浙江工业大学

硕士学位论文

论文题目：云计算环境下面向调度优化时效的预测方法研究与实现

作者姓名 _____

指导教师 _____

学科专业 计算机技术

所在学院 计算机科学与技术学院

提交日期 2018 年 4 月 5 日

浙江工业大学硕士学位论文

云计算环境下面向调度优化时效的预测方法研究与实现

作者姓名：

指导教师：

浙江工业大学计算机科学与技术学院

2018 年 4 月

Dissertation Submitted to Zhejiang University of Technology for the Degree of Master

RESEARCH AND IMPLEMENTATION OF PREDICTION METHOD FOR SCHEDULING OPTIMIZATION TIMELINESS UNDER THE CLOUD COMPUTING ENVIRONMENT

Candidate:

Advisor:

College of Computer Science and Technology
Zhejiang University of Technology
Apr 2018

浙江工业大学

学位论文原创性声明

本人郑重声明：所提交的学位论文是本人在导师的指导下，独立进行研究工作所取得的研究成果。除文中已经加以标注引用的内容外，本论文不包含其他个人或集体已经发表或撰写过的研究成果，也不含为获得浙江工业大学或其它教育机构的学位证书而使用过的材料。对本文的研究作出重要贡献的个人和集体，均已在文中以明确方式标明。本人承担本声明的法律责任。

作者签名：　　日期：　　年　　月　　日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权浙江工业大学可以将本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和汇编本学位论文。

本学位论文属于

1、保密□，在_____年解密后适用本授权书。

2、不保密□。

（请在以上相应方框内打“√”）

作者签名：　　日期：　　年　　月　　日

导师签名：　　日期：　　年　　月　　日

云计算环境下面向调度优化时效的预测方法研究与实现

摘　　要

云计算的任务调度和资源分配一直是当前云计算研究的热点。随着大数据和物联网的发展，如何在最短时间内寻找最优结果是云计算调度分配的核心问题。当前各种改进算法仍然是在算法复杂度和性能之间寻找平衡，并不能从根本上解决该矛盾。针对上述问题，利用深度学习模型训练群组优化调度后的数据来实现对调度结果的直接预测是一条可行的路径。但是由于深度学习复杂的层次结构和数量众多的神经元节点，导致其训练时间比传统的机器学习更长，如何提高训练速度优化深度学习模型是须解决的现实问题。论文主要从两个方面研究，一是从研究优化调度算法的角度以获取更精确的训练样本，另一个方面是从研究改进深度学习模型的角度以提高训练速度。具体工作如下：

(1) 针对云计算任务调度问题，建立了基于任务最短执行时间的调度模型。

(2) 针对调度优化效果问题，研究了两种优化算法。在鸟群算法的基础上结合差分进化算法并根据鸟群中个体飞入飞出群体的行为，提出改进型鸟群算法(ISBSA)。在新蝙蝠群算法的基础上加入动态因子，二阶震荡以及差分进化算法等三种改进方式。提出改进型新蝙蝠群算法(ONBA)，并使用这两种改进的算法求解本文提出的多目标问题。同时对两种改进算法的求解结果进行对比分析给出各自的使用范围。

(3) 针对调度优化效率问题，提出改进的深度学习模型(IDBN)，在传统的 DBN 网络前向训练 RBM 阶段和反向微调阶段引入自适应学习率，并使用上面的学习率改进算法对模型的训练次数进行控制以达到加快训练速度的目的。紧接着对上面的 IDBN 训练方法进行实验仿真以验证该方法确实能够加快模型的训练速度。

(4) 最后，论文对前面提出的改进的鸟群算法和改进的新蝙蝠群算法进行多目标任务调度模型的求解以获取调度数据集，并使用数据集训练 IDBN 模型，之后使用训练完成的 IDBN 模型对任务的调度结果进行预测并和传统的调度方法在时间上进行对比，以验证本文提出的方法具有实际应用价值。

关键词： 云计算，鸟群算法，新蝙蝠算法， DBN 模型，预测调度

RESEARCH AND IMPLEMENTATION OF PREDICTION METHOD FOR SCHEDULING OPTIMIZATION TIMELINESS UNDER THE CLOUD COMPUTING ENVIRONMENT

ABSTRACT

Task scheduling and resource allocation of cloud computing have always been a hot issue in the current research of cloud computing. With the development of large data and Internet of things, how to find the best results in the shortest time is the core problem of the cloud computing scheduling. The current various improved algorithms still seek to balance the algorithm complexity and performance, however it can't fundamentally resolve this contradiction. About this problem, using the deep learning model to train the data of the swarm optimal scheduling, then using it to predict the scheduling result is a feasible path. But deep learning is due to its complex hierarchical structure and a large number of neuron nodes, the training time is longer than the traditional machine learning, how to improve training speed and optimize deep learning model are practical problem that must be solved. Thesis mainly research from two aspects: One is to optimize the scheduling algorithm to obtain more accurate training samples; Another is to improve the training speed. This paper mainly finished the following several aspects of the research:

(1)For cloud computing task scheduling problem, this paper gives the mathematical expressions and scheduling model based on the shortest execution time of the task is proposed.

(2)For multi-objective optimization problems, based on the traditional bird swarm algorithm, introducing Differential Evolution Algorithm and according to the behavior of individual birds flying into the fly-out group, we improved bird swarm algorithm (ISBSA). Based on the traditional new bat colony algorithm, introducing dynamic factors, second-order oscillation and Differential Evolution Algorithm to optimize new bat colony algorithm (ONBA). And use these two improved algorithms to solve the multi-objective problem proposed in this paper. At the same time compare the results of the two improved algorithms.

(3)For problems with deep learning model training, introducing adaptive learning rate to training RBM stage and reverse tuning stage in traditional DBN network, and using the above

learning rate improvement algorithm to control the training times of the model to speed up the training of the model. finally we simulate the above improved DBN(IDBN) training method to verify that the method can speed up the training of the model.

(4)Finally, using ISBSA and ONBA solve multi-objective task scheduling models to obtain scheduling data sets, then using data sets to train IDBN models. Using the trained IDBN model to predict the scheduling results of the task and compare with the traditional scheduling method in time to verify this paper proposed method has practical application value.

Key Words: cloud compute, ISBSA, ONBA, IDBN, Predict scheduling

目 录

第1章 绪 论

1.1 研究背景与意义

近年来随着互联网技术的创新和发展以及物联网、大数据、人工智能等技术的掀起，网络上的数据量呈现爆发式增长，尤其是大数据，人工智能领域动辄需要处理 TB 甚至 PB 级的数据量，传统的处理方式已经远远满足不了人们对这些数据处理能力的需要[1]。因此云计算作为近十年前掀起的新型计算模型不仅没有衰落反而受到了更大的关注和发展，今天的基于云计算的产品如：云盘，云相册等已经被人们广泛使用。

云计算作为目前最流行的分布式计算模式，它能够按照用户的需求提供给用户不同的服务。云计算的核心技术是将互联网中海量的、可扩展的、弹性的 IT 资源进行封装[2]并以虚拟化[3]的方式提供给用户使用。用户在使用的过程中无需关心内部的实现细节，只需专注自身的服务需求并按照使用的服务支付相应的费用。因此云平台供应商按照客户的需求提供资源，并从中收取费用，当前谷歌、微软、亚马逊、阿里巴巴等大型企业纷纷向外界提供云计算服务。云计算平台从实际应用角度来看主要分为三类[4]:软件及服务(SaaS)、平台即服务(PaaS)、基础设施即服务(IaaS)。尽管它们提供不同的服务满足不同的应用，但是它们始终都避不开调度问题。

云计算分为三个主体：云计算供应商，云计算系统以及用户，对于云计算用户而言，由于用户在使用云计算系统提供的服务时是需要付出相应的费用的，因此当用户提交任务到云计算系统后如何能在最短的时间内将任务处理完，成为用户选择云计算服务商的重要考虑因素之一，特别是当前大数据和人工智能的掀起，这一类应用对于云计算处理任务的时间有着更高的要求，如果云计算不能在最短的时间内将任务处理完并及时给出相应的结果将严重的影响用户体验。对于云计算供应商而言能够在最短的时间内处理完用户的任务除了能够给用户更好的体验之外，在相同的时间段内则能够服务更多的用户从而获取更多的利润。因此合理的调度策略能够保证在给用户最好体验的同时也给云计算供应商带来最大化的利润，对于云计算系统而言合理的调度策略能够保持自身的稳定性，延长物理硬件的使用寿命，优秀的调度算法能有高效的利用自身的资源，维持任务负载的均衡减少不必要的调度开销。

据 Google2015 年提供的一份关于自家 Borg[5]调查显示，工作在 Borg 上的任务主要分

为两种：第一种是应该“永远”运行下去的长期的服务。这种服务主要用于终端产品如Gmail、网页搜索等；第二种是批处理任务，需要几秒到几天来完成。在长期的服务中给任务分配 70%的 CPU 资源实际用了 60%；分配了 55%的内存资源然而实际用了 85%。据Hyewon[6]等人研究云计算中资源的平均利用率不到一半。由上可知一方面是资源在调度时无法合理的分配另一方面却是资源的利用率低。因此任务调度问题是云计算急需解决的问题。

云计算任务的分配对于云计算系统的运行效率，能量消耗，使用寿命，经济收益，用户体验至关重要。特别是当前云计算服务器集群规模庞大，资源异构多样且动态多变，云计算服务的用户群体广泛，用户提交的任务多种多样，如何高效的调度和执行用户提交的任务对于减小用户任务执行的时间跨度异常重要，然而现如今大部分的调度方法都侧重于如何调度任务使得任务的执行时间最短，负载最均衡等方面，很少有人研究调度算法本身运行时所需的时间，用户从提交任务到获取结果这一时间跨度内，在云计算中会经历两个阶段：第一个阶段就是云计算系统接收到任务后会根据任务的需求和云计算环境中的资源使用情况对任务进行分配，如果任务过大可能还会对任务进行划分以获取细粒度的任务再进行分配，这一过程中主要执行调度算法，各种以诸如负载均衡，功耗最小等目标的方法在这一个阶段进行实现；第二个阶段就是按照调度算法执行后的任务分配结果将任务分配到对应的虚拟机上进行执行。因此任务的执行时间主要由调度算法的运行时间和任务在虚拟机上的运行时间共同决定。因此要减小任务的执行时间提高时效就是要减小调度算法的运行时间和任务在虚拟机上的执行时间。在此背景下如何建立任务的调度模型以及如何减小调度算法本身的执行时间，具有重要的理论意义和研究价值。当前各种改进算法仍然是在算法复杂度和性能之间寻找平衡，并不能从根本上解决该矛盾。针对上述问题，利用深度学习模型训练群组优化调度后的数据来实现对调度结果的直接预测是一条可行的路径。但是由于传统的深度学习复杂的层次结构和数量众多的神经元节点，导致其训练时间比传统的机器学习更长，如何提高训练速度，优化深度学习模型的训练时间是须解决的现实问题。

1.2 国内外研究现状

1.2.1 云计算国内外研究现状

自从 Google 提出“云计算”概念以来，特别是在各高校以及 IT 企业的推动下，云计算得到来迅猛的发展，国外以 Google、IBM、微软等为代表的公司，国内以阿里，百度，

腾讯等为代表的公司纷纷推出云计算相关的商业应用平台，云计算已经进入了百家争鸣的时代。如：Google 将 GFS、BigTable、MapReduce 等技术进行整合推出了 Google App Engine 等相关的云计算服务[7-9]；Amazon 则推出了基于 EC2、SDB、S3 等技术的云计算平台 Amazon Web Sevice (AWS)；IBM 则推出“蓝云”计算平台为用户实现即买即用的云计算平台；微软则推出 Windows Azure platform 云计算服务平台，并为用户提供了公有云、合作伙伴运营、私有云等三种不同的运营模式。云计算在国外迅猛发展的同时，国内也得到了快速的发展。阿里云则通过推动云合计划和云服务商共建生态系统，为企业、政府等用户提供云服务；腾讯云则通过发布“云+计划”来吸引云计算公司进行合作。浪潮发布“云腾计划”吸引合作企业。

云计算在商业方面取得成功的同时，学术界对云计算的研究也进入了高潮阶段，云计算中最核心的技术之一就是调度算法。因为云计算的运行模式决定了它必须保障每个用户的任务按其特定的需求进行处理，良好的云计算任务调度方法不仅可以提高用户任务的处理速度，减少任务执行开销，并且能够给云计算系统提供更理想的任务和虚拟机之间的映射方法，使资源得到合理的利用。因此，对云计算任务调度的研究十分重要。

云计算调度的核心问题之一就是如何调度分配资源和任务使得整个云系统的性能达到最大化同时保持负载均衡，针对这一问题许多大公司如 VMware，HP，IBM 等都做了研究并且提出了方案。VMware 通过研究资源虚拟化技术以及虚拟机的迁移从而提高负载均衡[10]，HP 通过自动放置虚拟机和动态迁移优化资源配置实现负载均衡[11]，IBM 为了满足用户的需求则采用性能优先的方案[12]。Fang Y, Wang F， Ge J 针对基于平衡云资源主机的负载度提出了一种新的任务调度算法[13] ，Juhnke E, Dornemann T, Bock D 等研究了在云工作流调度的瓶颈之一——数据传输，提出了一种新的调度算法，该算法在考虑两个主要的目标因素时间和成本的同时，还着重考虑了资源之间的数据传输对调度算法带来的影响[14]。Frincu M E, Craciun C 提出一种多目标优化算法，在高可用性和成本受限的目标要求下，同时考虑了资源负载和容错性等因素[15]。Oliveira[16]等人采用了自适应性调度算法解决云环境下工作流并行资源调度的问题。Pandey S[17]等人采用的云工作流调度算法既考虑了任务的执行时间开销，也考虑了任务之间数据传输的时间开销。Wu Q[18]提出了一种基于减少任务成本开销的调度算法。同时，依靠对供应商可信度、声誉等非性能特性方面的评估而进行资源调度的研究也越来越多。Mohammadi[19]基于可信度将任务分配到信誉度较好的供应商。Chen W[20]解决了基于用户自定义 Qos 要求的调度问题。Li W[21]，Spitz S[22]则将信任度作为任务调度的主要参考因素。李建峰[23]等人为了减小作业的平均完成时

间，提出了双适应度遗传算法。汤小春[24]等人提出了元区间的分配算法从而提高了云计算中资源的利用率；华夏渝[25]等人提出基于蚁群优化的资源分配算法，通过分析云计算的特点获取资源分配以提高服务质量减小响应时间。罗南超[26]等人提出基于量子优化的多服务器联合调度实现负载均衡。而 Ebadifard[27]等人则提出了一种黑洞的调度算法以平衡用户的需求和供应商的利益。

云计算任务调度实质上是对多目标优化问题的求解。求解任务调度问题的方法主要分为两大类：一类是传统的任务调度方法，如何晓珊等人提出 Min-Min[28]方法中总是优先调度短任务，用最快的速度分配调度队列中的任务用虚拟机上执行从而缩短任务的完成时间。Etminani 等人提出的 Max-Min[29]方法中最小化由于需要长执行时间的任务而导致的部分资源负载过重而部分资源空闲的极度负载不均衡的后果。杨武军等提出改进贪心算法[30]中在分配任务前感知数据，找出它所需要的数据所在的节点上，然后将任务部署到相应的计算节点上去。李靖[31]等人提出基于对策论的调度方法利用贝叶斯纳什均衡保证服务商对于组合任务的公平竞争，从而达到最小化执行时间。刘美林[32]等人提出博弈论的任务调度策略，利用博弈的效益函数计算节点的稳定状态从而提高负载均衡。第二类方法是启发式的搜索算法如：粒子群优化算法[33-34]通过模拟鸟类觅食来优化云计算资源调度，模拟退火算法[35-36]以固体加热，冷却时内部粒子的状态来优化资源调度，遗传算法[37-38]通过模拟自然进化过程搜索最优解，烟花算法[39]则是比较新的一种算法，主要基于烟花在夜空中爆炸产生火花优化调度。布谷鸟算法[40]则是通过模拟布谷鸟寻找适合产卵的鸟窝位置来求解云计算中的资源优化问题。

1.2.2 深度学习

深度学习自 2006 年以来获得了迅猛的发展，深度学习因其模仿人脑特征建立的多层次神经网络，通过组合低层特征形成更抽象的高层表示属性类别或特征，以发现数据的分布式特征表示，因此在短短的十年间不仅在图像识别[41-42]，语音识别[43-44]等方面突破传统的机器学习成果，甚至在预测药物分子活性[45]，预测 DNA 突变[46]，重建大脑回路[47]，分析粒子加速器[48]等传统机器学习难以企及的领域也获得了重大的成果。因此国内外许多大型企业纷纷投入研究，如：Google 推出的 Google Assistant，AlphaGo，微软的同声传译系统，百度的度秘，科大讯飞的讯飞输入法等都是基于深度学习。

虽然深度学在在语言识别，图像识别等领域表现优异，在商业上更是引起了人们的极大热情，在学术上掀起了人们的研究热潮，然而深度学习模型训练过程中如何确定模型的

层数，每一个层的神经元数，学习率以及模型训练时间等问题仍然没有一个公认的解决方案。通过研究，人们提出了许多的优化方法如 Ioffe[49]等人通过优化内部变量之间转化加速模型训练。Dean[50]等人将模型训练分不到成千上万的 CPU 上训练以加快训练速度。Duchi[51]，Senior[52]等人使用自适应学习率加快模型训练，刘凯[53]等人提出一种改进卷积玻尔兹曼机建立卷积 DBN 网络在处理较大尺度图像上缩短训练时间，赵志勇[54]等人将极限学习机和 DBN 结合以避免模型在训练时对每一个受限 RBM 的权重和偏执进行调整从而节约模型的训练时间，杨春德[55]等人在基于自适应深度置信网络的图像分类方法中将步长引入深度学习模型的训练中使得在相同层数的情况下模型的训练结果更优。Chen[56]提出在训练 RBM 时使用模糊参数构建模糊的 RBM 增加抗噪声能力，提高训练精度。Syulistyo[57]使用 PSO 对 CNN 的输出向量进行优化以减小 CNN 的训练时间。这些方法在图像分类，识别以及语音识别方面表现得更好。

1.3 存在的问题与拟解决方法

正如上面所说过的那样尽管对于云计算的调度算法国内外已经有大量的专家学者进行了研究，然而这些算法都集中于任务执行的时间，负载均衡，能量功耗最小等方面，对于调度算法本身运行时所需的时间和成本这一方面的研究还很少。当前各种改进算法仍然是在算法复杂度和性能之间寻找平衡，并不能从根本上解决该矛盾，随着深度学习的掀起，近年来虽然一些学者将深度学习运用于云计算领域以期望改进云计算的性能[58-61]，但是我们也可看到这些方法仅仅只是用来对 CPU 的使用率、CPU 的负载、虚拟机的时间跨、内存的使用等方面做一些精确的预测，而没有考虑到将预测方法用于任务的调度中，除此之外还有如何获取训练模型的数据集等问题，模型的预测结果和训练模型的数据集密切相关，训练集中任务的调度数据如果是关于任务最短执行时间的那么预测的任务调度数据也就是关于任务最短执行时间的，因此如何建立调度模型以获得任务最短执行时间训练数据集是关键问题。最终深度学习由于其复杂的模型结果和需要大量的训练数据，模型在训练时间本省就需要消耗一定的时间和资源，传统的深度学习由于其复杂的层次结构和数量众多的神经元节点，导致其训练时间比传统的机器学习更长，尽管训练模型所需的时间相对于整个云计算系统几年甚至十几年的服务时间来说相对短暂，但是提高深度学习模型的训练速度，缩短模型的训练时间能够更好地改善整个云计算系统的时效性。

针对上面的问题本文主要从两个方面研究，一是从优化调度算法角度出发以获取更精确的训练样本，深度学习模型的预测结果和训练数据集密切相关，要使深度学习模型的预

测结果是执行时间最短的任务分配方案，那么训练的数据集就应该是任务执行时间最短的调度数据集，因此本文建立一个任务最短时间执行模型，由于该模型是一个多目标优化模型，因此在求解该模型时将其转化成多目标优化问题并使用启发式算法进行求解，任务通过该模型后获得的调度方案即为任务的最短执行时间方案。本文研究的另一个方面是从深度学习模型改进角度出发以提高训练速度。深度学习在训练时主要分为前向无监督的训练和反向的调整阶段，无论是在前向的训练阶段还是反向的调整阶段，学习率和训练速度都密切相关合适的学习率既能够加快模型的训练速度又能够保证模型的训练精度。因此本文提出一中自适应的学习率改进算法以动态的调整学习率以使得模型加快训练的同时保证精度。针对整个云计算调度的时效问题本文提出使用经过最短任务执行时间数据集训练完成的改进深度学习模型对到达任务的调度方案直接进行预测以避免使用上述的启发式算法从而减小任务的调度时间和执行时间从而提高云计算调度的时效性。

1.4 本文的研究内容

针对上述的问题以及所提到的解决方法本文主要做了以下几方面的研究：

（1）针对云计算任务调度问题，建立了基于任务最短执行时间的调度模型。

（2）针对调度优化效果问题，研究了两种优化算法。在鸟群算法的基础上结合差分进化算法并根据鸟群中个体飞入飞出群体的行为，提出改进的鸟群算法(ISBSA)。在新蝙蝠群算法的基础上加入动态因子，二阶震荡以及差分进化算法等三种改进方式。提出改进的新蝙蝠群算法(ONBA)，并使用这两种改进的算法求解本文提出的多目标问题。同时对两种改进算法的求解结果进行对比分析给出各自的使用范围。

（3）针对调度优化效率问题，提出改进的深度学习模型，在传统的 DBN 网络前向训练 RBM 阶段和反向微调阶段引入自适应学习率，并使用上面的学习率改进算法对模型的训练次数进行控制以达到加快训练速度的目的。紧接着对上面改进的 DBN 训练方法进行实验仿真以验证该方法确实能够加快模型的训练速度。

（4）最后，论文对前面提出的改进的鸟群算法和改进的新蝙蝠群算法进行多目标任务调度模型的求解以获取调度数据集，并使用数据集训练改进的 DBN(IDBN)模型，之后使用训练完成的 IDBN 模型对任务的调度结果进行预测并和传统的调度方法在时间上进行对比，以验证本文提出的方法具有实际应用价值。

1.5 论文组织结构

论文的正文部分主要分为六个章节进行阐述，每一章节的具体内容如下：

第 1 章:绪论部分。这一节主要介绍本文研究内容的一些相关的背景以及研究的意义。与此同时介绍了与本文研究内容密切相关的当前国内外的一些研究状况。

第 2 章：介绍了本文研究内容涉及到的知识中的一些关键技术的研究和发展。介绍了云计算和云计算任务调度，深度学习的相关概念，以及在解决云计算任务调度的研究中使用的一些新的调度技术和调度方案。

第 3 章：提出了本文所研究的关于任务最短执行时间模型，并对模型的求解提出了改进的 NBA 算法(ONBA)和改进的鸟群算法(ISBSA)对建立模型进行求解，并给出了算法仿真和对比。

第 4 章:对于深度学习部分提出自适应的学习率改进算法和模型的训练次数控制算法对深度学习模型进行改进，并给出算法仿真和算法对比。

第 5 章：实验仿真部分。基于本文提出的使用深度学习的预测方法对云计算中的任务进行预测，将预测结果和 ONBA，ISBSA，PSO 等算法的求解结果进行对比以分析预测误差。此外将深度学习预测所需的时间和 ONBA，ISBSA，PSO 等算法的求解时间进行对比以验证本文提出的任务调度的时间的优化。

第 6 章：总结和展望。总结本文的主要研究内容和不足，提出下一步的研究方向。

1.6 本章小结

本章首先描述了课题的研究背景和国内外相关研究现状。在这个基础上，提出了当前云计算任务调度的存在的问题以及本文的解决方法和技术路线。之后对本文的研究内容作了简要的介绍。最后分章节对文章的结构内容作了介绍。

第 2 章　相关研究与关键技术

2.1　云计算与任务调度的相关介绍

2.1.1　云计算的定义

自 Google 提出“云计算”概念以来，云计算在业界掀起了新的浪潮。然而对于到底什么是云计算目前仍然没有一个公认的定义。工业和信息化部在 2012 年发布的云计算白皮书中指出，云计算主要利用分布式计算和虚拟资源管理技术将网络中的 ICT 资源集中使之成为共享的资源池，并以动态按需和可度量的方式向用户提供服务[62]。云计算其本质是通过网络将分散的硬件以及相应的软件等资源连接起来并通过虚拟化技术给用户提供的按需付费服务。云计算具有以下特点：

1.规模大：一个云计算系统一般都具有一定的规模。如：Google 有 100 多万台服务器。

2.虚拟化：用户可以在任意的位置，使用任意的终端请求服务而不用关心服务运行在哪个地方。

3.高可靠性：云计算通过使用数据备份容错，节点的可互换等保证服务的高可靠。

4.通用性：云计算中提供的服务可以各种各样的而不是针对特定的应用，云计算的服务是按用户的需求提供的。

5.高可伸缩性：云计算中资源可以动态的改变。

6.按需服务：用户按需购买服务。

7.极其廉价：云计算提供的服务成本相对较低，因此用户可以使用较低的费用就能够完成任务。

2.1.2　云计算的分类和关键技术

云计算按照服务类型大致可以分为以下三个部分：

1.基础设施即服务(IaaS)：通过虚拟化技术给用户提供计算基础设施，包括 CPU、内存、存储、网络等计算资源。由于 IaaS 在提供服务时已经将这些资源进行了封装，因此相当于用户拥有了一台虚拟的计算机，用户能够在上面安装不同的操作系统，运行不同的软件，完成不同的任务。IaaS 通用性广，灵活性高，能够实现资源的高度共享；

2.平台即服务(PaaS)：构建在 IaaS 上，PaaS 的抽象化程度更高，主要将计算环境，开

发环境封装成平台，并将平台提供给用户，用户在其平台上开发自己的服务；

3.软件即服务(SaaS)：服务商将应用软件部署在自己的服务器上面，并根据用户的需求通过网络给用户提供软件服务。SaaS 平台中的用户自主权限较低，但是安全性和可靠性较高。

其服务类型关系图如下：

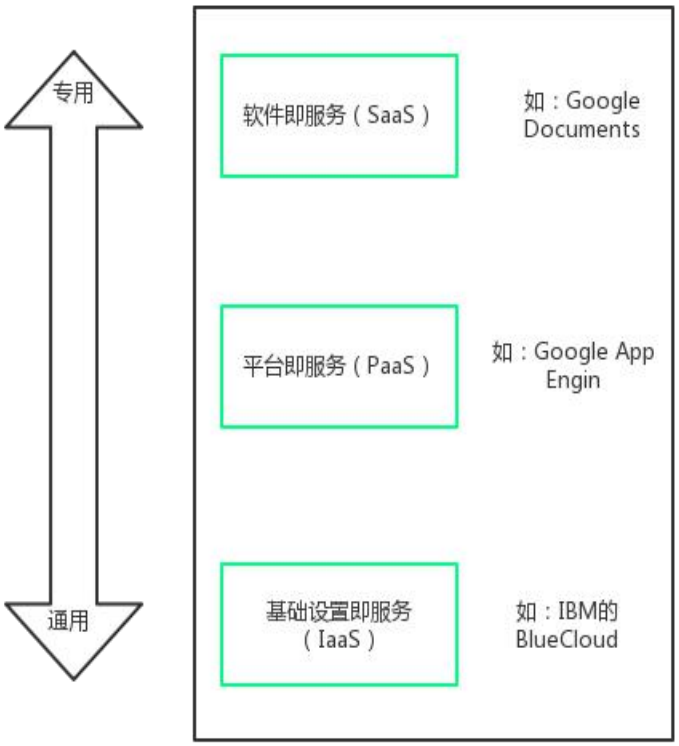

图 1　云计算服务类型

云计算中涉及到的技术有很多除了调度算法之外，虚拟化技术，数据的存储管理，安全管理等技术。

虚拟化：云计算使用存储虚拟化，网络虚拟化，软件虚拟化等虚拟化的方法整合整个系统中的硬件资源、虚拟服务器等其他的软硬件设施。并通过一种高效任务资源调度方法为应用提供基础设施平台，从而满足云计算的需求。

