****



**研 究 生 毕 业 论 文**

**（申请硕士学位）**

|  |  |
| --- | --- |
| **论文题目** | 面向负载均衡的云计算资源分配方法研究 |
| **作者姓名** | 刘祥 |
| **学科、专业名称** | 计算机科学与技术 |
| **研 究 方 向** | 云计算 |
| **指导教师** | 窦万春 教授 |

**2018年5月18日**

**学 号： MG1533031**

**论文答辩日期： 2018年5月18日**

**指 导 教 师： （签字）**

****

**Resource Allocation Methods for Load-Balance over Cloud Computing Environment**

by

**Liu Xiang**

Supervised by

Professor **Dou Wanchun**

Department of Computer Science and Technology

Nanjing University

**南京大学研究生毕业论文中文摘要首页用纸**

毕业论文题目：面向负载均衡的云计算资源分配方法研究

计算机科学与技术 专业 2015 级硕士生姓名： 刘祥

指导教师（姓名、职称）： 窦万春 教授

**摘要**

近年来，随着互联网和信息技术的发展，云计算受到了工业界和学术界的广泛关注。云平台规模的扩大，一方面，给用户带来了计算能力更强的云服务，另一方面，加剧了云平台日益突出的能耗问题。诸多研究表明，云平台的能耗与其资源利用率密切相关，资源利用率低下将造成能源浪费，不仅增加平台运营成本，也违背了绿色云计算的理念。同时，通过提高云平台负载均衡，也有助于提高平台稳定性和资源利用率，降低平台能耗。最近，越来越多的高性能计算（High Performance Computing，HPC）应用抛弃了传统运行方式，选择成本更低的按需付费的云服务来运行。在此场景下，云平台的资源管理面临着诸多挑战：1）对于运行前知晓所有HPC应用资源需求信息的静态场景，如何利用这些已知信息提前计算出资源分配方案，以提高云平台运行HPC应用时的资源利用率和负载均衡。2）对于用户可能随时提交HPC应用的动态场景，如何基于当前空闲资源状态为新提交的HPC应用分配资源，以确保云平台高资源利用率和负载均衡。

针对运行大规模HPC应用的云平台在资源分配上面临的上述挑战，本文围绕该场景，对面向负载均衡的云计算资源分配方法进行了相关研究。具体而言，本文的主要工作包括如下几个方面：

1. 为了阐述运行HPC应用的云平台和HPC应用的关系，本文首先提出了一个结合两者的分层系统框架，考虑到HPC应用可以细分为一个或多个服务需求，该框架将其定义为元服务，云平台使用云服务来为元服务分配资源并运行。为了对方法开展进一步的设计与分析，本文基于此框架创建系统模型，推导出性能指标资源利用率和负载均衡的计算公式，并给出了目标优化函数。
2. 在静态场景中，运行HPC应用之前，所有组成HPC应用的元服务信息都已知。围绕该场景，本文提出了面向负载均衡的云计算资源静态分配方法。该方法分为四个步骤：元服务预处理、云服务资源使用监控、元服务子集静态资源分配和全局静态资源分配方法，利用已知的所有元服务的资源需求和时间需求信息计算出资源分配方案，HPC应用按照这些计算好的资源分配方案运行。该方法通过选择负载均衡最佳的云服务为元服务分配资源，以及根据元服务结束时间对其进行负载均衡再次优化，提高了静态场景中运行大规模HPC应用的云平台的资源利用率和负载均衡。
3. 在动态场景中，任意时刻都可能有新的HPC应用需要云平台运行，HPC应用运行过程可以看成云平台动态地为新到达的元服务分配资源并运行。围绕该场景，本文提出了面向负载均衡的云计算资源动态分配方法。该方法分为四个步骤：可分配资源表的维护、针对元服务到达的动态资源分配、针对元服务结束的动态资源分配和全局动态资源分配方法。该方法通过选择负载均衡最好的云服务为元服务分配资源，以及在负载均衡降低时实现负载重新均衡，在满足良好资源利用率的同时能够有效提高系统负载均衡。

**关键字**：云计算、负载均衡、资源利用率、资源分配

**南京大学研究生毕业论文英文摘要首页用纸**

THESIS：Resource Allocation Methods for Load-Balance over Cloud Computing Environment

SPECIALIZATION: Computer Science and Technology

POSTGRADUATE: Liu Xiang

MENTOR: Prof. Dou Wanchun

**Abstract**

In recent years, with the development of the Internet and information technology, cloud computing has received extensive attention from the industry and academia. The expansion of the scale of cloud platforms, on the one hand, brings users more cloud computing services with greater computing capacity. On the other hand, it exacerbates the increasingly prominent energy consumption problem of cloud platforms. Many studies have shown that the energy consumption of cloud platforms is closely related to the utilization of their resources. Low resource utilization will cause energy waste, which not only increases the operating costs of the platforms but also violates the concept of green cloud computing. At the same time, by improving load-balance of cloud platforms, it will also help improve stability and resource utilization, and reduce energy consumption. Recently, more and more high performance computing (HPC) applications have abandoned traditional operating methods and selected lower-cost pay-as-you-go cloud services. In this scenario, the resource management of cloud platforms faces many challenges: 1) For the static scenario that resource requirement information of all HPC applications before running is known, how to use the known information to calculate resource allocation scheme in advance to improve resource utilization and load-balance for cloud platforms. 2) For the dynamic scenario where users may submit HPC application at any time, how to allocate resources for the newly submitted HPC applications based on the current idle resource status to ensure high resource utilization and load-balance of cloud platforms.

Based on the aforementioned challenges in resource allocation for cloud platforms running large-scale HPC applications, this paper focuses on this scenario and conducts related research on load balancing methods for cloud computing resources. Specifically, the main work of this paper includes the following aspects:

1. To illustrate the relationship between HPC applications and cloud platforms running HPC applications, this paper first proposes a layered system framework that combines them. Considering that HPC applications can be subdivided into one or more service requirements, which the framework defines as meta services, cloud platforms use cloud services to allocate resources for meta services and execute them. In order to carry out further design and analysis of the method, this paper builds the system model based on the framework, deduces the calculation formula of resource utilization and load-balance, which are performance indicators in this paper, and gives the target optimization function.
2. In the static scenario, all the information of meta services which make up HPC applications is known before running the HPC applications. Based on this scenario, this paper proposes a method of static allocation for cloud computing resources on load-balance. The method is divided into four steps: meta service preprocessing, resource usage monitoring for cloud services, static resource allocation for meta service subset, and global static resource allocation method. The resources allocation schemata are calculated using the known resource requirements and time requirement information of all meta services. Then, the HPC applications are executed according to these calculated resource allocation schemata. This method allocates resources for meta services by selecting the cloud service with best load-balance, and optimizes load-balance according to the end time of meta services. This method can improve the resource utilization and load-balance of cloud platforms running large-scale HPC applications in static scenarios.
3. In the dynamic scenario, there may be new HPC applications that need to be executed at any time. The running process of HPC applications can be seen as cloud platforms dynamically allocate resources for the newly arrived meta services. Based on this scenario, the paper proposes a method of dynamic allocation for cloud computing resources on load-balance. The method is divided into four steps: maintenance of assignable resource tables, dynamic resource allocation for the arrival of meta services, dynamic resource allocation for the end of meta services, and global dynamic resource allocation method. This method allocates resources for meta services by selecting the best cloud service on load-balance, and achieves re-balance when load-balance is reduced. It can effectively improve system load-balance while satisfying good resource utilization.

**Keywords**: cloud computing, load-balance, resource utilization, resource allocation

目 录

[第一章 绪论 1](#_Toc514929097)

[1.1 课题背景 1](#_Toc514929098)

[1.2 云计算资源分配方法研究现状 3](#_Toc514929099)

[1.2.1 服务质量 3](#_Toc514929100)

[1.2.2 资源分配公平 4](#_Toc514929101)

[1.2.3 负载均衡 5](#_Toc514929102)

[1.3 HPC应用研究现状 7](#_Toc514929103)

[1.4 论文研究工作 9](#_Toc514929104)

[1.5 论文组织与结构 10](#_Toc514929105)

[第二章 面向负载均衡的云计算资源分配框架 12](#_Toc514929106)

[2.1 引言 12](#_Toc514929107)

[2.2 面向负载均衡的云计算资源分配框架 12](#_Toc514929108)

[2.3 面向负载均衡的云计算资源分配模型 15](#_Toc514929109)

[2.4 本章小结 18](#_Toc514929110)

[第三章 面向负载均衡的云计算资源静态分配方法 19](#_Toc514929111)

[3.1 引言 19](#_Toc514929112)

[3.2 问题实例分析 20](#_Toc514929113)

[3.3 云计算资源静态分配方法设计 22](#_Toc514929114)

[3.3.1 元服务预处理 23](#_Toc514929115)

[3.3.2 云服务资源使用监控 25](#_Toc514929116)

[3.3.3 元服务子集静态资源分配 27](#_Toc514929117)

[3.3.4 全局静态资源分配方法 29](#_Toc514929118)

[3.4 本章小结 30](#_Toc514929119)

[第四章 面向负载均衡的云计算资源动态分配方法 31](#_Toc514929120)

[4.1 引言 31](#_Toc514929121)

[4.2 问题实例分析 32](#_Toc514929122)

[4.3 云计算资源动态分配方法设计 33](#_Toc514929123)

[4.3.1 可分配资源表的维护 35](#_Toc514929124)

[4.3.2 针对元服务到达的动态资源分配 35](#_Toc514929125)

[4.3.3 针对元服务结束的动态资源分配 37](#_Toc514929126)

[4.3.4 全局动态资源分配方法 38](#_Toc514929127)

[4.4 本章小结 39](#_Toc514929128)

[第五章 实验评估与分析 40](#_Toc514929129)

[5.1 引言 40](#_Toc514929130)

[5.2 面向负载均衡的云计算资源静态分配方法实验分析 40](#_Toc514929131)

[5.2.1 实验部署 40](#_Toc514929132)

[5.2.2 实验结果及分析 41](#_Toc514929133)

[5.3 面向负载均衡的云计算资源动态分配方法实验分析 48](#_Toc514929134)

[5.3.1 实验部署 48](#_Toc514929135)

[5.3.2 实验结果及分析 49](#_Toc514929136)

[5.4 本章小结 53](#_Toc514929137)

[第六章 结束语 54](#_Toc514929138)

[6.1 工作总结 54](#_Toc514929139)

[6.2 未来展望 55](#_Toc514929140)

[参考文献 56](#_Toc514929141)

[致 谢 61](#_Toc514929142)

[**附录A 攻读硕士学位期间参与的论文** 63](#_Toc514929143)

[**附录B 攻读硕士学位期间申请的专利** 63](#_Toc514929144)

[**附录C 攻读硕士学位期间获奖情况列表** 63](#_Toc514929145)

[**附录D 攻读硕士学位期间参加的科研项目** 63](#_Toc514929146)

# 绪论

## 课题背景

近些年，随着互联网和通信技术的发展，越来越多的人利用通信设备（例如笔记本、智能手机和平板电脑等），来使用互联网提供的各种应用和存储服务。云计算模式提供了一种新颖的效用计算模型，它给用户带来了弹性的、按需获取的计算资源和基础设施服务。云计算资源通常以具有一定资源配置的云服务的形式提供，在最近的十年里，很多云服务提供商，如Amazon、Google、Microsoft等，它们的数据中心已经拥有了上百万台的服务器。得益于虚拟化技术，所有服务器的资源可以统一调度来提供用户所需的计算资源，于是如何在云平台上对规模庞大的资源进行合理分配成为了一个研究热点。然而，云平台规模的扩大，给用户带来了更大计算能力的同时，也增加了平台资源管理的难度。资源利用率一直是云平台提供商为了提高性能并降低运营成本而关注的焦点[1][2]。但事实上，对于目前的云平台资源利用率远非理想。低效的资源分配方法伴随着资源利用率的低下，不仅会带来资源的极大浪费[3][4]，还会增加云平台的运营成本，不符合绿色云计算的理念。由此可见，有效的资源管理和分配方法必不可少[5][6]。

虚拟化技术带来了诸多好处。一方面，支撑云平台的硬件基础设施（例如计算能力、内存、存储、网络带宽等）被虚拟化成逻辑资源，组成了弹性的、资源共享的云环境，为用户提供按需付费的云服务，而不用单独搭建自己的计算设施[7][8] [9][10]。具体而言，云计算资源通常以云服务的形式提供[11][12]，云环境为运行在网络上的各种应用程序提供了一个弹性资源池，以类似于电力、水和天然气等传统公用资源的付费方式提供几乎无限的计算资源[9][13][14]，并试图通过将大量的计算资源汇集到云平台来降低总体成本[15][16]。如图1-1所示，用户可以方便地使用自己的通信设备（例如个人笔记本、智能手机等）从云环境获取所需的计算资源。另一方面，计算资源可以由平台统一管理与分配，使得为云平台设计合理的资源分配方法成为了可能。



图1-1 用户使用通信设备从云环境获取计算资源

通常，云平台的资源管理策略和机制旨在满足不同的应用要求，例如让服务器更高效地运行[17]，降低平台的运营成本[8]，为用户提供更加透明的使用环境[16]，以可扩展的方式支持某种形式的互操作性[18]等。为了更具体地描述不同的应用要求，在文献[7]中，云平台资源分配策略被分为体现管理层的五个方面：访问控制、性能分配、负载均衡、能耗优化和QoS。为了改进资源管理策略，一些其他领域的概念也被引入，例如访问理论[19][20]、机器学习[21]、效用理论[22]和经济模型[23]等。

与此同时，近些年，人们越来越多地考虑高性能计算（High Performance Computing，HPC）应用，HPC应用程序是资源密集的科学工作流（就数据、计算和通信而言）。传统方式中，这些HPC应用一直针对网格和超级计算集群[13][24]，需要支付非常高的成本来运行。在过去的几年中，考虑到云环境正好能够弥补上述不足，越来越多的HPC应用在云环境中开发和运行[13][24][25][26]。对于云环境中运行的大规模HPC应用程序，在计算和通信过程中，如果资源利用率较低，将会造成能源的浪费[27][28]，导致负面的生态影响。如同文献[16]中提到的，合理的资源分配不仅可以大幅度降低运营成本、保护环境，而且可以保证长期情况下快速灵活的资源供给。同时，考虑到负载均衡有助于实现较高的用户满意度和资源利用率，以及避免瓶颈和过度配置等，在这一场景下，如何通过合理的资源分配来提高云平台的资源利用率和负载均衡具有很大的现实意义，但是目前很少有这方面的研究。

由上文可见，运行大规模HPC应用的云平台的资源管理面临着巨大挑战。在这一背景下，本文首先提出了一个集成大规模HPC应用和云平台的面向负载均衡的云计算资源分配框架，云服务作为中间层来协调HPC应用需求和资源分配。然后将该场景细分为静态场景和动态场景，针对这两个场景分别提出了面向负载均衡的云计算资源分配方法。

## 云计算资源分配方法研究现状

近年来，越来越多的学者和社会各界人士开始关注云平台的资源分配方法。该研究方向的许多工作主要针对资源分配方法的不同性能要求展开。例如，[7]中将云计算资源管理方法的要求划分为管理层的五个方面，分别是访问控制、性能分配、负载均衡、能耗优化和QoS。当然也有很多其他性能要求，例如资源分配公平、响应时间等。限于篇幅，本节将重点介绍面向几个常见性能要求的云计算资源分配方法的研究现状。

### 服务质量

近些年，服务质量（Quality of Service，QoS）在信息和通信领域变得越来越重要[18]。随着人们对按需付费的云服务的需求逐渐扩大，放大了云平台资源管理策略在QoS方面的不足。QoS指的是用户对云平台提供的云服务的整体效果的满意程度。不同的用户有着不同的需求，因此云平台提供的云服务也需要满足用户的QoS要求。

广泛用于提供可靠云服务的云平台，能耗大和资源利用率低下使平台提供商面临着巨大的挑战。对此，文献[29]中针对QoS和平台能耗，研究了如何确保良好QoS的同时，提高云平台的资源利用率并降低能耗。文中假设服务提供商提供不同类型的云服务，其中每个服务都具有不同类别的QoS要求。不同于在虚拟机（Virtual Machine，VM）上分配具有不同QoS类型的云服务的传统方法，该文引入了一种新的方法，即使对于不同的QoS要求，也将相同服务类型的请求迁移至同一个VM中，并删除潜在的故障请求来及时提高资源利用率。该方法在提升QoS、资源利用率和降低能耗方面明显优于其他传统方法。

