面向云–端融合的移动容器云平台

吴 松[1] 牛 超[2] 金 海[1]
[1]华中科技大学
[2]亚马逊公司

关键词：云 - 端融合应用 计算卸载 容器

引言

根据友盟2015年底发布的中国移动互联网趋势报告，我国活跃移动设备数量已达10.8亿，比2014年增长20%。移动设备不再是简单的通信工具，越来越多的移动应用尝试完成更加复杂的逻辑功能，诸如大型游戏、人工智能等。移动云计算也应运而生，通过云端和终端的融合，利用云计算近乎无限的资源来扩展移动设备受限的硬件性能。目前，大多数研究工作是面向云 - 端融合的具体应用及智能终端的调度节能工作，优化了移动应用的用户体验。然而，云 - 端融合应用为云计算平台带来的新需求却鲜有研究。本文从云计算系统角度出发，研究云 - 端融合场景对云计算平台的需求与挑战，通过分析传统云计算平台在该场景中存在的问题，针对移动云的应用特征，探索性地提出了一套基于容器的轻量级云计算平台架构。

云–端融合应用场景

针对目前工业界和学术界的研究现状，我们根据云计算平台在移动应用中扮演的角色，将云 - 端融合应用场景分为三类（如图1所示）。

苹果公司于2011年10月发布了iOS上的个人语音助手Siri，如今Siri已成为苹果智能手机用户日常生活的一部分。Siri通过在本地设备对语音内容加以识别分析，连接云计算服务分析指令内容。这种类似传统客户机 / 服务器 (C/S) 应用模型的移动云结构目前在工业界最为流行，在这种场景中，云计算平台是软件服务的提供者。

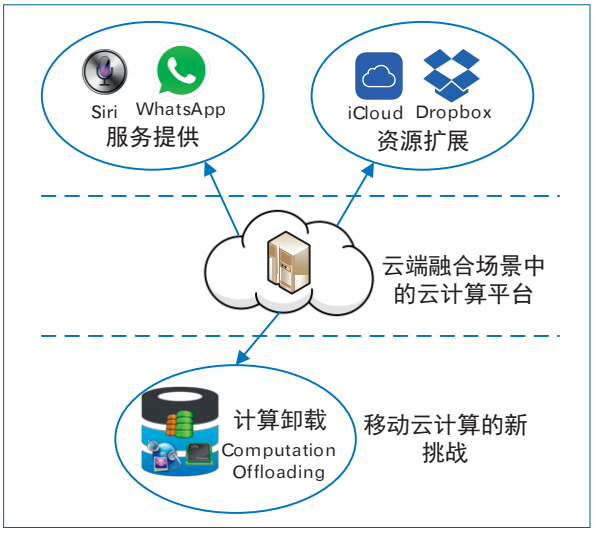


图1 云计算平台扮演的角色

与Siri同时发布的另一款苹果公司的产品iCloud，则侧重于为用户提供私有空间，苹果用户可通过iCloud方便地将设备中的文件存储到云端以突破移动设备物理存储的限制。截至2016年2月，iCloud累计用户数量已达7.82亿。在这类云 - 端融合场景中，云计算资源是移动设备的资源扩展。

上述两种移动云计算场景被广泛应用于生产环境中，也是目前工业界实现移动应用和云资源结合的主要手段。除此之外，学术界提出的应用模型——计算卸载 (computation offloading) 为移动设备从云

端受益提供了新的思路。在计算卸载中，移动设备不再从云端的服务提供者直接获取数据，而是把部分移动应用中的工作负载迁移到云上，利用云平台各类资源来帮助收集、存储和处理数据。图2介绍了典型的计算卸载的架构流程。

