag{4.20}$$

式（4.18）-（4.20）分别为一次平滑过程、二次平滑过程和三次平滑过程，其中：$x_i(k)$为第k个周期内$task_i$类型任务的实际负载值；α为平滑系数，取值在(0,1)之间。使用三次指数平滑法需要关注初始值 $p_i^1(0)$ 、 $p_i^2(0)$ 和 $p_i^3(0)$ 的选取问题。在通常情况下可以用最初几个实测值的平均值来代替，或直接采用首个周期的实测值来代替，本章采用首个周期的实测值来

代替。分析可知，通过预测误差修正原预测值得到每一次的平滑预测值。α 的大小表明了修正的幅度。α 的值大，说明修正的幅度大，反之亦然。

实验表明，不论采用何种预测算法，预测值都有可能存在或多或少的误差波动，导致出现预测值小于实时负载值的情况，系统实时性要求得不到保障，除此之外，预测值的频繁波动，导致系统稳定性极差，不能满足实际需求。为了解决上述问题，本章在采用三次指数平滑法预测的基础上，采取保守控制策略：假设在第 *k+1* 个周期，$task_i$ 类型的任务等待集群系统处理的任务总量在 $(c'_{i\min}(k+1), c'_{i\max}(k+1))$ 之间，其中 $c'_{i\min}(k+1)$ 和 $c'_{i\max}(k+1)$ 大小如式（4.21）和（4.22）所示：

$$c'_{i\min}(k+1) = c_i(k) + x'_i(k+1) - \sum_{j=1}^{N} a_{ij} \cdot u_{ij} \cdot T \tag{4.21}$$

$$c'_{i\max}(k+1) = c_i(k) + [x'_i(k+1) + \delta_i] - \sum_{j=1}^{N} a_{ij} \cdot u_{ij} \cdot T \tag{4.22}$$

其中，$c_i(k)$ 表示集群当前时刻需要处理 $task_i$ 类型的任务总量；δ_i 表示预测波动误差；$\sum_{j=1}^{N} a_{ij} \cdot u_{ij} \cdot T$ 表示在 T 周期内当前集群对 $task_i$ 类型的任务处理能力。定义实时响应比 γ：

$$\gamma = \frac{V'_i(k+1)}{c'_i(k+1)} \tag{4.23}$$

表示下一时刻 $task_i$ 类型的任务等待处理的任务总量（$c'_i(k+1)$）和需要相应类型的虚拟机总数（$V'_i(k+1)$）之间的比值。当 γ 为 1 时，理论上达到最高响应比。由分析可知：$\sum_{j=1}^{N} a_{ij}$ 表示当前集群中 VM_i 类型虚拟机总数。

（1）如果 $\gamma c'_{i\min}(k+1) > \sum_{j=1}^{N} a_{ij}$，则最大化开启 VM_i 类型的虚拟机的数量为：

$$y'_{i\,start} = \left\lfloor \gamma c'_{i\max}(k+1) - \sum_{j=1}^{N} a_{ij} \right\rfloor \tag{4.24}$$

（2）如果 $\sum_{j=1}^{N} a_{ij} > \gamma c'_{i\max}(k+1)$，则最小化关闭 VM_i 类型的虚拟机的数量为：

$$y'_{i\,shut} = \min\{\left\lfloor \sum_{j=1}^{N} a_{ij} - \gamma c'_{i\max}(k+1) \right\rfloor, \sum_{j=1}^{N} \sum_{k=1}^{q_j} [1 - R_{ik}]\} \tag{4.25}$$

其中 q_j 表示主机 $host_j$ 启动的虚拟机总数，$\left\lfloor \sum_{j=1}^{N} a_{ij} - \gamma c'_{i\max}(k+1) \right\rfloor$ 表示该种类型的虚拟机冗余量，$\sum_{j=1}^{N}\sum_{k=1}^{q_j}[1-R_{ik}]$ 表示当前该种类型的虚拟机空闲量，关闭的个数取二者较小值。

（3）如果 $\gamma c'_{i\max}(k+1) > \sum_{j=1}^{N} a_{ij} > \gamma c'_{i\min}(k+1)$，则维持现状。

通过采用上述保守控制策略，可以有效的提升系统的响应比和稳定性。

4.2.2 基于概率匹配的资源配置算法

我们已经将任务的调度问题抽象为“以尽可能最优的方式，将能够满足用户需求的多个虚拟机调度到集群规模为N的主机上”。本章从低能耗绿色计算出发，提出了两种低能耗的资源配置算法，旨在实现激活主机集合之间更好的负载均衡性和资源的最大化利用。首先，介绍基于概率匹配的资源配置算法（RA-PM）。

在开启/关闭的过程中，RS会评估可用物理主机的适用性，从而决定所需控制的虚拟机选择哪些物理主机来处理相应的控制策略。这取决于多种因素，包括物理主机的硬件资源使用情况、已启动虚拟机的分布和虚拟机的资源要求等[76]。由于本章采取了“关闭冗余，开启需求”类型的虚拟机，因此选择每个物理主机当前开启的虚拟机的总数 q_j 来对主机的适用性进行划分，定义适用性划分阈值 $q_{threshold}$ ：

（1） 选出 $q_j > q_{threshold}$ 的主机，加入高适用主机集合；

（2） 选出 $0 < q_j < q_{threshold}$ 的主机，加入低适用主机集合；

（3） 选出 $q_j = 0$ 的主机，加入休眠主机集合。

对于需要关闭类型的虚拟机，RA-PM算法优先从低适用主机集合中关闭相应空闲的虚拟机，适当迁移低适用主机集合中的虚拟机，减少开启的主机个数；如果在低适用主机集合中的虚拟机都不满足关闭要求，RA-PM算法会从高适用主机集合中关闭相应空闲的虚拟机。对于需要开启类型的虚拟机，RA-PM算法优先从高适用主机集合中开启相应的虚拟机；如果在高适用主机集合中的虚拟机都不满足开启要求，RA-PM算法再从低适用主机集合中开启相应的虚拟机；如果还是不能满足开启要求，RA-PM算法就从休眠主机集合中激活新的主机。

在资源配置的过程中，需要考虑到高适用主机集合之间负载的均衡和资源的最大化利用。在虚拟机调度过程中，需要均衡物理主机硬件资源（包括CPU与内存资源）的使用，防止其

出现 “木桶效应”。该效应表现为CPU利用率过高、但内存利用率低下，或者CPU利用率低下、但内存利用率过高。所以RA-PM算法在选择控制目标主机的过程中，考虑了待放置虚拟机与目标物理主机的匹配程度，定义VM_i类型的虚拟机对主机$host_j$的匹配函数：

$$MR_{ij} = e^{-\mu\left|\frac{r_i}{R_j}-1\right|} \tag{4.26}$$

其中，$r_i = \frac{mem_i}{mips_i}$，$R_j = \frac{Mem_j^{remain}}{MIPS_j^{remain}}$，$\mu$为常系数。$MR_{ij}$越大，表明$VM_i$类型的虚拟机对主机$host_j$匹配程度越高。主机对当前待分配虚拟机接受的概率取决于匹配概率和对该虚拟机的容量大小。考虑到所选主机集合的负载状况，能够使得激活主机集合获得更好的负载均衡，待分配虚拟机被当前主机接受的概率定义为：

$$P_{ij} = \frac{MR_{ij} \cdot h_j^{Capacity}}{\sum MR_{ij} \cdot h_j^{Capacity}} \tag{4.27}$$

依据式（4.27）选择一个合适的物理主机来放置该类型的虚拟机。

基于概率匹配的资源配置算法，算法的伪代码如图4.1所示：

Resource allocation algorithm based on probabilistic matching，RA-PM

INPUT：VMInfo, HostInfo;

OUTPUT：(Solution, TotalPower), HostInfo;

```
Preprocessing:
01      VMInfo = VMEnQueue() ;
02      (High-fitnessHosts, Low-fitnessHosts, SleepHosts) = ClassificationProcess(HostInfo)
Begin:
03      (Solution, TotalPower) = GetNewSolution(HostInfo, VMInfo){
04          For( Each: VMInfo) {
05              VMi = Traversing(VMInfo);
06              MatchingVM2Host(High-fitnessHosts,Low-fitnessHosts,SleepHosts,VMi,Solution,TotalPower){
07                  If(Available(High-fitnessHosts)){
08                      SelectHost (Pij);
09                      count(Solution, TotalPower)
10                  }Else If(Available(Low-fitnessHosts)){
11                          SelectHost (Pij);
12                          count(Solution, TotalPower)
13                          InsertSelectedHost→High-fitnessHosts
14                  }Else{SelectHost (Pij);
```

```
15                    count(Solution, TotalPower)
16                    InsertSelectedHost→ Low-fitnessHosts
17                }
18            }///End MatchingVM2Host()
19        }///End For()
20        Return: (Solution, TotalPower);
21    }///End GetNewSolution()
End:
22    DeployVM2Host(Solution, HostInfo);
23    RefreshHostInfo();
End Over.
```

图 4.1 RA-PM Algorithm

RA-PM算法虚拟机和主机匹配过程详细如下：

步骤 1：遍历待放置虚拟机队列。记当前需要放置的虚拟机类型为VM_i；

步骤 2：选择满足条件的主机集合，策略如下：

a) 计算当前高适用主机集合中每个物理主机所能容纳VM_i类型的虚拟机能力：

$h_j^{Capacity} = \min\left\{Mem_j^{remain}\%mem_i, Cpu_j^{remain}\%cpu_i\right\}$，若$\sum h_j^{Capacity} != 0$，则转步骤3，否则：

b) 计算当前低适用主机集合中每个物理主机所能容纳VM_i类型的虚拟机能力:

$h_j^{Capacity} = \min\left\{Mem_j^{remain}\%mem_i, Cpu_j^{remain}\%cpu_i\right\}$，若$\sum h_j^{Capacity} != 0$，则转步骤3，否则：

c) 计算休眠主机集合中每个物理主机所能容纳VM_i类型的虚拟机能力:

$h_j^{Capacity} = \left\{Mem_j^{remain}\%mem_i, Cpu_j^{remain}\%cpu_i\right\}$，转步骤3；

步骤 3：根据公式（4.26）计算所选主机集合中每个物理主机与VM_i类型的虚拟机的匹配程度：MR_{ij}；

步骤 4：从所选主机集合中选择物理主机来放置该类型的虚拟机。依据公式（4.27）选择一个合适的物理主机来放置该类型的虚拟机，且$Start_{ij}++$；

步骤 5：用下式刷新被选物理主机资源($host_j$)：

$$\begin{cases} MIPS_j^{remain} = MIPS_j^{remain} - mips_i \\ Mem_j^{remain} = Mem_j^{remain} - mem_i \\ ++TotalVm \end{cases} \quad (4.28)$$

若所选主机集合为低适用主机集合，将该主机归类为高适用主机集合类型；若所选主机集合为休眠主机集合，将该主机归类为低适用主机集合类型。

步骤 6：若虚拟机放置队列遍历结束，则程序终止，否则转步骤 1。

4.2.3 基于改进型模拟退火的资源配置算法

RA-PM算法是一种低能耗资源配置算法，先对可用物理主机的适用性进行评估，考虑受控虚拟机与目标物理节点的匹配程度。相比传统的任务调度算法，在资源配置的过程中，RA-PM算法能够实现激活的主机集合之间更好的负载均衡和资源大的最大化利用，激活更少的主机，降低系统的能耗，实现高效能绿色云计算。但该算法本身存在固有的缺陷：得到的解只是可行解，并非最优解。为了尽可能得到资源配置最优方案，进一步地，本章引入基于改进型模拟退火的资源配置算法，搜索全局近似最优解，完成资源的配置。

由公式（4.8）-（4.9）可知，对于求解多目标约束最优解的过程为NP-难问题[77]。为快速搜索出近似最优解，本章采用基于改进型模拟退火的资源配置算法完成资源的配置。在传统模拟退火算法的基础上增加“存储环节”来记录最优解，避免搜索过程中由于执行“概率接受”而遗失当前遇到的最优解。算法的伪代码如图4.2所示：

Resource allocation algorithm based on improved simulated annealing，RA-ISA

INPUT：VMInfo, HostInfo;

OUTPUT：(OptimalSolution, OptimalTotalPower), HostInfo;

```
Preprocessing:
01      Define: (OptimalSolution, OptimalTotalPower)→(OS, OTP)=0,
                (CurrentSolution, Current TotalPower)→(CS, CTP)=0,
                (PerturbedSolution, PerturbedTotalPower)→(PS, PTP)=0;
02      Initial:   T_max , T_min , K , r , t = T_max;
03      VMInfo   = VMEnQueue();
04      HostInfo   = GetHostInfo();
Begin:
05      While(t > T_max ){
06          (CS,CTP) = GetNewSolution(HostInfo, VMInfo);
07          For( k=0; k<K; k++){
```

```
08              If(First){
09                  (OS,OTP) = (CS,CTP);
10              }Else{
11                  (PS,PTP) = GetPerturbedSolution(HostInfo, VMInfo, CS);
12                  p= exp(-(PTP - CTP)/ κ *t);
13                  If(CTP > PTP || p > Rand(0,1) ){
14                      (CS,CTP) = (PS,PTP);
15                      If(CTP < OTP){
16                          (OS,OTP) = (CS,CTP);
17                      }///End If()
18                  }///End If()
19              }///End Else
20              Initial: (PS,PTP);
21          }///End For()
22          t =   t * r;
23      }///End While()
End:
24      DeployVM2Host(OptimalSolution, HostInfo);
25      RefreshHostInfo();
End Over.
```

图 4.2 RA-ISA Algorithm

下面对上述算法作进一步说明：

（1）RA-ISA 算法 1-4 行：预处理内容，其中第 1 行定义了算法运算过程中涉及的三个解方案，分别用于记录最优解、当前解和扰动解及对应的能耗；第 2 行初始化算法所需的最高温度、最低温度、当前温度迭代次数和当前温度值；第 3、4 行分别用于获取所需启动的虚拟机信息和当前主机集群信息。

（2）RA-ISA 算法 5~23 行：算法主体过程，从产生的初始解和初始温度值开始，重复“产生扰动解（GetNewSolution()）→ 接受或舍弃扰动解（PerturbationSolution）作为当前解（CurrentSolution）→ 记录最优解（OptimalSolution）”的迭代，并逐步衰减 t 值，算法终止得到最优解 OptimalSolution。RA-ISA 算法中，在当前解的领域结构内以一定概率“产生扰动解”，尽可能保证了扰动解遍布全部的解空间，其中解空间满足公式（4.10）-（4.13）的约束条件；“接受或舍弃扰动解”依据了 Metropolis 准则：在温度 t 时趋于以 $e^{-(\Delta E/\kappa t)}$ 为概率接受扰动解，其中 ΔE 为扰动解和当前解能耗差值，由式（4.8）-（4.9）计算可得，κ 为 Boltzmann 常数；“记录最优解”是为了记忆到当前状态为止的最优解。

（3）RA-ISA 算法 24~25 行：采用算法得到的最优解，部署需求虚拟机到主机集群中；刷新主机信息。

不同的开启控制矩阵 $Start_{M\times N}$ 构成算法的解空间，其中解空间满足（4.10）-（4.13）的约束条件。虚拟机采用的放置策略是 RA-ISA 算法产生解空间的关键因素。通过 RA-PM 算法产生初始解，在 RA-ISA 的内循环内，对当前解进行适当扰动，产生扰动解，实现局部最优解搜索；当 RA-ISA 的内循环结束，进入外循环，重新通过 RA-PM 算法产生新解，跳出当前解的局部近似最优解搜索。RA-ISA 算法通过内外循环完成全局近似最优解搜索，使得资源预配置的过程消耗更少的能耗。

4.3 实验验证与性能分析

本节从不同角度出发，衡量调度算法性能。实验将针对预测算法和控制策略的准确性、算法的耗时和能耗、开启的主机集群内资源的利用率和激活的主机数等指标来展开实验分析。将本章提出的低能耗任务调度算法对比于基于性能选择的贪婪算法（Greedy）和文献[47]提出的分组遗传算法，我们将其标记为 GABA 。Greedy 算法在选择物理主机放置虚拟机的过程中，总是选择能够放置该虚拟机最多的主机，以此获得当前集群性能上的最优；GABA 则基于二次指数平滑算法实现负载的预测。下文所有的实验结果均为实验 25 次后取平均值所得的结果。

4.3.1 实验环境

通过 C 语言实现 Greedy、RA-PM、GABA 和 RA-ISA 算法，运行在 Intel 酷睿双核 2GHz 主频的 PC 机上，测试各个算法耗时和能耗大小。构建用于仿真的开源云计算测试平台 CloudSim[15]，结合预测算法测试整体方案各项运行指标。本章在复用 CloudSim 框架的基础上，继承默认类 *VmAllocationPolicy* 实现了新的虚拟机放置策略，并在源码基础上修改了大量重要函数的功能代码和执行策略，使之符合本文需要的资源配置模型。

