无服务器计算平台冷启动优化的研究综述

洪涛

华中科技大学计算机科学与技术学院

摘要 无服务器(Serverless)计算正成为部署云应用程序的一种具有广阔发展前景的范式。其实现了一种真正的所付即所用的计费方式，将对资源的浪费降到了最低。无服务器计算允许开发人员将传统的单块服务器应用程序分解为更细粒度的云函数，实现了构建和扩展应用程序和服务的新方法。开发人员只需编写函数逻辑，把供应、扩展和管理运行函数的后端服务器等极其乏味的任务交给云服务供应商，从而降低了开发人员的门槛。无服务器模式的转变虽然带来了很多机遇，但也引入了容器函数冷启动耗时、资源利用不足等问题。为此，本文中对无服务器计算平台冷启动耗时优化技术做了调查和分析，重点阐述了轻量级虚拟机技术以及容器的快速部署技术原理和相关研究现状。例如基于 type-1 hypervisor 的轻量级虚拟化系统 LightVM，对 Xen 架构进行了全面改革，引入 unikernel 和 Tinyx 以减少虚拟机镜像大小和虚拟机内存占用；以及基于 KVM 的轻量级虚拟化系统 Firecracker，其完全取缔了 QEMU 并建造了新的超轻量的 VMM（virtual machine manager）、设备模型以及 API 来管理和配置 MicroVMs；使用 P2P 网络架构实现容器镜像分发的 Kraken，引入新的中间件 FT 来实现自适应管理容器，体高容器扩容速度的 FaaSNet。此基础上分析总结并指明了无服务器平台冷启动优化未来的主要研究方向：即虚拟化技术升级与优化以及容器镜像分发与管理算法优化。

关键词 无服务器计算；函数即服务；容器；轻量级虚拟机；镜像分发；云基础设施；

Review of Research on Cold Start Optimization of Serverless Computing Platforms

Tao Hong

Department of Computer Science and Technology, HuaZhong University of Science and Technology

Abstract Serverless computing is emerging as a promising paradigm for deploying cloud applications. It really implements pay-as-you-go billing with least resources waste. Serverless computing enables a new way of building and scaling applications and services by allowing developers to break traditionally monolithic server-based applications into finer-grained cloud functions. Developers write function logic while the service provider performs the notoriously tedious tasks of provisioning, scaling, and managing the backend servers that the functions run on, thus lowering the threshold for developers. Although the change of serverless model brings opportunities, it also brings problems such as cold startup delay of platform function and insufficient resource utilization. For example, LightVM, a lightweight virtualization system based on type-1 hypervisor, has comprehensively reformed the Xen architecture and introduced unikernel and Tinyx to reduce the virtual machine image size and virtual machine memory footprint; And Firecracker, a lightweight virtualization system based on KVM, has completely banned QEMU and built a new ultra-lightweight VMM (Virtual machine manager), and API to manage and configure MicroVMs; Kraken, which uses P2P network to implement container device deployment. Using new middleware-FT to implement model management containers, and FaaSNet with high container scaling speed. Summarized and explained the main research

directions of serverless platform cold start optimization in the future: optimization and update of virtualization technology, optimization of container images distribution and management algorithms.

Key words Serverless computing; Function as a service; Lightweight Virtual Machines; Image distribution; Cloud Infrastructure;

1 引言

近年来，云计算，特别是基础设施即服务提供的虚拟机技术已经被各行各业广泛接受并采用，企业越来越多地采用云的主要因素是它的现收现付模式，即客户只需为向云提供商租用的资源付费，并且能够获得所需的任意规模的资源。然而，数据中心[1]报告中云资源使用情况的研究表明：云客户支付的资源（租用虚拟机）与实际资源使用（CPU、内存等）之间存在巨大差距，这造成了资源的浪费，增加了云客户的成本。无服务器（Serverless）计算的诞生为解决这一问题提供了思路，其正成为部署云应用程序的一种新颖范例。云原生计算基金会（CNCF）[2]给出了无服务器计算的权威定义：无服务器计算描述了一种更细粒度的部署模型，其中包含一个或多个函数的应用程序被捆绑到平台上，然后应用程序根据当前的确切需求执行：伸缩和计费。无服务器计算实现了一种真正的现收现付(以毫秒为粒度)的计费方式，有效地节约了资源，并将所有的操作复杂性委托给云提供商，从而降低了开发人员的门槛。由于其研发交付速度和成本的优势，无服务器计算越来越受欢迎，Amazon、IBM、Microsoft、谷歌、腾讯和阿里巴巴等已经发布了无服务器计算能力的云供应商，以及由行业和学术机构共同推动的一些开源库。

无服务器计算使得开发人员无需担心可用性，可伸缩性、容错性、虚拟机资源的供应、服务器管理和其他基础设施问题，而是专注于应用程序的业务方面。但是，在无服务器计算中，提供商必须决定如何调度用户传入的函数，其中一个难题就是用户应用程序集的多样性。无服务器平台的优势在于其不需要明确的配置、自动且几乎无限的扩展，并按使用计费，因此吸引了不同应用社区的兴趣。调度的另一个难点是如何利用容器来托管应用程序。虽然容器有助于打包应用程序并简化函数调用过程，但它们确实会带来不良的性能影响。在大多数函数即服务(Functions as a Service, FaaS)平台上，函数首先会注入容器中，然后该容器将运行在虚拟机或物理机器上。如果没有请求函数的实例，FaaS 平台将派生出新的容器，并注入函数，配置依赖项，这称为冷启动延迟[3]。函数在容器中的冷启动可能会影响用户的服务水平协议(Service Level Agreement, SLA)。理想情况下，云应用程序的所有者希望维护用户的 SLA，从而不会导致更高的成本或增加系统复杂性。

