浙江工业大学

硕士学位论文

论文题目：　面向云边协同的节点调度研究与实现

作者姓名　平靖

指导教师　张美玉教授

第二导师　简峥锋副教授

学科专业　软件工程

学位类型　工程硕士

培养类别　全日制专业学位硕士

所在学院　计算机科学与技术学院

提交日期：2020 年 06 月

Research and Implementation on Cloud Edge-based Node Scheduling

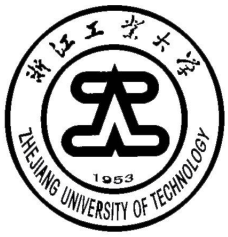
Dissertation Submitted to

Zhejiang University of Technology

in partial fulfillment of the requirement

for the degree of

Master of Engineering



by

Dissertation Supervisor: Prof.

Associate Supervisor: Associate Prof.

June., 2020

浙江工业大学学位论文原创性声明

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

作者签名：　　日期：　　年　　月

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权浙江工业大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于 1、保密□，在一年解密后适用本授权书。

2、保密□，在二年解密后适用本授权书。

3、保密□，在三年解密后适用本授权书。

4、不保密□。

（请在以上相应方框内打“√”）

作者签名：　　日期：　　年　　月

导师签名：　　日期：　　年　　月

中图分类号　TP391　　学校代码　10337

UDC　004　　密级　公开

研究生类别　全日制专业型硕士研究生

浙江工业大学

工程硕士学位论文

面向云边协同的节点调度研究与实现

Research and Implementation on Cloud Edge-based Node Scheduling

作　　者　平靖　　第一导师　张美玉

申请学位　工程硕士　　第二导师　简峥锋

学科专业　软件工程　　培养单位　计算机科学与技术学院

研究方向　边缘计算　　答辩委员会主席　***

答辩日期：　2020　年　　月　　日

面向云边协同的节点调度研究与实现

摘　要

近几年随着边缘计算的快速发展，越来越多的计算任务由传统的云计算中心向边缘计算节点迁移，面向云边协同的计算需求越来越突出。由于边缘异构计算资源的复杂性和不确定性，云边协同的节点调度是影响云边协同计算性能的关键。

当前面向云边协同的节点调度面临如下问题：（1）在云边协同环境中，面对不同类型的任务需要不同的策略（上传云中心或边缘端调度），云边协同的调度问题不再是传统的云中心集中批处理的调度模式，而是要综合考虑两种计算模式的需求，进行全局调度；（2）与传统云中心调度策略的“自上而下”不同，云边协同的调度策略是“自下而上的”，即用户与边缘节点交互，由边缘节点调度决策（上传或调度迁移到其他边缘节点）；（3）由于边缘节点的计算资源有限，如果进行批次调度会导致应用程序的延迟增加从影响用户体验。

可见，传统的云计算中心集中式调度策略已不再适合云边协同场景。论文针对上述问题，提出了面向云边协同的基于改进蝙蝠算法的实时调度模型，并在此基础上研究了 Storm 的边缘节点调度优化方法。主要工作如下：

1. 针对云边协同节点调度需求，提出了基于改进蝙蝠算法的实时调度模型。并结合 EdgeCloudSim 边缘计算仿真平台搭建调度仿真模型进行理论论证。

2. 在上述基础上研究了面向云边协同的 Storm 边缘节点调度优化方法及具体实现。首先建立了 Storm 边缘节点的调度模型，并提出了一种启发式动态规划算法；然后针对 Storm 拓扑任务的并发度受限于 JVM 栈深度的缺陷，提出了一种基于蝙蝠算法的调度策略；最后搭建 Storm 集群实验环境进行实验验证。

关键词：云边协同，节点调度，蝙蝠算法，storm，启发式算法

RESEARCH AND IMPLEMENTATION ON CLOUD EDGE-BASED NODE SCHEDULING

ABSTRACT

With the rapid development of edge computing in recent years, more and more computing tasks have been migrated from traditional cloud computing centers to edge computing nodes. The computational requirement for cloud-edge collaboration are becoming increasingly outstanding. Due to the complexity and uncertainty of heterogeneous computing resources at the edge node, cloud-edge collaborative node scheduling is the key to affecting cloud-edge collaborative computing performance.

At present, the node scheduling for cloud-edge collaboration faces the following problems: (1) in the cloud-edge collaboration environment, different strategies are required for different types of tasks (uploading cloud center or edge scheduling). The cloud-edge collaboration scheduling problem is no longer the traditional cloud center centralized batch processing scheduling model, but rather needs to comprehensively consider the needs of the two computing modes for global scheduling; (2) different from the "top-down" of traditional cloud center scheduling strategies, the cloud-edge collaboration scheduling strategy is "bottom-up" which is users interact with edge nodes and the edge nodes schedule decisions whether to upload or migrate to other edge nodes; (3) due to the limited computing resources of the edge nodes, if batch scheduling is performed, the delay of the application will increase, which will affect the user experience.

It can be seen that the traditional centralized scheduling strategy of the cloud computing center is no longer suitable for cloud-edge collaboration scenarios. For the above problems, this paper proposes a real-time scheduling model based on an improved bat algorithm for cloud-edge collaboration, and on this basis, researches Storm's edge node scheduling optimization method. Main tasks in this paper as follows:

1. For the cloud-edge collaboration node scheduling requirements, a real-time scheduling model based on an improved bat algorithm is proposed. Meanwhile, Combined with EdgeCloudSim edge computing simulation platform to build a scheduling simulation model for theoretical verification.

2. Based on the above, the optimization method and implementation of Storm edge node scheduling for cloud-edge collaboration are researched. Firstly, a Storm task offloads scheduling model for edge computing is established. And then a heuristic dynamic programming algorithm is put forward to realize real-time dynamic allocation of topological tasks among edge heterogeneous nodes. For the problem that the concurrency of topological tasks may be greater than the maximum depth of the JVM stack, a scheduling strategy based on bat algorithm is put forward. Finally, a Storm cluster experimental environment is set up for experimental verification.

KEY WORDS: cloud-edge collaboration, edge scheduling, bat algorithm, storm, heuristic algorithm

目 录

插图清单

附表清单

第一章 绪　论

1.1 研究背景和意义

目前，大数据处理已经从以云计算为中心的集中式处理时代逐渐跨入以物联网为核心的边缘计算模式的大数据处理时代[1,2]。通过将计算服务由网络中心节点移往边缘节点来处理，边缘计算可以有效地降低云计算系统中数据传输的成本[3]。然而，边缘计算的出现并不是替代云计算，两者既有区别，又互相配合，可以说边缘计算是云计算的补充和延伸。实际情况边缘设备往往资源有限，不像云中心那样拥有大量的计算资源（CPU，内存，硬盘）和强大的海量数据批处理能力。对于计算密集型任务（如 AI 任务），由于边缘设备的处理能力或数据存储内存不足，采用在边缘端处理会影响应用程序的性能和运行效率。但是, 对于延迟敏感型任务（如 VR 任务），当计算任务受延迟约束时，采用云计算的方式处理会影响任务处理的实时性。所以，由边缘端与云端形成的多层混合架构，更能综合发挥两种计算模式的优势。边缘计算与云计算必然将彼此融合，进入“云边协同”的新阶段[4]。

由于边缘异构计算资源的复杂性和不确定性[5]，计算任务的上传策略和边缘调度策略，都是影响云边协同计算性能的关键。只有实现了计算资源的合理调度与分配，才能充分发挥出云中心、边缘服务器、边缘设备之间相互协同的优势，实现资源利用率、网络延时、带宽、能耗等全方面优化[6,7]，因此，云边协同的资源调度策略是影响云边协同计算性能的关键。

传统的云计算调度策略是集中式的批处理调度方式，简单来说就是将任务批次分配给相应数量的云中心的服务器。但在云边协同的环境下，面对不同类型的任务需要不同的策略（上传云中心或边缘端调度），所以云边协同的调度问题不再是传统的云中心集中批处理的调度模式，而是要综合考虑两种计算模式的需求，进行全局调度。其次，传统云中心的调度模式是“自上而下”的，即用户与云中心交互，由云中心自上而下进行资源卸载。而云边协同的调度策略是“自下而上的”，即用户与边缘节点交互，由边缘节点调度决策（上传或调度迁移到其他边缘节点）。此外，由于边缘节点的计算资源有限，对于延迟敏感型任务如果进行批次调度会导致任务出现较大的延迟，从影响用户体验。因此，云边协同的调度策略需要更好的实时性。

综上所述，传统的云中心集中式计算的调度策略已不在适合云边协同节点调度的复杂场景。因此找到一种高性能的云边协同节点调度方法，会极大推动云边协同计算技术以及物联网的发展，将大数据计算能力和用户体验提升到新的层次。

1.2 研究现状

目前，产业界已经认识到边云协同的重要性并开展了积极的探索，如文献[8]提出了一种用于个人直播的基于云边协同的实时转码系统，该系统将空闲的资源与云结合起来，大规模地对大量视频进行转码。文献[9]提出一种基于云边缘协作框架的智能电子胃镜系统，在该系统中，边缘计算平台和云平台协同工作以实现实时病变定位和胃镜视频的细粒度疾病分类。文献[10]提出一种基于云边协同的缓存搜索系统等等。

由于云边协同环境下异构计算资源的复杂和不确定性，面向云边协同的调度策略是影响云边协同计算性能的关键[11]。其中，异构节点的资源调度目标主要是最小化端到端的数据传输延迟[12,13]以及最大化能源效率[14,15]（Energy-efficient）。在云边协同环境中，边缘端的任务卸载延迟是优化的关键目标，如文献[16]为了提高延迟性能并减少任务执行的失败概率，开发了一种用于本地计算和边缘计算的最优计算任务调度策略。文献[17]中开发了基于 Lyapunov 优化的最优动态卸载策略，并提出了一种协同任务卸载策略，以通过负载共享为多用户 MEC 系统提供低延迟性能。

云边协同计算不仅仅只考虑优化边缘端的任务卸载，需要有效地协同云计算和边缘计算的策略。Merlino, G 等人[18]提出了一个基于 OpenStack 的中间件平台，通过该平台，可以发现，组合和配置边缘，雾和云级别的资源容器，并将其提供给最终用户和应用程序，从而促进和协调卸载过程。所提出的体系结构可以细粒度地组合不同的卸载模式，从而提供新的工作负载工程模式。

Hao, Y 等人[19]研究了云边协同环境中智能应用程序（例如虚拟现实和情感检测）如何实现个性化和细粒度的任务卸载问题。并提出了一种称为智能任务卸载（iTaskOffloading）的方案。其具体思想是将智能应用程序任务划分为多个不同的子任务，每个子任务都可以在不同的边缘位置进行计算。最后基于情感检测的智能应用，构建了一个真实的测试平台来验证其 iTaskOffloading 方案的延迟比传统的云计算模式要短。但此研究主要针对的是智能应用程序，没有更好的考虑对于不同类型任务的切换问题。

Ahn, S 等人[20]研究了云边协同环境下的计算密集型视频分析应用的调度问题，提出了一种用于物联网（IoT）视频分析的新型边缘云边交互的框架。并针对视频

分析的位置切换延迟、边缘与云之间的数据传输延迟构建了分析模型。将从边缘到云的部分计算分流决策问题表述为两阶段混合整数问题，提出了两阶段混合整数问题的公式及其启发式贪婪算法，以实现边缘服务器上的工作负载平衡和任务卸载调度的最佳决策。但是，他们提出的模型只考虑了单个边缘服务器和云中心的协同调度的场景，而实际场景应该是分布在不同地理位置的多边缘服务器。

Wu, G 等人[21]研究了云边协同环境下的分布式数据存储的调度问题，提出了移动边缘云的协同存储架构。针对大多数现有的存储调度算法都无法很好地解决云边协同的调度问题，提出了一种称为 ACMES 协作存储调度算法，该算法考虑并整合了移动边缘云中节点的异构信息，能够在不降低功耗的情况下最大程度地降低功耗和节点撤出风险节点存储的可靠性，同时直接在边缘环境中做出存储调度决策，并以分布式和并行的方式工作。但其调度模型主要针对的是对批量任务的自适应分配。且主要其主要针对的是边缘端的自适应调度，而缺少与云中心的交互。

Shao, Y 等人[22]研究了云边协同环境下的数据放置问题，针对如何在截止日期约束下放置大量数据副本以降低访问成本的问题。提出了一种新颖的数据副本放置策略，用于在云边协同的环境中协调处理数据密集型 IoT 工作流。他们将数据副本的放置建模为 0-1 整数编程问题，然后提出一种基于 ITÖ（DRIW_ITÖ）的数据副本放置算法，以最大程度地降低总数据访问成本，同时满足云边协同环境中 IoT 工作流的数据可靠性和用户协作需求。

从以上研究可见，现有的云中心调度策略已不再适合云边协同的复杂场景。因此需要找到一种高效综合的云边协同调度策略。首先需，要针对任务的类型进行任务个性化调度，其次针对数据量的不断增加，需要考虑到边缘异构计算资源信息的实时变化。

1.3 面向云边协同的节点调度研究分析

云计算与边缘计算的融合发展已成为新趋势，使得面向云边协同的调度问题研究成为了热点。但是该研究依然存在着很多挑战，一方面是云和边缘的协同调度问题现有的调度策略已难以满足。另一方面是云计算平台的调度机制已不满足对云边协同计算的支持。因此，这些难点都是目前的研究亟待解决的问题。

1.3.1 面向云边协同节点调度的难点和解决思路

传统的云计算调度策略是集中式的批处理调度方式，简单来说就是将一批任务分配给相应数量的云中心的服务器，分配的过程需要考虑的因素各有差异，主

要是任务执行的时间和节点的负载。常见的策略是利用轮询机制分配或智能调度算法（如粒子群）计算最优的分配方案，再按调度方案进行分配。但在云边协同的环境下，面对不同类型的任务有不同的策略（上传云中心或边缘卸载）。由于边缘计算算力下沉的特色，每个边缘节点都位于不同的地理位置，其中每个边缘节点中又包含一定数量的边缘设备（智能机器人，服务器等各种终端）。其次，传统云中心的调度模式是“自上而下”的，即用户与云中心交互，由云中心自上而下进行资源卸载。而云边协同的调度策略是“自下而上的”，即用户与边缘节点交互，由边缘节点调度决策（上传或调度迁移到其他边缘节点）。最后，由于边缘节点的计算资源有限，所以如果进行批次调度会导致应用程序的延迟增加，从影响用户体验。因此云边协同的调度策略需要更好的实时性同时，由于边缘端又有较高的实时性需求。所以，调度问题不再是简单的集中式处理，而是要全局考虑所有边缘节点以及边缘设备的实时负载和配置信息，进行全局的实时调度。

针对上述问题，本文考虑从以下几个方面解决：一方面，针对任务的类型，根据其自带的信息进行实时判断，决定上传或卸载策略。另一方面，针对上文提到的“自下而上”的实时调度需求，本文的调度模型的边缘节点之间（与云中心）都能进行通信调度，用户可直接与边缘节点进行交互。由于蝙蝠算法[23]相比遗传算法等智能搜索算法，具有精度高、收敛快等优点，获得了在云边环境下资源调度方面的广泛运用[24]。因此本文考虑使用蝙蝠算法对每个任务进行实时全局计算，根据全局节点的实时负载信息动态计算出一个具体边缘节点的具体边缘设备的全局分配方案。

但传统蝙蝠算法在随机游走产生新解的过程中依赖的步长扩张系数是固定的，参数的取值很大程度会影响算法在全局搜索上的能力，收敛速度上也存在缺陷。因此本文对其进行改进优化，在算法全局搜索产生随机解的过程中引入可变步长的策略[25]来防止划分算子陷入早熟，并添加一个自然扰动因子震荡新解来提高全局收敛能力。