为了提高云平台提供商的利润并且避免违反服务水平协议（Service Level Agreement，SLA[30]），Alsarhan等人考虑云平台可以分配的各种资源（如CPU、内存和存储等），租赁给具有QoS保证的客户端，并提供了一种新颖的SLA框架，使用价格参数来满足市场上所有类别的QoS要求[31]。该框架使用强化学习来推导虚拟机分配策略，该策略可以适应系统中的服务成本、系统容量以及服务需求的变化，以保证所有客户端的QoS。该方法首次尝试结合资源分配与基于强化学习的服务质量，以提高平台提供商的利润并避免违反SLA。

众所周知，具有安全风险的应用直接影响系统的性能，因为安全性和性能是两个成反比的量。由于云平台提供的云服务很受欢迎，需要提供有效的机制来应对各种各样的安全威胁。这意味着如果云平台提供商无法有效管理他们的计算基础设施，QoS可能无法满足客户的要求。为此，文献[32]中基于不同安全机制的服务性能评估结果，为云平台定义QoS驱动的资源管理与分配方法。方法中试图动态地、即时地改变计算资源来降低为提升QoS而带来的额外开销。

另外，也有部分研究对QoS的大小提出了的衡量的方法。文献[33]中引入SLA来规定每个用户需求的QoS。文中指出根据SLA来计算用户为达到所需QoS而必须支付的价格，以及云服务供应商在违反QoS时需要向用户支付的罚款。在此基础上提出了一个支持QoS并行数据流的资源分配架构。在架构中，每个节点都有一个调节和控制数据访问的资源管理器，以实现资源动态分配和选择。

### 资源分配公平

云平台通过提供计算资源来满足用户需求，并向用户收取资源使用费用。通过多资源分配来建立的云平台公平性是网络和资源管理的一项关键任务[34]，同时，资源共享是提高资源利用率的有效方法，但是用户的需求经常随时间变化，使得实现公平的资源分配成为了一个挑战。

对于云平台中的多资源分配场景，Ghodsi等人第一次对云平台中多资源分配问题进行了系统研究[35]。他们提出了一种名为DRF的资源分配方法来平衡所有用户的主导份额，并表明在分配结果中保证了一些理想的公平性。DRF很快吸引了大量的关注，并且已经推广到很多方面。后来，Joe-Wong等人对DRF进行优化并将其纳入一个统一的框架，该框架试图在资源分配公平和效率之间做出权衡[36]。类似的，Parkers等人以几种方式扩展了DRF，引入了资源零需求、对用户需求加权赋值以及不可分割任务这三个方面，并且还研究了DRF下的社会福利损失[37]。再后来，文献[38]中设计了一种称为DRFH的多资源分配机制，将DRF的概念从单个服务器推广到多个异构服务器。

然而，在YARN[39]和Mesos[40]等当前流行的云计算资源管理系统中实施的现有多资源公平策略，如主导资源公平（DRF[35]），并不适用于按需付费的云环境。对此，文献[41]中证明了这是因为云环境的无记忆特性导致了如下两个漏洞：用户可以通过作弊获得资源，以及用户可能无法获得他们在资源贡献方面应该获得的资源总量。于是文献[41]针对云平台资源分配的公平性提出了一个新的资源分配策略H-MRF，将DRF和资源贡献公平相结合，并开发出一种名为MRYARN的模型，并将它实现到YARN中。类似的，Haikun等人针对资源分配公平提出了一种针对基础设施即服务（Infrastructure as a Service，IaaS）云的新型资源分配机制--互惠资源公平（Reciprocal Resource Fairness，RRF）[42]。RRF采用两种互补的分层机制：云用户间资源交易和内部权重调整，可以在多用户之间实现对多种资源的公平分配。对于云平台提供商而言，与当前的IaaS云模式相比，RRF能够将平台收入提高了2.2倍。对于平台用户而言，RRF能够保证多用户之间95％的经济公平。

也有针对云平台的某个特殊资源的公平性的研究。与CPU和内存等资源不同，依赖于传统传输层协议的网络资源由于缺乏VM级别的网络隔离而受到不公平的影响。对此，文献[43]中，作者提出了一种预留策略来满足虚拟机的带宽需求，保持虚拟机带宽分配公平性的同时提高了平台资源利用率。另外，Jian等人提出了一种新的带宽分配协议Falloc[44]，用于保证虚拟机使用网络资源的公平性。该协议主要有两个目标：根据基本带宽需求保证虚拟机的带宽，和根据虚拟机的权重来分享剩余带宽。为了设计Falloc，它们为云平台带宽分配创建了价格博弈模型，并提出一种分布式算法来实现非对称Nash议价解决方案，最终实现了云平台虚拟机之间的高资源利用率和带宽分配公平。

### 负载均衡

云平台上的负载均衡指的是平台将动态工作负载分布到多个节点上，以确保没有单个资源被淹没或未被充分利用。通过确保每个计算资源的高效、公平分配，负载均衡有助于实现较高的用户满意度和资源利用率。适当的负载均衡还有助于最大限度地减少资源消耗、实现故障转移和可扩展性、避免瓶颈和过度配置等[45]。因此研究面向负载均衡的资源分配方法很有意义。

目前，很多相关研究都集中在虚拟机层面，实时虚拟机迁移有助于实现云平台的负载均衡和操作整合。文献[46]中，作者提出了一种基于云平台虚拟机迁移的动态负载均衡算法。该算法提出了基于分形方法的触发策略。该策略通过预测虚拟机的可能迁移时间来确定虚拟机迁移时机，这样做可以避免单一虚拟机负载瞬时升高的问题。与蚁群算法和蜜蜂算法等算法相比，该算法能够更加均衡地分配资源，并提高系统性能。在文献[47]中，Tsakalozos等人专注于大型IaaS云中的实时虚拟机迁移，以便及时完成迁移并且不会对已商定的SLA产生负面影响。该文提出了一个可扩展的分布式经纪人网络，利用一种称为MigrateFS的专用文件系统，监督云平台所有正在进行的虚拟机迁移操作的进度。文献[48]中，作者针对虚拟机调度中的负载均衡问题，根据任务执行时间跨度提出了虚拟机负载模型，并设计了一种改进的遗传算法。通过改进遗传过程中的选择和交叉算子，可以提高算法的优化精度，与传统的遗传算法和循环调度算法相比，改进后的遗传算法在虚拟机负载均衡方面能获得更好的表现。

文献[49]中认为云平台因为有大量资源存在，可能会产生资源利用不足。该文创新地把资源管理理解为对软件（例如数据库服务器管理系统、负载均衡器等）和硬件资源（例如CPU、存储器和网络）的选择、部署和运行时管理，以确保应用程序的性能。该文提出以一种基于资源利用率和预测即将到来的负载的资源管理方法来提升资源利用率和负载均衡。

文献[50]中从云服务的角度出发，指出负载不均衡是云平台在运行这些服务时经常面临的问题。当现有云服务被用户或主机周期性停止时，负载不均衡便会出现，在这种情况下实现负载均衡的一种方法是执行负载重新均衡。负载重新均衡是主机之间迁移服务的过程，以迁移成本为代价，确保资源分配的统一。该文使用一种改进的遗传算法，达到了很好的资源负载均衡。

文献[51]中从降低云平台运营成本的角度出发，介绍了一种基于阈值的动态比较和平衡算法（Dynamic Compare and Balance Algorithm，DCABA），用于云平台服务器的优化。不同于仅考虑基于服务器中CPU、内存和带宽的使用情况的传统资源管理算法，DCABA能最大限度地减少了要启动的主机数量，从而降低了云平台的运营成本。该方法可以有效利用可用资源，最大限度地减少运行主机的数量，从而达到降低平台运营成本的目的。

另外，还有一些研究关注资源管理算法本身。在文献[52]中，作者提出了一种基于遗传算法（Genetic Algorithm，GA）的新型负载均衡策略。该策略能够平衡云平台云基础架构的负载，同时尽量减少给定任务集的构建范围。该算法优于现有的先到先服务算法、Round Robing算法和局部搜索算法。文献[45]中提出了一种名为“中央负载均衡器”的负载均衡算法，以平衡云平台虚拟机之间的负载。结果表明，与以前的负载均衡算法相比，该文中的算法可以在大规模云计算环境中实现更好的负载均衡。

## HPC应用研究现状

高性能计算（HPC）应用程序需要高处理能力来满足研究、工程、医疗等项目高度密集和复杂的特点。传统方式中，不得不支付非常高的成本来运行HPC应用程序。最近的研究发现云环境对于执行HPC应用程序越来越有吸引力，HPC云（High Performance Computing Cloud）正好可以满足弥补上述不足[53][54]。对此，出现了很多关于HPC应用程序和云平台的研究。

Peter等人针对各种经典的HPC体系结构进行了基准测试，并对这些体系结构进行了成本分析，尝试以排队机制探索隐藏成本[55]。他们观察到具有大内存和高带宽的节点架构更适合用于HPC应用中的密集计算。同时，他们还得出结论，HPC应用规模越大越具成本效益。

随着HPC应用规模变大，能源消耗已成为HPC系统发展过程中最关键的问题之一[56][57]。针对HPCC的能耗问题，Alvarruiz 等人提出了一个适用于HPC集群和云平台架构的通用能源管理系统，可以在平台节点未使用时关闭该节点，在需要时运行该节点[57]。该系统使用一组连接器与不同的HPCC中间件（例如批处理排队系统或云管理系统）集成在一起，能够在需要时对不同的节点实施通电和断电操作（如Wake电力设备单元）。该方案的优点是可以与不同的资源管理中间件集成，而不需要对该中间件进行任何修改，并且能够有效地降低平台能耗。同样针对HPCC的系统能耗，Thang等人针对过度配置和功耗受限的HPC系统提出了需求感知电源管理框架[58]。作业调度器根据可用的硬件资源选择要运行的作业。电源管理器持续监控电源使用情况，预测执行作业时的性能并优化每个CPU的功耗上限，以便满足每项作业所需的性能要求，同时充分利用可用功率进行预算来提高系统吞吐量。类似的，Pierre-François等人提出了另一种作业调度机制，将回填算法扩展为能耗感知，同时，采用节点关闭技术改进资源管理方法来尽可能减少能源消耗[56]。这种组合在实现高资源利用率的同时，也可以让云平台在一段时间内能实现高效的能耗预测。

同样，云平台中节点间异构、动态环境以及其他虚拟机的干扰是高效HPC的瓶颈。例如，对于紧密耦合的迭代应用程序，一个较慢的处理器会降低整个应用程序的运行速度。对此，Abhishek等人为云平台中的紧耦合迭代的HPC应用提供了一个动态负载均衡器，不仅能够推断出虚拟化环境中的静态硬件异构性，也能适应由于多用户引起的动态环境[54]。通过对虚拟机进行持续的实时监控、检测和定期细化任务与分配，该负载均衡器可以适应云平台内部的动态变化。

另外，还有一些针对HPC完成时间、通信开销和运营成本的研究。在云平台上，由于HPC应用在许多节点上运行并涉及应用之间的密集通信，网络传递延迟造成了HPC应用要等待很长时间才能完成。为了减少通信开销来降低HPC应用的完成时间，文献[59]中介绍了一个仿真框架，以详细描述HPC应用的性能。该文提出的框架能够模拟已知的消息传递接口（Message Passing Interface，MPI[60]），以及具有给定算法和网络拓扑的应用。最后证明了通过系统设置和工作负载分配算法的正确组合，通信开销可以降低约44％。通常，云平台资源管理系统同样可以应用于HPCC中的计算、存储和网络等资源。但是，它面临着可扩展性、互操作性以及QoS的挑战。针对这些挑战，Govindarajan等人提出基于分布式云资源管理框架的解决方案[61]。该框架能够处理大量的应用请求并以可互操作的方式管理多种云资源。考虑到货币成本和性能是云平台上开发和运行HPC应用程序的重要优化因素，文献[24]中，提出了一种基于MPI的货币成本优化方法，该方法是开发HPC应用程序的一种主流编程范例。

## 论文研究工作

本文对面向负载均衡的云计算资源分配方法开展了一定的研究。首先，本文介绍了课题背景以及云计算资源分配方法和HPC应用的研究现状。然后给出了一个结合HPC应用和云平台的分层系统框架。接着针对静态场景提出了面向负载均衡的云计算资源静态分配方法。其次对动态场景提出了面向负载均衡的云计算资源动态分配方法。最后对这两个方法分别进行了实验对比与分析。具体而言，本文主要研究工作包括以下几个方面：

1. **介绍了云计算平台资源分配方法的课题背景与现状**。本文首先简单介绍了课题背景。然后介绍了云计算资源分配方法的研究现状，包括不同研究者提出的针对不同性能要求的云计算资源分配方法的研究现状，以及云平台和HPC应用的研究现状。
2. **提出了面向负载均衡的云计算资源分配框架**。为了阐述运行大规模HPC应用的云平台的分层结构，提出了一个结合HPC应用和云平台的分层系统框架，在这个框架中，将组成HPC应用的服务需求定义为元服务，云服务作为中间层来协调资源分配与管理。为了开展进一步的分析，本文还根据这个框架进行了系统建模，并对资源利用率和系统负载均衡这两个指标提出了具体的计算方法，并基于此给出了目标优化函数。
3. **提出了面向负载均衡的云计算资源静态分配方法**。静态场景中，在运行HPC应用之前，所有应用的元服务信息都是已知的。也就是说，在进行资源分配之前，所有待分配资源的元服务的资源需求信息都已获得，在此基础上对所有元服务进行资源分配方案的计算。计算完成后，这些元服务按照这些计算好的分配方案运行，运行过程中不再进行资源分配方案的选择。该方法在运行时能结合BFD算法的思想并考虑负载均衡，能够有效地提高系统资源利用率与负载均衡。
4. **提出了面向负载均衡的云计算资源动态分配方法**。动态场景中，在任意时刻都可能有新的HPC应用需要运行，这意味着新的元服务出现并请求分配资源，每当此时对它们进行动态的资源分配与运行。另一方面，当有元服务运行结束时，系统负载均衡很可能会降低，此时寻找负载均衡最差的云服务，对其上的元服务进行资源再次分配，以实现负载再次均衡。同样的，该方法在运行时能结合BFD算法的思想并考虑负载均衡，在满足良好的资源利用率的同时能够有效地提高系统负载均衡。
5. **分别对上述不同场景下的方法实施了相关实验并进行了对比分析**。本文最后针对上述两种方法分别实施了实验并进行了对比分析。第一个实验将静态场景中的资源分配方法与静态场景中的Greedy[62]和BFD[63]方法进行对比，证实了该方法在提升资源利用率和负载均衡方面的有效性。第二个实验将动态场景中的资源分配方法与动态场景下的Greedy[62]和BFD[63]方法进行对比，证实了该方法在保证良好资源利用率的同时能有效提升系统负载均衡。

## 论文组织与结构

基于上述课题背景和研究现状，本文将围绕运行大规模HPC应用的云平台，针对面向负载均衡的云计算资源分配方法展开研究。本文主要内容分为6章，论文组织结构如图1-2所示。

第1章绪论主要介绍了本文的课题背景、云计算资源分配方法的研究现状、主要研究工作和论文结构。

第2章介绍了面向负载均衡的云计算资源分配框架，将云服务作为中间层来协调资源分配和管理，并针对该框架中进行系统建模，提出了性能指标的计算方法和目标优化函数。

第3章围绕静态场景，详细介绍了面向负载均衡的云计算资源静态分配方法。该方法共分为四个步骤，分别是元服务预处理、云服务资源使用监控、元服务子集静态资源分配、全局静态资源分配。该方法在步骤3和步骤4中对负载均衡进行优化，目标是提高系统资源利用率与负载均衡。

第4章围绕动态场景，详细介绍了面向负载均衡的云计算资源动态分配方法。该方法同样共分为四个步骤，分别是可分配资源表的维护、针对元服务到达的动态资源分配、针对元服务结束的动态资源分配、全局动态资源分配。该方法在步骤2和步骤3中对负载均衡进行优化，目标是在满足良好的资源利用率的同时有效提高系统负载均衡。



图1-2 论文组织与结构

第5章详细介绍了实验配置和实验对比分析，使用Greedy[62]和BFD[63]算法作为对比方法。使用资源利用率和负载均衡这两个指标，分别对面向负载均衡的云计算资源静态分配方法和动态分配方法进行对比分析。

第6章作为结束语对全文做出了总结，根据研究中发现的不足以及在研究过程中遇到的问题，对今后可以开展的工作进行了展望。

# 面向负载均衡的云计算资源分配框架

## 引言

近年来，伴随着互联网和信息技术的发展，以及笔记本、移动终端、智能手机的普及，云计算受到了各界人士的广泛专注。凭借着规模效益，云平台借用最新的通信技术，可以降低提供服务的代价。然而，云平台给用户带来极大的数据处理能力的同时，资源利用率一直不太理想，云平台规模的扩大尤其加重了这个问题。得益于虚拟化技术，云平台的所有资源，例如计算能力、存储、带宽等，可以由平台统一调度与分配。低效的资源分配方法将会导致资源利用率的低下，不仅会对资源造成浪费，还会带来平台运营成本的增加。因此云平台需要合理高效的资源分配方法。