（1）参数设定

根据不同类型的任务对计算资源的不同需求，云服务提供商可以提前决定提供的虚拟机粒度。为了尽可能接近云计算系统复杂的实际情况，实验中分别测试了1、3、5和8种类型虚拟机的划分，即*M*取值分别为1、3、5、8。参考CloudSim云计算仿真平台，我们假设相应类型的虚拟机对CPU和Mem的要求分别如表4.1所示：

表4.1 虚拟机CPU，Mem参数

虚拟机类型	(MIPS，Mem(MB))	虚拟机类型	(MIPS，Mem(MB))
1	(250 , 512)	**5**	(1000 , 512)
2	(500 , 512)	**6**	(500 , 1024)
3	(1000 , 1024)	**7**	(1000 , 2048)
4	(2000 , 2048)	**8**	(2000 , 1024)

本节实验中假设将各种不同类型的虚拟机部署于3种不同配置的主机上，每种类型主机数量相等。将开启和关闭虚拟机的瞬时功率（$\Delta p_j^{vmStart}, \Delta p_j^{vmShut}$）等效为开启每个虚拟机需要增加的功耗（$p_j^{vm}$），关闭主机的瞬时功率（$\Delta p_j^{hostStart}$）等效为主机当前功耗（$p_j^{host}$）。不同类型主机的其它配置、功耗参考文献[42, 78]，分别如表4.2所示：

表4.2 主机对应的CPU，Mem参数

主机类型	(MIPS，Mem(MB))	p_j^{host} (watt)	p_j^{vm} (watt)	$\Delta p_j^{hostStart}$ (watt)	$\Delta t_j^{hostStart}$ (s)
1	(2000,2048)	220	10	288	175
2	(4000,4096)	245	8	299	175
3	(8000,8192)	300	6	331	175

集群主机划分阈值 $q_{threshold}$ 设定为3；ISA法极限高温*T_max*取值100，极限低温*T_min*取值1，内循环迭代次数*K*设定为25，温度衰减系数*r*取值0.8。

（2）数据集选择

在仿真测试平台CloudSim中验证整体方案的可行性，需要提供多组不同类型的任务请求。本章在900个主机节点的数据中心部署3种类型的任务，其中2种类型采用2005年WLCG（Worldwide LHC Computing Grid）数据中心采集的数据[79]。该数据集收集了2005年11月20号到2005年12月5号内提交的多种不同类型的任务执行状况。依据数据集合中用于区分系统资源分配大小的组ID，选择前10天内提交的任务，每隔4小时来进行统计。采用其中较为典型的2组数据：lcg-1、lcg-2。第3种类型的任务请求采用现实中常见的泊松分布p-3。数据集如图4.3所示：

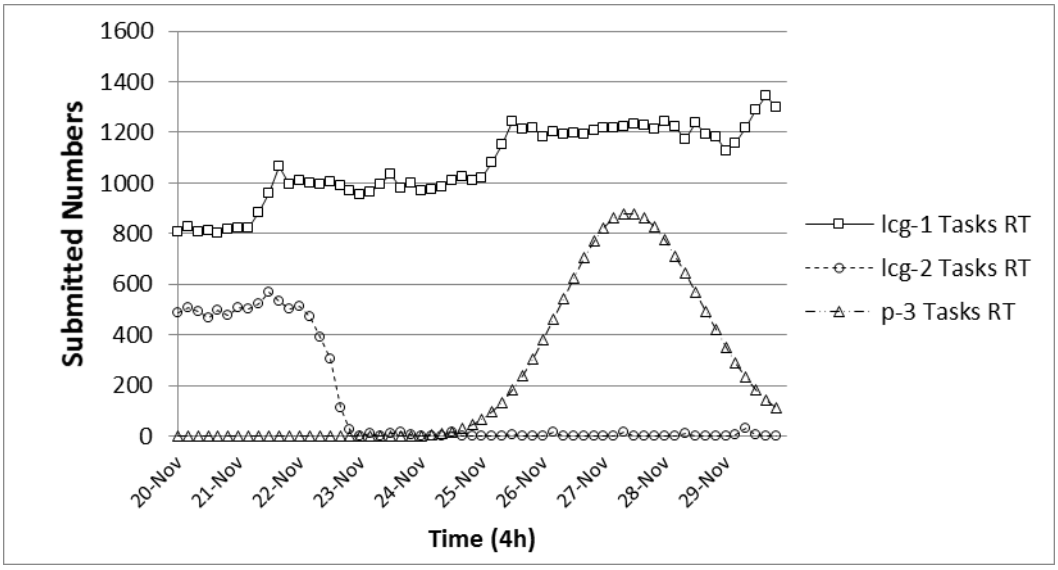
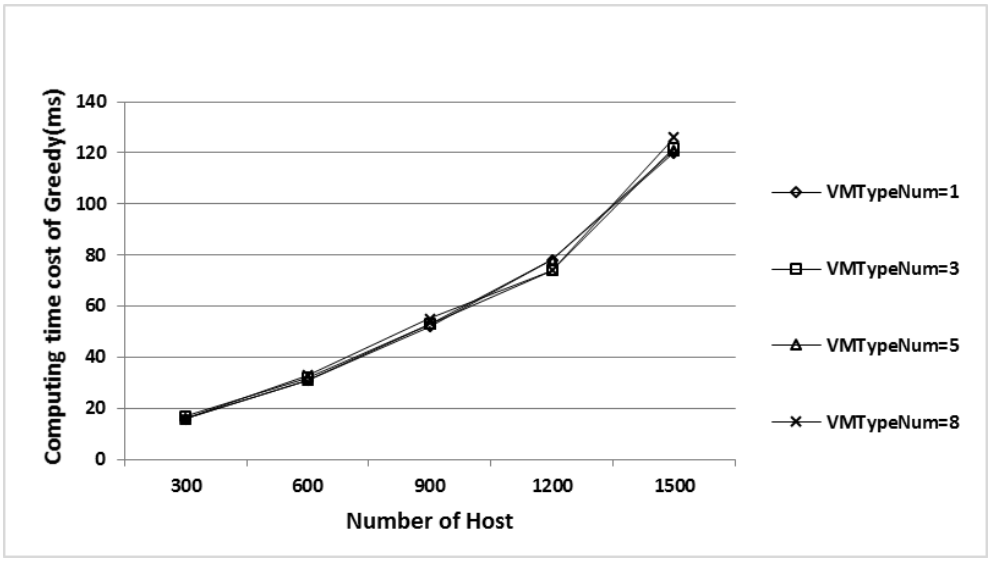


图 4.3 WLCG 2005 workload

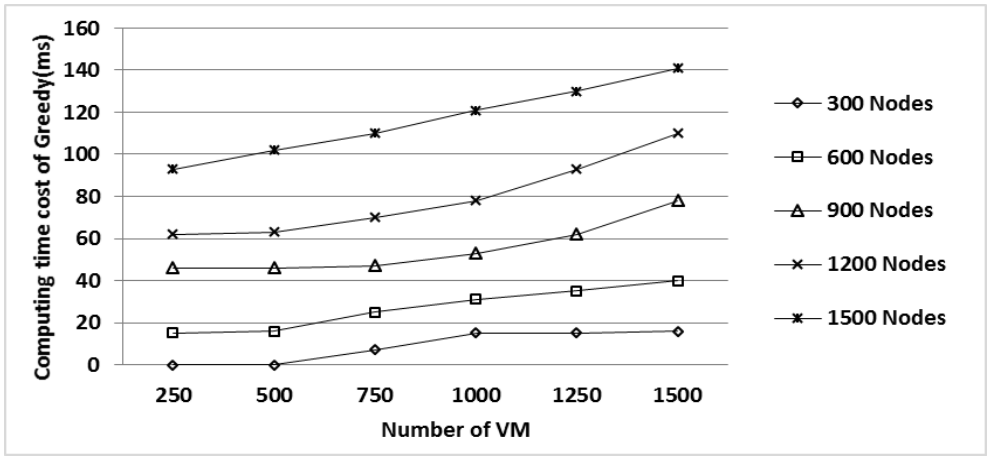
4.3.2 实验结果分析

（1）资源预配置算法的时间开销

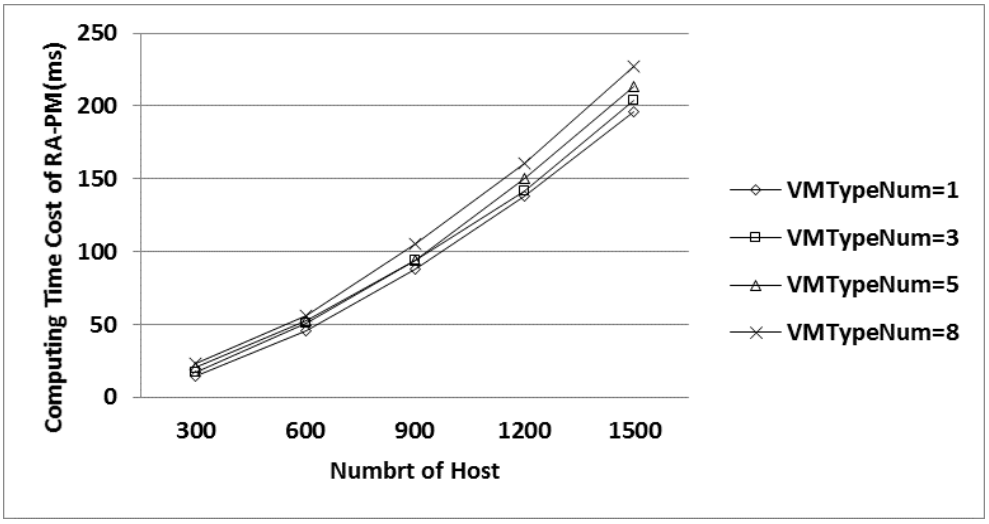
通过 C 语言实现 Greedy、RA-PM、GABA 和 RA-ISA 算法。算法的时间开销受到待配置虚拟机的数量、种类和主机节点的规模等因素的影响。实验中，每一种类型的待配置虚拟机数量相等。图 4.4(a)(c)(e)(g)：控制所有类型的待配置虚拟机总数为 1000（不同类型的虚拟机数量相等），考察待配置虚拟机的种类和主机节点的规模对 Greedy、RA-PM、GABA 和 RA-ISA 算法耗时的影响，从图中可以看出 Greedy、RA-PM、GABA 和 RA-ISA 算法随着集群主机规模的增大，算法耗时呈现类线性增长，而待配置虚拟机的种类对算法的影响很小；图 4.4(b)(d)(f)(h)：控制待配置虚拟机种类为 5，考察待配置虚拟机的数量（不同类型的虚拟机数量相等）和主机节点的规模对 Greedy、RA-PM、GABA 和 RA-ISA 算法耗时的影响，从图中可以看出 Greedy 算法随着待放置虚拟机数量的增大，算法耗时增速小于 RA-PM、GABA 和 RA-ISA 算法。由分析可知：RA-PM 算法耗时略大于 Greedy 算法，对待配置虚拟机的数量（任务数量）和集群规模的时间复杂度为 O(MN)；GABA 和 RA-ISA 算法耗时远大于 RA-PM 算法和 Greedy 算法。在采用 GABA 和 RA-ISA 算法时，需要考虑到资源预配置的周期：采用 GABA 和 RA-ISA 算法对资源预配置的周期有较高的要求，如果资源预配置的周期过短，即预测周期小于 GABA 和 RA-ISA 算法耗时，则 GABA 和 RA-ISA 算法失效，这也是所有求解多目标约束最优解的共性，宜采用 RA-PM 算法。RA-ISA 对比于 GABA 算法（进化 250 次），我们可以发现 RA-ISA 算法耗时更短，效率更高。