数据的存储管理：为保持高性能、高可靠性的存储性能。云计算采用分布式存储技术，将非必要使用数据存储于大量分布式存储设备中，以减轻客户在设备中需要的存储空间，降低在运行大型应用时对设备的等级需求。

安全管理：由于用户使用云计算时会将数据上传到云端，因此如何保证用户的数据安

全性关系到云计算是否能够得到广泛运用。

2.1.3 云计算任务调度

调度问题一直是很多领域都绕不开的一个问题，尤其是在计算机科学中更是研究的重点[63]。云计算中资源的差别较大，因为资源可能是普通的计算机，服务器或是虚拟机，因此这些资源的计算能力、存储能力和网络带宽等各方面因素有很大差异，又因为云计算由于其用户人群范围广泛，因此用户所提交的任务和对资源的需求也各不相同，如何调度任务到可用的资源上执行成为了云计算领域研究的一个重点问题。从整体上来看云计算中的任务调度本质上是云计算服务平台将用户提交的任务和可用的资源之间的映射过程，它根据云环境中资源、任务之间的状态信息，在一定的约束条件下，将相互独立的用户所提交的任务，映射到适当的虚拟机资源上执行，最后返回处理结果给客户。一个合适的任务调度算法不仅能够高效的分配虚拟机资源还能减小任务的执行时间降低资源耗费量，使云计算性能最佳。云计算任务调度主要有两个目的，(1)使用户方便的将任务提交给资源；(2)通过任务和资源之间的分配方法，增加资源的利用率，满足用户的任务需求。也就是说，通过良好的云计算任务调度算法不仅能够保证云计算环境中虚拟机资源的利用率同时也能够保证用户获得更好的云计算服务质量以达到双赢的局面。

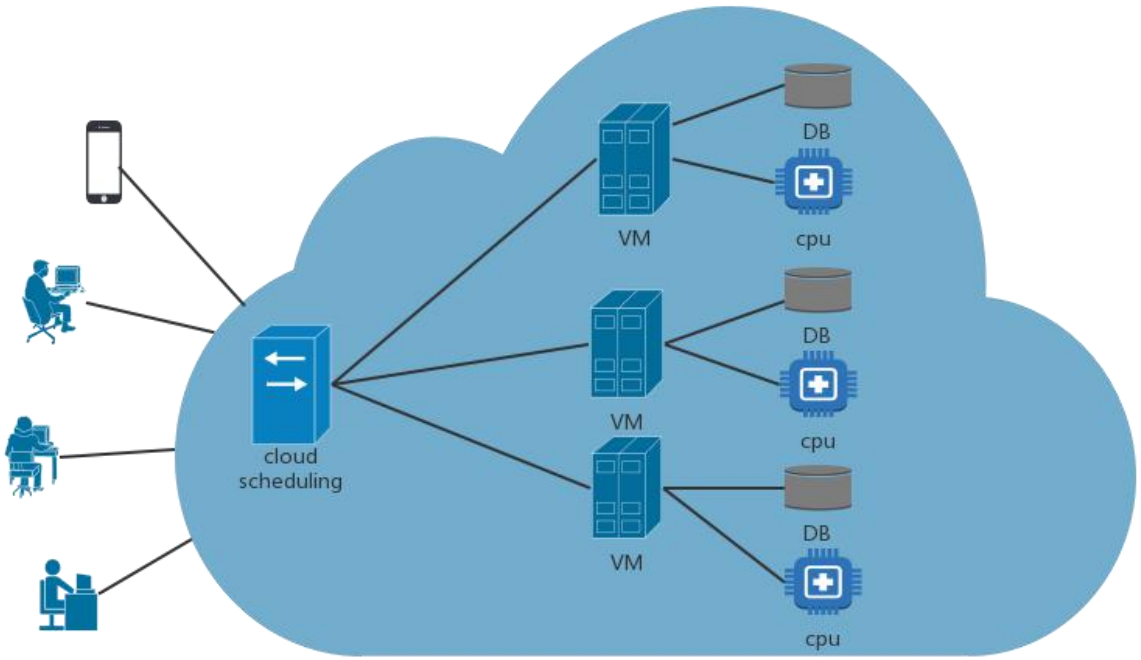

图 2-1　云服务调度体系结构

从上图可以看出，云计算系统中的调度模块不仅连接了用户所提交的任务，而且还连接云服务中的虚拟机资源，同时还要监控整个系统中的资源使用状况。用户在提交任务到云计算系统的同时可能还会提交一些相应的信息和要求等，云计算调度模块则根据这些任务和相应的内容要求调度云计算中的各种资源以满足各个任务的执行需求。

由于云计算环境中的异构、动态、分布和自治等特性，如何调度任务以满足用户的需求是一个极具挑战性的问题。在用户提交的一般性任务中，由于云计算平台面向的用户人群数量庞大用户提交的任务大小不同，用户提交的任务往往不能作为独立的调度单位，因此比较常见的方法是，如果该任务的任务粒度大，那么它便会被逻辑划分为若干个彼此相关的子任务，然后以这些子任务作为调度的基本进行调度和分布式化处理，如果该任务的任务粒度较小，那么它在进行调度和分布式化的转化过程中便不需要进行分割，一个独立的子任务就能够很好地完成它的工作；当所有的子任务执行完成后则整个任务便执行完毕。

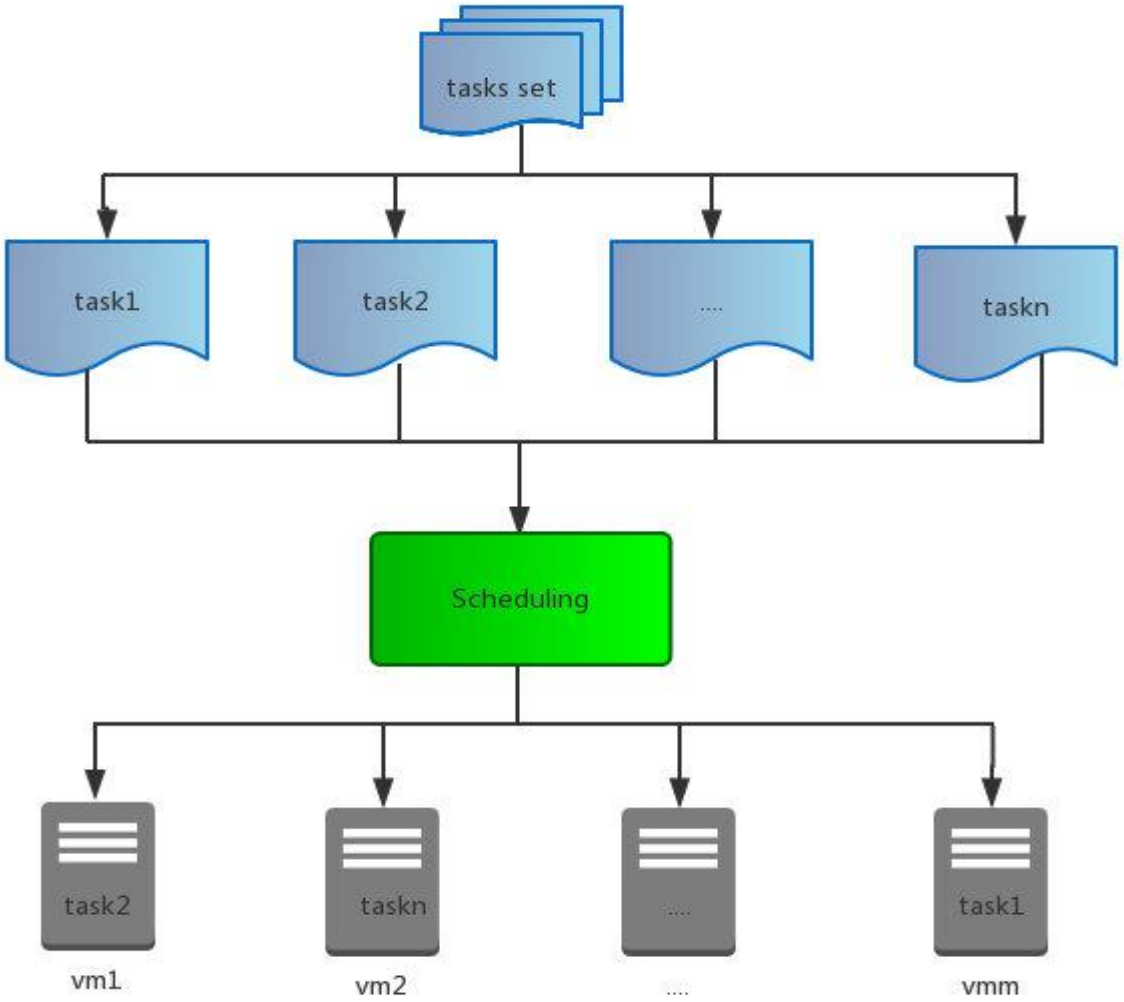

图 2-2 子任务划分与任务调度

如上图所示当用户提交任务到云计算平台后，云平台首先会将用户的任务进行逻辑划分成相互独立的子任务，云计算平台会按照每一个子任务的资源请求，任务所指定的功能以及自身所维护的云计算环境中的资源状况将每一个任务都分配到对应的虚拟机上执行。在任务进行分配时可能存在多台虚拟机同时满足一个任务的需求，也可能存在不同的任务同时分配到同一个虚拟机上运行。因此任务的分配需要按照一定的调度策略达成一定的调度目的进行分配。

2.2 常用的资源调度算法研究

云计算环境下的任务调度实质是将经过逻辑分割成的 n 个子任务分配到 m 个资源上

面，以便高效完成任务，任务调度其实是一个 NP 难问题，因此很多专家学者也在致力于解决云计算调度中的 NP 难问题并提出如下几种算法：

2.2.1 传统的资源调度算法

1.Min-Min 算法：Min-Min 算法的思想主要是将最短的任务分配给执行速度最快的资源上面。主要方法是获取任务在每一个虚拟机上执行的最早完成时间并将任务分配给具有最小最早完成时间的那台机器上。具体算法步骤如下：

(1)对于任务集合 T={t1,t2,…,tn}中的每一个任务 ti 分别计算出该任务在集群中的 m 台机器上的最早执行完成时间，这样就能得到一个时间矩阵 MT。

(2)利用上一步的 MT 矩阵，找到待分配的任务中最早完成的那个任务 p 和对应的分配的机器 k 并将该任务真正的分配到对应的机器上。

(3)将上面的任务 p 从任务集合中删除并跟新时间矩阵 MT。

(4)当任务矩阵不为空时不断的重复上面的步骤(1)-(3)中的操作。

该算法实时起来较简单，时间复杂度也不高，但是该算法在执行的过程中会将大量的任务分配给执行速度最快的机器上，不能充分利用性能较低的机器，这样容易造成负载不均衡。

2.Max-Min 算法：Max-Min 算法和上面的 Min-Min 算法类似都要计算每一个任务到每台机器上执行的最早完成时间，和 Min-Min 算法不同的是 Max-Min 算法是找出最大的最早完成时间，从而将任务分配到对应的机器上面。任务一旦分配就将该任务从任务集合中删除，并不断重复上面的操作直到任务集为空。该算法是为了优化由于需要长时间执行的任务在执行时导致的负载不均衡。由于 Max-Min 算法和 Min-Min 算法类似，虽然对于长任务执行时的负载进行了优化，但是依旧存在负载失衡的问题。

3.Sufferage 算法[64]：Sufferage 算法主要解决任务在调度过程中出现的竞争问题，sufferage 算法的基本思想为：计算每个任务在节点上的最短完成时间和次短完成时间的差值的绝对值，这个值称为损失值也称为 Sufferage 值，这个值越大就说明如果不将任务调度到完成时间最小的资源上，则会造成较大的损失，因此总是优先调度 Sufferage 值最大的任务进行执行，当任务在调度过程中发生抢占或竞争时会比较 Sufferage 值，如果 Sufferage 值小于已经分配的任务则任务继续执行，如果 Sufferage 值大于已经分配的任务的 Sufferage 值，则已经分配的任务会让出资源，并分配具有较大的 Sufferage 值的任务。从上面的过程可以看出 Sufferage 算法本质上是贪心算法，因此在进行任务分配时由于其

总是将任务执行效率最高的节点上，所以会造成负载不均衡。

4.FIFO 算法：其思想为将用户提交的任务按照提交时间插入到一个队列，再从对头不断的取出任务调度到资源上面，如此往复知道队列为空，缺点为对先提交的任务有利，后提交的任务需要等待，无法顾及任务的执行时间和效率并且资源利用率较低。

2.2.2 智能化的调度算法

传统调度算法的优点实现简单快速，缺点是适应性不强，局限性较大，容易导致负载不均衡，资源的利用率较低。因此基于智能的启发式算法很快就用来替代上述的传统调度算法。启发式算法主要有：遗传算法，粒子群算法，查分进化算法，鸟群算法，新蝙蝠群算法等，这些算法已被证明在求解关于调度的 NP 难问题比传统的算法表现更好。

1.遗传算法：遗传算法(GA)是 Holland[65]等人在 1975 年提出后由 DeJong, Goldberg 等人总结的一种全局优化搜索算法，其基本思想是模拟自然界遗传机制和生物进化论而形成的一种过程搜索最优解的算法。遗传算法通过模拟生物在自然环境中发生的繁殖、交叉和基因突变等现象。求解问题时在每次迭代过程中都保留一组候选解形成解种群，并按事先定义的标准从解种群中选取较优的个体，利用遗传算子(选择、交叉和变异)对这些个体进行组合，产生新一代的候选解群，重复此过程，直到满足给定的条件为止。

在遗传算法中问题的解是由个体表示的，一个个体就代表问题的一种解决方法，每一个个体又是由基因组成，因此个体也叫做染色体，多条染色体组成了一个种群。为了模拟自然界中生物的杂交和基因突变等现象，遗传算法在求解过程中会执行选择、交叉、变异等操作。为了实现自然界中的优胜劣汰现象，每一个个体会由适应度函数计算出一个适应度值，这个适应度值就是对个体的好坏进行的一个评价，适应度值高的说明个体更好因此会被留下，相反适应度值低的会被淘汰掉。当个体选择好了后紧接着进行杂交即由这些个体生成新的个体。从而组成新的种群。由于每一个个体的产生都是由适应度较高的父代杂交而来，因此个体朝着较优的方向进化，问题的解也就朝着越来越优的方向发展。其算法过程如下：

(1)将问题进行编码；

(2)初始化迭代次数，种群个数并按照(1)中的编码生成种群；

(3)计算当前种群中每一个个体的适应度值；

(4)按照一定的方法并遵循适应度越大被选中的概率越大的准则，从种群中选出一定的个体作为父代中的个体。

(5)对父代中的个体值进行随机配对，对配对的个体进行交叉以产生新的个体；

(6)对新产生的个体按照一定的概率对其基因进行变异；

(7)重复以上的操作指导迭代完成。

由于遗传算法中的编码可以是二进制编码也可以是实数编码，不同的编码方法求解的适应度值可能不一样，因此算法的准确性不稳定，此外遗传算法的收敛速度慢，易陷入局部最优解。

2.粒子群算法：粒子群算法(PSO)[66]是由 Kenned 和 Eberhart 在 1995 年提出的，该算法主要源于对鸟群觅食行为的研究，该算法和 GA 相似都是通过不断的迭代来求解最优解和 GA 不同的是该算法没有交叉，变异操作只是简单的寻找最优的粒子。在 PSO 算法中每一个粒子代表问题的一个解，每个粒子都有各自初始的速度和位置并随机的分布，为了衡量粒子位置的好坏，每一个粒子都会经过适应度函数计算出一个适应度值，此外每个粒子都记得自己搜索过的历史最好位置和最好适应度值，每个粒子也知道当前整个群体的最好位置，粒子根据自己当前的位置和历史最好位置，当前的速度以及粒子当前的位置和群体的最好位置来不断的调整自己的速度和下一次的位置。其调整公式如下：

$$V_{ij}^{t+1} = V_{ij}^{t} + c_1 r_1 (P_{ij}^{t} - X_{ij}^{t}) + c_2 r_2 (g - X_{ij}^{t}) \tag{2-1}$$

$$X_{ij}^{t+1} = V_{ij}^{t+1} + X_{ij}^{t} \tag{2-2}$$

上述公式中 V_{ij}^{t} 为当前迭代次数为 t 时的粒子 i 的第 j 维的速度， P_{ij}^{t} 表示粒子 i 在第 j 为上的最好位置， X_{ij}^{t} 表示当前粒子的位置， c_1, c_2 表示粒子的学习因子通常为 2， r_1, r_2 位取值范围在(0,1)之间的随机数，g 为群体的最好位置。算法步骤如下：

(1)初始化粒子种群的数量，迭代次数，粒子的维度，搜索范围等参数；

(2)初始化粒子的速度 V，粒子的位置 X，计算每个粒子的适应度值并求解粒子群的群体最佳位置 g 和每个粒子的历史最佳位置 P；

(3)开始迭代，按照公式(2-1)和公式(2-2)更新粒子的速度和位置；

(4)计算每个粒子的适应度值和粒子的历史最好适应度值比较，如果当前适应度值比历史最佳适应度值要好则更新历史最佳适应度值和 P；

(5)将每一个粒子的适应度值和整个种群的最佳适应度值进行比较，如果粒子的适应度值比种群的最佳适应度值要好则更新种群的最佳适应度值，并更行对应的 g；

(6)判断迭代是否结束，如果没有结束的返回步骤(3)重新开始否则转到(7)；

(7)迭代结束输出 g 即为问题的最优解。

粒子群算法虽然参数少，易实现相对于 GA 省去了交叉，变异等步骤，尽管如此 PSO 也存在着一些缺点，如：早期虽然收敛速度快，但是收敛精度较低，易发散甚至会出现无法收敛。达到收敛时由于所有的粒子都相互靠拢这样会导致陷入局部最优解等问题。

3.差分进化算法：差分进化算法(DE)[67]是 Storn 等人在 1997 年提出的，和 GA，PSO 算法一样也是基于群体智能的全局搜索算法。和 GA 算法相似差分进化算法也有选择，交叉，变异等操作，与 GA 算法不同的是差分进化算法在进行变异的时候会随机的选择一定比例的个体的差分信息作为扰动量。差分进化算法的基本思想是：从种群中随机的选取三个不同的个体并将期中的两个个体的相减产生差分向量，将产生的差分向量乘以一个权重因子后与第三个个体相加产生变异个体，然后将变异个体和种群中的其他个体进行交叉以产生新的个体，将新的个体和父代的个体都经由适应度函数计算出适应度值，比较新的个体的适应度值和父代个体的适应度值若新个体的适应度值优于父代个体的适应度值则保留新个体，否则保留父代个体。然后进行下一个的选择，变异，交叉操作。该算法中最重要的就是变异和交叉操作，其公式如下：

$$V_{ij}^{t+1} = X_{p1j}^{t} + \varpi(X_{p2j}^{t} - X_{p3j}^{t}) \tag{2-3}$$

$$u_{ij}^{t+1} = \begin{cases} v_{ij}^{t+1}, rand(0,1) \le cr \quad or \quad j = jr \\ x_{ij}^{t}, rand(0,1) > cr \quad or \quad j \ne jr \end{cases} \tag{2-4}$$

$$x_i^{t+1} = \begin{cases} u_i^{t+1}, f(u_i^{t+1}) > f(x_i^t) \\ x_i^t, otherwise \end{cases} \tag{2-5}$$

公式(2-3)主要是用于个体的变异操作，其中 V_{ij}^{t+1} 表示产生的变异的新个体，$i \ne p1 \ne p2 \ne p3$，ϖ 表示缩放因子通常为一个[0,2]之间的定值，X_{ij}^{t} 表示第 t 代种群中的第 i 个个体所对应的第 j 个维度上的值，有时为了保证变异后的解的有效性必须判断每一个变异后的个体每个维度上的值是否超出边界限制，如果没有超过边界则参与计算适应度值，如果超过边界则直接取边界值代替或者随机生成边界值范围内的值代替。公式(2-4)用于交叉时计算获得子代个体，其中 u_{ij}^{t+1} 表示通过交叉后获取的子代个体，cr 表示发生交叉的概率取值范围为[0,1]，jr 表示个体的维度，进行交叉时会对个体的每一个维度按照概率 cr 决定是否参与交叉。公式(2-5)用于决定子代和父代个体是否要参加下一代的运算，函数 f 表示适应度函数用来计算子代和父代个体的适应度值，如果子代的适应度值优于父代的适应度值则子代参与下一次的迭代父代被淘汰，反之则父代参与下一次的计算子代被

淘汰。算法过程如下：

(1)参数初始化，设置当前进化的代数，设置最大的循环迭代次数，种群大小，缩放因子，交叉概率；

(2)初始化种群中的个体，并计算每个个体的适应度值；

(3)设置 i=1；

(4)随机生成三个整数 p1,p2,p3 并使得 i≠p1≠p2≠p3；

(5)按照公式(2-3)生成变异个体；

(6)按照公式(2-4)生成交叉后的子代个体；

(7)分别计算子代和父代的个体的适应度值并按照公式(2-5)进行选择操作；

(8)令 i=i+1 并转到步骤(4)，直到 i 的值等于种群数；

(9)判断迭代是否结束如果没有结束则转到步骤(3)，否则输出结果。

该算在早期的时候，由于种群中个体的差异较大因此整个扰动量也较大。算法能够在较大范围内搜索，具有较强全局搜索能力；但是算法导论后期的时候由于算法趋向于收敛此时种群中个体的差异很小算法会在个体的附近进行搜索，这使得算法具有较强的局部搜索能力[68]。尽管 DE 算法拥有以上这些优点但是恰恰因为算法后期个体的差异性较小从而使得算法的收敛速度慢、容易陷入局部最优解，其次参数的控制对算法的性能有着重要的影响、算法在搜索时没有用到前面的历史搜索信息从而导致算法有时需要过多的迭代次数才能搜索全局最优解。

2.3 深度学习

深度学习是目前机器学习领域发展最热的一个分支，深度学习也被称为特征学习，深度神经网络等，深度学习的本质是模型了大脑视觉的分层神经网络，从接受视觉信号开始，然后做初步处理提取出物体的边缘信息，接着做抽象处理判定物体的形状，最后再做进一步的抽象并识别物体。整个过程就是神经网络由低层到高层不断地对信号进行抽象处理，最后获得的特征就能很好的表示该信号。深度学习的主要发展经历了两个阶段：

第一个阶段为浅层学习阶段：上个世纪 80 年代由于反向传播算法(BP)的发明开启了人们研究利用基于统计模型的机器学习的大门，人们发现使用 BP 算法训练神经网络能够使其从大量的训练样本中学习统计规律，从而对未知事物做出预测。这个时期的神经网络只有输入层，隐层，输出层。因此也被称为浅层学习模型。90 年代后许多的浅层学习模型相继被提出如：支持向量机(SVM)、Boosting、逻辑回归等模型，这些模型无论是理论

研究还是实际应用都获得了巨大的成功，然而神经网络因为其理论分析难度大，训练方法需要很多的经验和技巧再加上当时计算能力的限制，使得神经网络的发展进展缓慢。浅层学习的缺点在于，当学习的样本数量有限并且计算单元也有限时不能很好的表示较为复杂的函数，因此不能很好的应用在一些复杂的分类问题上面，此外浅层学习模型训练时需要人工来提取样本的特征，比较费时费力。

第二阶段为深度学习阶段：2006 年 Hinton[69]首次提出了深度学习的概念开启了深度学习在学术和商业应用的新浪潮，该论文有两个主要观点：一、多个隐层的人工神经网络具有很好的特征学习能力，通过它学习得到的特征能够很好的刻画数据的本质，因此多隐层的神经网络对于可视化或分类问题具有很好的应用；二、可以通过“逐层初始化”的方法来减小深度神经网络的训练难度。在这篇文章中，逐层初始化是通过无监督学习实现的。基于此 Hinton 提出深度神经网络的训练主要分为两步，一是逐层构建单层神经网络，并每次训练一个单层神经网络；二是、当所有层训练完成后，自上而下进行微调。通常我们认为第一步为自下而上的前向训练以确定每一层之间的连接参；第二部为微调阶段主要使用 BP 算法对已确定的参数进行精细的调整。整个训练过程如下：

一、使用自下而上的非监督学习，先用无标定数据训练第一层，训练时先学习第一层的参数，得到的模型能够学习到数据本身的结构，从而得到比输入更具有表示能力的特征；在学习得到第 n-1 层后，将 n-1 层的输出作为第 n 层的输入，训练第 n 层，由此分别得到各层的参数；

二、自上而下的监督学习，但前向训练完成后在使用带标签的数据去训练模型，使得误差自顶向下传输，以减小误差达到对网络微调的目的。

在短短的十年内，深度学习在语音识别、语义分析、图像分类等众多领域都突破了传统机器学习取得的成果。在语音识别领域，国外的微软、苹果、谷歌和国内的百度、科大讯飞等公司采用各自的深度学习模型将错误率降到 30%以下。

2.4 本章小结

本章首先介绍了云计算领域的相关概念，关键技术以及任务的调度定义并给出了当前常用的任务调度算，对常用的调度算法分析了其优缺点，最后介绍了深度学习的发展以及主要的训练方法。

第 3 章　面向任务最短执行时间的算法优化

3.1　云计算资源调度的目标

云计算任务调度的本质是寻找一种调度策略，基于该策略，在任务与计算资源之间建立起一种合适的映射关系，以实现应用任务在计算资源之间的合理分配与高效调度执行，尽管如此云计算任务调度必须考虑复杂的目标和用户需求，一个良好的调度策略必须尽量的满足以下几个目标：

1.最优跨度。这里所说的最优跨度通常是指时间上的，对于云计算而言最优时间跨度具体为以下两方面：第一，调度时间跨度，指的是系统从开始接受接收第一个子任务并开始按调度算法进行任务调度分配到所有的子任务调度分配结束时所需要的时间。第二，任务执行时间跨度，指的是从第一个任务分配到某台虚拟机上执行开始到所有的任务执行完毕所花费的时间，该跨度越短，调度分配方案越好。

2.负载均衡。在云计算的发展过程中，负载均衡一直是衡量一个调度算法优劣的重要标准。不能因为一些资源的性能较优就把大量的任务都调度分配到它上面而部分性能一般的资源却一直闲置，因此，如何将任务调度到虚拟机上执行并且在满足用户的时间等要求的同时保证云计算环境中虚拟机上分配的任务达到负载均衡是云服务供应商需要解决的关键问题之一，因为这关系到数据中心资源是否得到了充分科学的利用并会对运营成本造成一定的影响。

3.服务质量。服务质量和用户对服务产品的满意程度密切相关。云计算供应商应该使用合适的调度方案尽可能的满足用户需求的 QoS 并降低 SLA 违反率。云计算供应商既要根据用户提交的任务在最短的时间内准确的找到可用的服务资源又要在云计算环境发生变化时重新选择资源和重定向配置以此保证云计算服务的廉价性和可靠性。从云计算平台来看，云计算中的调度算法应该能够有效的控制用户所占用的资源同时避免用户之间的相互影响导致任务失败。

4.经济原则。由于云计算的商业性，云计算服务由用户、运营商、提供商等多方人员参与，存在复杂的资源交换、服务利益，不仅要满足用户需求和保证其 QoS 调度，而且要最大化云服务提供商的收益。经济原则成为云计算任务调度的一个重要目标。在市场经济下，使各方互惠互利的经济原则才能使云计算服务发展良好。

3.2 云计算调度模型的分析与建立

通过上面的分析可知云计算调度的目标多种多样除了上面的几种之外还有以能量效率最高[70]，提高云计算环境中的 GPU 利用率[71]为目标。本文是以任务最优跨度为目标建立调度模型，使得任务整体完成时间最小。

云计算中资源是通过虚拟化的方式提供给用户, 资源通常包装在虚拟机中，通常情况下云计算中任一虚拟资源可以用如下公式描述：

$$VM = \{Id, Mips, Mem, Size, Bw, Cpunum\} \tag{3-1}$$

上面的公式中 Id 表示资源的编号，Mips 表示虚拟资源中 CPU 的指令执行速度，Mem 表示内存的大小，Size 代表外存的大小如：磁盘，光盘等，Bw 表示资源的带宽，Cpunum 表示资源中封装的 CPU 的数量。随着技术的发展外存的容量越来越大，越来越廉价，带宽的接入都是百兆的光纤所以影响虚拟资源性能的主要方面在 CPU 和内存上, 这也是本文主要考虑的方面。