最近几年，人们因为各种需求越来越多地考虑HPC应用。因为使用云平台提供的云服务来运行HPC应用可以降低成本，越来越多的HPC应用在云平台上进行开发和部署运行。云计算资源分配方法的性能要求有多种，其中备受关注的一种是负载均衡，良好地负载均衡有助于实现较高的用户满意度和资源利用率，同时还有助于最大限度地减少能源消耗、实现故障转移和可扩展性、避免瓶颈和过度配置等[45]。然而，目前很少有围绕该场景，针对负载均衡的云计算资源分配方法的研究，因此，本文选择该方向开展研究。

在介绍面向负载均衡的云计算资源分配方法之前，本章将首先基于此场景给出一个面向负载均衡的云计算资源分配框架，并由上到下逐步分析每一层的含义和作用，指出该方法具体适用的层次。其次，为了方便下文对资源分配方法的研究，本章将会根据这个资源分配框架创建面向负载均衡的云计算资源分配模型，对相关概念给出定义，并指出本文设计的资源分配方法中性能指标的具体计算方法和目标优化函数。

## 面向负载均衡的云计算资源分配框架

众所周知，任何计算机相关应用的最上层一定是来自用户需求，最下层一定是一定是物理硬件的支持，云平台也是如此。对于拥有大规模HPC应用的云平台，本节对其由上而下进行分层，给出了在此场景下的面向负载均衡的云计算资源分配框架。如图2-1所示。

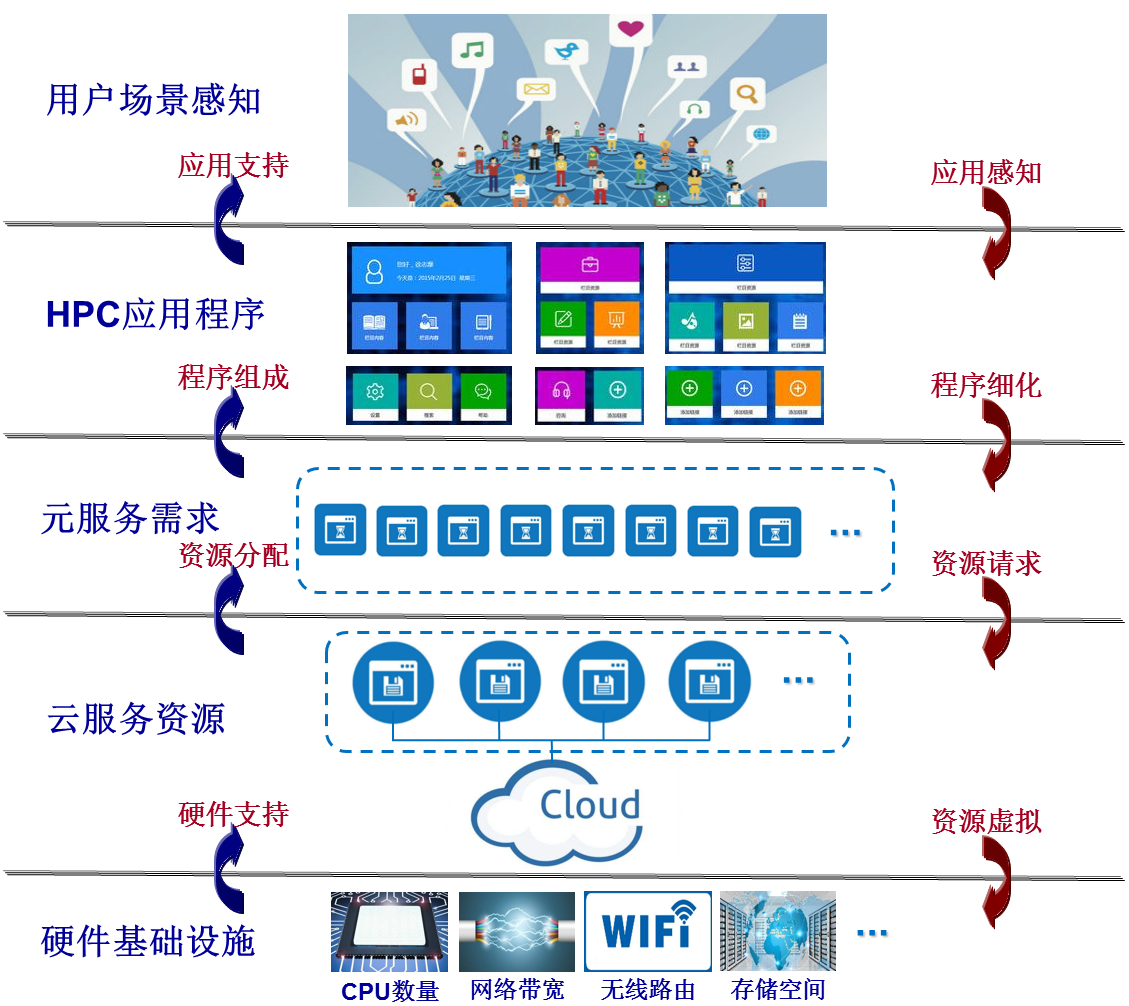


图2-1 面向负载均衡的云计算资源分配框架

从图2-1中可已看出，面向负载均衡的云计算资源分配框架分为5层，分别是：用户场景感知层、HPC应用程序层、元服务需求层、云服务资源层和硬件基础设施层。下面对每层进行详细说明。

（1）用户场景感知层。

用户在日常生产生活中，鉴于自己身处的场景，可能会遇到各种各样的具体需求，最简单的例如肚子饿了需要吃饭，复杂一点的例如需要计算从某个地点到另一个地点的最佳路径等。在这些需求中，有一种比较特殊，他们经过程序化处理能够交由计算机执行，例如计算机可以通过地点、距离等信息和输入的起点终点计算出这两个地点之间的最佳路径。这一类需求从用户场景感知层产生，可以被设计成应用程序交由计算机执行。

（2）HPC应用程序层

来自于用户场景感知层产生的用户需求，可以被设计成应用程序。当然，这样的应用程序首先可以尝试本地计算机来运行。但是本地的计算机毕竟处理器、存储和网络等资源有限，或者并不能满足应用程序运行环境，例如日常外出携带的手机、平板或者笔记本，它们的计算能力很小，这时就可以借助于资源弹性伸缩的云环境。于是，这样的用户需求可以设计成资源密集的工作流程，也就是HPC应用程序，由云平台来运行。

（3）元服务需求层

每个HPC应用是一个资源密集的工作流，由一个或多个更小粒度的服务需求组成，本文称之为元服务。每个元服务有具体的资源需求，例如需要多少处理能力（CPU）、内存和网络等；也有具体的时间需求，例如开始时间和运行时间。使用时间属性可以避免引入组成HPC应用的元服务之间的先后依赖关系，那么HPC应用的资源分配可以看成是对组成他们的若干元服务的资源分配。

（4）云服务资源层

得益于虚拟化技术，云数据中心的计算机和服务器资源可以统一管理，提供弹性、让用户按需付费使用的云环境。通常，云环境通过具有一定资源配置的云服务来运行上层传下来的元服务。类似于元服务，每个云服务也有一定的资源配置，例如多少处理能力（CPU）、内存和网络等。同一时间内，云服务因为资源有限，其能运行的元服务数量也因此受限。云平台通过使用这些云服务对元服务进行资源分配和运行。

（5）硬件基础设施层

如同运行应用程序的单个计算机一样，面向负载均衡的云计算资源分配框架的最底层也是由硬件构成。云平台统一管理着云数据中心的计算机和服务器，也就管理着它们的处理能力（CPU）、内存、存储和网络带宽等物理资源，为上面的云服务资源层提供硬件支持。

图2-1中，面向负载均衡的云计算资源分配框架从最上方的用户需求到最下方的云平台硬件基础设施分成5层，并给出了这5层之间的关系。一方面，用户场景感知层给出了HPC应用程序的来源，且HPC应用程序层可以细化为若干元服务。另一方面，通过云数据中心硬件基础设施层的支持，借用虚拟化技术，云平台可以通过提供具有一定资源配置的云服务来满足用户需求。这样就转换成了云平台利用其上的云服务来对元服务进行资源分配，这正是本文研究的内容，本文提出的方法将对云服务资源层与元服务需求层之间的资源分配进行负载均衡优化。为了对资源分配方法开展进一步的研究，下面一节将会依据此框架建立云计算资源分配模型。

## 面向负载均衡的云计算资源分配模型

为了方便下文进行进一步的分析，本节基于上一节提出的资源分配框架，对云服务层进行具体的建模，并给出了性能指标的计算方法和目标优化函数。表2-1列出了本节涉及到的关键符号。

表2-1 关键符号

|  |  |
| --- | --- |
| 符号 | 描述 |
| *N* | 元服务个数 |
| *M* | 元服务集合，*M* = {*m*1, *m*2, …, *mN*} |
| *W* | 云服务个数 |
| *S* | 云服务集合，*S* = {*s*1, *s*2, …., *sW*} |
| *mn* | *M*中第 *n* (1 ≤ *n*≤ *N*) 个元服务 |
| *sw* | *S*中第 *w* (1 ≤ *w*≤ *W*) 个云服务 |
| *Q* | 云计算资源种类个数 |
| *R* | 云计算资源种类集合，*R*= {*r*1, *r*2, .., *rQ*} |
| *rq* | *R*中第 *q* (1 ≤ *q* ≤ *Q*) 个资源 |
| *an,q* | 元服务 *m*n 对 *rq* 的资源需求数量 |
| *hw,q* | 云服务 *sw* 对 *rq* 的资源拥有量 |
| *uw* | 云服务 *sw* 的资源利用率 |
| *U* | 系统资源利用率 |
| *vw* | 云服务 *sw* 的负载均衡 |
| *V* | 系统负载均衡 |

云平台上的HPC应用通常可以从平台的云服务获取需要的虚拟化资源来运行，HPC应用通常由一个或多个元服务组成，这些元服务对云平台虚拟化资源（例如处理能力、内存、存储和网络带宽等）具有一定的需求，且在某个时刻需要运行一段时间。定义元服务如下：

**定义2-1（元服务）**。记 *mn* (1 ≤ *n* ≤ *N*) 表示某一时刻需要处理的第 *n* 个元服务，则所有元服务的集合可以表示为 *M* = {*m*1, *m*2, …, *mN*}，其中 *N* 表示 *M* 中元服务的个数。

元服务从云平台提供的云服务获取所需的虚拟化资源。定义云服务如下：

**定义2-2（云服务）**。记 *sw* (1 ≤ *w* ≤ *W*) 表示云平台第 *w* 个云服务，则所有云服务的集合可以表示为 *S* = {*s*1, *s*2, …, *sW*}，其中 *W* 表示 *S* 中云服务的个数。

通过虚拟化技术，云平台可以统一调度分配所有的虚拟化资源。云计算资源通常有很多种，例如处理能力（CPU）、内存、存储和网络带宽等。每个云服务对每种资源有着一定的容量，每个元服务对每种资源也有着一定的需求。定义云计算资源如下：

**定义2-3（云计算资源）**。云平台提供不同种类的虚拟化资源，表示为 *R*= {*r*1, *r*2, .., *rQ*}，其中 *rq* 表示第 *q* 种资源，*Q* 表示资源的种类数。

每个元服务对每种云计算资源有着不同的数量需求，当分配足够的资源才能运行，定义元服务资源需求如下：

**定义2-4（元服务*mn*的资源需求）**。元服务 *mn* (1 ≤ *n* ≤ *N*) 的资源需求表示为 *an* = {*an,q*| 1 ≤ *q* ≤ *Q*}，其中，*an,q* 表示元服务 *mn* 对第 *q* 种云计算资源的需求。

每个云服务对每种云计算资源有着一定的容量。定义如下：

**定义2-5（云服务*mn*的资源容量）**。云服务 *sw* (1 ≤ *w*≤ *W*) 的资源容量表示为 *hw* = {*hw,q*| 1 ≤ *q* ≤ *Q*}，其中，*hw,q* 表示云服务 *sw* 对第 *q* 种云计算资源的容量。

为了计算资源利用率，令 表示任意时刻元服务 *mn* 是否交由云服务 *sw* (1 ≤ *w* ≤ *W*) 运行， 可以由如下式子计算：

(2-1)

云服务 *sw* 对第 *q* 种云计算资源的利用率可以由如下式子计算：

(2-2)

云服务 *sw* 中所有不同种类云计算资源的利用率可以由如下式子计算：

(2-3)

在任意时刻，如果某个云服务上有元服务在运行，则表示其处于运行状态，否则为空闲状态。令 *K* 表示任意时刻处于运行状态的云服务的个数，可以由如下式子计算：

(2-4)

其中 *Fw* 是一个标志，用来表示云服务 *sw* 是否空闲，可以由如下式子计算：

(2-5)

那么任意时刻系统平均资源利用率可以由如下式子计算：

(2-6)

在本文中负载均衡由正在运行的云服务的资源利用率的方差来衡量。方差越大表示云服务之间的资源利用率越不稳定，也代表着负载均衡较差。相反，方差越小表示云服务之间的资源利用率越稳定，也代表着负载均衡较好。云服务的负载均衡可以由如下式子计算：

(2-7)

那么系统的负载均衡，也就是处于运行状态的云服务的平均负载均衡可以由如下式子计算：

(2-8)

基于以上定义与计算，面向负载均衡的云计算资源分配方法的目标在于最小化负载均衡，因此，其目标优化函数可以定义为最小化系统负载均衡，可以由如下式子表示：

(2-9)

s.t. (2-10)

(2-11)

(2-12)

其中公式(2-11)中的 表示任意时刻 *M* 中所有元服务对资源种类 *q* 的需求总和， 表示所有云服务对 *rq* 的总量。公式(2-12)中的 表示任意时刻，云服务 *sw* 上运行的所有元服务对 *rq* 的需求总和。

## 本章小结

本章对运行大规模HPC应用的云平台提出了一个分层的资源分配框架，并基于此对资源分配进行建模，提出了性能指标的计算方法和目标优化函数。

首先，本章根据云平台的架构和HPC应用的内部组成提出了一个面向负载均衡的云计算资源分配框架，该框架从最上层的用户场景感知层到最底层的硬件基础设施层分别介绍了每一层的作用以及层与层之间的关联。最后聚焦到与资源分配相关的元服务需求层和云服务资源层，具体地指出本文设计的方法以及负载均衡优化在框架中的位置。

接着对面向负载均衡的云计算资源分配创建模型，定义了组成HPC应用程序的元服务、云服务以及其他相关概念，然后一步步推导出优化指标的计算方法，最后给出目标优化函数。

# 面向负载均衡的云计算资源静态分配方法

## 引言

随着互联网和通信技术的发展，云计算平台无论是规模还是数量都逐渐增大。然而，云平台规模的扩大，增加了平台的资源管理难度，对平台资源利用率带来了不好的影响，资源利用率的低下会引发云平台能源消耗的增加，这样一来，不仅造成了资源的极大浪费，也违背了绿色云计算的理念。因此需要设计合理的云计算资源分配方法来提升系统资源利用率。

近些年，越来越多大规模HPC应用运行在云端，这些HPC应用由一个或者多个更小粒度的服务需求组成，每一个细粒度的服务需求包含一些资源需求，例如需要多少计算能力、内存、存储、带宽等，也包含时间需求，例如在何时运行、运行多长时间，在2.3节将其定义为元服务。而云平台通过提供具有一定资源配置的云服务来满足用户的需求。因此HPC应用的运行转换成了使用云服务为元服务分配云计算资源，并按照时间属性来运行。考虑到良好的负载均衡能够提升资源利用率和用户满意度，同时还能减少资源消耗、避免瓶颈和过度配置等[45]，因此，围绕该场景，研究面向负载均衡的云计算资源分配方法很有意义。

具体的，该场景可以细分为静态场景和动态场景。在静态场景中，在运行HPC应用之前，任何HPC应用的元服务的资源需求和时间需求都已获得。采用时间需求来避免引入HPC应用内部的先后依赖关系，那么所有HPC应用的运行实际上可以看成使用云平台提供的云服务来为所有元服务分配资源，并且云服务的资源容量有限，因此在任意时刻只能运行有限的元服务，在元服务获取到足够的云计算资源后按照时间需求来运行。因此，对于静态场景，可以在所有HPC应用运行之前，提前计算出所有元服务的资源分配方案，HPC应用运行时按照既定的方案运行，不再进行资源分配方案的选择。在动态场景中，最初并不知道任何HPC应用的信息，而是在任意时刻可能会有HPC应用需要分配资源来运行。因此，可以看成当有元服务到达并需要分配资源时，动态地为其选择合适的云服务来分配资源。

