硕士学位论文

面向云服务的弹性调度算法的研究与实现

THE STUDY AND REALIZATION ON ELASTICITY SCHEDULING ALGORITHM ORIENTED CLOUD SERVICES

张淼

哈尔滨工业大学

2017 年 6 月

国内图书分类号：TP302.8　　学校代码：10213
国际图书分类号：681.39　　密级：公开

工程硕士学位论文

面向云服务的弹性调度算法的研究与实现

硕士研究生：张淼
导　　师：左德承教授
申请学位：工学硕士
学　　科：计算机技术
所在单位：计算机科学与技术学院
答辩日期：2017 年 6 月
授予学位单位：哈尔滨工业大学

Classified Index: TP302.8
U.D.C: 681.39

Dissertation for the Master Degree in Engineering

THE STUDY AND REALIZATION ON ELASTICITY SCHEDULING ALGORITHM ORIENTED CLOUD SERVICES

Candidate: Zhang Miao
Supervisor: Prof. Zuo Decheng
Academic Degree Applied for: Master of Engineering
Speciality: Computer Science and Technology
Affiliation: School of Computer Science and Technology
Date of Defence: June, 2017
Degree-Conferring-Institution: Harbin Institute of Technology

摘 要

云服务是一种基于互联网的服务模式，通常包括了从服务申请、使用到支付的全部过程。云计算的吸引力就在于其计费标准是按照用户实际使用的资源付费的，云服务有能力在任何时间灵活的增加或者减少资源以满足用户的需求。弹性作为云计算模式下的一个重要特征，资源的动态分配机制实现在需求增加时，相应的增加资源；在需求降低时，将分配的资源回收。弹性通过按需付费的模式灵活适应用户波动的需求，在保证服务等级协议（SLA，Service Level Agreement）和服务质量（QoS，Quality of Service）的前提下使资源供应与资源需求尽可能接近。

弹性调度是实现云计算系统中动态、频繁以及自动变更资源配置关键，其性能很大程度上决定了云计算系统提供服务的能力。传统面向可扩展性的调度因无法随着系统资源需求的变化而动态伸缩资源，导致大量的资源浪费，以致无法满足当前云服务的要求。一个可以快速伸缩、精确配置的弹性调度策略将会有效的提升云计算系统的性能。但是目前的弹性算法主要还存在两方面的问题：一方面体现在面临大规模突发式与激增式应用负载涌现时，出现调度时延长（传统方式下虚拟机从镜像启动到服务可用的时间通常在 5~15 分钟左右）、大量 QoS 不满足以及导致 SLA 违约等问题；另一方面是目前的弹性调度算法大多只从用户或者服务提供者单方面进行优化，很少有从双方共同的角度，以费用最低为原则，以寻求 Provision-Cost 平衡为目标进行研究。

为解决上述问题，本文通过对调度算法进行深入研究，提出了一种结合反馈与多级预测机制的敏捷弹性在线调度算法，采用预测机制提前预留资源有效避免了由于资源延迟而导致的服务违约问题；加入预测式补偿机制，在资源分配过程中结合系统反馈信息，在保证预测结果准确性的同时提高资源分配的合理性。文章从资源配置速度与配置精确度两个方面出发，既考虑到用户的经济效益又结合服务提供者的切身利益，在保证 QoS 和 SLA 的基础上，以最小的成本实现最大的资源利用率，并设计罚金模型作为评测指标进行验证。最后，在 CloudStack 云平台上，部署适用于云环境下的弹性应用，提供多种资源、多种负载验证本文算法的有效性。

关键词：云服务；弹性；调度算法；预测；CloudStack

Abstract

Cloud services is an Internet-based service model whose typical process includes services application, use and payment. Pay-as-you-go is the most attractiveness feature of cloud computing.Cloud services can increase or reduce the resources flexibly at any time to meet the users' needs. As an important feature of cloud service, elasticity can allocate or release resources with the increasing or decreasing of demand. Elastic scheduling can meet the users' changeful demands through the pay-as-you-go model. It can make resource supply and resource requirments as close as possible under the premising of guaranteeing Service Level Agreement (SLA, Service Level Agreement) and Quality of Service (QoS, Quality of Service).

Elastic scheduling is key to achieve the dynamic, frequent and automatic resource allocation in cloud computing systems.The performance of the elasticity algorithm highly affects the capability of cloud systems to provide services. The traditional scalability oriented scheduling cannot dynamically reduce the resources with the change of system resource requirments. It wastes too many sources to meet the current requirements of cloud services. A fast scalable, accurate configurable of the elastic scheduling strategy will efficiently improve the resource utilization of the cloud computing system.However, the current elastic algorithms have two main shortages:1)when large-scale demands burst and surge application load appears, the algorithms cause large time delay (in the traditional solution, the virtual machines takes the 5~15 mintues to become ready from a mirror), a large number of insatiable QoS, and lead to break SLA contract and other issues. 2) Most of current elastic scheduling algorithms optimize from the user or service provider, few tries to minimum cost and studies the provision of Provision-Cost balance.

In order to solve the above problems, this paper proposes an agile on-line scheduling algorithm based on feedback and multi-level prediction mechanism. It applies the forecasting mechanism to reserve the resources to avoid breaking the SLA caused by the delay of resources. And a predictive compensation mechanism is integrated to feedback the system information allocating the resources. This solution can ensure the accuracy of the forecast results as well as improving the

rationality of resource allocation. Based on the two aspects of resource allocation speed and configuration accuracy, the paper considers both the economic benefits of users and the vital interests of service providers. On the basis of guaranteeing QoS and SLA, this solution achieves the maximum resource utilization rate with minimal cost, and then the penalty model is designed as the evaluation index to verify the solution. Finally, it is tested on the CloudStack cloud platform with elasticity applications. The effectiveness of the algorithm is verified with resources workloads.

Keywords: cloud service, elasticity, scheduling algorithm, forecasting,CloudStack

目 录

第 1 章 绪　论

1.1 课题背景

1.1.1 课题来源

本课题来源于国家 863 项目“云计算测试与评估系统研制”，针对云计算中弹性调度算法的研究，根据云弹性按需供给、快速配置资源的特点，以云服务供应商和服务使用者双方视角，提出一个有效的弹性调度策略。

1.1.2 课题的研究目的与意义

目前云计算已经成为一种强大的计算模式，支持潜在的众多远程用户多样化的需求。按照美国国家标准及技术研究所（NIST）的定义：云计算是一个可以快速配置、弹性伸缩的模型，它通过一个可配置的计算资源共享池（例如，网络、服务器、存储、应用和服务）使网络访问变得按需、便捷以及无处不在[1]。如今，云计算平台不仅已经被一些创新型公司广泛使用，而且得到了传统企业的密切关注。不少开源社区纷纷推出了自己的产品，如 CloudStack、OpenStack、Eucalyptus、Open Nebula 等[2-6]。以市场发展趋势来看，预计 2020 年全球云服务市场规模将从 2015 年的 1700 亿增长到 3900 亿；同样预计我国公有云市场规模将达到 570 亿[7]。因此，从我国乃至全球范围来看，云计算作为一种新兴产业，正发挥着越来越重要的作用。

云计算的吸引力在于其计费标准是基于使用的定价模型按照用户实际使用的资源付费，云服务有能力在任何时间灵活的增加或者减少资源以满足用户的需求，这也是云服务中非常重要的特性--弹性的体现。相比于传统方法中需要租用大量资源来维护高峰期时的 IT 基础架构配置，使用弹性云服务可以极大地节省租用成本。由于服务供应商可以通过提供基础设施获取利益；而服务使用者可以借助云平台按需付费的计费模式，降低租用成本，提高使用灵活性。因此，弹性云服务的使用无论对服务提供商还是对服务使用者，都提供了极大的便利。当然，随之而来的如何站在双方共同的角度寻求利益最大化将成为云计算持续快速发展过程中亟待解决的重要问题之一。

目前，世界上最大的云计算服务提供商 Amazon 的具有核心战略意义的亚马逊弹性云平台（EC2，Elastic Compute Cloud)就是以“弹性”二字冠名的，该平台主要通过按需调配、整合数据中心可用资源来减少不必要的资源浪费，

“弹性”二字也表明了该云平台具有很好的可伸缩性和可扩展性[8-12,14]。由于云计算所采用的是“pay-as-you-go”的收费模式[13]。过度配置导致的闲置资源会大大提高服务提供商和用户方的成本，而资源配置不足导致的服务能力饱和会在很大程度上影响服务质量，严重的甚至会导致服务降级或者服务失效。弹性是云计算系统中动态、频繁、自动改变资源配置能力的主要体现，因此云服务中弹性调度算法的性能在很大程度上决定了云计算提供服务的能力。研究表明，云服务中增加或者删除资源的调度频率与对工作负载变化的适应能力密切相关。传统的可扩展性就因为其滞后性与静态性而不能满足当前云服务的要求，因此快速弹性、精确配置的调度策略会给云计算的未来带来更多的可能性。

1.2 国内外研究现状

目前国内外学术界已经对相关问题展开了大量的研究。本文将弹性云按照不同角度进行分类，结果如表 1-1 所示。

（1）从伸缩方式来看：资源缩放分为水平缩放和垂直缩放[15]。在水平缩放（缩/放）过程中，以虚拟机为最小资源单位按需要添加或释放虚拟机[16-20]。与此相反，垂直缩放（上/下）通过改变分配给已经运行的虚拟机的资源来实现，例如增加（或减少）分配的 CPU 或内存[21]。垂直伸缩相对于水平伸缩有更好的稳定性，但是由于最常见的操作系统均不允许在线操作（不需重启）虚拟机，因而大多数云供应商只提供水平缩放。

（2）从经济角度来看：现有研究分别针对供应方和用户方两个立场。在云服务中，供应方利益的研究目标主要包括提升系统性能、提高服务质量、最大化资源利用率以及提高服务效率等，用以获取最大利益[22-26]。文献[22]和文献[24]均通过概率模型策略提高供应方的经济效益。文献[25]通过对目前弹性云服务中存在的问题进行改进，设计并实现了基于负载自动感知和资源动态调整的弹性云平台保证供应商的利益。对于服务提供者来说，如何以最少的资源配置和最高的资源利用率来保证系统性能和应用服务质量，仍是一个尚未解决的复杂问题[22]。对于用户方来说，在不降低应用性能的前提下降低租用费用就是其关注的重点问题。[26,27]针对用户盲目租用资源造成费用过高的问题，提出一种近似算法用于降低不必要的资源开销，但该算法是在用户需求已知的前提下进行计算的，在实用性方面还存在一定的问题。[28]中 Stephane Genaud 等人通过对任务资源匹配进行相关研究，提出了传统启发式算法和二维装箱结合的优化策略，寻求虚拟机资源量伸缩过程中速度和费用的平衡，用以降低用户资源租用代价。

（3）从实现机制的角度来看：弹性伸缩策略主要分为反馈触发、前瞻预

测以及二者结合的机制。目前比较流行的反馈触发策略就是根据一些性能指标和预定义的阈值来触发伸缩的阈值策略，由于其（明显的）的简单性而受到用户的欢迎。例如亚马逊 EC2 以及一些第三方工具如 RightScale[5,30-34]。但是由于其在资源部署过程中存在较大的时延问题，同时对于负载大规模增加的情况应对能力有限，所以要想广泛适用于各种场景，还需要对负载趋势有更深的理解与把握[24]。文献[35]中提出了一种简单的阈值策略，CPU 利用率超过 80%时触发器根据预先设定的伸缩规则进行调度；[36]中设置了 70%和 30%的上下阈值；[37]和[38]中对性能指标进行了改进，分别采用应用的平均响应时间以及 CPU、内存和网络带宽多个指标整合的结果作为性能评价的指标。但是如果资源无法及时供应造成的影响也将是巨大的，有例子表明互联网应用程序会由于意外过载造成资源供应不足而中断。例如，Amazon.com 网站就曾在购物季由于过载造成了四十分钟的停机。后者意味着在负载请求还未到来之前通过预测提前准备所需资源，降低虚拟机从资源申请到服务可用的时延问题，实现快速、高效的弹性云服务。文献[39]中通过对比多种预测技术，提出 ANOVA-AR 算法对离线数据进行预测。另外一种机制就是将两者进行有效结合，研究表明将阈值与预测两种策略结合在一起能更好的实现弹性调度。文献[40]提出了一种混合的方法，采用了主动机制和被动机制结合的弹性算法，在扩展时采用阈值策略，收缩时采用预测方法。通过实验表明该方法相比于现有的弹性方法减少了到达 CPU 瓶颈的数量，同时有效控制了收缩过程中发生错误的比例。但该算法处理的是稳定类型的负载，对于负载规模大幅增加的情况将发生大量的服务失效。文献[41]中通过排队论模型设计了两个控制器分别用来实现阈值策略和预测方法，该实验考虑了实际场景下的负载发生特征，降低了 SLA 违约率，但该实验没有考虑到虚拟机启动与释放所需要的延迟时间。在多数情况下，结合反馈机制获取的性能指标一般是直接从监视器获得，以一种完全被动的方式来执行弹性伸缩，然而将先前获取的历史监视数据通过分析来预测或期待系统的行为从而以主动的方式来执行弹性伸缩也是可行的[42]。因此如何扬长避短，选择合适的实现机制对弹性性能起着至关重要的作用。

（4）从实现技术来看：主要可分为手动策略和自动策略两种。手动策略是由用户根据自己监测到的资源情况而进行响应的弹性行为，GoGrid、Rackspace 和 Microsoft Azure 在最初都是通过手动方式来弹性伸缩的[43-44]。但由于 Web 应用负载是高度动态的，因此其自动调度过程中有两方面困难：一方面是如何应对负载上升时资源供应不足造成的资源瓶颈（尤其是 CPU 瓶颈）造成的服务失效、产生 SLA 违约、影响 QoS 甚至导致性能降级的问题；另一方面是如何应对负载降低产生的大量资源闲置，造成不必要的资源浪费的问题

[42]。传统的手动策略已经明显无法满足用户需求，针对这一问题国内外主要采用自动伸缩技术解决此类问题，其中自动伸缩技术按类别划分可以总结成如下五类：阈值规则，控制理论，强化学习，排队论和时间序列分析[43-46]，其中阈值规则属于反馈触发方式，深受云供应商们的青睐，时间序列分析是属于主动方式，控制论、排队论和增强学习在两种方式中都有体现。a）阈值策略，使用阈值规则若想实现较好的性能通常需要调度者细心调节参数以避免产生系统震荡，设置系统冷却时间、提出多个不同阈值，均为解决这一问题应运而生的[47,48]。文献[49]中 RightScale 的弹性伸缩算法在规则中加入了投票机制，对每一台虚拟机按照特定的规则进行单独评估最后以投票的方式选择集群是否需要进行弹性伸缩，一般根据集群申请虚拟机的准备时间加入冷却时间。

表 1-1 弹性云计算研究类别划分

分类方法	类别	研究目的
伸缩方式	水平伸缩	以虚拟机为最小资源单位按需要添加或释放虚拟机
	垂直伸缩	通过改变分配给已经运行的虚拟机的资源，调整资源配置
经济角度	云供应方的角度	提升系统性能、提高服务质量，获取最大经济利益
	云用户的角度	在不降低应用性能的前提下降低租用费用
	双方的角度	在满足基本 QoS 和 SLA 的基础上，站在双方视角利益最大化
实现机制	前瞻预测	主动机制，通过预留机制提前准备资源
	反馈触发	被动机制，通过监控触发
	混合机制	预测与触发机制相结合
实现技术	手动策略	手动调度，用户根据自己监测到的资源情况而进行响应
	阈值规则	随着定义阈值的变化，只有当这些变化已被检测到，才采取相应的策略进行调度
	控制论	研究各类系统的调节和控制规律
	排队论	通过对服务对象以及服务时间的研究得出统计规律来寻求某些指标最优或费用最经济问题
	增强学习	根据每个应用的状态以及给定的输入负载做出最佳的弹性伸缩决策
	时间序列分析	根据时间序列历史信息进行观察、研究，找出其内部的发展变化规律并完成未来时间的信息预测与模式匹配

[50]最初基于活动会话数提出了一组被动的弹性策略，[51]根据 RightScale 的方法扩展了该策略：如果所有的 VM 的活动会话数超过上限阈值则提供一个新的 VM，如果存在 VM 的活动会话数低于下限阈值，且同时至少存在一个不含活动会话的 VM，则回收该空闲 VM。b）强化学习（RL），RL 中两种常见的算法包括 SARSA 和 Q-learning，但这两种算法都存在一些缺陷：（1）初始性能不好，需要较长的学习时间。[52]通过启发式方法进行状态空间的搜索，又通过 Q-function 方法进行初始化估算，从而在迭代中更新加速收敛过程，另外也可以通过在每步中访问更多状态[55]或者使用并行方式学习[47]降低训练时间。（2）较大的状态空间。[53]使用单纯优化方法来选择能够返回高回报的状态。并行的方法也能解决状态空间过大问题，[54]为每一个虚拟机都创建一个代理，每个代理都管理自身的小规模查找表。使用查找表来代表 Q-function 并不高效，因此可以使用其他类型非线性逼近函数，例如：神经网络、CMACs、回归树、支持向量机、小波和回归样条函数。（3）环境适应性较差，RL 在弹性伸缩中的使用与应用相关性较高。[56,57]使用一种基于 ANN 的 RL 代理来根据应用和虚拟机的性能自动调节参数。c）排队论，[58]中使用排队论进行预测平均响应时间和工作负载的值，来实现应用在服务器间的分配，是供应商的利益最大化问题。[59,60]将云计算应用建模为 G/G/n 队列，其中 n 代表服务器数量。该模型可以在给定工作负载 λ 或请求的平均响应时间，以及服务器配置前提下，估算所需的资源。[57]中使用 G/G/1 队列构成的队列网络来建模。首先使用直方图来预测负载峰值，基于该负载峰值以及队列模型计算应用中每层需要的服务器数量，每层所需服务器数量可以使用被动方法来校正。这种模型最大的问题是当面对峰值负载时，可能造成大量的资源浪费。而且并没有考虑到用户的服务质量与最小化费用问题。d）控制论，固定增益控制器（包括 PID、PI 和 I）是控制论中最简单的控制器类型，[61,62]使用 I 型控制器来根据平均 CPU 使用情况调整虚拟机的数量。[63]使用 PI 型控制器根据批处理作业的执行进度来管理资源需求。自适应控制器也是使用较为广泛的一种控制器。[64]提出一种 MIMO 自适应控制器，该控制器使用二阶 ARMA 来对非线性的时变的资源量与性能之间的关系进行建模，该控制器能够调节 CPU 和磁盘 IO 的使用率。模糊模型是控制系统中重要的性能模型，可用来描述负载与所需资源之间的关系。[65]使用一种自适应的模糊控制器来对 Web 应用的业务逻辑层建模，并估算给定输入负载下的 CPU 资源量。模糊模型的一种改进方式为神经网络模糊控制器，该控制器使用 4 层的神经网络来表示模糊模型。[58]使用神经网络模糊控制器，通过快速的在线学习实现自适应的调节模型参数。[16]中将主动方式与被动方式结合起来，运用控制理论设计两个控制器，一个用于被

动扩展，而另一个用于主动收缩。但是在负载剧增的情况下，其瞬时处理能力不足以支撑大量的负载请求从而造成大量的服务违约，无法保证 SLA 和 QoS 指标。e)时间序列分析，文献[66,67]使用移动平均进行预测，这种模型结构简单，预测结果也较差，基于此这种方法一般都用来对时间序列进行去除燥声的操作。文献[68,69]中使用指数平滑法进行预测。自回归方法使用范围很广泛，[70-72]均使用该方法进行预测，但其一般与移动平均组合使用构成 ARMA 模型（自回归移动平均法），提高预测准确率。但其预测精度受历史窗口和最大匹配数量影响，可见不同的实现技术对弹性性能有较大影响。

1.3 本文的主要工作

本文针对目前云计算环境下资源利用率低、资源浪费严重、大规模突发式负载下因应对能力差而导致服务质量降低、QoS 违约上升等现象，拟从以下几个方面进行研究：

1.研究弹性云的实现机制，改善云环境下的弹性服务能力。搭建 CloudStack 实验平台，部署 TPC-W 应用并在客户端使用 LoadRunner 作为负载发生，设计完成实验。

2.通过对国内外云服务弹性机制的研究，发现由于虚拟机组织会产生很大的时延，并且这一时延会随着系统规模的增大而显著增长造成服务失效，严重影响云环境下的弹性服务性能。针对这一问题，本文充分利用预留机制，提出了一种基于反馈与多级预测机制的敏捷弹性在线调度算法（AEOAS）来实现虚拟机的快速弹性供给。通过长短周期相结合的方式，既实现了资源预留提前部署虚拟机，又完成了预测补偿减小预测误差；在资源分配过程中融合系统实时反馈信息，提高弹性性能。

3.针对前文提到的经济效益问题，站在云供应商和用户双方的角度，在保证 QoS 和 SLA 的前提下，设计多种不同类型的虚拟机寻求以最小的租用成本实现最大的资源利用率。

4.设计 Kafka 信息采集系统，实时收集集群日志作为访问请求，利用其时间开销可忽略不计的特性充分保证弹性云环境对时间开销的要求，设计多种负载类型验证算法可行性与适用性。

1.4 本文的主要结构

第 1 章是绪论，本章主要介绍了课题来源、课题的研究目的与意义，详细分析了本课题在国内外的研究现状并介绍了论文的主要工作。

第 2 章是弹性云服务，本章首先介绍了弹性以及弹性云服务的概念，之后通过对目前研究现状的分析与总结提出本文 AEOAS 弹性调度算法的基本设计思想。

第 3 章是基于改进的动态 SARIMA-BPNN 在线负载预测算法，通过对目前预测技术与预测方法的研究提出适用于本文负载特征的 ARIMA 预测算法，并针对该算法存在的问题提出若干改进：首先，原有算法不适用于实时动态变化的弹性云环境，本文结合云环境的特点通过实时更新预测模型设计了在线 ARIMA 预测算法；其次，针对本文负载数据的周期性与季节性特点，在原有算法基础上加入了季节特征；再次，由于原有算法在应对波动较大数据时相应的数据偏差会增大从而产生具有明显非线性特征的预测误差，为了提高预测精确度本文加入适用于拟合非线性数据的神经网络算法拟合预测误差，同时为了适应本文的弹性云环境提出了在线 BPNN 模型拟合误差；最后设计实验，验证预测算法的准确度。

第 4 章是基于 CloudStack 平台的 AEOAS 弹性调度，本章详细介绍了本文弹性调度机制的设计与实现，之后介绍了本文基于反馈与预测补偿机制的弹性调度算法 AEOAS 的设计与实现，最后通过对整个系统的测试原理以及测试流程的介绍，为后文实验验证做好准备。

第 5 章实验结果与分析，本章详细介绍了本文的实验验证部分，其中包括各种软硬件配置的实验环境、实验中相关数据分析以及实验结果与实验分析。设计了两种不同类型的负载，既验证了本文中算法对于简单波形负载的有效性，又通过对四组算法三组对比实验的设计，验证在复杂突发负载的情况下本弹性算法在精确度、违约率以及使用费用等方面的优势，并证明该算法具有一定的实际意义，在本文中提出的弹性云环境下具有可行性。

最后是结论，包括对本文主要研究内容的总结以及对未来相关研究的建议。

第 2 章 弹性云服务

2.1 弹性云服务的概念

2.1.1 弹性的定义

从物理层面以及经济层面来看，弹性分别代表了一种材料属性和一种多变量之间的灵敏程度，在这两种情况下的弹性是可以用计算公式精确描述的，但如今弹性这个术语已被转移到云计算背景下大量使用并且被认为是云范式的核心属性之一[73]。然而云环境下的弹性由于没有一个统一的量化公式而缺乏一个明确的定义，表 2-1 列出了几种不同的研究视角下为弹性做出的不同定义。