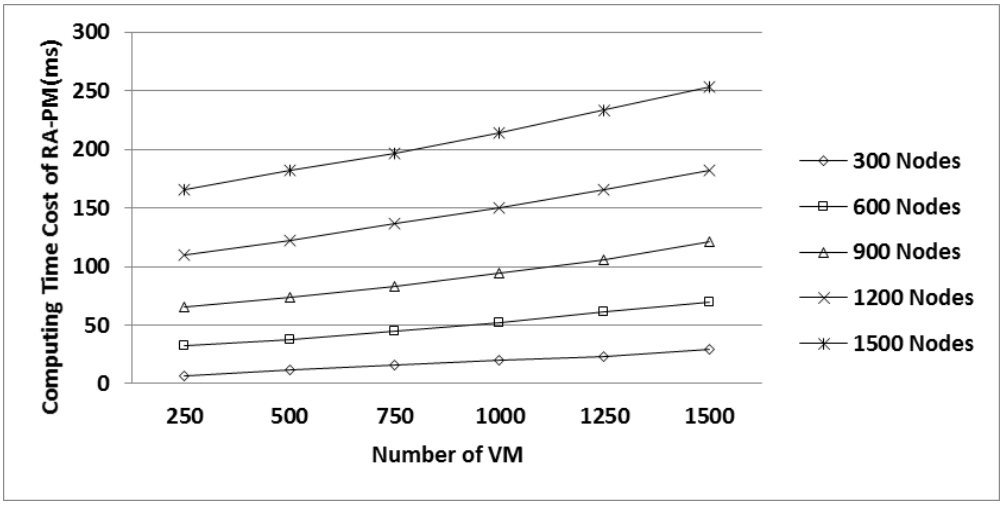
(a) 虚拟机种类和主机节点规模对Greedy算法的影响



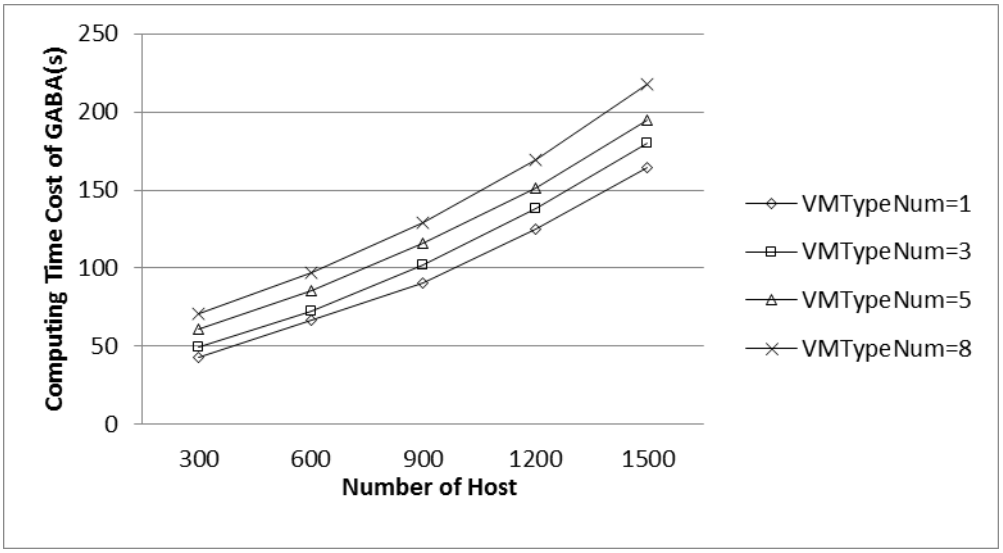
(b) 虚拟机数量和主机节点规模对Greedy算法的影响



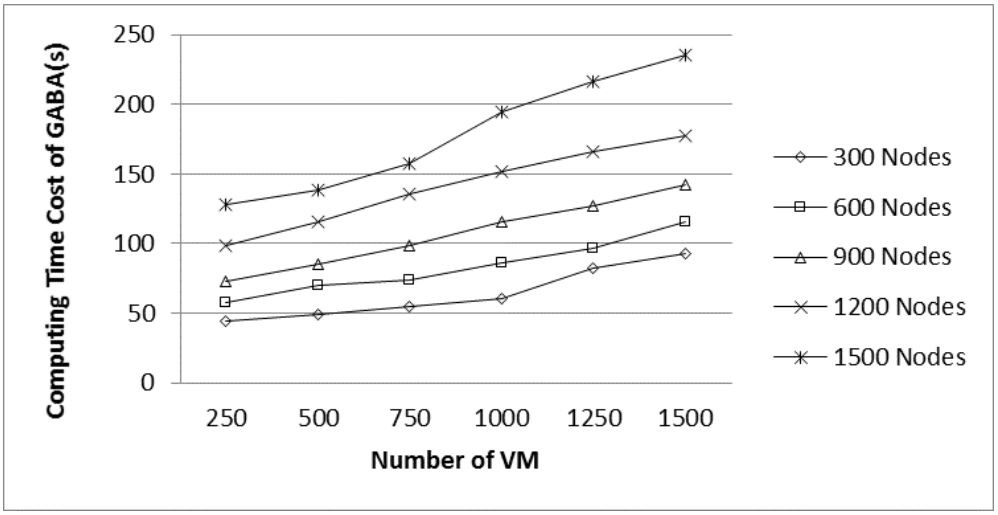
(c) 虚拟机种类和主机节点规模对RA-PM算法的影响



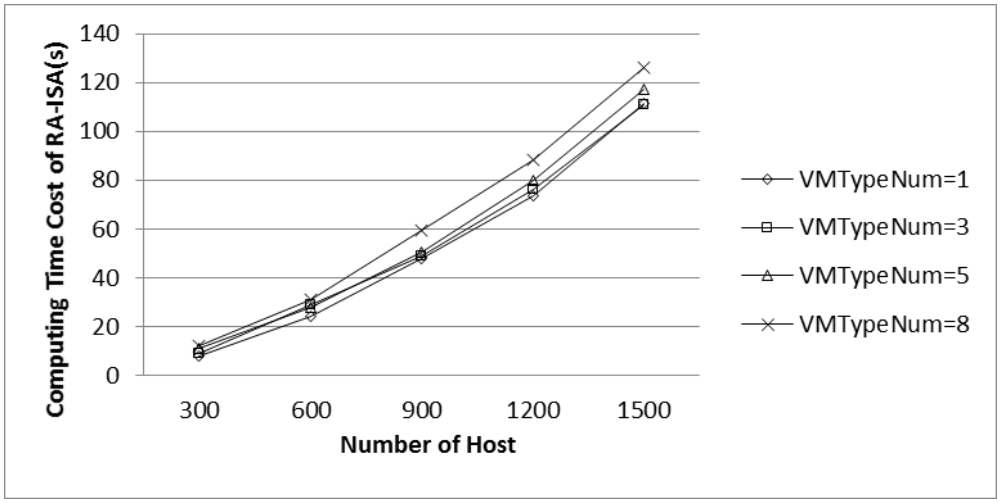
(d) 虚拟机数量和主机节点规模对RA-PM算法的影响



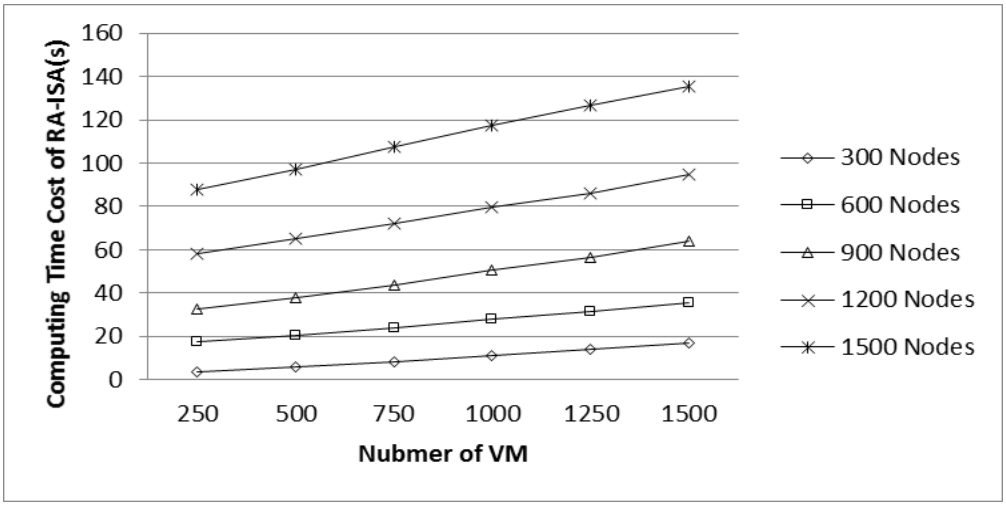
(e) 虚拟机的种类和主机节点的规模对GABA算法的影响



(f) 虚拟机数量和主机节点规模对GABA算法的影响



(g) 虚拟机的种类和主机节点的规模对RA-ISA算法的影响



(h) 虚拟机数量和主机节点规模对RA-ISA算法的影响

图4.4 Greedy、RA-PM、GABA和RA-ISA算法的时间开销

（2）资源预配置算法的能耗

为了衡量两种算法的能耗，我们将这两种算法对比于 Greedy 和 GABA 算法。仿真设定系统中存在 5 种粒度的虚拟机划分，集群规模为 1500 个节点，图 4.5 显示了待放置虚拟机数量和系统能耗的关系。从图 4.5 分析可知：GABA、RA-PM 和 RA-ISA 算法相比 Greedy 算法在能耗上减少了 2-4 倍；随着待放置虚拟机数量的增加，RA-ISA 算法比 RA-PM 和 GABA 算法的优势愈加明显，消耗更少的能耗。在资源预配置周期允许的情况下，采用 RA-ISA 算法优于RA-PM和GABA算法。但是我们应该注意到RA-PM算法耗时明显快于GABA和RA-ISA算法，如果预测周期很短，RA-PM 算法更好。

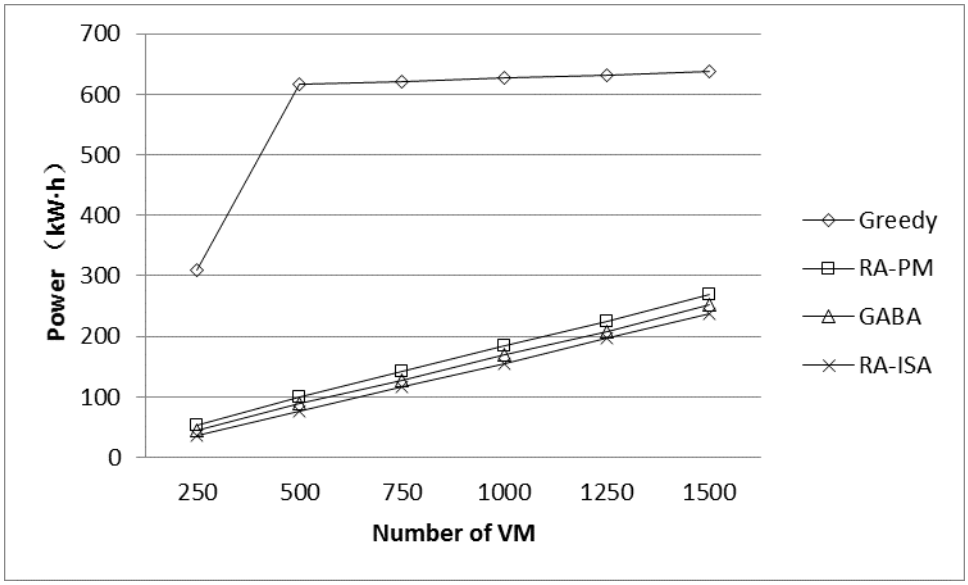
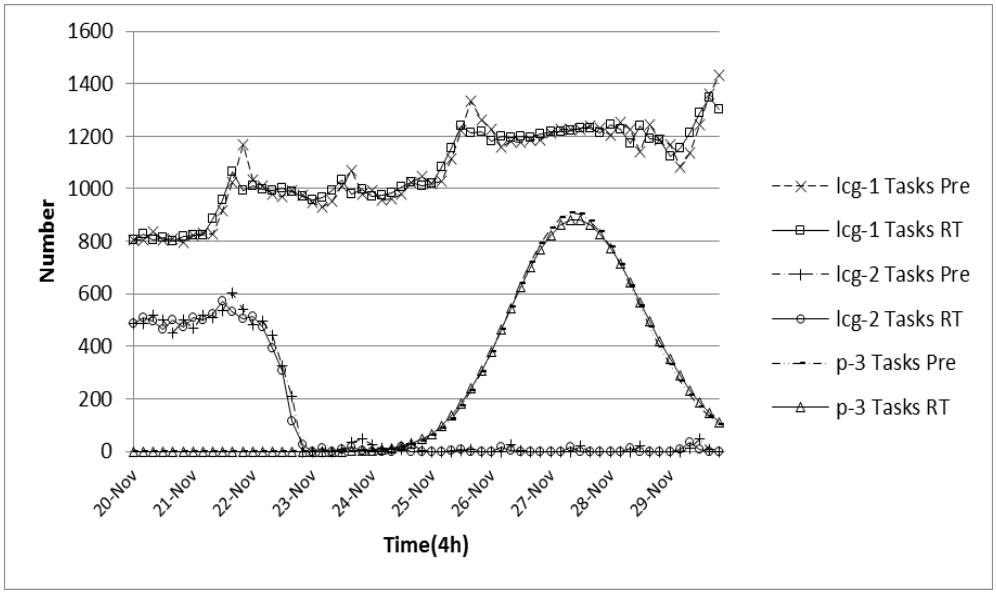


图4.5 Greedy、RA-PM、GABA和RA-ISA算法能耗

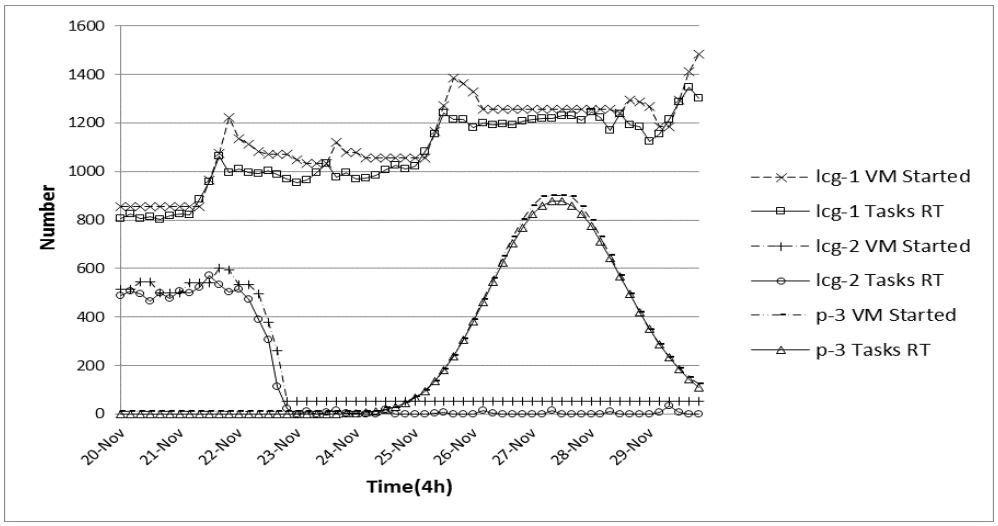
（3）整体方案的性能

(a) 预测与控制的准确度

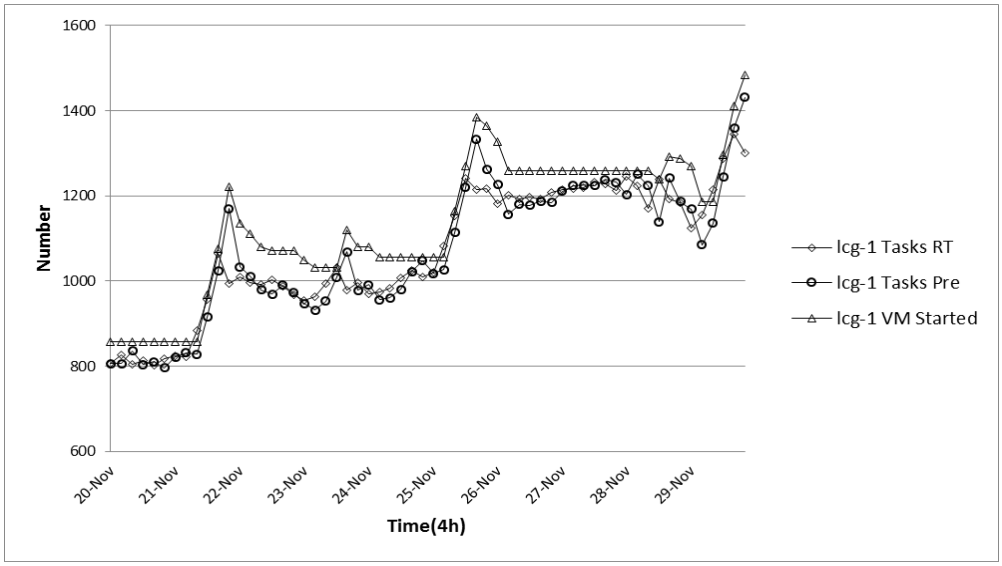
通过云计算仿真系统 CloudSim，在 900 个主机节点的数据中心部署 3 种类型的任务，分别服从 lcg-1、lcg-2 和泊松分布。利用三次指数平滑算法实现各类负载大小的预测，平滑系数 α 取值为 0.5，为了衡量系统预测与保守控制的效果，我们取实时响应比 γ 为 1，各类实时提交的任务与三次指数平滑算法预测的任务如图 4.6(a)中所示。由图 4.6(a)分析可知：存在较多的预测值小于实时提交的任务。对于实时性要求比较高的任务，将导致该种任务不能达到响应比为 1 的要求。本章在三次指数平滑算法的基础上提出保守控制策略，实验效果如图 4.6(b)所示。为了更加直观的分析控制效果，我们以 LCG 第一种任务为例，图 4.6(c)显示了实时提交的任务、预测值和控制值之间的关系。由图 4.6(c)分析可知：对于存在较大波动的任务，采取的保守控制策略有效的提高了系统的实时响应比和稳定性。同时，我们以 lcg-1 为例，给出 GABA 算法中采用的二次指数平滑预测算法和本章提出的预测算法实际开启虚拟机的个数，详细结果如图 4.6(d)所示。



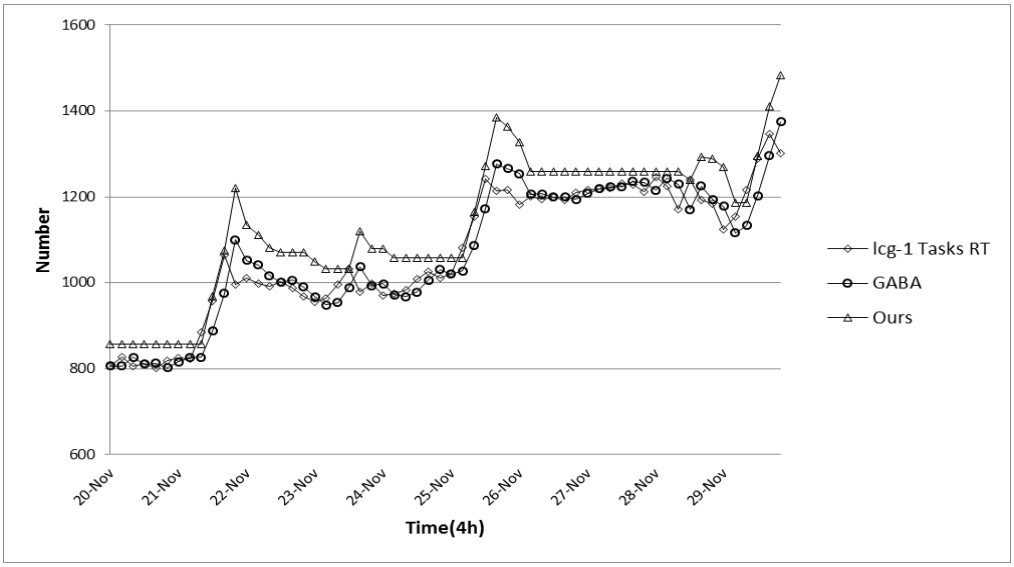
(a) 各类任务的预测值与实际值



(b) 各类任务的控制值与实际值



(c) lcg-1 的实际值、预测值与控制值



(d) lcg-1 的实际值、GABA 和预测控制开启虚拟机个数

图 4.6 预测与控制效果

本章采用的三次指数平滑算法和相应的控制策略存在一定的误差。依据均方差（MSE）分析实验结果：

$$MSE = \frac{1}{n}\sum_{i=1}^{n} e_i^2 \qquad (4.29)$$

其中，e_i 表示第 i 周期的预测误差。平滑系数 α 取值为 0.5，表 4.3 给出本章提出的预测与控制的均方差；表 4.4 给出 GABA 和我们采取预测控制后的均方差。

表4. 3 预测与控制均方差

group	MSE(Pre)	MSE(Control)
LCG1	2237.383	6482.55
LCG2	620.55	2721.167
P3	127.1833	212.1833

表4. 4 GABA和预测控制的均方差

group	MSE(GABA)	MSE(Ours)
LCG1	2348.146	6482.55
LCG2	711.232	2721.167
P3	112.27	212.1833

从表 4.3 和表 4.4 可以看出，控制值产生的均方差均大于预测值和 GABA 产生的均方差，从直观分析可知，这是必然的现象，采取的保守控制策略以适当增加能耗来换取系统性能上的稳定和任务响应比的提高。

(b) 资源使用情况：

Greedy、RA-PM、GABA 和 RA-ISA 算法都采取“关闭冗余，开启需求”的策略。在相

同的预测算法和控制策略情况下，图 4.7 列出了数据中心在资源预配置的过程中采用三种不同预配置算法开启主机的 MIPS 利用率。理论上通过 CloudSim 仿真可以达到的 MIPS 利用率为 100%。从图 4.7 可以看出，Greedy 算法的整体利用率大约在 25%-60%之间，RA-PM 算法的整体利用率大约在 80%-90%之间，两者波动比较大；GABA 和 RA-ISA 算法利用率保持在 90%左右，系统表现平稳。

