Energieeffizienz durch Computation Offloading in der Cloud

Matthias Fey
TU Dortmund, Fakultät für Informatik
Otto-Hahn-Straße 14
44227 Dortmund, Germany
matthias.fey@tu-dortmund.de

ABSTRACT

Die Popularität von Smartphones und dessen Applikationen ist innerhalb der letzten Jahre stark gestiegen. Immer komplexer werdende Programme wie Objekterkennung oder Videobearbeitung stehen mittlerweile auch mobil zur Verfügung. Je aufwändiger die Applikationen werden, umso mehr mindern sie dabei die Akkulaufzeit der Geräte. Energieeffizienz ist daher ein wichtiges Thema bei der Entwicklung von mobilen Applikationen. Computation Offloading, d. h. die Auslagerung von komplexen Berechnungen an leistungsstärkere Computer, ist ein möglicher Anwärter für die Erstellung energieeffizienterer Applikationen.

Diese Arbeit beleuchtet das Computation Offloading und zeigt den aktuellen Stand in der Forschung auf. Neben der Grundidee des Computation Offloadings und der Vorstellung bisheriger Arbeiten, beschäftigt sich diese Arbeit mit der Frage, ob Computation Offloading eine geeignete Möglichkeit darstellt, energieeffizientere Applikationen zu entwickeln.

EINLEITUNG

Das Smartphone revolutioniert unser Leben. So warb Apple schon 2010 mit dem Slogan „There's an app for everything" und zeigt damit auf, wohin die Reise geht. Mittlerweile surfen wir auf unseren mobilen Endgeräten im Internet, checken unsere Emails, schauen uns Videos an und betreiben Social Media. Dabei können wir dank zusätzlicher Komponenten und Sensoren am Smartphone wie Kamera oder GPS sogar weitaus mehr bewerkstelligen als am Computer und sind dabei noch mobil. Es stehen täglich neue innovative und revolutionäre mobile Applikationen zum Download zur Verfügung. Laut Forschung wächst die App-Industrie im Jahr 2015 exponentiell auf einen Marktwert von 37 Milliarden $ an [7].

Das Smartphone ist alltäglich geworden. Haben wir es nicht bei uns oder neigt sich der Akku dem Ende, fühlen wir uns unverzüglich eingeschränkt in unseren Aktivitäten. Das hat zur Folge, dass wir bei geringer Akkulaufzeit unseren Konsum ungewollt drosseln und unser Smartphone daher nicht zu vollem Potenzial nutzen können. Je mehr unsere Nachfrage nach komplexer werdenden Programmen auf Smartphones steigt, umso unglücklicher werden wir mit der Akkulaufzeit. Eine Umfrage mit tausenden von Smartphone-Nutzern hat ergeben, dass die Verbesserung der Akkulaufzeit der größte Wunsch des Endnutzers ist [9].

Der Nutzen eines Smartphones ist und bleibt limitiert durch dessen Akkulaufzeit. Die Batteriekapazität wird dabei gravierend eingeschränkt durch die Größe und das Gewicht der Endgeräte. Das hat zur Folge, dass die Energiedichte von Batterien in Smartphones mit einer recht unbedeutenden Rate von nur 5% jährlich anwächst [20]. Größere Smartphones mit stärkeren Akkus sind jedoch keine attraktive Alternative. Ebenso beschränken thermische Anforderungen die Leistung von Smartphones auf ungefähr 3 Watt [17].

Energieeffizienz ist daher ein wichtiges aber ebenso unterschätztes Thema bei der Entwicklung von benutzerfreundlichen mobilen Applikationen. Optimales Energiemanagement erfordert ein gutes Verständnis über die Energienutzung von Smartphones [4]. Applikationen wurden in der Vergangenheit nur mit wenig Rücksicht auf die Energiekosten eines Smartphones gestaltet [19]. Bspw. verbrauchen 6 der 10 beliebtesten Apps im Google-Play-Store über einer Laufzeit von 30 Sekunden bis zu 0,75% einer vollen Akkuladung [19].

In dieser Arbeit wird das *Computation Offloading in der Cloud* als Möglichkeit zur Energieeinsparung in Smartphone-Applikationen vorgestellt. Computation Offloading bedeutet, dass komplexe Aufgaben oder Berechnungen in mobilen Applikationen an leistungsstärkere Computer (z. B. die Cloud) ausgelagert werden. Im nachfolgenden Kapitel wird auf die Grundidee des Offloadings und insbesondere auf dessen Entscheidungsfindung eingegangen: Wann ist Offloading überhaupt praktikabel? Wie werden mögliche Bereiche einer Applikation für das Offloading bestimmt? Und welche Typen von Applikationen können bspw. durch Computation Offloading energieeffizienter gestaltet werden?

Daraufhin werden bereits entwickelte Systeme bzw. Frameworks zum Computation Offloading näher vorgestellt. Mit besonderem Augenmerk wird hier das Framework ThinkAir [11] präsentiert, welches sich die Frameworks MAUI [6] und CloneCloud [5] zu Nutze macht.

Im nachfolgenden Abschnitt werden Ergebnisse aus verschiedenen Studien mit unterschiedlichen Testszenarien und Übertragungstechnologien präsentiert [3, 13, 16, 11]. In einer

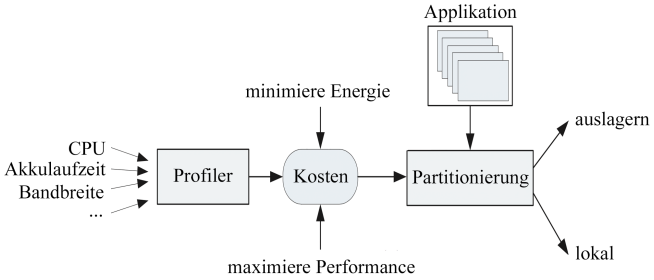

Figure 1. Architektur von Systemen zum Computation Offloading nach [22]

anschließenden Diskussion wird kritisch auf die Frage eingegangen, ob sich Offloading im alltäglichen Gebrauch lohnt und wo mögliche Schwächen liegen. Zum Abschluss wird ein Ausblick auf die Zukunft des Computation Offloadings gegeben und alternative Ansätze für energieeffiziente Applikationen vorgestellt.