表 2-1 弹性定义

作者	定义
OCDA[74]	依靠用户负载量向上扩展和向下扩展的能力
Reuven Cohen[76]	可管理、可测量、可预测、自适应的基于实时需求的应用程序的可量化能力
Rich Wolski[75]	用户请求到不同资源之间的映射关系
Herbst et al[73]	系统能够通过自动提供或移除资源适应负载变化的程度，以使每时每刻所提供的资源都能够恰当匹配当前的资源需求。
Cooper et al[77]	通过向一个运行时系统添加组件新实例从而增强系统能力。
Aisopos et al[78]	服务提供商能够为一个特定的服务提供动态资源分配（CPU、内存、磁盘），从而使得服务能力能够根据配置变化。
Espadas et al[79]	能够根据应用需求创建可变数量虚拟机的能力。
Li et al[80]	系统能适用负载变化的最短时间。
Perez et al[81]	系统为适应负载变化在不中断服务情况下提供或移除资源的能力。
Garg et al[82]	高峰时期服务能够被扩展的程度。
Han et al[83]	为应对变化的负载要求而自适应的调整资源规模的能力。
Pandey et al[84]	系统扩展和收缩的能力。

综合表中各篇论文中给出的不同定义，本文所讨论的弹性更偏重于文献[73]中所表达的观点，也就是说弹性是一种用来评价系统能够通过自动提供或移除资源，从而适应负载变化程度的度量，使得在每个时间点上系统所提供的资源都能够恰当匹配当前的资源需求。

这一定义准确的反应出了弹性中两个最核心的特征：速度和精确度。其中速度代表着弹性过程中的时间概念：当外界负载发生变化时，系统捕捉到这种变化及时调整资源配置，调整配置适应需求的时间越短、速度越快，则提供的弹性效果越好。弹性中的速度既包括需要增加资源时向上扩展的速度（也就是添加虚拟机提供服务的速度）和需要减少资源时向下扩展的速度（也就是资源释放的速度）。同样精确度代表着弹性过程中的精准程度：对每一时刻来说，当外界负载发生变化时，系统实际分配的资源与资源需求之间的拟合程度代表着弹性的精确度，所需的资源量与系统实际分配的资源量越接近则说明系统的弹性能力越强；不然，若分配的资源大于系统需要的资源则代表着资源分配过度，造成相应的资源浪费；若分配的资源小于系统需要的资源，则代表着此时资源分配不足，影响服务质量严重的会造成服务降级甚至服务失效。我们以图2-1 来说明。

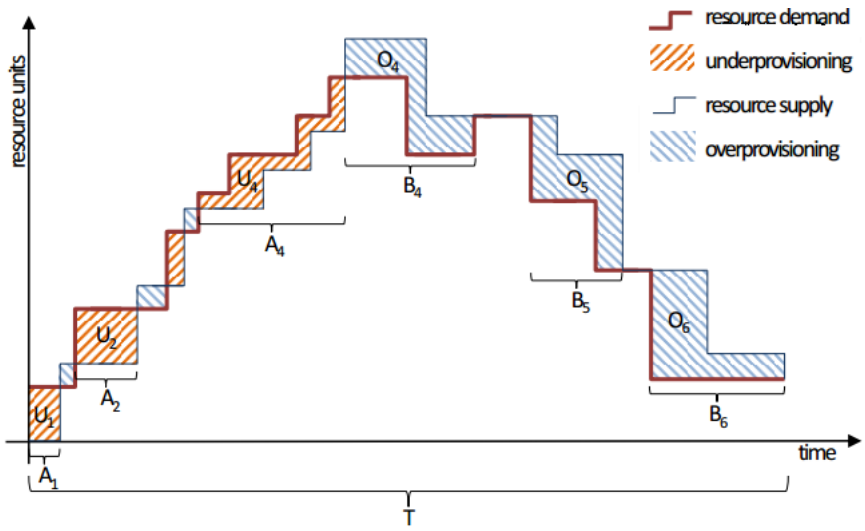


图 2-1 弹性核心指标

该图表示随着时间变化资源需求与资源供应曲线的变化过程，其中红色折线代表了资源需求情况，蓝色折线表示当资源需求随着时间不断变化时，系统弹性的调整资源供应情况，使资源供应曲线与需求曲线尽可能的接近。其中红色阴影表示资源供应不足的情况，用U_i来表示；蓝色阴影部分表示资源供应过量的情况，用O_i表示，U_i、O_i则分别对应着弹性过程中精确度的概念。图中

A_i代表资源供应不足的时间，B_i代表处于资源供应过量的时间长度，A_i和B_i反应了弹性过程中速度的概念，时间越小则表明资源供应的速度越快，弹性能力越强。本文对弹性的定义与评测充分围绕这两个核心特征进行，在这里我们用一个量化值来衡量弹性性能的好坏，分别定义了两个罚金指标 p 和 q，其中 p 代表资源供应不足情况下由于 SLA 违约所产生的罚金，q 代表资源供应过量时浪费的费用，则两者的总和即为整个弹性调度过程中基于 SLA 和 QoS 的罚金，罚金值越低则表明相应弹性性能越好[85]。计算过程如公式（2-1）所示：

$$C = \int_{t \in A_i} p dt + \int_{t \in B_i} q dt \qquad (2\text{-}1)$$

在概念上，弹性容易被人混淆的就是与扩展性的区别，[85]中甚至直接将弹性定义为可扩展性的重命名。事实上，弹性与扩展性是两个完全不同的概念，总结地说可以从如下两个方面考虑：（1）动态性、实时性，动态与实时是弹性的重要属性，而扩展性则更多的反映了系统的静态属性。（2）伸缩性，弹性在资源调度过程中既包含了向上扩展又包含向下收缩的过程，而扩展性主要强调了资源的向上扩展过程。由此可见弹性是有别于扩展性的重要属性，在云计算背景下有十分重要的意义。

2.1.2 弹性云服务的概念

在云计算中引入弹性的概念，是云计算深受用户欢迎并保持快速发展的重要原因之一。引入弹性的云服务可以根据需求调整自身的软件规模，并且为了更好的支持弹性，云供应商提出两个变革：（1）提出定价模型“pay-as-you-go”的概念，允许用户按照使用的资源和使用的时间付费；（2）提供了 API 接口，其中包含了申请、释放等操控虚拟机实例的指令。那么，所谓的弹性云服务就是云计算“弹性伸缩”（Auto-Scaling）服务，这项服务充分体现了云计算“弹性”、“灵活”的核心服务要义。

事实上，我们正处于用户需求催生“弹性伸缩”的阶段，针对突发型负载的应用场景越来越常见，无论是以提供基础设施为主的大型企业还是租用服务的小型公司，我们已经可以预见到随之而来的访问量的激增以及业务高峰期过后的缩容现象，云供应商迫切的需要更好的弹性方案帮助客户适应当前的业务需求。因此提供可伸缩的弹性云服务要求云供应商通过弹性云平台恰到好处的将计算、存储、网络等资源进行合理调配和自适应规划，既要避免因为资源供应不足而导致的服务中断又要避免资源分配过度造成的大量闲置资源空转。与此同时，在保证“弹性”的基础上供应方也有义务提供灵活而精确的计费模式，让用户只为使用的资源和时间计费。好的弹性可以让提供商在保证服务质量的

同时节省能源，也可使云基础设施用户在保证性能的前提下，节省资源租用费用[85]。

例如一个提供云服务的应用程序，在服务能力满足用户需求时可以提供有效服务，但当用户需求增加到资源所能提供的最大限度时若要保证服务的有效性就要增加新的资源，也就是弹性扩展的过程；若随着用户需求的降低，访问量下降，此时多余的资源应被移走释放到云中。一个具有弹性的云服务系统应该能够立即检测出这种需求的变化并提供适当的资源来有效应对用户请求，服务模型如图 2-2 所示：

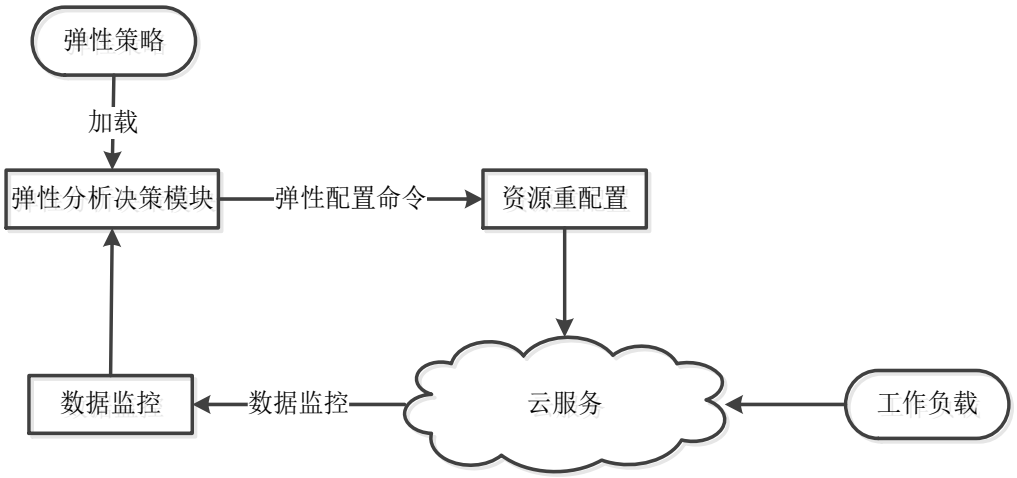


图 2-2 弹性云服务结构模型

2.2 AEOAS 算法的基本思想

根据图 2-3 所示的弹性云服务组织分类图所示：

（1）在伸缩方式上由于大多数供应商都只提供水平伸缩方式，不允许在线操作虚拟机，因此本文的算法也采用水平伸缩的调度方式。

（2）从经济角度上看，目前的研究内容主要集中在实现服务提供方或服务使用方某一方的利益最大化方面，而对于双方共同利益的考虑则比较少[26,28,29]。在用户需求日益激增的今天，如何有效的平衡双方利益实现经济共赢，是保证弹性云服务健康、快速发展的重要问题，因此本文的研究视角将站在双方的角度，寻求 Provision-Cost 平衡。

（3）从实现机制的角度看，采用反馈式触发方式意味着随着系统工作负载的变化，只有当这些变化被检测到，才采取相应的策略进行调度。由于资源配置需要一定的时间，因此会产生相应的时延问题，由于云服务的弹性伸缩是用户需求催生下的结果，大量突发式与激增式应用负载的涌现使得采用反馈式弹性调度算法时面临着调度时延长（传统方式下虚拟机从镜像启动部署到服务可用的时间通常在 5~15 分钟左右）、大量 QoS 不满足以及造成 SLA 违约等

问题。对于 Web 类型的服务来说，服务质量（通常包括响应时间、可用性、可靠性、安全性等）至关重要，因为用户通常难以忍受超过 5 秒钟的响应时间。而采用传统的反馈式弹性调度算法会造成大量的服务超时（一般超过 5s 就视为服务超时）；若采用提前分配大量资源的策略来应对由于负载请求量大规模增加而造成的服务失效问题，则会造成大量的资源浪费。与此相对的是基于预测机制的弹性调度策略。采用前瞻预测方式意味着资源的提前部署与预留，采用预测技术向上扩展时，可以在峰值负载到来之前准备好所需资源提前加入到集群中，从而有效解决弹性调度当中资源分配滞后的问题；在向下收缩时，在资源开始计费之前回收多余资源避免资源浪费。可见将预测机制应用到弹性调度的过程中可以在很大程度上提高弹性算法的精度，因此本文采用预测的方式实现资源弹性管理。

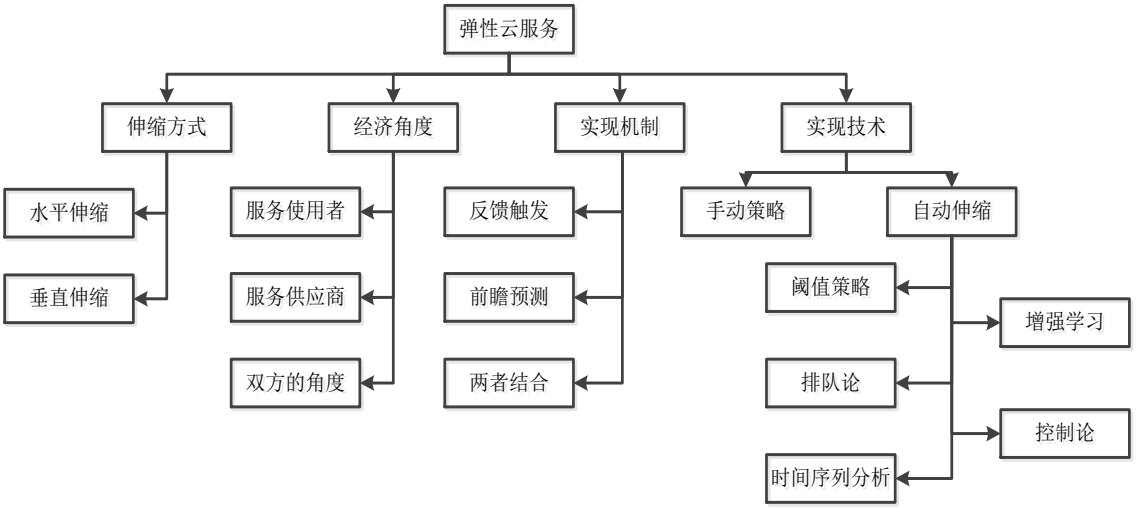


图 2-3 弹性云服务组织分类图

（4）从弹性调度过程中采用的实现技术上来看，a）采用阈值策略在应对突发型工作负载时，其规则的有效性难以保证，因为只有阈值条件满足后才能进行相应的资源分配过程。另外，该技术高度依赖于管理者定义的阈值以及工作负载的特征。这也是阈值策略的最主要缺陷：阈值选取难度大、受负载特征影响明显。b）基于强化学习的弹性策略能够在没有先验知识的情况下捕获最佳的管理策略。使用该策略时，用户不需要为应用定义一种特定的模型，该策略能够在线学习并适应应用的负载或者系统状态的变化。强化学习是解决一般应用资源弹性管理的一种较好的方法。但是该策略目前还不成熟，不能满足真实生产环境的场景，且该策略存在初始性能较差、学习时间较长、对突发负载适应性较差和搜索空间较大等问题。c）排队论模型已经在应用和系统建模中被广泛使用，经常用来为固定的系统结构建模，因此系统结构中任何微小的改变都需要重新建模。频繁的重新建模使得排队论在资源的弹性管理的代价很高，

因为该模型需要同时应对复杂的负载变化。当前也有很多论文尝试使用排队论来为多层复杂应用进行建模，但为一般应用设计弹性管理策略时，排队论模型不是最佳选择。d）基于控制论的弹性策略的应用效果依赖于控制器的种类以及目标系统的动态性。为使控制器实现对应用自动添加或者移除资源，需要创建一个可靠的性能模型，使得该模型可以将输入变量映射为系统的资源需求。但是采用简单的被动式控制器虽然方便但是准确度不高，采用主动式控制器受应用模型影响较大，对一般应用的弹性管理适用度不高。

基于时间序列分析的弹性策略是实现资源弹性管理的较为受欢迎的方案，由于其能够预测应用的未来需求，因此使用该策略可以提前进行资源的申请，从而有效避免由于虚拟机启动和应用部署所花费的时间。该策略的潜力较大，在涵盖金融、工程、经济学和生物信息学等多个领域广泛引用，其特有特征也决定了该方法同样适用于云计算环境下资源预测，其中包括：移动平均模型、自回归模型、ARMA 模型、指数平滑法和基于机器学习等不同方法。在学术界中，时间序列分析几乎都用于进行工作负载或资源情况的预测与分类，由于其可以根据时序历史信息完成对未来时间的信息预测与模式匹配，并且不依赖于应用模式，从而有效避免上述方法中存在的问题。云计算中的弹性调度旨在以最少的资源在满足 SLA 的前提下为用户提供服务，随着用户需求的实时变化，云中供应的资源也应随之动态变化，以实现供应与需求之间的拟合。因此本文采用时间序列分析中广泛使用的 ARIMA 算法进行预测。但是由于预测误差难以避免、随机事件难以预知，难免会影响到预测结果的准确性进而影响着云服务的弹性效果。目前针对这一问题进行补偿或改进以提高预测准确度的算法有很多，一方面是从算法本身出发提高预测的准确度，但此种方法仍然无法应对随机事件的发生，并且当随机事件出现时系统没有相应的应对措施；另一种方法就是加入监控机制，结合阈值策略通过实时采集系统资源信息发现需要补偿的点，控制系统及时做出响应。但由于阈值策略是被动触发式策略，当监控到阈值规则满足后由于时延等问题的存在仍然会造成系统响应不及时、发生服务违约的现象，因此本文提出了一种预测式补偿机制用以解决上述问题，并且在资源分配过程中充分结合当前系统资源监控信息。

综上所述，本文从四个方面出发，站在用户和供应商双方的角度，采用水平伸缩的方式，通过时间序列分析技术将前瞻预测和反馈触发机制结合在一起，在保证 QoS 和 SLA 的基础上，寻求 Provision-Cost 平衡，考虑到云计算环境下弹性资源分配动态性与实时性的特点，因此本文提出了一种结合反馈与多级预测机制的敏捷弹性在线调度算法（AEOAS，Agile Elasticity Online Auto Scaling）来实现虚拟机的快速弹性供给，其中第一级预测采用 SARIMA-BPNN

算法预测下一时刻的资源量，对于预测误差采用二级预测 ARIMA 在线算法进行及时补偿，同时在资源分配过程中结合反馈信息完成负载量到资源量的映射过程。

2.3 本章小结

本章第一小节从不同角度介绍了弹性的定义以及弹性云服务的概念，分析了弹性的两个核心指标速度以及精确度的概念，并介绍了弹性中罚金模型的计算方法为后文的评测工作做准备，第二小节详细分析了弹性云服务相关技术在国内外的研究现状、各个技术的优缺点以及存在的问题，最后总结并提出了相应的改进方法，提出了本文的弹性调度算法的总体设计思想。

第 3 章 改进的动态 SARIMA-BPNN 在线预测算法

3.1 SARIMA-BPNN 预测算法设计思想

随着如今用户需求的转变，传统的调度算法越来越难以适应弹性云服务的需求，本文的预测算法主要用于解决云系统中发生资源供应不足的情况时由于资源分配滞后而引起的服务失效、SLA 违约的问题，以及发生资源供应过量时造成的大量资源浪费的问题，通过提前部署虚拟机确保服务质量、避免系统震荡。本文选用的 SARIMA 算法是加入了季节性因素的 ARIMA 算法，结合负载数据的周期性特征，采用 SARIMA 算法更能学习到数据中的周期信息。以 ARIMA 为基础的相关算法均对时间有较强依赖性的特征数据具有很好的预测性。但研究表明，ARIMA 算法对线性数据的拟合效果更好、预测结果更精确，对于非线性数据是通过对非平稳序列差分平稳化处理，变成平稳序列之后再进行移动回归。从原理上来看，该算法虽然可以用于对非线性数据的拟合，但当数据波动过大时，算法的平稳化原理也决定着相应时刻预测误差的增大。对于云环境下的用户需求，无论是用户行为的随机性还是用户规模的庞大性都导致了用于预测的负载数据具有很大的波动性，单纯使用 ARIMA 算法进行预测难以满足复杂的云环境对弹性的要求。对于系统负载的精确预测是弹性云服务面临的挑战之一，目前已有一些对 ARIMA 算法进行改进的研究，[86]中通过引入平均误差修改预测精确度；[87]中通过动态调整用于预测的历史窗口大小进而调整预测模型；[88]中通过对 ARIMA 进行傅里叶变换提高精确度；[89]中通过资源预测误差在下一预测周期进行补偿，进而保证误差不会进一步扩大。但是这些方法都是在算法本身的基础上进行改进，改进方法更多依赖于作者本身的经验，例如在下一个时间段应该如何补偿；历史窗口的大小随着预测误差的出现怎样调整。

针对这一问题，本文设计了一种自动式误差调整方式--通过神经网络的自学习方式进行调整。由于神经网络算法具有很强的自学习能力，并且对于非线性数据有较强的拟合性[90]，根据 SARIMA 算法存在的预测误差偏大难以满足弹性云服务要求这一问题，本文通过神经网络算法对 SARIMA 算法产生的误差进行学习，并用学习到的参数预测下一时刻 SARIMA 算法的预测误差，进而决定下一时刻的预测值，提高预测精确度。本文中选用了最常见的神经网络之一，前馈神经网络进行误差学习，虽然这种融合算法在医学、生物信息学等领域已经有过使用，但目前很少有人将其用于解决弹性云环境下的资源调度问

题，由于云环境下的资源弹性调度具有高度的实时性与动态性，当外界负载发生变化时，我们应该及时更新历史数据信息以及算法参数完成下一时刻的资源预测，因此本文在原有算法的基础上进行了改进，设计了适用于本文弹性云环境下的 SARIMA-BPNN 在线负载预测算法，实验表明该算法的预测精度要好于使用单一算法的预测结果，并且适用于负载突变明显的实时弹性云环境下的资源调度。

3.2 基于改进的动态 SARIMA-BPNN 在线负载预测算法

本文的 SARIMA-BPNN 在线预测算法主要包括两个部分，第一部分是采用加入了季节特征信息的 SARIMA 在线算法实时预测下一时刻的负载请求信息，预测结果连同当前时间信息一同作为 BPNN 预测算法的输入，通过神经网络学习到的参数，预测出下一时刻 SARIMA 算法的预测误差，两者相加后的结果作为下一时刻的最终预测结果。

3.2.1 ARIMA 预测模型

ARIMA 模型（AutoRegressive Integrated Moving Average Model，简记为 ARIMA）又称为 box-jenkins 模型[91]。根据原时间序列平稳性和回归项的差异性，可以分为自回归过程（AR）、移动平均过程（MA）、自回归移动平均过程（ARMA）和自回归积分移动平均过程（ARIMA）[81]。根据本实验负载特征，本文选用了差分自回归移动平均模型 ARIMA（ p,d,q ）进行预测，其中 p 为自回归项数（即 p 阶自回归）， q 为移动平均项数（即 q 阶移动回归），d 为对非平稳序列平稳化过程所需要的差分次数（即 d 次差分）。该模型既考虑了时间序列在时间上的依存性，又考虑了随机因素的干扰性波动，因其具有较高的预测精度而被广泛采用。其模型形式如（3-1）所示：

$$\begin{cases} x_t = \varphi_1 x_{t-1} + \varphi_2 x_{t-2} + \ldots + \varphi_p x_{t-p} + \varepsilon_t - \theta_1 \varepsilon_{t-1} - \theta_2 \varepsilon_{t-2} - \ldots - \theta_q \varepsilon_{t-q} \\ w_t = \Delta^d x_t \\ \Phi(L) w_t = \Theta(L) \varepsilon_i \\ E(\varepsilon_t) = 0, Var(\varepsilon_t) = \sigma^2, E(\varepsilon_t \varepsilon_s) = 0, s \neq t \\ E x_s \varepsilon_t = 0, \forall s < t \end{cases} \tag{3-1}$$

其中 $x_i, i = 1, 2, \ldots, n$ 是随时间变化的负载量时间序列， $\varepsilon_i, i = 1, 2, \ldots, n$ 为均值为零的白噪声序列， $\varphi_i, i = 1, 2, \ldots, n$ 和 $\theta_i, i = 1, 2, \ldots, n$ 分别为模型的一系列待定系数，p 和 q 分别代表自回归项和移动平均项数，因而 $\Phi(L) = 1 - \varphi_1(L) - \cdots - \varphi_p(L)$ 和 $\Theta(L) = 1 - \theta_1(L) - \cdots - \theta_q(L)$ 分别代表着平稳可逆的自回归移动平均模型的自回