用户提交的任务主要分为计算密集型任务和 I/O 型任务，计算密集型任务主要对 CPU 的性能要求较高对磁盘的要求较低，I/O 型的任务一般对 CPU 的性能要求不高但是可能会经常会对磁盘进行读写任务，由于磁盘相对较廉价跟换磁盘的成本相对较低，所以本文主要考虑的是计算密集型任务。其任务可以按照如下的公式进行描述：

$$T = \{Tid, Length, Tmem, Tsize, Tbw\} \tag{3-2}$$

上述公式中 Tid 表示的是任务的编号，Length 表示任务的长度，Tmem 表示任务的内存需求，Tsize 表示任务对磁盘的要求，Tbw 表示任务对网络的需求，因为有的任务不需要网络所以这个属性不是必须的。

本文的云计算调度可以描述为：在云计算环境中将 n 个任务分配到 m 个虚拟资源上执行。任务集合可以表示为：T={T1,T2,…,Tn}，虚拟资源集合表示为：VM={VM1,VM2,⋯,VMn}将任务 T 分配到虚拟资源 VM 上的分配关系矩阵如下：

$$M = \begin{pmatrix} m_{11} & & m_{1n} \\ & & \\ m_{m1} & & m_{mn} \end{pmatrix} \tag{3-3}$$

上式中的 m_{ij} 表示任务 T_j 在资源 VM_i 上的分配关系，其取值只有 0 或 1 两种情况，0 表示该任务不分配到对应的资源上面执行，1 表示对应的任务分配到资源上面执行。为了简化调度过程我们做如下的约束条件：

(1)每一个任务只能调度到一个节点上面执行，每一个节点一次只能同时执行一个任

务；

(2)任务在执行期间不能被中断且任务的执行只需要一个处理器，如果有多个任务要在一个节点上面执行，则按照先来先服务的原则进行执行。

任务的执行时间为 E_{ij}，则其表达式如下：

$$E_{ij} = \begin{cases} \frac{L_i}{Mip_i}, M_{ij} = 1 \\ 0, M_{ij} = 0 \end{cases} \tag{3-4}$$

其中 L_j 为任务 j 的长度， Mip_i 为对应的 CPU 的执行速度。任务的完成时间

$$ET_{ij} = ST_{ij} + E_{ij} \tag{3-5}$$

ST_{ij} 表示当前任务 j 执行的开始时间，当 j 为第一个任务时 $ST_{ij}=0$，当 j 不是第一个任务时 $ST_{ij}=ET_{ij-1}$ 即为上一个任务的结束时间。节点上的任务执行时间为:

$$EVM_i = ST_{iend} + E_{iend} \tag{3-6}$$

其中 end 表示节点上的最后的一个任务。云计算中每一个节点都有一个执行时间，当整个任务集的执行时间就是执行时间最长的那个节点的完成时间所以，整个任务执行时间为：

$$FN = Max(EVM_i) \tag{3-7}$$

上式中 FN 即为任务的总的执行完成时间，FN 越小则表明任务的执行时间越短，相反则表明任务的执行时间越大，因此 FN 的值越小越好。

云计算中除了时间跨度最优外，另一个重要的目标就是负载均衡，负载均衡通常用来衡量云计算中的资源的利用情况，一个好的调度方案应当兼顾最小化任务执行时间和保证负载均衡两方面，这两点也是本文的重要对比指标。云计算中的负载均衡主要分为两个方面；一是虚拟资源的负载均衡也就是每一个节点执行任务的负载情况，另一个就是虚拟资源在物理机器上的负载均衡即每一个物理节点建立的虚拟资源的差异性很小[72]。由于本文中主要研究的是任务在虚拟机上的调度优化问题，因此选取的负载均衡也是任务在虚拟机上的负载均衡问题。

对于负载均衡而言常用的标准有两个：一是统计每台虚拟机上分配的任务数量，其值波动范围越小则说明每台虚拟机分配的任务数量约接近，所以负载也就越均衡，这种衡量标准的优点就是简单，其缺点就是使用范围较小，值适用于虚拟资源性能相近，任务长度相似的情况，如果资源性能差异较大或任务的长度变动较大, 只简单的统计资源分配的任

务数量会引起严重的负载失衡；所以我们使用下面的第二种衡量标准：使用每一个虚拟机执行任务的时间作为标准，如果每台虚拟机执行的任务所用的时间波动范围越小则负载越均衡，本文使用标准差作为主要的负载均衡优劣的衡量标准：

$$LB = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(EVM_i - EVM_{avg})^2} \tag{3-8}$$

负载均衡标注差越小说明云计算任务调度过程中，任务在虚拟机上的负载越均衡。这样以处理整个云计算环境下任务集合所需的运行时间、以及任务在虚拟机集群上的负载均衡为优化多目标的任务调度过程的一般性模型已经建立完成，接下来需要确定该模型下的目标函数。基于多目标优化的云任务调度模型的目标函数如公式如下：

$$\begin{cases} \mathrm{Min}(FN) = Min(Max(EVM_i)) \\ Min(LB) = Min\left(\sqrt{\frac{1}{m}\sum_{i=1}^{m}(EVM_i - EVM_{avg})^2}\right) \end{cases} \tag{3-9}$$

3.3 改进鸟群算法和新蝙蝠群算法

3.3.1 改进的鸟群算法

鸟群算法(BSA)是 Meng 等人[73]于 2015 年提出的新型的智能启发式全局搜索算法，鸟群算法主要是基于鸟群的群体行为和群体中信息交流，通过模仿鸟群的觅食，迁徙，警戒等行为以及这些行为中的信息共享机制和搜索策略而建立的一种新型的搜索优化算法。该算法具有调节参数少，收敛精度高，鲁棒性好等优点，近年来也受到了越来越多的关注。鸟群算法中鸟的觅食，迁徙以及警戒等行为能够被抽象成如下的规则：

规则一：鸟群中的每一只鸟都能选择觅食行为或警戒行为，无论是选择觅食还是保持警戒都是随机决定的；

规则二：在觅食时，每只鸟都能及时记录和更新自身搜索食物的历史最好路径和整个群体中的历史最好路径。在整个群体中这种信息是共享的，每一只鸟都能从群体中获取食物的位置信息以及将自己的信息共享到群体中；

规则三：当个体保持警戒时，每一只鸟都会试图向着群体的中心飞去，但是这种行为会受到群体中其他的鸟的竞争因素干扰，储备量高的鸟更容易飞到鸟群的中心；

规则四：鸟群会定期的迁徙到另一个地点觅食，当鸟群飞到另一个地点时鸟群的个体会分为生产者和乞食者，对于整个群体而言群体中那些食物储备最多的个体将会成为生产者，而群体中那些食物储备最少的个体将会成为乞食者，其他的鸟则随机的成为生产者和

乞食者

规则五：群体中的生产者会积极的寻找食物而乞食者则会随机的跟随一个生产者寻找食物。

假设鸟群的大小为 N 每只鸟在 D 为的空间中飞行，则在时间 t 时个体 i 的位置信息为 X_i^t 。

(1)觅食或保持警戒：鸟是觅食还是保持警戒是随机的进行选择，如果鸟的一个选择概率小于指定的一个概率值则鸟会选择觅食否则鸟保持警戒。

(2)鸟类觅食：当鸟儿选择觅食时的搜索位置变换公式如下：

$$X_{ij}^{t+1} = X_{ij}^t + (gbest_{ij} - X_{xi}^t) * c1 * rand(0,1) + (Pbest - X_{ij}^t) * c2 * rand(0,1) \quad (3\text{-}10)$$

上式中 $j \in [1, 2, \quad D]$， $gbest_{ij}$ 为个体 i 历史最好位置， $Pbest$ 为群体最好位置，$c1, c2$ 为正数表示认知行为和群体社会行为， X_{ij}^{t+1} 为个体 i 下一次的位置。

(3)保持警戒：当鸟保持警戒时按照规则三，他们会向群体的中心移动，但是由于鸟群中的个体之间存在着竞争的关系所以每个个体不是直接的飞往群体中心而是按照下面的公式移动：

$$X_{ij}^{t+1} = X_{ij}^t + A_1(mean_j - X_{ij}^t) * rand(0,1) + A_2(gbest_{kj} - X_{ij}^t) * rand(-1,1) \quad (3\text{-}11)$$

$$A_1 = a_1 * \exp(-\frac{p_i}{sump} * N) \quad (3\text{-}12)$$

$$A_2 = a_2 * \exp\left((\frac{p_i - p_k}{|p_k - p_i| + \partial}) * \frac{N * p_k}{sump + \partial}\right) \quad (3\text{-}13)$$

$mean_j$ 表示整个鸟群平均位置的第 j 个元素，k(k!=1)为一个[1,N]之间的随机整数 a_1, a_2 为[0,2]之间的常量。 p_i 为个体 i 的最优值， sump 为群体的最佳适应度值之和， ∂ 为最小常量。A_1 表示周边环境对鸟儿飞向群体中心的影响，A_2 表示一个特定的干扰对鸟儿飞向群体中心的影响，如果第 k 个个体的适应度值优于第 i 个个体的适应度值则 $A_2 > a_2$，这也意味着第 i 个个体遭受的干扰比第 k 个个体要大，第 k 个个体比第 i 个个体更有可能移动到群体的中心。

(4)飞行觅食：鸟儿为了寻找食物会定期的从一个地方飞到另一个地方，期间一些鸟会作为生产者而另一些鸟会作为乞食者，其中生产者的位置变换公式如下：

$$X_{ij}^{t+1} = X_{ij}^{t} + rand(0,1) * X_{ij}^{t} \tag{3-14}$$

乞食者的位置变换公式如下：

$$X_{ij}^{t+1} = X_{ij}^{t} + (X_{kj}^{t} - X_{ij}^{t}) * FL * rand(0,1) \tag{3-15}$$

上式中$k \in [1, 2, \quad , N]$，FL（$FL \in [0, 2]$）表示乞食者跟随生产者去寻找食物。

由上面的公式可以看出鸟群算法进行的是个体向更加优秀个体靠近的过程，而靠近的方式分为向自己历史最优位置和种群最优位置两种。个体下一次的位置由当前位置和个体的行为决定。鸟群的定期迁徙扩大了种群的搜索范围增加了粒子的多样性。

尽管 BSA 算法在求解多目标函数方面由于 PSO 算法和 DE 算法，但是 BSA 算法也存在着一些缺点，由于初期种群多样性不足，会导致在解决一些多极值函数（如 Ackley 函数）优化问题时会出现局部收敛的情况，同时 BSA 算法的认知和社会行为调节参数的变化易导致种群收敛精度变差和收敛迭代次数偏大。近年来有一些对 BSA 算法进行改进，如文献[74]提出使用 *Levy* 飞行策略改进 BSA，当鸟类进行迁徙时引入 *Levy* 飞行来提高算法的性能。文献[75]使用非线性反余弦的方法对 BSA 中的认知行为和群体的社会行为进行动态的调整以增大粒子的搜索范围, 以期提高粒子的质量, 丰富粒子的多样性。文献[76]提出使用交叉边界的方法对个体中的边界位置进行改进。这些改进算法虽然在一定程度上改善了 BSA 算的多样性，但是后期还是易陷入局部最优解。

鸟群算法中虽然通过鸟群的定期迁徙能够扩大其搜索范围，但是算法后期仍然存在着搜索能力不足，易陷入局部最优解等缺点，鸟群算法中通过观察鸟类的觅食，警戒，迁徙等行为提出鸟类的位置变换公式，同样通过观察鸟群的迁徙行为可知，鸟类在迁徙的时候途中会不断的有鸟加入到鸟群中，这些鸟所携带的信息会影响群体中的其他的鸟的搜索路径，同时群体中的鸟由于各种因素会离开群体进行单独的觅食或加入到其他的鸟群中因此无论鸟是离开群体还是新的鸟加入群体都会对群体的觅食信心产生影响。为了简单起见我们做以下的约束：

约束一：新加入的鸟已经在本地进行了觅食，警戒等行为，所以它们都带有初始位置信息，由于鸟群中的鸟对于该地点的环境不熟悉所以新加入的成员的位置信息对于鸟群中的其他鸟而言是随机的，鸟群中的鸟对新加入的鸟的信息也会产生影响。

约束二：由于鸟群在迁徙时中途不断地有鸟加入群体，同时群体中的鸟也不断地会飞出，这里我们假设加入的鸟的数量和离开的鸟的数量是相等的，因此鸟群的规模不变。

约束三：鸟群中食物储备量大的鸟由于在觅食时搜索范围更广所以更容易离开鸟群，

同理其他的鸟群中的那些食物储备量大的鸟也更容易离开原来的鸟群加入到新的鸟群，与之相反鸟群中食物储备量较少的鸟离开鸟群的可能性较小。

约束四：鸟群在迁徙觅食的时候总是寻找新的觅食地点而不是搜索已经觅食过的地点，因此个体的历史记录会对个体的下一次的搜索位置产生影响，本文这里假设个体中只有上一次的搜索位置和现在的位置会影响个体的下一次搜索位置。现对公式(3-14)进行改进：

$$X_{ij}^{t+1} = X_{ij}^{t} + rand(0,1) * X_{ij}^{t} + \kappa X_{ij}^{t-1} \tag{3-16}$$

上式中κ表示个体的历史位置对个体的影响程度，为一个范围在[0,1]中的定值。因为食物的储备量是一个相对的概念，一只鸟可能在鸟群中食物储备量较少但是在其他的群体中食物储备量较高，为了简单本文中新加入的鸟随机的成为生产者和乞食者。

新加入的鸟的位置变换公式如下：

$$X_{ij}^{t+1} = \begin{cases} \chi XN_{ij}^{t} + (1-\chi)\ XO_{ij}^{t} + \kappa XO_{ij}^{t-1}, rand(0,1) > \zeta \\ XN_{ij}^{t}, rand(0,1) < \zeta \end{cases} \tag{3-17}$$

上式中XN_{ij}^{t}表示鸟群中新加入的个体，XO_{ij}^{t}表示鸟群中已有的个体，χ表示新加入个体的位置信息和已有的个体的位置信息对鸟的位置变换的改变影响程度，通常为一个常数本文中取 0.5，ζ表示个体中每一个维度的变换阈值。

上述鸟群算法中粒子的位置变换和粒子对于认知和社会群体行为，以及迁徙的环境密切相关，不同的认知或迁徙的行为会影响粒子的搜索能力，实际的云计算中任务的长度，任务的需求不同也会影响上述的位置变换从而影响粒子的多样性，因此为尽可能的保持粒子多样性的稳定，我们引入 DE 算法，DE 算法由于存在交叉，变异等操作能够扩大种群的搜索范围，增加离子的多样性，并且该多样性和外部环境无关，只由粒子本身决定。因此本文除了上面的方法之外结合 DE 算法对鸟群算法进行改进，当鸟群中的鸟的位置进行变换后使用 DE 算法的选择操作将适应度值较好的个体选择出来作为父代进行杂交易产生子代，并将较优秀的子代保留下来从而朝着最优化的方向进化。改进的鸟群算法如下：

Set the number of individuals N, number of iteration M, the frequency of birds' flight FQ, the probability of foraging P, c1,c2,a1,a2,FL,PL, κ

Initialise X,t=0

Evaluate the N individuals' fitness value, gbesti=Xi, Pbest=argmaxXifitness(Xi)

While(t<M)

    If(t%FQ!=0)

        For i=1:N

```
            If rand(0,1)<P
                Update Xi with equation (3-10)
            Else
                Update Xi with equation (3-11)- (3-13)
            End if
        End for
Else
divide the swarm into producers and scroungers
For i =1:N
        If Xi is producer
            Update Xi with equation (3-16)
            If rand(0,1)<PL
Update Xi with equation (3-17)
                End if
            Else
                Update Xi with equation (3-15)
            If rand(0,1)>PL
Update Xi with equation (3-17)
                End if
            End if
        End for
    End if
    Randomly select individual and generate perturbance with (2-3)
    Crossover and mutation operators generates individual with (2-4)
    Choose a better individual with (2-5)
    Evaluate new X
    Update gbesti,Pbest= argmaxXifitness(Xi)
t=t+1
End while
```

改进鸟群算法通过补充鸟群迁徙过程中的周围鸟儿的加入以及鸟群中鸟儿的离开行为，以增强鸟群搜索后期的粒子多样性避免陷入局部最优解，通过引入 DE 算法扩大搜索前期的搜索范围也能增加粒子的多样性在后期避免陷入局部最优解。

3.3.2 新蝙蝠群算法

新型蝙蝠群算法(NBA)是 2015 年 Xian-Bing Meng 等人提出的智能启发式全局搜索算法该算法主要是在蝙蝠群算法[77]上面进行改进而来，该算法主要通过观察蝙蝠利用回声定位捕食行为，蝙蝠在捕食时会通过调整波长，音量以及脉冲发射频率来改变自己的位置，新型蝙蝠算法在传统的蝙蝠算法中加入了栖息地的选择和自适应回声补偿多普勒效应，新

蝙蝠群算法中的个体能随机的选择在栖息地中进行捕食，并且能够根据目标的接近程度自适应的调整多普勒补偿率。为了模拟蝙蝠探测食物，躲避障碍，蝙蝠算法做了以下假设：

(1)所有的蝙蝠利用回声定位的方法感知距离，并且它们也能区别食物和障碍物之间的不同；

(2)蝙蝠在位置 x_i 时的飞行速度为 V_i，并且蝙蝠在飞行的时候会以固定的频率 $f_{\min}$，不同的波长 λ 和响度 A_0 来搜素食物。它们能够根据距离食物的范围自动的调节发射脉冲的波长或频率并且同时调整脉冲的发射率 $r \in [0,1]$；

(3)蝙蝠发出的声波的响度从一个大的正数 A_0 逐渐减小到一个小的固定值 $A_{\min}$。

一般情况下频率 $f \in [f_{\min}, f_{\max}]$ 其对应的波长范围为 $\lambda \in [\lambda_{\min}, \lambda_{\max}]$，在实际应用中由于 λf 是一个固定的值，可以根据不同的问题调整波长，可以通过调整波长或频率来调整搜索范围，为了简化问题，通常 $f \in [0, f_{\max}]$，频率越高波长就越短，蝙蝠飞行的距离也就越短。脉冲发射率在[0,1]之间 0 表示没有发射脉冲，1 表示最大的发射率。

在一个 d 为的搜索空间中，蝙蝠的位置 x_i 和速度 v_i 的变换公式如下：

$$f_i = f_{\min} + (f_{\max} - f_{\min})b \tag{3-18}$$

$$v_i^{t+1} = v_i^t + (x_i^t - x_*)f_i \tag{3-19}$$

$$x_i^{t+1} = x_i^t + v_i^{t+1} \tag{3-20}$$

上述表示中 $\beta \in [0,1]$，x_* 是当前全局最佳位置，$f_{\min}$ 和 $f_{\max}$ 的大小依赖于问题的领域大小而定。

如果一个生成的随机数比脉冲的发射率 r 大则会进行局部搜索，蝙蝠进行局部搜索时的位置变换公式为:

$$x_n = x_o + \varepsilon A_{mean}^t \tag{3-21}$$

$\varepsilon \in [-1,1]$ 是一个随机数，A_{meam}^t 是群体的平均响度，一旦蝙蝠发现了食物则超声波的响度会逐渐的降低同时脉冲发射率会增加，其公式如下：

$$A_i^{t+1} = \alpha A_i^t \tag{3-22}$$

$$r_i^{t+1} = r_i^0[1 - \exp(\gamma t)] \tag{3-23}$$

其中 $\alpha \in [0,1]$，$\gamma > 0$，它们都是一个给定的常数值，通常情况下为 0.9，A_i^0 通常在

[1,2]之间，初始的 r_i^0 通常接近 0。

新蝙蝠算法上面的基础上引入了觅食环境选择和自适应的多普勒补偿机制，为了简化觅食过程做了如下的规定：

(1)所有的蝙蝠都能都随机的在不同的栖息地觅食；

(2)所有蝙蝠都可以在多普勒效应获得补偿，补偿率会依据目标接近程度自适应地行调整。

蝙蝠的栖息地选择主要分为量子行为和机械行为，蝙蝠的量子行为会使得蝙蝠在更大，更广泛的空间中进行觅食，蝙蝠的机械行为会使得蝙蝠在有限的环境中进行觅食。蝙蝠的量子行为位置变换公式如下：

$$x_{ij}^{t+1} = \begin{cases} g_j^t + \rho\left|m_j^t - x_{ij}^t\right| \ln\left(\frac{1}{u_{ij}}\right), & r < p \\ g_j^t - \rho\left|m_j^t - x_{ij}^t\right| \ln\left(\frac{1}{u_{ij}}\right), & r \geq p \end{cases} \tag{3-24}$$

上式中 g_j^t 表示当前群体中的全局最优位置，m_j^t 表示当前群体中的平均适应度值，u_{ij} 为区间在[0,1]之间的随机数，ρ 为收缩系数，p 为一个常数通常情况为 0.5。

蝙蝠的机械行为位置变换公式如下：

$$f_{ij} = f_{\min} + (f_{\max} - f_{\min})r \tag{3-25}$$

$$f_{ij}^t = \frac{c + v_{ij}^t}{c + v_{gj}^t} f_{ij} \left(1 + C_i \frac{g_j^t - x_{ij}^t}{\left|g_j^t - x_{ij}^t\right| + J}\right) \tag{3-26}$$

$$v_{ij}^{t+1} = wv_{ij}^t + (g_j^t - x_{ij}^t)f_{ij}' \tag{3-27}$$

$$X_{ij}^{t+1} = X_{ij}^t + V_{ij}^{t+1} \tag{3-28}$$

$f_{\min}, f_{\max}$ 为频率的最小值和最大值，c 为一个定值通常为 340，C_i 为多普勒补偿率，为[0,1]的一个随机值，ϑ 为一个随机值，w 是权重为一个（0,1）的定值，v_{ij} 为粒子的速度。蝙蝠在接近食物会增加脉冲发射频率同时减小响度以增加局部搜索能力，此外响度的影响因素除了自身原因外也会受其它的蝙蝠的影响，其位置更新公式如下：

$$x_{ij}^{t+1} = g_j^t + (1 + rand(0, \sigma^2)) \tag{3-29}$$

$$\sigma^2 = \left|A_i^t - A_{mean}^t\right| + \varepsilon \tag{3-30}$$

Ai 响度，ε 为一个随机值。

该算法在早期使用栖息地选择的方式使得蝙蝠在早期的搜索范围广，粒子的多样性得到增强，在进行局部搜索时多样性会减弱，因此易陷入局部最优解，特别是响度和脉冲发射频率参数的设置，如果太大容易陷入局部最优解，如果太小又不利于算法的快速收敛。

新蝙蝠群算法是在蝙蝠群算法的基础上引入了蝙蝠对栖息地环境的选择以及自适应的多普勒补偿机制虽然在一定程度上弥补了蝙蝠算法在收敛精和收敛速度，然而在算法搜索后期由于蝙蝠向着猎物聚集减小搜索范围使得粒子很难跳出局部的最优解。通过分析蝙蝠的机械行为可知，该行为主要是蝙蝠在有限的范围空间进行食物的搜寻，其位置的变换主要依靠蝙蝠的飞行速度，其变换公式优点类似于粒子群算法，遇到的问题也和粒子群算法的早熟现象相似。针对粒子群的早熟问题文献[78]将二阶振荡方法加入到粒子群速度更新公式，改进后的算法在求解粒子第 $k+1$ 次迭代速度公式中，加入了第 k-1 次迭代时粒子的速度信息。充分利用第 $k-1$ 代速度信息加强粒子的自学习能力，同时丰富种群的多样性。

本文对其蝙蝠的速度变换公式中同样引入二阶振荡环节，现将公式(3-27)加入二阶振荡后的改进公式如下：

$$\begin{aligned} v_{ij}^{t+1} &= \omega v_{ij}^{t} + (\beta_1(p_{ij} - (1+\varepsilon_1)x_{ij}^{t} - \varepsilon_1 x_{ij}^{t-1}) \\ &+ \beta_2(g - (1+\varepsilon_2)x_{ij}^{t} + \varepsilon_2 x_{ij}^{t-1}))*f_{ij}' \end{aligned} \tag{3-31}$$

其中 $\beta_1 = c_1 r_1$，$\beta_2 = c_2 r_2$。ω 与 c_i 为惯性权重和学习因子。p_{ij} 表示蝙蝠 i 的第 j 维上的历史最好位置，g 表示群体的最好位置。$T_{\max}$ 为最大迭代次数，当迭代次数 $k < T_{\max}/2$ 时，如果 $\varepsilon 1 < \frac{2\sqrt{\beta_1}-1}{\beta_1}$ 并且 $\varepsilon 2 < \frac{2\sqrt{\beta_2}-1}{\beta_2}$，此时算法搜索最优解的能力较强。当迭代次数 $k \geq G_{\max}/2$ 时，如果 $\varepsilon 1 \geq \frac{2\sqrt{\beta_1}-1}{\beta_1}$ 并且 $\varepsilon 2 \geq \frac{2\sqrt{\beta_2}-1}{\beta_2}$，此时算法的收敛速度快。为了使算法同时拥有较快的收敛速度和较好的全局搜索能力 $\varepsilon 1, \varepsilon 2$ 的取值如下：

$$\varepsilon 1 = \begin{cases} \dfrac{2\sqrt{\beta_1}-1}{\beta_1} = \dfrac{2\sqrt{c_1 r_1}-1}{c_1 r_1}, t < G_{\max}/2 \\ \dfrac{(2\sqrt{\beta_1}-1)(1+r_3)}{\beta_1} = \dfrac{(2\sqrt{c_1 r_1}-1)(1+r_3)}{c_1 r_1}, t \geq G_{\max}/2 \end{cases} \tag{3-32}$$

$$\varepsilon 2 = \begin{cases} \dfrac{2\sqrt{\beta_2}-1}{\beta_2} = \dfrac{2\sqrt{c_2 r_2}-1}{c_2 r_2}, t < G_{\max}/2 \\ \dfrac{(2\sqrt{\beta_2}-1)(1+r_4)}{\beta_2} = \dfrac{(2\sqrt{c_2 r_2}-1)(1+r_4)}{c_2 r_2}, t \geq G_{\max}/2 \end{cases} \tag{3-33}$$

上式中 r_3, r_4 为(0,1)范围内的随机数。为了避免因定值学习因子造成的种群多样性降

低和陷入局部最优解，对$c1, c2$引入动态参数机制

$$c_1 = \beta_1' + \beta_1 \cos(\rho_2 \times t / G_{max}) \tag{3-34}$$

$$c_2 = \beta_2' + \beta_2 \cos(\rho_2 \times t / G_{max}) \tag{3-35}$$

上式中$\beta_1' = c_1' + \frac{\alpha_1}{2}$，$\beta_1 = \frac{\alpha_1}{2}$，$\rho_2 = 2\rho_1$，$\beta_2' = c_2' + \frac{\alpha_2}{2}$，$\beta_2 = -\frac{\alpha_2}{2}$，$\rho_2 = 2\rho_1$，$c_1', c_2'$为学子因子，为一个定值，$\alpha_1, \alpha_2, \rho_1$为权重因子是一个常量。此外本文使用的缩放系数$\rho$，多普勒补偿率 Ci，栖息地的选在概率 P，惯性权重ω也不是定值而是根据当前的蝙蝠的位置而动态的改变，其公式如下：

$$\rho = \frac{(\rho_{max} - \rho_{min})(G_{max} - t)}{M} + \rho_{min} \tag{3-36}$$

$$C_i = rand(0,1)(C_{max} - C_{min}) + C_{min} \tag{3-37}$$

$$P = rand(0,1)(P_{max} - P_{min}) + P_{min} \tag{3-38}$$

$$\omega = \frac{(\omega_{max} - \omega_{min})(G_{max} - t)}{M} + \omega_{min} \tag{3-39}$$

上述公式中r_{max}, r_{min}分别表示缩放系数的最大值和最小值，通常为一个给定的定值，C_{max}, C_{min}为多普勒补偿率的最大值和最小值，为一个常数，P_{max}, P_{min}为蝙蝠栖息地的选择概率的最大值和最小值，w_{max}, w_{min}为惯性权重的最大值最小值。

正如上文所提到的，在实际的云计算环境下，不同的任务所需求的资源不同，学习因子，脉冲的发射频率，权重因子等都会对粒子的多样性产生影响，使得算法在搜索的后期粒子的多样性受影响很大，同理我们在上面的改进方法中再结合 DE 算法以增强离子的多样性，扩大搜索范围。具体算法如下：

Set the number of individuals N, number of iteration M, probability of habitat selection P, the frequency of updating the loudness and pulse emission rate, $\alpha_1, \alpha_2, \rho_1, \vartheta, r$.