因此，本文从轻量级虚拟机技术和云基础设施快速部署技术两个角度来讨论降低冷启动耗时问题。虚拟机和容器是云计算平台常用的虚拟化技术，但是它们都存在一定的不足，虚拟机技术虽然可以很好的保证隔离性、安全性以及兼容性，但是由于其规模较大，对物理主机要求较高，很难满足无服务器计算快速扩容的要求。对于容器而言，虽然其规模比虚拟机要小，启动过程也比虚拟机快，但是在新的主机上拉取容器镜像也是一个非常耗时过程，同时容器天生的安全性短板也让其成为无服务器计算多租户条件下必须要解决的问题。而轻量级虚拟机是在保持基本虚拟化功能的条件下，尽可能减少 VM 的规模以及运行时所占用的内存空间，在保证安全性和兼容性的同时，也加快了 VM 部署的速度，甚至超过了一般 Docker 容器的部署速度，大大减少了冷启动的耗时。比如对 Xen 进行改进，引入 unikernel 和 Tinyx 以减少虚拟机镜像大小和虚拟机内存占用的 LightVM[4]，Linux 内核 KVM 的虚拟化设施来提供最小化虚拟设备的 Firecracker[5]，都是在这方面进行努力。在容

器部署过程中，其实最耗时的部分是容器镜像的拉取过程，通过更改镜像分发所使用的网络架构或容器供应机制可以一定程度改变这一现状。比如大致基于 BitTorrent 协议的开源 P2P 容器仓库 Kraken[6]，以及加速无服务器容器供应的基于函数树的去中心化高拓展性的中间件系统 FaaSNet[7]，其更近一步消除了现有的容器分发技术依靠强大的根节点来服务一系列任务，包括数据播种、元数据管理和 P2P 拓扑管理的问题。本文接下来内容主要是分析介绍这些技术的原理和优势，以及这些技术当前的研究现状，并在文末做出总结和展望。

2 原理和优势

2.1 传统的虚拟机与容器

2.1.1 虚拟机

虚拟机是指运行在 Windows 或 Linux 计算机上的一个应用程序，这个应用程序模拟了一个基于 x86 的标准 PC 的环境。这个环境和普通的计算机一样，都有芯片组、CPU、内存、显卡、声卡、网卡、软驱、硬盘、光驱、串口、并口、USB 控制器、SCSI 控制器等设备，提供这个应用程序的“窗口”就是虚拟机监视器（VMM）。在一台电脑上将硬盘和内存的一部分拿出来虚拟出若干台机器，每台机器可以运行单独的操作系统而互不干扰，这些“新”机器各自拥有自己独立的 CMOS、硬盘和操作系统，你可以像使用普通机器一样对它们进行分区、格式化、安装系统和应用软件等操作，还可以将这几个操作系统联成一个网络。在虚拟系统崩溃之后可直接删除不影响本机系统，同样本机系统崩溃后也不影响虚拟系统，可以下次重装后再加入以前做的虚拟系统。同时它也是唯一的能在 Windows 和 Linux 主机平台上运行的虚拟计算机软件。虚拟机软件不需要重开机，就能在同一台电脑使用好几个 OS，不但方便，而且安全。

虚拟机技术虽然提供了强大的隔离性，但是虚拟化毕竟是在硬件层之上进行了封装，相比直接基于物理机，必然会损失一部分性能。从之前的经验来看，当一台物理机上并行运行多个虚拟机时，物理机资源的使用率越高，虚拟机性能下降的越剧烈。主要有两个原因：

(1) CPU 的限制，若 VM 配置了多核 CPU，当 VM 内应用要求多核同时处理某一任务时，此时物理机若只有少于所需的核数空闲，那么 VM 会锁住可用的核再去等待其他的核释放。若虚机较多并且物理机 CPU 很繁忙，此操作会造成类似于死锁的现象，导致表象性能急剧下降。近几年各大厂商在此方面做了很多的优化，该 bug 有很好的改进。

(2) IO 的限制，通常的 IO 是直接写在物理硬盘上，而 VM 的 IO 则是先写在 host 系统的虚拟机镜像文件上，再由 host 以某种策略写入物理硬盘。一来这种二段式的写入会造成读写的延迟，二来当计算部署的虚机数量时，硬盘的 IOPS 和多机造成的随机写方式，往往是非常容易忽略的一个指标。例如，7200 转的 SATA 盘理论 IOPS 是 76，一般 windows 工作时的 IOPS 是 10~30，因此理论上一块 SATA 盘只能支持 2~3 个 VM。当虚拟机数量超配后，多个 VM 繁忙时会造成所有 VM 处于 io_wait 的状态。

2.1.2 容器

容器是一种轻量级虚拟化技术，基于容器的虚拟化技术可以将物理资源抽象集中，逻辑划分，重新分配为虚拟实体。虚拟实体必须满足隔离性，包括用户隔离 (或者说权限隔离) 、进程隔离、网络隔离、文件系统隔离等，即虚拟实体只能感知其内部的资源，并且自以为是独占整个资源空间,它既不能感知其所在宿主机的真实资源，也不能感知其他虚拟实体的资源。实现容器技术需满足两个关键条件：一是资源隔离，二是资源控制。

（1）资源隔离

Linux 主要是通过内核提供的 namespace 技术，实现对 UTS、IPC、Mount、PID、Network、User 等的隔离。namespace 技术的目的之一就是实现轻量级虚拟化（容器）服务，在同一个 namespace 下的进程可以感知彼此的变换，而对外界的进程一无所知，这样可以让容器进程产生错觉，仿佛自己置身于一个独立的系统缓解中，以达到独

立和隔离的目的。

（2）资源控制

Linux 通过 cgroups（control groups）实现资源控制。简单地说，cgroups 可以限制、记录任务组所使用的物理资源（包括 CPU、memory、I/O 等），为容器实现虚拟化提供基本保证。资源控制为虚拟实体分配一定量的资源，虚拟实体得到所分配的资源，不能超出资源的最大使用量。cgroups 的主要目的是为了不同用户层面的资源管理，包括从单个任务的资源控制到操作系统层面的虚拟化，提供一个统一化的接口。