归系数多项式和移动平均系数多项式；其中 ARIMA（ p , d , q ）模型经过 d 次差分变成了平稳序列 w_t ，其中 $w_t = \Delta^d x_t = (1-L)^d x_t$ 表示时间序列的 d 阶差分过程，于是可对 w_t 建立 ARMA（ p , q ）模型 $\Phi(L) w_t = \Theta(L) \varepsilon_i$ ，由此转化为 ARMA 模型进行下一步的拟合，$\Delta^d x_t = \Theta(L) \varepsilon_t / \Phi(L)$ 即是时刻 t 的预测值，但是此值需要经过逆差分操作恢复到原来区间。ARIMA 建模流程如图 3-1 所示。由前面的介绍可知，ARIMA 与 ARMA 模型的不同之处主要在于时间序列是否平稳，而平稳性是序列回归的前提条件，因为我们在建模的过程中很多是建立在大数定理和中心极限定理满足的前提下，定理中要求的样本同分布原则等同于时间序列中平稳性概念，若不满足则得到的很多结论都是不可靠的，因此平稳性检查是必不可少的步骤。单位根检验法是经常使用的检验序列平稳性的方法，实际应用中的时间序列通常具有季节性、趋势性和随机性三个特征，因而大部分的 Web 应用负载都是具有季节性的非平稳时间序列，通常的平稳化方法有对数变换、指数平滑法、差分、移动平均等。本文通过对原始序列进行 d 阶差分将其转化为平稳时间序列，之后用 ARMA（ p , q ）模型进行拟合，模型拟合的过程通过自相关系数（ACF）和偏相关系数（PACF）确定拟合的模型 AR（ p ）、MA（ q ）中 p 、 q 的值，在通过一系列的参数估计、假设检验的过程，确定其是否有统计意义以及残差序列是否为白噪声序列。参数估计的方法包括最小二乘法、最大似然法和矩估计等，检验其统计意义同时确定模型中的未知参数。通过拟合的残差序列分析模型的有效性，最后完成预测分析。从建模、分析、运行到精确度整体代价来看，使用 ARIMA 方法进行预测符合本文对弹性预测算法的要求。

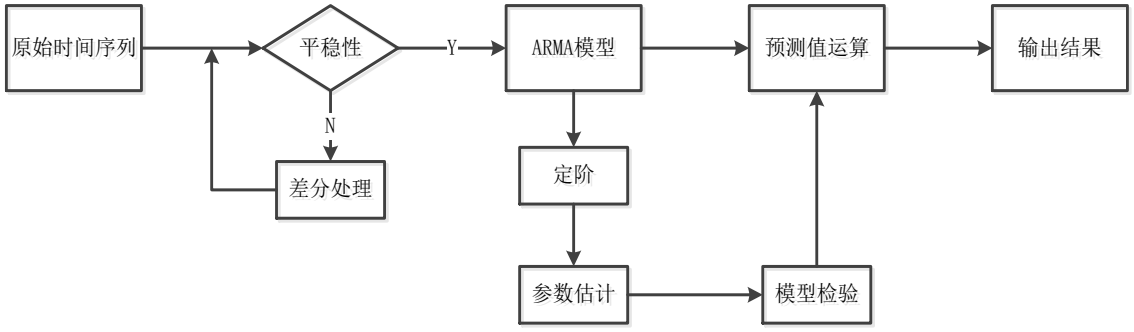


图 3-1 ARIMA 建模流程

实现过程为：

Step1：在时刻 t ，收集 x_{t-l} □ x_t 长度的访问量信息作为历史数据，按照 ARIMA 建模过程建模；

Step2：根据建立的模型预测 $t+\Delta T$ 时刻的访问量数据；

Step3：在$t+\Delta T$时刻真实数据到来之后更新历史数据列表，将最新数据加入到历史记录中，并将相隔最远的数据移出，确保数据的实时更新；转 Step1 直到算法结束。

由于本文的算法主要针对负载进行预测，在某种程度上用户行为具有很好的周期性特征，加入周期性信息将更有利于模型拟合，提高数据在时间上的相关性。因此本文以 ARIMA 算法为基础，采用了加入季节信息的 SARIMA 算法进行预测，设计过程如下所述。

3.2.2 带有季节信息的 SARIMA 预测模型

在某些时间序列中，由于季节性变化（年、季、月、周、日等）以及某些固有因素存在而引起的具有明显周期性特征的时间序列就称为季节性时间序列（SARIMA）[91]。本文的算法在 ARIMA 模型的基础上加入了季节性因素，设其季节周期为 s，周期的选择根据实际应用场景可以自行调整，模型表示如（3-2）所示。

$$\begin{cases} x_t = \alpha_1 x_{t-s} + \alpha_2 x_{t-2s} + \ldots + \alpha_P x_{t-Ps} + \varepsilon_t - \beta_1 \varepsilon_{t-s} - \beta_2 \varepsilon_{t-2s} - \ldots - \beta_Q \varepsilon_{t-Qs} \\ w_t = \Delta_s{}^D x_t \\ A_P\left(L^s\right) w_t = B_Q\left(L^s\right)\varepsilon_i \\ E\left(\varepsilon_t\right) = 0, Var\left(\varepsilon_t\right) = \sigma^2, E\left(\varepsilon_t \varepsilon_s\right) = 0, s \neq t \\ Ex_s \varepsilon_t = 0, \forall s < t \end{cases} \tag{3-2}$$

其中$x_i, i = 1,2,..,n$是随时间变化的具有周期性的时间序列，$\varepsilon_i, i = 1,2,\ldots,n$为均值为零的白噪声序列，$\alpha_i, i = 1,2,\ldots,n$和$\beta_i, i = 1,2,\ldots,n$分别为模型的一系列待定系数，$P$和$Q$分别代表季节性自回归项和移动平均项最大滞后阶数，$A\left(L^s\right) = 1 - \varphi_1 L^s - \varphi_2 L^{2s} - \ldots - \varphi_P L^{Ps}$和 $B\left(L^s\right) = 1 + \theta_1 L^s + \theta_2 L^{2s} + \ldots + \theta_Q L^{Qs}$分别称为季节性自回归算子和季节性移动平均算子，$\Delta_s{}^D = \left(1 - L^s\right)^D$ 代表D阶季节性差分，由此确定 SARIMA（P, D, Q）模型。

考虑到本文的实际负载序列同时具有季节性与非季节性的特点，因此本文综合（3-1）与（3-2）所示的非季节性与季节性模型，建模过程如下：（1）平稳化处理：由原来的d阶差分变为d阶非季节性差分和D阶季节性差分，表示为$w_t = \Delta^d \Delta_s{}^D x_t$；（2）模型定阶：$P$、$p$、$Q$、$q$分别代表季节性、非季节性的自回归、移动平均算子的最大滞后阶数，模型表达式为$\Phi_p(L) A_P\left(L^s\right)\left(\Delta_s{}^D \Delta^d x_t\right) = \Theta_q(L) B_Q\left(L^s\right)\varepsilon_i$，通过 AIC 准则确定包括$d$、$D$、$P$、$p$、$Q$、$q$在内的参数，最终确定$(p,d,q)\times(P,D,Q)_s$模型。（3）后续参数估计、

模型检验、预测运算的过程与 ARIMA 模型相同，之后采用 BPNN 算法拟合预测误差。

3.2.3 BPNN 模型

BP 神经网络（BPNN）对于非线性数据的处理具有独到的优势[92]，也是最常见的人工神经网络（ANN）模型之一。它是一种利用反向传播算法将误差逆向传播来进行训练的前馈神经网络。一般来说，BPNN 由输入层、隐藏层和输出层组成，其中输入和输出均为一层，隐藏层可以有一层或多层。在神经网络中每层均可包含多个节点，相邻两层任意节点间包含一条从上层节点到下层节点间的信号传输边。输入信号从输入层进入网络，经各个节点的处理逐层传播，最后由输出层输出。

其中，网络中的隐藏层和输出层节点也称之为感知器，每个感知器的输入是一个数值型向量，到达的输入向量经过感知器的权重和偏斜，计算对应输入向量的组合函数。一般来说这一组合结果还要经过一个激活函数的处理，最后得到的激活值即为上一层输入向量经过该感知器处理后得到的输出值，同样该输出值与由同层感知器处理后得到的输出值一起构成下一层感知器的输入向量，整个数据通过前一层的输出作为下一级输入的过程逐级传递直至到达输出层，经过输出层激活函数的处理产生回归结果，通常神经网络中常用的激活函数通常有 sigmd、tanh 和 relu 等。上述过程既为经过该网络处理的数据流走向也是神经网络拟合非线性数据进行回归预测的过程，图 3-2 表示了其中一个感知器模型示意图，其中感知器的前馈表达式如公式（3-3）所示：

$$z = activation\left(\sum_{i=1}^{n} w_i * x_i\right) \tag{3-3}$$

其中 x_i 、 w_i 分别代表第 i 个输入值以及该感知器对应该输入值的权值，对 w_i 来说， $i>0$ 时表示输入节点 $x_i\left(1 \le i \le m\right)$ 到隐藏节点的权重， $i=0$ 时表示到该隐藏节点的偏斜， z 表示经过激活函数 $activation$ 后的输出值。

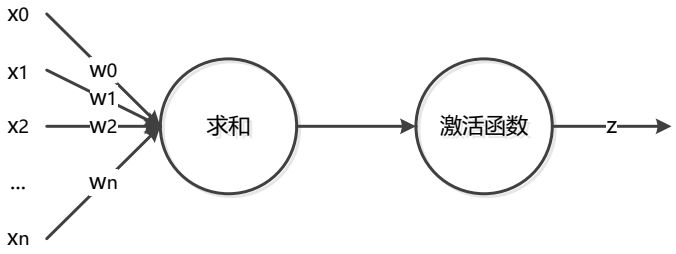


图 3-2 感知器模型

通过大量实验对比，本文最终选用了三层 BPNN 来进行数据拟合，本文采用的网络中包含一组输入层节点，在这里用向量 x 表示，对于每一条输入向量 p 代表一条实例长度；包含若干隐藏层节点以及一个输出层节点，用来拟合预测结果。我们用向量 h 表示隐藏层的激活值，y' 表示输出节点的输出结果也就是拟合结果，y 表示真实值。经实验对比，我们用 tanh 作为隐藏层节点的激活函数，sigmod 作为输出层节点的激活函数，这样的设置相比使用其他激活函数得到的效果更好。其网络实现如图 3-3 所示，W 为输入层到隐藏层的权重矩阵，如公式（3-4）所示。

$$W=\begin{vmatrix} W_{01} & \cdots & W_{0n} \\ W_{11} & \cdots & W_{1n} \\ \cdots & \cdots & \cdots \\ W_{m1} & \cdots & W_{mn} \end{vmatrix} \tag{3-4}$$

矩阵中的 W_{ij} 在 $i>0$ 时表示输入节点 $x_i\left(1\le i\le m\right)$ 到隐藏层第 j 个节点上的权重，i=0 时表示隐藏层第 j 个节点的偏斜，如图 3-3 中 x_0 所示。

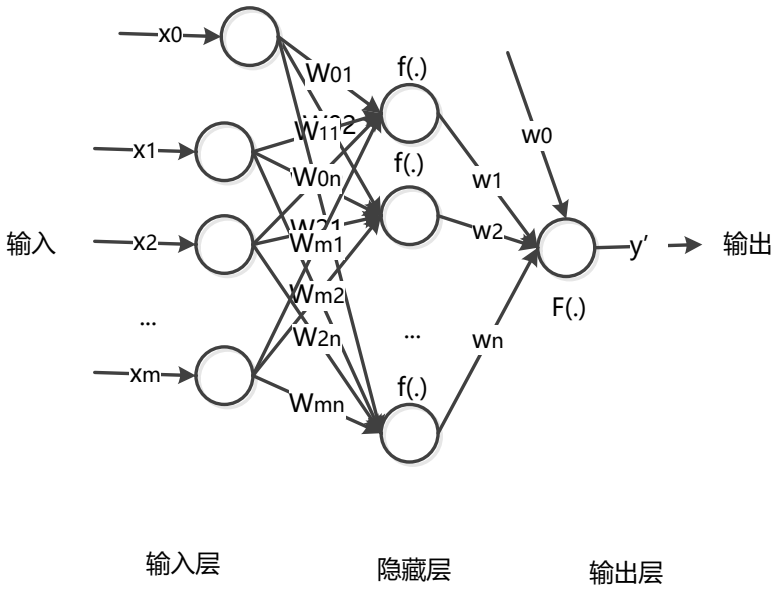


图 3-3 BPNN 网络图

我们令 x_0= 1，它不表示任何输入。w 表示从隐藏层到输出节点的权重向量 $w=\left[w_0,w_1,w_2,\ldots,w_n\right]$，同理 w_i 在 i > 0 时表示 h_i 到输出节点的权重，i=0 时表示输出节点上的偏斜，令 h_0= 1，其中 h_i 的计算公式如（3-5）所示，输出结果如公式（3-6）所示。

$$h_i=\tanh\left(\sum_{i=0}^{m}x_iW_{ij}\right) \tag{3-5}$$

$$y' = sig\,\mathrm{mod}\left(\sum_{i=0}^{n} w_i h_i\right) \tag{3-6}$$

$f(.)$、$F(.)$分别对应着隐藏层和输出层的激活函数 tanh 和 sigmod，i=0 分别对应着偏斜值。

由于 BPNN 中采用反向传播算法来训练网络，根据训练样本的误差对权重 W 和 w 进行更新。例如，对于一条训练样本来说网络的误差表示为（3-7）：

$$E = \frac{1}{2}(y - y')^2 \tag{3-7}$$

根据误差的定义以及前馈过程的表达式可以推导出误差关于权重矩阵或向量的梯度。反向传播算法用梯度下降法更新权重与偏置值，其思想是让权重矩阵或偏置向量沿着梯度相反的方向更新。其更新函数表达式如下，b 由 W_0、w_0 表示：

$$W \leftarrow W - \alpha \frac{\partial E}{\partial W} \tag{3-8}$$

$$w \leftarrow w - \alpha \frac{\partial E}{\partial w} \tag{3-9}$$

其中 $\alpha > 0$ 代表训练过程中的学习率，表示了权重更新的快慢。α 的取值对于网络的训练非常重要，取值过低会导致模型收敛缓慢，取值过高又会造成拟合效果震荡，通过反复实验，我们把学习率定为 α = 0.1。其中 BPNN 算法实现过程如下：

Step1：数据归一化；

Step2：使用随机函数初始化隐藏层和输出层权值矩阵与偏置向量；

Step3：确定输入样本信息并投入网络进行训练；

Step4：按照梯度下降算法对权值矩阵以及偏置向量进行更新；

Step5：重复 step3 和 step4，直到误差小于某一特定值或者达到迭代次数，算法结束。

3.2.4 SARIMA-BPNN 模型

本文的预测算法就是在上述模型的基础上设计完成的，我们充分考虑到预测结果中线性与非线性因素并存的情况，用 SARIMA 模型拟合数据中线性部分，并将其输出作为 BPNN 算法输入的一部分，设计了 SARIMA-BPNN 在线预测模型。在 BPNN 模型中，我们将 SARIMA 模型的预测值和时间信息组合为样例的特征向量，将 SARIMA 预测结果的误差作为拟合目标。其中时间信息分为星期和小时两部分，星期代表了一周当中第几天的信息，我们用一个长

度为 7 的向量来表示，例如$[1,0,0,0,0,0,0]$第一个位置为 1 其余位置均为 0，表示这一天为星期一；同样小时代表了一天的时间信息，我们使用长度为 24 的向量来表示，例如$[1,0,0,\ldots,0,0]$共 24 位表示该天的第一个小时。因此我们构造了一个含有 32 个输入节点的 BPNN 网络用来拟合上一预测结果的误差。隐藏层节点数设置为 64 个。

我们用目前流行的 TensorFlow 来实现 BPNN 网络并对其进行训练。TensorFlow 是谷歌推出的一款开源软件库，它既可以用于包括深度神经网络在内的机器学习模型的构建，也可以用于其他领域的数值计算。计算所用的模型在 TensorFlow 中称之为图，图中的每个节点表示一个数学操作，因此节点也成为操作，线表示节点间的数据传递，传递的形式为多维数组，也就是张量（tensor）。另外，TensorFlow 支持使用 GPU 来加速张量的运算，从而大幅度提升神经网络模型的训练速度。用 TensorFlow 进行机器学习任务的过程可以分为构建图和训练两个阶段。根据我们使用的 3 层 BPNN 网络，我们用代码构建了 TensorFlow 流图，为了更直观的展示，我们将它抽象为图 3-4 所示的示意图。

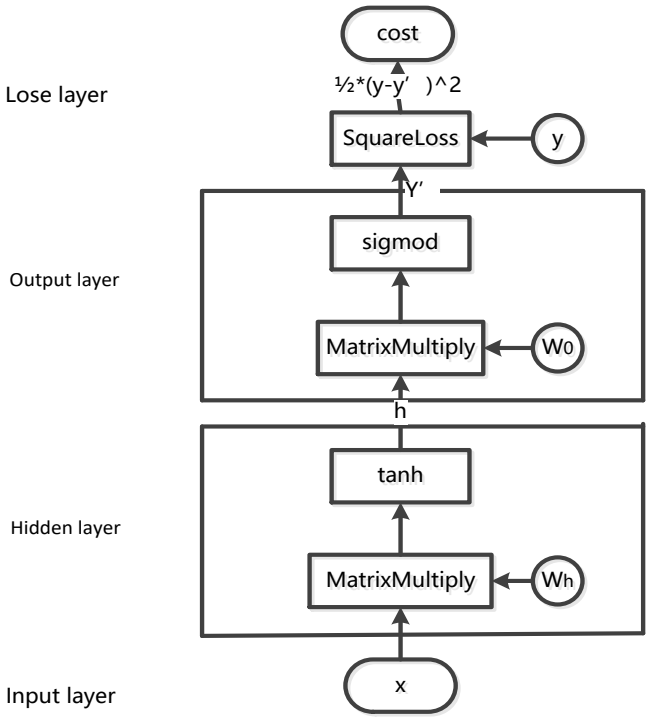


图 3-4 TensorFlow 构建模型

图中用 TensorFlow 构建的 BPNN 模型分为 4 层，即 BPNN 网络结构中的输入层、隐藏层、输出层以及训练模型所用的损失函数层。在 TensorFlow 流图中网络各层的多节点被整合成向量，而节点间的权重连线被整合成矩阵，它们都统一用张量的形式表示。输入层为模型的输入 x，它来自样例的特征

向量，向量的长度即为输入层节点的个数。隐藏层通过矩阵的乘法运算将输入向量的线性组合传入隐藏层各节点，再经过激活函数 tanh 得到隐藏层激活向量。输出层与隐藏层结构相似，激活函数采用了 sigmod，我们的网络输出层只有一个输出结点，它的输出就是样例的拟合结果。在损失函数层，我们用平方误差（squaer loss）作为模型的误差并输出为 cost，整个网络训练的目的是通过梯度下降法调整模型中的隐藏层参数矩阵 W_h 和输出层参数向量 W_o，使误差 cost 尽可能的最小化。

算法描述：

Algorithm3-1：SARIMA-BPNN

输入：负载时间序列 L、周期 T、样例位数 feature_num、训练样例条数 batch_size、迭代次数 n、隐藏层节点数 hidden_size

输出：最终预测值 forecast

```
1. Forc ← SARIMA(L_T)     //forecast workload by SARIMA model
2. 创建输入样例[SARIMA,weekday,hour]和输出样例[request-forecast]
3. 创建[feature_num,hidden_size,1]形状的神经网络，初始化权值
4. define loss founction and average error
5. train_step = tf.train.GradientDescentOptimizer(0.1).minimize(loss)//梯度下降法
6. for t in range(n):
       if   i - batch_size >= 0:
            神经网络训练最近的 batch_size 大小的样例
       else:
            sess.run(train_step, feed_dict={xs: x[:i], ys: y[:i]})//全部投入训练
7. compensate ←将 Forc 投入网络预测误差值
8. forecast = Forc+compensate
```

由于我们采用的是 SARIMA 和 BPNN 的组合模型对序列进行在线预测，所以 BPNN 的训练和预测过程应该是在线的。当我们获得真实访问量序列的最新值时，我们结合 SARIMA 模型之前对该访问量的预测值以及它对应的时间信息构成一条新的样例。我们需要将这条新的样例投入 BPNN 模型中对模型进行在线训练，为了达到更好的训练效果，通常 BPNN 模型要求每条样例多次训练。但是如果集中对某条样例多次训练，非常容易造成模型陷入局部最优解或过拟合于该条样例的特性，因此我们不仅仅用最新的样例进行训练，而且用最新的多个样例进行模型的训练。在这里我们设置每次新样例到来时，用

最新的 128 条样例重复训练模型 1000 次。这样做还有另一个好处，随着时间的推移，旧的数据将不再参与训练，它们对模型的预测结果的影响将会越来越小，这符合一般的时间序列的特点。用最新的样例训练完模型后，结合 SARIMA 模型对下一个访问量的预测值，我们可以用 BPNN 来预测其误差，并将两者相加作为组合模型的预测结果，算法描述如算法 3-1 所示。

3.3 实验结果及分析

3.3.1 负载数据准备

本文选用的实验验证数据是具有典型云用户需求特征的负载数据，通过对获取的 98 年世界杯足球赛 Web 直播网站连续三个月的访问日志进行解析[93]，分析了该网站完整的 Web 访问记录。由于原始文件是二进制形式，本文根据实验需求按照小时为单位进行统计（云计算环境下的虚拟机租用最后单位为小时），获取了该期间内的负载曲线。

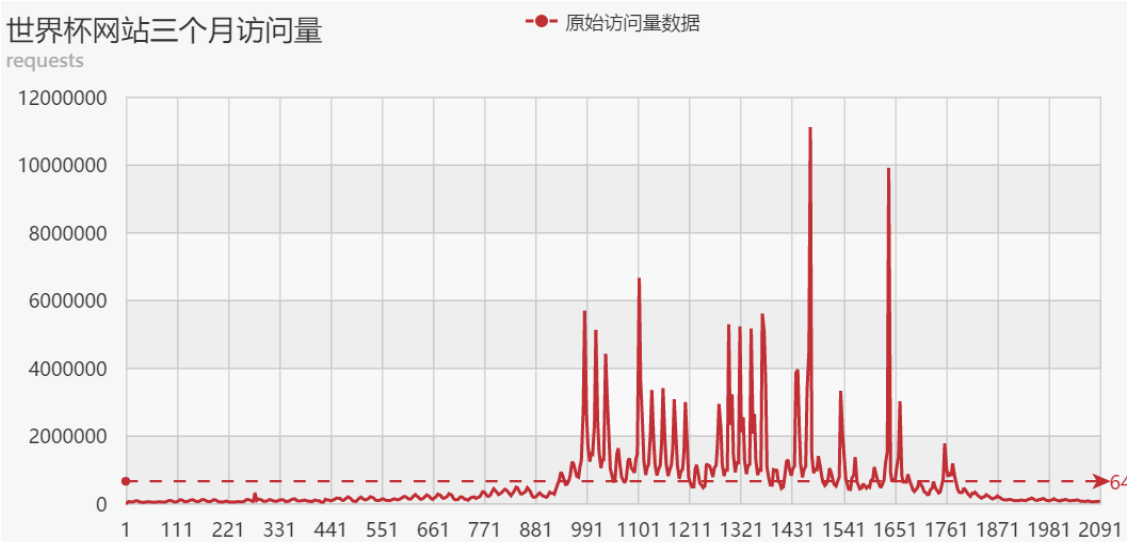


图 3-5 原始数据量曲线图

从相关统计信息中发现：该 Web 网站的实际访问量达到 135 亿且主要集中在比赛期间的一个月当中，从 6 月 10 日开始急剧增长到 7 月 1 日达到最高点，其他时间相对平缓但仍然高于正常访问量，符合突变型负载。通过对其负载变化情况的分析发现该网站访问量随时间变化明显，并且具有一定的周期性和季节性，比如：周末的访问量会比较低，因为更多的人会选择通过电视观看，而工作日期间大多人只能通过网站观看；又比如每天有两个时间段会出现爆炸性增长，如法国时间的下午五点半（这是由其赛事时间决定的），这些增长一般都是突发性的，但在较长的时间尺度上（例如：小时以上）这些爆发性是可以

