软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2021,32(12):3945−3959 [doi: 10.13328/j.cnki.jos.006113]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

功能分发网络:基于容器的智能边缘计算平台*

杨 术[1], 陈子腾[1], 崔来中[1], 明中行[1], 程 路[2], 唐小林[3], 萧 伟[4]

[1](深圳大学 计算机与软件学院,广东 深圳 518060)

[2](中国移动通信集团浙江有限公司,浙江 杭州 310016)

[3](华润集团-润联智慧科技有限公司,广东 深圳 518060)

[4](深圳清华大学研究院,广东 深圳 518057)

通讯作者: 崔来中, E-mail: cuilz@szu.edu.cn

摘 要: 随着大数据、机器学习等技术的发展,网络流量与任务的计算量也随之快速增长.研究人员提出了内容分发网络(CDN)、边缘计算等平台技术,但 CDN 只能解决数据存储,而边缘计算存在着难以管理和不能跨集群进行资源调度等问题.容器化技术广泛应用在边缘计算场景中,但目前,边缘计算采取的容器编排策略普遍比较低效,导致任务的计算延迟仍然过长.提出了功能分发网络 FDN(function delivery network),一方面为用户提供了访问边缘计算资源的统一接口和容器化的计算平台,无需进行繁琐的计算资源配置;另一方面,FDN 平台优化系统的资源利用和任务的计算延迟,能将任务所需的容器编排到合适的边缘计算集群.开发了一种基于启发式的容器编排策略,实现了跨集群的容器编排功能,进一步优化了执行的计算延迟.基于 Openwhisk 软件实现了 FDN,并在中国移动的网络中部署了该系统,而且对 FDN 和容器编排策略进行测试.实验结果表明,FDN 计算平台能够降低任务的计算延迟;同时,启发式容器编排策略的性能相比传统的算法有了较大的提升.

关键词: 功能分发网络;边缘计算;容器编排

中图法分类号: TP393

中文引用格式: 杨术,陈子腾,崔来中,明中行,程路,唐小林,萧伟.功能分发网络:基于容器的智能边缘计算平台.软件学报,2021,32(12):3945−3959. http://www.jos.org.cn/1000-9825/6113.htm

英文引用格式: Yang S, Chen ZT, Cui LZ, Ming ZX, Cheng L, Tang XL, Xiao W. Function delivery network: Container-based smart edge computing platform. Ruan Jian Xue Bao/Journal of Software, 2021,32(12):3945−3959 (in Chinese). http://www.jos.org.cn/1000-9825/6113.htm

Function Delivery Network: Container-based Smart Edge Computing Platform

YANG Shu[1], CHEN Zi-Teng[1], CUI Lai-Zhong[1], MING Zhong-Xing[1], CHENG Lu[2], TANG Xiao-Lin[3], XIAO Wei[4]

[1](College of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518060, China)

[2](China Mobile Communication Group Zhejiang Co., Ltd., Hangzhou 310016, China)

[3](Runlian Intelligent Technology Co., Ltd, China Resources, Shenzhen 518060, China)

[4](Research Institute of Tsinghua University in Shenzhen, Shenzhen 518057, China)

Abstract: With the development of big data and machine learning, the network traffic and data computation are growing fast. Researchers developed the content network delivery (CDN), edge computing, etc. for these challenges. Nevertheless, the CDN only

* 基金项目: 国家重点研发计划(2018YFB1800302, 2018YFB1800805); 国家自然科学基金(61772345, 61902258); 深圳市基础研究计划(RCYX20200714114645048, GJHZ20190822095416463, JCYJ20190808142207420); 广东省"青年珠江学者计划"

Foundation item: National Key Research and Development Program of China (2018YFB1800302, 2018YFB1800805); National Natural Science Foundation of China (61772345, 61902258); Major Fundamental Research Project in the Science and Technology Plan of Shenzhen (RCYX20200714114645048, GJHZ20190822095416463, JCYJ20190808142207420); Pearl River Young Scholars

收稿时间: 2020-02-26; 修改时间: 2020-04-02; 采用时间: 2020-06-16

addresses the data storage, and it is still challenging to manage and schedule resources among edge clusters in edge computing. Containerization has been widely employed in edge computing, but the current container orchestrators utilize the inefficient orchestration schemes, which leads to high computation latency. Thisstudy proposes the function delivery network (FDN). On the one hand, FDN provides the interface and containerization computation platform for users to access the edge computing resources. On the other hand, FDN optimizes the resource utilization and computation latency by orchestrating the containers to the appropriate edge clusters. Moreover, a heuristic container orchestrating algorithm is developed that enables the inter-cluster container orchestrating. The FDN system is implemented based on Openwhisk and the FDN system is deployed in China Mobile network, and the FDN system is evaluated. The results show that the proposed FDN system can decrease the task computation latency, and the heuristic container orchestration algorithm outperforms the traditional container orchestration schemes.

Key words: function delivery network; edge computing; container orchestration

随着物联网、机器学习、VR/AR 等应用的发展,网络数据量正在快速增大.5G 时代的到来,加快了数据产生的速度,网络的流量也会大幅增加,并会出现更多基于 5G 的新型应用.这些应用都将大幅度增加数据规模和网络流量,对数据实时处理的速度提出了更高的要求[1].

学术界和工业界也提出了很多提高数据处理效率和应对网络流量的解决方案.Pallis 等人提出了内容分发网络 CDN(content delivery network)[2],将网络资源存放到分布式的服务器上,而用户可以根据网络情况访问响应速度更快的服务器上.但是 CDN 只解决了数据存储,而不涉及数据计算.边缘计算(edge computing)[3,4]将服务器部署到距离用户更小的边缘端,从而高性能的边缘服务器能够以更小的延迟处理用户的数据请求和流量.但是边缘计算存在大量分布式的边缘计算集群,面对用户不同的计算任务需求,管理者经常需要将不同的计算环境从管理中心迁移到相应的边缘服务器,这加大了应用的开发和部署难度.

容器化技术是当前学术界和工业界研究的热点问题之一,它能够快速直接在操作系统层级向用户开放多个独立的用户态空间实例,并支持携带程序包与软件库,让容器直接在机器部署并运行,并在不同计算实体上实现资源的快速迁移,省去了开发人员繁琐的资源部署和配置步骤.研究人员也提出了一些容器化的边缘计算平台,改善了边缘计算中环境迁移困难的问题.但由于其分布式架构的特点,边缘计算平台仍然存在着分布式资源难以统一管理的问题,包括网络流量和计算资源的管理和调度等.因此,我们需要改进现有的架构,开发一种方便部署、管理和统一调度的边缘计算平台.

本文中,我们提出了功能分发网络 FDN(function delivery network),能够有效管理和调度分布式边缘服务器的计算资源,并结合无服务化计算(serverless computing)[5],为用户提供了访问计算资源的接口.我们将网络分成了控制器、DNS(domain name server)以及边缘计算集群.其中:DNS 会解析用户的地址以及提交的任务等信息;而控制器会收集网络流量和用户的信息,计算出一个容器编排策略,将任务对应的容器分配到不同的边缘计算器,最小化任务的计算延迟;边缘计算集群会根据编排策略将相应容器部署并运行,并最终将任务计算后的结果返回给用户.FDN 保持了边缘计算中网络延迟小、响应速度快的优点,同时,通过中心化控制器统一管理网络中的边缘计算集群和流量信息,实现了分布式资源的收集、管理和调度.另外,通过容器技术并结合无服务化计算,FDN 将计算任务函数化,实现了计算环境的快速迁移.