对于传统的容器而言，安全性一直是很大的问题。罪魁祸首是强大的内核系统调用 API，容器使用该 API 与主机操作系统进行交互。这个 API 非常广泛，因为它提供了内核对进程和线程管理、内存、网络、文件系统、IPC 等的支持：例如，Linux 有 400 个不同的系统调用，大多数具有多个参数，许多具有重叠的功能；此外，系统调用的数量也在不断增加。系统调用 API 从根本上来说比虚拟机提供的相对简单的 x86 ABI 更难保护，因为虚拟机提供的内存隔离(带硬件支持)和 CPU 保护环就足够了。尽管在过去的几年里引入了许多隔离机制，如进程和网络名称空间、根监狱、seccomp 等，容器仍然是不断增加的攻击目标。更复杂的是，任何能够垄断或耗尽系统资源的容器(如内存、文件描述符、用户 id、forkbomb)都会导致该主机上所有其他容器受到 DoS 攻击。

2.2 轻量级虚拟机

对于多租户部署来说，这让云服务提供商面临一个艰难的权衡：无服务器和容器服务的实现可以选择基于 hypervisors 的虚拟化技术（QEMU/KVM，可能会导致难以接受的过载）或者 Linux 容器技术（LXC，可能会导致相关的兼容性和安全性之间的权衡）。引入轻量级虚拟机技术就是为了避免做这种选择。

轻量级虚拟机技术就是在 Hypervisor 的技术基础上提供轻量级的虚拟化服务，同时要实现一些容器的典型特征，比如快速启动和高实例化密度。实现上述设计目标，首要是缩小虚拟机的镜像大小和运行时内存的大小。容器就是由于较小的 root 文件系统且共享系统内核，其实例所占用的内存也相当小，由此能实现高实例化密度。此外，虚拟机镜像越大，将其加载入内存就需要更长的时间，拖慢了启动速度。通过观察发现，多数 VM 和 Container 都是运行单应用的，若将 VM 的功能裁剪至刚好仅能满足所运行应用的需求，将可以极大地降低 VM 的内存占用。现有的轻量级虚拟机实现保留了 Hypervisor 基本的虚拟化功能部分，并通过自定义轻量级的虚拟机镜像来实现与用户的交互，既保留了虚拟机强隔离性、高安全性、高兼容性的特点，也加快了虚拟机的部署速度，满足无服务计算面对突发流量所要求的高可拓展性。

2.3 云基础设施快速部署

在容器启动的过程中，拉取镜像占了启动时间的 76%，但只有 6.4%的数据被读取。所以加快容器镜像拉取速度是实现云基础设施快速部署的关键。传统的镜像获取方式就是每个新加入的主机向镜像仓库发起请求，分别下载函数所需要的镜像，使用映像的序列化复制，部署数千个虚拟机实例将耗费大量时间。这种延迟在大多数环境中是不可接受的，因为这些环境是按需提供的，或者解决方案映像文件太大(以 GB 为单位)。同时，在耗时的全映像交付之前，客户请求的虚拟机和服务通常无法在限定的时间内启动，导致服务可用性低下。

BitTorrent 作为一个流行的 P2P 应用程序，实现简单，BitTorrent 被证明能够快速、高效地分发大文件，是在配置云基础设施中加速镜像部署的合理选择。在此基础上，研究人员进一步探讨了如何将其用于云供应，并研究了与 FUSE 系统集成的改进 BitTorrent 的系统性能。提出了一种 P2P 辅助的云供应方法，以实现前所未有的部署速度和可伸缩性。基本上，服务/解决方案的虚拟机映像文件被描述为 BitTorrent(BT)种子文件，它被发送到目标物理机，以便在其上提供云基础设施。通过共享所有机器的上传能力，克服传统的发放服务器带宽瓶颈，同

时充分利用高速的企业内部网，预计将大大加快云发放的速度。在原型系统上的实验证明，研究人员提出的 p2p 辅助云配置在节省部署时间、提高部署可扩展性和服务可用性、降低流量成本等方面具有较好的配置性能。

另一种方式就是通过改变容器供应机制，引入新的中间件来自适应管理容器。比如使用一个叫做 Function Tree（FTs）的抽象来支持大规模的高效容器供应，一个 FT 是一个二叉树，连接多个 Host VM 来形成一个快速的可扩展的容器供应网络。在 FT 的设计中，（1）每个函数具有单独的 FT，也就是说，中间件可以在函数粒度上管理 FTs。数据平面与控制平面解耦，（2）FT 中的每个 VM worker 都负责一个等价的的、单一的职责即容器供应(数据平面)，以及对于调度器的全局树管理(控制平面)。（3）FT 的平衡二叉树的结构能够动态的适应工作负载，每个代表中间节点的 VM 最多只有两个子节点通过它获取镜像。最终实现最小化容器镜像的 I/O 负载和后台注册表的镜像数据下载量、消除中心根节点的树管理瓶颈和数据播种瓶颈以及适应虚拟机动态加入与离开。在函数需要扩容时，引入新的镜像分发机制可以大大加快容器基础设施的部署速度，降低冷启动延迟。

3 研究进展

3.1 轻量级虚拟机

虚拟机可以像容器一样灵活，只要它们足够小并且工具栈足够快。研究人员经过研究提出了 LightVM 的概念，这是一种基于 Xen 的新的虚拟化解决方案，经过优化后，无论活动虚拟机的数量如何，都能提供快速的启动时间。LightVM 的特点是对 Xen 的控制平面进行了彻底的重新设计，将其集中式操作转变为分布式操作，从而将与管理程序的交互减少到最低限度。LightVM 可以在 2.3ms 内启动一个 VM，相当于 Linux 上的 fork/exec (1ms)，比 Docker 快两个数量级。LightVM 可以在内存和 CPU 使用量与进程相当的中等硬件上打包数千个 LightVM 客户机。

