密级：公开

编号：19214793

中山大學

工程 硕士专业学位论文

移动边缘计算场景中跨平台动态迁移机制的设计与实现

Design and Implementation of Heterogeneous-ISA Task Live Migration in Mobile Edge Computing

学位申请人：金 航

导师姓名及职称：陈志广 副教授

专业、领域名称：工程（计算机技术）

2021 年 5 月 26 日

中山大学硕士学位论文

移动边缘计算场景中跨平台动态迁移机制的设计与实现

Design and Implementation of Heterogeneous-ISA Task Live Migration in Mobile Edge Computing

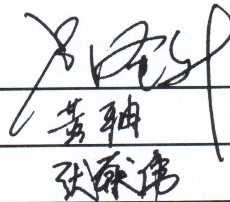
专业: 工程（计算机技术）

学位申请人: 金航

指导教师: 陈志广 副教授

论文答辩委员会主席:

成员:

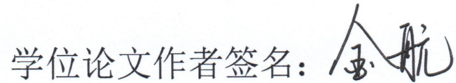


二〇二一年五月

原创性及使用授权声明

论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。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。本人完全意识到本声明的法律结果由本人承担。

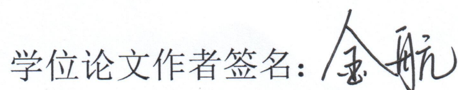
学位论文作者签名：

日期：2021 年 5 月 26 日

学位论文使用授权声明

本人完全了解中山大学有关保留、使用学位论文的规定，即：学校有权保留学位论文并向国家主管部门或其指定机构送交论文的电子版和纸质版；有权将学位论文用于非赢利目的的少量复制并允许论文进入学校图书馆、院系资料室被查阅；有权将学位论文的内容编入有关数据库进行检索；可以采用复印、缩印或其他方法保存学位论文；可以为存在馆际合作关系的兄弟高校用户提供文献传递服务和交换服务。

保密论文保密期满后，适用本声明。

学位论文作者签名：　　导师签名：

日期：2021 年 5 月 26 日　　日期：2021 年 5 月 26 日

摘　要

随着移动互联网和5G通信技术的普及，移动用户数量呈现快速增长趋势，用户对服务即时性和带宽的需求给移动应用服务带来了巨大的挑战。边缘计算作为一种新型的计算范式，能够充分利用位于网络边缘的计算和存储资源，为用户提供低延迟、高网络吞吐量的服务。移动终端设备在移动过程中其信号强度会不断变化，甚至因偶发的网络中断导致无法继续使用边缘服务器提供的计算服务。任务动态迁移允许当前执行的计算任务无缝转移到另一个设备上继续执行，减少重新执行任务所带来的额外计算开销。然而，由于边缘环境中的移动终端设备或服务器通常具有不同指令集体系架构的处理器，导致部分程序无法直接在异构平台上运行。如何在异构的设备之间实现任务的动态迁移是当前边缘计算中的挑战之一，设计一个低开销、高性能的任务动态迁移机制具有重要的研究意义。

本文对基于边缘计算的跨平台任务动态迁移进行了研究，主要工作内容如下：

（1）针对边缘计算场景中移动终端设备指令集异构的特点，设计了一种轻量级的异构平台间自适应任务动态迁移机制ALM-HIP。该机制基于Unikernel构建一个具有跨平台转换和通信功能模块的定制内核，能够在无宿主环境的情况下独立完成跨平台迁移。针对5G+移动边缘救护场景中网络环境不稳定的特点，提出了一个基于网络状态感知的自适应进程同步策略NAPS。该策略根据网络环境情况动态调节同步间隔，减少同步过程本身和因网络中断导致的重复计算开销。

（2）基于本文所设计的ALM-HIP机制和NAPS策略，实现了一个跨平台进程迁移框架。本文基于嵌入式平台开展实验，采用一些典型的测试程序对原型系统的性能进行测试与分析。实验结果表明，本文所提出的ALM-HIP机制可以有效实现Unikernel应用实例在无宿主环境下独立完成跨平台迁移，且迁移时间开销相对现有工作减少了约20%到30%，设计的NAPS策略在波动网络环境中数据同步传输网络开销和重复计算开销都比固定间隔同步策略更低。

最后，本文将所设计的ALM-HIP机制与NAPS策略进一步整合到一个面向多目标响应时间优化的卸载调度系统中，使该系统在基本卸载调度功能的基础上，具备了跨平台的任务迁移和进程同步能力，从而可以较为全面地支持移动边缘场景下异构节点之间的任务卸载与迁移。

关键词：移动边缘计算，任务动态迁移，跨平台迁移机制，进程同步

ABSTRACT

With the popularization of 5G network and the increasing quantity of mobile users, users have stronger requirements for immediate and high-bandwidth service nowadays, which causes a great challenge for mobile services. As a brand-new computing diagram, edge computing can make full use of computation and storage resources at the edge of network to provide low-latency services with high network throughput. When a mobile device is moving, its signal strength will vary frequently, in some worst situation, loss the connection with the edge server. Live migration makes it possible for the running processes to resume running on another server seamlessly to reduce the overhead caused by repetitive computation as much as possible. However, for the reason that devices at the edge are usually equipped with heterogeneous processors, it's unable to migrate the process directly. To migrate process between devices with heterogeneous processors can be an essential challenge, designing and implementation of a low-overhead and high-performance live migration system is with great value.

Heterogeneous-ISA task live migration mechanisms and related techniques are studied in this thesis. The main work contents can be described as follows.

1. A task migration mechanism named Adaptable and Lightweight task dynamic Migration between Heterogeneous-ISA Platforms (ALM-HIP) for the heterogeneous, low-performance and unstable nature of edge devices is proposed. The key of the mechanism is to implement a customized Unikernel with heterogeneous-ISA status conversion and communication ability, which makes it possible to migrate itself without the external support of host environment. Furthermore, a strategy named Network-aware Adaptable Process Synchronization (NAPS) for the 5G+ mobile medicare scenario is also proposed in this thesis. This strategy can adapt its synchronization interval to the varying network quality to reduce the overhead of synchronization and repetitive computation.

2. A prototype of the proposed mechanisms is implemented to evaluate the ALM-HIP mechanism which also employed the NAPS strategy. Based on the experiments conducted on real-world hardware with some typical benchmark applications, the performance of the prototype is evaluated and analyzed. The results show that the proposed ALM-HIP mechanism can make it possible for Unikernel applications to accomplish

heterogeneous-ISA status conversion and migration without the support of host hypervisor, with the time overhead reduced by about 20% to 30%. And the proposed NAPS strategy can perform better in fluctuating network environment to reduce the synchronization overhead and repetitive computation, compared to fixed interval synchronization strategy.

At last, the proposed mechanisms are combined into a task completion time optimization oriented offloading and scheduling system. With the combined mechanisms, the system is able to migrate and synchronize tasks between heterogeneous-ISA platforms besides the basic function of offloading and scheduling. In that way, the system is now with a versatile support for task offloading and scheduling between heterogeneous-ISA platforms.

Keywords: Mobile Edge Computing, Task Live Migration, Heterogeneous-ISA Migration, Process Synchronization

目　录

摘　要 I
ABSTRACT III
第一章　绪论 1
1.1　研究背景及研究意义 1
1.2　国内外研究现状 3
1.3　本文研究内容与创新点 6
1.4　本文章节安排 7
第二章　边缘计算中跨平台任务迁移技术对比与分析 9
2.1　面向任务动态迁移的虚拟化技术 9
2.2　跨平台任务迁移 12
2.3　任务迁移所面临的挑战 18
2.4　本章小结 19
第三章　一种异构平台间自适应和轻量级的任务动态迁移机制 21
3.1　问题描述和建模 21
3.2　异构平台间自适应和轻量级的任务动态迁移机制 25
3.3　基于网络状态感知的自适应进程同步策略 29
3.4　本章小结 32
第四章　自适应的跨平台任务迁移框架的实现与评估 33
4.1　原型设计与实现 33
4.2　实验测试与分析 33
4.3　本章小结 39
第五章　边缘计算中多节点弹性任务卸载调度原型系统 41
5.1　任务卸载调度框架与策略 41
5.2　原型实现与实验评估 44
5.3　本章小结 49
第六章　总结与展望 51
6.1　工作总结 51

6.2　研究展望 53
参考文献 55
在学期间完成的相关学术成果 59
致　谢 61

第一章 绪论

1.1 研究背景及研究意义

随着移动互联网和 5G 通信技术等的逐步发展，物联网技术也在日益成熟, 基于高速移动互联网的应用逐渐增加。中华人民共和国国家互联网信息办公室发布的第 47 次《中国互联网络发展状况统计报告》[1] 数据显示：2020 年 1 至 12 月，我国移动互联网接入流量达到了 1656 亿 GB，较上年增长了 35.7%，且在近五年内加速增长；2020 年 12 月，我国手机网民规模更是达到了 99.7% 的历史最高点；在应用方面，基础应用类、商务交易类、网络娱乐类和公共服务类等四大类型应用的用户规模和网民使用率都整体较年初有稳步增长。

上述统计数据表明，目前我国的移动互联网规模非常巨大，每年产生的数据量都在不断增加。用户在互联网中所扮演的角色从传统互联网中的消费者逐渐向生产者转变，产生于网络边缘的数据量呈爆发式增长。这一转变为移动互联网带来了巨大的挑战，如果不能适当地处理好这些产生于网络边缘的数据，将会对整个互联网骨干链路产生巨大的负面影响。同时，随着 5G 高速网络技术的普及，一些对通信带宽、网络时延非常敏感的应用的用户群基数也在增加。

为了应对互联网边缘日益增大的数据产生量和消费量，边缘计算这一崭新的计算范式进入了人们的视野。边缘计算这一概念一经提出，便迅速得到了国内外学界的广泛关注，并持续稳健发展[2]。在边缘计算这一计算范式中，位于网络边缘的计算、存储和网络资源都被充分利用起来，将计算任务部署到距离用户更近的网络位置上，为用户提供低延迟、高带宽的服务，同时还减轻了传统云计算架构中数据中心的压力。

移动边缘计算所解决的问题，主要在于如何充分利用位于网络边缘的移动设备和各类具有计算或存储能力的网络设施（如路由器、网关、无线 AP 等）或服务器的计算、存储和网络资源，满足用户对服务即时性、隐私性等的需求。在面向移动互联计算的边缘网络中，移动设备还受到网络信号、电池续航、散热能力等诸多因素的束缚[3]，这些限制条件决定了一些计算负载往往不能在移动设备上长时间高效地运行。因此，如何根据应用场景不同，对应地解决边缘网络中的端-边协同问题，是边缘计算研究中的主要挑战之一。

1.1.1 边缘计算中任务的跨平台迁移

边缘计算环境中的移动设备通常采用无线通信技术（如无线局域网、4G/5G 通信）与基站连接。随着移动设备所处位置的变化，其与边缘服务器的信号质量也在不断变化，甚至随时可能离开当前边缘网络的服务区域，导致移动设备频繁离开或加入某个边缘网络，具有不稳定性。此外，位于边缘网络中的设备通常都由不同指令集的 CPU 芯片构成[4-5]，如手提电脑、专业服务器通常采用 x86 架构指令集 CPU，智能手机、平板电脑通常采用 ARM 指令集 CPU，而一些网关等网络设置则采用 MIPS 或 ARM 指令集芯片……由于各种设备所采用的 CPU 芯片不同，芯片指令集这些设备所能够运行的程序或操作系统也受到平台指令集架构的限制，这就给边缘计算带来了异构性上的挑战。

任务迁移是边缘计算中的一个重要研究方向。由于边缘计算的应用场景对任务执行的环境有较大弹性需求，这就要求边缘计算的任务执行框架对当前网络状态进行评估，将计算任务放置于最佳的网络位置上，为用户提供低延迟、高性能的服务。但是对于一些对实时性比较敏感的应用，如 VR/AR 应用[6-8]、即时游戏[9]、自动驾驶[10-11] 等。这些应用对网络延迟和带宽的要求都比较高，且计算量较大，属于计算密集型任务，其计算负载不适合在计算性能和续航能力都相对较差的移动设备上执行。这就需要一定的机制，使这些应用在边缘服务器上为移动设备提供高性能服务的同时又在不间断地工作。如图 1-1所示，当移动设备的物理位置改变造成网络拓扑变化时，如何让这些服务应用能够“跟随”用户，无缝迁移到这一时刻最适合用户的边缘服务器上，成为边缘计算中关于任务迁移的一个重要的挑战，具有较大的研究价值和实用性。

在任务实时迁移问题上，现有的工作大致可分为两个大类：迁移决策研究和迁移机制的设计与实现。对于迁移决策研究，通常是将一个应用场景建模为一个数学问题，然后从宏观迁移算法的角度对问题进行研究；这类工作的特点是往往不涉及某个具体的平台，将具体的迁移机制作为黑盒使用。迁移机制的设计与实现较前者而言则更加基础，也更加靠近底层，通常是针对迁移机制本身的设计或优化，达到性能提升或增强迁移机制泛用性等目的；这类工作的特点是涉及到具体的任务运行环境和系统底层机制，涉及到操作系统、计算机体系架构等领域。

1.1.2 边缘计算中任务的卸载调度

在一些数据处理计算复杂度高且时延敏感的环境下，移动终端设备对边缘计算系统提出了较高的计算资源需求，而通常单一的计算节点通常因为其计算、存储和通信能力有限，难以独立完成大量的数据处理和反馈。因此，需要设计一个

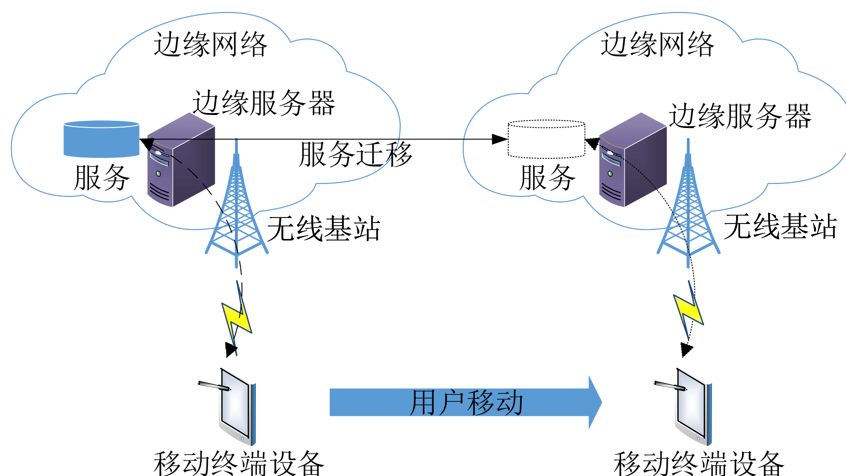


图 1-1 边缘计算场景中计算“跟随”用户

具有多节点弹性支持的分布式任务卸载调度机制，保障这些计算密集且时延敏感任务的快速响应和反馈，提升边缘计算系统的资源利用率。

对于计算卸载机制，任务调度是最关键的环节之一。根据任务调度的策略针对当前任务集合给出调度结果，将任务集分散卸载到边缘计算环境中各个相对合适的节点上执行。由于边缘计算环境中的计算、存储和网络资源都是不均等地分布在多个节点上，这也就导致了每个计算节点的计算能力、可用存储资源和通信带宽等都各不相同，使任务调度问题更加复杂。对于实际的任务调度执行，应用不同的任务调度策略会导致每个任务的响应时延各不相同，甚至可能出现节点因并不满足任务资源的需要导致任务延误或者失败等情况。

综上所述，在当前移动互联网数据激增的大背景下，移动边缘计算还将继续发展。由于边缘计算和传统云计算模式相比更加“靠近”用户，具有后者不可替代的低延迟、高带宽等特点。如何让边缘计算环境下的任务执行更具有弹性和可扩展性，如何高效地实现任务的动态迁移问题具有重要的研究意义。更进一步地，本文将对边缘计算场景下的跨平台动态迁移问题进行更深入的讨论和研究。

1.2 国内外研究现状

关于移动边缘计算中任务动态迁移的问题，学界已经开展了许多研究。总体来说，大致可以分为移动边缘计算场景下的任务动态迁移决策研究和任务迁移机制的设计与实现等两大类别。

1.2.1　移动边缘计算场景下的任务动态迁移决策研究

关于任务动态迁移的研究中，迁移决策算法研究倾向于解决宏观的迁移问题，通常是根据应用场景将迁移问题建模为数学优化问题，并设计相应算法予以求解。塔帕尔大学针对雾计算数据中心能源效率和服务可用性问题进行了研究，为任务选择和调度问题进行了建模，并通过合作博弈论方法解决该问题[12]。该工作所实现的是一个应用其所提出决策算法的任务卸载调度框架，具有任务调度、任务卸载和任务实时迁移等功能特性。佐治亚理工学院提出了一个雾计算中面向地理分布位置感知的应用增量部署和迁移框架[13]。该框架中应用了若干适用于在雾计算节点之间运行和迁移应用的算法，可以根据应用计算需求动态迁移计算任务。康奈尔大学的研究提出了虚拟机迁移策略 Supercloud，适用于在公共云服务商之间动态迁移虚拟机应用[14]。该工作将虚拟机应用的调度建模为数学问题，并设计了一个调度框架，通过在框架内的调度器上运行自动迁移算法调度虚拟机的迁移，以此获得最优的性能，且对服务和用户都透明。巴黎第六大学的工作则提出了一个云接入协议框架[15]，该协议框架应用了一个在线云调度算法，用于最优化虚拟机路由，并在数据中心之间迁移虚拟机实例。蒙特利尔大学的研究[16] 在迁移开销和用户体验（Quality of Experience，QoE）之间做出权衡，提出了一个基于移动性的服务迁移预测方法。该工作面向微数据中心之间的服务器迁移，预先对微数据中心之间的数据传输吞吐量进行评估，根据用户的移动模式将服务迁移至最优的微数据中心。考虑到当前许多研究计算弹性的工作更多的是面向动态编排增删容器实例的水平弹性问题，里尔大学提出了 ELASTICDOCKER，用于提升 Docker 容器的垂直弹性[17]。该工作所提出的主要策略是根据容器的动态资源占用率，动态调整为容器保留的各项计算、存储资源的量，充分利用宿主机资源，避免未使用资源浪费。当容器所需资源增加但宿主机又无法满足所需求的资源量时，该策略将通过容器迁移技术，将容器迁移到可以满足动态运行所需资源量的宿主机上继续运行。华中科技大学的工作将虚拟机迁移建模为一个连续时间马尔科夫决策过程[18]（Markov Decision Process, MDP），并设计了一个策略来决定当用户远离某个数据中心时，是否需要启动虚拟机迁移。早稻田大学采用深度强化学习的方法预先预测用户的移动模式，通过优化的迁移策略迁移容器，使容器的迁移更有预见性，更好地为用户提供计算服务[19]。该工作将用户在不同边缘网络之间移动时是否以及何时启动迁移的问题建模为一个顺序决策模型，并使用 MDP 予以求解，提出了一个基于深度 Q 网络（Deep Q-Network，DQN）的方法。该方法对用户的移动加以预测，判断迁移的必要性，并选择合适的时机在边缘网络之间有计划地迁移任务容器。华中科技大学提出了一个边缘认知计算（Edge Cognitive Computing，ECC）

的概念，并基于这个概念提出了一个动态服务迁移策略[20]。这项概念来源于对边缘计算和认知计算两个计算范式的结合，旨在边缘计算网络能够提供动态和弹性计算及存储资源的基础上，通过认知计算实现数据和资源认知，为附近的用户提供定制化的服务，使网络具有更深层次的、以人类为中心的智能认知能力。