COMPUTATION OFFLOADING

Computation Offloading (oder *cyber foraging* bzw. *surrogate computing* [12]) ist die englische Bezeichnung für die Auslagerung von komplexen Aufgaben oder Berechnungen an leistungsstärkere Computer (z. B. die Cloud) über Netzwerkübertragung. Dafür werden die benötigten Daten für die Berechnung vom Smartphone an den entsprechenden Offloading-Server gesendet. Das Smartphone wartet auf die Abarbeitung der Berechnung und empfängt nach Fertigstellung die Ergebnisse vom Server. Durch die Auslagerung verspricht man sich zum einen den Gewinn an Performance (d. h. die schnellere Abarbeitung der Aufgabe und die Nutzung der freigewordenen Ressourcen für andere Anforderungen) und zum anderen die Einsparung von Energiekosten auf dem Smartphone.

Computation Offloading unterscheidet sich von der traditionellen Client-Server-Architektur. Ein Client-Server-System baut auf einem leichtgewichtigen Client auf, der *jede* Berechnung an einen Server auslagert. Im Gegensatz dazu wird beim Computation Offloading nur dann eine Berechnung ausgelagert, wenn sich diese unter bestimmten Gesichtspunkten lohnt. Ebenso muss eine Applikation, die Offloading nutzt, im Vergleich zu einer Client-Server-Architektur nicht zwangsläufig als solche geplant werden. Näheres dazu findet sich im weiteren Verlauf dieses Kapitels.

Die typische Architektur von Systemen zum Computation Offloading zeigt Abbildung 1. Sie besteht typischerweise aus drei Hauptkomponenten: den Profilern, dem Kostenmodell und der Partitionierung.

Profiler sammeln Informationen des Smartphones und dessen Netzwerkumgebung wie CPU-Auslastung, Akkulaufzeit und Bandbreite [22]. Sie sollen dabei möglichst genaue Angaben liefern und dabei so wenig Leistung wie möglich in Anspruch nehmen, um den zusätzlichen Mehraufwand der Applikation für das Computation Offloading gering zu halten [11].

Die Schlüsselkomponente der Architektur ist das *Kostenmodell*. Hier wird anhand von Kriterien und den Profiler-Informationen die Entscheidung getroffen, wann und welche Komponenten oder Methoden einer Applikation in die Cloud ausgelagert werden sollen. In der Abbildung werden die Kriterien „minimiere Energie“ und „maximiere Leistung“ genannt. Es sind aber ebenso auch andere Kriterien denkbar wie das Maximieren von Sicherheit und Datenschutz oder der Robustheit der Applikation [22]. Diese Arbeit beschäftigt sich im weiteren Verlauf nur mit der Einsparung von Energie als Kriterium.

Die *Partitionierung* teilt die Applikation anhand der Ergebnisse des Kostenmodells in zwei Bereiche: eine lokale Partition, die auf dem mobilen Endgerät ausgeführt wird und eine Partition, die in die Cloud ausgelagert wird [22]. Die auszulagernde Partition wird als *Klon* der Anwendung bezeichnet [3].

Die Cloud bietet dabei den idealen Platz für die Auslagerung von komplexen Berechnungen dank ihrer einfachen Skalierbarkeit. Sie bietet nahezu unendlich viele Ressourcen die auf Nachfrage und je nach Auslastung zur Verfügung stehen [16]. Ebenso bietet sie den Vorteil des Multitaskings um mehrere Applikationen und Prozesse simultan laufen zu lassen [12]. Die Popularität der Cloud liegt unter anderem auch daran, dass sie relativ kostengünstig ist [22].

Trade-off

Computation Offloading stellt keinesfalls eine Pflicht zur Erstellung energieeffizienter Applikationen dar. Das liegt insbesondere an den zusätzlichen Energie- und Netzwerkkosten für die Datenübermittlung, die bei der Entscheidung zur Auslagerung bedacht werden müssen [22, 3]. Die Abwägung (engl. *Trade-off* [13]) zwischen dem Energieverbrauch zur Auslagerung der Aufgabe und der Energieeinsparung dank der Cloud entscheidet darüber, wann und ob Computation Offloading sinnvoll ist. Dieser Trade-off lässt sich mathematisch wie folgt definieren [13]:

$$E_{trade} = E_{lokal} - E_{offload}$$

E_{lokal} bezeichnet dabei die Energie, die durch lokale Berechnung auf dem Smartphone verbraucht wird und $E_{offload}$ kennzeichnet den Energieverbrauch des Smartphones, der durch die Auslagerung der Aufgabe entsteht. Nur wenn E_{trade} positiv ist, wird mit Hilfe von Computation Offloading Energie auf dem Smartphone eingespart.

Da sich Energie E durch die durchschnittliche Leistung P über einen bestimmten Zeitraum T bestimmt, also $E = P \times T$, lässt sich E_{lokal} weiter auflösen. Sei C dafür die Komplexität der Aufgabe, d. h. die Anzahl auszuführender Anweisungen, X_{lokal} die Geschwindigkeit der lokalen Berechnung und P_{lokal} die benötigte Leistung zur lokalen Berechnung, dann gilt [13]:

$$E_{lokal} = \frac{P_{lokal} \times C}{X_{lokal}}$$

Die Energiekosten zur Auslagerung der Aufgabe $E_{offload}$ setzen sich dagegen aus den Kosten zum Senden der Daten an

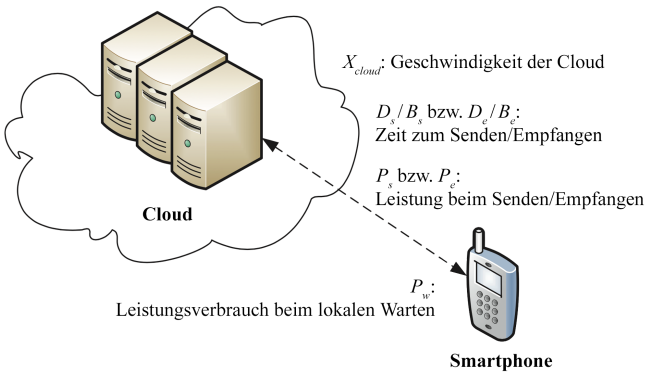

Figure 2. Energiekosten eines Offloading-Prozesses nach [22]

die Cloud E_{senden}, zum lokalen Warten auf die Beendigung der Berechnung in der Cloud E_{warten} und zum Empfangen des Ergebnisses $E_{empfangen}$ zusammen [13]. Die Kosten für das Profiling werden hier nicht berücksichtigt:

$$E_{offload} = E_{senden} + E_{warten} + E_{empfangen}$$

Diese Energien lassen sich erneut mit Hilfe der Verbindung zu Leistung und Zeit auflösen [13]:

$$E_{offload} = P_s \times T_s + P_w \times T_w + P_e \times T_e$$

Abbildung 2 veranschaulicht den Energieverbrauch bei der Auslagerung der Berechnung und die davon abhängigen Variablen. Die Zeit des lokalen Wartens T_w ist von der Ausführungsgeschwindigkeit der Cloud X_{cloud} abhängig. Ebenso lassen sich die Zeitintervalle zum Senden T_s und Empfangen T_e der Daten über die Menge der zu verschickenden und zu empfangenden Daten D_s, D_e sowie den aktuellen Bandbreiten B_s, B_e bestimmen, also $T_s = D_s/B_s$ und analog $T_e = D_e/B_e$ [13]. Hierfür wird die Bandbreite nicht ganz sinngemäß als Übertragungsgeschwindigkeit verstanden. Vorhandene Netzwerklatenzen werden ignoriert bzw. in der Bandbreite mit berücksichtigt. Es ergibt sich damit:

$$E_{trade} = \frac{P_{lokal} \times C}{X_{lokal}} - \frac{P_w \times C}{X_{cloud}} - \frac{P_s \times D_s}{B_s} - \frac{P_e \times D_e}{B_e}$$

$$\Leftrightarrow \quad E_{trade} = C\left(\frac{P_{lokal}}{X_{lokal}} - \frac{P_w}{X_{cloud}}\right) - P_s\frac{D_s}{B_s} - P_e\frac{D_e}{B_e}$$

$P_{lokal}/X_{lokal} - P_w/X_{cloud}$, P_s und P_w sind geräteabhängige, unveränderliche Werte des jeweiligen Smartphones und der jeweiligen Cloud-Plattform. Es wird damit ersichtlich, dass eine hohe Komplexität der auszulagernden Aufgabe das Computation Offloading befürwortet, wohingegen eine geringe Bandbreite oder ein hoher Datenaustausch zwischen den Geräten die Vorteile von Offloading mindern [13].

Statisch vs. dynamisch

Beim Computation Offloading wird zwischen statischer und dynamischer Entscheidungsfindung zur Ermittlung geeigneter auszulagernder Applikationspartitionen unterschieden. Bei der statischen Entscheidung wird die Applikation bereits während der Entwicklung in die jeweiligen lokalen und auszulagernden Bereiche partitioniert [12]. Die Applikation wird direkt als Computation-Offloading-System geplant. Diese Entscheidung kann deshalb im späteren Verlauf nur schwer rückgängig gemacht werden. Das hat aber den Vorteil, dass kein Mehraufwand der Applikation für Profiler während der Laufzeit bewerkstelligt werden muss [12]. Im Vorfeld müssen jedoch die Parameter, die die Trade-off-Ungleichung bestimmen, möglichst präzise prognostiziert werden. Die Parameter C, P_{lokal}, P_w, X_{lokal}, P_s, P_e und D_e können üblicherweise vorausschauend kalkuliert werden. Die Geschwindigkeit der Cloud X_{cloud} kann variieren, jedoch garantieren die meisten Cloud-Anbieter ein Minimum an Performance [12]. Die Bandbreiten B_s und B_e können nur schwer während der Entwicklung geschätzt werden. Ebenso können die zu versendenden Daten D_s stark von der jeweiligen Anforderung abhängen. Ein statisches Computation-Offloading-System kann aber mittels Wahrscheinlichkeitsrechnung oder Fuzzy-Regelungen versuchen, auch diese Parameter vorherzusagen [12].

Im Gegensatz dazu kann ein dynamisches Computation-Offloading-System auf Veränderungen in der Umgebung während der Laufzeit reagieren, z. B. bei schwankenden Netzwerkbandbreiten [12]. Ein Profiler und ein Kostenmodell entscheiden dann während der Ausführung der Applikation darüber, ob es sich lohnt, eine Berechnung auszulagern (vgl. Abbildung 1). Das hat zur Folge, dass die Parameter zur Entscheidungsfindung über den gesamten Zeitraum der Applikation überwacht werden müssen. Die Überwachung dieser Parameter ist aber relativ kostengünstig, sodass sich der Mehraufwand dafür in Grenzen hält [11]. Auf der anderen Seite sind der Profiler und das Kostenmodell im Idealfall auch im späteren Verlauf der Entwicklung oder nach der Veröffentlichung leicht erweiterbar bzw. anpassbar, um auch auf zukünftige Ansprüche schnell zu reagieren. Welche Bereiche einer Applikation potenziell ausgelagert werden, wird auch in einem dynamischen System meistens statisch identifiziert. Eine Partitionierung während der Laufzeit stellt einen zu hohen, unerwünschten Mehraufwand zur Analyse des Programms dar [12].

Forschungsarbeiten, die sich mit Systemen zum Computation Offloading beschäftigen, erlebten im Laufe der Jahre einen Umbruch von statischen hinzu dynamischen Systemen [12]. Das liegt unteranderem auch daran, dass dynamische Systeme weitaus automatisierter arbeiten und dem Entwickler damit mehr Arbeit abnehmen.

Applikationen

Es gibt eine unzählige Anzahl von Applikationen, die auf verschiedene Art und Weise für Computation Offloading in Frage kommen. Generell ziehen Applikationen mit komplexen Methoden und wenig Datenaustausch den größten Nutzen aus der Auslagerung (vgl. Trade-off). Bspw. müssen Texteditoren

nur eine überschaubare Menge von Daten D_s versenden (den Text) und können darauf z. B. eine Rechtschreibüberprüfung in der Cloud ausführen [12]. Als weiteres Beispiel für geringen Datenaustausch sei hier eine künstliche Intelligenz für Schach zu nennen. Die Entscheidung zur Auslagerung korreliert dabei zusätzlich mit der Fähigkeitsstärke des Spielers. Je besser der Spieler wird, umso mehr Aktionen müssen in der Berechnung des nächsten Spielzugs berücksichtigt werden. Damit steigt die Komplexität der Berechnung und folglich wird Computation Offloading gegen erfahrene Spieler profitabler [12].