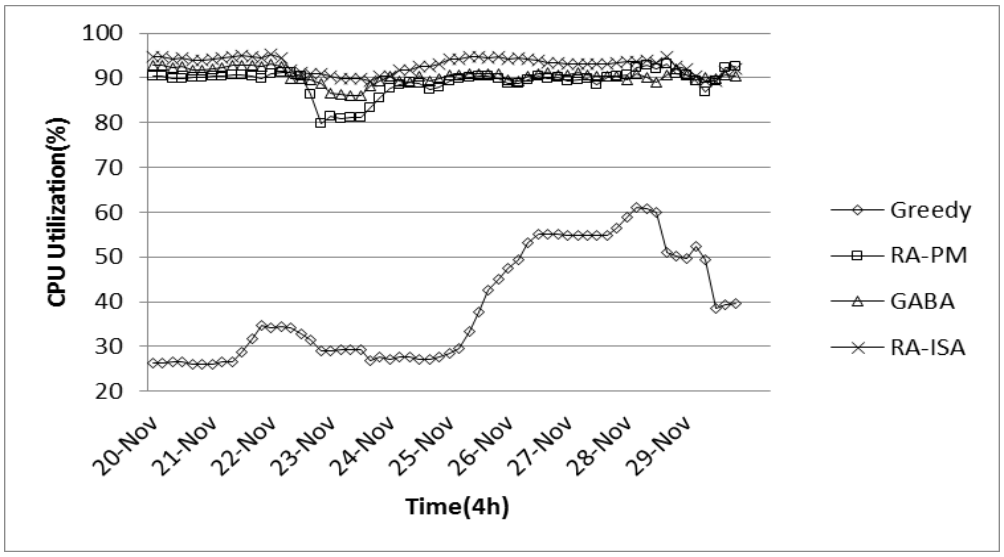


图4.7 Greedy、RA-PM、GABA和RA-ISA算法开启主机的平均利用率

为了更加直观的表现在该云计算系统模型下的资源使用情况，我们对 Greedy、RA-PM、GABA 和 RA-ISA 算法激活的主机数做出分析，如图 4.8 所示：数据中心共 900 个主机节点，部署 3 种类型的任务，相比传统的 Greedy 算法，RA-PM 、GABA 和 RA-ISA 算法激活了更少的主机数，实现了激活主机集合之间更好的负载均衡和资源的最大化利用。

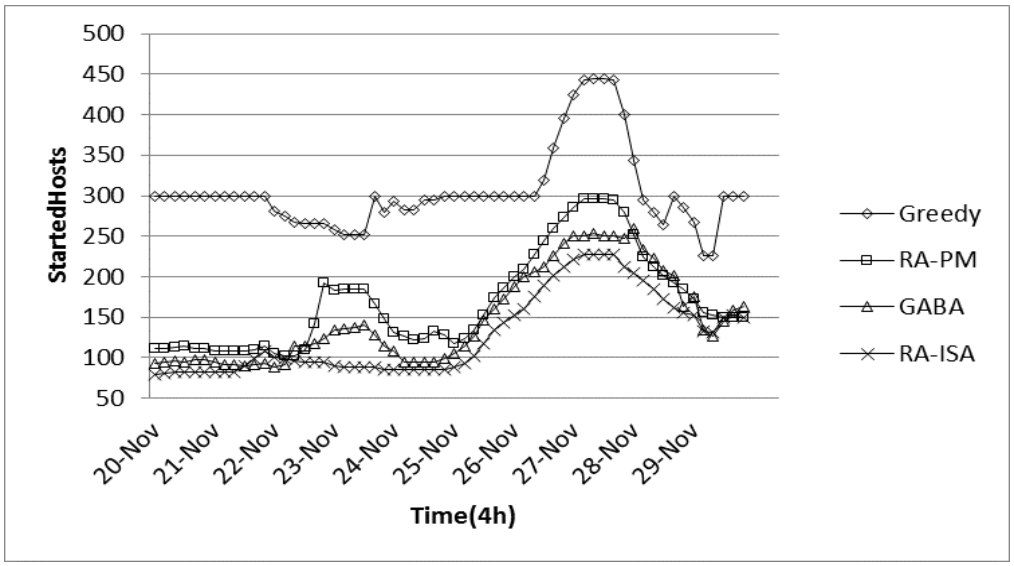


图4.8 Greedy、RA-PM、GABA和RA-ISA算法激活的主机数

(c) 系统功耗：

数据中心在虚拟机部署时分别采用 Greedy、RA-PM、GABA 和 RA-ISA 算法，在各个时

刻产生的功耗如图 10 所示。由图 4.9 分析可知：在资源预配置的过程中，RA-PM 算法相比传统的 Greedy 算法，平均减少约 40%的功耗；GABA 算法相比于 RA-PM 算法，平均减少约 10%的功耗；同时，RA-ISA 算法相比于 GABA 算法，平均减少约 8%的功耗。RA-PM 和 RA-ISA 算法能够降低系统的能耗，实现高效能绿色云计算。尽管 GABA 算法相比于 RA-PM 算法可以节约更多的能耗，但是其耗时是巨大的。我们再次重申：如果预测周期很短，RA-PM 算法优于 GABA 和 RA-ISA 算法。

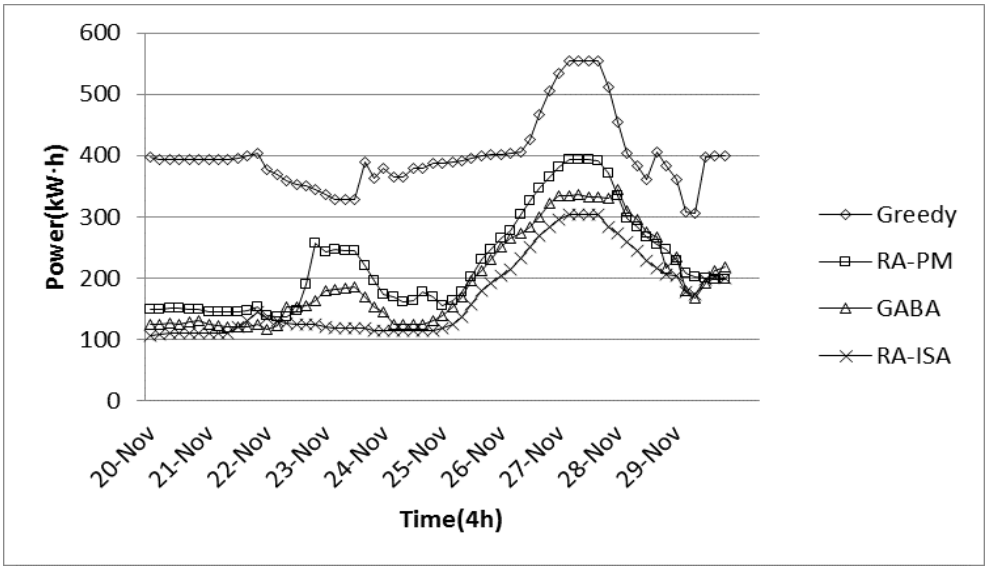


图 4.9 Greedy、RA-PM、GABA 和 RA-ISA 算法产生的系统能耗

4.3.3 性能分析

实验分析可知：采取的三次指数平滑预测算法能够有效解决资源配置落后于用户请求的问题。但是在实时任务波动比较大的情况下使得预测存在一定的误差，导致系统响应比低，稳定性差。本章在预测算法的基础上引入保守控制策略，以适当能耗的增加换取了系统性能上的稳定和任务响应比的提高。RA-PM 和 RA-ISA 算法使得资源预配置更加合理化，激活更少的主机数，提高了系统的资源利用率，降低了系统能耗。系统应用 RA-PM 算法可以减少约 40%的功耗，并且时间开销很小。进一步地，如果迭代周期足够长，RA-ISA 算法能够比 RA-PM 算法减少约 18%的系统功耗。所以，在资源预配置的过程中，RA-PM 算法可以快速地得出局部最优解，在一定程度上降低了系统的能耗，算法耗时少，能够适应短周期资源预配置的需求；RA-ISA 算法通过对全局最优解的搜索，使得云计算系统消耗更低的能耗，但是该算法复杂度高，迭代周期长，不能适用于短周期资源预配置的需求。可以根据实际情况选择其中的某种算法。

4.4 本章小结

本章提出了一套完整的绿色云计算数据中心的节能机制，包括节能模型的建立、负载大小的预测与控制、虚拟化数据中心的全局调度管理和虚拟机的预配置等。

本章最主要的贡献在于提出了两种资源预配置算法：RA-PM 和 RA-ISA。RA-PM 和 RA-ISA 算法对云计算系统的能耗和资源的利用率进行了不同程度的优化，更好地实现了节能降耗，提高了资源的利用率。从部署于 CloudSim 的整体方案看，RA-PM 和 RA-ISA 算法不存在任何硬件的投入，具有较高的可扩展性，能够适应于不同云计算系统的要求，从资源配置的角度实现了绿色计算的目的。

第五章 基于动态负载调节的自适应云计算任务调度策略

配合提出的任务调度模型，需要设计高效的任务调度算法，提高系统资源的利用率，减少整个云计算系统的总能耗。本章首先提出一种面向云计算平台任务调度的多级负载评估方法（Multi-level Load Assessment Method for Cloud Computing Task Scheduling，MLAM）。该方法充分考虑任务节点自身负载的动态变化、不同任务节点性能的差异以及不同任务负载需求的差异，选取了运行队列平均进程数、平均 CPU 利用率、平均内存利用率、平均网络带宽利用率作为评估所需的负载参数，并对其赋予不同的优先级，为大规模服务器集群的任务调度提供一种多级负载评估方法。在此基础上，提出一种基于动态负载调节的自适应云计算任务调度策略（Adaptive Task Scheduling Strategy based on Dynamic Workload Adjustment for Cloud Computing，ATSDWA）。任务节点在运行的过程中及时地自适应负载的变化，按照计算能力获取任务，实现各个节点自调节，同时避免因采用复杂的调度算法，使得管理节点承载巨大的系统开销，成为系统性能的瓶颈。

5.1 面向任务调度的多级负载评估方法

面向任务调度的多级负载评估方法所要解决的技术问题在于如何克服现有云计算平台负载评估机制存在的不足，提出一种面向云计算平台任务调度的多级负载评估方法，在尽可能提高负载评估精度的同时，减少负载评估方法本身带来的系统开销。

5.1.1 负载参数的选择

为了对节点的负载状况进行有效的评估，ATSDWA 算法需要选择合适的负载参数作为评估的标准。CPU 利用率的统计能够反映 CPU 被使用的情况，高 CPU 利用率，说明 CPU 超负荷运作，且硬件不能再接受更多的任务。然而，即使 CPU 的利用率低，CPU 的负载仍然可能很大，这体现在 CPU 维护的任务队列，其中的各个任务（tasks）处在休眠（Sleep）或运行（Runable）状态[80]。理想情况下，调度器会不断地让任务队列中的任务根据获得的 CPU 时间片的大小依次执行，但是当任务队列过长时，由于各个任务对资源的竞争，使得 CPU 在一段时间内处于未响应的状态[81]。这种现象也在我们实验的过程中得到了验证，表现为 CPU 利用率低下，但资源竞争激烈，节点亦处于超负荷的工作状态。此外，实际应用中存在对内存（MEM）资源需求较高但对 CPU 需求较少的任务，因此 MEM 亦是不可忽略的负载参数。

在云计算集群中，各个节点通过网络连接进行通信。如果网络出现阻塞，那么节点之间将无法交互，影响任务的顺利执行。所以网络带宽利用率也需列入考查范围。传统的评估准则仅从 CPU 的利用率来判断节点是否处于超负荷状态，这存在较大的局限性，常常不够科学，必须结合多方面因素来评估节点负载情况。

综上所述，本章提出的面向云计算平台任务调度的多级负载评估方法选取 CPU 每个内核运行队列的平均长度（LoadAverage）、CPU 利用率（CpuUsage）、内存利用率（MemoryUsage）和网络带宽利用率（NetworkBandwidthUsage）作为负载评估的参数。主要出于以下考虑：

（1）运行队列平均进程数。运行队列平均进程数。统计某段时间 CPU 正在执行以及等待 CPU 执行的进程数。运行队列平均进程数较大表明 CPU 处于超负荷状态。例如对于 I/O，Socket 等应用来说，运行队列的平均进程数最容易出现偏大的情况，对该参数进行评估能够有效地避免这种情况的发生。

（2）平均 CPU 利用率、平均内存利用率。任务队列存在多个正在执行的任务，平均 CPU 利用率以及平均内存利用率能够可靠的反映正在执行任务占用系统资源的大小，判断当前节点有无足够大的资源去执行新的任务。

（3）平均网络带宽利用率。反映节点带宽负荷的大小，判断当前节点有无足够网络带宽接收新的任务。如果不考虑网络带宽的因素，会造成网络阻塞现象的发生。

5.1.2 多级负载评估方法

根据上节分析，负载评估方法从集群节点的运行队列平均进程数、平均 CPU 和内存利用率、平均网络带宽利用率等方面考虑。下面首先定义相关参数。

(1) 评估模型

(a) 负载模型

定义 5.1： 运行队列的平均长度定义为 LoadAverage，用 *LA* 来标识，表示在某段时间内的运行队列平均进程数。

定义 5.2： CPU 利用率定义为 CpuUsage，用 *CU* 来标识，表示当前采集周期内的平均 CPU 利用率。

定义 5.3： 内存利用率定义为 MemoryUsage，用 *MU* 来标识，表示当前采集周期内的平均内存利用率。

定义 5.4： 网络带宽利用率定义为 NetworkBandwidthUsage，用 *NBU* 来标识，表示当前采集周期内的平均带宽利用率。

(b) 状态模型

根据上述负载指标，统计任务节点当前已使用的系统开销。根据开销的大小，为任务节点定义当前所处状态。评估模型中将节点分为三种状态，每个任务节点都可能处在下列三种状态之一：

状态 A（饥饿态，HUNGER）：代表任务节点的负载较轻，可以承担更多任务。

状态 B（最优态，OPTIMAL）：代表任务节点的负载合理，当前并行执行任务数合理。

状态 C（饱和态，SATURATION）：代表任务节点的负载较重，当前并行执行任务数超出承受范围。

三种状态之间可以相互转换，如图 5.1 所示。

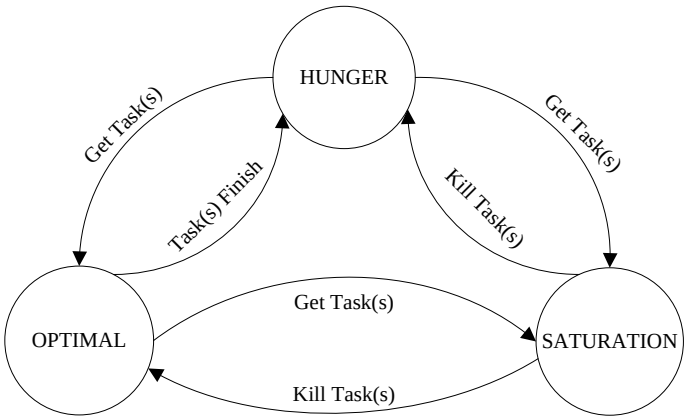


图 5.1 任务节点的状态转换示意图

为了描述和判断节点状态的属性，MLAM 为每个负载参数定义了负载最优值和负载饱和阈值。两个参考值定义如下：

为了描述和判断节点状态的属性，负载评估方法中为每个负载参数分别定义了负载最优阈值和负载饱和阈值，且最优阈值小于饱和阈值，参数的定义如下：

定义 5.5： 负载最优阀值定义为 OptimalValue，用 *OV* 来标识。OV_1—OV_4 分别表示运行队列平均长度、CPU 利用率、内存利用率、网络带宽利用率的最优阈值。当前任务节点各负载参数均小于相应的最优值时，该节点处于“饥饿态”。

定义 5.6： 负载饱和阈值定义为 ThresholdValue，用 *TV* 来标识。TV_1—TV_4 分别表示运行队列平均长度、CPU 利用率、内存利用率、网络带宽利用率的饱和阈值。当前任务节点的任一负载参数大于相应的饱和阈值时，该节点处于“饱和态”。

其它情况下，节点处于“最优态”。

(2) 多级负载评估方法

多级负载评估方法引入负载优先级的概念，将负载参数分为三级。首先，选取运行队列平均长度作为高优先级的负载参数；其次，选取 CPU 利用率和内存利用率作为中优先级的负

载参数；最后，选取网络带宽利用率作为低优先级的负载参数。当高优先级的负载信息大于预设的饱和阀值时，判定该任务节点处于饱和态，无需再采集下一级的负载信息，这样能够有效减少系统开销。

根据上述分析，形成如表 5.1 所示的云计算平台任务调度的负载评估标准：

表5.1 云计算平台任务调度的负载评估标准

优先级	负载指标	指标意义	评估标准		
0	运行队列平均长度	客观反应当前节点正在执行的任务总数	$LA<OV_1$	$OV_1<=LA<TV_1$	$LA>=TV_1$
1	CPU 利用率 内存利用率	统计当前节点正在执行的任务占用系统资源大小	$CU<OV_2$ && $MU<OV_3$	$OV_2<=CU<TV_2$ && $OV_3<=MU<TV_3$	$CU>=TV_2$ && $MU>=TV_3$
2	网络带宽利用率	客观反应当前节点有无足够网络带宽获得新任务	$NBU<OV_4$	$OV_4<=NBU<TV_4$	$NBV>=TV_4$
节点状态 STATE			上述条件均满足时	上述条件均满足时	任一条件满足时
			饥饿态	最优态	饱和态

