一种弱 C/S 架构的计算迁移模型的设计与实现

陈泽堃，李 强†，逯峻雨
（四川大学 计算机学院，成都 610065）

摘 要：近年来提出了许多计算迁移模型，但这些模型都存在部署困难、耦合性高等问题，针对这些问题提出了一种弱 C/S 架构的计算迁移模型。该模型开发了一个面向所有第三方应用的服务端，为第三方应用提供通用的计算服务。该模型不仅部署起来简单便捷，而且服务端和智能手机端低耦合的特性使得该模型对第三方的应用开发人员十分友好。实验结果表明，该模型能有效提升计算密集型应用的执行效率。

关键词：计算迁移；低耦合；计算密集型应用；弱 C/S 架构
中图分类号：TP338.8 **文献标志码**：A **文章编号**：1001-3695(2018)05-1421-05
doi:10.3969/j.issn.1001-3695.2018.05.030

Design and implementation of weak C/S structure computing migration model

Chen Zekun, Li Qiang†, Lu Junyu
(*College of Computer Science, Sichuan University, Chengdu* 610065, *China*)

Abstract: Many models of cloud computing have been proposed in recent years, but these models shared the same problem which was difficult to deploy and high coupling, and so on. For this problem, this papaer proposed a new model for computing migration which had a weak C/S structure. This model developed a server which opened to the third-party application and provided common computing service for them. Not only this model was easy and convenient to be deployed, but it was friendly for third-party application developers because of its low coupling feature of the server and the smart phone. The result of experiment indicates that this model can improve the execution efficiency of the computing intensive applications effectively.

Key words: computing migration; low coupling; computing intensive application; weak C/S structure

0 引言

智能手机的崛起带来了移动互联网的大潮，在近十年内对人类的生活产生了深远的影响，人们的使用习惯已经从传统的个人电脑和其他的电子设备过渡到了智能手机。根据中国互联网络信息中心的报告显示，截至到 2015 年底，中国网民规模达到了 6.88 亿，而其中手机网民达到了 6.2 亿[1]。在享受着移动设备带来便捷性的同时，人们也在承受着移动设备自身性能的局限性所带来的约束和不便。为了迎合越来越广泛的智能手机的功能需求，手机应用开发者开发出了各式各样的手机应用，如增强现实、大型游戏等，这些应用共同的特点是都会消耗大量的计算资源或者存储资源。虽然人们致力于提升手机的性能来解决这些问题，但事实上，移动设备的性能始终处于需求大于供给的状态。此外，尽管现在移动设备的处理能力相较于几年前的确有了很大的提升，但是电池的技术革新速度却远远跟不上处理能力的革新速度，每年电池的容量只能增长 5% 左右，远远不能满足人们对于性能的追求[2]。

正是由于这个趋势，移动云计算(mobile cloud computing, MCC)应运而生，作为智能手机和云计算结合的产物、云计算的核心技术之一，移动云计算在未来也有着广阔的前景[3]。作为一种新型的云计算模式，移动云计算能够为移动设备带来革新，不仅可以通过云平台为移动设备带来更有保障的服务，而且还能摆脱传统的移动互联网的劣势。

作为移动云计算中的一项重要技术，计算迁移是解决移动终端资源受限问题的有效途径[3]。计算迁移技术通过将智能手机等移动设备上的任务通过代码卸载和计算迁移等方法迁移至计算能力和存储能力都相对较强的云端，让移动设备的任务能够更快、更稳定地在云端执行。现有的计算迁移模式虽然能初步解决手机对性能的需求并且能降低能耗，然而由于这些模型的高耦合性以及部署困难等问题，第三方应用想要运用这些模式时不仅成本高，而且维护起来也相当困难。因此为了克服现有的计算迁移模式中的不足与缺陷，本文提出了一种新的计算迁移模型。

1 现有模型介绍

在过去的十年左右时间内，已经有大量的计算迁移相关的研究工作，并提出了许多的计算迁移模型。本文介绍 MAUI、Cloudlet 和 COFA 这三种比较有代表性的计算迁移系统，并讨论其优缺点。

1.1 MAUI 系统

MAUI[4]是基于 surrogate 的计算迁移系统[3]，通过应用开发者来实现代码的管理和决定计算迁移的粒度，最细支持到方法级的迁移粒度，从而最大、最细致地做到计算的迁移。MAUI 的系统架构如图 1 所示，包括两部分，即智能手机端和 MAUI 服务器端。

收稿日期：2017-05-18；**修回日期**：2017-07-17

作者简介：陈泽堃(1991-)，男，福建德化人，硕士研究生，主要研究方向为移动互联、云端融合计算；李强(1963-)，男(通信作者)，副教授，博士，主要研究方向为移动互联、云端融合计算(liq@scu.edu.cn)；逯峻雨(1992-)，男，博士研究生，主要研究方向为移动云计算.