另外，考虑到仿真环境的问题，目前研究者常用的 CloudSim 仿真平台依然是云计算集中式批处理的调度方式，已并不适合本文的云边协同场景。而最近开源项目，基于 CloudSim 计算内核的 EdgeCloudSim[26-27]，则在 CloudSim 的基础上搭建了支持模拟云边协同场景的仿真框架，它对其中每个边缘节点都附加了位置信息，并通过网络模块模拟与云中心的通信，使其更接近真实的云边协同环境。因此，本文在 EdgeCloudSim 搭建了本文的调度模型并进行试验的验证。

1.3.2 基于 Storm 边缘节点调度优化的解决思路

在云边协同的环境中，尤其是边缘节点的高实行性要求，对于以 Storm 为代

表的流式处理平台的实时处理能力有着很大的挑战。而 Storm 传统的云中心模式调度机制已不适合从云计算模式向边缘端延伸。

Storm 的底层调度机制，简单来说就是根据自定义的拓扑信息，将用户自定义的一定数量的线程实例分配到 Storm 架构中执行具体线程实例的容器 Slot 中。而 Storm 默认采用的 Even-Scheduler 调度机制采用轮询方式来分配任务，特点是简单高效。但其机制过于简单，没有考虑集群的实时状态信息和节点的配置信息。会导致在边缘环境下出现边缘节点间的通信代价高、负载不均以及边缘节点资源利用率低等问题，严重影响边缘端的计算性能。此外， Storm 改进版本新增的 Resource-Aware-Scheduler 调度机制通过对节点资源感知来优化通信代价和资源利用率，但其复杂的资源感知评价体系和参数配置增加了额外的调度复杂度，对节点负载的均衡优化的效果并不明显。因此，拓扑的分配部署方式是决定调度性能的关键，而拓扑的分配涉及平台底层调度模块中的相关定义，也是本文研究的难点。

本文针对上述问题，在 Storm 调度模块的源码底层改变了拓扑的分配方式，以及线程实例和容器 Slot 的映射关系。抽象出了一个集合形式的全局调度方案并建立了启发式调度模型，根据集群的实时状态信息和节点的配置信息，选择最优的调度方案。但是，此思路需要计算出所有集合形式的全局调度方案，然后在其中选择一个最优解。此计算过程算法的复杂度较高，如果拓扑配置的线程数超过了边缘服务器中 JVM 虚拟机的栈深度，会导致服务器崩溃。

所以，本文提出了基于蝙蝠算法的调度策略，即不用计算出所有的调度方案，只需根据抽象出的集合，初始化一个集合种群，并通过蝙蝠算法不断迭代最终计算出最优的调度解。此思路极大的降低了算法复杂度，对实际场景也能很好的适用。

1.4 本文的主要工作

针对上述提到的国内外研究现状存在的问题和解决方案，本文将研究任务主要分为两个阶段：面向云边协同的节点调度理论研究和面向云边协同的 Storm 边缘节点调度优化研究。

1.4.1 面向云边协同的节点调度

本文针对云边协同环境下的任务上传或边缘卸载的节点的实时调度问题，提出和建立了云边协同环境下基于改进蝙蝠算法的实时调度模型。

其中，由于传统 BA 算法在随机游走产生新解的过程依赖的步长扩张系数是固

定的，参数的取值很大程度会影响算法在全局搜索上的能力，收敛速度上也存在缺陷。因此本文在 BA 算法全局搜索产生随机解的过程中引入可变步长的策略[3]来防止划分算子陷入早熟，并模拟自然环境对蝙蝠回声定位产生的影响，添加一个自然扰动因子震荡新解来提高全局收敛能力。提出了带有扰动因子和具有可变步长的蝙蝠算法（VSSBA）。

在改进蝙蝠算法的实时调度模型中，用户可以通过移动设备提交计算任务到距离最近的边缘节点，边缘节点根据不同用户同时提交的数据流实时地决定是否上传至云中心，或通过改进的蝙蝠算法，根据全局节点的负载信息动态计算出一个具体边缘节点的具体边缘设备的全局分配方案。最后将计算任务实时地分配到相应的边缘节点所包含的边缘设备中。本文提出的调度模型通过云边协作的方式，针对不同任务类型采用最优的分配方案，能极大提高计算系统的性能。最后，本文在支持模拟边缘计算场景的 EdgeCloudSim 仿真平台构建了云边协同实时调度模型，并对基于改进的蝙蝠算法的调度模型的性能进行对比分析。

1.4.2 面向云边协同的 Storm 边缘节点调度优化

边缘异构节点间的调度时耗长、通信时延高以及负载不均是影响边缘计算性能的核心问题，传统的云计算平台难以满足云边协同环境中边缘端的计算需求[7]。本文研究了面向云边协同的 Storm 边缘节点的调度优化方法。首先建立了 Storm 边缘节点任务卸载调度模型，并提出了一种启发式动态规划算法。通过改变 Storm 调度框架中 Task 实例的排序分配方式以及 Task 实例和 Slot 任务槽的映射关系，然后根据边缘节点配置检测的结果来计算出最优的全局调度方案。本算法的特点是在拓扑任务设置的并发度低于 JVM 设置的最大栈深度的范围内，能准确计算出全局最优方案。缺点是如果设置的拓扑并发度高于阈值，会造成栈溢出问题。针对此问题，本文提出一种基于蝙蝠算法的调度策略，结合 Storm 的调度问题根据 Task 实例和 Slot 任务槽的映射关系初始化一个随机的解，通过蝙蝠算法的不断迭代来计算出最优解。本文算法复杂度低，运行速度快，适合任何并发情况，且该方法无需手动配置参数，能将属于同任务的线程最大化分配到相同节点，保证边缘节点的通信代价最低。

1.5 论文的结构安排

论文的结构安排如下：

第一章：阐述了本文研究的背景及其主要意义，介绍了目前相关领域的国内外研究现状。对本文研究所面临的一些重点难点进行了具体分析，并提出了相应

的解决思路。最后简要说明了本文的研究内容和创新之处。

第二章：主要介绍了本文涉及相关理论基础和一些关键技术，包括云边协同计算的相关概念、目前主流的流式处理平台的对比、Storm 的架构理论及其调度机制，最后对面向云边协同的资源调度定义和常用的调度算法做了简单的介绍。

第三章：提出了面向云边协同环境，基于改进蝙蝠算法的实时调度模型。并对本模型进行了详细的介绍与分析说明。并通过实验证明了本文模型架构相比云中心模型架构的优势。

第四章：提出了带有扰动因子和可变步长的蝙蝠算法。本章详细介绍了传统蝙蝠算法，然后详细说明了传统算法的缺陷以及本文对其的改进。最后对基于改进蝙蝠算法的调度模型的性能进行了实验论证和分析。

第五章：本章研究了面向云边协同的 Storm 边缘节点的调度优化方法，并建立了 Storm 边缘节点任务卸载调度模型。针对拓扑任务在 Storm 边缘异构节点间的实时动态分配问题，提出了一种启发式动态规划算法。同时针对拓扑任务的并发度受限于 JVM 栈深度的缺陷，提出了一种基于蝙蝠算法的调度策略。

第六章：是 Storm 边缘节点调度优化方法的实验部分。本章详细描述了实验环境、对比指标，以及具体的实验结果对比和分析。

第七章：对本文所涉及到的要点进行了总结，并提出了本文的一些不足之处。根据目前技术发展的动态，展望后续的研究方向。

第二章 相关研究与关键技术

2.1 引言

本章介绍了本文研究涉及的理论基础以及相关技术。理论基础方面，包括边缘计算概念和优势以及云边协同计算的概念和优势。此外，还介绍了面向云边协同资源调度的定义以及常用的调度算法。在本文面向云边协同的 Storm 边缘节点调度优化方法的研究方面，为了更好地理解本文提出的优化方法，本章首先对目前主流的流式计算平台进行对比，然后介绍了 Storm 的架构、数据模型及其调度机制。

2.2 边缘计算与云边协同计算的相关介绍

边缘计算是最近几年才提出的概念，目前暂无统一的定义。文献[28]中给边缘的计算的定义是一种分散式运算的架构，将计算任务由网络中心节点移往网络逻辑上的边缘节点来处理。边缘计算将原本完全由云中心节点处理大型计算服务加以分解成一定数量的字服务，并分散到更接近于用户终端设备的边缘节点去处理，可以极大提高数据的处理与传送速度，并减少延迟。简而言之，边缘计算是在靠近数据源的地方提供计算服务，任务在边缘端发起，可以产生更快的网络服务响应，能满足大数据处理在实时计算、应用智能、安全与隐私保护等方面的基本需求[29]。

随着边缘计算的兴起，在很多场景中需要计算庞大的数据并且得到即时反馈。这些场景开始暴露出云计算存在的一些问题，主要有以下几点：

1. 大数据的传输的带宽瓶颈：越来越多的边缘设备连接到互联网并生成数据，以云计算中心服务器为节点的可能会遇到带宽瓶颈；
2. 数据处理的即时性：云计算可能无法很好满足对海量数据的即时处理；
3. 隐私及能耗的问题：云计算将 VR、医疗、智能制造等设备采集的隐私数据传输到云数据中心的路径较长，容易导致数据丢失或者信息泄露等风险；
4. 数据中心的高负载导致的高能耗。

边缘计算的发展前景广阔，更是被称为 “人工智能的最后一公里”。但它还在发展初期，仍有许多问题需要解决，如：框架的选用，通讯设备和协议的规范，终端设备的标识，更低延迟的需求等。随着 IPv6 及 5G 技术的普及，其中的一

些问题将被解决，虽然这是一段不小的历程。相较于云计算，边缘计算有以下这些优势。

1. 更多的边缘节点来负载流量，使得数据的传输速度更快；
2. 更靠近用户的终端设备，传输更安全，数据处理更即时；
3. 更分散的边缘节点相比云计算故障所产生的影响更小，具备较强的容灾能力。

虽然边缘计算有众多优势，但边缘计算并不能替代云计算，两者既有区别，又相互协同。边缘计算的特点是可以实时或更快速的进行数据处理和分析，而云计算可根据这些即时产生的历史数据对处理结果进行批量的数据分析。二者需要通过紧密协同才能更好地满足各种需求场景，从而最大化体现云计算与边缘计算的应用价值。目前，产业界已经认识到边云协同的重要性，并开展了积极的探索[30]。如今，IT 巨头们纷纷推出了各自的边云协同产品，亚马逊的 AWS Greengrass、微软的 Azure IoT Edge、谷歌的 Edge TPU、华为的 Kube Edge 以及百度的 Open Edge 等等。在 2018 年 3 月 28 日云栖大会深圳峰会上，阿里云宣布 2018 年将战略投入到边缘计算技术领域，并推出首个 IoT 边缘计算产品 LinkEdge，将阿里云在云计算、大数据、人工智能的优势拓宽到更靠近端的边缘计算上，打造云端一体化的协同计算体系。

从实际的需求上看，只有实现边缘计算和云计算的紧密协同，才能够更好地匹配各类应用场景的需要，从而最大化边缘计算和应用的应用价值。换句话说，边缘计算凭借“边缘”的特性，可以更好地支撑云端的应用，而云计算则能够基于大数据分析，完成边缘节点无法胜任的计算任务，助力边缘计算更加满足本地化的需求[31]。

2.3 资源调度相关介绍

资源调度是指将计算资源分配给应用程序，并将该应用程序的组成部分映射到这些资源上，以满足某些服务质量（QoS）和节约资源等目标的过程[32]。应用程序本身可以抽象或具体表示成不同的形式（如进程、线程、任务、作业、工作流等）。其计算资源可能是多种多样的，可以是本地主机上的处理器到分布式资源（例如集群中的节点，云中心的虚拟机（VM）、边缘节点部署的边缘设备或移动设备等）。具体而言，资源调度可以指在特定的资源环境下，根据一定的服务质量(QoS)参数，在不同的计算资源之间进行资源调整的过程[33]。这些资源的使用者对应着不同的计算任务，每个计算任务在操作系统中对应一个或者多个进程。通常有两种途径可以实现计算任务的资源调度：在计算任务所在的计算资源中不断调整它

的工作负载，或者将计算任务迁移到其他计算资源上，以达到负载均衡的目的。

2.3.1 面向云边协同的资源调度

在云边协同的环境中的资源调度，边缘端存在资源的移动性（如用户的移动设备）。这些计算任务可能对数据传输过程和数据的逻辑移动性提出要求。此外，网络的状态可能会发生变化，需要系统进行调度和协调，以满足各种 QoS 目标（如延迟、负载、能量和金钱限制等）。因此，在复杂的协同计算系统中，资源调度涉及许多在线实时协调和优化决策，以及可能影响用户体验和系统性能。

资源调度的首要目标是确定合适的资源，以便根据合适资源的工作负载进行调度，以提高资源利用率[34]。为了更好地调度资源，需要最佳的资源工作负载的映射。资源调度还需要确定支持多个计算任务调度的足够和适当的工作负载，以便能够满足计算任务的众多 QoS 要求，例如 CPU 利用率、可用性、可靠性、安全性等[35]。因此，资源调度考虑每个不同资源的工作负载的执行时间，但最重要的是，总体性能也基于资源工作负载的类型，即具有不同的 QoS 要求。在虚拟机（资源）上分配计算任务之前，需要监测运行资源并计算每个资源的负载。如果任何虚拟机处于过度负载的情况，则计算任务不会分配给此类资源。并检查正在运行的虚拟机是否足以执行计算任务的条件。如果运行中的资源不足，则使用水平可伸缩性概念增加资源，否则将分配给工作负载并计算所需的 QoS 参数。

2.3.2 常见资源调度算法

目前常见的调度算法主要可分为静态调度和动态调度两部分[36]。静态调度算法需要事先获得有关任务的信息（如任务的长度，任务的数量和所需的计算需求）以及资源（节点的处理能力，内存等）的信息。当工作负载的变化非常小且系统的行为不经常变化时，静态算法可以很好地工作，但节点负载在云、边缘环境中会往往是动态变化的，因此静态算法不是云计算的合适选择。静态算法很容易实现，但是这些算法不能优化服务参数的质量，并且不能在实际环境中提供良好的性能。因此需要针对云环境的动态任务调度算法。静态算法的示例包括先进先出（FIFO），循环（RR），最短作业优先（SJF）等；动态调度算法中不需要事先获取有关任务和节点信息，但需要时刻监听节点。动态调度算法在云环境中更加高效和准确，因为如果任何节点处于过载状态，那么它们会将任务从过载节点转移到负载不足的节点，即算法条件随节点负载变化（增加或减少）而频繁变化。动态算法的示例包括动态轮循，异构最早完成时间（HEFT），基于聚类的异构重复复制（CBHD），加权最小连接（WLC），以及使用进化计算（EC）算法[37]，包括如遗传算法（GA）、粒子群优化（PSO）、蚁群优化（ACO）等启发式智能搜索算法。

启发式智能搜索算法的思想主要是通过初始化一定数量的随机解，作为智能搜索种群，然后通过迭代找到最优解。在每一次迭代中，种群会通过不同需求的适应度函数来跟踪个体极值和全局极值，并根据不同算法自身的迭代搜索机制来更新自己，最后可以计算出 NP 难问题的近似解。

2.4 流式计算平台与 Storm 的相关介绍

Hadoop 是研究社区和商业社区中众所周知的框架，它允许在海量数据上表达和处理分布式计算。其中 Hadoop 的 MapReduce 模型专门为云环境中的静态数据的离线批处理而设计，这使其不适用于处理云中的流数据的应用程序。为了缓解这个问题，Storm 于 2011 年问世，成为有前途的流数据处理计算平台[38]。Storm 旨在实时执行数据流的计算，主要可被应用于大数据的实时推荐、机器学习、分布式的 RPC 等场景。