1.2.2 边缘计算场景下任务迁移机制的设计与实现

除了上述任务动态迁移决策研究外，还有许多工作的研究内容是针对任务迁移机制的设计与实现的。和面向迁移决策算法的研究相比，这类研究通常针对更底层的机制开展研究，涉及网络通信机制设计与优化、系统级架构设计与优化等。纽约州立大学的研究针对同一物理机上运行的虚拟机之间的容器迁移场景进行了定向优化[21]。该工作通过内存重映射机制将原虚拟机中被容器所占有的内存空间的所有权转交给目标虚拟机，避免了额外的数据传输，减少了迁移的时间开销。IBM 的研究针对容器迁移中大量数据传送造成的时间开销进行了定向优化[22]。他们提出了数据联合和惰性复制等两个主要数据迁移机制，策略的核心思想在于优先使迁移的数据在目标宿主机上可用，随后根据需要逐步将数据传送到目标位置上。美国东北大学针对一些需要长时间运行，但大部分时间都处于空闲状态的应用容器（Long-Lived and Mostly Idle，LLMI，如私有邮件服务器、私有 DNS 服务器等），提出了一个介于基础设施即服务[23]（Infrastructure as a Service，IaaS）和平台即服务（Platform as a Service，PaaS）之间的平台框架[24]。目前已有的 IaaS 模式为用户提供虚拟机平台用于部署用户服务，但要求用户自行配置和对虚拟机进行管理，而基于容器的 PaaS 则要求用户的服务需要满足服务商所提供 API 的要求。该框架为用户提供一个传统的应用执行接口，允许大量容器在相互隔离的情况下安全地同时长时间运行。这就需要将部分长时间运行但没有服务请求的容器的内存状态截取并暂存至磁盘中。当有用户请求时，立刻选择一台计算资源满足容器需求的物理服务器，将该容器的内存状态快速传送并载入到服务器中恢复运行。通过使用该框架，云服务商可以为用户提供更加弹性的服务基础设施，且这个过程对用户是透明的，能够满足用户和服务提供商双方的需求和利益。帝国理工学院和 IBM 的研究针对任务动态迁移提出了一个分层框架[25]。该框架支持虚拟机和容器的实时动态迁移，将运行时环境层化为基层（一般是操作系统层）、应用层和实例层。当需要进行迁移时，可以由参与迁移的双方协商，仅传输那些目标服务器上没有的层数据，降低了动态迁移中传输的数据量，减少了迁移过程中的停机时间。类似地，威廉与玛丽学院的研究也提出了使用分层迁移的方式动态迁移容器[26]。该工作面向的应用场景为用户的终端设备在移动的过程中，通过分层迁移的方式将服务迁移到距离用户最近的边缘服务器上。该工作基于 Docker 实现，继承了 Docker

在数据层管理的方式，对分层的管理更加细化，减少了动态迁移需要的时间以及网络带宽。卡尔顿大学的研究主要面向容器动态迁移中的广域网网络传输中的网络拥塞和网络中断问题[27]。该工作提出了一个多路 TCP 协议作为容器迁移中的传输手段，通过多个子连接提升进程迁移的稳定性，减少迁移时间。罗格斯大学的工作提出了一个交通感知的容器迁移方法[28]，致力于设计一种通用的方法用于边缘计算场景下的容器迁移。该方法综合考虑了在不同负载、计算资源和网络带宽条件下的多种指标，包括用户体验（QoE）、系统响应时间和迁移开销等，在这些指标的基础上做出迁移决策。

1.3　本文研究内容与创新点

本文的主要研究内容与创新点如下：

（1）针对当前一些有代表性的跨平台迁移机制工作开展了对比研究。首先从应用虚拟化技术方面对目前主流的迁移工作进行分类，对这些主流的虚拟化技术的实现原理、系统架构、系统特点、适用场景和技术优缺点等方面进行了分析和比较。之后，根据应用类型和优化目标的不同分别对迁移任务进行了分类，列举了一些有代表性的工作，对不同运行机理的应用程序适合采用的迁移技术进行了概述。该研究的目的在于协助开发者和用户针对不同的应用场景和可用的硬件设备，选择合适的跨平台迁移技术和代表性迁移机制实现应用跨平台迁移。

（2）针对移动边缘计算场景下移动设备具有移动性、不稳定性，且边缘网络内设备具有异构性的特点，本文设计了一种异构平台间自适应和轻量级的任务动态迁移机制（Adaptive and Lightweight task dynamic Migration between Heterogeneous-ISA Platforms，ALM-HIP）。ALM-HIP 机制将跨平台迁移相关进程状态检查点转换组件集成到 Unikernel 内核模块中，允许在没有虚拟机管理器的支持下由双方应用实例自行协商迁移方式并完成跨设备迁移，支持端和边的任意组合。由于 Unikernel 应用在运行等级上属于系统级，因此即使是在计算和存储资源都严重受限的移动设备或嵌入式设备上，Unikernel 技术也仍支持系统级的用户程序直接运行于硬件之上，相较于基于虚拟化的运行技术（如虚拟机、容器等），能够获得更加优秀的性能表现。相较于有管理器支持的迁移机制，ALM-HIP 允许应用实例在直接运行于硬件上时，从系统层面完成迁移过程。这样的迁移机制能够满足实际应用场景需求，适应更多的硬件环境或任务执行环境。此外，本文将该机制应用于 5G+ 移动医疗救护场景，设计了一个基于网络状态感知的自适应进程同步策略（Network-aware Adaptable Process Synchronization，NAPS）。NAPS 策略旨在进程同步频率和网络通信状况间做出权衡，在减少进程同步开销的同时，将边缘服务器上尽可能

新的内存状态同步到临时服务设备。当移动设备和边缘服务器之间的连接断开时，临时服务设备可实例化最新的边缘端进程状态备份作为临时服务端，为边缘设备继续提供不间断的服务。当与边缘服务器的连接恢复或连接到新的边缘服务器时，则将最新的进程状态同步到边缘服务器上为边缘设备提供更高性能的计算服务。

（3）基于本文所提出的 ALM-HIP 机制，实现了一个适用于异构平台之间进行任务动态迁移的任务迁移框架，通过实验验证本文所提出的自适应跨平台迁移机制的可行性。此外，还针对本文所设计的 NAPS 策略进行了模拟实验，与固定间隔的进程同步策略进行对比，评估该策略的性能，在此基础上进行总结与讨论。

（4）实现了一个基于 Kubernetes 的多节点弹性任务卸载调度原型系统。该部分通过研究多节点任务调度问题，给出若干任务的集合在一个具有多节点的集群中的最优分配方案，优化任务的响应时间，提高系统的吞吐量，增强分布式系统的性能。设计和实现了一个适用于弹性任务计算场景的任务卸载调度原型系统，用于将一系列用户提交的计算任务卸载到一系列计算节点上，使任务的总响应时间最短。上述 ALM-HIP 机制和 NAPS 策略还被进一步整合到该系统中，使该系统在基本卸载调度功能的基础上，具备了跨平台的任务迁移和进程同步能力，从而可以较为全面地支持移动边缘场景下异构节点之间的任务卸载与迁移。

1.4　本文章节安排

本文结构分为六章，章节安排如下：

第一章是本文的绪论，首先介绍边缘计算场景下的任务动态迁移问题的研究背景和研究意义，然后对国内外针对边缘计算中任务迁移问题的研究现状进行调研，最后提出本文的研究内容及创新点。

第二章对主要关键技术进行了对比和分析，首先介了目前主流的虚拟化技术以及各自的适应场景及优缺点，然后针对不同类别的任务负载和不同的优化目标分别概括了一些具有代表性的典型工作，最后对这些代表性工作进行了对比分析，为后文选取相应的技术路线基础提供依据。

第三章提出了一种异构平台间自适应和轻量级的任务动态迁移机制 ALM-HIP，以及基于网络状态感知的自适应进程同步策略 NAPS。针对移动边缘计算环境中移动设备具有移动性、不稳定性和异构性的特点，设计一种自适应的、支持多种端-边组合的迁移机制，并详细阐述了该机制的实现过程。然后选取了 5G+ 移动救护的实际应用场景，对网络状态感知与自适应频度调节问题进行了问题建模，通过权衡同步频率和额外重复计算开销，最大化进程同步收益。

第四章主要介绍 ALM-HIP 的系统框架实现，详细介绍其系统架构和实现细

节，并通过实验对机制的有效性进行验证。此外，还针对提出的 NAPS 策略进行了性能评估，选取了不同网络环境评估该同步策略的性能。

第五章设计和实现了一个适用于弹性任务计算场景的任务卸载调度原型系统，可以将一系列用户提交的计算任务集合中的任务卸载到一系列计算节点上，使得各个任务的总响应时间最短。上述 ALM-HIP 机制和 NAPS 策略还被进一步整合到该系统中，为本章所提出卸载调度原型系统提供进一步的跨平台任务迁移与同步功能支持。

第六章对本文所做的工作进行总结和归纳，分析本文目前工作的不足以及可进一步优化的工作点。

第二章 边缘计算中跨平台任务迁移技术对比与分析

任务卸载与迁移问题是边缘计算中一个重要的研究对象，目前已有许多具有代表性和应用价值的工作。本章主要研究边缘计算中跨平台任务迁移相关技术，为第三章和第四章本文所提出的自适应任务跨平台迁移机制选择合适的基本技术路线。本章先对比边缘计算中跨平台任务执行与任务迁移的基本虚拟化技术，然后简述不同的应用种类和不同应用场景下的若干代表性工作，最后概述边缘计算中的跨平台任务迁移面临的挑战和相关研究进展。

2.1　面向任务动态迁移的虚拟化技术

边缘计算中的任务迁移通常需要依赖虚拟化技术以达到屏蔽某些系统差异的目的，从而为任务迁移提供有利的运行时环境和系统支持。虚拟化技术是一种用于将服务器硬件划分为若干物理上共享、逻辑上独立虚拟环境的技术。目前，虚拟化技术广泛地被云服务提供商和边缘计算场景采用。其中，最常用的虚拟化技术可分为虚拟机技术和容器技术。

2.1.1　虚拟机技术

基于虚拟机技术的虚拟化环境是一个系统级的虚拟执行环境，具有高度的隔离性。一般来说，虚拟机技术要求宿主物理服务器运行一个虚拟机管理器，该管理器既可以直接运行于服务器硬件设备上[29]，也可以先安装一个宿主操作系统，再安装虚拟机管理器[30-31]。该宿主机系统将占有物理服务器上的所有可用硬件资源，然后在该系统上新建虚拟机并为每个虚拟机安装一个内部操作系统（或操作系统级的应用），该系统只能占有该虚拟机所分配的所有逻辑资源。

在云计算场景下，虚拟机技术通常被用来将大型数据中心中的服务器划分为若干逻辑上隔离的服务器作为系统和任务的部署环境。虚拟机只需要通过复制虚拟磁盘镜像和相关配置信息，即可以被简单地从一台物理服务器上导出并迁移到其他物理服务器上或生成新的实例。然而，当需要动态迁移一个虚拟机实例时，需要将运行中虚拟机的所有内存状态、寄存器状态、网络连接状态和虚拟磁盘数据等传送到目标物理服务器上[32]。目前虚拟机内存数据和磁盘数据进行动态迁移的主流机制包括预拷贝[33]、后拷贝[34] 和混合拷贝[35] 等三类。

预拷贝方法[33,36] 指的是宿主机得到迁移指令后，将当前所有内存页面发送到

目标服务器上，此后迭代若干次，将系统运行过程中产生的“脏”内存页面迭代地更新到目标服务器后挂起，将最后一轮迭代之后到停机前产生的脏页面将被传送至目标服务器上，并启动迁移后的虚拟机。预拷贝方法的优势在于能够大大减少最后一步停机拷贝时需要传送的页面数量，减少了停机时间。然而，当某个虚拟机读写内存频繁，导致脏页面产生速率大于脏页面的传输速率时，迁移效率将大大降低，即迁移收敛问题[32]。

和预拷贝方法相反，后拷贝方法[34,37] 首先停止当前物理主机上虚拟机的运行并在目标服务器上启动被迁移虚拟机的新实例后恢复运行。新实例恢复运行后，通过按需抓取、主动推送和预先分页[34] 等方法逐步地将需要访存的页面抓取到目标服务器上。后拷贝方法可以减少拷贝过程中传送的数据总量，然而在动态迁移的过程中，新旧内存页面分别被放置在原服务器和目标服务器上。当出现迁移失败等异常情况时，可能会造成数据不完整或导致迁移后的虚拟机损坏。

针对上述两种拷贝方式的不足之处，有工作也提出了混合拷贝的方法[35,38]。混合拷贝方法先采用类似预拷贝的方法迭代复制若干轮内存数据，然后进入停机拷贝阶段，恢复目标服务器上的虚拟机运行后再迭代若干轮，将新的脏页面取回。混合拷贝方法通过有限的预拷贝节约了部分网络流量，但同时也继承了后拷贝方法鲁棒性较差的缺点。

由于预拷贝方法具有较强的鲁棒性，从而被 VMware[36]、Xen[33] 和 KVM[39] 等主流虚拟机管理器采用。虽然目前针对虚拟机动态迁移已有大量的研究，但是由于虚拟机具有体积庞大，部分系统级机能具有平台依赖性等特点，并不适用于边缘计算场景下的跨平台迁移。

2.1.2 容器技术

容器技术通过共享宿主机操作系统内核和为每个容器应用使用独立的文件系统实现轻量的隔离环境。容器的创建和销毁开销相对虚拟机而言都更小，便于应用部署和管理。容器通常采用共享宿主机操作系统内核的方式创建容器进程，而不是重新启动一个新的操作系统，这使得容器的创建时间远低于虚拟机的开机时间。但另一方面，容器的隔离性相对虚拟机较弱。典型的容器运行和管理平台或技术实现包括 Docker[40]、LXC/LXD[41] 以及 Podman[42] 等。

随着边缘计算的不断发展，越来越多的运行在网络边缘的设备（如网关、路由器、无线接入点和移动设备等）的计算能力和存储资源等逐渐受到了学界的关注。这些边缘网络中的设备通常具有受限的计算资源，不适合将体积较为庞大的虚拟机作为任务执行的载体卸载到这些设备上。为了充分利用这些位于网络边缘的计算和存储资源，需要通过一定的策略和机制将任务卸载或迁移到这些相对专业服

务器而言性能较低的设备上运行。

由于容器通常是运行在宿主操作系统上的一个或一组进程，容器迁移有时也会被描述为进程迁移。用户空间内的检查点截取/恢复[43]（Checkpoint/Restore In Userspace，CRIU）技术是容器/进程迁移中最常使用的技术之一。CRIU 技术通过将被迁移进程的各个线程的核心信息、所使用的文件描述符、进程程序树、进程的地址空间信息、网络设备及网络通信信息等保存为检查点文件。目标宿主系统通过载入检查点文件，可以恢复被进程被截取时的运行状态及相关运行时数据。

容器具有轻量、快速、隔离性强的特点，作为一种轻量的虚拟化方案，容器技术在边缘计算场景下得到了广泛的应用。为了吸取虚拟机技术和容器技术各自的长处，运行层次介于虚拟机和容器之间的 Unikernel 技术[44-45]也逐渐受到学界的关注。基于 Unikernel 技术，可以将用户程序与一个定制的精简 Linux 内核集成到一个应用镜像中，此镜像既可以运行于宿主机上，也可直接运行于嵌入式硬件或裸金属服务器上[46]。Unikernel 可被视作一个单应用的虚拟机，Unikernel 中的应用运行于单地址空间模式，独占整个 Unikernel 虚拟机。用户进程在专属的虚拟环境内运行，具有较强的隔离性，同时由于没有地址空间转换，运行性能更高。此外，Unikernel 具有容器单一进程、启动快的特点，因此被视作介于虚拟机和容器之间的一种程序执行环境。对于 Unikernel 而言，其唯一的不足在于需要为所有程序制作对应的镜像，这对于开发者而言并不够友好。

2.1.3　面向任务动态迁移的虚拟化技术对比分析

图2-1展示了程序在几种虚拟化机制中运行时的系统层级架构图，表2-1则从应用场景、性能、数据量、实例迁移速率、实例迁移开销、跨平台迁移支持和环境隔离性等若干方面对虚拟化技术进行了对比。

表 2-1　几种虚拟化机制的对比

虚拟化方式	面向场景	运行性能	数据量	迁移速率	迁移开销	异构平台支持	隔离性
虚拟机	云计算	一般	大	低	一般	差	良好
容器	云计算、边缘计算	好	小	高	低	一般	一般
Unikernel	云计算、边缘计算	好	小	高	低	一般	较好

虚拟机技术可以为任务提供高隔离性的执行环境，然而虚拟机内部通常需要安装一个完整的操作系统，导致一个虚拟机需要占用较多的存储资源，许多系统组件的运行也需要占用一定的计算资源。此外，虚拟机的启动速度与容器相比更

图 2-1　若干虚拟化机制的运行架构

慢，不适合对启动速度有较高要求的应用场景，虚拟机对硬件的依赖性也导致其对于跨平台迁移的支持性较差。

和虚拟机相比，容器与和 Unikernel 应用在性能和轻量性上都更胜一筹，更加适合边缘计算环境。然而，由于容器技术是基于共享宿主机内核实现的，容器实例之间的隔离性必然比虚拟机实例之间更差，无法满足某些应用的高隔离性要求。由于容器实例相当于进程级别应用，可以使用进程迁移相关技术有针对性地解析程序内存状态信息，有利于跨平台迁移的实现。

2.2　跨平台任务迁移

任务迁移是一项用于将任务运行的载体（虚拟机或容器）从一个运行环境（物理机、虚拟机或容器）迁移到另一个的过程。从迁移形式上分类，任务迁移可分为非动态迁移和动态迁移两类：非动态迁移主要指将一个进程的运行状态保存到磁盘上作为后备，可用于实例克隆、状态备份和恢复等；相对地，动态迁移指的是在

任务进程运行的过程中将任务进程状态拷贝到另一个运行环境中，在迁移的过程中仍保持进程的运行状态为用户提供持续的服务，对用户透明，可用于服务的快速复制[47]、负载平衡[48]、跟随用户移动性[49] 等。任务迁移是云服务提供商进行负载平衡或机器维护时转移检修设备上运行计算任务的手段之一。传统云服务提供商通常应用虚拟机迁移方法，而在当前新兴的边缘计算环境下，基于容器的动态迁移方式则更受到云服务提供商和用户的青睐。

2.2.1　边缘计算中动态迁移的异构平台问题

通常来说，一个可执行程序是若干平台相关机器指令和相关数据的结合体。当需要执行程序时，由操作系统将程序各部分代码和数据分别加载到内存上对应的位置，然后 CPU 从程序入口点处开始依次取出命令并执行。若试图将一个不同平台版本的可执行文件直接在当前平台上加载，则由于指令集体系架构的不同，会导致载入到代码段内存的指令无法被当前平台的 CPU 所识别，进而无法执行异构的命令。

对于迁移而言同样也是如此。动态迁移的实质是将正在执行中的进程状态截取，并在目标设备上通过恢复截取状态的过程，截取的状态包括程序在被迁移时刻的内存状态和寄存器组状态。恢复时，由宿主机操作系统载入寄存器组，找出迁移时刻的运行位置，让程序计数器指向该内存地址并继续取出指令和执行。与加载程序相同，若强行加载非当前设备 CPU 指令集的进程状态，则同样会导致载入后的代码段指令无法被当前设备的 CPU 识别和执行，导致程序运行失败，故直接迁移只适用于相同平台之间，不适用于异构平台之间的进程迁移。