进一步地，为了避免系统性能的抖动，影响采集节点信息的准确度与精度，引入了负载信息队列，包括：运行队列平均长度、CPU 利用率、内存利用率、网络带宽利用率（LoadAverageQueue、CpuQueue、MemoryQueue、NetworkBandwidthQueue）。每个任务节点维护一个长度为 N 的负载信息队列（N 的值取决于采集周期的长短，表明在一个采集周期内节点取 N 次自身负载信息值），当进入下一个采集周期时，重新采集节点信息，取代上一个周期的数据。则根据当前采集周期获得的负载信息队列，分别对其取平均值，得到当前采集周期内的运行队列平均进程数、平均 CPU 利用率、平均内存利用率、平均网络带宽利用率。

面向云计算平台任务调度的多级负载评估方法的流程图，如图 5.2 所示。

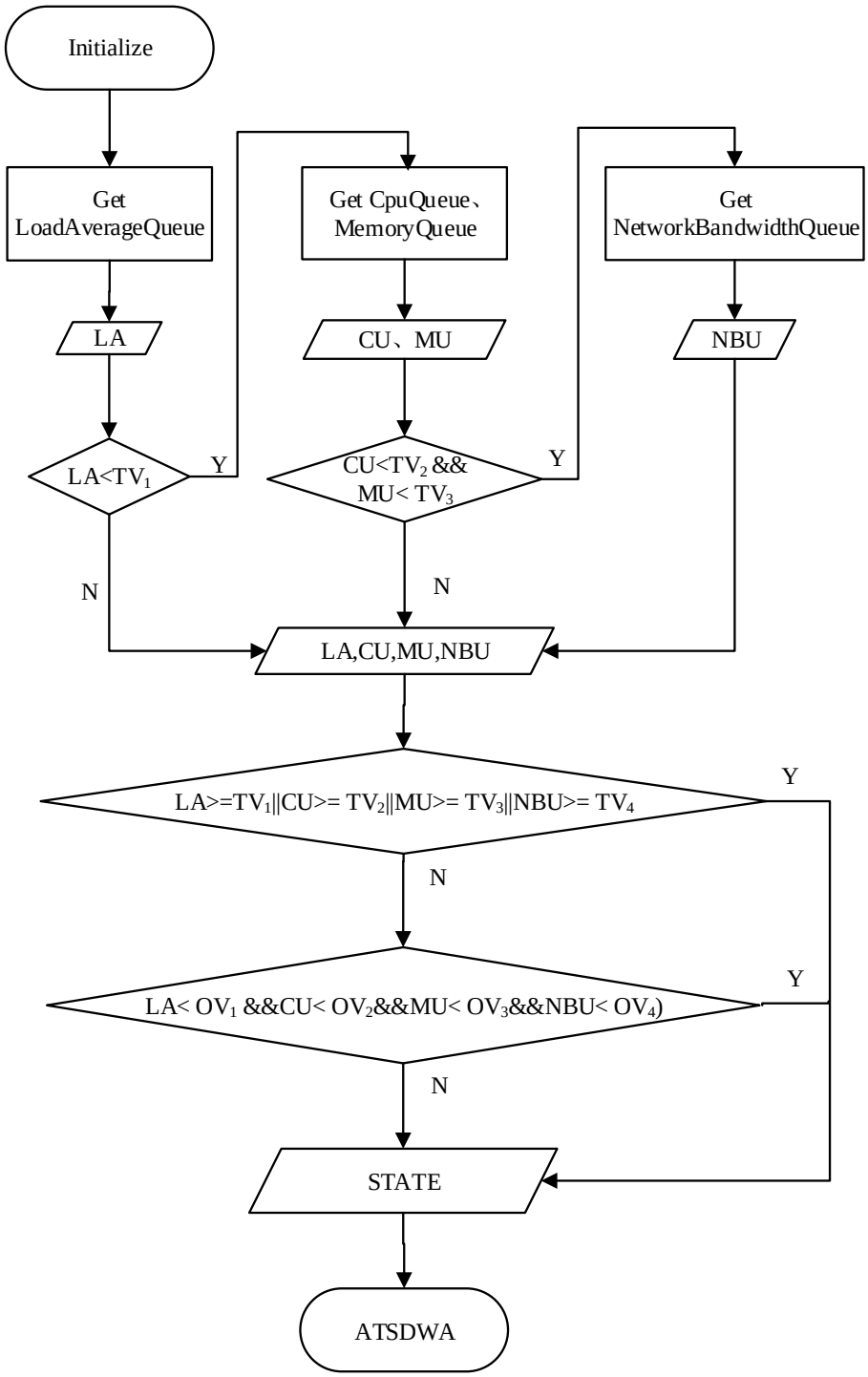


图 5.2 MLAM 方法的流程图

具体包括以下步骤：

步骤 1：初始化相关参数：包括各负载参数的最优阈值（OV_1—OV_4）、饱和阈值（TV_1—TV_4），以及负载信息队列长度 N；

步骤 2：在采集周期内，首先获得一组运行队列进程数，并将其写入负载信息队列 *LoadAverageQueue[N]*；

步骤 3：根据负载信息队列 *LoadAverageQueue[N]*计算当前采集周期内的运行队列平均进程数 *LA*，即：

$$LA=\frac{\sum LoadAverageQueue[i]}{N},\quad (i=0,1,...,N-1) \tag{5.1}$$

比较 *LA* 与 TV_1 的大小，若 $LA< TV_1$，则进入步骤 4，否则跳转至步骤 8；

步骤 4：在采集周期内，分别获得一组 CPU 利用率和一组内存利用率的值，并分别写入负载信息队列 *CpuQueue[N]*和 *MemoryQueue[N]*；

步骤 5：根据负载队列 *CpuQueue[N]*和 *MemoryQueue[N]*计算当前采集周期内的平均 CPU 利用率 *CU* 和内存利用率 *MU*，即：

$$CU=\frac{\sum CpuQueue[i]}{N},\quad (i=0,1,...,N-1) \tag{5.2}$$

$$MU=\frac{\sum MemoryQueue[i]}{N},\quad (i=0,1,...,N-1) \tag{5.3}$$

分别比较 *CU* 与 TV_2，*MU* 与 TV_3 的大小，若 $CU< TV_2$ 且 $MU< TV_3$，则进入步骤 6，否则跳转至步骤 8；

步骤 6：在采集周期内，获得一组网络带宽利用率的值，写入负载信息队列 *NetworkBandwidthQueue[N]*；

步骤 7：根据负载队列 *NetworkBandwidthQueue[N]*计算当前采集周期内的平均网络带宽利用率 *NBU*，即：

$$NBU=\frac{\sum NetworkBandwithQueue[i]}{N},\quad (i=0,1,...,N-1) \tag{5.4}$$

步骤 8：按照以下准则，判断当前节点的状态 STATE：

IF (LA>=TV_1||CU>= TV_2||BU>= TV_3||NBU>= TV_4)

STATE = SATURATION

ELSE IF (LA< OV_1 &&CU< OV_2&&BU< OV_3&&NBU< OV_4)

STATE = HUNGER

ELSE STATE = OPTIMAL

通过上述过程，各个任务节点实现了对自身的负载状态进行准确评估，将该评估结果传递给 ATSDWA，为下一步的任务调度提供负载评估参数。

5.2 基于动态负载调节的自适应任务调度策略

基于动态负载调节的自适应任务调度策略所要解决的问题在于节点性能的差异所带来的

问题，提出一种任务节点在运行的过程中能够及时地自适应负载的变化，按照计算能力获取任务，实现各个节点自调节，同时避免因采用复杂的调度算法，使得管理节点承载巨大的系统开销，成为系统性能瓶颈。

5.2.1 问题分析

随着设备的不断分批换代和升级，很多数据中心实际的集群计算环境已经异构化[82-83]，当前的任务调度算法往往很少考虑集群异构带来的问题[11]。除此之外，即便是同构类型的集群，不同类型的任务对系统资源的需求也不同，也会导致系统负载产生巨大的差异性。现有的任务调度算法未能考虑到各个节点负载的动态变化以及不同节点的性能差异，不能够满足集群的性能要求主要表现在系统的稳定性、快速响应和负载均衡等方面[84]。

云计算平台在实现任务调度时，受限于任务节点有限的资源和计算能力，不能无限制地分配任务。一种有效而简单的方法是通过预先设定任务节点最大可并行执行的任务总数。但由于各个节点负载的动态变化以及不同节点的性能差异，可能导致以下两种情况发生：

（1）若该任务节点属于高性能计算节点，当正在执行任务数已经达到最大可并行执行任务数时，该任务节点将无权继续获取新任务；然而，若此时该任务节点的负载仍然较轻，表明还有能力执行更多的任务，则将产生“饥饿”现象，造成空闲资源的浪费。

（2）若该任务节点属于低性能计算节点，当正在执行任务数小于最大可并行执行任务数时，该任务节点将继续申请新任务；然而，若此时该任务节点的负载已经很重，表明已无能力执行更多的任务，就会出现“饱和”现象，甚至导致节点过载情况的发生，造成节点宕机，发生灾难性事故。

这不仅在很大程度上影响集群的性能，还会造成集群资源的浪费。考虑任务节点自身实际负载的动态变化、不同任务节点性能的差异以及不同任务负载需求的差异，一种更加可靠且高效的方法是对任务节点的相关参数及节点的任务进行自适应调整，使各个任务节点尽可能处于最优态，从而减少集群资源的浪费，提高集群的性能。

5.2.2 主要思想

针对上述问题，本章提出一种适用于异构云计算平台的基于动态负载调节的自适应任务调度策略 ATSDWA。自适应性体现在算法实现过程中：在任务节点运行的过程中，根据自身资源及负载的变化，做出相应的调整，使节点运行性能达到最优，实现自适应性调节。各个任务节点通过周期性地采集负载参数，权衡节点负载状况，动态调节最大可并行执行任务数

（标记为 *MaxTasksCapacity*，取代固定任务槽的概念）的大小。改变传统调度器“一次心跳，全额分配”的方式，使任务节点每次心跳只能获取部分任务，而不是一次填满容量，这将提高集群中每个节点的动态可控性。任务节点在下一个心跳周期重新评估自身的负载情况，然后决定是否接受新的任务。

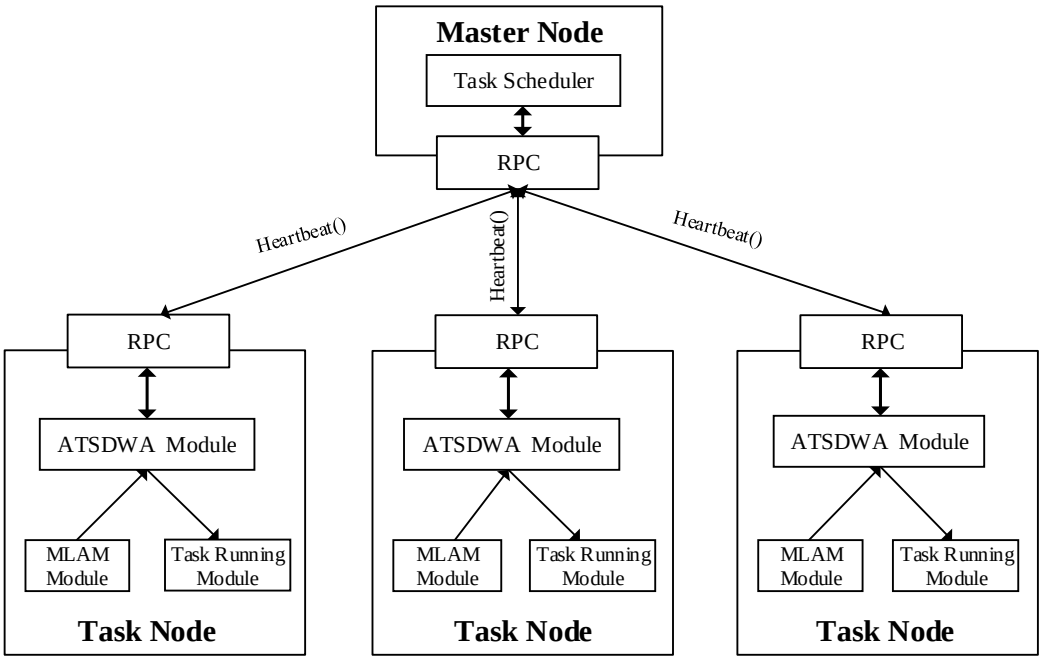


图 5.3 ATSDWA 运行环境

如图 5.3 所示，管理节点（Master Node）端的任务调度器（Task Scheduler）负责任务的分发，当它收到来自任务节点（Task Node）端的心跳（heartbeat）时，判断如果满足条件就分配任务。心跳的传递是通过 Master Node 和 Task Node 之间的远程调用协议（Remote Procedure Call Protocol，RPC）调用实现的。Task Node 端通过负载评估模块（MLAM Module）判断（应用上节提出的 MLAM 方法）节点的负载状况。ATSDWA 模块（ATSDWA Module）实现本章提出的基于动态负载调节的自适应任务调度策略，节点根据负载评估模块得到自身状态，依此进行调节，适应自身的负载变化，实现集群节点自调节，在保证集群响应实时性的同时，避免因采用复杂调度算法导致 Master Node 产生巨大的系统开销。Task Node 在心跳周期内判断自身负载状况，统计负载状态计量值（*LightLoadStatusCount* 和 *OverLoadStatusCount*），以此为判断依据，决定是否改变最大可并行执行任务数。任务执行模块（Task Running Module）实现任务的具体执行。

ATSDWA 策略的算法流程如图 5.4 所示。

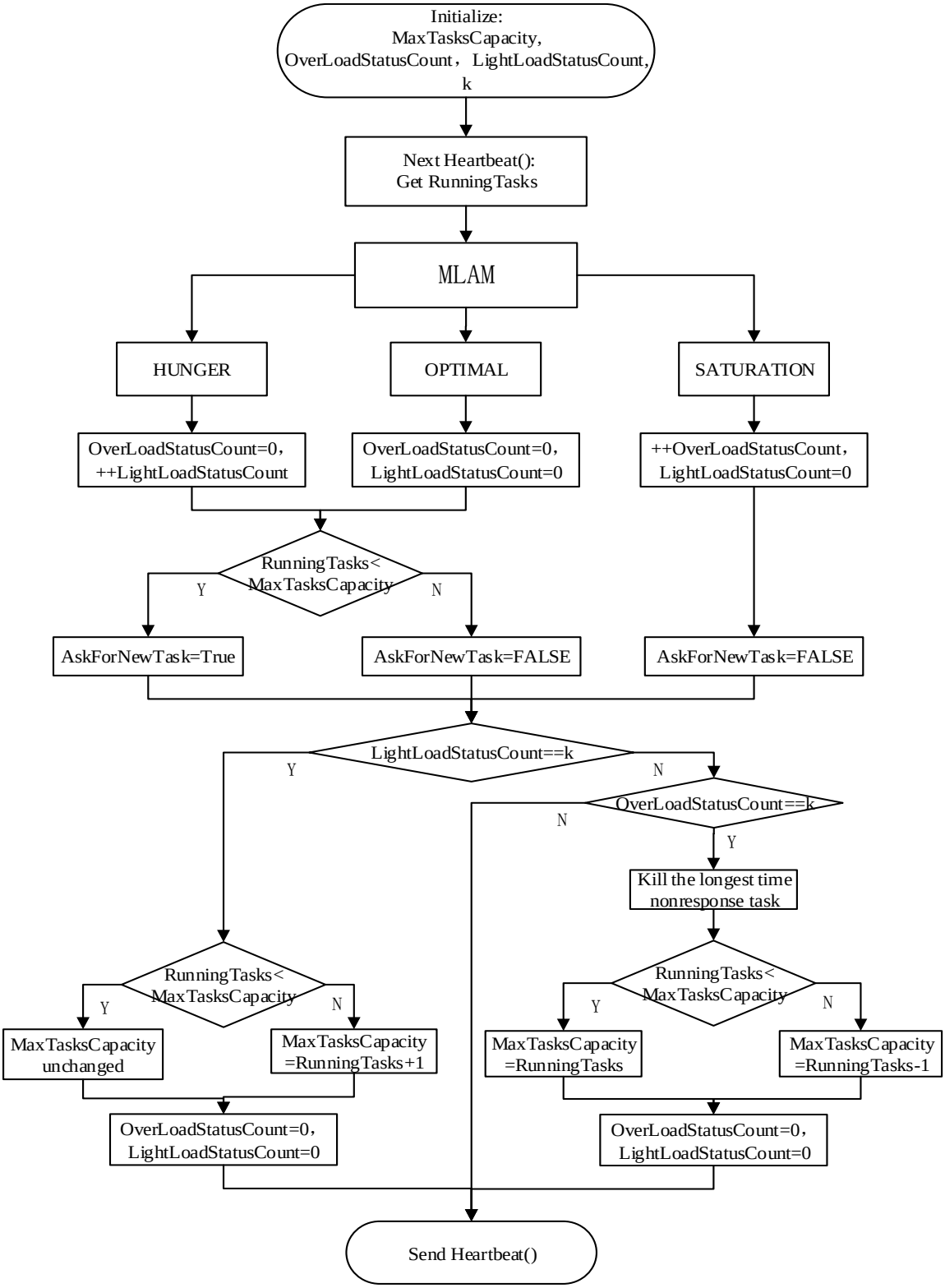


图 5.4　ATSDWA 策略算法流程图

在同构集群中，最大可并行执行任务数（*MaxTasksCapacity*）的大小设置为常数，且这个值在集群运行的过程中不可更改；在异构集群中，为了减少集群自适应调节周期，可以根据任务节点的配置高低设置 *MaxTasksCapacity* 的最小值 *Min*，最大值 *Max*。*Max* 通常与集群中主机的最大 CPU 核数相关，*Min* 与集群中主机的最小 CPU 核数相关。