预见的。图 3-5 为统计的访问量信息，该负载符合突变类型且可以用来进行预测，用该负载设计完成实验具有可行性。

3.3.2 预测结果分析

为了评价预测算法的准确性，本文采用了预测分析中最常使用的两个指标平均绝对误差（MAE）和平均均方根误差（RMSE）来作为评价依据，其中 MAE 能更好的反映出预测值误差的实际情况，其计算公式如下（3-10）所示：

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|f_i - y_i| \tag{3-10}$$

其中 f_i 表示预测值， y_i 表示真实值。RMSE 则可以评价数据的变化程度，RMSE 的值越小说明用预测模型来描述实验数据具有的精确度更高，其计算公式如下：

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - f_i)^2} \tag{3-11}$$

由于基于时间序列分析的预测算法对于历史数据窗口的大小、预测步长的选取都有很大的依赖性,。本文的预测步长充分结合云环境下的实际使用特征，选择了虚拟机最小计费单位（1h）；而历史数据窗口的选择如图 3-6 所示，本文以小时为跨度、以 24 小时为周期，分别测试了 1-7 天数据长度的历史窗口的预测精确度，采用 MAE 和 RMSE 两个指标来进行衡量。发现 MAE 从一天到六天依次递减，之后会随着历史数据窗口的增加依次变大；RMSE 随着历史窗口的增加依次降低并在 6 天 144 个小时时达到最低，之后呈现平缓上升的趋势。因此本文选用 L=144 作为历史数据窗口的长度，进行下一时刻的预测。

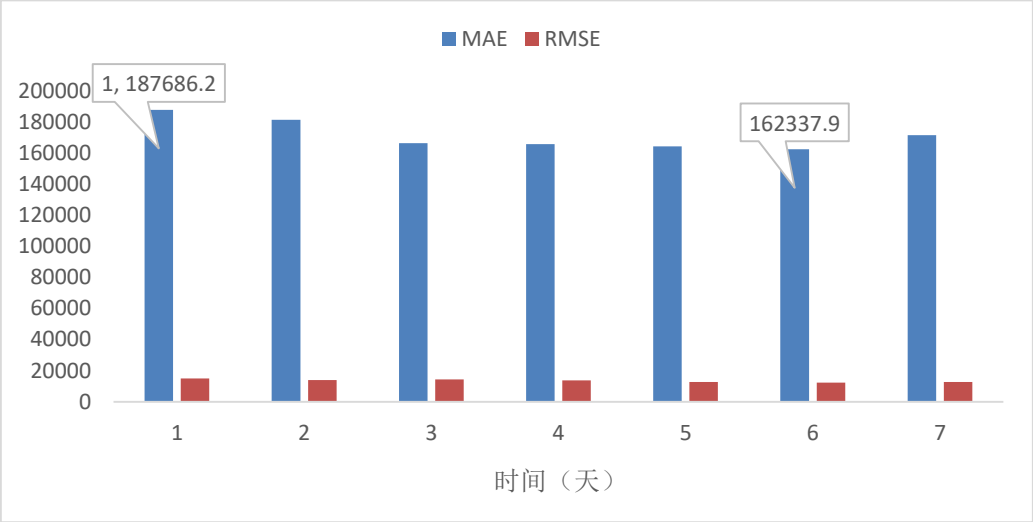


图 3-6 不同历史数据长度下预测精确比较

为了便于说明实验结果，本文在这里选择了世界杯期间 6 月 19 日到 6 月 24 日之间六天 144 个小时访问量数据用于在线预测 6 月 25 日一天的访问量信息，其中历史数据访问量变化趋势如图 3-7 所示。

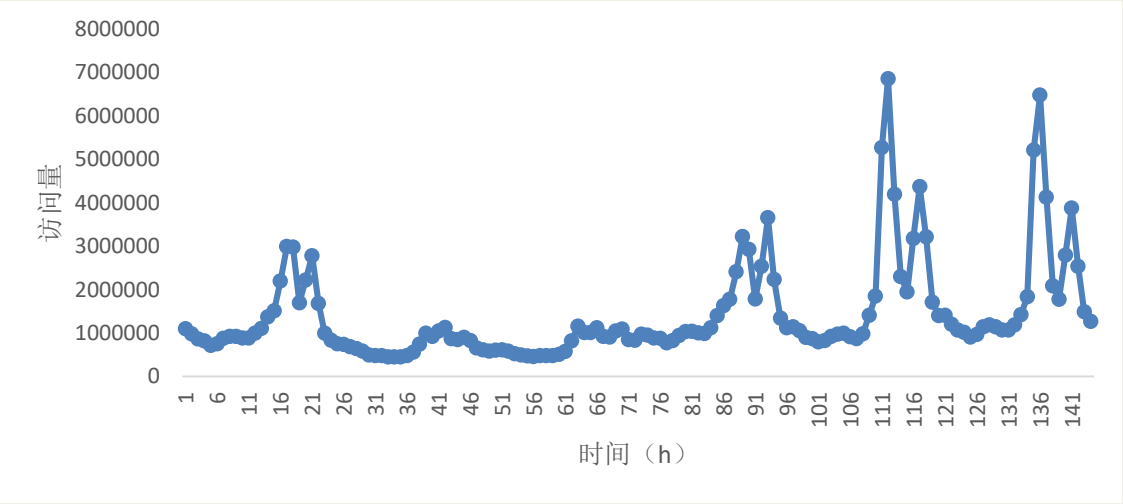


图 3-7 历史数据信息

图 3-8 表明了不同预测算法之间的预测结果，其中紫色曲线代表了真实负载值，蓝色曲线代表使用 SARIMA 算法进行预测的结果，红色曲线代表了使用神经网络模型进行预测的结果，绿色曲线代表使用本文 SARIMA-BPNN 融合算法进行预测的结果。

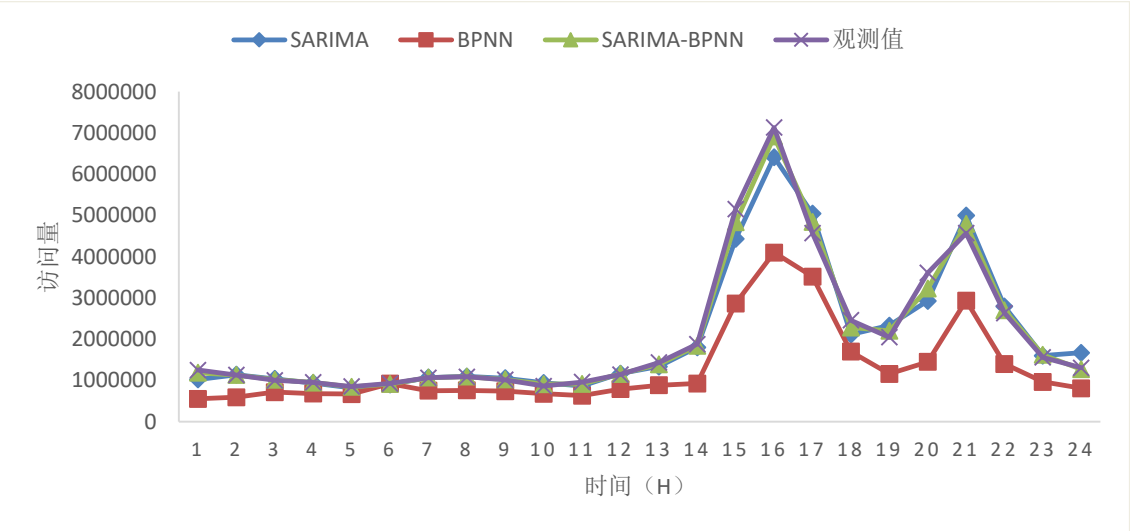


图 3-8 预测结果对比图（一）

图 3-9 表明了本文中改进的 SARIMA-BPNN 预测算法与[86]中提出的 ARIMA-improve 算法的比较结果，其中红色表示真实负载值，蓝色表示 SRAIMA-BPNN 预测结果，绿色表示使用 ARIMA-improve 进行预测得出的结果。

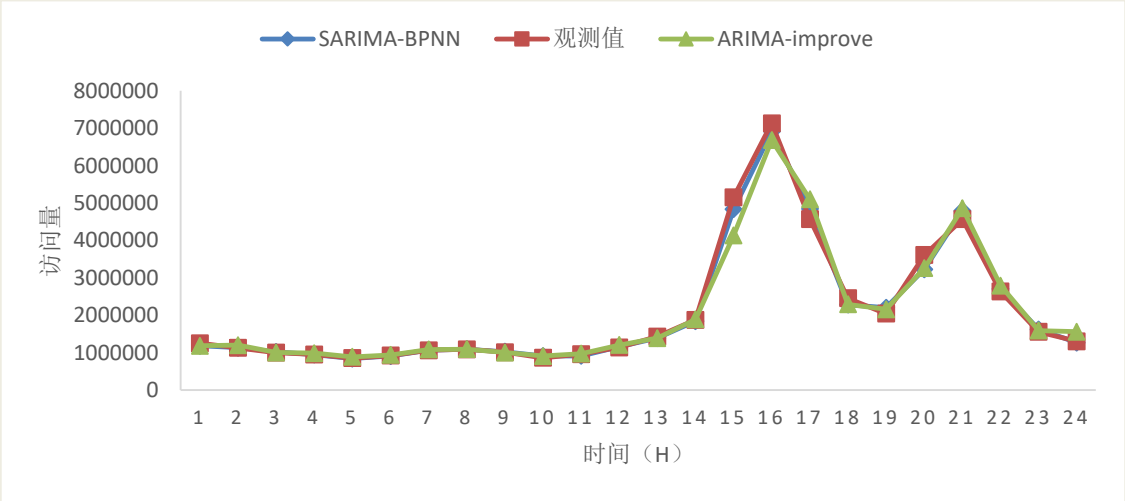


图 3-9 预测结果对比图（二）

不同算法的预测误差指标如表 3-1 所示，数据表明使用两者结合的 SARIMA-BPNN 预测方法要优于使用单一预测方法，其中平均绝对误差（MAE）相较于 SARIMA 和 BPNN 分别提高了 51%和 88.5%，平均均方根误差（RMSE）相较 SARIMA 和 BPNN 分别提高了 49.1%和 86.9%，可见使用 SARIMA-BPNN 进行预测的准确度在使用单一算法的基础上有很大的提升，此外，平均绝对误差百分比（MAPE）相较于 SARIMA 的 7%和 BPNN 的 35%提升到 4.5%，可以很好的拟合出负载变化趋势。另外，本文中的 SARIMA-BPNN 预测算法的平均绝对误差比 ARIMA-improve 降低了 41%，平均均方根误差降低了 44.6%，平均绝对百分比误差降低了 19%，因此本文的 SRAIMA-BPNN 算法有更高的预测精确度，且改进算法具有合理性。从全局来看，采用本文的预测算法，误差在十万范围内的准确率达到 90%以上，因此，采用本文的预测算法可以满足弹性云环境下的性能需求。

表 3-1 不同算法误差比较

算法名称	MAE	RMSE	MAPE
SARIMA	190554.239	283828.347	0.07745
BPNN	808449.167	1100518.06	0.35264
SARIMA-BPNN	92951.25	144372.588	0.04544
ARIMA-improve	159000.167	260796.337	0.05623

3.4 本章小结

本章主要介绍了基于改进的动态 SARIMA-BPNN 在线负载预测算法，通过对目前预测技术与预测方法的研究提出适用于本文负载特征的 ARIMA 预

测算法，并针对该算法存在的问题提出若干改进：首先，原有算法不适用于实时动态变化的弹性云环境，本文结合云环境的特点通过实时更新预测模型设计了在线 ARIMA 预测算法；其次，针对本文负载数据的周期性与季节性特点，在原有算法基础上加入了季节特征，形成 SARIMA 算法；再次，由于原有算法在应对波动较大数据时相应的数据偏差会增大从而产生具有明显非线性特征的预测误差，为了提高预测精确度本文加入适用于拟合非线性数据的神经网络算法拟合预测误差，同时为了适应本文的弹性云环境提出了在线 BPNN 模型拟合误差。最后设计实验，验证预测算法的准确度。

第 4 章 基于 CloudStack 平台的 AEOAS 弹性调度

4.1 基于 CloudStack 的资源弹性调度器的设计

本文使用了 CloudStack[94]作为弹性资源管理平台，在其上部署了适用于云环境下的弹性应用提供弹性服务，并运用弹性管理流程（MAPE 循环）完成本文的弹性调度过程。

4.1.1 弹性云服务管理流程-MAPE 循环

为了保证云服务的弹性性能，一个高效的云管理系统是十分重要的。由于云服务中的资源是以虚拟机为最小单位进行调度的，若没有一个高效的管理平台会存在如下几个问题：（1）每台虚拟机都需要配置操作系统；（2）每台新申请的虚拟机都需要经历镜像加载、应用重配置等过程，大大延长了服务可用的延迟时间；（3）没有统一的管理平台，虚拟机之间彼此独立，会导致资源分配不均衡现象的产生。因此本文采用了学术界广泛使用的 MAPE 循环完成资源弹性管理。

MAPE 循环包含四个阶段：Monitoring (M)、Analyser (A)、Planning(P)和 Execution (E)。首先，负载与资源监控模块（M）主要负责收集系统进行资源管理所需要的当前系统信息，通过云基础设施提供商提供的 API 接口，实时获取所监控基础设施的基本信息作为资源调度器进行资源调度的输入。资源的弹性管理主要体现在资源弹性分析模块（A）和资源弹性规划模块（P）。资源弹性分析模块（A）主要用于处理从监控模块中获取到的当前系统资源负载信息（如 CPU 资源利用率）和系统请求负载信息，之后用加载在其上的弹性调度策略进行相应的资源分析，并完成对未来资源使用量与需求量的预估；资源弹性分析模块主要按照加载策略按照相应的预测周期进行资源预测，并将得到的预测结果提交给资源规划模块形成相应的指令。资源规划模块（P）主要完成弹性伸缩与 SLA 之间的折衷，通过获取资源分析阶段的结果，一旦系统的当前状态可知或未来状态可预知，则按照预先制定的资源分配策略进行资源映射，形成相应的扩展或收缩列表，在适当时刻交给执行模块执行。执行模块（E）主要用来执行经过前面的弹性策略所提供的伸缩列表，例如待创建的虚拟机类型以及数量，待移除的虚拟机 id 等，此步骤可通过直接调用云服务提供商提供的 API 实现。

4.1.2 基于 CloudStack 的弹性资源调度管理

本文搭建了 CloudStack 平台用于资源弹性管理，在弹性管理的过程中采用 MAPE 循环，弹性管理机制如图 4-1 所示：

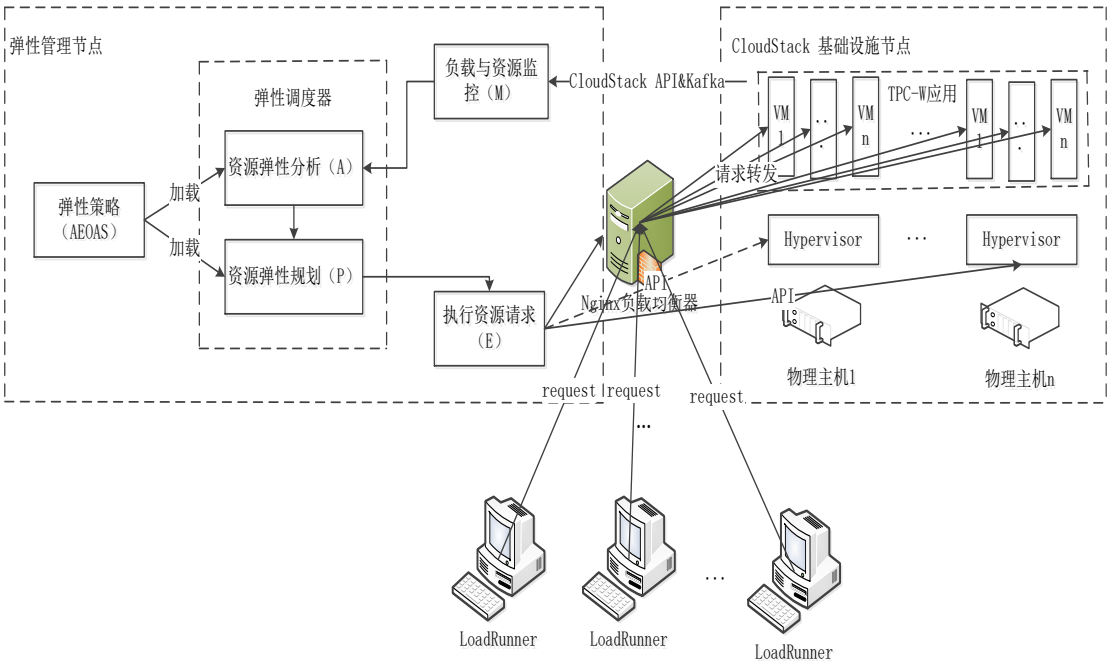


图 4-1 基于 CloudStack 的弹性管理

其中（M）Monitor 监控器主要用于完成系统资源的监控和负载信息的收集，监控资源的获取主要通过 CloudStack 提供的 API 接口读取实时资源监控信息，负载采集则通过搭建的 Kafka 消息队列机制实时采集系统日志，并解析出所需负载信息；（A）Analyzer 和（P）Planner 模块为弹性策略所在，也是系统中的弹性调度器部分：Analyzer 利用从监控器中获取到的负载访问量信息完成资源预测，并将预测结果传给 Planner 模块；Planner 模块根据 Analyzer 模块预测的请求量信息以及监控模块收集到的当前系统资源信息完成从请求量到资源量的映射过程，生成相应的资源伸缩列表控制（E）Executor 模块形成相应的指令通过调用 CloudStack 的 API 完成资源伸缩的过程。

本文将 Kafka[95]集群作为一个日志采集组件集成到应用系统中，用于系统实时日志的采集。一般 Kafka 连接通信的两端分别为 Producer 和 Consumer，本文中用于发布消息的 Producer 为集群中参与弹性伸缩的虚拟机应用实例，每台虚拟机作为一个 Producer 将 tomcat 访问日志发布到 Kafka 集群，因此集群中所有提供服务的虚拟机构成 Kafka 集群的 Producer；消息订阅端 Consumer

在本文中代表着消息处理服务器，通过对采集的系统实时访问日志进行收集与处理，生成每个特定周期下的访问量信息，提供给 Monitor 监控器进而用于 Analyzer 的弹性调度。

4.1.3 基于 CloudStack 的 TPC-W 弹性应用部署

CloudStack 作为一个弹性云管理平台，需要在其上部署一个能提供弹性云服务的应用程序。这个应用程序需要在负载增加时，自动增加可以提供服务的资源；在负载降低时，自动减少提供服务的资源；并且在资源动态伸缩的过程中维持保证服务质量所需要的性能指标。

TPC-W[96]作为一个事务性的 Web 基准测试程序，通过仿真一个在线电子书店的情景，进一步明确地模拟出顾客在网上商店浏览并购买商品的过程，从而用于测试不同软硬件运行环境的相关性能。但是由于 TPC-W 是一种传统的性能评测标准，其性能指标和本身的体系结构并不适用于当前的弹性云环境，因此将其作为一种应用向云环境下移植，并重新设计了符合云环境特征的部署结构如下图所示。

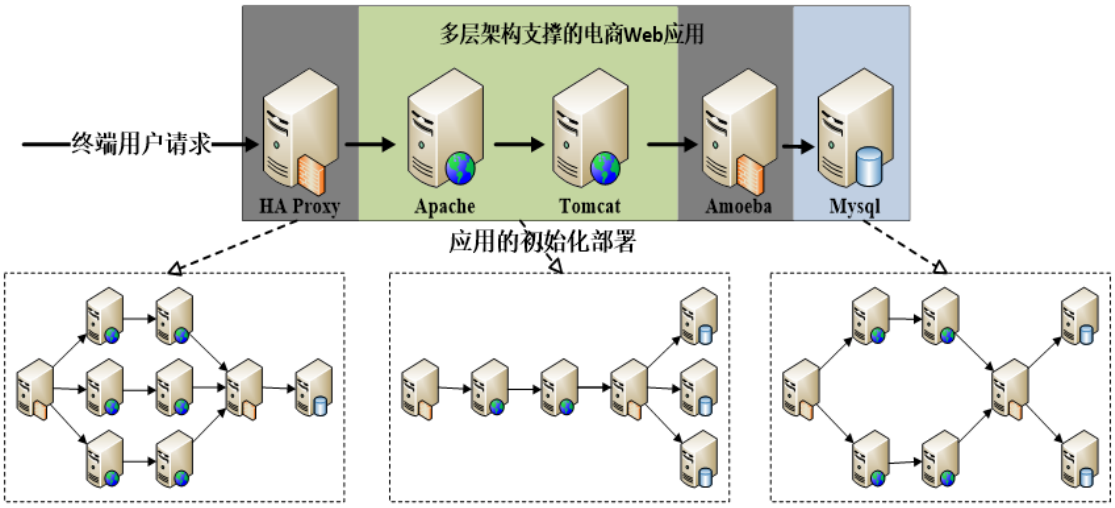


图 4-2 TPC-W 部署架构

其应用程序的层次组成依次为：HAProxy（High Availability Proxy）负载均衡服务器、Apache HTTP 服务器、Tomcat 应用服务器、Amoeba 数据库 Proxy 和 MySQL 数据库服务器。这些多层次服务器共同工作来处理终端用户的请求，维护系统整体性能，支撑弹性伸缩。其中，本文中的 HAProxy 负载均衡服务器采用 Nginx 实现，通过负载均衡将发送的请求进行分配，从而转发到集群当中。

4.2 弹性调度算法 AEOAS 的设计与实现

本文的调度算法主要集中在 Analyzer 和 Planner 组成的调度器部分，通过监控器中收集到的相关信息决定系统是否需要进行资源伸缩，以及伸/缩的时间、虚拟机类型以及数量等。

本文在调度过程中采用了预测机制，通过上章所述的 SARIMA-BPNN 在线算法作为一级预测机制进行负载信息的预测，并通过相应的资源分配策略分配系统所需资源。

此外，针对前面提到的预测误差等相关因素的存在，在调度过程中采用预测式补偿机制，通过 OCA 算法作为二级预测机制进行资源预测，以资源预留的方式提前部署虚拟机及时修正预测误差，实时调整资源分配以应对突发问题的产生，避免由于反馈触发方式带来的延时等问题而影响服务质量。另外，在资源分配过程中充分结合系统实时反馈信息，设计了基于反馈与多级预测机制的在线敏捷弹性调度算法（AEOAS）。

4.2.1 基于补偿的二级 ARIMA 在线预测算法

在对下一时间片信息进行预测的过程中，我们假设未来行为是对历史以及当前行为的一种延续，由此可知预测结果的准确性对历史数据的依赖性非常大。由于历史数据中随机误差因素的存在以及事件本身的不可预测性都导致预测偏差是无法避免的。为了最大程度的避免资源供应不足和资源浪费现象的发生，避免补偿不及时造成的资源分配滞后，使供应曲线与资源需求尽可能接近，本文引入了第二级细粒度的在线预测式补偿算法。

实验发现在一定的时间范围内，预测间隔时间越短，预测序列拟合效果也越好。由于本级预测算法的主要目的是解决 SARIMA-BPNN 模型出现预测偏差的情况下及时进行资源的补偿问题，通过时间上更细粒度的资源预测可以达到更精确的预测效果，提前发现系统资源供不应求的情况从而提前准备资源以避免服务失效。但是在资源准备过程中（包括虚拟机的创建、开机到服务可用）需要一定的时延（这个时间通常在分钟级别），因此权衡历史数据收集的代价和预测准确度，本级的预测算法选择了轻量级的、时间代价小的 Online Compensate ARIMA（OCA）算法。以更细粒度的负载信息作为时间序列，实时修正系统运行中的资源误差，预测步长 Δt 选为分钟级别，历史数据长度为 l_2。本文的 OCA 算法是对前文中 ARIMA 模型的延续，其建模与实现过程同实现过程如算法 4-1 所示。