MAUI 通过. NET 实现了运行时的计算迁移,为应用开发者提供了一种程序架构,由应用开发者去决定应用中的哪些部分需要迁移至 MAUI 服务器去执行。MAUI 有以下不足:a)由于 MAUI 基于. NET,所以 MAUI 有跨平台特性,但是该系统仅能用于. NET 应用,基于现在 Android 和 iOS 几乎统领智能手机操作系统的大背景下,该系统已经无法适用于今天的移动云计算环境;b)MAUI 这种客户端/服务器(client/server,C/S)结构使得应用开发者需要同时关注到智能手机和 MAUI 服务器。因此,这种计算迁移模式相对于第三方应用来说不能达到很好的兼容。

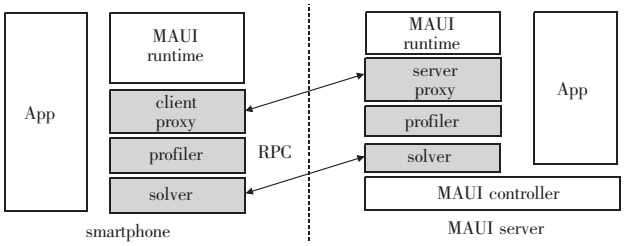

图 1 MAUI 系统架构

1.2 Cloudlet 系统

Satyanarayanan 等人提出了一种计算迁移模式来降低智能手机等移动设备的能耗,文献[5]提出了一种新的概念:Cloudlet。Cloudlet 的设计思路是将移动设备的计算工作迁移至一个就近的局域网内的服务器上去运行。Cloudlet 系统的迁移粒度属于虚拟机(virtual machine,VM)级别,即通过在局域网的服务器上去动态合成一个 VM 来执行移动设备的应用。该系统有以下不足:

a)基于 VM 粒度迁移的优点在于消除了语言上的兼容问题,但随之而来的问题就是 VM 迁移耗时过长。文献[5]中的实验表明,合成 VM 需要 60 ~ 90 s,这样的耗时已经不太适用于大部分的手机应用。虽然文献[6 ~ 9]对合成一个 VM 的时间有所缩短和优化,但是相比于普通的应用开发者来说,开发和维护的成本仍然太高。此外,如果将移动设备转移到另一个局域网,那么需要消耗的代价太大,如果只放在某一个特定的局域网内,却又丧失了移动设备本身的初衷——便携性。

b)动态合成一个 VM 执行环境的方式十分复杂,如果没有统一约束性的策略,不仅部署会很困难,而且在这个过程中将极易发生安全性相关的各种问题。

1.3 COFA 系统

COFA[10] 系统是基于 Android 智能手机的计算迁移系统,并且该系统的迁移粒度属于线程级别。作者通过对 Dalvik 虚拟机进行修改,使得 Dalvik 虚拟机有了通信能力,分为正常模式、服务器模式和客户端模式三种模式,并通过 Dalvik 虚拟机的通信机制来完成线程的迁移。COFA 系统架构如图 2 所示,系统分为服务器和客户端两个部分。COFA 系统的线程调度方式如下:a)将正在客户端执行的线程挂起;b)程序计数器和其他数据被客户端虚拟机捕获并序列化,然后将这些数据发送至服务器虚拟机;c)由服务器虚拟机重新恢复执行线程,在调度后执行该线程;d)最后将结果返回给客户端虚拟机。

COFA 系统的不足有以下几点:a)由于 Dalvik 虚拟机不支持 x86 平台架构,服务器的实现是基于 x86 平台的 Android 模拟器实现的,所以该系统从严格意义上来讲只是 Android 系统之间的迁移,不属于移动云计算的范畴;b)COFA 系统虽然具有线程级别的迁移粒度,但是由于该系统的计算迁移相当于移动设备之间的计算迁移,那么迁移的任务如果不是并行执行的话,并不会显著提高计算迁移的效率,甚至有可能比在本地执行更低。

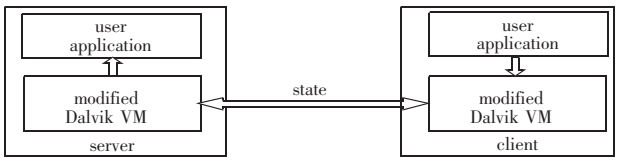

图 2 COFA 系统架构

1.4 小结

研究这些现有的计算迁移系统可以发现,这些系统的架构可分为客户端/服务器模式(C/S)和虚拟化模式两种。这两种方式的比较如表 1 所示。

表 1 现有计算迁移模式对比

比较项	C/S 模式	虚拟化模式
架构特点	需要提供特定的 SDK,与传统的 C/S 架构相差不大	需要为计算迁移开发特定的软/硬件基础设施,如特定的虚拟机等
优点	相对于虚拟化模式来说易于开发部署,安全性可控	由于有特定的执行环境,所以迁移具有更好的动态性和易用性
缺点	迁移不具有动态性,开发手机应用的同时需要服务器端进行匹配的开发	软/硬件基础设施开发难度较高且没有统一的规范,部署费时且困难
典型代表	MAUI	Cloudlet