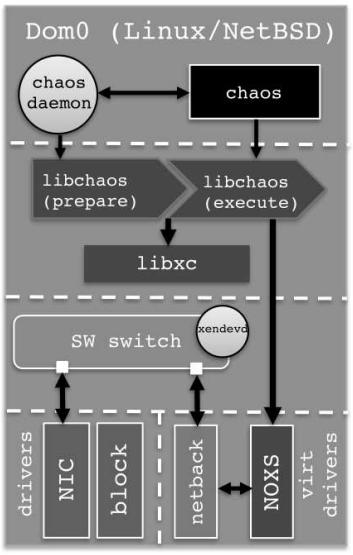

图 1 LightVM 架构图，显示了 noxs、xl 的放置(chaos)，分割工具栈和伴随守护进程，以及负责快速添加虚拟接口到软件交换机的 xendevd

LightVM, 这是一个基于 type-1 Hypervisor 的轻量级虚拟化系统。LightVM 保留了虚拟机的强隔离特性，同时提供了如容器那般具有吸引力的性能。研究人员分析阻碍传统虚拟化系统实现类似容器的动态的性能瓶颈，对 Xen 架构进行了全面改革，完全删除了它的后端注册表(即 XenStore)，这是一个性能瓶颈，研究人员称之为 noxs(没有 XenStore)，它的实现对引导和迁移时间以及其他指标有显著的改进。同时对 Xen 的工具栈进行了改进，包括一些优化，以及引入了一个分割工具栈，将可以定期运行的功能与必须执行的命令(如创建虚拟机)分离开来，其架构图如图 1 所示。Tinyx 的开发，一个用于构建基于 linux 的极简虚拟机的自动化系统，以及许多单内核的开发。这些轻量级虚拟机是实现高性能的基础，也是发现底层虚拟化平台的性能瓶颈的基础。一个典型的实现和广泛的性能评估表明，LightVM 能够在 2.3ms 内启动一个(单内核)虚拟机，达到多达 8000 个虚拟机的同一主机虚拟机密度，迁移和挂起/恢复时间分别为 60ms 和 30ms/25ms。每个虚拟机的内存占用只有 480KB(磁盘上)和 3.6MB(运行中)。为了展示它的适用性，研究人员使用 LightVM 实现了四个用例：个性化防火墙、

即时服务实例化、高密度 TLS 终止和一个轻量级计算服务，类似于 Amazon Lambda 或谷歌的云功能，但基于 Python 单内核。

XenStore 的一个基本问题是其集中的类似文件系统的 API，在虚拟机创建和引导期间使用速度太慢，需要进行数十次中断和权限域转换。与此相比，fork 系统调用只需要一个软件中断即一个用户内核交叉。要实现毫秒级的启动时间，研究人员需要的不仅仅是对现有 Xen 代码库的简单优化。LightVM，这是对基本 Xen 控制平面的完全重新设计，经过优化可以提供轻量级虚拟化，LightVM 的结构图如下所示。LightVM 不再使用 XenStore 来创建和引导虚拟机，取而代之的是一个叫做 noxs 的精益驱动程序，它通过允许前端和后端驱动之间通过共享内存直接通信来解决 XenStore 的可伸缩性问题，而不是通过 XenStore 来传递消息。因为 noxs 不依赖于消息传递协议，而是依赖于映射在客户地址空间中的共享页面，从而减少了虚拟机操作(创建/保存/恢复/迁移/销毁)所需的软件中断和域交叉的数量。LightVM 提供了一个分割工具栈，将 VM 创建功能分割为准备阶段和执行阶段，减少了 VM 创建的工作量。研究人员还实现了 chaos/libchaos，这是一个新的虚拟化工具栈，它比标准的 xl/libxl 要精简得多，此外还有一个名为 xendevd 的小守护进程，它可以快速地向软件交换机添加虚拟接口，或者处理块设备映像的设置。

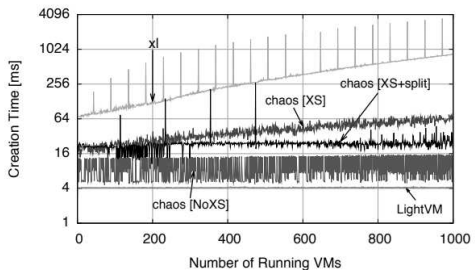

图 3 对于 LightVM 机制的所有组合，最多 1000 个白天单内核实例的创建时间。“xl”表示没有优化的标准 Xen。

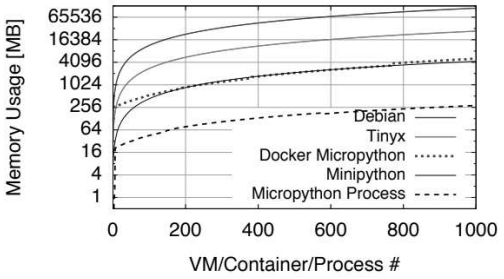

图 4 虚拟机、容器、进程虚拟机内存的使用的可扩展性

一个典型的实现和广泛的性能评估表明，如图 3 实验结果所示，LightVM 能够在 2.3ms 内启动一个(单内核)虚拟机，达到多达 8000 个虚拟机的同一主机虚拟机密度，迁移和挂起/恢复时间分别为 60ms 和 30ms/25ms。如图 4 实验结果所示，每个虚拟机的内存占用只有 480KB(磁盘上)和 3.6MB(运行中)。

此外轻量级虚拟机技术最新的进展是 Firecracker 的 microVM 技术，Firecracker 是一个 VMM，使用 Linux 内核 KVM 的虚拟化设施来提供最小化虚拟设备（MicroVMs），支持现代的 Linux 主机、Linux 和 OSv 的客户机。Firecraker 提供了基于 REST 的配置 API，硬盘的设备仿真，网路和串行控制台，网络和磁盘吞吐量和请求率的可配置速率限制。一个 MicroVM 运行一个 Firecraker 进程，提供一个简单的安全模型。Firecraker 的实现基于了很多现有的组件，包括 Linux 中的块 IO、进程调度器、内存管理器，以及谷歌的 crosvm；Firecracker 支持的仿真设备比 QEMU 少的多，同时为了安全性，Firecraker 选择支持块设备作为存储，而不是通过文件系统。REST APIs 的存在是为了定制客户内核、启动参数、网络配置、块设备配置、客户机配置以及 cpuid、logging、metrics、rate limiters 和元数据服务。