算法描述：

Algorithm4-1:ARIMA

输入:ΔT 长度的历史时间序列

输出:未来时间段内预测负载值 forc

```
1. library(forecast)
2. sequence_length = Δt
3. ts_1 <- ts(data[i:(i+(sequence_length - 1 ))])
4. fit <- auto.arima(ts_1, trace = FALSE)
5. forc <- forecast(fit,1)
```

由于本文的 OCA 补偿算法是基于预测机制进行补偿，由第三章可知，预测结果的准确性受历史数据窗口、预测步长的大小影响很大，为了保证算法的预测精度，本文分别测试了不同历史数据长度下 OCA 算法的预测误差，如图 4-3 所示，其中蓝色曲线代表不同历史窗口下 MAE 误差值，红色曲线代表不同数据窗口下的 RMSE 误差值，由测试结果可知随着窗口大小的增长，历史数据长度为 45min 时预测误差最低，窗口大小为 1 小时时误差稍大但并不明显，之后随着历史数据的增加预测误差逐渐增大，其中 MAE 与 RMSE 计算方法同公式（3-10）和（3-11）。

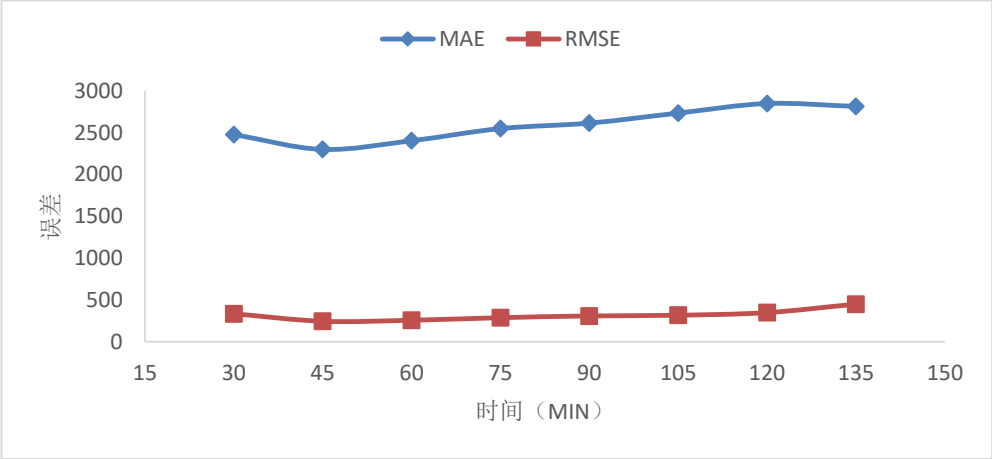


图 4-3 不同历史数据长度下 OCA 预测误差

因此文本在上述测试结果的基础上选择了 45min 和 60min 两个历史数据长度进一步测试了不同预测步长下 OCA 算法的预测误差，误差结果如图 4-4 所示，实验表明预测误差与预测步长成正相关，随着预测间隔的增加预测误差随之增大，并且数据长度为 45 分钟时预测误差相较于 60 分钟更小。考虑到本文实验环境下虚拟机从资源申请到服务可用的延迟时间通常在 1-3 分钟左右，为了避免造成系统震荡选择的间隔时间应该大于延迟时间，因此综合考虑预测

误差以及系统性能的稳定性，本文的 OCA 算法取 Δt=5，l_2=45，单位均为分钟。

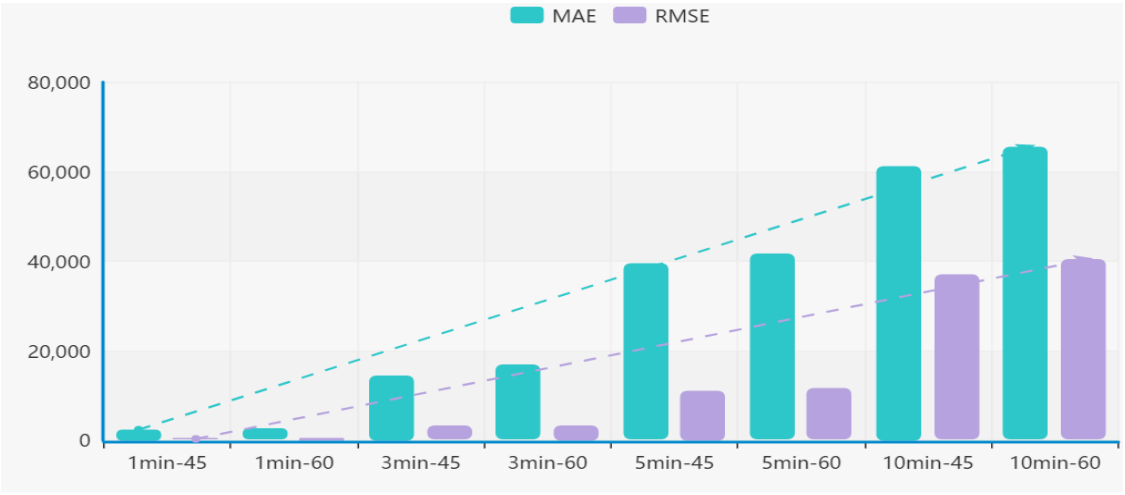


图 4-4 不同历史数据与预测步长下 OCA 预测误差

4.2.2 资源分配

在伸缩过程中采用反馈机制分配资源虽然会造成分配滞后、响应时间超时等一系列问题，但它是系统当前运行压力以及集群整体负载能力的最直接体现，因此本文在 OCA 算法预测过后的资源分配过程中融合了反馈信息这一因素，实现更精确地资源分配。本文使用集群平均 CPU 利用率作为反馈信息，其计算过程如（4-1）所示：

$$S_{cpu}(t)=\frac{1}{n}\sum_{i=1}^{n}S_{cpu}{}^{i}(t)\in\left[T^{L}{}_{cpu},T^{H}{}_{cpu}\right] \tag{4-1}$$

$S_{cpu}(t)$为系统采集到的实时监控信息，$T^{L}_{\bullet}$和$T^{H}_{\bullet}$分别代表相应资源的下界阈值和上界阈值，该阈值可由用户设定。在整个弹性调度过程中好的弹性算法应该保证相应的资源信息保持在阈值范围内，否则将触发相应的阈值策略，例如 Amazon Auto Scaling，但在实际应用中很难根据监控到的系统指标直接确定需要分配的资源量，因而通常采用特定比例与数量的虚拟机进行弹性伸缩，例如当$S_{cpu}{}^{1}(t)\geq S_{cpu}{}^{2}(t)\geq T^{H}_{\bullet}(t)$时，直观来看前者分配的资源应该不少于后者分配的资源，本文的调度算法就将相应的资源监控信息融合到资源分配的过程中。

为了更好的满足用户需求，实现“按需供应”，云中提供了多种类型的虚拟机供用户选择，例如，小型、中型、大型、计算型、I/O 型等。本文的弹性调度算法根据不同的用户需求提供了 M 种不同类型的虚拟机

$Server_1, Server_2, \ldots, Server_M$，不同类型的虚拟机处理能力与租用费用均不相同。在实际应用中，根据不同的应用场景在保障服务质量的前提下测出不同配置下的虚拟机能承受的外部负载压力值$W_i, i=1,2,\ldots,M$。据此，可以根据公式（4-2）预估出相应的资源需求量，比如：需要的虚拟机型号以及每种型号对应的虚拟机数量。

$$\min \sum_{i=1}^{M} n_i p_i, s.t. \sum_{i=1}^{M} n_i W_i \geq \lambda \tag{4-2}$$

其中，n_i和p_i分别代表第i种类型虚拟机的数量和租用费用，λ代表由SARIMA-BPNN算法预测出的负载请求量。对于一级预测SARIMA-BPNN的资源分配采用公式（4-2）进行运算，根据计算出的资源量确定下一时间片需要投入集群中的虚拟机种类与数量，在保证服务质量的前提下使租用虚拟机的费用最低。而在采用OCA算法进行补偿时为了提升资源分配精度，结合了系统资源反馈信息，形成的资源预估公式如（4-3）所示：

$$\min \sum_{i=1}^{M} n_i p_i, s.t. \sum_{i=1}^{M} n_i W_i \geq \lambda' e^{L(t)} \tag{4-3}$$

$$L(t) = \begin{cases} \dfrac{S_\bullet(t) - T_\bullet^H}{1 - T_\bullet^H}, S_\bullet(t) > T_\bullet^H \\ 0, T_\bullet^L \leq S_\bullet(t) \leq T_\bullet^H \\ \dfrac{S_\bullet(t) - T_\bullet^L}{T_\bullet^L}, S_\bullet(t) < T_\bullet^L \end{cases} \tag{4-4}$$

其中λ'代表由OCA算法预测出的下一时间片的负载量，n_i和p_i分别代表补偿后的第i种类型虚拟机的数量以及租用费用，使其租用费用最低。$e^{L(t)}$为系统反馈指标，其中$L(t)$代表系统中$S_{\text{cpu}}(t)$相对于阈值的比率。若在阈值范围之内则不触发阈值策略，$L(t)$取值为0，$e^{L(t)}$取值为1，此种情况不影响资源分配情况；其他情况下，若$S_{\text{cpu}}(t)$大于最高阈值$T_\bullet^H$，则$L(t)$大于0，$e^{L(t)}$大于1，此时意味着需要分配更多的资源；若$S_{\text{cpu}}(t)$小于最低阈值$T_\bullet^L$则反馈指标$e^{L(t)}$小于1，相应分配资源减少，该阈值可由用户根据业务需求自行设定，本文根据实验环境与实验效果选择0.9和0.1分别作为上下阈值。

由于公式（4-2）与（4-3）所示的计算方法可以抽象为多目标优化问题，即如何在保证服务质量、SLA 等指标的前提下，使租用虚拟机的费用代价最小。比如：如果我们提供了三种类型的虚拟机，其中CPU核数分别为1核、2核、4核，此时若系统中需要8个核的虚拟机资源，则我们可以分配的方案

如下：8 个 1 核的虚拟机、4 个 2 核的虚拟机、2 个 4 核的虚拟机、3 个 2 核 2 个 1 核、2 个 2 核 1 个 4 核、4 个 1 核 1 个 4 核等等，并且每种分配方案下随着用户需求的改变，又会随之产生新的需求，需要提供新的配置；随着虚拟机种类的增多系统规模的增大，其算法复杂度成指数级增长并且很难收敛到最优解。

考虑到实际应用中算法的实用性与可行性原则，本实验采用贪心的思想进行优化，按照性价比 W_i/p_i 从高到低进行排序，即在保证服务质量的前提下，单位费用处理负载能力最高的虚拟机类型最优先分配，之后按照处理能力递减的顺序依次确定系统中的虚拟机类型与数量，若待处理资源小于性价比最低的虚拟机处理能力时，按最小类型处理，保证整体租用费用最低。资源释放的过程只要服务质量满足，优先释放价格最高的虚拟机类型并依次递减。在资源补偿过程中由于虚拟机已经按小时提前计费，因此不再释放虚拟机，即资源不足时不考虑虚拟机类型的替换而是直接加入新的虚拟机，资源充足时不做处理。因而（4-2）、（4-3）中的资源映射函数分别表示为公式（4-5）、（4-7）。

其中 W_i 分别代表性价比依次降低排序的虚拟机类型所能处理的请求量，λ、η 分别代表 SARIMA-BPNN、OCA 算法预测出的请求量，λ'、η' 分别代表一二级预测算法中计算最后一种类型虚拟机数量之前系统中还需要的资源量。$R(\mathrm{n}_1,\mathrm{n}_2,\ldots)$、$r(\mathrm{n}_1,\mathrm{n}_2,\ldots)$ 中的 n_i 表示经过资源分配后按照性价比由高到低的顺序排序后的每种类型虚拟机所对应的数量，ξ、ω 分别代表经过 SARIMA-BPNN 以及 OCA 算法进行资源分配之前集群中已有的资源总量，通过系统所需资源作为是否进行资源伸缩的依据，其他变量的意义与之前保持不变。其中公式（4-5）为 SARIMA-BPNN 预测之后进行资源增加的过程，公式（4-7）为采用 OCA 算法进行资源预测过后的资源增加过程，公式（4-6）对应着 SARIMA-BPNN 预测算法中需要释放虚拟机时的计算过程。

$$R_{增}(\mathrm{n}_1,\mathrm{n}_2,\ldots)=\begin{cases}\left(\left\lfloor\dfrac{\lambda-\xi}{W_1}\right\rfloor,\left\lfloor\dfrac{\lambda-\xi-\left\lfloor\dfrac{\lambda-\xi}{W_1}\right\rfloor W_1}{W_2}\right\rfloor,\ldots,\left\lfloor\dfrac{\lambda'}{W_M}\right\rfloor\right), & \lambda>\xi\wedge\lambda'/W_M=0\\ \left(\left\lfloor\dfrac{\lambda-\xi}{W_1}\right\rfloor,\left\lfloor\dfrac{\lambda-\xi-\left\lfloor\dfrac{\lambda-\xi}{W_1}\right\rfloor W_1}{W_2}\right\rfloor,\ldots,\left\lfloor\dfrac{\lambda'}{W_M}\right\rfloor+1\right), & \lambda>\xi\wedge\lambda'/W_M\neq 0\end{cases} \quad (4\text{-}5)$$

$$R_{减}\left(\mathrm{n}_1,\mathrm{n}_2,\ldots\right)=\left(\left\lfloor\frac{\xi-\lambda}{W_1}\right\rfloor,\left\lfloor\frac{\xi-\lambda-\left\lfloor\frac{\xi-\lambda}{W_1}\right\rfloor W_1}{W_2}\right\rfloor,\ldots,\left\lfloor\frac{\lambda'}{W_M}\right\rfloor\right),\ \lambda<\xi \tag{4-6}$$

$$r_{增}\left(\mathrm{n}_1,\mathrm{n}_2,\ldots\right)=\begin{cases}\left(\left\lfloor\frac{\mathrm{e}^{\mathrm{L}(t)}(\eta-\omega)}{W_1}\right\rfloor,\left\lfloor\frac{\mathrm{e}^{\mathrm{L}(t)}(\eta-\omega)-\left\lfloor\frac{\mathrm{e}^{\mathrm{L}(t)}(\eta-\omega)}{W_1}\right\rfloor W_1}{W_2}\right\rfloor,\ldots,\left\lfloor\frac{\eta'}{W_M}\right\rfloor+1\right),\eta>\omega\wedge\eta'/W_M\neq 0\\ \left(\left\lfloor\frac{\mathrm{e}^{\mathrm{L}(t)}(\eta-\omega)}{W_1}\right\rfloor,\left\lfloor\frac{\mathrm{e}^{\mathrm{L}(t)}(\eta-\omega)-\left\lfloor\frac{\mathrm{e}^{\mathrm{L}(t)}(\eta-\omega)}{W_1}\right\rfloor W_1}{W_2}\right\rfloor,\ldots,\left\lfloor\frac{\eta'}{W_M}\right\rfloor\right),\eta>\omega\wedge\eta'/W_M=0\end{cases} \tag{4-7}$$

4.2.3 AEOAS 算法详述

本文的 AEOAS 算法主要分为两部分，第一部分是基于外部负载请求量信息，按照预测间隔为 ΔT 、历史窗口长度为 l_1 作为实验参数的一级在线 SARIMA-BPNN 预测算法，预测结果中若要增加资源则根据（4-5）进行资源分配同时生成扩展列表确定需要增加的虚拟机类型以及数量，并调用相应 API 完成虚拟机创建过程，在下一预测周期开始时将准备好的资源加入到集群当中，实现弹性增加的过程；若需要减少资源，则根据公式（4-6）进行资源映射，同时形成相应的资源收缩列表确定需要释放的虚拟机类型以及数量，在本预测周期结束后调用相应的 API 完成资源释放的过程，由于资源释放过程较短，资源收缩可以在很快的时间之内完成。

第二部分是对 SARIMA-BPNN 预测误差的补偿，在每个 ΔT 的时间范围内，基于负载请求信息按照时间间隔 Δt 、历史窗口 l_2 进行预测，采用二级在线 OCA 预测算法，由于本级预测算法主要以补偿作为目标，因此结合云计算中资源计费标准，只在发生资源供应不足时向集群中添加资源；当发生资源供应过量时，由于相应资源以及计费，因此不进行回收操作，保证最大的资源利用率。当发现资源供应不足时，结合实时反馈信息按照公式（4-7）进行资源映射，根据映射结果判断是否需要进行补偿，若需要增加虚拟机则形成相应的补偿列表，同时调用 CloudStack 中的 API 及时添加到集群当中完成资源补偿，其中算法设计如算法 4-2 所示。

算法描述：

Algorithm4-2:AEOAS

输入：Δt、ΔT、l_1、l_2、$W_i\left(i\in[1,M]\right)$、$\xi$、$\omega$

输出：扩展或伸缩的虚拟机列表

```
1.  while true
2.      forecast ← SARIMA-BPNN_ΔT (l1)
3.      if forecast > ξ
4.          if 分配的资源整除  // by equation(4-5)
5.              add n1 Server1,n2 Server2,…,nM ServerM
6.          else
7.              add n1 Server1,n2 Server2,…,nM+1 ServerM
8.      else if forecast < ξ  // by equation(4-6)
9.          remove n1 Server1,n2 Server2,…,nM ServerM
10.         else
11.             nothing to do
12.     for t in ⌊1, ΔT/Δt⌋
13.         forc ← ARIMA_Δt (l2)
14.         if forc > ω  // by equation(4-7)
15.             if 分配的资源整除
16.             add m1 Server1,m2 Server2,…,mM ServerM
17.         else
18.             add m1 Server1,m2 Server2,…,mM+1 ServerM
19.     endfor
20.     next ΔT
21. endwhile
```

4.3 测试原理与测试流程

4.3.1 被测系统工作原理

本文使用 LoadRunner[97]模拟客户端的用户行为，因此本文中的系统主要分为两个部分：部署在 CloudStack 上的 TPC-W 弹性应用集群和用于负载产生的 LoadRunner 负载发生集群。测试过程中通过控制 LoadRunner 负载发生端以 HTTP 报文的方式，模拟用户访问服务端的行为。客户端是由多台

LoadRunner 负载发生器组成的集群构成，集群中的负载发生器生成的 Web 访问请求，发送到部署的多层 TPC-W 弹性应用系统并等待接收系统返回的应答，根据收集到的相关信息评价被测系统弹性性能的好坏，其中 Nginx 作为多层弹性应用系统中与客户端接触的第一层，通过其负载均衡的作用将请求分发到集群中不同的节点中，之后根据 Monitor 模块监控的资源信息以及获取的访问日志，交给调度器根据其上制定相应的弹性调度方案，经由执行模块调用 API 完成集群中资源的删减（也就是本文中的 MAPE 循环管理流程），之后调用修改 Nginx 配置文件的脚本，将新添加的实例加入 Nginx 配置文件中用于承担负载以及分担服务，同样对于新删除的实例调用脚本完成从 Nginx 配置文件中删除其对应 IP 的操作，从而修改整个系统中的资源配置，完成整个弹性资源分配的过程；以期能用最少的资源在最大程度上满足客户的 SLA 和 QoS 保障服务需求。

4.3.2 测试工作的流程

本文的测试工作主要包括测试前的准备阶段和测试执行阶段，具体流程如下：

(1) 测试准备阶段：开始测试之前首先需要做好准备工作（主要是 LoadRunner 的准备工作），主要包括测试脚本的录制、请求负载和测试场景的设计等，可以通过设置集合点、定义事务以及设置思考时间等更真实地模拟用户行为，是测试过程中至关重要的一环；

(2) 开启资源调度器，执行弹性管理流程，由于本文中提供服务的 TPC-W 应用设置了自启动服务，因此在资源弹性伸缩的过程中不需要重启服务，节省了资源分配到资源可用的延迟时间；

(3) 运行测试场景开始测试，根据设定的不同场景分别执行测试脚本完成测试过程，若发生中断则测试终止，否则等待测试场景运行完毕，测试终止；

(4) 根据每种不同测试场景下收集到的统计数据，分析各自的实验结果，得出结论。

4.4 本章小结

本章详细介绍了本文弹性调度机制的设计与实现和基于反馈与预测补偿机制的弹性调度算法 AEOAS 的设计与实现，最后通过对整个系统的测试原理以及测试流程的介绍为后文实验验证做好准备。

第 5 章 实验结果与分析

5.1 实验环境

本文依照 TPC-W 性能评测为基准，以尽可能真实的还原现实业务场景为原则设计了本文的实验负载，采用符合 TPC-W 规范的应用程序作为基准测试程序，并使用 LoadRunner 进行负载发生，通过 Nginx 将请求转发到 CloudStack 管理的虚拟机集群之中进行处理，软件版本如表 5-1 所示。

表 5-1 实验软件配置

软件名称	版本号
CloudStack	CloudStack 4.6
操作系统	CentOS 6.5
JDK	jdk1.7.0_67
tomcat	apache-tomcat-6.0.18
MySQL	mysql Ver 14.14 Distrib 5.1.73
nginx	nginx/1.4.4
LoadRunner	LoadRunner 11.0

本文根据不同软硬件配置将系统划分为基础设施节点、资源管理节点和基准测试节点三个部分。

基础设施节点提供完全虚拟化的资源，其配置为：Intel Xeon E5-2620 X2 处理器，虚拟机管理器采用的是 XenServer。云管理节点为基础设施节点的总控节点，包括管理平台和负载均衡器两个部分，使用 CloudStack 作为弹性管理平台，Nginx 作为负载均衡器，操作系统均为 CentOS 6.5。具体配置为：Intel Xeon E5-2620 X2 处理器，2 核计算节点，2.1GHz CPU 主频，8GB RAM 内存。基准测试节点负责运行测试框架和负载发生器。测试框架用于控制实验和监控系统。负载发生器使用 LoadRunner 来模拟真实负载对系统加压。

在实验中，测试框架和负载发生器所属机器的配置为：Windows XP 2002 系统，处理器为 Intel(R) Atom(TM)，计算节点为四核，CPU 主频为 2.5 GHz，内存大小为 8 GB RAM。本文基于 CloudStack 资源管理平台，借助 TPC-W 基准实现了典型的 Web 应用场景，集群伸缩规模达到 35 个节点，并在此基础上提供三种不同类型的虚拟机用于弹性调度。

5.2 实验相关数据

5.2.1 评测指标

根据弹性的定义可知，本文中弹性调度的核心特征反应在速度和精确度两个方面，由图 2-1 以及公式（2-1）可知，本文中对于弹性算法的评测标准主要参考 SLA 标准中规定的 QoS 指标[98]。QoS 中规定的罚金主要包括两部分，一部分是资源分配不足时由于违背了 90%以上的响应时间低于 5 秒钟的 QoS 参考指标造成性能下降而产生的罚金[85]，这部分的罚金值定义为：设 f_i 代表从 t_{i-1} 到 t_i 时间段内，由于服务质量较低、性能指标不满足，而造成的相关经济损失，我们在这里定义整个调度过程中由于资源分配不足而造成的罚金值如公式 5-1 所示，其中 t_m 代表开始时间、t_n 代表结束时间。

$$P_l(t_m,t_n)=\int_{t_m}^{t_1} f_1dt+\int_{t_1}^{t_2} f_2dt+\cdots+\int_{t_{i-1}}^{t_i} f_idt+\cdots+\int_{t_{n-1}}^{t_n} f_ndt \tag{5-1}$$

另一部分是当资源分配过量时由于造成了资源浪费而产生的罚金，这部分罚金值定义为：设 $p_i(t_s,t_e)$ 代表从 t_s 到 t_e 时间段内由于资源空转多花费的罚金，其中 i 表示不同的资源类型，在本文中共有 ω 种不同资源，$M_i(t)$ 表示在 t 时刻用户在资源 i 上花费的金额，c_i 表示第 i 种资源的计费单价，$N_i(t)$ 表示在 t 时刻第 i 种资源在系统中稳定运行的数目，计算结果如公式 5-2 所示。

$$\begin{cases} p_i(t_s,t_e)=\int_{t_s}^{t_e}\left(M_i(t)-c_i\times N_i(t)\right)dt \\ P_o(t_s,t_e)=\sum_{i=1}^{\omega} p_i(t_s,t_e) \end{cases} \tag{5-2}$$