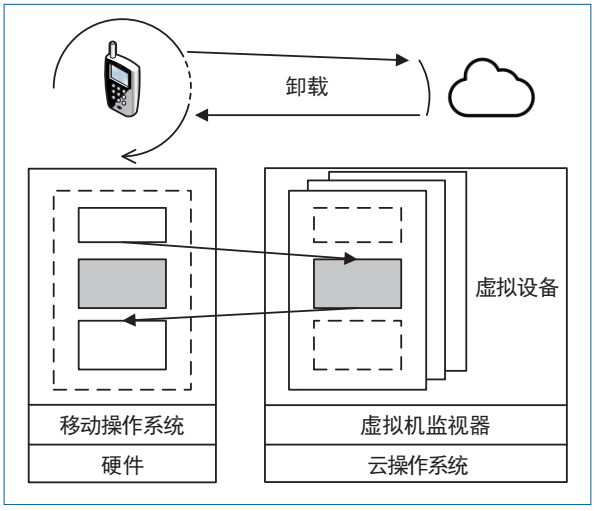


图2　计算卸载架构流程

作为云 - 端融合的一种新型应用场景，计算卸载近些年一直是移动云计算领域的研究热点。杜克大学提出的 MAUI[1] 和英特尔伯克利实验室提出的 CloneCloud[2] 是最早的计算卸载实现模型；德国电信实验室索科尔・索斯塔 (Sokol Kosta) 提出的 ThinkAir[3] 和密歇根大学马克・戈登 (Mark S. Gordon) 提出的 COMET[4] 都是早期卸载框架优化扩展的代表成果。北京大学张颖等人研究的 DPartner[5] 和华盛顿大学张艾琳（Irene Zhang，音译）等人提出的 Sapphire 则尝试从新的角度实现高效的计算卸载。

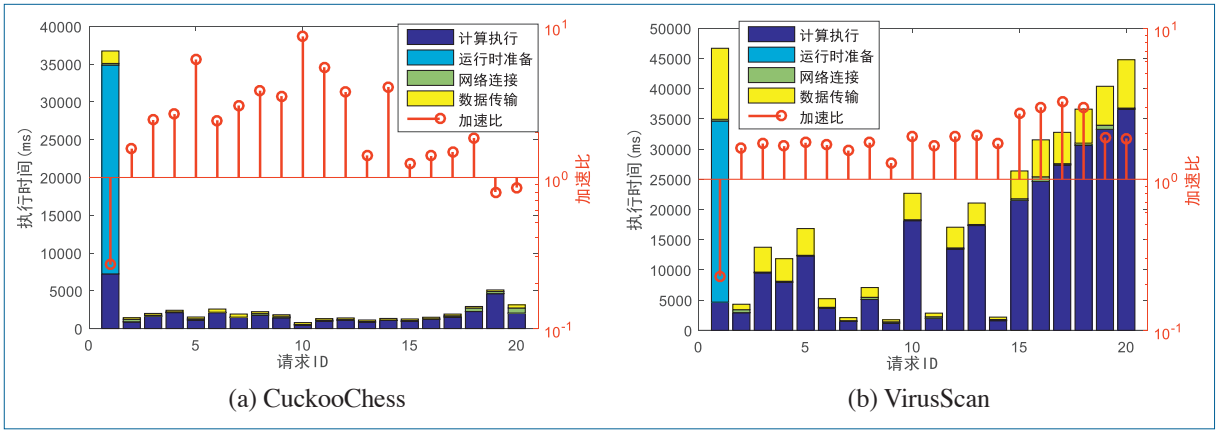
然而，尽管有国内外学者的大量研究成果作为基础，计算卸载却迟迟没有成为主流的移动云计算解决方案。其中最重要的原因在于，现有的云计算平台结构在支撑计算卸载请求时存在缺陷，云 - 端融合场景下的针对云计算平台的研究相对稀缺。因此，构建高效的移动云计算平台成为目前移动云计算相关技术发展的重要挑战。

本文针对计算卸载场景中的移动应用请求，探索新的云计算平台模式，尝试解决计算卸载难以实用的难题，进而希望为传统云 - 端融合模型带来启发。

面临的问题

要针对云 - 端融合场景构建高效的云平台，面临的首要问题就是搞清楚移动应用对云计算平台的需求特征。我们以 6 个典型移动应用为例，对计算卸载中的请求进行了初步分析。用于分析的基准应用是：OCR（光学字符识别）、FaceDetect（人脸识别）、Linpack（浮点性能测试）、Calculus（定积分计算）、CuckooChess（国际象棋 AI 引擎）、VirusScan（病毒扫描）。为了发现现有云平台存在的问题，云端采用最常见的虚拟机模拟移动设备环境的解决方案。

