

**UNIVERSIDAD NACIONAL DE SAN AGUSTIN DE
AREQUIPA**

ESCUELA DE POSGRADO

**UNIDAD DE POSGRADO DE LA FACULTAD DE
INGENIERÍA PRODUCCIÓN Y SERVICIOS**

**POTENCIANDO LOS DISPOSITIVOS MÓVILES MEDIANTE
TECNICAS DE OFFLOADING EN ARQUITECTURAS
CLOUDLET**

Tesis presentada por el Bachiller Enrique Arturo Soto Mendoza,

para optar por el Grado de Maestro en Informática,

con mención en Tecnologías de la Información,

Asesor: Alvaro Mamani Aliaga

Co-Asesor: Arlindo Flavio da Conceição

AREQUIPA-PERU

2016

Potenciando los Dispositivos Móviles mediante Técnicas de *Offloading* en Arquitecturas *Cloudlet*

El presente documento es una versión sometida a calificación
del jurado de Enrique Arturo Soto Mendoza

Resumen

El uso de *offloading* basado en nube de tareas costosas puede mejorar el rendimiento de una aplicación y reducir el consumo de energía de un dispositivo móvil, dicho objetivo se logra migrando la ejecución de código a servidores más potentes en la nube. Sin embargo, el *offloading* a la nube no siempre es la mejor opción, ya que de por medio existe una latencia elevada en entornos WAN. En este trabajo se propone Pytos, un framework de *offloading* para Python, el cual consiste de una arquitectura jerárquica de 3 niveles: 1) nube, 2) *cloudlet* y 3) dispositivo móvil. Este sistema permite la migración de código al borde de la red (*cloudlet*), mitigando las limitaciones de latencia. Pytos provee un mecanismo transparente de *offloading* al usuario final, ya que los *cloudlet* disponibles en una red serán habilitados automáticamente. Para los desarrolladores, Pytos provee un modelo simple de programación a una granularidad a nivel de métodos. Las evaluaciones empíricas iniciales mostraron que Pytos puede mejorar el rendimiento y consumo de energía hasta 20 veces.

Palabras clave: computación en nube para móviles, offloading de cómputo, framework, *cloudlet*, Python

Abstract

Cloud-based offloading may improve mobile application performance and reduce energy consumption by migrating mobile code execution to the cloud. Nevertheless, offloading to the cloud is not always the best bet, because of the high WAN delay. This work presents Pytos, a framework for Python, that allows code migration to the network edge (cloudlet), mitigating latency limitations. Pytos provides a transparent offloading mechanism to end users, hence available cloudlets in the network are discovered automatically. To developers, Pytos provides a simple programming model at method-level granularity. Empirical evaluation showed that Pytos can improve performance and energy consumption up to 20 times.

Key-words: mobile cloud, computation offloading, framework, cloudlet, Python

Índice general

Resumen	II
Abstract	III
Contenido	IV
Indice de Figuras	VI
Indice de Tablas	VII
Abreviaciones	VIII
1. Introducción	1
1.1. Motivación	4
1.2. Objetivos	4
1.3. Trabajos Relacionados	5
1.4. Organización del Documento	8
2. Conceptos	9
2.1. Computación Móvil	9
2.2. Computación en Nube	11
2.3. Computación en Nube para Móviles	13
2.3.1. Visión	13
2.3.2. Heterogeneidad	14
2.3.2.1. Taxonomía de la heterogeneidad en MCC	15
3. Offloading de Cómputo	22
3.1. Offloading	24
3.1.1. Viabilidad de <i>offloading</i>	25
3.1.1.1. Mejora del rendimiento	26
3.1.1.2. Conservación de la energía	27

3.1.2.	Factores de decisión en <i>offloading</i>	29
3.1.2.1.	Características del dispositivo móvil	29
3.1.2.2.	Características de la aplicación	30
3.1.2.3.	Características de red	30
3.2.	Algoritmo de offloading	31
3.3.	Criterios para la clasificación de Offloading	32
3.3.1.	Particionado de la Aplicación	34
3.3.2.	Tipos de Offloading	35
3.3.3.	Plataforma de despliegue	36
3.3.4.	Campo de Aplicación	36
4.	Cloudlets	39
4.0.5.	Arquitectura Cloudlet	40
5.	Propuesta: Sistema Pytos	43
5.1.	Arquitectura de Pytos	43
5.2.	Diseño de Pytos	47
5.3.	Modelo de Programación	48
5.3.1.	Flujo del Sistema	49
5.4.	Implementation	49
5.4.1.	Recopilador	50
5.4.2.	Evaluador	51
5.5.	Resultados Iniciales	51
5.5.1.	Configuración experimental	51
5.5.2.	Escenarios de Evaluación	53
5.5.3.	Resultados Iniciales	54
6.	Cronograma de Trabajo	57
7.	Conclusiones	58
	Bibliografía	60

Índice de figuras

2.1.	Taxonomía de los enfoques de mejora de en móviles. Figura adaptada de [ASG12]	14
2.2.	Una vista general a MCC [SAGB14]	15
2.3.	Taxonomía de los orígenes de la heterogeneidad en MCC [SAGB14]	16
2.4.	Heterogeneidad de plataformas en MCC y la difícil tarea para el desarrollador elegir la correcta [SAGB14]	19
3.1.	Consumo de energía por Bit de transferencia. WLAN vs 3G. Experimento realizado en celular Nokia N900 [MN10]	31
3.2.	Proceso general de <i>offloading</i> . Figura Adaptada de [KOMK14]	32
3.3.	Arquitectura del <i>framework</i> Cuckoo [KPKB12]	34
4.1.	Diferencias entre las arquitecturas basadas en nube y las basadas en cloudlet [SBH ⁺ 13]	42
5.1.	Vista global de la arquitectura propuesta de Pytos	45
5.2.	Detalles de interacción entre los componentes de Pytos	47
5.3.	Flujo del sistema de Pytos	50
5.4.	Metodología para la medición	53
5.5.	Comparación de tiempo de respuestas usando diferentes escenarios.	54
5.6.	Comparación de energía consumida usando diferentes escenarios.	55

Índice de cuadros

3.1. Algunas soluciones para el offloading de aplicaciones móviles	33
4.1. Diferencias más notorias entre los entornos <i>cloudlet</i> y <i>cloud</i> [SBCD09]	40
5.1. Datos de los diferentes conjuntos de imágenes usados en la aplicación de reconocimiento de rostros	54
6.1. Plan de trabajo	57

Abreviaciones

MCC	Mobile Cloud Computing
TI	Tecnologías de la Información
QoS	Quality of Service
NIST	National Institute of Standard and Technology
IoT	Internet of Things
API	Application Programming Interface
CISC	Complex Instruction Set Computer
RISC	Reduced Instruction Set Computer
PHP	Hypertext Preprocessor
MP	Mega Pixels
OS	Operating Systems
RPC	Remote Procedure Call
HTTP	Hiper Text Transfer Protocol
SaaS	Software as a Service
PaaS	Platform as a Service
IaaS	Infrastructure as a Service
VM	Virtual Machine

Capítulo 1

Introducción

En los últimos años, las tarjetas de desarrollo destinadas a los dispositivos *wearables* [PLJC14] están ganando popularidad entre los desarrolladores. Nuevas aplicaciones para estos sistemas embebidos están siendo desarrolladas como realidad aumentada, reconocimiento de rostros, procesamiento de lenguaje natural, video juegos, software de modelado en 3D, entre otros. Dichas aplicaciones usualmente son ávidas de recursos, requiriendo un cálculo de cómputo intensivo y un alto consumo de energía. Por otro lado, los dispositivos móviles, debido a su naturaleza portátil son limitados de recursos en términos de poder computacional y duración de batería [VRC13].

A pesar del notorio avance tecnológico en términos de hardware en Computación Móvil, la densidad de energía en baterías ha sido relegada y ha tenido una tendencia menor en comparación con los componentes restantes [PS05], consecuentemente la calidad de la experiencia de usuario es reducida. De acuerdo al estudio titulado “2012 Satisfacción de los Clientes de Smartphones en Estados Unidos”¹, la duración de la batería es el aspecto menos satisfactorio en los smartphones, calificado con un valor de 6.7 sobre 10. Por lo tanto, la ejecución fluida de estas tareas costosas en tales dispositivos limitados es un reto abierto [SAGB14].

¹ <http://www.jdpower.com/press-releases/2012-us-wireless-smartphone-and-traditional-mobile-phone-satisfaction-studies-volume>, último acceso en setiembre 2015

Inicialmente la Computación en nube para móviles (MCC, Mobile Cloud Computing) [RRL⁺14] fue propuesto como una opción prometedora para solucionar tal reto, proponiendo mecanismos de *offloading* para aplicaciones a fin de aliviar la sobrecarga de los recursos limitados en los sistemas embebidos. Esta meta es satisfecha integrando los recursos en nube en el entorno móvil de manera bajo demanda y elástica. En tal integración, las tareas intensivas son enviadas a servidores de cómputo de alto rendimiento en la nube, por lo que, MCC podría reducir el tiempo de respuesta de un algoritmo y ahorrar energía reduciendo procesamiento local.

Sin embargo, en entornos reales MCC presenta una cuestión crucial para ser resuelta: la elevada latencia en las Redes de Área Amplia (WAN, Wide Area Network) [SBCD09]. Considerese el escenario donde se tenga uno de los más potentes computadores en la nube, pero supongamos que desafortunadamente el ocasional proveedor de servicio (ISP, Internet Service Provider) no puede establecer una conectividad sin problemas en el área. Es altamente probable que tal limitación implicará una pérdida de paquetes, haciendo que la experiencia del usuario se vea degradada.

Con el objetivo de solucionar dicho inconveniente, los *cloudlets* [SCH⁺14] fueron propuestos como una nube más próxima al usuario final. Tal proximidad se logra desplegando dicho *cloudlet* en la Red de Área Local Inalámbrica (WLAN, Wireless Local Area Network), de tal forma que los *cloudlets* asisten a los dispositivos limitados en la red local actual con mecanismos de *offloading* sin ser perjudicados por la alta latencia en entornos WAN. La arquitectura *cloudlet* fue modelada por Satyanarayanan *et. al.* [SBCD09], donde tres principales características son resaltadas:

- **Estado flexible:** Es uno de los atributos más importantes. Una vez que el *cloudlet* es instalado, es enteramente auto-administrado y requiere poca o ninguna asistencia profesional.

- **Potente y con buena conexión a Internet:** Es una computadora rica en recursos o un cluster de computadoras disponible para ser usada por dispositivos cercanos. Similarmente, los *cloudlets* tienen una eficiente y confiable conexión a Internet, usualmente a través conexión cableada.
- **Disponible para ser usado por dispositivos móviles cercanos:** Está lógicamente cerca a los dispositivos, esto significa que cualquier dispositivo móvil ligado a la red WAN tiene una baja latencia con el *cloudlet* y un elevado ancho de banda disponible para transferir gran cantidad de datos.

El *offloading* de computación es una técnica que potencia las capacidades de los sistemas móviles (p. ej. wearables, teléfonos móviles, sensores, internet de las cosas, entre otros dispositivos portátiles). Dicha mejora es realizada migrando el programa entero o solo sus cálculos intensivos a computadoras más potentes ubicadas en la nube o en los *cloudlets* (infraestructura próxima). En contraste con la arquitectura cliente-servidor, la mayoría de los frameworks para *offloading* efectúan una decisión inteligente acerca de qué enviar y cómo migrar tareas. Esta decisión es ejecutada por algoritmo en segundo plano que considera algunas variables al contexto móvil: (1) disponibilidad del servidor, (2) ancho de banda de la red inalámbrica, (3) tamaño de datos enviados a través de la red (4) energía gastada por el proceso (en estado ocioso y en estado de ejecución), (5) velocidad de procesamiento en el servidor, (6) velocidad de procesamiento en el móvil, entre otros factores subyacentes a la heterogeneidad de MCC [SAGB14].

Con el fin de tomar ventaja del auge de las placas de desarrollo disponibles en el mercado [PLJC14] (Intel Edison Board ², Raspberry Pi ³, Intel Galileo ⁴, entre otros), y aprovechando que tales plataformas soportan distribuciones ligeras de Linux, se propone el sistema Pytos,

²<http://www.intel.com/content/www/us/en/do-it-yourself/edison.html>, último acceso en setiembre 2015

³ <https://www.raspberrypi.org/>, último acceso en setiembre 2015

⁴<http://www.intel.com/content/www/us/en/embedded/products/galileo/galileo-overview.html>, último acceso en setiembre 2015

un framework basado en Python que permite migrar tareas intensivas desde el dispositivo móvil hacia la infraestructura *cloudlet*. El mencionado procedimiento podrá ser realizado con facilidad para el programador, además que el proceso de descubrimiento de *cloudlets* será transparente al usuario final.

1.1. Motivación

La experiencia del usuario al utilizar una aplicación es un aspecto fundamental, que en muchos casos determina su aceptación o rechazo por los usuarios. Especialmente en aplicaciones que requieren fluidez para una interacción en tiempo real, como: realidad aumentada, procesamiento de imágenes, vídeo streaming, etc. Por lo que una eventual mejora de dicha característica en términos de rendimiento y/o reducción del consumo de energía usando la infraestructura de *cloudlets*, generaría una mayor satisfacción del usuario con la aplicación móvil. Adicionalmente, la investigación se vuelve más desafiante considerando la heterogeneidad inherente a la computación en nube para móviles.

1.2. Objetivos

Los objetivos del siguiente trabajo son los siguientes:

- *Estudio de los diferentes mecanismos de offloading*, se realizará un estudio y comparación de las diferentes técnicas y *frameworks* disponibles actualmente que permitan el *offloading* de tareas. Se mostrará un análisis comparativo de las arquitecturas, objetivo principal, tipo de *offloading*, plataformas de despliegue y el campo de aplicación donde fueron probados.
- *Validación de la prueba del concepto*, con el fin de verificar la supremacía de los entornos WLAN sobre los entornos WAN en el offloading de tareas, se realizará un estudio

empírico comparativo entre tales contextos. Estos experimentos se efectuarán usando una aplicación práctica de reconocimiento de rostros, empleando métricas de tiempo de respuesta y consumo de energía.

- *Propuesta de la Arquitectura de Pytos*, se propondrá una arquitectura, que permita la migración de código al borde de la red (*cloudlets*). Pytos proveerá mecanismos de *offloading* transparentes al usuario final, y a los desarrolladores, Pytos aprovisionará un modelo de programación a una granularidad a nivel de métodos.

1.3. Trabajos Relacionados

En el año 1983, Powell y Miller [PM83] introdujeron el aprovechamiento recursos computacionales disponibles en una red, su enfoque fue migrar procesos durante el tiempo de ejecución. Sin embargo, no fue hasta el advenimiento del boom de la computación móvil en el año 2001, el cual llevó este concepto a ser profundamente estudiado nuevamente. Sathyaranayanan *et. al.* [Sat01] modeló el concepto a *cyber-foraging* o también conocido como *offloading*, el aumento dinámico de los recursos computacionales de una computadora móvil inalámbrica explotando la infraestructura de hardware fija anexa a dicha red. Con lo cual, la computación viene a ser más copiosa, acrecentando la experiencia de usuario.

Similarmente a la definición de *cloudlet*, el término *fog computing* o computación en la niebla [BMZA12] traslapa la idea de procesar al borde de la red. No obstante, la principal diferencia entre los mencionados conceptos es que mientras la computación en la niebla potencia los dispositivos móviles usando componentes de red directamente (enrutadores de red, hubs, etc), los componentes *cloudlet* son computadoras tradicionales que pueden ser vistas como servidores de la nube ubicados próximos al usuario.