许多文献中都指出两种不同的计算迁移模式可以不同程度地提升手机的性能、降低能耗。但无论是对于客户端与服务器的高耦合 C/S 模式还是对于需要进行基础设施搭建的虚拟化模式,如果第三方的应用开发人员想要以某种计算迁移的模型来开发或者拓展自己的应用,那么就需要去搭建自己的服务器或者开发特定的虚拟化迁移环境。这对于第三方的应用来说,计算迁移模式开发成本过高、维护困难,两种计算迁移模式都是不友好的。

2 应用场景说明

针对计算迁移来说,计算任务按照完成任务主导条件来分,一般有两种常见的类型:计算密集型(CPU-bound)和 I/O 密集型(I/O-bound)。计算密集型任务顾名思义是指完成该任务所需要的时间主要是由处理器的执行速度来决定,换句话说就是该任务在执行过程中 CPU 的利用率很高,可能在大多数情况下会达到 100% 的情况[11]。I/O 密集型任务是指完该计算任务的时间主要花费在等待输入/输出设备上,换句话说就是该类型任务对数据的请求时间大于对数据的处理时间。

2.1 计算密集型任务的应用场景

1)图像处理应用

图像处理是对图像进行分析、加工和处理,使其满足视觉、心理以及其他要求的技术[12]。典型的应用场景有几何变换、颜色处理、图像融合、降噪、边缘检测等。图像处理属于计算密集型任务,此类应用通常来说都对 CPU 和 GPU 的要求很高,尤其是对于处理能力有限的智能手机等终端设备来说,上传至云端由硬件更为强大的服务器或者服务器集群进行图像处理,则显得更有优势。

2)人工智能应用

人工智能(artificial intelligence,AI)是指由人工制造出来的系统所表现出来的智能[13]。典型的应用场景有机器视觉、

人脸识别、指纹识别、自然语言处理等。此类应用也属于计算密集型任务,对 CPU 的要求很高,尤其是此类应用的计算过程中,大量依赖到各种支持程序的库,因此将此类应用的计算部分放置在智能手机端不太适合,比较适合迁移至服务器或者服务器集群去处理。

2.2　I/O 密集型任务的应用场景

I/O 密集型的应用相比于计算密集型应用来说,需要大量网络资源的传输。从能耗角度来讲,网络传输尤其是在网络条件比较差的情况下会增加手机的能耗,因此相对于计算密集型应用来说,I/O 密集型应用总体上不太适合进行计算迁移。

但是计算密集型和 I/O 密集型的划分具有相对性,因此也会有些特例。例如手机端的杀毒软件,这类应用相对于一般手机应用来说,用于 I/O 的时间较多,但是病毒扫描同样也会给手机带来计算方面的压力,而且病毒库需要大量的存储空间,将病毒库放在智能手机端十分不现实,所以此类问题也比较适合将病毒扫描过程迁移至云端去执行。

3　模型设计

本文提出的计算迁移模型架构包含手机端和服务端两部分。系统架构如图 3 所示。

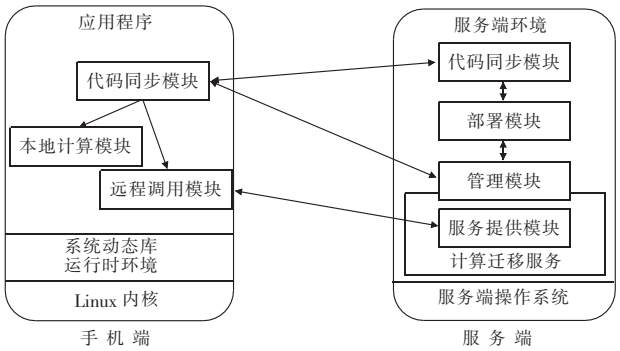

图 3　系统架构图

3.1　手机端部分

手机端部分基于底层为 Linux 内核的 Android 操作系统,第三方的应用开发人员只需要引入供远程调用的库即可完成应用框架的部署。在进行开发时,将应用中需要进行计算迁移的部分分为代码同步模块、本地计算模块和远程调用模块等主要组件,其中组件的主要功能如下:

a)代码同步模块。它是该迁移模型中不可或缺的一部分,主要任务是将 Android 手机端中需要迁移的代码同步至服务端。

b)本地计算模块。它是应用程序控制逻辑作出不迁移的决定或者计算迁移因为某些原因无法完成时,在本地执行任务的模块。该模块保证了应用程序功能的完整性,不会因为特殊的原因导致应用程序该部分功能无法执行。

c)远程调用模块。它同样也是计算迁移中十分重要的一部分,主要任务是在同步模块完成代码远端部署后进行方法的远程调用。

3.2　服务端部分

服务端的组件主要包括代码同步模块、部署模块、管理模块、服务提供模块等组件。组件的主要功能如下:

a)代码同步模块。位于服务端的代码同步模块,主要任务是与 Android 手机端的代码同步模块通信,将 Android 手机端发送的代码保存在服务端指定的位置,以供部署模块下一步的调用及部署。

b)部署模块。它是服务端中十分重要的一个组件,主要任务是完成对已同步代码的修改和部署。

c)管理模块。它是对服务端所提供的服务进行管理的模块,管理模块可以与 Android 代码同步模块通信,也可以与部署模块通信。管理模块管理了服务的产生和消亡。

d)服务提供模块。服务提供模块主要任务是为 Android 手机端提供了远程调用服务,主要由服务开启组件和服务集两部分构成。

3.3　设计说明

通过表 1 的比较可以看出现有的计算迁移系统存在的问题,C/S 模式客户端与服务端的高耦合,虚拟化模式需要进行基础设施搭建,这对第三方应用开发都不太友好。

仅仅为单一的应用开发提供计算迁移服务的服务端,会增加额外的开发和维护成本。因此开发一个面向所有的第三方应用的服务端是可行的,这种开发模式不仅可以减少第三方应用的开发成本,而且可以对服务端进行统一的服务管理。

所谓的弱 C/S 计算迁移模型,是指在计算迁移过程中智能手机端和服务端呈现一种松散耦合关系。服务端作为计算服务的提供者,不需要部署任何与智能手机端业务相关的代码,而是为第三方应用提供一种通用的服务。第三方应用开发者运用服务端提供的开发模型与约束开发其手机客户端,客户端通过自身业务逻辑将自身需要迁移的计算任务动态地同步至服务端,在服务端生成对应的计算迁移服务,然后经过远程调用完成计算密集型任务的远程执行,从而降低这部分任务的执行时间和手机功耗。这样设计计算迁移模型是为了让计算迁移成为一种流行的服务形式,对第三方的应用及其开发人员提供便利,主要特点如下:

a)在某个手机应用第一次发起计算迁移请求前,负责提供计算迁移功能的服务端不存在任何与该应用相关的计算模块;

b)第三方应用开发者不用考虑如何开发对应的服务器,只需要遵循服务端所提出的规范;

c)服务端在第一次接收到该应用的计算迁移请求后,根据应用提供的计算代码在服务端动态地部署好;

d)计算迁移服务端,作为一个计算服务的提供者,可以为各种不同的应用提供“即插即用”的计算迁移服务,而不仅仅作为某一个应用独占的服务器。

4　模型的实现

4.1　手机端的实现

本文模型中手机端的实现基于 Android 操作系统。之所以采用 Android 作为手机端应用的主系统,主要是因为 Android 操作系统是世界上最广泛的手机操作系统,十分具有代表性;此外,由于该系统具有的开源特性,应用程序拥有着更好的扩展性。

1)代码同步模块

代码同步模块需要 Android 应用开发人员预先将可迁移的代码部署在应用运行过程中可读的文件内,此过程需要遵循几点规范:a)为需要迁移的方法提供一个包含该方法的接口,

该接口名为 OffloadService;b)将需要部署的方法与上下文方法打包成一个名为 OffloadServiceImp 的类,并继承 OffloadService 接口;c)需要迁移的方法只能声明为 public 成员方法,不能声明为类方法。

在第三方应用程序作出进行计算迁移的决定后,代码同步模块将与服务端建立 TCP(transmission control protocol)连接,通过基于 HTTP(hypertext transfer protocol)的 Web service 接口,将预先部署好的代码同步至服务端。

应用程序控制逻辑可以灵活地决定代码同步的时间,既可以在应用启动后马上进行代码同步工作,也可以在应用启动后的任何时间,甚至在该段代码执行前同步至服务器。这样决定代码的同步时间,为第三方应用的计算迁移带来了极大的灵活性。

2)远程调用模块

远程调用模块是基于 Java 中传统的远程方法调用(remote method invocation,RMI)技术实现的。由于 Android 应用运行在 Dalvik 虚拟机上,可执行文件是. dex 文件,而在服务端的运行环境是标准的 Java 虚拟机(Java virtual machine,JVM),可执行文件是. class 文件,传统的 RMI 无法实现远程调用。