运行时准备　对于移动应用卸载到云平台的



(a) CuckooChess　　(b) VirusScan

图3　卸载请求各阶段执行时间及加速比

工作任务，云平台需要及时准备合适的运行时环境来运行这部分任务。图 3 展示了计算开始后前 20 个请求在各个阶段的执行时间，以及计算卸载相比本地执行带来的加速比。由于篇幅限制，我们选取 CuckooChess 和 VirusScan 两个基准应用作为代表。可以发现，首次卸载请求执行中存在长时间的运行时准备，可导致计算卸载反而比本地执行耗费更长的时间，这是由于虚拟机需要较长的启动时间所致。以 Amazon EC2[6] 为例，其单一实例的平均创建时间在 26 秒左右，这对于计算时间较短的移动应用任务来说是不可接受的。

云平台负载 图 4 展示了云平台响应上述请求过程中的负载，可以发现请求呈现计算密集和短时 I/O 的特点。图 4(a) 中，CuckooChess 工作负载以 CPU 资源为主，由于其单个请求的计算量较小，其 CPU 利用率表现出剧烈的波动，这也是大部分移动应用和传统云计算中的计算负载的一个明显区别。VirusScan 是将指定文件和数据库中的病毒特征值进行对比，因此产生了更多的 I/O 操作，如图 4(b) 所示。由于虚拟机中 I/O 虚拟化带来的开销相比 CPU 虚拟化的开销更加严重，因此尽管这部分 I/O 操作持续时间短，我们仍然认为这很可能会影响计算卸载的整体性能。

新型云平台探索

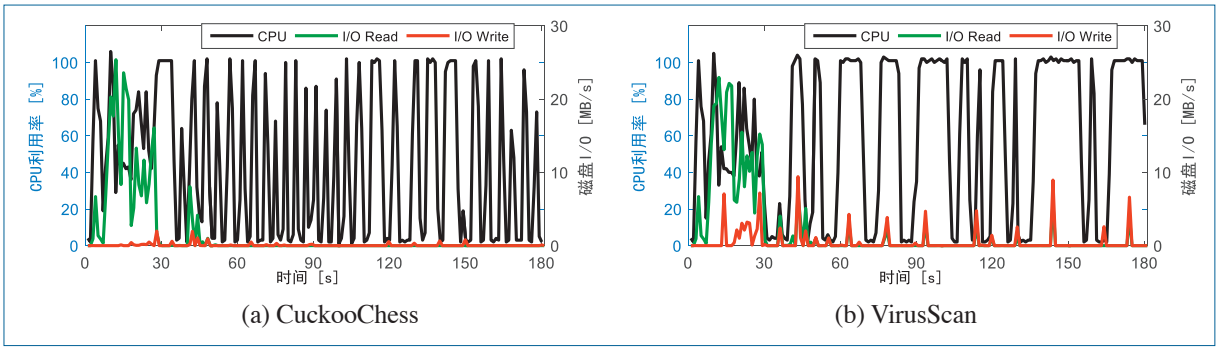
上述分析说明了云计算平台在云 - 端融合场景中的两个问题：过长的资源准备时间；虚拟化带来的 CPU 和 I/O 开销。由于移动应用对请求执行延迟非常敏感的特点，这两个问题严重影响了用户的应用体验，制约了云 - 端融合场景下的云计算平台发展。为了解决这些问题，我们认为需要从根本上改变云平台的结构，从运行时级别提升云计算平台的效率。

