云–端融合下的端设备能耗优化

曹 春 陆子凌 马晓星
南京大学

关键词：云 - 端融合 能耗优化

引言

移动应用技术的快速发展让我们能在移动设备上使用各类越来越复杂的应用，然而电池技术的停滞不前限制了移动设备的应用场景和能力，成为影响用户体验的重要因素。针对这一问题，在对电池技术本身的研究之外，软件技术层面的研究也尝试对设备能耗进行优化。传统意义上通过代码优化、漏洞检测和消除等技术实现的能耗优化，仅能在一定程度上缓解终端设备的能耗问题。云 - 端融合计算模式和相应技术平台为此问题提供了一个全新的解决方案。本文基于国内北京大学、南京大学等单位和国外学者在此方向所开展的较为典型的工作，对以计算迁移实现云端资源融合的计算模式采用的基本技术方法进行介绍，讨论该技术领域的发展路线和其在解决终端能耗问题中存在的技术挑战，为更近一步发挥云 - 端融合技术优势、形成新型的云 - 端融合应用形态及商业模式提供部分思路。

端设备能耗问题

从 1996 年 Palm 公司发布具有 128K 内存、16MHz CPU 的 Palm Pilot 个人数字助理 (PDA) 产品，仅仅过了 20 年，市场上随处可见的运行着安卓、苹果和微软视窗等系统的移动设备已具有了与当下主流桌面型电脑相当的硬件配置。移动终端的计算能力增长了成百上千倍，移动应用软件从当初简单的文本、数值存取处理逐步演化为各类复杂信息、娱乐和通信等应用。在用户期待以移动终端设备替代传统桌面和膝上型设备之时，发展相对缓慢的电池技术限制了这一趋势的实现。设备电力续航条件的限制，使得应用无法以不计开销的方式运行，这在很大程度上影响了应用性能和用户体验。

应用能耗过度问题的主要原因包括应用本身存在恶意行为（例如过度显示广告内容等）和应用本身的设计 / 实现缺陷。前者源自设计者的主观因素，可以通过识别具有恶意行为的应用让用户终端的电量资源避免被恶意消耗 [1,2]。后者则是软件技术研究的重点，通过应用设计和实现的改进优化终端系统的能耗水平，提升用户体验。

应用软件能耗优化

针对终端应用能耗过高这一问题，相关研究从应用开发运行相关技术和工程角度尝试优化应用能耗。例如，李鼎（Ding Li，音译）等人 [3] 从编码的角度出发，提出了一些可以降低能耗的编程经验，并且对其他人推荐的编程经验（例如安卓开发者网站上给出的一些最佳实践 [4]）进行了实验验证。马里奥·利纳雷斯 - 巴斯克斯 (Mario Linares-Vásquez) 等人将外部设备测出的能耗数据与执行路径对应起来，统计每个应用编程接口 (API) 调用的能耗，并提出避免使用能耗高的 API，而尽量使用功能相同的能耗低的 API，进而降低应用能耗 [5]。安卓系统（因其他主流移动平台的封闭性，当前研究主要针

对安卓系统开展工作，本文也主要针对安卓平台展开技术介绍和讨论）自身也在针对应用能耗进行优化。例如，安卓运行时 Dalvik 采用即时编译策略，在程序运行时将字节码翻译成机器码来执行，这样运行时开销就增加了；新的运行时 ART(Android Runtime) 采用预编译策略，在程序安装时就将字节码翻译成机器码，降低了运行时开销。

应用开发中能耗相关的不当设计称为能耗漏洞或能耗 Bug。这种漏洞不会影响应用的功能，也不会引起应用崩溃，只会让应用消耗更多的电量，并且用户往往很难发现。部分研究工作通过静态或动态方式进行检测。典型的能耗漏洞为“无法进入休眠的漏洞 (No-sleep Bug)”：安卓系统为了节省能耗，会在用户无操作一段时间后进入休眠状态，但这经常会影响一些应用的功能，于是安卓系统提供了唤醒锁 (wakelock) 机制。这是一种锁机制，只要系统中有应用申请了唤醒锁，系统就无法进入休眠状态，直到唤醒锁被释放。然而有些应用申请了唤醒锁，却忘记将它释放，或由于某些原因没有执行到释放唤醒锁的代码，导致系统永远无法进入休眠状态，这样就产生了一个无法进入休眠的漏洞。阿比纳夫·帕沙克 (Abhinav Pathak) 等人 [6] 通过数据流分析的方法检测可能导致无法进入休眠的漏洞的运行路径，并添加释放唤醒锁的代码来进行修复。又如，某些能耗漏洞与传感器使用不当相关，某些安卓应用会申请一些能耗较高的传感器资源，如全球定位系统 (GPS)，但在使用完后忘记将其释放，导致后台不断地获取传感器数据，而这些数据又无任何实际用途，造成能耗漏洞。可以看出，传感器相关漏洞 (Sensor-related Bug) 与无法进入休眠的漏洞的模式很类似，文献 [7] 同样通过数据流分析的方法进行检测并修复。能耗漏洞不仅可以从应用的字节码来检测，还可以从应用的实际能耗行为来分析。例如，班纳吉 (Abhijeet Banerjee) 等人 [8] 利用外部设备测量应用运行过程中的能耗，如果在应用运行之前和运行完毕后，设备的能耗行为不相似，则认为执行路径中存在能耗漏洞。