本章围绕静态场景，对面向负载均衡的云计算资源静态分配方法进行研究，主要目标是提升资源利用率和负载均衡。在方法设计中，将针对负载均衡进行优化。根据第二章创建的模型将该方法分为四个步骤，分别是元服务预处理、云服务资源使用监控、元服务子集静态资源分配和全局静态资源分配方法，其中，第四个步骤用于结合前面三个步骤，对全局提出一个静态分配方法，计算出所有元服务的资源分配方案。在第五章将会给出具体的实验分析，以证明该方法在提升系统资源利用率和负载均衡这两个方面的有效性。

## 问题实例分析

3.1节指出了负载均衡对提高云平台稳定性以及降低能耗方面的作用。围绕运行大规模HPC应用的静态场景，本节通过一个例子进行问题实例分析，指出该场景下资源分配方法中负载不均衡的问题。

如图3-1所示，在运行所有HPC应用之前，已经获得组成所有HPC应用的元服务，即*m*1、*m*2和*m*3，以及3种云计算资源，即*r*1、*r*2和*r*3，在图中分别用长方形、三角形和圆形表示，且元服务*m*1、*m*2和*m*3对这三种云计算资源的需求分别为*m*1: (4, 0, 1)，*m*2: (6, 1, 1)和*m*3: (3, 1, 1)。如图3-1(a)所示，云平台提供了两个云服务*s*1和*s*2用来为元服务分配资源，资源容量分别为*s*1: (8, 3, 4)和*s*2: (12, 4, 3)，假设这三个元服务具有相同的开始运行时间。

(a) (b)

图3-1 对元服务*m*1、*m*2和*m*3进行资源静态分配

考虑较为直接的资源分配方案。分别使用云服务*s*1和*s*2给元服务*m*1和*m*2分配资源，然后重新启用另一个空闲云服务*s*3为剩下的那个元服务*m*3分配资源，云服务*s*3的资源容量为*s*3: (12, 9, 9)。可以计算出云服务*s*1、*s*2和*s*3的资源利用率分别为0.25、0.361和0.0524。使用公式(2-6)和公式(2-8)可以计算出按照该方案运行元服务时的系统资源利用率和负载均衡，分别为0.221和0.0163。

接着介绍引入负载均衡优化的另一个资源分配方案。首先为元服务*m*1选择云服务，在云服务*s*1和*s*2之间，使用*s*1为*m*1分配资源将会获得更好的资源利用率，所以*m*1的资源分配方案为*s*1，此时云服务的剩余空闲资源量为*s*1: (4, 3, 3)和*s*2: (12, 4, 3)。接着为元服务*m*2选择云服务，在云服务*s*1和*s*2中，*s*1没有足够的资源为*m*2分配，而*s*2有足够的资源，因此，*m*2的资源分配方案为*s*2，此时云服务的剩余空闲资源量为*s*1: (4, 3, 3)和*s*2: (6, 3, 2)。最后为元服务*m*3选择云服务，若元服务*m*3选择云服务*s*1来分配资源，此时根据公式(2-8)计算出负载均衡为0.0108；若元服务*m*3选择云服务*s*1来分配资源，此时计算出负载均衡为0.056。负载均衡的值越小代表负载越均衡，因此*m*3的资源分配方案为*s*1。

图3-1(b)显示引入负载均衡优化的资源分配方案，使用*s*1共同给*m*1和*m*3分配资源，并使用*s*2给*m*2分配资源，按照这个方案运行元服务时，可以获得系统资源利用率0.465，高于第一种资源分配方案；在负载均衡方面，该方案获得的负载均衡值为0.0108，比第一种资源分配方案得到的0.0163更低，表示该方案获得了更好的负载均衡。因此，在利用已知的所有元服务的资源需求信息计算资源分配方案时，引入负载均衡优化，能够在提升资源利用率的同时提升负载均衡。

由此可见，对于运行大规模HPC应用的云计算平台静态场景，在计算元服务资源分配方案的过程中存在着负载不均衡的问题。可以对资源分配中某些过程针对负载均衡进行优化，来提高系统的资源利用率和负载均衡。因此，下文将围绕静态场景，提出面向负载均衡的云计算资源静态分配方法，在方法中对一些步骤进行面向负载均衡的优化。目标在于提升资源利用率和负载均衡。该方法将在第五章通过实验验证其有效性。

## 云计算资源静态分配方法设计

根据3.1和3.2.1节的介绍，本节将围绕静态场景，提出面向负载均衡的云计算资源静态分配方法（Static Resource Allocation），简称为SRA。具体来说，该方法由四个步骤组成，如图3-2所示。

**步骤1:** **元服务预处理**。对于HPC应用映射过来的元服务，它们有着不同的资源需求和时间需求。需要首先根据元服务开始时间对它们预处理。

**步骤2:** **云服务资源使用情况监控**。在进行资源分配方案的计算时，需要获取当前状态下云服务的资源使用情况。因此需要监控云平台上云服务的空闲资源情况。

**步骤3:** **元服务子集静态资源分配**。基于上述资源监控，正在使用中的云服务需要被元服务优先选择。如果这些云服务不够，需要租用更多空闲的云服务。

**步骤4:** **全局静态资源分配**。对所有的元服务，就需要一个全局的静态资源分配方法，来使用前三个步骤，为所有元服务计算出资源分配方案。

图3-2 面向负载均衡的云计算资源静态分配方法的步骤

为了描述该资源分配方法的各个步骤在第二章提出的资源分配框架中的位置和作用关系，该框架被稍微简化并突出了资源分配过程，得到了图3-3。如图3-3所示，在方法中，步骤1利用来自元服务需求层的元服务并为步骤3和步骤4输出元服务子集，每个子集中的元服务具有相同的开始时间。步骤2监控云服务的空闲容量，并将这些空闲容量值传送给步骤3，以实现对步骤2获取的元服务子集的资源分配。步骤4则针对整个过程进行管理，给出了全局的静态资源分配方法。该方法在步骤3和步骤4中引入负载均衡优化，在下文对方法步骤的具体分析中将会进行介绍。

图3-3 面向负载均衡的云计算资源静态分配方法与框架

### 元服务预处理

云计算平台上有大量的虚拟化资源，可用于运行HPC应用程序。如3.3.2节所述，HPC应用程序可以映射为多个具有一定资源需求和时间属性的元服务，云平台以云服务的形式为元服务的执行提供虚拟化资源。为了为元服务分配资源，首先需要对元服务预处理。

通常，除了资源需求，元服务还具有不同的时间需求，包括元服务运行的开始时间和持续时间。定义元服务*mn*的时间需求如下：

**定义3-1（元服务*mn*的时间需求）**。元服务 *mn* 的时间需求是一个二元组，表示为 *trn* = (*stn*, *dtn*)，其中 *stn* 和 *dtn* 分别表示为 *mn* 的开始运行时间和持续时间。

对于具有相同开始时间的元服务，它们需要同时分配所需的云计算资源。因此，*M*中具有相同开始时间的元服务将经过预处理并放到相同的元服务子集中。因此，最初所有的元服务可以按照开始时间分为若干个子集。元服务子集可以定义为：

**定义3-2（元服务子集）**。*M* 中具有相同开始时间的元服务将被分类到同一个元服务子集中。经过元服务预处理之后，所有的元服务子集可以表示为 *L* = {*l*1, *l*2,…, *lP*}，其中 *P* 表示 *L* 中子集的个数。

图3-4显示了一个元服务预处理的例子。在例子中，有5个元服务，即*M*1 ~ *M*5，这些元服务的开始时间分别为1.0，2.0，1.0，2.5和2.0。在对这些元服务进行预处理之后，它们被划分为3个子集，即*l*1，*l*2和*l*3。更具体地说，*l*1 = {*M*1，*M*3}，*l*2 = {*M*2, *M*5}，*l*3 = {*M*4}。



图3-4 元服务预处理示例

算法3.1显示了元服务预处理以及元服务子集的获取步骤。该算法的输入是元服务集合*M*，输出是元服务子集*L*。首先对元服务集合*M*按元服务开始时间的升序排序，然后将具有相同开始时间的元服务放入相同的子集中，最后将*M*分成几个子集。

|  |
| --- |
| **算法3.1:** 获取元服务子集 |
| **输入**：元服务集合 *M* = {*m*1, *m*2, …, *mN*}  **输出**：元服务子集 *L* = {*l*1, *l*2,…, *lP*}   1. 对*M* 中的元服务按开始时间递增排序; 2. *P* = 1, *f* = *st*1; // *st*1表示*m*1开始时间 3. **for** *n* = 1 to *N* **do** 4. **if** *stn* != *f* **then** 5. *P* = *P* + 1; 6. *f* = *stn*; 7. **end if** 8. 把*mn*加入*lP*; 9. **end for** 10. **Return** *L*; |

### 云服务资源使用监控

云平台以云服务的形式为元服务分配云计算资源。在计算元服务资源分配方案的过程中，需要经常获取平台上云服务的资源使用情况。因此，需要对其进行监控。

对于云平台上存在的各种类型的云计算资源，包括处理器资源（CPU）、带宽、存储、网络等。资源监控主要用于随时获取云平台上所有云服务的各类资源的空闲数量。云服务的空闲资源数量与每种资源的总量和已使用资源数量有关。对于每个特定的云服务，资源总量是不变的。因此，正在运行的云服务的空闲数量主要与资源使用有关。通过结合元服务已分配资源的记录，可以计算出当前分配状态下云服务的空闲资源数量。定义已分配资源的记录如下：

**定义3-3（资源分配记录）**。资源分配记录用来记录元服务的资源分配情况，表示为 *G* = { *g*1, *g*2,…, *gN*}，其中 *gn* (1 ≤ *n* ≤ *N*) 是元服务 *mn* 的资源分配记录。

在计算资源分配记录的过程中。元服务可能会分配给不同的云服务，因此*gn*中可能包含多个记录。对于*gn*中的第*i* (1 ≤ *i* ≤ |*gn*|)个资源分配记录，它是一个三元组，表示为*gn*,*i* = {*gsn*,*i*, *rsn*,*i*, *rdn,i*}，其中*gsn,i*是运行元服务*mn*的云服务，*stn*,*i*和*dtn,i*分别是分配给该云服务的开始时间和持续时间。通过对资源分配记录的分析和统计，可以获取当前时刻每个云服务的各种云计算资源的使用情况。由此，*R*中每种资源的空闲数量都可以相应地计算出来。使用空闲资源矩阵来保存每个云服务对每种云计算资源的剩余量，定义如下。

**定义3-4（空闲资源矩阵）**。空闲资源矩阵用来保存每个云服务中每种云计算资源的剩余量。表示为 *D* = (*dw*,*q*)*W*×*Q* (1 ≤ *w* ≤ *W*, 1 ≤ *q* ≤ *Q*)，其中 *dw*,*q* 是云服务 *sw* 第 *q* 种云计算资源的剩余量。

算法3.2显示了计算空闲资源矩阵的过程。该算法的输入是资源分配记录的集合*G*和请求计算空闲资源的时间*t*。该算法的输出是时间*t*的空闲资源矩阵*D*。在这个算法中，首先遍历整个云服务，并对其每种资源总量进行统计。然后通过资源分配记录集合*G*来确定开始时间小于*t*并且大于*t*的记录，从而计算出当前状态每种资源的剩余量。

|  |
| --- |
| **算法3.2:** 空闲资源监控 |
| **输入**：资源分配记录集合 *G* = { *g*1, *g*2,…, *gN*}  获取空闲资源的时间 *t*  **输出**：空闲资源矩阵*D*   1. **for** *w* =1 to *W* **do** 2. **for** *q* =1 to *Q* **do** 3. *f = hw*,*q*; 4. **for** *n* = 1 to *N* **do** 5. **for** *i* = 1 to |*gn*| **do** 6. **if** *gsn*,*i* ==*sw* **then** 7. **if** *t* ≥ *rsn,i* && *t* ≤ *rsn,i* + *rdn,i* **then** 8. *f -=* *an,q*; 9. **end if** 10. **end if** 11. **end for** 12. **end for** 13. *dw*,*q*=*f*; 14. **end for** 15. **end for** 16. **Return** *D*; |

### 元服务子集静态资源分配

为了实现优化负载均衡的目标，由于同一子集中的元服务具有相同的开始时间，因此应同时对其选择云服务，进行云计算资源的分配。

每个元服务对所有云计算资源都有一定的资源需求，其中有一个是主导资源。例如，当运行计算密集型元服务时，计算资源（CPU）是最重要的资源，可被视为主导资源。为了有效地执行元服务，应确认相同子集中所有元服务的主导资源类型。阈值*δq*用于判断元服务的主导资源类型是否为资源类型*rq*。如果元服务对*rq*的资源需求量超过此阈值，则应将该元服务置于另一类元服务子集。最终，*lp*可以被分成*Q*个子集，记为*lp* = {*l'p*,1, *l'p*,2, …, *l'p*,*Q*}。

由于元服务需要从云服务获取资源，因此云服务也需要按照主导资源类型进行分类。这里，阈值*δ'q*用于云服务类型判断。如果云服务*sw*的资源量*hw,q*大于*δ'q*，并且通过算法3.2判断出该云服务具有足够的空闲空间，则该云服务对应于第*q*类云计算资源，被分类到子集*s'q*中。最后，云服务集*S*被划分为*Q*个子集，即*S* = {*s'*1, *s'*2, …, *s'Q*}。

给元服务选择运行它的云服务时，*l'p*,*q*中的元服务优先在*s'q*中选择云服务。在本节中，元服务的资源分配可以看作是典型的背包问题。BFD算法的思想被用于资源分配过程。也就是说，元服务倾向于选择空闲资源最少的云服务来运行。

在上述过程之后，可能会有一些元服务无法分配资源。将这些主导资源种类为*q*的元服务都放在集合*vp*,*q*中。在*vp*,*q*中的元服务将遍历*s'z* (1 ≤ *z* ≤ *Q*, *z* ≠ *q*)中的云服务来选择。如果这些元服务的一部分仍然无法获得足够运行的云计算资源，那么*S*中空闲的云服务将被用于给这些元服务分配资源。

算法3.3显示了为元服务子集进行资源分配的过程。在算法3中，选择负载均衡最小的云服务对元服务进行资源分配，实现了对负载均衡的优化。算法的输入是元服务子集*lp*，输出是*lp*中元服务的资源分配记录。在该算法中，根据阈值*δq*判断每类资源的主导资源量，将*lp*中的元服务分为多个子集。同样，用于资源配置的云服务也被分为几个子集。在元服务和云服务分类之后，对于每个子集，利用BFD的基本思想，为元服务选择相关的云服务子集来分配资源。最后，剩余的元服务将被分配给空闲的云服务以进行资源分配。

|  |
| --- |
| **算法3.3:** 元服务子集静态资源分配 |
| **输入**：元服务子集*lp*  **输出**：*lp*中元服务的资源分配记录   1. **for** *i* =1 to |*lp*| **do** 2. **if** *ap,i,q* > *δq* **then** // *δq*是用于元服务类型判断的阈值 3. 把*lp,i*加入*lp*,*q'*; 4. **end if** 5. **end for** 6. 把未分配的元服务加入相关集合中; 7. **for** *w*= 1 to *W* **do** 8. 确定资源监控时间t; 9. 通过**算法3.2**获取 *dw*,*q*; 10. **if** *hw,q* >*δq'* && *dw*,*q* **>** 0 **then //***δq'*是用于云服务类型判断的阈值 11. 把*sw*加入*sq'*; 12. **end if** 13. **end for** 14. **for** *q* = 1 to *Q* **do** 15. 对*l'p*,*q*中的云服务按需求递减排序; 16. **for** *j* =1 to **|** *l'p*,*q***| do** 17. 找到*sq'*中可以支持*l'p*,*q*,*j*的云服务; 18. 使用公式(2-7)计算负载均衡; 19. 使用负载均衡最小的云服务为*l'p*,*q*,*j*分配资源; 20. **end for** 21. 把还未分配资源的元服务加入集合*vp*,*q*; 22. **end for** 23. **for** *q* = 1 to *Q* **do** 24. **for** 每个元服务*sq'* in*vp*,*q* **do** 25. *q'* = 1, *f*=0; 26. **while** *f* ==0 **do** 27. **if** 存在云服务可以给*sq''*分配资源 **then** 28. 使用这个云服务给*sq''*分配资源; 29. *f* = 1; 30. **else** *q' = q' +* 1; 31. **end if** 32. **end while** 33. **end for** 34. **end for** 35. 使用空闲云服务为剩下的元服务分配资源; |