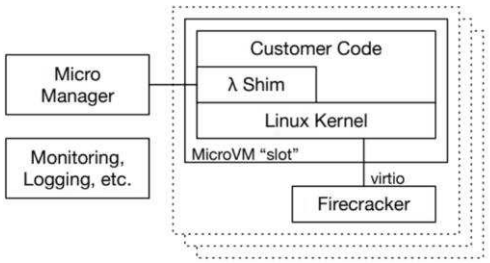

图 5 Lambda worker 的架构图

图 5 展示了 Lambda worker 的架构图，Firecracker 提供了关键的安全边界来保证可以在单个服务器上运行很多不同的工作负载。每个 Worker 运行成百上千的 MicroVMs（每个 MicroVMs 提供一个 slot），具体的数量取决于每个 MicroVM 配置的大小，每个 VM 消耗的 CPU、内存和其他资源。每个 MicroVM 为了客户的函数都包含了一个简单的沙盒，搭配最小化的 Linux 内核和用户区以及一个垫片控制进程。每个 MicroVM 中的垫片控制进程通过 MicroVM 的边界使用 TCP/IP 套接字来和 Woker Manager 通信，负责控制 Firecracker 进程。

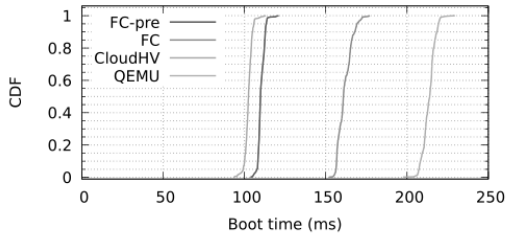

图 6 在 FC-pre、Firecracker、Cloud Hypervisor、QEMU 不同情况下顺序启动 microVMs 的累计时间

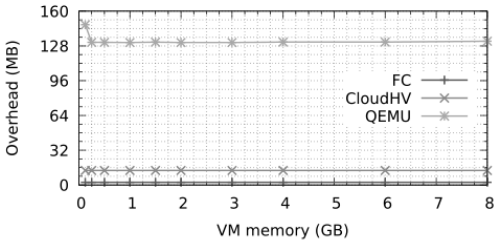

图 7 不同 VMM 下的内存负载

研究人员启动 1000 个 microvm，同时启动 50 个 vm。尽管任何高度并发的测试都会在不同的运行中给出不同的结果，但图 6 中的结果是具有代表性的：Firecracker 和 QEMU 在端到端比较方面的表现类似，大致保持了前面的串行启动时间中所看到的 50ms 间隔。与连续启动时间一样，预配置的 Firecracker 比 Cloud Hypervisor 显示出明显更好的结果，预配置的 Firecracker 在平均启动时间和更紧凑的分布上都优于 Cloud Hypervisor，前者的平均启动时间为 146ms，而后者为 158ms。

图 7 显示，不管配置的 VM 大小如何，所有三个 vmm 都有一个固定的内存开销(除了 QEMU，对于 128MB 大小的 VM,开销略高一些)。在所有测量的 VMM 尺寸中，Firecracker 的开销最小(约 3MB)。Cloud Hypervior 每个虚拟机的开销约为 13MB。QEMU 的开销最高，大约为每个 MicroVM 131MB。特别是对于较小的 microvm, QEMU 的开销是显著的。我们还使用 free 命令测量了 MicroVM 中的可用内存。不同 vmm 之间可用内存的差异与上面给出的数据一致。

3.2 云基础设施快速部署

为了提供云计算服务，新的数据中心需要首先实现云基础设施，即系统和应用发放，数据中心级虚拟化，以及支持极度数据密集型的工作负载。但是，在大量的物理机器(例如数千台)上配置这样的基础设施是一个耗时和费力的过程。通常，在为企业数据中心提供服务时，如图 8 所示，云供应服务器负责分发预配置的解决方案映像文件，并为企业内部网中的每个目标物理机器设置相应的环境(即硬件和所需的中间件)。使用映像的序列化复制，部署数千个虚拟机实例将耗费大量时间。这种延迟在大多数环境中是不可接受的，因为这些环境是按需提供的，或者解决方案映像文件太大(以 GB 为单位)。同时，在耗时的全映像交付之前，客户请求的虚拟机和服务通常无法在限定的时间内启动，导致服务可用性低下[6]。

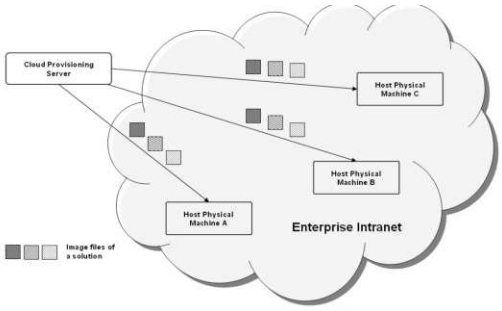

图 8 传统网络架构

研究人员提出了一种p2p辅助的云供应方法，以实现前所未有的部署速度和可伸缩性[7]。基本上，服务/解决方案的虚拟机映像文件被描述为BitTorrent(BT)种子文件，它被发送到目标物理机，以便在其上提供云基础设施。通过共享所有机器的上传能力，克服传统的发放服务器带宽瓶颈，同时充分利用高速的企业内部网，预计将大大加快云发放的速度。在实现过程中，采用了消息驱动的片选择方案，将改进后的 BitTorrent 与基于 FUSE 的虚拟机供应系统相结合，充分利用了 BitTorrent 加速下载和 FUSE 提高服务可用性的优势。在原型系统上的实验证明，研究人员提出的 P2P 辅助云配置在节省部署时间、提高部署可扩展性和服务可用性、降低流量成本等方面具有较好的配置性能。