Initialise x,v,t=0

Evaluate the N individuals' fitness value, pi=xi, g=argmaxXifitness(xi).

While(t<M)

    Compute P using equations (3-38)

    If rand()<P

        xpr=x

```
        update x with equation (3-24),(3-36)
    else
        update x using equations (3-25),(3-26),(3-28), (3-31)-(3-35),
        (3-37),(3-39)
        xpr=x
        update x using equation(3-28)
    end if
    if rand()>ri
        update x using (3-29),(3-30)
    end if
    select p1,p2,p3 generate perturbance using (2-3)
Crossover and mutation operators generates individual with (2-4)
    Choose a better individual with (2-5)
    Evaluate new x
    Update pi,g= argmaxXifitness(xi)
    if(rand(0,1)<Ai&&fitness(xi)<fitness(g)
        update A,r with(3-22)( 3-23)
    end if
    t=t+1
end while
```

改进新蝙蝠群算法不仅通过引入二阶振荡的方法增强了种群的自学习能力，丰富了种群在搜索后期的粒子多样性，还通过引入 DE 算法不断的进行进化择优使得算法保持多样性的同时能够搜索到最优解，这种算法可以更加适应于多类型的最优值求解问题。

3.3 改进鸟群算法和改进新蝙蝠群算法仿真

为了验证本文提出的改进的鸟群算法和新蝙蝠群算法，对于求解本文提出的任务最短执行时间模型的有效性，我们分别对改进的鸟群算法和改进的新蝙蝠群算法进行了实验仿真，为了证明本文提出的改进算法的适用性，我们在对上述的改进算法进行仿真的时候分别使用了随机生成的模拟任务数据和卢森堡大学高性能计算中心所公开的任务调度数据集进行验证，以证明本文提出的算法不仅能够用于理论研究也能用于实际的调度环境。

本文主要的对比标准为任务的完成时间和任务的负载均衡，负载均衡我们主要通过三个方面衡量：

(1)极差值：最大完成时间和最小完成时间差，其值越小则负载越均衡。

(2)相对极差值：极差值和完成时间的平均值相除。

(3)标准差。上述三个衡量公式如下。

$$LB_1 = T_{\max} - T_{\min} \tag{3-40}$$

$$LB_2 = \frac{T_{\max} - T_{\min}}{T_{avg}} \tag{3-41}$$

$$LB_3 = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(T_j - T_{avg})^2} \tag{3-42}$$

上式中$T_{\max}, T_{\min}, T_{avg}$分别表示虚拟的最大执行时间，最小执行时间和平均执行时间。

3.3.3 改进鸟群算法仿真

在云计算中，若一个大粒度的任务经过逻辑划分后可以得到 m 个子任务，当这些子任务被调度执行完成后则整个任务就执行完毕，若用户提交的任务本身粒度较小，就无需划分直接进行调度，此时假设用户提交的任务数量也是 m。在本文的优化算法求解过程中，一次任务调度对应一个 m 维的粒子。一个粒子的位置可以由 m 维向量$x_i = \{x_{i1}, x_{i2}, x_{i3}.....x_{im}\}$来表示，$x_{ij}$表示个体$P_i$第 j 维方向的位置。其中粒子的位置$x_{ij}$在任务调度过程中表示子任务$T_j$选择的虚拟机编号。下面的示例图表示了粒子与虚拟机调度的对应关系

粒子维度	1	2	3	4	5	6

任务编号	T1	T2	T3	T4	T5	T6
虚拟机编号	3	2	1	3	4	5

由上面的分析可知$pbest_i$和$gbest$分别代表了个体P_i当前的最优位置和整个鸟群中的全局最优位置。$pbest_i = \{Pb_{i1}, Pb_{i2}, Pb_{i3}.....Pb_{im}\}$，$Pbest_{ij}$代表个体$P_i$在第 j 维方向上的最优位置。$gbest = \{gb_1, gb_2, gb_3.....gb_m\}$表示子任务集$T = \{T_1, T_2, ...T_m\}$所选择的最优虚拟机调度方案。在算法最初时随机生成个体P_i，并将其赋值给$pbest_i$，并将经过适应度函数(3-9)求解的适应度值最小的个体值赋给$gbest$。之后的迭代过程中，不断的更新$pbest_i$与$gbest$。当迭代停止的时候最终的$gbest$就是全局最优值，也就是我们要求的解。

在进行本算法仿真验证时，虚拟机 MIPS 在[1000,5000]之间, MEM 在[200,500]之间，任务的长度在[5000,10000]之间，任务数量从[50,500]之间，虚拟机的数量在[10,50]之间，鸟群的大小为 100，鸟群的迁徙频率为 5，c1,c2 位 1.5，a_1,a_2 为 1，交叉概率为 0.9，变异概率为 0.5，个体离开群体的概率为 0.5，为了减少偶然因素的影响我们对实验进行了多次仿真，在对其结果取平均值进行对比。

本文主要和 GA，PSO，DE，BSA 算法进行对比，结果如所示(m 代表任务数，n 表示虚拟机数）：其任务数量和完成时间如下：

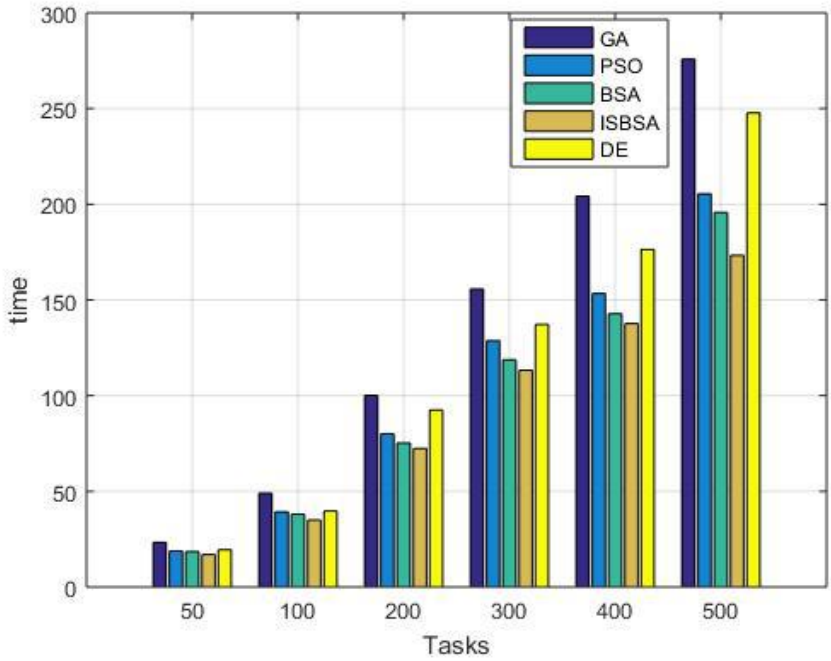

图 3-1　虚拟机数量为 10

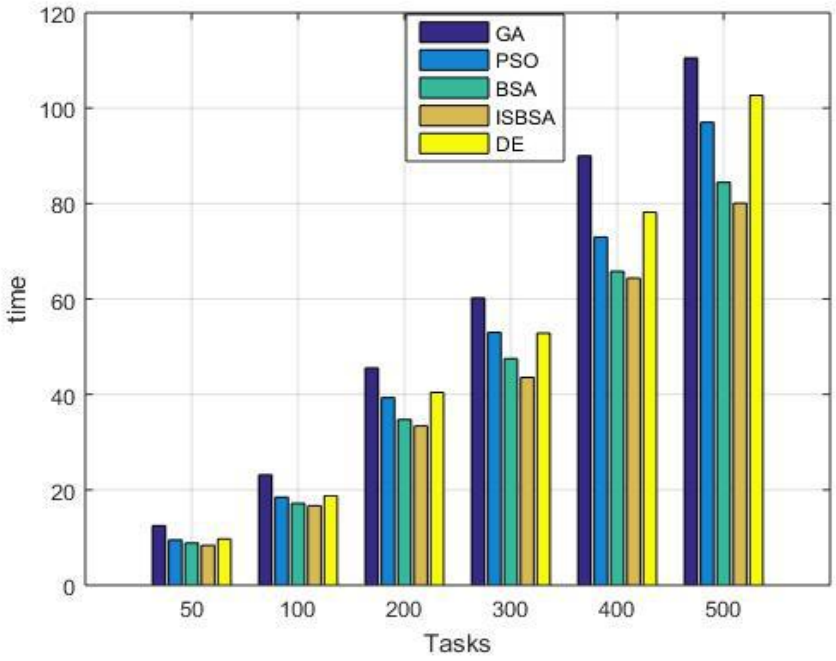

图 3-2　虚拟机数量为 30

其负载均衡值如下：

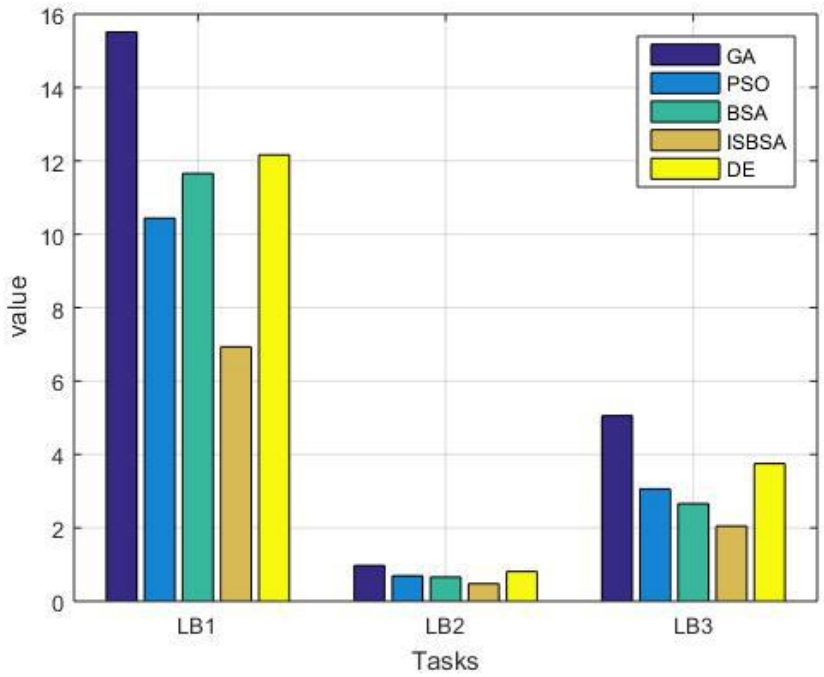

图 3-3　任务数为 50，虚拟机数为 10

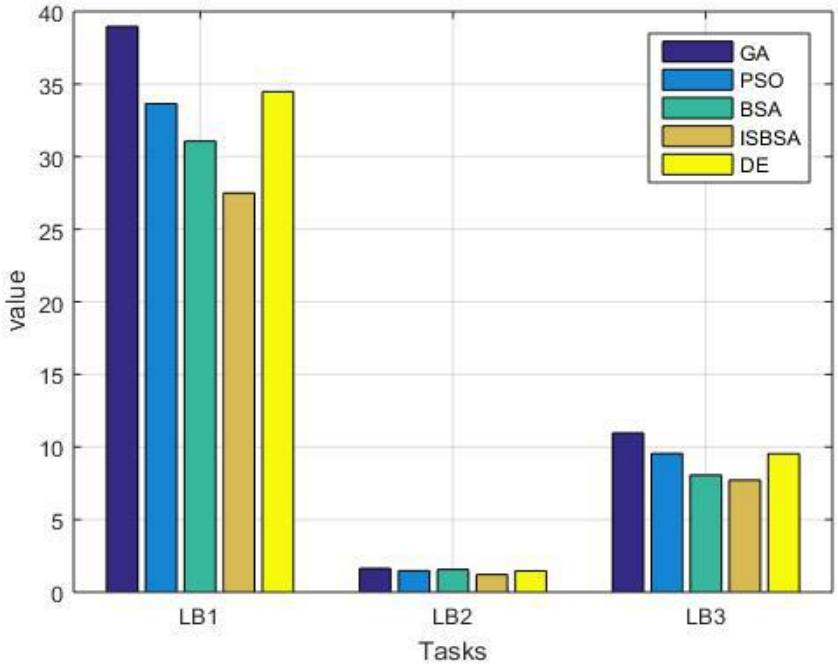

图 3-4　任务数为 200，虚拟机数为 30

从上面的图中可以看出本文提出的 ISBSA 算法无论是是完成时间上还是在均衡度上都要优于上述的其他的算法，从图 3-1 和图 3-2 可以看出随着任务的增加完成时间都在增长，但是本文提出的算法的增长速度最慢，这也从另一个方面说明本文提出的优化算法在进行任务分配时任务并没有集中的分配到某一台虚拟机上面，而是均匀的分配，从图 3-3，图 3-4 可以看出在本文给出的三种负载均衡标准上 ISBSA 的结果都是最优的，并且随着任务的增加和虚拟机的增加 ISBSA 算法的负载均衡依旧由于其他的算法，尽管随着任务数和虚拟机数的增加，问题的解空间也在增加但是 ISBSA 仍然能够在巨大的解空间中搜索到最优解。为了证明本算法在小规模任务和大规模任务都能够搜索到最优解本文任务数从 50 到 500 的完成时间和负载均衡如下表 3-1,3-2 所示：

表 3-1：任务完成时间

m	n	GA	PSO	BSA	IBSA	DE
50	15	19.324951	15.404700	14.712309	13.545967	15.591722
50	25	13.050131	10.861492	9.314073	9.313182	11.209752
50	35	12.610549	10.016182	8.926777	8.209944	9.995159
50	45	10.574416	8.801991	8.003431	7.819766	8.492614
100	25	25.428078	21.434100	20.307119	17.738449	21.560326
100	35	19.710097	16.663485	15.287984	14.478463	16.484296
100	45	18.612414	15.695191	14.097561	13.212735	15.476050

100	55	16.798921	13.792695	12.288375	11.906649	13.252545
200	15	78.139973	62.606172	52.873345	52.529040	67.561847
200	25	52.861281	43.747376	41.132673	38.563580	44.326202
200	35	34.409623	30.917369	26.567289	25.578874	30.756798
200	50	32.713030	27.967210	24.298678	23.656510	28.316929
300	15	110.143185	87.435944	81.538667	72.645575	94.347304
300	35	62.470060	51.701541	46.843201	44.643168	52.782725
300	45	47.086533	39.869690	34.674380	34.648762	39.579561
400	15	136.821922	116.638891	105.735953	103.010409	128.124573
400	25	100.069628	80.410542	71.834517	67.454025	83.708315
400	35	75.198086	65.465631	56.352928	56.035757	68.165152
400	45	65.096855	55.526335	49.154237	47.460199	56.507259
500	35	95.356789	83.269934	74.620759	67.595388	85.437679
500	45	81.332150	72.013541	63.043866	63.754684	72.295835
500	50	75.027626	65.567327	54.432041	54.412776	66.277070
500	55	67.339707	60.855249	54.486910	54.355332	60.266711

表 3-2：虚拟机负载均衡

		GA			PSO			BSA			ISBSA			DE		
m	n	LB1	LB2	LB3	LB1	LB2	LB3	LB1	LB2	LB3	LB1	LB2	LB3	LB1	LB2	LB3
50	**15**	**18.20**	**1.64**	**5.36**	**12.39**	**1.19**	**3.65**	**12.60**	**1.60**	**3.67**	**10.01**	**0.99**	**2.90**	**12.77**	**1.22**	**3.67**
50	**25**	**13.05**	**2.11**	**3.69**	**10.86**	**1.83**	**3.03**	**9.31**	**2.21**	**3.11**	**9.34**	**1.62**	**2.72**	**11.00**	**1.84**	**3.19**
50	**35**	**12.61**	**2.59**	**3.67**	**10.01**	**2.17**	**3.15**	**8.93**	**2.43**	**3.02**	**8.21**	**1.86**	**2.68**	**10.00**	**2.12**	**2.91**
50	**45**	**10.57**	**2.83**	**3.11**	**8.80**	**2.49**	**2.81**	**8.00**	**2.50**	**2.70**	**7.82**	**2.30**	**2.58**	**8.49**	**2.40**	**2.73**
100	**25**	**23.57**	**1.74**	**6.39**	**18.31**	**1.41**	**5.30**	**18.17**	**1.56**	**5.87**	**15.18**	**1.22**	**4.23**	**18.69**	**1.42**	**5.12**
100	**35**	**19.28**	**2.09**	**5.41**	**16.26**	**1.88**	**4.68**	**15.29**	**1.89**	**4.56**	**14.48**	**1.72**	**4.17**	**16.25**	**1.86**	**4.38**
100	**45**	**18.61**	**2.43**	**5.11**	**15.70**	**2.12**	**4.32**	**14.10**	**2.09**	**4.10**	**13.21**	**1.87**	**3.81**	**15.48**	**2.12**	**4.33**
100	**55**	**16.80**	**2.75**	**4.45**	**13.79**	**2.36**	**3.92**	**12.29**	**2.58**	**3.99**	**11.91**	**2.10**	**3.58**	**13.25**	**2.24**	**3.79**
200	**15**	**56.28**	**1.20**	**17.41**	**47.52**	**1.07**	**14.41**	**39.03**	**1.03**	**9.03**	**29.63**	**0.70**	**8.95**	**49.99**	**1.11**	**15.11**
200	**25**	**44.69**	**1.52**	**12.86**	**34.75**	**1.26**	**10.44**	**35.88**	**1.85**	**9.88**	**34.06**	**1.26**	**9.33**	**36.48**	**1.28**	**10.55**
200	**35**	**30.66**	**1.70**	**7.96**	**26.93**	**1.52**	**7.36**	**23.28**	**1.87**	**6.27**	**22.01**	**1.28**	**5.98**	**26.81**	**1.52**	**7.12**
200	**50**	**31.73**	**2.22**	**8.11**	**27.37**	**2.03**	**6.94**	**23.60**	**2.00**	**6.60**	**22.63**	**1.70**	**5.80**	**27.52**	**1.97**	**7.27**
300	**15**	**76.23**	**1.12**	**24.10**	**57.53**	**0.91**	**17.84**	**59.50**	**1.00**	**14.50**	**41.48**	**0.68**	**12.41**	**65.78**	**1.00**	**20.36**
300	**35**	**53.38**	**1.73**	**14.85**	**45.10**	**1.53**	**12.66**	**42.29**	**1.40**	**11.25**	**36.94**	**1.27**	**10.66**	**45.69**	**1.52**	**12.5**
300	**45**	**42.48**	**1.88**	**10.55**	**35.99**	**1.66**	**9.20**	**30.33**	**1.53**	**8.50**	**30.07**	**1.40**	**8.17**	**35.51**	**1.61**	**9.28**
400	**15**	**94.60**	**1.05**	**29.59**	**75.24**	**0.88**	**23.46**	**76.14**	**0.86**	**20.14**	**64.23**	**0.77**	**19.99**	**93.29**	**1.05**	**30.59**
500	**55**	**59.40**	**1.87**	**15.44**	**54.03**	**1.76**	**14.07**	**50.23**	**1.72**	**13.25**	**50.14**	**1.65**	**12.90**	**54.90**	**1.78**	**14.02**

上面两个表描述的是本文提出的基于 ISBSA 的任务调度算法解决任务服务请求的调度问题得到的总时间开销情况和相关的负载情况。无论是在问题规模情况较大的情况下还是较小的情况下，本文提出的算法在不仅在求解任务执行时间上面小于另外四种算法求解

的任务执行时间，同时在负载均衡上面也表现得比其他的四种算法更好。虽然随着任务数量的增加，问题的求解规模和难度都在呈几何倍数增加但是本文提出的基于 ISBSA 的调度算法能够在这种背景下在保证收敛速度较快的同时保证解的质量较好以及解的质量的稳定性，说明 ISBSA 算法在求解大规模多目标优化问题时比其他的四种算法更优秀。

ISBSA 算法在保证收敛速度同时的具有较强的避免局部优化的能力。下面三个图是任务总数分别为 100,300,400 虚拟机数为 25,30,50 的情况下基于三种算法对应求得的完成时间随迭代次数增加的趋势图。

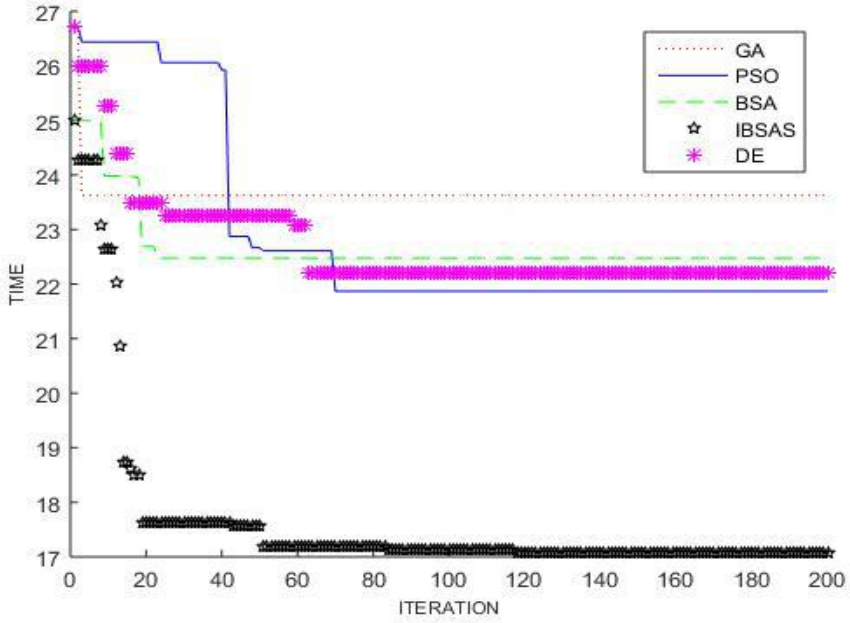

图 3-5　任务数 100 虚拟机数 25

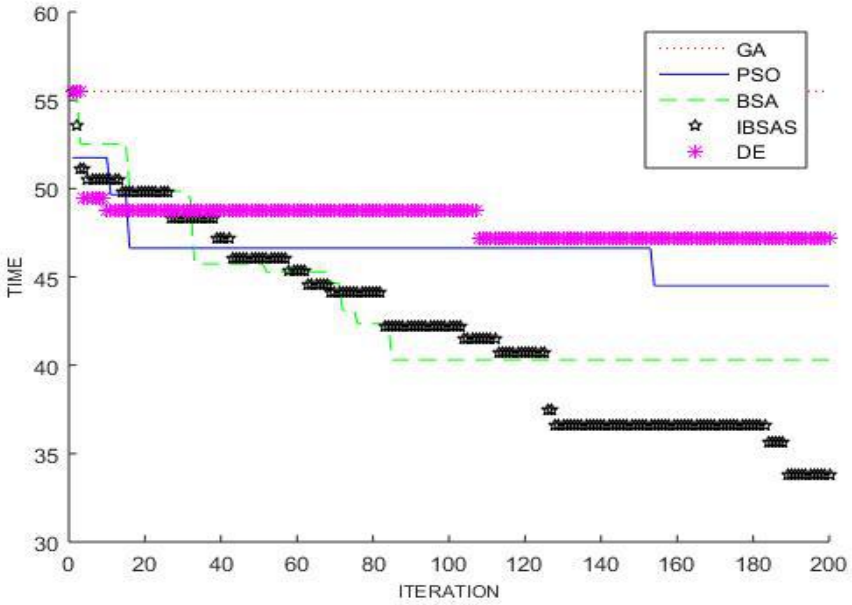

图 3-6　任务 300 虚拟机数 30

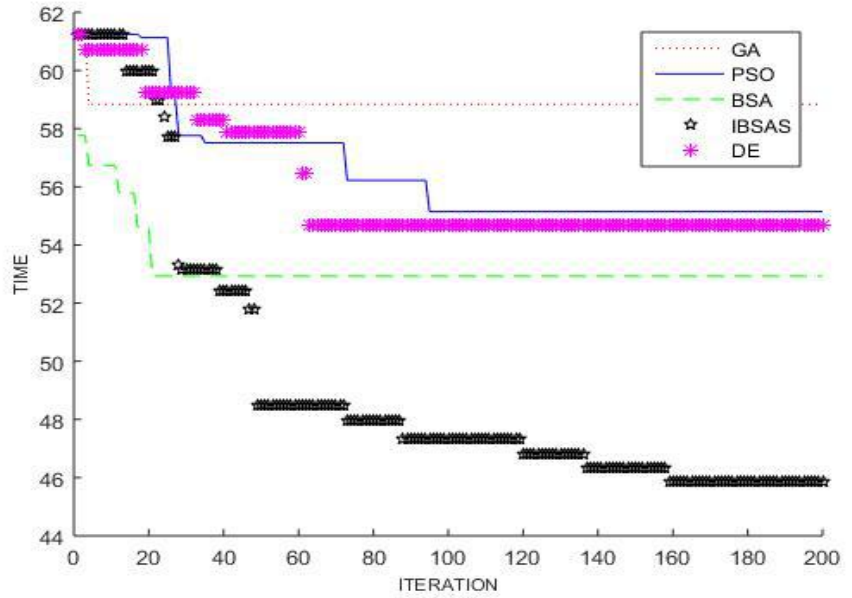

图 3-7　任务数 400 虚拟机数 50

从图中可以看出，五种算法中 ISBSA 展现了较强的跳出前期局部优化的能力以及快速收敛能力。ISBSA 算法在保持了各次迭代中种群优良性的情况下，不断和其他种群进行信息互换，不仅扩展了种群的多样性，且提高了最优解的质量。ISBSA 算法在传统鸟群算法的基础上加入了鸟群的飞进飞出操作不仅能增强全局搜索能力，并且通过引入差分进化算法还进一步丰富了种群的多样性。在迭代过程中出现停滞时，ISBSA 可以及时扩展种群的多样性和优良性，跳出局部最优，同时这些改变也是保持解质量前提下迭代速度

进一步提高的原因所在。为了证明本文的算法也适用于真实的数据，我们使用卢森堡大学的数据进行试验，实验室我们分别取任务数为 50,100,200,400 虚拟机数量为 10,20,30,40 实验结果如下：

表 3-3：任务完成时间

m	n	GA	PSO	BSA	ISBSA	DE
50	10	183.319103	153.785185	167.290055	148.619482	177.561500
50	20	151.787075	131.749793	101.210648	95.522324	119.172422
100	10	200.873859	163.911744	165.204364	141.027102	169.214096
100	20	155.100828	116.941479	106.038741	99.266310	120.281764
100	40	113.597828	106.795479	105.103541	99.266310	114.987350
200	10	340.300831	305.479214	334.119120	283.552957	324.328093
200	30	133.638889	122.792703	109.444502	85.986472	122.841902
400	30	123.960215	103.604872	104.471546	87.182268	102.765319

表 3-4：负载均衡

		GA			PSO			BSA			ISBSA			DE		
m	n	LB1	LB2	LB3	LB1	LB2	LB3	LB1	LB2	LB3	LB1	LB2	LB3	LB1	LB2	LB3
50	**10**	**165.28**	**1.32**	**52.41**	**53.60**	**0.41**	**17.89**	**144.01**	**1.20**	**46.58**	**52.02**	**0.47**	**17.72**	**121.55**	**0.98**	**38.21**
50	**20**	**148.64**	**2.26**	**39.65**	**131.75**	**2.07**	**45.05**	**95.93**	**1.51**	**30.45**	**95.52**	**1.50**	**30.35**	**119.17**	**1.78**	**44.84**
100	**10**	**138.00**	**1.04**	**35.44**	**56.60**	**0.43**	**18.08**	**113.76**	**0.86**	**33.01**	**23.69**	**0.18**	**7.77**	**81.95**	**0.63**	**23.26**
100	**20**	**155.09**	**2.22**	**50.93**	**116.89**	**1.68**	**39.30**	**106.05**	**1.73**	**38.05**	**99.26**	**1.46**	**33.57**	**115.32**	**1.85**	**33.66**
100	**40**	**100.08**	**2.02**	**38.95**	**100.87**	**1.58**	**37.25**	**99.98**	**1.48**	**36.54**	**95.87**	**1.34**	**33.24**	**101.58**	**1.60**	**36.66**
200	**30**	**133.17**	**2.29**	**41.93**	**122.79**	**2.23**	**39.94**	**102.21**	**1.78**	**34.67**	**85.98**	**1.61**	**29.41**	**120.40**	**2.13**	**39.11**
400	**30**	**121.72**	**2.29**	**33.08**	**102.49**	**2.03**	**31.05**	**101.24**	**1.97**	**29.81**	**87.18**	**1.83**	**28.00**	**102.76**	**2.03**	**30.84**