因此总的罚金计算公式如 5-3 所示，由于第一种情况下造成的后果要远远大于第二种情况下产生的影响，因此本文中用于计算罚金的相关指标如下：若是资源分配不足导致的 QoS 指标不满足，则相应时间超时的部分按照 90%的标准每超过 1%产生 0.2$/min 的罚金；由于资源分配过多导致的浪费部分，每台虚拟机产生 0.1$/h 的罚金，不足一小时的时间按一小时计算。

$$P(t_m,t_n)=P_l(t_m,t_n)+P_o(t_s,t_e) \tag{5-3}$$

5.2.2 资源服务能力

由于在弹性调度过程中我们需要根据预测出的负载信息进行相应的资源分配，因此了解不同类型服务器的服务能力是十分必要的。由于我们在实验中提供了多种不同类型的服务器，因此我们采用单台服务器加压的方式，逐一测

出每种类型服务器负载能力基准，并在实验过程中认为服务能力与资源数量成正相关。

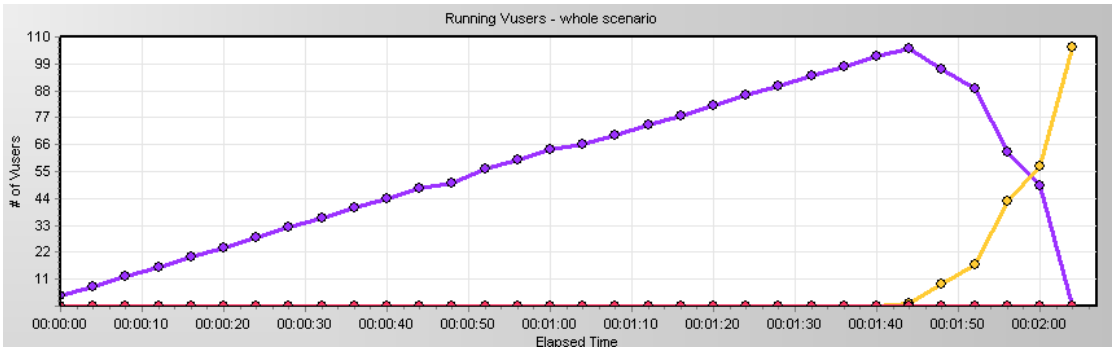


图 5-1 压力负载

在本文的实验环境下，我们逐渐增加对服务器的压力，测试在满足服务响应时间小于 5s 的限制，并且没有错误发生的情况下，服务器能处理的系统最大压力，当压力继续增大时，将出现响应时间超时甚至发生错误。我们以 t2.medium 类型的服务器举例说明，测试结果如图 5-1 所示。

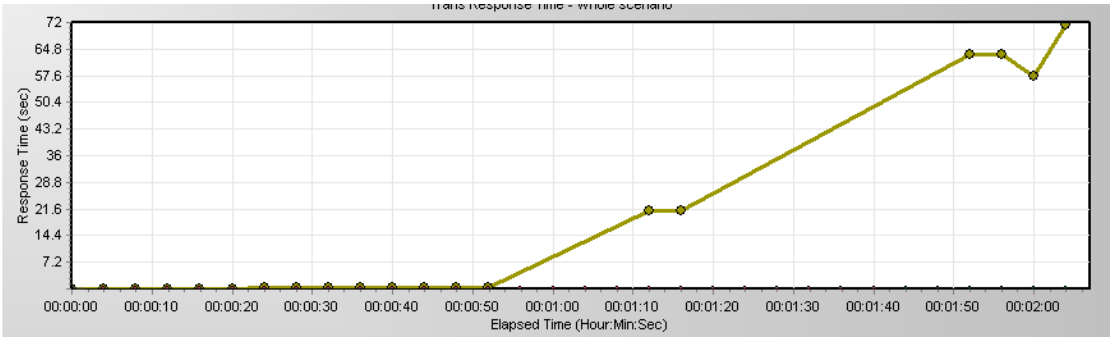


图 5-2 请求响应时间

其中图 5-1 表示逐渐增大的系统负载，随着系统压力的增大，响应时间变化曲线如图 5-2 所示，由图可知前 50s 系统一直保持着良好的性能，随着负载的增加，当用户并发数达到 50 左右时，响应时间呈指数型增长，此时系统已经达到负载能力的上限，无法及时响应。本文中选取了其中最具有代表性的三种不同虚拟机实例用于弹性伸缩，分别是 t2.small（2GiB 内存、1 个 vCPU）、t2.medium（4GiB 内存、2 个 vCPU）和 t2.large（8GiB 内存、2 个 vCPU）,这三种类型既包括 CPU 资源的增加以应对计算密集型资源需求，又包括内存资源的增加用于应对存储密集型资源需求，可以满足基本的用户需求。我们通过对 kafka 实时采集的系统日志信息进行统计，确定该类型服务器能承受的最大服务能力，其他类型的基准测试过程与此相同，因为算法需要，本文给出了两种不同时间粒度下的不同粒度虚拟机对资源的处理能力用于资源映射，统计结果如表 5-2 所示。

表 5-2 不同类型虚拟机处理能力

虚拟机类型	服务能力（/小时）	服务能力（/5min）
t2.small	115218	9581
t2.medium	234105	19588
t2.large	475256	39714

5.2.3 资源计费标准

参考公有云环境下，每种资源实例以小时为单位收取费用，本文中的资源计费标准在此基础上参考了 EC2 中 T2 系列的实例计费标准[99]。在 EC2 中 T2 系列共有 7 种不同类型的虚拟机，对于亚太地区来说，1 个 vCPU，2GiB 内存的实例类型收费标准是 0.032$/h，其中费用为 EC2 中真实租用价格，结果如表 5-3 所示。

表 5-3 不同类型虚拟机计费标准

虚拟机类型	费用（$/小时）
t2.small	0.032
t2.medium	0.064
t2.large	0.128

5.2.4 负载设计

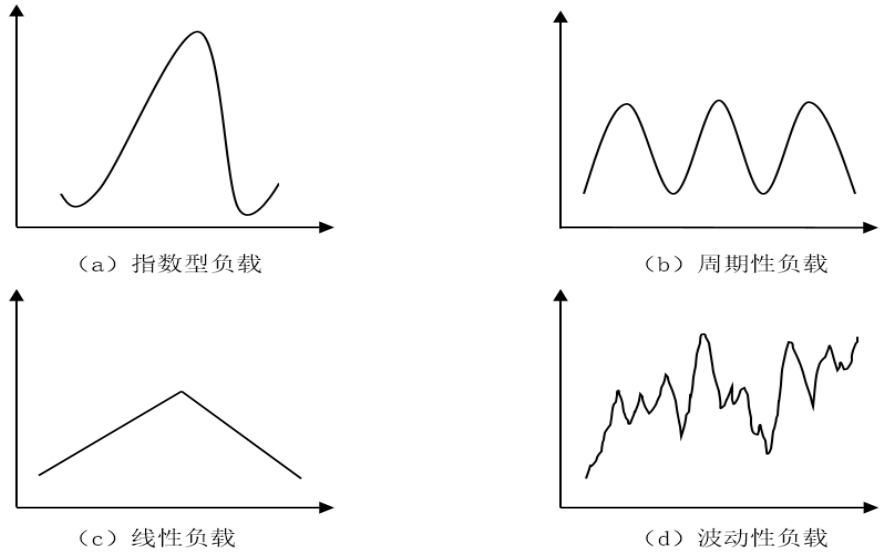


图 5-3 不同类型的负载特征

在实验过程中，我们以实际运行场景为原型，提取了四种不同类型的负载特征：指数型负载、周期性负载、线性负载和波动性负载。理想情况下的负载特征曲线分别如图 5-3 中的（a）、（b）、（c）、（d）所示。本文采用的实

验方法均需要经过一定时间段的数据采集工作用于负载预测的历史数据。为了验证算法的有效性，本文分别设计了两种不同类型的负载，第一种是以图 5-3（b）中的带有周期性的简单波形负载作为负载发生，证明在简单实验场景下，本文的算法仍然适用，负载设计如图 5-4 所示；

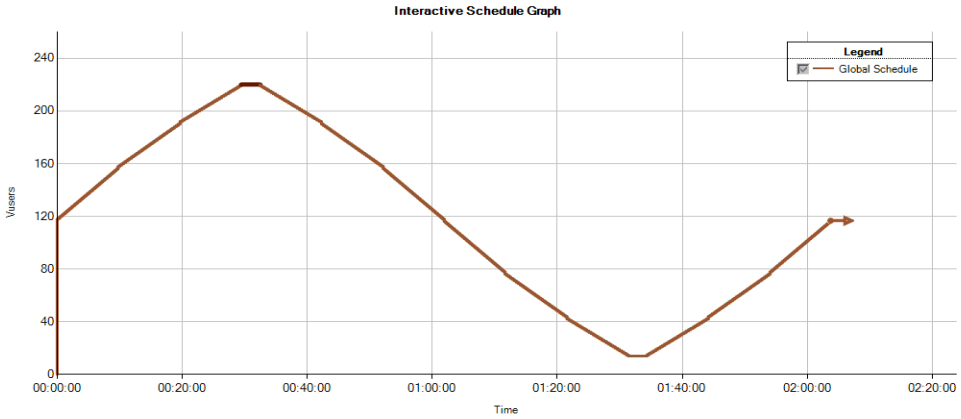


图 5-4 简单波形负载

第二种是以图 5-3 中的四种负载特征为原型，考虑到在目前的云计算环境下，由于用户规模的增大，用户需求的增加，线性增长的同时伴随着突发式指数型增长是最为常见的负载类型，下面几个目前比较常见的应用场景中均包含此种类型的负载：（1）电商活动大促、限时秒杀；（2）热门视频直播：重要比赛、大型晚会、热门电视剧、综艺节目；（3）游戏用户的夜间高峰期。

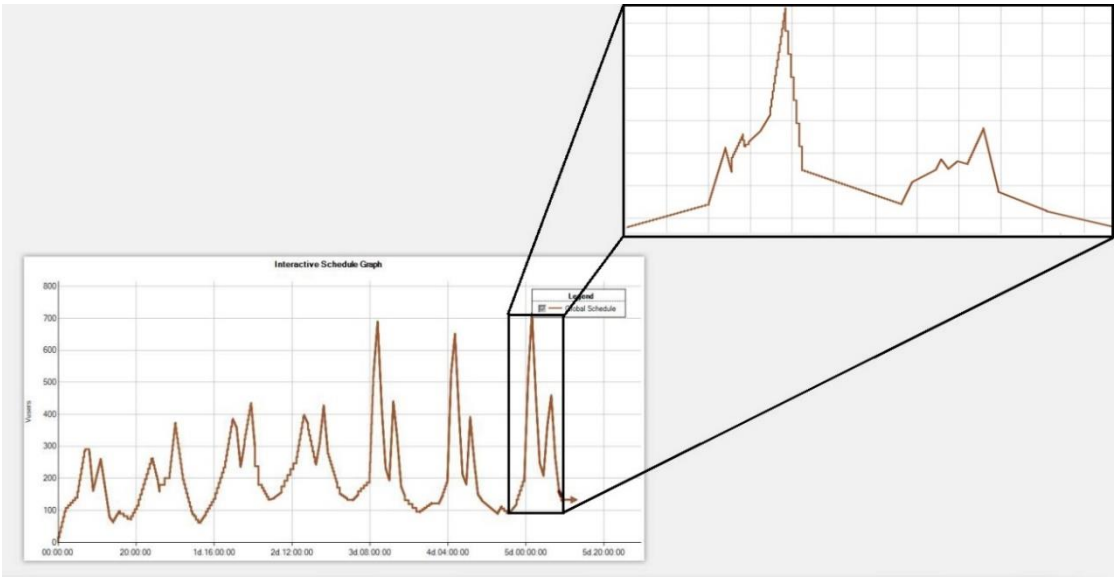


图 5-5 复杂混合型负载

我们已经可以预见到随之而来的访问量的激增以及业务高峰期过后的缩容现象，由于本文的主要研究目标在于提高云服务系统在应对含有突发性负载时的处理能力，也就是通过弹性云平台恰到好处的将计算、存储、网络等资源

进行合理调配和自适应规划，既要避免因为资源供应不足而导致的服务中断又要避免资源分配过度造成的大量闲置资源空转。因此本文设计了如图 5-5 所示的复杂负载发生曲线，在缓慢增长的同时含有突发性指数增长的特征，同时具备波动性与周期性的特征，在较长时间范围内用户的行为是可以进行预测的。实验过程中通过前面的负载进行数据采集，并用方框中的负载通过采集的历史信息进行实验，验证算法的有效性。由于本文的负载设计充分参考了真实业务场景并且涵盖了上述四种类型的负载特征，分别包含线性增加、线性减少、指数增加、指数减少的趋势，既保证了实际负载的波动性特征又具有可预测性的周期性特点，因此本文中设计的负载具有一定的可行性。

5.3 实验结果与分析

5.3.1 简单波形负载

本文通过两个小时的测试数据验证了简单负载下的弹性性能，资源供需对比结果如图 5-6 所示。

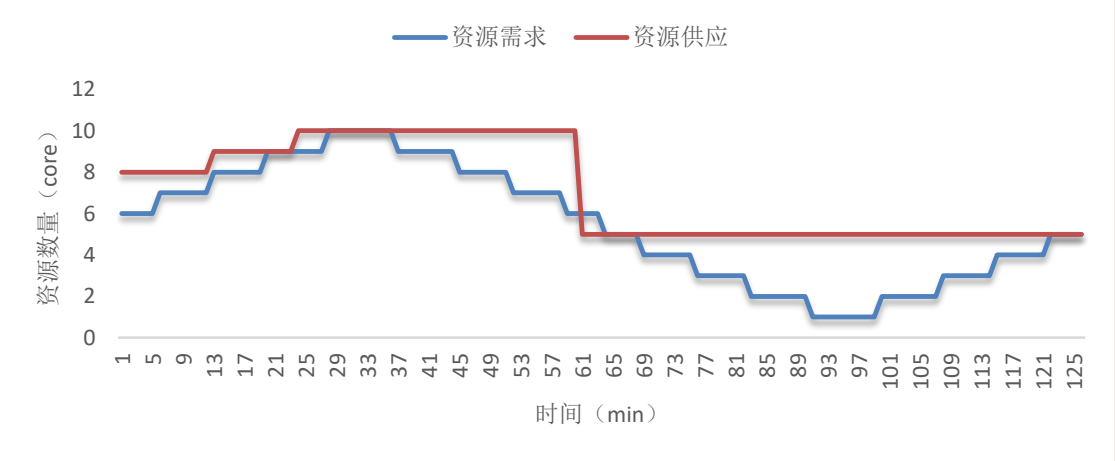
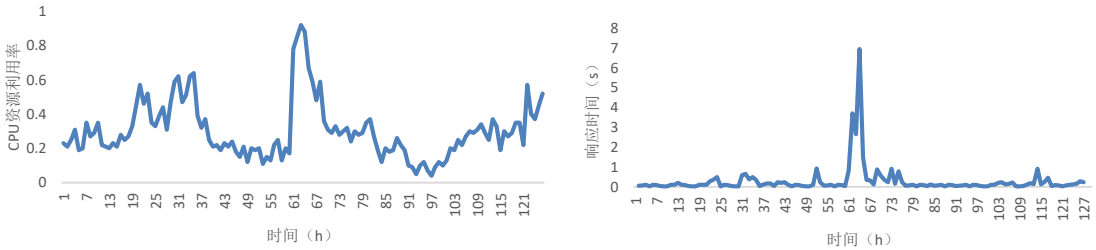


图 5-6 波形负载下资源供需曲线

其中蓝色曲线代表资源需求，红色曲线代表资源供应情况，结果显示使用本文 AEOAS 算法可以准确的完成资源预测并提前分配资源，通过较少的调度次数完成弹性伸缩。其中 CPU 资源利用率和响应时间变化曲线分别如图 5-7 中（a）和（b）所示，实验表明虽然在第 61 分钟时出现了资源供应不足导致的服务违约情况，但从整体误差来看，由于资源供应不及时导致的误差不足整体误差 1%，在整个调度过程中出现此种误差的比例更是低于 0.8%；同时整个调度过程中的 CPU 资源利用率均处于 10%-90%之间，并且 99%以上的响应时间在 1s 之内，满足弹性云服务的要求。虽然资源供应过度的误差占总体误差的 79.3%，并且产生了一定的罚金，但从弹性云服务的计费标准来看，过度的

资源并没有产生额外的租用费用，很多的完成了弹性伸缩的过程。综上可知，本文的 AEOAS 弹性策略对于简单应用下的波形负载仍然可以很好的完成资源弹性伸缩的过程，具有一定的普适性。



（a）CPU 资源利用率　　（b）响应时间

图 5-7 利用率与响应时间曲线

5.3.2 复杂突发型负载

为了验证本文中提出的弹性调度算法 AEOAS 在应对大规模复杂负载下的弹性性能，本文设计了三组对比实验进行说明。

第一组实验对比了本文中的算法与不采用预留机制的反馈触发式算法之间的性能，本文在此选用了工业中广泛使用的经典阈值策略[100]，资源供需对比结果如图 5-8 所示。

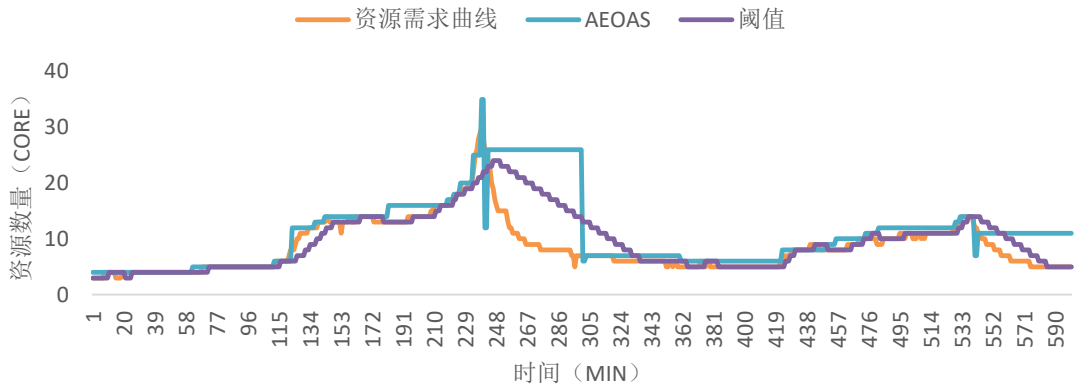


图 5-8 资源供需对比（a）

其中橘黄色曲线代表资源需求，蓝色曲线代表采用本文中设计的 AEOAS 算法的资源供应情况，紫色曲线代表采用经典阈值策略后的资源供应。本文的资源需求用资源核数来表示。由图可知，采用预留机制进行资源分配可以达到提前部署资源保证服务质量的效果，从图中可以看出阈值策略从第 9 分钟开始出现资源供应滞后，到第 112 分钟资源需求开始增加一直到 243 分钟资源较大规模增长的过程中一直处于资源供应不足，资源相应不及时状态。

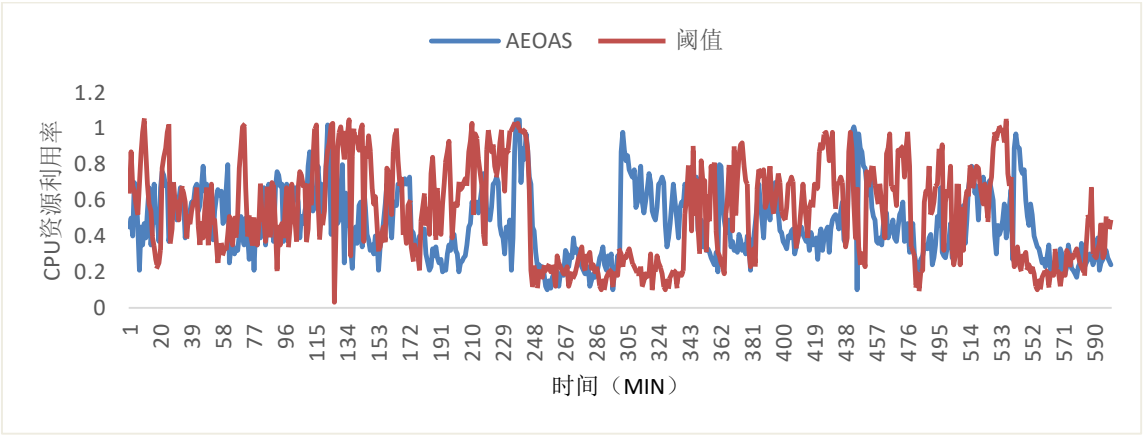


图 5-9 CPU 资源利用率对比（a）

其中资源利用率与响应时间对比结果分别如图 5-9 和 5-10 所示，发现在 112-243 分钟之间的时间段内采用阈值策略其 CPU 资源利用率几乎维持在 80%以上，响应时间也达到了 5s 的上限并且出现了服务失效、请求失败的情况，可见采用反馈式阈值触发策略无法应对此种类型的资源分配。而采用本文的 AEOAS 调度策略，由于资源预分配机制的存在，基本实现了资源的提前部署工作，从而应对大规模负载发生的情况，从资源利用率来看，在 112-243 分钟资源增长迅速的阶段 CPU 资源利用率基本维持在 20%-80%之间，响应时间满足 90%以上均在 5s 之内的 QoS 限制，证明本文的算法在应对突发负载的情况下弹性性能要大大好于采用阈值策略的调度机制。

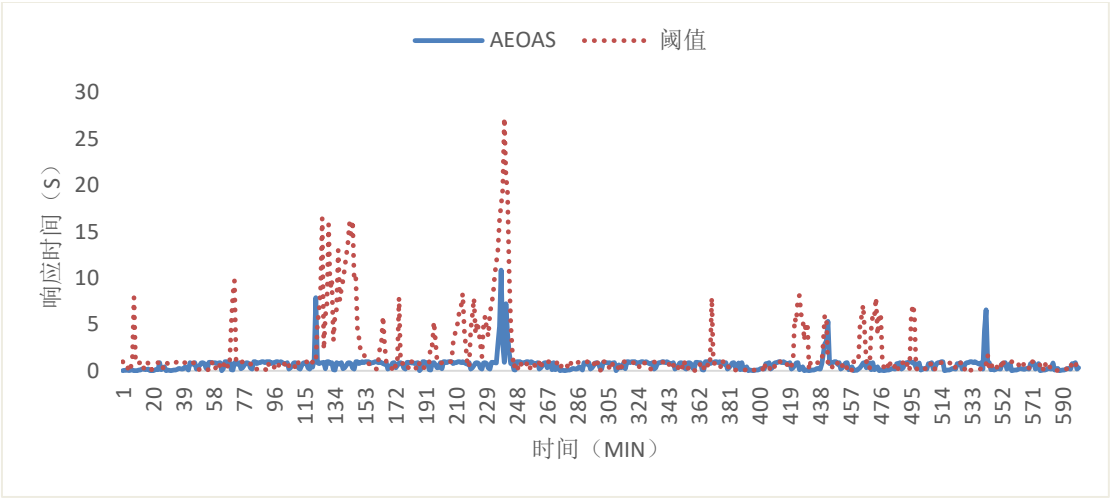


图 5-10 响应时间对比（a）

第二组实验对比了采用本文的 AEOAS 预测算法与只有一级预留没有误差补偿机制的 GABA 算法[101]进行对比实验，实验结果如图 5-11 所示。分析可知采用该算法调度的精确性与预测结果息息相关，当预测发生较大偏差或者随机事件发生时，系统无法及时响应并导致下一预测周期到来之前出现大量 QoS 违约现象，其中红色曲线表示 GABA 算法的资源供应曲线，采用该算法

进行资源调度可以解决上面没有预留机制时资源供应不及时的问题，但实验结果表明从第 124 分钟到 256 分钟持续出现了预测资源偏差较大的情况，并且系统无法及时作出相应，CPU 资源利用率以及响应时间对比结果分别如图 5-12 和 5-13 所示。