2.4.1 实时流式数据处理框架对比

在实时流式计算处理框架中，目前主流的 Storm 是一个容错的、分布式的及实时的流式计算系统，此外，Spark、Flink[39]也是目前较流行的实时处理平台。与 Storm 不同的是，Spark 和 Flink 并不是完全意义上的流处理，其中 Spark 通过“Micro-batch”策略的实现微批处理，其原理是将数据流切分成一批批微型数据流，其本质还是批处理的模式，因此在数据处理上的延迟比 Storm 要高。Flink 则是新兴的混合处理框架，对批处理和流式处理都有良好的支持，但其还处于刚起步阶段，很多方面存在不成熟、使用不方便的问题。需要未来的更深入研究[40]。

Storm 作为目前主流的流式处理框架和 Spark 以及 Flink 相比，Storm 有如下特点：

1. 高性能，超低延迟：基于 Native Streaming 的方式，使 Storm 拥有比 Spark 更低的延迟，使其更适用于云边协同计算中的边缘端数据处理。
2. 可扩展性高：相比 Flink 而言，Storm 开源程度更高。并且 Storm 提供了用户可自定义的调度器接口，使用户可以更灵活的扩展自己的系统；
3. 项目成熟度高：Storm 是第一个主流的流式处理框架，目前已经成为长期的工业级的标准，并在像 Twitter，Yahoo 等大公司使用。Spark 是最近最流行的 Scala 代码实现的流式处理框架。现在已被公司（Netflix, Cisco, DataStax, Intel, IBM 等）日渐接受。而 Flink 还是新兴项目，其强大功能还需更深入的挖掘，有着更好的前景；

4. 有成熟的社区环境：根据 GitHub 数据显示，Storm 的开源社区最活跃。目前 Storm 大概有 180 个代码贡献者；Spark 有 720 多个；而 Flink 有 130 个代码贡献者。

目前Storm在互联网企业中被广泛运用。淘宝利用Storm实现商品的实时推荐、日志的实时分析等场景；Twitter 利用 Storm 实时处理海量推文数据。此外，Storm 在美团、360、腾讯、Yahoo 等国内外知名互联网企业中都有重要的应用。

2.4.2 Storm 架构及其数据处理模型

Storm 集群由一个主节点（Master node）和多个工作节点（称为 Slaves node 或者 Work node）组成。其中，主节点运行一个称为“Nimbus”的后台进程，该进程是集群的管理中心进程，主要用于任务分配、任务监控、以及故障检测。每个工作节点运行名为“Supervisor”的后台守护进程，该进程负责启动和停止属于自己管理的 Worker 进程。Worker 进程负责处理运行的具体计算任务，在一个 Worker 进程中运行着一个或多个称为 Executor 的执行器线程。在 Executor 线程中运行着一个或多个最终执行计算任务的 Task 实例。此外，Nimbus 和 Supervisor 是无状态的，其节点是快速失败的。主节点和工作节点通过 Apache Zookeeper[41]集群进行协同操作，Storm 会将节点的运行数据和日志持久化到 Zookeeper 中，因此一个节点因故障而宕机并不会影响整体系统的运行。

在 Storm 将应用程序建模成一种称为 Topology 的 DAG 图，Storm 通过 DAG 图对应用程序的数据流进行路由和分组。特别是，有多种分组策略可通过 DAG 图控制数据流的路由，包括 Field grouping，Global grouping，All grouping 和 Shuffle grouping。Topology 中包含 Spout、Bolt、Ack 等消息处理组件，其中 Spout 是数据流源（Tuple 序列），它们从不同的源（包括数据库，消息传递框架和分布式文件系统）读取数据并向 Topology 发出消息；Bolt 用于数据处理，是计算的具体逻辑的实现者。简单的 Topology 逻辑可以通过单个 Bolt 实现，复杂逻辑则通过多个 Bolt 联合实现。每个消息组件有一个或多个 Executer 线程 Task 实例，而 Task 实例个数则取决于 Topology 配置的并行度参数。Topology 由 Nimbus 提交，根据一定的调度算法分配到相应的工作节点。收到 Topology 任务后，相应的工作节点的 Supervisor 进程会启动一个或多个 Worker 进程，启动的最大进程数由这个节点配置的 Slot 决定（Slot 指的是该节点的端口）。

2.4.3 Storm 调度器及其调度机制

Storm 通过调度器及其中的调度算法为 Topology 任务分配当前集群中可用的计算资源。Storm 提供了 IScheduler 接口，用户可通过实现该接口自定义调度器，

并为不同的实际场景设计相应的调度算法。

调度器的调度流程为：（1）获得当前集群的可用空闲资源；（2）获取 Topology 配置的 Worker 进程信息和 Executor 线程信息；（3）计算可释放的资源；（4）将 Executor 线程分配到 Slot 中。

Storm 提供了多种调度器供用户选择，目前比较常用的 2 种调度器如下：

EvenScheduler：将集群中可用的 Slot 资源排序，通过轮询机制为多个 Topology 分配资源。

ResourceAwareScheduler：在 Topology 中指定各个组件所需的计算要求（包括 CPU、内存等）。当 Storm 集群中具有空闲资源时，资源感知调度器将能够以公平的方式为用户分配额外的资源。

2.5 本章小结

本章首先介绍了本文研究涉及的理论基础以及相关技术。其中包括云边协同计算的概念，面向云边协同资源调度的定义以及常用的调度算法的介绍。然后本文对目前主流的流式计算平台进行对比，并介绍了 Storm 的架构数据模型及其调度机制。最后对本文工作的展开进行了说明。

第三章 调度模型的建立和分析

3.1 引言

由于传统的云中心集中式批处理的调度模式已不再适合云边协同计算环境的复杂场景。本文针对云边协同环境下的调度问题，提出和建立了云边协同环境下基于改进蝙蝠算法的实时调度模型。用户可以通过移动设备提交计算任务到距离最近的边缘节点，边缘节点根据不同用户同时提交的数据流实时地决定是否上传至云中心，或通过改进的蝙蝠算法，根据全局节点的负载信息动态计算出一个具体边缘节点的具体边缘设备的全局分配方案。最后将计算任务实时地分配到相应的边缘节点所包含的边缘设备中。

3.2 调度模型的架构与优势

3.2.1 调度模型架构

提出的基于云边协同的实时流调度模型架构由三层组成：第一层是云调度中心，它能处理边缘节点无法承载的大型计算任务；第二层由一定数量的边缘节点组成，边缘节点具有调度功能，它们负责各边缘节点之间的调度和通信，以及和云中心的通信。其中每个边缘节点由一定数量的边缘设备（如服务器）组成，边缘设备是最终处理计算任务的物理资源；第三层是用户的移动设备。每个移动设备都附带有位置信息，边缘节点会根据移动设备的位置信息来接受距离最近的移动设备提交的任务。此外，各边缘节点之间构成边缘云，边缘云与云中心通过各边缘节点进行通信。系统详细架构图如图 3-1 所示。

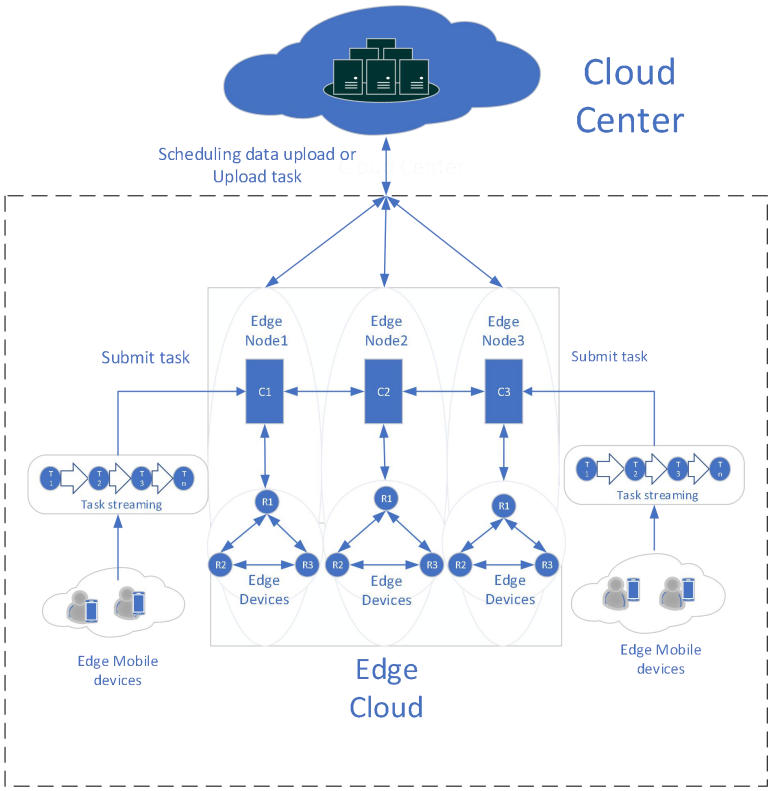


图 3- 1　基于改进蝙蝠算法的实时调度模型（三层）

Figure 3-1. Real-time scheduling model based on improved bat algorithm (three-layer)

用户可以通过移动设备提交计算任务到距离最近的边缘节点，边缘节点根据不同用户同时提交的数据流实时地决定是否上传至云中心。如果任务是复杂任务如（AI 任务），则边缘节点会实时地将复杂计算任务上传到云中心进行处理。如果是延迟敏感型任务（如 VR 任务），边缘节点会实时地获取边缘云中其他边缘节点的动态负载信息，再通过改进的蝙蝠调度算法将计算任务实时地分配到相应的边缘节点所包含的边缘设备中。此时，边缘设备则是执行计算任务的基本单位。

此外本文进行约定：本文中移动设备不作计算处理设备，仅仅作为数据上传设备。即不考虑将移动设备作为边缘设备来处理任务。

3.2.1 调度模型的优势

不同于集中式批处理的调度模型，本文提出的调度模型可以实时调度处理多个用户同时提交的计算任务。相比云计算的架构模式，本模型可以根据用户移动设备的位置信息，使得用户可以不用直接和云中心进行数据交互，而是可以将任务提交至最近的边缘节点。极大的减少计算任务从移动设备上传至云中心处理的数据延时，从而保障了对于延迟敏感型任务的良好支持。

此外，为了体现本文提出的基于云边协同架构的调度模型结构比基于云中心架构的调度模型具有更好的优势，我们在 EdgeCloudSim 中分别构建了云边协同架构的调度模型和传统云中心架构的调度模型。并对比两种架构处理任务的平均网络延迟和平均处理时间，实验结果如图二所示。

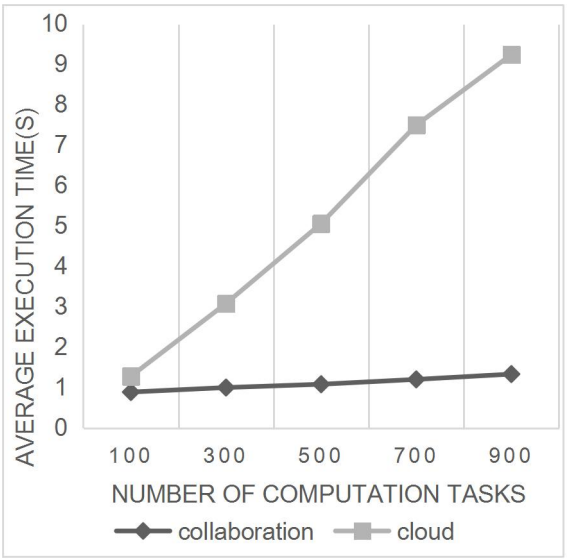


图 3- 2 任务的平均执行时间

Figure 3-2. Task average execution time

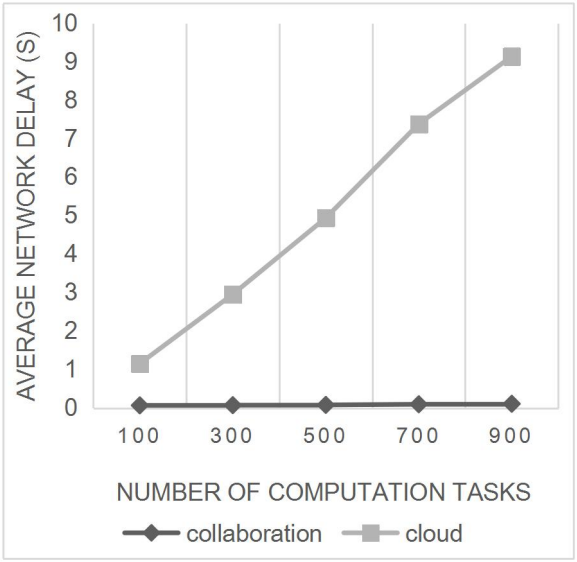


图 3- 3 任务的平均网络时延

Figure 3-3. Tasks average network delay

图 2 为两种架构处理任务的平均网络延迟和平均处理时间的实验对比。可以很明显的看出，当同时提交的计算任务不断增多，基于云中心架构的调度模型执行任务的平均时间和平均网络延迟也明显呈正比增加，而基于云边协同的调度模

型的依旧保持着高效的任务执行效率和较低的网络延迟。

此外，我们讲两种架构的调度模型的特点进行了归纳总结，如表 3-1 所示：

表 3-1　云边协同架构和云中心架构的对比

Table 3-1. Comparison of Edge Cloud-based architecture and Cloud-centric architecture

	云边协同架构	云中心架构
分布式结构	边缘端到边缘端&边缘端到云端	云端到云端
调度方法	实时	集中式
资源管理	云边分布式	云中心集中式
节点自治能力	有	无

3.3 调度模型的相关定义与分析

云边协同环境下，基于改进蝙蝠算法的调度模型考虑的是全局调度模式，即通过改进的蝙蝠算法将计算任务实时地分配到相应的边缘节点所包含的边缘设备中。其中，由改进的蝙蝠算法实时计算得到的调度方案由边缘节点的 id 和该边缘节点中的边缘设备的 id 组成，具体定义在下文会进一步解释。因此调度问题可以简化为：如何将第 i 个计算任务 T_i 分配给第 j 个边缘节点 C_j 中包含的第 k 个边缘设备 R_k，亦或是直接上传至云中心进行处理（本文中只有一个默认的云中心），使得模型的调度性能最大化。改进的蝙蝠算法会根据各边缘节点的实时负载信息，实时计算出的最优调度方案。此方案可有效提高边缘设备执行计算任务的速度，并保证节点的负载均衡。调度模型的具体定义如下：

为了具象化计算任务的执行时间，我们抽象出了任务长度的概念。用户提交的任务长度可以描述为：$L = \{L_1, L_2, L_3, \cdots, L_i\}$。其中，$L_i$ 表示用户提交的第 i 个任务的长度。其中，每个任务都包含诸如任务长度、位置信息以及对边缘设备计算能力的要求等属性。

任务首先需要根据其复杂度（任务长度）、延迟敏感度等需求来决定是否上传到云中心进行处理。如果选择上传，则需要考虑任务上传和下载数据的数据量和时延。而任务上传的时间由每个任务的数据大小决定，任务的数据量可以描述为：$S_i = \{S_1, S_2, S_3, \cdots, S_i\}$, S_i 表示用户提交的第 i 个任务的数据量。节点通信的吞吐量取决于每个边缘节点的带宽，边缘节点之间的带宽可以描述为矩阵 BwC：

$$BwC = \begin{Bmatrix} bwc_{11} & bwc_{12} & \cdots & bwc_{1m} \\ bwc_{21} & bwc_{22} & \cdots & bwc_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ bwc_{n1} & bwc_{n2} & \cdots & bwc_{nm} \end{Bmatrix} \tag{3-1}$$