网络的边缘附近充斥着大量的网络设备和移动终端设备，这些设备往往不是由相同指令集体系架构的 CPU 构成。这就导致一个任务进程不能在不同指令集体系架构的设备之间迁移，这样的设备体系结构异构性催生了学界对跨平台迁移的研究。通过应用跨平台迁移技术，一个任务可以在异构的设备之间迁移，消除了设备异构性对任务卸载设备的隔阂。本节将详细讨论一些典型的应用迁移工作。

2.2.2　面向不同种类应用的迁移机制

应用程序通常由一个或多个高级语言编写，这些高级语言按照执行方式可大致分为三类：解释型语言、编译型语言和混合型语言。

（1）解释型语言

由解释型语言所编写的程序，其特点是需要引入一个解释器程序对用户程序代码解释运行，并为其应用提供相应的运行环境，典型的解释型语言如 Java、Python、Perl 等。例如，Java 语言所编写的程序在编译为字节码后通常需要使用 Java 虚拟

机（Java Virtual Machine，JVM）作为其运行环境，而 Python 语言编写的程序则需要 Python 解释器来解析和执行其用户程序代码。这些解释器为用户程序提供的运行时环境包括但不限于 I/O、系统功能、垃圾回收（Garbage Collection，GC）等，这些由解释器提供的运行环境使得运行平台本身的操作系统环境和指令集等问题对于用户程序而言透明。正是由于解释器为用户程序提供了这样透明的运行环境，只需要为相应平台编译对应的解释器程序，用户程序即可在不同指令集平台上运行[48]。目前有一些工作面向解释型语言所编写应用的迁移与优化。

Chen 等提出了一个编程框架 COCA[50]，用来解决云计算中的卸载问题。COCA 利用了面向切面编程思想（Aspect-Oriented Programming，AOP）卸载 Java 应用程序。该工作通过为应用程序插入额外的信息实现将基于 Java 的应用程序卸载到其他的设备上运行。

Bruno 和 Ferreira 提出了一个 JVM 动态迁移机制 ALMA[51]，该工作的关键思想在于在垃圾回收和数据传送之间做出权衡，最小化迁移代价。ALMA 通过周期性检查应用的堆区内存为各个堆区评估垃圾回收所需要的代价，并将该代价和相应堆区内存的迁移速率比较，决定回收垃圾后迁移还是直接迁移各个堆区到目标服务器上。该工作通过综合评估每个堆区评估垃圾回收耗时和传送数据耗时，使得总体迁移耗时更少。

此外，这类解释型语言也可以引入即时编译（Just In Time，JIT）或预编译方法，将高级语言代码提前编译成对应语言的解释器内部表示，或者编译成平台相关的指令码。但就性质而言，这类程序的平台依赖性通过解释器或即时编译器得到了消除，使得它们可以较为容易地在任意安装有相应解释器的平台上运行。

（2）编译型语言

编译型语言程序通常需要一个高级语言编译器生成目标平台可执行代码，在运行时由操作系统将程序各部分对应直接载入到相应内存区域，最后从程序入口点开始执行。由于编译型语言的可执行程序的平台对应性极强，因此和解释型语言程序相比，编译型语言的跨平台迁移更加复杂。编译型语言程序的迁移，要求迁移时对程序的相关寄存器状态、内存区域、系统文件、网络连接等进行识别、收集、迁移和恢复。目前也有不少工作面向编译型程序的跨平台迁移开展，以期消除异构平台之间的隔阂。

Barbalace 等[48] 在操作系统层面实现了一套完整的工具链，包括基于 Popcorn Linux[52]、自定义的多平台编译链和异构平台对迁移的运行时支持。该工作通过其设计的多平台编译链编译和生成多平台版本的可执行程序代码，将异构平台程序代码中的符号地址对齐，确保函数入口点和变量地址在跨平台转换前后保持一致。

上文所提到的 CRIU 技术可以将一个容器或进程的所有状态存储为一系列检查点文件用于在其他设备上通过恢复检查点实现进程迁移。然而值得注意的是，CRIU 所截取的是一个特定进程的所有信息，包括寄存器组、代码段内存等。这些信息和平台高度相关，如果仅仅只是简单地将 CRIU 所截取的镜像传送到一个异构平台上，是不能够直接恢复被截取进程的状态。因此，当需要应用 CRIU 技术进行检查点截取/恢复时，需要对检查点文件进行额外的操作，将检查点文件转换为目标异构平台相关的形式。目前，有许多跨平台迁移的工作是基于 CRIU 的。

Barbalace 等提出了 H-Container[49]，可以用于将一个容器迁移到异构的平台上运行。H-Container 包含一个反编译器，可以将被迁移进程的可执行程序反编译为 LLVM IR 作为中间表示，同时向程序中插入若干“迁移点”，被反编译成 LLVM IR 的中间表示将被重新编译为目标异构平台的可执行程序。这些迁移点被用来收集和转换程序状态，进程必须运行到下一个迁移点处才能触发迁移。H-Container 的主要贡献在于它对运行中进程的当前状态进行了平台转换，使其能够适合于在目标异构平台上运行。

Unikernel 技术同样也被用于跨平台迁移。Oliver 等提出了一个跨平台执行卸载框架 HEXO[53]，它利用 Unikernel 技术，将高性能计算数据中心中高性能专业服务器上的计算任务迁移到若干硬件和能源成本更低的嵌入式平台上运行，获取更高的能效比。HEXO 要求被迁移的任务在源代码中插入迁移点相关代码调用后使用其提出的跨平台工具链，编译并制作该程序的多平台 Unikernel 镜像。

进程迁移的研究同样出现在针对异构单芯片多处理器（Chip MultiProcessors，CMP）的应用场景中。DeVuyst 等[54] 研究了如何以较小的性能损失实现将一个进程在 CMP 的异构核之间迁移，从而充分利用 CMP 的计算资源。他们的工作包括识别应用进程状态、定制编译器以及二进制翻译。该工作通过使用定制编译器，使一个程序本身及其数据在执行时能够放置于内存中合适的地址上以减小迁移开销。工作的核心思想在于使一个程序在运行时，其相应平台所对应的进程在内存中保持原生形式运行，以此来达到一个可接受的性能等级。

Bhat 等则提出了一个多内核操作系统[55]，实现进程在一个同时具有 ARM 和 x86 处理器的异构平台上不同处理器，该系统通过 Popcorn Linux 扩展而来，并带有一个定制的编译器。该系统属于典型面向特殊异构平台的工作。在该异构平台上，两个架构均分别运行一个原生编译的内核，两个内核之间通过 Popcorn 通信层进行消息传递。Popcorn 镜像和用户应用都分别同时运行在两个内核之上，其中应用由定制编译器构建而来，编译出的应用程序同时含有两个架构版本的机器码。执行时，分别载入两个架构版本程序到内存中，操作系统负责将对应部分的代码

映射到对应的异构平台上，对应的函数入口点和变量的地址则在同一虚拟地址上。当迁移进程的时候，只需要将源平台的寄存器组信息通过通信层传送到目标平台内核，即可快速实现任务进程的迁移。

（3）混合型应用

混合型应用指的是由一个或多个上述解释型语言和编译型语言编写的程序，或是程序中含有两种类型的程序片段。目前，Android 系统是世界上使用最广泛的移动（嵌入式）操作系统之一，被众多厂商安装在其生产的智能设备上，如智能手机、智能电视等。根据官方文档[56] 所述，一个安卓应用可以调用由 C 或 C++ 语言编写的原生库代码或模块，以达到提高 Android 应用性能的目的。这些库或模块通常是那些在程序中被频繁调用或需要进行大量计算的模块，可以通过官方所提供的 NDK 组件实现调用。

Lee 等[57] 最先对 Android 平台上混合型应用的迁移开展了研究。该工作的统计数据显示，应用市场上大量的 Android 应用都选择使用大量的编译型模块以提高应用性能。目前安装量和使用量最多的 Android 应用程序，如 Firefox 浏览器、VLC 播放器等移动应用等，其中由 C 或 C++ 编写的代码比例超过了整个应用 50%。目前市场上大多数安装有 Android 系统的智能设备都是基于 ARM 处理器，而商用服务器和个人电脑通常采用 x86 系列处理器。由于目前移动设备受到散热、功耗等限制因素的影响，其性能与 x86 系列服务器仍与专业服务器有较大的鸿沟。为此，该工作提出了一个原生卸载器（Native Offloader），用来将移动应用中的部分繁重计算模块独立地迁移到 x86 服务器进行运算。该工作的核心贡献是设计了一个定制的编译器分析应用代码，并将原始代码划分为客户端中间代码和服务端中间代码两部分，再将中间代码编译到两个平台上。当工作负载运行时，客户端将标识符、栈指针和页表等相关数据通过网络发送给服务器，然后从服务器接收脏页面，实现计算迁移。

2.2.3　面向特定场景优化的迁移机制

本小节将讨论现有工作，并将它们从适用的场景以及主要用途进行分类。主要分为跟随用户、提升性能、优化能源效率等三大类。

（1）跟随用户

边缘计算的初衷是将计算卸载到物理上和网络拓扑上更靠近用户的位置上，以此为用户提供低延迟、高带宽的服务。然而，在边缘计算场景下的终端设备通常具有移动性和不稳定性，如何解决当用户移动时任务的动态卸载和迁移问题，是研究的主要目标之一。

H-Container[49] 的主要目标是为任务在移动边缘云环境中的迁移创造尽可能

多的可供选择的迁移目标，从而使对低延迟需求较为迫切任务，如游戏、实时计算等服务，能够选择到最靠近用户的边缘设备上。边缘网络中的设备具有极强的平台异构性，边缘网络中参与计算的设备不再限于服务器和个人电脑，包括路由器、网关设备、无线 AP，甚至任意的嵌入式单板设备都有可能作为任务迁移的目标，而这些设备则可能由 x86、ARM，甚至其他的 CPU 构成。通过让容器具有迁移到异构平台的能力，当一个容器需要迁移时，边缘环境中将会有更多的候选设备可供选择用来为用户提供更高质量的服务。

（2）提升性能

边缘网络环境中的设备以移动设备、嵌入式设备和物联网设备等为主，由于这些设备的性能往往比主流的服务器等专业设备要差的多。因此，当用户需要在移动设备上运行一些计算密集型任务时，用户就会希望将计算负载卸载或迁移到一个拥有更强计算性能的边缘服务器上以达到加速计算，提升应用性能的目的。这样的思想在 Lee 等[57] 的工作上可见一斑。该工作通过将一个完整的应用划分出计算密集型的部分，并将这部分对性能需求较高的代码迁移到服务器上执行，使应用的整体性能得到了提升。

（3）提升能效比

能源问题和环境问题越来越受到全世界的关注。目前，关于边缘计算的工作中也有越来越多的相关工作为了达到减少能源消耗或高效利用能源的目的，提出了相应的改进措施。例如，Barbalace 等[48] 工作中通过实验评估表明，通过将应用迁移至 ARM 嵌入式平台，可以减少约 30% 的能源消耗。

相似的观点还可以在 HEXO[53] 的工作中看到。该工作通过将高性能计算负载迁移到若干嵌入式设备上同时运行，用更少的能源完成同样的任务计算，能源效率得到了提升。加利福尼亚大学[54] 和弗吉尼亚理工大学[55] 工作同样希望通过将进程迁移至异构单芯片多处理器平台上不同架构的核上运行，获得更高的能源效率。

2.2.4 对比分析与讨论

前文对基本技术路线和典型工作进行了概述，描述了这些技术和典型工作的特点和技术实现的大致原理。本节将对前文所涉及的相关工作进行梳理，讨论如何根据一个特定的应用场景，选取合适的技术路线或典型工作。

（1）对比分析

表2-2展示了部分上述工作在若干方面上的对比情况，图2-2则根据工作提出的先后顺序列出了相应的时间线。

（2）面向特定应用场景的技术路线选择

图 2-2　典型工作的提出时间线

图2-3展示了一条技术路线的选择路径，针对开发者和用户的需要提出建议，根据特定的应用场景选择相应的技术路线实现任务迁移。需要值得注意的是，图中列举的工作可能不会完全符合某个新的特定场景需求，但可以基于一些相似的工作针对实际需求定制。例如，由于 Python 和 Java 同为解释型语言，二者具有一定的相似性。当需要迁移一个 Python 应用程序时，可以采用一个类似于 ALMA[51] 的方法或其他尚未列出的方法实现。

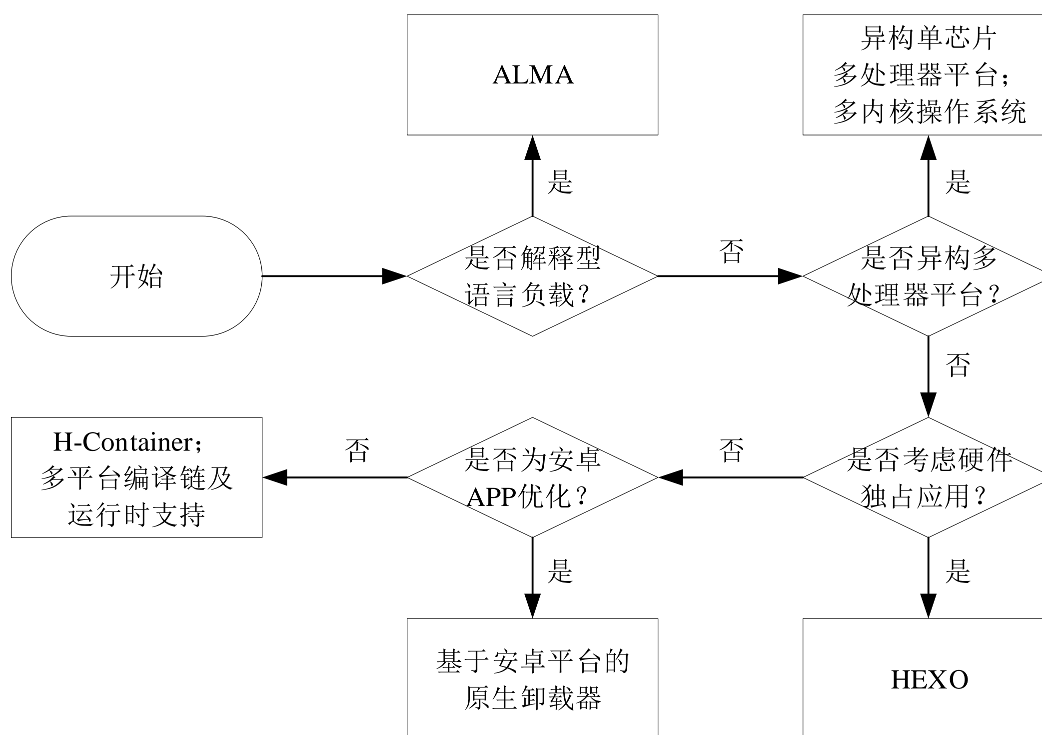


图 2-3　面向特定应用场景的技术路线选择

2.3　任务迁移所面临的挑战

虽然目前学界提出了许多应用迁移相关的工作，但目前仍有许多亟待解决的开放问题。随着边缘计算的不断发展，任务迁移将会有更重要的意义，同时还会

有更多的挑战和问题需要得到不断的关注。

（1）用户隐私和迁移安全问题

当前，隐私保护问题受到了个人用户和服务提供商等的强烈关注。任务迁移作为边缘计算中的一项关键研究领域，其隐私问题和安全保护问题应该得到更多的关注。

任务的执行离不开数据，而数据即隐私。在边缘计算场景下，虽然用户能够受到任务迁移带来的便利，能够享受延迟更低、带宽更高的高性能服务，如何确保用户产生的数据不被泄露或窃取，是一个值得讨论的问题。所谓保护，不仅仅指确保任务所运行的虚拟机或容器的安全性，在应用的迁移过程中的数据在网络上传输的安全性，同样也应该由更加有效的措施来确保。

（2）弹性的跨平台应用迁移问题

正如典型工作所展现的，目前的诸多跨平台迁移方法大多需要获取源程序代码，或对源程序代码进行一定适应性修改以满足相应的解决方案。然而，程序的源代码大多时候属于商业机密，大多开发商并不希望自己的源代码被公之于众。在不公开代码的情况下，则跨平台的适应性修改由开发商负责，又将导致巨大的程序维护难度，也不利于应用的版本迭代。

（3）面向特定边缘场景的优化问题

虽然现有的关于跨平台迁移的工作针对不同的可能出现于边缘场景中的异构平台进行了一定研究，现有研究中仍罕见面向真实用户场景下的应用研究。为了使这些现有的解决方案更加适应边缘计算场景，还需要对这些解决方案进行更进一步的优化和适配，并在真实的边缘场景中对其进行性能评估。

2.4 本章小结

本章对一些典型的任务迁移基本技术和典型工作进行了概括、对比与分析。首先，本章概括了几项基本的技术路线，即虚拟机、容器和模拟器等，对比了它们的适用场景和优缺点。然后，针对不同的应用类型和优化目标，对一些典型的工作进行了概括性论述，还进行了对比与分析。其次，本章给出了一个当需要应用此类技术时的技术选择路线图，帮助开发者和用户分辨和选择相关技术。最后，对当前边缘计算中任务跨平台迁移所面临的挑战进行了梳理。

此外，本章的论述内容同样也是本文主要工作开展的重要依据。

表 2-2　典型工作的对比情况[a]

对比点	异构单芯片多处理器平台[54]	基于安卓平台的原生卸载器[57]	多内核操作系统[55]	ALMA[51]	多平台编译链及运行时支持[48]	HEXO[53]	H-Container[49]
虚拟化技术	容器	无	无	Java 虚拟机	容器	Unikernel	容器
工作年份	2012	2015	2016	2017	2017	2019	2020
面向平台	异构单芯片多处理器平台	移动平台	异构单芯片多处理器平台	通用平台	通用平台	通用平台	通用平台
负载程序类型	编译型	混合型	编译型	解释型（Java 应用）	编译型	通用	编译型
应用场景	异构单芯片多处理器平台	安卓原生代码	异构单芯片多处理器平台	通用	通用	高性能计算数据中心	边缘场景
主要目的[b]	能、性	能、性	能	错、负、能、快、热等	能	能、性	用
系统组件[c]	编、翻、扩、特	检、中	编、扩、特	JVM、CRIU	编、中	编、扩	编、中、翻、CRIU
局限性	边缘环境中此类平台较少	缺少基于实际移动应用的评估	边缘环境中此类平台较少	仅适用于 Java 应用	需要获取程序源代码	需要获取程序源代码	缺少针对安全性的保护措施
移动设备支持	否	是	否	否	是	是	是

[a] 按工作发表年份从左至右排列。
[b] 主要目的的缩写包括：错（错误容忍）、负（负载均衡）、能（能源效率）、快（快速启动）、热（热升级）、用（用户移动性）、性（性能）。
[c] 系统组件的缩写包括：编（定制编译器）、扩（扩展系统）、特（特殊平台）、中（中间表示）、翻（二进制翻译）、检（检查点截取/恢复）。

第三章 一种异构平台间自适应和轻量级的任务动态迁移机制

针对移动边缘计算环境中的用户移动设备具有异构性、移动性和不稳定性的特点，本章提出一种异构平台间自适应和轻量级的任务动态迁移机制 ALM-HIP 和一种基于网络状态感知的自适应同步策略 NAPS。