### 全局静态资源分配方法

为了提高元服务执行过程中的负载均衡，本节提出了全局静态资源分配方法。利用算法3.1，元服务被分成若干个元服务子集。然后迭代地采用算法3.3对所有的子集进行资源分配。此外，由于相同云服务上的元服务很少同时结束运行，因此应调整资源分配记录以实现负载均衡的优化。

|  |
| --- |
| **算法3.4:** 全局静态资源分配方法 |
| **输入**：元服务集合*M*  **输出**：资源分配记录集合*G*   1. 使用**算法3.1**获取元服务子集*L*; 2. **for** *p* =1 to *P* **do** 3. 使用**算法3.3**对*lp*进行资源分配; 4. **for** *i* = 1 to |*lp*| **do** 5. *f* = 0, *j* = 1; 6. **while** *f* == 0&& *j* < *N* **do** 7. **if** *lp,i* == *mj* **then** 8. 更新资源分配记录*gj*; 9. *f* = 1; 10. **else** *j* = *j* + 1; 11. **end if** 12. **end while** 13. **end for** 14. **end for** 15. **for** *n* = 1 to *N* **do** 16. *ftn* = *stn* + *dtn*; 17. **end for** 18. 对*M*按照完成时间递减的顺序排序; 19. 获取不同完成时间的个数*C*; 20. **for** *c* = 1 to *C* **do** 21. 获取正在运行的元服务; 22. 使用**算法3.3**更新资源分配记录; 23. **end for** 24. **Return** *G*; |

元服务子集根据元服务开始时间来区分。在为元服务子集计算资源分配记录时，会为子集中的所有元服务生成资源分配记录。类似的，当有元服务结束时，部分处于运行状态的元服务将转移到其他云服务。这种优化的资源分配也可以看作是一个背包问题，算法3.3的基本思想也可以应用，即正在运行的元服务将被识别出来对这些元服务进行资源重新分配。

算法3.4显示了全局静态资源分配的过程。 该算法的输入是元服务集合*M*，输出是资源分配记录*G*。在该算法中，元服务子集*L*应该首先由算法3.1获得。然后，遍历所有获得的元服务子集，采用算法3.3为每个子集分配资源。同时，应该更新元服务的相应资源分配记录。最后，按照元服务结束时刻对资源分配记录进行调整。

## 本章小结

本章对面向负载均衡的云计算资源静态分配方法进行相关研究。在有着很多HPC应用的云平台，由于HPC应用由一个或多个元服务组成，当这些元服务在运行时，事实上是平台上的云服务在运行这些元服务。对元服务合理的资源分配可以降低平台的能耗开销和提升稳定性。因此，围绕静态场景，如何对元服务进行资源合理分配是一个挑战。

为了解决上述挑战，本章提出了面向负载均衡的云计算资源静态分配方法，在方法中的适当位置对负载均衡进行优化。在静态场景中，在所有元服务运行之前，它们的具体运行信息已经全部知晓，因此可以利用这些信息计算出每个元服务的资源分配记录，然后根据这些记录直接运行。本方法由四个步骤组成，目标在于提升资源利用率和负载均衡，将在第五章验证该方法的有效性。

# 面向负载均衡的云计算资源动态分配方法

## 引言

第三章围绕静态场景，对面向负载均衡的云计算资源静态分配方法进行了研究。如3.1节所述，低效的资源分配方法不仅会造成资源利用率差，也可能会造成一些其他不良的影响，例如低负载均衡。通过实现云平台的负载平衡，反过来也能够达到较高的用户满意度和资源利用率。此外，负载均衡还在如下方面有所帮助，例如减少资源消耗、实现故障转移、实现可扩展性、避免瓶颈和过度配置等[45]，符合绿色云计算的理念。因此，研究支持负载均衡的资源分配方法很有意义。

然而，现实生活中也经常出现需要实时为HPC应用分配资源来运行的场景。每当用户开发了新的HPC应用，提交到云平台运行，此时相当于需要实时为组成HPC应用的元服务动态分配资源。具体的，在动态场景中，任意时刻可能有一个或者多个云服务到达并请求分配云计算资源，每当此时，对这些元服务动态地分配资源并运行。显然第三章提出的面向负载均衡的云计算资源静态分配方法并不适用于动态场景。

因此，本章围绕动态场景，对面向负载均衡的云计算资源动态分配方法进行研究。主要目标是在保证良好的资源利用率的同时提升负载均衡。该方法同样分为四个步骤，分别是可分配资源表的维护、针对元服务到达的动态资源分配、针对元服务结束的动态资源分配和全局动态资源分配方法。其中，第四个步骤用于结合前面三个步骤，对全局提出一个动态资源分配方法，实时监控步骤二和步骤三的触发情况，并分别调用这两个步骤来运行。在第五章将会给出具体的实验分析，以证明该方法在系统资源利用率和负载均衡这两个方面的有效性。

## 问题实例分析

在4.1节已经提到围绕运行大规模HPC应用的云平台动态场景，对面向负载均衡的资源分配方法进行研究具有很强的现实意义，本节通过动态场景下的一个实例对问题进行分析。

如图4-1所示，在某一时刻，有3个元服务到达并请求资源分配与运行，即*M1*、*M2*和*M3*，以及2种不同类型的云计算资源，即A和B。元服务*M1*、*M2*和*M3*对虚拟化资源A和B的资源需求量分别为*M1*: (3, 3)、*M2*: (3, 1)和*M3*:(1, 2)。此时，有两个云服务s1和s2，它们的云计算资源容量分别为s1: (6, 5)和s2: (4, 7)，空闲资源剩余量分别为s1: (4, 3)和s2: (3, 4)。



图4-1 对元服务*M*1、*M*2和*M*3进行动态资源分配

一种较为直接的资源分配过程是依次为元服务*M1*、*M2*和*M3*分配资源。首先，对于元服务*M1*，使用云服务s1为其分配资源，此时云服务s1和s2的空闲资源剩余量变成s1: (1, 0)和s2: (3, 4)。接着，对于元服务*M2*，处于运行状态的云服务中只有s2有足够的资源为其分配，使用云服务s2为其分配资源后，云服务s1和s2的空闲资源剩余量变成s1: (1, 0)和s2: (0, 3)。最后，为元服务*M3*分配资源，此时云服务s1和s2都没有足够的资源为其分配，所以需要开启另外一个空闲的云服务s3为其分配资源，s3的资源容量为s3: (6, 6)，分配后空闲资源剩余量变成s3: (5, 4)。在这种分配过程下，可以计算出资源利用率和负载均衡分别为0.651和0.0833。

对于上述资源分配过程引入负载均衡优化。首先，对于元服务*M1*，云服务s1和s2均有足有的空闲资源。若使用云服务s1为其分配资源，可计算出分配后的负载均衡值为0.0833；若使用云服务s2为其分配资源，可计算出分配后的负载均衡值为0.0789。因为负载均衡值越大代表负载均衡越差，所以选择云服务s2为其分配资源，此时，云服务s1和s2的剩余空闲资源量为s1: (4, 3)和s2: (0, 1)。接着，对于元服务*M2*，处于运行状态的云服务中只有s1有足够的资源为其分配，使用云服务s1为其分配资源后，云服务s1和s2的空闲资源剩余量变成s1: (1, 2)和s2: (0, 1)。最后，使用云服务s1为元服务*M3*分配资源，云服务s1和s2的空闲资源剩余量变成s1: (0, 0)和s2: (0, 1)。按照如上分配过程，如图4-1右侧所示，可以计算出此时的资源利用率和负载均衡分别为0.964和0.0013。因此，相比于之前的方法，引入负载均衡优化后获得了更好的资源利用率和负载均衡。

由此可见，在动态场景的资源分配方法中，同样存在着负载不均衡的问题。针对上述问题，本章节围绕运行大规模HPC应用的云平台动态场景，设计了面向负载均衡的云计算资源动态分配方法，在方法的一些步骤中引入负载均衡优化。目标在于保证良好资源利用率的同时优化负载均衡。本文将在第五章通过实验验证该方法的有效性。

## 云计算资源动态分配方法设计

根据4.1和4.2.1节的介绍，本节将围绕动态场景，提出面向负载均衡的云计算资源动态分配方法（Dynamic Resource Allocation），简称为DRA。具体来说，该方法由四个步骤组成，如图4-2所示。

**步骤1: 可分配资源表的维护。**当元服务部署在云服务上时，需要获取云服务的空闲资源量。此外，在元服务被部署或撤销之后，也应该更新相应空闲资源量。

**步骤2: 针对元服务到达的动态资源分配。**一个或多个元服务可能在整个过程中的任意时刻到达并请求资源分配。到达后，所有这些元服务应该动态地分配资源。

**步骤3: 针对元服务结束的动态资源分配。**当元服务运行结束，整个系统的负载均衡将受到影响。因此，具有最差负载均衡的云服务上的元服务将被再一次动态地分配资源。

**步骤4: 全局动态资源分配方法。**在整个过程中，元服务可能随时到达或结束。此外，可分配资源表应根据这些操作进行刷新。因此，全局动态资源分配方法是必要的。

图4-2面向负载均衡的云计算资源动态分配方法的步骤

为了描述该资源分配方法的各个步骤在第二章提出的资源分配框架中的位置和作用关系，该框架被稍微简化并突出了资源分配过程，得到图4-3。

在面向负载均衡的云计算资源动态分配方法中，步骤1对云服务可分配资源表进行维护。步骤2检测任意时刻到达的元服务并动态地在云服务上进行资源分配，每次选择负载均衡最好的云服务，来优化系统负载均衡。每当元服务结束时，步骤3重新调整负载均衡，对负载均衡最差的云服务上的元服务进行又一次动态资源分配，以实现负载再次均衡。步骤4结合前三个步骤提出全局的动态资源分配方法。该方法在步骤2和步骤3中引入负载均衡优化，在下文对方法步骤的具体分析中将会进行介绍。

图4-3 面向负载均衡的云计算资源动态分配方法与框架

### 可分配资源表的维护

云计算平台上有大量云服务，可用于运行HPC应用程序。如前文所述，HPC应用可以映射为多个元服务，云平台以云服务的形式为元服务分配云计算资源并运行。为了找出具有足够资源的云服务来运行元服务，需要维护云服务的可分配资源表。可分配资源表的定义如下：

**定义4-1（元服务*sw*的可分配资源表）**。元服务 *sw* 的可分配资源表可以表示为*cw*={*cw*,*q*|1 ≤ *q* ≤ *Q*}，其中 *cw*,*q* 表示元服务 *sw* 中云计算资源 *rq* 的可分配数量。

当元服务部署在云服务上时，需要减少可分配资源表中的对应资源，并且在元服务运行结束时增加可分配资源表中的相应资源，如算法4.1所示。

|  |
| --- |
| **算法4.1:** 可分配资源表的维护 |
| **输入**：元服务*mn*，云服务 *sw*  云服务到达或者运行结束标记*Iwn*  **输出**：更新后的可分配资源表*cw*   1. **if** *Iwn* == 0 **then** 2. **for** *q* = 1 to *Q* **do** 3. *cw,q* = *cw,q* - *an,q*; 4. **end for** 5. **else** 6. **for** *q* = 1 to *Q* **do** 7. *cw,q* = *cw,q* + *an,q*; 8. **end for** 9. **end if** 10. **Return** *cw*; |

算法4.1显示了可分配资源表维护的主要步骤。该算法的输入是元服务*mn*，输出是更新后的可分配资源表。如果*Iwn*等于1，表示元服务*mn*部署在*sw*上，需要更新相应云服务的可分配资源表。否则表示元服务运行结束，也需要更新相应云服务的可分配资源表。

### 针对元服务到达的动态资源分配

当整个过程持续运行时，在任意时刻可能会有一个或多个元服务到达，需要分配资源并运行。这些元服务中的每一个都应该动态地选择适合的云服务来分配云计算资源。最基本的，所选择的云服务首先应该具有足够的分给元服务的可分配资源，然后，在分配给元服务之后，系统应该具有好的负载均衡。这些元服务保存在一个元服务集合中。

**定义4-2（待分配资源元服务集合*L*）**。在任意时刻，需要分配资源的元服务将保存在集合*L*中，表示为 *L* = {*lp*|1 ≤ *p* ≤ *P*}。其中 *P* 表示集合 *L* 中元服务的个数。

当这些元服务到达时，它们中的每一个都应该被动态地选择合适的云服务来分配资源。具体过程可见算法4.2。

|  |
| --- |
| **算法4.2:** 针对元服务到达的动态资源分配 |
| **输入**：待分配资源的元服务集合*L*  **输出**：*L*中元服务的资源分配方案   1. 对*L*中的元服务按资源需求总量递减排序; 2. 将所有运行中的元服务放入集合*E* ; 3. **for***i* = 1 to |*L*| **do** 4. **if** *E*中可以找到拥有足够资源分配给*l*i的云服务 **then** 5. 使用公式(2-7)计算负载均衡; 6. 将*li*分配给具有最小负载均衡值的云服务; 7. **else** 8. 将*li*分配给具有最小总资源量的空闲云服务; 9. **end if** 10. 使用**算法4.1**更新可分配资源表; 11. **end for** |

算法4.2给出了元服务资源分配的关键过程。该算法的输入是在任意时刻到达的元服务集合*L*，该算法的输出是*L*中每个元服务的资源分配方案。在该算法中，为了降低动态选择元服务的时间，首先按照总资源的递减顺序对*L*所有元服务排序。为了对负载均衡进行优化，只要正在运行的云服务能够找出满足相应元服务的资源需求，就可以将元服务*li*分配给负载均衡值最小的云服务。否则，将选择具有最小资源总量的空闲云服务分配给*li*。最后，使用算法4.1更新相该云服务的可分配资源表。

### 针对元服务结束的动态资源分配

当有元服务运行结束时，涉及的云服务的负载均衡可能会发生变化。如果该云服务上此时没有任何元服务在运行，该云服务的状态将会变成空闲，该云服务的负载均衡不变。否则，云计算平台的负载均衡可能会变差，为了实现负载均衡的目标，对于这种情况，应该检测系统的负载均衡，实现负载重新均衡。算法4.3显示了这样的过程。

|  |
| --- |
| **算法4.3:** 针对元服务结束的动态资源分配 |
| **输入**：结束运行的元服务*mn*，涉及到的云服务*sw*  **输出**：*L'*中的元服务的资源分配方案   1. 使用**算法4.2**更新*sw*的可分配资源表; 2. **if** *sw* 上没有任何元服务 **then** 3. 设置*sw*为空闲; 4. **for** *q* = 1 to *Q* **do** 5. *cw,q* = *hw,q*; 6. **end for** 7. **else** 8. 使用公式(2-7)计算运行中云服务的负载均衡; 9. 找出具有最高负载均衡值的云服务*sw*; 10. 把*sw'*中的元服务放入集合中*L'*; 11. **for** *q* = 1 to *Q* **do** 12. *cw',q* = *hw',q*; 13. **end for** 14. 设置*sw'*为空闲的; 15. 使用**算法4.2**对*L'*中的元服务进行动态资源分配; 16. **end if** |

算法4.3显示了针对元服务结束的动态资源分配过程。算法的输入是元服务*mn*和相关的云服务*sw*。在算法中，如果元服务*mn*结束，则应检查对应的云服务是否有其他元服务仍在其上运行。如果没有则可以设置*sw*为空闲，并相应地更新其可分配资源表。否则，正在运行的云服务的负载均衡可能会发生变化。因此，算法4.3通过公式(2-7)计算运行中的云服务的负载均衡的值，并获得具有最高负载均衡值的云服务*sw’*。因为*sw’*是具有最大负载或最小工作负载的云服务，因此需要为*sw’*中的所有元服务重新进行资源动态分配，由此对负载均衡进行了优化。

### 全局动态资源分配方法

为了提高云服务运行期间的负载均衡，本节介绍了全局动态资源分配方法。 利用算法4.1，为所有云服务维护可分配资源表以监视云服务的空闲资源剩余量。 然后采用算法4.2和4.3进行动态资源分配和资源收集。

|  |
| --- |
| **算法4.4:** 全局动态资源分配 |
| **输入**：任意时刻到来的元服务  **输出**：元服务资源分配方案   1. **for** *w* =1 to *W* **do** 2. **for** *q* =1 to *Q* **do** 3. *cw,q* = *hw,q*; 4. **end for** 5. **end for** 6. **while** 系统没有关闭 **do** 7. **if** 元服务集合*L*在任意时刻到达**then** 8. 使用**算法4.2**进行动态的资源分配; 9. **end if** 10. **if** 云服务*sw*上有元服务*mn*运行结束 **then** 11. 使用**算法4.3**进行动态的资源分配; 12. **end if** 13. **end while** |

算法4.4详细说明了全局动态资源分配的过程。该算法的输入可以是任何时刻到达的元服务，并且该算法可以动态地为相应的元服务分配所需的资源。在这个算法中，每个云服务的资源分配表应该在所有元服务到来之前被初始化。然后，对于任意时刻到来的元服务集合*L*，使用算法4.2为这些元服务分配资源；并在有元服务运行结束时使用算法4.3进行动态的资源分配。