硬件部件能耗优化

优化终端能耗的另一个思路是在硬件部件层面进行能耗优化过程。从根本上讲，设备运行时的能量是在硬件部件上消耗的，因此部分工作着重优化能耗较高的设备部件。例如，针对高能耗的 GPS 调用，庄振云 (Zhenyun Zhuang, 音译) 等人 [9] 提出利用能耗更低的基于无线网络的位置感知技术来代替 GPS 调用，利用加速度传感器来判断用户的运动状态，以便在用户静止时限制不必要的 GPS 调用，将来自不同应用的 GPS 调用请求进行同步以减少调用次数，调整 GPS 调用的请求参数等等，最终 GPS 调用能耗降低了 98%。

针对高能耗的 Wi-Fi 接入点扫描，金玉韩 (Kyu-Han Kim) 等人 [10] 提出根据用户的运动状态和 Wi-Fi 接入点的密度来调整 Wi-Fi 接入点扫描的间隔，最终降低了 79% 的 Wi-Fi 接入点扫描的次数。

屏幕能耗占应用总能耗的一大部分。而如今许多安卓设备都配有有机发光二极管 (OLED) 屏幕，其能耗受到屏幕亮度、显示内容的颜色的影响。在屏幕亮度相同的条件下，显示稍暗的黑色、灰色等要比显示稍亮的白色、黄色等的能耗要低。南京大学 [11] 据此对安卓系统能耗配置文件中的屏幕能耗模型做了改进。董勉 (Mian Dong) 等人 [12] 指出，许多 Web 应用的背景色都是白色，这并不是一个节省能耗的做法。他们调整 Web 应用的配色方案，使面积更大的背景色变为黑色，以此来降低 Web 应用的能耗。这种方法不仅限于 Web 应用，许多背景色是白色的安卓应用都可以针对屏幕能耗做出改进。

云–端融合的终端能耗优化

应用软件层面的编程优化、能耗漏洞检测和硬件层面的部件能耗优化提升了终端能耗效率，但这些立足于终端本身的技术方法只是在一定程度上缓解了设备的过度耗能问题，其优化空间的最终上限是完成应用逻辑本身所必需的基础能耗。云 - 端融合技术为终端能耗优化研究提供了一个完全不同的

技术方向，其基本思路是在终端之外的云端提供可用资源，支持终端应用的运行，从而节省应用在终端本身运行时在各个耗电部件（CPU、网络、传感器等）上所需消耗的能量。换言之，云 - 端融合技术将云端作为终端的延伸，通过将应用计算（代码）从终端迁移至云端运行，实现云端资源与终端资源的融合，共同完成应用运行支撑。

在云 - 端融合的技术理念下，将应用的部分计算任务迁移到云端来执行，在利用云端丰富的计算资源（如 CPU、内存等）的同时，也可以优化端移动设备上应用的能耗，其问题的关键在于如何进行计算任务迁移。计算迁移实现的基本方法是将应用的部分或全部代码预先或运行时拷贝至远程的其他计算设备，并在运行时的某一时刻在该远程设备上执行这些代码。这一技术在 20 世纪 90 年代时曾获广泛研究，在当时的互联网技术条件，特别是移动互联网初现的情况下，传统分布式计算模式面临计算节点间通信信道带宽窄、延时高、稳定性差等因素导致的远程调用低效问题，相应研究所产生的 Aglets、DAgent、TCL、Mogent 等代码移动技术（称为移动代理技术），通过在运行时将部分代码（特别是远程方法调用部分的代码）迁移至被调用方，避免了应用需要长时间维持可靠稳定通信信道的要求，实现了更为可靠、灵活的面向互联网的分布式计算模型。