ALM-HIP 基于 Unikernel 技术，通过为 Unikernel 应用内核添加跨平台迁移模块及通信模块，使 Unikernel 应用能够在无宿主操作系统支持的嵌入式平台上运行时的跨平台迁移。由于 ALM-HIP 摆脱了应用对操作系统或虚拟化管理器等的依赖，且支持跨平台迁移，因此能够适应各种边缘服务器、移动设备之间任意设备作为迁移对象和目标平台的任意组合，具有很强的自适应性。

另外，本章还提出一种基于网络状态感知的进程同步策略 NAPS。NAPS 基于网络动态评估设计，通过对当前移动设备和边缘服务器之间连接的信号强度、传输时延、测量带宽等若干指标进行综合考量，评估当前设备与边缘服务器的连接情况以及连接中断的概率。基于上述评估值，可以动态调节移动设备和边缘服务器之间进程的同步间隔。当信号较差时，在不影响服务的前提下尽可能将边缘服务器上最新的状态同步到本地作为后备，以备连接中断时由本地临时服务设备恢复运行，临时为移动设备连续提供相较于边缘服务器性能相对较差的服务，直至移动设备与原边缘服务器连接恢复，或与新的边缘服务器建立连接。

3.1 问题描述和建模

3.1.1 边缘计算场景下的端-边协同应用

随着移动计算技术的普及，越来越多的移动设备都作为互联网的终端，加入到整个网络环境中。由于移动设备天生受到电池续航能力、散热能力和设备体积等因素限制，其续航能力和计算性能都受到了很大的制约。

“云应用”诞生和流行于云计算场景，是一类将大部分的计算和数据放置于云端处理，本地设备仅作为交互和显示终端使用的应用。当用户使用云应用所提供的服务时，需要和云端建立持续的网络连接。通常云应用和云端交互通常采用移动 APP 或浏览器页面等方式实现，两种方式的实质均为通过调用云端提供的 API 进行服务的交互，云应用是“软件即服务”（SaaS[58]）的一种典型形式。其中，比

较有代表性的应用包括网盘类应用（如 Dropbox、百度网盘等）、团队协同类应用（如谷歌文档、腾讯文档等）、社交服务类应用（如 Facebook、Twitter 等）和部分游戏应用。

传统云应用在边缘场景下，仍然能够熠熠生辉。马来亚大学的研究[59] 指出，将应用的部分计算负载迁移到云端执行，移动设备仅作为交互终端这一“云应用”的通常做法虽然能够为移动设备减轻一部分计算负担，但这样的执行模式易受到网络质量的影响，从而影响到用户使用。于是，一种新型的“云应用”策略可根据边缘计算的特点提出：将“云应用”的计算密集型部分负载卸载到边缘服务器上执行，利用边缘网络低延迟的特性，为用户提供更即时的服务，在一些延迟敏感型的应用（如游戏应用）上进一步提升用户的使用体验。本文将这类在边缘网络中运行的“云应用”统称为端-边协同应用。

3.1.2 边缘移动设备的不稳定性

目前无线网络仍在发展过程中，对于全球大多数国家和地区而言，无线网络基站的建设都主要完成了城市、集镇等人员相对密集区域的无线信号覆盖。即便是在信号覆盖率较高的城市区域，根据所处位置的不同，移动设备能够接收到的信号强度变化也非常大。在一部分楼房、隧道、地铁等区域，无线信号也仍然会受到一定的阻隔，导致网络信号减弱或暂时中断。马来亚大学的研究针对某城市乘坐地铁过程中智能手机所接收到的 3G 网络信号强度进行了评估、收集和统计分析[59]。统计结果显示，在 20min 的地铁旅途中有 35% 的时间移动设备处于与云服务器断开连接的状态，而有 33% 的时间连接延迟超过 400ms，这样的网络环境势必导致基于云的应用的用户体验（QoE）下降。

与在城市中相比，医疗救护则面临更严峻的问题。当出现疫情等重大突发公共卫生事件时，往往需要快速组建重症监护体系。由于医院重症监护设备数量有限，为了快速组建重症监护环境，需要临时调用重症监护设备，例如为急救车部署重症监护设备，在接收病人时当即开始对其生命体征的监护和初步病情筛查，珍惜急救时间。针对上述移动重症监护医疗场景，目前的重症监护设备往往只能作为数据采集终端，其计算能力非常有限。在移动重症监护过程中，需要对病人的监测数据做即时分析，若将监测数据上传到云数据中心进行分析诊断具有较高的时延和网络带宽开销，且隐私数据更倾向于在本地进行分析，故选择采用端-边协同模式对数据进行分析和处理，将服务端卸载到距离救护车所处位置最近的边缘服务器上。救护车在移动过程中，会受到途经不同区域移动网络信号覆盖强度差异带来的网络波动，导致和边缘服务器的连接不稳定，或是在进入另一个区域后失去和原边缘服务器的连接。

如何解决移动设备在网络边缘环境中的移动性和不稳定性，是本章要解决的主要问题之一。

3.1.3　边缘计算节点间的异构性

和传统云计算场景不同，边缘计算的特点在于充分利用位于网络边缘的各种各类计算和存储资源，给用户提供低延迟、高性能的服务。然而，一个区域网络的边缘则充斥着数以万计的移动设备和网络设备，如智能手机、路由器、网关、无线 AP 等。这些设备或是出于能源消耗需求，或是出于性能需求，通常会使用不同架构指令集的处理器，以更好地适应其应用环境。程序是由处理器执行的指令和所需数据的集合，因此一个程序必须能够被对应的处理器架构所识别和支持，才可在该架构上运行，进而导致一个应用必须发布对应平台的版本，才能允许应用在不同的平台上运行。

3.1.4　问题建模

本文针对上述边缘环境中的设备具有移动性、不稳定性和异构性的特点，提出一种异构平台间自适应和轻量级的任务动态迁移机制 ALM-HIP。该机制下的端-边协同应用的网络拓扑结构如图 3-1所示。

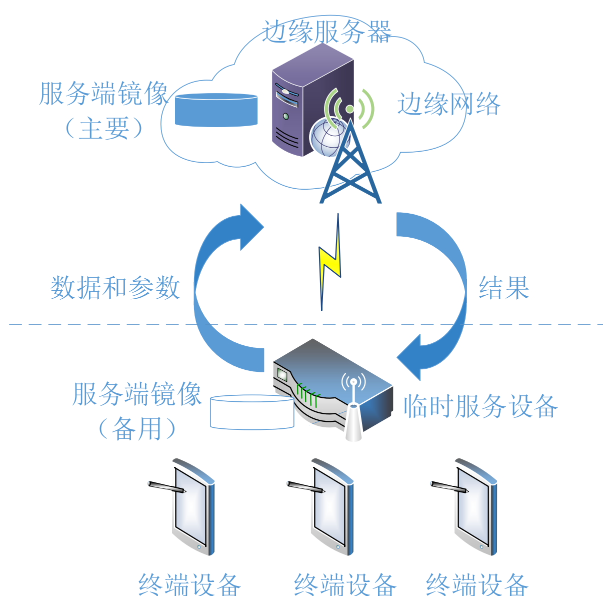


图 3-1　移动救护场景下端-边协同应用的网络运行架构图

以在移动的救护车上组建重症监护的移动救护场景为例，重症监护通常含有

若干终端采集设备，对病人的生命体征等进行实时监控，并将这些数据经由收集设备采集后上传到边缘服务器上进行数据分析。以下介绍本文所提出机制中涉及的若干概念。

边缘网络：一个边缘网络由一个或多个充当边缘服务器角色的设备，以及若干移动终端设备组成，移动终端设备通过无线基站和边缘服务器相连。移动设备作为应用的交互终端，用于将计算所使用的数据和参数提交到边缘服务器上，由边缘服务器所运行的对应应用的 Unikernel 服务端镜像为移动设备提供服务。移动设备由于具有移动性和不稳定性，和边缘服务器的网络连接状态会有一定波动，网络连接状态由两个指标衡量：延迟和带宽。当移动设备与边缘网络连接情况较差时有可能会随时离开边缘网络，然后在一定时间内与原来的边缘云网络重新建立连接，或连接到一个新的边缘网络中。

本文中所提及的边缘服务器和移动设备都是相对宽泛的概念。边缘网络中任何具有计算能力、具有稳定的能源供应、可以用来长时间卸载服务端镜像的设备统称为边缘服务器。边缘服务器被认为是长期稳定存在，边缘服务器到基站之间的连接的长期有效且可靠的。端-边协同应用中的移动设备则指仅作为应用交互终端，只用于数据显示、用户交互的设备，可能是智能手机，也可能是嵌入式设备，移动设备通过无线网络和基站相连，进而连接到边缘网络中。一个边缘服务器上可能同时运行多个应用的 Unikernel 镜像，每一个镜像既相当于一个虚拟机，又相当于一个系统级进程，这也是 Unikernel 应用介于虚拟机和容器之间的重要特征。

临时服务设备：临时服务设备是车载无线通信网关（后文简称移动网关），各终端设备通过车内无线或有限网络与该网关相连。移动网关通过广域无线通信方式（如 4G、5G 等）与无线基站建立连接，最终连接到某个移动边缘网络上的边缘服务器。临时服务设备上放置一个备用的服务端镜像，当与边缘网络断开连接时，该设备上将启用备用服务端 Unikernel 镜像，伺候将由该移动网关为各终端设备提供临时服务。由于该设备实际上由无线通信网关兼任，故其通常为性能相对较低的嵌入式平台，且由于厂商的不同，通常具有异构性。

进程迁移：进程迁移指将服务端镜像的实时运行状态根据需要进行架构转换，并将转换后的进程状态传送到本地临时服务设备上用以恢复运行，该设备可以为任意架构的嵌入式设备，安装有服务端镜像的备份镜像。

进程同步：本地临时服务设备在大多时间内仅仅和边缘服务器建立进程同步关系，每隔一定间隔将服务端镜像的运行状态同步到本地作为备份，期间终端设备仍直接和服务端镜像进行交互。当本地终端设备和边缘服务器连接中断时，启用本地临时服务设备，为终端设备不间断提供低延迟，但性能相对边缘服务器稍

低的服务。

3.2　异构平台间自适应和轻量级的任务动态迁移机制

本节详细描述本文提出的异构平台间自适应和轻量级的任务动态迁移机制ALM-HIP。假设整个任务执行的过程中不会因为程序本身问题导致程序异常退出，且当前实验环境中有且只有参与实验的设备和任务，不会与其他类型任务竞争计算和网络资源，在该前提下构建本文所提出的自适应跨平台任务状态同步机制。

3.2.1　跨平台进程迁移的关键技术

Unikernel 是一个新兴的容器技术，它将一个精简的内核与用户程序封装在一个镜像中。一个 Unikernel 实例可以认为是一个单进程的虚拟机，这一运行机制通过在操作系统级别运行用户程序，消除了传统虚拟机技术中的虚拟机内部的页表转换过程，提升了单进程实例的性能。同时由于镜像中仅包含程序需要的库和系统服务，因此也更加安全。

一个 Unikernel 实例的地址空间排列如图 3-2所示。从图中可以看出，Unikernel 实例内部是一个单地址的内存空间，用户程序直接调用内核的相关功能模块使用所需的系统服务（如文件读写、网络服务等）。

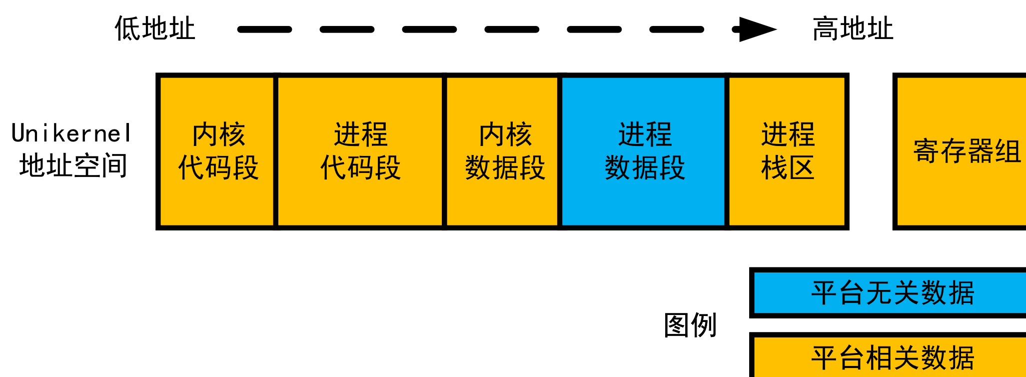


图 3-2　Unikernel 实例的地址空间排列

基于 Unikernel 的跨平台迁移机制技术架构如图 3-3所示。

在部署 Unikernel 镜像前，需要使用程序源代码生成两个平台对应的程序，并分别生成两个 Unikernel 镜像。关于源程序，本方法要求在程序中插入“迁移点”，其实质是一个定制的系统调用，执行一段特定存储程序状态和进行跨平台转换的代码。不同平台的设备分别载入对应平台版本的 Unikernel 镜像，当程序得到迁移

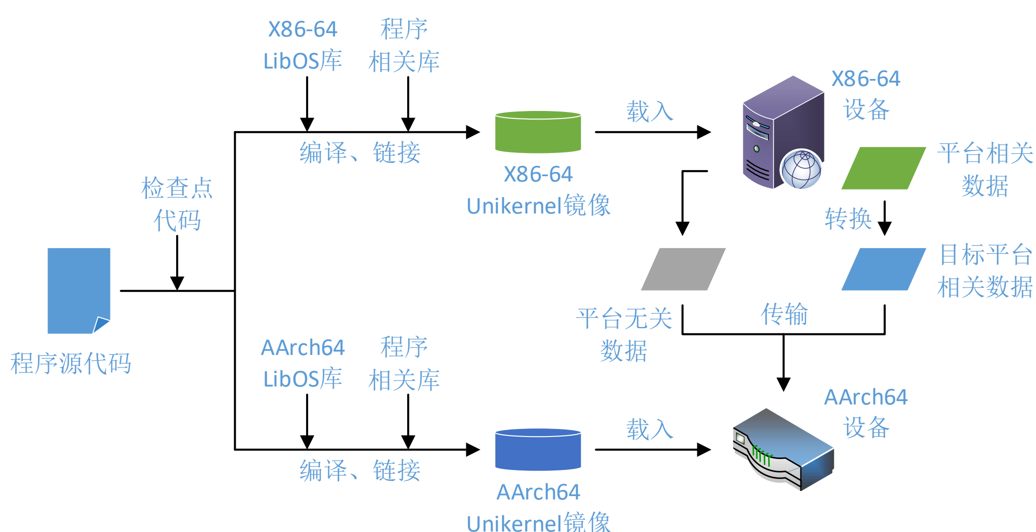


图 3-3　基于 Unikernel 的跨平台迁移机制技术架构

指令后，会运行到下一个迁移点处执行迁移代码，触发进程迁移的系统调用。系统调用触发后，宿主机管理器会和内核进行交互，根据实例内存地址空间种类的不同识别出需要迁移的部分内存，并将该部分内存划分为需要转换状态和不需要转换等两类，如图 3-4所示。

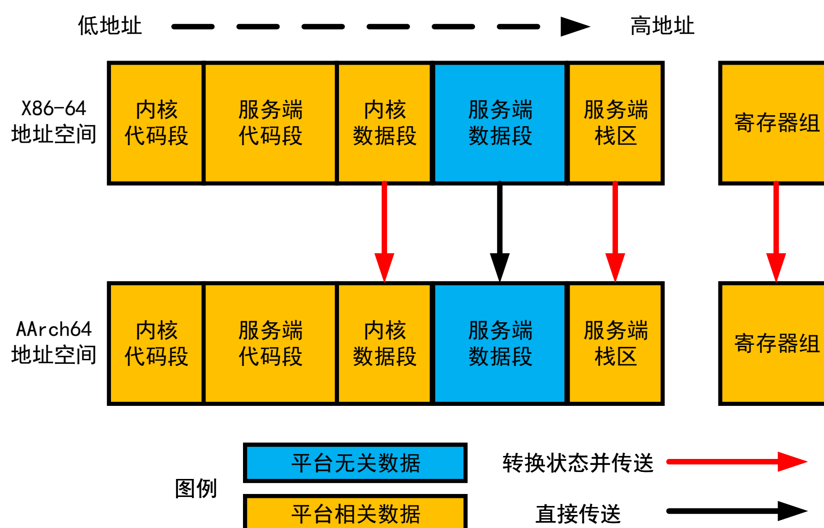


图 3-4　待迁移实例之间的地址空间关系

对于需要状态转换的部分内存（如用户程序栈区内存），管理器将执行状态转换相关代码，将其转换为目标平台的平台相关形式后传送；对于平台无关的部分内存（如用户程序数据段内存），管理器将直接将截取到的内存数据传送至目标设

备上。

恢复运行状态的过程是相似的，也需要宿主机管理器和客户内核之间共同协作，将接收到的内存数据恢复到地址空间中相应的位置上。HEXO 所提出的跨平台迁移机制是通过虚拟机管理器和客户系统内核共同协调完成的。

然而，并不是在任何应用场景下都具有宿主机支持的虚拟机运行环境，这就为其工作带来了一定的局限性。如何在无宿主机支持的环境中通过 Unikernel 内核独立完成内存截取和转换，是本文的主要研究目标。

3.2.2 异构平台间自适应和轻量级的任务动态迁移机制

本文基于 Unikernel 运行机制以及弗吉尼亚理工大学的工作 HEXO[53] 设计了一种异构平台间自适应和轻量级的任务动态迁移机制 ALM-HIP，该机制如图 3-5所示。ALM-HIP 的自适应主要体现在能够满足边缘环境中各种端、边之间的组合作为被迁移的实例的迁出设备和迁入设备，无需虚拟机管理器等上层管理程序支持。

如图 3-5，假设边缘服务器采用 x86-64 架构平台，临时服务设备采用 ARM 架构平台。由于终端设备架构与迁移无关且仅运行客户端，和服务端设备通过网络通信交互，因此可以是任意平台的设备。算法 3.1说明了 ALM-HIP 机制中一次跨平台迁移的过程。

算法 3.1 自适应的跨平台迁移机制（ALM-HIP）

输入： $Image_{ISA-A}, Image_{ISA-B}$

1: $Host_A \leftarrow Image_{ISA-A}$
2: $Host_B \leftarrow Image_{ISA-B}$
3: $start(Host_A)$
4: $start_migration(Host_A)$
5: $syscall(SYSMIG)$
6: $mem_stat \leftarrow identify_mem(Host_A)$
7: $mem_stat \leftarrow convert_mem(mem_stat)$
8: $checkpoint_file \leftarrow save_to_checkpoint(mem_stat)$
9: $Host_B \leftarrow checkpoint_file$
10: $restore(Host_B, checkpoint_file)$
11: $start(Host_B)$

以进程从 x86-64 设备上动态迁移到 AArch64 设备上为例，结合图 3-5中标注的流程顺序（以字母 A~G 注明），说明如下：

首先编译出应用的 x86-64 版本和 aarch64 镜像，分别载入 x86-64 和 aarch64 设备中，之后 x86-64 设备上的实例启动运行，aarch64 上的实例不启动运行，等待迁移的状态到来。接着，x86-64 设备启动迁移流程，运行到下个迁移点处触发系统调用，进入跨平台迁移过程。进入迁移过程后，首先通过内核本身识别待迁移进