## 本章小结

本章围绕动态场景，对面向负载均衡的云计算资源动态分配方法进行相关研究。在动态场景中，任意时刻都可能会有一个或多个元服务到达，每当有新的元服务需要云服务来分配资源时，本方法将会动态地为其选择合适的云服务并分配资源。该方法在元服务分配资源时对负载均衡进行优化，同时在负载均衡发生变化时，对负载均衡动态调整。目标是在保证良好的资源利用率的情况下提升系统负载均衡，第五章将会通过实验验证该方法的有效性。

# 实验评估与分析

## 引言

本文在第三章和第四章分别针对静态和动态场景设计了面向负载均衡的云计算资源分配方法，在方法中针对负载均衡进行优化。第三章的目标在于提升资源利用率和负载均衡，考虑到动态资源分配过程中的计算量不能过多，第四章的目标略有降低，目标在于确保良好资源利用率的同时能优化负载均衡。为了验证方法的有效性，本章将分别对这两个方法进行对比实验与分析，首先对每个实验分别介绍实验环境和参数设置，然后对获得的实验数据统计资源利用率和负载均衡，并分析这两个方法的有效性。

本章的结构安排如下：第二节对第三章中提出的面向负载均衡的云计算资源静态分配方法进行实验验证，第三节对第四章中提出的面向负载均衡的云计算资源动态分配方法进行实验验证，最后一节对本章进行总结。

## 面向负载均衡的云计算资源静态分配方法实验分析

### 实验部署

在实验中，流行框架CloudSim[26]被用来模拟元服务、云服务和云计算平台，同时，使用亚马逊云服务作为实验的模拟运行环境。亚马逊为用户提供了各种虚拟机实例，包括计算优化实例、内存优化实例、通用实例等。表5-1说明了客户端和远程云的具体配置。

表5-1 实验环境

|  |  |  |
| --- | --- | --- |
|  | 客户端 | 远程云 |
| 硬件 | CPU：Intel® Core™ i5-2520M 2.50GHz  内存：8.00GB | 亚马逊云VM实例类型：  m4.2xlarge (8 vCPUs，32G内存，1000Mbps带宽) |
| 软件 | Windows 10 | ubuntu 16.04，Cloudsim-3.0.3 |

表5-2显示了实验中的参数设置。为了评估本文第三章提出的面向负载均衡的云计算资源静态分配方法，使用6个具有不同规模的元服务数据集用于性能评估。每个数据集中的元服务数量设置为{500, 1000, 1500, 2000, 2500, 3000}。考虑四种不同类型的云计算资源，即CPU、内存、带宽和存储，然对这四种虚拟资源量化来衡量元服务和云服务的资源需求和供给关系。对于每个元服务，其数据记录是一个具有不同时间需求和资源需求的七元组。例如，对于元服务记录(2, 3.44, 4.21, 6, 2, 7, 3)，“2”是元服务的序号，“3.44”是元服务请求运行的开始时间，“4.21”是运行持续时间，“6”、“2”、“7”和“3”分别是对应的四种云计算资源的需求量。

表5-2 参数设置

|  |  |
| --- | --- |
| 参数 | 范围 |
| 元服务个数 | {500, 1000, 1500, 2000, 2500, 3000} |
| 云计算资源种类数 | 4 |
| 元服务资源需求量 | [1, 9] |
| 云服务资源容量 | [10, 30] |
| 元服务开始时间 | [0.0, 400.0] |
| 元服务持续时间 | (0.0, 20.0] |

### 实验结果及分析

在本节中，将进行一些对比分析以证明本文第三章提出的方法（缩写为SRA）在云计算资源分配上对于提升负载均衡和资源利用率的有效性。具体来说，文献[62]中基于Greedy算法的资源分配方法（缩写为Greedy），和文献[63]中基于BFD算法的资源分配方法（缩写为BFD），被用来与该方法做对比分析。在静态场景中，Greedy方法为具有相同开始时间的元服务计算资源分配方案时总是选择剩余空闲资源最少的云服务为其分配资源，而BFD方法需要首先对这些元服务按照所需资源总量由高到低排序，接着寻找当前情况下最适合的云服务为元服务分配资源。

（1）步骤1性能评估

如3.3.1节所述，利用步骤1，根据元服务开始时间将已知的元服务集合分为多个子集。每个子集中的元服务具有相同的开始时间。对于六个数据集，经步骤1处理后，获得的子集数分别为20、31、42、48、57和66。例如，当元服务的数量是500时，总共有20个子集，并且这20个子集中的元服务的数量分别是{24, 26, 24, 26, 28, 26, 27, 27, 24, 28, 29, 25, 25, 28, 25, 25, 26, 24, 27, 6}。

（2）步骤2性能评估

如3.3.2节所示，步骤2旨在监控所有云服务的剩余空闲资源。 在这一步中，通过对云服务资源分配记录的分析统计，可以随时检测到所有云计算资源的空闲资源量。例如，对于具有1000个元服务的数据集，当检测时间为10.06时，第55个云服务的空闲资源量为{9,17,10,7}，其中“9”、“17”、“10 ”和“7”分别是四种云计算资源的空闲资源量。

（3）步骤3性能评估

在第3.3.3节中，第3步旨在为元服务子集分配资源。通过步骤3，计算出该子集中的元服务的资源分配记录。例如，当用500个元服务的数据集时，生成的第101个元服务的资源分配记录为(3,17.78,3.71)，表明这个元服务将由第3个云服务分配云计算资源并运行，开始时间和持续时间分别为17.78和3.71。

（4）步骤4性能评估

在第3.3.4节中的步骤4中，采用全局静态资源分配方法来为所有元服务计算资源分配记录，并对步骤3生成的分配记录进行优化。在这个过程之后，将会得到每个元服务的资源分配记录。例如，对于规模为3000的元服务数据集，第223个元服务的分配记录是{(26, 100.47, 6.98)}。表示这个元服务仅部署在第26个云服务上。而对于第489个元服务，其相应的分配记录包含两个，即{(21, 73.92, 4.25), (28, 78.17, 5.62)}，表明它在21号云服务上运行时间段为[73.92, 78.17]，然后在28号云服务上运行时间段为[78.17, 83.79]，因此包含两条资源分配记录。

经过步骤4之后，将生成所有元服务的资源分配记录。下面将根据公式(2-6)和公式(2-8)分别对资源利用率和负载均衡进行统计与评估。

（5）资源利用率性能分析

资源利用率与使用到的云服务数量密切相关。通过对5.2.1节提到的6个不同规模的数据集分别实施Greedy、BFD和SRA这三个不同方法，统计实验中这三个不同方法使用到的云服务的数量。图5-1显示了Greedy、BFD和SRA在使用到云服务数量上的对比情况。如图5-1所示，Greedy、BFD和SRA所使用到的的云服务数量都随数据集规模的变化而变化。从图中我们还可以发现，更多的元服务需要更多的云服务来运行。此外，SRA方法在这三个方法中所使用到的云服务数量最少。

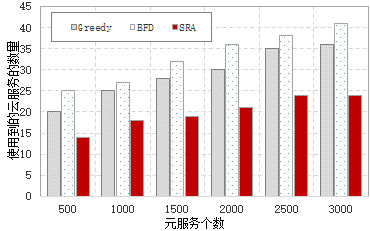
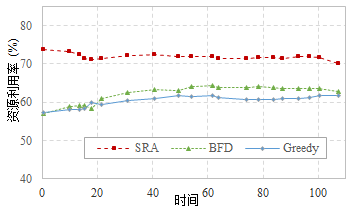
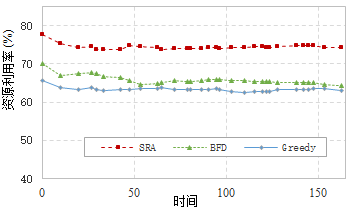


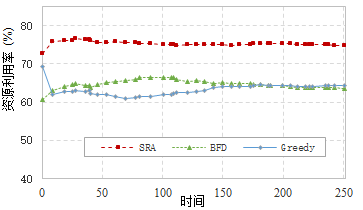
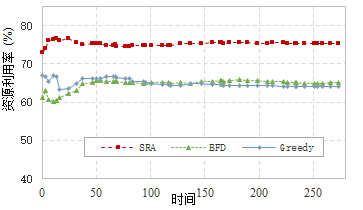
图5-1 Greedy、BFD和SRA在使用到的云服务数量方面的比较

上文从使用到的云服务数量的角度分析了不同方法，下面将进一步从整个运行过程的不同时刻分析BFD、Greedy和SRA这三个方法在平均资源利用率方面的比较情况。图5-2中的六个子图显示了按照Greedy、BFD和SRA这三个方法计算出的资源分配记录来运行元服务时，平均资源利用率随时间的变化情况。从图中可以看出，在使用SRA计算出的资源分配记录运行元服务的过程中，对所有数据集都达到了比Greedy和BFD更好的资源利用率。

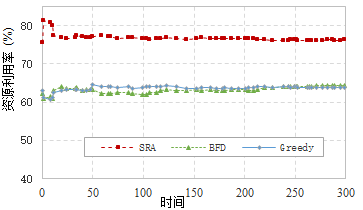
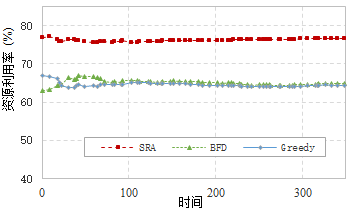
例如，在图5-2(f)中，我们可以发现，在不同的时刻，本文提出的面向负载均衡的云计算资源静态分配方法SRA可以比Greedy和BFD取得更高的资源利用率。当时间点为200时，BFD和Greedy的资源利用率大约为64％，而SRA可以获得将近75％的资源利用率。

(a) 元服务数量为500 (b) 元服务数量为1000

(c) 元服务数量为1500 (d) 元服务数量为2000

(e) 元服务数量为2500 (f) 元服务数量为3000

图5-2 Greedy、BFD和SRA在实时资源利用率方面的比较

根据图5-2中的数据，可以计算出这三种方法对整个过程的平均资源利用率。图5-3显示了Greedy、BFD和SRA的整个过程的平均资源利用率的比较情况。从图中可以发现，对于不同规模的数据集，本文提出的资源分配方法SRA比Greedy和BFD获得更好的整体平均资源利用率。

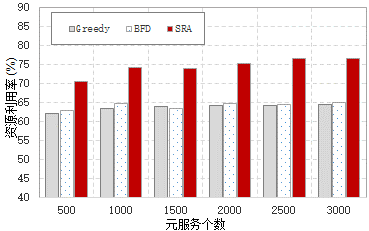


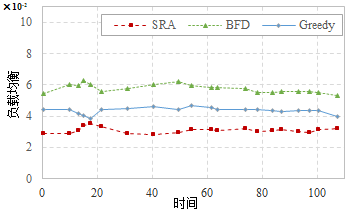
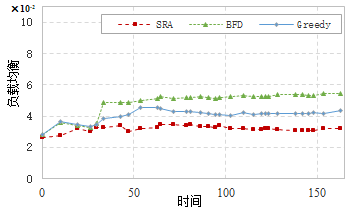
图5-3 Greedy、BFD和SRA在整体资源利用率方面的比较

（6）负载均衡性能分析

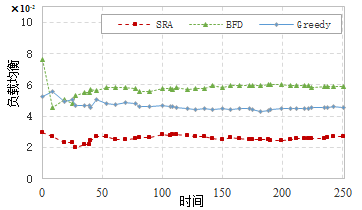
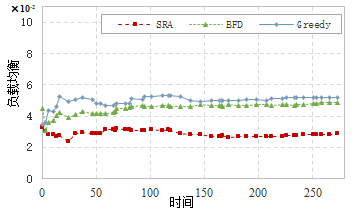
与上文对资源利用率的对比分析类似，下文将对Greedy、BFD和SRA这三个不同方法得到的资源分配记录做进一步负载均衡方面的分析。

图5-4中的六个子图显示了按照Greedy、BFD和SRA计算出的资源分配记录来运行元服务时，在不同时刻负载均衡的比较情况。从图中可以看出，本文提出的方法SRA在这三个方法中获得了最小负载均衡值，即SRA在负载均衡方面优于Greedy和BFD方法。

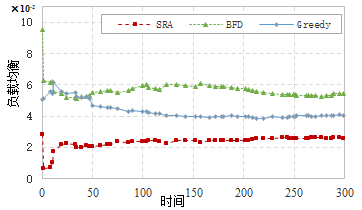
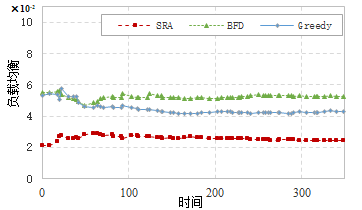
例如，在图5-4(c)中，我们可以发现，在不同的时刻，SRA可以获得比Greedy和BFD获得更低的负载均衡值。当时间点为150时，BFD达到接近0.06的负载均衡值，Greedy的负载均衡值达到0.04以上，而本文设计的的SRA方法得负载均衡值仅为约0.025。

(a) 元服务数量为500 (b) 元服务数量为1000

(c) 元服务数量为1500 (d) 元服务数量为2000

(e) 元服务数量为2500 (f) 元服务数量为3000

图5-4 Greedy、BFD和SRA在负载均衡方面的比较

同样，通过对图5-4中的数据进行统计，可以计算出这三种方法作用于不同数据集时，整个过程的平均负载均衡。图5-5显示了Greedy、BFD和SRA方法的整体平均负载均衡的比较情况。从图中我们可以发现，SRA取得了这三个方法中最好的整体负载均衡，这表明SRA在负载均衡方面可以比Greedy和BFD发挥更好的效果。

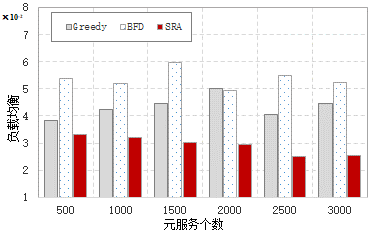


图5-5 Greedy、BFD和SRA在整体负载平衡方面的比较

上文详细地对资源利用率和负载平衡进行了对比分析。从图5-1到图5-5，我们可以推断，面向负载均衡的云计算资源静态分配方法SRA比BFD和Greedy能更好地实现负载均衡。为了更清楚地进行比较分析，本节计算了SRA、BFD和Greedy相比，在资源利用率和负载均衡方面提高的百分比。

表5-3显示了SRA和BFD与Greedy相比资源利用率提高的百分比。从表中可以直观地看出，在资源利用率方面，SRA优于BFD和Greedy。例如，当元服务数量为500时，SRA在资源利用率方面比Greedy好13.73％。表中的数字“-0.75”低于0，这意味着BFD在资源利用率方面比Greedy低 0.75％。

表5-3 SRA和BFD与Greedy相比资源利用率提高的百分比

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 方法 | 元服务个数 | | | | | |
| 500 | 1000 | 1500 | 2000 | 2500 | 3000 |
| SRA (%) | 13.73 | 17.06 | 15.69 | 17.28 | 19.08 | 18.98 |
| BFD (%) | 1.24 | 2.05 | -0.75 | 0.81 | 0.258 | 0.73 |

表5-4显示了与Greedy相比，SRA和BFD在负载均衡方面提高的百分比。与上面对资源利用率的对比结果相似，我们可以直观地发现SRA在负载均衡方面优于BFD和Greedy。例如，当元服务数量为2000时，BFD在负载均衡方面比Greedy好1.56％，而SRA比Greedy好41.05％。

表5-4 SRA和BFD与Greedy相比负载均衡提高的百分比

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 方法 | 元服务个数 | | | | | |
| 500 | 1000 | 1500 | 2000 | 2500 | 3000 |
| SRA (%) | 12.97 | 24.11 | 32.62 | 41.05 | 37.88 | 43.28 |
| BFD (%) | -40.84 | -22.67 | -34.24 | 1.56 | -35.88 | -17.35 |

## 面向负载均衡的云计算资源动态分配方法实验分析

### 实验部署

为了检验为本文提出的面向负载均衡的云计算资源动态分配方法在云平台支持负载均衡方面的有效性，本节进行了一些实验，实验环境同5.2节。

表5-5列出了本次实验中相关参数设置。为了上述方法，将6个具有不同数量的元服务数据集用于性能分析。每个数据集中的元服务数量设置为{1000, 2000, 3000, 4000, 5000, 6000}，考虑四种云计算资源，即CPU、内存、带宽和存储。元服务的记录是一个6元组。例如，元服务记录(2.2, 6.8, 3, 7, 5, 9)意味着元服务在时间点“2.2”开始，持续运行时间为“6.8”，并且对四种云计算资源的需求分别为“3”、“7”、“5”和“9”，分别代表对四类资源的需求量。