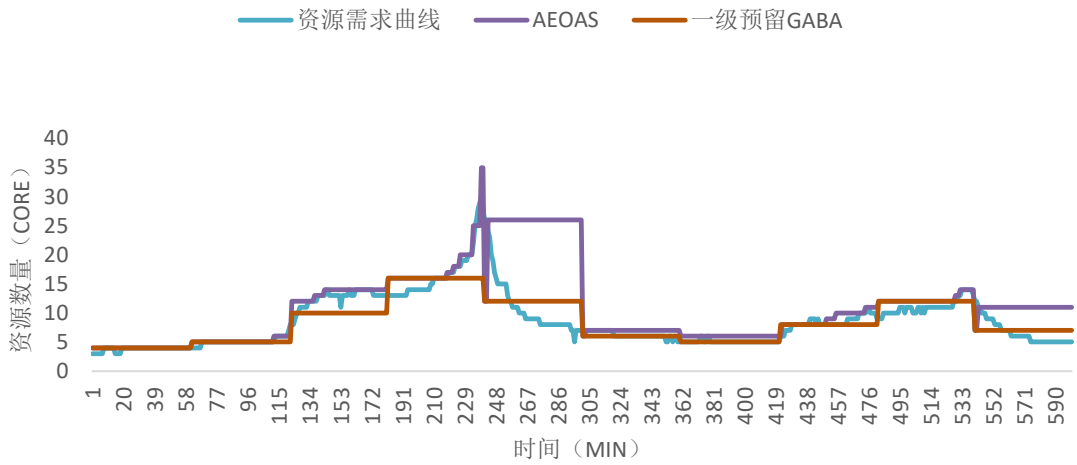


图 5-11 资源供需对比（b）

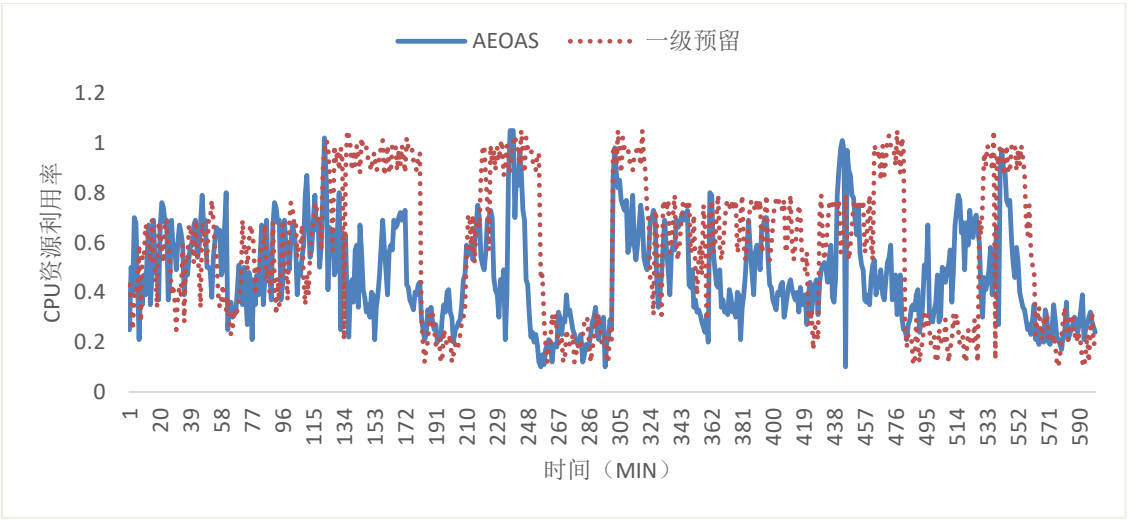


图 5-12 CPU 资源利用率对比（b）

由图 5-12、5-13 所示，该时间段出现了 CPU 剧烈波动的情况，在 124-178 分钟以及 210-256 分钟由于资源预估偏低，CPU 持续满载，同时响应时间大量增加，出现了服务违约、性能降低的情况；而在 170-206 以及 267-292 分钟之间的时间段 CPU 利用率又很低，造成了一定的资源浪费。而采用本文的 AEOAS 策略，由于加入了补偿机制，很好的避免了由于资源预测误差导致的性能降低问题，从第 241 分钟可以看出，由于预测算法出现了很大的误差在下一时间段开始之后释放了一部分虚拟机，但当补偿机制检测到预测误差之后及时补偿并申请了资源而避免了大量服务违约，从而更好的应对资源负载大规模

增加的情况，从响应时间上看，满足 SLA 规定的要求，可以较好的实现资源分配。

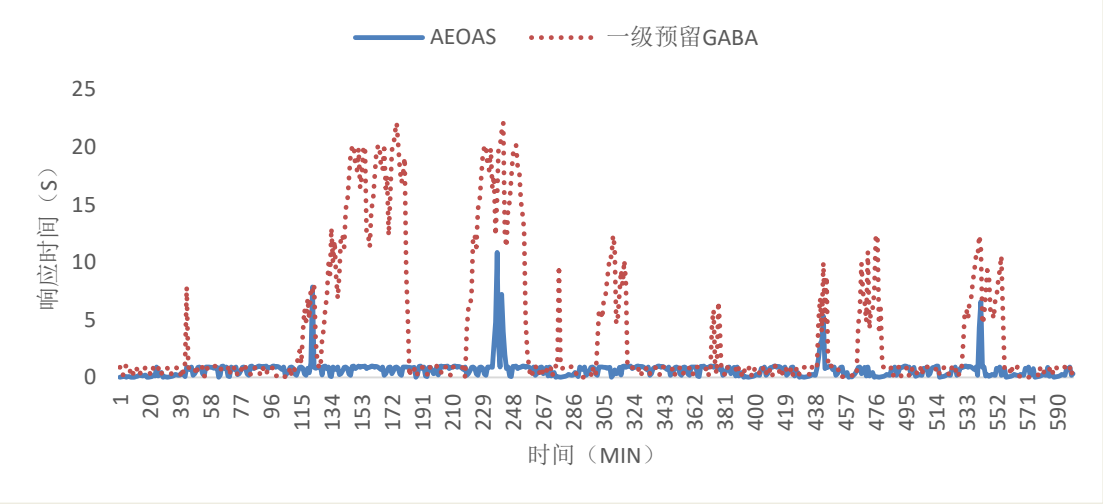


图 5-13 响应时间对比（b）

第三组实验实现了 AEOAS 算法与基于混合策略的弹性调度算法[39]的对比，Hybrid strategy 既使用了预留策略又加入了反馈机制，在资源分配过程中根据当前系统的反馈信息决定是否需要进行资源分配，若监控信息满足调度条件则按照神经网络预测的分配方案实施资源调度，若监控信息条件不满足，则当前周期不进行资源分配，比较结果如图 5-14 所示。

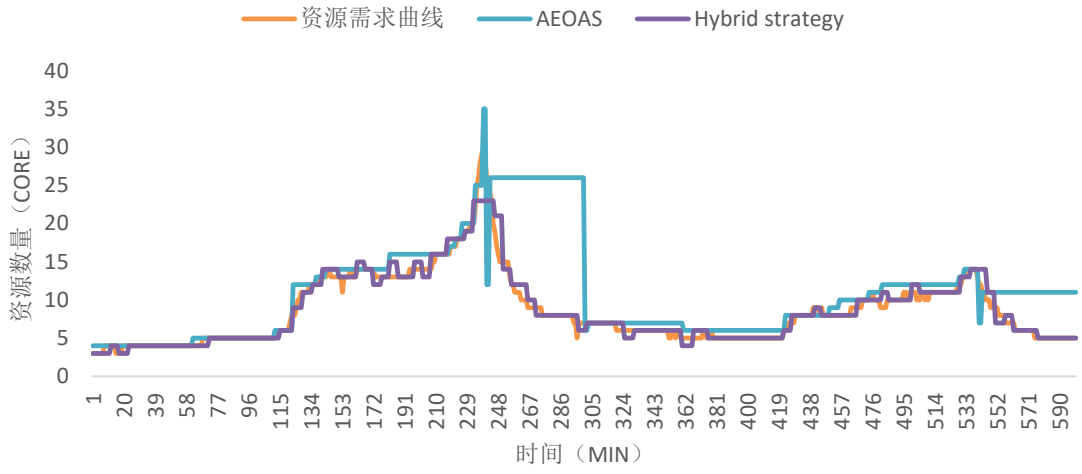


图 5-14 资源供需对比（c）

其中资源利用率与响应时间对比结果分别如图 5-15 和 5-16 所示。由图可知，由于 Hybrid strategy 算法采用分钟级别的短周期资源预测，其资源分配更贴近于资源需求曲线，虽然此种方式可以有效避免发生资源分配不足时仍然预测出需要减少资源或者资源分配过量但预测需要增加资源的情况，但该算法由于反馈信息只作为资源是否分配的依据无法修正资源预测的结果，因此该算法

的预测准确度对实验结果有较大的影响。另外，对于需要增加资源的时刻若无法触发反馈条件则无法及时响应资源变化，造成误差增加，例如第 234-244 分钟，同时由于该算法预测间隔是分钟级别的，资源准备过程需要一定的延时，频繁资源调度过程会导致系统抖动从而影响整体性能，从资源利用率调度曲线来看，CPU 变化明显资源伸缩频繁不适用于当前的弹性云环境，从而造成大量的响应时间超时，QoS 不满足。

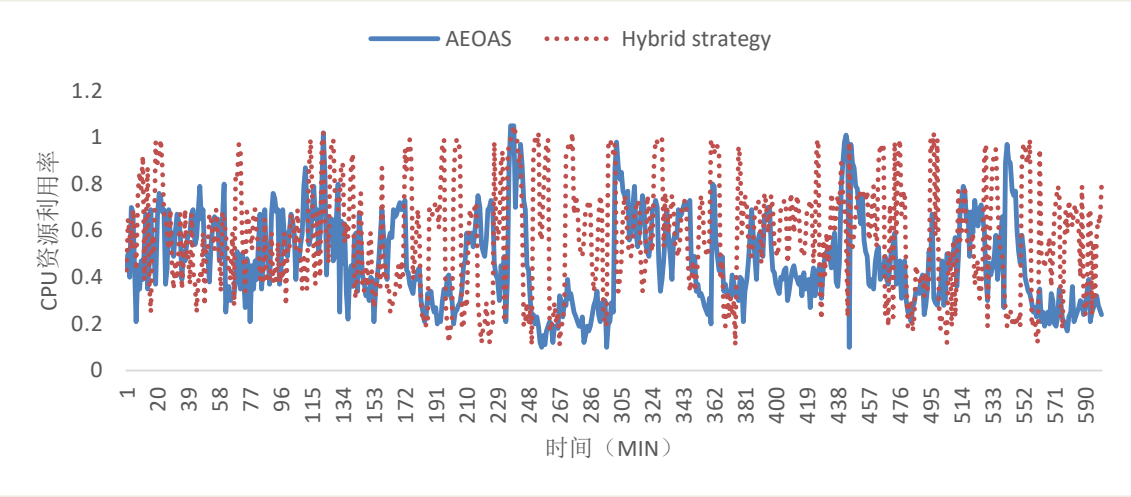


图 5-15 CPU 资源利用率对比（c）

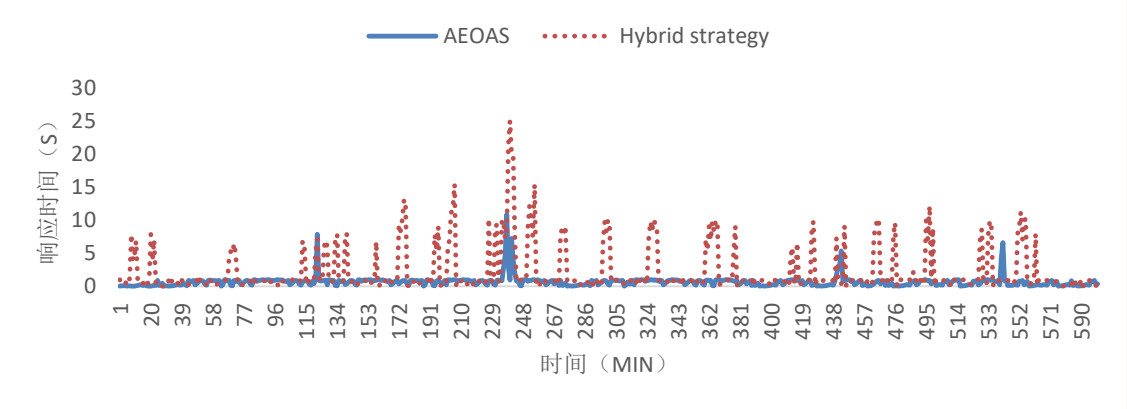


图 5-16 响应时间对比（c）

本文通过对四种算法在弹性调度过程中产生的误差结果进行统计分析发现，采用本文的 AEOAS 算法时资源供应不足所导致的误差只为 2%，相比于其他三种算法的误差大大降低了因资源供应不及时导致的服务性能降低以及影响服务质量的问题，此外本文的算法虽然在一定程度上增加了资源供应过高的误差比例，但由于云中资源计费标准是以小时为单位，此种分配方式不仅没有造成资源浪费，而且还会用于应对突发问题的产生以及避免资源频繁伸缩导致的系统震荡，统计结果如图 5-17 所示。

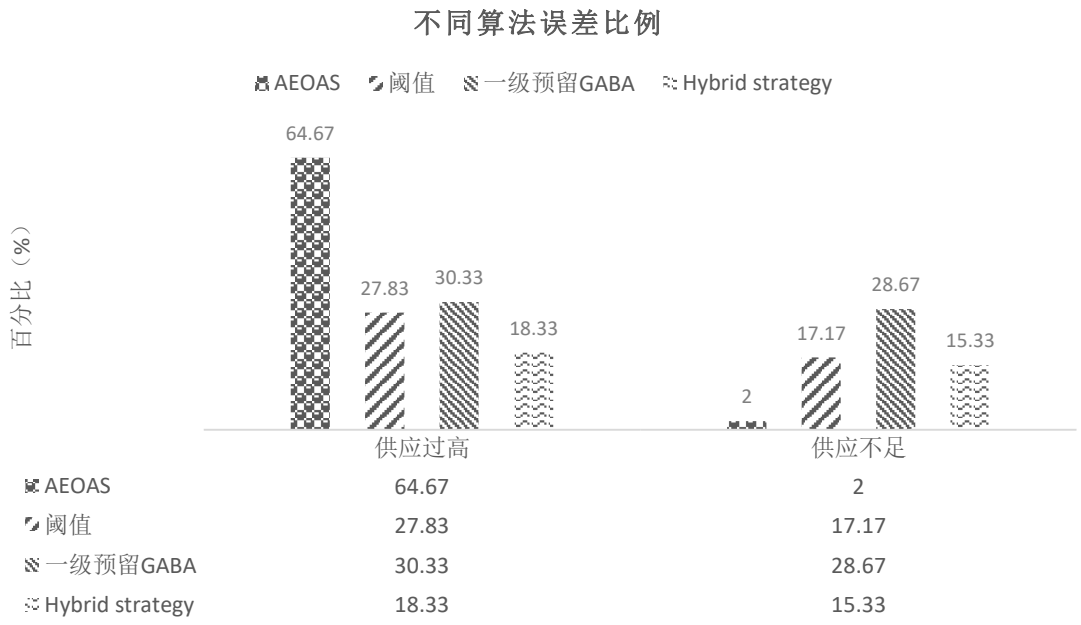


图 5-17 算法误差统计结果

由于本文算法分别实现了采用单一类型和多种类型虚拟机的调度，其中多种类型下的虚拟机调度情况如图 5-18 所示，该图为每个小时最终参与计费的虚拟机使用情况，实验结果表明增加虚拟机的类型在没有增加总金额的情况下，不仅减少了虚拟机数量同时又增加了弹性调度的灵活性，给用户提供了更多的选择空间，适应于更多的业务模式，具有一定的现实意义。

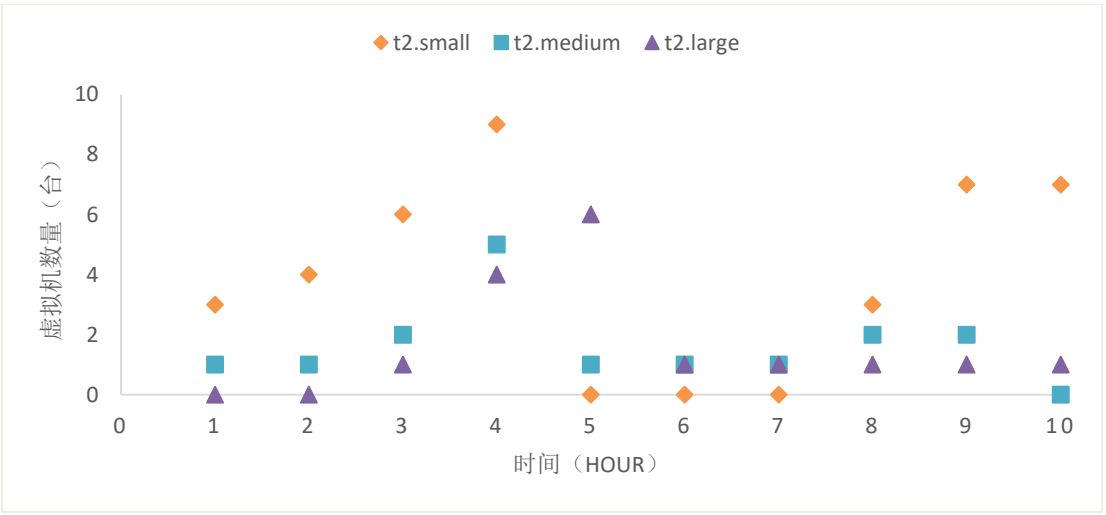


图 5-18 每小时提供服务的虚拟机类型以及数量

表 5-4 中分别统计了不同算法在调度过程中产生的罚金（按照公式 5-3 计算），其中本文的资源计费标准选择小时为最小时间单位（即不满足 1 小时的按 1 小时计算）。结果表明采用本文的 AEOAS 算法产生的罚金相比于采用

GABA、Hybrid strategy 和阈值策略分别降低了 80.7%、83.6%和 85.2%，证明本文算法具有更好的弹性性能。

表 5-4 不同算法产生的罚金

算法名称	AEOAS	GABA	Hybrid strategy	阈值
罚金（$）	2.495	12.975	15.19	16.858

另外，由于本文算法采用在线式资源调度算法，整个调度过程中资源配置时间如图 5-19 所示，其中 96%以上的配置时间在 3min 之内，在最高峰时达到 252s，小于本文 OCA 算法中设定的 5min 时间间隔，另外，由于本文中的算法在数据收集过程中通过将 Kafka 集群集成到系统中作为日志采集组件进行实时采集，该过程只占用少量的 CPU 资源，时间代价可以忽略不计，因此本实验具有一定的可行性。

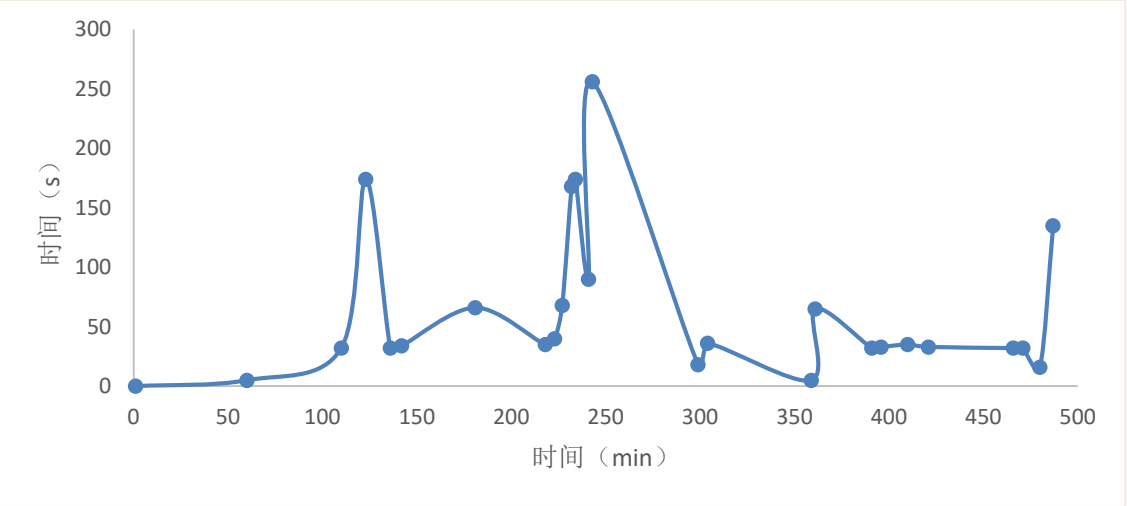


图 5-19 资源配置时间

综上所述，本文的 AEOAS 算法有效的解决了目前云计算环境下资源利用率低、资源浪费严重以及在面对大规模突发式负载发生的情况下，因应对能力不足而导致的服务质量降低、QoS 违约上升的问题，实现了在保证 SLA 与 QoS 的前提下，从云供应商和用户双方的角度出发，达到最小的租用成本，具有一定的可行性。

5.4 本章小结

本章详细介绍了本文的实验结果以及实验分析。设计了两种不同类型的负载，既验证了本文中算法对于简单波形负载的有效性，又通过对四组算法三组对比实验的设计，验证在复杂突发负载的情况下本弹性算法在精确度、违约率

以及使用费用等方面的优势，并证明该算法具有一定的实际意义，因此本文中提出的 AEOAS 算法在弹性云环境中具有可行性。

结　论

云计算作为一种新兴技术产业模式和基础架构解决方案，已经在国际市场中扮演着越来越重要的作用。弹性作为云计算背景下至关重要的特征之一也受到越来越多的关注。用户需求催生的弹性云服务也正在改变着人们的生产和生活方式，“按需付费”的云服务使用模式，也为云计算带来更大的发展空间和更多的挑战与机遇。用户需求的不断升级演变以及“服务质量至上”的消费理念的出现，也注定了云计算庞大规模下的性能保障刻不容缓。针对目前云服务在面对大规模负载发生时因为资源分配不及时、资源分配延迟时间长而导致的服务质量降低、SLA 违约上升这一问题，以及大量资源分配回收不及时导致的资源利用率低、资源浪费严重等问题，本文结合云服务中的计费模式，设计了改进的弹性调度策略，站在用户和供应商双方的角度，在保证服务等级协议与服务质量的前提下，寻求最大的经济效益，紧密结合用户需求提供更敏捷、更精确的资源弹性配置方案。本文的主要工作如下：

(1) 通过对相关研究现状的分析，总结了目前国内外学术界在相关领域的研究进展以及现有工作中存在的不足之处。

(2) 深入研究弹性以及弹性云服务的概念，紧密结合云弹性中速度与准确度这两个核心特征，针对目前研究中存在的问题，提出改进的解决方案。

(3) 分析不同预测方法的适用性与优缺点，结合 ARIMA 算法在时间序列分析中的优势以及神经网络算法对非线性数据强大的学习能力，将两种算法结合在一起提出改进的在线 SARIMA-BPNN 预测方法，并验证这种预测算法在本文的弹性云环境下仍然适用，有效的提高预测精度。

(4) 在传统阈值策略与预测机制的基础上，将两种策略有针对性的加以融合，结合云环境下动态实时的特征，提出了在线敏捷弹性伸缩算法，在多级预测机制的基础上加入反馈信息实现更精确的资源分配，实验验证本文的算法在保证服务质量以及相关等级协议的基础上，进一步节省费用开销。

(5) 提出多种虚拟机类型下的服务解决方案，满足用户需求多样性，实现更灵活的资源分配方案，也为云服务的发展提供更多的可能。

(6) 提出弹性云服务管理流程，设计适用于弹性云环境下的应用场景，并设计具有针对性的负载发生验证算法的有效性，制定基于罚金的评测指标，计算不同算法的罚金指标，验证算法准确性。

(7) 实现基于 CloudStack 的弹性资源调度过程，针对 LoadRunner 无法准确还原真实负载这一问题，设计 Kafka 信息收集系统，实时采集集群日志作为访问请求，验证算法可行性。

云计算作为一种复杂的、大规模的基础架构解决方案，其弹性调度过程需要解决的问题还有很多，虽然本文进行了深入的研究，但仍然存在以下问题需要改进：

(1) 快速的资源分配速度：本文中采用以模板创建虚拟机的方式实现资源供给，但到服务可用仍需要一定的准备时间，这也是算法调度周期的瓶颈限制之一，如果能提供更快的资源伸缩技术，相信整个系统的弹性调度效果将会有更大的提升。

(2) 设计动态阈值策略：阈值策略中阈值的确定一直是难以定论的问题，不仅取决于系统性能，更取决于用户需求，设计出可以根据系统动态信息自动更新的实时阈值策略将有更大的说服力。

主要参考文献

[1] Amanatullah Y, Lim C U, Ipung H P, et al. Toward cloud computing reference architecture: Cloud service management perspective[C]//ICT for Smart Society (ICISS)：2013 International Conference on. IEEE，2013: 1-4.

