密级：　　保密期限：

北京郵電大學

博士学位论文

题目：混合云环境下资源调度与管理若干问题研究

学　　号：2012010135

姓　　名：姜海鸥

专　　业：计算机科学与技术

导　　师：宋俊德

学　　院：计算机学院

2017 年 6 月 20 日

独创性（或创新性）声明

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

申请学位论文与资料若有不实之处，本人承担一切相关责任。

本人签名： 日期：2017.6.20

关于论文使用授权的说明

本人完全了解并同意北京邮电大学有关保留、使用学位论文的规定，即：北京邮电大学拥有以下关于学位论文的无偿使用权，具体包括：学校有权保留并向国家有关部门或机构送交学位论文，有权允许学位论文被查阅和借阅；学校可以公布学位论文的全部或部分内容，有权允许采用影印、缩印或其它复制手段保存、汇编学位论文，将学位论文的全部或部分内容编入有关数据库进行检索。（保密的学位论文在解密后遵守此规定）

本人签名：  日期：2017.6.20

导师签名：  日期：2017.6.20

混合云环境下资源调度与管理若干问题研究

摘　要

随着云计算技术的不断发展，越来越多的国内外公司推出各种云产品和云解决方案，越来越多的企业使用云计算技术部署系统，企业的云环境日益复杂。根据云资源提供者和使用者的关系可以分为私有云、公有云、混合云，用户可以选择使用多个云解决方案和公有云服务。部署在云上的应用内部任务的相互关系，可能相互独立，也可能是相互依赖，任务之间有数据传输的相互依赖关系的应用，也称为科学工作流。

本文关注的混合云环境包括三个方面：运行在云平台上的异构QoS（Quality of Service，服务质量）和 SLA（Service Level Agreement，服务等级协议）需求的任务或应用的混合；单个云内部资源性能异构、动态变化、节点失效等不确定性因素的混合；私有云和公有云等不同云平台的混合。分别针对相互独立的任务在单个私有云数据中心、单个科学工作流在单个或多个私有云平台、多个科学工作流在混合云平台、企业系统在多个混合云平台四种情况下，研究调度与管理问题。

1）提出了基于混合预测的异构任务调度和管理方法 MPHW（Multi-Prediction based scheduling for Heterogeneous Workloads），解决异构 QoS 和 SLA 需求混合的独立任务在单个私有云数据中心的调度与管理问题，保障高优先级任务的性能，同时提高了有效资源利用率。建立了异构任务的优先级模型，设计了离线训练的 ARMA（Auto-Regressive and Moving Average，自回归移动平均）模型和在线反馈的 AR（Auto-Regressive，自回归）模型混合的负载预测算法，并提出了基于预测的动态资源预留和任务抢占的调度策略。仿真实验结果表明，相对于基于即时可用资源量的抢占式调度算法，使用基于混合预测和动态资源预留的异构任务调度算法，能够减少 70%主机过负载和任务失败，减少超过 50%的由失败任务造成的资源浪费，提高有效资源利用率超过 65%。此外，对于低 QoS、SLA 任务来说，时间延迟也是可以接受的。

2）提出了一种科学工作流在不可靠私有云上的动态调度算法 DEFT（Dynamic Earliest-Finish-Time），减小了工作流完工时间，提高了调度的健壮性。DEFT 根据运行期资源的计算和传输速度，进行实

时单次调度，应对资源性能异构、动态变化、存在失效等不确定性因素。定义了新的调度健壮性衡量指标 CV（Coefficient of Variations，变异系数），更准确衡量调度算法产生短的、平稳分布的完工时间的能力。实验表明，使用 DEFT 比现有的最优的基于任务列表的静态调度算法 PEFT 和动态调度算法 DCP- G，工作流完工时间缩短超过 20%，且健壮性更好。

3）提出了一种数据密集型科学工作流在混合云上的动态调度算法 HCOD（Hybrid Cloud Optimized Data），减小了云平台之间数据传输量，同时，降低了费用开销。设计了快速的双层禁忌搜索图划分算法 DLTS（Double-Level Tabu Search），划分大型数据密集型科学工作流。将子工作流作为调度的最小单位，减小了调度在不同云平台的任务之间的数据传输量。仿真实验表明，本算法在降低费用开销、减小数据传输量等方面都有很好的表现。

4）提出了一种规范化的系统混合云迁移的策略制定方法，引导企业逐步理顺目标约束、分析决策、制定云迁移方案。提出了循环演进的企业系统混合云迁移策略制定步骤，确保了迁移策略的可实现性。提出了层次化分析方法，为互斥的、不可量化的目标、约束、云服务提供了统一的分析比较方法。使用网络服务运营支撑系统作为用例，分析其云化需求和约束，制定云迁移方案，确定系统各个应用和功能模块应该使用何种云部署和迁移方式。本方法作为指导思想，应用在国内某电信运营商运营支撑系统云化项目的需求分析和设计阶段。

关键词：云计算、异构任务、科学工作流、动态调度、混合云、混合云迁移

RESEARCH ON SOME KEY TECHNOLOGIES OF RESOURCE SCHEDULING AND MANAGEMENT IN THE HYBRID CLOUD ENVIRONMENT

ABSTRACT

With the development of cloud computing, more and more companies around the world has developed public as well as private cloud products. As enterprises employing the private and public cloud products for their systems, the environment of the enterprise system is increasingly complex. Cloud can be classified as private cloud, public cloud, and hybrid cloud according to the relationship between the cloud provider and the user. Tasks of applications running on the cloud can be independent and interdependent. Scientific workflows are those with data transferred between tasks.

The hybrid cloud environment we focus on in this paper consists three aspects. The first is the hybrid tasks with heterogeneous QoS (Quality of Service) and SLA (Service Level Agreement) requirements. The second is the hybrid uncertainties of resources in the cloud, including heterogeneous performances, dynamic changes, and failures. And the last is the hybrid clouds including private clouds and public clouds. We focus on the scheduling and management of independent tasks on a private cloud, single scientific workflow on the private cloud or several private clouds, constant incoming scientific workflows on hybrid clouds, and enterprise system on multi clouds.

The major contributions of this paper are as following:

1) We propose a scheduler MPHW (Multi-Prediction based scheduling for Heterogeneous Workloads) to schedule heterogeneous tasks with heterogeneous QoS and SLA requirements in the private cloud data center. The scheduler guarantees the performance of the tasks with high QoS and SLA requirements as well as increase the resource utilization of the cloud

data center. We propose a priority model for the heterogeneous tasks, hybrid prediction models including the offline-trained ARMA (Auto-Regressive and Moving Average) model and the feedback based online AR (Auto-Regressive) model, and a scheduling strategy based on the prediction based dynamic resource reservation and task eviction. Evaluations show that the scheduler can reduce the host overload and task failures by nearly 70%, reduce average resource waste caused by task failures by more than 50%, increase effective resource utilization by more than 65%. The delay performance degradation is also acceptable for tasks with low QoS and SLA requirements.

2) We propose a dynamic scientific workflow scheduling algorithm called DEFT (Dynamic Earliest Finish-Time) in the unreliable cloud, improving both makespan and robustness. To resolve the hybrid uncertainties like resource heterogeneity, performance variations and unknown failures, DEFT schedules tasks according to the computing rate and resource availability in the runtime. CV (Coefficient of Variation) is proposed to measure how DEFT can schedule workflows with more stable and shorter makespans. Experiments show that DEFT can reduce makespans by more than 20 percent, and produce schedules with larger robustness against uncertainties than existing list-based static scheduler PEFT and dynamic scheduler DCP-G.

3) We propose an online data-intensive scientific workflow scheduling algorithm HCOD (Hybrid Cloud Optimized Data) in the hybrid clouds, with the aim of completing the deadline-constrained workflows as many as possible at a low price. We propose a fast Double-Level Tabu Search graph partition algorithm, called DLTS, to partition the large scale DAG workflow into a set of sub-workflows. HCOD uses the sub-workflow as the scheduling unit to reduce data transferred between tasks on different clouds. The evaluation in a realistic setting shows that our scheduling strategy can achieve a promising performance with respect to the cost saving and data movement sizes.

4) We propose a formalized step-to-step decision making methodology for deciding which part of an enterprise system can be migrated to the cloud, helping enterprise to analyze migration objects, constraints, and various

cloud services. We propose a cycling decision making method for hybrid cloud migration of the enterprise system, so that the migration decision is verified. A hierarchical analyzing method is used for the tradeoff and comparison of the mutually exclusive, quantifiable, and unquantifiable criteria. An Internet service operation support system is introduced as a use case. The cloud migration objects and constraints are analyzed, and the migration strategy is made. The methodology is used in the analysis and design phase in the migration project of the operation support system for a famous telecom operator.

KEY WORDS: cloud computing, heterogeneous tasks, scientific workflow, dynamic scheduling, hybrid cloud, hybrid cloud migration

目 录

图表目录

第一章 绪论

1.1 论文研究背景

近年来，云计算[1]以其高可靠性、快速部署和按需扩展的服务理念，获得快速发展。根据云资源提供者和使用者的关系可以分为私有云、公有云、混合云。私有云是企业内部构建和使用的专有云环境，由专用数据中心或服务器集群搭建而成，提供安全可控的本地云服务。公有云是由第三方服务提供商以租赁的方式提供给企业或个人用户使用的服务，公有云可扩展性高、投入的开销少、维护成本低，但可控性低、存在安全隐患。混合云是二者的结合，企业可以将重要数据和应用放在私有云上，把一些非重要任务和数据放在公有云上执行和保存，在享受私有云的安全可控性的同时，利用公有云提高系统的弹性扩展能力[2,3]。

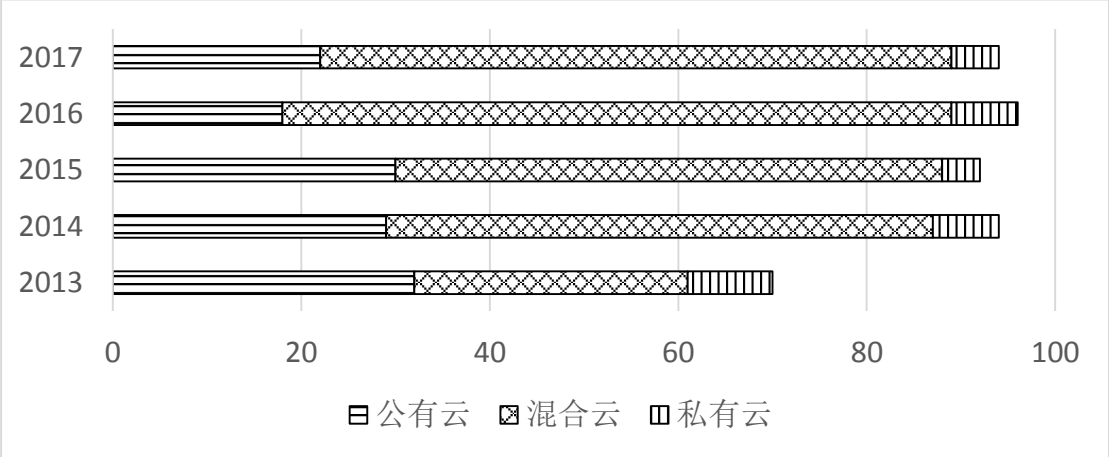


图 1-1 2013-2017 年各类型云使用情况

RightScale 公司 2013-2017 历年对全球 1000 多个企业用户使用云计算情况的调查报告显示[4,5]，云计算的使用从 2013 年的 75%增长到 2014 年的 94%，并在之后几年相对稳定。公有云的增长在 2014 年之后达到了相对饱和，私有云从 2013 年的 38%到 2017 年翻倍，混合云从 2013 年的 29%到 2017 年增长了 1.5 倍。而思科公司 2016 年底的《第六次年度思科云产业调研报告（2015-2020）》预测到：“2020 年云流量将增长 3.7 倍，从 2015 年的每年 3.9 ZB 增长到每年 14.1 ZB。到 2020 年，92%的工作负载将由云数据中心处理，传统数据中心处理的工作负载仅为 8%”[6]。在国内，工信部编制发布的《云计算发展三年行动计划（2017-2019 年）》[7]提出具体发展目标，包括扩大云计算产业规模、突破核心关键技术、提高云服务能力、推进在制造和政务等领域的应用水平提升、优化云数据中心布局建设、增加在全球市场所占份额等，明确推动国内云计算产业在整体规模、核

心技术、应用领域、生态规范、网络安全等方面的提升和突破。

云服务根据提供方式，可以分为 IaaS（Infrastructure as a Service，基础设施即服务），提供计算服务、存储服务、带宽服务等，如 Amazon 弹性云 EC2 和简单存储服务 S3、IBM 的蓝云、国内的阿里云、腾讯云、电信天翼云等。PaaS（Platform as a Service，平台即服务），提供应用开发环境等服务，如 Google App Engine、Microsoft Azure、中国移动的移动云、新浪云和百度云提供的几十种 PaaS 类型应用。SaaS（Software-as-a-Service，软件即服务），将应用或软件封装提供服务，如 Gmail、Google 地球、Google 地图、聊天等 Google App，微软的 Office Live 和 Dynamics CRM 等服务。

随着越来越多的企业系统使用云计算技术进行部署，企业选择使用的云服务和云平台日益多样，部署在云上的应用种类越来越多，云环境趋于复杂。部署在云上的应用种类繁多、需求异构，构成了混合的输入任务；云平台内部资源性能异构、动态变化、存在失效等不确定性因素混合，构成了不可靠的云平台；多种云平台、云产品混合，构成了混合的云平台。在这种多维度混合的云环境下，资源的统一调度和管理，成为云计算关键研究问题之一。

1.1.1 资源调度与管理框架

现有的资源统一调度与管理包括针对单个私有云内部混合应用的管理和对混合云上各云的管理。

1. 单个私有云内部混合应用的调度与管理

为了提高私有云平台的资源利用率，通常在平台内混合运行大量的异构的工作负载，如 CPU 密集型和内存密集型应用，长任务和短任务，批处理任务和低延迟任务等。混合应用的调度与管理架构主要分为三类：分别是：中央式架构，如 Hadoop JobTracker 的实现[8]，双层架构，如 Hadoop YARN[9,10]、Apache Mesos[11,12]，共享状态架构 Omega[13]。

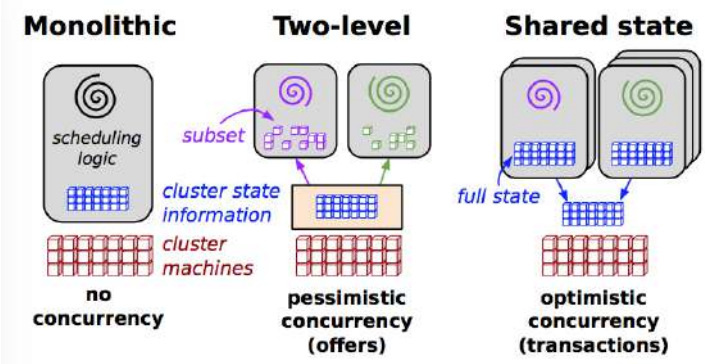


图 1-2 单云内部应用的调度与管理架构[13]

1）中央式架构（Monolithic）

中央式架构为所有任务和应用的调度和管理提供一个单一的、集中的调度管理模块。中央式架构的缺点在于：新的调度策略和实现、新的任务类型很难加入到该架构中，例如之前仅支持 MapReduce 作业，现在要支持流式作业，而将流式作业的调度策略嵌入到中央式调度器中是一项很难的工作[8]；其次，云平台规模受限，难于扩容和升级。

2）双层架构（Two-level）

双层调度和管理采用分而治之策略或者是策略下放机制，将中央式架构中的统一调度管理单元拆分，中央调度和管理功能简化，策略下放到各个应用的节点调度管理器完成。

YARN 包括资源管理器（Resource Manager, RM）、节点管理器（Node Manager, NM）、应用管理器（Application Master, AM）。CPU、内存、磁盘、网络等资源被封装成资源容器，提供给任务使用，并实现资源隔离。RM 是简化的中央管理器，负责整个集群的资源管理和分配；NM 部署在各台主机上，管理每个主机上的资源和任务，负责该节点程序的运行、资源的管理和监控，定时向 RM 汇报本节点资源使用情况和容器运行状态；AM 负责与 RM 协商以获取容器，并将容器内的资源分配给任务，与 NM 通信以启动/停止任务，监控所有任务运行状态，并在任务运行失败时重新为任务申请资源以重启任务[14]。

Mesos 包括主服务器 master、从服务器 slave、管理的异构应用框架 framwork、执行器 executor。Master 是简化的中央管理器，管理接入的各个 framwork 和 slave，决定将任务调度到哪个 slave 所在的容器中，并将 slave 上的资源按照某种策略分配给各个 framework；slave 运行于每个节点，接收并执行来自 master 的命令、管理节点上的任务，并为各个任务分配资源；framework 是指异构的应用框架，如 Hadoop、Dyrad 等，通过注册的方式接入 Mesos，以便 Mesos 进行统一管理和资源分配。Executor 启动 framework 内部的任务，不同的 framework 启动任务的接口或者方式不同，当一个新的框架要接入 Mesos 时，需要编写一个 excuter，说明如何启动该框架中的任务。整个 Mesos 系统采用了双层调度框架：第一层，由 master 将资源分配给 framework；第二层，framework 自己的调度器将资源分配给内部的任务。

双层调度和管理框架提供了一定的灵活性和并行度，但保守的资源可视化和采用悲观锁机制仍有一定的缺点，包括：

①各个应用无法知道整个集群的实时资源使用情况。很多应用不需要知道整个集群的实时资源使用情况就可以运行的很顺畅，但是对于其他一些应用，为之提供实时资源使用情况可以提供潜在的优化空间。例如一个任务或应用失败后，无法通过整个集群状态来判断应该迁移到其他节点还是继续等待，若集群非常繁

忙，迁移可能需要的开销比等待更大，若集群空闲，则更适合任务迁移。

②降低灵活度和并发粒度。如 Mesos 实际上采用一个全局的悲观锁机制控制并发，在任意一个时刻，master 只会将资源推送给某一个 framework，等到该 framework 返回资源使用情况后，才能够将资源推送给其他 framework，这大大限制了系统的并发性。

3）共享状态架构(Shared state)

Omega[13]是基于共享状态的调度器，允许每个应用都可以获得整个集群的实时资源使用信息的共享数据，并通过声明来获得某个资源的使用权，使用“乐观锁”MVCC（Multi-Version Concurrency Control，多版本并发控制）完成共享数据的并发访问控制，大大提升性能。对整个集群中的所有资源分组、限制每类应用程序的资源使用量、限制每个用户的资源使用量等功能全部由各个应用程序调度器自我管理和控制，在共享数据的验证代码中设置优先级管理，当同时有多个应用程序申请同一份资源时，优先级最高的那个应用程序将获得该资源，各个子调度器管理其他资源约束。该机制的问题是，系统性能受资源访问冲突的次数的影响，虽然 Google 通过实际负载测试证明，这种方式的冲突次数是完全可以接受的，但是在后续的框架 Kubernetes 中，还是引入了中央式架构的全局管理策略对共享状态架构进行补充。

4）混合架构 Kubernetes[15]。

Google 基于容器 Docker 的管理系统从 Borg[16]发展到 Omega[13]，再发展到开源的 Kubernetes。Kubernetes 主要包括多个容器的统一管理和操作单元 Pod、定义和访问 Pod 的服务、Pod 副本管理器、用于选择和管理 Pod 的标签。Kubernetes 的管理框架与双层调度管理框架相同，分为 Master 和 Node。Master 节点负责系统调度、对外接口、访问控制、对象的生命周期维护等工作。Node 负责维护容器的生命周期，例如创建、删除、停止 Pod 容器，负责容器的服务抽象和负载均衡等工作[17]。

Kubernetes 提供了混合的调度和管理架构，一方面提供了类似 Omega 的共享式调度管理的弹性和可扩容性，同时加强系统范围内的变量、策略和数据传输，通过中央式的 API 服务器实现对存储的访问控制，隐藏了存储实现的细节，提供对象验证、初始化、版本控制等服务。共享式架构中，节点间相互隔离，可以单独升级或更换，而中央式架构便于管理共同语义、固定参数和策略[18]。

2. 多个私有云和公有云构成的混合云的资源统一管理

用户可以选择使用单一的云解决方案或混合使用多个云服务。RightScale 公司调查报告显示，2016 年用户应用平均运行在 1.5 个公有云及 1.7 个私有云上，同时评估并计划扩展 1.5 个公有云及 1.3 个私有云[4]。混合云管理调用每个云平

台自身提供的管理接口或第三方提供的管理服务，对云资源进行一系列管理操作。不同类型私有云、公有云的服务模型和部署模型具有很大的差异性，造成相同的管理任务在应对不同云时的管理操作往往不同。例如，IaaS 云主要是对虚拟机、存储、带宽等基础设施级别的云资源进行管理，PaaS 云主要是对操作系统、系统运行环境等平台级别的云资源进行管理，SaaS 云则是对应用级别的云服务进行管理。

近年来，市场上出现了很多混合云管理产品，主要包括：只管理企业同构产品组成的同构混合云的产品和管理来自不同企业的异构产品组成的异构混合云的产品。

同构混合云的管理产品主要有：VMware 的 vRealize 管理套件，管理 VMware 私有云和公有云（包括 VMware vCloud Air、AWS、Azure、IBM SoftLayer 等）上的 VMware VPC；微软的 System Center 管理本地和 Windows Azure 上的 Windows Server 集群；Amazon 的 AWS Management Portal for vCenter，以插件的形式安装在 vCenter 环境中，管理 AWS 资源[19] 。

异构混合云的管理产品主要包括：Red Hat CloudForms、Cisco CloudCenter 等。Red Hat CloudForms[20]基于容器环境，提供跨虚拟化、私有云、公有云的全面统一管理，提供部署、集成、资源管理、运营管理、停用和财物报表等完整生命周期管理，提供可视化运营管理，支持全面的业务分析功能和详细日志记录。CloudForms 管理的混合云环境包括：虚拟化平台，如 Red Hat 虚拟化平台、VMware vRealize、Microsoft Hyper-V；基于 Red Hat OpenStack 的私有云平台；公共云平台，如 Amazon Web Services、Microsoft Azure 和 Google Cloud Platform 等。思科 CloudCenter 是一个以应用为中心的混合云管理平台[21]，安全地操作基础架构资源，在数据中心、私有云和公共云环境下部署应用，包括：数据中心，如 Cisco UCS Director、Cisco Application Centric Infrastructure、VMware vCenter、以及其他由软件定义的基础设施管理解决方案；多种架构的私有云，CloudStack、VMware vCloud Director、Microsoft Azure Pack；公有云，包括 AWS 和 AWS GovCloud、Microsoft Azure 和 Azure Government cloud、Google computing platform、Dimension Data platform、IBM SoftLayer、Rackspace platform、VMware vCloud Air 等。CloudCenter 管理混合云的关键模块是对用户透明的协调器，提供针对特定云资源的多租户编排，用于对应用程序进行建模、部署、和管理。协调器部署在各个本地数据中心、私有云、公有云端，使用 REST API 与管理器通讯。在部署阶段，根据管理器发送的信息，包括应用配置、运行策略、应用生命周期信息等，协调初始部署；在运行阶段向上屏蔽底层云环境的差异，将命令翻译成本地云环境下的执行命令，将应用程序的路径对应到本地云环境的路径。

开源的混合云管理框架有 OpenStack 和 Eucalyptus。OpenStack[22]是当今最流行的开源的 IaaS 解决方案, 它可以管理整个数据中心计算、存储和网络资源，由一系列具有 RESTful 接口的服务组件组成, 目前共涵盖了计算、对象存储、认证、用户界面、块存储、网络和镜像服务组件等七个核心组件。OpenStack 社区和不同的厂家提供了工作在不同层面的若干种混合云实现方式，包括：OpenStack 管理已有的 VMware 和 Hyper-V 虚拟化环境、IBM Blue Box 统一运维不同位置的多个 Blue Box 本地云和 Soft Layer 上的 Blue Box 专有云、OpenStack 的 multi-region 技术支持统一用户界面登录多个 OpenStack 云、华为提出了 OpenStack 云级联方案、99cloud 和阿里云提出了类似于 AWS 的混合云方案[19]。Eucalyptus[23]是加利福尼亚大学开发的一个基于 Amazon EC2 的、用于实现云计算的开源软件基础设施，具有简单的组织结构和模块化的设计，每个组件由若干个 Web 服务组成，具有定义良好的 WSDL 接口，且通过使用 WS-Security 策略支持安全通信，因此支持混合多虚拟机集群环境，兼容 vSphereTM、ESXTM、ESXiTM、KVM 和 XEN。当前 Eucalyptus 自助服务管理门户通过统一的界面管理 XEN、KVM 和 VMware 虚拟环境。

学术界对混合云管理也进行了深入的研究，针对不同的云资源实现了特定的管理任务。对混合云的管理平台主要有：欧洲研究项目 SeaClouds，提供异构的 PaaS 平台上应用的分布式管理、监控、和迁移[24]；2015 年美国自然科学基金和欧盟地平线 2020 计划的支持项目 SuperCloud[25]，在现有云平台基础上通过网络层、数据层和计算层的多级抽象，在多个云服务提供者之上构造虚拟用户云[26]；文献[27]将运行时模型（Models at Runtime）引入到云管理过程中，建立多样化云资源的管理方法；网构软件 (Inter-netware)，联接并管理互联网上数量众多的异构、自治的软硬件资源[28]；国防大学开发的 iVCE[29]，以网络资源的按需聚合与自主协同为核心，建立在开放的互联网基础设施之上，通过对中心资源、边缘资源和端资源的按需弹性聚合，为终端用户或应用系统提供服务。

1.1.2 调度与管理的对象

部署在单个或多个云上的应用内部任务的相互关系，可能相互独立，也可能相互依赖[30]，任务之间存在相互依赖关系的应用，包括通常用 DAG（Directed Acyclic Graph, 有向无环图）表示任务之间数据传输的科学工作流，和用 Petri 网表示任务之间逻辑关系的商业应用流程[31]。

1.相互独立任务的调度算法研究现状

相互独立任务的调度算法主要包括基于启发式的调度和基于随机搜索的调度。

启发式调度算法主要包括：随机将任务调度到最近空闲计算节点上；将任务调度到提供最小执行时间的计算节点上；将任务调度到提供最小完成时间

（Minimum Completion Time，MCT）的计算节点上；min-min，将完成时间最短的任务优先调度到提供 MCT 的计算节点上；max-min，将完成时间最长的任务优先调度到提供 MCT 的计算节点上；Xsufferage，将最短完成时间与次短完成时间差值最大的任务优先调度到提供 MCT 的计算节点上。

随机搜索算法包括：遗传算法、模拟退火算法、禁忌搜索算法、遗传算法与模拟退火算法的混合算法、基于 A*树的搜索算法等。

2. 科学工作流调度算法研究现状

科学工作流调度算法包括基于构造式算法和元启发式算法。

基于构造式算法主要包括：基于任务列表的算法、基于任务复制算法、聚类算法。基于任务列表的算法首先将工作流中的任务根据一定的规则进行排序，如 $rank_u$ 值（见本文 3.3.1 节）、关键路径等，在按序将任务调度到能够提供 MCT 的计算节点上。基于任务复制和聚类算法都是特别针对数据密集型工作流，分别使用任务复制、聚类算法，将通信量大的任务打包运行在同一节点/云平台上，以减小跨节点/平台的数据传输量和数据传输时间。

元启发式算法包括：遗传算法、蚁群算法、粒子群算法等。元启发算法虽然结果好，但是通常效率较低，可以先将大的工作流进行分割，再对各个子工作流使用元启发算法，从而降低搜索空间，提高计算速度。

1.1.3 调度与管理的时机

根据任务调度的时机，可以分为静态调度算法和动态调度算法。

静态调度是在任务集合、工作流或工作流集合提交的时候进行调度，云平台的所有资源节点计算、传输速度等性能信息，工作流结构，任务运行时负载信息、通信数据量等信息，都需要提前获得，且假设在运行时期不会发生改变。同时也认为每个任务调度到指定资源节点之后，该节点的空闲时间也可以立刻重新计算并更新[32]。此时，任务运行和数据传输时间都能准确计算出来，根据调度目标获得最佳调度。静态任务可以快速高效的将大型应用调度到大规模云平台，同时，从开发者的角度来说，也易于设计实现。

若上述信息不能事先获得，或者计算资源动态变化，则需要使用**动态调度**。动态调度在任务或工作流运行期进行调度，只在有空闲资源且任务就绪时，根据实际运行期的资源情况进行任务调度。动态算法比静态算法复杂得多，因为需要在运行期实时获取系统的更新信息[33]。动态算法分为在线 online 模式和批处理模式。Online 模式是指每当新任务到达时立刻调度，适用于任务到达率较低的情况；批处理模式适用于任务到达率较高的情况，一次性调度一组任务。

1.1.4 调度与管理的目标和约束

调度与管理的目标和约束决定调度算法，主要包括时间、费用开销、健壮性、可靠性、能耗等[34]，通常云计算环境下的调度约束还可能是多目标约束[35]，例如完工时间和费用的取舍。

时间开销，主要是一组任务或工作流的完工时间，包括硬性的约束条件，如截止时间约束，可伸缩性的优化目标，如最短完工时间。

费用开销，云计算中，尤其是在公有云环境中，费用开销是一个很重要的目标约束，包括计算开销和数据传输开销。硬性约束如费用预算，更常见的是可伸缩性的优化目标，如在截止时间之前完成且费用开销最小。完工时间和费用开销通常是相对立的两个指标，费用开销越低，意味着使用的公有云实例越便宜，计算速度通常比较慢，完工时间越长；反之，完工时间越短，意味着需要使用的公有云速度越快，费用越高，因此，很多研究关注完工时间和费用的平衡[36]。

可靠性，需要确保为任务选择的资源能够成功完成该任务，如果有任务失败，通常需要检测并重启。或者使用任务复制技术，将任务的多个副本调度到不同资源上运行[37]，避免任务失败。这两种方法或者浪费时间、或者浪费资源，但是在不可靠的云平台，如 Amazon EC2 Spot Instance，则需要考虑可靠性，这是一种用时间、空间换可靠性的方法。此外，可以为资源失败率建模，预测任务执行时间内资源的失败率，使任务尽可能调度到不会失败的资源上[38,39]。

健壮性，主要是衡量调度算法的性能，与任务或工作流的截止时间关联，表示一个任务在不超出整个工作流截止时间的情况下，最长可以延迟的时间，如果没有截止时间约束，健壮性通常指在资源计算传输速度动态变化的情况下，同一调度映射对应的不同完工时间的分布的平稳性[40]。

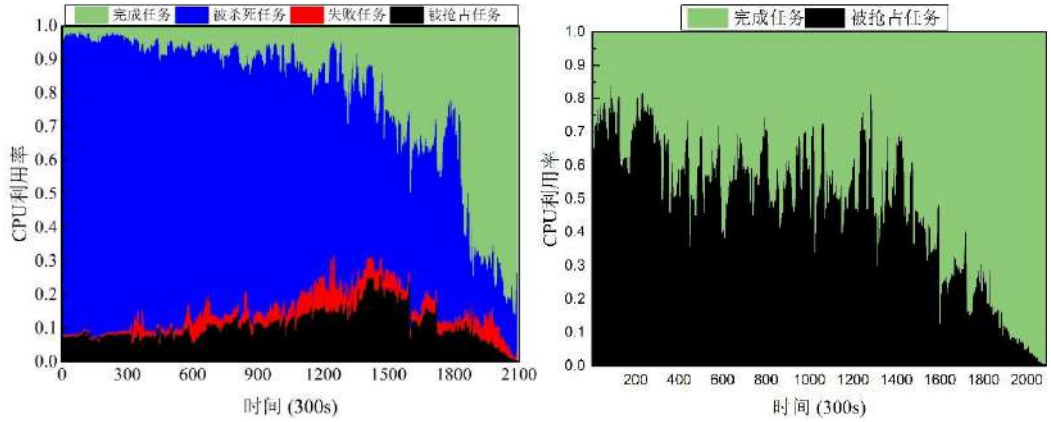
能耗，随着对环境的关注，能耗问题也得到越来越多的关注。能耗主要包括资源的消耗，如用电量，和污染，如二氧化碳排放量。常用的能耗估算方法通常建立能耗与资源利用率之间的线性、多元线性、非线性关系[41]或基于仿真[42]。

1.1.5 混合云调度与管理的问题与挑战

随着云产品的日益丰富，越来越多的企业系统部署到云上，云平台上的应用调度与管理面临一系列挑战：

1）相互独立的任务调度到私有云数据中心上遇到的问题与挑战。云计算和虚拟化的发展，使得生产性任务不再独占服务器，可以与其他低 QoS（Quality of Service，服务质量）和 SLA（Service Level Agreement，服务等级协议）需求的应用整合，构成混合异构任务。这种云数据中心的异构任务具有一定的优先级，不同优先级任务的资源竞争会造成大量任务失败，浪费资源。图 1-3a)是 Google 集

群[43]7 天内所使用的 CPU 中各类型任务所占的比例，每 300s 采样一次。在 Google 集群的说明文档[44]中，“被抢占任务”是指由于高优先级任务的抢占或主机过负载而失败的任务，“被抢占任务”所占的CPU比例最高达28.9%，平均为11.7%。“被杀死任务”是指由于所依赖任务的失败而造成的失败，这意味着“被抢占任务”会造成其他任务的失败，即浪费在最终没能成功完成的任务上的 CPU 高于最高 28.9%和平均 11.7%。部分“被杀死任务”和“失败任务”是由于用户取消操作造成的，如果我们除去原因未知的“被杀死任务”和“失败任务”，如图 1-3b)所示，大多数情况下只有一半的 CPU 使用在成功完成的任务上，另外一半的 CPU 则都浪费在由于被高优先级抢占、主机过负载而杀死、依赖任务失败而失败的任务上。



a)全部类型任务　　b)完成任务与被抢占任务

图 1-3 Google 任务 CPU 利用率

基于资源预留的调度算法为高优先级任务预留资源，保障性能。对于剧烈动态变化的负载，固定的资源预留量需要设定为负载的最高值，在非峰值时刻，会造成资源空闲和浪费，因此，资源预留量应该是可变的。对于通过预测负载来计算资源预留量的研究问题，Google Cluster 的数据分析显示（图 2-1），异构任务的运行负载具有异构的分布特性，现有的基于离线训练的静态预测方法对不平稳负载的预测精度不高，需要研究基于在线反馈的动态负载预测算法；而使用动态算法预测平稳负载则会带来不必要的计算开销；因此，预测模型需要综合考虑预测精度和算法开销两个方面。

2）科学工作流调度到一个或多个私有云平台上遇到的问题和挑战。云平台上包含大量计算、存储、网络等资源，这些硬件设备的更新换代，使得同一类型的硬件资源实际计算能力不同[45]，“同一”类型的资源在“服务”同一个任务时，性能也不同；实际运行期内计算节点的计算速率、网络传输速率实时变化，部署在同一主机上的虚拟机之间的干扰也会影响资源性能[46]；大规模复杂的系统中，虚拟资源发生异常、设备发生故障情况不可避免的时有发生。如图 1-4 所示，为

几周内 Amazon EC2 的 Intel Xeon 和 AMD Opteron 主机上的虚拟机 CPU 计算速度、磁盘顺序读写速度、网络传输速度[47]，每个点代表一个时间点内单个虚拟机的性能参数，可见，云平台上同类型资源呈现明显的性能异构性和动态性。静态调度算法在工作流提交期预估任务完成时间，显然会不准确。由于云平台上资源量大，尽管单个虚拟机的动态变化频率是以小时为单位，从整个云平台来看，资源动态变化频率很高，因此基于重调度机制的算法也不能很好的应对云平台的不可靠性。需要有效的动态调度算法，使得科学工作流调度到混合了资源性能异构、动态变化、有失效可能等不确定性因素的不可靠云平台上，获得最小完工时间。

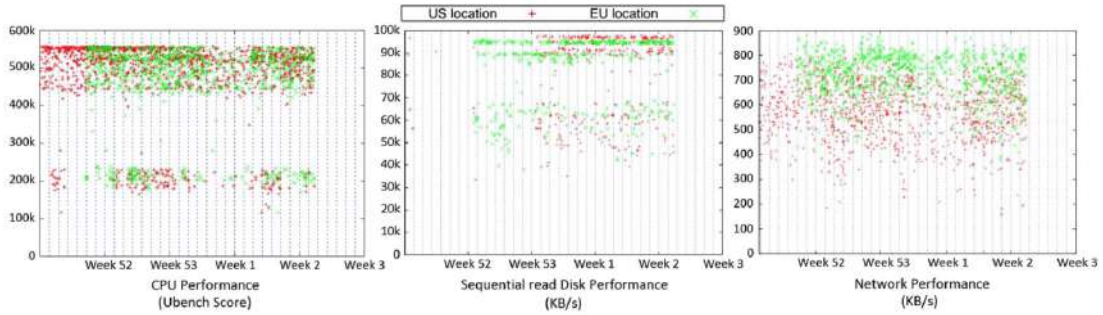


图 1-4 Amazon EC2 云实例 CPU、磁盘、网络性能[47]

3）多个科学工作流调度到混合云上遇到的问题和挑战。混合云能够提供弹性扩展能力，能够很好的应对输入工作流量动态变化的情况，同时，工作流优先调度运行在免费的私有云上，能够节约费用开销。目前市场上有大量多种多样的公有云实例类型，它们的计费模式、服务能力、接入带宽种类繁多，为公有云上的费用开销问题带来挑战[47]。随着大数据应用的发展，数据密集型科学工作流大幅增加，数据通信时间在工作流执行时间占据的比例不可忽视，同时，大量中间数据的传输也提高了网络拥塞的可能性，运行在不同云平台上的同一工作流的不同任务之间的数据传输量，也成为调度需要考虑的重要问题。因此，混合云上的数据密集型科学工作流的调度，是一个多目标优化问题。此外，工作流的持续输入特性，需要动态的调度算法实现，调度开销也是需要考虑的一个重要方面。

4）企业系统迁移到云上遇到的问题和挑战。随着云计算的发展，企业系统云化需求越来越强烈，私有云可以保证核心数据和应用的安全性，但是一次性建设开销较大，公有云可以提高弹性扩展能力，但是安全性可靠性低。如何指导企业分析系统云化目标，分析对比各种云产品性能，制定决策将系统拆分，并选择合适的云产品进行迁移，是需要考虑的问题。企业系统迁移的目标之前通常是互斥的，例如费用开销低和使用高质量的云产品之间互斥。互斥的目标之间需要权衡。同时，企业系统迁移目标、约束、云产品的部分性能是定性的指标，部分是数值指标，需要为定性指标设计量化方法，需要设计定性指标和定量指标的统一分析比较方法。

1.2 论文研究内容

本文分别针对相互独立的任务在单个私有云数据中心、单个科学工作流在单个或多个私有云平台、多个科学工作流在混合云平台、企业系统在混合云平台四种情况下，研究应用的调度与管理问题，研究内容架构如图 1-5 所示。

- 针对单个数据中心上，QoS 和 SLA 需求异构的、多优先级混合的相互独立任务，提出基于混合预测的调度算法，提高有效资源利用率，确保高优先级任务性能；
- 针对云平台混合了资源性能异构、动态变化、存在失效等不确定因素的情况，提出科学工作流的动态调度算法，缩短工作流的完工时间，提高调度健壮性；
- 针对私有云公有云混合的云平台上，持续到达的一组具有截止时间约束的数据密集型科学工作流，提出动态混合调度算法，减小云之间的数据传输量，降低租用公有云实例的费用开销；
- 针对企业系统从传统的本地数据中心迁移到混合云上的问题，提出具体的迁移步骤，在一定的策略和约束下，制定迁移方案。