由于不同的容器编排策略会对系统的计算性能产生影响,因此在 FDN 中,我们需要考虑容器编排的效率问题.目前,工业界存在一些容器编排工具,例如 Mesos[6],Kubernetes[7],Yarn[8],Borg[9]等,但它们不能取得很好的编排性能.一方面,它们采用的是低效的容器编排策略,例如先来先服务等;另一方面,它们没有考虑计算任务的数据结构.对于以有向无环图(directed acyclic graph,简称 DAG)作为底层的数据结构的复杂计算任务,可能存在着多个子任务,并且子任务间存在着数据依赖和执行顺序要求,因此,我们需要考虑不同容器的编排顺序.同时,目前的编排器只支持单一数据中心,无法在多集群系统中实现跨集群的容器资源编排.

为了在边缘计算平台中取得更好的容器编排效果,本文为 FDN 系统设计了一种基于启发式的跨集群容器编排策略.我们证明了计算最优化容器编排问题是一个 NP 难问题,无法在多项式时间内解决.因此,我们开发了基于启发式的容器编排算法.它综合考虑了有向无环图任务中子任务的先后执行顺序以及每个子任务的计算量、计算需求等信息,同时考虑了用户到集群的访问延迟以及集群本身的网络流量、闲置资源等信息.算法会

根据任务和网络流量信息生成一个待编排的容器列表,并按照优先级进行排序.算法会不断访问待编排容器列表,按顺序把容器贪心地编排到完成时间最快同时满足计算要求的边缘计算集群,直到把所有容器编排完毕.每当用户提交计算任务时,FDN 控制器将收集全局的网络流量和任务信息并执行上述过程,最终将容器编排策略传送给相关的集群和用户.相比传统的简单容器编排算法,启发式容器编排策略能够更合理地实现容器的编排过程,进而优化系统的计算延迟.

我们基于 IBM 开发的无服务化计算开源软件 Openwhisk 实现了功能分发网络技术,并在中国移动网络中部署了 FDN 计算平台.我们租用 6 个阿里云实例作为核心网环境,其中 3 个实例作为 Master 节点,另外 3 个作为 Worker 节点.同时,部署了两个边缘计算中心,每个中心由 1 台机架式服务器和 6 台配置不同的台式机模拟.我们在网络中部署了客户端,并进行真实的人脸识别的任务测试.实验结果表明,FDN 计算平台能够取得 137.82ms 的平均计算延迟.同时,我们在不同条件下模拟了基于启发式的容器编排策略,并与传统的编排算法作性能对比.实验结果表明了,我们的编排策略能够自适应于不同的网络环境.与传统的编排算法相比,启发式的容器编排策略能够取得更好的计算性能,提高了 FDN 的计算效率.

本文的贡献列举如下:

- 提出了功能分发网络 FDN,它为用户提供了统一的接口,使得用户能够上传自己的计算任务,并且系统会将该任务分配给最合适的边缘计算集群,让用户享受到低延迟高性能的计算服务;
- 提出了一种基于启发式的容器编排策略,实现容器的跨集群编排,优化用户任务的计算延迟;
- 实现并部署了 FDN 系统并评估了其性能,实验结果表明:我们提出的 FDN 架构和容器编排算法能够取得很好的计算延迟,能够满足当前网络中的数据计算需求.

本文第 1 节将会介绍当前的计算平台和容器编排工具的相关工作.第 2 节和第 3 节将会展示提出的 FDN 平台和容器编排模型.FDN 系统的实现细节、部署和测试会在第 4 节进行介绍.容器编排算法模拟的实验过程和结果将会分别在第 5 节进行展示.最后,结论会在第 6 节中给出.

1 相关工作

1.1 流量优化与计算平台

当前,网络数据量呈指数级的增长速度.如何更高效快速处理这些庞大数据和流量,成为网络领域的最大挑战之一.学术界和工业界提出了很多流量优化和计算平台的改进方案.CDN[2]是一种分布式的存储机制,它将数据内容分发到分布式服务器上,而用户能够在靠近的服务器上直接获取数据内容,避免访问距离更远的中心服务器.但是 CDN 只针对数据存储,无法提供数据的计算等功能.云计算[10,11]通过搭建高性能的计算中心,使得用户能够把计算任务和数据上传到云计算平台,并由高性能的机器进行计算.同时,用户能够按需购买指定数量和性能的机器和计算服务,为用户节省了购买服务器的费用.但是文献[12]指出:当前的云计算平台是中心化的,数据中心部署在距离用户较远的地方,这会导致用户到中心服务器的延迟较大,无法满足当前实时性的要求.同时,在使用云计算平台过程中,开发人员需要花费大量时间进行服务器的环境搭建及数据和计算资源的部署等,加大了开发人员的开发难度.

1.2 边缘计算

为了解决中心化平台延迟大的问题,研究人员提出了边缘计算[3,4].边缘计算采用了分布式的网络拓扑结构,多个边缘服务器集群被放置在距离用户更近的边缘端,而用户能够把计算任务和数据提交给相对距离更近的边缘计算集群并由其进行计算.相比云计算,分布式的边缘计算平台降低了用户到计算资源的距离和延迟,从而提高了数据流量的处理和计算速度.边缘计算现已成为业界研究的热点之一,并应用到不同的领域中,例如物联网[13]、5G[14]等.

但是边缘计算是基于分布式的网络结构,如何管理、调度和优化分布式的边缘计算资源,成为了最大的挑战之一.Teixeira 等人[15]指出:由于缺少统一的管理和调度中心,分布式的边缘计算平台无法获得网络流量和计

算资源的实时情况,因此无法得到最优的资源管理和调度方案,也无法达到更好的计算性能.因此,我们为分布式的边缘计算平台提供一个中心化的结构,负责管理和调度边缘计算平台的流量和分布式计算资源,提高边缘计算平台的性能表现.

1.3 无服务化计算与容器编排模型

无服务化计算(serverless computing)[5]是一种新的计算平台,为用户提供了函数化的计算服务.用户只需要调用统一接口,就能够进行函数化计算,无需额外管理和调度运行时需要的资源,使得用户可以更好的关注自己的业务逻辑.利用容器化技术,用户提交的函数将被快速部署到容器,并运行在对应的计算实体上,这免去了复杂的计算资源部署、配置和调度过程.同时,容器能够在不同的计算实体进行快速迁移,这提高了计算平台的可扩展性,提高用户的开发效率.

在容器技术使用过程中,我们需要把容器部署和运行在恰当的服务器上,这也是容器编排的概念.工业界提出了许多容器编排器,包括 Mesos[6],Kubernetes[7],Yarn[8],Borg[9]等.但是目前的容器编排器采用的仍是一些简单的编排策略,例如,Kubernetes 和 Borg 只提供了简单的先来先服务(FCFS)和基于优先级顺序的容器编排策略,而 Mesos,Yarn 等编排工具只支持先来先服务的编排策略.它们没有考虑到任务和容器本身先后执行顺序等信息,并且现有的容器编排器都是基于单个数据中心的,无法进行跨集群的容器编排.对于分布式边缘计算网络,现有的容器编排策略无法在多集群平台中使用.因此在本文中,为了适应 FDN 网络中多个边缘计算集群的分布式结构,我们设计一种跨集群的容器编排器,提供一种基于有向无环图结构的容器编排策略,优化用户到边缘集群的计算延迟.

2 FDN 技术架构

2.1 FDN系统概况

我们设计了一种新的边缘计算平台,功能分发网络 FDN,系统结构如图 1 所示.它是一种分布式的边缘计算平台,能够为用户提供基于函数的计算服务.在 FDN 平台中,用户只需要通过 FDN 提供的统一接口,上传计算任务数据等相关信息,就能够访问 FDN 中边缘计算的资源.此时,复杂的计算任务就会被上传到系统,并且控制器将根据任务和网络流量等信息,把相关容器编排和部署到合适的边缘计算集群,最终将计算结果返回给用户.