Hoy en día existe una variedad de frameworks para realizar *offloading*, los cuales entregan aplicaciones móviles intensas computacionalmente de forma parcial o enteramente a centros

de datos en la nube, o a computadores más potentes en la red de área local (analizados a más detalle en el Capítulo 3). Por ejemplo, MAUI [CBC⁺10], un framework de *offloading* basado en la nube, diseñado solo para soportar aplicaciones escritas en el lenguaje de programación .NET de Microsoft, realiza la tarea de migración a nivel de métodos, requiriendo que el desarrollador anote las funciones que pueden ser migradas. El motor de decisión de MAUI decide en tiempo de ejecución la opción más adecuada considerando la optimización de energía.

Asimismo, Cuckoo [KPKB12] es un framework que puede descargar tareas costosas en ambos entornos (definidos a priori): en la nube y en *cloudlets*. Dicho sistema está enfocado a plataformas Android y permite una granularidad de migración a nivel de métodos. Los métodos a migrar son declarados en un archivo de Lenguaje de Definición de Interfaces (AIDL, Android Interfaces Definition Language). Para resolver el tema de comunicación con los componentes nativos de Android, Cuckoo usa el modelo por defecto *stub/proxy* con el fin de separar las actividades (interfaz de usuario) de los servicios. En otro ejemplo, CloneCloud [CIM⁺11] transforma automáticamente (sin algún esfuerzo de programación) aplicaciones móviles de ejecución única, en aplicaciones distribuidas optimizadas, migrando secciones costosas a la nube. Este objetivo es alcanzado configurando una máquina virtual de Java (JVM) en el lado del servidor que ejecuta un clon de la aplicación móvil. En el caso del sistema COSMO, sus esfuerzos se enfocan en reducir el costo monetario que provoca el uso intensivo de los recursos en nube mientras que al mismo tiempo mejora el rendimiento.

El *offloading* basado en *cloudlets* tiene como propósito reducir la alta latencia WAN que presenta la Computación en Nube para Móviles. Satyanarayanan et al. [SBCD09] propusieron el concepto novel de adaptar los modelos de *offloading* vigentes a una arquitectura *cloudlet*, de tal forma que los usuarios móviles se beneficien de tal infraestructura próxima con demoras significativamente menores. Satyanarayanan et al. [SCH⁺14] en una investigación posterior, valida la arquitectura *cloudlet*, utilizando un dispositivo *google glass*⁵ para

⁵<https://www.google.com/glass/start/>, Último acceso en setiembre 2015

aumentar la capacidad cognitiva de resolver problemas en tiempo real mediante *streaming* de vídeo. Igualmente, Fesehaye et al. [FGNW12] estudió el impacto de los *cloudlets* en aplicaciones de nube interactivas. Sus resultados demostraron que los *cloudlet* superan la nube en términos de reducir la demora e incrementar el rendimiento. Soyata et al. [SMF⁺12] propuso la arquitectura MOCHA, en la cual el componente *cloudlet* determina como particionar el cálculo entre si mismo y múltiples servidores de la nube sobre los diferentes enlaces/rutas con el objetivo de mejorar la calidad de servicio (QoS, Quality of Service).

Un ejemplo de framework de *offloading* basados exclusivamente en componentes *cloudlets* es el caso de Scavenger [Kri10]. Scavenger es un sistema que emplea la migración de tareas mediante anotaciones de métodos en el lenguaje Python (de manera similar a Cuckoo y MAUI). Este software usa el enfoque de código móvil para particionar y distribuir tareas. El principal aporte de Scavenger es un planificador de recolección doble, que permite un uso inteligente de los recursos incluso en entornos no preparados.

Muchas aplicaciones pueden ser mejoradas con el *offloading* de tareas: operaciones matemáticas complejas [SCB11], procesamiento de señales en juegos de tiempo real [CBC⁺10] [KPKB12], seguridad de dispositivos móviles a través de análisis de virus [KAH⁺11], aumento de la cognición [SCH⁺14], entre otras tareas intensivas.

En este trabajo después de un análisis de las técnicas de *offloading* actuales, seleccionamos las más adecuadas y las mejoramos para poder brindar un framework basado en la arquitectura *cloudlet* que: 1) sea altamente móvil, 2) recolecte información del entorno móvil de manera correcta para la posterior planificación de tareas (local o remoto) y 3) se conserve recursos del dispositivo móvil como energía y mejore el rendimiento.

1.4. Organización del Documento

Este trabajo está organizado como sigue: En el Capítulo 2 damos una vista general a los conceptos fundamentales requeridos para entender las técnicas de *offloading*, como computación móvil, computación en nube y computación en nube para móviles. Estudiaremos las principales técnicas de *offloading* usadas en MCC en el Capítulo 3. En el Capítulo 4 introducimos el concepto de *cloudlet*, componente necesario para construir la arquitectura propuesta y reducir la demora en redes WAN. En el Capítulo 5 haremos una propuesta de arquitectura llamado Pytos, framework basado en Python para implementar *offloading* hacia *cloudlets* en aplicaciones móviles. Finalmente en los Capítulos 6 y 7 consideraremos el cronograma del trabajo y las conclusiones de la presente investigación.

Capítulo 2

Conceptos

A fin de llevar un claro entendimiento a lo largo del trabajo, es necesario conocer y entender los términos fundamentales en los cuales se basa la presente investigación. Se describen los conceptos básicos de computación móvil y computación en nube.

2.1. Computación Móvil

Según Talukdar [Tal10], la computación móvil puede ser definida como un entorno de computación de movilidad física. En el cual, el usuario de un entorno de computación móvil estará disponible para acceder a sus datos, información, u otro objeto lógico desde cualquier dispositivo en cualquier red mientras se está en movimiento. Un sistema de computación móvil permite al usuario realizar tareas en todas partes como: (1) navegar en Internet, (2) acceder a datos corporativos (negocios e información) o (3) acceder a datos de información personal (registro médico o agenda personal). Con el objetivo de que el entorno de computación móvil sea ubicuo, es necesario que la comunicación sea expandida sobre los dos medios: cableados e inalámbricos.

En tanto, Imielinski y Korth [KLLB13] afirman que “La computación móvil ya no necesita que los usuarios mantengan una posición fija y universalmente conocida en la red y permite casi movilidad sin restricciones”, mientras que Satyanarayana [Sat11] aclara que la visión de la computación móvil es “ Información al alcance de tus dedos en cualquier momento, en cualquier lugar”. Podemos definir el entorno de computación como móvil si este soporta uno o más de las siguientes características[Tal10]:

- *Movilidad del Usuario:* El usuario debe tener libertad para trasladarse desde una ubicación física a otra y usar el mismo servicio. El servicio podría estar en una casa o una red remota. Por ejemplo, un usuario que se desplaza desde Arequipa hacia São Paulo y usa Internet para acceder a una determinada aplicación corporativa, lo hará de la misma forma que el usuario lo usa en casa.
- *Movilidad de Red:* La movilidad de red lidia con dos tipos de casos de uso. En un caso, el usuario se mueve desde una red a otra y usa el mismo servicio sin problemas. Un ejemplo podría ser, cuando un usuario se traslada desde la red WiFi de la Universidad y cambia a una red 3G fuera del campus. En el otro caso, la red por si misma es móvil como en una red Ad hoc Móvil (MANET, Mobile Ad hoc Network). En una red MANET, cada nodo en la red es una combinación de un anfitrión o *host* y un punto de acceso. A medida que los nodos se mueven, los puntos de acceso dentro de la red también se desplazan cambiando dinámicamente la estructura de la tabla de ruta. Tales tipos de redes son usadas en campos de guerra o redes de sensores, donde los puntos de acceso/nodos se mueven constantemente.
- *Movilidad del dispositivo:* Se debe permitir al usuario cambiar de un dispositivo a otro y usar el mismo servicio sin problemas. Un ejemplo podría ser el caso de una agenda personal virtual que pueda ser accedida desde la rúa, de igual forma que desde el hogar.

2.2. Computación en Nube

La computación en nube es la entrega de servicios en vez de un producto. Como el Instituto Nacional de Estándares y Tecnología (NIST, National Institute of Standards and Technology), la describe [MG09]: “un modelo que permite de manera conveniente, acceso de red bajo demanda, ubicuo y conveniente a un conjunto compartido de recursos computacionales configurables (p. ej. Redes, servidores, almacenamiento, aplicaciones y servicios) que pueden ser otorgados rápidamente y liberados con el menor esfuerzo administrativo o interacción con el proveedor del servicio”.

Esta definición incluye arquitecturas en nube, seguridad, y despliegue de estrategias. Según [DWC10], cinco elementos esenciales describen a la computación en nube:

1. **Servicio automático bajo demanda:** Un consumidor con una necesidad instantánea puede disponer de recursos (como CPU, almacenamiento, uso de software, entre otros) de manera automática sin requerir interacción directa con el proveedor de estos recursos.
2. **Amplio acceso de Red:** Los recursos computacionales son entregados por la red de Internet y usados por varias aplicaciones de clientes con plataformas heterogéneas (celulares, objetos de Internet de las Cosas, navegadores web móviles, etc.)
3. **Agrupamiento de Recursos:** Los recursos de un proveedor en nube están agrupados por dos motivos: economía de escala y especialización. El resultado de este modelo es que los recursos de computación físicos son invisibles al consumidor, quien generalmente desconoce la ubicación, formación, funcionamiento, de estos recursos.
4. **Rápida elasticidad:** Para los usuarios, los recursos computacionales son elásticos sin dependencias contractuales, es decir, que un consumidor puede escalar hacia arriba rápidamente si la demanda lo requiere, o escalar hacia abajo rápidamente sin diligencias.

5. **Medición del servicio:** A pesar de que los recursos están agrupados en una misma infraestructura, la computación en la nube permite usar mecanismos para medir el uso de estos recursos por cada consumidor individual.

Adicionalmente a estas 5 características esenciales, la computación en nube se categoriza en 3 principales tipos.

- **Software como Servicio (SaaS, por sus siglas en inglés):** En este modelo los desarrolladores liberan el software sobre un entorno de *hosting* el cual puede ser accedido mediante aplicaciones de usuario como navegadores web o aplicaciones móviles. Ejemplos de SaaS tenemos Google Docs, Google Mail ¹, Google Drive ², Outlook ³, entre otros.
- **Plataforma como Servicio (PaaS, por sus siglas en inglés):** PaaS es una plataforma de desarrollo que soporta todo el ciclo de desarrollo del software, esto permite a los consumidores de la nube, liberar y desplegar sus aplicaciones directamente sobre la capa de aplicación. A diferencia de PaaS, SaaS solo sirve aplicaciones completas, mientras que PaaS ofrece adicionalmente la plataforma de desarrollo. Ejemplo de PaaS son Google App Engine ⁴, Heroku ⁵, Microsoft Azure ⁶.
- **Infraestructura como un Servicio:** Los consumidores de la nube directamente administran la infraestructura del servidor (redes, procesamiento, almacenamiento, gestión de paquetes, nuevos programas, entre otros). La creación de Máquinas Virtuales

¹<https://mail.google.com/>, último acceso en setiembre 2015

²<https://drive.google.com/drive/>, último acceso en setiembre 2015

³<https://www.live.com/>, último acceso en setiembre 2015

⁴<https://cloud.google.com/appengine/docs>, último acceso en setiembre 2015

⁵<https://www.heroku.com/>, último acceso en setiembre 2015

⁶<http://azure.microsoft.com/en-us/>, último acceso en setiembre 2015

(VM, Virtual Machines) son intensamente usadas en este modelo a fin de aislar, integrar diferentes recursos físicos. En este modelo podemos listar a : Amazon EC2 ⁷, Google Compute Engine ⁸, Digital Ocean ⁹ , entre otros.

2.3. Computación en Nube para Móviles

El área de la computación en nube para móviles al ser un campo de investigación relativamente nuevo, no existe una definición ampliamente aceptada y puede ser vista desde muchos ángulos como lo realizó el trabajo de Fernando, Loke y Rahayu [FLR13], tomando hasta 3 definiciones distintas. En este trabajo nos referimos a MCC como un conjunto de técnicas que aprovechan los potentes recursos computacionales en nube para potenciar aplicaciones móviles con el principal objetivo de mejorar la calidad de servicio hacia los usuarios que tienen dispositivos limitados en términos de energía, procesamiento y almacenamiento. En este capítulo detallaremos la visión actual de MCC, del mismo modo explicaremos la heterogeneidad que la envuelve, y los desafíos por solucionar en las técnicas de *offloading*.

2.3.1. Visión

MCC permite a los usuarios móviles acceder a sus aplicaciones, datos, y servicios en nube a través de internet aprovechando la web móvil [DLNW13], consiguiendo no depender totalmente de los recursos locales de los móviles, , por tanto esta nueva ola tecnológica puede ser aplicada en varias áreas como la salud, negocios de TI, educación, entretenimiento y redes sociales.

En el trabajo de Abolfazli, Sanaei y Gani [ASG12] se clasifica los enfoques que puede tomar la mejora en los dispositivos móviles entre hardware y software los cuales se muestran en

⁷<http://aws.amazon.com/ec2/>, último acceso en setiembre 2015

⁸<https://cloud.google.com/compute/>, último acceso en setiembre 2015

⁹<http://www.digitalocean.com/>, último acceso en setiembre 2015

la gráfica de la Figura 2.1. Si bien los avances de hardware son virtuosos, son aún lentos en comparación a la creciente expectativa de los usuarios y desarrolladores de aplicaciones. Otro enfoque alternativo para la mejora de las capacidades computacionales de los dispositivos es usar técnicas de software para la conservación de recursos locales y la reducción de requerimientos de recursos al instalar aplicaciones. En consecuencia, sostenemos que la computación en nube tendrá una participación destacada aventajándose de valiosos recursos remotos con el fin de mejorar la experiencia del usuario.

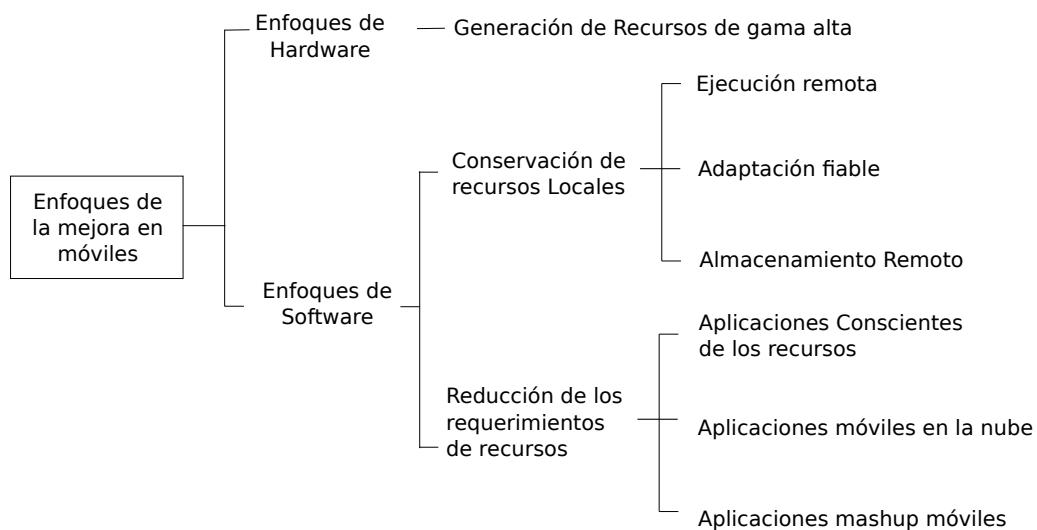


FIGURA 2.1: Taxonomía de los enfoques de mejora de en móviles. Figura adaptada de [ASG12]

2.3.2. Heterogeneidad

La innegable popularidad que han tomado los dispositivos móviles como los smartphones, wearables y objetos de IoT crea un mercado dinámico y exigente, que ha diversificado a los componentes de MCC en diferentes dimensiones de una manera no estándar. Esta variedad de hardware, arquitecturas, infraestructura y tecnologías de dispositivos móviles, nubes, y redes inalámbricas genera un entorno Heterogéneo. En esta sección describimos como la heterogeneidad abarca a todos los componentes de MCC: computación móvil, computación en nube y redes inalámbricas [SAGB14]. Este escenario es representado en la Figura 2.2.