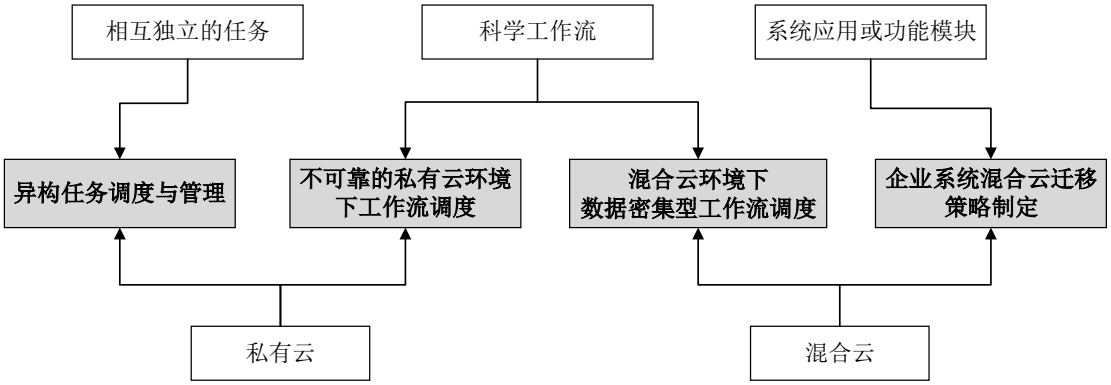


图 1-5 混合云调度与管理研究架构

1.3 论文主要创新点

本文主要的创新点如下：

1）基于混合预测的异构任务调度和管理方法，MPHW(Multi-Prediction based scheduling for Heterogeneous Workloads)。

针对异构 QoS 和 SLA 需求混合的独立任务在单个私有云数据中心的调度与管理问题，提出异构任务调度和管理方法 MPHW，保障高优先级任务的性能，同时提高了私有云数据中心的有效资源利用率。建立了异构任务的优先级模型，

设计了离线训练的 ARMA（Auto-Regressive and Moving Average，自回归移动平均）模型和在线反馈的 AR（Auto-Regressive，自回归）模型混合的负载预测算法，并提出了基于预测的动态资源预留和任务抢占的调度策略。仿真实验结果表明，相对于基于即时可用资源量的抢占式调度算法，使用基于混合预测和动态资源预留的异构任务调度算法，能够减少 70%主机过负载和任务失败，减少超过 50%的由失败任务造成的资源浪费，提高有效资源利用率超过 65%。此外，对于低 QoS、SLA 任务来说，时间延迟也是可以接受的。（第二章，对应学术成果[2][3]和专利成果[2]）

2）一种科学工作流在不可靠的私有云上的动态调度算法，DEFT (Dynamic Earliest-Finish-Time)。

针对用 DAG 表示的科学工作流在混合了资源性能异构、动态变化、存在失效等不确定性因素的不可靠私有云平台上的调度问题，提出动态调度算法 DEFT，减小完工时间，提高调度的健壮性。DEFT 根据运行期资源的计算和传输速度，进行实时单次调度，应对资源性能异构、动态变化、存在失效等不确定性因素。定义了新的调度健壮性衡量指标 CV（Coefficient of Variations，变异系数），更准确衡量调度算法产生短的、平稳分布的完工时间的能力。实验表明，使用 DEFT 比现有的最优的基于任务列表的调度算法 PEFT 和动态调度算法 DCP- G，工作流完工时间缩短超过 20%，且健壮性更好。（第三章，对应学术成果[1]）

3）一种数据密集型科学工作流在混合云上的动态调度算法，HCOD(Hybrid Cloud Optimized Data)。

针对将持续到达的有截止时间约束的数据密集型科学工作流在混合云上的调度问题，提出动态调度算法 HCOD，减小了多个云平台之间数据传输量，同时，降低了费用开销。设计了快速的双层禁忌搜索图划分算法 DLTS（Double-Level Tabu Search），划分大型数据密集型科学工作流。将子工作流作为调度的最小单位，减小了调度在不同云平台的任务之间的数据传输量。仿真实验表明，本算法在降低费用开销、减小数据传输量等方面都有很好的表现。（第四章，对应学术成果[6]）

4）一种规范化的企业系统混合云迁移的策略制定方法。

针对企业将系统安全高效的迁移到多种云平台上的需求，提出了一种规范化的策略制定方法，引导企业逐步理顺目标约束、分析决策、获得云迁移方案。提出了循环演进的企业系统混合云迁移策略制定步骤，确保了迁移策略的可实现性。提出了层次化分析方法，为互斥的、不可量化的目标、约束、云服务提供了统一的分析比较方法。使用网络服务运营支撑系统作为用例，分析其云化需求和约束，制定云迁移方案，确定系统各个应用和功能模块应该使用何种云部署和迁移方式。

本方法作为指导思想，应用在国内某电信运营商运营支撑系统云化项目的需求分析和设计阶段。（第五章，对应学术成果[4][5]和专利成果[1]）

1.4 攻读博士期间主要工作

本人在攻读博士学位期间，主要从事云计算技术的研究工作，以主要研究人员和设计人员身份，先后参与了教育部-中国移动科研基金项目“面向互联网的业务支撑系统关键技术及方案研究”、国家科技支撑计划“劳动者就业信息服务关键技术及服务模式研究：劳动者全生命周期的就业信息服务系统及应用示范”、国家科技支撑计划“公众保险一站式服务体系研究与系统开发”、企事业合作项目“跨域大数据关键技术架构研究”和“基于移动互联网大数据的业务精细洞察方法”等科研和企业项目。

在教育部-中国移动科研基金项目“面向互联网的业务支撑系统关键技术及方案研究”中，关注云计算技术在业务支撑系统中的应用研究。调研了面向互联网的业务支撑系统的云化现状，研究系统云迁移的目标、约束，提出一种进行系统混合云迁移的策略制定方法，指导中国移动业务支撑系统云化项目的分析、设计与实施。相关研究成果为第五章的研究奠定了基础。

同时，调研了面向互联网业务数据中心和私有云的建设情况，针对目前按业务类型分别构建本地或私有云数据中心的情况，提出了利用服务器的剩余资源，整合其他低 QoS、SLA 需求的应用，从而提高数据中心和私有云的资源利用率的方案。针对多种异构 QoS、SLA 需求混合的应用的调度问题，提出基于混合负载预测的调度方法。相关研究成果为第二章的研究奠定了基础。

在国家科技支撑计划“劳动者就业信息服务关键技术及服务模式研究：劳动者全生命周期的就业信息服务系统及应用示范”项目和国家科技支撑计划“公众保险一站式服务体系研究与系统开发”项目中，研究服务系统在私有云上的构建、部署和应用。针对云平台异构、动态、存在资源失效等不可靠因素，提出了动态的科学工作流调度算法，缩短了工作流完工时间，同时提高了调度的健壮性。相关研究成果为第三章的研究奠定了基础。

在企事业合作项目“跨域大数据关键技术架构研究”和“基于移动互联网大数据的业务精细洞察方法”中，针对大数据背景下，大量数据密集型应用在云上的调度需求，提出持续到达的数据密集型科学工作流在混合云上的动态调度算法，使得在工作流截止时间的约束下，费用开销最小的同时，云平台之间的数据传输量最小。相关研究成果为第四章的研究奠定了基础。

1.5 论文的组织结构

全文共分为六章进行阐述。

第一章，介绍课题的研究背景和现状，从云计算的行业发展现状出发，研究并总结了主要的资源调度与管理框架、对象、时机、目标约束，分析了当前混合云环境下调度与管理的挑战和问题，进而明确了本课题的研究内容和主要创新点，并阐述其与攻读博士期间主要工作的关系。

第二章，研究云数据中心混合的异构任务调度问题。首先提出了云数据中心利用生产性服务空闲资源运行低 QoS、SLA 需求的其他应用的模式，介绍了异构任务的分类、优先级、调度策略。然后分析异构负载的时间序列特征，提出针对平稳分布的生产性负载的 ARMA 预测模型和针对不平稳分布的主机负载的基于离线训练的 AR 预测模型。随后，给出了异构任务调度与管理的系统架构和具体算法描述。最后通过实验仿真，分析负载预测算法的精度，分析基于混合预测的异构任务对云数据中心性能的改进，并对比分析不同的预测时间窗口长度对调度性能的影响。

第三章，研究不可靠云平台的科学工作流调度问题。首先介绍了云平台资源性能异构、动态变化、存在失效现象等不确定因素，分析了现有静态调度算法、动态调度算法在不可靠云平台上的不足。然后建立科学工作流在私有云平台上的调度模型，并提出调度的健壮性定义和计算方法。随后介绍动态调度算法 DEFT，分析时间复杂度，并结合一个用例对比分析了 DEFT 和现有经典、高效的静态、动态调度算法 HEFT、PEFT、DCP-G 的完工时间。最后，仿真实验对比 PEFT、DCP-G、DEFT 算法的工作流完工时间和调度健壮性。

第四章，研究混合云环境下数据密集型科学工作流的调度问题。首先介绍了现有混合云环境下任务和工作流调度的主要算法，介绍数据密集型科学工作流调度的主要应用场景和算法。然后建立数据密集型科学工作流在混合云上的调度模型。随后，介绍图划分的概念，提出了 DLTS 算法将大型的 DAG 工作流快速划分成一组子工作流；提出动态混合调度算法，包括 2 种工作流排序策略、2 种子工作流选择策略、子工作流在私有云上的调度算法 EDF-EFT、公有云上的调度算法 HLCF。最后，仿真实验对比分析了 DLTS 算法与已有算法 MITS 和 PATS 工作流划分的性能，并分析了不同私有云规模、工作流数据量、等待队列检索频率的情况下，4 种排序和选择策略的截止时间满足率、公有云运行的任务数、费用开销、数据传输量、调度额外时间开销等性能。

第五章，研究一种规范化的企业系统混合云迁移策略的制定步骤和方法。首先介绍了现有的云迁移策略和决策系统。然后给出循环演进的云迁移策略制定的

步骤，分析应用的迁移目标、可量化的应用性能约束、定性的企业和系统约束、云服务性能，基于 AHP 方法对目标、约束、性能进行分层分析，统一分析定量约束与定性约束，转变成可用数值表示的相关重要度，来求解判断系统各个功能模块的迁移方案。最后，将网络服务运营支撑系统的云迁移作为用例，分析并制定了运营支撑系统的云迁移策略。

第六章，总结了全文，并指出了需要进一步研究的问题。

参考文献

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

[2] 刘鹏.云计算 [M]. 第二版. 北京: 电子工业出版社, 2011: 15-23.

[3] 李学俊,吴洋,刘晓等.混合云中面向数据中心的工作流数据布局方法[J]. 软件学报, 2016,27(7):1861-1875

[4] RightScla2016 云计算使用调查报告及历年报告数据分析 [EB/OL], http://cloud.51cto.com/art/201602/506316.htm, 2017-04-01.

[5] RightScale , State of the Cloud Report [EB/OL], http://www.rightscale.com/?utm_expid=41192858-87.kWDMP6I0Qe-C5Om6akr_Zg.0, 2017-04-01.

[6] Cisco, 第六次年度思科云产业调研报告(2015-2020) [EB/OL], http://www.cisco.com/c/zh_cn/about/press/corporate-news/2016/11-14.html, 2017-04-01.

[7] 工信部. 云计算发展三年行动计划(2017-2019 年)[J]. 电子政务, 2017, 4:93-94

[8] 解析 Google 集群资源管理系统 Omega [EB/OL], http://dongxicheng.org/mapreduce-nextgen/google-omega/, 2017-04-01.

[9] Apache Yarn [EB/OL], https://hadoop.apache.org/docs/r2.4.1/hadoop-yarn/hadoop-yarn-site/index.html, 2017-04-01.

[10] Vavilapalli V K, Murthy A C, Douglas C, et al. Apache Hadoop YARN: yet another resource negotiator [A]. //Proceedings of the 4th annual Symposium on Cloud Computing [C], California: ACM Press, 2013: 1-16.

[11] Apache Mesos [EB/OL], http://mesos.apache.org/, 2017-04-01.

[12] Hindman B, Konwinski A, Zaharia M. Mesos: a platform for fine-grained resource sharing in the data center [A]. //Proceedings of the 8th USENIX conference on

Networked systems design and implementation [C], Boston: USENIX, 2011: 295-308

[13] Schwarzkopf M, Konwinski A. Abd-El-Malek M, et al. Omega: flexible, scalable schedulers for large compute clusters [A]. //Proceedings of the European Conference on Computer Systems [C], Prague: ACM Press, 2013:351-364.

[14] YARN 基本框架介绍 [EB/OL], http://www.cnblogs.com/BYRans/p/5513991.html, 2017-04-01.

[15] Google Kubernetes [EB/OL], http://kubernetes.io/, 2017-04-01.

[16] Verma A, Pedrosa L, Korupolu M., Large-scale cluster management at Google with Borg [A]. //Proceedings of the 10th European Conference on Computer Systems [C], Bordeaux: ACM Press, 2015: 1-17.

[17] Kubernetes 架构设计与核心原理 [EB/OL], http://www.dockerinfo.net/1048.html, 2017-04-01.

[18] Burns B, Grant B, Oppenheimer D,et al. Borg, Omega, and Kubernetes, lessons learned from three containermanagement systems over a decade [J]. Queue Containers, 2016, 14(1): 1-24.

[19] OpenStack 企业私有云的若干需求(4): 混合云支持 (Hybrid Cloud Support) [EB/OL], http://www.cnblogs.com/sammyliu/p/5290855.html, 2017-04-01.

[20] Red Hat, 红帽 CLOUDFORMS 混合环境的统一管理产品规格说明 [EB/OL], https://www.redhat.com/cms/managed-files/cl-red-hat-cloudforms-unified-management-for-hybrid-environments-inc0451306lw-201611-a4-zh.pdf, 2-17-04-01.

[21] Cisco, Cisco CloudCenter Solution: Architecture Overview [EB/OL], http://www.cisco.com/c/dam/en/us/products/collateral/cloud-systems-management/cloudcenter/white-paper-c11-737224.pdf, 2017-04-01.

[22] OpenStack [EB/OL]. https://www.openstack.org, 2017-04-01.

[23] Nurmi D, Wolski R, Grzegorczyk C, et al. The Eucalyptus: open-source cloud-computing system [A]. //Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid [C], Washington DC: ACM Press, 2009: 24-131.

[24] Brogi A, Ibrahim A, Soldani J, et al. SeaClouds: a European project on seamless management of multi-cloud applications [J]. ACM SIGSOFT Software Engineering, 2014, 39 (1): 1-4.

[25] SuperCloud [EB/OL]. https://supercloud-project.eu/, 2017-04-01.

[26] 陈海波, 夏虞斌, 糜泽羽. 跨云计算的机遇、挑战与研究展望 [J]. 中国计算机学会通讯, 2017, 13 (3): 30-35

[27] 陈星,兰兴土,李隘鹏,郭文忠,黄罡.基于运行时模型的混合云管理方法.软件学报,2017,28(7)

[28] 梅宏,刘譞哲.互联网时代的软件技术:现状与趋势[J]. 科学通报, 2010, 13:1214-1220.

[29] Lu X, Wang H, Wang J, et al. Internet-based Virtual Computing Environment: Beyond the data center as a computer [J]. Future Generation Computer Systems, 2013, 29(1): 309-322.

[30] Ruby A. J, Aisha B. W, Shriram. A taxonomy and survey of scheduling algorithms in cloud: based on task dependency [J]. International Journal of Computer Applications, 2013, 82(15): 20-26.

[31] Liu L, Zhang M, Lin Y, et al. A survey on workflow management and scheduling in cloud computing [A]. //Proceedings of the 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing [C], Chicago: IEEE Press, 2014: 837-846.

[32] Arabnejad H, Barbosa J G. List scheduling algorithm for heterogeneous systems by an optimistic cost table [J]. IEEE Transaction of Parallel and Distributing Systems, 2014, 25 (3): 682-694.

[33] Nargunam K L G, Shajin A. Compatibility of hybrid process scheduler in green it cloud computing environment [J]. International Journal of Computer Applications, 2012, 55 (5), 27-33.

[34] Smanchat S, Viriyapant K, Taxonomies of workflow scheduling problem and techniques in the cloud [J]. Future Generation Computer Systems, 2015, 52: 1-12.

[35] 沈尧,秦小麟,鲍芝峰.一种云环境中数据流的高效多目标调度方法[J]. 软件学报, 2017, 28 (3): 1-20.

[36] Grandinetti L, Pisacane O, Sheikhalishahi M, An approximate constraint method for a multi-objective job scheduling in the cloud [J], Future Generation Computing System. 2013, 29: 1901-1908.

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

[38] Kianpisheh S, Charkari M N, Kargahi M. Reliability-driven scheduling of time/cost-constrained grid workflows [J]. Future Generation Computer Systems. 2016, 55: 1-16.

[39] Zhao L, Ren Y, Sakurai K. Reliable workflow scheduling with less resource redundancy [J]. Parallel Computing 2013,39: 567-585

[40] Canon L-C, Jeannot E. Evaluation and optimization of the robustness of DAG schedules in heterogeneous environments [J]. IEEE Transactions on Parallel and Distributed Systems, 2010, 21 (4): 532-546.

[41] 林伟伟, 吴文泰. 面向云计算环境的能耗测量和管理方法 [J]. 软件学报, 2016, 27(4): 1026-1041.

[42] 罗亮,吴文峻,张飞.面向云计算数据中心的能耗建模方法[J].软件学报,2014,25(7):1371−1387.

[43] Google cluster data, http://code.google.com/p/googleclusterdata/wiki/ClusterData2011_1, 2017,4,10.

[44] Reiss C. and Wilkes J., Google cluster usage traces: format + schema (version of 2011.10.27, for trace version 2), https://drive.google.com/open?id=0B5g07T_gRDg9Z0lsSTEtTWtpOW8&authuser=0, 2017,4,10.

[45] Dejun G P J, Chi C H, EC^2: performance analysis for resource provisioning of service-oriented applications [A]. //Proceedings of the 2009 International conference on Service-oriented computing [C], Sweden: Springer Press, 2009: 197-207.

[46] Farley B, Venkatanathan V, Bowers K D, More for your money: exploiting performance heterogeneity in public clouds [A]. //Proceedings of Symposium on Cloud Computing [C], California: ACM Press, 2012: 1-14.

[47] Schad J, Dittrich J, Quiané-Ruiz J-A. Runtime measurements in the cloud: observing, analyzing, and reducing variance [J]. Proceedings of the VLDB Endowment, 2010, 3 (1): 460-471.

[48] Lin B, Guo W, Lin X. Online optimization scheduling for scientific workflows with deadline constraint on hybrid clouds [J]. Journal of Concurrency and Computation: Practice & Experience, 2015, 28 (11): 3079-3095.

第二章 基于混合预测的异构任务调度和管理方法

2.1 引言

在传统数据中心里，企业为了保障生产性应用的动态需求，应对可能的资源需求高峰，为应用提供过量资源，通常每个应用独占一个或多个服务器。虽然每个服务器都不是空闲的，但是上面运行的应用通常只占用了 10-50%的资源[1, 2]，造成了浪费。因此，如何有效利用数据中心的空闲资源，成为企业和机构节省费用开销的重要途径之一[3]。云计算和虚拟化能够将低 QoS 和 SLA 需求的应用整合到服务器上，从而很好的解决资源过度提供的问题。原有的生产性应用和其他的低 QoS、SLA 需求的应用混合，构成了异构的任务。

现有的异构任务调度和管理框架的优先级管理，主要基于抢占策略，在新任务到来时根据现有空闲资源量为其分配资源，在资源不足时，高优先级任务可以抢占低优先级任务后进行调度。这种基于“资源尽力分配-任务抢占”的策略的主要问题有：1）如果将主机空闲资源全部分配，在下一时刻负载增加时，主机过负载，需要杀死任务释放资源，从过负载到释放资源的过程中，重要的生产性应用的性能将受到影响；2）过负载造成的杀死任务、不同优先级之间的任务抢占等各种操作产生的失败任务占用的资源，是一种资源浪费，任务抢占、杀死、重调度、迁移等操作，也会带来额外开销。

因此调度算法需要针对私有云数据中心上的异构任务，避免过多任务被调度到主机上引起主机过负载，减少资源竞争带来的任务抢占，同时减少失败任务量，从而减少资源浪费。

2.2 相关研究工作

2.2.1 异构任务调度与管理方法

任务或应用的异构性包括很多方面，如 QoS 和 SLA 需求不同、任务或应用内部结构不同、任务长度不同、任务负载不同等。表 2-1 根据管理的异构任务的类型不同对现有异构任务调度与管理研究成果进行分类。

现有的开源异构任务调度与管理主要研究异构的批处理架构，如 Apache YARN 可以管理 Spark、Dryad、Giraph、Hoya、REEF、Storm、Tez 等框架[6]；Mesos 可以管理 MapReduce、Dryad、MapReduce Online、Pregel 等框架，或者研究 QoS、SLA 需求不同的任务[7]。Google Omega[8]、Google Kubernetes[9]研究 CPU

密集型和内存密集型应用异构、长任务和短任务异构、批处理和低延迟任务异构等任务异构的情况。

表 2-1 异构任务调度与管理研究分类

可以管理的异构任务	文献
异构的批处理计算框架	Apache YARN[6]，Mesos[7]
QoS、SLA 需求不同的任务	Mesos[7]
CPU 密集型和内存密集型应用； 长任务和短任务； 批处理和低延迟任务；	Google Omega[8] Google Kubernetes[9]
批处理任务和事务型任务	文献[10-13]
生产性任务和尽力提交的数据处理类任务	文献[15]
交互式应用和批处理任务	文献[16]

文献[10,11]研究批处理任务和事务型任务混合的异构负载，在各任务满足公平调度的同时，根据性能模型最大化各任务的性能，根据排队论为事务型任务建立性能模型，根据完成时间估算设计批处理任务的性能模型。文献[12,13]研究批处理任务和事务型任务混合的异构负载，基于准入控制机制和虚拟机重调度机制最大化资源利用率，并且保障不同工作流的 SLA。文献[15]研究生产性任务和数据处理类的尽力提交任务混合，保障生产性任务的 SLA，同时尽量减小尽力提交任务的延迟，将当前和未来资源使用映射成混合整数线性规划 MILP 问题，并提出了伸缩的启发算法来平衡分配生产性任务和尽力提交任务。文献[16]研究交互式应用和批处理应用混合、虚拟机和物理服务器混合的异构数据中心，利用被过度提供的剩余资源运行其他低优先级任务，通过干扰预防系统实时监控任务干扰，在干扰产生时杀死低优先级任务来保障高优先级任务的性能。

2.2.2 异构任务优先级管理方法

现有根据优先级管理异构负载的算法大多为优先级高的负载分配更多的可使用的资源量上限。YARN 使用优先级作为每个应用能够获得多少比例的系统资源量的上限，同优先级的任务之间使用公平调度[4]和容量调度[5]。Mesos 使用 DRF 算法[14]，初始情况下按公平原则为各计算框架预留一定比例的资源，高优先级的应用可以动态获得更多的资源。在资源冲突时，YARN、Mesos、Quasar[17]使用简单的抢占策略，高优先级任务杀死低优先级任务，失败的低优先级任务并不保存运行进度，直接释放资源，并排队等待重调度。Google Omega 和 Kubernetes 在资源申请阶段如果发生资源争用，则优先级高的任务/应用可以获得资源，优先

级低的任务/应用等待。文献[16]基于实时监控技术，杀死低优先级任务来保障高优先级任务的性能。

2.2.3 负载预测算法

目前主机负载预测算法主要用在虚拟机迁移、服务器整合、能耗管理等方面。Quasar[17]根据用户定义的服务约束，基于集群当前性能，使用协同过滤技术估算资源最小使用量，Quasar 预测任务的实际需求量，而非依据资源预留，为任务提供资源。文献[18]使用人工神经网络 ANN 预测未来资源可用量和每个应用的资源需求量。文献[19]使用高斯预测模型，参数通过曲线拟合预测和遗传算法，预测主机负载。文献[20]首先将虚拟机分组，然后使用隐马尔科夫模型分析获得每组虚拟机的 CPU 负载特征。文献[21]基于进化算法，使用相空间重建（Phase Space Reconstruction）和数据处理的组方法（Group Method of Data Handling）结合的方法进行负载预测。文献[22]提出一个比多次的单步预测更加准确的多步预测方法，包括 4 个步骤：为序列找出适配函数、预测序列的变化模式特征、判断变化范围、预测变化模式。文献[23,24]将预测问题变成一个基于贝叶斯模型的分类问题，预测的时间窗口不是定长的，而是随着距离当前时间越远而呈指数分布。随着预测窗口增长，预测精度会降低。文献[25]使用 ARIMA 模型预测主机能耗，文献[26]针对虚拟机整合问题，使用 ARIMA 模型预测虚拟机负载。文献[28]首先将工作流根据特征进行聚类，分成高、中、低三类，然后使用三类工作流对混合的隐马尔科夫模型进行离线训练，最后用该模型预测工作流模式。

2.2.4 现有研究存在的问题

现有的异构任务调度算法和负载预测算法主要有以下几个问题和不足：

（1）对于有优先级的异构任务的管理，通常是在资源分配时根据空闲资源量尽力分配，而在资源发生冲突时，高优先级任务直接抢占低优先级任务，在监控高优先级任务性能下降时，直接杀死低优先级任务。在数据中心负载较高时，资源尽力分配原则会造成过多的任务抢占、杀死、重调度，造成资源浪费；同时，高优先级任务性能也会受到影响。

（2）现有预测算法大多基于离线训练，参数和模式通常是静态的，在预测过程中不变，对于符合平稳分布的序列精度良好，但是对于快速变化的不规则负载，静态的模型性能不够好。

针对第一个问题，本文提出基于动态资源预留的资源分配模型和基于抢占的任务调度策略，通过预测确定新任务的可用资源量上限，此外，在资源短缺时采用抢占方式，高优先级任务抢占低优先级任务，获得资源。

针对第二个问题，本文针对异构负载提出混合预测模型，对于平稳分布的生

产性负载序列，使用静态的离线训练的 ARMA（Auto-Regressive and Moving Average，自回归移动平均）模型，精度符合需求且时间开销较低，对于快速变化的不规则主机负载，使用基于反馈的 AR（Auto-Regressive，自回归）模型，对不平稳分布的负载预测精度高。

2.3 异构任务的优先级模型

在多种类型应用混合的云数据中心中，相互独立的、异构的任务根据重要性、QoS、SLA 需求，可以分成三类优先级，分别对应不同的调度策略：

高优先级的生产性任务：生产性任务是指企业或组织原有的与生产性、盈利性服务相关的应用，这部分负载原先独占服务器，具有最高的优先级。生产性任务通常对延迟敏感，具有最高的 QoS 和 SLA 需求，新到达的生产性任务应该立刻被调度并执行，如果可用资源不足，则抢占其他非生产性任务。由于在原有情况下，企业或组织中的生产性任务独占服务器，且服务器资源提供量远远大于生产任务的实际需求量，因此单个主机全部运行生产性任务且过负载的情况不会出现。生产性任务不能被其他任务抢占，也不能在主机过负载时被杀死。

中等优先级任务：非交互式的批处理任务、科学计算工作流、企业日常事务管理等应用，通常只有一个截止时间约束，对延迟不敏感，在数据中心负载较高时可以等待，在负载较低时再运行。这部分任务具有中等优先级，如果数据中心可用资源足够，可以被调度，否则等待。在中等优先级任务运行过程中，如果生产性任务的资源请求量、使用量增大，可用资源不足，生产性任务性能会受到影响。因此，调度中等优先级任务时，应预留出未来一段时间内生产性负载的增加量。一个主机可以提供给中等优先级任务的可用资源量为当前可用资源量减去未来一段时间内主机上生产性负载的增量。首先预测生产性应用负载，当前可用资源量减去生产性应用负载增量，即为中等优先级任务的可用资源量。

低优先级的免费任务：测试任务、公测应用等免费任务，对延迟不敏感，对成功率要求不高，在数据中心负载高时可以等待，任务失败也无所谓，优先级最低。在调度时，如果可用资源不足，则排队等待。在其他较高优先级任务可用资源不足时，低优先级任务可以被抢占，若主机过负载，可以被杀死，释放资源供其他较高优先级任务使用。在免费任务运行过程中，已调度任务的资源使用量增大、其他高优先级任务的资源请求量增大、主机过负载等情况，都会抢占免费任务，频繁的任务调度和抢占会造成资源浪费。因此，调度低优先级任务时，应预留出未来一段时间内主机已调度的所有负载的增加量，一个主机可用提供给低优先级任务的可用资源量为当前可用资源量减去未来一段时间内主机上负载的增

量。首先预测主机负载，可用负载量为主机资源总量减去预测的主机负载量最大值。

2.4 异构负载的混合预测模型

工作负载可以看作时间的连续函数，将连续的时间分割成离散的时槽(slot)，将每个 slot 内负载最大值设为该 slot 对应的负载，则可以得到主机的离散时间序列$\{X_n\}$。

2.4.1 异构负载的时间序列分布特性

生产性负载、中等优先级负载、免费负载的时间序列具有不同的分布特性，本文分析 Google 集群[29,30]的各类不同优先级任务负载特征，如图 2-1 所示，为一天内 CPU 利用率的时间序列，slot 为 300s。生产性任务优先级大于 8，其他任务优先级小于 9[31,32]，包括中等优先级任务和低等优先级任务。其中，生产性任务的 CPU 利用率通常较为平稳，负载的时间序列符合平稳分布，使用 ARMA 模型[33,34]来预测生产性负载，模型的参数通过离线训练获得。其他较低优先级任务的 CPU 利用率呈现出更高的峰值-均值比，负载的时间序列也显示出不平稳的特性。对于同样不平稳的主机负载的时间序列，使用一阶差分去除负载的不平稳性，然后使用 AR 模型[33,34]进行预测，并利用在线的反馈机制实时计算 AR 模型的参数。

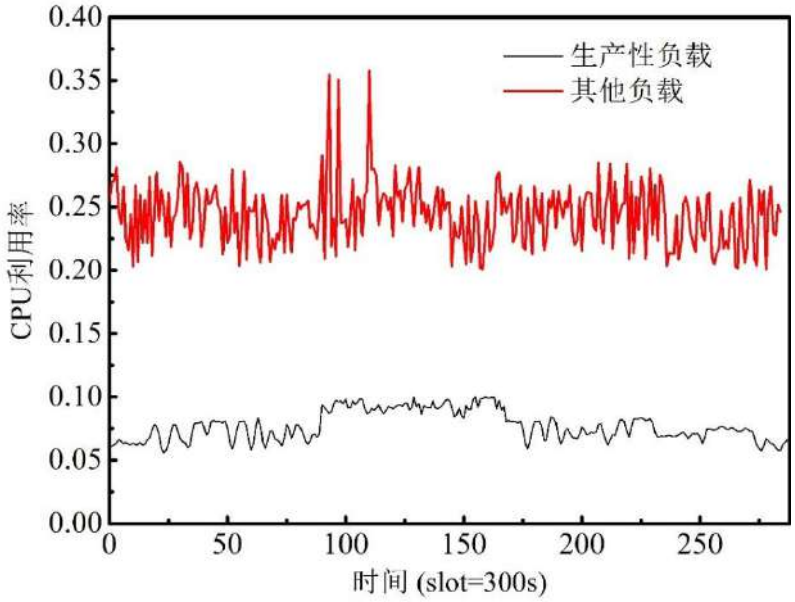


图 2-1 Google 集群异构负载时间序列

2.4.2 基于离线训练的 ARMA 模型

在时间序列分析中，ARMA 模型用于分析平稳分布的时间序列，包含一个自回归模型 AR(p)和移动平均过程(Moving Average)MA(q)。

定义 1. 对于满足平稳分布的负载的时间序列$\{X_n\}$满足 ARMA(p,q)模型，且前 n-1 个负载值$\{X_{n-1}\}$已知，则时刻 n 的负载 X_n 为：

$$X_n-\varphi_1X_{n-1}-\cdots-\varphi_pX_{n-p}=\varepsilon_n-\theta_1\varepsilon_{n-1}-\cdots-\theta_q\varepsilon_{n-q} \tag{2-1}$$

其中，X_t, $t=n\text{-}p,n\text{-}p\text{-}1,\ldots,n$ 是时刻 t 的负载，p 是 AR 过程的阶，q 是 MA 过程的阶，其中 φ_i, $i=1,2,\ldots,p$，θ_j, $j=1,2,\ldots,q$ 是从前 n-1 个负载$\{X_{n-1}\}$计算得到的参数。E_k, $k=1,2,\ldots,q$ 是预测误差，是独立同分布的高斯白噪声序列，即：

$$E(\varepsilon_n)=0,\ \ E(\varepsilon_n\varepsilon_{n+k})=\begin{cases}\sigma_\varepsilon^2, & k=0\\ 0, & k\neq 0\end{cases} \tag{2-2}$$

使用最小信息量准则 AIC (Akaike Information Criterion)和贝叶斯信息准则 BIC (Bayesian Information Criterion)[34]准则，可以找到合适的最小 p 和 q，确定过去有多少个状态对预测值产生影响。AIC 准则和 BIC 准则基于信息论原理，用于判断由极大似然估计得到的模型参数的质量，通常 AIC 与 BIC 值越小，模型参数质量越好。

AIC=-2*ln*(模型中极大似然函数值)+2(模型中未知参数个数) (2-3)

BIC=-2*ln*(模型中极大似然函数值)+*ln*(*n*)(模型中未知参数个数) (2-4)

在选择了合适的 p 和 q 后，ARMA 模型参数 φ_i, $i=1,2,\ldots,p$, θ_j, $j=1,2,\ldots,q$ 可以使用最小二乘法确定。假设通过对$\{X_n\}$的一个样本序列$\{X_{n-1}\}$的识别判定$\{X_n\}$为 ARMA(p,q)序列，令向量 $\bar{\boldsymbol{\varphi}}=(\varphi_1,\cdots,\varphi_p,\theta_1,\cdots,\theta_q)^{\mathrm{T}}$，递推的计算 ε_n 的估计值 $\hat{\varepsilon}_n$ 为：

$$\begin{cases}\hat{\varepsilon}_n=0, n\le p\\ \hat{\varepsilon}_n=x_n-\sum_{i=1}^{p}\varphi_i x_{n-i}+\sum_{i=1}^{q}\theta_i\hat{\varepsilon}_{n-i}, n=p+1,p+2,\cdots,N\end{cases} \tag{2-5}$$

定义 $\hat{\varepsilon}_n$ 的残差平方和 $S(\bar{\boldsymbol{\varphi}})$ 为 $S(\bar{\boldsymbol{\varphi}})=\sum_{n=p+1}^{N}\hat{\varepsilon}_n^2$，使 $S(\bar{\boldsymbol{\varphi}})$ 达到极小的 $\hat{\bar{\boldsymbol{\varphi}}}^L=(\hat{\varphi}_1^L,\cdots,\hat{\varphi}_p^L,\hat{\theta}_1^L,\cdots,\hat{\theta}_q^L)$，为 $S(\bar{\boldsymbol{\varphi}})$ 的最小二乘估计。

2.4.3 基于在线反馈的自回归模型

定义 2. 自回归模型 AR(p)表示时刻 n 的负载 X_n 是前 n-1 个负载的线性组合。

$$X_n=\varphi_1X_{n-1}+\varphi_2X_{n-2}+\cdots+\varphi_pX_{n-p}+\varepsilon_n \tag{2-6}$$

定义 3. 移动平均过程 MA(q)表示时刻 n 的负载 X_n 是前 n-1 个预测误差的线

性组合。

$$X_n = \varepsilon_n - \theta_1\varepsilon_{n-1} - \theta_2\varepsilon_{n-2} - \cdots - \theta_q\varepsilon_{n-q} \tag{2-7}$$

由于主机负载的不规则性，预测误差会很大，即 MA(q)中的 $\varepsilon_k, k=1,2,...,q$ 值较大，如果使用 ARMA 模型预测主机负载，预测误差会加重。此外，固定参数也无法适应连续变化的时间序列模式。因此，仅使用自回归模型预测主机负载，并基于实时的实际工作负载的反馈，动态获得 AR(p)模型的参数。

现有 AR(p)模型参数的预测方法包括：最小二乘法、马尔科夫蒙特卡洛算法、基于 Yule-Walker 方程算法等[33]。本文使用 Yule-Walker 方程估计模型参数，Yule-Walker 方程的基本形式为：

$$\begin{cases} \rho_1 = \varphi_1 + \varphi_2\rho_1 + \cdots + \varphi_p\rho_{p-1} \\ \rho_2 = \varphi_1\rho_1 + \varphi_2 + \cdots + \varphi_p\rho_{p-2} \\ \quad\vdots \\ \rho_p = \varphi_1\rho_{p-1} + \varphi_2\rho_{p-2} + \cdots + \varphi_p \end{cases} \tag{2-8}$$

其中 ρ_k 是自相关函数，定义为：

$$\rho_k = \rho_{-k} = \gamma_k / \gamma_0, k \geq 0 \tag{2-9}$$

γ_k 是自协方差，定义为：

$$\gamma_k = \gamma_{-k} = E[(X_n - \mu)(X_{n-k} - \mu)], k \geq 0 \tag{2-10}$$

特别的

$$\gamma_0 = E[X_nX_n] - E[X_n]E[X_n] = \sigma^2 - \mu^2 \tag{2-11}$$

根据 AIC 准则和 BIC 准则，AR(p)的最佳阶数为 2。根据 Yule-Walker 方程，AR(2)模型参数为：

$$\varphi_1 = \frac{\rho_1(1-\rho_2)}{1-\rho_1^2} = \frac{r_0r_1 - r_1r_2}{r_0^2 - r_1^2}, \varphi_2 = \frac{\rho_2 - \rho_1^2}{1-\rho_1^2} = \frac{r_0r_2 - r_1^2}{r_0^2 - r_1^2} \tag{2-12}$$

主机负载不具有平稳分布的特性，对序列使用一次差分去除不平稳性。负载的一阶差分时间序列{ΔX_n}的自回归模型如下：

$$\Delta X_n = \varphi_1\Delta X_{n-1} + \varphi_2\Delta X_{n-2} + \cdots + \varphi_p\Delta X_{n-p} + \varepsilon_n \tag{2-13}$$

对于 AR(2)模型，我们有：

$$\Delta X_n = \varphi_1\Delta X_{n-1} + \varphi_2\Delta X_{n-2} + \varepsilon_n$$

$$X_n - X_{n-1} = \varphi_1(X_{n-1} - X_{n-2}) + \varphi_2(X_{n-2} - X_{n-3}) + \varepsilon_n$$

$$X_n = (1+\varphi_1)X_{n-1} + (\varphi_2 - \varphi_1)X_{n-2} - \varphi_2X_{n-3} + \varepsilon_n \tag{2-14}$$