基于计算（代码）迁移的云 - 端融合本质上采用同样的技术方法，通过将代码执行迁移到云平台，实现对云端计算、存储和网络资源的利用，减轻了移动设备本身的资源负担，提高了应用性能并降低了移动设备能耗。基于这一思路，出现了一系列研究工作 [18]，包括 MAUI、CloneCloud、DPartner 和 ThinkAir 等。

MAUI

MAUI 是爱德华多·凯尔弗 (Eduardo Cuervo) 等人在 2010 年发布的一项较为全面实现移动设备计算迁移的研究工作 [13]。与本领域其他工作类似，为屏蔽不同平台硬件异构问题（例如移动设备基于的 ARM 平台和云端基于的 x86 平台），MAUI 在微软公司 .NET 公共语言运行时 (.NET CLR) 上实现代码和计算的运行时迁移，支持在应用运行时细粒度地决定将哪些代码迁移到云端运行，并通过实验验证了 MAUI 系统可有效降低应用在移动设备上的能耗。

MAUI 设计了一个较为简单的开发框架，移动应用开发人员在该框架下可对其开发的应用进行注释，将那些可以迁移到云端运行的方法标注为 Remoteable。在应用运行时，通过自省技术 (reflection) 对标注为 Remoteable 的方法进行辨识，当某个 Remoteable 方法被调用并且有可用的远程服务器时，MAUI 就用它的决策引擎来决定该方法是否应该迁移到远程执行以降低应用的能耗。MAUI 根据 Remoteable 方法迁移到远程执行所需要传输的状态信息的数量，来分析迁移的成本，通过统计迁移节省的 CPU 周期的数量，来分析迁移的收益。另外，MAUI 持续地监控移动设备的网络情况，评估带宽和延迟，作为其决策引擎的参数，以此适应网络环境的变化。图 1 为 MAUI 的系统架构。在移动设备

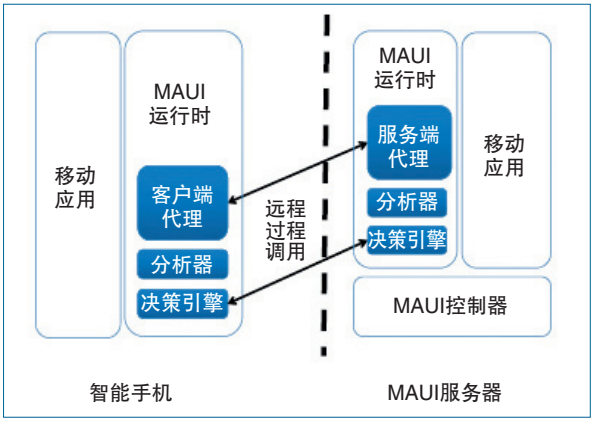


图1　MAUI系统架构

上，MAUI 主要包括三个组件：**客户端代理** (client proxy)，负责待迁移方法的状态信息的传输；**分析器** (profiler)，负责分析迁移的成本和收益；**决策引擎** (solver)，为了降低能耗，它实际运行在 MAUI 服务器上。MAUI 服务器主要包括四个组件：**服务**

端代理 (server proxy) 和**分析器**，与移动设备上相应的组件功能一致；**决策引擎**，定期对方法是否需要迁移进行决策；**MAUI 控制器**，用于对方法迁移请求进行身份验证和资源分配。

文献 [13] 针对 MAUI 平台做了一组实验。作者选取了面部识别、视频游戏和国际象棋三类应用通过 MAUI 平台进行计算迁移，开展了应用在移动设备上的性能和能耗等方面的比对评估。实验结果表明，计算迁移除了可以较为显著提升应用性能外，在使用 Wi-Fi 进行通信的场景下也能有效降低在移动设备端所消耗的电能，特别是将面部识别这个 CPU 密集型的核心算法迁移到远程执行，可降低算法 90% 左右的能耗。

CloneCloud

MUAI 对代码可迁移性的判断依赖于开发人员手工进行，一方面为开发人员带来了额外负担，另一方面开发人员可能错误地将不可迁移的代码指定为可迁移，例如，将某些访问移动设备特有的传感器等本地资源的代码迁移到云端运行会发生错误。这一问题在 CloneCloud 系统中得以解决 [14]。CloneCloud 使用静态代码分析和动态环境分析相结合的方法，对应用代码进行划分 (partitioning)，划