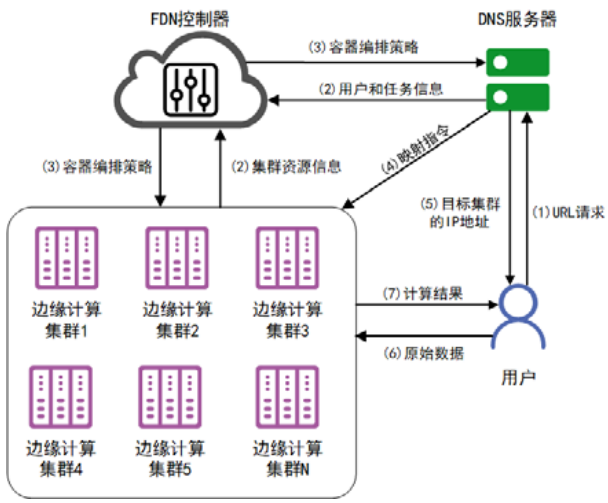


Fig.1 Overview of FDN system

图 1 FDN 系统概况

在 FDN 中,我们主要有以下角色.

- 用户:向 FDN 系统上传相应的计算任务和数据,访问边缘计算资源,并且支付对应的费用;
- 边缘计算集群:负责计算用户提交的复杂的计算任务,并且将计算结果返回给用户;
- 控制器:负责收集用户和分布式边缘集群的信息(例如网络流量、闲置资源等),并且通过算法得出一个容器编排和调度策略;
- DNS:负责为用户提供统一的接口,让用户能够访问边缘计算资源.它根据用户的请求解析出用户所在的位置,同时接收控制器的容器编排和调度策略,将用户与对应的边缘计算集群进行绑定.

FDN 的运行过程如图 2 所示,主要分为以下步骤进行.

(1) 用户会通过 FDN 提供的统一接口,向 DNS 发出请求,并提交相应的任务和网络等信息;

(2) DNS 会解析用户的地址信息,并将其发送给控制器;控制器同时也会采集边缘计算集群的状态信息,包括网络状况、计算资源被占用率和闲置情况等;

(3) 控制器根据收集到的信息计算得到一个合理的调度和容器编排策略,并将其回传给 DNS,并将容器部署到对应的边缘计算集群.由于不同容器编排策略将直接影响系统的计算效率,因此我们将在第 3 节研究 FDN 的容器编排问题;

(4) DNS 将根据编排策略,将用户与对应的边缘计算集群进行绑定;

(5) DNS 将目标边缘计算集群的网络信息传给用户,使用户能够连接到目标集群;

(6) 用户将原始数据发送给边缘计算集群,最后,边缘计算集群将会进行计算并把计算结果返回给用户.

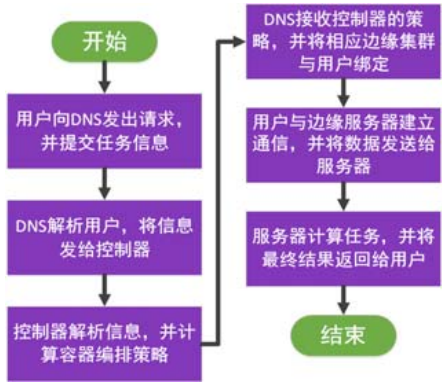


Fig.2 Flow chart of FDN system

图 2 FDN 运行流程图

在设计上,FDN 结合了无服务化计算及边缘计算的特点.通过系统提供的统一接口,用户可以访问到无服务化计算的计算资源.在实际运行过程中,我们采用 URL 链接的形式作为接口,用户只需要输入 URL 地址,就能够访问 FDN 资源.当用户通过接口上传自己的任务和位置信息后,利用容器化技术,计算任务所对应的容器将会被部署并运行在边缘计算集群,而用户只需提交自己的任务信息和原始数据,就能够获得无服务化计算的计算服务.

我们在 FDN 系统中会维护一个代码库(code repository),其中包含一些基本的容器镜像,例如人脸识别、大数据分析、目标检测等,保证了用户常用的计算需求都能得到满足.用户能够方便高效地调用并使用无服务化计算资源,不需要再进行繁琐的环境和计算资源部署过程;同时,我们改善了边缘计算的分布式架构.我们引入了控制器和 DNS 等中心化组件,负责管理和优化边缘计算集群和用户的资源和网络流量信息.控制器也能够根据网络状况计算合理的容器编排策略,保证了 FDN 平台合理正常运行,克服了传统边缘计算平台难以管理的缺点.FDN 既拥有方便使用、管理和调度的优点,又能够保证更好的计算延迟,满足了许多应用实时性和计算速度的需求.

2.2 FDN控制器

控制器是 FDN 中心化的调度组件,负责整个系统的正常运转.它负责流量优化和容器的编排,让计算任务调度到更合适的边缘计算集群.同时,控制器还负责用户接入、代码库的维护、网关等功能,保持系统稳定高效运行.FDN 控制器包含了以下几个组件.

1) FDN-Web:负责存储用户的个人信息、项目信息以及功能函数的创建等功能;
2) FDN 分发器:负责计算并实现容器的跨集群编排,优化每个计算任务的计算延迟.编排算法会在第 3 节展示;
3) FDN 费用中心:负责记录用户使用平台所需要支付的费用;
4) FDN 网关:负责收集网络的流量信息以及函数的分发与路由、集群内的负载均衡等功能.同时,FDN 网关还承担着流量控制、安全策略管理、用户授权等功能,保证 FDN 系统安全可靠的运行;
5) FDN 代码库管理:负责管理和维护代码库的稳定和更新,并且判定用户上传的容器镜像是否合法以及有效.

为了更好地管理分布式边缘集群,我们还设计了分层的控制器管理结构.在每个边缘计算集群内部有一个控制器,分别管理自己所在的集群.而全网还有一个中心控制器,作为全网的指挥,收集来自各个集群控制器的信息,以实现对全网信息的统一管理和调度.这种分层结构能够同时在中心和边缘端更便捷高效地管理各类计算资源和网络流量,保证 FDN 平台的可靠运行.

2.3 边缘计算集群

为了让不同地理位置的用户享受更低的计算延迟,多个边缘计算集群分布在网络的不同地点,并由控制器进行管理.我们为每个边缘服务器安装了容器的运行环境.当用户提交计算任务时,目标边缘计算集群会根据控制器的容器编排策略,从代码库中下载、安装并运行任务所需要的容器.同时,它会接收 DNS 的映射指令并与对应的用户绑定,与用户进行直接的数据通信,并最终将计算结果返回给用户.

2.4 DNS

DNS 为用户提供了统一的接口,用户在使用时,只需要输入一个 URL 地址,就可以访问计算资源.DNS 会解析用户的地址,并且将地址和任务信息传送给中心控制器.同时,在容器编排时,DNS 将与控制器协同工作.当控制器计算出容器编排策略后,DNS 会根据调度策略将目标边缘计算集群与用户进行绑定,通过发送映射命令,让它们进行数据通信.我们指出 DNS 在 FDN 系统中起到了桥梁的作用,连接了用户与中心控制器,为控制器提供准确的用户位置和任务信息.DNS 将解析用户信息与计算调度策略功能进行解耦,为系统的可扩展性提供了有力支撑;同时也能够减轻中心控制器的压力,保证平台的可靠运行.在实际运行中,为了让用户以更小的延迟访问 FDN 资源,我们采用了 Smart DNS 技术,优化地址访问和解析策略,以适应 FDN 在不同用户规模下的实际使用效果.FDN 中采用的 Smart DNS 具体分为两个部分:1) 控制器根据容器编排算法,计算出最优的用户流量调度策略,然后将用户的 IP 地址与边缘服务器对应,更新 DNS 服务器;2) DNS 本身不需要做任何修改,但它直接与控制器联动,共同组成 FDN 的 Smart DNS 功能.