为了解决手机端和服务端在相互调用中的不兼容问题,该模型建立了统一的远程调用的基本单位:RemoteMessage 接口和 RemoteInstance 类,RemoteMessage 和 RemoteInstance 实现了 Serializable 接口,从而使得其子类能够实现序列化传输。RemoteCall 和 RemoteReturn 类作为调用和返回的基本单位,实现了 RemoteMessage 接口。

此外,该模型建立了持有客户端 Client 类与 Server 服务端类。在客户端,Client 持有一个 HashMap 数据结构,用来保存需要远程调用的 RemoteInstance 对象实例,而在服务端也持有一个 HashMap 数据结构,用来保存远程实例的代理对象。客户端 client 与服务端 server 通过基于 TCP 的可靠连接实现了通信,从而实现了手机端与服务端的正常远程调用。

4.2 服务端的实现

本文提出的计算迁移模型的服务端只需部署最新版的 JDK(Java development kit)运行环境和 Apache 服务器与 PHP(hypertext preprocessor,PHP)环境,并进行简单的部署和配置即可完成服务端的搭建。服务端的实现主要有两部分:a)以 PHP 语言实现的组件,包括代码同步模块、部署模块、管理模块;b)以 Java 语言实现的组件,包括服务提供模块。

1)部署模块

为了实现代码的部署,该模块需要借助一些工具代码,这些代码存放在已经预先配置好的指定目录下的 serverTools 文件中。部署模块在代码完成同步后,读取已同步的代码与工具代码,动态地将两部分进行修改和组合,从而完成服务端代码的部署。

在代码部署过程中也需要遵循一些规范:a)第三方的应用开发人员必须在代码同步时发送代码所属类所在包的全名,如果 Android 手机端和服务端两处包名不统一,则无法完成部署;b)服务端需要根据包名为不同的计算迁移服务建立不同的存储路径。

2)服务提供模块

服务提供模块中的服务开启组件是由管理模块来驱动的,当服务端的部署模块完成代码部署后,发送消息给管理模块,管理模块启动服务开启组件,从而开始对部署好的代码进行编译与运行,最终生成一个远程调用服务。每个服务可以提供多次服务,也就是说,相同的第三方应用可以多次调用同一个服务。此外,一个服务不会自动停止运行,只有当管理模块发出停止指令后,该服务才会停止。

每一个远程调用服务都会占用一个服务端的端口,因此一个服务端可以持有多个服务,这些服务构成了服务集。服务集的大小取决于可用端口数量和服务端的处理能力。

4.3 系统执行过程

完成一次完整的计算迁移的执行过程如图 4 所示,Android 手机端模块和服务端模块由中间的虚线隔开。

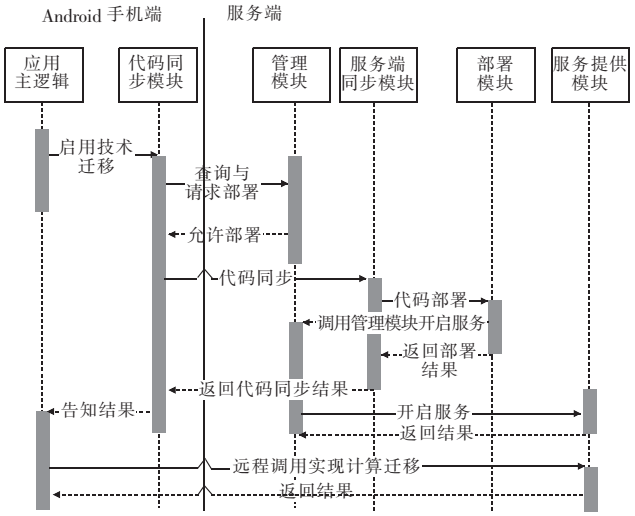

图 4 计算迁移执行过程

需要说明的是,在服务端的部署模块进行代码部署的时候,在此过程中调用了服务端的管理模块,目的是为了调用服务提供模块,对部署好的代码进行编译和运行。之所以这样做是因为服务端程序在运行编译好的字节码文件时,会因为服务处于持续开启状态而一直阻塞在那个状态,所以需要重新对服务端的管理模块产生一个新的 HTTP 请求,这样做可以避免部署模块阻塞,及时对手机端进行应答。新的 HTTP 请求的功能类似于开启一个新的线程。

每个服务可以提供多次调用,也就是说相同的第三方应用即使在不同的手机端都可以多次调用同一个服务。如果在 Android 手机端的代码同步模块查询和请求部署阶段,发现服务端已存在需要的服务时,可以省略代码的同步、部署、编译及运行等阶段,直接进入远程调用阶段。远程服务已存在的情况下,计算迁移任务的执行时间就是远程服务执行的时间,可以很好地满足服务的实时性要求。远程服务不存在的情况下,远程服务的部署、编译及运行有一定的耗时,这也是计算迁移相对于本地执行多出来的一个阶段。因此如果代码同步提前进行、服务提前部署好,远程执行的任务的执行时间就直接等于远程调用服务的时间,减少了代码同步的时间,可以更好地保证服务的实时性。