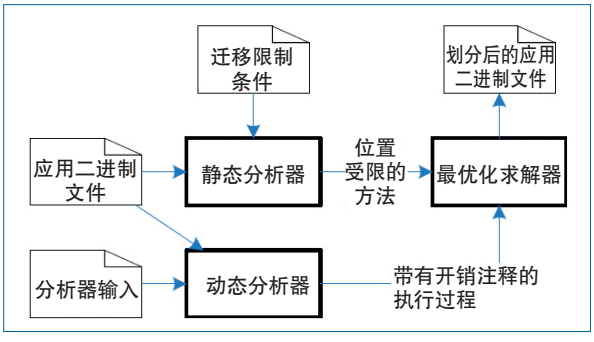


图2　CloneCloud工作流程

分的流程如图 2 所示。其中，静态分析器 (static analyzer) 通过静态代码分析，根据一系列限制条件识别出可以迁移到云端运行的方法，这些限制条件包括 (1) 需要访问本地资源的代码必须留在本地执行；(2) 共享本地状态的代码必须在同一设备上执行；(3) 防止出现嵌套迁移。通过静态分析器划分相当于 MAUI 系统中通过开发人员注释的 Remoteable 方法，因此 CloneCloud 不需要开发人员的参与。动态分析器 (dynamic profiler) 结合本地和云端的网络环境等条件分析应用代码迁移的成本和收益，构造代码迁移的开销函数。最后由最优化求解器 (optimization solver) 给出一个执行时间最短或能耗最低的运行时应用划分方法，将应用的一部分留在本地运行，另一部分迁移到云端运行。

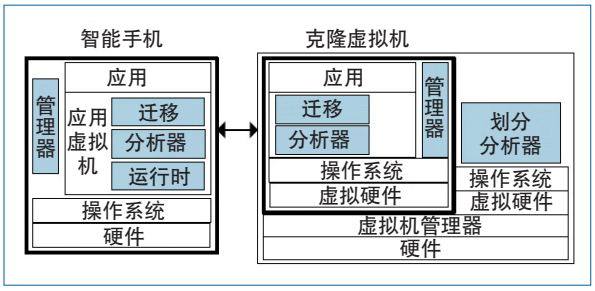


图3　CloneCloud系统结构

图 3 为 CloneCloud 的系统架构。CloneCloud 在云端为移动设备创建克隆的虚拟机。在应用的运行过程中，如果遇到一个迁移节点，正在运行的线程就会被阻塞，它的相关状态信息被发送到云端，由云端继续运行该线程，它可以访问云端的各种资源，例如更快的 CPU、网络，更大的内存等等；本地应用的其他功能（线程）不会受到影响，但如果它们试图访问迁移到云端的线程的相关状态信息，就会进入阻塞状态；当迁移到云端的线程执行完毕时，相关的状态信息被发送回本地，合并到本地被阻塞的线程中去，本地被阻塞的线程将被唤醒并继续执行。

CloneCloud 原型系统在病毒扫描、图像搜索和用户行为追踪等三个应用上开展了实验。实验结果显示，使用计算迁移技术在提升应用运行性能的同时，能在大部分情况下降低应用对移动端设备电能的消耗，特别在计算量大的情况下效果尤为显著。实验中图像搜索应用的输入达到 100 个图像时，搜索过程的能耗降低了 95%。

DPartner

CloneCloud 系统支持自动对代码进行划分并在运行时完成方法粒度的计算迁移，这一技术依赖于对应用运行底层的虚拟机 (Dalvik VM) 进行修改以实现应用计算线程暂停、计算现场获取和运行恢复合并等机制。通过这种方式所实现的计算迁移能力虽强，但对底层机制的修改限制了其适用场景，并降低了用户接受度，因为对虚拟机修改可能导致各类安全性问题，同时也带来了技术可维护性问题，例如，当前安卓平台的底层虚拟机正迁向新一代的 ART，原有 Dalvik VM 上所做的实现能否有效地在 ART 上进行实现还处于未知状态。

北京大学 [15] 提出了基于应用自动化重构实现计算迁移的技术，较为有效地解决了上述问题。安卓应用本质上是由许多类组成的 Java 程序，一项计算任务被实现为某个类中的某个方法，可以被本类或其他类中的方法调用，因此安卓应用计算任务迁移可以实现为包含该计算任务的某个类的远程部署和调用。该工作首先提出了一种支持安卓应用计算任务按需远程执行的程序结构，它主要包含 Proxy 和 Endpoint 两个核心元素，如图 4 所示。它将调用者