Auf der anderen Seite transferieren Applikationen wie Videobearbeitung oder Objekterkennung große Datenmengen zur Cloud und die Einsparung von Energie hängt damit größtenteils von der Bandbreite B_s ab. Im Idealfall müssen große Datenmengen gar nicht erst übertragen werden, weil sie bereits in der Cloud vorliegen. So werden die Kosten für die Übertragung nahezu komplett gespart und Computation Offloading wird deutlich lukrativer [12].

Weitere Typen von Applikationen sind z. B. Spracherkennungs- [11], Augmented Reality- [22] und Datenanalyse-Programme für Gesundheit und Fitness [5] oder Virenscanner [11].

SYSTEME ZUM COMPUTATION OFFLOADING

Der Energieverbrauch einer Applikation hängt stark davon ab, auf welchem Smartphone mit welcher Hardware diese ausgeführt wird [15]. Eine vor kurzem erfolgte Studie zeigt auf, dass mehr als 10.000 mögliche Kombinationen bezüglich Hardware und Software für Android-Applikationen existieren [18]. Es gibt natürlich Kombinationen, die allgemein üblicher sind als andere, aber für Entwickler sind es dennoch zu viele, um diese zu berücksichtigen [15].

Es wurden über die Jahre eine Menge von Frameworks (hauptsächlich für Android-Systeme) vorgestellt, die dem Entwickler bei der Aufgabe des Computation Offloadings helfen [12]. Neuere Systeme sind z. B. *ThinkAir* [11], *MAUI* [6], *CloneCloud* [5], *Cuckoo* [10], *MOCA* [2] und *SmartDiet* [21]. Diese bieten unterschiedliche Ansätze und Lösungen zur Verbesserung des Computation Offloadings und beschäftigen sich unter anderem auch mit Thematiken wie Transparenz und Sicherheit [12].

MAUI unterstützt bspw. die Annotierung von Methoden, die potenziell für die Auslagerung in Frage kommen [6]. Allerdings berücksichtigt MAUI nicht die Möglichkeit zur Skalierbarkeit der Cloud und verschenkt damit Potenzial [6, 11].

CloneCloud hingegen migriert einen laufenden Thread im Smartphone zu einem bestimmten Zeitpunkt zu einem Applikationsklon in der Cloud [5]. Es bestimmt diesen Zeitpunkt mit Hilfe einer statischen Analyse. Dafür werden die Threads auf dem Smartphone und in der Cloud unter verschiedenen Laufzeitbedingungen ausgeführt und analysiert [5]. CloneCloud bestimmt damit, welche Threads zu welchem Zeitpunkt und unter welchen Bedingungen in die Cloud ausgelagert werden sollen. Damit limitiert sich dieser Ansatz durch die statische Analyse und berücksichtigt keine Veränderungen der Umgebung während der Laufzeit. [11]. Außerdem muss die statische Analyse für jede neue Applikationsversion neu ausgeführt werden [11].

ThinkAir bedient sich aus dem Besten der beiden oben genannten Frameworks [11]. Dabei adressiert es die fehlende Skalierbarkeit von MAUI, in dem es je nach Auslastung dynamisch eine parallel laufende Anzahl von virtuellen Maschinen eines Smartphone-Klons in der Cloud erstellt oder zerstört [11]. Im Gegensatz zu CloneCloud bietet ThinkAir die Auslagerung von Methoden anstatt von Threads und einen Profiler, der während der Ausführung Informationen zur Entscheidungsfindung liefert [11]. Im folgenden Verlauf des Kapitels soll das ThinkAir-Framework nach [11] näher erläutert werden.

ThinkAir besteht aus drei wesentlichen Komponenten: der Smartphone-Umgebung, dem Applikationsserver und den Profilern.

Smartphone-Umgebung

Die potenzielle Auslagerung von Methoden geschieht in ThinkAir über Annotierung. Jede Methode, die es wert sein könnte, ausgelagert zu werden, wird vom Entwickler als solche mit `@Remote` markiert. Der Aufruf dieser Methode erfolgt dann automatisch im `ExecutionController`, der mittels verschiedener Kriterien darüber entscheidet, ob die Methode lokal oder in der Cloud ausgeführt werden soll. Die Entscheidung ist abhängig von den gesammelten Umgebungsdaten der Profiler sowie vorangegangenen Entscheidungen und Informationen zur Ausführung dieser Methode. Gibt es keine vergangene Aufrufe dieser Methode basiert die Entscheidung des `ExecutionControllers` rein auf den Umgebungsparametern der Profiler. Bspw. wird bei einer signalstarken WiFi-Verbindung die Methode aller Voraussicht nach ausgelagert. Eine signalschwache Verbindung spricht eher für eine lokale Ausführung. Für den Aufruf in der Cloud müssen alle Parameter und Instanzobjekte der Klasse mitgeschickt werden, die für eine erfolgreiche Ausführung der Methode nötig sind. Beim Aufruf sammeln die Profiler dann bspw. Daten über Ausführungszeit und Energieverbrauch der Methode und trainieren den `ExecutionController` zu besseren Entscheidungen. Schlägt eine Verbindung zur Cloud fehl, wird die Methode alternativ lokal ausgeführt. Damit bleibt die Funktionsfähigkeit der Applikation auch bei Verbindungsabbrüchen garantiert.

Applikationsserver

Der Applikationsserver kümmert sich auf Seiten der Cloud um den auszulagernden Quelltext und dessen Ausführung. Dafür verbindet sich der `ExecutionController` beim Start der Anwendung mit einem `ClientHandler` in der Cloud. Ist die Applikationssoftware der Cloud noch unbekannt lädt der `ClientHandler` den Quelltext vom Smartphone und kümmert sich um die Einbindung aller nötigen nativen Bibliotheken. Ebenso antwortet der `ClientHandler` auf Ping-Nachrichten für die Bestimmung der aktiven Verbindungslatenz.

Wirft der Applikationsserver während der Ausführung einer Methode eine `OutOfMemory`-Exception, kümmert sich dieser selbstständig um die Umschichtung der Aufgabe auf ei-

ne leistungsstärkere virtuelle Maschine und garantiert damit auch bei vielen Anfragen eine zeitgemäße Antwort.

Der Applikationsserver nutzt eine Virtualisierungsplattform und kann damit auf privaten sowie kommerziellen Clouds zum Einsatz kommen. Als Plattformen sind z. B. Xen, QEMU und VirtualBox von Oracle zu nennen. Virtualisierung in der Cloud hat gegenüber Smartphones den enormen Vorteil, dass Multitasking weitaus effizienter über Multiprozessoren und der Erweiterung zu mehreren virtuellen Maschinen genutzt werden kann.