Aunque la heterogeneidad en la MCC ha sido inherente desde los orígenes de la computación en nube y móvil, su complejidad la hace desafiante y única, por lo que necesita un estudio exhaustivo.

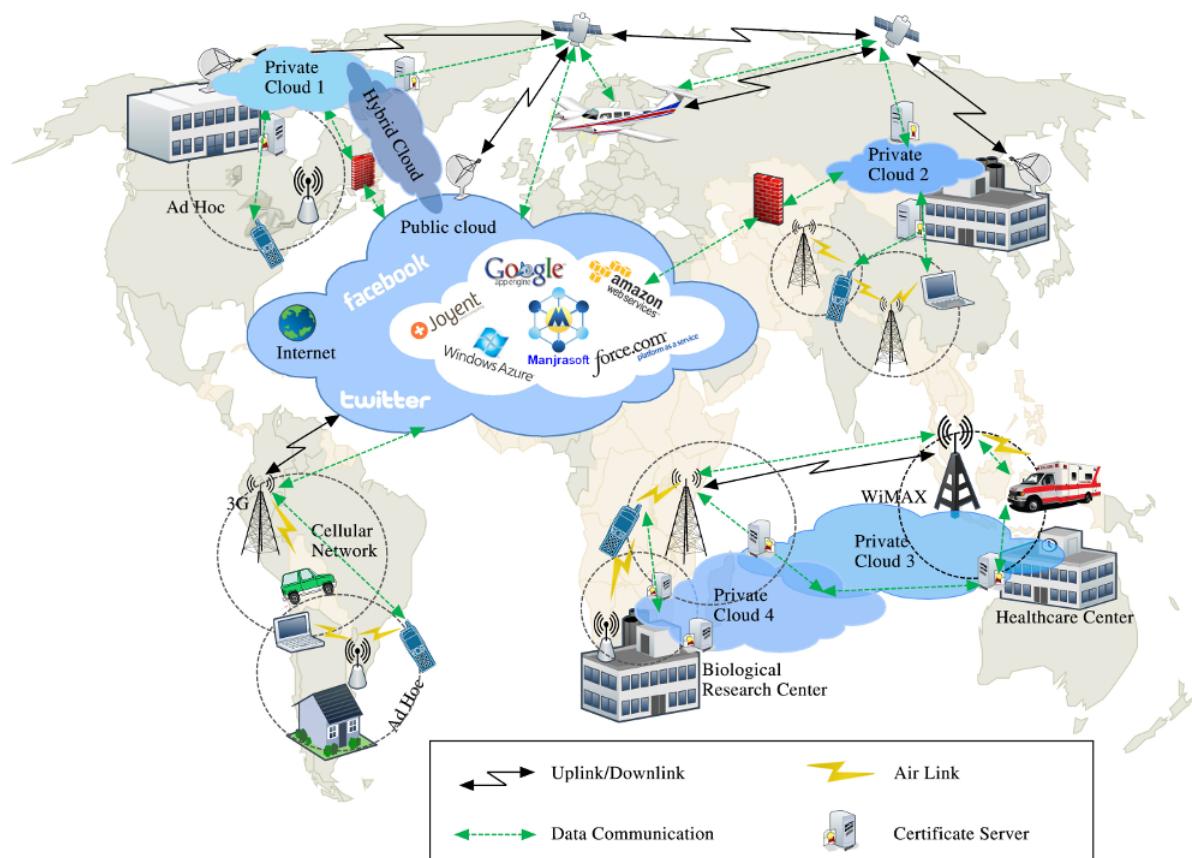


FIGURA 2.2: Una vista general a MCC [SAGB14]

2.3.2.1. Taxonomía de la heterogeneidad en MCC

Para entender claramente la heterogeneidad que no es una característica específica de un dominio, podemos percibir casos en la vida real: los automóviles que están compuestos de partes totalmente diferentes que logran interactuar entre ellos con un solo fin; el caso de las partes del cuerpo que son intrínsecamente heterogéneas logran cooperar para mantener la funcionalidad como un todo. Análogamente, los tres componentes principales de la MCC

(computación en la nube, computación móvil y redes inalámbricas) deben interactuar sin problemas y consistentemente. Esta división de los orígenes de la taxonomía está representada en la imagen de la Figura 2.3.

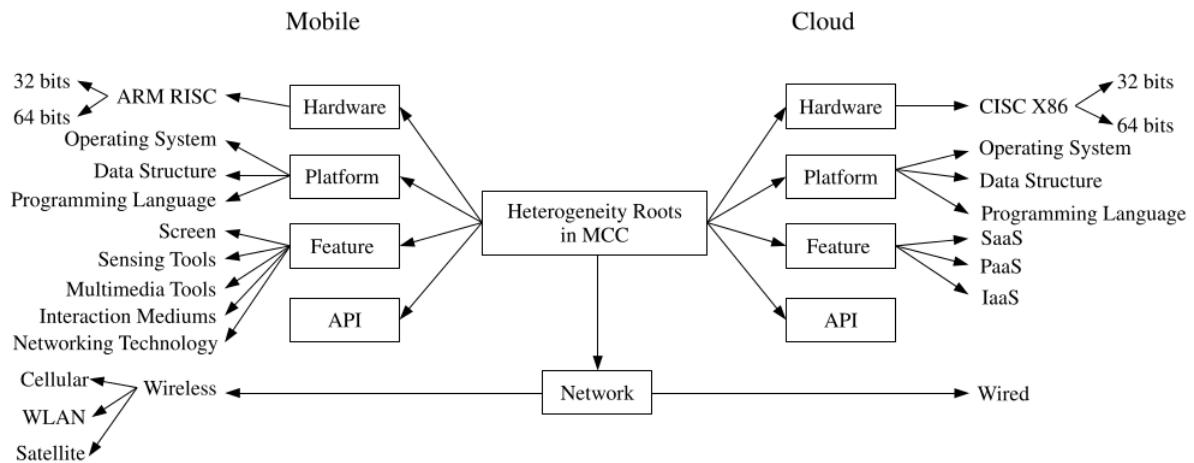


FIGURA 2.3: Taxonomía de los orígenes de la heterogeneidad en MCC [SAGB14]

- 1. Heterogeneidad de hardware:** La fuerte demanda de los usuarios de móviles y la competencia entre las grandes compañías fabricantes de hardware han permitido que diversifiquen sus arquitecturas internas entre dispositivos móviles, servidores en nube, e infraestructuras de redes.

En el entorno de la computación en nube, la variedad de proveedores de servidores otorgan una variedad en infraestructuras y diseño de arquitecturas de Computadoras con un Conjunto de Instrucciones Complejas (CISC, por sus siglas en inglés) con las variaciones de 32 y 64 bits.

Similarmente, la diversidad de arquitecturas en los dispositivos móviles, que generalmente están basadas en Computadoras con Conjunto de Instrucciones Reducidas (RISC, por sus siglas en inglés) de 32 bits; sin embargo, existe una considerable variación en términos de velocidad del procesador, número de nucleos, cantidad de caché del

procesador. Igualmente existe una variedad importante entre los recursos del dispositivo móvil como la memoria interna, fiabilidad de la antena inalámbrica y la capacidad de la batería.

La heterogeneidad en hardware entre los dispositivos móviles y las computación en nube, crea los siguientes problemas:

- La variación de hardware de los servidores en nube conlleva a una diferencia de rendimiento y calidad de servicio, además del casi nulo soporte para la portabilidad de aplicaciones entre los diferentes proveedores, entonces fuerza al usuario a tomar una decisión importante entre la vasta cantidad de opciones disponibles. Los desarrolladores pueden usar la herramienta en línea de *Software Insider* para optar por la mejor propuesta de servicios en nube ¹⁰.
- El conocido principio usado por desarrolladores “codifica una vez, ejecútalo donde sea” se cumple parcialmente en el heterogéneo ambiente móvil, dado que requiere dependencia de herramientas privadas multi-plataforma como Xamarin ¹¹ o Qt para móviles ¹². Las herramientas en mención son un gran aporte en el entorno de los *smartphones* pero tienen una limitante en entornos de dispositivos *wearables* y de Internet de las cosas, ya que en su arquitectura se ejecuta sobre diferentes distribuciones ligeras de Linux.
- Un parámetro fundamental en las técnicas de *offloading* es la estimación precisa de la energía [SAGB14], para que se decida si el enviar partes de la aplicación conservará energía o no [KL10]. Es por tal motivo que se necesita estimar de manera precisa este valor

2. Heterogeneidad de Plataforma:

De modo equivalente a la heterogeneidad de hardware, existe una amplia disponibilidad de sistemas operativos, lenguajes de programación y estructuras de datos en MCC que transforman a las plataformas en entornos

¹⁰ <http://cloud-computing.softwareinsider.com>, último acceso en setiembre 2015

¹¹ <http://xamarin.com/>, último acceso en setiembre 2015

¹² <http://www.qt.io/mobile-app-development/>, último acceso en setiembre 2015

heterogéneos. En el contexto de los sistemas operativos móviles encontramos algunos como Android ¹³ y iOS ¹⁴, de Google y Apple respectivamente, cada uno con múltiples versiones y arquitecturas que soportan diferentes lenguajes de programación y estructuras de datos. Android ofrece el desarrollo sobre la máquina virtual de java basado en bytecodes; mientras que iOS soporta Objective-C, un lenguaje compilado a nivel de código máquina.

En el ámbito de la computación en nube los proveedores más populares tales como App Engine ¹⁵, Microsoft Azure ¹⁶ y Amazon Web Services ¹⁷ soportan una diversa cantidad de lenguajes de programación, sistemas operativos y arquitecturas de almacenamiento, p. ej. Azure Soporta .Net, PHP, Ruby, Python y Java para desarrollar aplicaciones, Google igualmente ofrece Java además de Python, PHP y Go; sin embargo, estas plataformas poseen almacenes de datos no relacionales con diseños estructurales y esquemas de particionado diferentes (Big table de Google [CDG⁺06], Dynamo de Amazon [DHJ⁺07] y DocumentDB de Azure ¹⁸), y por lo tanto estos proveedores hacen cumplir sus restricciones de uso para proveer mayor flexibilidad de su servicio.

Tal heterogeneidad convierte la portabilidad en un proceso tedioso, costoso y de alto riesgo. Transferir aplicaciones entre diferentes nubes demanda un costo extra que incluye descargar la aplicación, realizar la conversión y desplegar el nuevo sistema a la nueva nube. Es peligroso, dado que la transferencia entre bases de datos de entre nubes compromete la privacidad e integridad de los datos cuando existe una diferencia entre sistemas de archivos y técnicas de encriptación [RS10]. Por lo tanto, para el desarrollador escoger la plataforma correcta no es tarea fácil. La variedad de plataformas se representa en la Figura 2.4.

¹³<http://developer.android.com/index.html>, último acceso en setiembre 2015

¹⁴<https://developer.apple.com/devcenter/ios/index.action>, último acceso en setiembre 2015

¹⁵<https://cloud.google.com/appengine/docs>, último acceso en setiembre 2015

¹⁶<https://azure.microsoft.com/en-us/>, último acceso en setiembre 2015

¹⁷<http://aws.amazon.com/>, último acceso en setiembre 2015

¹⁸<http://azure.microsoft.com/en-us/services/documentdb/>, último acceso en setiembre 2015

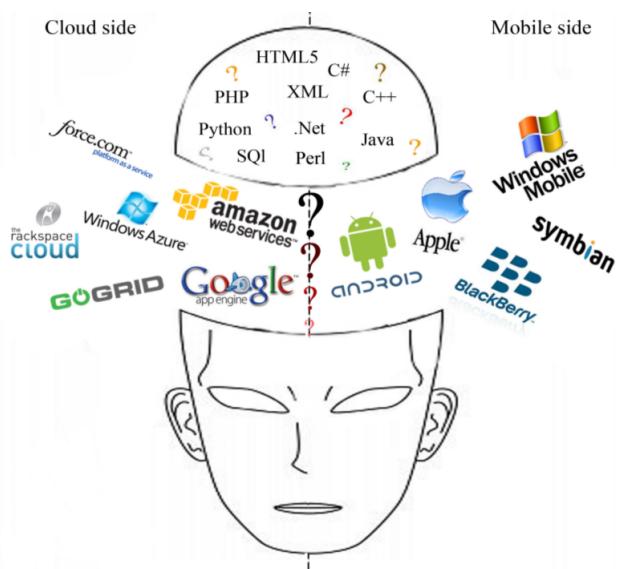


FIGURA 2.4: Heterogeneidad de plataformas en MCC y la difícil tarea para el desarrollador elegir la correcta [SAGB14]

3. Heterogeneidad de Características: Es la variación de características en los *smartphones* tales como los sensores, el área de visualización, multimedia, y tecnologías de red. Una de las características que más varía entre fabricantes es la cámara. P. ej. El celular motorola X, un celular de gama media, posee una cámara de 10PM¹⁹, entretanto el celular Samsung Galaxy Edge 6, un celular de gama alta, dispone de una cámara de 16 MP²⁰. Esta desigualdad puede significar una variación en las técnicas de offloading (las cuales detallaremos en el Capítulo 4) si es que consideramos algún tipo de procesamiento de imagen en la nube, debido a que el envío de datos a la nube sería más lento.

Las variaciones en el área de visualización entre los celulares impiden que los desarrolladores diseñar una interfaz de usuario que se adapte a todos los tamaños de pantalla, disminuyendo la experiencia de usuario. En el entorno Android existe el soporte para la construcción de aplicaciones que se adecúan al contenido de la pantalla en su versión

¹⁹<https://www.motorola.com/us/Moto-X/FLEXR1.html>, último acceso en setiembre 2015

²⁰<http://www.samsung.com/uk/galaxys6/>, último acceso en setiembre 2015

3.2 en adelante ²¹.

La heterogeneidad en nube está dada por los servicios que cada proveedor ofrece en particular. P. ej. *Google App Engine* ofrece el servicio de Big Query ²² que permite a los administradores realizar consultas sobre *Big Data* en tiempo real. Microsoft Azure a su vez, es el único que ofrece un servicio de almacenamiento de backups encriptado ²³

4. Heterogeneidad de APIs: Las Interfaces de Programación de Aplicaciones (API, por sus siglas en inglés) son bibliotecas de código que permiten un acceso rápido a datos específicos o funciones de un proveedor en particular. Los usuarios de *smartphones* tienen una alta expectativa respecto a la experiencia de usuario, es por tal motivo que las mas grandes compañías de sistemas operativos móviles (Android, iOS, Blackberry ²⁴ y Firefox OS ²⁵) otorgan una amplia gama de APIs a los desarrolladores. Esta es una ventaja para desarrollar aplicaciones enriquecedoras en experiencia de usuario, de una forma ágil y fácil sin necesidad de acceder al kernel. Sin embargo, las APIs son dependientes de los proveedores y la portabilidad nuevamente se convierte en una tarea complicada, dadas las diferencias de las APIs.

De manera similar, en la computación en nube, la casi totalidad de proveedores diseñan APIs que son usados exclusivamente por sus clientes p. ej. Google App engine ofrece la API de documentos e Indices ²⁶ para realizar búsquedas de manera rápida y eficiente sobre espacios geográficos y texto principalmente. Otro ejemplo que resaltamos es en el envío de notificaciones push, Google, Azure y Parse ²⁷ ofrecen sus APIs basadas

²¹ <http://developer.android.com/about/versions/android-3.2.html>, último acceso en setiembre 2015

²²<https://cloud.google.com/bigquery/>, último acceso en setiembre 2015

²³<http://azure.microsoft.com/en-us/services/backup/>, último acceso en setiembre 2015

²⁴<http://us.blackberry.com/>, último acceso en setiembre 2015

²⁵<https://www.mozilla.org/en-US/firefox/os/2.0/>, último acceso en setiembre 2015

²⁶<https://cloud.google.com/appengine/docs/java/search/>, último acceso en setiembre 2015

²⁷<https://www.parse.com/>, último acceso en setiembre 2015

en *Google Cloud Messaging*²⁸, *Notification Hubs*²⁹ y *Parse Push Notifications*³⁰ respectivamente. Esta heterogeneidad además que impide la portabilidad, demanda un tiempo considerable al desarrollador poder aprender cada una de las APIs en nube.