通过以上各个组件的协同工作,用户能够享受到便捷的计算服务,同时保证了用户的计算延迟.但是我们指出,对于容器化技术,目前的容器编排器采用的是一些较为简单的编排策略(例如先来先服务、优先级等).这种编排策略没有考虑到容器和网络流量等重要信息,会加大任务的计算延迟;同时,对于分布式的网络架构,现有的编排器无法支持跨集群的资源调度,不能很好地支持边缘计算的系统架构.相比传统的单集群内部调度,跨集群容器编排需要进行容器的迁移,需要考虑集群间的传播延迟;而单集群调度不需要考虑延迟,因为集群内部的传播延迟约等于 0.我们指出:传统的容器编排方案(例如,使用 docker 和 kubernetes 分别作为容器引擎和编排器)只支持单集群的容器编排,无法扩展到多集群容器编排.因此,本文为 FDN 设计了一种支持跨集群的容器编排策略,支持在多个边缘计算集群间进行容器调度,发挥 FDN 控制器统一管理的优势,进一步优化容器化计算平台的延迟.

3 容器编排

3.1 问题定义

定义的记号见表 1.

Table 1 Notation list

表 1 记号表

符号	意义	符号	意义
P	边缘计算集群集合	$r(c)$	容器 c 运行时所需要的资源
p	一个边缘计算集群	$\delta(c)$	容器 c 的计算量
$\theta(p)$	集群 p 能分配的算力大小	$\bar{t}_c$	容器 c 的平均计算时间
u	一个用户	$I(p)$	计算设备 p 的闲置资源
$J=(V,E)$	一个计算任务以及对应的图结构	$f(c,p)$	容器 c 是否被编排到设备 p 上
$\omega(c_i,c_j)$	容器 c_i 和 c_j 的通信代价	$EST(c,p)$	容器 c 在集群 p 上的最早执行时间
C	任务运行时的容器集合	$EFT(c,p)$	容器 c 在集群 p 上的最早完成时间
c	任务运行时的一个容器	$AST(c)$	容器 c 的实际开始时间
$prev(c)$	容器 c 的父容器集合,早于 c 计算	$AFT(c)$	容器 c 的实际完成时间
$succ(c)$	容器 c 的子容器集合,晚于 c 计算		

在 FDN 系统中,定义 $P=\{p_1,p_2,\dots,p_n\}$为系统中的 n 个边缘计算集群集合,其中,$p_i\in L(1\leqslant i\leqslant n)$表示第 i 个边缘计算集群.当一个用户 u 申请 FDN 服务时,它会向中心控制器提供该计算任务 J 运行时所需要的容器镜像信息.对于一个计算任务 J,它可能存在着多个子任务,而且这些子任务之间存在着一定的依赖关系,执行时需要遵循一定的先后顺序.任务 J 可抽象成一个有向无环图 $J=(V,E)$,其中,$V=\{v_1,v_2,\dots,v_{|V|}\}$表示子任务集合,而 $E=\{e_1,e_2,\dots,e_{|E|}\}$表示子任务间的数据依赖关系.对于节点 v_a 到节点 v_b,假设它们之间存在有向边(e_a,e_b)从节点 v_a 指向节点 v_b,表示只有先执行节点 v_a 才能执行节点 v_b.对于一个节点 v,我们将它的父节点集合和子节点集合分别定义为 $prev(v)$和 $succ(v)$;同时,我们假设对于每一个任务 J,它都会存在唯一的入口任务(entry job,也称为 v_{entry}),而出口任务(exit job,也称为 v_{exit})可能有多个.也就是说:入口任务的 v_{entry} 入度为 0,而出口任务 v_{exit} 的出度大于等于 0.当任务在被调度和编排的时候,系统都会从 v_{entry} 开始执行,并且最终某一个 v_{exit} 结束计算.我们令执行入口任务的容器称为 c_{entry},令执行出口程序的容器称为 c_{exit}.

对于任务 J,我们假设它运行时可分配的容器集合为 $C=\{c_1,c_2,\dots,c_m\}$,其中,$c_i\in C$ 代表每一个独立的容器.我们假设每一个子任务 j 都由一个独立的容器 c 进行计算,因此一个任务 J 的执行需要由多个容器间的协同工作来完成.由于子任务之间存在着一定的数据依赖,因此容器的编排顺序也受到了要求;同时,容器间也存在着数据通信.当容器 c_i 和 c_j 被编排到两个不同的边缘计算集群时,定义通信代价为$\omega(c_i,c_j)$,其中,$\omega(c_i,c_j)$为正整数.而如果容器 c_i 和 c_j 被编排到相同的边缘计算集群,那么通信代价为 0,即$\omega(c_i,c_j)=0$.由于中心云到边缘云的延迟比较稳定,而且容器在边缘会有缓存时间,所以用户后续上传的数据都可以享受低延迟的服务.在本文中,我们主要考虑用户的流量调度,暂不考虑中心云下发容器的过程延时.在以后的研究中,我们将会进一步研究容器从中心云下放到边缘云的延迟这一重要因素.

对于每一个容器 c,当它被编排到某一个边缘计算集群时,需要集群内的机器花一定时间进行计算.我们假设容器 c 的计算量为$\delta(c)$,其中,$\delta(c)$为正整数,并且假设边缘计算集群 p 当前的闲置算力为$\theta(p)$,$\theta(p)$也为正整数.那么容器 c 在系统中的平均完成时间可以表示为

$$\bar{t}_c=\sum_{p\in P}\frac{\delta(c)}{\theta(p)}\Big/n.$$

每一个容器都会有对于计算设备的需求,我们把容器 c 的需求(包括内存、存储等)定义为 $r(c)$,其中,$r(c)$的值为正整数;同时,对于集群 $p\in P$,它都会有当前闲置的资源量,我们将其定义为 $I(p)$,并且 $I(p)$为正整数.如果容器 c 要在集群 p 上顺利运行,那么 c 的计算需求必须小于等于 p 的闲置资源,即 $r(c)\leqslant I(p)$.

在容器计算的过程中,用户需要实时把数据传送给对应的容器进行计算.定义 $d(u,c,p)$为用户 u 到容器 c 所在的边缘计算集群 p 的延迟.为了简单起见,令每 $d(u,c,p)$在 0 到 1 之间变化,也就是 $0\leqslant d(u,c,p)\leqslant 1$.我们令 p_{entry}

和 p_{exit} 分别表示入口容器和出口容器所在的集群,那么用户 u 到 p_{entry} 和 p_{exit} 的延迟我们也简单表示为 d_{entry} 和 d_{exit}.