Profiler

Das Profiler-System ist modular gestaltet, um die einfache Einbindung von neuen Profilern zu ermöglichen. Aktuell besitzt ThinkAir vier Profiler (Hardware, Software, Netzwerk, Energie), die sich um das Sammeln von verschiedenen Daten kümmern. Insbesondere werden `Listener` registriert, um Informationen über die Akkulaufzeit, die Verbindungsstärke und deren Technologie (Wifi, 3G, ...) zu erhalten. Dadurch werden keine zusätzlichen Ressourcen für die Ermittlung dieser Daten benötigt. Die vier Profiler ermitteln unter anderem folgende Daten:

- **Hardware:** CPU-Auslastung, Display-Helligkeit, Wifi, 3G
- **Software:** Ausführungszeit und Komplexität von Methoden, Anzahl von Methodenaufrufen, ... (Ermittlung über Android Debug API)
- **Netzwerk:** Round Trip Time und Datenrate (Abschätzung der vorliegenden Bandbreite), Anzahl gesendeter und empfangener Pakete pro Sekunde
- **Energie:** Energieverbrauch von Methodenaufrufen

Um den Energieverbrauch einzelner Methoden dynamisch zur Laufzeit zu ermitteln, macht sich ThinkAir das *PowerTutor*-Modell zu Nutze [23]. PowerTutor ist ein Energiemodell zur Abschätzung der Energiekosten von CPU, LCD, GPS, WiFi und 3G der Smartphones HTC Dream und HTC Magic. ThinkAir modifiziert dieses Modell leicht und berücksichtigt dabei, dass Komponenten wie Audio oder GPS nicht ausgelagert werden können. Mit einer getesteten maximalen Fehlerrate von 6,27% wurde dieses Energiemodell auf eine Vielzahl anderer Smartphones angewendet [11]. ThinkAir toleriert diese Fehlerrate und kann somit den ungefähren Energieverbrauch einzelner Methoden vorhersagen.

TESTERGEBNISSE

Der Gewinn an Energieeinsparung von Computation Offloading und die Auswertung von Systemen zum Computation Offloading ist Bestandteil vieler Forschungsarbeiten. Bspw. beschäftigt sich die Studie von Barbera et al. mit der Netzwerkverfügbarkeit und dem mobilen Datenverkehr von Smartphones in einem Testumfeld von 11 Endnutzern [3]. Die Studie will damit die Realisierbarkeit von Computation Offloading im Alltag überprüfen. Insbesondere legt sie die durchschnittliche tägliche Dauer dar, in der Benutzer über der Woche entweder mit 3G/2G oder WiFi zum Internet verbunden sind (vgl. Abbildung 3). Es zeigt sich, dass

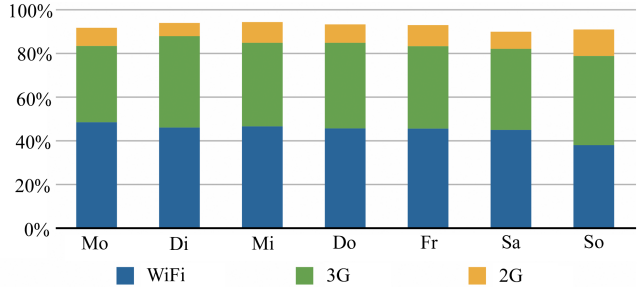

Figure 3. Durchschnittliche tägliche Konnektivität für unterschiedliche Übertragungstechnologien [3]

die tägliche Konnektivität über der ganzen Woche mit einer Übertragungstechnologie mit 90% sehr hoch ist [3]. Die Zeiten der Verbindungsabbrüche erklären sich durch signallose Gebiete (z. B. U-Bahnen) oder durch das Ausschalten des Smartphones [3]. Die Dauer der Konnektivität der unterschiedlichen Technologien (WiFi, 3G, 2G) ist ebenfalls recht stabil. WiFi erreicht eine durchschnittliche Konnektivität von 45%, 3G von 40% [3]. Damit sind Benutzer rund 85% ihres Tages mit einer Technologie verbunden, die hohe Bandbreiten erlaubt.

Die Studien von Lagerspetz et al. und Miettinen et al. analysieren die Energieeinsparung durch Computation Offloading anhand der Trade-off-Ungleichung in Kapitel 2 am Beispiel eines Nokia N900[13, 16] . Dabei vernachlässigen sie die entstehenden Energiekosten des mobilen Geräts beim Warten auf die Beendigung der Berechnung der ausgelagerten Funktion. Die Arbeiten nehmen für ihre Testzwecke an, dass die entsprechende Cloud-Plattform die Aufgabe schnell lösen kann und das Handy wenig Energie beim Warten verbraucht [13]. Dann gilt die folgende vereinfachte Trade-Off-Ungleichung [13]:

$$C\frac{P_{lokal}}{X_{lokal}} > P_s\frac{D_s}{B_s} + P_e\frac{D_e}{B_e}$$

Nach [13] läuft das Nokia N900 mit einer Geschwindigkeit X_{lokal} von 600MHz während einer Berechnung und verbraucht 0,9W Leistung P_{lokal}. Miettinen et al. verifizieren diese Ergebnisse [16] . Damit ergibt sich nach X_{lokal}/P_{lokal} eine ungefähre Anzahl von 650 Millionen Befehlen pro Joule. Nach einer Studie der Universität von Helsinki [13] sendet und empfängt das Nokia N900 im WiFi bei einer Geschwindigkeit von 700kB/s ungefähr $B_s/P_s = 600$kB pro Joule. Bei einer gedrosselten Verbindung von 100kB/s beträgt die Energieeffizienz 110kB/J. Ebenso wurde die Energieeffizienz beim Senden und Empfangen im 3G-Netzwerk an unterschiedlichen Standpunkten gemessen (nah und weit entfernt von der Basisstation) [16]. Es zeigt sich, das die Energiekosten zum Übertragen weitaus höher sind als zum Empfangen und die Entfernung zur Basisstation einen signifikanten Unterschied im Bezug auf den Energieverbrauch ausmacht [16]. Unter besten Umständen müssen für eine nutzbringende Auslagerung ungefähr eine Millionen Befehle in der Berechnung ausgeführt werden, wenn ein Megabyte an Daten übertragen wird [16]. Dieses Ergebnis deckt sich mit