在本模型中,手机端开发人员可以灵活地决定代码同步的时间,可以在应用启动后马上连接服务端进行代码同步工作,也可以在任务执行之前。如果某个业务实时性要求很高,手机端开发人员可以在应用启动过后立刻进行同步工作,当然如果这个业务最后没执行就会做一些无用功;如果业务实时性要求没那么高或者本身这个业务就耗时,代码同步时间远小于服务

执行的时间,可以在任务执行的时候再进行代码同步,避免做无用功,手机端开发人员自己权衡。这样根据不同的业务需求灵活决定代码的同步时间,可以满足不同的服务要求。

5　实验结果与分析

5.1　实验环境

本文的实验环境主要参数如表 2 所示,网络环境是 IEEE 802.11bgn 无线网络。

表 2　实验环境参数

参数项	Android 手机端	服务端
型号	OPPO R829T	联想 ThinkPad T450
操作系统	Android 4.2.2	Windows 10
处理器	联发科 MT6582	Intel Core i7-5500U
处理器主频	1.3 GHz	2.4 GHz
运存	1 GB	8 GB

5.2　实验说明

本文以在 Android 手机端的哈夫曼编码为例进行实验,使用哈夫曼编码作为计算迁移中的计算部分,主要有以下特点:

a)当所需编码的字符串固定以后,每次编码的结果也一样,所以哈夫曼算法具有稳定性,实验结果也具有稳定性和可重复性。

b)哈夫曼编码具有 I/O 密集型任务的特点,可以通过增加所编码的字符串的长度来增加网络资源传输量。

c)可以通过控制哈夫曼编码的主体算法循环次数来实现计算量的控制。如果增加循环次数,则意味着算法的计算量也相应增加。

为了避免在实验过程中因为出现某些特殊原因而影响实验结果,本文所采用的方法是:在相同条件下进行 12 次实验,并记录每次的数据,舍弃掉最高和最低的数据,最终求得中间 10 次实验数据的平均值,从而达到较为客观的实验结果。

由于本文所提出的计算迁移模型特点所致,所迁移代码的传输、部署、运行需要一定的预部署时间,所以在计算量较小的实验条件下会出现使用迁移来进行计算的任务执行时间反而比在本地执行时间长的情况。

5.3　实验结果

本文以大小为 100 Byte 和 15 KB 的文件作为哈夫曼编码的对象,通过控制哈夫曼编码主体算法的循环次数来测试不同条件下的计算迁移性能。

图 5 ~7 所示的分别是大小为 100 Byte、1 KB 和 15 KB 的文件编码的实验结果,图中横坐标代表主体算法循环次数,纵坐标代表编码执行时间,虚线代表的是使用计算迁移的任务执行时间,实线代表的是在本地的任务执行时间。

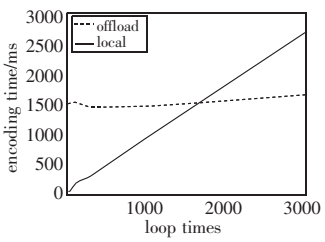

图 5　100 Byte 文件编码

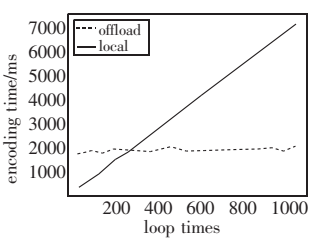

图 6　1 KB 文件编码

5.4　结果分析

本文提出的计算迁移模型,相比于其他计算迁移模型来说,由于实验环境不同、实验采用的算法不同,因此本文实验结果只针对本文提出的计算迁移模型,与其他文章中的实验结果没有统一的衡量标准,不具有可比性。

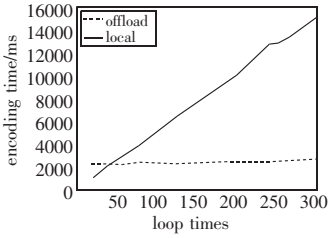

图 7　15 KB 文件编码

实验结果表明:对于大小为 100 Byte 的文件,由于一次编码计算量较低,所以当循环次数较少的时候,本地任务执行比计算迁移更有效率,执行时间更短。对于 1 KB 的文件,执行一次编码计算量大一些,在 Android 客户端执行压力较大,在循环 250 次左右后通过计算迁移执行编码任务就可以比本地编码所需的时间更短。而对于大小为 15 KB 的文件,因为执行一次编码的计算量最高,所以对于在 Android 手机端本地执行的压力更大,在循环次数大于约 40 次以后,通过计算迁移技术执行编码任务就可以比本地编码所需的时间更短,效率也更高。