另外,为了表示某个容器是否被编排到某个边缘集群,我们令 $f(c,p)$表示为

$$f(c,p)=\begin{cases}1, & \text{容器}c\text{被编排到集群}p\text{上}\\ 0, & \text{容器}c\text{不被编排到集群}p\text{上}\end{cases}.$$

对于用户 u 提交的任务,当容器被编排到对应的集群时,任务的计算延迟包括 3 部分:一部分是在服务器上的计算消耗时间,一部分是容器间的通信时间,另一部分是用户传送数据给容器所在的边缘服务器的延迟.我们定义容器 c 的实际开始执行时间(actual start time)为 $AST(c)$和实际完成执行时间(actual finish time)为 $AFT(c)$.类似地,我们定义将容器 c 在集群 p 上最早的开始执行时间(earliest start time)为 $EST(c,p)$,并且:

$$EST(c,p)=d(u,c,p)+\max\{avail[p],\max_{c_m\in pred(c)}(AFT(c_m)+\omega(c_m,c))\}.$$

其中,$avail[p]$表示为集群 p 准备好执行容器 c 的时间,而公式后面的 max 函数指的是集群 p 等待执行完容器 c 的父容器的最大时间.特别地,对于入口容器 c_{entry},它的最早开始执行时间为用户到入口容器所在的集群延迟,即 $EST(c_{entry},p_{entry})=d_{entry}$.我们同时定义容器 c 在集群 p 上的最早完成执行时间(earliest finish time)为 $EFT(c,p)$,其中,该值的大小包括容器 c 的执行时间加上最早执行时间,即:

$$EFT(c,p)=\overline{t_c}+EST(c,p).$$

我们指出,$EST(c,p)$和 $EFT(c,p)$的值可以通过迭代的方式计算得到.同时,当容器已经确认被编排到某个计算集群上时,它在集群上的最早执行时间就等于实际开始执行时间;同时,它在集群上的最晚完成时间就等于实际完成实行时间.在本文中,我们的目标是让所有任务的计算延迟最小(其中包括各个容器在集群上的计算时间、容器间的数据通信时间以及用户到容器的延迟).因此,算法的目标是让某个任务的最后一个出口容器的计算延迟最小,同时满足设备的闲置资源和容器的资源需求,我们将其形式化为

$$\min\max AFT(c_{exit})$$
$$\text{s.t.}\sum_{c\in C,\,p\in P}r(c)\times f(c,p)\leqslant I(p)$$

3.2 编排算法

我们指出,存在最优算法得到计算延迟最小的容器编排策略.最优算法的主要思路是:遍历所有满足限制要求的容器编排策略,并最终选出一个计算延迟最小的编排方案.我们指出:只要等待足够长的时间,最优算法始终能计算出最优的容器编排方案,任务的计算延迟也必然是最小的.但是,最优算法不能够在线性时间内解决,这对于一个实际应用的系统显然是不合适的.接下来,我们会证明使用最优算法来得到最优的容器编排策略是一个 NP 难的问题.

定理 1. 找到一个最优的容器编排策略是一个 NP 难问题.

证明:在线性时间内,验证一个给定的容器编排策略是可行的.因此,找到一个最优的容器编排策略属于 NP 问题的范畴.为了证明该问题是一个 NP 难问题,我们将最大子集和问题归约为最优容器编排问题.其中,最大子集和问题已经被证明为 NP 完全问题[16].最大子集和问题是:给定一个有限集合 Q,对于每一个 $q\in Q$,都有一个特定的值 $v(q)\in Z^+$、一个正整数 $B\in Z^+$,找到一个子集 $Q'\subseteq Q$,让 $\sum_{q\in Q'}v(q)\leqslant B$ 并且 $\sum_{q\in Q'}v(q)$ 最小化.

假设现在有两个边缘计算集群 p_A 和 p_B,并假设用户 u 到它们的延迟分别为 $d(u,p_A)$和 $d(u,p_B)$,其中, $d(u,p_A)$的值很小,而 $d(u,p_B)$的值很大.同时假设 p_A 和 p_B 的可分配算力为 $\theta(p_A)$和 $\theta(p_B)$,并假设 $\theta(p_A)$的值很大,而 $\theta(p_B)$的值很小.并且假设任务在 p_A 预计完成的时间很短,而 p_B 预计完成的时间很长.因此,对于待编排容器列表 $C=\{c_1,c_2,\ldots,c_m\}$来说,如果有更多的容器被编排到 p_A,那么系统就能取得更好的计算时间性能.

令 Q 代表待编排的容器,并且每一个容器 c 都会请求 p_A 的服务,同时,$r(c)|c\in C$ 等同于 $v(q)|q\in Q$.由于 $d(u,p_A)$与 $d(u,p_B)$以及 $\theta(p_A)$与 $\theta(p_B)$之间的值相差很大,因此 p_A 是唯一可选的边缘计算平台,并且令它的闲置资源 $I(p_A)=B$.我们接下来证明,找到最优的容器编排策略等同于找到最大子集和问题的解决办法.最优的容器编排策

略是:找到一个容器的子集和 $C'\subseteq C$,让 $\sum_{c\in C'} r(c)\leqslant I(p_A)$,并且 $\sum_{c\in C'} r(c)$ 是最大的.因此,找到最优的容器编排策略与解决最大子集和问题是等价的. □

参考了文献[17]的做法,我们首先计算每个容器的优先级,并且用贪心的方式将每个容器编排到当前响应最快、同时满足资源需求的集群上.我们根据子任务信息计算出每个子任务对应的容器的 *score* 值(也称为优先级,它综合考虑了容器的数据依赖以及容器的执行时间等),并且按照 *score* 值的递减顺序形成一个容器列表.其中,*score* 值的计算方式如下:

$$score(c)=\overline{t_c}+\max_{c_m\in succ(c)}\{\omega(c_m,c)+score(c_m)\}.$$

平均计算时长越低,同时与其他节点通信代价越低的节点,它所在容器的 *score* 值就越高,并且 *score* 值越大,说明它应该先被执行.在容器编排的过程中,算法会依次根据容器列表依次调度每一个容器,并根据计算时间编排到响应最快的边缘计算集群.如果存在某个边缘计算集群的闲置资源无法满足容器的需要,那么算法会考虑响应时间第二快的计算集群,判断它是否能容纳和运行当前的容器.以此类推,直到找到满足计算要求的边缘集群为止.

基于启发式的容器编排算法具体细节见算法 1.

算法 1. *Heu-Orche*(*J*,*P*,*C*).

1. 初始化任务 *J* 的图结构,并将每个子任务对应到不同的容器;
2. 收集和计算每个容器的信息,并且计算它们的 *score* 值,并按照 *score* 值递减的顺序将它们进行排序,存于 $C=\{c_1,c_2,\ldots,c_m\}$中;
3. **While** 列表中存在未被编排的容器
4. 选出列表中的 *score* 值最小的第 1 个容器 *c*;
5. **For** $p\in P$
6. **If** $r(c)\leqslant I(p)$ **then**
7. 计算 *EFT*(*c*,*p*)值;
8. **EndFor**
9. 将 *c* 编排给 *EFT* 值最小的同时满足计算需求的集群;
10. **EndWhile**
11. 更新网络中的任务和闲置资源使用信息,并为下一轮编排做准备.

第 1 行、第 2 行是容器优先级计算过程.首先,在第 1 行,算法会先获取每个容器运行的计算量、资源需求等,以及在每个集群的预计开始执行时间和完成执行时间等.接着,在第 2 行,算法会根据信息计算每个容器的 *score* 值,并按照 *score* 值的递减顺序依次对容器进行排序,并形成列表 $C=\{c_1,c_2,\ldots,c_m\}$,而接下来算法也会按照该列表对它们进行调度.

接着是集群选择阶段.从第 3 行~第 10 行,算法不断选出列表中未被编排的容器计算出合适的编排策略,计算方式则是采用贪心的方式,直到待编排容器列表为空.在第 4 行,算法每次都会选出列表中第 1 个未被编排的容器 *c* 进行操作,并且会收集 *c* 的计算量和需求信息.从第 5 行~第 8 行,算法遍历所有可用的边缘计算集群,收集所有集群的资源使用信息、集群间的传输代价以及用户与集群的延迟信息.第 6 行、第 7 行,对于每个集群 *p*,算法首先会判断该集群当前是否满足容器的需求,也就是判断 *r*(*c*)是否小于等于 *I*(*p*):如果满足,算法会继续计算容器 *c* 编排到该集群的预计完成执行时间 *EFT*(*c*,*p*);如果不满足,则直接跳到下一个集群进行判断.在遍历完所有的集群后,算法会选择预计完成时间最小、并且满足容器计算需求的那个边缘计算集群,并把容器编排给它,并更新网络的信息.最后,所有容器就会被贪心地编排到合适地边缘计算集群,并在第 11 行更新网络的信息.