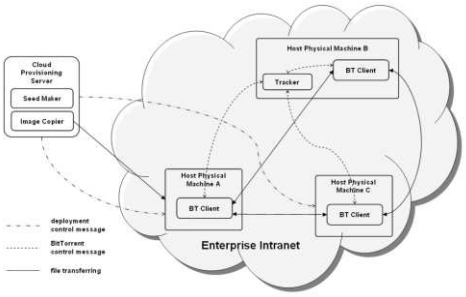

图 9 P2P 网络架构

系统概述如图 2 所示。此项研究的目标场景是将解决方案映像从 Internet 上的云供应服务器部署到企业内部网的所有主机物理机器上。在将解决方案映像包的一个副本及其对应的 BitTorrent 种子文件部署到选定的物理机器之后，使用类似于 BitTorrent 的 P2P 协议将解决方案映像发送到企业内部网中的其他物理机器。此外，Cloud Provisioning Server 的框架如图 10 所示。系统中的核心组件是供应管理器，它处理供应逻辑。支持供应逻辑的组件包括物理机器管理器，解决方案映像管理器，种子制造器，跟踪器选择器，BT 客户端安装程序，BT 跟踪器安装程序，和镜像复制器。Physical Machine Manager 负责从数据库中获取主机物理机信息的 CRUD(创建、检索、更新和删除)操作。解决方案镜像管理器负责解决方案镜像信息的 CRUD 操作，以及带数据库和本地文件系统的文件。种子生成器可以在将解决方案添加到物理供应服务器时自动生成种子。跟踪器选择器处理企业内部网中合适跟踪器的选择。BT Tracker Installer 和 BT Client Installer 可以安装 BT Tracker 和客户端到目标主机物理机器。此外，Image Copier 负责使用适当的传输协议将镜像文件复制到目标主机物理机。

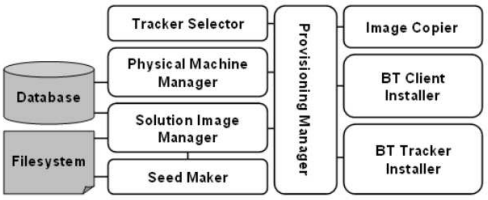

图 10 云提供服务器的架构

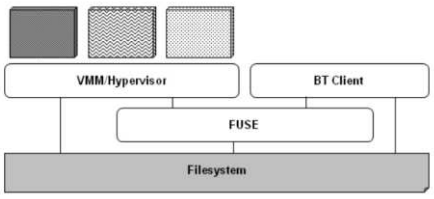

Fig.4. Framework of Host Physical Machine.

图 11 主物理机的架构

对于客户端，主机物理机框架如图 11 所示。除了普通物理机中存在的 VMM /Hypervisor 和本地文件系统外，系统还添加了 FUSE (Filesystem In Userspace)和 BT Client。FUSE 是一种提供功能性文件系统的操作系统插件，可以拦截操作系统用户的文件操作，判断该文件是否存在于本地文件系统中。如果没有，可以调用 BT Client 从其他 BT Client 获取图像文件。

表 1 像下载性能总结

Peer Domain	Downloaded Bytes	Downloading Time	Uploaded Bytes	Share Ratio	Average Downloading Speed
Group A	12G	35m16s	6.25G	0.521	5.81MB/s
Group B	12G	40m26s	7.74G	0.645	5.07MB/s
Group C	12G	63m40s	6.53G	0.529	3.22MB/s

采用 P2P 技术后，镜像的传输速度可以达到 5M/sec，一般情况下，将一个 12G 的解决方案镜像分发到所有目标机器上，整个部署过程花费不到 1 小时。由于节点共享率超过 0.5，并且减少了初始种子的负担，此方法在处理更大的机器组时可以很好地扩展。同时，虚拟机和相关服务可以在下载完整映像之前启用，服务可用时间在 20 分钟以内。

函数计算允许用户使用自定义容器映像和容器工具构建和部署 FaaS 应用程序。图 1 显示了功能部署和调用的典型工作流。为了部署(或更新)一个容器化函数，用户发送`create/update`请求，以便将容器映像推到一个集中的容器注册表(图 12 步骤 1)。调用一个部署的函数，用户发送`invoke`请求到前端网关(步骤 2)，前端网关检查用户的请求和注册表中容器镜像的状态(步骤 3)。然后前端网关将请求转发到后端法 Host VM 集群(步骤 4)。最后，主机虚拟机创建函数容器，并从注册表中拉出其容器数据(步骤 5)。以上步骤完成后，主机虚拟机准备就绪，开始为调用请求提供服务。

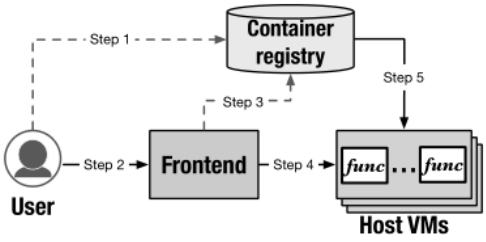

图 12 阿里巴巴 FaaS 容器工作流概要

FaaSNet 将虚拟机之间的容器供应进行了去中心化和并行化。引入了一个叫做 Function Tree（FTs）的抽象来支持大规模的高效容器供应。FaaSNet 集成了一个 FT 管理器组件到调度器（Scheduler）中，一个 worker 组件到虚拟机代理（VM agent）中，来协调 FT 的管理[8]。FaaSNet 架构如图 13 所示。