2013 年，基于 go 语言的开源项目 Docker 走进了开发者的视线 [7]。这个最初来源于 Linux Container 的高级容器引擎，截至 2016 年初用户使用量已达 20 亿。Docker 本质上是一种容器解决方案，它通过系统内核本身提供的命名空间 (namespace) 和 cgroup 来实现多个模拟环境的隔离。不同的容器和宿主操作系统共享一个内核，容器内部的执行单元对底层内核请求时不再通过虚拟机监视器 (hypervisor) 层，几乎不存在性能损失。同时，容器具有快速部署的特性，其启动时间一般在秒级之内完成。Docker 的出现引发了近些年 IT 界新的容器热潮，2014 年 11 月，谷歌和亚马逊接连发布各自基于容器的新业务，容器开始在云计算平台中崭露头角。

在云 - 端融合场景中引入容器作为云计算平台的资源组织单元，既可以解决虚拟机带来的性能开销，也可以利用容器的轻量级特性满足移动应用请求的高实时响应需求。

挑战与解决方案

构建基于容器的移动云计算平台面临关键性的挑战。虚拟机通过虚拟化技术能够做到对多种内核的支持，而容器在设计之初就不具备这个能力。以安卓操作系统为例，由于云上的宿主机内核多采用



(a) CuckooChess (b) VirusScan

图4 计算卸载请求中的云计算平台负载

标准 Linux 内核，和安卓内核存在较大差异，因此容器无法像虚拟机一样简单地实现安卓模拟环境，图 5(a) 描述了这个问题。解决这一挑战的关键在于

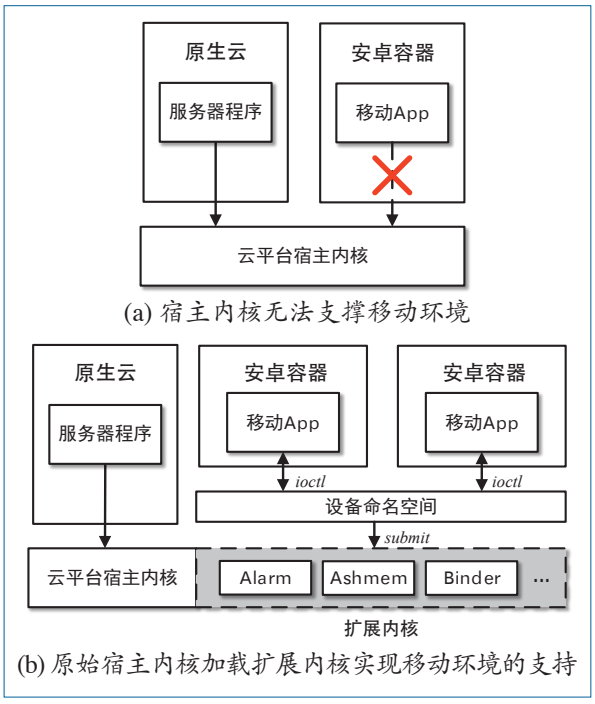


图5　云平台运行安卓容器的挑战和解决

安卓的内核特性 [8](Android kernel feature) 必须被云平台的宿主系统内核支持，进而为安卓容器的运行提供可能。

为了解决这一难题，我们设计了针对移动环境 (Android) 的扩展内核模块，其中集成了关键的安卓内核驱动。这部分核心驱动包括 Alarm（定时触发器）、Binder（安卓系统进程间通信机制）、Logger（轻量级的 RAM 日志驱动）、LMK（Low Memory Killer，低内存管理）、timed output/gpio（通过超时时间控制通用输入 / 输出）以及 Ashmem（匿名共享内存）。

核心驱动在不同安卓容器间的复用和隔离通过设备命名空间 (device namespace) 实现。设备命名空间在哥伦比亚大学的研究项目 Cells[9] 中提出，用于实现同一个手机上多个安卓系统对设备的并发使用。它本质上是对传统内核命名空间 (kernel namespace) 的扩展。

如图 5(b) 所示，宿主操作系统通过动态的加载来扩展模块，实现了安卓核心驱动的初始化，不需要重启云服务器或重新编译云平台宿主内核，具备扩展灵活性。值得一提的是，扩展内核模块中包含的是安卓核心驱动，它们保障了最基本的安卓系统运行，而蓝牙和相机等硬件设备驱动则被排除在外，因为在移动云计算平台中的请求任务中一般不包含硬件相关的操作任务，这部分硬件驱动的移除降低了内核扩展开销。