定理 2. 算法 1 的时间复杂度为 O(*n*|*E*|).

证明:在第 2 行,算法需要考虑子任务间的数据依赖以及每个子任务在边缘计算集群的平均计算时间,因此,第 2 行的时间复杂度为 O(*n*|*E*|).而从第 3 行~第 10 行为循环遍历过程,其时间复杂度为 O(*mn*).因此,算法 1 的时间复杂度为 O(*n*|*E*|). □

基于贪心策略的启发式容器编排算法能够充分考虑容器间的数据依赖关系以及每个容器的计算延迟和计算需求,为任务运行提供合理的容器编排顺序,是一种简单易用并且高效的编排算法,并且将控制器的计算负载维持在一个较低的水平,以适应大规模的用户数量.算法的时间复杂度为 $O(n|E|)$,即使在稠密图的情况下,算法的时间复杂度为 $O(nm^2)$.因此,随着用户数量和边缘服务器数量的增长,算法的执行时间也呈多项式时间增长的速度,具有较强的扩展性,能够很好地保证大规模用户下容器的编排效率和算法执行时间.

算法将运行在 FDN 控制器中.每当有用户提交任务请求时,FDN 控制器会先收集任务、集群和网络等重要信息,再执行容器优先级计算过程和集群选择阶段.它会将容器编排策略放进分发器,并按规则编排到目标边缘计算集群.DNS 也会根据编排策略,将用户与集群进行映射绑定.经过以上过程,用户提交的计算任务能够更合理地部署到边缘计算集群进行计算,优化任务的计算延迟,提高系统的计算效率.

4 FDN 系统实现、部署与测试

4.1 FDN实现细节

我们实现了 FDN 系统,包括了 FDN 控制器和各个边缘计算集群,系统架构如图 3 所示.对于 FDN 控制器,我们实现了客户管理后台(FDN-WEB)、FDN 分发器、中心网关和费用中心等功能,每部分的功能也如第 2 节描述.其中,我们使用了 RabbitMQ 作为消息队列代理软件,用于处理网络中用户发出的计算请求;同时,用户和容器等信息也都存放在数据库中,以进行费用和其他用户信息的统一管理.当消息队列不为空时,FDN 分发器就会依次处理队列中的消息,并找到代码库中对应的代码、容器等信息,进而将容器分发到对应的边缘计算集群上.对于 FDN 网关,我们也使用了 Nginx 作为服务器代理软件,用于进行负载均衡、安全管理和流量监控等网络状态管理.另外,我们在 FDN 系统中也部署了一系列高性能的边缘计算集群(我们以 3 个为例,如图 3 所示).对于系统中的边缘计算集群,我们安装了 K8S 和 Openwhisk 并进行了一定的改进,作为底层的容器编排工具,实现系统的容器化管理和基于函数的计算功能.每个边缘计算集群内部也会实现边缘网关,功能与控制器网关类似,用于维护边缘集群内部的正常运转.我们也使用并改进了 Prometheus 软件,将其作为分布式集群的管理工具,让多个边缘计算集群间协同工作.

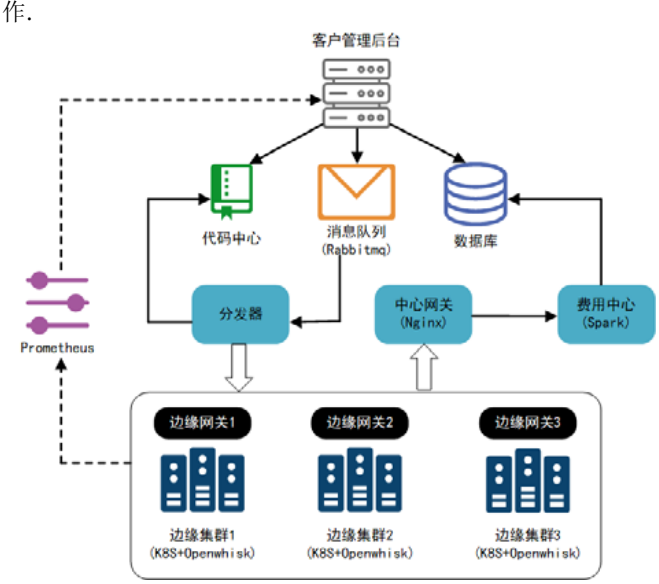


Fig.3 Implementation of FDN system

图 3 FDN 系统实现

4.2 部署设定

我们在中国移动网络中部署了 FDN 系统,如图 4 所示.

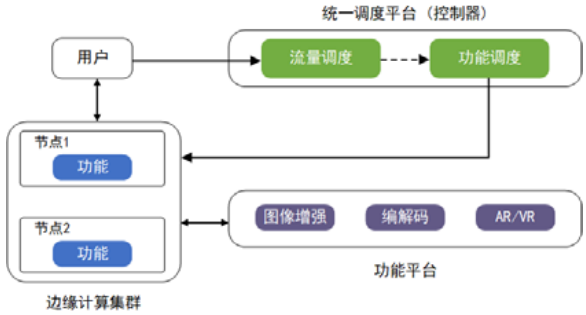


Fig.4 Deployment of FDN system

图 4 FDN 系统部署

我们将 FDN 控制器以及功能平台部署在杭州,同时在边缘端分别部署了两个性能相同的边缘计算集群,分别放置于北京和深圳.在功能中心中,我们准备了不同的常见容器镜像,例如图像增强、视频编解码、VR 应用等常见计算任务,为边缘节点提供软件支持.

在本实验中将以人脸识别作为功能测试,评估 FDN 的计算延迟性能.具体来说,我们在深圳部署了用户(客户端),通过 DNS 提供的统一接口,每隔 10s 向 FDN 平台提交计算量相同的人脸识别任务.同时,为了保证测试结果的准确性,我们令客户端请求过程持续大约 10 分钟,测试任务总数为 60 个,并统计任务的计算延迟.为了简单起见,两个边缘计算集群的初始状态都是一样的,包括闲置资源、网络流量以及各类硬件设备等.我们定义一个任务的计算延迟为该任务从上传到边缘服务器返回结果所需要的时间.我们将分别测试并比较两个边缘计算集群的计算延迟,以判断不同边缘计算集群对于任务计算延迟的影响.同时,我们将展示 FDN 控制器的资源使用情况,并用 CPU 利用率作为评估指标,以评估控制器计算编排策略时的计算负载情况.

4.3 测试结果

两个边缘计算集群的计算延迟如图 5 所示,其中,横轴是任务的索引,纵轴是任务的计算延迟,单位是 ms.从图中可以看到,位于深圳的边缘计算集群的计算延迟明显小于北京的计算集群.其中:深圳计算集群的平均计算延迟为 137.82ms;而北京计算集群的平均延迟为 392.16ms,超过了深圳计算集群平均延迟 65.1%.这是因为位于深圳的边缘计算集群距离用户更近,因此任务的计算延迟更小,而这些任务也被编排到深圳的边缘计算集群进行计算.这说明我们的 FDN 系统能够根据延迟等计算资源信息,将任务调度到更合适的集群,最小化任务的计算延迟.同时,对于用户来说,FDN 系统能大大提高复杂任务的计算速度,满足了许多应用实时性的需求.FDN 控制器的 CPU 资源利用率如图 6 所示,其中,横轴代表任务的索引,纵轴代表 CPU 的资源利用率并且均为百分数.