其中，BwC 表示边缘节点C_n和边缘节点C_m之间的通信带宽。需要注意的是，当 $n = m$ 时，bwc_{nm}的值为 0。bwc_{nm}的取值与边缘节点之间的距离有关。则任务 T_i上传下载的数据延迟 D_i 可以表示为：

$$D_i = \frac{S_i}{bwc_{nm}} \tag{3-2}$$

本文中每个边缘节点之间的带宽由 EdgeCloudSim 的网络模块模拟。本文的调度策略首先需要根据计算出的数据延迟和每个计算任务对边缘设备计算能力的要求来决定是否上传至云中心处理。如果计算任务较轻量化或是延迟敏感型任务，则卸载到边缘节点中的工业设备中。其中，每个边缘节点中的边缘设备的计算能力可以描述为：$Mips = \{M_1, M_2, M_3, \cdots, M_k\}$，其中$M_k$表示第 k 个边缘设备的计算能力。

为了模拟真实的计算场景，每个边缘设备的计算能力应当是有限的，其计算能力会随着负载的增大而衰减。定义第 P 个调度分配方案的任务执行时间评估函数为：

$$\begin{cases} f_{P_i^{j_k}}(x) = \dfrac{L_i}{M_{j_k} - cap_i \times LB_{j_k}} + D_i \times flag_i \\ P_i^{j_k} = [j_i, k_i] \quad j \in [0,y], k \in [0,z] \end{cases} \tag{3-3}$$

其中，$P_i^{j_k}$表示第 i 个任务分配给第 j 个边缘节点的第 k 个边缘设备，即分配方案的属性。属性中，j_i表示第 j 个边缘节点的 id, k_i表示第 k 个边缘设备的 id。此外，$j \in [0,y], k \in [0,z]$表示一共有 y 个边缘节点和 z 个边缘设备。M_{j_k}表示第 j 个边缘节点的第 k 个边缘设备的计算能力。cap_i 表示第 i 个任务对边缘设备计算能力的要求，LB_{j_k}表示第 j 个边缘节点的第 k 个边缘设备的负载情况，可以理解为某时刻边缘设备中正在处理的计算任务的数量。$flag_i$表示第 i 个任务是否上传至云中心处理，如果选择上传，其值为 1，否则值为 0。

为了模拟实际计算场景，每个边缘设备需要有足够的额外内存空间来存储已处理的任务，否则任务数据将丢失。同时，计算任务的带宽不能超过边缘节点的总带宽：

$$\begin{cases} \sum_{i=1}^{x} RM_i \le RAM_{j_k} \\ \sum_{i=1}^{x} RB_{i_j} \le B_j \end{cases} \tag{3-4}$$

其中，x 是被分配到边缘设备的任务的数量。RM_i表示第 i 个任务的内存需求，RAM_{j_k}表示第 j 边缘节点的第 k 个边缘设备的可用内存。RB_{i_k}为第 i 个任务分配给第 j 个边缘节点的带宽需求，B_j是第 j 个边缘节点和云中心之间的带宽。

需要注意的是，上述所有的信息都是实时变化的，即会对每个任务流执行上述的调度过程。系统模型和问题表述中使用的相关符号和定义如表 3-2 所示：

表 3-2　符号和定义
Table 3-2. Notations

Notations	Definitions
T_i	第 i 个计算任务
C_j	第 j 个边缘节点
R_k	第 k 个边缘设备
L_i	第 i 个计算任务的长度
S_i	第 i 个任务的数据大小
BwC	边缘节点之间的带宽可以描述为矩阵
bwc_{nm}	边缘节点C_n和边缘节点C_m之间的通信带宽
D_i	任务 T_i上传下载的数据延迟
$Mips$	边缘设备的计算能力
M_k	第 k 个边缘设备的计算能力
$f_{P_i^{j_k}}(x)$	第 P 个调度分配方案的任务执行时间评价函数
$P_i^{j_k}$	第 i 个任务分配给第 j 个边缘节点的第 k 个边缘设备
M_{j_k}	第 j 个边缘节点的第 k 个边缘设备的计算能力
cap_i	第 i 个任务对边缘设备计算能力的要求
LB_{j_k}	第 j 个边缘节点的第 k 个边缘设备的负载情况
$flag_i$	第 i 个任务是否上传至云中心处理
j_i	第 j 个边缘节点的 id
k_i	第 k 个边缘设备的 id

RM_i	第 i 个任务的内存需求
RAM_{j_k}	第 j 边缘节点的第 k 个边缘设备的可用内存
RB_{i_j}	第 i 个任务分配给第 j 个边缘节点的带宽需求
B_j	第 j 个边缘节点和云中心之间的带宽

3.4 本章小结

本章详细介绍了云边协同环境下基于改进蝙蝠算法的实时调度模型的架构。并对调度的流程和相关定义做出了详细解释。本文提出的调度模型，用户可以通过移动设备提交计算任务到距离最近的边缘节点，边缘节点根据不同用户同时提交的数据流实时地决定是否上传至云中心，或通过改进的蝙蝠算法将计算任务实时地分配到相应的边缘节点所包含的边缘设备中。通过云边协作的方式，针对不同任务类型采用最优的分配方案，能极大提高计算系统的性能。最后，通过实验证明了基于云边协同架构的调度模型的优势。

第四章 基于改进蝙蝠算法的调度模型与性能分析

4.1 引言

针对传统蝙蝠算法中固定的步长缩放因子会影响局部搜索精度和收敛速度等问题，本文提出了一种带有扰动因子和可变步长的蝙蝠算法（本文称为 VSSBA 算法）。该算法能有效提升算法的全局搜索能力和局部搜索能力，以及搜索精度。在第三章所述的调度模型中，该算法能够计算出实时的最优分配方案，通过最小化边缘设备处理任务的执行时间，优化边缘节点之间的平均负载来提高整个调度模型的性能。本章节将详细描述了传统蝙蝠算法的思想和本文对其做出的相关改进，并对基于改进蝙蝠算法的实时调度模型性能进行了实验证明和分析。

4.2 基础蝙蝠算法简介

蝙蝠算法（Bat Algorithm）[42]是由 Yang 教授于 2010 年提出的高效生物启发式算法，同粒子群算法、遗传算法等类似，是一种搜索全局最优解的算法。蝙蝠算法的思想来自于蝙蝠通过回声定位来感知距离和障碍物。该算法根据距离自动调整脉冲发射的频率或波长，以此来不断迭代搜索直到计算出最优解。Chen，S 等人[43]利用马尔可夫链理论对 BA 算法进行了数学分析，证明 BA 算法可以收敛到最优解。Kongkaew，W 等人[44]对离散搜索空间中的 BA、PSO 和 GA 算法进行了比较，结果表明 BA 算法具有较好的收敛速度。该算法首先初始化一组随机解，然后通过迭代搜寻最优解，且在最优解周围通过随机飞行产生局部新解，加强了算法的局部搜索能力。与其他算法相比，蝙蝠算法特色是参数少，不需要过多依赖参数且在准确性和有效性方面远优于其他算法。

4.2.1 算法描述

简而言之，蝙蝠算法就是模拟蝙蝠回声发射与检测这样的一个机制，所有的蝙蝠都使用回声定位来感知距离，并且可以判断出是食物还是障碍物。算法具体步骤如下：

步骤 1：初始化蝙蝠的位置信息x_{ij}和速度信息v_{ij}。i 代表种群中的第 i 个解，j 代表第 i 个解中的第 j 个索引值。然后模拟蝙蝠的位置更新和速度更新，得到更新

后的位置信息，公式如下：

$$v_{ij}^{t+1} = v_{ij}^{t} + (x_{ij}^{t} - x_{*}) \times Q_i \tag{4-1}$$

$$x_{ij}^{t+1} = x_{ij}^{t} + v_{ij}^{t+1} \tag{4-2}$$

$$Q_i = Q_{min} + (Q_{max} - Q_{\min}) \times \beta \tag{4-3}$$

式中，$\beta \in [0,1]$是服从均匀分布的随机数，t 为迭代的时刻，Q_{max}和Q_{min}分别为频率的最大和最小值，x_*是全局最优解，；需要注意的是：算法第一次迭代时，x_*全局最优解是初始种群的局部最优解。即通过第三章中的适应度函数，根据初始随机种群计算出的最优解。

步骤 2: 根据公式（4-2）计算得到的更新后的位置，再采用蝙蝠的随机游走产生一个新解x_{new}，公式如下：

$$x_{new} = x_{old} + \varphi \times \varepsilon \times {A_{avg}}^{t} \tag{4-4}$$

其中，$\varepsilon \in [-1,1]$，φ是步长缩放因子，此处取值为默认的 0.001。${A_{avg}}^{t}$是 t 时刻所有蝙蝠的平均响度。

步骤 3：首先，根据第三章的时间评价函数计算得到的新的适应度评价值f_{new}；然后随机生成一个 0 到 1 之间的随机数 *rand*，若 *rand* 小于平均响度${A_{avg}}^{t}$，且f_{new}小于之前的适应度评价值f_i，则更新响度$A_{avg_i}^{t+1}$和脉冲发射率r_m^{t+1}，以及新的适应度评价值为f_i；公式如下：

$$A_{avg_i}^{t+1} = \alpha A_{avg_i}^{t} \tag{4-5}$$

$$r_i^{t+1} = r_i^{t}(1 - e^{-\gamma t}) \tag{4-6}$$

$$f_i = f_{new} \tag{4-7}$$

其中，α和γ在本文实验中均参照文献[44]取值为 0.9。

步骤 4：更新当前的全局最优解，并继续搜索迭代，直到达到最大迭代次数，循环结束迭代完成，输出全局最优解x_*。

4.3 蝙蝠算法的改进

4.3.1 可变步长

在基础蝙蝠算法中，使用公式（4-4）确定了一个解后，再使用随机游走产生一个新解，以加强算法的局部搜索能力。但是在公式（4-4）中控制步长缩放因子φ是固定的，在搜索早期阶段，如果控制参数过大，会容易跳过最优解。在搜索后期，如果控制参数过小，则很容易导致陷入局部最优。因此本文引入了可变步长策略[]，根据当前迭代次数来动态更新控制参数，其步长公式为：

$$\varphi(t) = \frac{0.4}{1 + e^{\frac{t - t_{max}}{200}}} \tag{4-8}$$

其中 t 是迭代次数，t_{max}是最大迭代次数。

4.3.2 干扰因子

为了防止算法陷入局部最优，本文使用一种自然扰动因子来对蝙蝠的飞行定位进行类似自然界的外界因素的波动干扰。对公式（4-2）进行改进，引入一个精度 来判定蝙蝠算子是否陷入早熟。同时为了避免扰动幅度过大导致偏离原来的位置，需将扰动幅度限制在 10%之内。加入自然扰动因子后的位置更新公式如下：

$$x_{ij}^{t+1} = x_{ij}^{t} + (1 - \tau) \times \omega \times v_{ij}^{t+1} \tag{4-9}$$

其中ω为 -1 到 1 之间的服从均匀分布的随机数。τ为扰动幅度。经过多次实验定值精度τ的取值定在 0.1 效果最好。

4.4 基于改进蝙蝠调度算法的云边协同节点调度描述

改进的 VSSBA 算法在蝙蝠算法的基础上加入了可变步长策略和自然干扰因子，以优化跳出局部优化和全局搜索能力。在边云协同计算环境下，利用 VSSBA 算法计算最优调度方案，即最优的任务调度方案。算法具体步骤如下：

步骤 1：初始化参数：设置种群数量，迭代次数，最大迭代次数。设置任务长度和虚拟机的计算能力值。

步骤 2：使用公式计算初始适应度。然后使用公式（4-1），（4-2）和（4-3）执行全局更新以生成初始解。最后使用公式（4-9）扰动初始解，以防止其陷入局部最优。

步骤 3：使用公式（4-4）随机行走生成新的解，并使用公式（4-8）根据迭代次数控制行走的步长。

步骤 4：如 4.21 小节步骤 3 所述，使用公式（4-5），（4-6），（4-7）更新平均响度、脉冲发射率和新的适应度评价值。

步骤 5：不断迭代当前全局最优解，直到达到最大迭代次数，循环结束迭代完成，输出全局最优解x_*。

算法伪代码如下：

The VSSBA Algorithm

Input: *gmax*: 迭代次数.
Pop: 随机初始种群
Task: 计算任务
Output: 最优调度分配方案
Initialize x_{ij} v_{ij} f_i $A_{avg_i}^t$ r_i
While (*t* < *gmax*) do
更新公式 (3-3) (4-1) (4-2) (4-3) (4-9)
if rand > r_i
通过随机飞行产生新解：更新公式(4-4) (4-8)
更新新解的 f_i 得到 f_{new}：更新公式(3-3)
if rand < A_i^t and $f_{new} < f_{min}$
接受新解并更新公式(4-5) (4-6) (4-7)
End if
计算全局最优解 x_*
End while

4.5 改进蝙蝠算法性能分析

为了验证本文提出的基于 VSSBA 算法的云边协同节点调度模型与算法的有效性，本章实验在 EdgeCloudSim 环境中实现了 VSSBA 算法，并和经典的智能调度算法（PSO,GA）以及在 EdgeCloudSim 中默认使用的最小负载算法在本文提出的云边协同环境中调度模型的性能进行比较。

4.5.1 实验环境

表 4-1 列出了本章的实验环境。在本实验中，我们利用 EdgeCloudSim 的边缘环境模拟平台，实现了云边协同架构的调度模型。

表 4-1 实验环境

Table 4-1. Running environment

System	Intel(R) Core (TM) i5-7500 CPU @ 3.20 GHz

Memory	8 GB
Operating system	Windows 10
Simulator	EdgeCloudSim
System	Intel(R) Core(TM) i5-7500 CPU @ 3.20 GHz

4.5.2 实验参数

在本节实验环境中，云中心配置了一个主机，并在主机中配置了四个大型调度服务器来处理计算密集型的处理计算任务。其中云中心 CPU 的总计算能力设置为 100000。同时，配置了 14 个边缘节点，每个边缘节点生成 8 个边缘设备（如服务器虚拟机、智能机器人等）。然后模拟 500 ~ 1500 个用户通过移动设备，同时向云边协同调度系统提交任务的场景。用户提交的计算任务会附带所需计算能力的需求属性和位置信息。实验的具体结构如表 4-2 所示。

表 4-2 实验参数

Table 4-2. Experimental parameter

Parameter	Value
云数据中心上的主机数	1
云主机上的服务器数量	4
每个边缘节点的边缘设备数量	8
边缘节点的数量	14
用户数量	100 to 1000
计算任务长度的范围	0 to 20000
计算任务所需的计算能力需求范围（%）	10-60

4.5.3 实验对比指标

实验的对比指标如下：

（1）每个边缘节点中边缘设备执行任务的平均执行时间。单位为 s，值越小，边缘调度系统的任务执行效率越高。

（2）边缘节点的平均负载率，单位为 %。如果节点之间的平均负载过高，可能意味着大部分任务都集中在某些边缘设备上，导致其计算执行性能严重降低，且造成了闲置计算资源的浪费。

在资源调度领域，常采用各种智能算法或轮询机制等算法来计算最优的调度分配方案。这些算法都有各自的工作原理，因此在调度问题中也有着不同的机制。本文在 EdgeCloudSim 环境中分别实现了比较经典的遗传算法和 EdgeCloudSim 中