从上面的表三和表四可以看出使用真实数据集进性验证时，无论是任务的完成时间还是负载均衡，本文提出的 ISBSA 算法都要优于其它的四个算法。由于真实的调度数本质上是上面随机调度数据中的特例，而真实数据的调度结果也证明了本文提出的调度方法具有一定的实用性。

3.3.4 改进新蝙蝠群算法仿真

这一部分仿真实验和上一部分仿真实验类似，即将一个大粒度的任务请求经过逻辑划分后得到 m 个子任务，当这些子任务被调度执行完成后则整个任务就执行完毕，若用户提交的任务本身粒度较小，就无需划分直接进行调度，此时假设用户提交的任务数量也是

m。在本文的优化算法求解过程中，一次任务调度对应一个 m 维的粒子。一个粒子的位置和速度可以由 m 维向量 $x_i = \{x_{i1}, x_{i2}, x_{i3}.....x_{im}\}$ 和 $v_i = \{v_{i1}, v_{i2}, v_{i3}.....v_{im}\}$ 来表示，x_{ij} 和 v_{ij} 分别表示个体 P_i 第 j 维方向的位置和速度。其中粒子的位置 x_{ij} 在任务调度过程中表示子任务 T_j 选择的虚拟机编号。下面的示例图表示了粒子与虚拟机调度的对应关系

粒子维度	1	2	3	4	5	6

任务编号	T1	T2	T3	T4	T5	T6
虚拟机编号	3	2	1	3	4	5

和上面一样， $pbest_i$ 和 $gbest$ 分别代表了个体 P_i 当前的最优位置和整个蝙蝠群中的全局最优位置。 $pbest_i = \{Pb_{i1}, Pb_{i2}, Pb_{i3}. . . . . .Pb_{im}\}$， $Pbest_{ij}$ 代表个体 P_i 在第 j 维方向上的最优位置。 $gbest = \{gb_1, gb_2, gb_3. . . . . .gb_m\}$ 表示子任务 $T = \{T_1, T_2, \ldots T_m\}$ 所选择的最优虚拟机调度方案。

在进行算法仿真时和上面一样，虚拟机 MIPS 在[1000,5000]之间, MEM 在[200,500]之间，任务的长度在[5000,10000]之间，任务数量从[50,500]之间，虚拟机的数量在[10,50]之间，蝙蝠群的大小为 100，栖息地的选择概率为最大值，最小值分别为 0.8, 0.5，惯性权重最大值，最小值分别为 0.8, 0.4，多普勒补偿率最大值，最小值分别为 0.9, 0.1，缩放系数最大值，最小值分别为 1, 0.5。为了减少偶然因素的影响我们对实验进行了多次仿真，在对其结果取平均值进行对比。

本文主要和 GA，MPSO，NBA，ONBA，DE 算法进行对比，结果如所示（m 代表任务数，n 表示虚拟机数）：

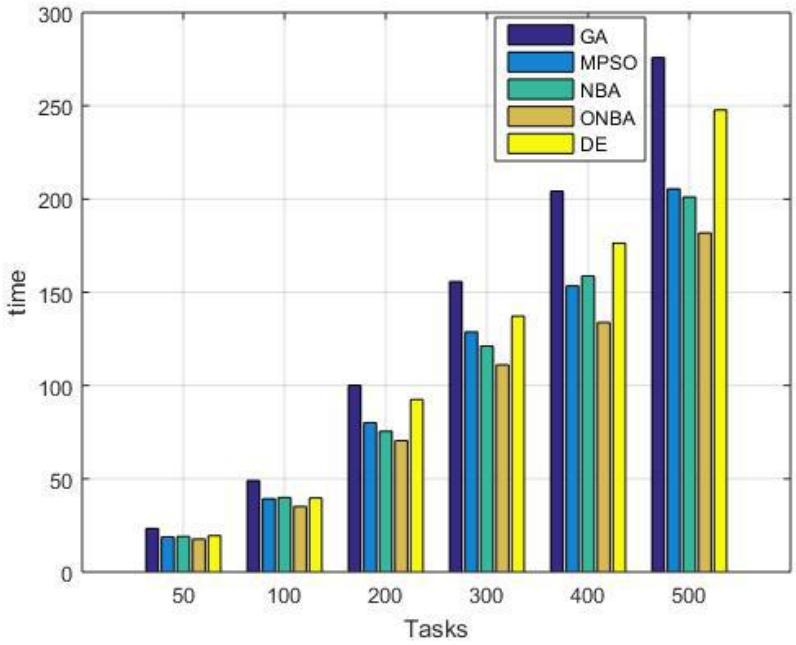

图 3-8　虚拟机数量为 10

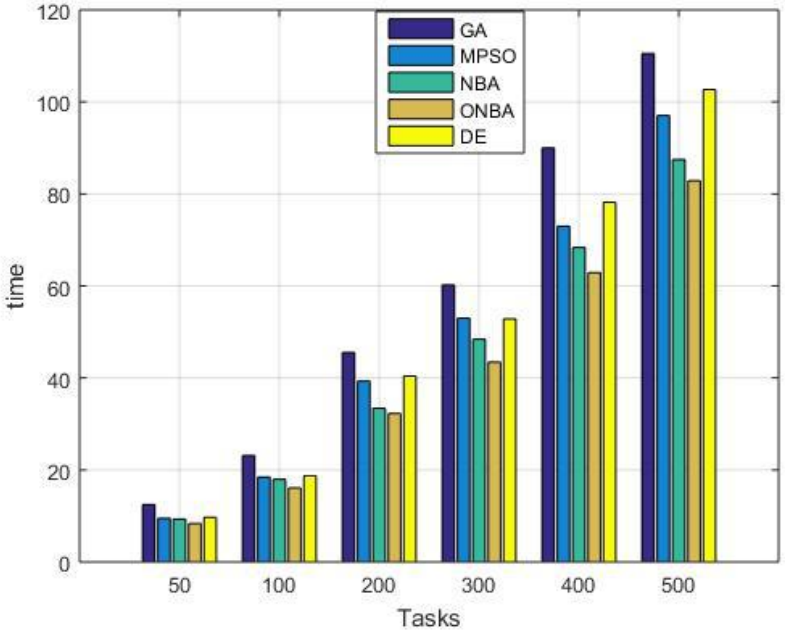

图 3-9 虚拟机数量为 30

负载均衡情况如下所示：

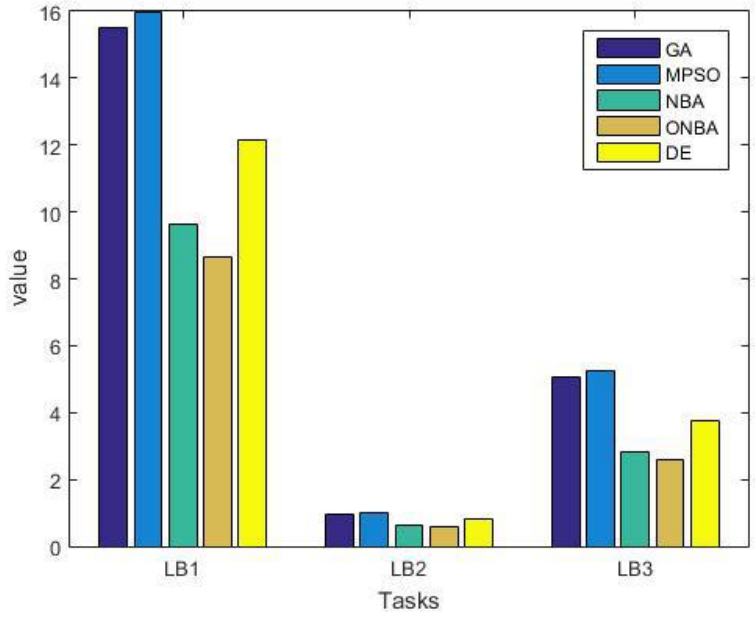

图 3-10　任务数量为 50，虚拟机数为 10

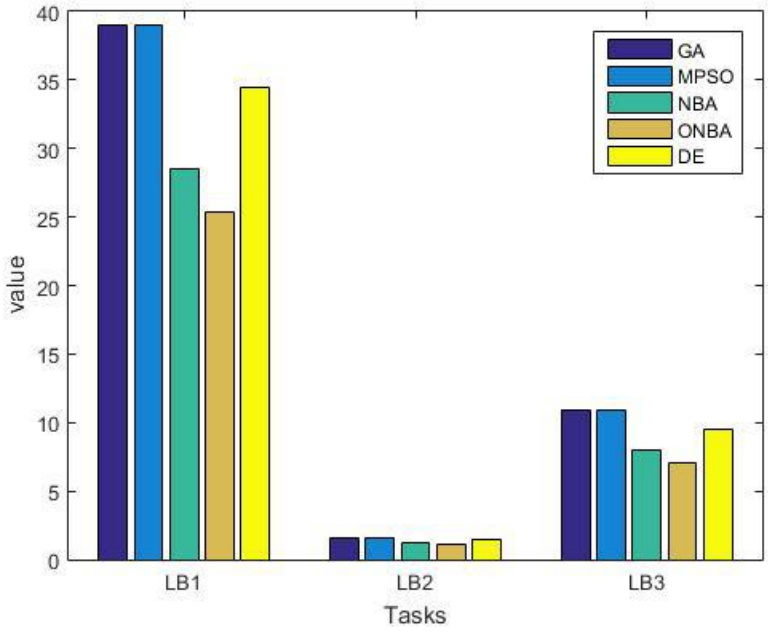

图 3-11　任务数量为 200，虚拟机数为 30

从上面的图中可以看出本文提出的 ONBA 算法在完成时间上和负载均衡都要优于上述的其他的算法，从图 3-8 和图 3-9 可以看出随着任务的增加完成时间虽然都在增长，但是本文提出的 ONBA 算法的完成时间依然最短，这也说明本文提出的算法在任务数量上有着更好的适用性，从图 3-10，图 3-11 可以看出在本文给出的三种负载均衡标准上 ONBA 的结果都是最优的，并且随着任务的增加和虚拟机的增加 ONBA 算法的负载均衡依旧由于其他的算法，尽管随着任务数和虚拟机数的增加，问题的解空间也增加但是 ONBA 仍

然能够在巨大的解空间中搜索到最优解。为了证明本算法在小规模任务和大规模任务都能够搜索到最优解本文任务数从 50 到 500 的完成时间和负载均衡如下：

表 3-5：任务完成时间

m	n	GA	MPSO	NBA	ONBA	DE
50	15	19.324951	15.404700	15.792500	14.632572	15.591722
50	25	13.050131	10.851492	9.409460	9.200116	11.209752
50	35	12.610549	10.016182	8.926777	8.209944	9.995159
50	45	10.574416	8.781991	7.711503	7.693663	8.492614
100	25	25.428078	21.433000	20.126577	18.469807	21.560326
100	35	19.710097	16.654385	15.986306	14.639473	16.484296
100	45	18.612414	15.655191	14.381032	13.837046	15.476050
100	55	16.798921	13.783195	12.488374	11.839655	13.252545
200	15	78.139973	61.596086	59.255568	53.002380	67.561847
200	35	34.409623	30.807358	27.223869	26.502978	30.756798
200	50	32.713030	27.821210	24.874986	23.443421	28.316929
300	15	110.143185	86.445144	84.119818	78.485922	94.347304
300	35	62.470060	51.451538	47.294624	44.644144	52.782725
300	45	47.086533	38.979983	37.291184	35.161460	39.579561
400	15	136.821922	114.879497	108.439787	102.062655	128.124573
400	25	100.069628	79.988542	76.671865	71.324126	83.708315
400	35	75.198086	64.965976	59.205584	57.950702	68.165152
400	45	65.096855	55.026715	51.908584	48.530527	56.507259
500	35	95.356789	82.941519	72.938800	69.998374	85.437679
500	45	81.332150	71.018194	64.230099	61.936040	72.295835
500	50	75.027626	64.883279	58.872767	55.663000	66.277070
500	55	67.339707	60.784249	54.619793	52.546994	60.266711

表 3-6：虚拟机负载均衡

		GA			MPSO			NBA			ONBA			DE		
m	n	LB1	LB2	LB3	LB1	LB2	LB3	LB1	LB2	LB3	LB1	LB2	LB3	LB1	LB2	LB3
50	**15**	**18.20**	**1.64**	**5.36**	**18.20**	**1.64**	**5.36**	**13.56**	**1.27**	**4.06**	**10.82**	**1.04**	**3.42**	**12.77**	**1.22**	**3.67**
50	**25**	**13.05**	**2.11**	**3.69**	**12.75**	**2.11**	**3.59**	**9.02**	**1.52**	**2.48**	**8.94**	**1.52**	**2.52**	**11.00**	**1.84**	**3.19**
50	**35**	**12.61**	**2.59**	**3.67**	**10.01**	**2.59**	**3.59**	**9.57**	**2.00**	**2.97**	**8.61**	**1.88**	**2.65**	**10.00**	**2.12**	**2.91**
50	**45**	**10.57**	**2.83**	**3.11**	**9.87**	**2.75**	**3.10**	**7.71**	**2.30**	**2.58**	**7.69**	**2.21**	**2.59**	**8.49**	**2.40**	**2.73**
100	**25**	**23.57**	**1.74**	**6.39**	**22.06**	**1.66**	**5.43**	**16.77**	**1.28**	**5.01**	**16.11**	**1.28**	**4.29**	**18.69**	**1.42**	**5.12**
100	**35**	**19.28**	**2.09**	**5.41**	**18.56**	**1.95**	**4.76**	**15.06**	**1.68**	**4.41**	**14.19**	**1.61**	**4.00**	**16.25**	**1.86**	**4.38**
100	**45**	**18.61**	**2.43**	**5.11**	**16.54**	**2.22**	**4.42**	**14.38**	**1.96**	**3.93**	**13.83**	**1.91**	**3.87**	**15.48**	**2.12**	**4.33**
100	**55**	**16.80**	**2.75**	**4.45**	**14.80**	**2.46**	**4.02**	**12.48**	**2.15**	**3.50**	**11.83**	**2.07**	**3.49**	**13.25**	**2.24**	**3.79**
200	**15**	**56.28**	**1.20**	**17.41**	**57.26**	**1.24**	**17.33**	**38.93**	**0.88**	**12.16**	**29.59**	**0.69**	**8.71**	**49.99**	**1.11**	**15.11**
200	**35**	**30.66**	**1.70**	**7.96**	**26.86**	**1.46**	**7.10**	**24.16**	**1.39**	**6.40**	**23.09**	**1.35**	**6.20**	**26.81**	**1.52**	**7.12**

200 50	**31.73**	**2.22**	**8.11**	**32.15**	**2.25**	**8.24**	**23.82**	**1.75**	**6.35**	**23.82**	**1.75**	**6.35**	**27.52**	**1.97**	**7.27**
300 15	**76.23**	**1.12**	**24.10**	**76.02**	**1.12**	**24.37**	**53.61**	**0.83**	**15.69**	**42.40**	**0.66**	**13.94**	**65.78**	**1.00**	**20.36**
300 35	**53.38**	**1.73**	**14.85**	**44.38**	**1.43**	**12.01**	**38.69**	**1.31**	**11.04**	**36.60**	**1.26**	**10.41**	**45.69**	**1.52**	**12.5**
300 45	**42.48**	**1.88**	**10.55**	**35.45**	**1.62**	**9.00**	**31.42**	**1.44**	**8.82**	**30.76**	**1.41**	**8.60**	**35.51**	**1.61**	**9.28**
400 15	**94.60**	**1.05**	**29.59**	**75.60**	**0.89**	**24.49**	**63.89**	**0.73**	**20.42**	**55.14**	**0.65**	**17.06**	**93.29**	**1.05**	**30.59**
500 55	**59.40**	**1.87**	**15.44**	**55.40**	**1.79**	**14.75**	**46.54**	**1.51**	**12.96**	**44.87**	**1.47**	**12.19**	**54.90**	**1.78**	**14.02**

表 3-5 和表 3-6 显示的是 ONBA 和其他的调度算法解决任务服务请求的调度问题得到的总时间开销情况和相关的负载情况。从上面的两个表中可以看出无论是在问题规模情况较大还是较小，ONBA 算法在时间开销和负载均衡上面也表现得更好。在任务数量和问题规模都逐渐增加的情况下，本文提出基于 ONBA 的调度算法能够在保持收敛速度的情况下，保持解的质量和稳定性，说明其能适应大规模的多目标优化问题的求解

ONBA 算法在保证收敛速度同时的具有较强的避免局部优化的能力。下面三个图是任务总数分别为 200，300，400 虚拟机数为 25,30,50 的情况下基于三种算法对应求得的完成时间随迭代次数增加的趋势图。

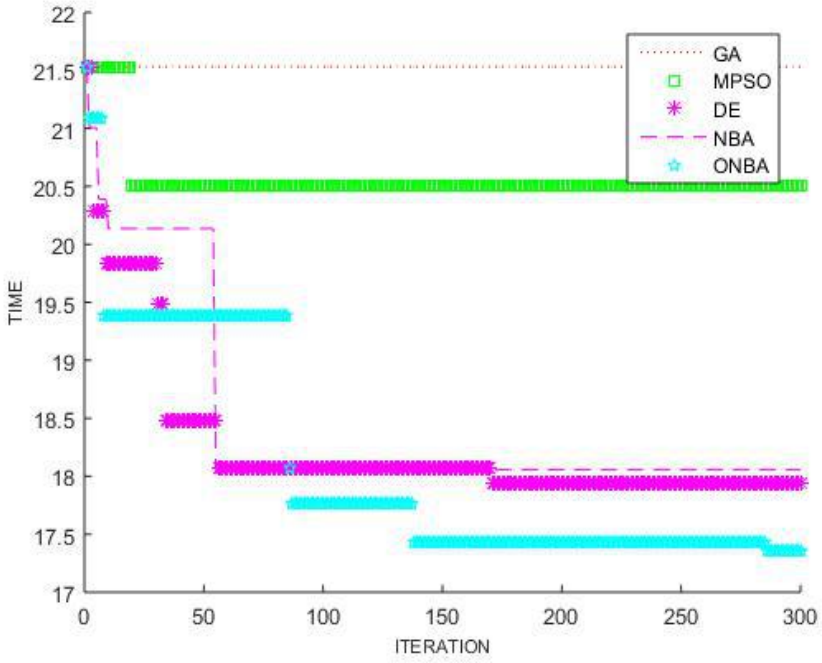

图 3-12　任务数为 100，虚拟机数 25

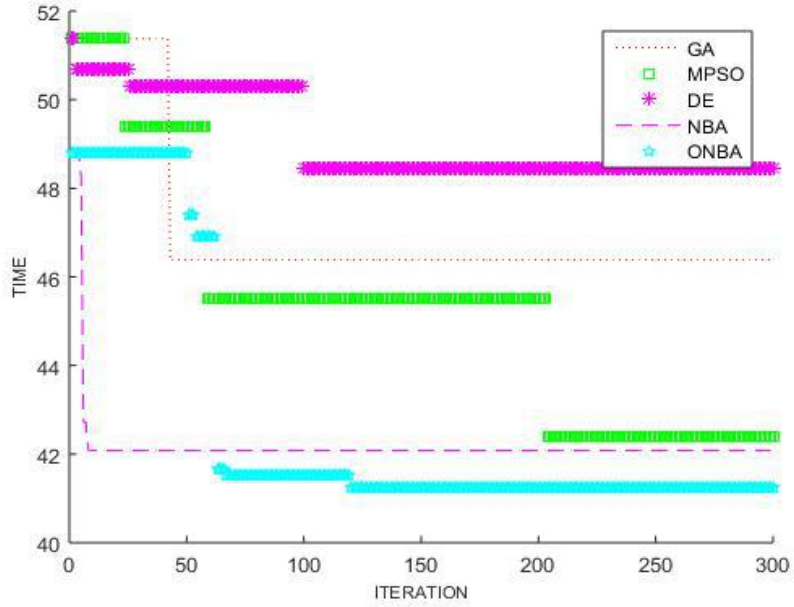

图 3-13　任务数为 300，虚拟机数为 30

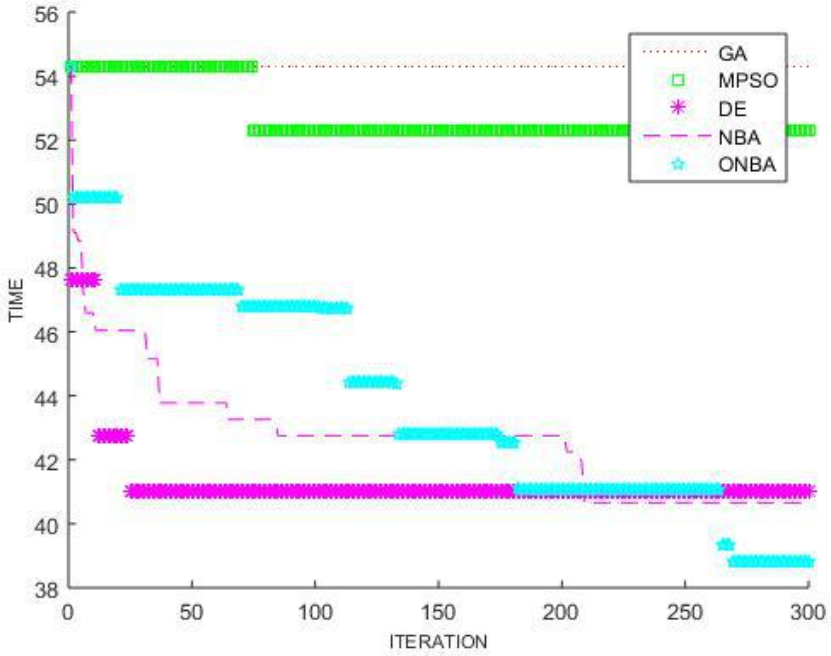

图 3-14　任务数为 400，虚拟机数为 50

从图中可以看出，五种算法中 ONBA 展现了较强的跳出前期局部优化的能力以及快速收敛能力。ONBA 算法在传统 NBA 算法的基础上加入了二阶振荡的环节增强全局搜索能力，并且通过引入差分进化算法还进一步丰富了种群的多样性。在迭代过程中出现停滞时，ONBA 算法可以及时扩展种群的多样性和优良性，跳出局部最优，同时这些改变也是保持解质量前提下迭代速度进一步提高的原因所在。ONBA 算法结合了 NBA 算法的局部搜索能力强，使用二阶振荡增强全局搜索能力以及差分进化算法的种群多样性等优点，

使得 ONBA 算法在防止局部最优解的同时能寻找到一个较好的解。同样的为了证明本文的算法也适用于真实的数据，我们使用卢森堡大学的数据进行试验，实验时我们分别取任务数为 50 , 100, 200, 400 虚拟机数量为 10, 20, 30,40 实验结果如下：

表 3-7：任务完成时间

m	n	GA	MPSO	NBA	ONBA	DE
50	10	183.319103	183.319103	158.841124	153.116220	177.561500
50	20	151.787075	147.318699	115.664795	94.558545	119.172422
100	10	200.873859	180.885285	143.434972	142.529186	169.214096
100	20	155.100828	154.598194	105.444207	100.986301	120.281764
100	40	113.597828	114.872948	103.457817	98.548762	114.987350
200	10	340.300831	335.732191	284.814059	261.522971	324.328093
200	30	133.638889	133.638889	98.708411	98.337935	122.841902
400	30	123.960215	123.960215	97.315464	91.486215	102.765319

表 3-8：负载均衡

		GA			MPSO			NBA			ONBA			DE		
m	n	LB1	LB2	LB3	LB1	LB2	LB3	LB1	LB2	LB3	LB1	LB2	LB3	LB1	LB2	LB3
50	**10**	**165.28**	**1.32**	**52.41**	**165.28**	**1.32**	**1.32**	**140.80**	**1.14**	**38.30**	**103.29**	**0.84**	**35.31**	**121.55**	**0.98**	**38.21**
50	**20**	**148.64**	**2.26**	**39.65**	**143.04**	**2.04**	**32.04**	**115.66**	**1.93**	**33.35**	**94.55**	**1.52**	**31.56**	**119.17**	**1.78**	**44.84**
100	**10**	**138.00**	**1.04**	**35.44**	**78.64**	**0.58**	**20.58**	**121.94**	**0.98**	**35.50**	**51.66**	**0.24**	**20.45**	**81.95**	**0.63**	**23.26**
100	**20**	**155.09**	**2.22**	**50.93**	**154.58**	**2.19**	**42.35**	**100.43**	**1.62**	**33.07**	**79.84**	**1.43**	**30.41**	**115.32**	**1.85**	**33.66**
100	**40**	**200.59**	**0.78**	**71.73**	**110.01**	**2.40**	**38.29**	**102.54**	**1.58**	**37.07**	**67.59**	**1.10**	**32.84**	**101.58**	**1.60**	**36.66**
200	**30**	**133.17**	**2.29**	**41.93**	**133.17**	**2.29**	**42.01**	**97.81**	**1.77**	**30.54**	**94.90**	**1.70**	**30.02**	**120.40**	**2.13**	**39.11**
400	**30**	**121.72**	**2.29**	**33.08**	**121.72**	**2.29**	**32.98**	**97.09**	**2.02**	**33.11**	**88.93**	**1.86**	**29.88**	**102.76**	**2.03**	**30.84**

从上面的两表可以看出无论是任务的完成时间还是负载均衡，本文提出的 ONBA 算法都要优于其它的四个算法。由于真实的调度数本质上是上面随机调度数据中的特例，而真实数据的调度结果也证明了本文提出的调度方法具有一定的实用性。

3.3.5 ONBA 和 ISBSA 算法对比分析

在求解上面提出的调度模型时，本文在 BSA 算法的基础上引入鸟群的飞入飞出行为，使得新的个体携带的信息能够在鸟群迁徙觅食时对群体的信息进行更新从而影响原有的个体信息以增强全局搜索能力，通过和差分进化算法的相结合使个体之间能够进行交叉变异从而增强种群的多样性。最终改进 BSA 算法原有的缺点。同样的在 ONBA 算法的基础

上本文结合二阶振荡算法和差分进化算法，在保留 NBA 算法的局部搜索能力的同时增强群体的全局搜索能力和群体的多样性以避算法免陷入局部最优解，从而优化原有的 NBA 算法。将 ISBSA 算法和 ONBA 算法进行对比，因为 ISBSA 算法中需要确定鸟群的警戒，觅食，迁徙等行为，算法的时间复杂度比 ONBA 的算法时间复杂度高，所以在求解问题时所用的时间也会多一些，先随机取虚拟机数分别为 20, 35 任务数从 50 增加到 400 时对算法的运行时间进行测试，如下：