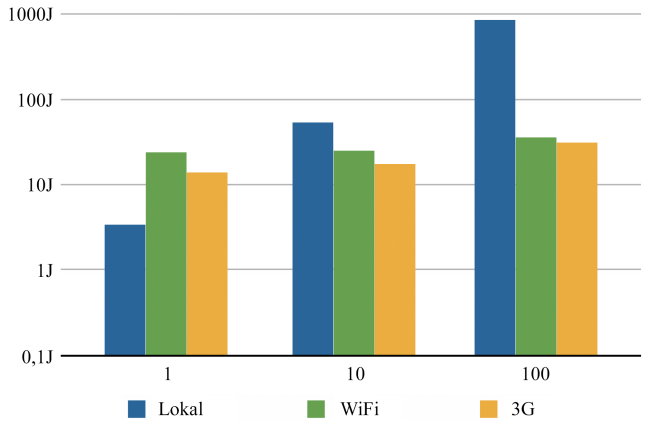

Figure 4. Energieverbrauch bei der Benutzung der unterschiedlichen Übertragungstechnologien für die Gesichtserkennungs-Applikation nach [11]

dem Ergebnis aus [13]. Das führt nach [13] zu folgenden Entscheidungshilfen:

1. Aufgaben mit keiner oder wenig Datenübertragung sollten immer ausgelagert werden.

2. Aufgaben mit aufwändigen Berechnungen (mehr als eine Millionen Befehle) können ausgelagert werden.

3. Aufgaben, die eine größere, sekundäre Datenübertragung als die Übertragung zur Auslagerung benötigen, sollten ausgelagert werden.

Für ThinkAir wird eine Evaluierung anhand von mehreren Szenarien unter drei verschiedenen Konnektivitäten (Lokal, WiFi und 3G) präsentiert [11]. Das 3G-Netzwerk besitzt dabei eine durchschnittliche Round Trip Time (RTT) von 100ms. Für das WiFi existieren zwei Testumgebungen, die in der Evaluierung als Durchschnitt berücksichtigt werden: ein WiFi-Router in einer gewöhnlichen Wohnung (50ms RTT) und ein öffentlicher Wireless Access Point für eine Vielzahl von Benutzern (200ms RTT) [11]. Die Studie testet das Framework anhand von vier Applikationsbeispielen, von denen zwei in dieser Arbeit näher vorgestellt werden sollen: eine Gesichtserkennung und ein Virenscanner [11].

Gesichtserkennung

Die Applikation zählt die Anzahl der Gesichter in einem Bild mit Hilfe der Android API `FaceDetector`. Damit ist die Applikation für Android optimiert und bietet eine relativ schnelle lokale Ausführung [11]. Der Test wird mit einer variablen Anzahl von zu überprüfenden Fotos durchgeführt (1, 10 und 100). Die Fotos liegen auf dem Smartphone sowie auf der Cloud vor. Abbildung 4 präsentiert die Ergebnisse der Gesichtserkennung auf einer logarithmischen Skala. Die Detektion eines einzelnen Bilds läuft auf dem Smartphone schneller und verbraucht dabei weniger Energie, als wenn die Aufgabe erst über WiFi oder 3G an die Cloud delegiert werden muss. Steigt die Anzahl der zu analysierenden Fotos, so zeigt sich die Cloud deutlich leistungsstärker und energieeffizienter. Die Bearbeitung von 100 Fotos verbraucht in der Cloud nur einen Bruchteil der ungefähr 1000J, die lokal zur Berechnung benötigt würden [11]. Der Unterschied des Energieverbrauchs zwischen WiFi und 3G ist zu vernachlässigen, da keine Daten an die Cloud übermittelt werden müssen.

Virenscanner

Der Virenscanner ermittelt über einer lokalen bzw. in der Cloud liegenden Datenbank mit 1000 enthaltenden Virussignaturen in einem Dateisystem die Anzahl der gefundenen Viren. Das Testszenario umfasst 3500 Dateien mit einer Gesamtgröße von 10MB [11]. Im Vergleich zur Gesichtserkennung müssen in diesem Fall große Datenmengen in die Cloud ausgelagert werden. Die Ausführung auf dem Smartphone beträgt für den Scanvorgang über eine Stunde, wohingegen die Auslagerung und Berechnung in der Cloud nur drei Minuten in Anspruch nimmt. Die Energieersparnis ist ebenso deutlich. Verbraucht das Smartphone bei lokaler Berechnung fast 3000 Joule, so werden für die Auslagerung nicht mehr als 70 Joule benötigt [11]. Der Unterschied der Energiekosten bei der Übertragung zwischen WiFi und 3G ist erneut marginal. Es zeigt sich aber im Gegensatz zu den Studien zum Nokia N900 (vgl. [13, 16]), dass die Benutzung von WiFi knapp weniger energieeffizient ist als von 3G [11]. Ähnliche Ergebnisse zeigte auch die Gesichtserkennung.

DISKUSSION

Die Testergebnisse des vorangegangenen Kapitels zeigen, dass Computation Offloading bei bestimmten Anwendungsszenarien nicht nur die Möglichkeit der Leistungssteigerung, sondern auch eine Minimierung des Energieverbrauchs von Smartphones bietet. Für das Computation Offloading ist es wünschenswert, dass das mobile Gerät jederzeit Daten zur Berechnung an die Cloud senden kann. Andererseits bremsen langsame oder nicht vorhandene Verbindungen die Akzeptanz und Benutzerfreundlichkeit des Systems [3]. Systeme zum Computation Offloading benötigen sorgsame Planung, um sichtbar lange Latenzzeiten für den Benutzer zu vermeiden [16]. Allerdings schmälern eine durchschnittliche verbindungsstarke Konnektivität von 85% und die bereits gute dynamische Entscheidungsfindung der gegenwärtigen Offloading-Frameworks diesen Kritikpunkt [3]. Computation Offloading ist in der heutigen Zeit keine utopische Vorstellung mehr. Die verbesserte und vermehrte Breitbandabdeckung mit Hilfe von 3G, 4G, Femtozellen (Funkzelle zur Erweiterung eines Mobilfunkanbieternetzes) und fester Drahtloskommunikation fördern die Bedingungen für Offloading [8].