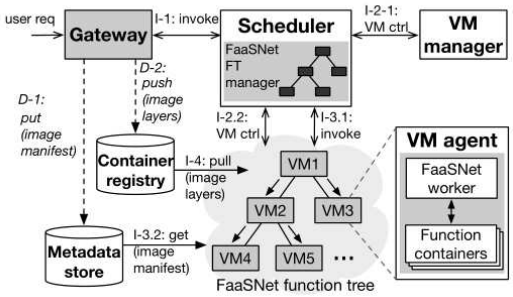

图 13 FaaSNet 架构图

gateway 负责：(1) 租户身份访问管理（IAM）认证，(2) 将函数调用请求转发到 FaaSNet 调度器，(3) 将常规容器镜像转换为高效的 I/O 数据格式。scheduler 负责：响应函数调用请求。一个 FaaSNet worker 负责：（1）服务调度器的指令来完成镜像下载和容器供应的任务，（2）管理虚拟机中的函数容器

FaaSNet 集成了一个 FT manager 到一个 scheduler 中通过`insert`和`delete`接口来管理 Function Trees。一个 FT 是一个二叉树，连接多个 Host VM 来形成一个快速的可扩展的容器供应网络，及架构如图所示。每个虚拟机运行一个 FaaS VM agent，用来负责虚拟机本地函数管理。为了容器供应任务，研究人员在 VM agent 中集成了一个 FaaSNet worker。在函数调用路径上，如果没有足够的虚拟机或者持有请求函数实例的所有虚拟机都很忙，调度程序首先与一个 VM Manager 通信，以从空闲的虚拟机池中向外扩展活动的虚拟机池。调度程序然后查询它的本地 FT 元数据，并向 FT 的 FaaSNet workers 发送 RPC 请求，以启动容器供应过程。还没有本地供应 container runtime 的所有虚拟机之间，在 FT 中供应 container runtime 的这个过程可以有效地去中心化和并行化。

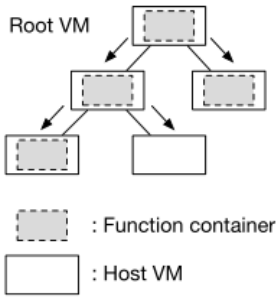

图 14 FaaSNet FT 的例子

FaaSNet workers 并行的从指定的对等虚拟机中按需获取函数容器层并创建 container runtime 时，调度程序决定关键路径。在函数部署路径上，网关通过从面向租户的容器注册表中提取通常镜像，将函数的通常容器镜像转换为高效的 I/O 格式，逐块压缩镜像层，创建包含格式相关信息的元数据文件(镜像清单)，并将转换后的层及其关联的清单分别写入阿里巴巴云内部容器注册表和元数据存储。

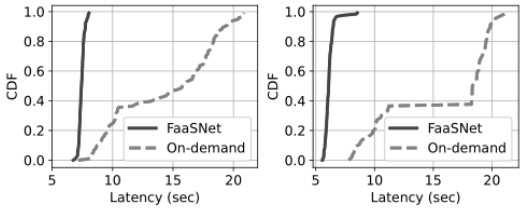

图 15 容器供应延迟

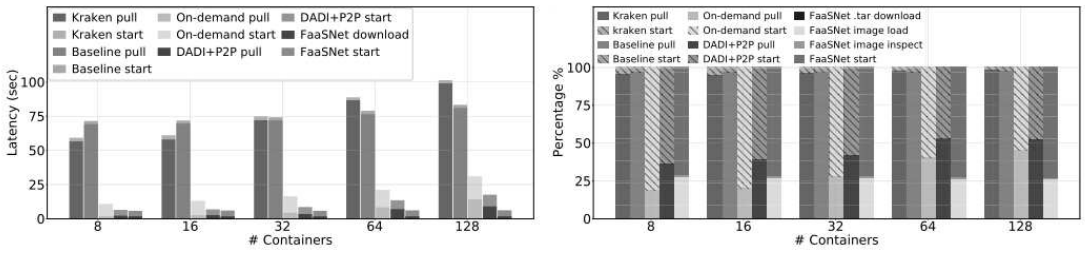

图 16 容器分发的可拓展性测试

研究人员使用两个 FaaS 应用程序来评估 FAASNET:一个物联网应用程序和一个游戏应用程序。由于最初的游戏工作负载在吞吐量上呈现出一个逐渐上升的趋势，因此研究人员将基于游戏工作负载创建一个合成的突发工作负载，以模拟压力测试目的的突发模式。测试集群有多达 1000 个 VM，因此将两个工作负载跟踪的峰值吞吐量按比例(约为原始吞吐量的 1/3)降低到集群大小，并将持续时间从 2 小时缩短到不到 1 小时。接下来，研究人员将分析在两个工作负载中看到的容器供应延迟。如图 13 所示，由于按需注册表会产生性能瓶颈，因此按需分配的容器供应延迟变化很大，从 7 秒到 21 秒不等。大约 80%的容器需要至少 10 秒才能启动。在 FaaSNet 中，容器启动延迟是高度可预测的，而且变化明显较小。对于合成工作负载，大约 96%的功能只需要 5.8 秒就可以启动。对于物联网工作负载，几乎所有的功能都在 6.8-7.9 秒的短时间内开始执行。这表明 FaaSNet 可以实现可预测的容器启动延迟。

图 16 显示，FaaSNet 在高并发性下可以很好地扩展，比基线速度提高了 13.4 倍，比 Kraken 速度提高了 16.3 倍。FaaSNet 比 on-demand 和 DADI+P2P 分别快 5 倍和 2.8 倍。如图 14(b)所示，FaaSNet 的平均容器供应延迟由两个操作支配:图像加载和容器启动。FaaSNet 消除了图像加载的两个操作上的瓶颈，FaaSNet 允许每个 FaaSNet 工作者从元数据存储中获取图像清单(开销可以忽略)，然后在本地并行启动图像加载过程，从而在每个 VM 上实现分散的图像加载(在功能方面相当于图像拉取);在容器启动时，每个 FaaSNet VM worker 直接从 peer VM 获取层块，并在获取足够的块时启动函数容器。通过所有这些优化，FaaSNet 在从 8 个函数扩展到 128 个函数启动时保持了几乎相同的延