[2] 朱莉,王鹏. 云计算在高校的部署与应用研究—以开源云计算产品Eucalyptus为例[J]. 吉林师范大学学报(自然科学版), 2011, 32(2): 131-133.

[3] Sefraoui O, Aissaoui M, Eleuldj M. OpenStack: Toward an Open-source Solution for Cloud Computing[J]. International Journal of Computer Applications, 2014, 55(3): 38-42.

[4] Yang C T, Huang K L, Liu J C, et al. On Construction of Cloud IaaS Using KVM and Open Nebula for Video Services[C]// International Conference on Parallel Processing Workshops. IEEE Computer Society, 2012: 212-221.

[5] Nurmi D, Wolski R, Grzegorczyk C, et al. The Eucalyptus Open-Source Cloud-Computing System[C]// Ieee/acm International Symposium on CLUSTER Computing and the Grid. IEEE, 2009: 124-131.

[6] Barkat A, Dos Santos A D, Ho T T N. Open Stack and Cloud Stack: Open Source Solutions for Building Public and Private Clouds[C]// International Symposium on Symbolic and Numeric Algorithms for Scienti-fic Computing. IEEE, 2014: 429-436.

[7] 2016 年中国云计算行业发展情况分析[OL].(2016-04-05). http://zhuanti.cww.net.cn/tech/html/2016/4/5/201645853126417.htm.

[8] Balduzzi M, Zaddach J, Balzarotti D, et al. A security analysis of amazon's elastic compute cloud service[C]//Proceedings of the 27th Annual ACM Symposium on Applied Computing. ACM, 2015: 1427-1434.

[9] Peng N Y, Wen X S, Chen J B, et al. Research on Power Transformer Fault Diagnosis with BPNN Method[J]. High Voltage Apparatus, 2004.

[10] Ben-Yehuda O A, Ben-Yehuda M, Schuster A, et al. Deconstructing Amazon EC2 Spot Instance Pricing[C]// IEEE Third International Conference on Cloud Computing Technology and Science. IEEE, 2011: 304-311.

[11] Walker E.Benchmarking amazon EC2 for high-performance scientific

computing[J]. Usenix, 2016: págs. 18-23.

[12] Ostermann S, Iosup A, Yigitbasi N, et al. A Performance Analysis of EC2 Cloud Computing Services for Scientific Computing[C]// International Conference. 2015: 115-131.

[13] Frank A, Steeb C A, Edelstein D B, et al. Business method for pay-as-you-go computer and dynamic differential pricing: US, US20060165005[P]. 2006.

[14] Breitgand D, Epstein A. Capacity over-commit management in resource provisioning environments: US, US9245246[P]. 2016.

[15] Sharma U, Shenoy P, Sahu S, et al. Kingfisher: A system for elastic cost-aware provisioning in the cloud[J]. Dept of Cs, 2010.

[16] Ali-Eldin, A, J. Tordsson, and E. Elmroth. "An adaptive hybrid elasticity controller for cloud infrastructures." Network Operations and Management Symposium IEEE, 2012: 204-212.

[17] Khatua S, Ghosh A, Mukherjee N. Optimizing the utilization of virtual resources in Cloud environment[J]. 2016: 82-87.

[18] Seung Y, Lam T, Li E L, et al. CloudFlex: Seamless scaling of enterprise applications into the cloud[C]// IEEE INFOCOM. IEEE, 2011:211-215.

[19] Wilder B. Cloud Architecture Patterns: Using Microsoft Azure[J]. Oreilly & Assoc Inc, 2012.

[20] Ghanbari H, Simmons B, Litoiu M, et al. Optimal autoscaling in a IaaS cloud[C]// International Conference on Autonomic Computing. ACM, 2012: 173-178.

[21] Buyya R, Garg S K, Calheiros R N. SLA-oriented resource provisioning for cloud computing: Challenges, architecture, and solutions[C]// International Conference on Cloud and Service Computing. IEEE Computer Society, 2011: 1-10.

[22] Hu, Ye, et al. "Resource provisioning for cloud computing." Conference of the Centre for Advanced Studies on Collaborative Research, November 2-5, 2009, Toronto, Ontario, Canada DBLP, 2016: 101-111.

[23] Li C, Li L Y. Optimal resource provisioning for cloud computing environment[M]. Kluwer Academic Publishers, 2012.

[24] Hadji, Makhlouf, and D. Zeghlache. "Minimum Cost Maximum Flow Algorithm for Dynamic Resource Allocation in Clouds." IEEE, International

Conference on Cloud Computing IEEE, 2016: 876-882.

[25] Akhani, Janki, S. Chuadhary, and G. Somani. "Negotiation for resource allocation in IaaS cloud." Bangalore Compute Conference, Compute 2011, Bangalore, India, March DBLP, 2011: 1-7.

[26] Han, Rui, et al. "A Deployment Platform for Dynamically Scaling Applications in the Cloud." IEEE Third International Conference on Cloud Computing Technology and Science IEEE, 2012: 506-510.

[27] Chang, Fangzhe, J. Ren, and R. Viswanathan. "Optimal Resource Allocation in Clouds." IEEE, International Conference on Cloud Computing IEEE Computer Society, 2015: 418-425.

[28] S. Genaud , J. Gossa.Cost-wait Trade-offs in Client-side Resource Provisioning with Elastic Clouds[C]. 2011 IEEE International Conference on Cloud Computing (CLOUD), Washington, DC,USA,2011,pp.1-8.

[29] Liu, Feifei, and X. Dong. "A Novel Elastic Resource Allocation Strategy of Virtual Cluster." Fourth International Symposium on Parallel Architectures, Algorithms and Programming IEEE, 2011: 168-173.

[30] Com I A. Amazon Elastic Compute Cloud[J]. 2011.

[31] Munz F. Middleware and Cloud Computing: Oracle on Amazon Web Services (AWS), Rackspace Cloud and RightScale[C]// munz & more publishing, 2011.

[32] Messias V R, Estrella J C, Ehlers R, et al. Combining time series prediction models using genetic algorithm to autoscaling Web applications hosted in the cloud infrastructure[J]. Neural Computing & Applications, 2016, 27(8): 2383-2406.

[33] Chieu T C, Mohindra A, Karve A A. Scalability and Performance of Web Applications in a Compute Cloud[C]// IEEE, International Conference on E-Business Engineering. IEEE, 2011: 317-323.

[34] Vaquero L M, Rodero-Merino L, Buyya R. Dynamically scaling applications in the cloud[J]. Acm Sigcomm Computer Communication Review, 2015, 41(1): 45-52.

[35] Koperek P, Funika W. Dynamic Business Metrics-driven Resource Provisioning in Cloud Environments[C]// International Conference on Parallel Processing and Applied Mathematics. Springer-Verlag, 2016: 171-180.

[36] Hasan M Z, Magana E, Clemm A, et al. Integrated and autonomic cloud

resource scaling[J]. 2012, 104(5): 1327-1334.

[37] Lorido-Botran T, Miguel-Alonso J, Lozano J A. Comparison of Auto-scaling Techniques for Cloud Environments[C]// CEDI. 2013.

[38] Xu W, Zhu X, Singhal S, et al. Predictive Control for Dynamic Resource Allocation in Enterprise Data Centers[C]// Network Operations and Management Symposium, 2006. NOMS 2006. Ieee/ifip. IEEE, 2009: 115-126.

[39] Urgaonkar B, Shenoy P, Chandra A, et al. Dynamic Provisioning of Multi-tier Internet Applications[C]// International Conference on Autonomic Computing, 2005. Icac 2005. Proceedings. IEEE, 2005: 217-228.

[40] Rimal B P, Choi E, Lumb I. A Taxonomy and Survey of Cloud Computing Systems[C]// Fifth International Joint Conference on Inc, Ims and IDC. IEEE Computer Society, 2015: 44-51.

[41] Bacigalupo D A, Hemert J V, Usmani A, et al. Resource management of enterprise cloud systems using layered queuing and historical performance models[C]// IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum. IEEE, 2010: 1-8.

[42] Ali-Eldin A, Kihl M, Tordsson J, et al. Efficient provisioning of bursty scientific workloads on the cloud using adaptive elasticity control[C]// The Workshop on Scientific Cloud Computing DATE. ACM, 2016: 31-40.

[43] Caron, E, F. Desprez, and A. Muresan. "Forecasting for Grid and Cloud Computing On-Demand Resources Based on Pattern Matching. " IEEE Second International Conference on Cloud Computing Technology and Science IEEE, 2010: 456-463.

[44] Galante, Guilherme, and L. C. E. D. Bona. "A Survey on Cloud Computing Elasticity." IEEE Fifth International Conference on Utility and Cloud Computing IEEE, 2013: 263-270.

[45] Coutinho E F, Sousa F R D C, Rego P A L, et al. Elasticity in cloud computing: a survey[J]. Annals of Telecommunications, 2015, 70(7): 289-309.

[46] Khatua S, Ghosh A, Mukherjee N. Optimizing the utilization of virtual resources in Cloud environment[J]. 2010: 82-87.

[47] Dutreilh X, Moreau A, Malenfant J, et al. From Data Center Resource Allocation to Control Theory and Back[C]// IEEE, International Conference on Cloud Computing. IEEE Computer Society, 2016: 410-417.

[48] Hasan M Z, Magana E, Clemm A, et al. Integrated and autonomic cloud resource scaling[J]. 2012, 104(5): 1327-1334.

[49] Messias V R, Estrella J C, Ehlers R, et al. Combining time series prediction models using genetic algorithm to autoscaling Web applications hosted in the cloud infrastructure[J]. Neural Computing & Applications, 2015: 1-24.

[50] Chieu T C, Mohindra A, Karve A A, et al. Dynamic Scaling of Web Applications in a Virtualized Cloud Computing Environment[C]// IEEE International Conference on E-Business Engineering. 2015: 281-286.

[51] Amiri M, Mohammad-Khanli L. Survey on Prediction Models of Applications for Resources Provisioning in Cloud[J]. Journal of Network & Computer Applications, 2017, 82: 93-113.

[52] Dutreilh X, Moreau A, Malenfant J, et al. From Data Center Resource Allocation to Control Theory and Back[C]// IEEE, International Conference on Cloud Computing. IEEE Computer Society, 2016: 410-417.

[53] Bu X, Rao J, Xu C Z. Coordinated Self-configuration of Virtual Machines and Appliances using A Model-free Learning Approach[J]. IEEE Transactions on Parallel & Distributed Systems, 2013, 24(4): 12-21.

[54] Barrett E, Howley E, Duggan J. Applying reinforcement learning towards automating resource allocation and application scalability in the cloud[J]. Concurrency & Computation Practice & Experience, 2013, 25(12): 1656–1674.

[55] Xu C Z, Rao J, Bu X. URL: A unified reinforcement learning approach for autonomic cloud management[J]. Journal of Parallel & Distributed Computing, 2012, 72(2): 95-105.

[56] Shen Z, Subbiah S, Gu X, et al. CloudScale: elastic resource scaling for multi-tenant cloud systems[C]// ACM Symposium on Cloud Computing. ACM, 2011: 5.

[57] Tesauro G, Jong N K, Das R, et al. A Hybrid Reinforcement Learning Approach to Autonomic Resource Allocation[C]// IEEE International Conference on Autonomic Computing. IEEE, 2006: 65-73.

[58] Urgaonkar B B, Shenoy P, Ch A, et al. Agile dynamic provisioning of multi-tier internet applications[C]// International Conference on Autonomic Computing. ACM, 2010: 217-228.

[59] Menasce D A, Dowdy L W, Almeida V A F. Performance by Design: Computer

Capacity Planning By Example[M]. Prentice Hall PTR, 2014.

[60] Alieldin A, Kihl M, Tordsson J, et al. Efficient provisioning of bursty scientific workloads on the cloud using adaptive elasticity control[J]. Sciencecloud Workshop on Scientific Cloud Computing, 2012: 31-40.

[61] Kumar R, Yadav S K. Scalable Key Parameter Yield of Resources Model for Performance Enhancement in Mobile Cloud Computing[J]. Wireless Personal Communications, 2017: 1-32.

[62] Lim H C, Babu S, Chase J S, et al. Automated control in cloud computing: Challenges and opportunities[J]. First Workshop on Automated Control for Datacenters & Clouds, 2016: 13-18.

[63] Lim H C, Babu S, Chase J S. Automated control for elastic storage[C]// International Conference on Autonomic Computing, Icac 2010, Reston, Va, Usa, June. 2010: 1-10.

[64] Park S M, Humphrey M. Self-Tuning Virtual Machines for Predictable eScience[C]// Ieee/acm International Symposium on CLUSTER Computing and the Grid. IEEE, 2009: 18-21

[65] Padala P, Hou K Y, Kang G S, et al. Automated control of multiple virtualized resources[C]// EUROSYS Conference, Nuremberg, Germany, April. 2015: 13-26.

[66] Ali-Eldin A, Tordsson J, Elmroth E. An adaptive hybrid elasticity controller for cloud infrastructures[J]. Network Operations & Management Symposium IEEE, 2012, 131(5): 204-212.

[67] Gong Z, Gu X, Wilkes J. PRESS: PRedictive Elastic ReSource Scaling for cloud systems[C]// International Conference on Network and Service Management. IEEE, 2016: 9-16.

[68] Khatua S, Ghosh A, Mukherjee N. Optimizing the utilization of virtual resources in Cloud environment[C]// IEEE International Conference on Virtual Environments Human-Computer Interfaces and Measurement Systems. IEEE, 2010: 82-87.

[69] Brown R G, Meyer R F, D'Esopo D A. The Fundamental Theorem of Exponential Smoothing[J]. Operations Research, 1961, 9(5): 673-687.

[70] Roy N, Dubey A, Gokhale A. Efficient Autoscaling in the Cloud Using Predictive Models for Workload Forecasting[J]. 2017: 500-507.

[71] Fang W, Lu Z H, Wu J, et al. RPPS: A Novel Resource Prediction and Provisioning Scheme in Cloud Data Center[C]// IEEE Ninth International Conference on Services Computing. IEEE, 2015: 609-616.

[72] Islam S, Keung J, Lee K, et al. Empirical prediction models for adaptive resource provisioning in the cloud[J]. Future Generation Computer Systems, 2012, 28(1): 155-162.

[73] Herbst, Nikolas Roman, S. Kounev, and R. Reussner. "Elasticity in Cloud Computing: What it is, and What it is Not." International Conference on Autonomic Computing 2013.

[74] OCDA. Master Usage Model: Compute Infratructure as a Service. Open Data Center Alliance (OCDA), 2012.

[75] WOLSKI, R. Cloud Computing and Open Source: Watching Hype meet Reality, May 2011.

[76] Reuven Cohen,COHEN, R. Defining Elastic Computing, September 2009.

[77] Cooper B F, Silberstein A, Tam E, et al. Benchmarking cloud serving systems with YCSB[C]// ACM Symposium on Cloud Computing. ACM, 2015: 143-154.

[78] Aisopos F, Tserpes K, Varvarigou T. Resource management in software as a service using the knapsack problem model[J]. International Journal of Production Economics, 2015, 141(2): 465-477.

[79] Espadas J, Molina A, Jiménez G, et al. A tenant-based resource allocation model for scaling Software-as-a-Service applications over cloud computing infrastructures[J]. Future Generation Computer Systems, 2013, 29(1): 273-286.

[80] Chen H, Kim M, Lei H, et al. Elastic and Scalable Publish/Subscribe Service: US, US20130007131[P]. 2016.

[81] Perez-Sorrosal F, Patiño-Martinez M, Jimenez-Peris R, et al. Elastic SI-Cache: consistent and scalable caching in multi-tier architectures[J]. Vldb Journal, 2015, 20(6): 1-25.

[82] Garg S K, Versteeg S, Buyya R. A framework for ranking of cloud computing services[J]. Future Generation Computer Systems, 2013, 29(4): 1012–1023.

[83] Han R, Ghanem M M, Guo L, et al. Enabling cost-aware and adaptive elasticity of multi-tier cloud applications[J]. Future Generation Computer

Systems, 2014, 32(2): 82-98.

[84] Pandey S, Voorsluys W, Niu S, et al. An autonomic cloud environment for hosting ECG data analysis services[J]. Future Generation Computer Systems, 2012, 28(1): 147-154.

[85] Islam S, Lee K, Fekete A, et al. How a consumer can measure elasticity for cloud platforms[C]//Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering. ACM, 2012: 85-96.

[86] Mehrmolaei S, Keyvanpour M R. Time series forecasting using improved ARIMA[C]// Artificial Intelligence and Robotics. IEEE, 2016: 92-97.

[87] El Desouky A A, Elkateb M M. Hybrid adaptive techniques for electric-load forecast using ANN and ARIMA[J]. Generation, Transmission and Distribution, IEE Proceedings, 2015, 147(4): 213-217.

[88] Vazquez C, Krishnan R, John E. Time series forecasting of cloud data center workloads for dynamic resource provisioning[J]. Dissertations & Theses-Gradworks, 2015.

[89] Messias V R, Estrella J C, Ehlers R, et al. Combining time series prediction models using genetic algorithm to autoscaling Web applications hosted in the cloud infrastructure[J]. Neural Computing & Applications, 2016, 27(8): 2383-2406.

[90] Hansen L K. Neural Network Ensemble[J]. IEEE Trans Pattern Analysis & Machine Intelligence, 2010, 12.

[91] ARIMA 模型[OL]. http://wiki.mbalib.com/wiki/ARIMA 模型.

[92] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[C]// Conference on Symposium on Opearting Systems Design & Implementation. USENIX Association, 2014: 10-10.

[93] M. Arlitt and T. Jin. 1998 world cup web site access logs[OL]. Available: http://www.acm.org/sigcomm/ITA/.

[94] Apache CloudStack[OL]. http://cloudstack.apache.org/.

[95] Apache Kafka[OL]. http://kafka.apache.org/.

[96] TPC[OL]. http://www.tpc.org/.

[97] Yang, Ping, and L. I. Jie. "Using LoadRunner to Test Web's Load Automatically." Computer Technology & Development (2007).

[98] Hwang R H, Lee C N, Chen Y R, et al. Cost optimization of elasticity cloud

resource subscription policy[J]. Services Computing, IEEE Transactions on, 2014, 7(4): 561-574.

[99] Amazon EC2 定价[OL]. https://aws.amazon.com/cn/ec2/pricing/on-demand/.

[100] Azure Auto Scaling[OL].https://blogs.msdn.microsoft.com/jianwu/2014/11/25/azure-auto-scaling/.

[101] Mi H, Wang H, Yin G, et al. Online Self-Reconfiguration with Performance Guarantee for Energy-Efficient Large-Scale Cloud Computing Data Centers[C]// IEEE International Conference on Services Computing. IEEE, 2016: 514-521.

攻读硕士学位期间发表的论文及其它成果

哈尔滨工业大学学位论文原创性声明和使用权限

学位论文原创性声明

本人郑重声明：此处所提交的学位论文《面向云服务的弹性调度算法的研究与实现》，是本人在导师指导下，在哈尔滨工业大学攻读学位期间独立进行研究工作所取得的成果，且学位论文中除已标注引用文献的部分外不包含他人完成或已发表的研究成果。对本学位论文的研究工作做出重要贡献的个人和集体，均已在文中以明确方式注明。

作者签名：张淼　　日期：2017 年 6 月 29 日

学位论文使用权限

学位论文是研究生在哈尔滨工业大学攻读学位期间完成的成果，知识产权归属哈尔滨工业大学。学位论文的使用权限如下：

（1）学校可以采用影印、缩印或其他复制手段保存研究生上交的学位论文，并向国家图书馆报送学位论文；（2）学校可以将学位论文部分或全部内容编入有关数据库进行检索和提供相应阅览服务；（3）研究生毕业后发表与此学位论文研究成果相关的学术论文和其他成果时，应征得导师同意，且第一署名单位为哈尔滨工业大学。

保密论文在保密期内遵守有关保密规定，解密后适用于此使用权限规定。

本人知悉学位论文的使用权限，并将遵守有关规定。

作者签名：张淼　　日期：2017 年 6 月 29 日

导师签名：胡德文　　日期：2017 年 6 月 29 日

致　谢

两年的硕士生涯一眨眼就要结束了，回想起这两年里发生过的点点滴滴，真的收获太多。这里不只有丰富的资源和浓郁的学术氛围使得自己各个方面的能力都有了很大的提升，更有来自良师益友的温暖和关怀，让我在学习知识、完成学业的同时，倍感温暖。还记得刚刚踏进哈工大校园时那个青涩的自己，太多的画面在脑海中闪现，太多的身影陪我一起走过，太多的日子因为有你们的存在而更加精彩。

首先，我要衷心的感谢我的导师左德承老师，老师在百忙之中耐心指导、及时给出非常有建设性的意见，从论文的选题到撰写完成，让我受益良多。另外要特别感谢张展老师，张老师对我而言是一位良师益友，很幸运能够师从张老师，从初入研究生涯时，张老师在学术上的指导带领我快速入门，到最后完成毕业论文的撰写认真批阅，数不清的点点滴滴……正是因为老师们的耐心指导让我一路克服困难、改正不足、顺利完成学业。在此，我衷心的感谢左老师以及张老师在我读研期间给予我的无私帮助和指导。

同时，也感谢实验的其他各位老师的教导与关心，感谢董剑老师对我关于毕业设计的指导，感谢吴智博老师在开题期间给我的中肯建议，感谢刘宏伟老师在中期期间给予我非常重要的建议，使我后期的工作更加明朗。感谢温老师除了学习上，在生活上还给予了我关心，让我感受到温暖。感谢实验室的所有老师，真诚的感谢这些老师将宝贵的知识传授给我。

我还想感谢实验室的师兄师姐，感谢李文浩师兄，在毕业设计上给予我耐心的指导，解答疑惑，谢谢李文浩师兄的仔细帮助。感谢戴荣倩师姐认真的指导，无论是学习中还是生活中都极尽所能，还要感谢师姐在找工作的过程中的指导，帮我走出迷茫。感谢周鹏师兄和赵岩师兄，两位师兄在我实验期间给予了非常多的帮助，谢谢你们对我的指导与帮助。此外，还要特别感谢吴娜师姐、冯丹青师姐、潘道华师姐，对我的指导以及生活上的关心。

感谢实验室的同学们：郭琪、李蒙、董文菁、付金伟、吴宽欣、陈超、赵伟博、成坚、刘雷，他们陪我走过学生生涯，感谢他们与我度过了这段难以忘记的时光，谢谢你们，难以割舍的友谊。

感谢我的室友们，对我而言，958 是一个永远的美好回忆，我们是闺蜜、是益友更是学习道路上的伙伴。因为她们，我的研究生涯变得丰富多彩，谢谢她们的陪伴和包容。

最后，我要极其感谢我的父母以及亲人，感谢他们对我的爱和呵护，还有一如既往的无私付出和支持。谢谢他们支持我做的每一个决定，给予我充分的信任与成长的空间。是爸爸妈妈无私的爱和教育让我拥有了完善的人格、感恩的心。我会把这些当做人生路上最宝贵的财富，好好珍藏，谢谢你们。