图 5.4 中 *MaxTasksCapacity* 初始值为 *Min*。设置两个负载计量值 *OverLoadStatusCount* 和

LightLoadStatusCount 值为 0，初始化负载状态计量阀值 *k*。其中，*OverLoadStatusCount* 和 *LightLoadStatusCount* 为负载计量值，分别用于统计节点为“饱和态”和“饥饿态”的次数。节点在运行过程中，负载频繁地调节 *MaxTasksCapacity* 会使系统不稳定，因此只有当“饱和态”或者“饥饿态”连续出现 *k* 次时，算法才进行自适应性调整，*k* 的大小决定了系统进行自适应调节的粒度。在系统实际运行时，可以根据云系统中实际运行情况来调整 *k* 的取值。

5.2.3 算法描述

本节结合 ATSDWA 策略算法的伪代码，进一步描述算法的实现。

ATSDWA 策略算法的具体实现伪代码如图 5.5 所示：

ATSDWA **策略算法**

输入：初始化 *MaxTasksCapacity*：最大可并行执行任务数，最大值为 *Max*，最小值为 *Min*；
OverLoadStatusCount，*LightLoadStatusCount*：负载计量值，值均为 0；
k：负载状态计量阀值

输出：*AskForNewTask*，*MaxTasksCapacity*

```
1   RunningTasks = GetRunningTasks(); /*获得系统当前正在运行的任务数*/
2   CurrentStatus = MLAM();
3   IF (CurrentStatus == HUNGER || CurrentStatus == OPTIMAL) { /*如果当前为饥饿态或最优态*/
4       OverLoadStatusCount = 0;
5       IF (CurrentStatus == HUNGER) { /*如果当前为饥饿态*/
6           LightLoadStatusCount++;
7       } ELSE { /*如果当前为最优态*/
8           LightLoadStatusCount = 0;
9       }
10      IF (RunningTasks < MaxTasksCapacity) { /*判断当前任务节点是否继续获取任务*/
11          AskForNewTask = TRUE;
12      } ELSE {
13          AskForNewTask = FALSE;
14      }
15  } ELSE { /*如果当前为饱和态*/
16      OverLoadStatusCount++;
17      LightLoadStatusCount = 0;
18      AskForNewTask = FALSE;
19  }
20  IF (LightLoadStatusCount == k) { /*如果连续出现 k 次饥饿态*/
```

```
21        IF (RunningTasks < MaxTasksCapacity) { /*判断是否改变 MaxTasksCapacity */
22            MaxTasksCapacity unchanged;
23        } ELSE {
24            MaxTasksCapacity = RunningTasks+1;
25        }
26        OverLoadStatusCount = 0;
27        LightLoadStatusCount = 0;
28    } ELSE IF (OverLoadStatusCount == k) { /*如果连续出现 k 次饱和态*/
29        Kill The Longest Time Nonresponse Task;
30        IF (RunningTasks < MaxTasksCapacity) { /*判断如何改变 MaxTasksCapacity */
31            MaxTasksCapacity = RunningTasks;
32        } ELSE {
33            MaxTasksCapacity = RunningTasks - 1;
34        }
35        OverLoadStatusCount = 0;
36        LightLoadStatusCount = 0;
37    }
```

图 5.5　ATSDWA 策略算法流程图

步骤 1：各个任务节点初始化最大可并行执行的任务数（*MaxTasksCapacity*），两个负载计量值（*OverLoadStatusCount*，*LightLoadStatusCount*），负载状态计量阀值 *k*；

步骤 2：各个任务节点在心跳周期内（HeartBeatTime）通过调用 *GetRunningTasks()*函数得到任务节点正在执行的任务数（*RunningTasks*），同时采用本章提出的负载评估方法进行负载评估，即调用 *MLAM()*函数得到节点当前状态 *CurrentStatus*。在该心跳周期内，若 STATE 为饥饿态（即 *CurrentStatus*=HUNGER），则 *LightLoadStatusCount* 的值加 1，同时将 *OverLoadStatusCount* 清零；若 STATE 为最优态（即 *CurrentStatus*=OPTIMAL），同时将 *LightLoadStatusCount* 和 *OverLoadStatusCount* 清零；若 STATE 为饱和态（即 *CurrentStatus*=SATURATION），则 *OverLoadStatusCount* 的值加 1，同时将 *LightLoadStatusCount* 清零；

步骤 3：任务节点达到心跳时间后，根据 *RunningTasks*、*MaxTasksCapacity* 和负载评估的结果（*CurrentStatus*）决定是否向管理节点索取新任务，若 *RunningTasks*<*MaxTasksCapacity* 且该任务节点处于饥饿态或最优态时，置 *AskForNewTask* 为 *TRUE*，向管理节点索取任务；否则置 *AskForNewTask* 为 *FALSE*，不向管理节点索取新任务；

步骤 4：比较负载计量值（*OverLoadStatusCount*，*LightLoadStatusCount*）和负载计量值阈值 *k* 之间的关系，具体如下：

(1) 当 *LightLoadStatusCount=k* 时，表明连续 *k* 个心跳周期内的负载状态都处于饥饿态，若 *RunningTasks<MaxTasksCapacity*，则置 *MaxTasksCapacity* 的值保持不变；否则置 *MaxTasksCapacity=RunningTasks*+1。将比较负载计量值（*OverLoadStatusCount*，*LightLoadStatusCount*）都清零；

(2) 当 *OverLoadStatusCount =k* 时，表明连续 *k* 个心跳周期内的负载状态都处于饱和态，杀死超时未得到响应的任务，并将此任务汇报给管理节点，请求管理节点对该任务重新分配给合适的任务节点；若 *RunningTasks<MaxTasksCapacity*，则置 *MaxTasksCapacity=RunningTasks*；否则置 *MaxTasksCapacity=MaxTasksCapacity*-1。将比较负载计量值（*OverLoadStatusCount*，*LightLoadStatusCount*）都清零。

(3) 当两个负载计量值（*OverLoadStatusCount*，*LightLoadStatusCount*）都不等于 *k*，则不采取任何措施，任务节点进一步的监控自己负载的变换；

步骤 5：任务节点以 HeartBeatTime 时间间隔周期性地向管理节点发送心跳包；转步骤 2，进行循环调度。

5.3 实验验证与性能分析

为了验证 MLAM 评估方法、分析并比较本章提出的 ATSDWA 策略，需要构建一种有效而可靠的云计算平台。我们结合第二章对 Hadoop 集群架构及现有调度策略的分析，将 MLAM 评估方法和 ATSDWA 策略应用于 Hadoop 云计算平台，并对比于 Hadoop 现有的调度策略 ORIGINAL 和文献[11]提出的 TRAS 算法。本章从不同角度出发，衡量任务调度算法的性能。实验将针对用户提交作业的总响应时间、资源的利用率和系统加速比等指标来展开实验分析。

5.3.1 性能指标

为了验证本章提出的 MLAM 评估方法和 ATSDWA 算法在异构 Hadoop 集群上运行的有效性和可靠性，对其性能优劣进行有效的评估，本章采用以下三个指标作为算法评估的性能指标：

1、作业总响应时间

作业总响应时间是指从提交任务到返回最终处理结果的时间。该指标反映了云计算系统对用户的服务和交互能力，响应时间越短表明系统的服务和交互能力越强。用户在短时间内就知道自己提交任务的运行情况，这使用户获得更好的服务和交互体验。

2、资源利用率

这里的资源利用率包括：CPU 利用率、内存利用率、CPU 每个内核运行队列的平均长度。恰当的资源利用率说明算法具有良好的自适应性，它可以最大化地利用资源，同时又不降低集群的执行效率和服务质量。但资源利用率也不是越大越好，资源利用率过高，会降低系统的执行效率。因此，实验用资源利用率来评价系统的自适应性。

3、加速比

加速比是指在相同的数据和任务处理量的情况下，增加计算集群系统规模对并行计算能力提升的比率。该指标的定义如下：

定义 5.7：固定用户提交作业量的大小，将集群的规模从 P_1 增加到 P_2，相应的，任务的响应时间将从 T_1 减少到 T_2。加速比的计算公式如下所示：

$$SP = \left| \frac{T_2 - T_1}{P_2 - P_1} \right| \tag{5.5}$$

该指标主要反映了云计算系统当前采用的任务调度算法时的可扩展能力。如果通过增加并行处理节点的数量，可以大幅提升系统响应能力，说明采用当前任务调度算法的云计算平台具有高可扩展性。

5.3.2 实验环境设置

本章提出的算法关注的是任务节点的行为，如果能够保证足够多的任务分配到节点上，可以使得实验集群处于高度的负载强度之下，使得实验集群具备“压力测试”的条件，所以即使小规模的分布式系统也可以用来验证我们提出的方案。我们采用 VMware 技术基于实验室设备（包括 PC，服务器和网络交换机）搭建了一个实验平台，将实验部署于这个异构计算集群上。每个虚拟机作为一个 Hadoop 节点，建立了包含 12 个异构节点的测试平台，其中 1 个为 JobTracker 节点（即管理节点），11 个为 TaskTracker 节点（即任务节点）。虚拟机的操作系统均为 ubuntu11.04，网络带宽资源为 1000M。表 5.2 列出了计算节点的配置信息。

表 5. 2 计算节点的配置信息

NodeType	**CPU**	**Memory**	VMInfo
1	Intel Core i5 3337U 2.7Ghz	4.00GB	1* VMType1+1* VMType2
2	Intel Core i5 3210M 2.5Ghz	4.00GB	2* VMType1
3	Intel Core 2 T6400 2.00Ghz	3.00GB	1* VMType1+1* VMType2
4	Intel Core 2 T9600 2.80Ghz	3.00GB	1* VMType1+1* VMType2
5	Intel Core i3-2330M 2.20Ghz	2.00GB	1* VMType1+1* VMType2
6	Intel Core M480 2.67Ghz	2.00GB	1* VMType1+1* VMType2

其中，集群内包含 7 个双核 CPU 节点，内存 512M，标记为 VMType1，其中 1 个为 JobTracter 节点，6 个为 TaskTracker 节点；5 个单核 CPU 节点，内存 218M，记为 VMType3，全部为 TaskTracker 节点。硬盘大小均为 20G。

1、我们对 Hadoop 集群参数进行重置，部分参数配置如表 5.3 所示，其余参数采用默认值[9]：

表 5.3　设定的配置参数

Hadoop 集群参数	默认值	设定值
dfs.replication	3	1
dfs.block.size	64M	32M
dfs.heartbeat.interval	3s	6s
mapred.child.java.opts	200M	150M

相关参数说明如下：

（1）*dfs.replication*：文件创建时产生的存放份数。默认 *dfs.replication* 的值为 3，即做 2 次的备份，当作业文件比较大，且磁盘空间有限时，可以将该值调低，测试时选择为 1，即不做备份，以减少磁盘开销。

（2）*dfs.block.size*：数据块大小。通过它来设置切分文件数据块的大小。数据块的大小决定了集群节点 map 任务的数量，这将直接影响集群的执行效率，默认块的 Hadoop 文件块的大小为 64M。数据块的大小设置取决于文件的大小，如果文件比较大，*dfs.block.size* 设置偏低则会导致过多的分片，造成执行效率的低下；数据块的大小设置还取决于集群规模。实验中我们将其设置为 32M。

（3）*dfs.heartbeat.interval*：心跳间隔，即向管理节点（JobTracker）发送心跳包的时间间隔。

（4）*mapred.child.java.opts*：用于设置每个 Map 或 Reduce 任务消耗的内存大小。默认的大小为 200M。这个参数，很大程度上决定了集群节点内存的利用率。如果它被设置的过低，将导致内存利用率不高，造成资源浪费。

2、不同的参数对实验结果有很大的影响。为了使实验更直观，表 5.4 中列出了 ATSDWA 算法相关参数的设置。这些参数在实验中经过了反复的调整和验证，最终确定了它们的设置值。

k 是负载状态计量阀值。在系统实际运行时，可以根据云系统中实际运行情况来调整 *k* 的取值，在本实验系统中取 10 时达到最优。OV_1—OV_4 分别表示运行队列平均长度、CPU 利用率、内存利用率、网络带宽利用率的最优阈值，TV_1—OV_4 分别对应它们的饱和阀值，当它

们采用表中的值时，算法达到预期效果。本实验设置最大可并行执行任务数的最大值 Max 为 4，最小值 Min 为 1。

表 5.4　设置参数

ATSDWA 相关参数	设定值
k	10
N	3
OV_1	2
TV_1	4
OV_2-OV_4	0. 30
TV_2-TV_4	0.90
Max	4
Min	1

3、为了观察本章提出的 ATSDWA 算法对集群性能影响，对比与现有的调度策略 ORIGINAL 和文献[11]提出的 TRAS 算法。对于调度策略 ORIGINAL，最大可并行执行任务数（MaxTasksCapacity）在集群启动之前通过 mapred.tasktracker.map.tasks.maximum、mapred.tasktracker.reduce.tasks.maximum 参数预先设定，本实验取值为 4。

监控环境：Ganglia[85]。

测试程序：TeraSort 排序基准测试[86]。

5.3.3 实验结果与性能分析

本章在 Eclipse 下编译基于 Hadoop-0.20.2 的源码，通过 jar cvf hadoop-0.20.2-core.jar *命令生成 jar 文件，实现任务调度策略的优化。将编译好的 jar 文件部署到各个集群节点上，然后重新启动集群。本实验中，我们考查计算密集型任务，比如公司的大数据报告、实验数据分析、数据挖掘技术等。它们有一个共性就是该类任务主要消耗 CPU 和内存资源。我们的实验中采用 TeraSort 排序基准测试程序，属于计算密集型测试。另外，TeraSort 是 Hadoop-0.20.2 中的一个排序作业。2008 年 5 月，在 910 个节点的集群上运行 TeraSort 代码，并对 100 亿条记录（1TB）排序，耗时 209 秒，由此 Hadoop 在排序基准评估中赢得了第一名[86]。所以，我们也可以用这个基准测试程序来验证我们提出的算法的性能[87]。为了比较算法的自适应性，我们通过减小任务的大小（减小到 32MB）来增加任务的数量。实验结果对计算密集型任务的任务调度技术有良好的参考价值。实验过程中，主要通过 Ganglia 来监控集群的运行状态，观察集群资源利用情况，对比源码改进前后的基准测试结果。

1、任务总响应时间统计

在建立的包含 12 个异构节点测试平台上，分别对 3072M、4096M、5120M 的信息进行排序统计，每个任务节点所产生的平均任务数分别为 96、128、160。为了保证所测数据的公平性，对每组数据进行三次测试，分别观察 Hadoop 的原始算法、TRAS 算法和本章提出的 ATSDWA 算法对集群性能影响。实验数据如表 5.5 所示：

表 5.5　排序基准测试结果

Files(MB) / Times(s)	3072			4096			5120		
	ORIGINAL	TRAS	ATSDWA	ORIGINAL	TRAS	ATSDWA	ORIGINAL	TRAS	ATSDWA
1	1603	1463	1194	2198	1960	1638	2956	2361	2085
2	1572	1438	1204	2182	1876	1636	2713	2464	2122
3	1533	1455	1221	2131	1934	1628	2624	2490	2078
Average	1569.33	1452	1206.33	2170.33	1923.33	1634	2764.33	2438.33	2095

从表 5.5 看出：集群系统的在每次执行多个任务时，总响应时间趋于平稳。从性能提升率得出：在相同集群规模下，随着任务数的增加，在采用 ATSDWA 策略后，任务节点性能得到相对提升，优化效果明显。ATSDWA 策略能够提高云计算平台和用户的交互能力。为了更加直观比较优化前后的集群性能，经过表 5.5 数据分析，得出集群系统改进前后性能对比情况。如图 5.6 所示：