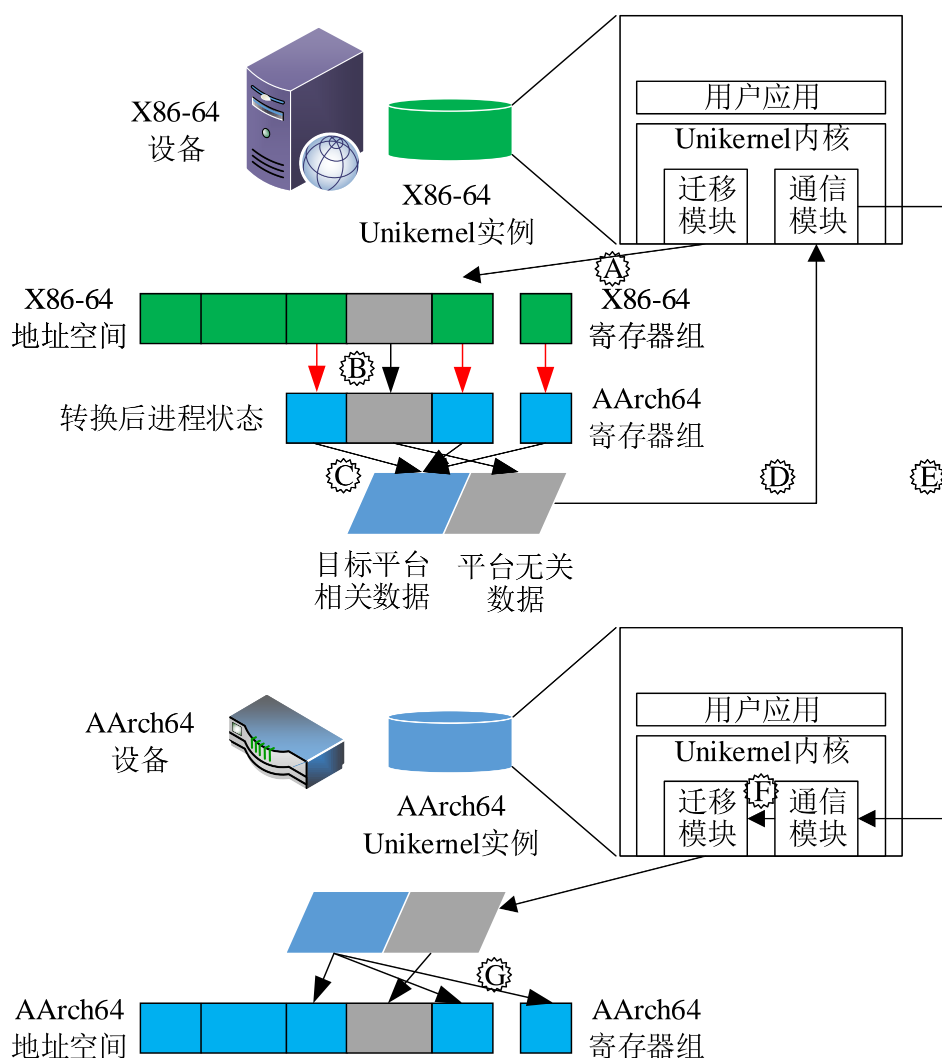


图 3-5 ALM-HIP 机制示意图

程的各个内存区域（A），然后将识别到的进程状态进行跨平台转换（B），并保存为检查点文件（C）。将上述检查点文件送入 x86-64 实例的通信模块（D），该模块与 aarch64 实例的通信模块交互，传送检查点文件到目标实例上（E）。目标实例接收到检查点文件，进行完整性校验并送至迁移模块等待进程恢复（F）。目标实例迁移模块解析收到的检查点文件，将其中的相关数据载入到内存对应地址上（G），载入完毕后即可在目标实例上恢复迁移进程的运行。

在 ALM-HIP 机制中，Unikernel 内核将负责以下所有的过程：

1. 原实例内存空间的地址区域识别与截取；
2. 需转换部分内存的跨平台转换；
3. 被迁移实例和目标平台上 Unikernel 实例之间的内核控制和数据通信；

4. 目标实例用检查点文件覆盖当前实例内存空间（或使用后拷贝方法）。

ALM-HIP 由若干部分组成，主要组件和工具链包括：跨平台应用镜像生成工具链、基于 Unikernel 的程序运行环境、跨平台迁移转换内核模块和跨平台迁移内核通信模块等。

跨平台应用镜像生成工具链： 该工具链用于对程序源代码进行多平台编译，利用定制的编译器，以 newlib① 作为程序的运行时库，将应用本身与运行时库进行静态链接。链接过程中，使每个符号的地址分别对齐在相同地址上，便于跨平台时内存的截取与恢复。

基于 Unikernel 的程序运行环境： 将生成的程序定制内核共同生成一个完整的 Unikernel 镜像，一个 Unikernel 镜像为一个应用程序单位。

跨平台迁移转换内核模块： 该模块是本文的主要贡献之一，通过为 Unikernel 内核增加一个具有应用内存映像和寄存器组状态截取、内存地址空间区域识别、内存和寄存器组状态跨平台转换等功能的模块。该模块功能由“迁移点”作为一个系统调用触发，触发后程序本身的运行将被挂起，由内核接管实例 CPU 占有权，执行内存和应用寄存器状态截取等操作。当转换模块准备好迁移的数据后，会将控制权转交给内核通信模块，由通信模块完成剩余的传送过程。此外，对于目标实例，当收到源实例发来的检查点状态或后拷贝数据后，还需要通过该模块将转换后的内存状态即时写入内存中，更新当前进程状态。

跨平台迁移内核通信模块： 该模块用于在异构平台实例之间进行所截取和转换后的内存状态的传输。源实例和目标实例之间建立控制通信通道，协商跨平台迁移过程。在控制通道的管理下，两个实例之间建立一个数据通道，用于检查点文件的传输或后拷贝模式下的内存数据传输。

3.3 基于网络状态感知的自适应进程同步策略

本节设计了一个基于网络状态感知的自适应进程同步策略 NAPS。NAPS 针对边缘环境中移动终端具有移动性和不稳定性的特点，通过每隔一定间隔执行一次跨平台迁移，将边缘服务器上的进程状态同步到本地备用临时服务设备上。本节将详细描述该进程同步策略的同步方案及一些优化方法。

3.3.1 概念定义与问题描述

一个进程在相同平台之间同步时，可以将程序从开始到某一时刻之间执行的指令数量作为一个进程的执行进度。然而由于指令集不同，相同的高级语言语句

① 是一个面向嵌入式系统的 C 语言运行库，官方网站：http://sourceware.org/newlib/。

会编译出数量不同、种类不同的若干指令序列，因此不能简单地用指令执行的数目来表记程序的计算量。本文面向的应用场景是针对异构平台之间相同程序不同平台版本之间的进程同步，于是用一个等效的表达方式来表记一个程序在不同平台上进程的执行进展，即等效计算进度。

定义 3.1 (等效计算进度（Equal Computation Progress）)： 排除设备性能对某进程执行效率以及同一程序异构版本的影响，在给定参数、数据、程序等确定，且程序中不含有随机因素的前提下，一个进程的计算过程中进行的有效计算量的多少，被称为等效计算进度。

根据上述定义，某一进程在效率完全相同的设备上执行的越久，其等效计算进度越大，本文用进程的状态号来代表等效计算进度：状态号越大，表示该进程当前执行的有效计算量更多，等效计算进度越大；反之，状态号越小，表示该进程当前执行的有效计算量更少，等效计算进度越小。

马来亚大学提出了一个固定间隔的进程同步方法[59]，然而这样的固定间隔进程同步并不能很好地适应边缘网络中错综复杂且不稳定的环境。进程状态同步频率问题，其实质上是进程同步开销和当网络意外中断时同步设备需要重新计算的未同步等效计算进度多少的权衡。一个固定间隔的进程同步的时序图如图 3-6所示。

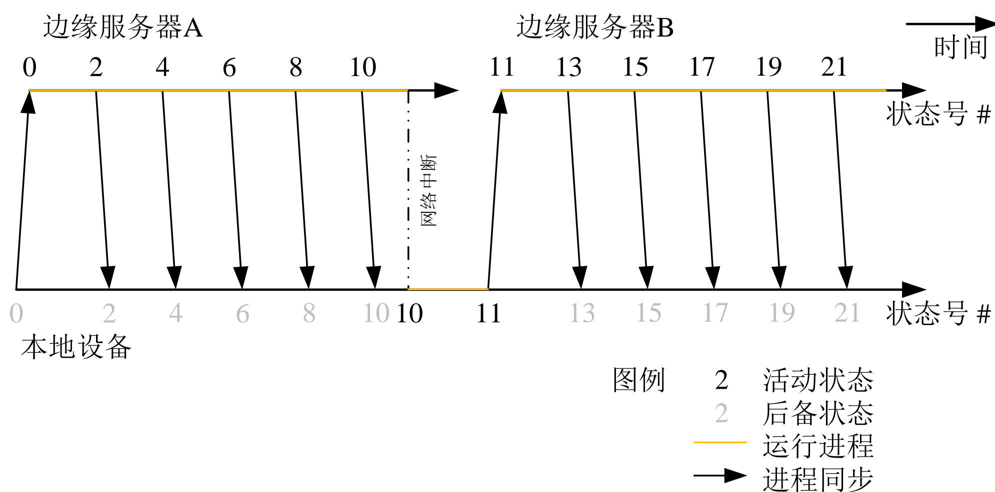


图 3-6　一个固定间隔的进程同步时序图

当终端设备在边缘网络环境内的网络情况较好时，若同步频率过于频繁，则会影响到服务端程序的性能，也会浪费移动终端设备用于网络通信的能源；若同步间隔过大，则当不可预料的网络中断发生时，会导致边缘服务器端仍有较多的最新状态没有同步到临时服务端设备，重复计算则会带来更多的时间和能源开销。

因此，NAPS 策略旨在通过动态对当前网络环境进行评估：当网络环境较好时，通过适当降低同步频率，降低同步带来的额外开销；当网络环境较差时，通过适当提高同步频率，在带宽等因素允许的前提下尽可能将边缘服务器端尽可能新的内存状态同步到本地的临时服务端设备上。

3.3.2 问题建模

本节将对所提出的进程同步问题进行数学建模。为了简化模型，本策略中的评估过程的最小单位为一个固定的采样间隔 I，每经过一个采样间隔根据上个采样间隔内的网络状况，对当前时刻的网络状态进行评估，进程同步总是在一个采样间隔的结束时刻触发。一些相关概念的表记与意义如表 3-1所示。

表 3-1　NAPS 策略中涉及的符号及其含义

符号	描述
I	评估采样间隔（Interval）
B_t	时刻 t 的评估网络带宽（Bandwidth）
L_t	时刻 t 的评估网络延迟（Latency）
S_t	时刻 t 的动态同步间隔
$E_{e,t}$	边缘服务器上当前进程在时刻 t 的等效计算进度
$E_{m,t}$	临时服务设备上当前进程在时刻 t 的等效计算进度
M_t	时刻 t 时需传送内存数据的量

本文采取一种基于网络带宽和网络延迟评估在 t 时刻后下个 I 内网络中断的算法，为进程同步计算动态同步间隔。当前环境中评估采样间隔为 I，则时刻 t 最近 n 次采样的网络带宽最大值和最小值分别为 B_t^{max} 和 B_t^{min}，最近 n 次采样的网络延迟最大值和最小值分别为 L_t^{max} 和 L_t^{min}。

假设 t 时刻网络带宽反映出接下来 S 时间的中断概率为 P_t^B，网络延迟反映出的中断概率为 P_t^L，则这两项概率分别为

$$P_t^B = \frac{B_t - B_t^{min}}{B_t^{max} - B_t^{min}}, \tag{3-1}$$

$$P_t^L = 1 - \frac{L_t - L_t^{min}}{L_t^{max} - L_t^{min}}, \tag{3-2}$$

由于网络带宽和网络带宽均可独立反映部分网络状况，故在接下来 S 时间的中断

概率为

$$P_t = 1 - \left(1 - P_t^B\right)\left(1 - P_t^L\right). \tag{3-3}$$

NAPS 策略的迁移代价包括网络连接正常时的同步数据量以及网络中断后需要重复计算的部分等效计算进度的计算开销。基于上述建模，可以假设出当进程状态同步间隔为 S 时的同步代价函数为

$$V(S) = \frac{M_t}{S} + P_t \times S \, (S > 0). \tag{3-4}$$

由式 3-4可以看出，代价函数 $V(S)$ 能够在 $S = \sqrt{\frac{M_t}{P_t}}$ 时取得最小值 $V = 2\sqrt{M_t P_t}$。

3.4 本章小结

本章首先针对移动边缘网络环境中移动设备的移动性、不稳定性及异构性进行了概述，提出了针对边缘计算环境下“云应用”模式应用的服务端程序在高实时性、高服务连续性和高性能需求的情况下的问题。然后，本章提出了一种异构平台间自适应和轻量级的任务动态迁移机制 ALM-HIP，该机制通过对 Unikernel 技术加以定制，为其内核设计了一个跨平台转换模块和一个迁移通信模块，使迁移过程免除了对虚拟机管理器的依赖，能够更好地适应无宿主机操作系统的嵌入式环境，具有更强的自适应性和泛用性。之后，又针对本章所提出的自适应跨平台任务状态迁移机制给出了一个基于网络状态感知的自适应进程同步策略 NAPS，通过对网络状况和迁移必要性等进行动态评估，按需触发进程同步，最小化进程同步代价和重复计算等效计算进度的综合代价。

第四章 自适应的跨平台任务迁移框架的实现与评估

本章节首先就本文所提出的 ALM-HIP 机制以及 NAPS 策略分别进行了代码实现，然后将两部分结合起来，形成一个完整的跨平台进程迁移框架。然后，本章基于该框架进行实验数据收集与分析，讨论和探究数据反映的系统特点，最后针对实验所反映出的问题进行了讨论，发掘该机制中存在的不足和改进点。

4.1 原型设计与实现

ALM-HIP 的原型系统根据图 3-5，基于 Popcorn Linux 及其工具链，以及开源的 Unikernel 工具链 HermitCore①等实现。ALM-HIP 的实现中所使用的工具链包括生成多平台版本，且支持跨平台迁移的可执行文件所需的定制编译器和链接器，以及 HermitCore Unikernel 内核。

基于 HermitCore 的跨平台支持要求用户为应用程序源代码手工添加“迁移点”，该迁移点的实质是一个系统调用，通过内核功能收集应用进程的内存地址空间中的栈空间和堆空间。

当程序运行到迁移点代码时，会进入到内核迁移模块中执行迁移相关代码逻辑。本系统采用了类似 CRIU 的做法，在用户空间内实现进程的检查点截取/恢复。通过读取内核在执行进程时记录的内存地址空间映射、文件描述符、socket 等信息，可以将一个进程所有的内存数据分别读出。同时，系统将 HEXO 所提供的跨平台信息转换功能模块移植到内核的迁移模块中，将截取到的进程数据转换后送至通信模块。

通信模块负责与迁移目标实例建立数据通信，将转换后的进程内存数据发送到目标实例上。目标实例收到转换后的进程状态数据后，按照不同内存区域分别将收到的内存映像恢复到当前实例的对应区域内存地址上。

4.2 实验测试与分析

4.2.1 实验设置

测试框架：基于上节所实验的原型系统，本节采用 KVM 虚拟机—嵌入式平台环境，基于数值空气动力学模拟（Numerical Aerodynamic Simulation，NAS）并行

① https://hermitcore.org/

基准测试[60]（NAS Pararell Benchmark，NPB）进行实验测试，分别收集实验数据并分析。

NAS 是美国国家航天局的一个数值空气动力学模拟项目，该项目为了客观地比较并行超级计算机的性能而设计了一组基准测试程序，称为 NAS 并行基准测试 NPB。NPB 的测试代码选取自计算流体动力学代码，具有代表意义，受到了广泛的认可。本实验采用 NPB 3.3.1 版本，含有包括 5 个内核测试和 3 个伪应用测试，本章选取其中 4 个基准测试程序进行实验，选取的基准测试如表 4-1所示。其中，NPB-IS 为原生 C 语言编写，其余测试程序采用首尔大学基于 NAS 的 Fortran 语言版本移植的 C 语言版本，各程序处理数据规模均选用 NPB 规定的 B 等规模配置（A 等为最小，C 等为最大）。

表 4-1　实验所使用的测试程序

程序名称	程序简称	编写语言	问题规模	程序说明
大整数排序	NPB-IS	C	2^{25}	基于桶排序的大整数排序
繁杂并行	NPB-EP	C	2^{30}	计算高斯伪随机数
共轭梯度法	NPB-CG	C	75000	求解大型稀疏对称正定矩阵的最小特征值的近似值
五对角线方程组	NPB-SP	C	102^3	求解五对角线方程组

硬件配置：模拟实验中采用一台配备有 Intel(R) Core(TM) i7-9400F 的 x86-64 架构计算机作为边缘服务器，采用一个树莓派 4B 作为临时服务端设备。首先通过一个使用 NPB-IS 性能测试，粗略估计两个设备的性能差异。实验平台的配置如表4-2所示。

表 4-2　实验硬件配置表

设备	台式计算机	树莓派 4B
CPU 型号	Intel(R) Core(TM) i7-9400F	ARM Cortex-A72
CPU 架构	x86-64	AArch64
CPU 主频	2.90 GHz	1.50 GHz
CPU 核数	6	4
内存大小	16 GB	4 GB
操作系统	Debian 9.3.0	Ubuntu 18.04 Server
NPB-IS 耗时	2.17 s	24.25 s
计算效率/$Mop \cdot s^{-1}$	154.67	13.84

由表4-2可以看出：作为边缘服务器的台式计算机针对计算密集型应用的计算效率约为作为临时服务端设备的树莓派 4B 的约 11.17 倍，可以体现出二者在执行相同应用时的性能差异。

实验架构：真机实验的实验设置架构图如图4-1所示。其中，计算性能较强的台式计算机用于作为边缘服务器端，用来模拟将计算负载卸载到边缘服务器上以获取更高计算性能的情况。而性能相对较弱的树莓派 4B 则作为临时服务设备，在服务端程序运行过程中每隔一定间隔从边缘服务器上同步最新内存状态到本地，用于在终端设备与边缘服务器连接中断时临时启动，为终端设备继续提供服务。

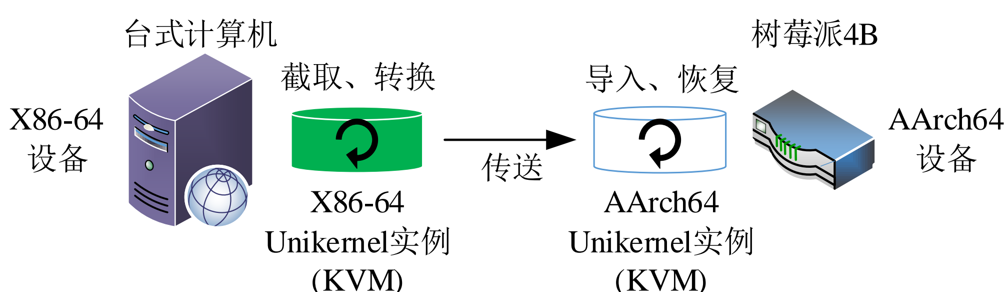


图 4-1　实验设置图

本文还设置了一个模拟实验，通过编程模拟该进程同步过程，用以单独测试 NAPS 策略，验证不同策略参数对策略本身性能的影响。和集成到原型系统中的实验不同，该模拟实验仅针对 NAPS 策略本身进行模拟。模拟实验将生成一个动态的网络环境参数模拟真实场景下网络环境的变化情况，其中边缘服务器和临时服务设备之间的带宽和延迟随时间变化而变化，观察同步策略的表现和运行情况。此外，本模拟实验还通过设定不同的评估采样间隔，观察评估采样间隔对 NAPS 策略造成的影响。