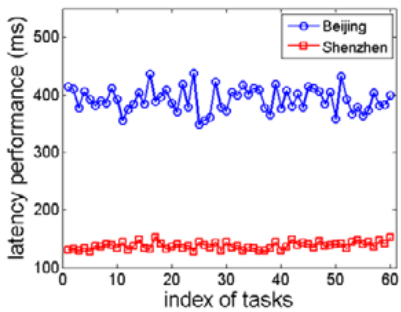


Fig.5 Latency performance of edge clusters

图 5 边缘计算集群的计算延迟

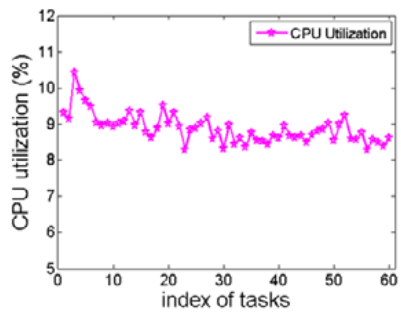


Fig.6 Utilization of FDN controller

图 6 控制器的 CPU 利用率

我们可以看到:位于杭州的 FDN 控制器的 CPU 资源利用率一直维持在稳定水平,其平均值为 8.87%.这说明我们的 FDN 控制器能够高效快速地对网络的信息进行收集,以及对任务所对应的容器编排策略进行快速的计算,同时保持较多的闲置资源,确保了 FDN 计算平台的稳定安全运行.

5 容器编排模拟实验

5.1 实验设定

我们实现并模拟了基于有向无环图结构的启发式容器编排算法(Heu-Orche),并评估了它在不同任务规模、用户数量、网络拓扑和设备算力下的计算延迟性能.我们首先生成了不同规模的有向无环图计算任务,每个计算任务中包含了不同数量的子任务,且每个子任务均封装为一个容器并编排到机器进行计算.我们随机模拟了每个子任务的计算规模和计算量,以及子任务之间的数据依赖和通信时长等必要信息.我们为每个计算任务模拟了 100~1000 个子任务,将容器编排到对应的计算集群,评估算法对不同任务规模的编排效果.

同时,为了评估不同的边缘计算网络环境,我们会改变网络的拓扑和规模,包括边缘计算集群的个数和集群的计算能力.在实验中,分布式边缘计算集群的数量会从 1 变化到 5,并随机分布在网络的不同位置.这些分布式边缘计算集群会根据控制器计算得到的容器编排策略,将容器部署到对应的机器,并根据容器的数据依赖关系开始并行计算,并最终将计算结果返回给用户.我们还会评估用户数对算法执行效率的影响,将用户数从 1 000 个变化到 10 000 个.默认情况下,子任务的数量为 100 个,边缘计算集群的个数为 3 个,用户数量为 1 000 个.

为了与其他编排算法进行对比,我们实现了在传统容器编排器中广泛采用的先来先服务算法(FCFS)和基于优先级(priority first)的编排算法[6–9].对于先来先服务算法,容器会根据先后顺序部署并运行在机器;而对于优先级编排算法,我们会根据子任务的规模顺序为容器进行优先级设定,并且让优先级更高的容器优先进行编排.另外,对于边缘计算的应用场景,我们开发了基于距离优先(distance first)的贪心编排策略,其中每个容器会被贪心地编排到距离最近的边缘计算集群.这种距离优先的调度策略广泛应用在边缘计算中[18],因此我们对其作了一定改进,设计了基于距离优先的容器编排策略.另外,我们还引入了最长剩余时间优先(longest remaining time first,简称 LRTF)算法.这是一种几年来广泛应用在容器编排的调度策略,容器会根据最长剩余时间进行抢占调度,以提高系统任务的计算延迟[19].我们将在同样的实验设定下,评估并比较这几种编排算法的计算延迟性能.在实验过程中,为了模拟更加真实的网络环境,我们为网络加入了 5ms 的网络抖动[20],并测试在该延迟抖动设定下的算法性能.

5.2 实验结果

1) 不同子任务与用户数量

我们首先测试了 5 个算法在不同数量的子任务(100~1 000)下和不同用户数量下(从 1 000 个~10 000 个用户)的编排性能,实验结果如图 7 和图 8 所示.

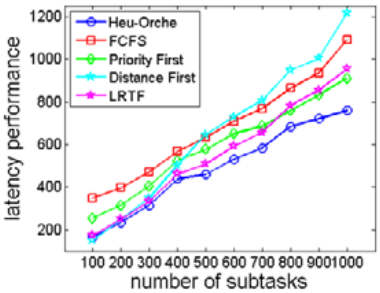


Fig.7 Computation latency with different workflows

图 7 不同子任务数量下的任务计算延迟

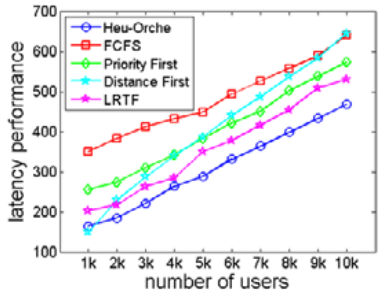


Fig.8 Computation latency with different users

图 8 不同用户数量下的任务计算延迟

在图 7 中,总体上,随着子任务数量的增加,5 个算法随着子任务数量的增加,任务的计算延迟都呈现上升趋

势.我们可以看到:传统的 FCFS、优先级编排算法和 LRTF 算法之间的性能差异不是很大,并且保持着相似的增长趋势.距离优先算法和 LRTF 算法在子任务数量较小时能够取得不错的效果,但当任务规模增加后,它们的性能缺陷显现出来了,其中,距离优先算法的性能甚至不如 FCFS 等传统的算法.我们提出的 Heu-Orche 的性能总体上优于其他 4 种算法,并且随着子任务数量的增加,它的性能优势就更大.例如:当子任务为 100 个时,Heu-Orche 的性能分别胜过 FCFS、优先级算法和 LRTF 算法 53.1%、35.7%和 5.49%;而子任务提高到 1 000 个时,Heu-Orche 的性能分别胜过它们 62.9%、15.1%和 25.5%.不同于那些简单的编排策略,Heu-Orche 能够考虑有向无环图的数据依赖关系以及计算集群的资源使用情况,因此它能够计算出一个更加合理的编排策略,优化任务的计算延迟.

而在图 8 中我们可以看到:当每个用户同时进行 1 000 个子任务的编排任务时,随着用户数量的增加,5 种算法的任务计算延迟基本上呈线性增长的趋势.并且 Heu-Orche 的性能随着用户数的增多,仍然保持较好的表现.相比 Kubernetes,Mesos,YARN,Borg 等传统容器编排器所使用的 FCFS 和优先级算法以及传统边缘计算中经常使用的距离优先算法和 LRTF 算法,我们提出的启发式算法能够更好地适应不同用户和流量规模,具有很强的拓展性,也能够取得更好的延迟性能,对基于有向无环图结构的计算任务有很大的优势.

2) 边缘集群数量

我们测试了 5 个算法在不同数量的边缘计算集群下(1 个~5 个)和不同数量的子任务(100 个子任务和 1 000 个子任务)的编排性能,实验结果如图 9 和图 10 所示.

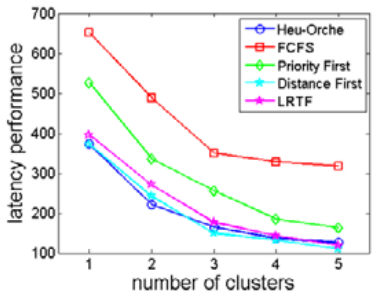


Fig.9 Computation latency with different clusters

图 9 不同边缘集群数量下的任务计算延迟

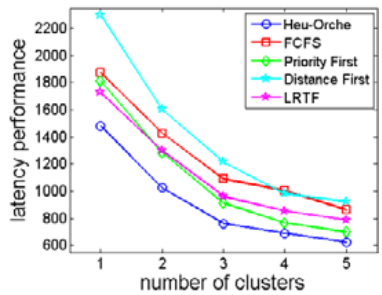


Fig.10 Computation latency with different clusters

图 10 不同边缘集群数量下的任务计算延迟

我们可以看到:随着边缘计算集群数量的增加,5 个算法的计算延迟都呈现下降的趋势;并且随着集群数量的增加,任务的计算延迟下降的幅度会减小,因为后面加入的边缘计算集群对于任务整体延迟的影响有限.对于 FCFS、优先级算法和 LRTF 算法,它们的变化趋势基本保持稳定和一致,而距离优先算法则出现很大的波动.例如:当子任务为 100 个时(如图 9 所示),距离优先算法总体上能够获得最优的性能,Heu-Orche 与距离优先之间的差距不是很大;而当子任务上升并固定为 1 000 个之后(如图 10 所示),距离优先算法的性能是最差的,而 Heu-Orche 依旧保持着很好的性能表现.这说明我们提出的启发式编排算法能够较好地适应不同的子任务和不同的边缘计算集群的数量和规模.