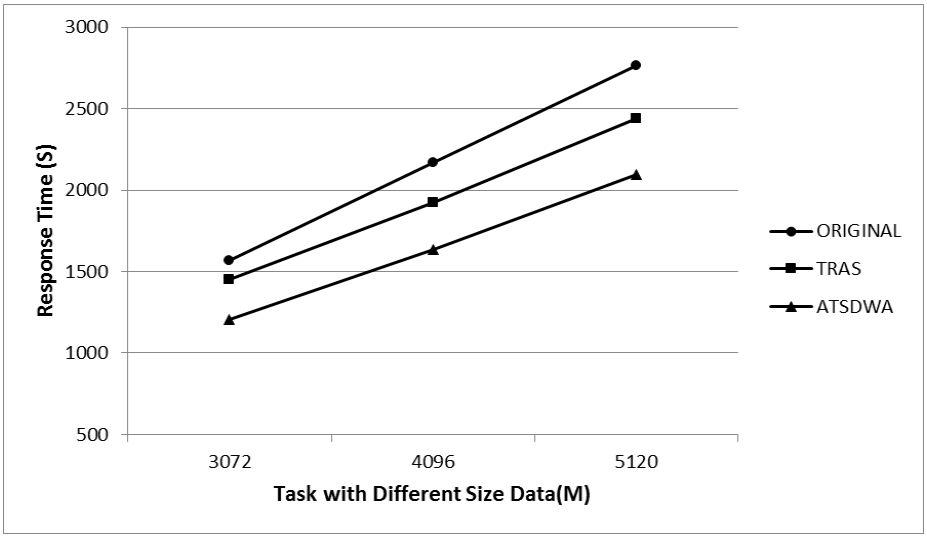


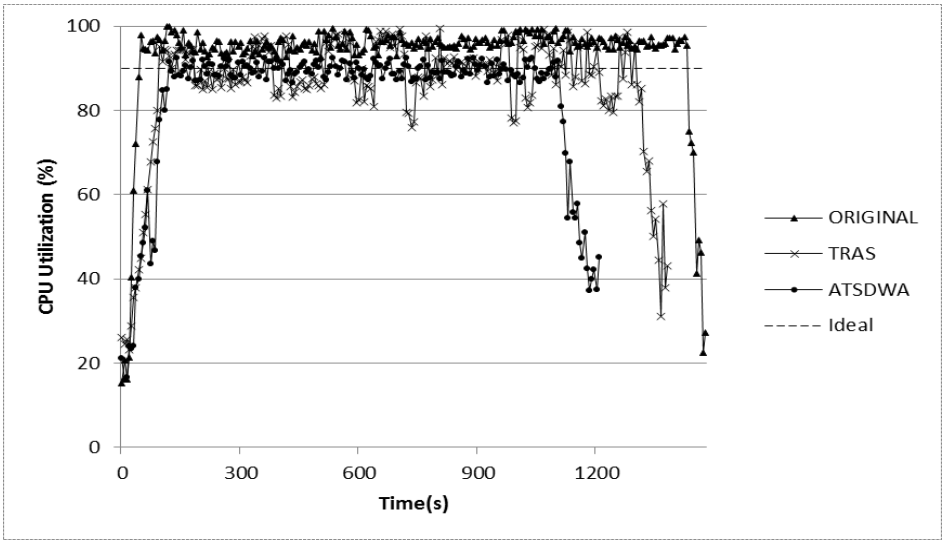
图 5.6 不同数据大小的响应时间对比

从图 5.6 中看出，采用的 ATSDWA 策略在异构 Hadoop 集群中高效可靠，性能优于 ORIGINAL 和 TRAS 算法。能够实现任务节点在运行的过程中自适应负载的变化，实现各个节点自调节。在异构环境下，避免了任务节点频繁的单点配置，提高了部署效率。

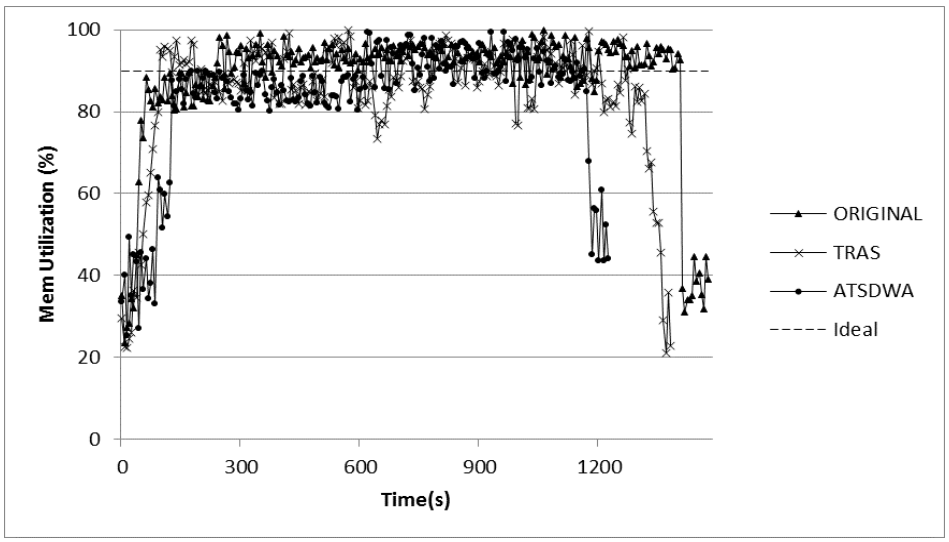
2、资源利用率

随机选取一台虚拟机作为我们的观察对象。同时，运行 3072MB 大小的排序任务来采集

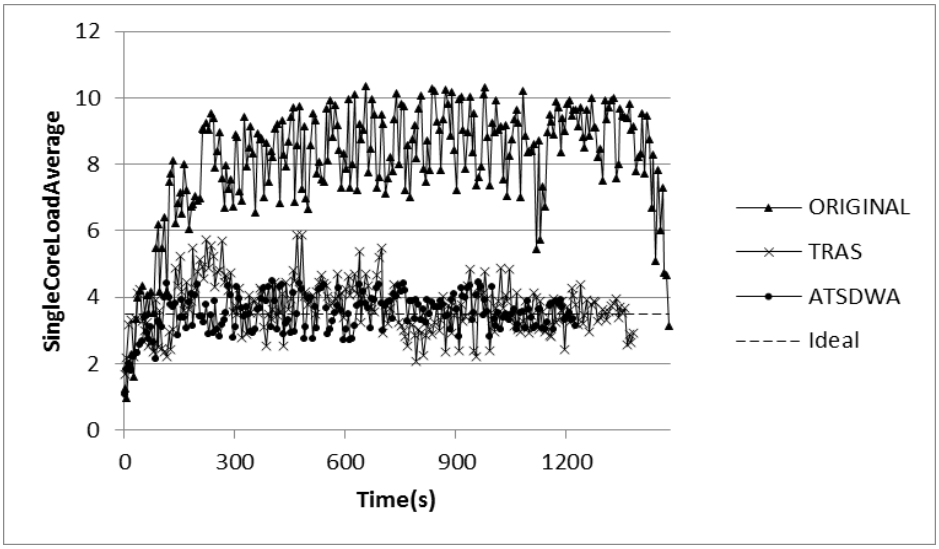
资源使用情况，包括 CPU 利用率、内存利用率以及 CPU 每个内核运行队列的平均长度。根据以上说明，我们记录了资源利用率，并显示在图 5.7(a)(b)(c)中。



（a）CPU 利用率



（b）内存利用率



（c）CPU 每个内核运行队列的平均长度

图 5.7 资源利用率

由图 5.7(a)(b)(c)可以看出 ORIGINAL 算法的资源使用情况一直处于超高的状态，如图(a)中 CPU 利用率一直很高，接近于 100%，当节点长期处于一种超负荷的情况下运行，会导致节点负载过重，节点的运算变慢，甚至出现宕机现象，从而导致任务运行变慢。然而 TRAS 算法考虑了节点的自身情况，但是由于 TaskTracker 将自身的资源信息报给 JobTracker，然后再由 JobTracker 来决定是否分配任务，所以调节滞后，造成资源的使用情况出现忽高忽低的现象，有时甚至达到了 100%。这种状况，造成节点的不稳定，也会使得节点的运算变慢，甚至宕机。再者，TRAS 算法没有考虑 CPU 每个内核运行队列的平均长度这一因素，所以在图(c)中表现不佳。ATSDWA 资源利用率（包括 CPU utilization、MEM utilization 和 SingleCoreLoadAverage）一直在 ideal（饱和阀值）附近，说明集群应用了 ATSDWA 后，能够根据节点自身的资源使用情况，调节任务的获取机制，有效地进行自适应性调节，使节点能够保持在最佳工作状态，提高节点的稳定性，从而提高任务的执行效率，减少作业的响应时间。

实验的网络资源为千兆带宽，且 TeraSort 作业对网络资源的需求不高，所以网络带宽利用率对本实验而言没有参考价值，故未在本节列出。

3、加速比统计

加速比是体现集群自适应性的关键性指标之一。图中折线的斜率（响应时间和集群规模的增量比）反映了集群的加速比。测试在输入数据文件大小均为 3072M 以及其它系统配置均相同的情况下，任务节点的增加（3，5，7，9 和 11 个节点）对任务执行速度的影响。分别观察 TeraSort 排序在 3、5、7、9 和 11 台任务节点下的任务执行时间情况，实验结果如图 5.8 所示。

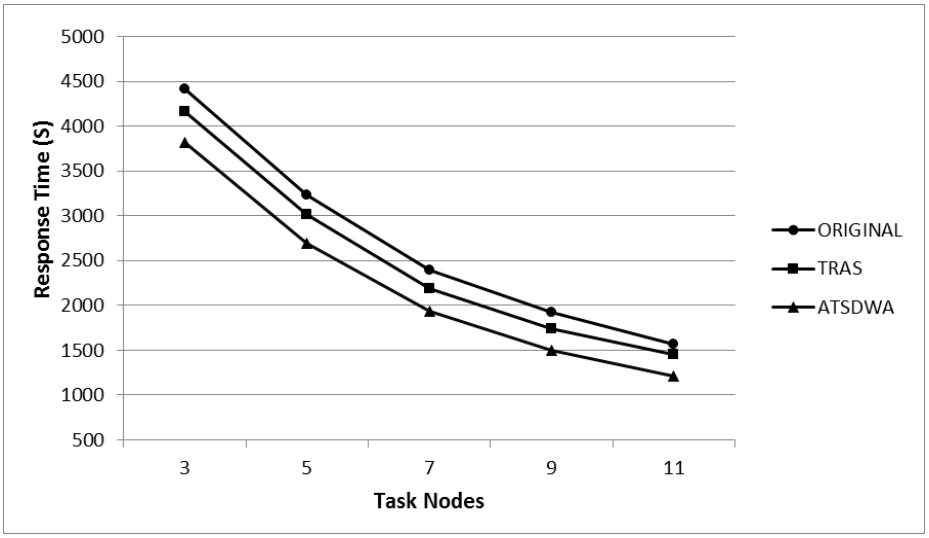


图 5. 8 不同任务节点数量的作业响应时间对比

从图 5.8 中可以看出，随着节点个数的增大，作业的响应时间逐渐减少，其斜率反映了

集群采用不同算法的加速比情况，ATSDWA 算法随着节点规模的增大，能够保值同原算法相近的加速比，优于 TRAS 算法。这说明 ATSDWA 策略有良好的加速比性能，在异构 Hadoop 集群中具有高可扩展性。

通过 Ganglia 监控观察可知，优化后的任务节点负载在阈值之内，负载状况比较平稳，完成作业时间缩短。

5.4 本章小结

在异构云计算集群中，采用 ATSDWA 策略可以有效提高系统性能和可靠性。上述实验表明，ATSDWA 算法对任务节点和管理节点都有良好的优化。在任务节点端任务执行时间减少，节点性能平稳，任务失败率也明显下降，同时避免了“饥饿”和“饱和”现象的发生。在管理节点端，还避免了因超负载而可能发生的节点失效现象。事实上，ATSDWA 同时适用于同构和异构集群。ATSDWA 在提高集群整体任务吞吐率的同时，自身不会增加任务节点端太多的额外开销，实现了各个任务节点自调节，避免任务节点频繁的单点配置，提高了部署效率。

第六章 总结与展望

6.1 总结

随着云计算技术的迅速发展，规模不断扩大的云数据中心每天都在消耗着巨大的能量。不合理的调度策略导致能源浪费严重，使得云数据中心的运营成本不断增加。因此，绿色云计算的概念应运而生。而目前针对绿色云计算的研究与应用已经取得了一定的成果，但仍存在诸多不足，而云数据中心的高能耗、低效率问题已刻不容缓，本文在这些方面进行了一系列的研究与探索，主要包括绿色云计算系统模型及相应子模块的构建、资源配置和任务调度算法的设计及验证等工作。

首先，从实现绿色云计算的内部和外部因素出发，设计了一种高效可靠的绿色云计算系统模型。该模型重点着手于系统内部的资源配置模块和任务调度模块，针对这两个子模块的实际需求，给出了具体的实施方案。资源配置模块使得系统资源配置更加合理，提高了系统资源的利用率，降低了云计算系统整体的能源消耗；任务调度模块能够保证云计算系统具有合理的任务调度机制。

其次，从全局角度出发，抽象任务调度问题为资源配置问题。通过预测用户请求的负载大小，并结合当前系统状态和资源分布，采取保守控制策略，计算下一个周期内任务对系统资源的需求量。然后，建立满足多目标约束的能耗模型，提出基于概率匹配的资源配置算法RA-PM，实现低能耗资源配置。在该算法的基础上，提出基于改进型模拟退火的资源配置算法RA-ISA，进一步降低系统能耗。实验结果表明，预测算法和保守控制策略能够有效避免资源配置滞后于用户请求的问题，提高系统的响应比和稳定性；RA-PM和RA-ISA算法能够激活更少主机，实现激活主机集合之间更好的负载均衡和资源的最大化利用，降低云计算系统能耗。

最后，为了设计合理的任务调度机制，提出一种面向云计算平台任务调度的多级负载评估方法。该方法充分考虑任务节点自身负载的动态变化、不同任务节点性能的差异以及不同任务负载需求的差异，选取合适的负载参数，并对其赋予不同的优先级，为绿色云计算系统的任务调度提供一种负载评估方法。在此基础上，提出一种基于动态负载调节的自适应云计算任务调度策略。任务节点自适应负载的变化，按照计算能力获取任务，实现各个节点自调节，同时避免因采用复杂的调度算法，使得管理节点承载巨大的系统开销，成为系统性能瓶颈。实验结果表明，基于动态负载调节的自适应云计算任务调度策略高效可靠，适用于异构

云计算集群的计算环境，性能优于现有的任务调度策略。

6.2 展望

本文的研究旨在于解决云数据中心的高能耗、低效率问题。在这些方面进行了一系列的研究与探索。一方面提高了异构云计算集群的执行效率，解决集群资源合理利用的问题，却忽略了任务执行过程中出现异常状态的情形。可靠的容错调度策略是云计算系统任务调度领域的一个重要研究课题，这也是我们今后研究的重点。

另一方面，从系统的全局出发，采取“关闭冗余，开启需求”的资源预配置策略，实现绿色云计算。然而，实现高效的绿色云计算还要考虑很多因素，例如主机的负载均衡、虚拟机迁移、网络间的通信量、主机节点的温度、不同地理位置的电费和配套设施的节能等等[88-89]，建立合适的数学模型来寻找开销最低、性能最佳的部署方案至关重要，这些都是今后作者的研究方向。

参考文献

[1] Armbrust M, Fox A, Griffith R, et al. A view of cloud computing [J]. Communications of the ACM, 2010, 53(4): 50-58

[2] Mell P, Grance T. The NIST definition of cloud computing (draft) [J]. NIST special publication, 2011, 800(145): 1-7

[3] Cook G, Van Horn J. How dirty is your data [J]. A Look at the Energy Choices That Power Cloud Computing, 2011

[4] 云计算发展与政策论坛. 数据中心能效测评指南. [EB/OL]. (2013-08-17)[2013-12-08]. http://www.3cpp.org/achievement/89.shtml

[5] Liu Fang, Tong Jin, Mao Jian, et al. NIST cloud computing reference architecture [J]. NIST special publication, 2011, 500(292): 1-28

[6] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters [J]. Communications of the ACM, 2008, 51(1): 107-113

[7] Ghemawat S, Gobioff H, Leung S. The Google file system [C] //ACM SIGOPS Operating Systems Review. New York: ACM, 2003, 37(5): 29-43

[8] Chang F, Dean J, Ghemawat S, et al. Bigtable: A distributed storage system for structured data [J]. ACM Trans on Computer Systems, 2008, 26(2): 1-26

[9] Apache. Hadoop [EB/OL]. (2013-7-10)[2013-7-20]. http://hadoop.apache.org/

[10] Apache. Mapreduce [EB/OL]. (2013-7-10)[2013-7-20].http://hadoop.apache.org/mapreduce/

[11] Yong M, Garegrat N, Mohan S. Towards a resource aware scheduler in hadoop [C] //Proc of the 2009 IEEE Int Conf on Web Services. Los Angeles: IEEE, 2009: 102-109

[12] Nightingale E, Chen P, Flinn J. Speculative execution in a distributed file system [C] //ACM SIGOPS Operating Systems Review. New York: ACM, 2005, 39(5): 191-205