默认实现的最小负载算法，并基于上述两个指标对比使用不同调度算法的调度模型的性能：

（1）EdgeCloudSim 中默认的最小负载算法（Min-Load）

（2）遗传算法（GA）

（3）默认的蝙蝠算法（BA）

（4）带有扰动因子和可变步长的蝙蝠算法（VSSBA）

4.5.4 性能对比分析

实验结果如下：

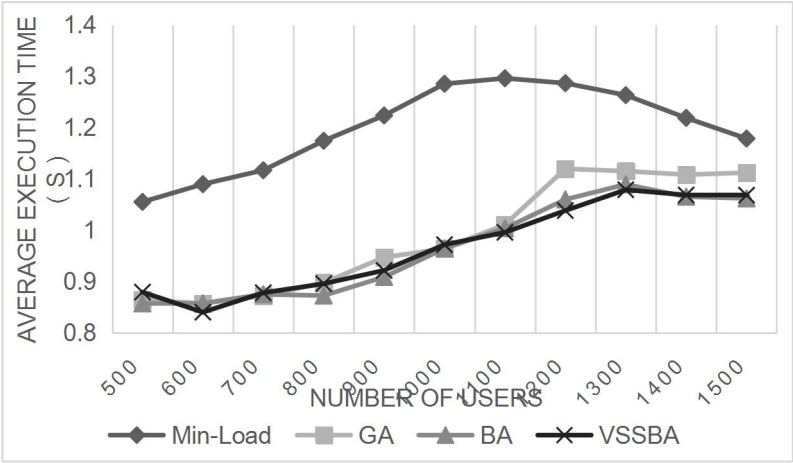


图 4- 1　边缘设备的任务平均执行时间

Figure 4-1. Average execution time of subtasks on edge devices

图 4-1 是使用不同调度算法的调度模型，它们的计算任务平均执行时间的对比。计算任务的平均执行时间能很好反映云边协同系统的计算性能，其中边缘设备的任务执行速度很大程度上取决于调度模型的性能和边缘节点间的负载情况。由对比实验结果可知，使用智能算法的调度模型其子任务的处理速度要高于使用最小负载调度算法的调度模型。其中，当云边协同计算系统中提交计算任务的用户数量小于 1000 时，使用智能调度算法的调度模型的性能之间的差异并不明显，而使用最小负载算法的调度模型的性能明显不如使用智能算法的调度模型。原因是每次分配调度时，最小负载算法需要轮询所有边缘设备节点并获取其负载情况。随着用户数量的不断增加，计算任务也不断增加，导致每次轮询所有边缘设备的成本也随之增大。相比之下智能调度算法通过随机初始解，然后通过全局和局部搜索不算迭代计算出最优解，其性能消耗相对较低。此外，扰动因子和可变步长策略加强了蝙蝠算法的全局搜索能力，能有效提高蝙蝠算法的收敛速度。当用户数量大于 1000 时，可以看出 GA 算法的性能明显不如 BA 算法和 VSSBA 算法。

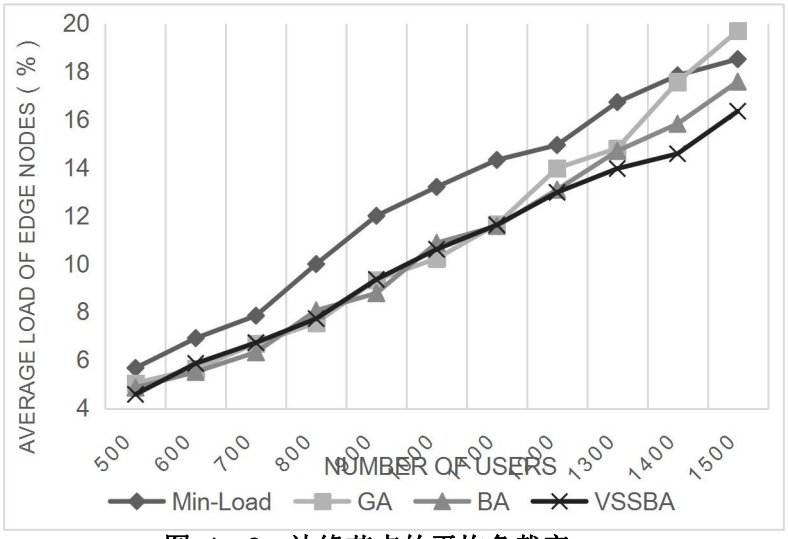


图 4- 2 边缘节点的平均负载率

Figure 4-2. Average load of edge nodes

边缘节点的平均负载情况是衡量调度算法性能最重要的指标。其中，边缘节点的平均负载率越高，则表明有部分计算任务被分配集中在某些边缘设备上，而此时有相当多的边缘设备处于空闲状态。当边缘设备被分配的任务过多时，将导致其计算性能的严重下降，且造成了空闲的计算资源的浪费并影响整个系统的计算性能。图 6 显示了边缘设备的平均负载变化，从实验结果可以看出，使用智能调度算法的调度模型的性能明显优于使用最小负载算法的调度模型。其中，当云边协同计算系统中提交计算任务的用户数小于 1200 时，智能调度算法间的性能差异不明显。当用户数大于 1200 时，使用本文提出的 VSSBA 算法的调度模型的性能明显优于其他算法。

4.5.5 实验小结

综合各种指标的性能对比，我们提出的 VSSBA 算法性能要优于其他算法。扰动因子和可变步长策略加强了蝙蝠算法的全局搜索能力，有效提高了蝙蝠算法的收敛速度。其中，在云边协同计算系统中提交计算任务的用户数不断增加的情况下，VSSBA 算法对边缘节点的负载的优化效果较其他算法有较明显的优势。

4.6 本章小结

本章是对第三章调度模型的重要补充，详细描述了蝙蝠算法和相关的改进优化，提出了一种带有扰动因子和可变步长的蝙蝠调度算法（VSSBA），并基于改进蝙蝠算法在云边协同计算系统中的节点调度性能进行了实验证明与分析。在实验部分详细介绍了在 EdgeCloudSim 环境中搭建本文的调度模型，通过对比使用不同

调度算法的调度模型，它们的边缘节点的平均负载率以及每个边缘节点中边缘设备执行任务的平均执行时间。实验证明，扰动因子和可变步长策略能有效提高蝙蝠算法的收敛速度，其改进的蝙蝠算法性能较其他算法有着明显的优势。

第五章 Storm 边缘节点调度优化方法

5.1 引言

本章节结合了第三章第四章的节点调度模型和蝙蝠算法等理论，研究了面向云边协同的 Storm 边缘节点的调度优化方法，并建立了面向边缘节点的 Storm 任务卸载调度模型。同时针对拓扑任务在 Storm 边缘异构节点间的实时动态分配问题，提出了一种启发式动态规划算法。通过改变 Storm 的 Task 实例的排序分配方式以及 Task 实例和 Slot 任务槽的映射关系实现全局的优化调度；同时针对拓扑任务的并发度受限于 JVM 栈深度的缺陷，提出了一种基于蝙蝠算法的调度策略。

5.2 Storm 调度优化研究现状

优化边缘节点调度问题和实时流式处理框架的一个重要挑战是优化拓扑任务的部署和分配[45]。最近相关研究已经以最适合异构场景的方式扩展了 Storm 的调度方式，通过设置额外的输入（如网络链接信息或节点资源利用率等）和附加的模块（如附加的系统监视器或复杂的调度程序）。例如，文献[46-51]描述了 Storm 的扩展，考虑了服务器之间的 CPU 和网络负载，以便重新平衡任务到节点的分配；此外，也有文献[52]使用 Metis 对拓扑任务进行多层的 K 划分；文献[53]提出使用 GPU 来提升 Storm 计算能力。但是上述相关研究依然存在一些问题，如文献[46]增加了额外资源感知检测模块，来优化节点通信代价和资源利用率，从而增加了调度方法的复杂度，导致调度耗时增加；文献[47]使用贪心算法来检测网络延时，容易陷入局部最优。这些方法仅仅额外附加感知模块，却没有本质改变拓扑的分配方式，因此依然存在调度时间过长、负载不均等问题。如何提高 Storm 集群边缘服务器之间的调度效率、降低通信时延，保证节点负载均衡，以及提高边缘节点的资源利用率是需要亟需解决的关键问题。

根据以上研究可以看出传统的 Storm 调度策略已无法满足云边协同环境中边缘端的高实行性需求。一种高效的拓扑调度分配方式是提高 Storm 边缘节点调度效率以及资源利用率的关键。

5.3 Storm 边缘节点的调度模型和优化方案

5.3.1 集群架构

图 5-1 中，云边协同场景下的 Storm 集群由 Storm 云中心的 Master 单元节点和若干异构边缘服务器组成。

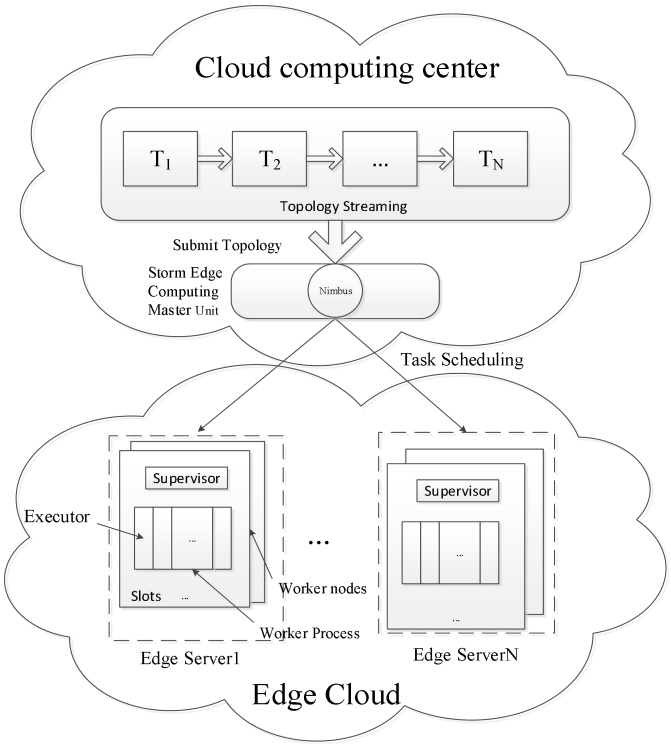


图 5- 1 Storm 集群架构

Figure 5-1. Storm's cluster architecture

其中 Storm Edge Computing Master Unit 为云中心与边缘服务器 Storm 集群节点间的通信 Master 单元节点，它负责管理云中心拓扑流任务卸载到各边缘服务器的 Storm 集群节点的通信与调度，并装载了 Storm 的核心组件 Nimbus， 其作用是收集和管理云中心卸载到边缘服务器的拓扑数据流，然后根据相应的调度算法将其分配到各边缘服务器的子节点。子节点装载了 Supervisor 组件，Supervisor 将有一个或多个 Worker 进程。Supervisor 将任务委派给 Worker 进程，Worker 进程将根据需要产生尽可能多的 Executor 线程并运行拓扑任务。此外，每个子节点运行在各个边缘服务器中，它们组成一个边缘云。

5.3.2 任务调度模型

Storm 集群中每个节点的 Supervisor 组件可以启动一个或多个 Worker 进程。所有的 Topology 都将在 Worker 进程里运行，进程启动的最大数量由这个边缘节点配置的 Slot 决定，这里 Slot 实际指 Worker Node 的端口号，默认端口号是

6700-6703。1 个单独的 Worker 进程会运行多个 Executor 线程，每个 Executor 线程只会运行 1 个 Topology 的 1 个 Component（Spout 或 Bolt）的 Task 实例。

现有的 Storm 调度机制的 Sort-slots 算法以轮询分配的方式，将拓扑任务中的 Executor 线程的 Task 实例以< node id+ port id, (start-task-id, end-task-id)>集合的形式逐一分配给可用的 Slots。这样的方式只是简单的排序再分配，在拓扑任务数量多的情况下会造成节点的负载不均，且没有考虑节点的配置信息。

本文提出的调度模型改变了 Storm 的分配方式，抽象出一种组合优化的集合形式，具体以每个 Slots 被分配到 Executor 线程中 Task 实例的数量组成一个集合，再将该集合分配给可用的 Slots，集合的形式实际表示为一个一维数组。此外，在调度模型中附加一个获取边缘节点配置信息的模块，图 5-2 给出了 Storm 任务调度模型。

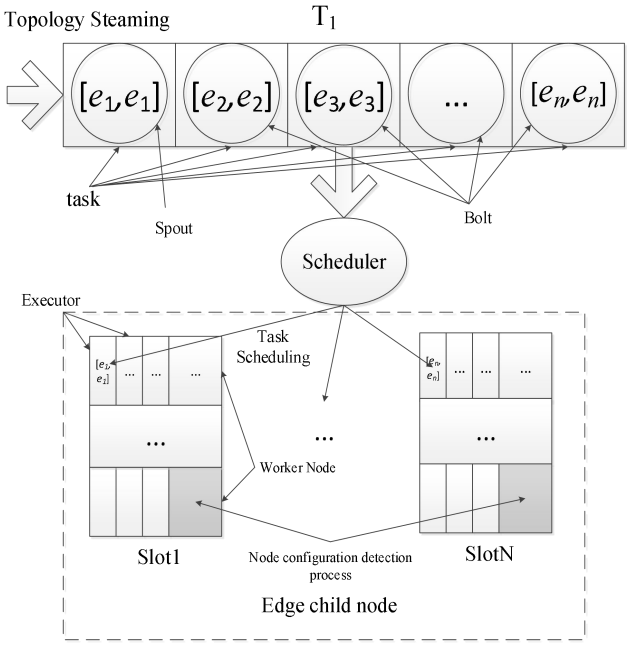


图 5- 2 Storm 边缘节点任务卸载调度模型

Figure 5-2. Storm edge node task offloading scheduling model

为了方便理解本文提出的调度模型的调度过程，本文对调度模型的结构作如下定义：

对于一个含有 n 个工作节点 Storm 集群：

$N = \{n_i | i \in [1,n]\}$，每个节点$n_i$配置有$S_i$个 Slot，那么集群可被分配的计算资源即 Slot 集合 R 可表示为：

$$R = \left\{ S_j^i =< i,j > | i \in [1,n], j \in [1,S_i] \right\} \tag{5-1}$$

S_j^i表示第n_i节点的第 j 个 Slot。集群中一共有 $Total_S(N)$ 个 Slot 资源。

对于即将提交至集群的 Topology T，一个任务的拓扑 T 由 Worker 进程中的 Executor 线程所组成，Executor 中的（Spout 或 Bolt）Task 实例是由开始和末尾的 task -id 组成的二维数组的序列，其单元格式可定义为：

$$[start-task-id,end-task-id] \tag{5-2}$$

一般情况 Task 中开始和末尾的 id 是相同的，即 $start-task-id = end-task-id$。这里统一定义为$E_i(i \in (1,N))$。

图 5-2 中 Scheduler 调度器以上述抽象出的集合的形式（集合中的每个维度为每个 Slot 被分配到 Executor 线程数量，集合的形式实际表示为一个一维数组）将拓扑数据流 T1 中的 Task 实例通过调度器 Scheduler 分配到相应的边缘节点。分配模型如图 5-3 所示。