仅仅解决了内核兼容并不能使安卓容器正常启动，因为原本设计运行在移动设备上的安卓启动过程和传统容器的启动过程差别很大。图 6(b) 描述了我们为实现安卓容器启动做的准备工作，对比图 6(a)，准备工作主要包含三个部分。首先，安卓容器共享宿主内核，因此启动过程中不必再加载内核。其次，容器的启动过程是直接在根文件系统 (rootfs) 上启动 init 进程，因此我们预先构建了安卓系统的根文件系统，从而启动过程中不必再准备文件系统。最后，原始的 init 进程在容器中已经无法正常工作，我们对 init 进程进行了定制，使其能够顺利启动，同时删除了不必要的系统服务，进一步优化了启动时间。

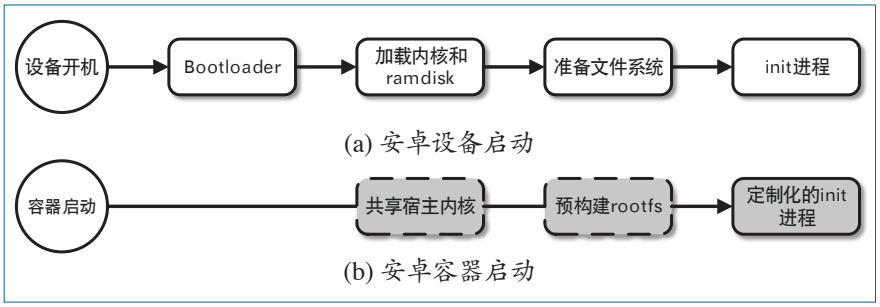


图6　安卓设备启动 vs. 安卓容器启动

平台的构建

以安卓容器为基础，我们得以构建面向云 - 端

融合场景的轻量级云平台，其系统架构主要包括运行时层、共享资源层和内核层三部分（见图 7）。

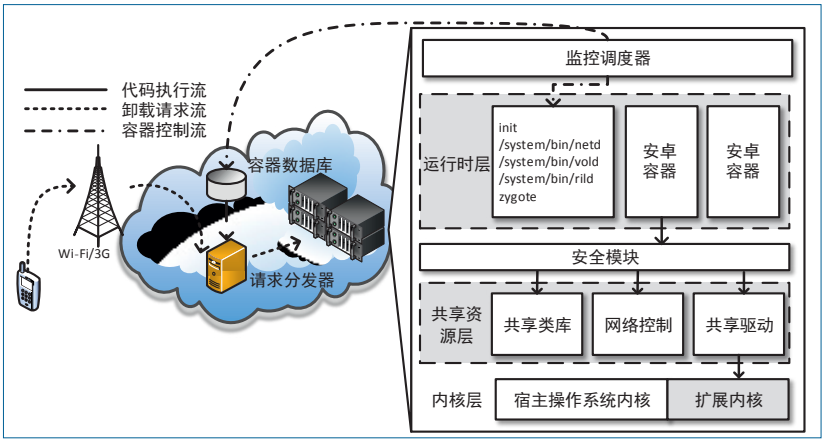


图7　基于容器的移动云平台架构

运行时层主要以安卓容器的形式为计算卸载请求提供工作负载的执行环境，利用安卓容器快速部署的特性实现了云端资源的高效响应，同时容器降低了运行时环境的性能损失，进而提高了卸载任务的执行效率。

共享资源层提供了不同容器间共享的资源，这类似于 Docker 采用 AUFS[10] 实现的分层结构。这部分共享资源主要包括安卓系统运行所需的类库、安卓的网络控制模块以及安卓核心驱动。