6 结 论

在本文中,我们提出了一种容器化的边缘计算平台 FDN.通过平台提供的统一接口,用户无需进行复杂的资源和环境配置就能够访问边缘计算资源,计算任务也能够被调度到最合适的边缘计算集群进行计算.同时,我们开发了一种基于启发式的跨集群容器编排策略,综合考虑了用户到集群的延迟、任务和集群闲置资源信息等,优化任务的计算延迟.我们在实际生产环境中实现并部署了 FDN 平台以及实验模拟了启发式的容器编排算法.实验结果表明:我们的 FDN 计算平台能够取得很快的计算延迟,同时,启发式的容器编排算法相比传统的编排策略有了较大的性能提升.

致谢 感谢华为技术有限公司基于图理论的网络优化算法研究项目 YBN2019125156 的支持.

References:

[1] Khan R, Kumar P, Jayakody DN, *et al*. A survey on security and privacy of 5G technologies: Potential solutions, recent advancements and future directions. IEEE Communications Surveys & Tutorials, 2020,22(1):196−248.

[2] Zhao J, Liang P, Liufu W, *et al*. Recent developments in content delivery network: A survey. In: Proc. of the Int'l Symp. on Parallel Architectures, Algorithms and Programming. Singapore: Springer-Verlag, 2019. 98−106.

[3] Shi WS, Zhang XZ, Wang YF, *et al*. Edge computing: State-of-the-art and future directions. Journal of Computer Research and Development, 2019,56(1):69−89 (in Chinese with English abstract).

[4] Qiu L, Jiang WX, Li YZ, *et al*. Trustworthy data collection method based on edge computing and trust value. Ruan Jian Xue Bao/Journal of Software, 2019,30(Suppl.(11)):71−81 (in Chinese with English abstract). http.//www.jos.org.cn/1000-9825/19008.htm

[5] Jonas E, Schleier-Smith J, Sreekanti V, *et al*. Cloud programming simplified: A Berkeley view on serverless computing. arXiv preprint arXiv: 1902.03383, 2019.

[6] Rattihalli G, Saha P, Govindaraju M, *et al*. Two stage cluster for resource optimization with Apache Mesos. arXiv preprint arXiv: 1905.09166, 2019.

[7] Kaur K, Garg S, Kaddoum G, *et al*. KEIDS: Kubernetes based energy and interference driven scheduler for industrial IoT in edge-cloud ecosystem. IEEE Internet of Things Journal, 2019.

[8] Yao Y, Gao H, Wang J, *et al*. New scheduling algorithms for improving performance and resource utilization in hadoop YARN clusters. IEEE Trans. on Cloud Computing, 2019.

[9] Verma A, Pedrosa L, Korupolu M, *et al*. Large-scale cluster management at Google with Borg. In: Proc. of the 10th European Conf. on Computer Systems. 2015. 1−17.

[10] Luo WP, Feng CS, Qin ZG, *et al*. Ciphertext sharing scheme for the public cloud. Ruan Jian Xue Bao/Journal of Software, 2019,30(8):2517−2527 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5468.htm [doi: 10.13328/j.cnki.jos.005486]

[11] Xian HQ, Liu HY, Zhang SG, *et al*. Verifiable secure data deduplication method in cloud storage. Ruan Jian Xue Bao/Journal of Software, 2020,31(2):455−470 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5628.htm [doi: 10.13328/j.cnki.jos.005628]

[12] Hassan N, Yau KL, Wu C. Edge computing in 5G: A review. IEEE Access, 2019,30(7):127276−127289.

[13] Liu Y, Yang C, Jiang L, *et al*. Intelligent edge computing for IoT-based energy management in smart cities. IEEE Network, 2019, 33(2):111−117.

[14] Qi YL, Zhou YQ, Liu L, *et al*. MEC coordinated future 5g mobile wireless networks]. Journal of Computer Research and Development, 2018,55(3):478−486 (in Chinese with English abstract).

[15] Teixeira FA, Pereira FM, Wong HC, *et al*. SIoT: Securing Internet of things through distributed systems analysis. Future Generation Computer Systems, 2019,92:1172−1186.

[16] Abboud A, Bringmann K, Hermelin D, *et al*. SETH-based lower bounds for subset sum and bicriteria path. In: Proc. of the 30th Annual ACM-SIAM Symp. on Discrete Algorithms. Society for Industrial and Applied Mathematics, 2019. 41−57.

[17] Arabnejad H, Barbosa JG. List scheduling algorithm for heterogeneous systems by an optimistic cost table. IEEE Trans. on Parallel and Distributed Systems, 2013,25(3):682−694.

[18] Yi S, Hao Z, Zhang Q, *et al*. Lavea: Latency-aware video analytics on edge computing platform. In: Proc. of the 2nd ACM/IEEE Symp. on Edge Computing. 2017. 1−13.

[19] Chen W, Zhou X, Rao J. Preemptive and low latency datacenter scheduling via lightweight containers. IEEE Trans. on Parallel and Distributed Systems, 2019.

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

附中文参考文献:

[3] 施巍松,张星洲,王一帆,等.边缘计算:现状与展望.计算机研究与发展,2019,56(1):69−89.

[4] 邱磊,蒋文贤,李玉泽,等.基于边缘计算与信任值的可信数据收集方法.软件学报,2019,30(Suppl.(11)):71−81. http://www.jos.org.cn/1000-9825/19008.htm

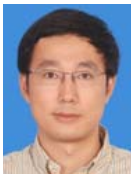
[10] 罗王平,冯朝胜,秦志光,等.一种面向公有云的密文共享方案.软件学报,2019,30(8):2517−2527. http://www.jos.org.cn/1000-9825/5468.htm [doi: 10.13328/j.cnki.jos.005486]

[11] 咸鹤群,刘红燕,张曙光,等.可验证的云存储安全数据删重方法.软件学报,2020,31(2):455−470. http://www.jos.org.cn/1000-9825/5628.htm [doi: 10.13328/j.cnki.jos.005628]

[14] 齐彦丽,周一青,刘玲,等.融合移动边缘计算的未来 5G 移动通信网络.计算机研究与发展,2018,55(3):478−486.

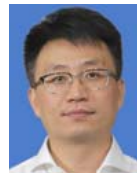
杨术(1988－),男,博士,副研究员,CCF 专业会员,主要研究领域为计算机网络体系结构,边缘计算,物联网.

陈子腾(1996－),男,硕士生,主要研究领域为边缘计算,区块链,物联网.



崔来中(1984－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为计算机网络体系结构,边缘计算,物联网.

明中行(1987－),男,博士,副研究员,CCF 专业会员,主要研究领域为计算机网络体系结构,边缘计算,物联网.



程路(1980－),男,博士,主要研究领域为计算机网络,大数据.

唐小林(1978－),男,硕士,主要研究领域为计算机网络,企业数字化,大数据.

萧伟(1981－),男,博士,副研究员,主要研究领域为人工智能,边缘计算,物联网.