5. Heterogeneidad de Redes: A diferencia de las computadoras de escritorio, la mayoría de dispositivos móviles solamente usan redes inalámbricas para conectarse a Internet, las cuales son comparativamente más intermitentes y menos fiables. La variedad infraestructuras de redes inalámbricas disponibles como Wi-fi, 3G y 4G dificultan el proceso de selección de la red más adecuada en el momento que el cliente móvil está en movimiento.

²⁸<https://developer.android.com/google/gcm/index.html>, último acceso en setiembre 2015

²⁹<http://azure.microsoft.com/en-us/documentation/services/notification-hubs/>, último acceso en agosto 2015

³⁰https://parse.com/docs/push_guide, último acceso en julio 2015

Capítulo 3

Offloading de Cómputo

En los últimos años, los usuarios de dispositivos móviles han sido testigos de un crecimiento considerable de aplicaciones que son útiles en diferentes áreas: salud, entretenimiento, educación, entre otras. Dichas aplicaciones están disponibles (en algunos casos de manera gratuita) en distintas tiendas virtuales como (*Google Play Store* ¹, *App Store* ² y/o *Amazon App Store* ³, entre otras. El crecimiento del mercado de programas móviles según *Transparency Market Research* [tra14] (empresa que realiza investigación de mercado electrónico) pronostica que el mercado móvil crecerá rápidamente en los próximos cinco años, aumentando su valor en aproximadamente US\$ 16.97 billones en el año 2014 a una valuación total de US\$ 54.89 billones en el año 2020.

El uso intensivo y en conjunto de tales aplicaciones demandan una cantidad importante de procesamiento y de energía, los cuales no son abastecidos, esto por las limitaciones en los móviles [VRC13] : (i) baja capacidad de procesamiento, (ii) memoria limitada, (iii) conexiones de red inestables, y (iv) duración de la batería limitada; debido a su naturaleza portátil.

¹<https://play.google.com/store> , último acceso en junio 2015

²<https://itunes.apple.com/us/genre/ios/id36>, último acceso en junio 2015

³<http://www.amazon.com/mobile-apps/b?node=2350149011>, último acceso en junio 2015

A pesar del gran avance tecnológico en hardware resaltado por la ley de Moore, el recurso más crítico en los dispositivos móviles es la densidad de la energía de las baterías, el cual ha tenido una tendencia de mejora considerablemente baja respecto a los demás componentes en la computación móvil [PS05]. Tales limitaciones reducen la calidad del servicio que los usuarios móviles esperan tener.

La Computación en Nube para móviles (*Mobile Cloud Computing*, MCC) tiene entre sus principales objetivos solucionar tales inconvenientes proponiendo técnicas y *frameworks* de *offloading* al integrar los recursos en nube al entorno móvil de una forma elástica y bajo demanda, no solamente para los dispositivos de gama baja sino para cualquier móvil que lo requiera [KOMK14].

En ese escenario, a pesar de que es un área emergente de investigación, la industria móvil ya se ha beneficiado, por ejemplo:

- El caso de la aplicación *DeepFace*: El algoritmo de reconocimiento de rostros de *Facebook* que alcanza una precisión del 97.35 % [TYRW14];
- La aplicación *Google Now*⁴ o *Siri*⁵: Son aplicaciones móviles para Android⁶ y iOS⁷ respectivamente, que utilizan algoritmos de reconocimiento de voz bajo una arquitectura basada en servicios (*Service Oriented Arqchitecture*, SOA).
- *Dropbox*⁸, el cual permite extender la memoria secundaria del dispositivo en la nube otorgando seguridad de la información en caso de pérdida o robo del dispositivo móvil que lo utiliza, así como también sincronización con los dispositivos enlazados, entre otros beneficios.

⁴<http://www.google.com/landing/now/>, último acceso en febrero 2015

⁵<http://www.apple.com/ios/siri/?cid=oas-us>, último acceso en febrero 2015

⁶<http://developer.android.com/index.html>, último acceso en febrero 2015

⁷<https://www.apple.com/ios/>, último acceso en febrero 2015

⁸<https://www.dropbox.com/>, último acceso en enero 2015

La técnica de *Offloading* en el campo de MCC es utilizada para aumentar virtualmente las capacidades del sistema móvil migrando partes de la aplicación a servidores en nube más potentes, de tal forma, que mejore el tiempo de respuesta, el ahorro de energía y en muchos casos la precisión de los algoritmos. Dicha técnica se encuentra implementada en diferentes *frameworks*: MAUI [CBC⁺10], CloneCloud [CIM⁺11], Cuckoo [KPKB12], COMET [GJM⁺12], COFA [SCB11].

La principal diferencia de estos *frameworks* con la arquitectura cliente-servidor es que el cálculo computacional puede o no ser enviado a ejecución remota. Esta decisión es provista por un algoritmo que considera parámetros variables: velocidad de ancho de banda, consumo de energía de la interfaz de red, tamaño de datos a enviar, entre otros. También existe una diferencia clave con el modelo *grid computing* ya que las técnicas de *offloading* migran los procesos a un entorno de computación diferente al del usuario, en tanto que el modelo de *grid* lo hace migrando a otro dentro del mismo entorno computacional.

Se debe también tener en cuenta la naturaleza heterogénea, la cual es inherente a MCC presentando nuevos retos y desafíos en este nuevo campo de estudio. Debido al crecimiento competitivo de las compañías proveedoras, se generó heterogeneidad entre dispositivos móviles, plataformas de computación en nube y redes inalámbricas, dificultando principalmente la interoperabilidad y portabilidad [SAGB14].

3.1. Offloading

El término *offloading* o *cyber-foraging* fue introducido por Satyanarayanan [Sat01] “*Cyber-foraging es una forma efectiva de lidiar con este problema. La idea es aumentar dinámicamente los recursos computacionales de una computadora con conexión inalámbrica aprovechando la infraestructura de hardware cableada. A medida que el cálculo es más barato y más abundante, es sensato “gastar” estos recursos para mejorar la experiencia de usuario* ”. Inicialmente, *Cyber-foraging* propone el aprovechamiento de computadoras estáticas en estado

inactivo de una red local para aumentar las capacidades de los móviles inalámbricos. Hoy en día, la mayor estabilidad de las redes inalámbricas permiten que estos conceptos se hayan expandido de igual forma al uso de la computación en la nube con el fin de aumentar las capacidades de los dispositivos, por tal motivo usamos los términos *cyber-foraging* y *offloading* indistintamente en el presente texto.

El *offloading* de aplicativos tiene como objetivo incrementar las capacidades de los sistemas móviles (extender la duración de la batería y mejorar el rendimiento de las aplicaciones [KL10]) integrando la Computación en nube en los móviles. Sin embargo, en la práctica no es tan sencillo ya que debido a la heterogeneidad en MCC existen muchos factores que influyen sobre si es más adecuado el uso de los servicios en nube sobre la ejecución local. Tales factores dependen del ancho de banda de las red, la interfaz de red a usar (Wifi, 3G and 4G), las cantidades de datos a transferir a través de la red, velocidad del servidor, memoria disponible y carga en el servidor. En este apartado damos un panorama general acerca de la viabilidad del *offloading*, considerando los beneficios principales: mejora de velocidad y ahorro de energía; además, son descritos algunos factores externos e internos que influyen en la toma de decisiones al momento de hacer *offloading*.

3.1.1. Viabilidad de *offloading*

Dado que el proceso de *offloading* transfiere parte del cómputo a un recurso computacional externo. Esta transferencia depende de muchos factores no estándares inherentes a MCC (ver Sección 3.1.2), por lo tanto se requiere saber si aplicar la técnica de *offloading* es necesaria y se tiene que analizar qué cálculo se migrará. En esta sección describimos dos factores de decisión para aplicar *offloading* y sus bases que la fundamentan.

3.1.1.1. Mejora del rendimiento

A medida que las aplicaciones consumen más recursos computacionales, el *offloading* se presenta como una opción atractiva [Bal06]. Por ejemplo, propongamos el caso en el que un módulo crítico de telemedicina [Ryu12] necesita realizar pre-procesamiento de señales en tiempo real; si el procesador de dicho módulo es demasiado lento, el cálculo pesado puede ser migrado a servidores más potentes para su procesamiento.

El siguiente proceso es basado en la investigación de *Kumar et al.* [KLLB13], el cual se ignoran los tiempos de conexión a la red y se asume que el tamaño del programa es despreciable, o puede ser descargado de otro sitio mediante una red de alta velocidad.

Se define a S_w como la velocidad del sistema móvil y w como la cantidad de cálculo a realizar de una parte P que *puede* ser migrada a la nube (no incluye funciones nativas, funciones que usen sensores, o funciones que administren dispositivos de entrada y/o salida). Entonces, el tiempo de ejecución de la parte P es :

$$\frac{w}{S_m} \quad (3.1)$$

Si la parte P es llevada a un servidor, enviar los datos de entrada d_i toma $\frac{d_i}{B}$ segundos a un ancho de banda B . En esta sección el proceso de *offloading* puede mejorar el rendimiento de la aplicación, incluyendo al cálculo y transmisión, puede ser realizado más rápido en el servidor. Ahora supongamos que S_s sea la velocidad del servidor. Tiempo necesario para *offload* al servidor y ejecutar la parte P es:

$$\frac{d_i}{B} + \frac{w}{S_s} \quad (3.2)$$

Entonces el proceso de *offloading* mejora el rendimiento cuando la ecuación 3.1 es mayor que la ecuación 3.2

$$\frac{w}{S_m} > \frac{d_i}{B} + \frac{w}{S_s} \Rightarrow w \left(\frac{1}{S_m} - \frac{1}{S_s} \right) > \frac{d_i}{B}. \quad (3.3)$$

Esta desigualdad sostiene que:

- Para un valor alto de w : El programa requiere computación pesada
- Para un valor alto de S_s : El servidor es más rápido.
- Para un valor pequeño de d_i : Una cantidad pequeña de datos es intercambiada.
- Para un valor alto de B : El ancho de banda es alto.

La desigualdad muestra efectos limitados de la velocidad del servidor. Si se cumple que $\frac{w}{S_m} < \frac{d_i}{B}$, incluso si la velocidad del servidor es infinitamente rápida (p. ej. $S_s \rightarrow \infty$), el proceso de *offloading* no puede mejorar el rendimiento. Por lo que, solo tareas que requieren un cálculo pesado (un valor alto en w) con un intercambio ligero de datos (valor pequeño de d_i) deben ser consideradas.

3.1.1.2. Conservación de la energía

La energía es la limitante de los sistemas móviles. Actualmente, los celulares ya no son usados solamente realizar llamadas; sino que también son usados para navegar por internet, realizar video-llamadas, reproducir videos, procesamiento de *streaming* de música, ejecución de juegos, entre otros. Como resultado de la ejecución de todas esas aplicaciones el consumo de energía aumenta y por consecuencia se acorta la duración de la batería. La técnica de *Offloading* tiene como objetivo conservar este recurso, enviando ejecuciones de tareas que demanden gran cantidad de procesamiento de la aplicación a servidores remotos.

El siguiente análisis propuesto por *Kumar et al.* [KLLB13] explica cuando las condiciones de las técnicas de *offloading* conservan energía. Se define p_m como la alimentación energética en

el sistema móvil. Entonces, la energía del dispositivo móvil es para realizar una tarea puede ser obtenida modificando la ecuación 3.1:

$$p_m \times \frac{w}{s_m} \quad (3.4)$$

Sea p_c la energía necesaria para enviar los datos desde un sistema móvil sobre la red. Por lo tanto, luego de enviar los datos, el sistema necesita obtener la interfaz de red, mientras se espera los resultados. En este tiempo muerto, la energía consumida es p_i . Incorporando estos parámetros en la Ecuación 3.2, se tiene:

$$p_c \times \frac{d_i}{B} + p_i \times \frac{w}{S_s} \quad (3.5)$$

Por lo tanto se puede inferir que el proceso de *offloading* conserva energía cuando la Ecuación 3.4 es mayor que la Ecuación 3.5.

$$p_m \times \frac{w}{s_m} > p_c \times \frac{d_i}{B} + p_i \times \frac{w}{S_s} \quad (3.6)$$

$$\Rightarrow w \times \left(\frac{p_m}{s_m} - \frac{p_i}{S_s} \right) > p_c \times \frac{d_i}{B} \quad (3.7)$$

Se observa que las ecuaciones 3.3 y 3.7 son muy similares. Para que el proceso de *offloading* conserve el energía, un cálculo pesado (un valor alto de w) y una ligera cantidad de datos (valor pequeño d_i) deben ser considerados.

3.1.2. Factores de decisión en *offloading*

En MCC, la heterogeneidad es consecuencia de la disputa de las grandes compañías por el mercado móvil. Esta competencia genera una variedad de servicios y/o productos singulares que en su mayoría no siguen un estándar, y por ende muchas veces se relega la interoperabilidad y/o portabilidad [SAGB14].

Sanae et al. [SAGB14] identificó y clasificó la heterogeneidad presente en MCC, el cual lo representó en la Figura 2.3. Esta taxonomía presenta tres componentes (dispositivos móviles, la nube y las redes inalámbrica) que poseen raíces de heterogeneidad a nivel de : hardware, plataforma, características, interfaces de programación de aplicaciones (API, *Application Programming Interface*) y Red. Esta diversidad dificulta la toma de decisiones de los diferentes *frameworks* de *offloading*. Por ejemplo si consideramos dos celulares con arquitecturas distintas es probable uno de ellos tenga un mejor tiempo de respuesta y la decisión más factible sea no aplicar la técnica de *offloading*. Por lo tanto, la aplicación es probable que se ejecute localmente. Similarmente, ocurre lo mismo en las entorno de las redes inalámbricas y en las características de la plataforma, lo cual se profundizará en esta sección.

Considerando las raíces de la heterogeneidad en MCC, clasificamos los factores de decisión en *offloading* en tres grupos: características del dispositivo móvil, características de la aplicación y características de red. Las cuales las presentamos a continuación:

3.1.2.1. Características del dispositivo móvil

La mayoría de dispositivos móviles están basados en computadoras con un conjunto de instrucciones reducidas (RISC, Reduced Instruction Set Computer) de 32 bits; sin embargo, existe una considerable variación en términos de velocidad del procesador, número de núcleos, cantidad de caché del procesador. Igualmente, existe una variedad importante entre los recursos del dispositivo móvil como la memoria memoria interna, fiabilidad de la antena inalámbrica y la capacidad de la batería. Tales parámetros necesitan ser contemplados al

momento de aplicar *offloading*. En un futuro cercano con la entrada de la serie de procesadores ARM de 64 bits, entre los ya tradicionales Cortex, se generará una heterogeneidad aún mas desafiante⁹.

Uno de los principales objetivos de proceso de *offloading* es el ahorro de energía. Es por eso que considerar la precisión del gasto energético individual de cada componente en tiempo real es un reto abierto.

3.1.2.2. Características de la aplicación

Una métrica efectiva para decidir el proceso de *offloading* es sobre las tareas que consumen alta carga computacional. No obstante, existen algunas partes que no son convenientes para transferir. Porciones de código que ejecutan servicios locales como las interfaces de usuario o códigos que interactúan con componentes de entrada y/o salida como los sensores [CBC⁺¹⁰], métodos nativos del lenguaje de programación ya que pueden tener diferentes implementaciones en diferentes sistemas operativos [GNM⁺⁰⁴] [GJM⁺¹²] y tareas que necesitan recursos locales para ejecutarse [OH98].