表5-5 参数设置

|  |  |
| --- | --- |
| 参数 | 范围 |
| 元服务个数 | {1000, 2000, 3000, 4000, 5000, 6000} |
| 云计算资源种类数 | 4 |
| 元服务资源需求量 | [1, 9] |
| 云服务资源容量 | [10, 30] |
| 元服务开始时间 | [0.0, 240.0] |
| 元服务持续时间 | (0.0, 20.0] |

### 实验结果及分析

在本节中，将会进行一些分析，以证明面向负载均衡的云计算资源动态分配方法（缩写为DRA）的有效性。具体来说，本文将DRA方法与文献[62]中提出的基于Greedy算法的资源分配方法（缩写为Greedy），以及文献[63]中提出的基于BFD算法的资源分配方法（缩写为BFD）进行对比分析。在动态场景中，Greedy方法在为新到达的元服务选择云服务来分配资源时总是选择可分配资源最少的云服务为元服务分配资源，而BFD方法需要对新到达的元服务按照所需资源总量由高到低排序，再依次为其根据当前处于运行状态的云服务的可分配资源表选择最合适的为其分配资源。

（1）步骤1性能评估

如4.3.1节所述，当元服务部署在云服务上或者离开使用的云服务时，将会调用步骤1。例如，当具有资源需求(4, 3, 7, 4)的元服务部署在具有可分配资源表(5, 6, 11, 16)的云服务上，该云服务的可分配资源表将会更新为(1, 3, 4, 12)。

（2）步骤2性能评估

如4.3.2节所述，步骤2被设计为在任意时刻有一个或多个元服务到达时，动态地为这些元服务分配资源。例如，几个元服务在某个具体时刻到达并请求资源分配，其中的每个元服务将会动态选择合适的云服务来分配云计算资源并运行。

（3）步骤3性能评估

如4.3.3节所述，第3步被设计为在运行结束的元服务离开云服务时，如果该云服务上仍有元服务运行，负载均衡很可能会降低，此时需要负载重新均衡。例如，当有元服务在某个具体时刻完成时，如果在相关的云服务上没有元服务运行，则该云服务将被设置为空闲。否则，整个过程的负载均衡将受到影响。因此，负载均衡最差的云服务上的所有元服务将被再一次动态分配资源。

（4）步骤4性能评估

如4.3.1节所述，当元服务到达并需要分配资源时，以及当元服务在运行结束时，需要设计全局的动态资源分配方法来实时监控步骤2和步骤3的两种不同情况，直到整个过程因满足条件而停止。

当整个过程持续按照步骤4运行时，元服务会动态选择云服务并得到资源分配。通过利用公式(2-6)来计算资源利用率，利用公式(2-8)来计算负载均衡，并在此基础上进一步具体分析。

（5）资源利用率性能分析

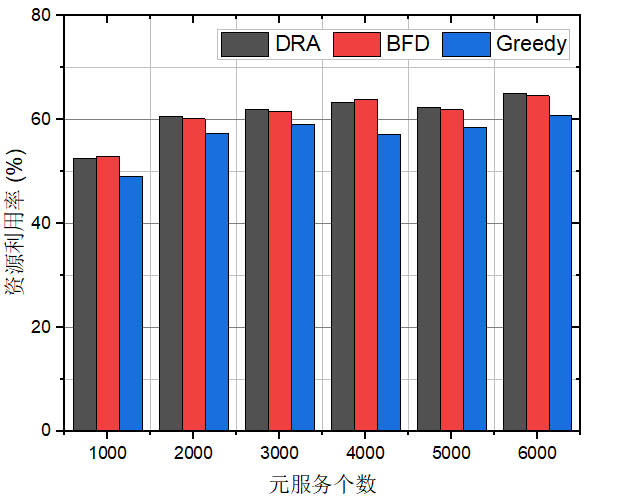
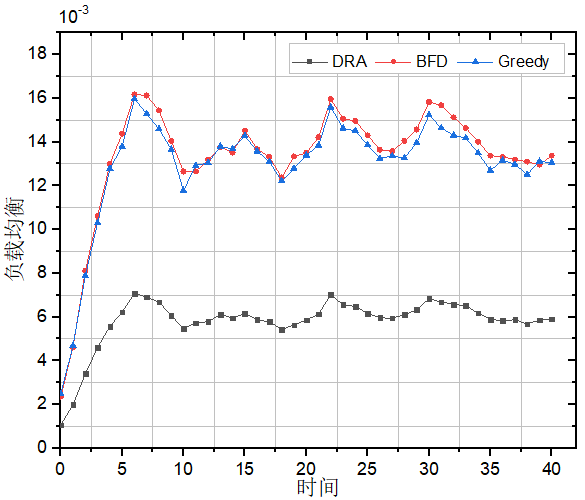
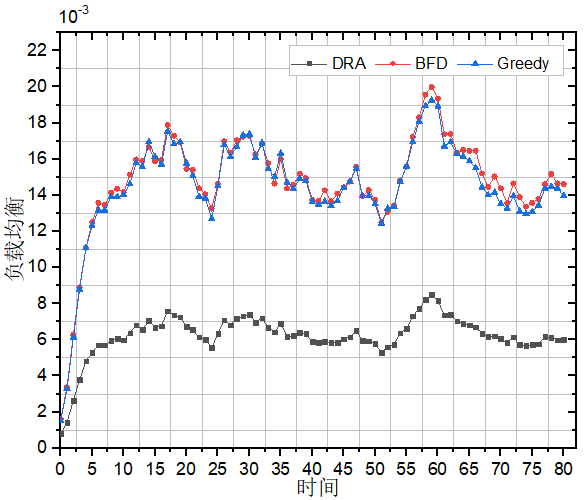
通过执行Greedy、BFD和DRA，可以计算出整个过程的平均资源利用率。图5-6显示了这三种方法在平均资源利用率方面的比较。从图中可以看出，本文提出的DRA方法在整体平均资源利用率方面与BFD相似，并且优于Greedy。

图5-6 DRA、BFD和Greedy在整体平均资源利用率方面的比较

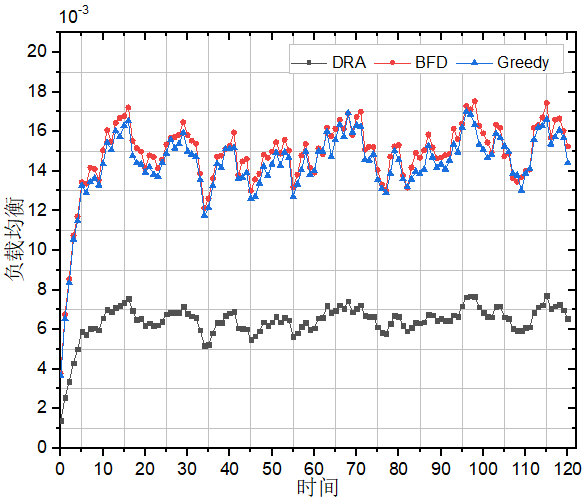
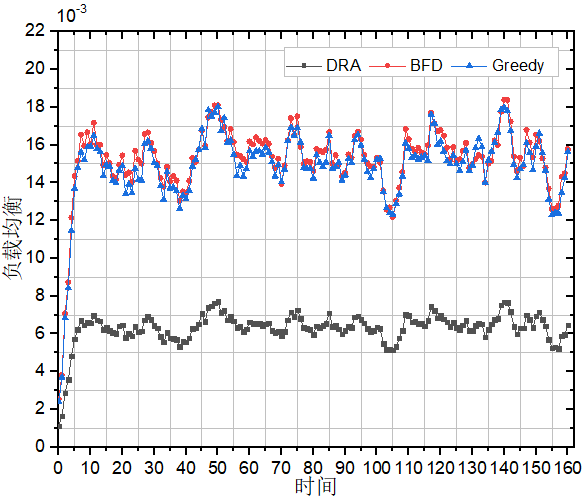
（6）负载均衡性能分析

图5-7中的六个子图显示了DRA、BFD和Greedy作用于六个不同规模数据集时，在不同时刻的负载平衡的比较情况。从这些图中，可以很容易地发现， DRA在负载平衡方面比BFD和Greedy更有效。

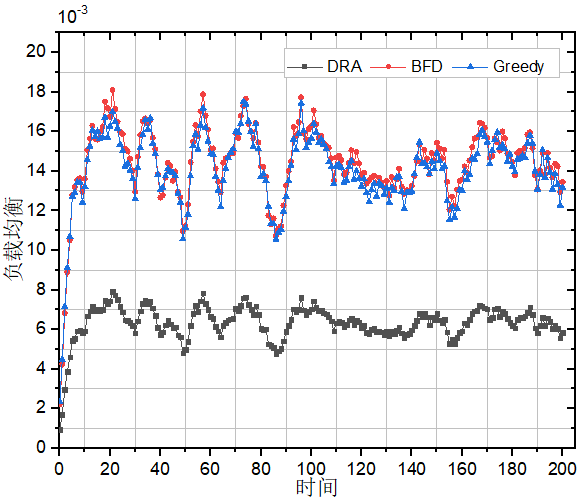
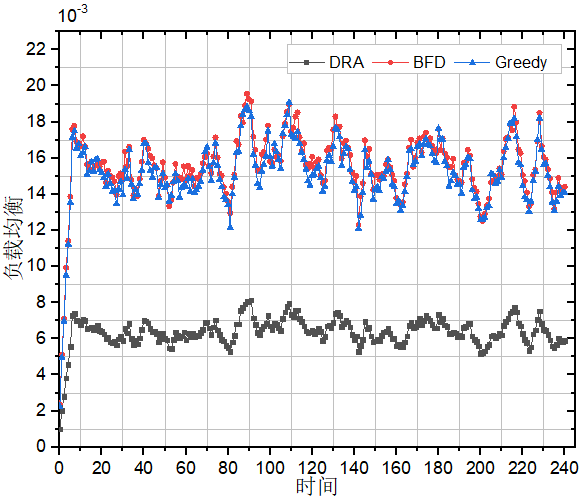
例如，如图5-7(a)所示，DRA在不同的时刻比BFD和Greedy取得了更好的负载均衡。当时间点为40时，DRA的负载均衡值为5.904，而BFD和Greedy分别为13.379和13.058，这意味着DRA在负载均衡方面表现比BFD和Greedy更好。

(a) 元服务数量为1000 (b) 元服务数量为2000

(c) 元服务数量为3000 (d) 元服务数量为4000

(e) 元服务数量为5000 (f) 元服务数量为6000

图5-7 DRA、BFD和Greedy在负载均衡方面的比较

与整体资源利用率类似，也可以计算出整个过程的平均负载均衡。图5-8显示了DRA、BFD和Greedy这三个方法在整体负载均衡方面的比较。从图中可以发现，DRA在这三种方法中，对六个不同规模的数据集都取得了最小的负载均衡值，这意味着它达到了最好的负载均衡。

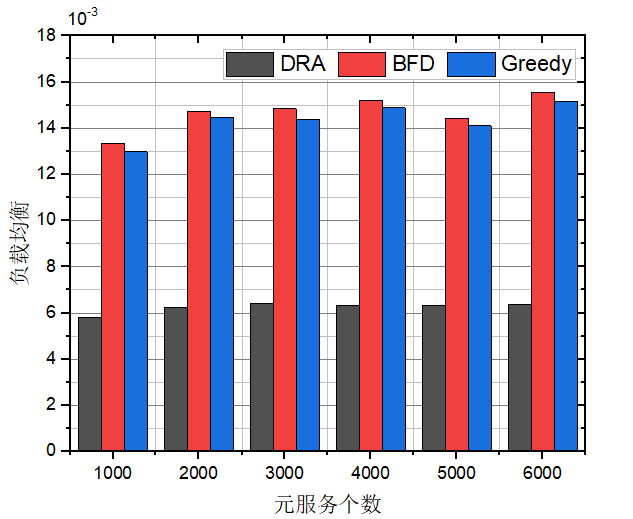


图5-8 DRA、BFD和Greedy在整体负载均衡方面的比较

通过对资源利用率和负载均衡的具体评估，我们可以得出结论：DRA在保证良好资源利用率的同时，比BFD和Greedy更好地实现了云计算平台上负载均衡的资源分配。为了显示更直观的比较结果，表5-6列出了DRA和BFD与Greedy相比的负载均衡提升的百分比。从表中可以明显看出，DRA比BFD和Greedy有更好的负载均衡。例如，对于规模2000的元服务数据集，DRA的整体负载均衡比Greedy好56.95%，而BFD比Greedy差24.86%。

表5-6 DRA和BFD与Greedy相比整体负载均衡提高的百分比

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 方法 | 元服务个数 | | | | | |
| 1000 | 2000 | 3000 | 4000 | 5000 | 6000 |
| DRA (%) | 55.25 | 56.95 | 55.32 | 57.58 | 55.01 | 57.90 |
| BFD (%) | -23.12 | -24.86 | -25.81 | -25.40 | -25.22 | -25.84 |

## 本章小结

本章主要对前两章提出的方法从实验角度证明其有效性，分别给出了实验环境和实验结果分析。

5.2节讨论的是面向负载均衡的云计算资源静态分配方法的实验结果。这一节首先给出了实验的运行环境和实验参数设置，然后使用6个不同规模的元服务数据集，将该方法与另外两种对比方法分别作用于这些数据集，计算出资源分配记录。最后对资源分配记录针对资源利用率和负载均衡两个方面进行统计分析。实验结果显示了该方法在资源利用率和负载均衡方面的有效性。

5.3节讨论的是面向负载均衡的云计算资源动态分配方法的实验结果。同样，这一节首先给出了实验的运行环境和实验参数设置，然后使用6个不同规模的数据集，通过运行该方法与两个对比方法，并统计运行过程中的资源利用率和负载均衡。最后对上述数据进行分析。实验结果显示了该方法在保证良好资源利用率的同时能有效提升系统负载均衡。

# 结束语

## 工作总结

近年来，伴随着互联网和云计算的飞速发展，云计算平台拥有着数以万计的服务器和虚拟机。在过去的几年中，越来越多的HPC应用程序在云计算平台开发和运行，并且存在着一定的性能要求，而且，通常这些HPC应用程序通过资源密集型方式启用，因此可以考虑统一的资源分配方法对其进行合理的资源分配来达到良好的资源利用率。同时，本文试图针对负载均衡这一方面进行优化，重点对以下几个方面进行了研究：

1. **面向负载均衡的云计算资源分配框架**。本文针对云计算平台和HPC应用程序提出了一个针对云计算平台的资源分配框架，指出了资源分配方法在其中的位置和作用。然后根据这个资源分配框架对系统进行建模，推导出性能指标的计算方法和目标优化函数。
2. **面向负载均衡的云计算资源静态分配方法**。本文围绕运行大规模HPC应用的云计算平台的静态场景，提出了一种名为SRA的静态资源分配方法。该方法由元服务预处理、云服务资源使用监控、元服务子集静态资源分配和全局静态资源分配方法这四个步骤组成，在计算资源分配方案的过程中对负载均衡进行了优化，目标在于提升资源利用率和负载均衡。
3. **面向负载均衡的云计算资源动态分配方法**。本文围绕运行大规模HPC应用的云计算平台的动态场景，提出了一种名为DRA的动态资源分配方法。该方法由可分配资源表的维护、针对元服务到达的动态资源分配、针对元服务结束的动态资源分配和全局动态资源分配方法这四个步骤组成。该方法在为元服务分配资源时对负载均衡进行优化，在负载均衡发生变化时，对负载均衡动态调整。目标在于保证良好资源利用率的同时提升负载均衡。
4. **实验评估与分析**。本文在最后对第三章和第四章提出的围绕不同场景的两个资源分配方法进行了实验对比分析。对于第三章提出的面向负载均衡的云计算资源静态分配方法，验证了其在提升资源利用率和负载均衡方面的有效性；对于第四章提出的面向负载均衡的云计算资源动态分配方法，验证了其在确保良好资源利用率的同时能有效地提升负载均衡。

## 未来展望

本文在对云计算资源分配方法的研究过程中仍然存在着不足和一些值得进一步研究的问题，具体表现在以下几个方面。

1. 对于本文提出的方法，在进行实验分析时，由于实际资源的有限，只是进行了仿真实验，来证明方法的有效性。因此，未来将尝试对一些真正的云平台部署提出的静态和动态资源分配方法，以验证其实用性。
2. 在本文提出的方法中，只考虑了元服务和云服务之间支持负载均衡的资源分配方法。因此，在未来的工作中，将对更多的性能要求进行研究，例如如何通过成本优化来调节资源配置的公平性和利用率。