每当获得新的工作负载 X_n 时，重新计算参数 φ_1, φ_2，从而使得预测模型不断调整，适应动态变化的工作负载。

2.5 基于混合预测的异构任务调度和管理

2.5.1 基于混合预测的异构任务调度和管理框架

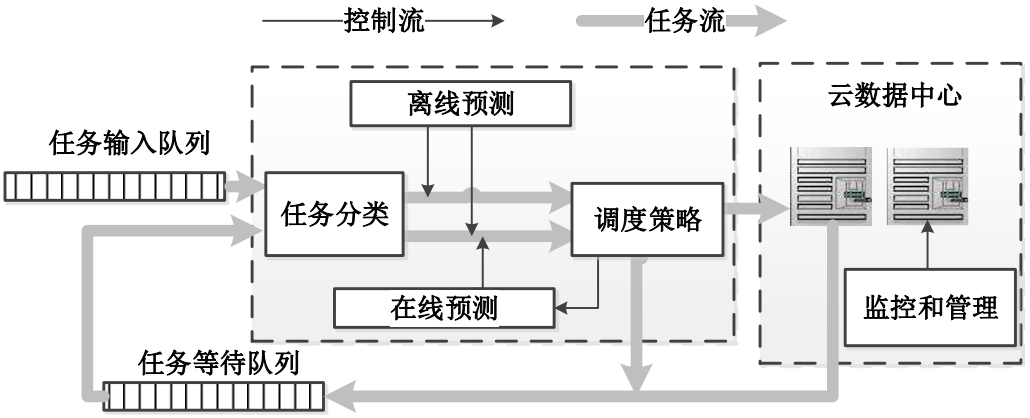


图 2-2 基于混合负载预测的异构任务调度和管理框架

异构任务调度和管理框架如图 2-2 所示，包括如下几部分：

任务分类：识别新任务的类型和优先级，并为该任务分配不同的预测和调度算法。对于中等优先级任务，使用离线的 ARMA 模型预测生产性负载，计算可用资源量，对于低优先级的免费任务，使用基于在线反馈的 AR 模型预测主机负载，计算可用资源量；

离线预测器：使用 ARMA 模型预测平稳的生产性负载，通过离线训练获得静态参数。预测在下一个时间窗内生产性负载的最大资源请求和使用量，然后给出当前可以分配给中等优先级任务的最大资源量建议。

在线预测器：对于低优先级的免费任务，预测在下一时间窗内主机负载的情况，然后给出当前可以分配给低优先级的免费任务的最大资源量建议。在线预测器使用反馈控制的 AR 模型，根据过去信息，动态调整 AR 模型参数，提高 AR 模型的预测精度。

调度决策模块：根据任务类型、优先级和预测器给出的资源分配建议，进行任务调度。如果根据预测没有足够的资源分配给新任务，对于中等优先级任务，随机抢占免费任务获得资源后调度，对于低优先级的免费任务，则将任务放到队列排队等待，避免调度后在未来的时间内再次被高优先级任务抢占。

监控和管理模块：监控和管理模块是一个后台程序，周期性使用 AR 模型预测各主机的总负载，一旦一个主机被预测过负载，则杀死低优先级任务释放资源，

保障高优先级任务的性能。

任务等待队列：包括未被调度、被杀死、被抢占的任务。这些任务根据截止时间排列，截止时间按照用户定义或预置的最晚完成时间减去任务预计运行时间计算。如果系统中有空闲资源，则队列中的任务按序提交运行，如果任务到截止时间仍未被调度，则应该调度到其他数据中心。本章只关注单个的运行异构任务的数据中心，不考虑数据中心间任务和数据的迁移，因此我们假设任务没有截止时间约束，可以无限期排队等待且最终都会被调度执行。

2.5.2 基于混合预测的异构任务调度算法

基于混合预测的异构任务调度算法如算法 2-1 所示。当一个新任务 T_i 到达数据中心，首先分析任务的类别和优先级，选择相应的预测算法和调度策略。若任务类别和优先级为中级（middle），使用 ARMA 模型预测主机上高级别的生产性应用的负载，得出各主机可用资源量（第 4 行），并使用 worst-fit 选择主机 h_j，若 h_j 可用资源足够分配给任务 T_i，则将 Ti 调度到 h_j 上（6-8 行）。否则随机选择主机 h_k，计算潜在空闲资源 ap_k。潜在空闲资源是指，通过不断抢占优先级更低的任务释放的资源与空闲资源的总和，如果潜在空闲资源能够满足 T_i 的需求，即 $ap_k \geqslant r_i$，则抢占这些低优先级任务，将 T_i 调度到 h_k 上（10-18 行）。选择抢占任务的策略包括且不止如下几种：1）最小优先，优先抢占占用资源量最小的任务；2）最优优先，优先选择所有资源占用量 r_v 大于 r_i 的任务中资源占用量最小的任务；3）最大优先，优先抢占占用资源量最大的任务；4）最小完成度优先，优先选择运行完成进度最小的任务；5）最晚完成时间优先：优先选择预计完成时间最长的任务；6）最晚开始时间优先：优先选择开始运行时间最晚的任务。本文随机选择目标任务抢占（第 13 行）。如果没有合适的主机，则将任务放入队列等待（第 19 行），等有任务完成并释放资源时，再进行调度。若任务 T_i 是低优先级的免费任务（gratis），首先更新 AR(p)模型参数（第 24 行），然后预测主机负载和能够提供给 T_i 的可用资源（第 25 行），并使用 worst-fit 选择主机 h_j（第 27 行），若 h_j 可用资源足够分配给任务 T_i，则将 T_i 调度到 h_j 上，否则将任务放入队列等待（28-30 行），等有任务完成并释放资源时，再进行调度。此外，还有一个监控程序周期性使用 AR 模型预测各主机是否过负载，如果一个主机过负载，则监控程序选择并杀死一些任务，直至过负载情况消失。优先选择低优先级任务，对于同一优先级的任务，选择策略与抢占任务的策略相同，本节使用随机选择的策略。

算法 2-1 基于混合预测的异构任务调度算法

输入： $T=\{T_1,T_2,...,T_n\}$，输入的任务队列

$WQ=\{Tw_1,Tw_2,...,Tw_n\}$，排队等待的队列

$H=\{h_1,h_2,...,h_n\}$，云数据中心中的主机集合

1: **for** each task $T_i \in \{T, WQ\}$ **do**
2: **if** *priority*(T_i)=*middle* **do**
3: **for** Each host $h_j \in H$
4: Use ARMA model to predict available resources a_j
5: **end for**
6: Choose the host h_j with maximum a_j
7: **if** $a_j \geqslant r_j$ **do**
8: $h_j \leftarrow T_j$
9: **else do**
10: **if** there is at least one target host **do**
11： Choose a target host h_k randomly, $ap_k=a_k$
12: **do**
13: Choose T_v in h_k randomly
14: $ap_k+=r_v$, $\{temp\} \leftarrow T_v$
15: **until** $ap_k \geqslant r_i$
16: Evict all the tasks in {*temp*}
17: $h_k \leftarrow T_j$
18： **end if**
19： **else** $\{WQ\} \leftarrow T_j$
20: **end else**
21: **end if**
22: **if** priority(T_i)=gratis **do**
23: **for** Each host $h_j \in H$ **do**
24: Update parameters of AR(*p*) model
25: Use AR(*p*) model to predict available resources a_j
26: **end for**
27: Choose the host h_j with maximum a_j
28: **if** $a_j > r_j$ **do**
29: $h_j \leftarrow T_j$

续上表

30: **else** $WQ \leftarrow T_j$

31: **end if**

32: **end for**

2.6 仿真实验

本节使用实测数据进行仿真，分析基于混合预测的异构任务调度算法对性能的改进。主要实验包括预测精度分析，任务调度性能分析，以及比较不同 slot 大小对调度性能的影响。

2.6.1 实验环境描述

本实验使用 Google 公布的云集群 Google cluster[29]实测数据构造异构工作负载。Google “task_event”表中记录了任务提交时间，优先级，以及归一化的 CPU、内存、硬盘需求量。文献[31]指出，优先级大于 8 的任务是生产性任务，小于 2 的是免费任务，其他为中等优先级任务。记录的采样时间是 300s，因此本文的默认 slot 长度也为 300s。Google 公布的数据为 2011 年 5 月份中 29 天内 67 亿个任务在超过 12,000 个主机上的执行跟踪情况。本实验使用了前 7 天的数据，大约 2100 个 slot，共约 1.15 亿个任务。

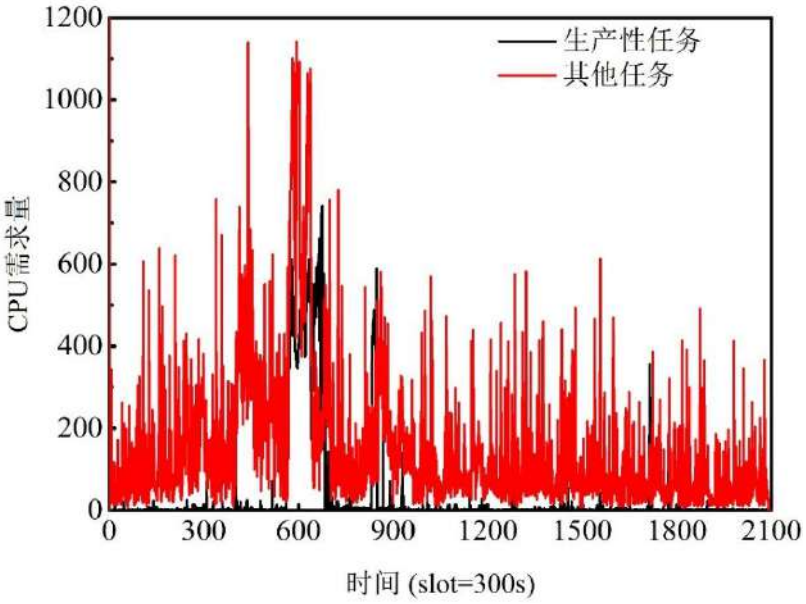


图 2-3 输入的混合负载

工作负载是同一 slot 内正在执行的所有任务的 CPU 需求量之和。图 2-3 显示了输入 2100 个 slot 内的生产性负载和其他负载的分布。大多时间下，生产性负载的时间序列符合平稳分布，在 slot600-700，slot900 时间段有 2 个高峰。其他

负载的时间序列更加不规则，slot400，slot650-700 时间段有 2 个高峰。高峰段代表大量数据同时提交到数据中心，这会引起资源冲突，导致大量的任务抢占和延迟调度，同时，高优先级的生产性任务性能也会受到影响。

云数据中心使用 CloudSim 3.0 [35]工具集仿真。CloudSim 是墨尔本大学网格实验室推出的离散事件仿真器, 支持大型数据中心建模、虚拟机建模以及并行任务建模, 可用于各种资源分配策略的仿真对比。数据中心包括 1000 个主机，上面运行 1000 个虚拟机，与 Google 实测数据一致，CPU、内存、硬盘容量都做归一化处理。

实验对比了如下四个调度算法的性能：

原有算法，基于当前资源可用量尽力分配，在资源不足时，不同优先级的任务进行抢占式调度；

ARMA，基于主机负载预测的资源预留，但仅使用 ARMA 模型预测主机负载，计算可用资源量，进行调度，ARMA 模型参数通过离线训练获得；

FOAR (Feedback based Online AR)，基于主机负载预测的资源预留，但仅使用 AR 模型预测主机负载，计算可用资源量，进行调度，AR 模型参数通过在线反馈实时计算；

MPHW(Multi-Prediction based scheduling for Heterogeneous Workloads)，基于主机负载预测的资源预留，针对不同负载分别使用 ARMA 和 AR 模型预测可用资源量。

2.6.2 预测精度分析

基于预测的调度算法中最重要的因素在于为任务确定合适的可用资源量，针对异构负载的不同模式特点，使用不同的预测模型。本实验用于分析预测精度。使用的数据是原有调度算法下的生产性负载和主机负载。ARMA 模型分别用来预测生产性负载和主机负载，首先将 7 天约 2100 个 slot 的实测数据分成两部分，第 1 天约 300 个 slot 的数据为训练集，用于离线训练，获得 ARMA 模型参数，后面 1800 个 slot 的数据为验证集，用于验证预测算法的精度，将 ARMA 模型的预测值与实测数据比较。使用 IBM 的 SPSS Statistics[36]工具分析训练集，确定模型阶数 $p = 1$, $q = 1$, φ_1, θ_1。在线 AR 模型用来预测主机负载，参数 φ_1, φ_2 基于在线反馈负载值实时更新。在线 AR 模型不需要离线训练阶段，仅使用后 1800 个 slot 作为验证集。

使用成功率和 MAPE（Mean Absolute Percent Error，平均绝对百分误差）来衡量预测精度。成功率是预测成功的数量与总数据量的比较，文献[32]中定义一个成功的预测是指预测值落在实际值的 10%以内的区间中。在本节的调度算法中，若预测值小于实际值，则计算的可用资源大于实际可用资源，这可能会导致资源

的过度提供。因此，本文认为若预测值小于实际值，预测结果失败，一个成功的预测是指预测值不比实际值小，且落在比实际值大于 10%以内的区间里，即：

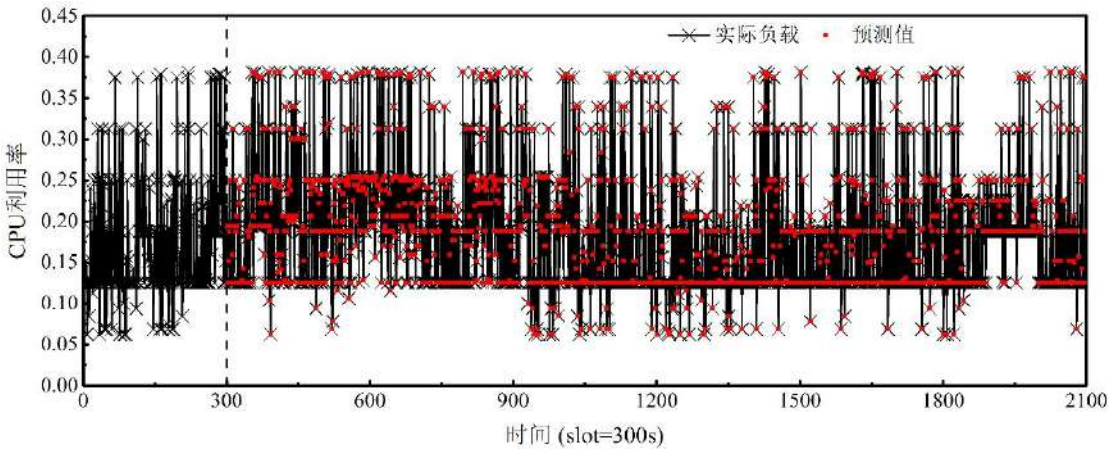
$$\frac{\hat{X}_i - X_i}{X_i} \times 100\% \le 10\% \tag{2-15}$$

MAPE 为：

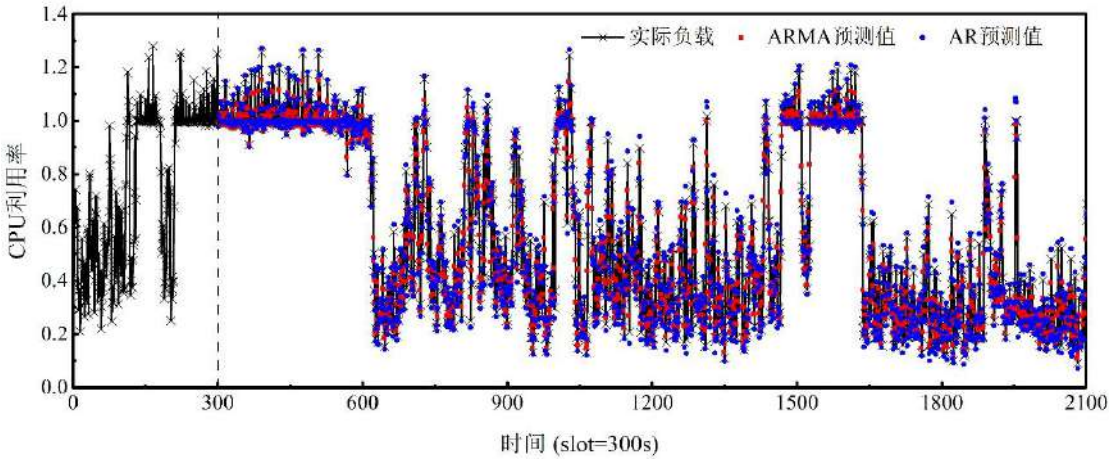
$$MAPE = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{\hat{X}_i - X_i}{X_i}\right| \times 100\% \tag{2-16}$$

其中，X_i 为实际值，$\hat{X}_i$ 为预测值，N 为预测的个数。通常，高的成功率和 MAPE 值，代表了高的预测精度。

图 2-4 显示随机选择的一个主机的实际负载和预测值，运行在该主机上的生产性负载和预测值。图 2-5 显示 1000 台主机的 1800 个 slot 的负载预测成功率和 MAPE 的累积分布 CDF。



a) 生产性负载的 ARMA 预测



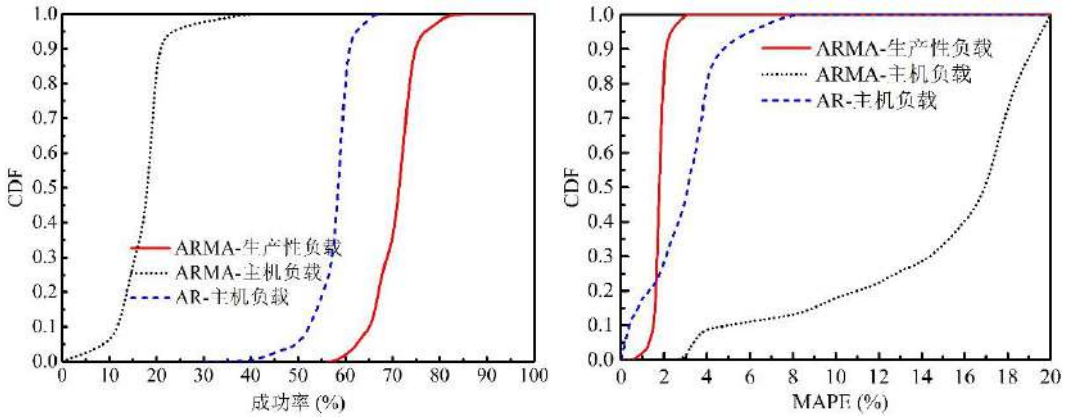
b) 主机负载的 ARMA 预测和 AR 预测

图 2-4 负载的实际值和预测值

图 2-4a)显示生产性负载的实际值与 ARMA 模型的预测值，预测值与实际值

非常接近，图 2-5 显示的平均成功率为 71.8%，MAPE 为 1.81%，表明离线训练的 ARMA 模型足够预测平稳分布的生产性负载。

图 2-4b)显示主机负载的实际值、ARMA 模型的预测值、和基于在线反馈的 AR 模型的预测值，其中，CPU 利用率超过 100%是由资源过度提供造成的，AR 模型的预测值比 ARMA 模型的预测值更加接近实际值。图 2-5 的成功率和 MAPE 的 CDF 分别显示基于在线反馈的 AR 模型性能高于 ARMA 模型。ARMA 模型的平均成功率为 17.22%，MAPE 为 15.14%，而 AR 模型分别为 58.3%和 2.63%。综上，基于在线反馈的 AR 模型更适用于预测不规则动态变化的主机负载。



a) 预测成功率　　b)预测 MAPE 值

图 2-5 不同预测方法的成功率和 MAPE

2.6.3 云数据中心性能分析

本节分析四种调度算法下，任务失败与重调度率、云数据中心有效资源利用率、调度延迟等性能。

1. 任务失败与重调度

图 2-6 对比了四种调度算法下数据中心主机过负载次数、杀死任务数量、抢占任务数量。主机过负载数分别为原有算法 31.8 万、ARMA 算法 25.2 万、FOAR 算法 12.5 万、MPHW 算法 14.4 万，相对于原有算法，ARMA 减少了 20.0%，FOAR 减少了 60.3%，MPHW 减少了 54.85%。失败任务的数量，包括被杀死的和被抢占的，分别为：原有算法 202 万，ARMA 算法 94 万，FOAR 算法 75.8 万、MPHW 算法 60.5 万，相对于原有算法，ARMA 减少了 53.2%，FOAR 减少了 62.3%，MPHW 减少了 69.92%。

失败任务在云数据中心有可用资源时可以被重新调度，重新调度的任务与新输入的任务叠加输入，如果重调度任务过多，也会造成输入高峰。图 2-7 显示了每小时系统输入的任务量，包括新输入的任务和失败重新调度的任务。原有调度算法下，输入任务数量仍有大的高峰，而 ARMA、FOAR、MPHW 算法的输入分

布依次平缓。ARMA 比原有算法总输入量少 8.55%，FOAR 比原有算法总输入量少 10.1%，MPHW 比原有算法、ARMA、FOAR 分别少了 11.22%、2.92%、1.34%。

主机过负载情况下，主机上所有任务的执行速度都会变慢，性能下降，输入高峰意味着很多任务同时调度，会造成资源冲突和任务抢占。从本实验可以看出，FOAR 和 MPHW 在降低系统过负载率、任务失败率方面具有较好的表现。

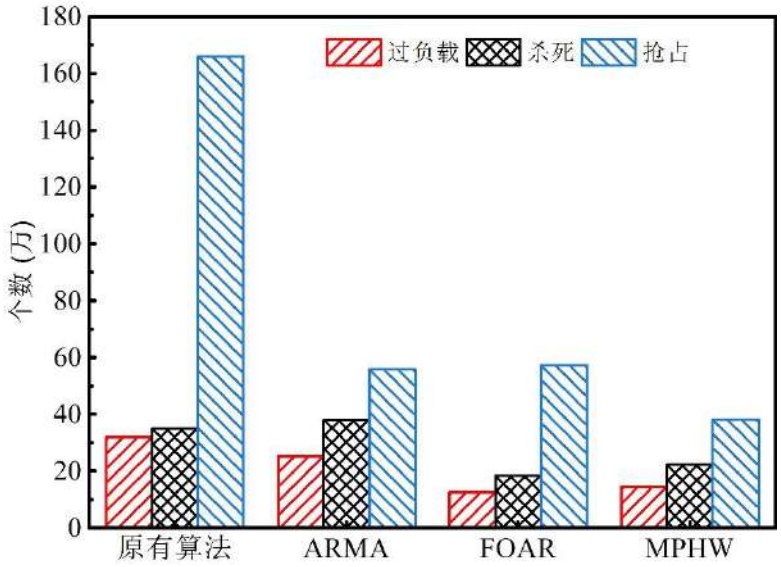


图 2-6 数据中心过负载、杀死、抢占次数

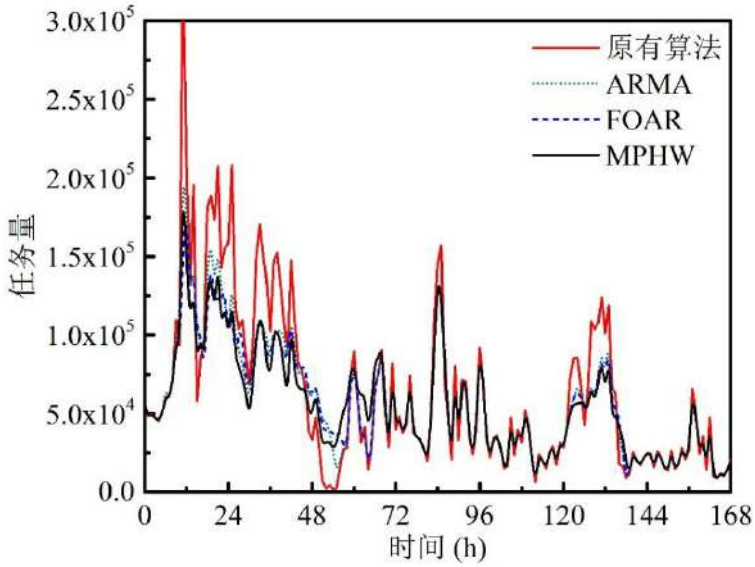


图 2-7 任务重调度数量

2. 云数据中心有效资源利用率

失败任务会造成 CPU 资源的浪费，重调度、抢占等操作也会带来额外开销。本节实验研究四种算法下，数据中心有效资源利用率的情况。

首先引入资源使用率 RUP （Resource Usage Percentage）的概念，来比较数据中心全部被抢占任务、被杀死任务、成功完成任务使用的资源比例。在 $slot_j$ 中，

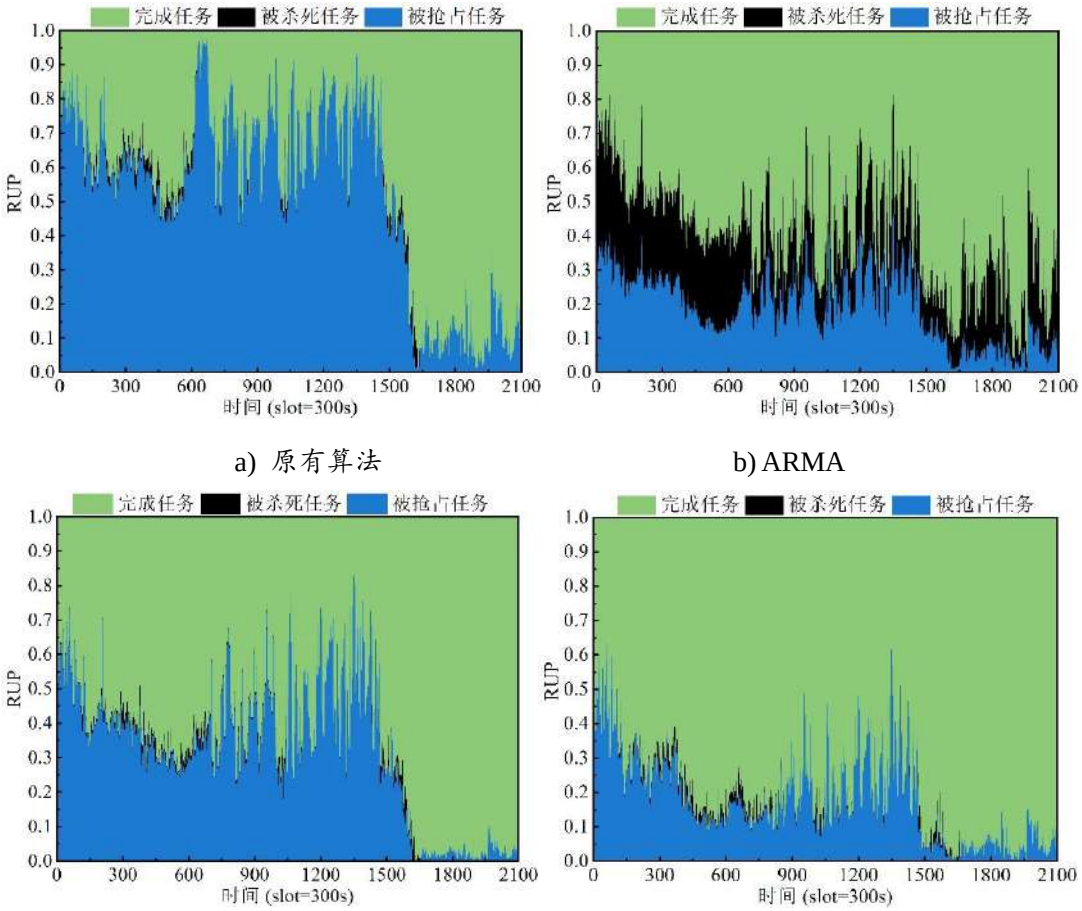
任务类别 $type_i$（被抢占，被杀死，成功完成）的资源使用量 RU_j^i 为该 $slot_j$ 内所有该类任务的 CPU 利用率之和：

$$RU_j^i = \sum_{\substack{T_k \in type_i \\ T_k \in slot_j}} CPU_k \tag{2-17}$$

其中，CPU_k 为任务 T_k 的 CPU 利用率。RUP 为单类任务 RU 与所有任务 RU 之和的比值。

$$RUP_j^i = RU_j^i \Big/ (RU_j^{kill} + RU_j^{evict} + RU_j^{finish}) \tag{2-18}$$

图 2-8 显示了四种调度算法下，数据中心的各类任务的 RUP。失败任务的平均 RUP 从原有算法的 59.17%下降到 ARMA 45.24%、FOAR 38.19%、MPHW 29.99%。成功完成任务的平均 RUP 从 40.83%上升到 ARMA 54.76%、FOAR 61.61%、MPHW 70.01%。



a) 原有算法　　b) ARMA

c) FOAR　　d) MPHW

图 2-8 完成、被杀死、被抢占任务的资源使用率

CPU 浪费率为使用的 CPU 中被失败任务浪费掉的部分，是失败任务的 RUP 与主机 CPU 利用率的乘积：

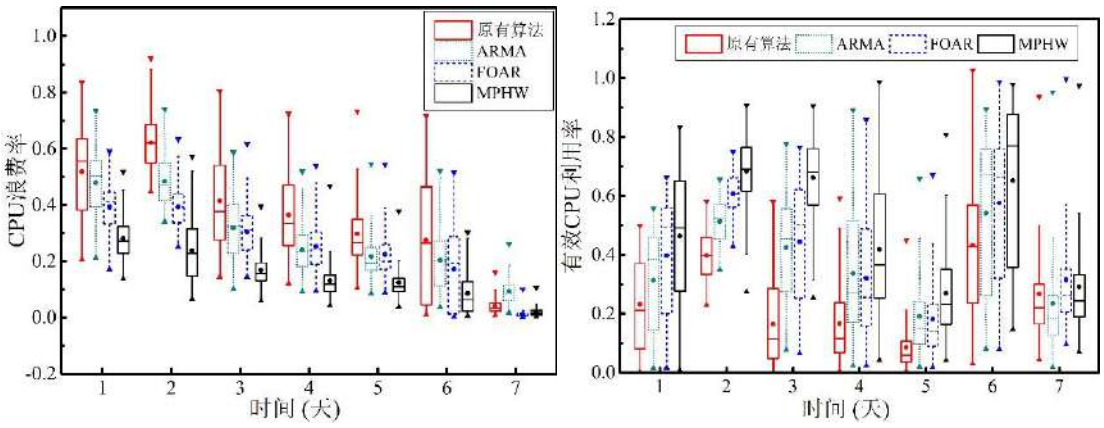
$$wasteCPU_j = (RUP_j^{kill} + RUP_j^{evict}) * \sum_{h_k \in DC} CPU_j^{h_k} \tag{2-19}$$

其中，$CPU_j^{h_k}$ 为在 $slot_j$ 中，主机 h_k 的 CPU 利用率。

数据中心的资源利用率通常指主机上 CPU 利用率，但是失败任务消耗的主机 CPU 是一种资源浪费，本节定义有效资源利用率来描述数据中心的资源利用率，为 CPU 中成功完成的任务使用的部分，是完成任务的 RUP 与主机 CPU 利用率的乘积：

$$\text{eff}CPU_j = RUP_j^{finish} * \sum_{h_k \in DC} CPU_j^{h_k} \tag{2-20}$$

图 2-9 显示了每天数据中心的资源浪费率和有效资源利用率的情况，箱图呈现了数据集的最小值、25 分位、中位数、75 分位、最大值。图 2-9a)显示四种算法的资源浪费率，最大浪费率分别为原始算法 92.0%、ARMA 73.97%、FOAR 63.30%、MPHW 57.18%。平均浪费率分别为原始算法 36.14%、ARMA 28.13%、FOAR 23.95%、MPHW 14.95%。在第二天对应图 2-3 的 slot 为 400 处的输入高峰期，四种调度算法的资源浪费率差别最大。图 2-9b)显示四种算法的有效资源利用率，平均有效利用率分别为原始算法 29.10%、ARMA 36.49%、FOAR 40.57%、MPHW 49.12%。MPHW 比原有算法节省超过 50%的资源，有效利用率提高超过 65%。



a)资源浪费率　　b)有效资源利用率

图 2-9 云数据中心的资源利用率

3. 调度延迟

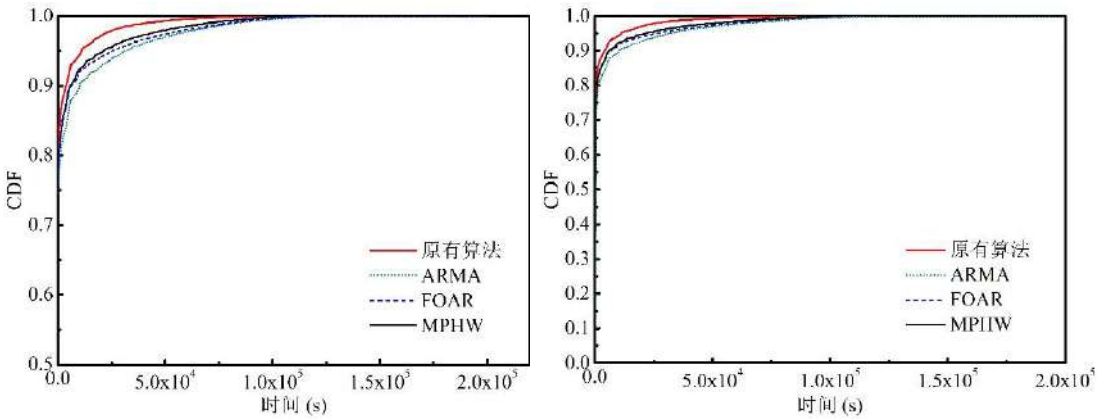
在基于预测的任务调度中，需要为更高优先级的负载未来资源需求增加预留出资源。对于 QoS、SLA 需求高的高优先级任务，采用立即调度方式，具体为，若可用资源足够，则立即分配调度；若可用资源不足，则抢占其他任务，释放资源后调度；此外，后台监控程序基于预测监控主机负载，当预测到主机有过负载可能时，杀死低优先级任务释放资源，保障高优先级任务的性能。因此，高优先

级任务没有时延或时延很小。

对于低优先级的任务，即使当前可用资源高于需求量，如果未来可用资源不足，也不能立刻调度。这会带来额外的调度延迟。传统的调度延迟是指从任务提交到任务被调度到某一资源节点上的时间跨度，这不能很好的描述调度性能，因为任务在被调度后还可能会被杀死或抢占，本文定义的调度延迟是从任务提交到数据中心到成功完成的那次调度的时间跨度。同时，本文定义任务响应时间为从任务提交到成功完成的时间跨度。

图 2-10 为任务调度延迟和响应时间的 CDF。任务立刻成功调度的比例为：原有算法 79.81%、ARMA 74.74%、FOAR 76.65%、MPHW 77.56%。相对于原有算法的任务平均调度延迟，ARMA 增加了 139.78%，FOAR 增加了 101.0%，MPHW 增加了 75.38%。相对于原有算法的任务平均响应时间，ARMA 增加了 122.25%，FOAR 增加了 83.38%，MPHW 增加了 65.62%。

调度延迟适当的增加是可以忍受的，因为中间优先级任务和免费任务通常是非实时任务，对延迟不敏感，而前面分析的输入高峰的缓解可以保证生产性任务的性能。



a)任务调度延迟　　b)任务响应时间

图 2-10 云数据中心任务的时间性能

上述实验通过对比原始算法和基于预测的算法，表明在计算可用资源时预测未来负载情况，可以减小失败任务数量、减小主机过负载情况、减少资源浪费，相对于 QoS 和 SLA 等级较低的应用，适当的额外延迟也是可以接受的。ARMA、FOAR、MPHW 算法的对比表明，负载预测精度可以影响调度性能。

2.6.4 slot 尺寸对调度性能的影响

基于预测的调度算法预测在任务执行过程中生产性负载或主机负载的最大值，因此，预测的时间段 slot 最好接近任务运行时长。新任务的准确运行时长通常很难确定，本文使用统计方法估算。图 2-11 显示了任务运行时长的 CDF。不

足 15%的任务在 100s 内完成，接近 60%的任务在 300s 内完成，89%左右的任务在 600s 内完成，超过 98%的任务在 900s 内完成。本实验比较不同 slot 长度下，系统性能，slot 长度分别设为 100s、300s、600s、900s。

连续的时间被分割成离散的 slot，每个 slot 内负载的最大值作为负载时间序列中对应于该 slot 的值，随着 slot 长度的增加，每个 slot 内的最大值也增加，负载时间序列变得更加平稳，未来 slot 的预测值也增大，为新任务预测的可用资源量减小，任务被立刻调度的概率减小，则失败任务数减小。在 slot 为 300s 时，失败任务数量比 slot 为 100s 时少 25.57%，比 slot 为 600s 时多 8.25%，比 slot 为 900s 时多 22.33%。图 2-12 显示不同 slot 长度下总的任务输入量，包括新提交的任务和重调度的任务，随着 slot 长度的增加，任务输入量的波动变缓。

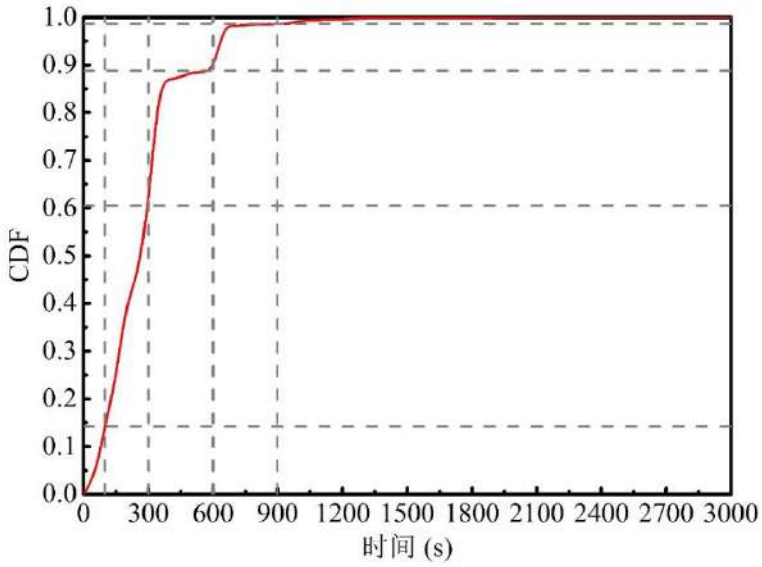


图 2-11 任务运行时长

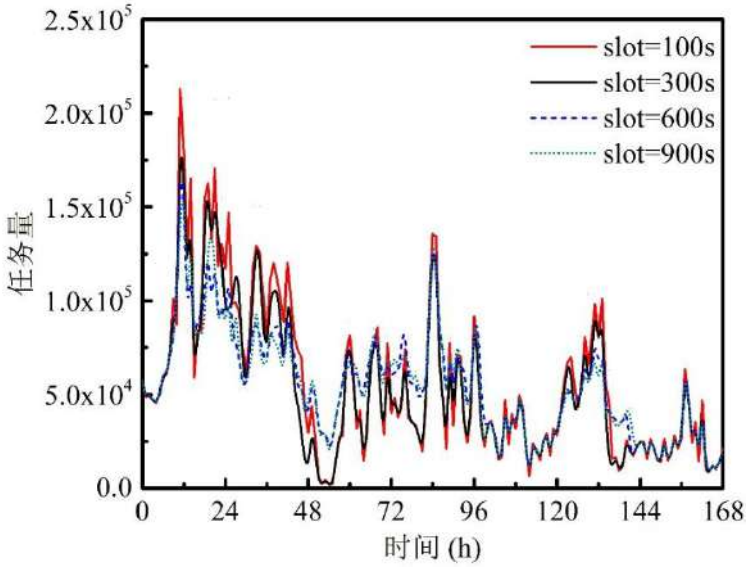
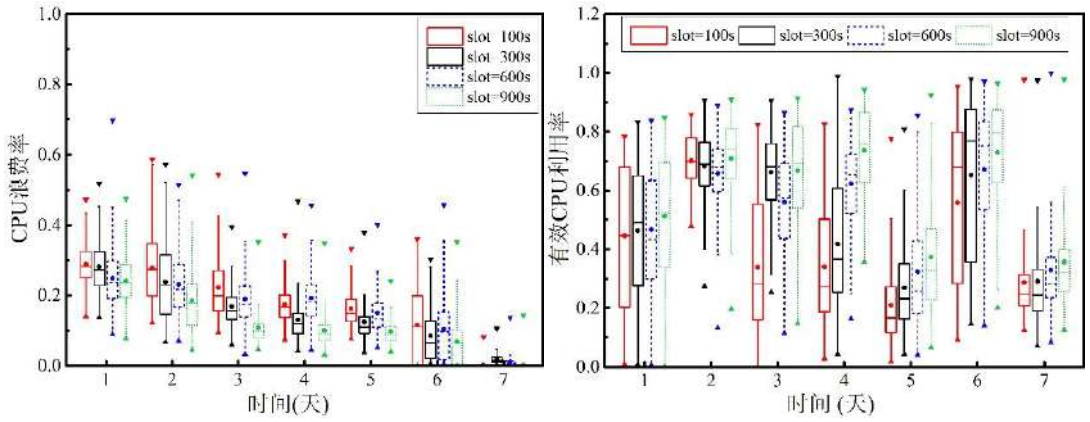