对比策略：模拟实验模拟了几种具有代表性网络变化情形下，针对不同评估采样间隔、不同迁移数据量大小下 NAPS 策略的表现情况，同时还与不同固定间隔的同步策略进行了对比。

性能指标：本文所提出功能的性能评价指标分为 ALM-HIP 机制和 NAPS 策略两部分。对于 ALM-HIP 机制而言，这部分工作的主要目标是去除跨平台迁移过程中虚拟机管理器的必要性，因此在性能指标方面，主要测试以下几个方面：

1. ALM-HIP 机制的有效性：即是否达到了由内核独立完成跨平台迁移相关功能；
2. 迁移数据量：即一次跨平台迁移过程中传送数据的量；

3. 迁移计算开销：进程从触发跨平台迁移到迁移结束中的额外计算开销；
4. 服务中断时间：跨平台迁移过程中服务中断时间长度。

对于 NAPS 策略而言，主要评估其算法性能，包括：

1. 同步通信开销：进程同步导致的额外数据通信量；
2. 同步计算开销：进程同步导致的额外计算开销；
3. 重复计算开销：当切换运行设备时，由于最新状态尚未同步到目标设备，从而造成目标设备需要重新计算的等效计算进度的量。

4.2.2　实验结果与分析

（1）异构平台间自适应和轻量级的任务动态迁移

在 ALM-HIP 机制的实验方面，选择采用若干计算密集型，且具有较强代表性的 NPB 基准测试集应用进行实验评估，这些程序及其意义如表4-1所示。本组实验的结果如图4-2所示，其中横坐标为不同的应用程序和应用不同的迁移机制，纵坐标为相应性能测试项目的数据。

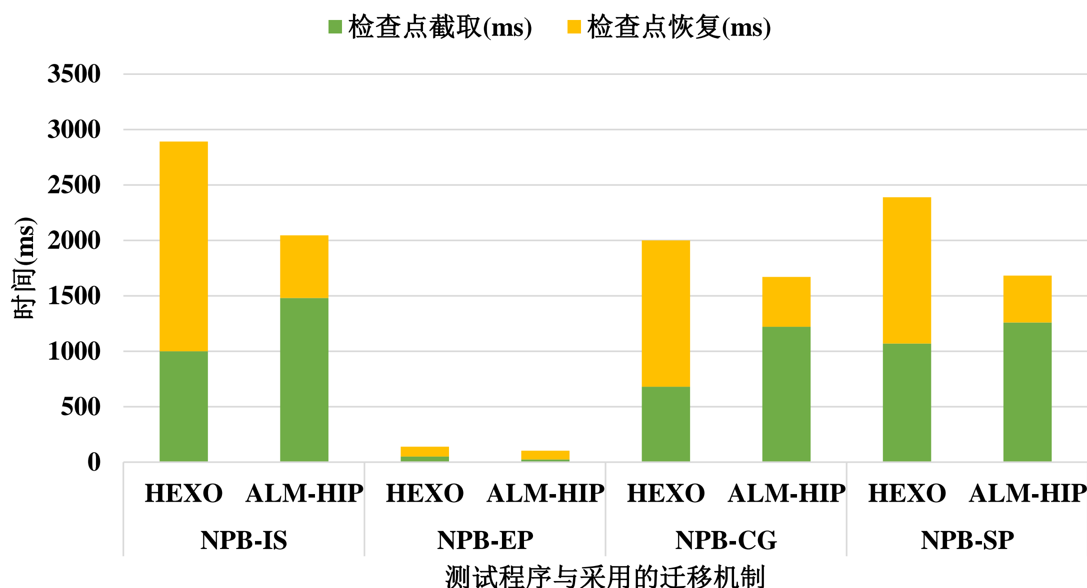


图 4-2　自适应迁移机制实验结果

本实验主要从进程状态收集转换和状态恢复耗时两个方面将本文所提出的自适应迁移机制与 HEXO 进行比较。从实验结果可以看出，ALM-HIP 的总体性能比 HEXO 较好，但分别从检查点截取和检查点恢复两个方面来看，则两个机制各有所长。ALM-HIP 用于检查点截取阶段的时间开销高于 HEXO，经分析这是由于自适应迁移机制在截取内存信息方面没有虚拟机管理器的支持，需要占用 Unikernel 实例时间在实例内进行相关信息收集操作，导致性能较 HEXO 略差。而从检查点恢复

的过程来看，由于在此阶段无需对内存本身的结构进行分析，可以直接根据截取检查点时记录的内存信息将相应内存信息载入并恢复进程的继续运行，故 ALM-HIP 的时间开销比 HEXO 更低。

就泛用性而言，由于 ALM-HIP 机制能够独立地完成进程的跨平台迁移，实现了等价于一个跨平台的用户空间内检查点截取/恢复（CRIU）技术，具有较多 CRIU 技术的特性。ALM-HIP 能够极大地降低 Unikernel 应用跨平台迁移的应用门槛，拓宽了跨平台迁移技术的适用场景，具有较大的工作价值。

（2）基于网络状态感知的自适应进程同步策略

针对 NAPS 策略，本节设计了若干网络状况的场景，并为这些场景分别选用不同的若干组参数进行实验测试。具体而言，本节设计的网络状况包括以下几种：

1. 网络情况较好：网络带宽和网络延迟比较稳定，会随机出现一些网络性能的波动，但不会出现网络中断的情况；
2. 网络情况一般：网络带宽和网络延迟有一定随机波动，有网络中断的情况出现；
3. 网络情况较差：网络带宽和网络延迟有较大波动，多次出现网络中断。

为以上网络环境分别生成一组网络带宽和网络延迟随时间的变化关系，如图4-3所示。其中，实线代表当前时刻的网络带宽，虚线代表当前时刻的网络延迟，横坐标为时间。

针对上述三种网络状况，分别取 $I = 4\,\text{min}$、$I = 8\,\text{min}$ 和 $I = 16\,\text{min}$，以及取固定间隔进程同步策略对照组（分别取固定间隔 $I^{fix} = 4\,\text{min}$、$I^{fix} = 8\,\text{min}$ 和 $I^{fix} = 16\,\text{min}$）得到模拟结果如表 4-3所示。

评估采样间隔的影响分析：当评估采样间隔设定值较小时，系统将频繁地对端-边的网络状态进行评估。较小采样间隔的好处是能够尽快对当前网络的变化做出评估，能够反映当前网络最新的状态，避免网络出现较大变化而系统的评估还仍基于一段时间前的状态造成评估不准确。然而，采样间隔较小同样会对系统产生一定的影响，如网络测速将会消耗一定的信号带宽，同样也会使系统的总体性能下降。

网络状况差异的影响分析：当网络状态整体较好，带宽较高且延迟相对较低时，由于网络中几乎不存在网络中断的情况，在这样的网络环境下启用临时服务设备的概率较低，无需使临时服务设备中暂存计算状态的等效计算进度紧跟边缘服务器上服务端进程的等效计算进度。此时适当增加进程同步的间隔，有利于进一步减少进程同步的网络开销、性能开销，自然地减少了移动终端设备的电量消耗，有利于提升移动设备的续航能力。与固定间隔的同步策略相比，NAPS 策略可

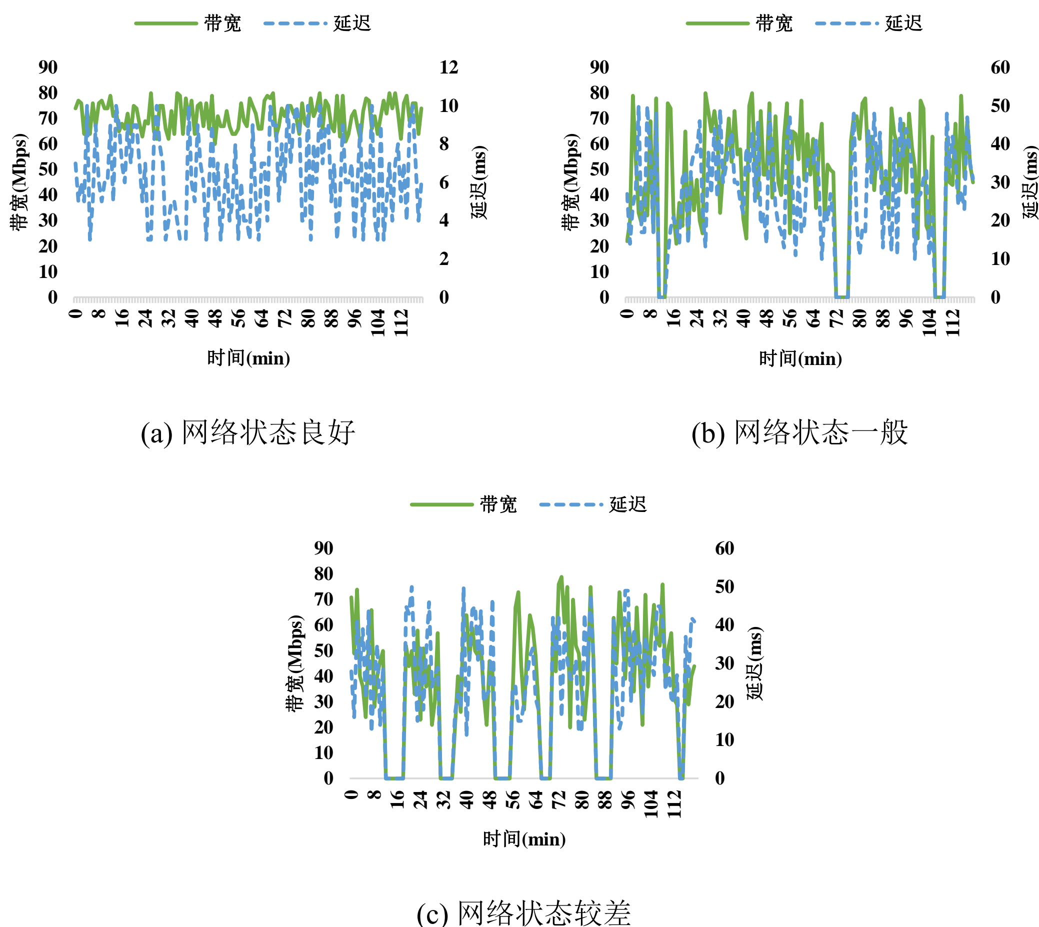


图 4-3　几种模拟的网络环境

以有效避免固定间隔同步带来的额外同步开销，同时可以防止因固定间隔过大导致来不及对网络状态的突变做出响应。

当网络状态较差时，由于网络中断非常频繁，如果进程同步的间隔过大，将导致边缘服务器上相应进程最新的内存状态无法及时同步到临时服务设备上，带来较多的等效计算进度损失。若采用固定间隔的同步策略，则为了应对此类网络环境，需要非常低的同步间隔，导致进程同步占用了本就较差的网络环境的信号资源，反客为主，得不偿失。与固定间隔的同步策略相比，当评估间隔值更小时，NAPS 策略能够更好地适应较差的网络环境。

当网络环境一般，偶尔出现中断情况时，此时根据网络状态的变化无法非常准确地估计出网络中断的发生。若网络中断出现在一次较长的动态间隔时间内时，将造成较大的等效计算进度的浪费，而此时若选取较低的评估间隔则又会使同步的开销增大。从实验结果来看，此时 NAPS 策略的性能优于固定间隔策略，这是因为固定间隔策略仍然需要较小的固定间隔才能适应偶发的网络中断。

综上所述，本文所提出的 NAPS 策略可以自然地适应大多数的网络环境，而

表 4-3　NAPS 策略在不同网络状况不同评估采样间隔下的表现

网络状态	同步策略	同步次数	重复计算开销
较好	感知（$I = 4\,\text{min}$）	17	0
较好	感知（$I = 8\,\text{min}$）	16	0
较好	感知（$I = 16\,\text{min}$）	14	0
较好	固定（$I^{fix} = 4\,\text{min}$）	29	0
较好	固定（$I^{fix} = 8\,\text{min}$）	14	0
较好	固定（$I^{fix} = 16\,\text{min}$）	7	0
一般	感知（$I = 4\,\text{min}$）	14	90
一般	感知（$I = 8\,\text{min}$）	14	50
一般	感知（$I = 16\,\text{min}$）	16	50
一般	固定（$I^{fix} = 4\,\text{min}$）	24	50
一般	固定（$I^{fix} = 8\,\text{min}$）	11	90
一般	固定（$I^{fix} = 16\,\text{min}$）	3	330
较差	感知（$I = 4\,\text{min}$）	8	230
较差	感知（$I = 8\,\text{min}$）	7	270
较差	感知（$I = 16\,\text{min}$）	7	250
较差	固定（$I^{fix} = 4\,\text{min}$）	16	120
较差	固定（$I^{fix} = 8\,\text{min}$）	6	280
较差	固定（$I^{fix} = 16\,\text{min}$）	1	600

无需人工干预为其设置固定的同步间隔。和固定间隔策略相比，NAPS 策略更能适应不稳定、不确定的网络环境，而固定间隔的同步策略则在网络状态较好或网络状态较差的环境下，可通过人为规定同步间隔达到最小化同步的综合开销。

4.3　本章小结

本章针对本文所提出的 ALM-HIP 机制实现了一个系统框架，详细介绍了该原型系统的实现原理和部分实现细节。其次，本章基于该原型系统对 ALM-HIP 机制进行了相关性能评估，验证了该机制的有效性，分析了其性能开销的来源，以及可能的优化方法。本章还针对 NAPS 策略进行了在多种复杂网络环境下的运行性能评估，量化了各项参数对整体同步额外开销的影响多少，在最小化同步带来的额外开销的同时最小化因意外网络中断造成的重复计算开销。

第五章 边缘计算中多节点弹性任务卸载调度原型系统

本章围绕计算任务复杂度高且时延敏感环境的实际情况，研究多节点的任务调度技术。根据卸载任务的具体资源和计算需求，结合边缘环境中各节点的实时状态信息进行实时调度，使得计算任务能够分散到边缘网络集群节点上，缩短任务的平均响应时间。具体地，本章将研究计算任务复杂度高且时延敏感环境下面向计算卸载的多节点弹性任务调度模型和功能框架。此外，针对边缘计算环境中各个服务节点的动态性，研究任务调度中的错误处理和容错机制，避免和最小化任务重调度和重新执行带来的额外开销。

本章设计和实现了一个适用于弹性任务计算场景的任务卸载调度原型系统，能够将一系列用户提交的计算任务卸载到一系列计算节点上，使得各个任务的总响应时间最短。此外，本章将前述的 ALM-HIP 机制与 NAPS 策略整合到该卸载调度原型系统中，为系统增加了跨平台迁移支持与进程同步支持。

5.1　任务卸载调度框架与策略

5.1.1　任务卸载调度总体框架

本节将围绕计算任务复杂度高且时延敏感环境的实际需求，研究如何高效地将待执行的任务集合调度到集群中各个分散节点上，使得任务集合能够并行执行，加速任务响应时间，提高集群资源利用率。该任务卸载调度的框架如图 5-1所示。

在该框架中有一个作为控制器角色的节点，该节点维护一个任务请求队列，并负责将用户所提交的任务集合根据最小化任务平均完成时间的任务分配策略（MBCT）进行调度，为每个任务分配一个合适的运行节点，并为每个计算节点分配适当的任务执行顺序，缩短任务的平均响应时间。本原型系统基于 Kubernetes[①]实现，详见本章“原型实现与评估”节。

5.1.2　面向多目标优化的任务调度策略

任务调度是系统的核心环节之一，它根据任务集合、集群内节点集合和使用的调度策略产生任务调度决策序列。任务调度策略需要合理选择优化目标作为调度的前提，由于在当前环境下计算任务复杂度高且时延敏感，这就需要任务的平均响应时间尽可能小。此外，由于集群内单个节点资源有限，如何充分利用每个

① 一个开源的、适用于管理云平台中多个主机上的容器化的应用。官方网站：https://kubernetes.io/。

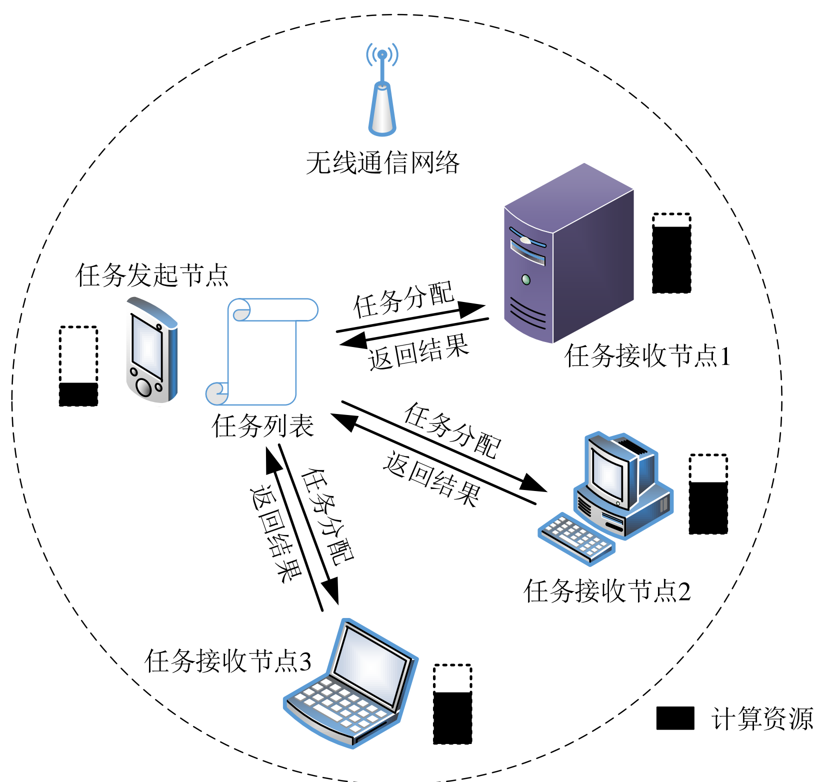


图 5-1　任务卸载调度框架

节点有限的计算、存储和网络带宽资源，做好资源管理，也是该任务调度策略需要考虑的问题之一。

（1）内存资源满足优先策略

在调度的过程中，CPU 和内存资源的管理与分配是一个重要的问题，它直接影响到整个集群的资源利用率。由于在本平台以容器形式执行和管理任务，在任务提交时，需要为每个任务制定 CPU 和内存需求，其中当可分配内存小于任务内存需求时，可能会导致任务执行失败。此时，本系统将使用内存资源满足优先策略，当本地资源满足时，优先在本地执行；否则对集群内节点按照可用内存剩余量从高到低排序，按任务队列依次分配任务。

然而，内存资源满足优先策略并未考虑任务的执行时间，仅凭该策略虽然在计算卸载时使用内存资源满足优先策略能够提高系统的资源利用率，但无法缩短任务的平均响应时间。因此内存资源满足优先策略仅适用于尚未在系统中执行过的任务使用，对于已经在系统中执行过至少一次的任务，将使用基于遗传算法的最小任务响应时间策略对其进行调度。