Computation Offloading kann weiterhin zum *Mobile Cloud Computing* erweitert werden. Mobile Cloud Computing bietet zusätzliche Leistungen zum bloßen, effizienteren Berechnen einer Aufgabe. Z. B. können Benutzerdaten in der Cloud gespeichert werden oder Inhalte über mehrere Benutzer geteilt werden [16]. Damit sind für viele Anforderungen die entsprechenden Daten bereits in der Cloud gespeichert und der Netzwerktransfer beim Computation Offloading reduziert sich auf ein Minimum [16]. Bereits heute lagern viele Benutzer ihre Daten an die Cloud aus. Damit wird ihnen eine Sicherheit ihrer Daten garantiert und auf verschiedenen Geräten parallel zugänglich gemacht.

Auf der anderen Seite ergeben sich neben möglichen langen Latenzzeiten und der dadurch entstehenden Minderung der Benutzerfreundlichkeit noch weitere Nachteile für Benutzer und Entwickler, die unbedingt beachtet werden sollten.

Benutzer von Computation-Offloading-Systemen sollten darüber im Klaren sein, dass ihr Smartphone während der Ausführung der Applikation im ständigen Kontakt zum Internet steht. Anstehende Traffic-Kosten in Zeiten von Internetflatrates mit Datenlimit können zum Ärgernis des Benutzers führen. Ebenso ist die Privatsphäre und Sicherheit von sensiblen Daten ein kritischer Punkt bei der Abwägung zum Offloading. Private Daten werden möglicherweise an Server gesendet, über die der Benutzer keine Kontrolle hat [12]. Der Benutzer sollte entscheiden können, ob er die Auslagerung der Berechnung auf diesen Daten befürwortet oder lokal auf dem Smartphone laufen lassen will. Der Sicherheitsaspekt ist bei der heutigen Angst der ständigen Überwachung ebenfalls ein wichtiges Thema [12]. Ein Dritter kann möglicherweise Daten bei der Übertragung an die Cloud abfangen. Mit Hilfe von Kryptografie und dem Aufbau von sicheren Verbindungen kann diesem zwar entgegengewirkt werden, mindert aber gleichzeitig den Gewinn an Energie durch die Auslagerung. Eine Ansammlung von Studien und Frameworks haben sich in der Vergangenheit bereits dem Thema Sicherheit beim Computation Offloading gewidmet (vgl. [12]).

Der Entwickler muss durch die Benutzung von Computation Offloading zusätzlichen Aufwand investieren. Es scheint schwierig, Entwickler von der Idee des Computation Offloadings zu überzeugen, nur weil Benutzer nicht gerne ihr Smartphone laden. So sollte neben der Einsparung von Energie ebenso der Gewinn an Performance beim Computation Offloading im Vordergrund stehen.

In der Arbeit zu ThinkAir wird auch auf mögliche Schwächen des Frameworks hingewiesen [11]. Bspw. werden bei der Datenübermittlung möglicherweise Instanzobjekte mitgesendet, die für die Berechnung der Methode nicht benutzt werden. Statische Code-Analysen und das Caching von Objekten in der Cloud sind zwei der vorgeschlagenen Verbesserungsmöglichkeiten [11]. So werden Daten, die nicht benutzt werden, und unveränderte Objekte, die bereits in der Cloud liegen, gar nicht erst übermittelt. Allerdings produziert das Caching von Daten und die Überprüfung auf Veränderungen einen gewissen Mehraufwand, der die Trade-off-Ungleichung verschiebt [11]. Ebenso wird auch auf die Sicherheits-Problematik im ThinkAir-Framework hingewiesen. Ein Vorschlag zur Verbesserung ist ein Authentifizierungsmechanismus für eine Verbindung über einen Shared-Secret-Service [11]. Private Daten, wie der Standort oder die Profildaten des Benutzers, sollen in einer zukünftigen ThinkAir-Version über die `SecureRemotable`-Klasse vor dem Zugang durch Dritte geschützt werden. Gleichzeitig wird dem Entwickler damit die Last abgenommen, sich um diese Angelegenheiten selbst zu kümmern [11].

AUSBLICK

Die „International Data Corporation“ sagt voraus, dass 70% der Unternehmen in der Informationstechnikbranche auf eine „Cloud first“-Strategie im Jahr 2016 setzen werden [8]. Damit rückt auch das Computation Offloading immer mehr in die Köpfe der Entwickler und Anwender. Fortschritt in der Technologie kann und wird das Computation Offloading nur allgegenwärtiger machen, denn die Trade-off-Beziehung wird damit immer weiter in Richtung Offloading verschoben [16]. Die mobile Breitbandabdeckung wächst und wird von Jahr zu Jahr schneller [8, 11]. Damit können relativ geringe Round Trip Times und hohe Bandbreiten für die Zukunft prognostiziert werden [11].

Dagegen benötigt die Aufrechterhaltung einer konsistenten drahtlosen Verbindungen ohne Verbindungsabbrüche bei sich bewegenden Nutzern weitere Forschung und Entwicklung [1]. Ein weiteres spannendes Forschungsgebiet ist das Umverteilen von laufenden virtuellen Maschinen ohne Verbindungsabbrüche auf Server, die näher am Benutzer sind, um lange Latenzzeiten zu vermeiden [1].

Ebenso ist das Verlangen nach komplexeren Energie-Profilern auf dem Markt enorm gestiegen [16]. Frameworks wie ThinkAir müssen sich auch heute noch mit Schätzungen vom Energieverbrauch einzelner Methoden zufriedengeben und implizieren somit eine gewisse Fehlertoleranz bei der Entscheidung zur Auslagerung [11].

Die zur Verfügung stehenden Ressourcen in der Cloud sind nahezu grenzenlos und bieten Spielraum und Möglichkeiten für gänzlich neue und innovative Applikationen [16]. Es wird spannend sein, die Entwicklung von Computation Offloading und der Cloud weiterhin zu beobachten.

ALTERNATIVE ANSÄTZE

Neben dem Computation Offloading gibt es zahlreiche weitere Forschungsarbeiten zur Erstellung von energieeffizienten Applikationen. Auf der ICSE 2014 wurden zwei weitere Ansätze, *SEEDS* und *Nyx*, vorgestellt, die Entwicklern automatisiert dabei helfen, Energie in ihren Anwendungen einzusparen.