3.1.2.3. Características de red

Debido al desplazamiento continuo del usuario, el dispositivo móvil necesita una red inalámbrica para iniciar la transferencia de cómputo a los recursos computacionales en nube. Las redes inalámbricas del tipo Wi-Fi, 3G, 4G y WiMax son las más adecuadas, aunque tenemos que considerar que cada una de estas interfaces consume una cantidad diferente de energía. Por ejemplo, como se observa en la Figura 3.1, las redes 3G consumen más energía que las redes Wi-Fi con una reducida tasa de transferencia de bits, sin embargo ocurre lo contrario con una tasa alta de transferencia [MN10].

⁹ <http://arm.com/about/newsroom/arm-discloses-technical-details-of-the-next-version-of-the-arm-architecture.php>, último acceso en junio 2015

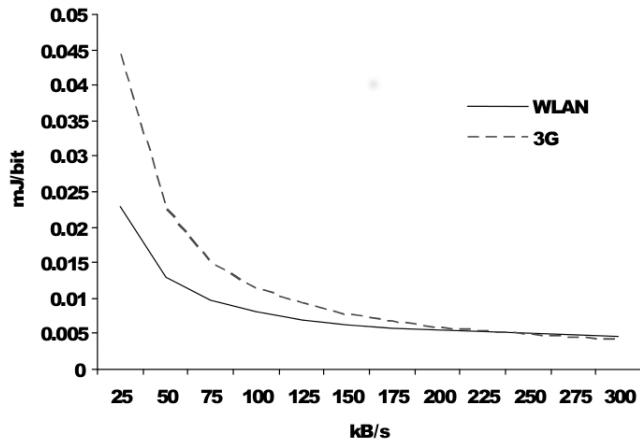


FIGURA 3.1: Consumo de energía por Bit de transferencia. WLAN vs 3G. Experimento realizado en celular Nokia N900 [MN10]

Similarmente, una demora cuantiosa debido al ancho de banda de una red inalámbrica puede ser causante de un consumo de energía más alto, ya que la aplicación se encuentra en estado inactivo a la espera del resultado.

3.2. Algoritmo de offloading

En esta sección se presenta un algoritmo básico para la aplicación de *offloading* de aplicaciones, el cual es representado en la Figura 3.2. Se debe resaltar que el algoritmo mostrado a continuación no es necesariamente aplicado en todos sistemas de *offloading*. El flujo comienza con la ejecución de la aplicación seguido por la revisión de los permisos del usuario para realizar *offloading*. En caso posea estos privilegios, la aplicación revisa si el servidor (sea de: red local o en la nube) está en conexión con el dispositivo móvil, revisando su conectividad. Posteriormente, se toma la decisión si realmente es viable y necesario el *offloading*, cual sea el objetivo principal: mejorar el rendimiento o ahorrar energía o ambos (descrito en la Sección 3.1.1). Si la cuestión anterior es positiva, todo el entorno es favorable para su ejecución remota por lo que se migra el cálculo a servidores en la nube. En otro caso, la aplicación se ejecuta de manera local.

El proceso *offloading*, en especial la toma de decisión de *offloading* es muy difícil de realizar con una adecuada precisión. Esto, debido a factores externos e internos del dispositivo móvil que fueron revisados en la Sección 3.1.2.

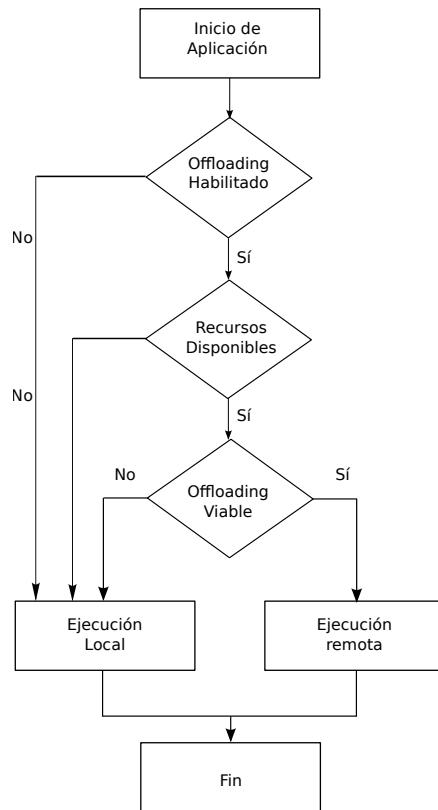


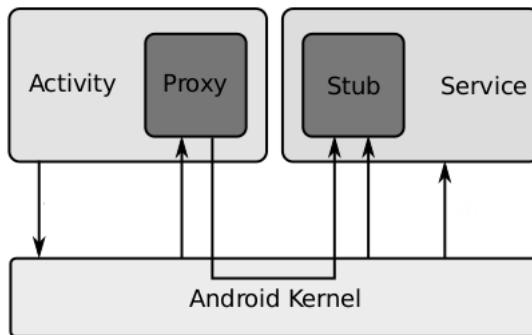
FIGURA 3.2: Proceso general de *offloading*. Figura Adaptada de [KOMK14]

3.3. Criterios para la clasificación de Offloading

En las secciones anteriores fueron descritos los objetivos de la técnica de *offloading* así como los factores que influyen en la toma de decisión para que ésta se realice. En este sección se describen los criterios de clasificación: tipo de *offloading*, particionado de la aplicación y plataforma de despliegue. Posteriormente, se muestra una clasificación de los más importantes y actuales *frameworks* de *offloading* disponibles en MCC, la Tabla 3.1 muestra dicha clasificación siguiendo los criterios anteriormente comentados.

CUADRO 3.1: Algunas soluciones para el offloading de aplicaciones móviles

Año	Framework	Arquitectura	Objetivo Principal	Tipo de Offloading	Plataforma Cliente/Servidor	Campo de Aplicación
2002	Spectra [FPS02]	Orientado a Servicios	Ahorro de Energía y reducción de tiempo de Ejecución	Estático	Entorno Simulado	Procesamiento de Audio y Documentos
2003	Chroma [BSP003]	Orientado a Servicios	Reducción de tiempo de Ejecución	Estático	Entorno Simulado	Procesamiento de Imágenes
2010	MAUI [CBC+10]	Virtualización	Ahorro de Energía	Dinámico	Windows Mobile / .Net	Procesamiento de Imágenes / Juegos
2011	CloneCloud [CIM+11]	Virtualización	Ahorro de Energía y reducción de tiempo de Ejecución	Dinámico	Android/Dalvik	Analisis de Virus / Búsqueda de imágenes
2011	COFA [SCB11]	Orientado a Servicios	Ahorro de Energía y reducción de tiempo de Ejecución	Estático	Android/Dalvik	Cálculos matemáticos
2012	COMET [GJM+12]	Virtualización	Reducción de tiempo de Ejecución	Dinámico	CyanogenMod (Android)/Dalvik	Aplicaciones Web
2012	Cuckoo [KPKB12]	Orientado a Servicios	Ahorro de Energía y reducción de tiempo de Ejecución	Estático	Android/JavaVM	Juegos / Procesamiento de Imágenes

FIGURA 3.3: Arquitectura del *framework* Cuckoo [KPKB12]

3.3.1. Particionado de la Aplicación

La presente investigación se basa principalmente en dos tipos de arquitecturas al momento de realizar el particionado: “arquitecturas basadas en servicios” y “virtualización” [FLR13]. El primero es realizado entre el dispositivo móvil y el servidor vía protocolos como por ejemplo: llamadas a procedimiento remoto (RPC, Remote Procedure Call), invocación de métodos remotos(RMI, Remote Method Invocation) y sockets. Dos ejemplos que utilizan estos protocolos son los *frameworks* Chroma [BSPO03] y Spectra [FPS02] los cuales poseen pre-instalado en sus servidores servicios RPC para el proceso de *offloading*. Otro ejemplo es el *framework* *Cuckoo* [KPKB12], *framework* para *offloading* basado en Android OS¹⁰, como se observa en su arquitectura de la Figura 3.3, utiliza interfaces para la comunicación con componentes nativos siguiendo el modelo *stub/proxy* en el lenguaje de programación Java. El *framework* *Cuckoo* utiliza el modelo por defecto de Android *activity/service* que separa las actividades (métodos de interacción con el usuario) de los servicios (métodos de cómputo pesado).

En el segundo tipo arquitectural, la virtualización, consiste en transferir la imagen de la memoria de una máquina virtual (VM, *Virtual Machine*) desde un servidor origen a un servidor destino sin detener su ejecución [CFH⁺05]. Durante la migración las páginas de memoria de la MV son copiadas previamente sin generar interrupción alguna en el sistema

¹⁰<http://developer.android.com/index.html>, último acceso en setiembre 2015

operativo, de tal modo que da la sensación de una migración natural. Una ventaja es que el código de la aplicación no necesita ser rescrito y provee una alta y segura ejecución, debido a que los límites de la MV lo aíslan. MAUI [CBC⁺10] es *framework* el cual usa una mezcla de migración, entre MV y partición de código. Las aplicaciones que usan MAUI pueden transferir calculos a servidores tanto locales como remotos que estén sobre la infraestructura en la plataforma .NET. Otro ejemplo es el *framework* CloneCloud [CIM⁺11] el cual usa migración de MV para transferir parte de la carga de la aplicación a servidores.

3.3.2. Tipos de Offloading

A fin de adaptar técnicas de *offloading* en una aplicación, es concerniente al desarrollador conocer las posibles maneras para implementar, así como sus posibles ventajas y desventajas. Tal adaptación puede ocurrir de dos maneras: de forma estática o dinámica [MM09]. En el primer enfoque el desarrollador define que secciones de código se ejecutarán remotamente. Esto ocurre en la fase de diseño o en la instalación y por lo tanto cuando se inicia el ejecutable, el *framework* conoce de antemano que partes del programa pueden ser enviados a ejecución remota. La principal ventaja es que no existe una prominente sobrecarga en la aplicación, por tal motivo en este enfoque es necesario que los factores que influyen en el algoritmo de decisión (que describimos en la sección 3.1.2) sean predichos lo más exacto posible. Parámetros como poder de cálculo o ancho de banda en el lado del servidor son valores muy estables debido a que los grandes proveedores garantizan un nivel mínimo de rendimiento. Algunos algoritmos de predicción de estos factores son: predicción basada en los registros históricos [GKW04] y predicción probabilística [RP03]. Por otro lado, en el enfoque dinámico la ubicación de la parte a transferir no esta predeterminado por el desarrollador y tiene que ser calculado generalmente en tiempo de ejecución, lo cual genera sobrecarga de cómputo. En este enfoque de igual manera existen mecanismos de predicción para la toma de decisiones. Por ejemplo algunas propuestas construyen modelos de Markov como un modelo de movilidad (basados en el histórico de conexiones a puntos de acceso Wi-Fi), entonces es

entrenado con patrones de movilidad del usuario [LS13]. En cambio, Wolski [WGKN08] en su investigación propone que el ancho de banda sea predicha usando un esquema bayesiano.

3.3.3. Plataforma de despliegue

Las soluciones para el *offloading* de aplicaciones requieren desplegarse en plataformas móviles y en plataformas en nube. Para este motivo existe una pluralidad de plataformas heterogéneas en la computación en nube y en la computación móvil . Esta heterogeneidad (como se observó en la Figura 2.3) generalmente limita a los sistemas de *offloading* a ejecutarse en todas las plataformas. Tal es el caso de CloneCloud [CIM⁺11] que explota el sistema operativo de código abierto AndroidOS para integrar su solución. Este sistema despliega una versión modificada de la máquina virtual de Dalvik (MV del sistema Operativo Android) [Ehr10] en un servidor en la Nube con el fin de que acepte la transferencia de la aplicación.

Otro ejemplo es el caso de MAUI [CBC⁺10], el cual implementa una arquitectura basada en *Windows Mobile*¹¹ para su implementación en el móvil, entre tanto que en el servidor se despliega en un arquitectura .NET. Cuckoo [KPKB12] por el lado del cliente está basado en Android, mientras que el servidor se ejecuta sobre la MV de Java. Este hecho hace posible que el servidor de Cuckoo sea genérico y puede desplegarse sin mayor esfuerzo en cualquier tipo de servidor. En el caso de COMET [GJM⁺12] en el lado del cliente se usa CyanogenMod¹², una versión libre de Android mantenida por la Comunidad CyanogenMod.

3.3.4. Campo de Aplicación

Múltiples dominios pueden ser beneficiadas debido a la aplicación de *offloading*. Para tal prueba del concepto, los investigadores han implementado aplicaciones sobre diferentes áreas. En esta sección presentamos ejemplos que demuestran el beneficio del empleo de *offloading*:

¹¹<https://www.windowsphone.com/en-us>, último acceso en enero 2015

¹²<http://www.cyanogenmod.org/>, último acceso en febrero 2015

1. Cálculos Matemáticos: Funciones matemáticas complejas, tal como la multiplicación de matrices de gran tamaño, o la secuencia de *fibonacci* de un número considerable, esta última fue implementada por *Shivarudrappa et al.* en su trabajo para aliviar carga computacional llamado COFA [SCB11]
2. Procesamiento de Señales: Las tareas de procesamiento de señales como voz o imágenes demandan una alta carga computacional, y por ende un consumo energético elevado. Por ejemplo, una aplicación de reconocimiento de patrones en imágenes como lo aplicaron los autores de MAUI [CBC⁺10] en su caso de estudio, o en el caso de la aplicación *eyeDentify*, la cual es implementa usando el framework *Cuckoo* [KPKB12] para el reconocimiento de objetos en imágenes.
3. Juegos: Las aplicaciones de juegos usualmente requieren un poder de cómputo pesado para mantener un renderizado suave de cada *frame*, de tal manera que la interacción con el usuario sea rápida. En el trabajo propuesto por *Kemp et al.* [KPKB12], se presenta la aplicación *PhotoShoot*, un juego multi-jugador de la categoría de disparos en primera persona (FPS, First Person Shooter) que sin la aplicación de *offloading*, a más lento el procesador del dispositivo móvil, más tiempo toma el disparo en ser analizado, lo cual da al usuario con el móvil limitado una seria desventaja. Por ende, aplicar *offloading* de cierto modo establece un juego más justo.
4. Análisis de Virus: Estimando la creciente amenaza de virus y *malwares*, las aplicaciones de la categoría antivirus están siendo una parte fundamental de los sistemas móviles. Sin embargo, el uso de dichos aplicativos demandan una gran cantidad de cálculo que consume una cantidad considerable de energía. El proceso de adaptar técnicas de *offloading* en este dominio , podría aliviar estas limitaciones al realizar un escaneo en la nube de una imagen del celular para ahorrar energía (asumiendo que el dispositivo móvil y su clon en la nube, están sincronizados). Un ejemplo en esta área es el de *CloneCloud* [CIM⁺11], el cual es usado para el caso de estudio de análisis de virus. [KAH⁺11].

Similarmente, existe una variedad de aplicaciones que pueden tomar ventaja del uso de MCC. Sin embargo, aún existen aún retos abiertos consecuencia de su heterogeneidad presente.

Capítulo 4

Cloudlets

Con la ayuda de la nube, los dispositivos móviles pueden migrar las partes computacionalmente intensivas de sus aplicaciones. Los ‘inacabables’ recursos de la nube pueden minimizar los costos de tiempo y energía de las aplicaciones móviles. Sin embargo, como se describió en la sección 4, uno de los aspectos que interfieren realizar *offloading* a la nube es la alta latencia entre la conexión entre el móvil y la nube. Adicionando un cloudlet [SBCD09] (una computadora local que provee de 100 a 1000 veces más poder computacional que un dispositivo móvil con mínima latencia), crea las posibilidades para ejecutar aplicaciones sensibles a la latencia y que demanden gran poder computacional de un dispositivo móvil. La noción de *cloudlet* fue introducida como solución a los obstáculos técnicos descritos anteriormente. La idea principal es proveer recursos abundantes que son necesitados por los dispositivos móviles no desde una *nube* distante, sino desde un *cloudlet* cercano. Satyanarayanan *et al.* resumió las diferencias entre *cloudlet* y *cloud*, las cuales están listadas en la tabla 4.1. Se debe notar que el estado flexible sugiere que las copias de cache, datos o código están en el dispositivo móvil o *cloud*. En tanto, el estado rígido se refiere a que existe solo una única copia de datos o código. Debido a que el *cloudlet* tiene estado flexible, la pérdida de un *cloudlet* no es algo grave.