图 2-12 不同 slot 长度下任务输入量

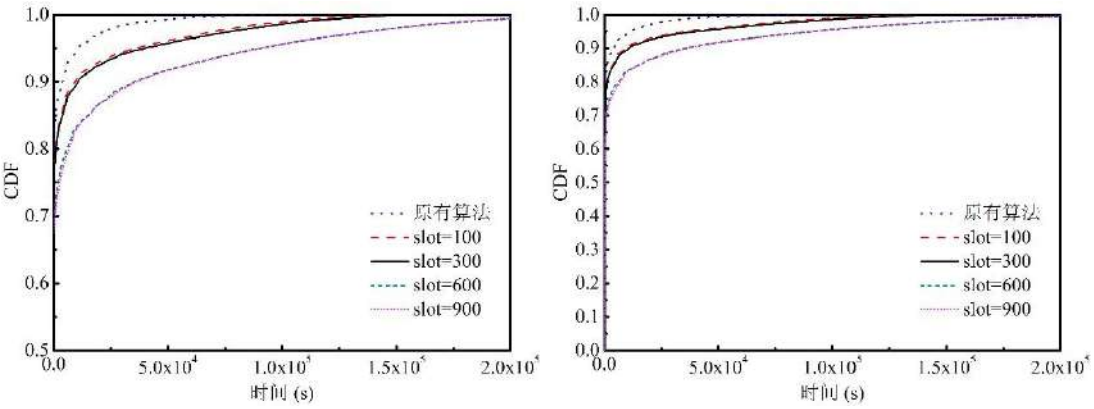
图 2-13 显示了不同 slot 长度下资源利用率的情况。随着失败任务的减少，资源浪费率降低，如图 2-13a)所示。平均 CPU 浪费率从 slot 为 100s 时的 17.72%减小到 slot 为 600s 时的 16.05%，到 slot 为 900s 时的 11.45%。随着失败任务的减少，有效 CPU 利用率提高，如图 2-13b)所示。从 slot 为 100s 时的 41.16%提高到 slot 为 600s 时的 51.87%，到 slot 为 900s 时的 58.33%。



a)资源浪费率　　b)有效资源利用率

图 2-13 不同 slot 长度下的数据中心资源利用率

图 2-14 显示了不同 slot 长度下任务调度时延和响应时间的 CDF。slot 长度为 100s 时的任务时间性能与原有算法的任务时间性能基本相同，随着 slot 长度的增加，延迟调度的任务比例从 slot 为 100s 时的 18.64%增加到 900s 时的 27.96%，平均时延从 slot 为 100s 时的 2023.84s 增加到 slot 为 300s 时的 3308.30s、slot 为 900s 时的 7066.61s，平均响应时间从 slot 为 100s 时的 2302.74s 增加到 slot 为 300s 时的 3589.31s、slot 为 900s 时的 7381.74s。slot 为 900s 时的平均调度时延和平均响应时间约为 slot 为 300s 时的 2 倍。



a)任务调度延迟的 CDF　　b)任务响应时间的 CDF

图 2-14 不同 slot 长度下任务的时间性能

该实验表明，预测时间段 slot 的长度由数据中心中任务的运行时长决定，并且影响任务的调度性能和数据中心资源利用率。如果 slot 长度太小，特别是不能

覆盖大部分任务的运行时长，则预测的可用资源量不能很好的反应任务未来运行期内的实际可用资源量。如果 slot 长度能够覆盖大部分任务的运行时长，随着 slot 长度的增加，数据中心资源浪费率会减小，有效资源利用率会增加，但是调度延迟和响应时间也都会随之增加。因此，slot 长度设置需要权衡两者的关系。

2.7 本章小结

本章针对混合了低 QoS 和 SLA 需求的应用与高优先级的生产性应用的云数据中心，提出基于混合预测的异构任务调度算法。异构任务被分为高、中、低三个优先级，时间连续的负载被划分为离散的时间序列。对于中等优先级任务，使用离线训练的 ARMA 模型对平稳分布的生产性负载序列进行预测，资源可用量为当前可用量减去未来时间段内生产性负载增加量，如果可用资源不足，则随机选择一个主机抢占其上的最低优先级任务直至释放的资源量足够调度该任务。对于最低优先级任务，使用基于在线反馈的 AR 模型对不平稳分布的主机负载序列进行预测，可用资源量为未来时间段内主机可用资源的最小值，如果可用资源不足，则将任务放至等待队列，直至资源足够时再提交调度。仿真实验表明，相对于基于即时可用资源量的抢占式调度算法，本章提出的基于混合预测的异构任务调度算法能够减小 70%主机过负载和任务失败次数，能够减少超过 50%的资源浪费，有效资源利用率提高超过 65%。此外，虽然调度延迟有所增加，但是对于 QoS、SLA 需求低的任务来说，是可以忍受的。

本章提出的异构任务调度算法中，对于生产性负载的时间序列分析是基于 Google cluster 的负载分析，符合平稳分布，使用离线训练的 ARMA 模型进行预测，若在其他环境中，生产性负载不符合平稳分布，则也使用在线反馈的 AR 模型进行预测。

参考文献

[1] Guenter B, Jain N, Williams C. Managing cost, performance, and reliability trade-offs for energy-aware server provisioning [A]. // Proceedings of IEEE INFOCOM [C], Shanghai, IEEE Press, 2011:702-710.

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

[3] Bhattacharya A A, Culler D, Friedman E, et al. Hierarchical scheduling for diverse

datacenter workloads [A]. //Proceedings of the 4th annual Symposium on Cloud Computing [C], California: ACM Press,2013:1-15.

[4] Apache Fair scheduler, http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/FairScheduler.html, 2017,4,10.

[5] Apache Capacity scheduler, http://hadoop.apache.org/docs/r2.7.0/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html, 2017,4,10.

[6] Apache Yarn, https://hadoop.apache.org/docs/r2.4.1/hadoop-yarn/hadoop-yarn-site/index.html, 2017,4,10.

[7] Apache Mesos, http://mesos.apache.org/, 2017,4,10.

[8] Schwarzkopf M, Konwinski A. Abd-El-Malek M, et al. Omega: flexible, scalable schedulers for large compute clusters [A]. //Proceedings of the European Conference on Computer Systems [C], Prague: ACM Press, 2013:351-364.

[9] Google Kubernetes, http://kubernetes.io/, 2017,4,10.

[10] Carrera D, Steinder M, Whalley I, et al. Enabling resource sharing between transactional and batch workloads using dynamic application placement [A]. //Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware archive [C], Belgium: Springer Press, 2008: 203-222.

[11] Carrera D, Steinder M, Whalley I, et al. Managing SLAs of heterogeneous workloads using dynamic application placement [A]. //Proceedings of the 17th International Symposium on High Performance Distributed Computing [C], Boston: ACM Press, 2008: 217-218.

[12] Garg S K, Gopalaiyengar S K, Buyya R, SLA-based resource provisioning for heterogeneous workloads in a virtualized cloud datacenter [A]. //Proceddings of the 11th International Conference on Algorithms and Architectures for Parallel Processing [C], Melbourne: Springer Press, 2011: 371-384.

[13] Garg S K, Toosi A N, Gopalaiyengar S K, et al. SLA-based virtual machine management for heterogeneous workloads in a cloud datacenter [J]. Journal of Network and Computer Applications, 2014, 45 (C): 108-120.

[14] Ghodsi A, Zaharia M, Hindman B, et al. Dominant resource fairness: fair allocation of multiple resource types [A]. //Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation [C], Boston: USENIX Press, 2011:323-336.

[15] Curinom C, Difallahu D E, Douglasm C,et al. Reservation-based scheduling: if you’re late don’t blame us! [A]. //Proceedings of the ACM Symposium on

Cloud Computing, Seattle: ACM Press, 2014: 1-14.

[16] Sharma B, Wood T, Das C R. HybridMR: a hierarchical MapReduce scheduler for hybrid data centers [A]. //Proceedings of IEEE 33rd International Conference on Distributed Computing Systems [C], Philadelphia: IEEE Press, 2013: 102-111.

[17] Delimitrou C Kozyrakis C. Quasar: resource-efficient and QoS-aware cluster management [A]. //Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems [C], New York: ACM Press, 2014:127-144.

[18] Dodonov E Mello R F. A novel approach for distributed application scheduling based on prediction of communication events [A]. Future Generation Computer System, 2010, 26(5): 740-752.

[19] Farahat M A, Talaat M. Short-term load forecasting using curve fitting prediction optimized by Genetic Algorithms [J]. International Journal of Enegry Engineering, 2012, 2: 23-38.

[20] Khan A, Yan T S X, Nikos A. Workload characterization and prediction in the cloud: a multiple time series approach [A]. //Proceedings of the IEEE Network Operations and Management Symposium, Westin Maui: IEEE Press, 2012: 1287-1294.

[21] Yang Q, Peng C, Yu Y, et al. Host load prediction based on PSR and EA-GMDH for cloud computing system [A]. //Proceedings of IEEE Third International Conference on Cloud and Green Computing [C], Karlsruhe: IEEE Press, 2013: 9-15.

[22] Yang D, Cao J, Yu C, et al. A multi-step-ahead CPU load prediction approach in distributed system [A]. //Proceedings of the Second International Conference on Cloud and Green Computing [C], Xiangtan: IEEE Press, 2012: 206-213.

[23] Dia S, Kondo D, Cirne W. Host load prediction in a Google compute cloud with a Bayesian model [A]. //Proceedings of the IEEE/ACM Conference on High Performance Computing Networking, Storage and Analysis [C], Salt Lake City: IEEE Press, 2012: 1-11.

[24] Dia S, Kondo D, Cirne W. Google hostload prediction based on bayesian model with optimized feature combincation [J]. Journal of Parallel and Distributed Computing, 2014, 74: 1820-1832.

[25] Zhang Q, Zhani M F, Zhang S, et al. Dynamic energy-aware capacity provisioning for cloud computing environments [A]. //Proceedings of the 9th International

Conference on Autonomic Computing [C], San Jose: ACM Press, 2012: 145-154.

[26] Cheng L, Zhang Q, Boutaba R. Mitigating the negative impact of preemption on heterogeneous mapreduce workloads [A]. //Proceedings of the 7th International Conference on Network and Services Management [C], Paris: IEEE Press, 2011:189-197.

[27] Huang Q, Shuang K, Xu p, et al. Prediction-based Dynamic Resource Scheduling for Virtualized Cloud Systems [J]. Journal of Networks, 2014, 9 (2): 375-383.

[28] Solanki A S, Kothair A. Predictive resource management in cloud using jybrid hidden markov model: HHMM [J]. International Journal of Engineering Science and Computing, 2016, 6 (7): 8244-8250.

[29] Google cluster data, http://code.google.com/p/googleclusterdata/wiki/ClusterData2011_1, 2017,4,10.

[30] Reiss C. and Wilkes J., Google cluster usage traces: format + schema (version of 2011.10.27, for trace version 2), https://drive.google.com/open?id=0B5g07T_gRDg9Z0lsSTEtTWtpOW8&authuser=0, 2017,4,10.

[31] Reiss C, Tumanov A. Heterogeneity and dynamicity of clouds at scale: Google trace analysis [A]. //Proceedings of the Third ACM Symposiumon Cloud Computing [C], San Jose: ACM Press, 2012: 7.

[32] Abdul-Rahman O A, Aida K. Towards understanding the usage behavior of Google cloud users: the mice and elephants phenomenon [A]. //Proceedings of IEEE 6th International Conference on Cloud Computing Technology and Science, Singapore: IEEE Press, 2014: 272-277.

[33] Box G E P, Jenkins G M, Reinsel G C. Time series analysis: forecasting and control [M]. 4th ed. San Francisco: Wiley Press, 2011: 56-81.

[34] 李华, 胡奇英. 预测与决策教程 [M]. 北京: 机械工业出版社, 2012:131-168.

[35] 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, Software, Practice and Experience, 2011, 41(1): 23-50.

[36] IBM SPSS Statistics, http://www.spss.co.in/, 2017, 4, 10.

第三章 不可靠私有云环境下科学工作流动态调度算法

3.1 引言

云平台上的资源有一系列不确定性因素需要考虑：1）性能异构，在云数据中心中，CPU、硬盘、网络交换机等硬件设备的更新换代，使得同一类型的硬件资源实际计算能力不同[2]，如文献[3]所述，“同一”类型的资源在“服务”同一个任务时，速度也不同，这意味着同一类型的资源，包括硬件资源和虚拟机资源，性能异构；2）性能动态变化，实际运行期内计算节点的计算速率、网络传输速率实时变化，部署在同一主机上的虚拟机之间的干扰[3]也会影响资源性能；3）节点失效，大规模复杂的分布式系统中，硬件故障、软件错误等资源失效问题不可避免。混合了一系列不确定因素的云平台，称为不可靠云平台。

在科学工作流中，任务之间的依赖关系主要是数据传输，通常用有向无环图DAG（Directed Acyclic Graph）表示。现有科学工作流调度算法能够很好的适应异构的私有云平台，但大多是静态的调度算法，在工作流提交时根据平台计算和传输速度来调度任务。在不可靠云平台上，资源计算和传输速度动态变化且可能失效，静态调度算法不能保证实际运行时的主机性能保持最优。少数应对不可靠云平台的算法基于重调度机制，在感知系统性能变化后重新调度，不适合资源动态变化、节点失效更加频繁的云平台。本章研究如何将科学工作流动态调度到不可靠的私有云平台上，在完工时间最小的同时，具有健壮性。

3.2 相关研究工作

任务调度是指将任务映射到计算节点上，满足一系列性能目标，例如执行时间最短、开销最小等[1]。DAG 工作流调度属于 NP-难问题，针对同构计算系统、异构计算系统、网格、云平台的科学工作流调度和完工时间最短的问题，目前有很多启发式、元启发式算法[1,4,5]。启发式算法、构造式算法的复杂度角度，主要分为任务列表算法、聚类算法、基于任务复制的算法三类，任务列表算法由于其复杂度最低而更受青睐[7]。在动态资源环境中，静态算法通常在虚拟机选择阶段，为虚拟机计算速率引入一个性能变化参数来估算任务完成时间；动态算法则大多针对 Grid 环境，使用重调度机制，监控到性能变化或任务实际完成时间与预计不同时，重新对剩余任务进行调度。

3.2.1 静态调度算法

HEFT (Heterogeneous Earliest Finish Time)[6]算法、CPOP (Critical Path on a Processor) [6]算法、PETS (low complexity Performance Effective Task Scheduling) [7]算法、HCPT (Heterogeneous Critical Parent Trees) [8]算法、HPS (High Performance task Scheduling) [9]算法、Lookahead 算法 [10]、SDBATS (Standard Deviation Based Algorithm for Task Scheduling) [11]算法、PEFT (Predict Earliest Finish Time algorithm) [12]算法都是目前比较流行的、性能较好的、静态的、基于任务列表的调度算法。

HEFT、CPOP、HCPT、SDBATS、Lookahead、PEFT 算法包括两个阶段：

- 任务排序阶段，根据任务节点的 $rank_u$ 值、关键路径、最佳开销表等参数，确定任务调度顺序；
- 资源选择阶段，为任务分配计算资源，使得工作流的完工时间最短。

HPS 和 PETS 算法在这两个阶段之前还有一个任务打包阶段，将能够并行执行的任务打包成组。

HEFT 是最经典、最常用的算法，算法的时间复杂度最低，PEFT 是当前调度结果最好的静态调度算法，且具有跟 HEFT 相同的时间复杂度。上述算法都是静态调度算法，在计算资源一直可用且速度不变的可靠静态环境下可以产生很好的调度结果。

3.2.2 性能变化感知的静态调度算法

在实际运行环境中，资源节点不是一直可用的、性能通常是动态变化的，实际运行时的计算和传输速度通常与运行之前的估算值相差很大，纯静态调度算法未必产生很好的调度结果。

一些静态 DAG 工作流的调度算法假设计算资源的计算速度或任务执行时间变化情况符合一定的分布规律，来修正虚拟机选择阶段计算的任务完成时间。文献[13]为虚拟机的计算速率引入一个基于正态分布的性能下降率指标。文献[14]假设计算时间和数据传输时间相互独立且服从指数分布或正态分布，统计分析历史数据获得分布的参数。文献[15]假设任务运行时间的变化率符合均值为 0 的正态分布。

通常，云环境下的虚拟机的计算速度或任务执行时间不符合固定的分布规律，文献[16, 17]使用蒙特-卡罗算法解决这一问题。首先，工作流的任务执行时间预测结果组成一个输入空间，从输入空间中抽样，对应每一组抽样的任务运行时间用静态调度算法可以生成一个调度结果，所有调度结果中平均完工时间最小的那个调度结果即为最终的调度结果。该算法的速度和效果依赖于输入空间的规模、抽样个数、静态调度算法的选择。

3.2.3 动态调度算法

目前的动态调度算法主要使用重调度机制，监控到性能变化或任务实际完成时间与预计不同时，重新对剩余任务进行调度。

文献[18]设计了一种离线调度器，根据运行期任务的开始时间和完成时间信息，动态调整调度策略，如果一个任务的延迟可能导致完工时间的增加，其后继任务被重新调度到更快的计算节点上。该算法能够减小一个任务延迟带来的连锁效应，但是只能保证完工时间不增加，只在任务延迟时启动的重调度机制并不能有效利用速度快的计算节点来主动缩短完工时间。

P-HEFT (HEFT for parallel tasks) [19]算法动态调度一组不同时间到达的 DAG 应用，某个机制触发下，如新应用到达、资源性能变化等，将现有的所有应用中，可以立刻调度的任务打包成一个 DAG，然后使用 HEFT 算法调度到异构集群中。

DCP-G (Dynamic Critical Path for Grid) [20]首先按照计算和传输开销最小的情况识别 DAG 应用的关键路径，按序调度就绪的关键路径上的任务，任务调度到节点上之后，动态更新该任务的计算和传输开销，并重新计算 DAG 应用的关键路径。运行期检测到资源性能变化时，进行重调度。

3.2.4 健壮性及调度算法

当前没有一个公认的调度健壮性的定义，一些定义将调度的健壮性与截止时间关联。文献[21]定义任务的松弛时间为在不影响整个工作流截止时间的情况下，最长可以延迟的时间，调度的松弛时间为工作流中所有任务松弛时间之和。文献[15]给出了健壮性的两种定义：1）工作流在截止时间之前完成的概率；2）工作流在不超出截止时间约束情况下，最长可以延迟的时间。文献[22]给出了健壮性的两种定义：1）拖延期，即工作流实际完成时间超出截止时间的长短；2）超时率，即工作流实际完成时间超出截止时间的频率。使用双目标的遗传算法，最小化完工时间，最大化健壮性。

在没有截止时间约束的调度中，健壮性通常与完工时间关联，根据在资源计算传输速度动态变化的情况下，同一调度映射对应的不同完工时间的分布，计算健壮性。文献[23]将 DAG 工作流的任务多次调度到同一动态平台，通过分析完工时间的分布，使用标准差、微分熵[21]、落入某个区间的概率[24]、99%分位数等指标计算健壮性。提出了演进的元启发算法，调度结果的完工时间较短但是算法速度较慢，提出了简单的启发式算法，算法速度较快但是调度结果的完工时间一般。

3.2.5 现有工作存在的问题

现有的工作主要存在以下问题：

（1）静态调度算法适用于可靠的静态异构环境，对资源动态性建模的方法依赖于模型的精确性，不能满足云平台资源节点性能高度动态变化且没有规律的情况；

（2）现有的动态调度算法大多是感知系统性能变化后重调度，云平台资源量大，从整个平台来看，所有资源动态变化较为频繁，重调度的时间节点难以确定，重调度操作过于频繁，带来的额外开销较大；

（3）现有的衡量是否能在性能动态变化的资源节点上生成平稳的完工时间的健壮性指标——标准差与平均完工时间相关，准确度不高，见 3.4 节实例分析，不能有效衡量调度算法产生平稳的、较短的完工时间的能力。

针对上述问题，本文针对异构、动态、有一定失败率的不可靠云平台，提出了动态的科学工作流调度算法，能在性能动态变化的资源节点上生成平稳的最短完工时间。

3.3 系统模型

IaaS 云和私有云以 VM（Virtual Machine，虚拟机）的形式向用户提供服务。VM 配置信息包括：OS、CPU（用每秒百万级指令数 MIPS 衡量）、内存、网络带宽等。当一个工作流到达云平台，某类符合用户需求或工作流配置信息的 VM 被提供给工作流使用。

科学工作流中的任务和任务之间的依赖关系通常建模为有向无环图 DAG，用 $G=\{V,E\}$表示，其中，$V=\{T_1,T_2,...,T_n\}$代表一个工作流中的 n 个任务，$E=\{(T_i,T_j)|\ T_i,T_j\in V\}$表示任务 T_i 到 T_j 的依赖关系，$e=(T_i,T_j)$的值为任务 T_i 向任务 T_j 传输的数据量。边$(T_i,T_j)\in E$ 表示任务 T_i 完成后才能执行任务 T_j，任务 T_i 称为 T_j 的父任务，记为 $parent(T_j)$，任务 T_j 称为 T_i 的子任务，记为 $child(T_i)$，没有任何父任务的任务称为 DAG 的入口，记为 T_{entry}，没有任何子任务的任务称为 DAG 的出口，记为 T_{exit}。本节的工作涉及以下定义：

定义 1. 计算开销。在时刻 t 任务 T_i 在虚拟机 VM_j 上的估计执行时间，记为 $w_{i,j}^t$。平均计算开销 $\overline{w_i^t}$ 是在时刻 t 任务 T_i 在 M 个虚拟机上的平均估计执行时间：

$$\overline{w_i^t}=\sum_{j=1}^{M}w_{i,j}^t/M=\sum_{j=1}^{M}(L_i/R_j^t)/M \tag{3-1}$$

其中，R_j^t 是 VM_j 在时刻 t 的计算速度，用 MIPS（Million Instructions Per Second，

每秒处理的百万级的机器语言指令数）衡量，L_i 是任务 T_i 的长度，用 Million Instructions（百万机器语言指令数）衡量。

定义 2. 数据传输开销 c_{ij}^t。在时刻 t 任务 T_i 向任务 T_j 传输数据的估计平均时间：

$$c_{ij}^t = delay + D_{ij} / B_{ij}^t \tag{3-2}$$

其中，$delay$ 是网络平均时延，B_{ij}^t 是时刻 t 的网络传输速度，D_{ij} 是任务 T_i 向任务 T_j 传输的数据量。值的注意的是，如果两个任务运行在同一个虚拟机上，或者分别运行在部署于同一物理主机上的不同虚拟机上，则 c_{ij}^t =0。

定义 3. 工作流的完工时间 makespan。一个工作流从提交到最后一个任务完成的时间：

$$makespan = \max\{FT(T_{exit})\} - ST \tag{3-3}$$

其中，ST 是工作流提交到云上的时刻，$FT(T_i)$是任务 T_i 完成的时刻。对于不止一个出口任务的工作流，所有出口任务的最晚完成的时间即为工作流的完成时间。

为了方便比较，makespan 通常需要归一化，称为 NSL(Normalized Schedule Length)，定义如下：

$$NSL = makespan / L_{CP_{\min}} \tag{3-4}$$

$$L_{CP_{\min}} = \sum_{T_i \in CP} \min_{VM_j \in \{VM\}} \{w_{ij}\} + \sum_{\substack{T_i, T_k \in CP \\ T_k \in child(T_i)}} D_{ik} / \max_{VM_j, VM_l \in \{VM\}} B_{jl} \tag{3-5}$$

其中，$L_{CP_{\min}}$ 是理论最小完工时间，是工作流 DAG 的关键路径（Critical Path，CP）的最短长度，关键路径是从入口任务到出口任务的最长路径。在计算 $L_{CP_{\min}}$ 时，任务的计算开销是将任务运行在最快节点上获得，传输开销通过将带宽设置为网络的最大值获得。工作流的实际完工时间不可能小于 $L_{CP_{\min}}$，即 NSL 的值永远大于 1.0。NSL 值越小，意味着调度的效果越好。

定义 4. 最早开始时间 EST(Earliest Start Time)。任务 T_i 在虚拟机 VM_j 上最早开始于所有父任务都完成，中间数据都传输到 T_i，并且实例 VM_j 空闲。

$$EST_{ij}^t = \max(\max_{T_p \in parent(T_i)} (FT(T_p) + c_{pi}^t), FreeSlot_j^t) \tag{3-6}$$

其中，$FreeSlot_j^t$ 是自时刻 t 开始算起，VM_j 的最早空闲时间，对于入口任务，$EST_{entry,j}^t = \max(FT, FreeSlot_j^t)$。

定义 5. 数据最晚传输时间 DLTT (Data Latest Transmission Time)。为了保证时刻 t 任务 T_i 在虚拟机 VM_i^t 上的最早开始时间 EST_{ij}^t，在 VM_p 上运行完成的父任务 T_p 最晚开始传输中间数据的时刻：

$$DLTT(T_i, VM_i^t, T_p, VM_p; t) = EST_{ij}^t - c_{ip}^t \tag{3-7}$$

定义 6. 最早完成时间 EFT(Earliest Finish Time)。任务 T_i 在虚拟机 VM_j 上最早完成时间为：

$$EFT_{ij}^t = EST_{ij}^t + w_{ij}^t \tag{3-8}$$

3.4 健壮性指标

工作流调度算法的健壮性，用来衡量同一调度在速度动态变化的资源上产生的完工时间的分布的平稳度。

定义 7. 将一个工作流调度到动态云上 N 次，M_i 是第 i 次调度对应的完工时间，μ 为 $M_i, i=1,2,...,N.$的均值，σ 为标准差，变异系数 CV (Coefficient of Variations) 即为调度的健壮性：

$$CV = \sigma / \mu \tag{3-9}$$

$$\mu = \frac{1}{N}\sum_{i=1}^{N} M_i \tag{3-10}$$

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(M_i - \mu)^2} \tag{3-11}$$

文献[23]对比了现有的几种与完工时间平稳度相关的健壮性定义和性能，包括标准差、微分熵、落入某个区间的概率、99%分位数，实验表明标准差是最具有代表性且易于计算的定义，标准差 σ 反应了完工时间的分布距离平均值 μ 的远近关系。

但是，对于两个调度算法在完工时间的均值 μ 不同的情况下，标准差 σ 并不总是能很好的比较二者的健壮性。例如：调度 1 和调度 2 分别运行 5 次，产生的完工时间分别为{50s,49s,50s,51s,50s}，{30s, 29s, 30s, 31s, 30s}，μ_1=50s，σ_1 = 2/5，μ_2=30s，σ_2 = 2/5，如果用标准差 σ 定义健壮性，则两个调度的健壮性相同。实际上，调度 1 的完工时间在均值的±2%区间波动，调度 2 的完工时间在均值的±3.33%区间波动，显然调度 1 的健壮性比调度 2 好。CV 衡量实际完工时间相对于平均完工时间 μ 的偏离比例，CV 值越小，意味着实际完工时间偏离 μ 的程度越小，平稳度越好，健壮性越好。按照本文的定义，CV_1=2/250，CV_2=2/150，调度 1 的

健壮性优于调度 2。CV 优于标准差，因为与 μ 无关，且同样易于计算。

3.5 动态调度算法 DEFT

本节详细介绍针对具有异构、性能动态变化、可能存在失效等不确定性因素的不可靠私有云平台上的 DAG 工作流动态调度算法，DEFT (Dynamic Earliest-Finish-Time)算法。

3.5.1 DEFT

DEFT 包括一组调度循环，每当一个任务完成并释放 VM 资源的时候，启动一次调度循环。DEFT 基于启发式方法，即如果在每次调度循环中调度的任务都基于当时的计算和网络速率得到的最小完工时间，那么整个工作流的完工时间也最小。每次调度循环都是基于任务列表的调度，包括三个阶段：任务排序、虚拟机选择、数据传输。

任务排序阶段。在单次调度循环中，未调度的任务首先根据 $rank_u$ 值的大小进行排序，$rank_u$ 值根据执行该次调度循环时虚拟机计算速度和网络带宽进行估算，在时刻 t 任务 T_i 的 $rank_u$ 值，记为 $rank_u(T_i;t)$：

$$rank_u(T_i;t) = \overline{w_i^t} + \max_{T_j \in child(T_i)} [rank_u(T_j;t) + c_{ij}^t] \tag{3-12}$$

其中 $\overline{w_i^t}$ 为在时刻 t 任务 T_i 在私有云上的预计平均计算开销，c_{ij}^t 是在时刻 t 任务 T_i 到子任务 T_j 的预计平均数据传输时间，对于出口任务，$rank_u(T_{exit};t)= \overline{w_{T_{exit}}^t}$。

虚拟机选择阶段。任务首先按照 $rank_u$ 降序排列，$rank_u$ 值高的任务具有更高的优先级，优先选择能够提供最小 EFT 的虚拟机，记为 *bestVM*。如果一个任务的所有父任务都完成，则该任务进入“就绪”状态，在 *bestVM* 可用时，任务调度到 *bestVM* 上执行。剩余任务，包括未进入“就绪”状态的任务和等待 *bestVM* 变得可用的“就绪”任务，只保存(*task, bestVM*)记录。在下一次调度循环中，如果虚拟机可用时间、虚拟机计算速率、网络带宽等资源性能变化导致能够提供一个任务最小 EFT 的虚拟机改变，则对应的(*task, bestVM*)记录也会对应更新。

数据传输阶段。当一个任务 T_i 完成时，需要传输给子任务的中间数据缓存在运行的虚拟机 VM_i 上，并在 *DLTT* 时刻开始传输。*DLTT* 根据公式(3-7)计算，子任务对应的虚拟机存储在(*task, bestVM*)映射记录中。在单次调度循环中，存储记录(*time*, *sourceTask*, *sourceVM*, *destinationTask*, *destinationVM*, *data*)，其中 *time*= $DLTT(T_j, VM_j^t, T_i, VM_i; t)$，数据的源任务 *sourceTask* 是刚完成的任务 T_i，运行在虚拟机 VM_i 上，目标任务 *destinationTask* 是 T_j，当前调度映射的最佳虚拟机是

VM_j^t。

如果在中间数据开始传输后，(*task, bestVM*)的映射变为(*task, bestVM'*)，重新计算新的 *DLTT'*。如果 *DLTT'*早于当前时间，则按照从当前时间开始传输数据的情况，计算新的最早完成时间 *EFT'*，对比 *EFT'*与旧的最早完成时间 *EFT*，若 *EFT'* ＜*EFT*，则意味着虽然在 *DLTT'*时刻没能传输数据，但是如果立刻开始传输数据，也可用获得更小的最早完成时间，则将(*task, bestVM*)和 *DLTT* 记录表中的信息更新为 *bestVM'*和 *DLTT'*；反之，则意味着由于传输时间被推迟， *bestVM'*不能作为最佳虚拟机，保持原有记录。

当一个虚拟机收到数据包，它首先查找(*task, bestVM*)和 *DLTT* 记录表中目的虚拟机为本机的记录，如果收到的数据包目的虚拟机 *destinationVM* 为本机，则数据被缓存给任务使用，否则该数据包被忽略。这保证了在中间数据开始传输后，(*task, bestVM*)映射记录变化后，已经在网络上传输的旧数据不会造成混乱。在资源节点失效可能发生的情况下，任务可能会失败，缓存在源虚拟机 *sourceVM* 上的中间数据、(*task, bestVM*)和 *DLTT* 记录表中的信息，要等到目标任务 *destinationTask* 完成后才能删除。

3.5.2 DEFT 算法描述

DEFT 的算法伪代码如算法 3-1 所示。在数据传输阶段，只需要考虑完成任务的所有未调度的子任务的 *DLTT*，计算 *DLTT* 时用到子任务与其 *bestVM* 的映射，因此在虚拟机选择阶段，只需要为这些子任务选择虚拟机即可，不需要考虑其他任务。用集合 *Scope* 存储单次循环中需要考虑的任务，即所有完成任务的未调度的子任务。*Readys* 集合存储单次调度中所有就绪状态的任务，*Maps* 是虚拟机选择阶段的(*task, bestVM*)调度映射的列表，*Datas* 存储中间传输数据信息(*DLTT*, T_f, VM_f, T_c, $bestVM_c$, *data*)的列表，包括数据的源、目的任务和虚拟机、最晚传输时间 *DLTT*、数据 ID 标识。在初始状态下，入口任务是就绪状态，放在 *Scope* 和 *Ready* 中。算法包括一组调度循环，在有任务完成并释放资源时候，启动单次调度循环。首先，完成任务的所有未调度子任务被添加到 *Scope* (第 3 行)，根据当前计算速度和网络带宽，计算 *Scope* 中所有任务的 $rank_u$ 值(4-5 行)，将所有父节点都完成的任务，即就绪任务添加到 *Readys* 集合中(6-7 行)。随后，*Scope* 中的任务按照 $rank_u$ 值降序排列(第 9 行)，并按需为每个任务选择最佳虚拟机(10-11 行)，新的(*task,bestVM*)记录添加到 *Maps* 列表(12-13 行)，若 *Maps* 和 *Datas* 列表中任务 T_i 对应的 $bestVM_i$ 在当前调度循环中有变化，则更新 *Maps* 和 *Datas* 列表(14-17 行)。对于所有 *Readys* 集合中的任务，如果对应的最佳资源当前可用，则将任务调度到最佳资源上(18-19 行)，并将该任务和最佳资源对应的记录从 *Maps*、*Readys*、*Scope* 集合中删除(第 21 行)。对于每一个完成的任务 T_f，计算从 T_f 到每

一个子任务的数据最晚传输时间，向 *Datas* 中添加相应记录(*DLTT*, T_f, VM_f, T_{child}, $bestVM_{child}$, *data*) (22-25 行)，并将 *Datas* 中 T_f 最为目的节点的记录删除(第 27 行)。此外，在运行期内，按照 *Datas* 内的记录，将中间数据从源节点传输到目标节点(第 32 行)。

算法 3-1 DEFT 算法

输入: 工作流 DAG, *n* 个任务{T_1,T_2,...,T_n}

VM ={VM_1,VM_2,...,VM_m}，云上的 *m* 个虚拟机的集合

Fins，单次调度循环开始前完成的任务集合

Socpe，单次调度循环考虑的任务范围，初始为{$T_{entrance}$}

Readys，就绪状态任务列表，初始为{$T_{entrance}$}

Maps，(*task, bestVM*)调度映射列表

Datas，中间传输数据记录

1: **while** *Socpe* is **not null do**
2: **if** *Fins* is **not null** at time *t* **do**
3: update *Scope*
4: **for** each task $T_i \in Scope$ **do**
5: compute $rank_u(T_i; t)$
6: **if** T_i is ready
7: $Readys \leftarrow T_i$
8: **end for**
9: sort tasks in *Scope* by non-increasing order of $rank_u$
10: **for** each task $T_i \in Scope$ **do**
11: find $bestVM_i$ with minimal $EFT(T_i, bestVM_i; t)$
12: **if** $T_i \notin Maps$
13: $Maps \leftarrow (T_i, bestVM_i)$
14: **else if** $(T_i, bestVM_i) \notin Maps$ **do**
15: update *bestVM* for T_i in *Maps*
16: update *records* whose destination is T_i in *Datas*
17: **end if**
18: **if** $T_i \in Readys$ and $bestVM_i$ is available
19: **schedule** T_i to $bestVM_i$
20: end **for**
21: delete scheduled tasks from *Maps*, *Readys*, *Scope*

续上表

22: **for** each task $T_f \in Fins$ **do**
23: **for** task $T_c \in child(T_f)$ **do**
24: compute $DLTT\ (T_f, bestVM_f, T_c, VM_c; t)$
25: $Datas \leftarrow (DLTT, T_f, VM_f, T_c, bestVM_c, data)$
26: **end for**
27: delete record whose destination is T_f from *Datas*
28: **end for**
29: **end if**
30: empty *Fins*
31: *wait*()
32: send data at time *DLTT* in *Datas*

3.5.3 算法复杂度

将一个有 n 个任务的工作流调度到 IaaS 的 m 个虚拟机上，在单次调度循环内，DEFT 为 *Scope* 内的每一个任务计算 $rank_u$ 值，并选择最佳虚拟机。在任务排序阶段，根据公式(3-12)，通过递归自底向上计算目标任务的 $rank_u$ 值，所有未调度任务都将遍历一遍。在单次调度循环内，对于 k 个未调度任务和 m 个目标虚拟机，任务排序阶段时间复杂度为 $O(k^2*m)$。虚拟机选择和数据传输阶段的时间复杂度为 $O(s*m)$，其中 s 是 *Scope* 中的任务数，且 $s \leqslant k$，单次循环内总的时间复杂度为 $O(k^2*m)$。在最差情况下，每次调度循环只能调度一个任务，则共需要 n 个调度循环，第 l 次调度循环的时间复杂度为 $O((n-l)^2*m)$。调度全部 n 个任务的总的时间复杂度为 $O(1^2*m+2^2*m+...+(n-1)^2*m)$，即 $O(n^3m)$。

3.6 实例分析

本节用一个例子详细解释 DEFT 算法，并与 HEFT [6]、PEFT [12]、DCP-G [20] 对比。图 3-1 和表 3-1 给出了一个简单的实例场景，包括一个包含 10 个任务的科学工作流和 3 个全连接的异构虚拟机。在初始状态下，虚拟机之间的带宽相等，任务之间的通信开销如图 3-1 左图所示，每个任务在每个虚拟机上的计算开销如表 3-1 左侧 4 列所示。在 t=30s 时，系统性能发生变化，VM3 的计算速度和带宽翻倍，各任务在 VM3 上的计算开销如表 1 最右列所示，若父任务和子任务分别调度在 VM3 和其他 VM 上，则任务之间的传输开销如图 3-1 右图所示。