SEEDS ist ein Framework für die Einsparung von Energie in Java-Applikationen, in dem automatisiert Änderungen auf Quelltext-Ebene vorgenommen werden [15]. Damit erlöst es den Entwickler von der Aufgabe, den Quelltext selbstständig auf Energieeffizienz zu optimieren. SEEDS wählt dafür aus den energieeffizientesten Implementierungen der Java Collection API und wechselt sie mit den bisherigen Implementierungen aus [15]. Ein Benchmark-Test zeigt, dass der Energieverbrauch bei 7 der 13 Implementierungen des `Collection`-Interfaces durch einen Austausch der Implementierung verbessert werden kann [15]. Applikationen, die durch SEEDS optimiert werden, zeigen eine Energieeinsparung von bis zu 17% [15].

Der Bildschirm ist mit eine der am meisten Energie verbrauchenden Komponenten eines Smartphones [4]. Nyx beschreibt eine Technik, welche Webapplikationen energieeffizienter für OLED-Bildschirme gestaltet [14]. OLED-Bildschirme verbrauchen mehr Energie für die Anzeige von hellen als von dunklen Farben [14]. Auf Basis dieser Kenntnis generiert Nyx ein neues, dunkleres Farbschema einer Webseite und versucht gleichzeitig Farbzusam-

menhänge und Ästhetik der Seite zu erhalten [14]. Die Evaluierung dieser Technik zeigt eine Reduzierung der Bildschirm-Energiekosten um 40%. Zusätzlich gaben über 60% der Befragten an, dass die transformierte Web-Applikation für den täglichen Gebrauch akzeptabel ist. 97% würden die Transformierung bevorzugen, wenn sich die Akkulaufzeit des Smartphones dem Ende neigt [14].

Die vielen aktuellen Forschungsarbeiten in den unterschiedlichsten Bereichen zum Thema „Apps and Energy“ beweisen das Bedürfnis nach energieeffizienteren Applikationen. Obwohl bereits viel Arbeit in diesem Gebiet steckt, ist die Forschung darin durch die anhaltende Popularität von Smartphones und dem Wunsch nach komplexeren Applikationen nicht aufzuhalten.

REFERENCES

1. S. Abolfazli, Z. Sanaei, M. H. Sanaei, M. Shojafar, and A. Gani. 2014. Mobile Cloud Computing: The-state-of-the-art, challenges, and future research. In *Encyclopedia of Cloud Computing*. Wiley.
2. A. Banerjee, X. Chen, J. Erman, V. Gopalakrishnan, S. Lee, and J. Van Der Merwe. 2013. MOCA: A lightweight Mobile Cloud Offloading Architecture. In *8th ACM International Workshop on Mobility in the Evolving Internet Architecture (MobiArch '13)*. 11–16.
3. M. V. Barbera, S. Kosta, A. Mei, and J. Stefa. 2013. To offload or not to offload? The bandwidth and energy costs of Mobile Cloud Computing. In *Proceedings IEEE INFOCOM 2013*. 1285–1293.
4. A. Carroll and G. Heiser. 2010. An analysis of power consumption in a smartphone. In *USENIX Conference on USENIX Annual Technical Conference (USENIXATC'10)*. 21–21.
5. B. G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. 2011. CloneCloud: Elastic execution between Mobile Device and Cloud. In *6th Conference on Computer Systems (EuroSys '11)*. 301–314.
6. E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. 2010. MAUI: Making smartphones last longer with Code Offload. In *8th International Conference on Mobile Systems, Applications, and Services (MobiSys '10)*. 49–62.
7. E. Daley. 2011. The mobile „App Internet“ recasts the software & services landscape. (2011). http://de.slideshare.net/nasscom-emerge/forrester-8840580
8. S. Hossain. 2013. What is Mobile Cloud Computing? (2013). http://archive.thoughtsoncloud.com/2013/06/mobile-cloud-computing/
9. CNN International. 2013. Battery life concerns mobile users. (2013). http://edition.cnn.com/2005/TECH/ptech/09/22/phone.study/
10. R. Kemp, N. Palmer, T. Kielmann, and H. Bal. 2012. Cuckoo: A Computation Offloading framework for smartphones. In *Mobile Computing, Applications, and Services*. Springer Berlin Heidelberg, 59–79.
11. S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang. 2012. ThinkAir: Dynamic resource allocation and parallel execution in the Cloud for Mobile Code Offloading. In *Proceedings IEEE INFOCOM 2012*. 945–953.
12. K. Kumar, J. Liu, Y. H. Lu, and B. Bhargava. 2013. A survey of Computation Offloading for Mobile Systems. *Mobile Networks and Applications* 18, 1 (2013), 129–140.
13. E. Lagerspetz and S. Tarkoma. 2011. Mobile search and the Cloud: The benefits of Offloading. In *IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*. 117–122.
14. D. Li, A. H. Tran, and W. G. J. Halfond. 2014. Making web applications more energy efficient for OLED smartphones. In *36th International Conference on Software Engineering (ICSE 2014)*. 527–538.
15. I. Manotas, L. Pollock, and J. Clause. 2014. SEEDS: A software engineer's energy-optimization Decision Support Framework. In *36th International Conference on Software Engineering (ICSE 2014)*. 503–514.
16. A. P. Miettinen and J. K. Nurminen. 2010. Energy efficiency of mobile clients in Cloud Computing. In *2nd USENIX Conference on Hot Topics in Cloud Computing (HotCloud'10)*. 4–4.
17. Y. Neuvo. 2004. Cellular phones as embedded systems. In *IEEE International Solid-State Circuits Conference*. 32–37.
18. OpenSignal. 2013. Android fragmentation visualized. (2013). http://opensignal.com/reports/fragmentation-2013/
19. A. Pathak, Y. C. Hu, and M. Zhang. 2012. Where is the energy spent inside my app?: Fine grained energy accounting on smartphones with Eprof. In *7th ACM European Conference on Computer Systems (EuroSys '12)*. 29–42.
20. S. Robinson. 2009. *Cellphone energy gap: Desperately seeking solutions*. Technical Report. Strategy Analytics.
21. A. Saarinen, M. Siekkinen, Y. Xiao, J. K. Nurminen, M. Kemppainen, and P. Hui. 2012. SmartDiet: Offloading popular apps to save energy. *SIGCOMM Comput. Commun. Rev.* 42, 4 (2012), 297–298.
22. H. Wu, Q. Wang, and K. Wolter. 2013. Tradeoff between performance improvement and energy saving in Mobile Cloud Offloading Systems. In *IEEE International Conference on Communications Workshops (ICC)*. 728–732.
23. L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R.P. Dick, Z. M. Mao, and L. Yang. 2010. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *8th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES/ISSS '10)*. 105–114.