CUADRO 4.1: Diferencias más notorias entre los entornos *cloudlet* y *cloud* [SBCD09]

	Cloudlet	Cloud
Estado	Solo estado flexible	Estado flexible y rígido
Administración	Autogestionado, poca o sin atención profesional	Administrado profesionalmente 24/7
Entorno	Propiedad descentralizada por los negocios locales	Propiedad centralizada por pocas Empresas (Amazon, Google, etc.)
Red	Latencia dependiente de la Red de Área Local (LAN)	Latencia de Internet o Ancho de Banda
Escalamiento	Pocos usuarios a la vez	Cientos, Miles o Millones de usuarios a la vez

4.0.5. Arquitectura Cloudlet

Como se describió anteriormente, los *cloudlets* pueden ser usados como intermediarios entre los dispositivos móviles y los servidores en la *nube*. Una de las primeras implementaciones de una arquitectura *cloudlet* fue un prototipo de sistema de computación cloudlet llamado Kimberley [SBCD09], desarrollado por Satyanarayanan *et al.*. En contraste con la arquitectura en nube, Kimberley está conectado a los móviles a través de LAN, no WAN, y es accedido por solamente unos usuarios a la vez. En la Figura 4.1 se muestra la arquitectura *cloudlet* en comparación con la arquitectura en nube. El rol principal que cumple un *cloudlet* es la administración de tareas. También puede colaborar al dispositivo móvil con algún tipo de procesamiento intermedio.

La arquitectura MOCHA [SBH⁺13] (Mobile CLoud Hybryd Architecture) fue creada como una solución a aplicaciones de nube para móviles que requieren una masiva cantidad de tareas paralelizables. Dicha arquitectura permite que laptops, touchpads, y smartphones se connecten a la nube via *cloudlets*, este servicio se realiza usando conexiones de redes múltiples como 3G/4G, Bluetooth, y WiFi. El *cloudlet* determina como dividir el cálculo entre si mismo y múltiples servidores para optimizar la calidad de servicio (QoS, Quality of Service) basado en modelos estadísticos de métricas QoS (latencia, costo monetario, poder de procesamiento, etc.). MOCHA fue demostrado usando tareas de reconocimiento de rostros. Una

arquitectura Similar a Kimberly y a MOCHA, fue propuesta por Ha *et al.*. Esta arquitectura de dos niveles aprovecha un nivel de infraestructura en la nube, y un segundo nivel en el borde o *edge* del internet, sirviendo a los dispositivos móviles incorporados a la red por medio de WiFi. Los *cloudlets* son pequeños centros de datos próximos al usuario, los cuales solo tienen estados flexibles guardados en caché de la nube o de los dispositivos móviles. En algunos escenarios como una construcción de oficina, múltiples *cloudlets* pueden estar localizados cerca a otros y pueden estar conectados de una forma punto-a-punto. En este caso, la ruta entre entre los móviles, servidores en nube, y los *cloudlets* tiene que ser tomada en cuenta. En la investigación de Fesehaye *et al.* [FGNW12] fueron propuestos dos tipos de esquemas de enrutamiento o *routing*: centralizado y descentralizado. En el enrutamiento descentralizado la tabla de enrutamiento está construida y mantenida por los *cloudlets*. Los *cloudlets* periódicamente emiten información de su presencia a los nodos vecinos y los otros *cloudlets*. Cuando un dispositivo móvil recibe un mensaje *broadcast* desde un *cloudlet*, almacena tal información en una tabla de *cloudlets* adjunto a un identificador (ID) del *cloudlet*. Cada móvil también emite periódicamente un mensaje tipo *broadcast* con su identificador, de tal forma que todos los *cloudlets* tengan conocimiento. En el enrutamiento centralizado el servidor central es responsable de preparar y mantener la tabla de rutas. El *cloudlet* periódicamente envía su ID, los ID de sus usuarios móviles y los ID de los *cloudlets* vecinos al servidor central. Consecuentemente, el servidor central calcula la tabla de rutas para cada *cloudlet* e instala la tabla de reenvíos en todos los *cloudlets*.

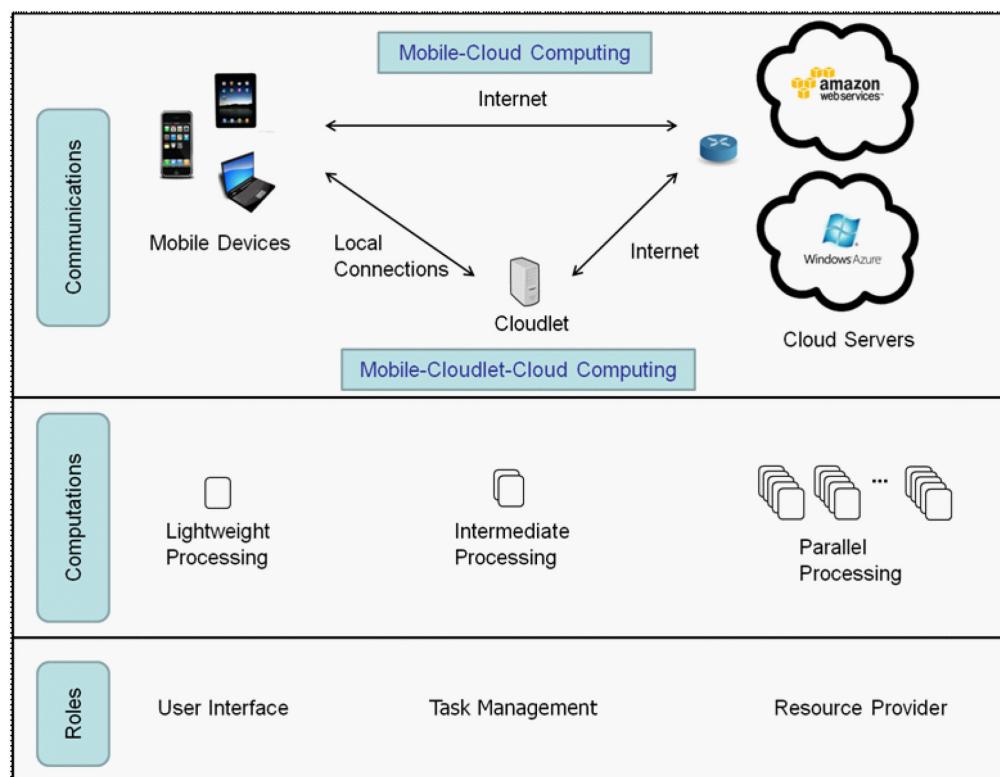


FIGURA 4.1: Diferencias entre las arquitecturas basadas en nube y las basadas en cloudlet
[SBH⁺13]

Capítulo 5

Propuesta: Sistema Pytos

En este capítulo son descritos los niveles jerárquicos involucrados en la Arquitectura de Pytos, a fin de realizar cómputo al borde de la red y optimizar los recursos móviles.

5.1. Arquitectura de Pytos

Pytos es un *framework* basado en Python que será implementado sobre un entorno *cloudlet* con soporte de la nube. En la figura 5.1 se muestran los componentes de Pytos desplegados sobre tres capas jerárquicas: (i) dispositivo cliente, (ii) *cloudlet* y (iii) servidor en nube. La arquitectura propuesta cuenta con un componente privado, el cual es alojado y administrado en servidores de nuestro grupo de investigación. Este componente da soporte en el proceso de descubrimiento de *cloudlets* disponibles en una determinada red de área local. Los propósitos principales de que los servidores de consulta de *cloudlets* sean centralizados, en nube y administrados por Pytos; son: (i) seguridad, los programadores que usan Pytos pueden confiar su código en los *cloudlets* certificados por Pytos, (ii) disponibilidad, los *cloudlet* disponibles en una red son consultados al servidor central de Pytos usando Internet desde cualquier lugar, y iii) facilidad, el desarrollador no se preocupa en instalar o desplegar el componente *cloudlet*

. Entonces, el procedimiento mencionado de *offloading* será realizado de forma transparente al usuario final y con facilidad para el desarrollador. Además de los beneficios en ahorro de energía y tiempo de respuesta en los móviles, consideramos una contribución importante la migración de tareas con envío a la nube, la cual se muestra en la parte a) de la Figura 5.1. A diferencia de la migración de tareas con data de retorno (donde las aplicaciones cliente están a la espera de un resultado desde el cloudlet imagen b) de la Figura 5.1), las aplicaciones con envío directo a nube entregan los resultados a una aplicación en nube especificada por el programador. Este modelo es útil cuando se realiza un pre-procesamiento antes de realizar *streaming* de datos sin necesidad de volver al dispositivo móvil (codificación de video y audio, procesado de imágenes en tiempo real, etc.). En el caso no esté disponible

El principal objetivo de la arquitectura propuesta es mejorar el rendimiento de los servicios basados en *cloudlets* en lugares públicos de gran afluencia, como hospitales, restaurantes, cafetería o centros comerciales. Además, a través de la localización de *cloudlets*, el framework puede facilitar y mejorar el rendimiento de aplicaciones que demanden gran cantidad de recursos, sensibles a la demora, trayendo los recursos de la nube más cerca al usuario final.

Se propone un entorno de programación donde los desarrolladores seleccionen los métodos deseados de una aplicación que podrían ser migrados para su ejecución remota. Cada vez que un método sea invocado y el servidor remoto esté disponible, Pytos usará su framework de optimización para decidir si el método debe ser migrado en tiempo de ejecución. Con el objetivo de maximizar los beneficios de la infraestructura *cloudlet*, se considera los siguientes componentes (ilustrados en la figura 5.2) :

1. **Biblioteca de Pytos:** Este componente provee la sintaxis para usar los decoradores de Pytos (descrito en la sección 5.3), el cual es precisado para interceptar el método de cómputo elevado. Además, inicializará un proceso demonio *Pytos Daemon* si no ha sido inicializado uno anteriormente.

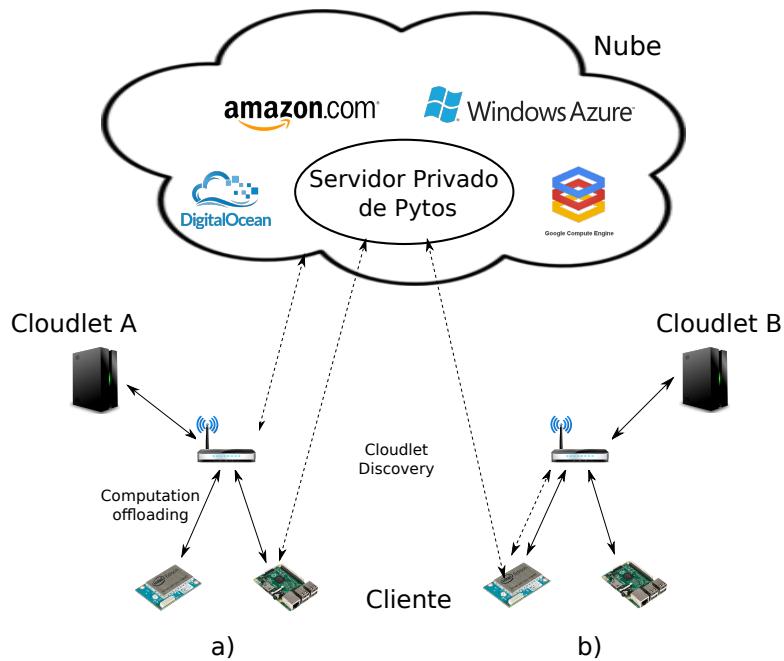


FIGURA 5.1: Vista global de la arquitectura propuesta de Pytos

2. Pytos Daemon: Es el componente fundamental del sistema de Pytos, el cual es compuesto de tres módulos principalmente:

- El recopilador, que recauda información del contexto del móvil como: el estado de la red inalámbrica y el tiempo de ejecución en el servidor y el *cloudlet*. Seguidamente, los valores del recopilador son enviados a la función de costo;
- El evaluador o planificador de tareas, es responsable de evaluar la función de costo, ya que toma la decisión de *offloading* más adecuada; y
- El explorador, destinado a buscar nuevos *cloudlets* disponibles. Como fue discutido anteriormente, este módulo consulta el repositorio de *cloudlets* de la aplicación localizados en el servidor en nube.

3. Modelo de predicción: En este componente se aborda el medio para optimizar la decisión de *offloading*. En esta escena, un modelo probabilístico de regresión lineal es implementado para pronosticar el tiempo de ejecución de un método migrado al lado

del *cloudlet*. Entonces, los valores estimados son retornados al evaluador como una entrada en la función de costo.

4. **Endpoints¹ de descubrimiento:** Ubicados en el componente *cloudlet* donde una Interface de entrada en un Servidor Web (Web Server Gateway Interface, WSGI)² se ejecuta con la intención de dar soporte al cliente en el proceso de descubrimiento de *cloudlets*. Se aprovecha el recurso centralizado en la nube para almacenar un repositorio general de *cloudlets*.
5. **Contenedor Pytos:** En este componente se ejecutará la tarea migrada de la aplicación. La función decorada definida en el código de programa inicial es convertida a *endpoints* abiertos y accesible con un token de autorización basado en el modelo distribuido REST. Los resultados son enviados nuevamente al cliente, o enviados directamente a la aplicación en nube especificada por el desarrollador. Cada tarea migrada al poseer un *token* único se podrá almacenar y utilizar nuevamente si es requerida por un periodo de tiempo. Los *endpoints* son declarados usando un Identificador de Recurso Único (URI, Uniform Resource Identifier) [WWP⁺].
6. **Aplicación web del Desarrollador:** Este componente es opcional. Si el desarrollador lo determinó en su aplicación cliente, el resultado será enviado desde el contenedor Pytos hasta el servidor definido por el desarrollador. Por ejemplo, puede ser implementado un juego cooperativo de realidad aumentada, el cual requiera procesar a cada frame de video antes de ser enviado a Internet. Creemos que tal proceso aliviaría el costo operacional de los servidores en nube, delegando la tarea pesada de procesamiento de los frames a los *cloudlets*.

En las secciones siguientes se describen los componentes indicados anteriormente en mayor detalle.