内核层通过宿主操作系统加载扩展内核，支撑了上层环境的正常运行。

采用容器替代虚拟机作为云端的资源管理单位也带来了安全问题，这源于容器相比虚拟机的弱隔离性。为了解决这个问题，系统设计了安全模块，通过检测从容器中发出的执行流的访问对象是否在其权限范围内，记录下非法请求。这些记录被用来定位计算请求的来源应用，当应用的非法请求数量达到阈值则标记其为恶意应用。

资源调度一直以来都是云计算环境中的研究热点，现有的研究工作 [11~13] 大多以虚拟机为调度单位，通过虚拟机迁移和资源配额调整来实现云端资源的管理。然而，从调度的粒度来看，虚拟机内部应用的调度才是云计算用户更加关心的。在云平台的监控调度器中，监控信息的获取得以深入到容器内部，监控的粒度从虚拟机级细化到进程级，这使得运行时内部应用的运行情况变得可见，从而有效地提高了资源调度的准确度。

初步验证

我们对比了新型云计算平台安卓容器和传统解决方案中的安卓虚拟机。

我们首先关注云 - 端应用运行环境的启动时间

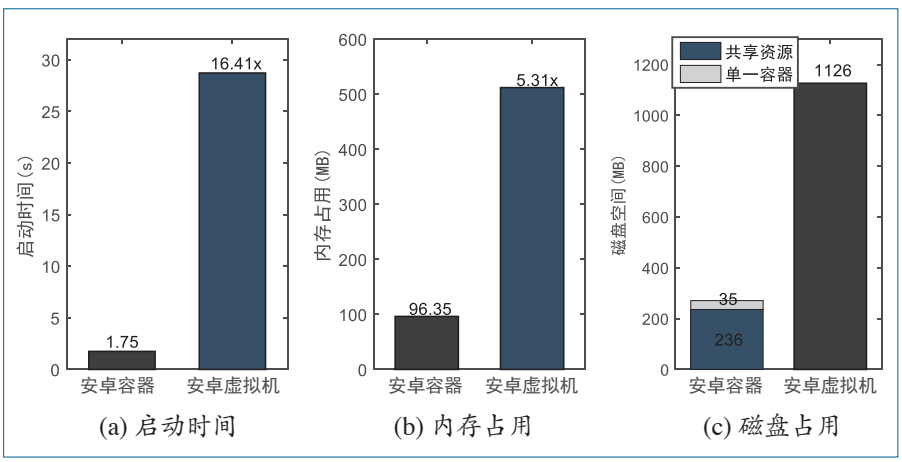


图8　安卓容器和安卓虚拟机的开销对比

和资源占用。图 8(a) 给出了两种资源模型的启动时间，安卓容器的启动时间仅有 1.75 秒，仅为安卓虚拟机启动时间的 1/16，很大程度上降低了云平台中运行环境扩展的延迟。图 8(b) 和图 8(c) 从内存占用和磁盘占用方面验证了安卓容器的轻量级特性。在内存占用方面，单一容器的内存占用仅为虚拟机的 1/5，这提升了在有限的物理资源条件下能同时生效

的运行环境数量。在磁盘占用方面，单一容器的体积仅为 35MB，而安卓虚拟机的磁盘占用在 1.1GB 左右。共享资源层是所有容器共享的一份存储，通过对系统中冗余部分的精简，其体积仅为 236MB。

为了评估云计算平台性能的提升，我们评估了前面提到的 6 个典型移动应用在不同云计算平台上的执行效果，如图 9 所示（AC 表示安卓容器）。可

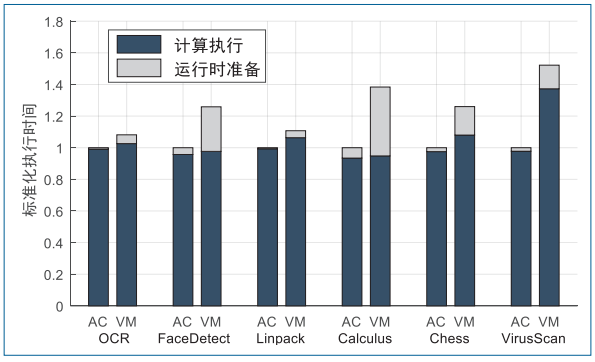


图9　不同应用的执行性能

以看到不同应用负载的执行性能均有不同程度的提升 (1.07x~1.52x)，主要来源于运行时准备和计算执行两个方面的提速。在计算性能方面，由于容器的 I/O 性能相比虚拟机提升较大，因此 VirusScan 这一 I/O 操作相对较多的应用有更好的优化效果。