迟。

可伸缩和快速的容器供应可以为 FaaS 提供商提供基本的弹性，这些提供商支持基于容器定制的云功能。FaaSNet 是第一个为优化的 FaaS container runtime 供应提供端到端集成解决方案的系统。FaaSNet 采用了轻量级、去中心化和自适应的 function tree，以避免主要的平台瓶颈。FaaSNet 为大型云提供商的 FaaS 平台(阿里巴巴云功能计算)提供了具体的解决方案。研究人员通过实验评估表明 FaaSNet 可以在数秒内启动数千个大型函数容器。这项工作将使基于容器的 FaaS 平台真正具有弹性，并为更广泛的依赖程度高的 FaaS 应用打开大门，包括机器学习和大数据分析。

4 总结与展望

无服务器计算平台是一种新型云计算范式，它由于不需要明确的配置、自动且几乎无限的扩展性，并按需计费的优势而更接近于云计算最初的期望。该模式的转变虽然带来了机遇，同时也带来了平台函数冷启动延迟、资源利用不足的问题。为此，本文中对无服务器计算平台冷启动耗时优化技术做了调查和分析，重点阐述了轻量级虚拟机技术以及容器的快速部署技术原理和相关研究现状。对 Xen 架构进行了全面改进的 LightVM，引入 unikernel 和 Tinyx 以减少虚拟机镜像大小和虚拟机内存占用，LightVM 能够在 2.3ms 内启动一个(单内核)虚拟机，达到多达 8000 个虚拟机的同一主机虚拟机密度，迁移和挂起/恢复时间分别为 60ms 和 30ms/25ms。每个虚拟机的内存占用只有 480KB(磁盘上)和 3.6MB(运行中)；以及基于 KVM 的轻量级虚拟化系统 Firecracker，其完全取缔了 QEMU 并建造了新的超轻量的 VMM（virtual machine manager）、设备模型以及 API 来管理和配置 MicroVMs，展示了优于 QUEM 和 Cloud Hypervisor 的启动速度和 VM 内存开销；使用 P2P 网络架构实现容器镜像分发的 Kraken，镜像的传输速度可以达到 5M/sec，一般情况下，将一个 12G 的解决方案镜像分发到所有目标机器上，整个部署过程花费不到 1 小时；引入新的中间件 FT 来实现自适应管理容器，体高容器扩容速度的 FaaSNet，FaaSNet 在高并发性下可以很好地扩展，比基线速度提高了 13.4 倍，比 Kraken 速度提高了 16.3 倍。FaaSNet 比 on-demand 和 DADI+P2P 分别快 5 倍和 2.8 倍。

从技术上讲，降低 FaaS 云平台的冷启动延时还有很多可行的办法。

（1）我们的工作可以通过 P2P 通信在虚拟机之间共享容器映像，它也有可能推广到更广泛的范围比如数据共享，将数据共享用于通用容器编排系统，如 Kubernetes。随着矩阵计算、数据分析、视频处理和机器学习等数据密集型应用的出现，FaaS 平台也出现了这种需求。它们中的大多数依赖于集中存储来进行数据交换，这与我们工作中的容器注册表类似，是一个瓶颈。所以接下来可以将数据共享考虑到 FaaS 平台的设计中，从而降低哪些数据密集型应用获取数据的负载。

（2）考虑云函数暂停并缓存调用函数一段固定的时间，以减少冷启动的数量。然而，这将增加供应商的 TCO。为了减少这样的成本，研究人员提出了一种预测方法，即及时对功能进行预热，以便传入的重复请求可能会击中温暖的容器。

（3）优化容器镜像的存储和检索。Slacker 通过利用惰性克隆和惰性传播来加快容器启动时间。从共享网络文件系统(NFS)中存储和获取映像，并从容器注册表中引用映像。

（4）功能环境缓存和预配置可以用来处理具有亚秒级持续时间和稀疏调用的短寿命函数，但需要额外的基础设施级成本。

参考文献

[1] KILCIOGLU C, RAO J M, KANNAN A, et al.Usage patterns and the economics of the public cloud[C]//Proceedings of the 26th International Conference on World Wide Web. 2017:83-91

[2] CNCF Serverless Whitepaper v1.0 [EB/OL].(2020-05-23)[2020-08-31].https://github.com/cncf/wg-serverless#white-paper

[3] MAHMOUDI, LIN C, KHAZAEI H,et al. Optimizing serverless computing:introducing an adaptive function

placement algorithm[C]//Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering.2019:203-213.

[4] Filipe Manco, Costin Lupu, Florian Schmidt, Jose Mendes, Simon Kuenzer, Sumit Sati, Kenichi Yasukata, Costin Raiciu, and Felipe Huici.. My VM is Lighter (and Safer) than your Container[C]. // Proceedings of the 26th Symposium on Operating Systems Principles Association for Computing Machinery, New York, NY, USA,2017:218–233.

[5] Alexandru Agache and Marc Brooker and Alexandra Iordache,et al. Firecracker: Lightweight Virtualization for Serverless Applications[J].//17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20), 2020:419--434.

[6] Zhijia Chen, Yang Zhao, Xin Miao, Ying Chen, and Qingbo Wang. Rapid provisioning of cloud infrastructure leveraging peer-to-peer networks.[C]// In 2009 29th IEEE International Conference on Distributed Computing Systems Workshops, pages 324–329, 2009.

[7] Kraken: A P2P-powered Docker registry that focuses on scalability and availability. https://github.com/uber/kraken.

[8] Cheng A W A S C A H T A H W A H Y A H L A R D A Y. FaaSNet: Scalable and Fast Provisioning of Custom Serverless Container Runtimes at Alibaba Cloud Function Compute[C].//2021 USENIX Annual Technical Conference (USENIX ATC 21), 2021:443--457.