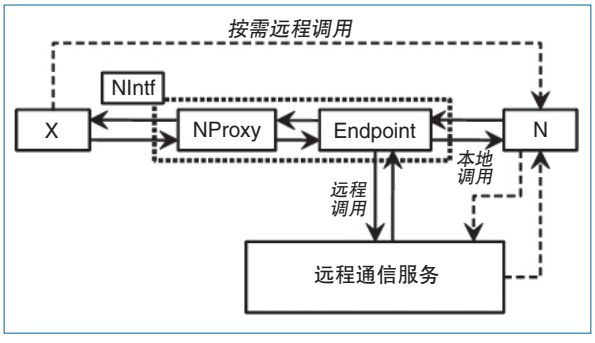


图4　DPartner原理

X 和被调用者 N 之间的直接内存调用以及通过远程通信服务 (Remote Communication Service, RCS) 的远程调用都转换成了经由 Proxy 和 Endpoint 进行的间接调用。NProxy 的外部行为和 N 完全一致，只是它本身不执行任何实际的计算操作，只负责将方法调用转发到 N 执行。Endpoint 负责获取 N 当前的位置并提供 N 的引用供 X 使用。若 N 运行在远程节点，则 Endpoint 会通过远程通信服务获得 N 的远程引用，并把该引用以 NProxy 的形式提供给 X 使用；若 N 运行在本地节点，Endpoint 会直接获得对 N 的内存引用，并同样以 NProxy 的形式提供给 X 使用。

在确定了支持安卓应用计算任务按需远程执行的程序结构之后，该工作设计实现了自动对安卓应用进行重构以符合该程序结构的工具——DPartner。它首先对安卓应用中的类进行分类：必须留在本地执行的 anchored 类（它们使用了一些只能在本地才能获取的特殊资源，如 GUI 或传感器等）和可以被迁移到云端执行的 movable 类；然后进行应用类转换，当一个 movable 类被迁移到云端时，与之交互的类都需要被转换成可按需远程调用的结构，即生成被调用者的代理类 proxy，并重写调用类来使用 proxy。接着进行应用类聚类，为了使计算任务远程执行能够提高应用的性能，降低应用的能耗，必须避免频繁的网络调用所带来的负面影响，因此需要把相互间调用频繁的类作为一个整体迁移到云端执行。最后进行应用类封装，生成两部分制品，一部分是转换后留在本地运行的应用，另一部分是转换后的 movable 类组成的集合，部署到云端以供远程调用。该工作还提出了计算任务是否需要远程执行的运行时决策模型。该模型基于各个应用类的工作负载、内存消耗，应用类之间的交互频率、传输数据量大小，本地和远程节点的处理能力、内存大小，以及它们之间的网络延迟等，以应用的性能与资源消耗的比值为目标，决定最终应用中的计算任务是否需要远程执行。

与 MAUI 等工作类似，作者选取了计算密集型 (Linpack)、交互密集型（3D 赛车）和计算交互兼具型（国际象棋）等三类应用开展实验。实验结果表明，计算密集型安卓应用能降低其在移动设备端 27%~83% 的能耗，交互密集型应用的能耗平均降低仅约 25%，所有类型应用均有较为明显的性能提升。

ThinkAir

在终端和云端间进行计算迁移的主要问题在

于云端缺乏真实的终端运行环境，因此前述工作对应用可迁移部分的限制条件较为苛刻，一般而言仅与移动平台无关的计算部分可迁移到云端执行。为此，索科尔·科斯塔 (Sokol Kosta) 等提出一项称为 ThinkAir[16] 的工作，在云端提供完整的安卓 x86 架构虚拟机环境，以运行从终端迁移而来的计算任务。