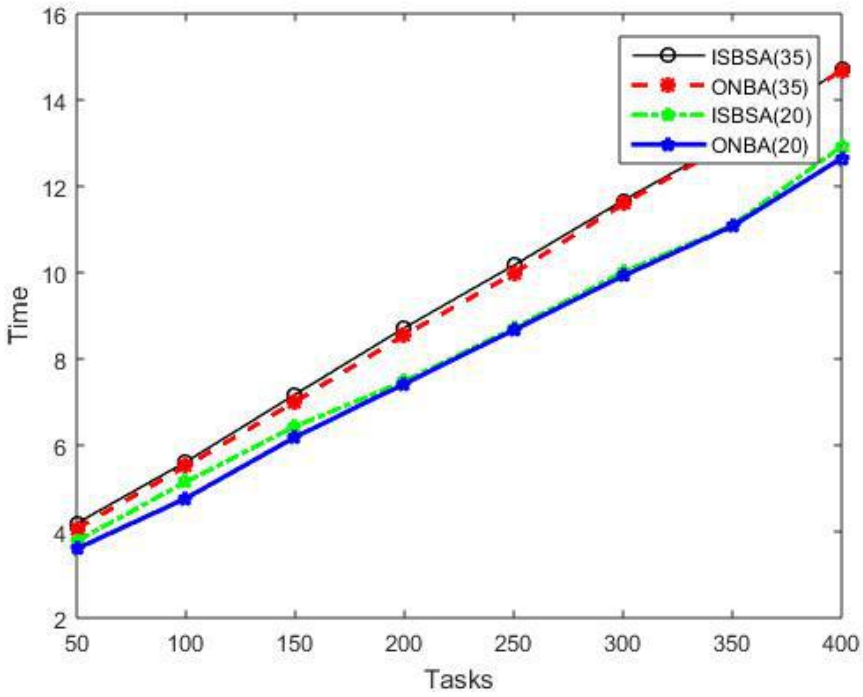

图 3-15　运行时间

从算法运行时间结果可以看出 ISBSA 算法运行时间要大于 ONBA 算法运行的时间，但是从上两节的仿真实验结果也可以看出，在求解任务执行时间最短的问题上面，ISBSA 算法求解的最短时间解的数要多于 ONBA，正是因为 ISBSA 在鸟类迁徙过程中的上述行为其部分的个体携带的信息很容易影响整个群体，因此种群的多样性较强，在求解最优解时效果较好，而在虚拟机负载均衡的问题上面进行对比时，能够发现 ONBA 算法无论在 LB1 还是 LB2，LB3 上表现的都比 ISBSA 算法要好，正是因为 ONBA 算法由于其二阶振荡的引入使得种群的搜索范围扩大，因此在求解负载均衡时的效果更好。通过以上分析观察我们还可以得出，尽管 ISBSA 算法的运行时间比 ONBA 算法的运行时间长，但是相差不大，同样的在求解任务执行最短时间和负载均衡上面两者的差异很小，所以当云计算系

统更注重任务的执行时间时可以使用 ISBSA 算法，当云计算系统更注重负载均衡时可以使用 ONBA 算法，但是如果要同时兼顾任务执行时间和负载均衡时 ISBSA 算法和 ONBA 算法都比传统的调度算法表现更好并且两种算法求解结果相差不大，因此两种算法选一种就行。

3.5 本章小结

本章针对云计算模型的研究提出了任务最短时间执行模型，该模型在保证虚拟机负载均衡的约束下使得用户提交的任务能在最短的时间内执行完成。其次在求解最短时间执行模型时，本章分别对 BAS 算法和 NBA 算法进行了分析和改进从而提出了 ISBSA 算法和 ONBA 算法并与传统的遗传算法，粒子群算法，差分进化算法进行实验对比，证明本章提出的改进算法在求解多目标问题时能够扩大种群的搜索范围，增加个体的多样性从而避免陷入局部最优解，在本章节的最后使用公开的调度数据集以验证本文提出的改进算法具有一定的实用性。

第 4 章 面向优化时效的改进深度学习模型

4.1 引言

上一章中主要是建立一个任务调度模型和对该模型衍生出的多目标问题进行求解，以实现分配的任务能在最短的时间内执行完毕，同时保证负载均衡，然而正如上面提到的要实现任务执行的最小时间跨度，云计算调度过程中不仅要考虑任务在虚拟机上的执行时间问题还要考虑调度算法运行时本身需要花费的时间问题，从上一章的实验结果可以看出本文提出的算法在求解调度模型的问题时，虽然任务的执行完成时间和负载均衡都要优于上面提到的其他所有算法，但是由于智能算法在进行求解的时候本身要通过不停的迭代从群体中找出最好的解，这一过程本身时间复杂度较高，由于本文还融合了 DE 算法进行了改进所以 ISBSA 和 ONBA 算的时间复杂度更高，所耗费的时间也更多。因此本文使用深度学习模型对任务的调度方案进行预测以替代智能算法，从而减小调度算法运行的时间开销。因为深度学习模型本身也是一个很复杂的模型，所以深度学习模型在虽然在训练的过程中也会耗费大量的时间和资源。因此改进深度学习 DBN 模型以加快模型的训练速度减小模型的训练时间对于整个云计算系统的调度时效优化意义重大。

4.2 RBM 训练改进

DBN 是 Hinton 教授在 2006 年提出的一种基于概率的深度学习网络模型，它是由多个受限玻尔兹曼机(RBM)堆叠后所形成的一种模型，先通过逐层贪婪无监督训练，然后经过有监督微调。最终实现模型的训练，对比传统神经网络参数初始化该方法提升了网络的训练速度，加强了网络的建模能力[79-81]。RBM 是 DBN 模型的主要组成部分，DBN 模型的训练主要就是通过训练 RBM 获取对应的权重等参数。

RBM 模型是由 Smolensky[82]在 1986 年提出的，由玻尔兹曼机(BM)进行改进而来的，和 BM[83]一样 RBM 也分为隐藏层和显示层，和 BM 不同的是 RBM 中同一层中的神经元之间无连接，隐藏层和显示层之间的神经元进行全连接。由于这一限制使得 RBM 拥有如下性质：当显示层中的神经元的状态已经确定时，隐藏层中神经元的激活条件互相独立，同理当给定隐藏层中的神经元的状态时，显示层中神经元的激活条件也是相互独立的。RBM 的结构图如下所示：

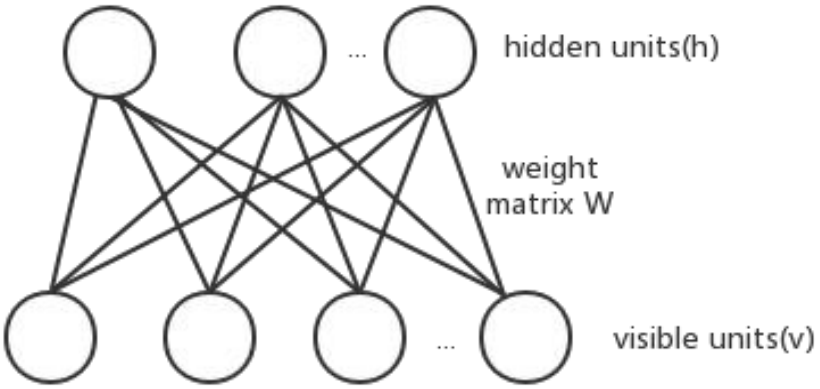

图 4-1　RBM 结构图

上图中的 v 表示显示层的神经元，h 表示隐藏层的神经元，文献[84]中指出上述的神经单元可以服从任意的分布，如高斯分布，泊松分布等，由于本文使用的神经元只有激活状态和未激活状态所以本文的神经元遵从二值分布，即 $v_i \in \{0,1\}$, $h_j \in \{0,1\}$ 其中 v_i,h_j 分别表示显示层中神经元 i 和隐藏层中神经元 j 的状态。

同 BM 一样，对于某一种特定状态(v,h)的 RBM 模型来说它的能量函数定义如下：

$$E_\theta(v,h) = -\sum_{i=1}^{m} b_i v_i - \sum_{j=1}^{n} c_j h_j - \sum_{i=1}^{m}\sum_{j=1}^{n} v_i W_{ij} h_j \tag{4-1}$$

上式中 m 表示显示层的神经元个数，n 表示隐藏层的神经元个数，其中θ为当前 RBM 的状态下的参数θ={W_{ij},c_j,b_i}，b_i 为显示层神经元 i 的偏执值，c_j 为隐层神经元 j 的偏执值，W_{ij} 为显示层神经元 i 与隐藏层神经元 j 之间的连接权重值。也可以使用如下矩阵的形式来定义：

$$E_q(v,h) = -b^T v - c^T h - v^T W h \tag{4-2}$$

其中 b^T 为可视层的偏置矩阵，c^T 为隐层的偏置矩阵，W 为对应的可视层到隐层的连接权值矩阵。基于上面的能量函数定义和受限玻尔兹曼机的性质可以推导出其状态下的(v,h)联合概率分布：

$$P_\theta(v,h) = \frac{e^{-E_\theta(v,h)}}{\sum_{v,h} e^{-E_\theta(v,h)}} \tag{4-3}$$

RBM 训练的一个重要的目标就是最大化数据 v 的边缘概率分布，其公式如下：

$$P_\theta(v) = \frac{\sum_h e^{-E_\theta(v,h)}}{\sum_{v,h} e^{-E_\theta(v,h)}} \tag{4-4}$$

给定数据训练集后，训练 RBM 就意味着要确定θ，以使得训练完成的模型能很好的拟合给定的训练集，也就是说，在该参数下由相应 RBM 表示的概率分布要尽可能地与训练样本表示的分布相符合，若给定的训练集为 T，则含有多个训练数据集的 RBM 训练目标就是最大化下面的似然函数：

$$L_\theta = \prod_{t=1}^{T} P_\theta(V^t) \tag{4-5}$$

从 RBM 的结构性质可知，当给定显示层 v 中的所有神经元状态时，隐层的第 j 个神经元的激活概率为：

$$P_\theta = (h_j = 1 \mid v) = \frac{1}{1+\exp(-c_j - \sum_{i=1}^{m} v_i W_{ij})} \tag{4-6}$$

同样的当给定隐层 h 中的所有神经元状态时，显示层的第 i 个神经元的激活概率为：

$$P_\theta(v_i = 1 \mid h) = \frac{1}{1+\exp(-b_i - \sum_{j=1}^{n} W_{ij} h_j)} \tag{4-7}$$

由于同一层中的神经元之间时相互独立的所以我们可以得到每一层的概率密度函数如下：

$$P(h \mid v) = \prod_{j=1}^{n} P(h_j \mid v) \tag{4-8}$$

$$P(v \mid h) = \prod_{i=1}^{m} P(v_i \mid h) \tag{4-9}$$

上面已经提到过 RBM 训练的目标就是要最大化似然函数(4-5)，对两边同时取对数可得：

$$\ln L_\theta = \sum_{t=1}^{T} \ln P_\theta(v^t) \tag{4-10}$$

由对数的单调性可知要最大化(4-5)就是要最大化式(4-10)，要最大化上面的似然函数就要找到一个θ来拟合输入的训练集，对上面的公式使用梯度上升法，通过迭代的方式进行逼近，可得θ的迭代公式如下：

$$\theta = \theta + \eta \frac{\partial \ln L_\theta}{\partial \theta} \tag{4-11}$$

上式中的η为一个给定的整数值，由上式可知要求解θ值就是要求解概率密度的似然函数对于各个变量的偏导数，式(4-10)可以改写成如下形式：

$$\ln L_\theta = \sum_{t=1}^{T} \left(\ln \sum_h \exp[-E_\theta(v^{(t)}, h)] - \ln \sum_v \sum_h \exp[-E_\theta(v, h)] \right) \tag{4-12}$$

此时对上式中(4-11)进行求偏导操作就是对式（4-12）进行求偏导，其过程如下：

$$\begin{aligned} \frac{\partial \ln L_\theta}{\partial \theta} &= \sum_{t=1}^{T} \left(\sum_h \frac{\exp[-E_\theta(v^{(t)}, h)]}{\sum_h \exp[-E_\theta(v^{(t)}, h)]} \frac{\partial\left(-E_\theta(v^{(t)}, h)\right)}{\partial \theta} - \sum_v \sum_h \frac{\exp[-E_\theta(v, h)]}{\sum_v \sum_h \exp[-E_\theta(v, h)]} \frac{\partial\left(-E_\theta(v, h)\right)}{\partial \theta} \right) \\ &= \sum_{t=1}^{T} \left(-\sum_h P_\theta(h \mid v) \frac{\partial E_\theta(v, h)}{\partial \theta} + \sum_v \sum_h P_\theta(v, h) \frac{\partial E_\theta(v, h)}{\partial \theta} \right) \end{aligned} \tag{4-13}$$

由于 T 为已经给定的训练集中的样本数，所以要求解上式中的偏导数的值就是要求解如下表达式：

$$\frac{\partial \ln L'_\theta}{\partial \theta} = -\sum_h P_\theta(h \mid v) \frac{\partial E_\theta(v, h)}{\partial \theta} + \sum_v \sum_h P_\theta(v, h) \frac{\partial E_\theta(v, h)}{\partial \theta} \tag{4-14}$$

上式中的 $P_\theta(h|v)$表示给定显示层 v 的神经元状态时隐藏层 h 中神经元的概率分布，由 RBM 的结构特征可知该项比较容易求解， $P_\theta(v,h)$表示显示层神经元和隐藏层神经元的联合概率分布，该分布比较难以直接计算，只能通过采样的方式进行间接计算

$$\sum_v \sum_h P_\theta(v,h) \frac{\partial E_\theta(v,h)}{\partial \theta} = \sum_v P_\theta(v) \sum_h P_\theta(h \mid v) \frac{\partial E_\theta(v,h)}{\partial \theta} \tag{4-15}$$

由于显示层已知，求解上式就是要计算出$\sum_h P_\theta(h \mid v) \frac{\partial E_\theta(v,h)}{\partial \theta}$，又因为θ是关于 W_{ij},c_j,b_i 的参数因此对其求偏导就是对 W_{ij},c_j,b_i 分别进行求偏导：

$$\sum_h P(h \mid v) \frac{\partial E(v,h)}{\partial W_{ij}} = -\sum_h P(h \mid v) h_i v_j = -P(h_i = 1 \mid v) v_j \tag{4-16}$$

$$\sum_h P(h \mid v) \frac{\partial E(v,h)}{\partial b_i} = -\sum_h P(h \mid v) v_i = -v_i \tag{4-17}$$

$$\sum_h P(h \mid v) \frac{\partial E(v,h)}{\partial c_j} = -P(h_j = 1 \mid v) \tag{4-18}$$

将上面的三个公式代入到式(4-14)中可推到出如下的求解公式：

$$\frac{\partial \ln L(\theta)}{\partial W_{ij}} = P(h_i \mid v)v_j - \sum_v P(v)P(h_i = 1 \mid v)v_j \tag{4-19}$$

$$\frac{\partial \ln L(\theta)}{\partial b_i} = v_i - \sum_v P(v)v_i \tag{4-20}$$

$$\frac{\partial L(\theta)}{\partial c_j} = P(h_j = 1 \mid v) - \sum_v P(v)P(h_j = 1 \mid v) \tag{4-21}$$

上面三个公式中都会涉及到对 $\sum_v P(v)$ 的计算，由于该项计算起来太复杂，时间复杂度太高因此不宜通过直接计算得到，而是通过采样的方法进行近似的计算。经典的近似计算方法有：随机近似法(SAP)[85], Gbbis 采样[86], 对比散度(CD)[87], 比例匹配[88]，状态翻转变换[89]，其中 CD 算法是最经典且被广泛使用的算法。

CD 算法是 2002 年由 Hinton 提出，该算法是在 Gibbs 采样的基础上提出的一种改进算法，该算法提出的初衷是为了克服 Gibbs 采样算法的训练效率差的缺点。在训练 RBM 时 CD 算法需要实施 K 步（通常情况下 K=1）Gibbs 采样，在 CD 算法中当给定显示层的训练样本 v(0)时可以根据公式(4-6)计算出所有隐藏层中神经元被激活和未被激活的概率，以此确定该层神经元的状态，使用公式(4-7)计算显示层神经元 i 的激活概率以实现显示层的重构，然后求解(4-19)-(4-21)，因此上面三式可转化为以下三式进行求解：

$$\frac{\partial \ln L(\theta)}{\partial W_{ij}} = P(h_i = 1 \mid v^{(0)})v_i^{(0)} - P(h_i = 1 \mid v^{(k)})v_i^{(k)} \tag{4-22}$$

$$\frac{\partial \ln L(\theta)}{\partial b_i} = v_i^{(0)} - v_i^{(k)} \tag{4-23}$$

$$\frac{\partial L(\theta)}{\partial c_j} = P(h_j = 1 \mid v^{(0)}) - P(h_j = 1 \mid v^{(k)}) \tag{4-24}$$

CD 算法的基本思想为：对于训练集首先使 $v(0)=v$, 接着进行 K 步 Gibbs 采样，最后利用概率公式 P(h(k-1)|v(k-1))获取隐藏层在 k-1 步时的状态 h(k-1)，利用公式 P(v(k)|h(k-1))获取重构的显示层在 k 步时的状态 v(k)，具体算法如下：

Input: K,Sample s,n,m,RBM(W,b,c)

Output:ΔW,Δb,Δc

Initialize ΔW=0,Δb=0,Δc=0

for v
    v(0)=v;
    for t=0,1,…K-1
        for i=1,2,…n
            if rand(0,1)<P(h_i=1|v)
                h_i(t)=1;
            else
                h_i(t)=0;
            end if
        end for
        for j=1,2,…m
            if rand(0,1)<P(v_j=1|h)
                v_j(t+1)=1;
            else
                v_j(t+1)=0;
            end if
        end for
    for i=1,2,…n
        for j=1,2,…m
            $\Delta W_{ij}=\Delta W_{ij}+[P(h_i=1|v^{(0)})v_j^{(0)}-P(h_i=1|v^{(K)})v_i^{(K)}]$;
            $\Delta b_j=\Delta b_j+[v_j^{(0)}-vj^{(K)}]$;
            $\Delta c_i=\Delta c_i+[P(h_i=1|v^{(0)})-P(h_i=1|v^{(K)})]$;
        end for
    end for
end for
end

使用 CD 算法输出的ΔW, Δb, Δc 后紧接着进行 RBM 的训练，RBM 的训练就是使用 CD 算法的输出结果不断地调整显示层和隐层之间的权重、偏置等参数，具体的算法过程如下：

Initialize W,b,c,learning rate η,iterations m
  for t=1,2,…m
      using the CD algorithm to obtain ΔW,Δb,Δc;
      update:W=W+ηΔW,b=b+ηΔb,c=c+ηΔc;
  end for
end

上述算法中η为 RBM 权重和偏置在训练过程中的学习率，在传统训练方法中该学习率要么保持不变，要么需要手动进行调整，而且学习率的设置和调整主要依靠训练人员的

经验，从而增加训练人员的工作量。尽管 CD 算法应用广泛，但是因为每一次采样时 Gibbs 链都是用样本数据进行初始化，因此会减小训练速度和精度，并且 CD 算法并不是传统意义上的梯度下降算法，所以 Tieleman 提出持续对比散度(PCD)[90]和快速持续对比散度(FPCD)[91]，Eesjardings 等人则提出并行回火 MCMC(PTMCMC)[92]等方法对 CD 算法进行改进。这些方法虽然在 RBM 的训练上做出了优越的改进，在训练精度和收敛速度上有了很大提高，但是上述算法复杂度太高实现起来较困难。RBM 本质上是一个概率神经网络，学习率对模型的训练有着重要的影响，但是 RBM 和神经网络一样，学习率的设置难点在于如果设置的太大训练速度虽然会增加但是可能会引起震荡，学习率如果设置的过小虽然会克服震荡的现象但是收敛速度慢。在针对神经网络学习率的问题方面文献[93-96]提出根据网络训练的重构误差结果对网络中的学习率进行自适应地增大或减小以避免固定学习率导致的缺点，文献[97]提出将自适应学习率方法引入 CD 算法中使得学习率随着 RBM 连续两次迭代后参数更新的方向进行增加或减小，文献[98]定义了 RBM 训练阶段的重构误差并根据重构误差的结果对学习率进行自适应的减半或保持不变等操作。上述方法中学习率的都是线性的变化虽然在一定程度改进了模型的训练速度和训练效果但是线性变换的学习率在求解参数减小误差方面存在搜索能力不强、局限性较大等缺点。综合上面的改进方法本章结合本文的调度模型数据的特点对 RBM 的学习率进行简单的改进使之呈非线性变化，以达到在训练调度数据时能更快的收敛从而减小因深度学习模型训练而占用的资源，保证云计算环境中的资源尽可能的用于执行用户提交的任务。参考文献[98]本文使用的重构误差公式如下：

$$rcerror = \sum_{i=1}^{m} (v_i^{(0)} - v_i^1)^2 / m \tag{4-25}$$

其中 $v^{(0)}$为显示层中神经元的状态，$v^{(1)}$为 CD 算法中的采样后的显示层神经元的状态，上述改进算法中学习率的改进中可以用以下公式进行大致的表示

$$\eta(t+1) = \begin{cases} \alpha\eta(t), error(t) < error(t-1) \\ \beta\eta(t), error(t) > error(t-1) \end{cases} \tag{4-26}$$

其中α为一个(1,2)之间的常量，β为一个(0,1)之间的常量，t 为迭代次数，error(t)为当前误差，error(t-1)为上一次迭代误差，当 error(t)<error(t-1)时以固定的速率增加学习率以增加模型的训练速度，当 error(t)>error(t-1)时以一定的速率减小学习率以确保模型的训练精度。实验结果表明虽然学习率能够根据实际情况进行动态调整从而实现加快误差的收敛程度，但是仍然很难满足寻优要求。因此在此基础上，根据 RBM 的重建误差进行动态的调

整学习率，公式如下：

$$\eta(t+1)=\begin{cases}\alpha\sqrt{1+e(t)/t}\eta(t), error(t)<error(t-1)\\ \beta\eta(t), error(t)>error(t-1)\end{cases} \tag{4-27}$$

其中 e(t)为当前误差，α为一个(1,2)之间的常量，β为一个(0,1), 学习率的调整过如下：

(1)error(*t*)<error(*t*-1)则说明随着 t 的增加误差呈下降趋势，此时适当的增加η有助于提高误差的下降速度，由于 error(t)训练次数呈现下降的趋势而 t 不断的上升，因此η的增加率呈下降的趋势，这样就避免因η增加过快而越过误差的极小值。

(2)当 error(t)>error(t-1)则说明误差很有可能越过了极小值此时减小 η 以使得误差继续减小直至收敛。

该模型中学习率的改变不再是按照固定的速率进行增加而是根据模型训练后的误差进行调整，并且由于模型训练时的重构误差直接参与调整，使的学习率的改变具有一定的自适应性，由于当学习率处于增加阶段时模型的重构误差在不断地下降导致整个学习率不断的下降，学习率缓慢的增加以使得误差改变的幅度缓慢减小以实现比较精确的调整。

4.3 模型训练次数的改进

DBN 模型的训练除了需要训练每一层的 RBM 外还会有一个 BP 的微调阶段，模型在训练时无论是 RBM 的训练阶段还是微调的阶段都会有多次的迭代，模型的训练速度和精度都与模型的训练次数是分不开的，如果模型的训练次数设置的太小则在训练次数内模型无法收敛，如果模型的训练次数设置的太大则会增加模型的训练时间。通常情况下训练次数 M 的值是由训练人员根据经验进行设置，为了保证模型在训练时能够收敛，M 通常需要反复的调整，这样使得训练人员的工作量成倍的增加。

结合本文提出的学习率的调整方法，现在对模型的训练次数也进行控制，在实际训练中我们结合误差和学习率来控制模型的训练次数。由上面的参数更新公式可知当η的值减小到一个极小值后模型的参数基本不变，模型的误差下降幅度极小，整个模型的误差范围在可接受范围内所以此时再增加 t 已经没有意义，则应该停止训练。另外考虑到 t 可能还没有下降到设置的范围时模型已经收敛，所以训练误差也是控制模型训练次数的因素之一。当模型满足以下两个条件(condition)之一时就停止训练：

$$\begin{cases}\eta(t)<\partial\\ \left|\beta(t-n)-\beta(t-n-1)\right|\le\alpha(n=1,2\quad)\end{cases} \tag{4-28}$$

其中∂为一个设置的阈值，b 为模型的训练误差，n 为连续统计误差变化率次数。a 为

一个误差波动变化率范围，取值为(0,1)。由上面的公式可知当$\eta>\partial$，连续的相连误差超出波动范围时，模型进行迭代训练以减小误差, 当$\eta<\partial$时误差下降幅度小，此时停止训练，当连续几代的误差变化率在α 范围内波动时基本认为模型收敛也停止迭代。

这样无论哪一个条件都能控制模型的 t。在训练时就无需人工反复的调整。当有一个条件不满足是模型就停止训练从而节约训练时间。

RBM 训练更多的时候是一次生成ΔW,Δb,Δc 而不是不断的调用 CD 算法，本文使用的 RBM 训练方法也是在直接生成ΔW,Δb,Δc 的方法上进行改进并结合本章节提出的训练次数控制提出改进的 RBM(IRBM)训练过程如下：

Initialize W,b,c,learning rate η,iterations m，
reconstruction error e(t),N,n,∂,pre,lr=η(0);

While $\eta(t) > \partial$ &&n<N&&t<m do

for all hidden units i do
compute $P(h_{1i}=1|v_1)$ with (4-6)
sample $h_{1i}\in\{0,1\}$ from $P(h_{1i}|v_1)$
end for
for all visible units j do
compute $P(v_{2j}=1|h_1)$ with (4-7)
sample $v_{2j}\in\{0,1\}$ from $P(v_{2j}=1|h_1)$
end for
for all hidden units i do
compute $P(h_{2i}=1|v_2)$ with (4-6)
end for
$W=W+\eta[h_1v_1^T- P(h_2=1|v_2)v_2^T]$;
$b=b+\eta[v_1-v_2]$;
$c=c+\eta[h_1-P(h_2=1|v_2)]$;
compute reconstruction error e(t);

$\eta(t)$=lr;

if $|e(t)-err|<\partial$
n++;

update $\eta(t)$ with (4-27);

else
n=0;

update $\eta(t)$ with (4-27);

end if

lr=$\eta(t)$;

pre=e(t);

```
        t++;
end while
end
```

上述算法中 v_1 为 RBM 输入的训练数据集，h_1 为根据输入的数据集而计算出的隐藏层神经元的状态，v_2 为根据隐藏层神经元的状态重构的显示层神经元的状态。在上一章节学习率的改进上集合本章节的训练次数控制条件以减小RBM的训练次数从而加快模型的训练速度。

4.4 BP 算法中学习率的改进

DBN 深度学习模型的训练第一步 RBM 训练完成后，需要用带标签的数据对整个网络进行微调，微调阶段就是神经网络中的 BP 阶段，DBN 的 BP 阶段和神经网络的 BP 阶段一样都是使用梯度下降算法对权重值进行调整以最小化，在神经网络中模型的损失函数定义如下：

$$E = \frac{1}{2}\sum_{t=1}^{m}\left|x_i - y_i\right|^2 \tag{4-29}$$

上式中 m 为训练集中的样本数量，x 训练的样本经过神经网络后的输出值，y 为样本所对应的标签值，BP 的目的就是要最小化 E，要最小化 E 就是要找到一组合适的参数 θ={W,b}使得网络输出结果和标签最接近，根据梯度下降算法权重和偏执的更新公式如下：

$$\theta = \theta - \eta\nabla E(\theta) \tag{4-30}$$

上式中^{h} 为学习率，$\nabla E(\theta)$ 为参数对应的梯度。由于梯度下降算法每更新一次都要输入所有样本，因此当训练数据很多时非常耗时，所以许多用于改进梯度下降算法的方法分别被提出[99-104]，梯度下降法中对误差影响最大的就是学习率的取值问题，学习率过小，则会导致收敛速度很慢。学习率过大，会影响模型的收敛性能使其发生震荡。文献[105]提出一种 AdaGrad 算法，对每一个参数使用不同的学习率以提高模型的训练精度，文献[106]则改进了 Adadelta 算法对 AdaGrad 方法中的学习率下降过快的问题进行了改进以提高模型的训练速度，文献[107]则提出了 Adam 算法利用梯度的一阶矩估计和二阶矩估计对学习率进行调整。文献[108]则提出在 AdaGrad 算法的基础上进行改进的 AdaDec 算法，对学习率进行动态的调整，以加快模型的训练速度。尽管上面的方法都对学习率进行了动

态的调整也取得了较好的结果但是上面的这些方法在改变学习率的时候都是不断地减小学习率，这样会使得模型的收敛速度越来越慢。

AdaGrad 算法中学习率变化可以用如下公式表示：

$$\eta(t+1)=\frac{\eta(0)}{\sqrt{K+\sum_{t=1}^{m}\nabla g(t)^2}} \tag{4-31}$$

t 为模型训练的迭代次数，$\eta(0)$位学习率的初始值，K 为一个定值，$\nabla g(t)$为梯度，从上面的公式中可以看出随着训练次数的增加梯度也不断地累加，因此学习率不断的减小。并且学习率的减小速度越来越快。