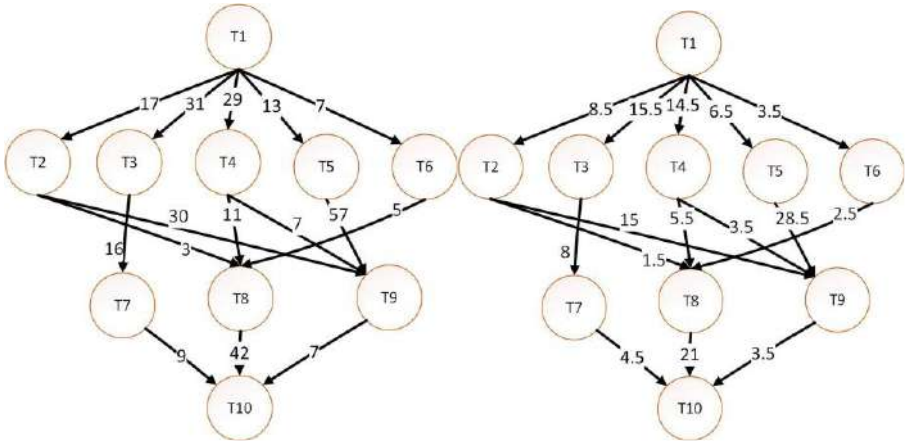


图 3-1 任务之间传输开销，t=0s（左），t=30s（右）

表 3- 1 各任务在各虚拟机上的计算开销

	w_{i1}	w_{i2}	w_{i3}(t=0-30s)	w_{i3}(t=30s-)
T1	22	21	36	18
T2	22	18	18	9
T3	32	27	43	21.5
T4	7	10	4	2
T5	29	27	35	17.5
T6	26	17	24	12
T7	14	25	30	15
T8	29	23	36	18
T9	15	21	8	4
T10	13	16	33	16.5

1）HEFT

HEFT 是经典的静态调度算法，在工作流提交时给出调度映射，并将任务调度到相应的计算节点上排队等待运行。HEFT 根据任务提交时的系统性能调度，按 $rank_u$ 值进行任务降序排列，并依次分配提供最小 EFT 的计算节点。在本场景中，使用 t=0s 时的计算开销和传输开销。

(1)任务排序阶段

各任务的平均计算开销为：{26.33, 19.33, 34, 7, 30.33, 23.33, 23, 29.33, 14.67, 20.67}

各任务的 $rank_u$ 为：{169, 114.33, 102.67, 110, 129.7, 119.3, 52.7, 92, 42.3, 20.7}

任务排序为：T1, T5, T6, T2, T4, T3, T8, T7, T9, T10

(2)虚拟机选择阶段，按序计算各任务在各虚拟机上的最早完成时间 EFT，并选择调度到 EFT 最早的虚拟机上，如表 3-2 所示。

各任务对应的最优虚拟机映射为：VM1←{T2, T8, T10}，VM2←{T1, T5, T3, T7}，VM3←{T6, T4, T9}，预计完工时间为 122s。

表 3-2 EFT_{ij} 及调度映射

	EFT_{i1}	EFT_{i2}	EFT_{i3}	VM 及起止时间
T1	22	21	36	VM2(0-21s)
T5	73	48	69	VM2(21-48s)
T6	54	71	52	VM3(28-52s)
T2	60	66	70	VM1(38-60s)
T4	67	58	56	VM3(52-56s)
T3	92	75	99	VM2(48-75s)
T8	89	98	99	VM1(60-89s)
T7	105	100	121	VM2(75-100s)
T9	120	121	113	VM3(105-113s)
T10	133	136	164	VM1(120-133s)

t=30s 时资源性能改变，但是任务调度映射不变，最终完工时间为 122s，任务的调度和执行时间如图 3-2 所示。

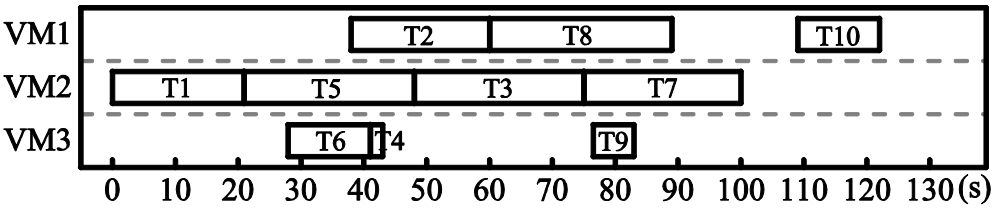


图 3-2 HEFT 算法调度结果

2）PEFT

PEFT 是目前完工时间最短的静态调度算法，且算法复杂度与 HEFT 相同。在工作流提交时给出调度映射，并将任务调度到相应的计算节点上排队等待运行。PEFT 在任务排序阶段使用 OCT (Optimistic Cost Table)，任务 T_i 在虚拟机 VM_j 上的 OCT 值为：

$$OCT(T_i,VM_j)=\max_{T_k\in child(T_i)}\{\min_{VM_p\in VM}[OCT(T_k,VM_p)+w_{kp}+c_{ik}]\} \tag{3-13}$$

其中，w_{kp} 是任务 T_k 运行在虚拟机 VM_p 上的计算开销，c_{ik} 是任务 T_i 到任务 T_k 的运行开销。出口任务在任意虚拟机上的 *OCT* 值都为 0，即 $OCT(T_{exit},VM)$=0。若 T_i 与 T_k 在同一个虚拟机上，或者 VM_j 与 VM_p 在同一主机上，c_{ik}=0。任务根据在各虚拟机上的 *OCT* 平均值降序排列选择虚拟机。在虚拟机选择阶段，T_i 映射到 VM_j 上，使得 $O_{EFT}(T_i,VM_j)$最小，$O_{EFT}(T_i,VM_j)$定义为：

$$O_{EFT}(T_i,VM_j)=EFT(T_i,VM_j)+OCT(T_i,VM_j) \tag{3-14}$$

在本场景中，使用 *t*=0s 时的计算开销和传输开销。

各任务的 *OCT* 值如表 3-3 所示：

表 3-3 各任务的 *OCT* 值

	OCT_{i1}	OCT_{i2}	OCT_{i3}	*avgOCT*
T1	64	68	86	72.67
T2	42	39	42	41
T3	27	41	43	37
T4	42	39	50	43.67
T5	28	37	28	31
T6	42	39	44	41.67
T7	13	16	22	16.67
T8	13	16	33	20.67
T9	13	16	20	16.33
T10	0	0	0	0

排序为：T1, T4, T6, T2, T3, T5, T8, T7, T9, T10。各任务按序计算在各虚拟机上的 O_{EFT} 值，如表 3-4 所示。

表 3-4 $O_{EFF}(T_i,VM_j)$值及调度映射

	$O_{EFT}(T_i,VM_1)$	$O_{EFT}(T_i,VM_2)$	$O_{EFT}(T_i,VM_3)$	VM 及起止时间
T1	86	89	122	VM1(0-22s)
T4	71	100	105	VM1(22-29s)
T6	97	85	97	VM2(29-46s)
T2	93	103	99	VM1(29-51s)
T3	110	121	139	VM1(51-83s)
T5	140	110	98	VM3(35-70s)
T8	125	93	139	VM2(54-77s)
T7	110	140	151	VM1(83-97s)
T9	155	164	109	VM3(81-89s)
T10	132	122	152	VM2(106-122s)

各任务对应的最优虚拟机映射为：VM1←{T1, T2, T4, T3, T7}，VM2←{T6, T8, T10}，VM3←{T5, T9}

t=30s 时资源性能改变，但是任务调度映射不变，最终完工时间为 122s，任务的调度和执行时间如图 3-3 所示。

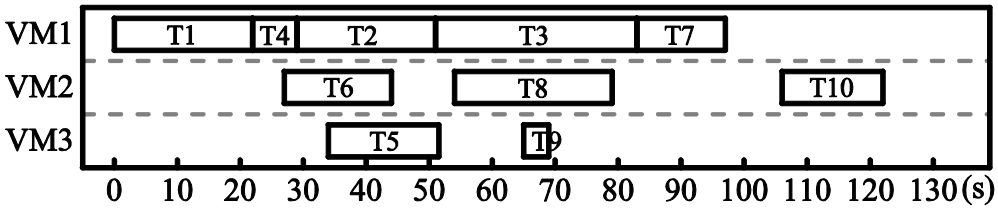


图 3-3 PEFT 算法调度结果

DCP-G 在工作流提交时，进行调度，运行期检测到资源性能变化时，进行重

调度。首先根据任务在系统上的最小计算开销和最小传输开销，检索 DAG 工作流的关键路径，在单步调度中选择关键路径上的“就绪”状态任务，即所有父节点都调度完的任务，并选择该任务的关键子任务，即最晚开始时间与最早开始时间差值最小的子任务，选择为关键子任务提供最小 EST 的计算节点作为任务的最佳节点，若任务 T_i 调度到虚拟机 VM_j 上运行，其子任务 T_k 的最早开始时间 $EST^{DCP\text{-}G}$ 为：

$$EST^{DCP-G}(T_i,T_k;VM_j)=FT_{ij}+D_{ik}/\overline{B_j} \tag{3-15}$$

其中，FT_{ij} 是任务 T_i 在 VM_j 上的完成时间，D_{ik} 是 T_i 传输给子任务 T_k 的中间数据量，$\overline{B_j}$ 是 VM_j 与其他虚拟机之间的平均带宽。在任务调度到该节点上后，该任务的计算开销确定，则重新计算 DAG 的关键路径，进行下一次的单步调度。

t=0 时，各任务的最小计算开销为：{21, 18, 27, 4, 27, 17, 14, 23, 8, 13}，任务间的最小传输开销为图 3-1 左图。

- Step1：关键路径为{T1,T5,T9,T10}，关键路径的长度为 146；就绪任务为 T1，T1 调度到 VM2 上，预计起止时间为 0-21s；
- Step2：关键路径为{T1,T5,T9,T10}，就绪任务为 T5，关键子任务为 T9；EFT_{51}=63s，EFT_{52}=48s，EFT_{53}=69s，EST_{91}=120s，EST_{92}=105s，EST_{93}=126s，T5 调度到 VM2，预计运行起止时间是 21-48s；
- Step3：关键路径为{T1,T4,T8,T10}，关键路径长度为 143；就绪任务为 T4，关键子任务为 T8，EFT_{41}=57s，EFT_{42}=59s，EFT_{43}=54s，EST_{81}=68s，EST_{82}=70s，EST_{83}=65s，调度到 VM3，预计运行起止时间是 50-54s；
- Step4：关键路径为{T1,T4,T8,T10}，关键路径长度为 143，关键路径上没有就绪任务，已调度任务的关键任务是 T2，T2 的关键子任务是 T8，EFT_{21}=60s，EFT_{22}=66s，EFT_{23}=72s，EST_{81}=63s，EST_{82}=69s，EST_{83}=75s，T2 调度到 VM1，预计运行的起止时间为 38-60s；
- Step5：关键路径为{T1,T4,T8,T10}，关键路径长度为 143，关键路径上没有就绪任务，已调度任务的关键子任务是 T3，T3 的关键子任务是 T7，EFT_{31}=92s，EFT_{32}=75s，EFT_{33}=97s，EST_{71}=108s，EST_{72}=91s，EST_{73}=113s，T3 调度到 VM2，预计运行的起止时间为 48-75s；
- Step6：关键路径为{T1,T4,T8,T10}，关键路径长度为 143，关键路径上没有就绪任务，已调度任务的关键子任务是 T6，T6 的关键子任务是 T8，EFT_{61}=110s，EFT_{62}=83s，EFT_{63}=78s，EST_{81}=115s，EST_{82}=88s，EST_{83}=83s，T6 调度到 VM3，预计运行的起止时间为 54-78s；
- Step7：关键路径为{T1,T4,T8,T10}，关键路径长度为 143，关键路径上的

就绪任务是T8，关键子任务是T10，EFT_{81}=112s，EFT_{82}=106s，EFT_{83}=114s，$EST_{10,1}$=154s，$EST_{10,2}$=148s，$EST_{10,3}$=154s，T8 调度到 VM2，预计运行的起止时间为 83-106s；

- Step8：关键路径为{T1,T4,T8,T10}，关键路径长度为 143，关键路径上没有就绪任务，已调度任务的关键子任务是 T9，T9 的关键子任务是 T10，EFT_{91}=120s，EFT_{92}=127s，EFT_{93}=98s，$EST_{10,1}$=127s，$EST_{10,2}$=134s，$EST_{10,3}$=105s，T9 调度到 VM3，预计运行的起止时间为 90-98s；
- Step9：关键路径为{T1,T4,T8,T10}，关键路径长度为 143，关键路径上没有就绪任务，已调度任务的关键子任务是 T7，T7 的关键子任务是 T10，EFT_{71}=105s，EFT_{72}=131s，EFT_{73}=128s，$EST_{10,1}$=114s，$EST_{10,2}$=140s，$EST_{10,3}$=137s，T7 调度到 VM1，预计运行的起止时间为 91-105s。
- Step10：就绪任务是 T10，$EFT_{10,1}$=161s，$EFT_{10,2}$=130s，$EFT_{10,3}$=181s，T10 调度到 VM2，预计运行的起止时间为 114-130s。

在 t=30s 时，系统性能发生变化，VM3 的计算和传输开销降低，除 T1 已经运行完成，T5 正在运行外，其他任务需要重新调度，T5 预计完成时间为 t=48s。剩余任务的最小计算开销为{21.5, 2, 17.5, 14, 18, 4, 13}，最小传输开销为图 3-1 右图。

- Step11：重新寻找关键路径，为{T1,T3,T7,T10}，关键路径的长度为 97.5；就绪任务为 T3，关键子任务是 T7；T3 原本调度在 VM2 上，T1 到 T3 的中间数据没有传输，因此计算在 VM1 和 VM3 上的 EFT 时，数据需要从 t=30s 开始传输，EFT_{31}=93s，EFT_{32}=75s，EFT_{33}=67s，EST_{71}=109s，EST_{72}=91s，EST_{73}=75s，T3 调度到 VM3，预计运行的起止时间为 45.5-67s；
- Step12：关键路径为{T1,T3,T7,T10}，关键路径的长度为 97.5，就绪任务为 T7，关键子任务是 T10，EFT_{71}=89s，EFT_{72}=100s，EFT_{73}=82s，$EST_{10,1}$=98s，$EST_{10,2}$=109s，$EST_{10,3}$=86.5s，T7 调度到 VM3，预计运行的起止时间为 67-86.5s。
- Step13：关键路径为{T1,T5,T9,T10}，关键路径长度为 97，关键路径上没有就绪任务，已调度任务的关键子任务是 T4，T4 的关键子任务为 T8，T4 本来调度在 VM3 上，T1 数据到达 VM3 的时间是 t=40s，到达 VM1 的时间是 t=30+29=59s，EFT_{41}=66s，EFT_{42}=58s，EFT_{43}=42s，EST_{81}=77s，EST_{82}=69s，EST_{83}=47.5s，调度到 VM3，预计运行起止时间是 40-42s；
- Step14：关键路径为{T1,T5,T9,T10}，关键路径长度为 97，关键路径上没有就绪任务，已调度任务的关键子任务是 T2，T2 的关键子任务是 T8，T2 本来调度在 VM1 上，T1 的数据到达 VM1 的时间是 t=38s，到达 VM3 的时间

是 t=30+8.5=38.5s，EFT_{21}=60s，EFT_{22}=66s，EFT_{23}=95.5s，EST_{81}=63s，EST_{82}=69s，EST_{83}=97s，T2 调度到 VM1，预计运行的起止时间为 38-60s；

- Step15：关键路径为{T1,T5,T9,T10}，关键路径长度为 97，就绪任务是 T9，T9 的关键子任务是 T10，EFT_{91}=120s，EFT_{92}=111s，EFT_{93}=90.5s，$EST_{10,1}$=127s，$EST_{10,2}$=118s，$EST_{10,3}$=94s，T9 调度到 VM3，预计运行的起止时间为 86.5-90.5s；
- Step16：关键路径为{T1,T5,T9,T10}，关键路径长度为 97，关键路径上没有就绪任务，已调度任务的关键子任务是 T6，T6 的关键子任务是 T8，T6 原来调度在 VM3 上，T1 的数据到达 VM3 的时间是 t=28s，到达 VM1 的时间是 t=30+7=37s，EFT_{61}=86s，EFT_{62}=65s，EFT_{63}=40s，EST_{81}=91s，EST_{82}=70s，EST_{83}=42.5s，T6 调度到 VM3，预计运行的起止时间为 28-40s；
- Step17：关键路径为{T1,T5,T9,T10}，关键路径长度为 97，关键路径上没有就绪任务，已调度任务的关键子任务是 T8，T8 的关键子任务是 T10，EFT_{81}=89s，EFT_{82}=86s，EFT_{83}=108.5s，$EST_{10,1}$=131s，$EST_{10,2}$=126s，$EST_{10,3}$=129.5s，T8 调度到 VM2，预计运行的起止时间为 63-86s；
- Step18：就绪任务是 T10，$EFT_{10,1}$=139s，$EFT_{10,2}$=120s，$EFT_{10,3}$=123.5s，调度到 VM2，预计运行的起止时间为 94-120s。

DCP-G 算法的最终调度结果如图 3-4 所示，完工时间为 120s。

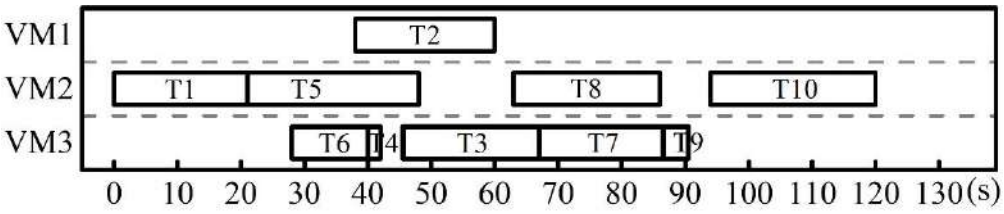


图 3-4 DCP-G 算法调度结果

DEFT 在运行时动态调度，每当有任务完成释放资源时，进行一次调度循环，具体步骤如下：

- Step1：起始时间 t=0s 时，*Scope*={T1} *Ready*={T1}，对于 T1，EFT_{11}=22s，EFT_{12}=21s，EFT_{13}=36s，T1 调度到 VM2，预计运行的起止时间为 0-21s；
- Step2：t=21s 时，T1 完成，*Scope*={T5,T6,T2,T4,T3} *Ready*={T5,T6,T2,T4,T3}，对于 T5，EFT_{51}=73s，EFT_{52}=48s，EFT_{53}=69s，T5 调度到 VM2，预计运行的起止时间为 21-48s；对于 T6，EFT_{61}=73s，EFT_{62}=71s，EFT_{63}=52s，T6 调度到 VM3，预计运行的起止时间为 28-52s；对于 T2，EFT_{21}=60s，EFT_{22}=66s，EFT_{23}=70s，T2 调度到 VM1，预计运行的起止时间为 38-60s；对于 T4，EFT_{41}=67s，EFT_{42}=58s，EFT_{43}=56s，T4 调度到 VM3，预计运行的起止时间

为 52-56s；对于 T3，EFT_{31}=92s，EFT_{32}=75s，EFT_{33}=99s，T3 调度到 VM2，预计运行的起止时间为 48-75s；

- Step3：t=41s 时，T6 完成，Scope={T8} Ready=null，对于 T8，EFT_{81}=88s，EFT_{82}=98s，EFT_{83}=78.5s，由于 T8 未进入“就绪”状态，只保存到 VM3 的映射，预计运行的起止时间为 60.5-78.5s；
- Step3：t=43s 时，T4 完成，Scope={T8,T9} Ready=null，对于 T8，与 VM3 的映射不变；对于 T9，EFT_{91}=120s，EFT_{92}=110s，EFT_{93}=82.5s，由于 T9 未进入“就绪”状态，只保存到 VM3 的映射，预计运行的起止时间为 78.5-82.5s；
- Step4：t=48s 时，T5 完成，Scope={ T9} Ready=null，对于 T9，与 VM3 的映射不变；
- Step5：t=59s 时，T2 完成，Scope={T8,T9} Ready={T8,T9}，对于 T8，T9，最佳虚拟机的映射不变，调度，T8 的预计运行的起止时间为 60.5-78.5s，T9 的预计运行的起止时间为 78.5-82.5s；
- Step6：t=75s 时，T3 完成，Scope={T7} Ready={T7}，对于 T7，EFT_{71}=105s，EFT_{72}=100s，EFT_{73}=98s，T7 调度到 VM3，预计运行的起止时间为 83-98s；
- Step7：t=78.5s 时，T8 完成，Scope={T10} Ready=null，对于 T10，$EFT_{10,1}$=,120s，$EFT_{10,2}$=118.5s，$EFT_{10,3}$=114.5s，由于 T10 未进入“就绪”状态，只保存到 VM3 的映射，预计运行的起止时间为 98-114.5s；
- Step8：t=82.5s 时，T9 完成，Scope={ T10} Ready=null，对于 T10，与 VM3 的映射不变；
- Step9：t=98s 时，T7 完成，Scope={T10} Ready={T10}，对于 T10，最佳虚拟机的映射不变，调度，预计运行的起止时间为 98-114.5s。

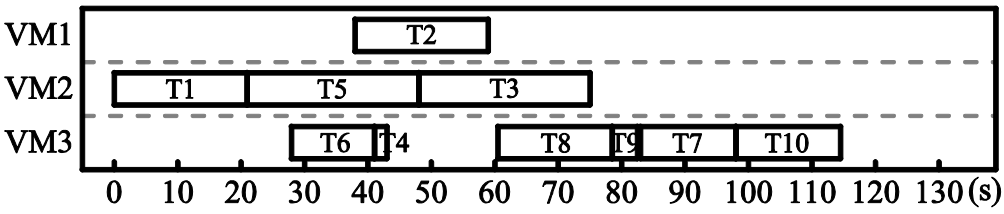


图 3-5 DEFT 算法调度结果

DEFT 算法的最终调度结果如图 3-5 所示，完工时间为 114.5s

3.7 仿真实验

本节仿真具有异构、动态、失效等不确定性因素的不可靠云，对比 DEFT 与 PEFT、DCP-G 算法的完工时间和健壮性指标。

3.7.1 实验环境描述

本节使用 DynamicCloudSim [25]仿真工具仿真不可靠云，DynamicCloudSim 基于 CloudSim 工具集，同时提供了：1）资源的异构性，新分配的虚拟机 CPU、I/O、带宽等性能从一个正太分布的函数中采样，RSD (Relative Standard Deviation, 相对标准偏差)可以自定义；2）性能的动态变化，CPU、I/O、带宽速率下次变化的时间间隔从指数函数中采样，变化后的值从正太分布中采样，指数分布的速率参数和正太分布的 RSD 参数越高，表明性能变化频率和幅度越大；3）拖后腿主机和任务运行失败事件，可以定义一个虚拟机拖后腿的概率和性能下降的比例，任务在调度到虚拟机上时为任务分配一个自定义的平均失败概率，任务失败的原因包括节点失效或故障等。

云平台的服务器按照 DynamicCloudSim 的默认设置，包括 100 台 Intel Xeon E5430 2.66GHz、200 台 AMD Opteron270 2GHz、200 台 AMD Opteron 2218HE 2.6GHz，服务器之间的网络全连接。100 台虚拟机随机分配在服务器上，I/O 为 20 MB/s，虚拟机外部带宽为 0.25 MB/s，网络延迟忽略不计，CPU、I/O、带宽的异构性正太分布的 RSD 分别为 0.4、0.15、0.2。虚拟机默认的拖后腿概率是 0.015, 性能下降率为 0.5，默认的任务失败概率是 0.002。

系统的不确定性用两个参数表示：1）性能变化次数，在工作流运行过程中，系统性能变化的次数，由于性能下次变化的时间间隔从指数函数中采样，不同调度下工作流的完工时间不一致，所以工作流运行过程中没有一个确定的变化次数，本实验用范围表示，{1-10 次, 10-20 次, 20-30 次, 30-40 次}；2）不确定度，分为低不确定性和高不确定性，低不确定性的动态 RSD 为默认值 CPU 0.054、I/O 0.033、带宽 0.04，虚拟机默认的拖后腿概率是 0.015, 性能下降率为 0.5，默认的任务失败概率是 0.002；高不确定性 CPU、I/O 和带宽的 RSD 均为 0.5，虚拟机默认的拖后腿概率是 0.05, 性能下降率为 1，默认的任务失败概率是 0.05。实验中不确定性参数简写为：{<10,*l*>, <10,*h*>, <20,*l*>, <20,*h*>, <30,*l*>, <30,*h*>, <40,*l*>, <40,*h*>}。

本实验使用美国地震中心从表征区域中提取地震灾害特征的 Montage 工作流，该工作流被广泛应用在科学工作流任务调度、资源分配等研究中。工作流的 DAG 结构，包括任务之间的依赖关系、运行时间信息、任务的输入输出文件等，都使用 Pegasus DAX(Directed Acyclic Graph in XML)[26]描述。工作流规模从 50 到 1000 个任务不等，WorkflowGenerator[27]为每个规模的工作流生成 20 个实例。本实验将各个工作流在实验平台上分别调度运行 100 次，计算平均完工时间、NSL、CV。

3.7.2 实验结果分析

实验对比分析了 PEFT、DCP-G、DEFT 算法在不可靠云上的完工时间和健壮性。图 3-6 显示了云平台默认配置下、包含 1000 个任务的 20 个不同工作流的平均 NSL 和 CV。所有的工作流使用 DEFT 都比使用 PEFT 调度产生的完工时间短，19 个工作流使用 DEFT 比使用 DCP-G 调度产生的完工时间短。19 个工作流使用 DEFT 比使用 PEFT 调度的健壮性更好，18 个工作流使用 DEFT 比使用 DCP-G 调度的健壮性更好。

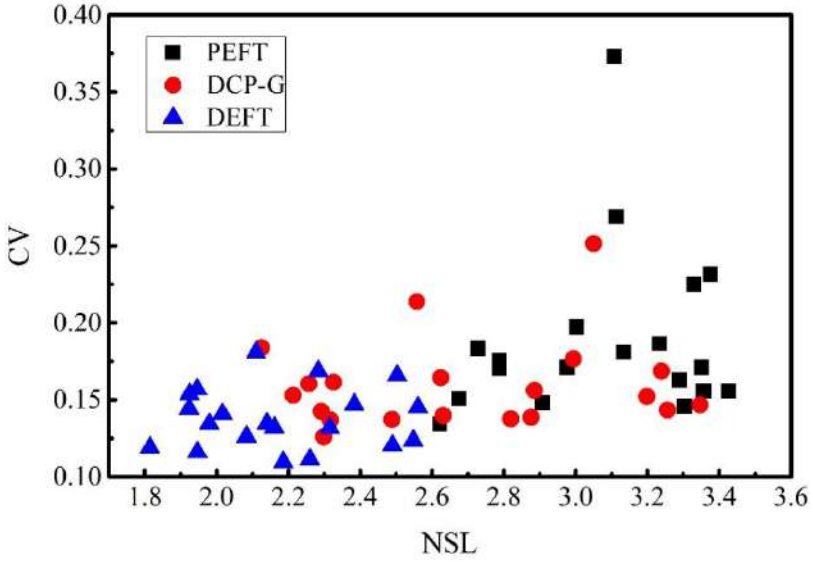


图 3-6 默认配置下的平均 NSL 和 CV

图 3-7 和图 3-8 显示了不同的不确定性参数下的完工时间和健壮性。工作流从图 3-6 的实验结果中选择三个算法产生的 CV 差值最小的那个工作流，静态确定环境下，PEFT、DCP-G、DEFT 调度该工作流的对应的 NSL 分别为 2.56、2.47、1.86。

图 3-7 显示了不同的不确定性参数下三种调度算法完工时间的箱线图，分别标识最小值、25 分位、平均值、75 分位、最大值。DEFT 一直可以获得最小的 NSL。性能变化次数小于 10 且不确定度低的情况下，PEFT、DCP-G、DEFT 的平均 NSL 分别为 2.83、2.52、1.82，性能变化次数大于 30 且不确定度低的情况下，三种算法的平均 NSL 分别为 2.79、2.47、1.95。在低不确定度环境中，变化频率低时工作流使用 DEFT 调度比使用 PEFT 和 DCP-G 调度的平均完工时间分别快 35.69% 和 27.78%，变化频率高时分别快 30.11%和 21.05%。性能变化次数小于 10 且不确定度高的情况下，PEFT、DCP-G、DEFT 的平均 NSL 分别为 5.04、3.91、2.92,，性能变化次数大于 30 且不确定度低的情况下，三种算法的 NSL 分别为 4.06、3.65、2.90。在高不确定度环境中，变化频率低时工作流使用 DEFT 调度比使用 PEFT 和 DCP-G 调度的平均完工时间分别快 42.06%和 32.89%，变化频

率高时分别快 28.57%和 20.55%。

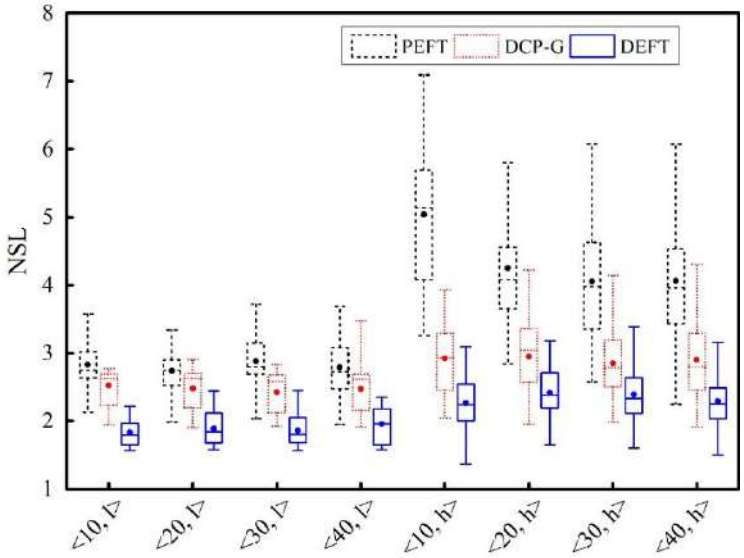


图 3-7 不同不确定性参数下工作流的完工时间

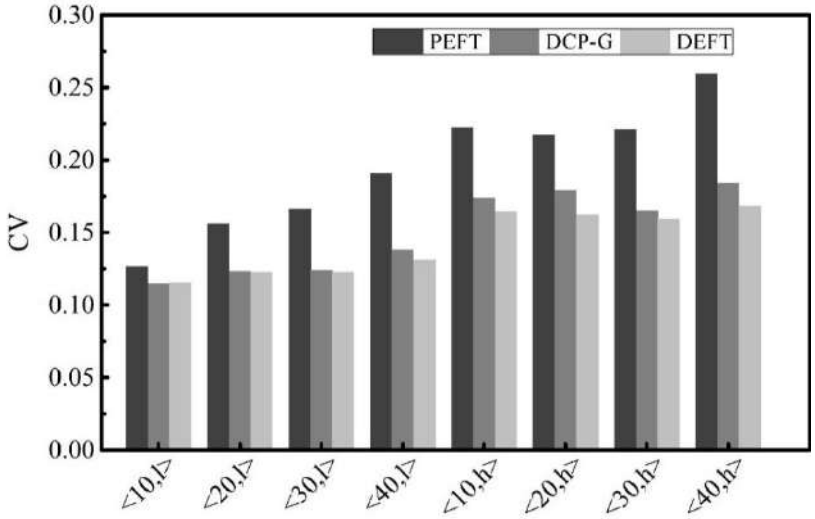


图 3-8 不同不确定性参数下的调度健壮性

图 3-8 显示了不同的不确定性参数下三种调度算法的健壮性柱状图。DCP-G 和 DEFT 的健壮性接近，因为二者都是动态调度，可以及时感知运行中资源性能的变化。不确定度低时，PEFT、DCP- G、DEFT 的 CV 分别为 0.16、0.124、0.123，不确定度高时分别为 0.23、0.176、0.164。性能变化次数小于 10 且不确定度低的情况下 DEFT 与 DCP-G 的 CV 比 PEFT 分别高 9.36%和 8.86%，性能变化次数大于 30 且不确定度高的情况下 DEFT 与 DCP-G 的 CV 比 PEFT 分别高 29.05%和 35.03%。

工作流执行过程中，为任务选择资源操作消耗的时间算是额外开销，如果额外开销过大，会影响工作流执行的性能。图 3-9 显示了云平台默认配置下，三种算法对于不同规模的任务的平均调度时间。小规模工作流（小于 300 个任务）情

况下，三种算法的调度时间相近，即 DEFT 的开销接近于静态调度算法。随着工作流规模的增大，动态调度算法 DCP-G 和 DEFT 的调度时间呈线性增加，而静态调度算法 PEFT 基本不变。另外，拥有 1000 个任务的工作流平均完工时间约 7000s，而 DEFT 的调度时间为 9s，与完工时间相比，调度的额外开销可以忽略不计。

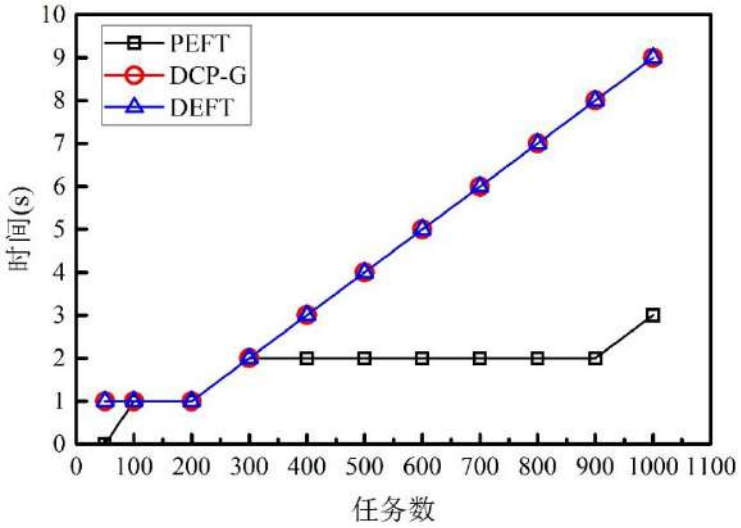


图 3-9 不同规模工作流的调度时间开销

3.8 本章小结

针对混合了资源性能异构、动态变化、有失效可能等不确定因素的不可靠云平台，本章提出了动态工作流调度算法 DEFT，在完工时间最小的同时，具有健壮性。任务只在所有父任务都完成之后，才进入就绪状态，根据当前系统的性能选择资源进行调度。DEFT 包括一组调度循环，每次有任务完成且有空闲资源时，进行一次调度操作。在每个调度操作中，所有未调度的任务根据当前资源速度，计算 $rank_u$ 值，并依据 $rank_u$ 降序排列，每个未调度任务按序映射到提供最小 EFT 的虚拟机资源，就绪状态的任务可以调度到相应的虚拟机，而其他任务只保留到虚拟机的映射关系，并传输中间数据。本章用变异系数 CV 计算调度的健壮性，用以衡量不可靠云环境下调度机制能否产生平稳的、最小的完工时间。实验表明，使用 DEFT 比现有的典型的基于任务列表的静态调度算法 PEFT 和动态调度算法 DCP-G，工作流完工时间缩短超过 20%，且健壮性更好。

参考文献

[1] Fakhfakh F, Kacem H H, Kacem A H. Workflow scheduling in coud computing: a survey [A]. //Proceedings of IEEE 18th International Enterprise Distributed

Object Computing Conference Workshops and Demonstrations [C], Washington DC: IEEE Press, 2014: 372－378.

[2] Farley B, Venkatanathan V, Bowers K D. More for your money: exploiting performance heterogeneity in public clouds [A]. // Proceedings of the Third ACM Symposium on Cloud Computing [C], San Jose: ACM Press, 2012: 1-14.

[3] Dejun G P J , Chi C H. EC2: performance analysis for resource provisioning of service-oriented applications [A]. // Proceedings of the 2009 international conference on Service-oriented computing [C], Stockholm: Springer Press, 2009: 197-207.

[4] Alkhanaka E N, Leea S P , Rezaeia R, et al. Cost optimization approaches for scientific workflow scheduling in cloud and Grid computing: a review, classifications, and open issues [J]. Journal of Systems and Software, 2016, 113 (1): 1-26.

[5] Amalarethinam D I G, Josphin A M. Dynamic task scheduling methods in heterogeneous systems: a survey [J]. International Journal of Computer Applications, 2015, 110 (6):12-18.

[6] Topcuoglu H, Hariri S, Wu M-Y. Performance-effective and low-complexity task scheduling for heterogeneous computing [J], IEEE Transactions on Parallel and Distributed Systems, 2002, 13 (3): 260-274.

[7] Ilavarasan E, Thambidurai P. Low complexity performance effective task scheduling algorithm for heterogeneous computing environments [J]. Journal of Computer Sciences, 2007, 3 (2): 94-103.

[8] Hagras T, Janecet J. A simple scheduling heuristic for heterogeneous computing environments [A]. //Proceedings of the Second international conference on Parallel and distributed computing, LjubljanaL IEEE Press, 2003: 104-110.

[9] Ilavarasan E, Thambidurai P, Mahilmannan R. High performance task scheduling algorithm for heterogeneous computing system [A]. //Proceedings of International Conference on Algorithms and Architectures for Parallel Processing: Distributed and Parallel Computing [C], Berlin: Springer Press, 2005: 193-203.

[10] Bittencourt L F, Sakellariou R, Madeira E R M. DAG scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm [A]. // Proceedings of the 18th Euromicro Conference on Parallel, Distributed and Network-based Processing [C], Washington DC: IEEE Press, 2010: 27-34.

[11] Munir E U, Mohsin S, Hussain A, et al. SDBATS: a novel algorithm for task

scheduling in heterogeneous computing systems [A]. // Proceedings of IEEE 27th International Symposium on Parallel and Distributed Processing Workshops and PhD Forum [C], Boston: IEEE Press, 2013:43-53.

[12] Arabnejadand H, Barbosa J G. List scheduling algorithm for heterogeneous systems by an optimistic cost table [J]. IEEE Transactions on Parallel and Distributed Systems, 2014, 25 (3): 682-694.

[13] Rodriguezand M A, Buyya R, Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds [J]. IEEE Transactions on Cloud Computing, 2014, 2 (2): 222-235.

[14] Tang X, Li K, Liao G, et al. A stochastic scheduling algorithm for precedence constrained tasks on Grid [J], Future Generation of Computing System, 2011, 27 (8): 1083-1091.

[15] Poola D, Garg S K, Buyya R, et al. Robust scheduling of scientific workflows with deadline and budget constraints in clouds [A]. //Proceedings of IEEE 28th International Conference on Advanced Information Networking and Applications [C], Victoria: IEEE Press, 2014: 858-865.

[16] Zheng W, Sakellariou R. A Monte-Carlo approach for full-ahead stochastic DAG scheduling [A]. //Proceedings of the 21st Heterogeneous Computing Workshop [C], Washington DC: IEEE Press, 2012: 99-112.

[17] Zheng W, Sakellariou R. Stochastic DAG scheduling using a Monte Carlo approach [J]. Journal of Parallel and Distributed Computing, 2013, 73 (12): 1673-1689.

[18] Zheng Q. Dynamic adaptation of DAGs with uncertain execution times in heterogeneous computing systems [A]. //Proceedings of IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum [C], Atlanta: IEEE Press, 2010: 1-8.

[19] Barbosa J G, Moreira B. Dynamic scheduling of a batch of parallel task jobs on heterogeneous clusters [J]. Parallel Computing, 2011, 37 (8): 428-438.

[20] Rahman M, Hassan R, Ranjan R, et al. Adaptive workflow scheduling for dynamic Grid and cloud computing environment [J]. Concurrency and Computation: Practice and Experience, vol.25, 2013, 13 (1): 1816-1842.