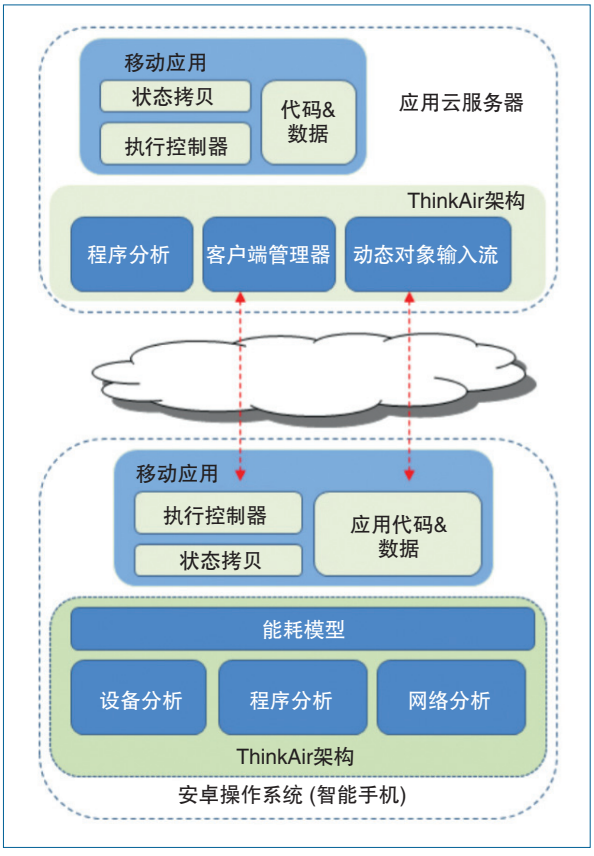


图5　ThinkAir系统结构

ThinkAir 提供一个与 MAUI 类似的简单编程框架，允许用户将某个可进行计算迁移的 Java 方法标注为“@Remote”，并由 ExecutionController 在运行时进行计算迁移的决策。迁移决策过程同样会综合考虑网络质量等环境因素，待迁移方法既往在本地的执行时间和能耗等应用因素，以及商业云资源的使用代价因素等。云端管理器 ApplicationServer 则通过 ClientHandler 构件与终端 ExecutionController 交互完成代码迁移、状态迁移和结果返回等通信过程，并在云端基础设施上开启虚拟机，接收并完成来自终端的计算请求。这些虚拟机运行定制版的安卓 x86 模拟器，因此具有完整的（但不包含科斯塔所裁剪掉的用户界面和内置应用等无用构件）安卓运行环境，可直接运行安卓应用字节码。科斯塔预先创建了六个不同的虚拟机资源配置（CPU、内存、Java 堆区大小）方案，实现为不同资源需求的计算任务提供合适的云端资源，ThinkAir 系统结构如图 5 所示。

科斯塔选取 N 皇后问题、人脸识别、病毒扫描等应用进行实验。实验结果显示，在应用的输入超过某个边界值时，采用云 - 端融合的计算迁移技术可有效提升应用性能，并节省终端的能耗开销达 1~2 个数量级，这与其他工作的结果基本一致。

ThinkAir 还在云 - 端融合的应用场景方面进行了拓展验证。一方面，科斯塔选取了一个图像合并应用，在云端完成移动设备因缺乏足够内存而无法完成的大尺寸图片拼接，验证了云 - 端融合技术对于资源密集型应用运行的有效支持；另一方面，ThinkAir 以参数划分和数据分块等方法，实现对计算任务的并行化，并通过同时启动多个虚拟机克隆并发完成计算任务提高性能的机制，以人脸识别和病毒扫描这两个应用验证了这一机制带来的显著性能提升，将原有简单的计算在端和云之间迁移，扩展为类似多路复用 (multiplexing) 形态的更为灵活高效的计算模式。

CoseDroid

为解决云端与终端运行环境的异构问题，南京大学提出 CoseDroid 框架 [17]，采用比 ThinkAir 更为激进的方式实现计算迁移：让计算迁移发生在移动终端之间。由于不同终端可能存在完全或部分相同的运行环境，因此可迁移部分可更为宽松。在该框架下，一个计算过程（对象方法）是否可迁移仅要求其是 (1) 非交互式的，即该代码执行过程中无须在两个设备间进行消息传递；(2) 安全的，即在满足非交互式要求的前提下，方法在本地和远程设备上执行的结果状态是一致的。作者使用 Soot 工具对代码进行静态分析并寻找满足这两个条件的方法，通

过代码插桩使得应用中的这部分代码可在运行时由CoseDroid框架将代码和序列化的对象状态，从当前设备（主机终端）发送到另一个终端设备（服务器终端）实现计算迁移。系统架构如图6所示。

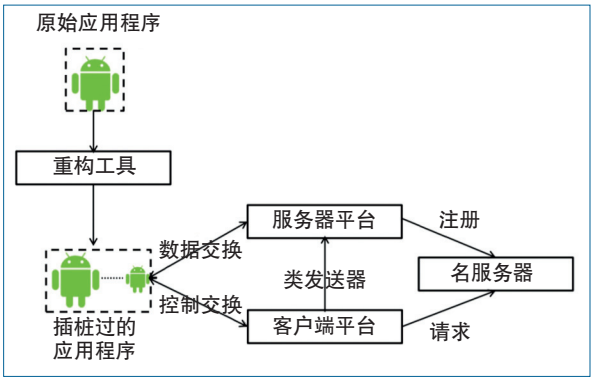


图6 CoseDroid的工作流程

虽然主机终端能耗开销被转移到服务器终端，并且由于网络通信等开销，两个设备合计消耗的电量甚至大于该应用在主机终端上独立运行的开销，但这一模式仍然存在现实意义，即CoseDroid可支持移动终端用户间的“借电”，非紧急用户可将其终端所拥有的电量赠予或出售给紧急用户，让后者在缺乏云端基础设施的场景下，仍能无缝地完成计算迁移，降低自身电量消耗。