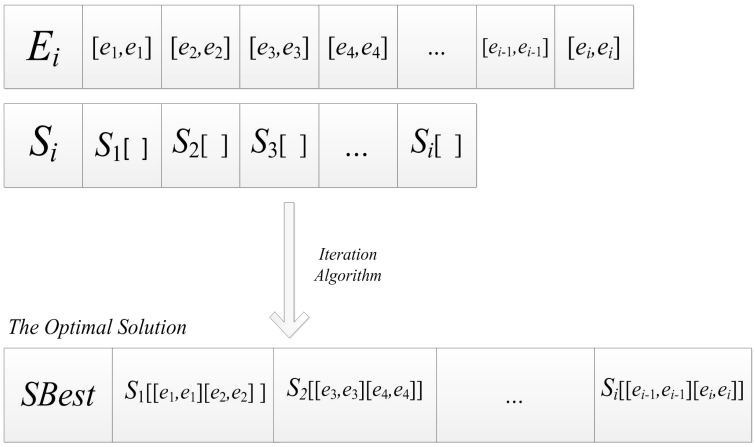


图 5- 3 Scheduler 分配模型

Figure 5-3. Scheduler assignment model

对于拓扑 T 所配置的$N_e(T)$个 Executor 线程，它们将以(start-task-id, end-task-id)的形式均匀分配到相应边缘节点所配置的 Slots 所组成的空集合中。最终 Executor 线程在每个 Slot 集合中储存形式是上述抽象的集合的形式（集合中的每个维度为每个 Slot 被分配到 Executor 线程数量，集合的形式实际表示为一个一维数组）。

简而言之，对拓扑 T 的资源调度可以表示为：

$$f(x) \rightarrow S \tag{5-3}$$

其中函数f表示 Executor 到 Slot 的映射。x 表示执行 Task 的 Executor 集合和容纳 Executor 集合，S 表示相对应的 Slot。同时一次资源调度需要满足以下要求：

（1）T 占用的 Worker 数应小于或等于集群的 Slot 数量；

（2）对于当前研究版本（即 1.06）的 Storm 来说，不允许不是同一个 Topology

的两个 Executor 线程被分配到同一个 Slot 中。

5.3.3 优化方案

综上所述，本文 Storm 的调度问题可以简化为：如何将$N_e(T)$个 Executor 线程分配到$N_S(T)$个 Slot 集合中，使得 Storm 在边缘节点的调度时间最短、资源利用率最高，并保证节点间的负载均衡。这是一个多项式求解 NP-Hard 问题。

为了优化上述调度过程，本文将$N_e(T)$个 Executor 线程分配到$N_S(T)$个 Slot 的分配方案看作群组的一个解。解的数量则由 Topology T 所配置的 Executor 线程数和 Worker 进程数决定。每个解的结构是上述抽象出的集合的形式。所有解的解集$res = \{res_1,\ldots,res_n\}$为 k 个解所组成的二维数组。Topology T 所配置的 Slot 数$N_S(T)$和 Executor 线程数$N_e(T)$视为优化的维度。通过附加配置检测模块获取的边缘节点配置信息的作为调度的输入，将一次任务调度的总执行时间和每个边缘节点的负载均衡标准差作为解优劣的评价值。最后利用启发式动态规划算法和基于蝙蝠算法的调度策略计算得到最优解即 Storm 调度的最佳分配方案。

5.4 算法设计

5.4.1 启发式动态规划算法

启发式动态规划算法首先根据边缘节点配置检测的结果（CPU 的利用率）作为适应度函数来评价解的优劣。定义：第N_{res}个分配方案即解为$res[i]$ $(i \in [1,N_{res}])$，提交至集群的 Topology T 所配置的 Executor 数视为任务的长度$L_{Ne(T)}$，获取的系统的分配给集群的 CPU 值为C_{Sys}，P_{Exe}为分配给每个 Executor 线程的 CPU 占集群 CPU 总值的百分比。则将既定数量的 Executor 线程分配到第 i 个 Slot 所需要的执行时间T_i的计算公式为：

$$T_i = \sum_{i=1}^{N_S(T)} \frac{res[i]}{\left(C_{Sys} \times \left(\frac{1}{res[i] \times P_{Exe}}\right)\right)} \tag{5-4}$$

其中，T_i越小说明该解的执行时间越短。在计算出每个解最短执行时间的同时需要考虑每个节点的负载均衡，即每个节点执行任务所用的时间。波动范围越小则负载越均衡。

定义负载均衡的标准差 LB 计算公式为:

$$LB = \sqrt{\frac{1}{N_S(T)} \sum_{i=1}^{N_S(T)} \left(T_i - T_{avg}\right)^2} \tag{5-5}$$

其中，T_{avg}为集群所有节点分配线程所需的执行时间T_i的总和的平均值。

启发式动态规划算法的目的是将$N_e(T)$个 Executor 分配到$N_s(T)$个 Slot 中，保证此方案解的总执行时间最短且负载均衡度最高。针对上小节建立的调度模型，计算所有可能的分配结果。

算法首先初始化解集 $res = \{res_1,\ldots,res_n\}$，定义当前节点的第 i 个 Slot 的索引为全局变量 Idx。为充分利用本地资源，算法设定了每个 Slot 所允许容纳的最大、最小 Executor 线程数，分别定义为：$Max\,N_e(T)$和 $Min\,N_s(T)$，其取值范围为$(1,N_s(T)+1)$。然后按照图 3 中的方式，根据式（5-4）、式（5-5）计算最优解。启发式动态规划的具体算法流程如下：

Step1：输入 Topology T 配置的 Executor 的数量$N_e(T)$、Slot 的数量$N_s(T)$、以及 $MaxN_e(T)$，$Min\,N_e(T)$；

Step2：初始化当前已经分配的 Executor 线程数为 0，即 $Currrent\,N_e(T) = 0$；

Step3：判断当前节点的 Slot 的索引 Idx ，如果当前索引值小于$N_s(T)$且当前已分配 Executor 线程数小于 $N_e(T)$，则根据全局变量索引 Idx，循环遍历将第 j 个值赋值到 $res[Idx]$中；

Step4：重复步骤 2.3，将还未分配的任务数补位到数组 $res[N_s(T)]$的位置；得到当前节点的调度方案集合；

Step5：若还存在没有处理的节点，则以没有处理的节点为新的当前节点（使用递归的形式重复上述循环遍历过程），返回 Step2，直至计算得到所有可能的调度方案集合；

Step6：对解进行评价，以得到的最优解作为 Storm 节点任务调度的最佳分配方案。

5.4.2 基于蝙蝠算法的调度策略

本策略首先通过根据图 3 中的方式随机初始化一个群组解，然后根据式（5-4）、（5-5）不断计算迭代出全局最优解。最后基于出入栈的思想将 Executor 线程以 (start-task-id, end-task-id)抽象出的集合的形式按照其每个维度的值分配到集群中。分配时该算法会将属于一个 Slot 的多个 Executor 线程分配在一起，优化其通信时延高的问题。基于蝙蝠算法的调度策略具体流程如下：

Step1：遍历 Topology T，判断拓扑是否需要调度，否则结束算法；

Step2：获取 T 从 Component *id* 到 Executor *id* 的映射的 Map 集合，将获取的 Component 的 *id* 的 Map 集合存入 Set 集合；T 配置的 Executor 的 *id* 按 Component 的 *id* 的顺序排序存入 Collection 集合；

Step3：根据 T 配置的 Worker 数来确定所需要的 Slot 数；

Step4：根据蝙蝠算法得出的解 *res* 代入，根据解的分配方案将 Executor 分配进 Worker 中，将结果存入 List 集合；

Step5：获取集群可用的 Slot 并排序存入 List 集合；如果 Slot 已满则释放；

Step6：调用 $Assign()$方法将分配好的 Slot 集合分配到集群的边缘节点中，到此算法结束完成分配。

5.4.3 算法复杂度分析

本文提出的启发式动态规划算法在生成初始解时运用了递归的方式，在拓扑任务并发度较低的情况下能精准计算全局最优的调度方案，算法的时间复杂度为 $O((Max - Min)\ M)$，其中 Max 和 Min 是每个 Slot 分配的 Executor 数量的范围，M 是 Slot 的数量，算法时间复杂度最差情况是 $n = m * max$。拓扑任务设置的并发度大小受限于 JVM 栈深度，所以只适用于并发度较低的情况。基于蝙蝠算法的调度策略是不确定算法，通过随机产生的初始解不断迭代到相对最优解，其精度不如启发式动态规划算法，算法的时间复杂度为 $O(N * M)$，其中 N 是蝙蝠的数量 M 是最大迭代次数，算法的时间复杂度的最好情况是 $O(N)$。由此可知，所以在时间复杂度上蝙蝠算法的性能要明显优于指数级的复杂度，且不受限与 JVM 的栈深度，但其缺点是可能得到全局的相对最优解。

5.5 本章小结

本章结合了前几章节的调度模型和算法理论知识，研究了 Storm 边缘节点的调度优化方法。首先建立 Storm 边缘节点调度模型，并提出了一种启发式动态规划算法，该算法能计算出满足条件的所有分配方案，并准确找到全局最优解。针对拓扑任务的并发度可能超过 JVM 设置的栈最大深度的问题，提出一种基于蝙蝠算法的调度策略，通过随机产生初始解，不断迭代计算出优化解。所提出的方法可以应用于最常见的场景，包括高拓扑并发性的情况，并且不需要手动配置参数。

第六章 Storm 边缘节点调度优化方法实验与结果分析

6.1 引言

本章是第五章 Storm 边缘节点调度优化方法的实验部分。本章节详细描述了实验环境、对比指标，以及具体的实验结果对比和分析。实验结果表明，和 Storm 调度算法相比，本文提出的 Storm 边缘节点调度优化方法与传统云中心模式的 Storm 调度模型相比，在边缘节点 CPU 利用率指标上平均提升了约 60%，在集群的吞吐量指标上平均提升了约 8.2%，因此能够满足边缘节点之间的高实时性处理要求。

6.2 实验介绍

6.2.1 实验环境

本文实验在一个安装了 ESXI6.0 系统的 Dell R710 服务器中构建虚拟化集群来模拟边缘服务器之间的交互。本文构建了 4 个不同配置的节点，其中包括 3 个工作节点。服务器的基本配置如下：CPU：Intel（R）Xeon（R）X5650；2.66GHz×6 core×2；RAM：128GB；1Gbps×4 的网卡以及 2 个 1T 硬盘组成的磁盘阵列。 虚拟节点的配置如下：

第 1 个虚拟节点的 CPU 为 2.67GHz×1 core、RAM 为 2GB；操作系统是 Ubuntu 16.04 x86_64；

第 2 个虚拟节点的 CPU 为 2.67GHz×2 core、RAM 为 4GB；操作系统是 Ubuntu 14.04 x86_64；

第 3 个虚拟节点的 CPU 为 2.67GHz×4core、RAM 为 6GB；操作系统是 Ubuntu 14.10 x86_64；

第 4 个虚拟节点的 CPU 为 2.67GHz×6core、RAM 为 8GB；操作系统是 Ubuntu 12.04 x86_64。集群配置如表 6-1 所列。

表 6-1 主机配置

Table 6-1. Storm node configuration

主机	IP 地址	功能
Storm-M	192.168.0.18	Nimbus、Zookeeper

Storm-S1	192.168.0.19	Supervisor、Zookeeper
Storm-S2	192.168.0.20	Supervisor、Zookeeper
Storm-S3	192.168.0.21	Supervisor、Zookeeper

本文 Storm 使用的版本为 1.06，利用 Pluggable Scheduler 实现本文的算法。实验中使用经典的 Word Count Topology 来测试集群的性能。为了保证时间测量数据的准确性，本文集群中的所有节点都配置了 NTP 时间同步协议来同步集群时间。

6.2.2 调度算法对比

本节对比了蝙蝠算法、PSO、GA 以及本文提出的启发式动态规划算法(IDP)。在 Storm 环境下，不同并发度(拓扑任务设置的 Worker 进程和 Executor 线程数量)时调度方案的计算时间的差异，如图 6-1 所示。

启发式动态算法的特点是并发度低时拥有最快的计算时间和计算全局最优解的能力，符合边缘计算的强实时性需求。但是由于时间复杂度较高，随着设置的并发数增加，计算时间会呈指数增长，在边缘计算环境下便不再适用。而智能算法通过随机产生初始解，根据初始解迭代更新计算出全局相对最优解。其算法的复杂度较低且受拓扑设置的并发数的影响较小。从图 6-1 中可以看出，IDP 算法在并发度较小的情况下的计算速度最快，但是并发度设置为 32 或 64 时，计算时间呈指数上升。智能算法受并发度的影响较小，且蝙蝠算法的计算时间较 PSO 和 GA 更短，更符合 Storm 在云边协同中边缘端的应用场景。

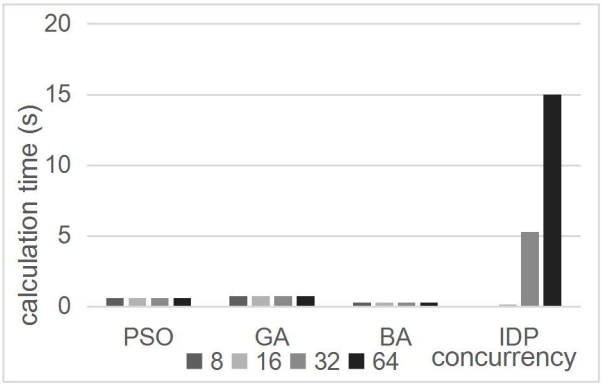


图 6- 1　不同并发度下计算时间的对比

Figure 6-1. Comparison of calculation time under different degrees of concurrency

6.2.3 调度模型评价指标

边缘环境下的高实时处理需求是 Storm 边缘节点调度优化问题的关键。集群吞吐量是影响集群实时处理性能的一个关键指标，它指的是单位时间内处理的数据量。其中，决定集群吞吐量的指标有 CPU 使用率和负载均衡等。本文使用以下指标来评估性能：

1. 单位时间 Tuple 的吞吐量（Tuple/s），即每秒处理的 Tuple 的数量，其直接反映集群的吞吐量。在 Storm 中一个数据流由无限的元组（Tuple）序列组成，这些元组会被分布式并发地创建和处理。单位时间内集群处理的 Tuple 能直接反映集群处理的数据流即吞吐量的大小。吞吐量值越大，集群处理数据的效率越高。
2. 指定时间内集群中所有边缘节点的平均 CPU 使用率。该指标的数值越大，每个节点的 CPU 资源利用率就越高，即越能充分利用节点的计算能力。
3. 指定时间内集群中所有边缘节点平均 CPU 使用量的标准差。该指标的值越小，说明每个节点的 CPU 利用率的差异越小，即节点负载越均衡。

6.3 实验结果与分析

6.3.1 实验对比对象

实验中的数据来源于 Storm 提供的 Trident RAS API 和 Storm UI REST API 以及本文算法实现的边缘节点配置检测模块获取。实验的比较对象为：

1. Storm：Storm 的默认均衡调度模型。
2. E-Storm：本文实现的面向边缘的 Storm 模型称为 E-Storm。
3. R-Storm：文献[46]提出的资源感知的实时 Storm 模型。

6.3.2 边缘节点 CPU 的使用情况

表 6-2 列出了 3 个调度模型的调度场景中，拓扑任务启动后 1min 内每 5s 记录的所有边缘节点 CPU 的使用情况的平均值。该值的单位是拓扑任务进程 CPU 使用率的百分比。为了更直观地展示这 3 种调度模型的优化效果比较，本文使用线形图来表示，如图 6-2 所示。Storm 默认的均衡调度模型采用单一的轮询方式分配 Executor 线程，而没有考虑到每个边缘节点的 CPU 负载情况，其特点是调度效率高但节点 CPU 利用率低，会造成一定的资源浪费。R-Storm 需要手动设置各个组件运行时每个实例所需要的资源数和 Topology 优先级数。对 CPU 的负载进行了详细优化，因此其资源利用率高但调度效率低，且集群的负载性能比较依赖于手动设置的参数。本文实现的 E-Storm 可根据动态获取的 CPU 信息计算一个全局负载最优的调度方案，同时优化了节点资源利用率和调度效率。如图 6-2 所示，E-Storm 的优化效果明显优于 Storm，其在某些时刻的 CPU 利用率高于 R-Storm。虽然 R-Storm 的整体 CPU 利用率略高于 E-Storm，但差距并不明显。