未来的方向

我们在新型云平台的探索中，将容器引入云计算平台作为移动应用的运行容器，具体地解决了传统云计算平台在云 - 端融合场景中的缺陷。在本节，我们从探索成果的应用场景 (DevOps) 和平台自身的优化前景 (Unikernel) 两个方面介绍云 - 端融合场景下云计算平台未来的研究方向，以期为云 - 端融合的发展提出新的思路。

DevOps

DevOps[14]（英文 Development 和 Operations 的组合）是一组过程、方法与系统的统称，用于促进软件开发者和其他 IT 技术专家之间的沟通、协作与整合，自动化软件的交付和基础设施变化的过程。Docker 的出现为 DevOps 带来了变革，这源于 Docker 对不同机器上运行环境的标准化，使开发环境可以轻松地部署到生产环境中。然而由于移动应用的运行大多以设备环境为主，使用 Docker 作为 DevOps 团队的部署管理工具不再有效。

在我们的新型云平台探索过程中，一个重要的贡献是实现了移动操作系统环境在云计算平台的容器中运行，即安卓容器。为了解决上述难题，安卓容器为云 - 端融合场景中的 DevOps 带来了希望，它连接了移动应用的开发和运维部署，从而可以高效地实现云 - 端融合应用部署到生产环境中去。

尽管容器的引入以及 DevOps 思想的结合为云 - 端融合应用带来了新的可能，但是新型云平台在 DevOps 中的使用，对运维而言还存在很大的挑战，这需要有经验的 DevOps 团队根据需求针对性地提出大规模容器部署的解决方案，同时对现有安卓容器进一步定制，例如，实现类似 Docker 的一套完善的运行环境定义管理。有关安卓容器在 DevOps 应用场景中的改进将作为未来重要的研究方向。

Unikernel

容器的引入提升了云 - 端融合场景中云计算平台的资源使用效率，但是容器的隔离性和安全性问题仍然是应用过程中的隐患。同时，要做到即时资源分配 (just-in-time)，运行环境的启动时间必须在毫秒级。即时资源分配意味着仅在请求到达后为其分配资源，且资源在请求执行后被立即回收，这是云 - 端融合场景下的云计算平台发展的最终目标，将云 - 端应用运行容器像物理资源一样提供给移动应用请求。

为了满足上述需求，Unikernel 看起来是一个很有潜力的解决方案，是新兴的一种很有前景的实现超轻量级云服务器的技术[15,16]，在 2016 年初，Docker 公司宣布收购 Unikernel System，尝试寻求容器与 Unikernel 结合的发展前景。所谓 Unikernel，是用库操作系统 (library operating system) 构建的专用的单地址空间镜像，可直接运行在 hypervisor 或

硬件上。它通过把操作系统精简到只保留运行特定应用所需的最少资源，以最小的依赖运行应用程序，将资源消耗做到最小化，其构建过程如图 10 所示。

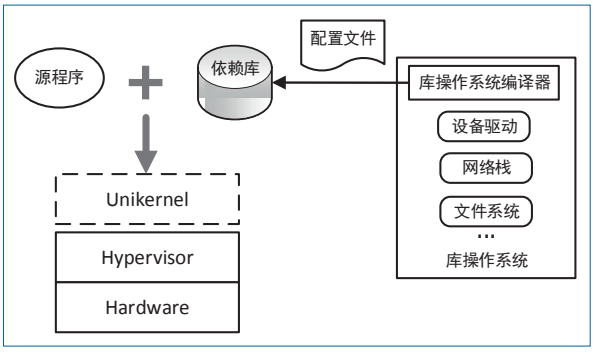


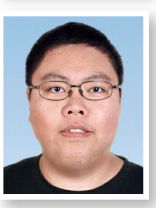
图10　Unikernel构建过程

Unikernel 具有镜像体积小（大约几百 KB）、启动速度快（毫秒级）、高安全性、高兼容性的优点，这些优点都从不同角度弥补了现有容器技术中存在的缺陷。因此，可以说 Unikernel 为云 - 端融合场景下的云计算平台提供了新的选择。Unikernel 作为云 - 端应用的运行容器，可以做到对用户请求的即时响应，同时保障不同环境间的强隔离性和不同移动环境的兼容性。