（2）基于遗传算法的最小任务响应时间策略

当一个任务在系统中被执行过后，系统将统计该任务的实际执行时间，计算

该任务最近若干次在系统上执行的实际执行时间的平均值，作为该任务的预估执行时间。下面介绍基于遗传算法的最小任务响应时间策略，该策略是整个调度系统的核心策略，适用于在有预估时间的前提下，为一个含有多个任务的任务集合中的各个任务调度到一个具有多个节点的集群中的具体节点上。

假设任务集为 $I = \left\{i_1, i_2, ..., i_n\right\}\left(n \in N^*\right)$，集群中的计算节点为 $J = \left\{j_0, j_1, ..., i_m\right\}(m \in N)$，其中 j_0 表示本地节点，任务 i 在节点 j 的总响应时间可以表示为

$$T_{i,j} = t_{i,j}^{exec} + t_{i,j}^{wait}, \tag{5-1}$$

其中，$t_{i,j}^{exec}$ 表示任务 i 在节点 j 的执行时间；$t_{i,j}^{wait}$ 为任务 i 在节点 j 的等待时间，其计算方式为

$$t_{i,j}^{wait} = q_j \times T_j^{avg}, \tag{5-2}$$

其中 q_j 和 T_j^{avg} 分别为节点 j 上等待队列的长度和该节点上任务的平均执行时间。值得注意的是，式 5-1中没有考虑传输任务本身容器的时间和网络延迟，因为这两项时间在本章介绍的边缘计算系统中远小于任务的执行时间和队列等待时间。

对于上述集合 I 和 J，记 $x_{i,j}$ 为一次分配决策：$x_{i,j} = 1$ 表示将任务 i 分配到节点 j 上，$x_{i,j} = 0$ 表示将任务 i 和节点 j 无分配关系。同时，$x_{i,j}$ 还满足

$$\sum_{j=0}^{m} x_{i,j} = 1\,(1 \leqslant i \leqslant n, 0 \leqslant j \leqslant m), \tag{5-3}$$

即每个任务分配且仅分配到一个节点上执行一次。

本章所实现的卸载调度系统的目标为使所有任务的总响应时间最短，即为每个任务 i 求解一个合适的节点 j，使

$$T = \sum \left(x_{i,j} \times T_{i,j}\right) \tag{5-4}$$

最小。

然而上述问题是 NP-难的，于是该系统使用了遗传算法为其求得一个近似最优解。一组分配决策 $X_{a,b} = \left\{x_{1,0}, ..., x_{1,m}, x_{2,0}, ..., x_{n,m}\right\}$ 为进化过程中一个世代中的一个个体，个体中的每个分配关系为一条染色体。将一个个体的染色体集合记为 $S = \left\{s_i = J_j, 1 \leqslant i \leqslant n, 0 \leqslant j \leqslant m\right\}$，即将任务 I_i 分配到节点 J_j 上。

对于每个个体，其约束条件函数和适应性函数分别定义为：

$$C(X) = \sum \left(x_{i,j} \times T_{i,j}\right) + \sum_{j=0}^{m} E_j^{cpu} + \sum_{j=0}^{m} E_j^{mem}, \tag{5-5}$$

$$fitness(X) = \frac{1}{C(X)}. \tag{5-6}$$

其中 E_j^{cpu} 和 E_j^{mem} 分别为节点 j 关于 CPU 资源约束条件和内存资源约束条件的惩罚项。当一个节点上被分配的任务量的资源需求量超过节点的对应资源量时，将其计算方法如式 5-7和式 5-8所示，用于避免调度策略将过多任务聚集在同一个节点上，因此当一个节点被分配的任务的资源需求量超出越多时，其任务的惩罚就越高。

$$E_j^{cpu} = \frac{\sum_{x=1}^{n}\left(x_{i,j} \times I_i^{cpu}\right) - J_j^{cpu}}{\frac{\sum_{x=1}^{n}\left(x_{i,j} \times I_i^{cpu}\right)}{\sum_{x=1}^{n} x_{i,j}}} \times T_j^{avg}. \tag{5-7}$$

$$E_j^{mem} = \frac{\sum_{x=1}^{n}\left(x_{i,j} \times I_i^{mem}\right) - J_j^{mem}}{\frac{\sum_{x=1}^{n}\left(x_{i,j} \times I_i^{mem}\right)}{\sum_{x=1}^{n} x_{i,j}}} \times T_j^{avg}. \tag{5-8}$$

其中，I_i^{cpu} 和 I_i^{mem} 分别表示任务 I_i 需求的 CPU 和内存资源量，J_j^{cpu} 和 J_j^{mem} 分别表示节点 J_j 拥有的 CPU 和内存资源量。

遗传策略上，本系统使用轮盘赌选择法，即每个个体进入下一代的概率与其适应性相等。每一代有 30% 的个体将进入下一轮迭代，交叉模式为单点交叉，突变率为 30%，迭代轮数为 500 轮。

5.2 原型实现与实验评估

5.2.1 原型系统实现

根据上节所述的实现途径，基于 Kubernetes 平台实现了一个卸载调度原型系统，该系统的架构如图 5-2所示。

该原型系统中包括一个任务发起节点和若干任务接收节点，任务发起节点即需要卸载调度服务的设备，即本地设备；任务接收节点则是用于卸载任务进行计算的节点，接收到的任务在执行完毕后，将结果传回任务发起节点。用户通过作为任务发起节点的本地设备提交任务集合，经控制器应用上述策略调度后将任务卸载到任务接收节点。

任务发起节点分为 Web 服务界面和控制器两部分。其中，Web 服务界面用于用户交互，用户通过 Web 服务界面管理任务集合、管理集群内节点列表和查看任务执行结果。Web 服务部分包括任务列表、提交任务和集群管理等三个页面。控制器部分则在用户提交任务后，通过上节所述任务卸载调度算法，为每个任务选择合适的节点并发起任务卸载。

（1）任务管理

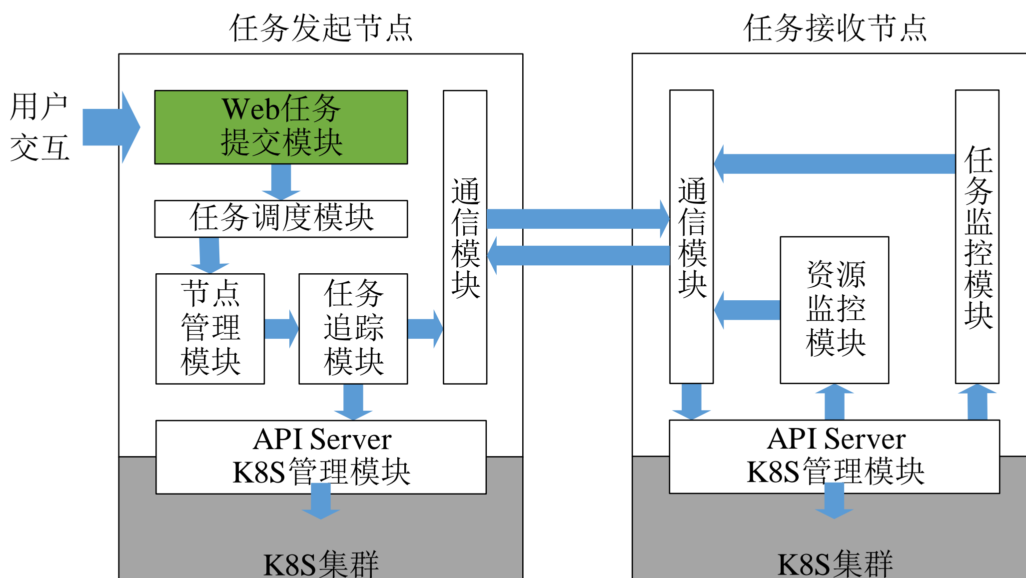


图 5-2　任务卸载调度原型系统组成架构

任务管理部分分为任务列表和任务提交两个页面。任务列表页面如图 5-3所示，其功能为展示已提交任务的实时状态信息。如图所示，任务具有“等待中”“运行中”和“已完成”等三个可能的状态。其中“等待中”包括两个可能，一是任务在等待队列，仍未分配至合适节点。二是由于节点的可用资源不满足需求，需要等待其他任务执行完成让出资源。“运行中”表示该任务正在运行，“已完成”表示任务已正常执行完毕。当任务执行完毕后，系统会显示其从提交至执行完成之间的完成时间。

任务管理
任务列表
新增任务
集群管理

任务列表　　删除所有任务

任务ID	任务名称	运行集群	IP地址	卸载调度	GPU需求	运行状态	完成时间	查看日志	操作
0	task012	Local	127.0.0.1	否	否	已完成	14.41s	查看	
1	task006	Local	127.0.0.1	否	否	已完成	15.47s	查看	
2	task004	Local	127.0.0.1	否	否	运行中	-	查看	
3	task009	Local	127.0.0.1	否	否	运行中	-	查看	
4	task014	Local	127.0.0.1	否	否	运行中	-	查看	
5	task013	Local	127.0.0.1	否	否	运行中	-	查看	
6	task007	Local	127.0.0.1	否	否	运行中	-	查看	
7	task008	Local	127.0.0.1	否	否	运行中	-	查看	
8	task010	Local	127.0.0.1	否	否	运行中	-	查看	
9	task016	Local	127.0.0.1	否	否	运行中	-	查看	
10	task001	Local	127.0.0.1	否	否	运行中	-	查看	

图 5-3　任务列表页面

任务提交页面如图 5-4所示。新增任务时，用户需提供任务名、任务的 CPU 资源需求量、内存资源需求量和容器镜像地址等信息。其中任务名不能重复，否则需要将原任务删除。在新增任务时，用户为任务指定本地运行，或将该任务交由控制器分配至合适的服务节点。新增任务页面支持批量新增任务，用户可将待提交任务写入一个 Json 格式文件批量提交。

图 5-4　任务提交页面

（2）集群管理

集群管理页面如图 5-5所示。集群管理界面主要有两个功能，一个是显示当前已连接服务节点的信息，如 IP 地址和可用 CPU、内存资源等；另一个功能是增加服务节点，用户可根据服务节点的 IP 地址和 TCP 端口添加新集群。该界面也支持通过 Json 文件批量添加新集群。

（3）异构平台的进程迁移和同步支持

为了使该原型系统具备跨平台任务迁移能力，本工作还将提出的 ALM-HIP 机制与 NAPS 策略进一步整合到该系统中。具体而言，ALM-HIP 机制主要是作为跨平台的技术实现，将这部分功能整合到任务发起节点的任务追踪模块和任务接收节点的任务监控模块。当有跨平台迁移需求时，可通过调用 ALM-HIP 的相关功能，将正在当前节点上执行的任务截取为检查点文件，然后传送至目标节点，由目标节点的相应模块负责将检查点文件恢复到系统中，完成任务的跨平台迁移。

此外，系统中所整合的 NAPS 策略也可按需启用，为原型系统中需要防止由节点退出导致重复计算的高鲁棒性需求任务提供进程同步服务。

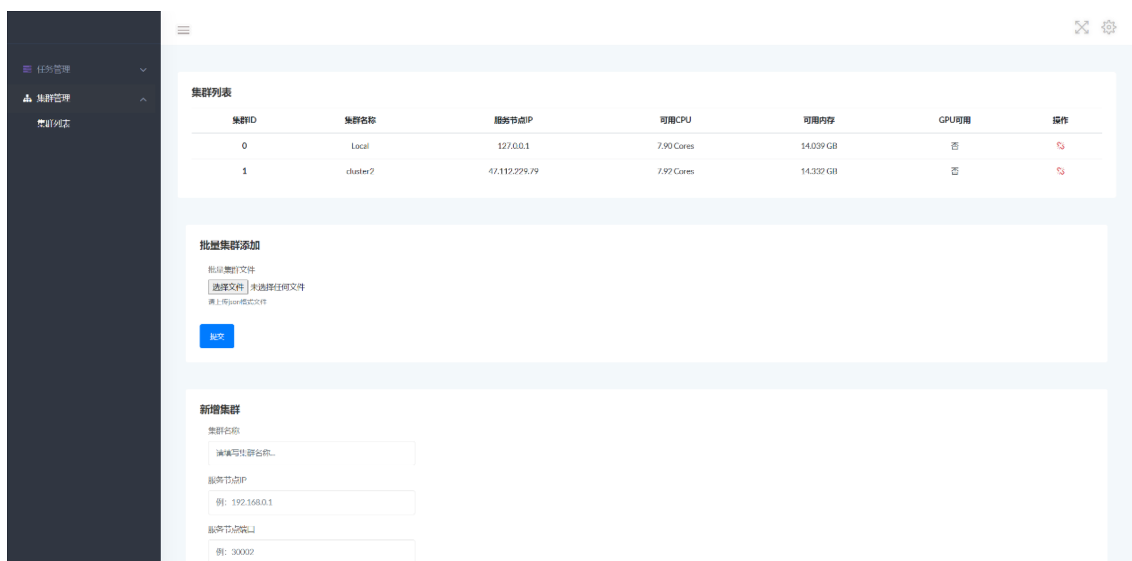


图 5-5　集群管理页面

5.2.2　实验设置

为了测试该原型系统的功能有效性以及弹性任务卸载调度策略的性能，本文选取了若干具有代表性的测试集任务用于测试，这些任务均为计算密集型任务，表 5-1展示了这些任务的名称及其现实意义。任务集合由表中随机抽选得到，任务集大小分别为 100、500 和 1000，针对每个任务集进行三次实验。考虑到每种任务在真实场景下并不是等量的，故在抽取时为每种任务均设置有选中概率。

表 5-1　测试集任务程序

程序名称	简称	语言	抽取概率	内存需求 (GB)	CPU 需求 (核)
迷宫寻路	maze	C++	0.02	0.4	0.5
SHA256 计算	sha256	C	0.30	0.4	1.0
N 皇后问题	13queen	C++	0.10	0.4	0.5
快速排序	quicksort	C++	0.14	0.4	0.5
文件验证	hashzip	C	0.30	0.4	1.0
动态规划	dynamic	C	0.04	0.4	0.5
哈夫曼编码	huffman	C++	0.06	0.4	0.5
计算圆周率	calcpi	C	0.02	0.4	0.5
矩阵运算	matrix	C	0.02	0.4	0.5

为了对比卸载调度算法的有效性，首先提交任务让其仅在本地节点运行，待运行结束后记录最后一个执行完毕任务从提交其到执行完毕的时刻之间的时间差，

作为本地运行的比较基准。由于每个任务都在本地节点被运行过一次，在本地节点运行后系统将学习到每个任务种类的预估执行时间数据，随后通过卸载调度方式再次提交该任务集，但不同的是本次提交允许任务根据调度策略的决策结果卸载到各个节点上运行。实验所使用的集群配置如表 5-2所示。

表 5-2　卸载调度原型系统测试集群配置

节点名称	节点总核数[a]	节点总内存	网络带宽
发起节点	4	8GB	1500Mbps
服务节点 1	8	16GB	1500Mbps
服务节点 2	8	16GB	1500Mbps
服务节点 3	4	8GB	1500Mbps

[a] CPU 型号均为 Intel® Xeon® Platinum 8269CY

5.2.3 实验结果与分析

按照上述方法进行实验测试，各次实验的任务集大小、本地运行总响应时间、调度卸载总响应时间以及加速比等数据如表 5-3所示。其中，加速比 A 的计算方法为

$$A = \frac{T^{local} - T^{off}}{T^{local}} \times 100\%, \tag{5-9}$$

T^{local} 表示本地运行总响应时间，即仅在本地节点运行任务集合时从任务集合提交到所有任务均执行完毕所经过的时间；T^{off} 表示调度卸载总响应时间，即以卸载调度方式运行任务集合时从任务集合提交到所有任务均执行完毕所经过的时间。

由表 5-3可以看出：对于大多数实验组，卸载调度方式的运行加速比例大约在 70% 至 80% 之间。根据表 5-2的设备硬件配置，本地运行的 CPU 核数为 4，卸载调度运行时所有节点总共的 CPU 核数为 24，因此理想情况下总响应时间的理论值为 83.33%。经分析，有以下原因导致加速比例未能达到预期值：

1. 本地运行时，由于 kubernetes 的 API Server 在添加任务时也会占用一定系统资源，已经添加的任务正在全速执行，这导致 K8S 具有的资源受限，API Server 和任务抢占系统资源，导致任务实际运行耗时估计与真实值有所偏差，预学习的任务执行时间不准确；
2. 调度策略对于异构多核的适配性不够好，当 CPU 核数不足时，调度策略向 CPU 核数较多的节点分配任务的倾向不够。

原型系统的设计还应针对上述可能的原因做进一步优化，如为 API Server 预留一定的资源，防止任务和 K8S 平台抢占资源，导致系统性能下降。另外，还应

表 5-3　调度卸载原型系统实验结果

任务集编号	任务集大小	T^{local}(s)	T^{off}(s)	A
1-1	100	687.69	150.77	78.08%
1-2	100	586.17	133.77	77.18%
1-3	100	675.46	185.19	72.58%
2-1	500	3005.07	919.13	69.41%
2-2	500	3992.50	826.33	79.30%
2-3	500	3390.49	855.06	74.78%
3-1	1000	7258.78	1666.98	77.03%
3-2	1000	7020.05	1824.68	74.01%
3-3	1000	6792.21	1526.02	77.53%
平均加速比				75.55%

针对核数不同的节点时任务调度的倾向性进行优化，防止有部分任务集聚在核数较少的节点上，导致多核数节点运行完毕，但仍有部分节点仍有大量任务在等待执行。

5.3　本章小结

本章根据面向多目标优化的任务调度中的内存资源满足优先策略和基于遗传算法的最小任务响应时间策略等两个调度策略，设计和实现了一个调度卸载原型系统。首先介绍了该原型系统的应用背景和技术途径，然后对该系统进行了代码实现和性能评估，验证了调度卸载系统的有效性。此外，本章还将设计的 ALM-HIP 机制和 NAPS 策略进一步整合到该卸载调度系统中，使该系统在基本卸载调度功能的基础上，具备了跨平台的任务迁移和进程同步能力，从而可以较为全面地支持移动边缘场景下异构节点之间的任务卸载与迁移。

第六章 总结与展望

本文针对面向边缘环境中的动态迁移问题和跨平台问题进行了概述和总结，阐述了跨平台动态迁移的研究背景和相关工作。本文提出了一种异构平台间自适应和轻量级的任务动态迁移机制 ALM-HIP，并设计了一个基于网络感知的自适应同步策略 NAPS，分别适用于边缘计算中设备异构的环境中的任务动态迁移和同步。

6.1 工作总结

目前，得益于网络技术和移动通信技术的快速发展，全球正进入一个万物互联的时代。在这样的大环境中，随着联网设备数量的急剧增加，网络中产生的数据也呈现爆发式增长，传统云计算的局限性也逐步展现。对于网络边缘所产生的海量数据，如果将这些数据全部不加处理发送至云端，将为云端产生极大的网络压力和数据处理压力。此外，由于网络边缘的设备通常和云数据中心距离较远，因此对于一些如 AR/VR、即时游戏、无人驾驶等延迟敏感型应用，其服务质量将大打折扣。在网络的边缘，通常分布着许多具有一定计算和存储能力设备，边缘计算这一计算范式旨在充分利用这些位于网络边缘设备的计算、存储和网络资源，为移动用户或物联网设备提供低延迟、高性能的服务。边缘计算通过将计算放置于网络边缘进行处理，能够极大地减轻云服务中心的计算负载以及骨干网络的网络流量，提高网络的整体效率。边缘计算中包含多项研究领域，但其研究的根本问题是如何做好边-端-云三者协同，更充分地利用边缘网络中的各项资源。边缘网络中通常含有大量异构平台的设备，设备的异构性给任务的迁移带来了挑战，如何打破设备异构性的隔阂，允许任务在异构设备之间迁移，是本文研究的主要问题。本文主要针对边缘计算中的跨平台迁移问题进行了深入研究，同时基于提出的跨平台任务迁移机制，设计了一个任务进程同步策略。本文的主要工作可总结为以下几点：