[13] Tang Zhou, Zhou Junqing, Li Kenli, et al. A MapReduce task scheduling algorithm for deadline constraints [J]. Cluster Computing, 2013, 16(4): 651-662

[14] Elghoneimy E, Bouhali O, Alnuweiri H. Resource allocation and scheduling in cloud computing [C] //Proc of the 2012 IEEE Int Conf on Computing, Networking and Communications. Hawaii: IEEE, 2012: 309-314

[15] Calheiros R, Ranjan R, Beloglazov A, et al. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms [J]. Software: Practice and Experience, 2011, 41(1): 23-50

[16] 刘鹏. 云计算[M]. 第二版. 北京：电子工业出版社, 2011: 270-271

[17] 郭兵, 沈艳, 邵子立. 绿色计算的重定义与若干探讨[J]. 计算机学报, 2009, 32(12): 2311-2319

[18] 过敏意. 绿色计算: 内涵及趋势[J]. 计算机工程, 2010, 36(10): 1-7

[19] 孙大为, 常桂然, 陈东等. 云计算环境中绿色服务级目标的分析、量化、建模及评价[J]. 计算机学报, 2013, 36(7): 1509-1525

[20] 林闯, 田源, 姚敏. 绿色网络和绿色评价:节能机制、模型和评价[J]. 计算机学报, 2011, 34(4): 593-612

[21] Tan T K, Raghunathan A, Jha N K. Energy Macro-modeling of Embedded Operating Systems [J]. ACM Trans on Embedded Computing Systems, 2005, 4(1): 231-254

[22] Feng Xizhou, Ge Rong, Cameron K W. Power and Energy Profiling of Scientific Applications on Distributed Systems [C] //Proc of the 19th Int Parallel & Distributed Processing Symp. Colorado: IEEE, 2005: 34-34

[23] Bui V, Norris B, Huck K, et al. A Component Infrastructure for Performance and Power Modeling of Parallel Scientific Applications [C] //Proc of the 2008 compFrame/HPC-GECO workshop on Component-Based High Performance Computing. Karlsruhe: ACM, 2008: 6-6

[24] Yao F, Demers A, Shenker S. A scheduling model for reduced CPU energy [C] //Proc of the 36th IEEE Symp on Foundations of Computer Science. Wisconsin: IEEE, 1995: 374-382

[25] Intel. Ssd-320-specification [EB/OL]. (2013-10-10)[2012-10-21]. http://www.intel.eu/content/www/eu/en/solid-state-drives/ssd-320-specification.html

[26] 马玮骏, 吴海佳, 刘鹏. MassCloud云存储系统构架及可靠性机制[J]. 河海大学学报(自然科学版), 2011, 39(3): 348-352

[27] Andersen D, Franklin J, Kaminsky M, et al. FAWN: A fast array of wimpy nodes [C] //Proc of the ACM SIGOPS 22nd symp on Operating systems principles. New York: ACM, 2009: 1-14

[28] Caulfield A, Grupp L, Swanson S. Gordon: using flash memory to build fast, power-efficient clusters for data-intensive applications [C] //Proc of the 14th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2009, 44(3): 217-228

[29] Leverich J, Kozyrakis C. On the Energy (In)efficiency of Hadoop Clusters [J]. ACM SIGOPS Operating Systems Review, 2010, 44(1): 61-65

[30] Lee K G, Veeravalli B，Viswanathan S. Design of fast and efficient energy-aware gradient-based scheduling algorithms for heterogeneous embedded multi-processor systems [J]. IEEE Trans on Parallel and Distributed Systems, 2009, 20(1): 1-12

[31] Kang J, Ranka S. Dynamic slack allocation algorithms for energy minimizationon parallel machines [J]. Journal of Parallel and Distributed Computing, 2010, 70(5): 417-430

[32] Kim KH, Buyya R, Kim J. Power Aware Scheduling of Bag-of-Tasks Applications with Deadline Constraints on DVS-enabled Clusters [C] //Proc of the 7th IEEE Int Symp on Cluster Computing and the Grid. Rio De Janeiro: IEEE, 2007: 541-548

[33] Wang Lizhe, Laszewski G, Dayal J, et al. Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with DVFS[C] //Proc of the 10th IEEE/ACM Int Conf on Cluster, Cloud and Grid Computing. Melbourne: IEEE/ACM, 2010: 368-377

[34] 张法, 王林, 侯晨颖等. 网络能耗系统模型及能效算法[J]. 计算机学报, 2012, 35(3): 603-615

[35] Nordman B. Saving Large Amounts of Energy with Network Connectivity Proxying [R]. California: Linux Collaborative Summit, 2009: 1-39

[36] 王巍, 罗军舟, 宋爱波. 基于动态定价策略的数据中心能耗成本优化[J]. 计算机学报, 2013, 36 (3): 599-612

[37] 宋杰, 李甜甜, 闫振兴等. 一种云计算环境下的能效模型和度量方法[J]. 软件学报, 2012, 23(2):200-214

[38] Stoess J, Lang C, Bellosa F. Energy management for hypervisor-based virtual machines [C] //USENIX Annual Technical Conference. California: USENIX, 2007: 1-14

[39] Nathuji R, Schwan K. VirtualPower: coordinated power management in virtualized enterprise systems [C] //ACM SIGOPS Operating Systems Review. New York: ACM, 2007, 41(6): 265-278

[40] Oh Y F, Kim S H, Eom H, et al. Enabling cosolidation and scaling down to provide power management for cloud computing [C] //Proc of the 3rd USENIX conf on Hot Topics in Cloud Computing. Portland: USENIX, 2011: 14-14

[41] Van H N, Tran F D, Menaud J M. Performance and power management for cloud infrastuctures [C] //Proc of the 3rd Int Conf on Cloud Computing. Florida: IEEE, 2010: 329-336

[42] Kusic D, Kephart J, Hanson J, et al. Power and performance management of virtualized computing environments via lookahead control [J]. Journal of Cluster Computing, 2009, 12(1):1−15

[43] Xu Jing, Fortes J A B. Multi-objective virtual machine placement in virtualized data center environments [C] //Proc of the 2010 IEEE/ACM Int Conf on Green Computing and Communications & Cyber, Physical and Social Computing. Hangzhou: IEEE, 2010: 179-188

[44] Xie Ruitao, Jia Xiaohua, Yang Kan, et al. Energy Saving Virtual Machine Allocation in Cloud Computing

[C] //Proc of the 33rd Int Conf on Distributed Computing Systems Workshops. Philadelphia: IEEE, 2013: 132-137

[45] Kord N, Haghighi H. An energy-efficient approach for virtual machine placement in cloud based data centers [C] //Proc of the 5th Conf on Information and Knowledge Technology. Shiraz: IEEE, 2013: 44-49

[46] Beloglazov A, Buyya R. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers [J]. Concurrency and Computation: Practice and Experience, 2012, 24(13): 1397-1420

[47] 米海波, 王怀民, 尹刚等. 一种面向虚拟化数字中心资源按需重配置方法[J]. 软件学报, 2011, 22(9): 2193-2205

[48] 叶可江, 吴朝晖, 姜晓红, 何钦铭. 虚拟化云计算平台的能耗管理[J]. 计算机学报, 2012, 35(6): 1262-1685

[49] Berral J L, Goiri I, Nou R, et al. Towards energy-aware scheduling in data centers using machine learning [C] //Proc of the 1st Int Conf on Energy-Efficient Computing and Networking. Passau: ACM, 2010: 215-224

[50] Mezmaz M, Melab N, Kessaci Y, et al. A parallel bi-objective hybrid for energy-aware scheduling for cloud computing systems [J]. Journal of Parallel and Distributed Computing, 2011, 71(11): 1479-1508

[51] Truong V T D, Sato Y, Inoguchi Y. Performance evaluation of a green scheduling algorithm for energy savings in cloud computing [C] //Proc of the 2010 IEEE Int Symp Parallel & Distributed Processing. Atlanta: IEEE, 2010: 1-8

[52] Song Ying, Sun Yuzhong, Shi Weisong. A two-tiered on-demand resource allocation mechanism for VM-based data centers [J]. IEEE Trans on Services Computing, 2013, 6(1): 116-129

[53] Valentini G L, Khan S U, Bouvry P. Energy-Efficient Resource Utilization in Cloud Computing [J]. Large Scale Network-Centric Distributed Systems, 2013: 377-408

[54] 谭一鸣, 曾国荪, 王伟. 随机任务在云计算平台中的能耗的优化管理方法[J]. 软件学报, 2012, 23(2): 266-278

[55] Liu Ning, Dong Ziqian, Rojas-Cessa R. Task Scheduling and Server Provisioning for Energy-Efficient Cloud-Computing Data Centers [C] //Proc of the 2013 IEEE 33rd Int Conf on Distributed Computing Systems Workshops. Philadelphia: IEEE, 2013: 226-231

[56] 李新, 贾智平, 鞠雷等. 一种面向同构集群系统的并行任务节能调度优化方法[J]. 计算机学报, 2012, 35(3): 591-602

[57] Xu Xiaolong, Wu Jiaxing, Yang Geng, et al. Low-power task scheduling algorithm for large-scale cloud data centers [J]. Systems Engineering and Electronics, 2013, 24(5): 870-878

[58] Lee Y C, Zomaya A Y. Energy efficient utilization of resources in cloud computing systems [J]. The Journal of Supercomputing, 2012, 60(2): 268-280

[59] 赵建峰. 基于遗传算法和蚁群算法的节能调度研究[D]. 济南: 山东大学, 2013

[60] Wang Lizhe, Khan S U, Chen Dan, et al. Energy-aware parallel task scheduling in a cluster [J]. Future Generation Computer Systems, 2013, 29(7): 1661-1670

[61] Thanavanich T, Uthayopas P. Efficient energy aware task scheduling for parallel workflow tasks on hybrids cloud environment [C] //Proc of the 2013 IEEE Int Conf on Computer Science and Engineering. Nakorn Pathom: IEEE, 2013: 37-42

[62] 李建敦, 彭俊杰, 张武. 云存储中一种基于布局的虚拟磁盘节能调度方法[J]. 电子学报, 2012, 40(11):2247-2254

[63] Fox A, Griffith R, Joseph A, et al. Above the clouds: A Berkeley view of cloud computing [R]. California: University of California at Berkeley, 2009: 1-23

[64] Ye L, Lu G, Kumar S, et al. Energy-efficient storage in virtual machine environments [C] //Proc of the 2010 ACM SIGPAN/SIGOPS Int Conf on Virtual Execution Environments. New York: ACM, 2010:75-84

[65] 郑湃, 崔立真, 王海洋等. 云计算环境下面向数据密集型应用的数据布局策略与方法[J]. 计算机学报,

2010, 33(8): 1472-1480

[66] 王聪，王翠荣，王兴伟等. 面向云计算的数据中心网络体系结构设计[J]. 计算机研究与发展，2012, 49(2): 286-293

[67] 丁泽柳，郭得科，申建伟等. 面向云计算的数据中心网络拓扑研究[J]. 国防科技大学学报，2011, 33(6):1-6

[68] 刘晓茜，杨寿保，郭良敏等. 雪花结构:一种新型数据中心网络结构[J]. 计算机学报，2011，34(1): 76-85

[69] Liu Shuo, Quan Gang, Ren Shangping. On-line preemptive scheduling of real-time services with profit and penalty [C] //Proc of the 2011 IEEE Southeastcon. Nashville: IEEE, 2011: 287-292

[70] Moore J, Chase J, Ranganathan P, Sharma R. Making scheduling cool: Temperature-aware workload placement in data centers [C] //USENIX Annual Technical Conference. Anaheim: USENIX, 2005: 61-75

[71] Verma A, Ahuja P, Neogi A. Power-aware dynamic placement of HPC applications [C] //Proc of the 22nd Annual Int Conf on Supercomputing. Austin: ACM, 2008: 175-184

[72] Durbin J, Koopman S. Time series analysis by state space methods [M]. Oxford: Oxford University Press, 2012:49-51

[73] Khan A, Yan Xifeng, Tao Shu, et al. Workload characterization and prediction in the cloud: A multiple time series approach [C] // Proc of the 2012 IEEE Symp on Network Operations and Management. Hawaii: IEEE, 2012: 1287-1294

[74] 魏亮，黄韬，陈建亚，等. 基于工作负载预测的虚拟机整合算法[J]. 电子与信息学报, 2013, 35: 6

[75] Brown R, Meyer R. The fundamental theorem of exponential smoothing [J]. Journal of Operations Research, 1961, 9(5): 673−685

[76] Microsoft. The deployment and migration for virtual machine [EB/OL]. (2009-8-1)[2013-07-08]. http://technet.microsoft.com/zh-cn/library/cc956131.aspx

[77] Lin Fengtse, Kao Chengyan, Hsu Chingchi. Applying the genetic approach to simulated annealing in solving some NP-hard problems [J]. Systems, Man and Cybernetics, IEEE Transactions on, 1993, 23(6): 1752-1767

[78] Intel. xeon-e5-brief.pdf [EB/OL]. (2013-7-10)[2013-07-20]. http://www.intel.eu/content/dam/www/public/us/en/documents/product-briefs/xeon-e5-brief.pdf

[79] LCG grid, LCG-2005-1.swf [EB/OL]. (2007-1-24)[2013-07-08]. http://lcg.web.cern.ch/LCG/

[80] A Yuan. LoadAverage. [EB/OL]. (2012-2-10)[2013-07-08]. http://edu.21cn.com/java/g_189_786430-1.htm

[81] Walker R. Examining load average [J]. Linux Journal, 2006, 2006(152): 1-5

[82] Qin Xiao, Jiang Hong. A dynamic and reliability-driven scheduling algorithm for parallel real-time jobs executing on heterogeneous clusters [J].Journal of Parallel and Distributed Computing, 2005, 65(8): 885-900

[83] Zhu Xiaomin, He Chuan, Li Kenli, et al. Adaptive energy-efficient scheduling for real-time tasks on DVS-enabled heterogeneous clusters [J]. Journal of parallel and distributed computing, 2012, 72(6): 751-763

[84] 吴毅华. 云计算环境下的调度策略研究[D]. 上海: 上海交通大学, 2011

[85] Ganglia Monitoring System [EB/OL]. (2012-7-20)[2013-7-08]. http://ganglia.sourceforge.net/

[86] Heger D. Hadoop performance tuning-A pragmatic & iterative approach [J]. CMG Journal, 2013: 1-16

[87] Schatz M. CloudBurst: highly sensitive read mapping with MapReduce [J]. Bioinformatics, 2009, 25(11): 1363-1369

[88] Gardner R, Mckee B, Watson B, et al. Migration of a virtual machine in response to regional environment effects: U.S., 8341626 [P]. 2012-12-25, http://www.freepatentsonline.com/8341626.html

[89] 邓维，廖小飞. 基于虚拟机的数据中心能耗管理机制[J]. ZTE TECHNOLOGY JOURNAL, 2012: 1-19

附录 1 攻读硕士学位期间撰写的论文

（1）曹玲玲，高效能绿色云计算研究分析，计算机应用与教育论文集，2013.12；

（2）Xu Xiaolong, Cao Lingling, et al. Adaptive Task Scheduling Strategy based on Dynamic Workload Adjustment for Heterogeneous Hadoop，已投稿；

（3）Xu Xiaolong, Cao Lingling, et al. Resource Pre-Allocation Algorithm for Low-Energy Task Scheduling of Cloud Computing，已投稿。

附录 2 攻读硕士学位期间申请的专利

（1）徐小龙，曹玲玲等，一种面向云计算平台任务调度的多级负载评估方法，201210146956.3，2012.05，已公示；

（2）徐小龙，曹玲玲等，一种面向绿色云计算的资源配置方法，201410061305.3，2014.02，已受理。

附录 3 攻读硕士学位期间参加的科研项目

（1）国家自然科学基金项目，基于安全 Agent 的可信云计算与对等计算融合模型及关键技术的研究（61202004）；

（2）江苏省自然科学基金，公共云计算环境中可信虚拟私有云模型 VPC 及其关键技术的研究（BK2011754）。

致谢

论文的最后，感谢导师徐小龙副教授的细心指导。在徐老师的耐心指导下，克服了论文中的诸多难点，使自己在云计算领域的领悟更深。在读研期间，徐老师的细心指导和帮助使我受益良多。特别在科研和论文的写作方面：从研究方向的确立和资料收集，到论文的选题和撰写，徐老师都给予了我很大的帮助。导师定期开展的学术研讨会更是使自己获益良多，不仅开拓了自己的学术眼界，而且提高了思维能力。这些收获必将使我受益终生。在此，向徐老师表示崇高的敬意和深深的谢意。

感谢帮助过我的老师以及师门同学们，是你们的热心帮助启发了我，使我不断克服论文撰写过程中遇到的种种困难。在此，向你们表示由衷的感谢。

感谢我的父母和家人，是你们时刻激励着我，在你们的鼓舞和支持下我才顺利完成了学业。

最后，感谢评审老师和答辩委员会的各位专家，感谢你们在百忙之中抽出时间审阅我的论文，感谢你们的宝贵意见和指导。