表 6-2 调度开始后 1min 内节点平均 CPU 使用率对比

Table 6-2. Comparison of average CPU usage of nodes within 1 minute after scheduling

Timestamp (s) / CPU average usage (%)	Storm	R-Storm	E-Storm
1	4.55	10.85	9.4
5	4.3	11.55	7.3
10	4.75	10.15	6.7
15	4.7	7.7	5.5
20	3.9	8.45	8.3
25	4.5	7.1	7
30	4.55	8.2	6.45
35	4.25	6.55	10.35
40	4.15	7.7	5.4
45	3.8	10.7	6.2
50	5.4	7.6	6.3
55	4.75	5.75	6.55
60	4.5	6.9	7.55

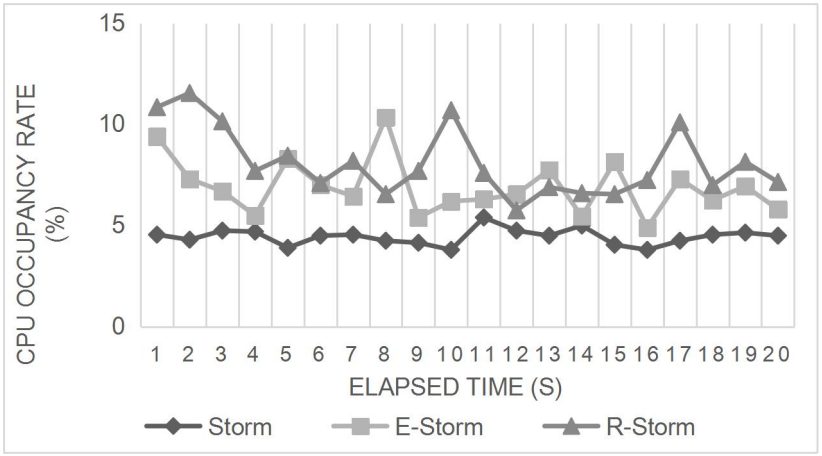


图 6- 2　节点 CPU 平均占用率

Figure 6-2. Average CPU usage between nodes

6.3.3 边缘节点负载均衡度的对比

图 6-3 给出了 3 个调度模型的调度场景中每个节点 CPU 使用情况的标准差。标准差越小，意味着负载越平衡。由于默认的 Storm 采用轮询机制，且每次轮询后会对节点剩余的 Slot 进行排序，因此能保证负载的相对均衡。R-Storm 着重优化了节点资源的利用率，但由于需要手动设置参数可能导致边缘节点的负载倾斜。E-Storm 是在资源利用率、调度效率、负载均衡中取一个相对平衡的策略。从图 6-2

中可以看出，在目前的实验环境中，Storm 的负载平衡程度最好，E-Storm 的负载平衡度明显优于 R-Storm。由此可见，R-Storm 在负载均衡方面存在不足。

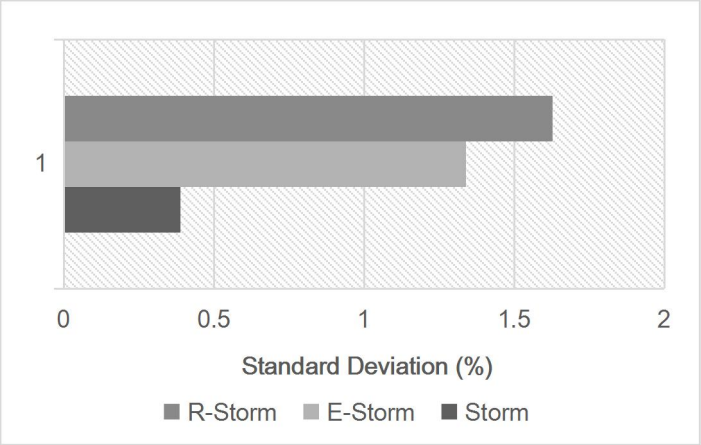


图 6- 3　节点 CPU 平均占用率的标准差

Figure 6-3. CPU average occupancy standard deviation

6.3.4 集群吞吐量的对比

集群吞吐量是上述各项指标的综合评价结果，是影响边缘环境下实时数据处理的关键因素。吞吐量越大，集群在单位时间内的数据处理能力就越强。表 6-3 列出了拓扑任务调度开始后 1 分钟内每 5 秒收集一次的每个调度模型的集群吞吐量数据，该数据的单位是每秒处理的元组数（Tuple/s）。为了更直观地显示集群吞吐量之间的差距，图 6-4 比较了调度启动后 1 分钟内每 5 秒收集一次的每个调度程序下的集群吞吐量。集群的吞吐量受节点 CPU 利用率、负载均衡度、调度处理效率等指标的综合影响，算法最好的平衡这些指标则可以使吞吐量的优化效果最理想。默认的 Storm 只考虑了调度的效率和负载均衡忽视了节点 CPU 利用率，R-Storm 着重优化提升节点的利用率，在负载均衡度和调度效率上略显不足。从图中可以看出，E-Storm 的集群 Tuple 处理量与 R-Storm 前期的集群 Tuple 处理量相差不大。而 Storm 的集群吞吐量明显低于前期的两个集群吞吐量。但在后期，直到系统最终稳定，E-Storm 的吞吐量高于其他两种模型。

表 6-3　调度开始后 1min 内各调度器的集群吞吐量对比

Table 6-3. Cluster throughput comparison of each scheduler within 1 minute after starting scheduling

Timestamp (s) / Throughput	Storm	R-Storm	E-Storm
5	220	600	420
10	1060	1420	1520
15	2200	2300	2420

20	3100	3480	3380
25	4040	4300	4540
30	5000	5380	5380
35	5980	6360	6380
40	6880	7240	7380
45	7880	8400	8240
50	8760	9260	9420
55	9700	10100	10180
60	10780	11240	12240
65	12780	12100	13120
70	13080	13800	14160

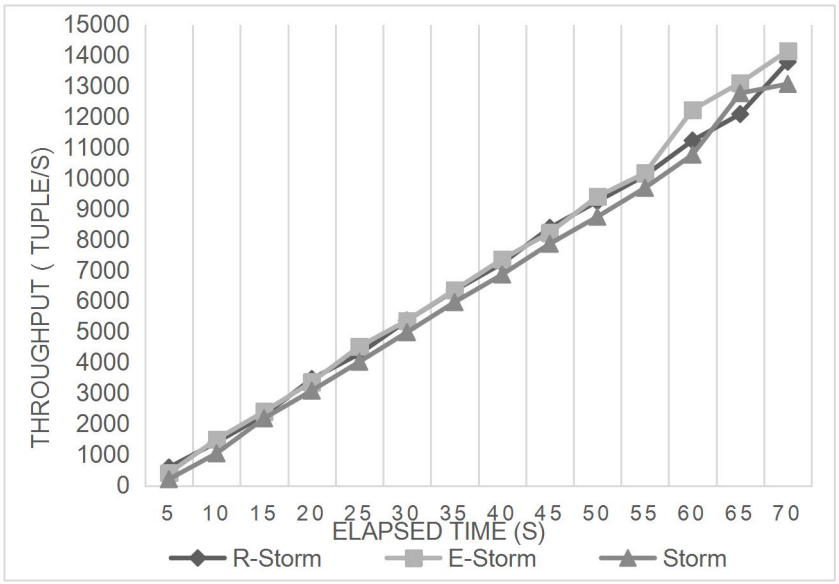


图 6- 4 单位时间的 Tuple 发送量

Figure 6-4. Tuple throughput Unit in unit time

6.4 本章小结

综合上述实验对比，本文实现的 E-Storm 在各项指标的优化上取得了较好的平衡。在节点 CPU 利用率和负载均衡指标的对比上，虽然 E-Storm 在负载均衡方面不如默认的 Storm，但是默认的 Storm 调度模型的 CPU 利用率较低，会影响集群的吞吐量；此外，R-Storm 侧重优化 CPU 利用率而在负载均衡的优化上略显不足，而 E-Storm 在负载均衡的优化上比 R-Storm 更出色。因此，本文实现的 E-Storm 模

型在资源利用和负载均衡的整体优化上优于其他两个调度模型。E-Storm 在最重要的吞吐量的优化效果最好，更符合边缘节点的调度需求。实验证明，对于边缘节点的调度优化，与当前的 Storm 调度算法相比，本文提出的算法在边缘节点 CPU 利用率指标上平均提升了约 60%，在集群的吞吐量指标上平均提升了约 8.2%，因而能够满足边缘节点之间的高实时性处理要求，可以有效提高边缘环境中的数据传输能力。

第七章 结论与展望

7.1 结　论

如今，边缘计算发展正如火如荼。在靠近数据源头的边缘端提供存储、分析和处理已是产业界的共识。全球各大主流电信运营商已经对 MEC（移动边缘计算）展开布局，以亚马逊、谷歌、华为等为代表的 Top 级公有云服务商也已经推出边缘计算解决方案。但边缘计算的出现并不是替代云计算，两者既有区别，又互相配合。二者不是替代关系，而是互补协同关系。在大多场景中，边缘计算并不是孤立存在，其实际落地离不开边云协同，云边协同将放大云计算和边缘计算的价值。

实际上，产业界已经认识到边云协同的重要性，并开展了积极的探索。例如，中国工业互联网产业联盟 AII 在其 2017 年发布的《工业互联网平台白皮书（2017）》中关于工业互联网平台功能架构图的描述，已经初步呈现了边云协同的理念；华为在其 HC2018 大会发布的智能边缘平台 IEF（Intelligent EdgeFabric）也明确提出了边缘与云协同的一体化服务概念；西门子 2018 年发布了 Industrial Edge 的概念，大致理念是通过云端部署 Industrial Edge Management 实现边缘计算与云计算的协同。

在工业场景中，通常通过将计算能力下沉到更为靠近数据源头的网络边缘侧，一方面利用边缘层运行实时分析服务，另一方面在云中心层进行 AI 模型训练，从而实现一个高性能的云边协同数据处理平台。其中，云边协同的资源调度是影响平台数据处理性能的核心问题。因此，面向云边协同资源调度的研究意义在于，目前传统的云中心批处理的调度策略已不在适合云边协同的复杂环境。更需要一种综合高效的协同调度策略。

本文的重点在于：分析了面向云边协同节点调度以及将理论问题应用于 Storm 边缘节点调度优化问题的重难点，并提供了相应的解决思路。本文的研究内容主要分成（1）面向云边协同的节点调度的理论调度模型和调度算法及其相关改进；（2）将理论问题应用于面向云边协同的 Storm 边缘节点调度优化方法两个部分。分别提出了一种面向云边协同环境的基于改进蝙蝠算法的实时调度模型，和面向云边协同的 Storm 边缘节点调度优化方法。

本文主要进行了以下几点研究：

1. 针对云边协同环境下的实时调度问题，从技术研究的重难点进行了详细分析。

这主要包括：传统的云计算调度策略是集中式的批处理调度方式，简单来说就是将一批任务分配给相应数量的云中心的服务器，分配的过程需要考虑的因素各有差异，主要是任务执行的时间和节点的负载。常见的策略是利用轮询机制分配或智能调度算法（如粒子群）计算最优的分配方案，再按调度方案进行分配。但在云边协同的环境下，面对不同类型的任务有不同的策略（上传云中心或边缘卸载）。由于边缘计算算力下沉的特色，每个边缘节点都位于不同的地理位置，其中每个边缘节点中又包含一定数量的边缘设备（智能机器人，服务器等各种终端）。同时，由于边缘端又有较高的实时性需求，所以，调度问题不再是简单的集中式处理，而是要全局考虑所有边缘节点以及边缘设备的实时负载和配置信息，进行全局的实时调度。

针对上述问题，本文考虑从以下几个方面解决：一方面，针对任务的类型，根据其自带的信息进行实时判断，决定上传或卸载策略。另一方面，针对上文提到的边缘端的特殊需求，由于蝙蝠算法相比遗传算法等智能搜索算法有精度高、收敛快等优点，在获得了在云边环境下资源调度方面的广泛运用。因此本文考虑使用蝙蝠算法对每个任务进行实时全局计算，根据全局节点的实时负载信息动态计算出一个具体边缘节点的具体边缘设备的全局分配方案。

但传统蝙蝠算法在随机游走产生新解的过程中依赖的步长扩张系数是固定的，参数的取值很大程度会影响算法在全局搜索上的能力，收敛速度上也存在缺陷。因此本文对其进行改进优化，在算法全局搜索产生随机解的过程中引入可变步长的策略来防止划分算子陷入早熟，并添加一个自然扰动因子震荡新解来提高全局收敛能力。提出了带有扰动因子和具有可变步长的蝙蝠算法（VSSBA）。

2. 针对面向云边协同的 Storm 边缘节点的调度优化问题，首先从技术研究的重难点进行了详细分析。

这主要包括：Storm 的底层调度机制，简单来说就是根据自定义的拓扑信息，将用户自定义的一定数量的线程实例分配到 Storm 架构中执行具体线程实例的容器 Slot 中。而 Storm 默认采用的 Even-Scheduler 调度机制采用轮询方式来分配任务，特点是简单高效。但其机制过于简单，没有考虑集群的实时状态信息和节点的配置信息。会导致在边缘环境下出现边缘节点间的通信代价高、负载不均以及边缘节点资源利用率低等问题，严重影响边缘端的计算性能。此外， Storm 改进版本新增的 Resource-Aware-Scheduler 调度机制通过对节点资源感知来优化通信代价和资源利用率，但其复杂的资源感知评价体系和参数配置增加了额外的调度复杂度，对节点负载的均衡优化的效果并不明显。因此，拓扑的分配部署方式是决

定调度性能的关键，而拓扑的分配涉及平台底层调度模块中的相关定义，也是本文研究的难点。

针对上述问题，本文提出了一种启发式动态规划算法，其思路是在 Storm 调度模块的源码底层改变了拓扑的分配方式，以及线程实例和容器 Slot 的映射关系。抽象出了一个集合形式的全局调度方案并建立了启发式调度模型，根据集群的实时状态信息和节点的配置信息，选择最优的调度方案。但是，此思路需要计算出所有集合形式的全局调度方案，然后在其中选择一个最优解。此计算过程算法的复杂度较高，如果拓扑配置的线程数超过了边缘服务器中 JVM 虚拟机的栈深度，会导致服务器崩溃。所以，在其基础上提出了基于蝙蝠算法的调度策略，即不用计算出所有的调度方案，只需根据抽象出的集合，初始化一个集合种群，并通过蝙蝠算法不断迭代最终计算出最优的调度解。此思路极大的降低了算法复杂度，对实际场景也能很好的适用。

7.2 展　望

随着人工智能的不断发展，和 5G 时代的来临。云边协同计算在越来越多的场景下将被应用，（如 5G 直播，VR 游戏，AI 智慧城市大脑等），并且可以为用户和应用程序提供良好的支持。然而，本文所提出的面向云边协同的实时调度模型想在实际的场景下应用，还存在很多不足。它主要存在以下几点问题：

1. 尽管本文方法已经考虑到了边缘节点的实时负载变化，并且能实现对计算任务的实时调度。但实际场景的情况非常复杂，如当大量的用户都通过同一个边缘节点提交任务，可能导致该节点的网络拥堵而导致任务提交失败；另外，当一个用户在一个边缘节点提交任务的过程中，移动到了另一个边缘节点的覆盖范围。此时也可能造成任务提交失败的情况；还有出现节点故障宕机的情况等等。本文并没有针对这些情况的调度机制。

2. 实际场景也不能仅仅只考虑节点的负载变化，其能量损耗、经济因素也是需要考虑的问题。其调度应涉及更高维度的多目标优化。

此外，对于实际边缘计算场景的 Storm 调度优化也存在一些问题：

1. 在云边协同的场景下，Storm 部署在边缘端进行实时运算，而缺少与云中心进行数据交互的架构。需要的是更好的拓展 Storm 的架构模式以更好的支持云边协同的模式。

2. 本文提出的调度优化方法只着重考虑了 CPU 利用率，而实际上 Storm 是基于内存的计算，其内存的分配、磁盘 IO 等也是影集群性能的重要因素，需要从多个维度优化调度。此外，本文提出的方法也缺少故障转移的重调度方案。

参考文献

[1] Tran TX, Hajisami A, Pandey P, Pompili D. Collaborative mobile edge computing in 5G networks: New Paradigms, Scenarios, and challenges[J]. *IEEE Communications Magazine*. 2017, 55(4): 54-61.