（1）本文研究了边缘计算中任务迁移的相关背景，对异构平台之间的任务迁移技术进行了深入研究。首先概述了边缘计算中常用的虚拟化技术，比较各项虚拟化技术的运行原理、系统特点和适用场景等，对比主流虚拟化技术之间的相同点和不同点，总结了这些虚拟化技术的特性。然后列举并根据迁移任务负载种类不同，分类分析了一些典型的跨平台工作，总结这些典型工作的实现原理和应用

场景，最后总结了跨平台迁移的相关挑战。通过总结和比较各项典型工作，提出了一个面向特定应用场景的技术路线选择方案，可协助用户和开发者选择相应或相似的解决方案。

（2）本文针对边缘网络中的设备异构问题，设计了一种异构平台间自适应和轻量级的任务动态迁移机制 ALM-HIP。由于目前的跨平台迁移机制中，在迁移任务时大多需要让虚拟运行环境与宿主环境交互，这就需要为宿主环境安装一个操作系统，占用了不必要的存储资源和计算资源。为了应用的隔离性和独占性，ALM-HIP 机制基于 Unikernel 技术实现，Unikernel 可以直接作为一个完整的操作系统级应用，直接运行于硬件上。当 Unikernel 实例直接运行于硬件上时，将失去宿主环境中的虚拟化环境支持。本文通过为 Unikernel 内核设计两个内核模块用于跨平台转换和迁移通信，使 Unikernel 实例能够独立完成跨平台迁移过程，因此该机制不仅仅适用于虚拟化环境，也充分利用了 Unikernel 支持直接在硬件环境上运行的重要特性。本文还设计了一个基于网络状态感知的自适应进程同步策略 NAPS。首先为边缘计算场景下进程的同步问题建立模型，然后构建了一个以网络评估为基础的进程同步策略 NAPS。在进程运行过程中，移动终端设备周期性评估自身与边缘服务器的网络状况，包括网络带宽和网络延迟等，通过测量到的网络状况对设备与边缘服务器的连接中断概率进行评估，适当调节任务进程同步的频率，减小进程同步和因未及时同步造成的重复计算开销。

（3）本文实现了一个自适应跨平台迁移机制系统框架，基于该框架评估其性能，验证了 ALM-HIP 机制的有效性。本文还针对设计的 NAPS 策略开展了实验，与固定间隔的进程同步策略对比评估该策略的性能，就实验结果开展了总结与讨论。

（4）本文针对计算复杂度高且时延敏感的环境，实现了一个适用于弹性任务计算场景的任务卸载调度原型系统。该系统的技术途径应用了内存资源满足优先策略和基于遗传算法的最小任务响应时间策略等两个策略，对于提交的任务集合，能够保证在节点资源满足需求的前提下，通过任务历史执行的时间数据预估每个任务的执行时间。此后，应用了遗传算法求解一个近似最佳的任务调度策略，使任务集合中的任务能够分配到合适的计算节点上，充分利用集群内的计算资源，最小化任务集合总响应时间。最后，基于该调度策略实现了一个卸载调度原型系统，并基于此系统评估调度策略的有效性。另外，本文将所设计的 ALM-HIP 机制与 NAPS 策略进一步整合到此原型系统中，使该系统在基本卸载调度功能的基础上，具备了跨平台的任务迁移和进程同步能力，从而可以较为全面地支持移动边缘场景下异构节点之间的任务卸载与迁移。

6.2　研究展望

在 5G 通信技术普及，物联网技术蓬勃发展的今天，众多具有低延时响应、高计算性能需求的应用应运而生。边缘计算的发展对于充分利用边缘网络的计算、存储和网络资源具有重要的意义，而如何通过实现端—边—云三者的协同实现资源的充分利用，为用户提供更佳的服务体验是边缘计算的研究问题。目前学术界对于边缘计算的研究热度仍处于上升阶段，随着边缘计算技术研究的层次加深，未来还会有更多的挑战。对于本文所涉及的技术而言，以下几个方面还值得更加深入地研究：

（1）本文所提出的 ALM-HIP 机制基于 Unikernel，由于 Unikernel 技术本身的构建和运行特性，新加入的跨平台迁移相关内核模块的开销仍需要进一步评估和优化。本文虽然实现了 Unikernel 实例独立完成跨平台迁移的功能特性，但尚未就该机制的复杂度进行进一步的优化，同时关于新的内核模块对于实例整体的能源消耗也尚未进行评估，即本文的工作目前尚停留于原型实现的阶段，关于自适应跨平台技术还需要更进一步的验证和优化。

（2）本文所提出的 NAPS 策略给出了对于网络状态描述的一种可能的建模形式，并以该策略对边-端网络中断概率进行了评估。然而在许多场景下，可能会出现网络评估情况较差，但网络中断概率相对很低的情况，需要针对这样的网络场景进一步优化该同步策略，使得其具有更强的适应性，进一步减少边缘网络中进程同步对被迁移实例因进程同步造成的性能开销。

（3）本文所实现的面向多目标优化的卸载调度原型系统中对于任务执行时间的预估采用了简单的平均方法。虽然就实验结果而言平均方法已经能够显现出原型系统的有效性，但就实际运行场景而言，还需要更加有效的方法，精准预估任务集合中每个任务的执行时间，避免因预估时间的偏差导致整个卸载调度算法的决策出现较大的误差。

参考文献

[1] 中国互联网络信息中心. 第 47 次中国互联网络发展状况统计报告 [R]. 北京: CNNIC, 2021.

[2] 施巍松, 张星洲, 王一帆, 等. 边缘计算: 现状与展望 [J]. 计算机研究与发展, 2019, 56(01): 69-89.

[3] Shi W, Cao J, Zhang Q, et al. Edge computing: Vision and challenges[J]. IEEE internet of things journal, 2016, 3(5):637-646.

[4] Hong C H, Varghese B. Resource management in fog/edge computing: a survey on architectures, infrastructure, and algorithms[J]. ACM Computing Surveys (CSUR), 2019, 52(5):1-37.

[5] Li C, Xue Y, Wang J, et al. Edge-oriented computing paradigms: A survey on architecture design and system management[J]. ACM Computing Surveys (CSUR), 2018, 51(2):1-34.

[6] Wang L, Jiao L, He T, et al. Service entity placement for social virtual reality applications in edge computing[C]//IEEE INFOCOM 2018-IEEE Conference on Computer Communications. IEEE, 2018:468-476.

[7] Ren P, Qiao X, Chen J, et al. Mobile edge computing–a booster for the practical provisioning approach of web-based augmented reality[C]//2018 IEEE/ACM Symposium on Edge Computing (SEC). IEEE, 2018:349-350.

[8] Liu Q, Han T. Dare: Dynamic adaptive mobile augmented reality with edge computing[C]// 2018 IEEE 26th International Conference on Network Protocols (ICNP). IEEE, 2018:1-11.

[9] Choy S, Wong B, Simon G, et al. The brewing storm in cloud gaming: A measurement study on cloud to end-user latency[C]//2012 11th Annual Workshop on Network and Systems Support for Games (NetGames). IEEE, 2012:1-6.

[10] Liu S, Liu L, Tang J, et al. Edge computing for autonomous driving: Opportunities and challenges[J]. Proceedings of the IEEE, 2019, 107(8):1697-1716.

[11] Peng H, Ye Q, Shen X S. Sdn-based resource management for autonomous vehicular networks: A multi-access edge computing approach[J]. IEEE Wireless Communications, 2019, 26(4): 156-162.

[12] Kaur K, Dhand T, Kumar N, et al. Container-as-a-service at the edge: Trade-off between energy efficiency and service availability at fog nano data centers[J]. IEEE wireless communications, 2017, 24(3):48-56.

[13] Saurez E, Hong K, Lillethun D, et al. Incremental deployment and migration of geo-distributed situation awareness applications in the fog[C]//Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems. 2016:258-269.

[14] Shen Z, Jia Q, Sela G E, et al. Follow the sun through the clouds: Application migration for geographically shifting workloads[C]//Proceedings of the Seventh ACM Symposium on Cloud Computing. 2016:141-154.

[15] Secci S, Raad P, Gallard P. Linking virtual machine mobility to user mobility[J]. IEEE Transactions on Network and Service Management, 2016, 13(4):927-940.

[16] Nadembega A, Hafid A S, Brisebois R. Mobility prediction model-based service migration procedure for follow me cloud to support qos and qoe[C]//2016 IEEE International Conference on Communications (ICC). IEEE, 2016:1-6.

[17] Al-Dhuraibi Y, Paraiso F, Djarallah N, et al. Autonomic vertical elasticity of docker containers with elasticdocker[C]//2017 IEEE 10th international conference on cloud computing (CLOUD). IEEE, 2017:472-479.

[18] Ksentini A, Taleb T, Chen M. A markov decision process-based service migration procedure for follow me cloud[C]//2014 IEEE International Conference on Communications (ICC). IEEE, 2014:1350-1354.

[19] Zhang C, Zheng Z. Task migration for mobile edge computing using deep reinforcement learning[J]. Future Generation Computer Systems, 2019, 96:111-118.

[20] Chen M, Li W, Fortino G, et al. A dynamic service migration mechanism in edge cognitive computing[J]. ACM Transactions on Internet Technology (TOIT), 2019, 19(2):1-15.

[21] Sinha P K, Doddamani S S, Lu H, et al. mwarp: Accelerating intra-host live container migration via memory warping[C]//IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). IEEE, 2019:508-513.

[22] Nadgowda S, Suneja S, Bila N, et al. Voyager: Complete container state migration[C]//2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS). IEEE, 2017: 2137-2142.

[23] Bhardwaj S, Jain L, Jain S. Cloud computing: A study of infrastructure as a service (iaas)[J]. International Journal of engineering and information Technology, 2010, 2(1):60-63.

[24] Zhang L, Litton J, Cangialosi F, et al. Picocenter: Supporting long-lived, mostly-idle applications in cloud environments[C]//Proceedings of the Eleventh European Conference on Computer Systems. 2016:1-16.

[25] Machen A, Wang S, Leung K K, et al. Live service migration in mobile edge clouds[J]. IEEE Wireless Communications, 2017, 25(1):140-147.

[26] Ma L, Yi S, Li Q. Efficient service handoff across edge servers via docker container migration[C]//Proceedings of the Second ACM/IEEE Symposium on Edge Computing. 2017:1-13.

[27] Qiu Y, Lung C H, Ajila S, et al. Lxc container migration in cloudlets under multipath tcp[C]// 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC): volume 2. IEEE, 2017:31-36.

[28] Maheshwari S, Choudhury S, Seskar I, et al. Traffic-aware dynamic container migration for real-time support in mobile edge clouds[C]//2018 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS). IEEE, 2018:1-6.

[29] VMware. What is esxi? bare metal hypervisor, esx, vmware[EB/OL]. 2021[2021-03-23]. https://www.vmware.com/products/esxi-and-esx.html.

[30] VirtualBox. Oracle vm virtualbox[EB/OL]. 2021[2021-03-23]. https://www.virtualbox.org/.

[31] VMware. Windows vm, workstation pro, vmware[EB/OL]. 2021[2021-03-23]. https://www.vmware.com/products/workstation-pro.html.

[32] Zhang F, Liu G, Fu X, et al. A survey on virtual machine migration: Challenges, techniques, and open issues[J]. IEEE Communications Surveys & Tutorials, 2018, 20(2):1206-1243.

[33] Clark C, Fraser K, Hand S, et al. Live migration of virtual machines[C]//Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2. 2005: 273-286.

[34] Hines M R, Deshpande U, Gopalan K. Post-copy live migration of virtual machines[J]. ACM SIGOPS operating systems review, 2009, 43(3):14-26.

[35] Hu L, Zhao J, Xu G, et al. Hmdc: Live virtual machine migration based on hybrid memory copy and delta compression[J]. Appl. Math, 2013, 7(2L):639-646.

[36] Nelson M, Lim B H, Hutchins G, et al. Fast transparent migration for virtual machines[C]// USENIX Annual technical conference, general track. 2005:391-394.

[37] Hirofuchi T, Nakada H, Itoh S, et al. Enabling instantaneous relocation of virtual machines with a lightweight vmm extension[C]//2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing. IEEE, 2010:73-83.

[38] Kim J, Chae D, Kim J, et al. Guide-copy: Fast and silent migration of virtual machine for datacenters[C]//SC'13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. IEEE, 2013:1-12.

[39] Kivity A, Kamay Y, Laor D, et al. kvm: the linux virtual machine monitor[C]//Proceedings of the Linux symposium: volume 1. Dttawa, Dntorio, Canada, 2007:225-230.

[40] Docker. Docker: Empowering app development for developers[EB/OL]. 2021[2021-03-23]. https://www.docker.com/.

[41] LXC. Linux containers[EB/OL]. 2021[2021-03-23]. https://linuxcontainers.org/.

[42] Podman. Podman[EB/OL]. 2021[2021-03-23]. https://podman.io/.

[43] CRIU. Criu[EB/OL]. 2021[2021-03-23]. https://www.criu.org/Main_Page.

[44] Madhavapeddy A, Scott D J. Unikernels: the rise of the virtual library operating system[J]. Communications of the ACM, 2014, 57(1):61-69.

[45] Unikernels. Unikernels - rethinking cloud infrastructure[EB/OL]. 2021[2021-03-23]. http://unikernel.org/.

[46] Lankes S, Pickartz S, Breitbart J. Hermitcore: a unikernel for extreme scale computing[C]// Proceedings of the 6th International Workshop on Runtime and Operating Systems for Supercomputers. 2016:1-8.

[47] Lertsinsrubtavee A, Ali A, Molina-Jimenez C, et al. Picasso: A lightweight edge computing platform[C]//2017 IEEE 6th International Conference on Cloud Networking (CloudNet). IEEE, 2017:1-7.

[48] Barbalace A, Lyerly R, Jelesnianski C, et al. Breaking the boundaries in heterogeneous-isa datacenters[J]. ACM SIGARCH Computer Architecture News, 2017, 45(1):645-659.

[49] Barbalace A, Karaoui M L, Wang W, et al. Edge computing: the case for heterogeneous-isa container migration[C]//Proceedings of the 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. 2020:73-87.

[50] Chen H Y, Lin Y H, Cheng C M. Coca: Computation offload to clouds using aop[C]//2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012). IEEE, 2012:466-473.

[51] Bruno R, Ferreira P. Alma: Gc-assisted jvm live migration for java server applications[C]// Proceedings of the 17th International Middleware Conference. 2016:1-14.

[52] Barbalace A, Sadini M, Ansary S, et al. Popcorn: Bridging the programmability gap in heterogeneous-isa platforms[C]//Proceedings of the Tenth European Conference on Computer Systems. 2015:1-16.

[53] Olivier P, Mehrab A F, Lankes S, et al. Hexo: Offloading hpc compute-intensive workloads on low-cost, low-power embedded systems[C]//Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing. 2019:85-96.

[54] DeVuyst M, Venkat A, Tullsen D M. Execution migration in a heterogeneous-isa chip multiprocessor[C]//Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems. 2012:261-272.

[55] Bhat S K, Saya A, Rawat H K, et al. Harnessing energy efficiency of heterogeneous-isa platforms[J]. ACM SIGOPS Operating Systems Review, 2016, 49(2):65-69.

[56] Android. Concepts, android ndk, android developers[EB/OL]. 2021[2021-03-23]. https://developer.android.com/ndk/guides/concepts.

[57] Lee G, Park H, Heo S, et al. Architecture-aware automatic computation offload for native applications[C]//Proceedings of the 48th international symposium on microarchitecture. 2015: 521-532.

[58] Dubey A, Wagle D. Delivering software as a service[J]. The McKinsey Quarterly, 2007, 6 (2007):2007.

[59] Ahmed E, Naveed A, Gani A, et al. Process state synchronization-based application execution management for mobile edge/cloud computing[J]. Future Generation Computer Systems, 2019, 91:579-589.

[60] Bailey D, Barszcz E, Barton J, et al. The nas parallel benchmarks rnr-94-007[J]. NASA Advanced Supercomputing Division, Tech. Rep, 1994.

在学期间完成的相关学术成果

学术论文：

[1] **Jin H**, Su Y, Liang F, Liu F. Heterogeneous-ISA Application Migration in Edge Computing: Challenges, Techniques and Open Issues[C]. The 7th International Conference on Artificial Intelligence and Security (ICAIS). Dublin, Ireland. July 19-23, 2021.（已录用，EI 索引，与学位论文第 2 章相关）

[2] Su Y, **Jin H**, Liu F, Li W. SACache: Size-Aware Load Balancing for Large-Scale Storage Systems[C]. The 7th International Conference on Artificial Intelligence and Security (ICAIS). Dublin, Ireland. July 19-23, 2021.（已录用，EI 索引）

[3] 刘芳, **金航**, 梁丰洲, 陈志广. 计算机组成原理实验改革——CPU 设计与汇编程序设计的结合 [C]. 南京: 中国高校计算机教育大会, 2020.（已录用）

专利：

[4] 刘芳, 梁家越, **金航**, 肖侬. 一种风险感知的边缘计算任务分配方法 [P]. CN112052092A, 2020-12-08.（已公开，与学位论文第 5 章相关）

在学期间参加的主要科研工作：

[5] **** 环境下的 **** 与弹性服务技术. 装备预先研究项目, 2018.12-2020.12.（学生骨干）

[6] 高效能可扩展边缘计算体系结构关键技术研究. 国家自然科学基金面上项目, 2021.01-2024.12.（学生骨干）

致 谢

白云山下，珠江之畔；羊城初夏，来之甚早。中山楼外，花开二度；逸仙路上，你来我往。槐月方始，阴雨连绵；举首回望，别离将至。

经师易遇，人师难求。恩师刘芳教授，引我入学术之堂，导我于迷途困境。务学不如务求师，修学不忘师者情。古来学者必有师，得良师者得天下。精钻细磨终成文，夜半未眠者何人。人非生而晓事理，皆仗先生传道业。学问学问，学术业之理，问日常点滴。先生非但学术之师，更为人生之师。先生桃李遍天下，何必寥寥数言诉。�堂前，绿荫如旧；先生之德，永存于心。

桃李无言，下自成蹊。恩师陈志广教授，伴我行学途之上，待我以细致关怀。善出于教，教本在师；片言相教，即为吾师。授书传业无终时，师恩定当涌泉报。筱园路开，拨云见日；先生之情，永无相忘。

师门诸君，郭烨婷师姐，梁家越师兄，蔡振华师兄，黎燊师兄，张振源师兄等。于新进之时答疑解惑，于启蒙之时传授经验。苏屹宏同学，梁丰洲同学等，亦为良友。于学术研究相互帮助，于日常之中相互支持。冥冥之中似有缘，五湖四海得相见。志同道合求学问，不负风华正茂时。

身体皮肤，受之父母；老生常谈幸听之，充耳不闻何成人。年过半百，银丝既见；日夜劳苦，是为何人。夜半思念，目泉润湿；万爱千恩，终有报时。

博学慎思，明辨笃行；审自身之品行，问术业之高峰。孙文先生，十字校训；铭记心中，伴我一生。感恩母校，予我机遇；感恩诸师，予我智慧。

中山大學
SUN YAT-SEN UNIVERSITY
1924