以 Unikernel 为核心构建云 - 端融合场景中的云计算平台存在着不小的挑战，如何构建移动环境下的库操作系统是一个技术难题，且工程量巨大。尽管如此，在容器技术产业背景下可以预见，未来几年 Unikernel 会和容器技术紧密结合并迅速发展，势必为云 - 端融合场景中的云计算平台带来更多的可能。■

吴　松

CCF高级会员，体系结构专委和高性能计算专委委员。华中科技大学教授，并行分布式计算研究所所长。主要研究方向为云计算、虚拟化和并行分布式系统。wusong@hust.edu.cn



牛　超

CCF专业会员。亚马逊公司云计算系统工程师。
527274127@qq.com

金　海

CCF会士、常务理事、出版工作委员会主任。华中科技大学教授。主要研究方向为虚拟化、网格计算、对等计算、集群计算和网络存储。
hjin@hust.edu.cn

参考文献

[1] Cuervo E, Balasubramanian A, Cho D, et al. MAUI: making smartphones last longer with code offload[C]// *Proceedings of the 8th international conference on Mobile systems, applications, and services*. San Francisco, California, USA: ACM, 2010. 49-62.

[2] Chun B G, Ihm S, Maniatis P, et al. Clonecloud: elastic execution between mobile device and cloud[C]// *Proceedings of the sixth conference on Computer systems*. Salzburg, Austria: ACM, 2011. 301-314.

[3] Kosta S, Aucinas A, Hui P, et al. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading[C]// *Proceedings of IEEE INFOCOM*. Orlando, FL, USA: IEEE, 2012. 945-953.

[4] Gordon M S, Jamshidi D A, Mahlke S, et al. Comet: Code offload by migrating execution transparently[C]// *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. Hollywood, CA, USA: USENIX Association, 2012. 93-106.

[5] Zhang Y, Huang G, Liu X, et al. Refactoring android java code for on-demand computation offloading[C]// *Proceedings of the 27th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*. 2012: 233-248.

[6] Amazon EC2, http://aws.amazon.com/ec2/.

[7] Docker, http://www.docker.com.

[8] Android Kernel Feature, http://elinux.org/Android_Kernel_Feature.

[9] Andrus J, Dall C, Hof A V, et al. Cells: a virtual mobile smartphone architecture[C]// *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP)*. Cascais, Portugal: ACM, 2011: 173-187.

[10]AUFS, http://aufs.sourceforge.net/.

[11]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 (NSDI)*. Boston, Massachusetts, USA: USENIX Association, 2005: 273-286.

[12]Hines M R, Gopalan K. Post-copy based live virtual

machine migration using adaptive pre-paging and dynamic self-ballooning[C]// Proceedings of the 5th International Conference on Virtual Execution Environments (VEE). Washington, DC, USA: ACM, 2009: 51-60.

[13]Kim J, Chae D, Kim J, et al. Guide-copy: fast and silent migration of virtual machine for datacenters[C]// Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC). Denver, CO, USA: ACM, 2013: Article No.66.

[14]DevOps, http://en.wikipedia.org/wiki/DevOps.

[15]Madhavapeddy A, Mortier R, Rotsos C, et al. Unikernels: Library operating systems for the cloud[J]. ACM SIGPLAN Notices, 2013, 48(4): 461-472.

[16]Madhavapeddy A, Leonard T, Skjegstad M, et al. Jitsu: Just-in-time summoning of unikernels. [C]// 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15). 2015: 559-573.