[2] Ning Z, Kong X, Xia F, Hou W, Wang X. Green and Sustainable Cloud of Things: Enabling Collaborative Edge Computing[J]. *IEEE Communications Magazine*. 2019, 57(1):72-8.

[3] Ferrer A, Marquès J, Jorba J. Towards the Decentralised Cloud: Survey on Approaches and Challenges for Mobile, Ad hoc, and Edge Computing[J]. *ACM Computing Surveys (CSUR)*. 2019, 51(6):1-36.

[4] Ghosh S, Mukherjee A, Ghosh SK, Buyya R. Mobi-IoST: Mobility-aware Cloud-Fog-Edge-IoT Collaborative Framework for Time-Critical Applications[J]. *IEEE Transactions on Network Science and Engineering*. 2019, 1-1.

[5] Yao H, Bai C, Xiong M, Zeng D, Fu Z. Heterogeneous cloudlet deployment and user‐cloudlet association toward cost effective fog computing[J]. *Concurrency and Computation: Practice and Experience*. 2017, 29(16).

[6] Panda SK, Jana PK. Uncertainty-Based QoS Min–Min Algorithm for Heterogeneous Multi-cloud Environment[J]. *Arabian Journal for Science and Engineering*. 2016, 41(8):3003-3025.

[7] Meng S, Li Q, Wu T, Huang W, Zhang J, Li W. A fault‐tolerant dynamic scheduling method on hierarchical mobile edge cloud computing[J]. *Computational Intelligence*. 2019, 35(3):577-598.

[8] Zhu Y, He Q, Liu J, Li B, Hu Y. When Crowd Meets Big Video Data: Cloud-Edge Collaborative Transcoding for Personal Livecast[J]. *IEEE Transactions on Network Science and Engineering*. 2018, 7(1):42-53.

[9] Ding S, Li L, Li Z, Wang H, Zhang Y. Smart electronic gastroscope system using a cloud–edge collaborative framework[J]. *Future Generation Computer Systems*. 2019, 100:395-407.

[10] Tang J, Zhou Z, Xue X, Wang G. Using Collaborative Edge-Cloud Cache for Search in Internet of Things[J]. *IEEE Internet of Things Journal*. 2020, 7(2):922-36.

[11] Ren J, Yu G, He Y, Li GY. Collaborative Cloud and Edge Computing for Latency Minimization [J]. *IEEE Transactions on Vehicular Technology*. 2019, 68(5):5031-44.

[12] Yang L, Cao J, Cheng H, Ji Y. Multi-User Computation Partitioning for Latency Sensitive Mobile Cloud Applications[J]. *IEEE Transactions on Computers*. 2015, 64(8):2253-2266.

[13] Yousefpour A, Ishigaki G, Gour R, Jue JP. On Reducing IoT Service Delay via Fog Offloading [J]. *IEEE Internet of Things Journal*. 2018,5(2):998-1010.

[14] You C, Huang K, Chae H, Kim B. Energy-Efficient Resource Allocation for Mobile-Edge Computation Offloading[J]. *IEEE Transactions on Wireless Communications*. 2017, 16(3):1397-411.

[15] Wu B, Zeng J, Ge L, Su X, Tang Y. Energy-Latency Aware Offloading for Hierarchical Mobile Edge Computing[J]. *IEEE Access*, 2019, 7:121982-97.

[16] Liu J, Mao Y, Zhang J, Letaief KB. Delay-optimal computation task scheduling for mobile-edge

computing systems[C]. *IEEE International Symposium on Information Theory – Proceedings*. 2016, 2016-1451-1455.

[17] Mao Y, Zhang J, Letaief KB. Dynamic Computation Offloading for Mobile-Edge Computing With Energy Harvesting Devices[J]. *IEEE Journal on Selected Areas in Communications*. 2016, 34(12):3590-605.

[18] Merlino G, Dautov R, Distefano S, Bruneo D. Enabling Workload Engineering in Edge, Fog, and Cloud Computing through OpenStack-based Middleware[J]. *ACM Transactions on Internet Technology (TOIT)*. 2019, 19(2):1-22.

[19] Hao Y, Jiang Y, Chen T, Cao D, Chen M. iTaskOffloading: Intelligent Task Offloading for a Cloud-Edge Collaborative System[J]. *IEEE Network*. 2019, 33(5):82-8.

[20] Ahn S, Lee J, Kim TY, Choi JK. A Novel Edge-Cloud Interworking Framework in the Video Analytics of the Internet of Things[J]. *IEEE Communications Letters*. 2020, 24(1):178-82.

[21] Wu G, Chen J, Bao W, Zhu X, Xiao W, Wang J. Towards collaborative storage scheduling using alternating direction method of multipliers for mobile edge cloud[J]. *The Journal of Systems & Software*. 2017, 134:29-43.

[22] Shao Y, Li C, Tang H. A data replica placement strategy for IoT workflows in collaborative edge and cloud environments[J]. *Computer Networks*. 2019, 148:46-59.

[23] Yang X. Bat algorithm: Literature review and applications[J]. *International Journal of Bio-Inspired Computation*. 2013, 5(3):141-9.

[24] Jian C, Chen J, Ping J, Zhang M. An Improved Chaotic Bat Swarm Scheduling Learning Model on Edge Computing[J]. *IEEE Access*. 2019, 7:58602-10.

[25] Yu S, Zhu S, Ma Y, Mao D. A variable step size firefly algorithm for numerical optimization[J]. *Applied Mathematics and Computation*. 2015, 263:214-20.

[26] Sonmez C, Ozgovde A, Ersoy C. EdgeCloudSim: An environment for performance evaluation of edge computing systems[J]. *Transactions on Emerging Telecommunications Technologies*, 2018, 29(11).

[27] C. Sonmez, A. Ozgovde and C. Ersoy, Performance evaluation of single-tier and two-tier cloudlet assisted applications[C]. *2017 IEEE International Conference on Communications Workshops (ICC Workshops), Paris*. 2017,302-307.

[28] Lopez PG, Montresor A, Epema D, Datta A, Higashino T, Iamnitchi A, et al. Edge-centric computing: Vision and challenges[J]. *Computer Communication Review*, 2015, 45(5):37-42.

[29] Top 10 Strategic Technology Trends for 2018: Cloud to the Edge (2018) [M]. *Gartner*, 2018.

[30] 工业互联网平台白皮书 (2017 年) [M]. *工业互联网产业联盟*, 2017.

[31] 云计算与边缘计算协同九大应用场景（2019 年）[M]. *中国信息通信研究院*, 2019.

[32] Varshney P, Simmhan Y. Characterizing application scheduling on edge, fog, and cloud computing resources. Software[J]. *Practice and Experience*. 2019, 50(5):558-95.

[33] Madni SHH, Madni SHH, Latiff MSA, Latiff MSA, Coulibaly Y, Coulibaly Y, et al. Recent advancements in resource allocation techniques for cloud computing environment: a systematic review[J]. *Cluster Computing*. 2017, 20(3):2489-533.

[34] Singh S, Chana I. A Survey on Resource Scheduling in Cloud Computing: Issues and Challenges[J]. *Journal of Grid Computing*. 2016, 14(2):217-64.

[35] Singh S, Chana I. QoS-Aware Autonomic Resource Management in Cloud Computing: A Systematic Review[J]. *ACM Computing Surveys (CSUR)*. 2016, 48(3):1-46.

[36] Kumar M, Sharma SC, Goel A, Singh SP. A comprehensive survey for scheduling techniques in

cloud computing[J]. *Journal of Network and Computer Applications*, 2019, 143:1-33.

[37] Zhan Z, Liu X, Gong Y, Zhang J, Chung HS, Li Y. Cloud Computing Resource Scheduling and a Survey of Its Evolutionary Approaches[J]. *ACM Computing Surveys (CSUR)*. 2015, 47(4):1-33.

[38] Soualhia M, Khomh F, Tahar S. Task Scheduling in Big Data Platforms: A Systematic Literature Review[J]. *The Journal of Systems & Software*. 2017, 134:170-189.

[39] Apache Software Foundation. Apache Flink Project [OL]. http://flink.apache.org/. March 2020.

[40] Espinosa CV, Martin-Martin E, Riesco A, Rodriguez-Hortala J. FlinkCheck: Property-Based Testing for Apache Flink[J]. *IEEE Access*. 2019, 7:150369-82.

[41] Junqueira F, Reed B. ZooKeeper: Distributed Process Coordination[M]. *O'Reilly Media*, Inc. 2013.

[42] Kar, A. K. Bio inspired computing – A review of algorithms and scope of applications[J]. *Expert Systems with Applications*. 2016, 59:20-32.

[43] Chen S, Peng G, He X, Yang X. Global convergence analysis of the bat algorithm using a markovian framework and dynamical system theory[J]. *Expert Systems with Applications*. 2018, 114:173-82.

[44] Kongkaew W. Bat algorithm in discrete optimization: A review of recent applications[J]. *Songklanakarin Journal of Science and Technology (SJST)*. 2017, 39(5):641-50.

[45] Cheng B. Edge-Computing-Aware Deployment of Stream Processing Tasks Based on Topology-External Information: Model, Algorithms, and a Storm-Based Prototype[C]. *IEEE International Congress on Big Data. IEEE*, 2016.

[46] Peng B, Hosseini M, Hong Z, et al. R-Storm: Resource-Aware Scheduling in Storm[C] *Middleware Conference. ACM*, 2015.

[47] Aniello L, Baldoni R, Querzoni L. Adaptive Online Scheduling in Storm [C] *Proceedings of the 7th ACM international conference on Distributed event-based systems. ACM*, 2013, 207-218.

[48] Cardellini V, Grassi V, Presti F. et al. Distributed QoS-aware Scheduling in Storm [C] *ACM International Conference on Distributed Event-Based Systems. ACM*: 2015, 344-267.

[49] 简琤峰,卢涛,张美玉.Storm 启发式均衡图划分调度优化方法[J].*小型微型计算机系统*, 2018, 39(11):2538-2544.

[50] Eskandari L, Mair J, Huang Z, Eyers D. T3-Scheduler: A topology and Traffic aware two-level Scheduler for stream processing systems in a heterogeneous cluster[J]. *Future Generation Computer Systems*. 2018, 89:617-32.

[51] Li C, Zhang J, Luo Y. Real-time scheduling based on optimized topology and communication traffic in distributed real-time computation platform of storm[J]. *Journal of Network and Computer Applications*. 2017, 87:100-15.

[52] Eskandari L, Huang Z, Eyers D. P-Scheduler: adaptive hierarchical scheduling in apache storm[C] *Proceedings of the Australasian Computer Science Week Multiconference*. ACM, 2016.

[53] Chen Z, Xu J, Tang J, Kwiat K, Kamhoua C. G-Storm: GPU-enabled high-throughput online data processing in Storm[C]. *2015 IEEE International Conference on Big Data. IEEE*, 2015, 307–312.

致　谢

岁月荏苒，时光飞逝。转眼间，即将结束三年的研究生求学生涯。回首过去三年的研究生时光，不禁感慨万千。三年的硕士学习生涯，在老师的指导和同学的帮助下，我的专业技能和项目实践能力都得到了提升，而且通过阅读文献、验证实验和撰写论文，我的理论知识得到了充实，科研能力也在逐步提高。这三年间，无论是艰难与挫折，还是成功与喜悦，都让我得到了历练，得到了成长。在毕业论文完成之际，我想对我帮助过我的导师、同学和家人表示衷心的感谢。

感谢我的导师张美玉教授和简琤峰副教授。感谢他们对我的生活和学习的关心，感谢他们传授我为人处世的经验，感谢他们在科研过程中对我犯的错误的包容和鼓励。师者传道授业解惑也，学术上，他们时刻把握我研究的大方向，总在我迷惘的时候为我解惑，时常牺牲休息的时间了解我研究中的问题并和我深入讨论，给予我有益的指导、建议和帮助，让我能顺利完成学业，感谢他们无私的师道精神。

感谢国家自然基金对研究的支持，感谢秦绪佳老师、马建平老师、汤颖老师、陈佳舟老师，感谢你们给予我学习上与生活上的帮助。感谢云计算小组的师姐师弟们，感谢他们在项目中的帮助与支持。感谢我的室友们在生活给予我的帮助。还要感谢我的女朋友在我研究生生涯中遭遇挫折的时候给予的安慰和鼓励。

感谢我的父母和家人，感谢他们的无私的付出和陪伴。在我遇到困难时会鼓励我坚持下去，正是有了他们的支持，我才能更加有动力继续前进，度过每一个困难，最终完成论文的编写。在此，祈愿我的家人健康，幸福。

最后，再一次由衷地感谢曾经基于我帮助和关心的人们，祝大家生活顺利，平安幸福！

作者简介

1 作者简历

××××年××月出生于××××。

××××年××月——××××年××月，××大学××院（系）××专业学习，获得××学硕士学位。

2 攻读硕士学位期间发表的学术论文

[1] 面向边缘计算的 Storm 边缘节点调度优化方法. 计算机科学（正刊），已录用.（第二导师为第一作者，本人为第二作者）

[2] An Improved Chaotic Bat Swarm Scheduling Learning Model on Edge Computing, IEEE Access, 2019, 7:58602-10. （SCI 源期刊，IF = 4.098）（第二导师为第一作者，本人为第三作者）

3 参与的科研项目及获奖情况

[1] 面向设计意图在线交换的语义云粒元重组方法研究, 国家自然科学基金. 编号: 61672461.

[2] 图像与视频不变性局部结构特征描述及应用研究, 国家自然科学基金. 编号: 61672461.

4 发明专利

学位论文数据集

密　级*	中图分类号*	UDC*	论文资助
公开	TP391	004	
学位授予单位名称	学位授予单位代码	学位类型*	学位级别*
浙江工业大学	10037	工程硕士	硕士
论文题名*			
关键词*			论文语种*
并列题名*	******		中文
作者姓名*	平靖	学　号*	2111712373
培养单位名称*	培养单位代码*	培养单位地址	邮政编码
浙江工业大学计算机科学与技术学院	10037	杭州市潮王路 18 号	310032
学科专业*	研究方向*	学　制*	学位授予年*
计算机技术	云计算、边缘计算	2.5	2020
论文提交日期*			
导师姓名*		职　称*	教授
评阅人	答辩委员会主席*	答辩委员会成员	
电子版论文提交格式：文本（　）图像（　）视频（　）音频（　）多媒体（　）其他（　）			
电子版论文出版（发布）者	电子版论文出版（发布）地	版权声明	
论文总页数*	**		
注：共 33 项，其中带*为必填数据，为 22 项。			

附件 2：学位论文书脊示例