Benlic U, Hao J-K. An effective multilevel tabu search approach for balanced graph partitioning[J]. Computers and Operations Research, 2011, 38 (7): 1066-1075.

[21] Boloni L, Marinescu D C. Robust scheduling of meta-programs [J]. Journal of

Scheduling, 2002, 5 (5): 395-412.

[22] Shi Z, Jeannot E, Dongarra J. Robust task scheduling in non-deterministic heterogeneous computing systems [A]. //Proceedings of IEEE International Conference on Cluster Computing [C], Barcelona: IEEE Press, 2006:1-10.

[23] Canon L-C, Jeannot E. Evaluation and optimization of the robustness of DAG schedules in heterogeneous environments [J]. IEEE Transactions on Parallel and Distributed Systems, 2010, 21 (4): 532-546.

[24] Shestak V, Smith J, Siegel HJ, et al. A stochastic approach to measuring the robustness of resource allocations in distributed systems [A]. //Porceedings of International Conference of Parallel Processing [C], Ohio: IEEE Press, 2006: 459-470.

[25] Bux M, Leser U. DynamicCloudSim: simulating heterogeneity in computational clouds [J]. Future Generation Computer Systems, 2015, 46: 85-99.

[26] Pegasus, WorkflowGenerator, https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator, 2017-04-01.

第四章 混合云环境下数据密集型科学工作流动态调度算法

4.1 引言

混合云确保用户可以在私有云资源不足时租用公有云，来获得更多的计算和存储能力，拥有了弹性扩展能力[1]。如今，工作流不止计算规模增长，数据规模也在大幅增长，电子商务、互联网应用、科学计算等领域出现了大量数据密集型应用，工作流运行中数据通信时间占据不可忽视的重要部分。例如，地震分析工作流 CyberShake 超过 97%的时间都在运行 SeismogramSynthesis 任务，每个 SeismogramSynthesis 任务平均读取 547.16MB 数据[2]。因此，对于数据密集型科学工作流，决定哪些任务调度到公有云哪些任务调度到私有云，数据传输时间是需要考虑的重要因素。

现有科学工作流在混合云上的调度通常只针对普通科学工作流，关注于减小费用开销，并没有关注数据密集型科学工作流可能产生的数据传输量过大的问题。而对于数据密集型科学工作流的调度，或者是在私有云平台上，或者只针对多个公有云平台，并没有考虑减小费用开销的问题。本章关注将持续提交的数据密集型科学工作流调度到混合云上，工作流在截止时间之前完成的情况下，解决最小费用开销和最小云之间的数据传输量的双重目标。

4.2 相关研究工作

对于云平台上的任务调度，费用开销是一个重要的目标约束，目前以节约费用开销为目标的混合云调度，通常优先将工作流调度到免费的私有云上，若空间不足，再选择任务/工作流调度到公有云上。对于数据密集型工作流，不论是在私有云还是在公有云上，数据传输量都是需要考虑的重要因素。

4.2.1 混合云环境下的调度算法

1. 减小费用开销的单目标调度

对于以减小费用开销为目标的单目标调度问题，现有的混合云环境下的调度方法通常关注于在工作流的截止时间约束下减小开销，优先把工作流部署在免费的私有云上，当工作流不能满足截止时间约束时，选择单个任务/工作流调度到公有云上。调度的两个关键步骤分别是选择调度到公有云的任务/工作流、选择合适的公有云实例。

静态调度方法在静态环境下，针对单个工作流，预测工作流完工时间，如果

工作流超时，则选择任务调度到公有云。文献[3]关注如何保证网格环境下的任务完成时间，当任务可能超时的时候，选择任务调度到 Amazon EC2 的竞价型实例 Spot Instance 上，使用改进的动态关键路径算法来确定云实例的类型、数量、启动时间。HCOC（Hybrid Cloud Optimized Cost）[1]算法首先使用 PCH（Path Clustering Heuristic）[4]算法将工作流调度到私有云上，估算工作流完成时间，如果超过截止时间，则选择优先级最高（该任务及所有后继任务的完成时间之和最大）的任务、或预计完成时间最晚的任务，调度到能够确保工作流在截止时间之前完成的、最低价公有云实例上。文献[5]对工作流分层，按层调度任务，并按层等比例划分截止时间为“子截止时间”，估算各层任务的完成时间，如果超过子截止时间，就调度到公有云上。

动态调度算法是在运行期动态调度工作流，在预测工作流会超出截止时间时，选择工作流调度到公有云上。文献[6]研究网格环境下的任务之间相互独立的应用，当一个任务延迟会导致整个应用延迟时，估算各个任务在当前可用的云资源上的最早完成时间，选择最早完成时间最小的任务调度到公有云上。文献[7,8]研究批处理应用在混合云上的开销最小化，并且尽量在截止时间之前完成任务。公有云使用按需计费模式，首先估算各应用如果调度到公有云上的费用，任务根据估计费用降序排列。周期遍历队列，如果一个应用预计不能在截止时间之前完成，该应用、或估计费用最低的应用被调度到最便宜的公有云实例上。文献[9]对文献[7,8]的工作进行扩展，研究 DAG 工作流在混合云上的开销最小化，并且尽量在截止时间之前完成任务，公有云使用按需计费模式。首先使用 HIAP（Hierarchical Iterative Application Partition，结构化的迭代的应用划分）算法将截止时间约束分解成各任务的子截止时间，然后计算各工作流的最长路径，根据最长路径降序排列，队列遍历时将超出子截止时间的任务所在工作流调度到最便宜的公有云实例上。

2. 多目标调度问题

对于多目标调度问题，目前的研究通常采用静态调度方式。TCHC (Time and Cost optimization for Hybrid Clouds)[10]算法在预计完成时间超过截止时间时，将开销最小、完成时间最短的任务调度到公有云上。文献[11]关注运行时长与费用开销的平衡。通常情况下，如果调度的目标是最短完工时间，则需要将任务放到公有云上，则费用开销大；若需要费用开销小，则任务尽量放在免费的私有云上，但是排队等待和资源冲突会延长完工时间。将优化问题建模成整数规划问题，并使用非支配排序算法 NSGA-II 进行修正。文献[12]假设在私有云和公有云实例数量有限，使用蚁群算法建模，分别针对减小完工时间和费用开销的目标，使用时间优先和开销优先的单目标优化策略，再使用熵优化模型最大化用户的 QoS 和

服务提供商的效益。文献[13,14]将混合云上任务调度问题建模为多维背包问题，设计了一种单目标差分演化算法，在交叉操作、选择策略以及控制参数的调整方面进行改进，同时，结合非支配排序算法 NSGA-Ⅱ中基于帕累托最优的快速排序的方法，提出了一种多目标差分演化算法，分别提高演化算法求解单目标约束的和多目标约束的混合云任务调度的时间效率。

4.2.2 数据密集型工作流的调度

对于数据密集型工作流的资源分配与任务调度，减小数据传输量是最重要的目标之一。主要的调度方法包括全局式调度方法、构造式调度方法，任务聚类和复制的调度方法，和基于工作流图划分的调度方法。

1. 全局式调度方法

全局式调度方法根据约束对调度问题进行建模求解。文献[15]将具有截止时间约束的数据密集型工作流在公有云上的调度问题建模成整数规划问题，同时考虑任务执行时间和数据存储位置，并减小总的费用开销。文献[16]针对数据密集型工作流在公有云上的调度，使用进化算法，用“分配染色体”对任务与计算节点的映射进行编码，“排序染色体”定义执行顺序，利用云计费模型和定制的演化算子，经过一系列染色体的交叉和变异操作，实现最小数据传输量和最短完工时间两个目标的优化。

2. 任务聚类和复制的调度方法

任务聚类和复制方法的主要思想是将较大通信量的任务放在同一节点/平台内。文献[17]针对数据密集型工作流在公有云上的调度，基于任务执行时间、任务之间的依赖关系、数据文件大小，利用数据重用、数据复制技术，实现任务到计算节点的映射，减小数据传输，实现减小执行时间和节省费用开销的权衡。文献[18]关注数据密集型工作流调度在不同数据中心时，数据中心之间总的数据通信时间最短。提出数据中心选择算法，减小使用的数据中心个数，同时提高数据中心之间的可用带宽。在调度初期使用 k-mean 聚类方法将通信量大的任务进行聚类，使用多层任务复制技术用来降低数据传输量。

3. 基于工作流图划分的调度方法

基于工作流图划分的调度方法首先在一定的目标约束下，使用图划分技术将数据密集型工作流划分成一系列子工作流，子工作流之间数据传输量最小，再以子工作流为基本单位，分别调度到不同的资源节点或云平台上，确保不同节点/平台之间的数据传输量最小。

文献[19]指出传统图划分算法不能识别 DAG 边的方向，因此不能识别在工作流中代表任务之间执行顺序的数据传输方向，可能导致任务不能很好地分配在离数据存储位置最近的计算节点上，使用 MCGP（Multi-Constraint Graph Partitioning,

多约束的图划分算法）[20]，为 DAG 中的每个任务顶点赋予一个矢量权重，MCGP 均衡每个子图内顶点的矢量各维度和，并且确保割值最小，实现减小数据传输量和最短完工时间。文献[21,22]针对数据密集型工作流在异构计算系统的调度问题，使用双重目标约束下的工作流划分，提高吞吐量，降低子工作流间数据传输量，并使用部分任务复制技术，进一步减小数据传输延迟，最后，将每个子工作流映射到能够最快完成该子工作流的计算节点上。文献[23]针对志愿计算系统与公有云组成的混合系统，调度具有截止时间约束的数据密集型工作流，提高满足截止时间的工作流的比例，并节省费用开销。首先，使用禁忌搜索算法，将工作流划分成一组节点数相近、总数据传输量最小的子工作流。基于志愿计算资源可用性和负载均衡目标，将子工作流调度到分布式的志愿系统上，估算每个子工作流在计算节点上的执行时间，如果超出子截止时间，则调度到公有云上，公有云的计费模型使用 Amazon EC2 的竞价型实例 Spot Instance。

4.2.3 当前研究存在的问题

通过对已有的数据密集型科学工作流调度、混合云环境下工作流调度的研究，当前主要存在的问题主要有以下几点：

（1）混合云上的调度通常只针对普通的科学工作流，并没有关注数据密集型科学工作流可能产生的数据传输量过大的问题。

（2）数据密集型工作流的调度研究大多数工作目标在于减小节点间的数据通信量；MCGP 算法、禁忌搜索算法等工作流划分算法虽然可以获得较好的划分效果，但是算法复杂度高、耗时长，不适合动态调度对响应时间的需求。

本章针对具有截止时间约束的数据密集型科学工作流在混合云系统上的调度问题，目标是在截止时间之前尽可能多的完成工作流，减小云之间的数据传输量，减小费用开销。将工作流划分为子工作流，作为公有云调度的最小单元，减小不同云之间的数据传输量，使用快速的双层禁忌搜索图划分算法进行工作流划分，减小对调度算法带来的时间开销。使用动态调度算法，在运行期内估算任务完成时间、调度任务，适应云环境下计算资源、网络资源的动态特性。

4.3 系统模型

本章提出的动态调度算法，主要关注私有云和公有云组成的混合云平台上，一组持续到达数据密集型科学工作流的调度问题。

4.3.1 混合云模型

混合云 $P=p_{private} \cup p_{public}$，包括用户或组织内部的私有云和若干个不同的公有

云。公有云服务提供商通过 Internet 向用户或组织提供一组不同的服务 $S_p=\{s_{p1},s_{p2},\ldots,s_{pm}\}$。本章主要研究 IaaS 服务，关注提供服务的 CPU 的数量、计算速度、网络传输速度。

在混合云中，组织内部的私有云、外部的公有云都是以虚拟机的形式提供计算资源，统一称为云实例。云实例的配置信息包括：内存、vCPU 个数、存储容量、网络能力等，实例通常被固定配置为某一种或几种实例类型提供给用户使用。例如，Amazon EC2 提供的 t2.small 类型实例包括 1 个 vCPU、2GB 内存、弹性块存储，t2.large 类型实例包括 2 个 vCPU、8GB 内存、弹性块存储[24]。

每当一个工作流到达时，根据用户定义或者工作流配置，可以选择一类云实例，作为资源池分配给工作流。通常来说，公有云和私有云的实例数量都是有限的，相对于私有云来说，公有云实例数量巨大，本章假设公有云实例数量足够大，每当用户云实例不够时，都可以租用新的公有云实例。

本章仅考虑任务是调度在私有云上还是公有云上，不考虑多个公有云的选择问题，因此，本章的混合云包括一个私有云和一个公有云。公有云计费模式参考 Amazon EC2 的按需计费模型[24]，计算费用以小时为单位计费，数据传输费用根据传入、传出公有云的数据量，以 GB 为单位计费。

4.3.2 科学工作流模型

科学工作流通常将工作流的任务和任务之间的依赖关系建模为有向无环图 (Directed Acyclic Graph, DAG)，用 $G=\{V,E\}$表示，其中，$V=\{T_1,T_2,\ldots,T_n\}$代表一个工作流中的 n 个任务，$E=\{(T_i, T_j)|\ T_i, T_j\in V\}$表示任务 T_i 与 T_j 的依赖关系，$e=(T_i,T_j)$的值为任务 T_i 向任务 T_j 传输的数据量。边$(T_i, T_j)\in E$ 表示任务 T_i 完成后将数据传输给任务 T_j，然后任务 T_j 才能开始运行，T_i 称为 T_j 的父任务，记为 $parent(T_j)$，任务 T_j 称为 T_i 的子任务，记为 $child(T_i)$，没有任何父任务的任务称为 DAG 的入口，记为 T_{entry}，没有任何子任务的任务称为 DAG 的出口，记为 T_{exit}。本章的工作涉及以下定义：

定义 1. 执行时间。任务 T_i 在云实例 it_j 上的执行时间记为 w_{ij}。平均执行时间 $\overline{w_i}$ 是任务 T_i 在 M 个云实例上的平均执行时间：

$$\overline{w_i}=\sum_{j=1}^{M}w_{ij}\Big/M=\sum_{j=1}^{M}(L_i/R_j)\Big/M \tag{4-1}$$

其中，R_j 是云实例 it_j 的计算速度，用 MIPS 衡量，L_i 是任务 T_i 的长度，用 Million Instructions 衡量。

定义 2. 数据传输时间 c_{ij}。任务 T_i 向任务 T_j 传输数据的平均时间：

$$c_{ij}=delay+D_{ij}/B \tag{4-2}$$

其中，*delay* 是网络平均时延，*B* 是网络平均带宽，D_{ij} 是任务 T_i 向任务 T_j 传输的数据量。值的注意的是，如果两个任务运行在同一个云实例上，或者分别运行在部署于同一物理主机上的不同云实例上，则 c_{ij}=0。

定义 3. 工作流的完工时间 makespan。一个工作流从提交到最后一个任务完成的时间：

$$makespan = \max\{FT(T_{exit})\} - ST \tag{4-3}$$

其中，*ST* 是工作流提交到云上的时间节点，$FT(T_i)$是任务 T_i 的完成时间。对于不止一个出口的工作流，所有出口任务中最晚完成的时间即为工作流的完成时间。

定义 4. 工作流截止时间 deadline。由用户或配置文件定义的约束，表示工作流必须在该时间之前完成，即 makespan≤deadline，否则认为该工作流失败。

定义 5. 最早开始时间 EST(Earliest Start Time)。任务 T_i 最早开始于所有父任务都完成，并且中间数据都传输到 T_i，任务 T_i 在云实例 it_j 上最早开始于所有父任务都完成，并且中间数据都传输到 it_j 上，并且 it_j 空闲。

$$\begin{aligned} EST_i &= \max_{T_p \in parent(T_i)} (FT(T_p) + c_{pi}) \\ EST_{ij} &= \max(EST_i, FreeSlot_j) \end{aligned} \tag{4-4}$$

其中，$FreeSlot_j$ 为 it_j 的最早空闲时间，对于入口任务，$EST_{entry}=ST$。

定义 6. 最早结束时间 EFT(Earliest Finish Time)。任务 T_i 在 it_j 上的最早结束时间为

$$EFT_{ij} = EST_{ij} + w_{ij} \tag{4-5}$$

定义 7. 任务的子截止时间为最晚结束时间 LFT(Latest Finish Time)。在工作流的截止时间约束下，为了保证工作流的完成时间不会超出截止时间，每个任务都有一个子截止时间，任务 T_i 的完成时间不应晚于 LFT_i：

$$LFT_i = \min_{T_c \in child(T_i)} (LFT_c - \overline{w_c} - c_{ic}) \tag{4-6}$$

对于出口任务，$LFT_{exit}=deadline$。

4.4 数据密集型科学工作流的动态调度算法

本节提出的数据密集型科学工作流的动态调度算法包括两部分，科学工作流划分和动态混合调度。新到达的工作流首先被划分为多个子工作流，子工作流之间的数据通信量最小。然后工作流进入私有云排队，队列每隔一段时间被遍历一次，估算队列中各个任务的结束时间，如果超出 LFT，表示该任务将被延迟，整个工作流会超出截止时间，此时，选择一个或多个子工作流调度到公有云上。

4.4.1 数据密集型科学工作流划分

数据密集型科学工作流的任务之间的通信数据量巨大，工作流划分的目的在于，将工作流划分为 k 个子工作流，每一个子工作流内部的任务都是高内聚的，子工作流之间的任务则是低耦合的，即将数据通信量大的任务封装到一个子工作流，而子工作流之间的数据通信量最小。在调度时，属于一个子工作流的所有任务同时调度到同一云平台上，确保云平台之间数据传输量最小、传输时间最短，且造成网络拥塞的可能性最小。

使用 DAG 图划分算法，可以将 DAG 划分成一组不相交的、拥有基本相同任务数的子集，并且子集之间通信量之和最小。

定义 8. 图划分的概念。

对于 DAG，将 n 个顶点的集合 V 划分成 k 个子集 $V_1,V_2,\ldots,V_k$，如果图划分后的子集能够满足以下条件，则称为对图 DAG 的一次“k-路划分”。

a) $V_i \cap V_j=\Phi$, $i,j=1,2,\ldots,n$, $i\neq j$;

b) $|V|\approx n/k$;

c) $V_1 \cup V_2 \cup \ldots \cup V_k=V$;

d) 划分的割值最小。

两个不在同一子集的顶点之间的边称为“割边”，割边的值称为“割值”。

图的划分问题是一个 NP 完全问题[25]，本文使用双层禁忌搜索图划分算法 DLTS（Double-Level Tabu Search）对一个包括 n 个任务的工作流进行划分，要求每个子工作流中任务个数不超过 m 个，如图 4-1 所示，包括 4 步：

Step1：图塌缩，使用 HEM（Heavy-Edge Matching，重边匹配）算法，将大图 G_0 逐步塌缩成小图 $G_1,G_2\ldots,G_m$，其中$|V_0|>|V_1|>|V_2|>\ldots>|V_m|$，当$|V_m|\leqslant k_1$ 时停止；

Step2：初始划分，使用禁忌搜索图划分算法对 G_m 进行 k_2-路划分，生成 k_2 个子集；

Step3：图恢复，将塌缩阶段合并的节点解耦合，将图恢复成原状，对应为 k_2 个子工作流；

Step4：细划分，若一个子工作流的内部任务数 $m_1>m$，则使用禁忌搜索图划分算法对该子工作流进行$\lceil m_1/m \rceil$-路划分。

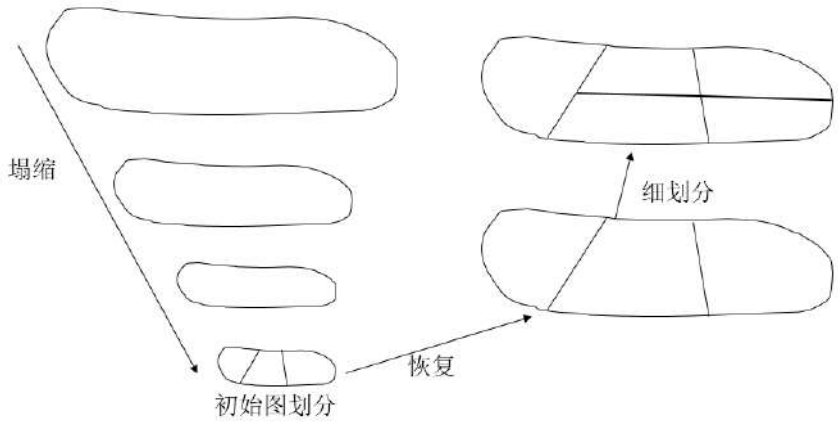


图 4-1 数据密集型科学工作流 DAG 图划分步骤

1. 图的塌缩和恢复

图的塌缩使用 HEM 算法通过将大的 DAG 图中数据通信量大的两个任务节点合并成一个节点，来逐步缩小图的规模。将图 G_{i-1} 塌缩成图 G_i 的主要步骤如下：

Step1：将 G_{i-1} 内部有数据传输的任务 T_p, T_q 的数据传输量 D_{pq} 按照降序排列，保存<D_{pq}, T_p, T_q>记录的列表 D_{list}；

Step2：对于 D_{list} 的第一条记录<D_{pq}, T_p, T_q>，$T_p+T_q=T_k^{'}\rightarrow G_i$；

Step3：从 D_{list} 中删除顶点为 T_p 或 T_q 的所有记录；

Step4：从 G_{i-1} 中删除顶点 T_p，T_q，以及与之邻接的边；

重复 Step2-4，直至 D_{list} 中没有记录；

Step5：将 G_{i-1} 中的剩余点加入 G_i，构造 G_i。

Step2 中任务 T_p 和 T_q 合并成一个顶点 $T_k^{'}$，即 $T_p+T_q=T_k^{'}$，$T_k^{'}$ 需要记录的信息包括内部子任务 id、子任务之间数据通信量、与其他外部任务节点的数据通信量。在 Step5 中，根据这些信息构造 G_i。在图恢复阶段，根据这些信息完整恢复出初次划分的子工作流图。

2. 禁忌搜索工作流图划分算法

禁忌搜索算法是一种元启发式的随机搜索算法，虽然对于较大规模的图的划分，禁忌搜索算法的时间开销较大，但是经过塌缩的 DAG 规模较小，它的性能较好。首先将 DAG 进行随机划分，然后经过一定次数的置换操作，达到最佳图划分，总割值最小。

定义 9. 相邻两个子工作流 sw_i 与 sw_j 的割值，记为 $NetCut(sw_i,sw_j)$，为两个子工作流之间所有传输数据量之和。

$$NetCut(sw_i,sw_j)=\sum_{\substack{T_p\in sw_i\\T_q\in sw_j}} D_{pq} \tag{4-7}$$

定义 10. 工作流图划分总割值 SumNetCut，工作流划分成的所有子工作流之间的割值之和。

定义 11. 子工作流的权重 Weight，是指子工作流内部任务执行时间和数据传输时间之和。

$$Weight(sw_i) = \sum_{T_p \in sw_i} \bar{w}_p + \sum_{\substack{T_p \in sw_i \\ T_q \in sw_i}} c_{pq} \tag{4-8}$$

定义 12. 置换收益 MoveGain，是指将一个任务 T_p 从 sw_i 置换到 sw_j 之后，$NetCut(sw_i,sw_j)$的增量。

$$MoveGain(sw_j, sw_i, T_p) = NetCut(sw_j + T_p, sw_i - T_p) - NetCut(sw_j, sw_i) \tag{4-9}$$

算法 4-1 描述了工作流的禁忌搜索图划分算法。首先，将工作流随机划分成大小接近的 k 个子工作流，作为目前的最佳划分 SW_{best}，并计算 *SumNetCut*、*Weight*、*MoveGain*（1-4 行）。用令牌环控制方式交替选择两种置换方式，不断置换相邻子工作流中的任务，直到 *SumNetCut* 在 λ 次迭代中保持不变（第 24 行）。

- 单次置换 SingleMove：随机选择一个子工作流 sw_{dest}，且 $Weight(sw_{dest}) \neq Weight_{max}$。对于所有的 $T_c \in sw_c$ 且 $sw_c=\{sw_c \in SW | Weight(sw_c) > Weight(sw_{dest})\}$，选择 $MoveGain(sw_{dest}, sw_c, T_c)$的最大值，对应的 sw_c 即为 sw_{source}，T_c 即为 T_{move}，将 T_{move} 从 sw_{source} 置换到 sw_{dest}。（5-9 行）
- 双次置换 DoubleMove：首先操作 SingleMove，选择 T_{move1}，$sw_{source1}$，sw_{dest1}。然后随机选择另一个子工作流 sw_{dest2}，$dest1 \neq dest2$ 且 $Weight(sw_{dest2}) \neq Weight_{max}$。对于所有的 $sw_{c2}=\{sw_{c2} \in SW | sw_{c2} \neq sw_{dest1}, sw_{c2} \neq sw_{dest2}\}$，且 $T_{c2} \in sw_{c2}$，选择 $MoveGain(sw_{dest2}, sw_{c2}, T_{c2})$的最大值，对应的 sw_{c2} 即为 $sw_{source2}$，T_{c2} 即为 T_{move2}。将 T_{move1} 从 $sw_{source1}$ 置换到 sw_{dest1}，T_{move2} 从 $sw_{source2}$ 置换到 sw_{dest2}。（10-15 行）

对于置换后的新的子工作流集合 *SW'*，若新的 *SumNetCut'* 小于当前的 *SumNetCut*，则将 *SW'* 保存成 SW_{best}（16-19 行），否则将当次置换取消（第 20 行）。

此外，通过扰动函数 Pertubation 避免陷入局部最优解（21-23 行）。具体操作方法是：

- 随机选择一组子工作流 $sw_m \in \{sw_1, sw_2, \ldots, sw_k\} - \{sw_{max}\}$，其中 sw_{max} 是 *Weight* 最大的子工作流；
- 随机选择 T_c，其所在子工作流 sw_c 满足 $Weight(sw_c) > Weight(sw_m)$；
- 将 T_c 从 sw_c 置换到 sw_m；
- 重复 p_{str} 次上述操作。

算法 4-1.工作流禁忌搜索的 k-路图划分算法

输入：DAG 工作流包括 n 个任务 $T=\{T_1,T_2,...,T_n\}$

输出：k 个子工作流 $SW_{best}=\{sw_1, sw_2, \ldots, sw_k\}$

1: Partition DAG randomly into k sub-workflows $SW_{best}=\{sw_1, sw_2, \ldots, sw_k\}$

2: Compute $SumNetCut\ (SW_{best})$

3: **for** each $sw_i \in SW$， compute $Weight_i$

4: **for** each sw_i and neighbor sw_j, for each $T_p \in sw_i$, compute $MoveGain(sw_j,sw_i,T_p)$

4: **repeat**

5: **if** token=SingleMove **do**

7: Choose sw_{dest}, T_{move}, sw_{source}

8: Move T_{move} from sw_{source} to sw_{dest}

9: **end if**

10: **if** token=DoubleMove **do**

11: Choose sw_{dest1}, T_{move1}, $sw_{source1}$

12: Move T_{move1} from $sw_{source1}$ to sw_{dest1}

13: Choose different sw_{dest2}, T_{move2}, $sw_{source2}$

14: Move T_{move2} from $sw_{source2}$ to sw_{dest2}

15: **end if**

16: **if** $SumNetCut\ (SW') < SumNetCut\ (SW_{best})$ **do**

17: $SW_{best} \leftarrow SW'$, $SumNetCut(SW_{best}) \leftarrow SumNetCut(SW')$

18: Updata Weights, MoveGains

19: **end if**

20: **else** roll back

21: **if** SW_{best} not improved after g iterations **do**

22: Perturbation(sw_{dest}, T_{move}, sw_{source})

23: **end if**

24: **until** $SumNetCut(SW_{best})$not improved after λ iterations

4.4.2 动态混合调度算法

混合云环境下持续输入的有截止时间约束的工作流的动态调度，包括 3 个部分：

- 私有云调度：所有未调度的任务首先被排序在私有云队列中。当一个任务的所有父任务都完成，该任务变成就绪状态，若私有云可用资源足够，

该任务的子工作流调度私有云实例上。

- 队列遍历：定期遍历私有云上的任务队列，估算各任务的完成时间，如果晚于 LFT，则意味着工作流可能无法在截止时间之前完成，选择一个或多个子工作流调度到公有云上。
- 公有云调度：调度子工作流到公有云上，选择保障每个任务在 LFT 之前完成、且费用最低的公有云实例。

1）私有云调度

所有工作流的任务首先在私有云上排队，任务在所有父任务都完成时变成就绪状态，若一个任务进入就绪状态，则认为该任务的子工作流也进入就绪状态。子工作流是调度的最小单位，不是在提交时调度，而是在子工作流就绪之后，如果私有云资源足够，保证就绪子工作流中的任务都能在 LFT 之前完成，则调度到私有云上执行；如果私有云资源不足，则估算如果继续排队等待，子工作流中任务的预计完成时间，若超出 LFT，则子工作流调度到公有云上。

任务在私有云上的资源选择和调度使用 EDF-EFT 策略与算法，任务按照 EDF（Earliest Deadline First，最小完成时间优先）策略排序，即将任务按照 LFT 升序排序，LFT 小的工作流排在前面，优先选择可以实现 EFT 的私有云实例调度。

2）公有云调度

公有云调度为子工作流中的任务选择合适的公有云实例，使用 HLCF（Heterogeneous Least Cost First，异构的最小开销优先）算法。子工作流 sw_j 中的任务首先按照 LFT 升序排序，依次为任务选择开销最小的公有云实例，使得任务能够在 LFT 之前完成，并且费用开销最低，即：

$$\begin{aligned} &\min \quad cost \\ &\text{Subject to} \quad FT_i \le LFT_i \end{aligned} \tag{4-10}$$

开销包括子工作流 sw_j 中所有任务的计算开销和数据传输开销：

$$cost = \sum_{T_i \in sw_j} cost_i^{comp} + cost^{data} \tag{4-11}$$

$$cost^{data} = PrivateCut(sw_j) * p^{trans} \tag{4-12}$$

$$PrivateCut(sw_j) = \sum_{T_i \in private} NetCut(T_i, sw_j) \tag{4-13}$$

$$cost_i^{comp} = \min_{k \in requiredtype} (w_{ik} * p_k^{comp}) \tag{4-14}$$

其中，p^{trans} 是基于“传入”和“传出”公有云的数据价格，以“每 GB”为单位计算。$PrivateCut(sw_j)$是子工作流 sw_j 与在私有云上等待、运行、完成的其他任务的割值。任务 T_i 的计算开销是 T_i 在实例 it_k 上的预计执行时间 w_{ik} 乘以 it_k 的

单价，记为 p_k^{comp}，如公式(4-14)所示。与文献[7-9]提出的每次选择单价最低的公有云实例不同，依据公式(4-14)，任务 T_i 可能不是调度在单价最低的云实例上，也可能调度到单价高速度快的云实例上，从而 $w_{ik} * p_k^{comp}$ 最低。

3）动态混合调度

所有的工作流首先在私有云上排队，工作流的排序原则可以是以下几种：

- FCFS（First Come First Served，先来先服务）：工作流按照到达顺序排列，新来的工作流被排到队尾，工作流根据提交时间先后调度。
- EDF（Earliest Deadline First，最小完成时间优先）：工作流按照截止时间排序，截止时间早的工作流排在前面。

周期性的遍历私有云上的任务队列，根据当前私有云的计算资源、网络资源性能，根据私有云调度策略 EDF-EFT 估算各任务的完成时间，当一个任务预计超出 LFT，意味着整个工作流将被延迟，该任务被称为“不可靠任务”，所在子工作流称为“不可靠子工作流”。需要选择子工作流调度到公有云上。子工作流的选择策略包括：

- 直接选择：将不可靠子工作流直接调度到公有云；
- 最少传输优先：选择排队在不可靠任务之前、执行时长不小于不可靠任务、*PrivateCut* 最小的子工作流调度到公有云。

算法 4-2 动态混合调度算法.

输入：$Q_{private}$ 已排序的 m 个工作流，$W=\{w_1,w_2,\ldots,w_m\}$，

排序的 n 个任务 $T=\{T_1,T_2,\ldots,T_n\}$，排序的 s 个子工作流 $SW=\{sw_1,sw_2,\ldots,sw_s\}$，

输出：Q_{public} 任务与公有云实例的映射$\{<T_1, it_1>, <T_2, it_2>,\ldots,<T_k,it_k>\}$

1 **for** each $T_i \in T$, **do**
2 compute FT_i
3 **if** $FT_i > LFT_i$ **do**
4 $sw_{move} \leftarrow sw(T_i)$ //直接选择 T_i 所在子工作流
5 $sw_{move} \leftarrow \min_{sw_p \in SW} PrivateCut(sw_p)$ //选择最小割值的子工作流
6 $Q_{public} \leftarrow$ public_scheduler(sw_{move})
7 remove sw_{move} from $Q_{private}$
8 **end if**
9 **end for**

动态混合调度算法如算法 4-2 所示。对于私有云队列中的任务，根据当前私有云平均计算、传输速率，使用 EDF-EFT 算法估算完成时间 FT，对于调度在公有云上的任务，执行时间、数据传输时间根据已选择的公有云实例的计算、传输速度来估算(1-2 行)。如果任务的预计完成时间 FT 晚于 LFT，即为不可靠任务

（第 3 行），子工作流根据不同的策略被选择出来，直接调度(第 4 行)、或者最小割值调度(第 5 行)，使用 HLCF 算法调度到公有云实例上。

4.5 实验分析

实验首先对比本章提出的 DLTS 工作流划分算法和已有算法划分实际工作流的时间效率和性能；随后仿真混合云环境，分别运行不同数据量的工作流，对比在私有云规模不同、等待队列遍历频率不同的情况下，几种排序和调度策略的性能。

4.5.1 实验环境描述

本节使用 CloudSim3.0 仿真混合云，私有云和公有云均包括 4 种实例类型，公有云计费参考 Amazon EC2 的亚太地区-地区价格[24]，云实例配置信息和带宽信息如表 4-1、4-2、4-3 所示。

表 4-1 私有云实例类型

名称	vCPU 数量	内存 GB
small	1	2
medium	2	4
xLarge	4	16
2xLarge	8	32

表 4-2 公有云实例类型和价格

名称	vCPU 数量	内存 GB	价格$/h
small	1	2	0.03
medium	2	4	0.06
xLarge	4	16	0.24
2xLarge	8	32	0.48

表 4-3 云带宽和价格

位置	带宽 MGB/s	价格$/GB
私有云内部	20	0
私有云向公有云	10	0
公有云向私有云	10	0.12
公有云内部	20	0

工作流分别使用 CyberShake 和 Montage 工作流，CyberShake 工作流是美国宇航局拼接太空图片的应用，是一种数据密集型工作流，Montage 工作流是美国

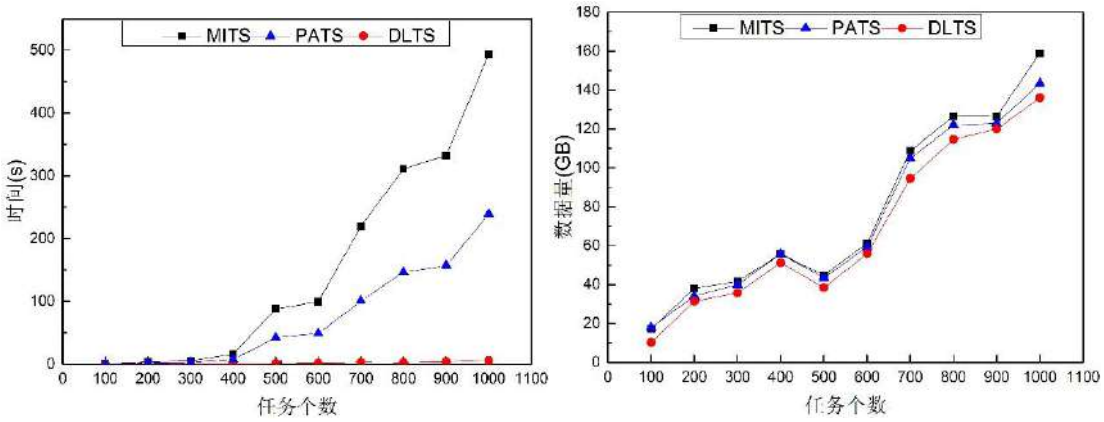
地震中心从表征区域中提取地震灾害特征的应用，是一种计算密集型工作流[2]。工作流的 DAG 结构，包括任务之间的依赖关系、运行时间信息、任务的输入输出文件等，都使用 Pegasus DAX 描述，随机选择 WorkflowGenerator 生成的包含 1000 个任务的 20 个工作流实例持续到达混合云，到达时间依照卢森堡大学的 Gaia 集群的日志数据[26]中的 job 到达时间，一个工作流的截止时间为平均最佳运行时间的 k 倍，k 取值范围是[3, 20]。工作流的最佳运行时间是工作流在足够多的资源上，不存在资源不足造成的计算、传输速度下降，运行的完成时间，将 20 个工作流分别运行可以得到平均最佳运行时间。CyberShake 和 Montage 工作流的时间、数据量等信息如表 4-4 所示。

表 4-4 CyberShake 和 Montage 工作流的性能参数

参数	CyberShake	Montage
单个任务平均运行时间	半数任务约 50s 另外一半任务不足 1s	10-20s
单个工作流平均最佳运行时间	1500s	90s
单个工作流平均数据量	250GB	18.5GB
全部工作流平均数据量	250TB	18.5TB

4.5.2 工作流划分算法性能分析

本实验将本文提出的 DLTS 算法与已有的性能较好的 MITS[27]和对其改进的 PATS[28]算法对比。对包括 n 个任务的 CyberShake 工作流进行划分，每个子工作流包括大约 10 个任务，塌缩阶段停止塌缩的终止条件是被合并顶点内部任务数不大于 10，初始阶段对塌缩图进行$\lceil n/10 \rceil$-路划分，恢复阶段各子工作流内部任务数为 m，且$m \le 10*\lceil n/10 \rceil$，细化分阶段对各子工作流进行$\lceil m/10 \rceil$-路划分。图 4-2 显示了不同规模的工作流划分的时间开销和子工作流之间的数据传输总量，使用 DLTS 算法对 1000 个任务的工作流划分大约需要 5s，约占工作流执行时间（1500s）的 0.4%。随着工作流规模的增加，DLTS 的时间开销基本变化不大，而数据传输量也基本略小于 MITS 和 PATS 算法。



a)时间开销　　b) 数据传输量

图 4-2 工作流划分的性能比较

4.5.3 动态混合调度算法性能分析

实验分别针对不同的私有云规模、工作流数据量、等待队列遍历频率，对比几种调度算法的性能。算法根据在私有云上排序方式和在发现不可靠任务后，调度到公有云上的子工作流选择策略的不同，分为：