在此思路的基础上，CoseDroid进一步实现了传感器“嫁接”(sensor offloading)概念，即允许主机终端使用服务器终端的传感器获取位置、加速度等环境信息。该技术的实现主要基于对安卓平台传感器数据获取模式的观察，通过应用代码插桩，让应用向CoseDroid框架中的虚拟传感器管理器(VSM)进行传感数据申请和监听器注册等过程。虚拟传感器管理器替代真实管理器，在框架通信层支撑下完成在远程服务器终端的传感器管理器进行数据获取申请和监听器注册等对应过程，并将运行时生成的传感器数据传输回主机终端。本质上，主机终端的虚拟传感器管理器被实现为服务器终端真实传感器管理器的代理对象，实现了主机终端应用对远程服务器终端传感器的使用。作者基于该技术改造了著名的涂鸦跳跃(Doodle Jump)游戏，游戏界面运行在主机终端，用户操作运行在服务器终端，验证了技术的有效性。

从能耗优化的角度来看，如果嫁接的是加速度、温湿度和磁场等低功耗传感器，由于通信开销较传感器本身功耗大若干数量级，使用这一技术反而会导致能耗增加。但是，对于某些功耗较大的传感器（如GPS），或在应用需要长时间对传感器数据进行监听获取的场景下，使用CoseDroid也能显著降低应用能耗开销。在作者实验中，一款Shake工具通过CoseDroid平台使用其他移动终端传感器后，能耗节省了约50%。

云–端融合技术问题讨论

在非实验设定场景下，计算卸载的省电效果并不显著，其主要原因是云-端融合研究仍存在较多技术挑战。从终端的角度来看，主要技术难点包括细粒度的可迁移代码划分和全局优化的精准迁移决策。

细粒度的可迁移代码划分 应用实际运行时能耗开销主要来自应用之外的终端平台框架接口调用[5]，底层框架部分代码被调用是完成上层应用逻辑功能的一部分。但当前大部分研究工作仅在应用层面考虑计算迁移，并不考虑框架层计算。代码的可迁移性判断也以是否与本地框架有交互调用为依据，而这类代码在应用中的占比并不高，因而实际运行中的能耗优化效果并不如实验中的设定场景那么明显。因此，进一步细粒度地划分出能进行计算迁移以利用云端资源的代码，是提升云-端融合技术有效性的关键。

全局优化的精准迁移决策 计算迁移的决策一方面受多种因素（包括网络状况、代码能耗预测）影响而相对复杂，并因移动终端网络状况频繁多变且能耗预测无法精确计算（如蜂窝通讯模块的能耗长尾效应等）而无法准确决策；另一方面，决策过程也因终端上多应用多线程的并行运行，难以做到全局优化，迁移既需要考虑单个应用的响应时间等性能指标，又需要从全局角度进行迁移请求的缓冲和合并，以降低通讯开销，优化全局能耗。是否能在两者间做出正确

权衡，也是迁移有效性的重要决定因素。

从云的角度考虑，云 - 端融合的技术挑战主要包括云端和终端的环境异构性屏蔽、普适化的云端资源提供。

云端和终端的环境异构性屏蔽 当前的云 - 端融合技术主要在托管代码环境中实现计算迁移，这一方式在达成实现简单的目标的同时，限制了云 - 端融合技术的应用范围。从云计算技术“按需使用”的基本理念来看，云端资源不仅应该在时间空间上按需可用，还要对资源形式灵活可定制，当终端须迁移的计算环境与云端环境异构不兼容时，云端资源如何通过虚拟化及“软件定义”等技术进行有效适配，是云 - 端融合技术研究的重要挑战。