6　结束语

1)总结

本文就现阶段智能手机与移动云计算的发展前景提出了计算迁移技术的可行性,并分析了国内外现存的计算迁移模型的优缺点。在此基础上,本文基于 Android 平台提出了一种弱 C/S 架构的计算迁移模型。该模型吸收了传统 C/S 架构迁移模型的细粒度优点,并摈弃了虚拟化模型的部署困难与高耦合的缺点。相比于虚拟化模式的计算迁移,该模式部署简单方便,对第三方的应用及开发者十分友好。

此外,本文基于哈夫曼编码,利用该计算迁移模型与传统的本地执行方法做了对比实验。实验结果表明,在计算量大于某个阈值后,利用该计算迁移模型来进行任务的远程执行,比在本地执行效率更高、时间更短。

2)展望

相比于前人所提出的计算迁移方法,本文虽然取得了一定的进步,但是该模型仍有许多方面需要改进与拓展。

a)服务端组件拓展。

本文的计算迁移模型中,服务端组件管理模块暂时只能进行简单的服务开启与关闭功能。一个第三方的应用如果多次请求进行计算迁移,每次都需要进行部署,那么将会有大量的时间与能耗浪费在相同的代码部署上。如果服务端组件能智能地根据应用的不同来维护不同的迁移服务,并且服务能根据应用需要实时更新,那么将会为相同应用的多次调用节省大量的时间与能耗。

b)计算迁移服务提供。

本文提出的计算迁移模型旨在为第三方的应用及其开发者提供便利的计算迁移服务。如果能够统一为不同的应用提供计算迁移服务,那么对于移动端的开发人员来说,服务端的部署与开发成本将降低至最小甚至是零,他们就能够专注于移动应用的开发。这样做能够促使计算迁移成为移动端一种常见的开发模式,而不仅仅是存在于实验室或某些小范围的应用中。

(下转第 1449 页)

元局、工贸、四公里),当 INPUT(输入)值修改至目标值时,OUTPUT(输出)值可达到目标值。目标值与初始值对比结果显示,各换乘站点的首末班线路数均未改变,表明该项 INPUT(输入)指标对换乘效率最大化的贡献率较低,无须对该项指标进行调整。而各换乘站点公交线路数与站台车位数普遍偏少,其中华新街、铜元局、四公里站亟待增加运营公交线路与站台车位,提高公交运能,接驳从轨道车站至公交站点换乘的大量乘客,从而提高换乘客流比。观音桥、牛角沱、铜元局站换乘距离过长,在具备一定的条件基础上,可考虑建设轨道公交换乘一体化设施,缩短乘客步行距离,协调不同方向客流组织,避免客流交织及减少冲突点,以实现缩短换乘时间,提高乘客满意度和忠诚度的最终目的。

表 3　各站点指标目标值与初始值对比

指标	站点					
	观音桥	华新街	牛角沱	铜元局	工贸	四公里
公交线路数 B_1	--	+17.000	+8.040	+21.000	+3.000	+18.895
首末班线路数 B_2	--	--	--	--	--	--
公交站台车位数 P	+4.400	+6.000	+4.224	+5.000	+2.000	+3.947
空间距离 D/m	-134.538	-23.333	-10.517	-140.000	--	-19.522
换乘距离 L/m	-202.000	-33.000	-149.368	-284.000	-66.000	-161.368
换乘时间 T/min	-3.720	-6.200	-7.659	-9.900	-4.953	-8.279
换乘综合满意度 S	+2.473	+1.408	+0.225	+2.908	+0.189	+2.551
换乘客流比 R	+0.189	+0.190	+0.024	+0.310	+0.262	+0.291

备注:+表示目标值对应原始值增加数,-表示目标值对应初始值减少数,--表示目标值与原始值相等。

4　结束语

近年来,城市轨道交通发展速度加快,如何与原有常规公交系统实现高效的协调换乘,提高城市交通运转效率和资源利用率,进而提升旅客运输服务质量是运输行业内值得探讨的重要课题。本文在综合分析轨道交通与常规公交换乘效率影响因素的基础上,选取了八个内部换乘效率的测评指标,并根据指标内容能否调整划分为输入指标和输出指标,基于 DEA 方法构建了城市轨道交通与常规公交换乘效率的综合测评模型。对重庆市城市轨道交通 3 号线嘉州路—四公里段 10 个轨道车站及站旁公交站点的相对效率进行测评计算,得到其换乘效率评价指数和在有效前沿面上的投影,从而实现量化辨识公交站点设施投入与轨道车站接驳是否匹配的问题,为轨道及公交衔接组织研究提供了理论依据。但模型在各车站的适用性及测评指标选取的客观性还有待进一步研究。

参考文献:

[1] 陈旭梅,林国鑫,于雷. 常规公共交通与轨道交通运营调度协调模型[J]. 系统工程理论与实践,2009,29(10):165-173.