¹<https://cloud.google.com/appengine/docs/java/endpoints/>, Último acceso en setiembre 2015
²<https://www.python.org/dev/peps/pep-0333/>, último acceso en junio 2015

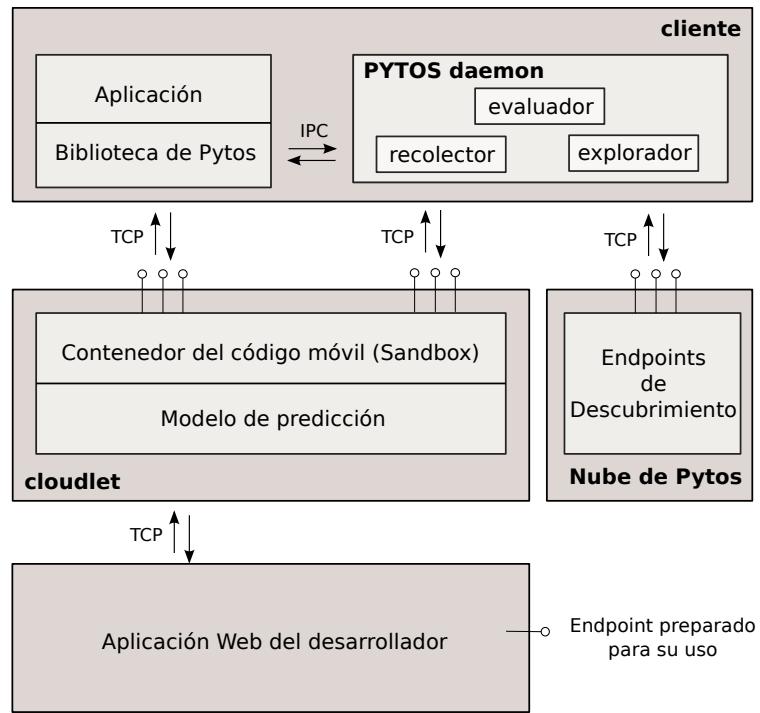


FIGURA 5.2: Detalles de interacción entre los componentes de Pytos

5.2. Diseño de Pytos

Después de la introducción inicial de las ventajas de migración de cómputo en el capítulo 4 como una técnica para mejorar el rendimiento y reducir el uso de la batería en aplicaciones con cálculos intensivos; en esta sección son descritos los detalles de diseño sobre la adición de la técnica de *offloading* a una aplicación de una manera simple sin un esfuerzo adicional. Nuestro enfoque tiene como objetivo la simplicidad, considerando:

1. El uso de Python ³, este lenguaje de programación de alto nivel es ampliamente usado para propósitos generales y tiene una gran variedad de bibliotecas, las cuales en su gran mayoría son actualizadas periódicamente.
2. El uso de los *decoradores* de Python ⁴, pues permite la selección a nivel de métodos. Tal cualidad permite a los desarrolladores agregar *offloading* de una manera sencilla.

³<https://www.python.org/>, último acceso en junio 2015

⁴<https://www.python.org/dev/peps/pep-0318/>, último acceso en junio 2015

3. El proceso de descubrimiento de *cloudlets* simple y transparente al usuario final

5.3. Modelo de Programación

Python tiene un modelo de programación potente, con características avanzadas y sintaxis expresiva. Uno de ellos son los *decoradores*, los cuales generalmente son usados para alterar dinámicamente la funcionalidad de un método sin tener que usar directamente el uso de subclases. Esta característica encaja adecuadamente en nuestro enfoque para extender la funcionalidad de un método que *puede* ser migrado fácilmente. En el código 5.1 se ilustra como un método es *decorado* con la sintaxis de Pytos. Por lo tanto, el motor de Pytos podrá decidir en tiempo de ejecución si el *offloading* es ventajoso o no.

```
@offload
def faceRecognition(self,img):
    faces = faceCascade.detectMultiScale(img,CTE1, CTE2)
    return faces
```

LISTING 5.1: Ejemplo del uso de la sintaxis de Pytos sobre un método que *puede* ser migrado

Luego que las funciones son seleccionadas, lo que el motor de Pytos hace es : 1) interceptará el código del método, 2) generar un código único o *token* que identifique al método, 3) encolar la tarea en el planificador inmediatamente, 4) revisar si es viable el envío de código al *cloudlet*. 5) instalar el código de la función en el *cloudlet* si es necesario, 6) ejecutar la tarea (local o remotamente) y esperar por el resultado (si no existe soporte de la nube). Las funciones *decoradas* serán convertidas en puntos de salida HTTP o *endpoints* de la aplicación en el *cloudlet* con identificadores URIs⁵.

⁵<http://www.ltg.ed.ac.uk/~ht/WhatAreURIs/>

5.3.1. Flujo del Sistema

Nuestra propuesta de framework llevará a las aplicaciones móviles basadas en *cloudlets* por los siguientes pasos antes de realizar la migración de computación al *cloudlet*. En la Figura 5.3 se muestra el proceso de *offloading* de cómputo.

1. *Proceso de descubrimiento de cloudlets*: En esta fase, el explorador de Pytos solicita los *cloudlets* disponibles en una red. Esta demanda toma el Identificador de Conjunto de Servicio⁶ (SSID, Service Set Identifier) de la red WLAN que esté conectada en el momento como un parámetro de entrada.
2. *Decisión de offloading*: Consecuentemente, en el siguiente paso es realizada la decisión inteligente de *offloading*. En el cual el objetivo de Pytos es optimizar el rendimiento, por lo que el ancho de banda de la red y los tiempos de ejecución son recolectados y transferidos a la función de evaluación (discutido en la sección 5.4.2)
3. *Ejecución de tareas pesadas*: La función de cómputo intensivo es ejecutada. Si es ejecutada remotamente, los parámetros del método y respuestas son colocados en un archivo con notación de Objeto JavaScript⁷ (JSON, JavaScript Object Notation). Entonces, estos datos livianos son enviados sobre la capa de transporte TCP.

5.4. Implementation

En esta sección resaltamos los detalles de implementación del framework Pytos. Se describe como el recopilador de red recolecta la información relevante del contexto móvil con el objetivo de realizar un *offloading* inteligente antes de enviar el cómputo al *cloudlet*. Finalmente, explicamos como el costo de evaluación es realizado priorizando el aspecto del rendimiento.

⁶<http://www.webopedia.com/TERM/S/SSID.html>

⁷<http://json.org/>, último acceso en febrero 2015

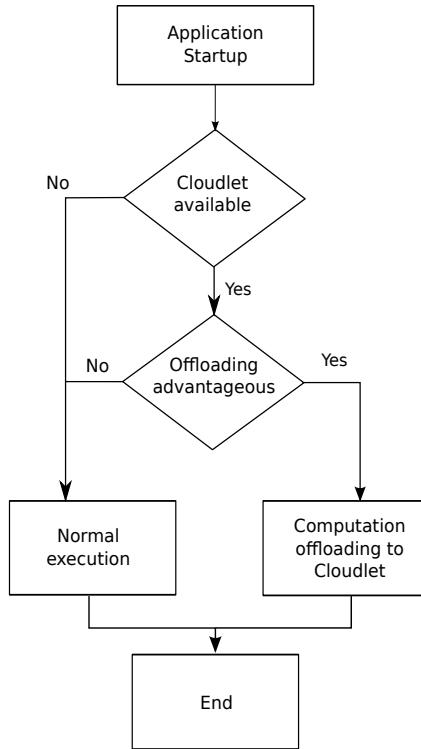


FIGURA 5.3: Flujo del sistema de Pytos

5.4.1. Recopilador

Como mencionamos anteriormente, Pytos será capaz de realizar una decisión de *offloading* inteligente en tiempo de ejecución: si la ejecución de la función debe ejecutarse remotamente o localmente. Para alcanzar tal objetivo, se implementará un recopilador a fin de recolectar toda la información importante del contexto móvil, como: 1) características de la Red WLAN (ratio de transferencia, latencia) y 2) características del contexto de la aplicación relacionadas al tiempo de ejecución en el *cloudlet* y localmente (dispositivo móvil).

Para reunir el estado de la red, nosotros consideramos la técnica usada por Cuervo *et al.* [CBC⁺10]. Se implementará un proceso demonio de Pytos para enviar un paquete de 10kb al cloudlet a fin de medir el tiempo de transferencia. Usando este simple enfoque permite que el recolector de Pytos obtenga los valores estimados de la latencia y ancho de banda. Tal procedimiento es realizado cada tres minutos para mantener un valor estimado

actualizado. Similarmente, el proceso demonio de Pytos consulta el *cloudlet* por información estadística basados en históricos sobre las funciones de cómputo pesado que fueron realizadas en el pasado. Luego, tales valores (1) y (2) son enviados a la función de costo (discutido en la Sección 5.4.2)

5.4.2. Evaluador

Nuestra investigación se centra en minimizar el tiempo de ejecución de una función de cómputo pesado. En tal sentido, se construirá una función de evaluación que considere lo siguiente: tiempo de ejecución en el *cloudlet* y en el cliente, además del tiempo de transferencia. Estos valores son obtenidos del componente recopilador de Pytos. Usaremos la ecuación 3.3 explicada en la sección 4 como función evaluadora.

5.5. Resultados Iniciales

En esta sección describimos la configuración experimental adoptada para evaluar la migración de tareas en un entorno *cloudlet*. Dicho experimento en que factor el *offloading* de cómputo es posible ahorrar energía y mejorar el rendimiento. Por lo que, se implementó una aplicación de reconocimiento de rostros usando *offloading* y fue probado sobre un conjunto de imágenes.

5.5.1. Configuración experimental

Con el objetivo de demostrar las ventajas del *offloading* en un entorno *cloudlet*, se consideraron los componentes de una jerarquía de tres niveles: cliente, *cloudlet* y nube.

En el cliente, se utilizó una mini-computadora Intel Edison que incorpora la placa de expansión de Arduino⁸ para direccionar la conexión con interfaces de Entrada/Salida externas como dispositivos de cámara web USB, tarjeta de expansión de memoria micro SD y entrada de energía. Esta placa incorpora un CPU Intel Atom con una frecuencia de 500 MHz de doble núcleo, 1 GB de memoria RAM y una antena WiFi interna IEEE 802.11a/b/g/n.

Se usó como sistema operativo Ubilinux⁹ de la distribución Debian (versión 150309). En el *cloudlet* se utilizó una computadora de escritorio equipada con un CPU Intel Core i7-4700MQ a una frecuencia de 2.40 GHz, 11.5 GiB de memoria RAM, y Ubuntu 14.04 como sistema operativo. En el servidor en nube se dispuso de una máquina virtual con Ubuntu 14.04 (7 núcleos y 16 GiB de memoria RAM), el cual pertenece a la empresa Digital Ocean¹⁰ y está físicamente localizado en New York. Durante los ensayos, el ancho de banda de la conexión entre el cliente y el *cloudlet*, medido con la herramienta Iperf benchmark¹¹, fue aproximadamente 8.90 Mbits/seg. Todas las conexiones fueron realizadas sobre el protocolo TCP, usando un Punto de Acceso provisto por TP-link, en su modelo Archer C2. Para obtener medidas de energía a grano fino, se consideró el uso de la herramienta PowerAPI [BNRS13], una biblioteca de software para monitorear la energía consumida a nivel de procesos. Dicha herramienta, nos permite conseguir de manera estándar y con alta precisión (similar a los multímetros convencionales) las unidades *watt* consumidas por el proceso deseado. Relegamos el consumo de energía realizado por la interfase de WiFi, ya que este valor es dominado por el consumo de energía del CPU [CDJ⁺15].

La metodología usada para obtener la energía y tiempo de respuesta se muestra en la Figura 5.4. En tal procedimiento, el recopilador de datos, que está compuesto por el PowerAPI y un colector de marcas de tiempo, requiere el identificador de proceso o PID de la aplicación que implementa *offloading* con Pytos.

⁸<https://www-ssl.intel.com/content/www/us/en/do-it-yourself/edison.html>, Último acceso en setiembre 2015

⁹<http://www.emutexlabs.com/ubilinux>, último acceso en setiembre 2015

¹⁰<https://www.digitalocean.com/>, último acceso en setiembre 2015

¹¹(<https://iperf.fr/>), último acceso en setiembre 2015

Antes de que se ejecute el *offloading* de una aplicación, tal aplicación se detiene hasta que una señal de aviso del recopilador de datos es recibida. Entonces, la aplicación que implementa Pytos comienza la tarea de cómputo pesado; simultáneamente, el consumo de energía y tiempo son recolectados para ser exportados.

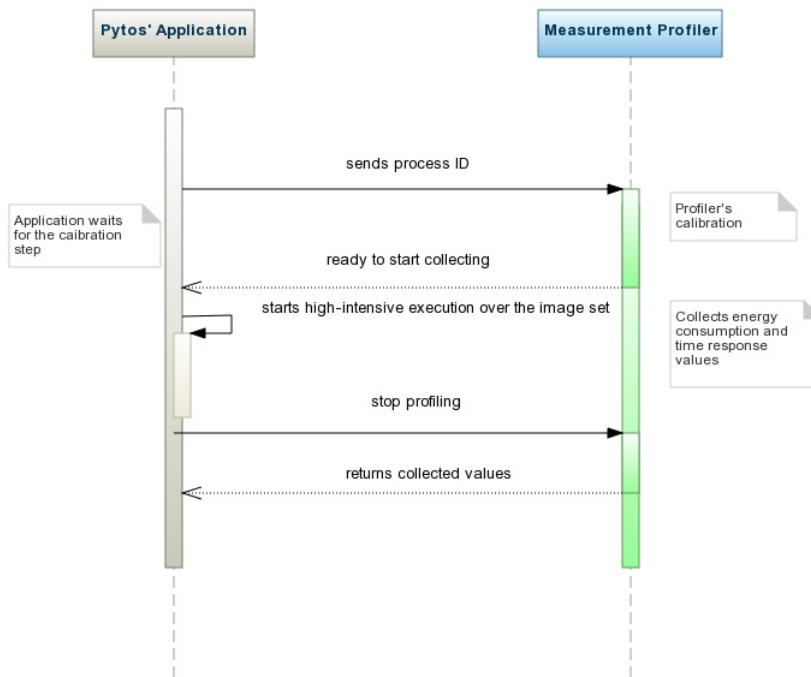


FIGURA 5.4: Metodología para la medición

5.5.2. Escenarios de Evaluación

Evaluamos la migración de tareas pesadas usando una aplicación de reconocimiento de rostros. Como mencionamos anteriormente, los desarrolladores deberán seleccionar en fase de diseño, los métodos costosos computacionalmente. Claramente, en este caso la operación que más necesita de recursos es la de reconocimiento de rostros. De tal forma, si el cómputo es enviado a un *cloudlet*, la aplicación principal espera por la respuesta a fin de ser mostrado en el dispositivo limitado. En el cuadro 5.1 se muestra el conjunto de imágenes que fueron

CUADRO 5.1: Datos de los diferentes conjuntos de imágenes usados en la aplicación de reconocimiento de rostros

Set	Resolution (width x height)	Average size (bytes)	Standard deviation
1	160x120	7612	141
2	176x144	8959	203
3	416x240	21478	1183
4	352x288	23548	1494
5	800x600	84668	810
6	960x544	89023	5098

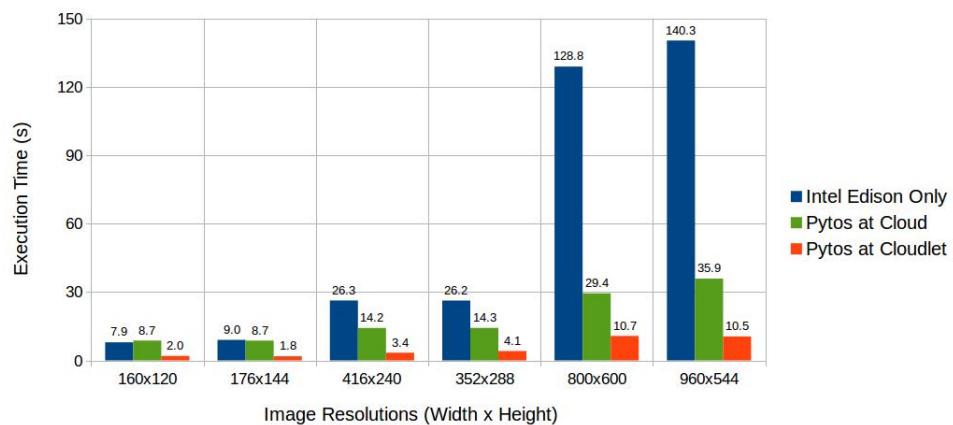


FIGURA 5.5: Comparación de tiempo de respuestas usando diferentes escenarios.

probados con nuestra aplicación. Cada conjunto contiene 20 imágenes que fueron obtenidas de una misma cámara web.

5.5.3. Resultados Iniciales

Se evaluó localmente (cliente) y remotamente (*cloudlet* y nube) el rendimiento y el ahorro de energía de una aplicación que usualmente consume abundantes recursos (descrito en la Sección 5.5.2). Los experimentos son ejecutados en tres escenarios: (1)solamente en el dispositivo Intel Edison (2) en la infraestructura *cloudlet*, y (3) sobre la infraestructura en nube. El último escenario no es un objetivo del framework propuesto, sin embargo, lo consideramos para notar el impacto de la demora en los escenarios WAN.