# 参考文献

|  |  |
| --- | --- |
| [1] | Breslow A D, Tiwari A, Schulz M, et al. Enabling fair pricing on HPC systems with node sharing[C]. *Conference on High Performance Computing, Networking, Storage and Analysis*, 2013: 1-12. |
| [2] | Tang S, Lee B S, He B. Fair resource allocation for data-intensive computing in the cloud[J]. *Transactions on Services Computing*, 2018: 20-33. |
| [3] | Islam M A, Ren S, Mahmud A H, et al. Online energy budgeting for cost minimization in virtualized data center[J]. *Transactions on Services Computing*, 2016, 9(3): 421-432. |
| [4] | Pantazoglou M, Tzortzakis G, Delis A. Decentralized and energy-efficient workload management in enterprise clouds[J]. *Transactions on cloud computing*, 2016, 4(2): 196-209. |
| [5] | Yang B, Li Z, Chen S, et al. Stackelberg game approach for energy-aware resource allocation in data centers[J]. *Transactions on Parallel and Distributed Systems*, 2016, 27(12): 3646-3658. |
| [6] | Xie Q, Yekkehkhany A, Lu Y. Scheduling with multi-level data locality: Throughput and heavy-traffic optimality[C]. *Conference on Computer Communications*, 2016: 1-9. |
| [7] | Marinescu D C, Paya A, Morrison J P, et al. Distributed hierarchical control versus an economic model for cloud resource management[J]. *Computer Science*, 2015, 21(2): 562-609. |
| [8] | Rahmanian A A, Dastghaibyfard G H, Tahayori H. Penalty-aware and cost-efficient resource management in cloud data centers[J]. *Journal of Communication Systems*, 2016, 15(7): 362-709 |
| [9] | Sampaio A M, Barbosa J G. Energy-Efficient and SLA-Based Resource Management in Cloud Data Centers[M]. *Advances in Computers*, 2016, 100: 103-159. |
| [10] | Nowzari C, Preciado V M, Pappas G J. Optimal resource allocation for control of networked epidemic models[J]. *Transactions on Control of Network Systems*, 2017, 4(2): 159-169. |
| [11] | Rygielski P, Kounev S. Network virtualization for QoS-aware resource management in cloud data centers: A survey[J]. *PIK-Praxis der Informationsverarbeitung und Kommunikation*, 2013, 36(1): 55-64. |
| [12] | Wang Z, Hayat M M, Ghani N, et al. Optimizing cloud-service performance: Efficient resource provisioning via optimal workload allocation[J]. *Transactions on Parallel and Distributed Systems*, 2017, 28(6): 1689-1702. |
| [13] | Lee E K, Viswanathan H, Pompili D. Proactive thermal-aware resource management in virtualized HPC cloud datacenters[J]. *Transactions on Cloud Computing*, 2017, 5(2): 234-248. |
| [14] | Yang S, Wieder P, Yahyapour R, et al. Reliable Virtual Machine Placement and Routing in Clouds[J]. *Transactions on Parallel and Distributed Systems*, 2017, 28(10): 2965-2978. |
| [15] | Jennings B, Stadler R. Resource management in clouds: Survey and research challenges[J]. *Journal of Network and Systems Management*, 2015, 23(3): 567-619. |
| [16] | Zhang H, Jiang H, Li B, et al. A framework for truthful online auctions in cloud computing with heterogeneous user demands[J]. *Transactions on Computers*, 2016, 65(3): 805-818. |
| [17] | Yoon J, Arslan M Y, Sundaresan K, et al. Self-organizing resource management framework in OFDMA femtocells[J]. *Transactions on Mobile Computing*, 2015, 14(4): 843-857. |
| [18] | Tuncer D, Charalambides M, Clayman S, et al. Adaptive resource management and control in software defined networks[J]. *Transactions on Network and Service Management*, 2015, 12(1): 18-33. |
| [19] | Kusic D, Kephart J O, Hanson J E, et al. Power and performance management of virtualized computing environments via lookahead control[J]. *Cluster computing*, 2009, 12(1): 1-15. |
| [20] | Dutreilh X, Moreau A, Malenfant J, et al. From data center resource allocation to control theory and back[C]. *Conference on Cloud Computing*, 2010: 410-417. |
| [21] | Tang C, Steinder M, Spreitzer M, et al. A scalable application placement controller for enterprise data centers[C]. *Conference on World Wide Web*, 2007: 331-340. |
| [22] | Tsompanidis I, Zahran A, Sreenan C. Towards utility-based resource management in heterogeneous wireless networks[C]. *Workshop on Mobility in the evolving internet architecture*, 2012: 23-28. |
| [23] | Stokely M, Winget J, Keyes E, et al. Using a market economy to provision compute resources across planet-wide clusters[C]. *Symposium on Parallel & Distributed Processing*, 2009: 1-8. |
| [24] | Gong Y, He B, Zhou A C. Monetary cost optimizations for mpi-based hpc applications on amazon clouds: Checkpoints and replicated execution[C]. *Conference for High Performance Computing, Networking, Storage and Analysis*, 2015: 1-12. |
| [25] | Zhai Y, Liu M, Zhai J, et al. Cloud versus in-house cluster: evaluating Amazon cluster compute instances for running MPI applications[C]. *State of the Practice Reports*, 2011: 1-10. |
| [26] | Calheiros R N, Ranjan R, Beloglazov A, et al. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms[J]. *Software: Practice and experience*, 2011, 41(1): 23-50. |
| [27] | Dabbagh M, Hamdaoui B, Guizani M, et al. Energy-efficient cloud resource management[C]. *Computer Communications Workshops*, 2014: 386-391. |
| [28] | Cao Y, Jiang T, He M, et al. Device-to-device communications for energy management: A smart grid case[J]. *Journal on Selected Areas in Communications*, 2016, 34(1): 190-201. |
| [29] | Homsi S, Liu S, Chaparro-Baquero G A, et al. Workload consolidation for cloud data centers with guaranteed QoS using request reneging[J]. *Transactions on Parallel and Distributed Systems*, 2017, 28(7): 2103-2116. |
| [30] | Goudarzi H, Pedram M. Hierarchical SLA-driven resource management for peak power-aware and energy-efficient operation of a cloud datacenter[J]. *Transactions on Cloud Computing*, 2016, 4(2): 222-236. |
| [31] | Alsarhan A, Itradat A, Al-Dubai A Y, et al. Adaptive Resource Allocation and Provisioning in Multi-Service Cloud Environments[J]. *Transactions on Parallel and Distributed Systems*, 2018, 29(1): 31-42. |
| [32] | Batista B G, Ferreira C H G, Segura D C M, et al. A QoS-driven approach for cloud computing addressing attributes of performance and security[J]. *Future Generation Computer Systems*, 2017, 68: 260-274. |
| [33] | Tolosana-Calasanz R, Bañares J Á, Pham C, et al. Resource management for bursty streams on multi-tenancy cloud environments[J]. *Future Generation Computer Systems*, 2016, 55: 444-459. |
| [34] | Poullie P, Bocek T, Stiller B. A Survey of the State-of-the-Art in Fair Multi-Resource Allocations for Data Centers[J]. *Transactions on Network and Service Management*, 2018, 15(1): 169-183. |
| [35] | Ghodsi A, Zaharia M, Hindman B, et al. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types[C]. *Nsdi*, 2011: 24-24. |
| [36] | Joe-Wong C, Sen S, Lan T, et al. Multiresource allocation: Fairness-efficiency tradeoffs in a unifying framework[J]. *Transactions on Networking*, 2013, 21(6): 1785-1798. |
| [37] | Parkes D C, Procaccia A D, Shah N. Beyond dominant resource fairness: Extensions, limitations, and indivisibilities[J]. *Transactions on Economics and Computation*, 2012, 3(1): 808-825. |
| [38] | Wang W, Liang B, Li B. Multi-resource fair allocation in heterogeneous cloud computing systems[J]. *Transactions on Parallel and Distributed Systems*, 2015, 26(10): 2822-2835. |
| [39] | "Apache yarn," [Online]. Available: https://hadoop.apache.org/docs/current2/index.html. |
| [40] | Isard M, Prabhakaran V, Currey J, et al. Quincy: fair scheduling for distributed computing clusters[C]. *Symposium on Operating systems principles*, 2009: 261-276. |
| [41] | Tang S, Niu Z, He B, et al. Long-Term Multi-Resource Fairness for Pay-as-you Use Computing Systems[J]. *Transactions on Parallel and Distributed Systems*, 2018: 1147-1160. |
| [42] | Liu H, He B. F2C: enabling fair and fine-grained resource sharing in multi-tenant IaaS clouds[J]. *Transactions on Parallel and Distributed Systems*, 2016, 27(9): 2589-2602. |
| [43] | Xie D, Ding N, Hu Y C, et al. The only constant is change: incorporating time-varying network reservations in data centers[J]. *SIGCOMM Computer Communication Review*, 2012, 42(4): 199-210. |
| [44] | Guo J, Liu F, Tang H, et al. Falloc: Fair network bandwidth allocation in iaas datacenters via a bargaining game approach[C]. *Conference on Network Protocols*, 2013: 1-10. |
| [45] | Soni G, Kalra M. A novel approach for load balancing in cloud data center[C]. *Advance Computing Conference*, 2014: 807-812. |
| [46] | Ren H, Lan Y, Yin C. The load balancing algorithm in cloud computing environment[C]. *Conference on Computer Science and Network Technology*, 2012: 925-928. |
| [47] | Tsakalozos K, Verroios V, Roussopoulos M, et al. Live VM Migration Under Time-Constraints in Share-Nothing IaaS-Clouds[J]. *Transactions on Parallel and Distributed Systems*, 2017, 28(8): 2285-2298. |
| [48] | Liu L, Qiu Z, Dong J. A load balancing algorithm for virtual machines scheduling in cloud computing[C]. *Conference on Modelling, Identification and Control*, 2017: 471-475. |
| [49] | Kothari N B, Mahalkari A. Uncertain cloud resource provisioning using the predictive approach[C]. *Conference on Information, Communication, Instrumentation and Control*, 2017: 1-7. |
| [50] | Sundararajan P K, Feller E, Forgeat J, et al. A constrained genetic algorithm for rebalancing of services in cloud data centers[C]. *Conference on Cloud Computing*, 2015: 653-660. |
| [51] | Sahu Y, Pateriya R K, Gupta R K. Cloud server optimization with load balancing and green computing techniques using dynamic compare and balance algorithm[C]. *Conference on Computational Intelligence and Communication Networks*, 2013: 527-531. |
| [52] | Dasgupta K, Mandal B, Dutta P, et al. A genetic algorithm (ga) based load balancing strategy for cloud computing[J]. *Procedia Technology*, 2013, 10: 340-347. |
| [53] | Sajay K R, Babu S S. A study of cloud computing environments for High Performance applications[C]. *Conference on Data Mining and Advanced Computing*, 2016: 353-359. |
| [54] | Gupta A, Sarood O, Kale L V, et al. Improving hpc application performance in cloud through dynamic load balancing[C]. *Symposium on Cluster, Cloud and Grid Computing*, 2013: 402-409. |
| [55] | Xenopoulos P, Daniel J, Matheson M, et al. Big data analytics on HPC architectures: Performance and cost[C]. *Conference on Big Data*, 2016: 2286-2295. |
| [56] | Dutot P F, Georgiou Y, Glesser D, et al. Towards Energy Budget Control in HPC[C]. *Symposium on Cluster, Cloud and Grid Computing*, 2017: 381-390. |
| [57] | Alvarruiz F, de Alfonso C, Caballer M, et al. An energy manager for high performance computer clusters[C]. *Symposium on Parallel and Distributed Processing with Applications*, 2012: 231-238. |
| [58] | Cao T, He Y, Kondo M. Demand-aware power management for power-constrained hpc systems[C]. *Symposium on Cluster, Cloud and Grid Computing*, 2016: 21-31. |
| [59] | Kaplan F, Tuncer O, Leung V J, et al. Unveiling the Interplay Between Global Link Arrangements and Network Management Algorithms on Dragonfly Networks[C]. *Symposium on Cluster, Cloud and Grid Computing*, 2017: 325-334. |
| [60] | Gropp W, Lusk E, Skjellum A. Using MPI: portable parallel programming with the message-passing interface[M]. *MIT press*, 1999, 1: 13-22. |
| [61] | Govindarajan K, Kumar V S, Somasundaram T S. A distributed cloud resource management framework for High-Performance Computing (HPC) applications[C]. *Conference on Advanced Computing*, 2017: 1-6. |
| [62] | Goudarzi H, Pedram M. Multi-dimensional SLA-based resource allocation for multi-tier cloud computing systems[C]. *Conference on Cloud Computing*, 2011: 324-331. |
| [63] | Mustafa S, Bilal K, Madani S A, et al. Performance evaluation of energy-aware best fit decreasing algorithms for cloud environments[C]. *Conference on Data Science and Data Intensive Systems*, 2015: 464-469. |

**致 谢**

2018年的5月，现在的南京正值春末夏初，坐在南京大学计算机科学与技术系系楼428往窗外看去，绿油油的一片爬山虎已然爬上了北大楼，可谓是一片生机，生意盎然。而此时，我的研究生生活即将结束。回首这三年时光，美好而又短暂，充实而又漫长，往事仍历历在目。从三年前本科结束有幸认识了窦老师并坚定地选择了跟窦老师读研，到组会上聆听窦老师的教诲，和组里师兄师姐讨论学术问题，以及在课堂上认真学习听讲，这三年一路走来真的收获很多，有许多许多的人要感谢。在硕士即将毕业之际，我谨向各位表示最真挚的谢意。

衷心感谢我的恩师窦万春教授。在读研这三年里，窦老师认真严谨的学术态度、求真务实的工作作风、渊博的学术知识和的诚恳的待人之道都深深地影响着我。依旧记得三年以来的每一次组会，窦老师都严谨全面地为我们指点迷津，帮我们解答疑难、拓宽研究思路，并鼓励我们积极创新、不断探索。在窦老师的谆谆教诲和孜孜不倦的指导下，我的学术思想和专业知识不断增多，分析问题的能力也逐渐增强。千言万语都表达不了我的感激之情，但我还是想向老师您说一句：窦老师，谢谢您！

感谢课题组内的刘孟、许小龙、孟顺梅、吴涛桃、吴小同、贾国超、吴诗颖、刘岩、杨君、李曙、马全、项昊龙、刘子凡、王文平、汤闻达、吴潮兵、赵烜、沈永康、沈炜、牛祥波、甘磊磊、费凡、刘佳和聂红丽等同学，感谢各位在我读研期间给我的无私帮助和各种关心指导。希望以后大家还能继续互相帮助，共同进步。在这里，祝大家身体健康、前程似锦。

感谢我的室友刘天驰和孟占帅同学。这三年的一起生活，大家培养了深厚的感情。由于性格相投，大家没有任何矛盾，平时学习和生活上面也都能互相帮助。忘不了无数个夜晚一起聊天，一起讨论学术问题。在这里，祝你们学业进步、事业有成。

感谢我的家人。读研的三年，感谢我的爸妈在物质上对我的大力支持，在学习和生活上对我无微不至的关怀和无条件的包容。在我遇到挫折时，对我耐心的开导；在我遇到疑惑时，给我细心的解答。这些都鼓励着我，让我更有动力地不断学习，不断进步。

感谢我的女友鲁梦和我自己。读研的三年，也是我和鲁梦从最初的认识到感情不断加深的过程。感谢三年以来你的陪伴与支持，未来让我们一起努力，创造更加美好的生活。同时，我也要感谢自已一直那么认真踏实、努力上进，不畏艰难地把目标都一一实现。研究生生活即将结束，这不是真正的结束，而是一个崭新的开始。未来的自己将会更有信心面对和迎接新的挑战。

再次感谢南京大学，感谢老师、同学和家人的帮助与支持。

**附 录**

**附录A 攻读硕士学位期间参与的论文**

1. Dou W, Xu X, **Liu X**, et al. A Resource Co-Allocation method for load-balance scheduling over big data platforms[J]. *Future Generation Computer Systems*, 2017. (CCF-C类期刊)
2. Tang W, **Liu X**, Rafique W, et al. A Dynamic Resource Allocation Method for Load-Balance Scheduling over Big Data Platforms[C]. *Conference on Cyber, Physical, and Social Computing*, 2018. (投稿中)

**附录B 攻读硕士学位期间申请的专利**

1. 窦睿涵，**刘祥**，苏丰，一种移动互联网环境下基于位置共享的游客管理系统，专利申请号：201610112229.3
2. 窦万春，刘心雨，**刘祥**，支持移动终端的时间空间关联的个性化定制图片管理方法，专利申请号：201710383223.4

**附录C 攻读硕士学位期间获奖情况列表**

1. 南京大学光华奖学金二等奖，2016.12
2. 全国大学生“挑战杯”南京大学校决赛二等奖，2017.05
3. 南京大学优秀研究生，2018.01
4. 南京大学优秀毕业生，2018.05

**附录D 攻读硕士学位期间参加的科研项目**

1. 江苏省重点研发计划项目：基于大数据分析的现代教育云服务平台及关键技术研发
2. 国家自然科学基金面上项目：大数据驱动的复杂系统协同创新理论与方法研究