AdaDec 算法中学习率的变换公式如下：

$$\eta(t+1)=\frac{\eta(0)\left(1+\frac{t}{R}\right)^{-q}}{\sqrt{K+G(t)}} \tag{4-32}$$

$$G(t)=\phi\nabla g(t-1)^2+\nabla g(t)^2 \tag{4-33}$$

该公式中 R 为最大迭代次数，q 为常数，ϕ为一个常数，可以看出该公式通过对历史最近的梯度进行累加，以保证减小学习率。

上面我们已经分析了学习率的设置对模型训练的影响，较大的学习率会加快模型的收敛速度因此结合上一章中 RBM 训练时的学习率的设置方法以及 AdaDec 公式中的关于学习率的减小方法，使用误差对学习率进行自适应调整以使得学习率能够根据训练的误差进行增加或减小以加快收敛速度，学习率的改进公式如下：

$$\eta(t+1)=\begin{cases}\eta(t)\left(1+\frac{\sqrt{e(t)}}{t}\right), e(t)<e(t-1)\\ \frac{\eta(t)}{\sqrt{1+G(t)}}, e(t)>e(t-1)\end{cases} \tag{4-34}$$

$$G(t)=\varsigma E(g(t-1)^2)+g(t)^2 \tag{4-35}$$

上述公式中 e(t)为当前模型训练的误差，t 为迭代次数，ς 为一个(0,1)之间的定值，$E(g(t-1)^2)$为上一次迭代梯度平方和的均值，学习率的调整过程如下：

(1)当 e(t)<e(t-1)说明当前误差值小于上一次误差，误差随着迭代的增加在不断的减小。由于随着 t 的增加，e(t)呈下降趋势所以学习率的增加速度减慢。使得梯度下降时步长增加速度减小以避免越过极小值。

(2)当 e(t)>e(t-1)时说明当前的学习率太大导致梯度下降太快，因此应当减小学习率以减小梯度下降的步长。

在进行梯度下降算法的时候模型也会进行循环的迭代，知道模型的误差收敛，因此反向调整同样面临着训练次数的问题，因此在进行梯度下降算法时也结合上一节中的模型训练次数控制方法对梯度下降进行训练次数控制。算法如下：

Initialize W,b,c,learning rate η,iterations m，
error e(t),N,n, ∂ ,pre,lr=η(0);

while $\eta(t) > \partial$ &&n<N &&t<m do

compute e(t)
compute current gradient g(t)2,pre=E(g(t-1)2);
update θ
$\eta(t) = lr$
If |e(t)-err|< ∂

update $\eta(t)$ with (4-34);

n++;
else

update $\eta(t)$ with (4-34),(4-35);

n=0;
end if
err=e(t)

lr= $\eta(t)$;

t++;
end while
end

上述算法较为简单，就是不断的根据当前的误差，决定是否增减或减小学习率，以减小模型的训练次数从而达到加快训练速度的目的。

4.5 深度学习模型改进仿真

为了验证本章节中提出的 RBM 改进算法和改进的梯度下降算法，本文使用上一章节中的调度数据经过归一化后的值对本节中的算法在 matlab 上进行仿真实验。在实际试

验中本文分为两部分。一部分主要针对训练 RBM 时的问题验证本文提出的训练方法(IRBM)和传统的训练方法(RBM)以及上述中的线性变化的训练方法(PRBM)进行对比。第二部分主要是针对模型在进行反向调整时的训练误差，使用本文提出的方法(IGD)和传统的训练方法(GD), 以及 AdaDec 算法进行对比。

在 RBM 训练中我们使用的初始学习率为 1，误差的波动范围为 0.1，初始训练值为 50，n 为 5。其 RBM 的训练次数和重构误差如下图所示：

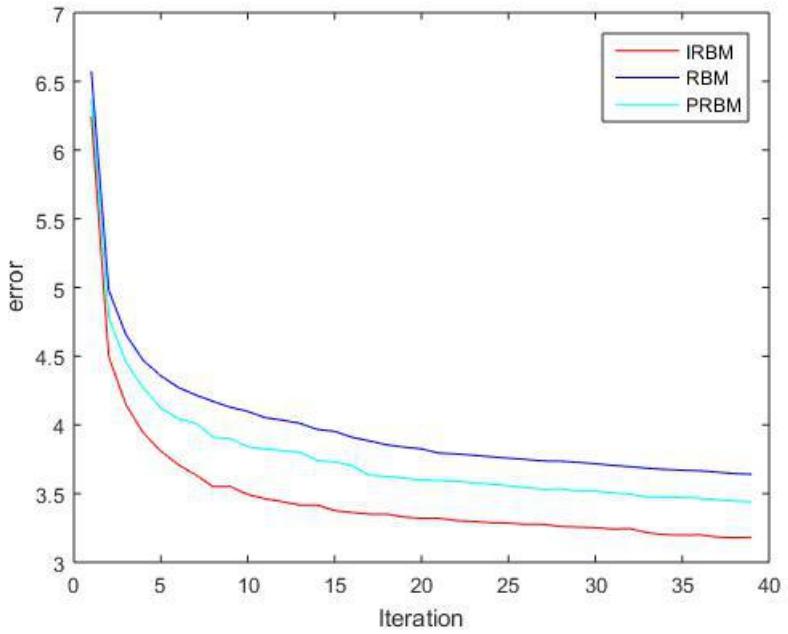

图 4-2　IRBM 训练误差

从上图中可以看出虽然我们设置的最大训练次数为 50 次，但是实际训练次数在 40 左右就已经停止，并且我们还能看出 IRBM 中后面模型在训练时的波动很小而 RBM 和 PRBM 中模型依然呈现下降的趋势，这说明本文提出的对于模型训练次数的控制方法却是能够减小模型的训练次数，现对 IRBM 进行继续训练使得训练次数达到最大的训练次数并和传统的 RBM 训练方法和 PRBM 训练方法进行对比，其结果图如下：

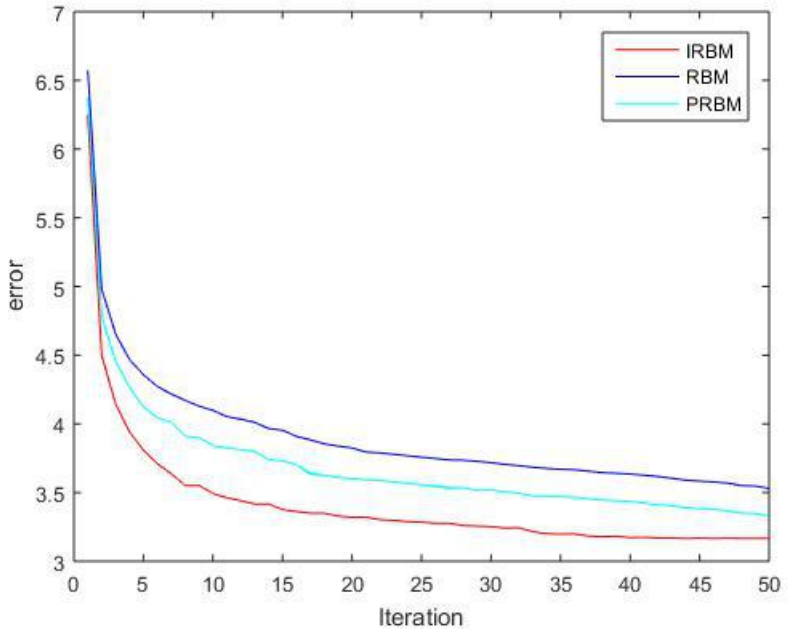

图 4-3　模型训练 50 次误差对比

从上图中可以看出即使模型训练到 40 次以后误差的波动范围很小但是 RBM 和 PRBM 的误差确实是随着训练次数的增加误差仍然在下降，此时我们可以认为随着迭代次数的增加，IRBM 中的误差基本保持不变，也即 40 次以后的迭代是不必要的，和传统的 RBM 训练算法对比我们可以看到 PRBM 的误差虽然也在减小并且比 RBM 的误差小，但是误差结果仍然要大于 IRBM 的误差。尽管随着模型的不断迭代训练，其误差在不断地减小，但是 RBM 和 PRBM 在相同的训练次数内其误差仍大于 IRBM 的误差并且当 IRBM 误差基本不变时 RBM 和 PRBM 的训练误差仍呈下降趋势，这说明本文提出的算法确实对 RBM 模型的训练起到了加速的作用。

在进行反向微调时，我们设置的误差波动范围为 0.0001，整个模型微调时的训练迭代次数设置初始值为 100，连续统计误差次数为 10，模型的训练结果如下图所示：

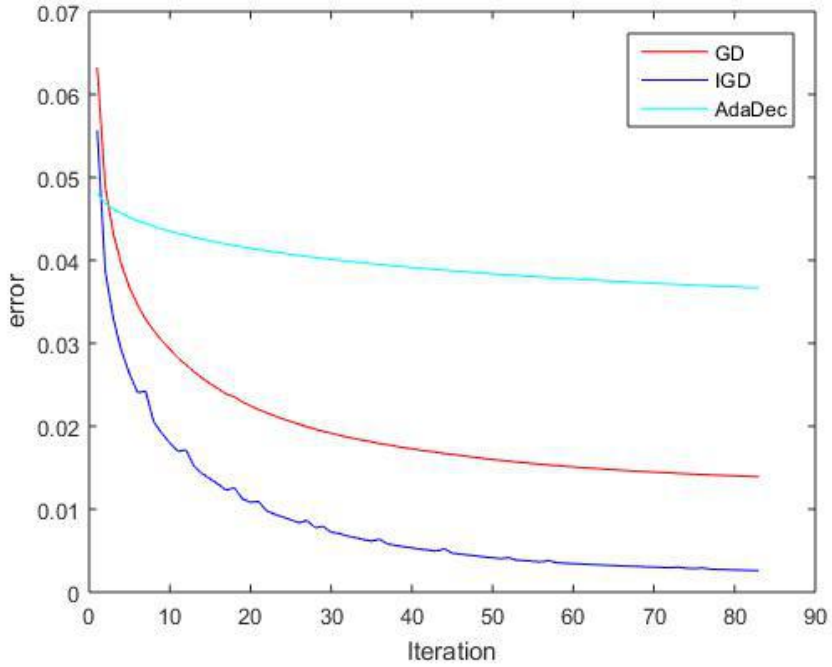

图 4-4　模型微调时训练误差

同样的从上图中可以看出虽然我们设置的最大训练次数为 100 次，但是实际训练次数在 80 多次时就已经停止，并且越往后模型越趋向于平稳，这也说明误差波动越来越小。现对模型的训练次数不加控制使其训练大最大的迭代次数，结果如下图所示：

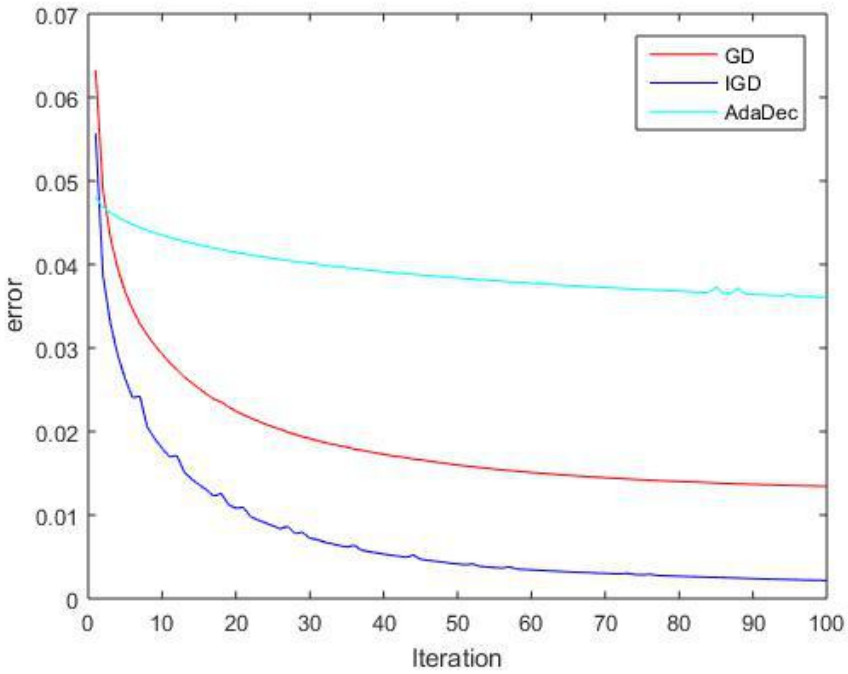

图 4-5　训练 100 次的误差对比

从上图中可以看出 80 多次以后模型的基本保持稳定，这也说明后里面的训练可以不

用进行，模型的训练条件对模型的训练次数起到的控制的作用，同时对比 GD 和 AdaDec 算法在相同的训练次数内模型的误差更低，这也说明模型的误差下降速度要更快，GD 和 AdaDec 算法中的误差下降曲线也可以从侧面反映出本文中学习率的增加确实能加快误差的下降速度。

4.6 本章小结

本章主要针对 DBN 模型的训练过程进行部分改进，模型无论是在前向训练 RBM 还是反向调整时的学习率和误差都是紧密关联的，针对这一个特点本文提出自适应的学习率调整算法以加快模型训练时的误差减小速度，针对模型训练过程中的迭代次数问题本文结合学习率的调整特点对模型的训练次数进行控制以减小不必要的迭代以达到减少模型的训练时间加快训练速度的目的。

第 5 章　实验仿真及结果分析

5.1　引言

尽管我们已经针对已有的启发式智能算法的缺点进行了改进，但是由于启发式算法的本质仍然是通过不断的循环迭代来求解一个最优解，算法的循环次数以及粒子的个数这些都会消耗大量的时间，从而延长任务的结束时间，不利于用户体验。本文使用历史的任务调度数据来训练深度学习模型，当模型训练完毕以使用其来预测任务的分配以替换调度算法从而减小任务分配所消耗的时间达到改进用户体验的目的。

5.2 深度学习预测调度仿真

为了验证本文的提出的深度学习预测算法，同样的本文在 matlab 上进行仿真实验，为了评估模型的预测效果，我们使用的误差为：平均绝对误差(MAE)，平均相对误差(MRE)和均方根误差(RMSE)来衡量本文的模型的预测值和真是之间的差距，其公式如下：

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|f_i - \hat{f}_i\right| \tag{5-1}$$

$$MRE = \frac{1}{n}\sum_{i=1}^{n}\frac{\left|f_i - \hat{f}_i\right|}{f_i} \tag{5-2}$$

$$RMSE = \left[\frac{1}{n}\sum_{i=1}^{n}\left(\left|f_i - \hat{f}_i\right|\right)^2\right] \tag{5-3}$$

其中 f_i 为数据的真实值 $\hat{f}_i$ 为预测值，n 为预测的数据大小。本文中使用改进的 DBN 模型对调度数据的特征进行提取并使用 RBF 进行预测，本文结构图如下：

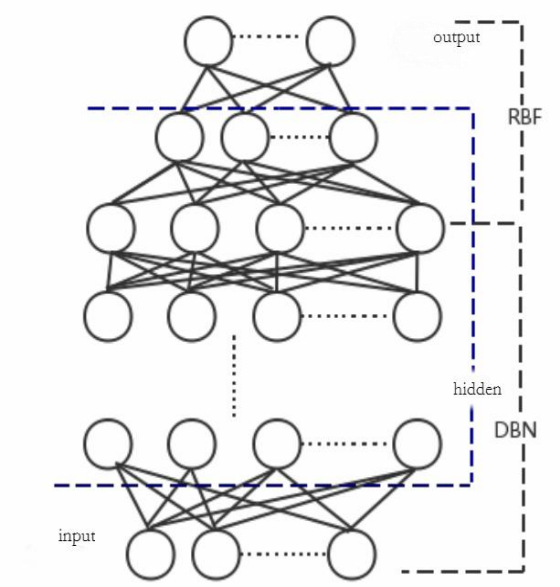

图 5-1　结构图

为了验证预测模型的适用性本文分别使用 ONBA，ISBSA，PSO 的调度数据对模型进行训练，因任务的长度，内存需求等数据的变化范围较大所以在训练模型前本文使用归一化处理，将调度数据的范围归一化到(0,1)之间，并使用训练完成的模型进行预测验证，验证的数据集中任务数 m 为 200，300，400，虚拟机数 n 为 20，35，45，60。

5.2.1　深度学习对于 ONBA 调度结果的预测

为了验证本文提出的基于深度学习的预测方法能够准确的预测任务的调度数据，以及通过训练完成的深度学习模型预测任务调度时所用的时间比 ONBA 短，本文分别使用任务数 200，300,400，虚拟及数量为 20,45 进行测试，其预测结果部分图如下：

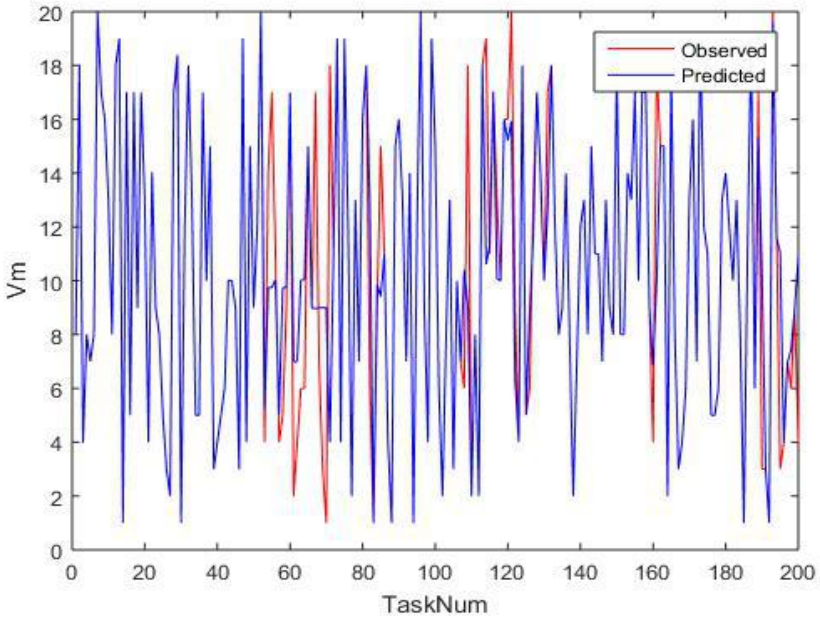

图 5-2　任务数 200，虚拟机数 20

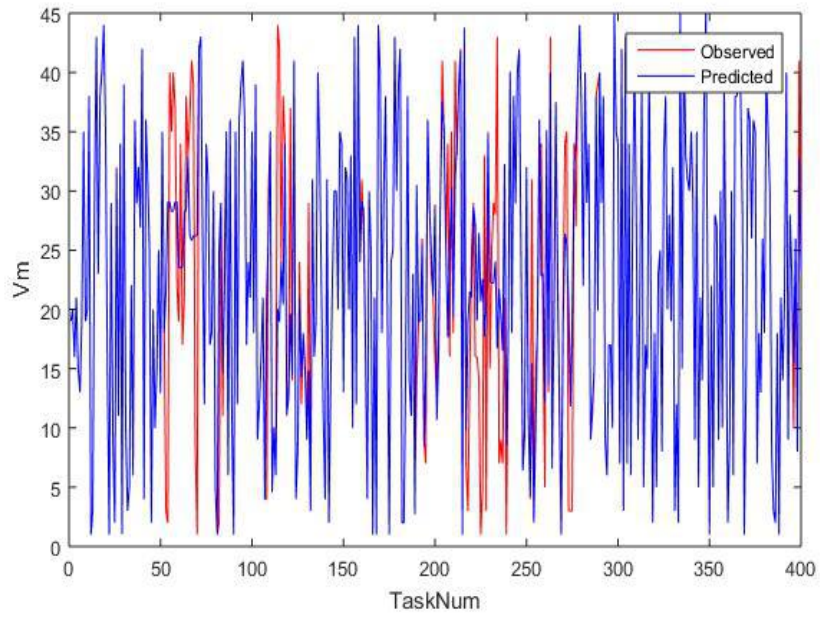

图 5-3　任务数为 400，虚拟机数为 45

表 5-1：误差表如下

m	n	MAE	MRE	RMSE
200	20	0.866468	0.090576	2.311344
200	45	2.072890	0.102661	5.721576
300	20	1.319155	0.134961	2.781888

300	45	2.072890	0.102661	5.721576
400	20	1.004769	0.102852	2.421846
400	45	2.169998	0.112233	5.514179

表 5-2：ONBA 和 IDBN 对于任务调度用时对比

m	n	ONBA（单位：秒）	IDBN（单位：秒）
200	20	5.202336	0.468154
200	45	6.964589	0.685310
300	20	7.023486	0.631514
300	45	9.363982	0.492413
400	20	8.783663	0.814632
400	45	12.107095	0.899639

上面的预测结果图中 Predicted 表示使用深度学习模型进行预测的结果，Observed 表示使用的是 ONBA 进行调度的结果，从结果对比图和误差表中可以看出预测的结果和 ONBA 调度的结果偏差很小，这说明深度学习模型精准的预测了任务的调度情况，从任务的调度时间对比表中发现当任务数相同时 IDBN 对任务的调度进行预测时所用的时间要远远小于 ONBA 调度所用的时间，结合时间表和误差表进行分析得出深度学习的预测结果和 ONBA 的调度结果之间的误差很小，但是所用的时间上面却要远远小于 ONBA 算法运行所用的时间，这说明使用深度学习预测的方法能够提高任务的调度时效。尽管从两表中还可以看出随着任务数量和虚拟机数的增加，模型在进行预测时对应的误差也会相应的增加，造成这种现象是因为任务和数量虚拟机数量的增加，数据间特征组合的可能性也在呈几何倍数的增加，导致了预测的难度增加，尽管模型的预测误差值在增加但是总体上看误差增加的幅度并不大并且最终的误差仍然在一个很小的范围内，这一较小的任务调度误差值相对于任务的数量和使用 ONBA 进行调度所需的时间角度来说可以忽略不计。所以基于本文的使用深度学习代替 ONBA 算法确实是能优化调度的时效性。

5.2.2 深度学习对于 ISBSA 调度结果的预测

为了验证本文提出的基于深度学习的预测方法能够准确的预测任务的调度数据，以及通过训练完成的深度学习模型预测任务调度时所用的时间比 ISBSA 短，本文分别使用任务数 200，300, 400，虚拟及数量为 35, 60 进行测试，其预测结果部分图如下：

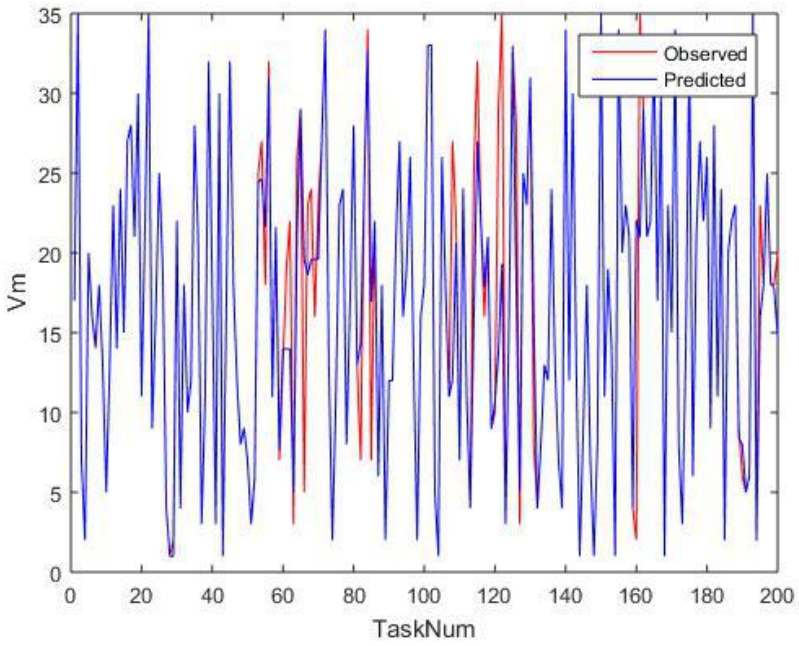

图 5-4　任务数为 200，虚拟机数为 35

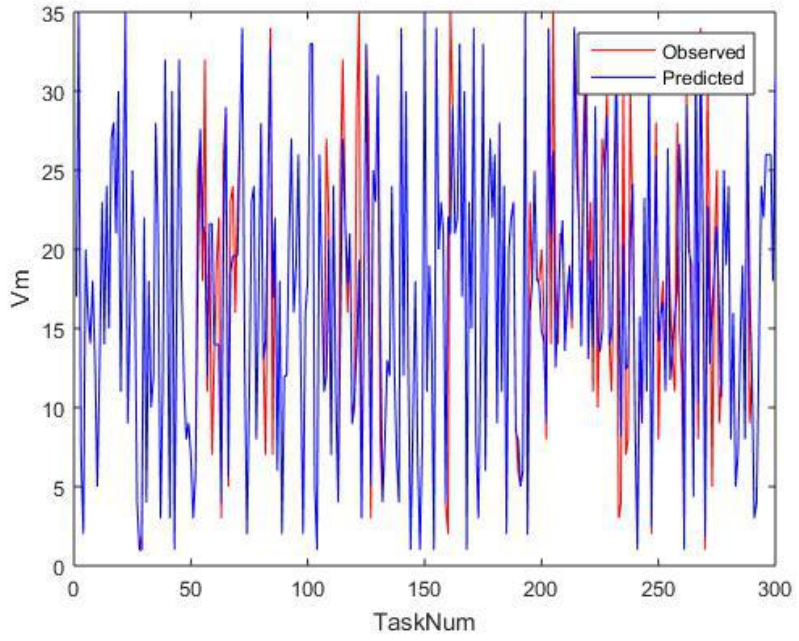

图 5-5　任务数为 300，虚拟机数为 35

表 5-3：误差表

m　　n	MAE	MRE	RMSE
200　　35	1.069422	0.071142	3.338031

200 60	1.817851	0.060905	5.362979
300 35	1.606828	0.105068	3.738363
300 60	3.173390	0.119269	6.849517
400 35	2.085470	0.108547	4.052864
400 60	3.5258746	0.142548	6.892456

表 5-4：ISBSA 和 IDBN 调度时间对比

m n	ISBSA（单位：秒）	IDBN（单位：秒）
200 35	8.822134	0.453304
200 60	11.788601	0.419309
300 35	12.222762	0.669537
300 60	15.682579	0.556462
400 35	14.457853	0.715485
400 60	16.384569	0.852469

和上一节一样预测结果图中 Predicted 表示使用深度学习模型进行预测的结果，Observed 表示使用的是 ISBSA 进行调度的结果，从结果对比图和误差表中可以看出预测的结果和 ISBSA 调度的结果相接近，说明深度学习模型精准的预测了任务调度，从调度的时间对比表中可以看出 IDBN 对任务进行预测时所用的时间要远远小于 ISBSA 调度所用的时间，结合时间表和误差表来看，深度学习的预测结果和 ISBSA 的调度结果误差很小，但是在所使用的时间上面却要远远小于 ISBSA 所用的时间，尽管从两表中还可以看出随着任务数量和虚拟机数的增加，模型在进行预测时对应的误差也会相应的增加，造成这种现象是因为任务和数量虚拟机数量的增加，数据间特征组合的可能性也在呈几何倍数的增加，导致了预测的难度增加，尽管模型的预测误差值在增加但是总体上看误差增加的幅度并不大并且最终的误差仍然在一个很小的范围内，这一较小的任务调度误差值相对于任务的数量和使用 ISBSA 进行调度所需的时间角度来说可以忽略不计。所以基于本文的使用深度学习代替 ISBSA 算法确实是能优化调度的时效性。