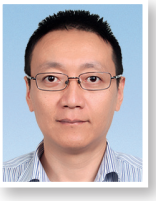
普适化的云端资源提供 当前研究工作的实验结果表明，网络通信延迟因素对能耗优化结果的影响较大，网络通信开销是云 - 端融合中能耗优化效果的主要影响因素，将计算迁移至远程通信开销较大的云端，反而可能带来更大的能耗开销。因此，需要研究如何将上述屏蔽异构性的云资源“随时随地”快速及时地部署并提供给终端用户，使终端能以高效低能耗的方式完成计算迁移，享受普适化云端对终端能力进行平滑无缝的资源扩展，从而突显云 - 端融合计算的性能及能耗优势。■

曹 春

CCF专业会员。南京大学计算机科学与技术系副教授。主要研究方向为Android能耗优化、软件动态更新技术等。
caochun@nju.edu.cn

陆子凌

南京大学计算机科学与技术系研究生。主要研究方向为Android能耗度量方法、能耗优化技术等。
nju10lzl@gmail.com



马晓星

CCF专业会员。南京大学计算机科学与技术系教授。主要研究方向为软件自适应技术等。
xxm@nju.edu.cn

参考文献

[1] Dini G, Martinelli F, Saracino A, et al. MADAM: A Multi-Level Anomaly Detector for Android Malware[C]// *Proceedings of International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security*. Springer Berlin Heidelberg, 2012: 240-253.

[2] Merlo A, Migliardi M, Fontanelli P. On Energy-Based Profiling of Malware in Android[C]// *Proceedings of High Performance Computing & Simulation (HPCS), 2014 International Conference on*. IEEE, 2014: 535-542.

[3] Li D, Halfond W G J. An Investigation into Energy-saving Programming Practices for Android Smartphone App Development[C]//*Proceedings of the 3rd International Workshop on Green and Sustainable Software*. ACM, 2014: 46-53.

[4] Performance Tips, Android Developer Documentation (http://developer.android.com/training/articles/perf-tips.html).

[5] Linares-Vásquez M, Bavota G, Bernal-Cárdenas C, et al. Mining Energy-Greedy Api Usage Patterns in Android Apps: An Empirical Study[C]//*Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014: 2-11.

[6] Pathak A, Jindal A, Hu Y C, et al. What is Keeping My Phone Awake? Characterizing and Detecting No-sleep Energy Bugs in Smartphone Apps[C]//*Proceedings of the 10th international conference on Mobile systems, applications, and services*. ACM, 2012: 267-280.

[7] Liu Y, Xu C, Cheung S C. Where Has My Battery Gone? Finding Sensor Related Energy Black Holes in Smartphone Applications[C]//*Proceedings of Pervasive Computing and Communications (PerCom), 2013 IEEE International Conference on*. IEEE, 2013: 2-10.

[8] Banerjee A, Chong L K, Chattopadhyay S, et al. Detecting Energy Bugs and Hotspots in Mobile Apps[C]//*Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2014: 588-598.

[9] Zhuang Z, Kim K H, Singh J P. Improving Energy Efficiency of Location Sensing on Smartphones[C]// *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 2010: 315-330.

[10]Kim K H, Min A W, Gupta D, et al. Improving Energy Efficiency of Wi-Fi Sensing on Smartphones[C]//*INFOCOM, 2011 Proceedings IEEE*. IEEE, 2011: 2930-2938.

[11] Lu Z, Cao C, Tao X P. Improing Screen Power Usage

Model on Android Smartphones[C]//Proceedings of 2015 Asia-Pacific Software Engineering Conference (APSEC). IEEE, 2015: 167-173.

[12] Dong M, Zhong L. Chameleon: A Color-adaptive Web Browser for Mobile OLED Displays[J]. IEEE Transactions on Mobile Computing, 2012, 11(5): 724-738.

[13] 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. ACM, 2010: 49-62.

[14] 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. ACM, 2011: 301-314.

[15] Zhang Y, Huang G, Liu X, et al. Refactoring Android Java Code for On-demand Computation Offloading[C]// Proceedings of ACM SIGPLAN Notices. ACM, 2012, 47(10): 233-248.

[16] Kosta S, Aucinas A, Hui P, et al. Thinkair: Dynamic Resource Allocation and Parallel Execution in the Cloud for Mobile Code Offloading[C]//INFOCOM, 2012 Proceedings IEEE. IEEE, 2012: 945-953.

[17] Wu X, Xu C, Lu Z, et al. CoseDroid: Effective Computation-and Sensing-offloading for Android Apps[C]//Proceedings of Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual. IEEE, 2015, 2: 632-637.

[18] Flores H, Hui P, Tarkoma S, et al. Mobile Code Offloading: From Concept to Practice and Beyond[J]. IEEE Communications Magazine, 2015, 53(3): 80-88.