[2] Iseki H, Taylor B D. Not all transfers are created equal: toward a framework relating transfer connectivity to travel behavior[J]. Transport Reviews,2009,29(6):777-800.

[3] Algers S, Hansen S, Tegner G. Role of waiting time, comfort, and convenience in modal choice for work trip[J]. Transportation Research Record,1975,534:38-51.

[4] Wardman M, Hine J, Stradling S. Interchange and travel choice-volumes 1 and 2[J]. Transport Research,2001.

[5] 郭淑霞,陈旭梅,于雷,等. 轨道交通换乘常规公交平均候车时间模型[J]. 交通运输系统工程与信息,2010,10(2):143-147.

[6] 孙凡松,张开冉,王若成. 交叉口处轨道交通与常规公交的换乘模型研究[J]. 交通运输工程与信息学报,2015,13(3):76-80.

[7] 许旺土,何世伟,宋瑞. 轨道交通接运公交发车间隔及票价优化模型[J]. 吉林大学学报:工学版,2009,39(6):1469-1474.

[8] 覃煜,晏克非. 轨道交通与常规公交衔接系统分析[J]. 城市轨道交通研究,2000,3(2):44-48.

[9] 陈坚,钟异莹,陈林,等. 城市轨道交通车站内部换乘效率测评模型[J]. 铁道科学与工程学报,2016,13(3):577-582.

(上接第 1425 页)

c)安全性问题。

安全性问题是互联网、移动互联网中常见的问题。在为用户带来计算迁移便捷性的同时,如何保证计算过程中数据的传输、用户隐私的隔离与保护以及如何保证在提供统一化的服务中,服务端是否会因为某些恶意代码的部署而产生拒绝服务的情况都是本文进一步努力的方向。

参考文献:

[1] CNNIC. 第 37 次中国互联网络发展状况[EB/OL]. (2016-01-22)[2017-07-08]. http://www.cnnic.net.cn/gywm/xwzx/rdxw/2016/201601/W020160122639198410766.pdf.

[2] Othman M, Hailes S. Power conservation strategy for mobile computers using load sharing[J]. ACM SIGMOBILE Mobile Computing and Communications Review,1998,2(1):44-51.

[3] 张文丽,郭兵,沈艳,等. 智能移动终端计算迁移研究[J]. 计算机学报,2016,39(5):1021-1038.

[4] Cuervo E, Balasubramanian A, Cho D, *et al*. MAUI: making smartphones last longer with code offload[C]//Proc of the 8th International Conference on Mobile Systems, Applications, and Services. New York: ACM Press,2010:49-62.

[5] Satyanarayanan M, Bahl P, Caceres R, *et al*. The case for VM-based Cloudlets in mobile computing[J]. IEEE Pervasive Computing,2009,8(4):14-23.

[6] Verbelen T, Simoens P, De Turck F, *et al*. Cloudlets: bringing the cloud to the mobile user[C]//Proc of the 3rd ACM Workshop on Mobile Cloud Computing and Services. New York: ACM Press,2012:29-36.

[7] Verbelen T, Simoens P, De Turck F, *et al*. Adaptive application configuration and distribution in mobile cloudlet middleware[C]//Proc of International Conference on Mobile Wireless Middleware, Operating Systems, and Applications. Berlin: Springer,2012:178-191.

[8] Verbelen T, Simoens P, De Turck F, *et al*. Adaptive deployment and configuration for mobile augmented reality in the Cloudlet[J]. Journal of Network and Computer Applications,2014,41(5):206-216.

[9] Lewis G A, Echeverría S, Simanta S, *et al*. Cloudlet-based cyber-foraging for mobile systems in resource-constrained edge environments[C]//Proc of the 36th International Conference on Software Engineering. New York: ACM Press,2014:412-415.

[10] Shivarudrappa D, Chen Minglung, Bharadwaj S. COFA: automatic and dynamic code offload for Android[R]. Boulde: University of Colorado, 2011.

[11] CPU-bound [EB/OL]. (2016-11-20)[2017-07-08]. https://en.wikipedia.org/wiki/CPU-bound.

[12] Gonzalez R C, Woods R E, Eddins S L. Morphological image processing[C]//Digital Image Processing. 2008:627-688.

[13] Artificial intellige[EB/OL]. (2016-12-19)[2017-07-08]. https://en.wikipedia.org/wiki/Artificial_intellige.