5.2.2 深度学习对于传统调度结果的预测

为了证明本文提出的调度算法具有一定的适用性，而不是仅仅针对本文提出的 ONBA

算法和 ISBSA 算法有效，先对传统的 PSO 调度算法获取的数据进行预测，其结果如下：

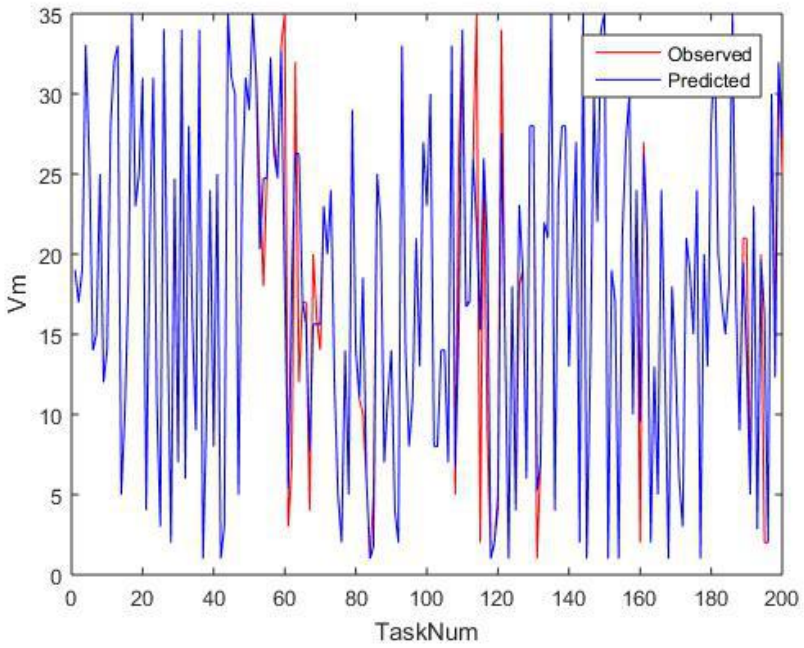

图 5-6　任务数为 200，虚拟机数为 35

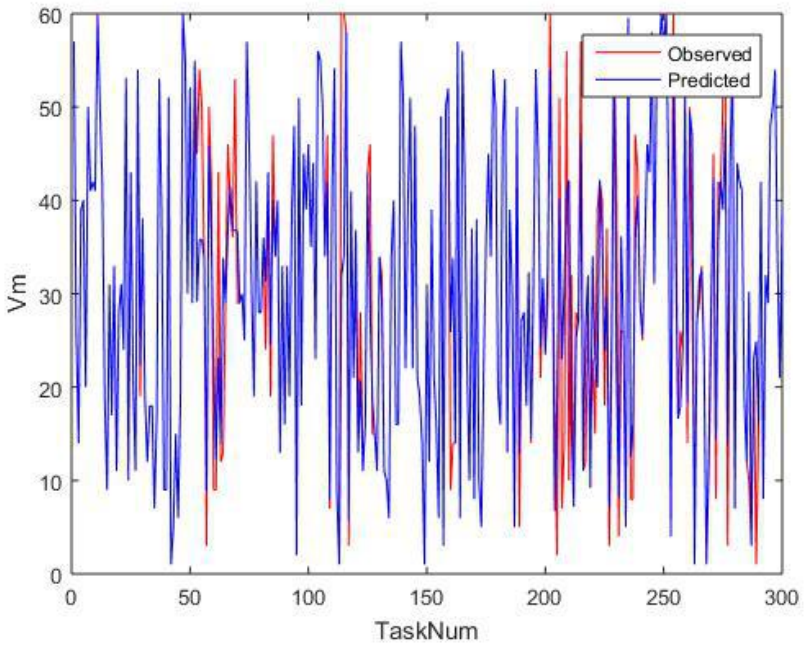

图 5-6　任务数为 300，虚拟机数为 60

表 5-5：误差表

m　　n	MAE	MRE	RMSE

200 35	0.960148	0.068999	3.039599
200 60	1.447235	0.053878	4.511500
300 35	1.487031	0.114056	3.483470
300 60	2.633169	0.116344	6.145972
400 35	1.756824	0.126857	3.752468
400 60	3.154896	0.135846	6.854976

表 5-6：PSO 和 IDBN 调度时间对比

m n	PSO（单位：秒）	IDBN（单位：秒）
200 35	3.733594	0.469225
200 60	4.621450	0.437560
300 35	5.235160	0.740911
300 60	6.112598	0.600144
400 35	5.879542	0.795426
400 60	8.647487	0.835489

从上面的仿真图和误差表可以看出深度学习模型对于任务调度预测时能够获取很好的预测结果，从中还可以看出虽然随着任务数量和虚拟机数的增加，模型在进行预测时对应的误差也会相应的增加，这是因为任务和虚拟机数的增加，数据间特征组合可能性也在呈几何倍数的增加，从而导致预测的难度也会增加，此外还能得出虽然模型的误差值也在增加但是总体上看误差增加的幅度和最终的误差在一个可接收的范围内。

5.3 本章小结

本章主要对前两章的算法进行了结合以及对本文提出的使用深度学习模型对云计算法中的任务的调度预测进行验证，同时实验证明，当模型训练完成后，如果有新的任务到达，使用 IDBN 模型对调度进行预测所用的时间和传统的 ONBA，ISBSA，PSO 的调度方法进行求解时所使用的时间缩短了 10 倍左右，从中我们还能看出 ONBA 算法和 ISBSA 算法求解相同的规模的任务时所需要的时间要比 PSO 所需的时间更长，这是因为这些算法为了避免求解问题时陷入局部最优解而使用了更为复杂的搜索方法消耗了更多的时间，结合前面的预测结果来看本文提出的深度学习模型在保证调度质量的前提下减小调度算

法本身所需的执行时间是有一定的实际应用价值的。

第 6 章　总结与展望

6.1　总结

随着大数据，人工智能的掀起人们对于云计算服务的处理任务的能力有着越来越高的要求，其中当任务提交到云计算集群后能否在最短的时间内获取到处理后的结果成为用户选择云计算服务的重要原因之一，云计算服务中如何在最短的时间内将任务调度到虚拟机上执行以及虚拟机如何在最短的时间内将分配的任务执行完毕成为提高调度优化时效领域中亟待解决的问题，这对这一问题，建立了最短任务执行时间调度模型，然后提出使用改进的 ONBA 算法和 ISBSA 算法求解该模型以获取任务最短执行时间的调度数据，并使用该数据对深度学习模型进行训练以预测任务的调度情况，从而避免用户以后提交的任务再次经过调度模型进行求解，以节约任务的调度时间提高任务的调度时效。论文主要做了一下研究：

1.针对云计算任务调度建立了最短任务执行时间调度模型，在确保用户提交的任务执行完毕所用的时间最短的同时保证虚拟机的负载均衡。

2.在对调度模型转化的多目标问题进行求解时在传统的鸟群算法基础上引入差分遗传算法并根据鸟群中个体的飞入飞出群体的行为，提出改进的鸟群算法(ISBSA)。在传统的新蝙蝠群算法的基础上引入了动态因子，二阶震荡以及差分进化算法的三种改进方式，提出了改进的新蝙蝠群算法(ONBA)，求解本文提出的多目标优化问题，并对这两种改进的算法进行仿真以验证在求解多目标问题时这两种算的优点。同时对两种改进算法的求解结果进行对比分析。

3.在改进深度学习模型方面，本文提出的自适应学习率算法和传统的算法对比不再只是保持学习率不变或者简单的减少学习率,而是根据训练误差情况分为自适应的增加或下降。其中自适应增长阶段用来增加学习率以加快模型的训练速度。下降阶段用来减小学习率以使模型收敛。此外本文所提出的模型训练次数控制方法能有效的控制因模型收敛而出现的不必要的训练。学习率自适应和训练次数控制方法分别从模型训练速度和模型迭代次数等方面结合，最终使得整个模型的训练时间缩短。

4.最后，论文对前面提出的改进的鸟群算法和改进的新蝙蝠群算法进行多目标任务调度模型的求解以获取调度数据集，并使用数据集训练 IDBN 模型，之后使用训练完成的 IDBN 模型对任务的调度结果进行预测并和传统的调度方法在时间上进行对比，以验证本

文提出的方法对于云计算调度优化时效的改进具有实际应用价值。

6.2 展望

虽然本文在对云计算环境中任务的调度时间做了优化但是仍然存在以下不足：

1.本文提出的任务调度模型还有待进一步的完善，在实际应用中影响任务执行速度的因素有很多如：带宽，能量功耗，磁盘容量等因素，用户选择云服务处了时间之外还有安全因素，经济因素等一些情况需要考虑纳入本文的模型之中。

2.本文提出的 ISBA 算法和 ONBA 算法在求解多目标问题时需要耗费更长的时间，并且算法后期的搜索能力还有待加强，因此这两个算法还有待进一步的改进。

3.深度学习中学习率的改进和训练次数的控制方法虽然在本文提出的调度数据验证中有较好的效果，但是对于其他的数据集的训练情况还有待验证，并且学习率的改进方法有待改进，模型训练到最后是否收敛到最小值还有待验证。

4.对于深度学习用来进行预测时使用不同的调度算法的数据集以及虚拟机的数量改变好深度学习模型需要重新进行训练以适应新的环境，因此如何增强深度学习模型对于云计算环境的适应性还需要研究。

参 考 文 献

[1] 周天君. 基于 Hadoop 的网络海量数据采集及处理平台开发[D]. 北京邮电大学, 2012.

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

[3] Uhlig R, Neiger G, Rodgers D, et al. Intel virtualization technology [J]. Computer, 2005, 38(5): 48-56.

[4] Singh S, Chana I. A Survey on Resource Scheduling in Cloud Computing: Issues and Challenges [J]. Journal of Grid Computing, 2016, 14(2):217-264.

[5] Verma A, Pedrosa L, Korupolu M, et al. Large-scale cluster management at Google with Borg [A]. Tenth European Conference on Computer Systems[C]. New York: ACM, 2015, 21-24

[6] Song H, Chang S B, Lee J W, et al. Utility adaptive service brokering mechanism for personal cloud service [A]. 2011 IEEE International Conference On Military Communications Conference [C]. USA:IEEE, 2011, 1622-1627.

[7] Bedra A. Getting Started with Google App Engine and Clojure [M]. IEEE Educational Activities Department, 2010.

[8] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters [J]. Communications of the Acm, 2008, 51(1):10-10.

[9] Ghemawat S, Gobioff H, Leung S T. The Google file system [A]. Nineteenth ACM Symposium on Operating Systems Principles[C]. New York: ACM, 2008, 29-43.

[10] Matthews J, Garfinkel T, Hoff C, et al. Virtual machine contracts for datacenter and cloud computing environments [A]. Proceedings of the 1st workshop on Automated control for datacenters and clouds [C]. New York: ACM, 2009, 25-30.

[11] Vouk M, Averitt S, Bugaev M, et al. Powered by VCL-using virtual computing laboratory (VCL) technology to power cloud computing [A]. Proceedings of the 2nd International Conference on Virtual Computing (ICVCI)[C]. NC: RTP, 2008, 15-16.

[12] 田文洪，赵勇. 云计算: 资源调度管理[M]. 北京: 国防工业出版社，2011

[13] Fang Y, Wang F, Ge J. A task scheduling algorithm based on load balancing in cloud computing,Web Information Systems and Mining. Springer Berlin Heidelberg[A]. Web Information Systems and Mining[C]. Germany: Springer, Berlin, Heidelberg , 2010, 271-277.

[14] Juhnke E, Dornemann T, Bock D, et al. Multi-objective scheduling of BPEL workflows in geographically distributed clouds [A]. IEEE International Conference on Cloud Computing[C]. USA:

IEEE, 2011, 412-419.

[15] Frincu M E, Craciun C. Multi-objective meta-heuristics for scheduling applications with high availability requirements and cost constraints in multi-cloud environments [A]. 2011 IEEE International Conference On Utility and Cloud Computing[C]. USA:IEEE, 2011, 267-274.

[16] Oliveira D, Ogasawara E, Ocaña K, et al. An adaptive parallel execution strategy for cloud‐based scientific workflows [J]. Concurrency and Computation: Practice and Experience, 2012, 24(13): 1531-1550.

[17] Pandey S, Wu L, Guru S M, et al. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments[A]. 2010 IEEE international conference on Advanced information networking and applications[C]. USA: IEEE, 2010, 400-407.

[18] Wu Q, Zhao Y. A cost-effective scheduling algorithm for scientific workflows in clouds [A]. 2012 IEEE International Conference On Performance Computing and Communications Conference[C]. USA: IEEE, 2012, 256-265.

[19] Fard H M, Prodan R, Fahringer T. A truthful dynamic workflow scheduling mechanism for commercial multicloud environments[J]. IEEE Transactions on Parallel and Distributed systems, 2013, 24(6): 1203-1212.

[20] Chen W N, Zhang J. A set-based discrete PSO for cloud workflow scheduling with user-defined QoS constraints,Systems[A]. 2012 IEEE International Conference on Man, and Cybernetics[C]. USA: IEEE, 2012, 773-778.

[21] Li W, Zhang Q, Wu J, et al. Trust-Based and QoS Demand Clustering Analysis Customizable Cloud Workflow Scheduling Strategies [A]. 2012 IEEE International Conference on Cluster Computing Workshops [C]. USA: IEEE, 2012, 111-119.

[22] Spitz S, Bök P B, Tüchelmann Y. Trust-based resource allocation and evaluation of workflows in distributed computing environments [A]. 2010 IEEE International Conference on Software Technology and Engineering [C]. USA:IEEE, 2010.

[23] 李建锋，彭舰. 云计算环境下基于改进遗传算法的任务调度算[J]．计算机应用,2011,31(1) : 184-186．

[24] 汤小春，刘健．基于元区间的云计算基础设施服务的资源分配算法研究[J].计算机工程与应用，2010，46(34) : 237-241．

[25] 华夏渝，郑骏，胡文心. 基于云计算环境的蚁群优化计算资源分配算法[J]．华东师范大学学报: 自然科学版，2010，1(1) : 127-134．

[26] 罗南超. 云计算中多服务器服务能力优化调度仿真[J].计算机仿真,2018(01):382-385.

[27] Ebadifard F, Babamir S M. Optimizing multi objective based workflow scheduling in cloud computing

using black hole algorithm[A]. 2017 IEEE International Conference on Web Research [C]. USA: IEEE, 2017, 102-108.

[28] 何晓珊, 孙贤和, Laszewski G V. QoS Guided Min-Min Heuristic for Grid Task Scheduling [J]. Journal of Computer Science and Technology, 2003, 18(4):442-451.

[29] Etminani K,Naghibzadeh M. A Min-Min Max-Min selective algorihtm for grid task scheduling [A]. Internet, 2007　IEEE/IFIP International Conference in Central [C]. USA: IEEE, 2007, 2-7.

[30] 杨武军, 郝凯. 基于贪心改进算法的云计算任务调度[J]. 传感器与微系统, 2016(12):143-145.

[31] 李靖, 乔蕊, 刘志中,等. 结合对策论与多目标 MILP 的 Web 服务组合调度问题求解[J]. 计算机工程, 2016, 42(1):11-17.

[32] 刘美林. 云计算中基于博弈论的任务调度算法研究 [D]. 北京工业大学, 2014.

[33] Dashti S E, Rahmani A M. Dynamic VMs placement for energy efficiency by PSO in cloud computing[J]. Journal of Experimental & Theoretical Artificial Intelligence, 2016, 28(1-2): 97-112.

[34] Alkayal E S, Jennings N R, Abulkhair M F. Efficient Task Scheduling Multi-Objective Particle Swarm Optimization in Cloud Computing [A]. 2016 IEEE Conference on Local Computer Networks Workshops[C]. USA: IEEE, 2016, 17-24.

[35] Sabar N R, Song A. Grammatical evolution enhancing simulated annealing for the load balancing problem in cloud computing [A]. Proceedings of the 2016 on Genetic and Evolutionary Computation Conference[C]. New York: ACM, 2016, 997-1003.

[36] 袁晓林,施化吉. 基于模拟退火算法的云计算资源调度模型[J]. 软件导刊,2015,(02):68-70.

[37] 徐文忠,彭志平, 左敬龙.基于遗传算法的云计算资源调度策略研究[J]. 计算机测量与控制,2015,(05):1653-1656.

[38] 李建锋,彭舰. 云计算环境下基于改进遗传算法的任务调度算法[J]. 计算机应用,2011,31(1): 184-186.

[39] 黄伟建,郭芳. 基于烟花算法的云计算多目标任务调度[J]. 计算机应用研究,2017,34(6): 1-5.

[40] 吴国芳.云环境中基于布谷鸟搜索算法的多目标任务调度方案[J]. 计算机应用研究, 2015,32(9): 2674-2677.

[41] Krizhevsky A, Sutskever I, Hinton G E. ImageNet classification with deep convolutional neural networks [J]. Advances in neural information processing systems, 2012, 25(2): 1097-1105.

[42] Tompson J, Jain A, Lecun Y, et al. Joint Training of a Convolutional Network and a Graphical Model for Human Pose Estimation [J]. Eprint Arxiv, 2014:1799-1807.

[43] Hinton G, Deng L, Yu D, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups [J]. IEEE Signal Processing Magazine, 2012, 29(6): 82-97.

[44] Sainath T N, Mohamed A R, Kingsbury B, et al. Deep convolutional neural networks for LVCSR [A].

2013 IEEE International Conference on Acoustics, Speech and Signal Processing[C]. USA:IEEE, 2013, 8614-8618.

[45] Junshui, Ma,Robert P, Sheridan,Andy, Liaw,George E, Dahl,Vladimir, Svetnik.Deep neural nets as a method for quantitative structure-activity relationships [J].Journal of chemical information and modeling,2015,55(2):263-74.

[46] Helmstaedter M, Briggman K L, Turaga S C, et al. Connectomic reconstruction of the inner plexiform layer in the mouse retina [J]. Nature, 2013, 500(7461):168-74.

[47] Xiong H Y, Alipanahi B, Lee L J, et al. The human splicing code reveals new insights into the genetic determinants of disease [J]. Science, 2015, 347(6218):1254806.

[48] Ciodaro T, Deva D, De Seixas J M, et al. Online particle detection with Neural Networks based on topological calorimetry information [A]. Journal of Physics Conference Series[C]. USA:IOP, 2012, 1453-1466.

[49] Ioffe S, Szegedy C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift [A]. Proceeding ICML'15 Proceedings of the 32nd International Conference on International Conference on Machine Learning Computer Science [C]. New York:ACM, 2015.

[50] Dean J, Corrado G S, Monga R, et al. Large scale distributed deep networks [A]. International Conference on Neural Information Processing Systems[C]. USA: Curran Associates Inc. 2012, 1223-1231.

[51] Duchi J, Hazan E, Singer Y. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization [J]. Journal of Machine Learning Research, 2011, 12(7):257-269.

[52] Senior A, Heigold G, Ranzato M, et al. An empirical study of learning rates in deep neural networks for speech recognition [J]. 2013, 22(4):6724-6728.

[53] 刘凯,张立民,范晓磊.改进卷积玻尔兹曼机的图像特征深度提取[J].哈尔滨工业大学学报,2016, 48(5):155-159.

[54] 赵志勇,李元香,喻飞等.基于极限学习的深度学习算法[J]. 计算机工程与设计, 2015(4):1022-1026.

[55] 杨春德,张磊. 基于自适应深度置信网络的图像分类方法[J]. 计算机工程与设计, 2015(10):2832-2837.

[56] Chen C L P, Zhang C Y, Chen L, et al. Fuzzy Restricted Boltzmann Machine for the Enhancement of Deep Learning [J]. IEEE Transactions on Fuzzy Systems, 2015, 23(6): 2163-2173.

[57] Syulistyo A R, Purnomo D M J, Rachmadi M F, et al. Particle swarm optimization (PSO) for training optimization on convolutional neural network (CNN)[J]. Jurnal Ilmu Komputer dan Informasi, 2016, 9(1): 52-58.

[58] Chang Y C, Chang R S, Chuang F W. A Predictive Method for Workload Forecasting in the Cloud

Environment [M] Advanced Technologies, Embedded and Multimedia for Human-centric Computing. Springer Netherlands, 2014:577-585.

[59] Qiu F, Zhang B, Guo J. A deep learning approach for VM workload prediction in the cloud [A]// 2016 IEEE International Conference on Software Engineering, Artificial Intelligence, NETWORKING and Parallel/distributed Computing[C]. USA: IEEE, 2016, 319-324.

[60] Xu D, Yang S, Luo H. A Fusion Model for CPU Load Prediction in Cloud Computing [J]. Journal of Networks, 2013, 8(11):2506-2511.

[61] Mao H, Alizadeh M, Menache I, et al. Resource Management with Deep Reinforcement Learning [A]// ACM Workshop on Hot Topics in Networks[C]. New York: ACM, 2016, 50-56.

[62] 云计算白皮书(2012 年)[M]. 工业和信息化部电信研究院,2012.

[63] 陈全,邓倩妮.云计算及其关键技术[J]. 计算机应用,2009,29(09):2562-2567.

[64] Munir E U, Li J, Shi S. QoS Sufferage Heuristic for Independent Task Scheduling in Grid [J]. Information Technology Journal, 2007, 6(8):1166-1170.

[65] Hayes-Roth F. Review of "Adaptation in Natural and Artificial Systems by John H. Holland", The U. of Michigan Press, 1975 [M]. ACM, 1975.

[66] Eberhat R, Kennedy J. A new optimizer using particle swarm theory [A] . Procof 6th Int Symposium on Mico Machine and Human Science[C]. USA:IEEE, 1995, 39-43.

[67] Storn R, Price K. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces [J]. Journal of global optimization, 1997, 11(4): 341-359.

[68] 武志峰.差异演化算法及其应用研究[D]. 北京交通大学, 2009.

[69] Hinton G E, Salakhutdinov R R. Reducing the dimensionality of data with neural networks [J]. science, 2006, 313(5786): 504-507.

[70] Dashti S E, Rahmani A M. Dynamic VMs placement for energy efficiency by PSO in cloud computing [J]. Journal of Experimental & Theoretical Artificial Intelligence, 2016, 28(1-2):97-112.

[71] 吕相文,袁家斌,张玉洁.云计算环境下多 GPU 资源调度机制研究[J]. 小型微型计算机系统,2016, 37(4):687-693.

[72] 冯小靖,潘郁.云计算环境下的 DPSO 资源负载均衡算法[J].计算机工程与应用,2013, 49(6):105-108.

[73] Meng X B, Gao X Z, Lu L, et al. A new bio-inspired optimisation algorithm: Bird Swarm Algorithm [J]. Journal of Experimental & Theoretical Artificial Intelligence, 2015(17): 20-22.

[74] 刘晓龙,宁芊,赵成萍,涂樺.基于莱维飞行的鸟群优化算法[J].计算机测量与控制,2016,24(12):194-197.

[75] 刘剑飞,王少影,曾祥烨,卢嘉,王蒙军.基于群智能算法的光 OFDM 系统 PAPR 抑制[J]. 光学学报,2017,37(01):100-107.

[76] Xu C, Yang R. Parameter estimation for chaotic systems using improved bird swarm algorithm[J]. Modern Physics Letters B, 2017, 31(36): 1750346.

[77] Yang X S. A new metaheuristic bat-inspired algorithm[M] Germany: Springer, Berlin, Heidelberg, 2010: 65-74.

[78] 简琤峰,王斌,张美玉,等.一种求解云服务组合全局QoS最优问题的改进杂交粒子群算法[J]. 小型微型计算机系统, 2017, 38(7):1562-1567.

[79] Y. Bengio. Learning Deep Architectures for AI [J]. Foundations and Trends in Machine learning, 2009, 2(1): 1-127.

[80] Y. Bengio, P. Lamblin, D. Popovici, et al... Greedy Layer-Wise Training of Deep Networks [C]. Advances in Neural Information Processing Systems, 2006, 19: 153-160

[81] Hinton G E. Learning to represent visual input [J]. Philosophical Transactions of the Royal Society B: Biological Sciences, 2010, 365(1537): 177-184.

[82] Smolensky. Information processing in dynamical systems: foundations of harmony theory [A]. Colorado Univ At Boulder Dept Of Computer Science[C]. USA: MIT Press, 1986,194-281.

[83] Hinton, G. E, Sejnowski, T. J. Learning and relearning in Boltzmann machines [A]. Colorado Univ At Boulder Dept Of Computer Science[C]. USA:MIT Press, 1986:45-76.

[84] Welling M, Rosen-Zvi M, Hinton G. Exponential family harmoniums with an application to information retrieval [A]. International Conference on Neural Information Processing Systems [C]. India: MIT Press, 2004, 1481-1488.

[85] NEAL R M.Connectionist learning of belief networks [J]. Artificial Intelligence, 1992, 56 (1):71-113

[86] Liu J S. Monto Carlo Strategies in Scienctific Computing [M]. New York: Springer-Verlag, 2001.

[87] G. E. Hinton. Training products of experts by minimizing contrastive divergence [J]. Neural Computation, 2002, 14(8): 1771-1800.

[88] Hyvärinen A. Some extensions of score matching [J]. Computational Statistics & Data Analysis, 2007, 51(5):2499-2512.

[89] Kai B, Fischer A, Igel C. The flip-the-state transition operator for restricted Boltzmann machines [J]. Machine Learning, 2013, 93(1): 53-69.

[90] T.Tieleman. Taining restricted Boltzmann machine using approximations to the likelihood gradient [A]. Proceedings of the 25th International Conference on Machine Learning [C]. New York: ACM, 2008, 1064-1071

[91] Tieleman T,Hinton G E. Using fast weights to improve persistent contrastive divergence[A].In Procedings of International Conference on Machine Learning(ICML) [C] . USA: ACM, 2009, 1033-1040.

[92] Eesjardings G, Courville A., Bengio Y,Vincent P, Dellaleau O. Parallel tempering for training of restricted Boltzmann Machines [A]. In Procedings of JMLRW&CP:AISTATS [C],2010, 145-152.

[93] 江国荐,顾乃杰,张旭,任开新.基于 SAE-LBP 的网页分类研究[J]. 小型微型计算机系统,2016,37(04):738-742.

[94] 张素芹.机器人 BP 神经网络避障控制模型构建及仿真[J]. 西安工业大学学报,2015,35(08):678-682.

[95] 陈伟根,潘翀,云玉新,王有元,孙才新.基于改进小波神经网络算法的电力变压器故障诊断方法[J]. 仪器仪表学报,2008(07): 1489-1493.

[96] 李金屏,王风涛,杨波.BP 小波神经网络快速学习算法研究[J]. 系统工程与电子技术,2001(08):72-75.

[97] 乔俊飞,王功明,李晓理,韩红桂,柴伟. 基于自适应学习率的深度信念网设计与应用[J]. 自动化学报,2017,43(08): 1339-1349.

[98] Luo L, Wang Y, Peng H, et al. Training restricted Boltzmann Machine with dynamic learning rate [A]// 2016 IEEE International Conference on Computer Science & Education[C]. USA: IEEE, 2016.

[98] A. Žilinskas. Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms, by Jan A. Snyman [J]. Thesis, 1993(6):613-615.

[100] Streeter M, Streeter M. Delay-tolerant algorithms for asynchronous distributed online learning [A]. International Conference on Neural Information Processing Systems[C].USA:MIT Press, 2014, 2915-2923.

[101] Zhang S, Choromanska A E, LeCun Y. Deep learning with elastic averaging SGD[A]. Advances in Neural Information Processing Systems[C]. 2015: 685-693.

[102] Recht B, Re C, Wright S, et al. Hogwild: A lock-free approach to parallelizing stochastic gradient descent[A]. Advances in neural information processing systems[C]. 2011: 693-701.

[103] Abadi M, Agarwal A, Barham P, et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems [J]. 2016.

[104] Qian N. On the momentum term in gradient descent learning algorithms [J]. Neural Networks the Official Journal of the International Neural Network Society, 1999, 12(1): 145-151.

[105] Duchi J, Hazan E, Singer Y. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization [J]. Journal of Machine Learning Research, 2011, 12(7): 257-269.

[106] Zeiler M D. ADADELTA: An Adaptive Learning Rate Method [J]. Computer Science, 2012.

[107] Kingma D P, Ba J. Adam: A method for stochastic optimization [J]. arXiv preprint arXiv:1412.6980, 2014.

[108] Senior A, Heigold G, Yang K. An empirical study of learning rates in deep neural networks for speech recognition [A]. 2013 IEEE International Conference on Acoustics, Speech and Signal Processing [C]. USA:IEEE,2013:6724-6728.

致　　谢

攻读学位期间的科研项目和学术论文

[1] 国家自然科学基金：面向设计意图在线交换的语义云粒元重组方法研究（No. 61672461）；

[2] 国家自然科学基金：图像与视频的不变性局部结构特征描述及应用研究（No. 61672463）；