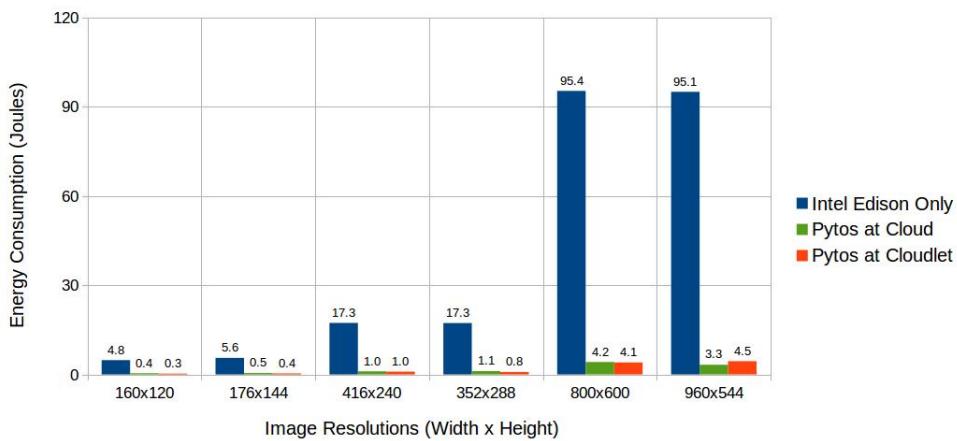


FIGURA 5.6: Comparación de energía consumida usando diferentes escenarios.

La Figura 5.5.2 muestra la diferencia de rendimiento al procesar los conjuntos (1), (2) y (3) de las imágenes variables (Cuadro 5.1). Como se esperaba, a medida que la imagen de entrada a ser procesada era más grande, más tiempo se requería para su procesamiento; en todos los escenarios. Por ejemplo, para la ejecución aislada en Intel Edison varía desde 7.94 segundos (s) (con una desviación estándar de 0.0215) para procesar el conjunto 1 de imágenes más liviano a 140 s (con una desviación estándar de 0.2571) para procesar el conjunto más pesado. Mientras tanto, empleando la infraestructura *cloudlet* se utilizó 1.98 s (con una desviación estándar de 0.4839) para procesar el conjunto 1, y 10.53 s para procesar el conjunto 6, eso significa que el rendimiento se acrecentó en un factor de 4 a 13 veces. Los experimentos usando la nube muestran un rendimiento bueno en comparación con la ejecución aislada en Intel Edison, pero son peores en comparación con los ejecutados en entornos *cloudlets* debido a la latencia WAN elevada. Por lo que se confirma que la ejecución de tareas pesadas en *cloudlets* es una mejor apuesta.

Similarmente, la Figura 5.6 muestra la misma tendencia en ahorro de energía. La energía en ejecución aislada en Intel Edison varía desde 4.81 Joules (J) (con una desviación estándar de 0.0822) para procesar el conjunto 1, a 95.06 J (con una desviación estándar de 0.3077) para procesar el último conjunto de imágenes 6. Por otra parte, migrando cómputo al *cloudlet*

varía de 0.29 J (con una desviación estándar de 0.0217) para procesar el conjunto 1 (economizando energía en un factor de 16), a 4,45 J (desviación estándar de 0.1884) para procesar el conjunto 6 (economizando energía en un factor de 21).

Capítulo 6

Cronograma de Trabajo

Actividades	Años y Semestres								
	2014		2015				2016		
	1	2	Febrero	Marzo	Abril	1	2	Enero	Febrero
Disciplinas Obligatorias	✓	✓				✓	□		
Estudio de la Bibliografía y estado del Arte			✓	✓	✓				
Estudio de métricas de evaluación					✓	✓			
Implementación de propuestas						✓	□		
Presentación de Artículos						✓	□	□	
Redacción de la tesis y defensa						□	□	□	□

CUADRO 6.1: Plan de trabajo

Capítulo 7

Conclusiones

En este trabajo se propone Pytos, un framework para emplear *offloading* en entornos *cloudlet*, lo cual permitirá ahorrar energía y optimizar el rendimiento de aplicaciones móviles que consuman abundantes recursos. Nuestro enfoque lleva la computación al borde de la red y visiona solucionar una de las principales limitaciones de los frameworks basados en nube: la demora de las redes WAN.

La arquitectura propuesta es transparente para los usuarios finales, ya que descubre los *cloudlets* disponibles y decide si el *offloading* es factible , o no, de manera automática. A los desarrolladores, nuestro modelo permitirá una integración simple y una migración de código de grano fino usando decoradores de Python. Como fue demostrado en las evaluaciones empíricas, la migración de cómputo a los *cloudlets* puede conservar energía hasta un factor de 20.

Además, la arquitectura propuesta puede ser ampliamente usada. Cada aplicación que precise de recursos adicionales (codificación de video, procesamiento de señales, juegos, etc.) puede ser asistida la computación en el borde de la red. Sin embargo, en el modelo propuesto, el desarrollador escoger la tarea conveniente que pueda ser migrada sin efectos secundarios en la aplicación. Por ejemplo, un método no puede ser migrado si este requiere llamadas al sistema

operativo, como lecturas de sensores, uso de instrucciones de Entrada/Salida, o consultas a base de datos locales. En versiones futuras, se visiona solucionar tales limitaciones.

La arquitectura Pytos, si es desplegada cerca a entornos de red inalámbricas, puede mejorar la experiencia de usuario, ofreciendo, por ejemplo, un mejor servicio en áreas comerciales, como cafeterías, centros comerciales, y restaurantes. Si es desplegado ampliamente, los *cloudlets* pueden reducir el tráfico de internet a nivel global.

Bibliografía

- [ASG12] Saeid Abolfazli, Zohreh Sanaei, and Abdullah Gani. Mobile cloud computing: A review on smartphone augmentation approaches. *arXiv preprint arXiv:1205.0451*, 2012.
- [Bal06] Rajesh Krishna Balan. *Simplifying cyber foraging*. School of Computer Science, Carnegie Mellon University, 2006.
- [BMZA12] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, New York, NY, USA, 2012. ACM.
- [BNRS13] Aurélien Bourdon, Adel Noureddine, Romain Rouvoy, and Lionel Seinturier. Powerapi: A software library to monitor the energy consumed at the process-level. *ERCIM News*, 92:43–44, January 2013.
- [BSPO03] Rajesh Krishna Balan, Mahadev Satyanarayanan, So Young Park, and Tadashi Okoshi. Tactics-based remote execution for mobile computing. In *Proceedings of the 1st international conference on Mobile systems, applications and services*, pages 273–286. ACM, 2003.
- [CBC⁺10] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: Making smartphones last longer with code offload. In *Proceedings of the 8th International Conference*

- on Mobile Systems, Applications, and Services*, MobiSys '10, pages 49–62, New York, NY, USA, 2010. ACM.
- [CDG⁺06] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, OSDI '06, pages 15–15, Berkeley, CA, USA, 2006. USENIX Association.
- [CDJ⁺15] Xiaomeng Chen, Ning Ding, Abiliash Jindal, Y Charlie Hu, Maruti Gupta, and Rath Vannithamby. Smartphone energy drain in the wild: Analysis and implications. 2015.
- [CFH⁺05] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 273–286. USENIX Association, 2005.
- [CIM⁺11] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, pages 301–314. ACM, 2011.
- [DHJ⁺07] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon’s highly available key-value store. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 205–220. ACM, 2007.

- [DLNW13] Hoang T Dinh, Chonho Lee, Dusit Niyato, and Ping Wang. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing*, 13(18):1587–1611, 2013.
- [DWC10] Tharam Dillon, Chen Wu, and Elizabeth Chang. Cloud computing: issues and challenges. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pages 27–33. Ieee, 2010.
- [Ehr10] David Ehringer. The dalvik virtual machine architecture. *Techn. report (March 2010)*, 4, 2010.
- [FGNW12] D. Fesehaye, YunLong Gao, K. Nahrstedt, and Guijun Wang. Impact of cloudlets on interactive mobile cloud applications. In *Enterprise Distributed Object Computing Conference (EDOC), 2012 IEEE 16th International*, pages 123–132, Sept 2012.
- [FLR13] Niroshnie Fernando, Seng W Loke, and Wenny Rahayu. Mobile cloud computing: A survey. *Future Generation Computer Systems*, 29(1):84–106, 2013.
- [FPS02] Jason Flinn, SoYoung Park, and Mahadev Satyanarayanan. Balancing performance, energy, and quality in pervasive computing. In *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pages 217–226. IEEE, 2002.
- [GJM⁺12] Mark S Gordon, Davoud Anoushe Jamshidi, Scott A Mahlke, Zhuoqing Morley Mao, and Xu Chen. Comet: Code offload by migrating execution transparently. In *OSDI*, pages 93–106, 2012.
- [GKW04] Selim Gurun, Chandra Krintz, and Rich Wolski. Nwslite: a light-weight prediction utility for mobile devices. In *Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 2–11. ACM, 2004.

- [GNM⁺04] Xiaohui Gu, Klara Nahrstedt, Alan Messer, Ira Greenberg, and Dejan Milojevic. Adaptive offloading for pervasive computing. *Pervasive Computing, IEEE*, 3(3):66–73, 2004.
- [KAH⁺11] Sokol Kosta, Andrius Aucinas, Pan Hui, Richard Mortier, and Xinwen Zhang. Unleashing the power of mobile cloud computing using thinkair. *CoRR*, abs/1105.3232, 2011.
- [KL10] K. Kumar and Yung-Hsiang Lu. Cloud computing for mobile users: Can offloading computation save energy? *Computer*, 43(4):51–56, April 2010.
- [KLLB13] Karthik Kumar, Jibang Liu, Yung-Hsiang Lu, and Bharat Bhargava. A survey of computation offloading for mobile systems. *Mobile Networks and Applications*, 18(1):129–140, 2013.
- [KOMK14] A.R. Khan, M. Othman, S.A. Madani, and S.U. Khan. A survey of mobile cloud computing application models. *Communications Surveys Tutorials, IEEE*, 16(1):393–413, First 2014.
- [PKKB12] Roelof Kemp, Nicholas Palmer, Thilo Kielmann, and Henri Bal. Cuckoo: a computation offloading framework for smartphones. In *Mobile Computing, Applications, and Services*, pages 59–79. Springer, 2012.
- [Kri10] M.D. Kristensen. Scavenger: Transparent development of efficient cyber foraging applications. In *Pervasive Computing and Communications (PerCom), 2010 IEEE International Conference on*, pages 217–226, March 2010.
- [LS13] Kilho Lee and Insik Shin. User mobility-aware decision making for mobile computation offloading. In *Cyber-Physical Systems, Networks, and Applications (CPSNA), 2013 IEEE 1st International Conference on*, pages 116–119. IEEE, 2013.

- [MG09] Peter Mell and Tim Grance. The nist definition of cloud computing. *National Institute of Standards and Technology*, 53(6):50, 2009.
- [MM09] Alin Florindor Murarasu and Thomas Magedanz. Mobile middleware solution for automatic reconfiguration of applications. In *Information Technology: New Generations, 2009. ITNG'09. Sixth International Conference on*, pages 1049–1055. IEEE, 2009.
- [MN10] Antti P Miettinen and Jukka K Nurminen. Energy efficiency of mobile clients in cloud computing. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pages 4–4. USENIX Association, 2010.
- [OH98] Mazliza Othman and Stephen Hailes. Power conservation strategy for mobile computers using load sharing. *SIGMOBILE Mob. Comput. Commun. Rev.*, 2(1):44–51, January 1998.
- [PLJC14] C. Perera, C.H. Liu, S. Jayawardena, and Min Chen. A survey on internet of things from industrial market perspective. *Access, IEEE*, 2:1660–1679, 2014.
- [PM83] Michael L. Powell and Barton P. Miller. Process migration in demos/mp. Technical Report UCB/CSD-83-132, EECS Department, University of California, Berkeley, Aug 1983.
- [PS05] J.A. Paradiso and T. Starner. Energy scavenging for mobile and wireless electronics. *Pervasive Computing, IEEE*, 4(1):18–27, Jan 2005.
- [RP03] Peng Rong and Massoud Pedram. Extending the lifetime of a network of battery-powered mobile devices by remote processing: a markovian decision-based approach. In *Proceedings of the 40th annual Design Automation Conference*, pages 906–911. ACM, 2003.

- [RRL⁺14] M.Reza Rahimi, Jian Ren, ChiHarold Liu, AthanasiosV. Vasilakos, and Nalini Venkatasubramanian. Mobile cloud computing: A survey, state of art and future directions. *Mobile Networks and Applications*, 19(2):133–143, 2014.
- [RS10] A. Ranabahu and A. Sheth. Semantics centric solutions for application and data portability in cloud computing. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 234–241, Nov 2010.
- [Ryu12] Seewon Ryu. Telemedicine: Opportunities and developments in member states: Report on the second global survey on ehealth 2009 (global observatory for ehealth series, volume 2). Technical Report 2, 2012.
- [SAGB14] Zohreh Sanaei, Saeid Abolfazli, Abdullah Gani, and Rajkumar Buyya. Heterogeneity in mobile cloud computing: taxonomy and open challenges. *Communications Surveys & Tutorials, IEEE*, 16(1):369–392, 2014.
- [Sat01] M. Satyanarayanan. Pervasive computing: vision and challenges. *Personal Communications, IEEE*, 8(4):10–17, Aug 2001.
- [Sat11] Mahadev Satyanarayanan. Mobile computing: The next decade. *SIGMOBILE Mob. Comput. Commun. Rev.*, 15(2):2–10, August 2011.
- [SCBD09] Mahadev Satyanarayanan, P. Bahl, R Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE*, 8(4):14–23, Oct 2009.
- [SBH⁺13] Tolga Soyata, He Ba, Wendi Heinzelman, Minseok Kwon, and Jiye Shi. Accelerating mobile cloud computing: A survey. *Communication Infrastructures for Cloud Computing*, pages 175–197, 2013.
- [SCB11] Deepak Shivarudrappa, M Chen, and Shashank Bharadwaj. Cofa: Automatic and dynamic code offload for android. *University of Colorado, Boulde*, 2011.

- [SCH⁺14] Mahadev Satyanarayanan, Zhuo Chen, Kiryong Ha, Wenlu Hu, Wolfgang Richter, and Padmanabhan Pillai. Cloudlets: at the leading edge of mobile-cloud convergence. IEEE, 11 2014.
- [SMF⁺12] T. Soyata, R. Muraleedharan, C. Funai, Minseok Kwon, and W. Heinzelman. Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture. In *Computers and Communications (ISCC), 2012 IEEE Symposium on*, pages 000059–000066, July 2012.
- [Tal10] A.K. Talukdar. *Mobile Computing, 2E*. McGraw-Hill communications engineering series. McGraw-Hill Education (India) Pvt Limited, 2010.
- [tra14] Mobile applications (store type - native (on-deck), third-party (off-deck)) market - global industry analysis, size, share, growth, trends and forecast 2014 - 2020. Technical report, Transparency Market Research, 2014.
- [TYRW14] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1701–1708. IEEE, 2014.
- [VRC13] N. Vallina-Rodriguez and J. Crowcroft. Energy management techniques in modern mobile handsets. *Communications Surveys Tutorials, IEEE*, 15(1):179–198, First 2013.
- [WGKN08] Richard Wolski, Selim Gurun, Chandra Krintz, and Daniel Nurmi. Using bandwidth data to make computation offloading decisions. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–8. IEEE, 2008.

- [WWP⁺] P N Walsh, Peter N. Walsh, D. Phil, Tulay A. Kansu, James J. Corbett, Peter J. Savino, Warren P. Goldburgh, Norman, and J. Schatz. Architecture of the world wide web. In *W3C, Tech. Rep., 2004. [Online]*. Available: <http://www.w3.org/TR/webarch>.