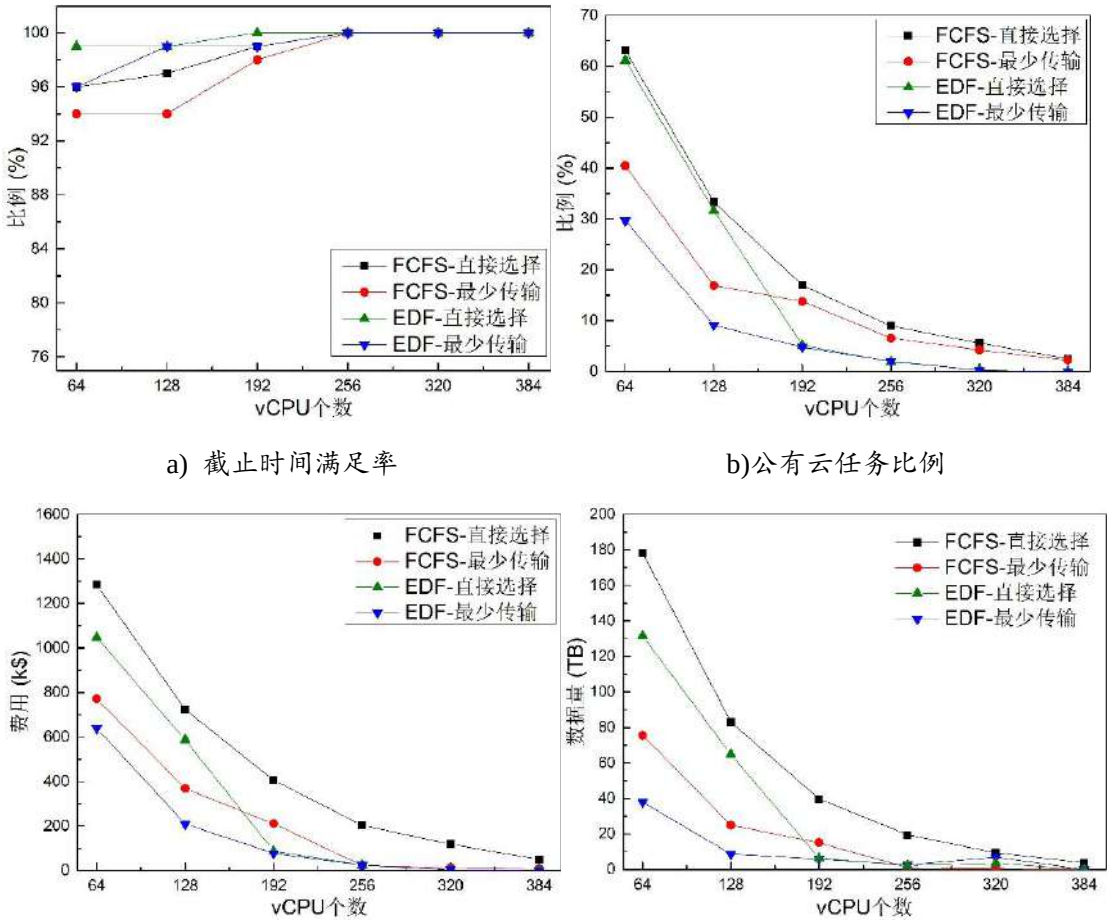
- FCFS-直接选择：按照 FCFS 排序，直接选择不可靠子工作流；
- FCFS-最少传输：按照 FCFS 排序，选择排在不可靠任务之前、PrivateCut 最小的子工作流；
- EDF-直接选择：按照 EDF 排序，直接选择不可靠子工作流；
- EDF -最少传输：按照 EDF 排序，选择排在不可靠任务之前、PrivateCut 最小的子工作流；

主要衡量的性能包括：a)截止时间满足率：能够在截止时间之前完成的工作流的比例；b)公有云任务比例：调度到公有云上运行的任务的比例；c)总的费用开销；d) 私有云和公有云之间传输的数据量。

1. 不同私有云规模的性能对比。

如图 4-3 所示，为数据密集型工作流 CyberShake 在不同私有云 vCPU 数量下的性能变化。随着私有云规模的增加，更多任务能够调度到私有云上完成，需要调度到公有云上的任务减少，相应的费用开销、数据传输量也会减小。从截止时间满足率来看，在私有云负载较高，导致私有云实例运行速度下降时，直接选择算法略优于最少传输算法。因为直接选择算法直接将不可靠子工作流调度到速度更快的公有云实例上，直接减小了工作流超出截止时间的可能性；而最少传输算法将不可靠任务之前的子工作流调度到速度更快的公有云实例上，减小了不可靠任务的等待时间，但是如果私有云实例运行速度仍旧很慢，工作流的完工时间可能还是很大，仍旧无法在截止时间之前完成。在公有云任务量、费用开销、数

据传输量方面，在私有云上的队列按照 EDF 排序要优于 FCFS 排序，最少传输算法的性能优于直接选择算法。小规模私有云中，FCFS-直接选择算法的公有云任务数比 EDF-最少传输算法多一倍，费用开销多一倍，而传输的数据量比 FCFS-最少传输算法多 1.25 倍，比 EDF-最少传输算法多 3.5 倍。



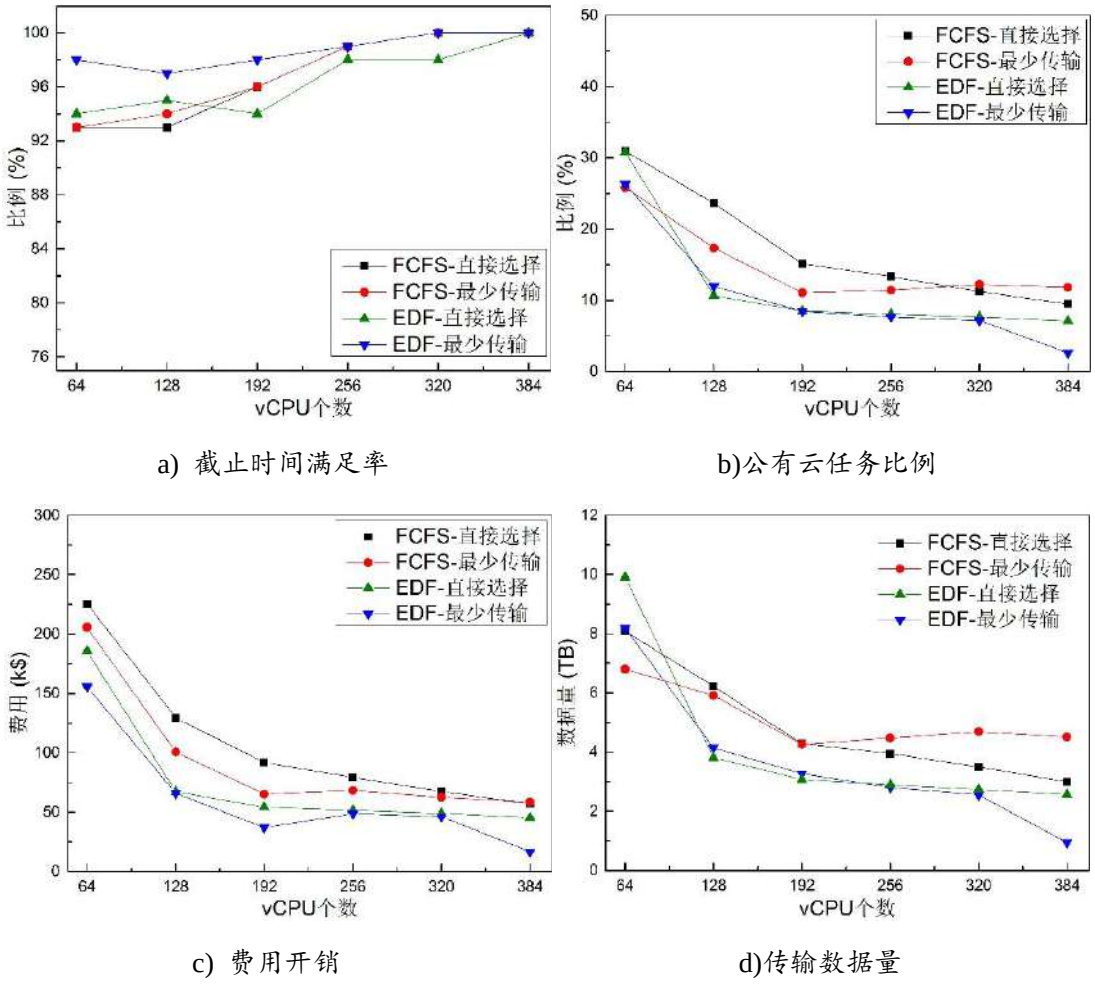
a) 截止时间满足率　　b)公有云任务比例

c) 费用开销　　d)传输数据量

图 4-3 私有云规模对 CyberShake 工作流的性能影响

2. 不同数据规模的性能对比。

图 4-4 为计算密集型工作流 Montage 在不同私有云 vCPU 数量下的性能变化。随着私有云规模的增加，工作流各项性能指标均有提高。EDF 排序算法的公有云任务数比 FCFS 排序算法少，因为在 EDF 算法中截止时间短、受云实例运行速度下降影响大的工作流先执行，截止时间长、受云实例运行速度下降影响小的工作流可以排队等待，而不需要将其中的任务调度到公有云上。对比 CyberShake 可以看出，不论私有云规模如何，四种算法下，工作流性能的区别不大。因为对于 Montage 工作流，数据量较小，子工作流调度与直接调度任务相比，产生的数据传输量差别不大，因此使用不同的选择算法选择出来的子工作流差别不大，对性能影响也不大。



a) 截止时间满足率　　b)公有云任务比例

c) 费用开销　　d)传输数据量

图 4-4 私有云规模对 Montage 工作流的性能影响

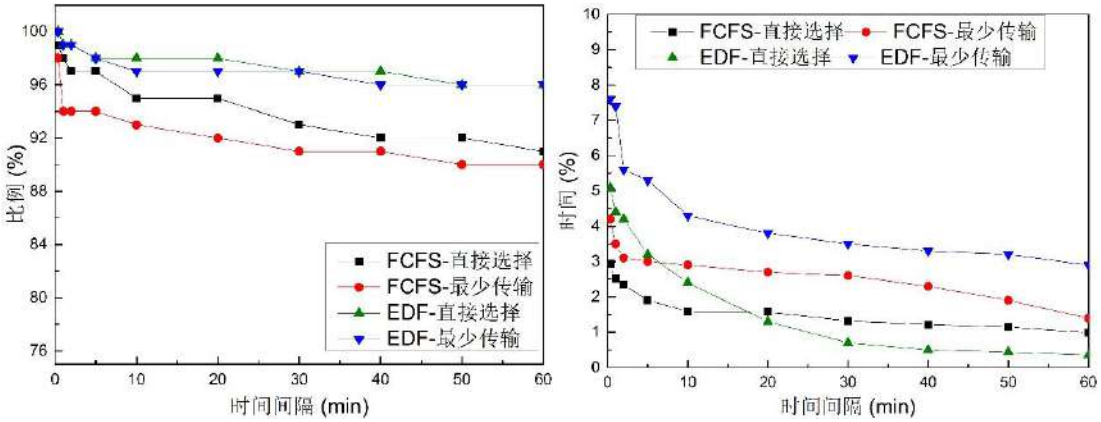
3. 等待队列遍历频率对性能的影响。

工作流在私有云中排队等待调度，队列被周期性遍历，估算各任务的预计完成时间，如果晚于任务的 LFT，则该任务为不可靠任务，需要选择子工作流调度到公有云上。遍历队列的频率决定了能否及时找到不可靠任务，消除工作流超出截止时间的隐患。如果频率过低，则不可靠任务不能被及时检测出来，工作流会超出截止时间，导致失败；如果频率过高，频繁的队列遍历、时间估计，则会带来更高的额外时间开销。图 4-5 显示了在不同的队列遍历频率下，截止时间满足率和调度的时间开销。私有云包含 128 个 vCPU，两次队列遍历的时间间隔为小于单个 CyberShake 任务的 20s，大于单个任务小于单个 CyberShake 工作流最佳完成时间的 1 分钟、2 分钟、5 分钟、10 分钟、20 分钟、30 分钟，和大于单个 CyberShake 工作流最佳完成时间的 40 分钟、50 分钟、60 分钟。

图 4-5 a)显示，随着遍历时间间隔的增加，FCFS 算法的工作流截止时间满足率下降幅度较大，EDF 算法下降幅度较小，因为在 EDF 算法中截止时间短、受云实例计算速度下降影响大的工作流先执行，截止时间长、受云实例计算速度下降影响小的工作流后执行，即使队列遍历频率低，也能相对及时的检测出不可靠

任务。

图 4-5 b)显示包括队列遍历、任务完成时间估算、子工作流选择在内的调度操作花费的时间占全部输入工作流总的完成时间的比例。随着遍历时间间隔的增加，额外开销占比减小。此外，所有情况下调度开销均小于 8%，特别是在遍历时间间隔大于 1 分钟后，调度开销小于 5%，对于输入工作流的总运行时间来说，可以忽略不计。



a)截止时间满足率　　b) 额外时间开销

图 4-5 等待队列遍历频率对性能的影响

4.6 本章小结

本章针对持续到达的有截止时间约束的数据密集型科学工作流，提出了动态混合调度算法，使得公有云和私有云之间数据传输量最小，公有云的费用开销最低。工作流使用双层禁忌搜索图划分算法 DLTS 进行快速划分，确保子工作流之间数据传输量最小。到达的工作流首先根据 FCFS 或 EDF 原则在私有云上排队。若私有云资源空闲，将就绪子工作流按照 EDF-EFT 策略调度到私有云上。将工作流的截止时间约束映射成各任务的最迟完成时间约束 LFT，周期性遍历等待队列，估算各任务完成时间，如果晚于 LFT，则直接选择该任务的子工作流，或者选择排队在该任务之前、PrivateCut 最小的子工作流，使用 HLCF 算法调度到公有云实例上。仿真实验分析了 DLTS 工作流划分算法的性能，并对比了 2 种排序策略 2 种选择策略下，调度的截止时间满足率、费用开销、数据传输量等的性能。

参考文献

[1] Bittencourt L F, Madeira E R M. HCOC: a cost optimization algorithm for workflow scheduling in hybrid clouds [J], Journal of Internet Services and

Applications, 2011, 2 (3): 207-227.

[2] Juve G, Chervenak A, Deelman E, et al. Characterizing and profiling scientific workflows [J]. Future Generation Computer Systems, 2013, 29: 682-692.

[3] Ostermann S, Prodan R. Impact of variable priced cloud resources on scientific workflow scheduling [A]. // Proceedings of International European Conference on Parallel and Distributed Computing [C], Rhodes Island: Springer Press, 2012: 350-362.

[4] Bittencourt L F, Madeira E R M. A performance-oriented adaptive scheduler for dependent tasks on grids [J]. Journal of Concurrency and Computation, Practice & Experience, 2008, 20 (9): 1029-1049.

[5] Kumar B A, Ravichandran T. Time and cost optimization algorithm for scheduling multiple workflows in hybrid clouds [J]. European Journal of Scientific Research, 2012, 89 (2): 265-275.

[6] Lee Y C, Zomaya A Y, Rescheduling for reliable job completion with the support of clouds [J]. Future Generation Computer Systems, 2010: 1192-1199.

[7] Bossche R V, Vanmechelen K, Broeckhove J. Cost-efficient scheduling heuristics for deadline constrained workloads on hybrid clouds [A]. //Proceedings of IEEE International Conference on Coud Computing Technology and Science [C], Athens: IEEE Press, 2011: 320-327.

[8] Bossche R V, Vanmechelen K, Broeckhove J. Online cost-efficient scheduling of deadline-constrained workloads on hybrid clouds [J]. Future Generation Computer Systems, 2013, 29: 973-985.

[9] Lin B, Guo W, Lin X. Online optimization scheduling for scientific workflows with deadline constraint on hybrid clouds [J]. Journal of Concurrency and Computation: Practice & Experience, 2015, 28 (11): 3079-3095.

[10] Chopra N, Singh S. Deadline and cost based workflow scheduling in hybrid cloud [A]. //Proceedings of Advances in Computing, Communications and Informatics [C], Mysore: IEEE Press, 2013: 840-864.

[11] Leena V A, Ajeena Beegom A S, Rajasree M S. Genetic Algorithm Based Bi-Objective Task Scheduling in Hybrid Cloud Platform [J]. International Journal of Computer Theory and Engineering, 2016, 8 (1): 7-13.

[12] Zuo L, Shu L, Dong S, et al. A Multi-objective Hybrid Cloud Resource Scheduling Method Based on Deadline and Cost Constraints [J]. IEEE Access, 2016, 99 (9): 1-13.

[13] 梁庆中. 混合云平台上多目标任务调度算法研究 [D].武汉:中国地质大学, 2015: 23-34.

[14] Fan Y, Liang Q, Chen Y, et al. Executing time and cost-aware task scheduling in hybrid cloud using a modified DE algorighm [A]. //proceedings of the 7th International Symposium on Computational Intelligence and Intelligent Systems [C]. Guangzhou: Springer Press, 2015: 74-83.

[15] Pereira W F, Bittencourt L F, Fonseca N L S. Scheduler for data-intensive workflows in public clouds [A]. //Proceedings of IEEE Latin American Conference on Cloud Computing and Communications [C], Maceio: IEEE Press, 2013: 41-46.

[16] Szabo C, Sheng Q Z, Kroeger T, et al. Science in the Cloud: allocation and excution of data-intensive scientific workflows [J], Journal of Grid Computing, 2014, 12 (2): 245-264.

[17] Casas I, Taheri J, Ranjan R, Wet al. A balanced scheduler with data reuse and replication for scientific workflows in cloud computing systems [J]. Future Generation Computer Systems, 2016, 1-11.

[18] Zhang J, Wang M, Luo J, et al. Towards optimized scheduling for data-intensive scientific workflow in multiple datacenter environment [J]. Concurrency and Computation Practice and Experience, 2015, 27 (18): 5606-5622.

[19] Tanaka M, Tatebe O. Workflow scheduling to minimize data movement using multi-constraint graph partitioning [A]. //Proceedings of 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing [C], Washington DC: IEEE Press, 2012: 65-72.

[20] Karypis G, Kumar V. Multilevel algorithms for multi-constraint graph partitioning [A]. //Proceedings of the 1998 ACM/IEEE conference on Supercomputing [C], Washington DC: IEEE Press, 1998:1-13.

[21] Ahmad S G, Liew C S, Rafique M M, et al. Data-intensive workflow optimization based on application task graph partitioning in hterogeneous computing systems [A]. //Proceedings of IEEE Fourth International Conference on Big Data and Cloud Computing, Sydney: IEEE Press, 2014: 129-136.

[22] Ahmad S G, Liew C S, Rafique M M, et al. Optimization of data-intensive workflows in stream-based data processing models [J]. The Journal of Supercomputing, 2017: 1-23.

[23] Ghafarian T, Javadi B. Cloud-aware data intensive workflow scheduling on

volunteer computing systems [J]. *Future* Generation Computer Systems, 2015, 51: 87-97.

[24] Amazon EC2 pricing, https://aws.amazon.com/cn/ec2/pricing/on-demand/, 2017,3,10.

[25] 马永刚. 图划分方法及其在分布式网络环境下的应用[D]. 大连：大连理工大学. 2012: 26.

[26] The University of Luxemburg Gaia Cluster log, http://www.cs.huji.ac.il/labs/parallel/workload/l_unilu_gaia/index.html, 2017, 4, 10."

[27] Benlic U, Hao J-K. An effective multilevel tabu search approach for balanced graph partitioning[J]. Computers and Operations Research, 2011, 38 (7): 1066-1075.

[28] 郑丽丽, 武继刚, 陈勇等.带全图的均衡 k 划分[J]. 计算机研究与发展, 2015, 52 (3):769-776.

第五章 一种规范化的企业系统混合云迁移策略制定方法

5.1 引言

适当的企业云部署能够帮助企业降低成本、改进企业敏捷性。对于大型企业，已有系统已经较为成熟。使用混合云迁移策略，根据应用的安全和性能要求，将一部分应用或功能模块迁移到云上，剩下的部分保留在原来的本地部署环境中，能够节省云迁移的资金和消耗，同时降低风险。

混合云迁移策略更细致地考虑了应用程序与云环境的结合，现有的系统云迁移研究或者只分析应用迁移时需要考虑的目标和约束，或者只针对特定应用或功能模块给出专有的技术层面的迁移方案，或者只提出了通用的迁移步骤，并没有完备的约束分析与策略制定方法。针对企业整个系统的混合云迁移需求，需要完整的策略制定步骤和方法，能够指导企业按步进行迁移需求、目标、约束的分析、求解，获得解决方案，并通过测试判断方案有效后进行实际迁移。

5.2 相关研究工作

目前对系统云迁移的研究工作主要分成两类，一类重点分析云迁移的约束和目标，另一类关注迁移决策的制定。

5.2.1 云迁移约束分析

在云迁移目标和约束分析方面，文献[1]关注的是最大化收益，同时最小化数据传输开销，涉及的约束包括政策、延迟、流量。文献[2]关注在满足应用的 QoS 和 SLA 需求的同时，选择合适的云服务提供商，使得费用开销最小。MANITICORE[3]关注的是性能和费用开销的平衡，针对用户希望能够最大化的利用公有云资源的目标，建立了模块间的依赖分析模型，得出了模块云迁移的开销计算公式。文献[4]关注的是内容分发系统迁移到混合云环境下的最小开销问题。文献[5]在迁移政策、QoS 等约束下，关注最大化收益的同时，最小化费用开销。文献[6]列举了云迁移过程中产生的跟费用开销相关的一系列可量化、不可量化约束。文献[7, 8]列出了企业系统混合云迁移的全部约束条件、需要做的决策和需要完成的工作。

5.2.2 云迁移决策制定

部分研究针对某个系统的某个功能层，从技术方面分析了迁移的步骤和方法。

文献[9, 10]提出了应用数据层云迁移的基本步骤：工具收集、分析、设计、迁移数据、测试、优化、部署。文献[11]为 Web 应用的云迁移决策制定提供了知识库，基于 4 个决策点：应用分布、弹性扩展、多租户需求、服务提供商选择，提供了 17 个决策，每个决策又对应多个可行方案，最终可以生成超过 50 种决策。CloudGenious [12]列举了 Web 应用向云环境迁移的步骤：云服务选择、虚拟机镜像选择、云虚拟机镜像配置、迁移策略制定、应用迁移。

部分研究描述了企业云迁移的步骤。Cloudstep[13]定义了通用的迁移策略生成步骤，包括应用和云服务分析、技术和经济约束分析、迁移策略分析、执行迁移。$(MC^2)^2$[14]提出了判断是否选择使用云解决方案的基本步骤，包括场景描述、定义解决方案、约束描述、需求分析、选择多约束的决策制定方法、决策制定方法参数配置、求解、应用。文献[15]定义了确定迁移哪个模块的四个步骤：1）识别原始系统的功能模块和架构；2）描述模块间依赖关系；3）生成所有迁移方案；4）使用费用开销函数过滤并得到最佳方案。文献[16, 17]提出云迁移包括三个阶段：1）智能阶段，定义问题、需求、云迁移的机遇；2）设计阶段，分析服务适用性，分析迁移风险；3）云服务选择。

文献[18,19]提出了基于 AHP 方法的针对单个应用的迁移决策制定方法，进行方案生成和最佳解决方案求解，并在文献[18]中以企业智能应用的迁移部署为例，进行迁移决策分析，判断应用是否适合迁移，在文献[19]提出了规范化的半自动式决策制定方法，使得任何企业都可以通过规范化的分析方法制定决策。

5.2.3 当前研究存在的问题

已有研究存在如下问题：

（1）对于分析迁移目标和约束工作，部分成果仅根据几个可量化的目标，生成多个可行方案，或者只提出政策约束、QoS 等非量化指标，部分成果虽然列出了约束的全集，但是没有对约束进行分析和结构化分类，也没有给出衡量非定量约束的方法，无法根据这些约束有效过滤备选方案，同时，认为各类约束的重要性相同，而在实际企业系统中，各类约束对于企业决策的重要性是不一样的。

（2）对于系统迁移制定决策、执行迁移的研究，部分成果只针对某个特定的系统功能模块，从技术层面给出迁移方法。部分成果仅列举迁移步骤，但没有提出如何制定迁移策略。部分成果给出了迁移策略的制定方法，根据定性约束求解，但是没有对目标和约束进行层次化分析，也没有对最终的策略进行验证。

本章给出了一种形式化的、基于循环改进的企业系统混合云迁移策略制定方法和步骤，重点分析迁移目标、约束、云服务性能，基于 AHP（Analytic Hierarchy Process，层次分析法）[20]提出分层分析求解的策略制定方法，将定量的和定性的目标、约束、性能进行分层分析，进行统一的重要性分析比较，求解获得最佳解

决方案。

5.3 基于循环的规范化的混合云迁移策略制定

制定企业系统的混合云迁移策略，关键在于判断哪些功能模块需要迁移到云上，以及如何选择云提供商，在一系列的目标、可量化的应用性能约束、定性的系统约束下，实现企业业务战略。生成解决方案后，需要在实际环境中部署，使用真实业务数据进行测试，判断决策的正确性。因此，迁移策略制定类似于软件工程中软件设计的瀑布模型，是一种基于循环的、逐步优化的策略制定方法，如图 5-1 所示：

迁移评估：收集云迁移的相关信息，包括企业现有的平台系统和项目管理信息、IT 环境、业务战略、迁移粒度等；

决策分析：分析待迁移应用或功能模块的迁移目标、定量的性能约束，分析整个企业和系统约束，分析备选云服务；

决策制定：对目标、约束、性能进行分层分析，将定性指标转变成可用数值表示的相关重要度指标，并将定量、定性指标统一分析对比，获得初级解决方案，若初级解决方案确定需要使用公有云服务，则对备选公有云服务进行层次化分析，选择合适公有云服务，生成最终解决方案；

解决方案试部署与运行：根据解决方案的迁移策略，将目标应用或功能模块的程序和数据迁移到选择的云平台上，包括程序代码修改、程序架构适应性修改、数据提取与迁移等活动，部署完成后，迁移部分的业务数据，试运行该应用或功能模块；

测试与优化：经过一段时间的试运行，判断应用或功能模块在云上的运行情况是否符合目标和约束，验证业务功能是否正常，进行调试和优化，确保弹性扩展、负载均衡、持续不间断运行能力、备份和恢复等；若经过测试与优化，该方案仍旧不能满足需求，则将结果重新返回决策分析阶段，在决策分析中调整约束和目标，并重新制定决策；

实际业务上线：试运行成功后，实际业务切割上线。

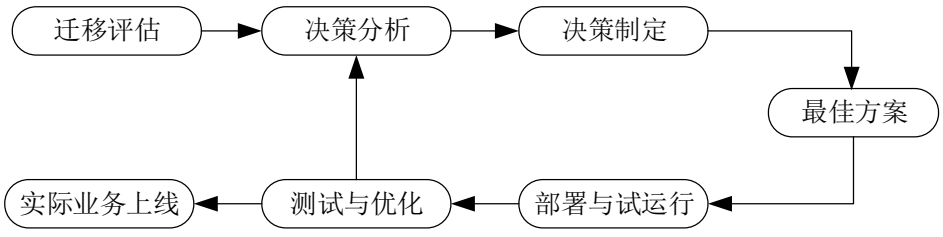


图 5-1 企业系统混合云迁移步骤

5.3.1 迁移评估

制定企业系统云迁移战略，首先需要收集相关信息，用以评估迁移的基本方向和策略，包括企业现有的平台系统和项目管理信息、迁移的动机、迁移粒度等。

对现有系统的描述包括：系统的技术框架、功能框架、业务流程和逻辑、事务、应用和数据流、内部外部用户、单个应用的 QoS、SLA 等需求、功能部署平台、第三方组件或框架、软件配置和运行情况、程序和代码、外部接口、最低软硬件配置要求等信息。

迁移粒度主要有以下几种类型[21]：

1）迁移系统的单个功能，例如，使用 Google App Engine Datastore 代替本地的 MySQL 数据库。是目前最常用风险最低的云迁移方法，需要注意一些基本的配置、重构、适应性问题。

2）迁移系统的某个功能层，如数据层、业务逻辑层、前台桌面。如果需要迁移数据层，决策中需要着重考虑的是数据一致性、数据隔离和数据安全问题；如果迁移的是业务流程，决策中需要着重考虑的是模块间的依赖性、信息交互、事务一致性、相关功能的用户体验等。

3）迁移系统的单个业务应用，是云迁移的一种最典型的方式，把单个业务应用部署在云上执行，并封装成云服务。

4）迁移整个系统到云上，意味着全部的数据层、业务逻辑层都使用云服务进行重构。

5.3.2 决策分析

1. 迁移目标

对于需要判断是否迁移的应用或功能模块，首先需要确定迁移的目标，迁移目标包括但不限于：节约费用开销、系统升级、提高资源利用率、弹性扩展、提高可维护性，如图 5-2 所示，为文献[22]分析当前云迁移的研究领域主要关注的目标。

1）节约费用开销。

RightScale 公司 2017 年对全球 1000 多个企业用户使用云计算情况的调查报告显示，53%的企业选择云平台部署系统都是出于节约费用开销的目的[23]。节省的费用开销包括：（1）企业可以选择租用计算硬件、存储资源、网络带宽等资源来代替购买硬件构建本地数据中心。（2）应用或功能模块迁移到云端，可以省去对软硬件的日常维护费用，包括：人力资源方面，减少雇佣相关软硬件维护人员，节省薪水支出；能耗方面，减小机房规模，节省机房散热、照明、场地维护的费用。（3）由于业务的动态变化，系统对资源的需求也是动态变化的，企业往往需

要购买能够承受业务忙时峰值的资源，这在业务闲时形成了大量的空闲资源，造成浪费，云计算按需付费的方式允许企业根据应用实际需求租用云服务，在业务需求峰值时增加租用量，需求减少时相对减少租用量，节省了运营费用。

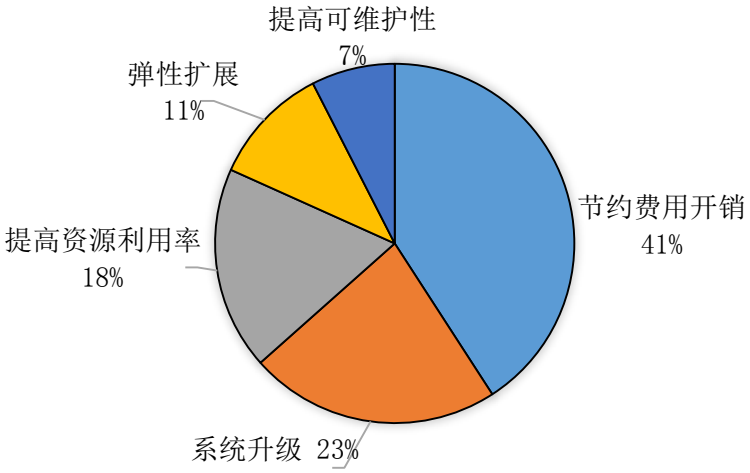


图 5-2 应用云迁移目标[22]

2）系统升级。

升级包括两方面，一方面是纵向升级，提高软硬件资源节点的性能，如提高计算节点的速度、存储节点的存储能力、网络节点的传输速度等，通过租用更高级的云服务，可以实现快速升级；另一方面是横向扩展，即增加资源节点数量，通过租用更多云服务，可以实现快速扩展。

3）提高资源利用率。

传统模式下，为了保障生产性应用的服务质量，一种应用通常独占单个或一组服务器，决定服务器数量的通常是应用负载或业务需求的峰值，而在日常运行中，实际负载通常仅占服务器容量的 10-50%[24]。云计算和虚拟化技术能够帮助整合多个应用，并实现资源隔离和性能保障，能够有效利用闲时剩余资源，提高整个数据中心的资源利用率。

4）弹性扩展。

公有云即用即付（pay-as-you-go）的模式意味着云计算的弹性扩展能力，在应用负载或业务需求动态变化时，动态调整资源节点数量，通过动态调整租用的云实例数量来实现。在业务需求峰值时增加租用量，需求减小时相对减少租用量。

5）提高可维护性。

在部分资源节点失效时，可维护性保证应用能够被正常访问。云计算和虚拟化技术提供了应用程序按照“虚拟机-镜像”的形式运行，同时，在应用运行过程中也可实时创建快照保存运行状态，当硬件、软件故障影响应用访问时，可以以镜像为模板重新启动一个实例，快速恢复运行[25]。

2 应用性能约束

应用或功能模块在云迁移后 QoS、SLA 性能指标不能降低。不同类型的应用有不同的性能约束。从应用的功能来说，事务型应用如 Web 应用、交互式应用，主要关注响应时间和吞吐率，非交互式的批处理任务主要关注完成时间，通常用截止时间做约束[26]。根据应用负载特征分类，CPU 密集型应用，也称计算密集型应用，大部份时间用来做计算、逻辑判断等 CPU 动作，通常 CPU 占用率较高，受资源节点的计算能力影响，主要约束是计算时间；I/O 密集型应用，如 Web server 的静态页面访问、基于数据库的应用等，需要大量 I/O 操作，对 CPU 占用率较低，受资源节点（硬盘/内存）的读写速度影响，主要约束是 I/O 时间；数据密集型应用，通常在分布式计算环境下处理高度密集的海量数据，主要受网络带宽、延迟等因素影响，约束是与数据传输相关的时间、延迟等。

3. 企业或系统的迁移约束

从企业角度考虑云迁移的约束问题，包括整个企业系统的安全问题、企业组织管理问题、此次云迁移的开销问题。

系统的安全性问题主要是由于部分应用或功能模块从系统中分离出来部署到云上，涉及到部分数据访问接口、应用调用接口暴露出来，相应的访问控制、数据鉴权、数据隔离、流程隔离等安全性保障需要加强。基于云计算和虚拟化技术的灾备管理需要更新。网络等原因造成的云服务失效、不可用、不可访问，会影响业务，以及系统其他应用和功能。如何确保可用性，也是需要考虑的重要安全问题。

企业的组织管理方面，需要分析企业业务和战略与云迁移是否一致；运维模式发生变化，传统的机房管理人员、软件运维人员，将转变成云服务的运维和管理人员，需要进行相关人员培训；另外，地方性政策法规也需要考虑，例如，不能将涉及国家、地方安全机密的数据存储在外国或其他敏感地区的云上。

表 5-1 企业系统混合云迁移各阶段的费用和时间开销

阶段	开销描述	费用	时间
前期规划	迁移策略制定		√
	应用或功能模块的迁移需要修改或申请新的牌照、制定新的合同	√	
开发阶段	雇佣新员工或聘请第三方开发商	√	√
	系统的切割上线、应用迁移和试运行		√
迁移阶段	应用的云迁移		√
	相关员工培训	√	√
运营维护阶段	云产品使用过程中的咨询和运维费用	√	

企业在进行系统的混合云迁移的整个周期内，需要完成一系列操作，如调研、设计、切割上线、运维等，这些活动需要花费一定的时间和金钱。企业需要事先预计时间和金钱方面的开销，确定是否符合企业目前战略安排。各阶段的费用和时间开销如表 5-1 所示。

4. 云服务选择

随着云计算的不断发展，国内外公有云 IaaS、PaaS、SaaS 产品越来越丰富，企业可以选择丰富的公有云产品或者在厂商的指导下部署私有云平台。选择私有云、公有云服务需要综合考虑以下约束：

1）效率。对于 IaaS 云，效率包括新的云服务从申请、创建、到返回结果的服务响应时间，和服务速度。例如，从 IaaS 云上租用一个新的虚拟机，从发出申请，经过虚拟机创建、配置，到提供商最终返回给用户使用接口的时间，即为响应时间，在使用虚拟机过程中，该虚拟机的计算速度，是云服务的速度。在 PaaS、SaaS 服务中，服务从申请、创建到返回给用户使用接口的时间，即为服务响应时间，服务速度是从发起单次服务请求，到返回服务结果的时间。

2）计费。云服务的费用是一个重要的指标，对于私有云，费用通常包括一次性软硬件购买安装费用，和长期的软硬件维护、机房能耗、运维人员薪水。对于公有云，主要是云产品按量、按时间的租金。Amazon 的计算服务 EC2 提供按使用时间的按需实例、基于月租的预留实例、只在闲时使用并按照使用时间计费的竞价型实例，数据传输以 GB 为单位计费[27]，存储服务 S3 根据不同存储容量区间按月计费[28]。Google Compute Engine 提供的计算、存储、大数据等服务均基于使用时间计费[29]，App Engine 提供的 PaaS、SaaS 云产品目前主要是免费的。国内的百度云目前提供 40 余款云服务产品，大多数为按月收费[30]。

3）可靠性。云服务需要具有可靠性，能够持续不间断的提供服务，服务效率等性能需要保持相对稳定，云服务面向用户的调用只有一个接口，屏蔽了底层实现，因此服务的动态迁移、升级等操作，应该对用户透明，不能影响服务质量。

4）适应性。云服务应该能够动态变更或升级，以应对用户的新需求，适应性一方面定义是否能够满足用户的新需求，另一方面衡量从新需求提出，到更新服务直至客户满意所需的时间。

5）可用度。衡量该项云服务培训、安装、操作、运维的难易程度，好的云服务应该是高可用的，方便没有任何基础的用户能够快速学习使用。

5.3.2 决策制定

对于云迁移的部分目标、企业系统或模块的迁移约束、部分云服务性能，都是定性的、不可量化的指标，同时部分优化指标互斥，本研究点基于 AHP，对决策分析阶段提取的迁移目标、应用性能和系统迁移约束进行层次化分解、定性指

标转化、重要性比较、初级解决方案生成，对于需要迁移到公有云的解决方案，再次进行公有云性能层次化分解、转化，获得最终解决方案。

首先对目标和约束进行分析，构造其递归化的层次结构，形成约束递归树；然后通过将递归树每层节点按照重要性两两比较，为每层递归树构造判断矩阵；最后计算各约束的重要性，并得到每个备选方案的权值，权值最高的备选方案即为解决方案。

1）构造约束递归树。首先对约束进行分析，构造其递归化的层次结构，形成约束递归树，根节点是目标层，包含应用进行云迁移需要实现的目标，中间为准则层，描述迁移约束，叶子节点为方案。对于多目标的迁移，根节点为空，根节点的下一层为多个目标，准则层的迁移约束也可按照大类-小类进行分层，对于初级迁移策略制定，方案包括不迁移、迁移到私有云、迁移到公有云，对于公有云服务选择的决策，方案为备选云服务。

2）构造判断矩阵。在约束递归树中，每个非根节点 m 都有一个判断矩阵 $\boldsymbol{A}_m$，将其子节点进行两两比较，他们相对于父节点的重要性比值为判断矩阵相应位置的元素值：

$$A_m=[r_{ij}]_{n\times n}=\begin{bmatrix}1 & r_{12} & \cdots & r_{1n}\\ r_{21} & 1 & \cdots & r_{2n}\\ \vdots & \vdots & \ddots & \vdots\\ r_{n1} & r_{n2} & \cdots & 1\end{bmatrix}=\begin{bmatrix}1 & r_{12} & \cdots & r_{1n}\\ 1/r_{12} & 1 & \cdots & r_{2n}\\ \vdots & \vdots & \ddots & \vdots\\ 1/r_{1n} & 1/r_{2n} & \cdots & 1\end{bmatrix} \tag{5-1}$$

其中，n 为节点 m 所包含的子节点的数目，i，j 分别表示节点 m 的子节点 i 和子节点 j，r_{ij} 表示子节点 i 与子节点 j 相对于所属节点 m 的重要度。对于可用数值计算的目标和约束，如费用、时间等，分别计算子节点对应的数值，相对重要度 r_{ij} 为子节点的比值。若两个子节点中至少有一个是不可量化的定性目标和约束，相对重要度 r_{ij} 取值根据心理学专家的定义[20]，取值范围是从 1 到 9 的整数，标示同样重要到非常重要，r_{ji} 表示节点 j 相对于 i 的重要性比值，故 $r_{ij}=1/r_{ij}$，节点与自身重要性相比总是 1，故矩阵对角线值为 1。通过上述方法，可以将无法用数值求解的定性约束转化成一组以数值表示的重要性判断矩阵，同时，实现了定量与定性指标的同一对比分析。

3）矩阵求解获得最佳方案。计算约束层各元素的权重，权重最高的备选方案即为最佳解决方案。

对于节点 m 的判断矩阵 $\boldsymbol{A}_m$，根据公式 $A_m\boldsymbol{x}=\lambda_{\max}\boldsymbol{x}$ 可以求得特征向量 $\boldsymbol{x}$ 和特征值 $\lambda_{\max}$，特征向量 $\boldsymbol{x}=\langle x_1,x_2,\cdots x_n\rangle$ 的元素 x_i 即为节点 m 的第 i 个子节点对应的单层权重，用 W_i^k 表示，k 表示子节点 i 所在的层。

需要衡量判断矩阵 $\boldsymbol{A}_m$ 一致性，即满足传递性 $r_{ij}=r_{ik}\times r_{kj}$。对于一致性比率 CR（Consistency Ratio），若 $CR\leq 0.1$，则认为判断矩阵具有一致性，据此而计算的值

是可以接受的，若 CR>0.1 则判断矩阵不具有一致性，需要重新设计判断矩阵元素值[20]。

$$CR = CI / RI = \frac{(\lambda_{\max} - n)/(n-1)}{RI} \tag{5-2}$$

其中，CI 为判断矩阵的一致性指标，RI 为判断矩阵的随机一致性指标，取值如表 5-2 所示[20]。

表 5-2 RI 取值表（n 表示节点 m 的子节点个数）

n	1	2	3	4	5	6	7
RI	0	0	0.52	0.89	1.12	1.26	1.36
n	8	9	10	11	12	13	14
RI	1.41	1.46	1.49	1.52	1.54	1.56	1.58

确定了每层的权重后，自目标层向方案层构造层次总排序 R_i^k，第 k 层的 R_i^k 计算如下：

$$R_i^k = \sum_{j=1}^{m} W_i^k R_j^{k\text{-}1}, \quad (i = 1, 2, ..., n) \tag{5-3}$$

其中，R_i^k 表示第 k 层的节点 i 的层次总排序值，R_j^{k-1} 表示第 k-1 层的节点 j 的层次总排序值，W_i^k 表示第 k 层的节点 i 对应的权重，$R_i^1 = W_i^1$。

层次总排序的一致性检验。设第 k-1 层共有节点 m 个，第 k 层共有节点 n 个，第 k 层节点 i 的对应于上一层节点 j 的单层一致性指标 CI_j^{k-1}，单层随机一致性指标为 RI_j^{k-1}，则其层次总排序一致性比率 CR_i^k 的计算方法如下：

$$CR_i^k = \frac{\sum_{j=1}^{m} CI_j^k R_j^{k\text{-}1}}{\sum_{j=1}^{m} RI_j^k R_j^{k\text{-}1}}, \quad (i = 1, 2, ..., n) \tag{5-4}$$

层次总排序通过一致性检验后，权重最大的备选方案即为最终方案。

5.4 系统混合云迁移实例分析

目前，国内网络服务，如电子商务、数字医疗、在线教育等，发展迅猛，服务需求多种多样，软硬件及网络等服务环境日益复杂化，虚拟服务需求不断增加。传统服务运营支撑系统采用定制资源、定制架构、定制应用的定制开发模式，将面临服务系统运维、业务扩展、系统扩容难度大、费用高等多方面问题，无法实现高效率、低成本地规模化交付服务，并满足灵活、多变的个性化服务的需求。因此，需要引进云计算技术，实现网络服务运营支撑系统云化，应对新业务模式、新技术和新架构带来巨大的机遇和挑战。

作为用例，本文分析了国内某网络服务运营支撑系统的云化需求。通过内部专家讨论，确定如图 5-3 所示的运营支撑系统混合云迁移策略制定的递归树，本文针对运营支撑系统模块的迁移约束，将目标、性能、约束分成性能、可用性、安全、经济、管理五个方面。

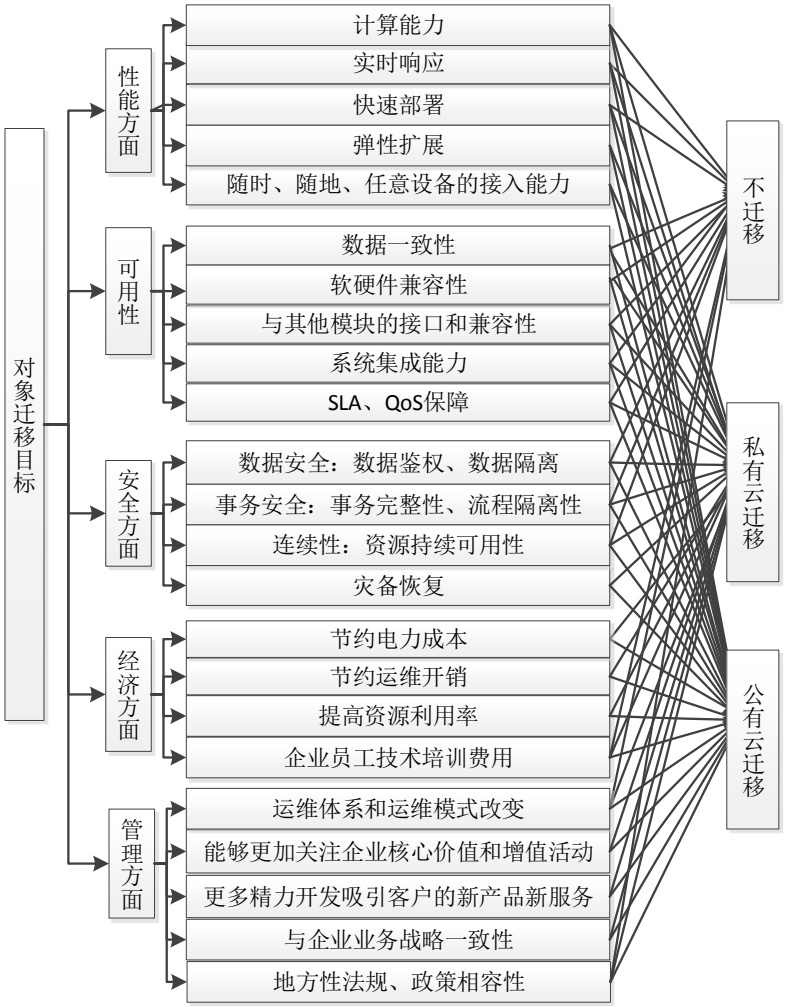


图 5-3 运营支撑系统混合云迁移的约束递归树

以系统各应用模块为基本迁移粒度，分析和制定决策。使用“Super Decision”软件[21]进行计算得到各约束的权重，并通过一致性检测，确定判断矩阵的合理性，以计费应用为例，迁移约束的 level1 权重如图 5-4 所示，各影响因素排序结果为安全、性能、可用性、经济、管理因素，同时，一致性指标为 0.0353，满足一致性约束。

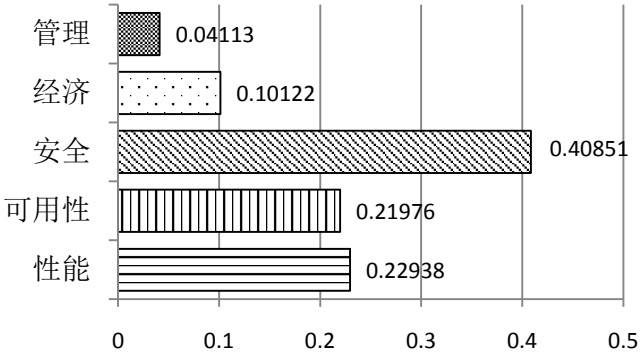


图 5-4 计费应用云迁移 level1 层的约束权重

level2 层各约束相对于 level1 层的相对权重按重要程度依次排序为：1）安全因素，数据安全（0. 35091）、事务安全（0.35091）、灾备（0. 18906）、业务连续性（0.10911）；2）性能因素，计算能力（0.38539）、实时响应能力（0.38537）、弹性扩展（0.14253）、多种接入（0.05628）、快速部署（0.03043）；3）可用性因素，SLA、QoS 保障（0.46563）、数据一致性（0.31875）、软硬件兼容（0.12241）、集成能力（0.06093）、接口兼容（0.03228）；4）经济因素，电力成本（0.39505）、资源利用率（0.39499）、运维开销（0.16268）、培训费用（0.04729）；5）管理因素，法律法规（0.46038）、企业战略（0.31859）、运维模式（0.12688）、关注核心活动（0.04045）、关注新产品和新服务开发（0.0537）。层次总排序如图 5-5 示，重要程度依次为数据安全、事务安全、计算能力、实时响应、SLA 和 QoS 保障等。

三种迁移方案的权重计算结果为不迁移 0.476069，迁移到私有云 0.346121，迁移到公有云 0.177810。从结果的反馈来看，计费应用目前暂时不需要迁移，在未来规划和测试通过后，可以考虑迁移到私有云上。

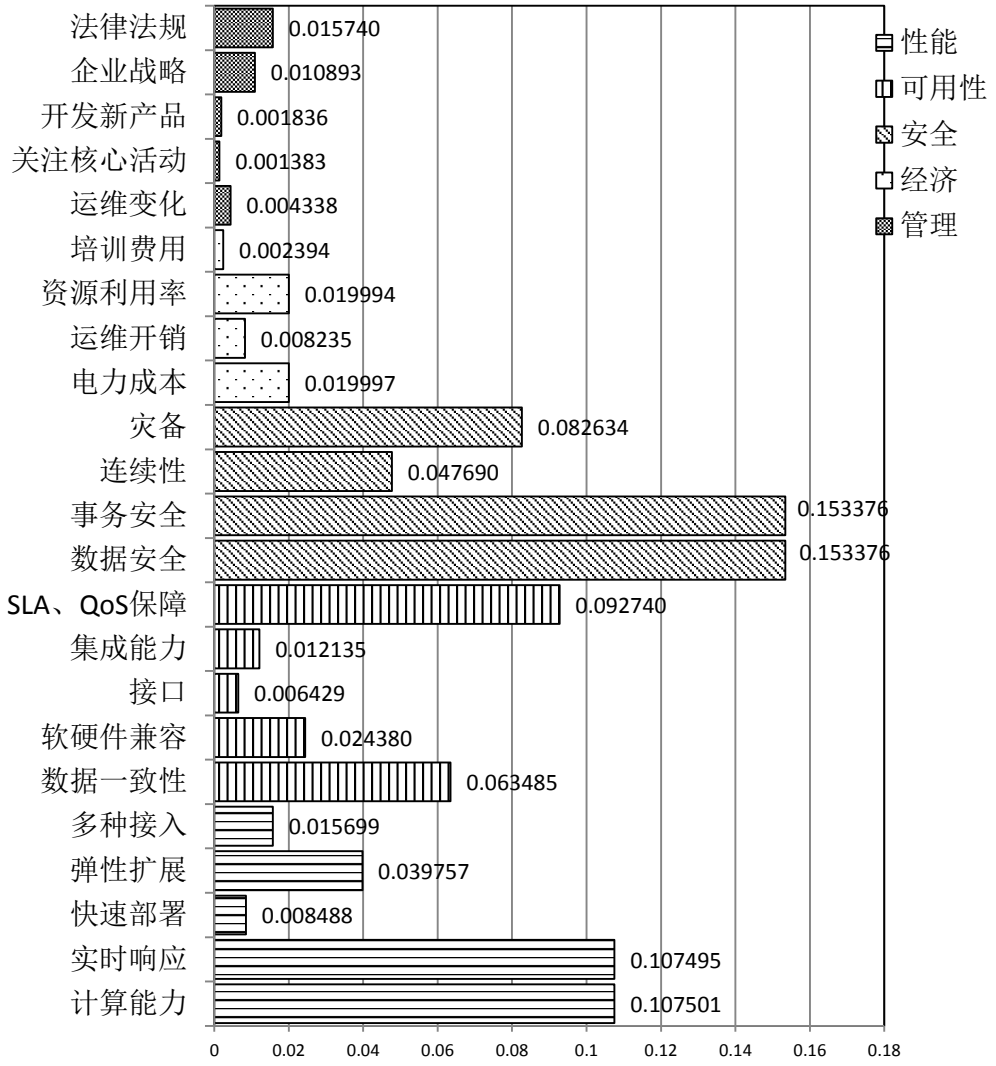


图 5-5 计费应用云迁移各约束的层次总排序

对运营支持系统其他主要应用进行讨论，确定递归树中各约束的权重，计算判断矩阵并通过一致性检测，各应用的三种迁移备选方案对应的标准权重结果如图 5-6 示。

- 对实时性和安全性要求很高、数据量和计算量很大的关键应用，如订单管理、问题处理、实时计费暂不迁移；
- 实时性或计算量要求稍有放松的关键和重要应用，如用户界面、销售管理、渠道管理、帐详单查询、服务开通、资源管理、S/P 结算可以迁移到企业已经搭建成熟的私有云平台；
- 一般系统如市场营销、优惠与促销、服务质量管理、综合采集、S/P 采购等，实时性要求不高、故障影响低，可以选择公有云提供商迁移。

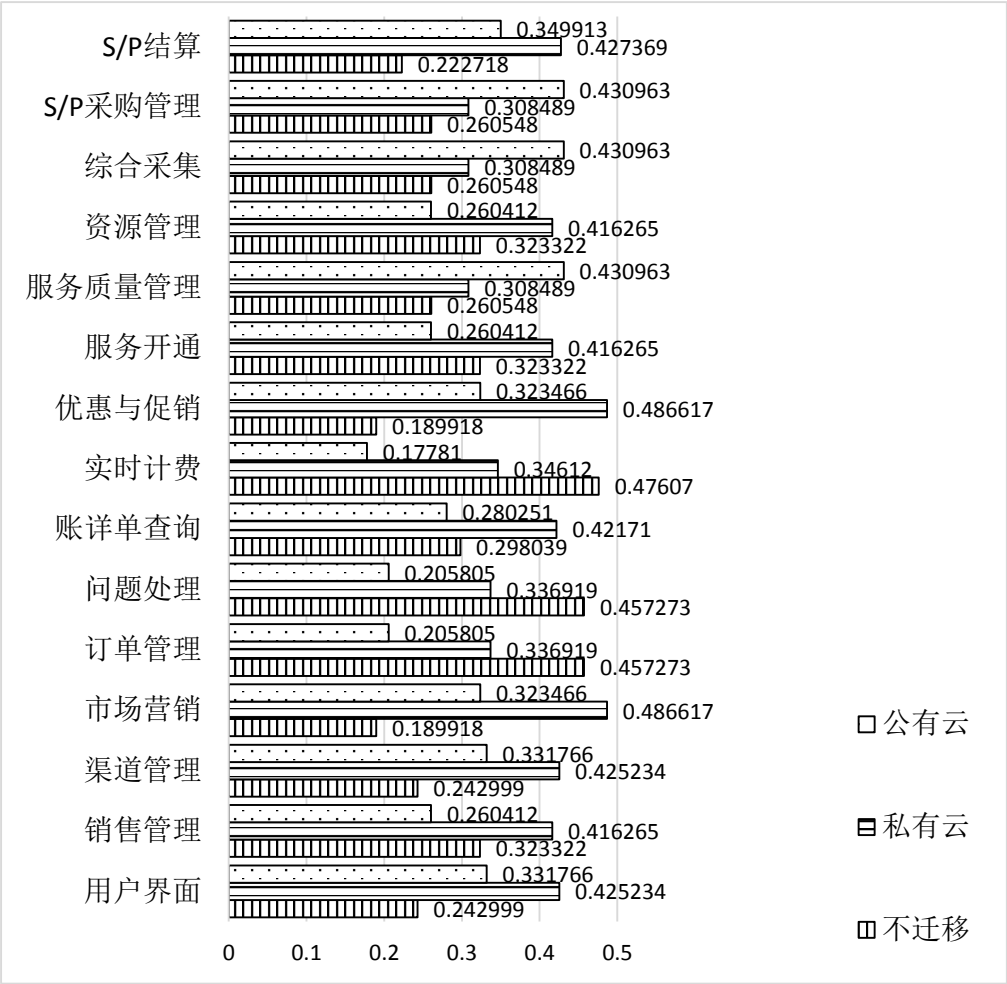


图 5-6 各应用模块三种迁移方案对应的标准权重

5.5 本章小结

混合云迁移方式允许企业根据应用的安全和性能要求，将一部分模块迁移到云上，剩下的保留在原有系统中，节省云迁移的资金和消耗，同时降低风险。本章主要提出一种制定企业系统混合云迁移的策略的步骤和方法。首先定义循环演进的企业系统混合云迁移步骤：评估、设计、决策、试运行、测试与优化、迁移，在测试策略不可行且优化失败时，返回设计阶段；然后重点分析迁移目标、约束、云服务性能，最后基于 AHP 方法对目标、约束、云服务性能进行层次化分解，将定量约束与定性的目标、约束统一对比获得相关重要度指标，求解并生成决策。作为用例，分析网络服务运营支撑系统云化的需求，利用本文提出的方法，制定了运营支撑系统混合云迁移策略。此外，本方法作为指导思想，应用在国内某电信运营商运营支撑系统云化项目的需求分析和设计阶段。

参考文献

[1] Hajjat M, Sun X, Sung Y-W E, et al. Cloudward bound: planning for beneficial migration of enterprise applications to the cloud [A]. // Proceedings of the ACM SIGCOMM 2010 conference, New Delhi: ACM Press, 2010: 243-254.

[2] Andrikopoulos V, Song Z, Leymann F, Supporting the Migration of Applications to the Cloud through a Decision Support System [A]. // Proceedings of IEEE Sixth International Conference on Cloud Computing [C], Santa Clara: IEEE Press, 2013: 565-572.

[3] Kaviani N, Wohlstadter E, Lea R. MANTICORE: a framework for partitioning software services for hybrid cloud [A]. // Proceedings of IEEE 4th International Conference on Cloud Computing Technology and Science [C], Taipei: IEEE Press, 2012: 333-340.

[4] Qiu X, Li H, Wu C, et al. Cost-minimizing dynamic migration of content distribution services into hybrid clouds [J]. IEEE Transactions on Parallel and Distributed Systems, 2015, 26 (12): 3330-3345.

[5] Huang D, Yi L, Song F, et al. A secure cost-effective migration of enterprise applications to the cloud [J]. International Journal of Communication Systems, 2014, 27 (12):3996-4013.

[6] Tak B C, Urgaonkar B, Sivasubramaniam A. To move or not to move: the economics of cloud computing [A], //Proceedings of the 3rd USENIX conference on Hot topics in cloud computing [C], Portland: USENIX Press, 2011: 1-6.

[7] Khajeh-Hossein A, Sommerville I, Bogaerts J, et al. Decision support tools for cloud migration in the enterprisen[A]. //Proceedings of IEEE International Conference on Cloud Computing [C], Washington DC: IEEE Press, 2011: 541-548.

[8] Andrikopoulos V, Strauch S, Leymann F. Decision support for application migration to the cloud: challenges and vision [A]. //Proceedings of the International Conference on Cloud Computing and Services Science [C], Aachen: Springer Press, 2013:1-8

[9] Strauch S, Andrikopoulos V, Bachmann T, et al. Decision support for the migration of the application database layer to the cloud [A]. // Proceedings of IEEE 5th International Conference on Cloud Computing Technology and Science [C], Bristol: IEEE Press, 2013: 639-646.

[10] Strauch S, Andrikopoulos V, Karastoyanova D, et al. Migrating enterprise applications to the cloud: methodology and evaluation [J], International Journal Big Data Intelligence, 2014, 1: 127-140.

[11] Andrikopoulos V, Darsow A, Karastoyanova D, et al. CloudDSF - the cloud decision support framework for application migration [A]. // Proceedings of the European Conference on Service-Oriented and Cloud Computing: Service-Oriented and Cloud Computing [C], Vienna: Springer Press, 2014:1-16.

[12] Menzel M, Ranjan R, Wang L, et al. CloudGenius: A Hybrid Decision Support Method for Automating the Migration of Web Application Clusters to Public Clouds [J]. IEEE Transactions on Computers, 2015, 64 (5): 1336-1348.

[13] Beserra P V, Camara A, Ximenes R, et al. Cloudstep: a step-by-step decision process to support legacy application migration to the cloud [A]. // Proceedings of IEEE 6th International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems [C], 2012: 7-16.

[14] Michael M, Marten S, Stefan T. $(MC^2)^2$: criteria, requirements and a software prototype for cloud infrastructure decisions [J]. Software: Practice and Experience, 2013, 43 (11): 1283-1297.

[15] Shawky D M. A cost-effective approach for hybrid migration to the cloud [J]. International Journal of Computer and Information Technology, 2013, 2 (1): 57-63.

[16] Alkhalil A, Sahandi R, John D. Migration to cloud computing: a decision process model [A]. //Proceedings of the Central European Conference on Information and Intelligent Systems [C], Varaždin: ELSEVIER Press, 2014:1-10.

[17] Alkhalil A, Sahandi R, and D. A decision process model to support migration to cloud computing [J], International Journal of Business Information Systems, 2016, 24 (1): 102-126.

[18] Juan-Verdejo A, Baars H. Decision support for partially moving applications to the cloud - the example of business intelligence [A]. // Proceedings of the 2013 international workshop on Hot topics in cloud services [C], Prague: ACM Press, 2013: 35-42.

[19] Juan-Verdejo A, Zschaler S, Surajbali B,et al. InCLOUDer: a formalised decision support modelling approach to migrate applications to cloud environments [A]. //Proceedings of the 40th Euromicro Conference on Software Engineering and Advanced Applications [C], Verona: IEEE Press, 2014: 467-474.

[20] 孙宏才, 田平,王莲芬, 网络层次分析法与决策科学[M]. 北京: 国防工业出版社, 2011: 33-39.

[21] Vasilios A, Tobias B, Frank L, et al. How to adapt applications for the cloud environment [J], Computing, vol. 95 (6), 2013: 493-535.

[22] Jamshidi P, Ahmad A, Pahl C. Cloud Migration Research: A Systematic Review [J]. IEEE Transactions on Cloud Computing, 2013,1(2): 142-157.

[23] RightScale, State of the Cloud Report [EB/OL], https://www.rightscale.com/lp/state-of-the-cloud?campaign=70170000000vFyh, 2017-04-01.

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

[25] 贾佩, 基于 Eucalyptus 的云计算应用迁移与部署研究开发[D], 西安：西安理工大学, 2014: 21

[26] Balaji P, Sadayappan P, Islam M, Techniques for providing hard quality of service guarantees in job scheduling [M]. New Jersey: Wiley Press, 2009: 403-425

[27] Amazon EC2 定价 [EB/OL], https://aws.amazon.com/cn/ec2/pricing/, 2017-04-01.

[28] Amazon S3 定价 [EB/OL], https://aws.amazon.com/cn/s3/pricing/, 2017-04-01.

[29] Google Cloud Platform Pricing [EB/OL], https://cloud.google.com/pricing/, 2017-04-01.

[30] 百度云 [EB/OL], https://cloud.baidu.com, 2017-04-01.

第六章 总结和展望

6.1 论文总结

随着云产品的不断推出，越来越多的企业系统使用云计算技术进行部署，企业的云环境日益复杂。私有云数据中心改变了传统数据中心服务器只能运行单个应用的情况，不同 QoS、SLA 需求的异构任务可以整合在同一个私有云数据中心，这样的异构任务可以划分为不同的优先级，调度和管理需要侧重满足高优先级任务的性能。云平台资源具有异构性、动态性、可能失效等不确定因素，在这样的不可靠云平台上调度工作流，需要考虑不确定因素，才能提高调度的健壮性。随着大数据应用的发展，数据密集型工作流日益增加，如何将数据密集型工作流调度到混合云上，不只需要满足以往混合云调度费用开销最小化的目标，还需要确保不同云平台之间数据传输量最小。当前企业系统云化需求不断提高，但是缺少完备的迁移策略制定方案，能够指导企业按步分析确定系统各个功能模块是否适合迁移、选择哪个云产品进行迁移。针对上述问题，本文提出了将异构的相互独立的任务调度到单个私有云数据中心的调度与资源管理算法、将工作流动态的调度到不可靠的私有云平台的算法、将持续到达的数据密集型工作流动态的调度到混合云的算法、指导企业系统进行混合云迁移的策略制定步骤和方法，具体研究点如下：

1）提出了基于混合预测的异构任务调度和管理方法，MPHW(Multi-Prediction based scheduling for Heterogeneous Workloads)。针对混合了不同 QoS 和 SLA 需求的异构任务的私有云数据中心，在保障高优先级任务的性能的同时，提高了数据中心的资源利用率。异构任务被分为高、中、低三个优先级，时间连续的负载被划分为离散的时间序列。对于中等优先级任务，使用离线训练的 ARMA 模型对平稳分布的生产性负载序列进行预测，资源可用量为当前可用量减去未来时间段内生产性负载增加量，如果可用资源不足，则随机选择一个主机抢占其上的最低优先级任务直至释放的资源量足够调度该任务。对于最低优先级任务，使用基于在线反馈的 AR 模型对不平稳分布的主机负载序列进行预测，可用资源量为未来时间段内主机可用资源的最小值，如果可用资源不足，则将任务放至等待队列，直至资源足够时再提交调度。仿真实验表明，基于混合预测的异构任务调度算法能够减小 70%主机过负载和任务失败次数，能够减少超过 50%的资源浪费，有效资源利用率提高超过 65%。此外，虽然调度延迟有所增加，但是对于 QoS、SLA 需求低的任务来说，是可以忍受的。

2）提出了科学工作流在不可靠的私有云上的动态调度算法，DEFT (Dynamic Earliest-Finish-Time)算法。应对资源性能异构、动态变化、有失效可能等不确定性因素混合的不可靠云平台，在完工时间最小的同时，具有健壮性。任务只在所有父任务都完成之后，才进入就绪状态，根据当前系统的性能选择资源进行调度。DEFT 包括一组调度循环，每次有任务完成且有空闲资源时，进行一次调度操作。在每个调度操作中，所有未调度的任务根据当前资源速度，计算 $rank_u$ 值，并依据 $rank_u$ 降序排列，每个未调度任务按序映射到提供最小结束时间的虚拟机资源，就绪状态的任务可以调度到相应的虚拟机，而其他任务只保留到虚拟机的映射关系，并传输中间数据。用变异系数 CV 计算调度的健壮性，用以衡量不可靠云环境下调度机制能否产生平稳的、最小的完工时间。实验表明，使用 DEFT 比现有的典型的基于任务列表的静态调度算法 PEFT 和动态调度算法 DCP-G，工作流完工时间缩短超过 20%，且健壮性更好。

3）一组数据密集型工作流在混合云上的动态调度算法，HCOD(Hybrid Cloud Optimized Data scheduling algorithm)。针对持续到达的有截止时间约束的数据密集型科学工作流，提出了动态混合调度算法，使得公有云和私有云之间数据传输量最小，公有云的费用开销最低。工作流使用双层禁忌搜索图划分算法 DLTS 进行快速划分，确保子工作流之间数据传输量最小。到达的工作流首先根据 FCFS 或 EDF 原则在私有云上排队。若私有云资源空闲，将就绪子工作流按照 EDF-EFT 策略调度到私有云上。将工作流的截止时间约束映射成各任务的最迟完成时间约束 LFT，周期性遍历等待队列，估算各任务完成时间，如果晚于 LFT，则直接选择该任务的子工作流，或者选择排队在该任务之前、PrivateCut 最小的子工作流，使用 HLCF 算法调度到公有云实例上。仿真实验分析了 DLTS 工作流划分算法的性能，并对比了 2 种排序策略 2 种选择策略下，调度的截止时间满足率、费用开销、数据传输量等的性能。

4）提出了一种规范化的企业系统进行混合云迁移的策略制定方法。混合云迁移方式允许企业根据应用的安全和性能要求，将一部分模块迁移到云上，剩下的保留在原有系统中，节省云迁移的资金和消耗，同时降低风险。首先定义循环演进的企业系统混合云迁移步骤：评估、设计、决策、试运行、测试与优化、迁移，在测试策略不可行且优化失败时，返回设计阶段；然后重点分析迁移目标、约束、云服务性能，最后基于 AHP 方法对目标、约束、云服务性能进行层次化分解，将定量约束与定性的目标、约束统一对比获得相关重要度指标，求解并生成决策。分析了网络服务运营支撑系统云化的需求，利用本文提出的方法，制定了运营支撑系统混合云迁移策略。此外，本方法作为指导思想，应用在国内某电信运营商运营支撑系统云化项目的需求分析和设计阶段。

6.2 进一步工作

随着云计算的不断发展，各种公有云、私有云、混合云产品的不断推出，部署在云上的应用种类越来越多，企业的云环境日益复杂，混合云环境还有很多技术问题亟待解决。本文针对混合云环境的任务调度与资源管理的几个关键问题展开研究，这些研究还很不全面，下一步需要深入研究的工作有如下几点：

1）为新任务计算可用资源时，增加干扰因素对性能的影响分析，干扰主要包括部署在同一虚拟机上多个应用的干扰、部署在同一主机上的多个虚拟机的干扰。

2）对于不可靠云平台的科学工作流调度，估算运行期任务完成时间时，只是根据当时资源计算和传输速度，也没有考虑节点失效可能，未来考虑引入对资源速度、节点失效可能性的预测，以求提高任务完成时间估算的准确度。

3）目前只研究调度数据密集型科学工作流的问题，未来将研究混合工作流的动态调度模型，包括数据密集型、计算密集型、I/O 密集型，对等待队列遍历频率、重调度对象的选择策略，都有更复杂的要求。

4）在企业系统混合云迁移策略制定中，我们基于 AHP 方法对目标、约束、云服务进行层次化分析、量化求解。而部分目标、约束并非严格的树形结构，而是构成网络结构，下一步考虑进一步分析目标、约束之间的关系，研究 ANP（Analytic Network Process，网络分析法）在混合云迁移决策制定中的适用度。

附录　缩略语

英文缩写	英文全称	中文解释
AHP	Analytic Hierarchy Process	层次分析法
AIC	Akaike Information Criterion	最小信息量准则
AM	Application Master	应用管理器
ANN	Artificial Neural Network	人工神经网络
ANP	Analytic Network Process	网络分析法
API	Application Programming Interface	应用程序编程接口
App	Application	应用
AR	Auto Regressive	自回归
ARMA	Auto Regressive Moving Average	自回归移动平均
ARIMA	Autoregressive Integrated Moving Average Model	差分自回归移动平均
AWS	Amazon Web Services	亚马逊 Web 服务
BIC	Bayesian Information Criterion	贝叶斯信息准则
CI	Consistency Index	一致性指标
CDF	Cumulative Distribution Function	累积分布
CP	Critical Path	关键路径
CPOP	Critical Path on a Processor	处理器上的关键路径
CPU	Central Processing Unit	中央处理器
CV	Coefficient of Variations	变异系数
CR	Consistency Ratio	一致性比率
CRM	Customer Relationship Management	客户关系管理
DAG	Directed Acyclic Graph	有向无环图
DAX	Directed Acyclic Graph in XML	XML 描述的 DAG
DCP-G	Dynamic Critical Path for Grid	网格环境下的动态关键路径
DEFT	Dynamic Earliest Finish Time	动态最早完成时间
DLTS	Double-Level Tabu Search graph partition	双层禁忌搜索图划分
DLTT	Data Latest Transmission Time	数据最晚传输时间
DRF	Dominant Resource Fairness	域资源公平调度算法
EC2	Elastic Compute Cloud	弹性计算云
EDF	Earliest Deadline First	最早截止时间优先

EFT	Earliest Finish Time	最早结束时间
EST	Earliest Start Time	最早开始时间
FCFS	First Come First Service	先来先服务
FOAR	Feedback based Online AR	在实时线反馈的自回归
FT	Finish Time	完成时间
NSGA-II	Non-dominated Sorting Genetic Algorithm II	带精英策略的非支配排序的遗传算法
GB	GigaByte	千兆字节
HCOC	Hybrid Cloud Optimized Cost	混合云的费用优化
HCOD	Hybrid Cloud Optimized Data	混合云的数据最优化
HCPT	Heterogeneous Critical Parent Trees	异构的关键父树
HEF	Heterogeneous Earliest Finish Time	异构的最早完成时间
HEM	Heavy-Edge Matching	重边匹配
HIAP	Hierarchical Iterative Application Partition	结构化的迭代的应用划分
HLCF	Heterogeneous Least Cost First	异构的最小开销优先
HPS	High Performance task Scheduling	高性能的任务调度
IaaS	Infrastructure-as-a-Service	基础设施即服务
I/O	Input/Output	输入/输出
LFT	Latest Finish Time	最晚结束时间
MA	Moving Average	移动平均
MAPE	Mean Absolute Percent Error	平均绝对百分误差
MB	MByte	兆字节
MCGP	Multi-Constraint Graph Partitioning	多约束的图划分算法
MCT	Minimum Completion Time	最小完工时间
MILP	Mixed-Integer Linear Programming	混合整数线性规划
MIPS	Million Instructions Per Second	每秒处理的百万级的机器语言指令数
MPHW	Multi-Prediction based scheduling for Heterogeneous Workloads	基于混合预测的异构负载调度
MVCC	Multi-Version Concurrency Control	多版本并发控制
NM	Node Manager	节点管理器
NP	Non-deterministic Polynomial	非确定多项式
NSL	Normalized Schedule Length	归一化的调度长度

PaaS	Platform-as-a-Service	平台即服务
PCH	Path Clustering Heuristic	路径聚类启发式算法
PEFT	Predict Earliest Finish Time	预测最早完成时间
PETS	low complexity Performance Effective Task Scheduling	低复杂度的性能高效的任务调度
P-HEFT	HEFT for parallel tasks	并行任务的 HEFT 算法
QoS	Quality of Service	服务质量
OCT	Optimistic Cost Table	最优开销表
OS	Operation System	操作系统
REST	Representational State Transfer	具象状态传输
RI	Random Index	随机一致性指标
RM	Resource Manager	资源管理器
RSD	Relative Standard Deviation	相对标准偏差
RUP	Resource Usage Percentage	资源使用率
S3	Simple Storage Service	简单存储服务
SaaS	Software-as-a-Service	软件即服务
SDBATS	Standard Deviation Based Algorithm for Task Scheduling	基于标准差的任务调度算法
SLA	Service Level Agreement	服务等级协议
S/P	Supplier/Provider	供应商/合作伙伴
ST	Submit Time	提交时间
TCHC	Time and Cost optimization for Hybrid Clouds	混合云的时间和费用优化
vCPU	Virtual CPU	虚拟 CPU
VM	Virtual Machine	虚拟机
VPC	Virtual Person Computer	虚拟个人电脑
WS-Security	Web Service Security	Web 服务安全
WSDL	Web Service Description Language	Web 服务描述语言
XML	eXtensible Markup Language	可扩展标记语言
ZB	Zetta Byte	十万亿亿字节

致　谢

在北京邮电大学 PCN&CAD 中心度过的几年硕士博士生活，我从学业到生活，都成熟了许多，为将来的工作、生活都打下了坚实的基础。在这里，我要衷心的感谢几年来帮助过我的老师和同学们。

感谢我的博士导师宋俊德教授和硕士导师宋美娜教授，为我提供了宝贵的参与项目和课题的机会，正是实际的项目训练了我运用所学知识分析解决问题的能力。宋俊德教授渊博的知识，严谨的治学态度，深邃的洞察力和对事业孜孜不倦的追求，是我一生学习的楷模，他豁达的性格启发了我对生活态度的改变，让我在面对困难的时候能够泰然处之。宋美娜教授在我硕士博士期间的学习、工作和生活上都给与了我极大的支持和帮助，她忘我的工作精神，认真负责的工作态度，时时激励着我不断进取，让我明白如何学习、如何做事、如何做人。

感谢鄂海红老师在我硕士和博士前期带领我参与实验室的项目和课题，一步一步悉心指导我如何带领师弟师妹们实施和完成项目、如何有效与合作方沟通和交流，带领我走过前期的茫然与无措。博士后期又花时间细致的指导我的博士论文工作。每每与鄂老师交流，我总能获得新的知识和鼓舞，鄂老师给予我的关爱，让我受益匪浅，终生难忘。

感谢实验室黎燕、于艳华、李艳丽、欧阳中洪、许可等老师对我的关心和帮助。感谢鄂新华博士、乐冠博士、童俊杰博士、李健博士先后在学业、课题项目以及学术论文上对我的指导和无私帮助。感谢几年来与我朝夕相处、相互鼓励、克服博士前期的茫然、中期的焦虑、后期的辛苦的同学和舍友王婷、陈诚、张铮，感谢亲密合作共事过的已经毕业的师兄师姐们，还没有毕业的师弟师妹们，以及和我同级的兄弟姐妹们。

感谢我的父母和丈夫，感谢他们多年来的支持和关怀，他们给了我读博的勇气和决心，容忍我科研不顺时的暴躁与无理，在我压力大时不断的鼓励我，在我延期时不断的开导我。

感谢每一个曾经关心我、帮助过我的人，没有他们，就没有成为合格博士的我。

攻读学位期间发表的学术论文

[1] Haiou Jiang, Haihong E, Meina Song, Dynamic Scheduling of Workflow for Makespan and Robustness improvement in the IaaS Cloud, IEICE TRANSACTIONS on Information and Systems, 2017, E100-D (4): 813-821，(SCI: 000399371100025)

[2] Haiou Jiang, Haihong E, Meina Song, Multi-Prediction based Scheduling for Hybrid Workloads in the Cloud Data Center, 投稿 SCI 刊源 Cluster Computing，目前 2 审后 minor review 修改中

[3] Haiou Jiang, Haihong E, Meina Song, Hierarchical Prediction based Task Scheduling in Hybrid Data Center, Proceedings of the International Conference on Parallel and Distributed Systems, 2015:17-24, (EI:20154601537525)

[4] 姜海鸥,宋美娜,鄂海红,邓江东. 运营支撑系统混合云迁移策略制定方法, 北京邮电大学学报, 2014,37(5): 1-5, (EI: 20145100353908)

[5] Haiou Jiang, Haihong E, Meina Song, Junde Song, A Generic Method of Migrating Enterprise Components Partially to the Cloud, Pervasive Computing and the Networked World Lecture Notes in Computer Science, vol. 8351, 2014: 206-217，(EI:20143118009286)

[6] Haiou Jiang, Haihong E, Meina Song, Online Scheduling for Data-Intensive Workflows on Hybrid Clouds, 已投稿

攻读学位期间申请的发明专利

[1] 鄂海红,宋美娜,姜海鸥,常倩,黄岩,舒琴,刘虹,宋俊德,江周峰. 迁移方法和装置. CN 103780688 A

[2] 鄂海红,黄岩,宋美娜,常倩,姜海鸥,宋俊德,舒琴. 一种资源调度的方法、装置和系统. CN 105808341 A

攻读学位期间参与的科研项目

[1] 面向互联网的业务支撑系统关键技术及方案研究，教育部-中国移动科研基金项目（项目编号：MCM20123031）

[2] 劳动者就业信息服务关键技术及服务模式研究，国家科技支撑计划（项目编号：2013BAH10F01）

[3] 公众保险一站式服务体系研究与系统开发，国家科技支撑计划（项目编号：2014BAH26F02）

[4] 跨域大数据关键技术架构研究，企事业合作项目

[5] 基于移动互联网大数据的业务精细洞察方法，企事业合作项目