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. Aber so sehr der App-Markt explodiert und immer komplexer werdende Programme wie Objekterkennung oder Videobearbeitung auf Smartphones zur Verfügung stehen, so unglücklicher werden Benutzer mit der Akkulaufzeit dieser. Energieeffizienz ist daher ein wichtiges Thema bei der Entwicklung von benutzerfreundlichen mobilen Applikationen. Computation Offloading, das heißt 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 dessen in der Forschung auf. Neben der Grundidee des Computation Offloadings und der Vorstellung bisheriger Arbeiten auf diesem Gebiet, beschäftigt sich diese Arbeit vor allem 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 im Internet, checken unsere Emails, gucken Videos und betreiben Social Media auf unseren mobilen Endgeräten. Dabei können wir dank zusätzlichen Komponenten und Sensoren am Smartphone wie Kamera oder GPS sogar weitaus mehr bewerkstelligen als am Computer und sind dabei noch mobil. Das App-Geschäft explodiert. So finden täglich neue innovative und revolutionäre Applikationen ihren Weg in die gegenwärtigen Smartphone Stores. Laut Forschung wächst die App-Industrie im Jahr 2015 exponentiell auf einen Marktwert von 37 Milliarden an $ [7].

Das Smartphone ist ebenso alltäglich geworden. Haben wir es nicht bei uns oder neigt sich der Akku dem Ende, fühlen wir uns unverzüglich eingeschränkt in unseren Fähigkeiten. 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 dieser. 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 [19]. Größere Smartphones mit stärkeren Akkus sind jedoch keine attraktive Alternative. Ebenso schmälern thermische Anforderungen die Leistung von Smartphones auf ungefähr 3 Watt [16].

Energieeffizienz ist daher ein wichtiges aber ebenso auch unterschätztes Thema bei der Entwicklung von benutzerfreundlichen mobilen Applikationen. Optimales Energiemanagement erfordert ein gutes Verständnis über die Energienutzung von Smartphones [4]. Es ist folglich nicht verwerflich, dass Entwickler sich bisher nur wenige Gedanken um Energieeffizienz in ihren Applikationen gemacht haben [18]. So verbrauchen 6 der 10 beliebtesten Apps im Google Play Store über einer Laufzeit von 30 Sekunden bis zu 0,75% einer vollen Akkuladung [18].

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 (zum Beispiel 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 beispielsweise durch Computation Offloading energieeffizienter gestaltet werden?

Daraufhin werden bisherige Arbeiten in diesem Forschungsgebiet vorgestellt. Es wurden in der Vergangenheit bereits eine Vielzahl von Frameworks entwickelt, die CPU intensive Aufgaben automatisiert auslagern [11, 6, 5, 10, 2, 20].

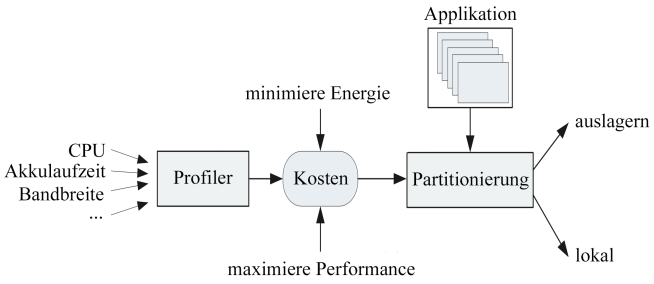

Figure 1. Architektur von Computation Offloading Systemen nach [21]

Mit besonderem Augenmerk wird hier das Framework ThinkAir [11] präsentiert, welches sich die Frameworks MAUI [6] und CloneCloud [5] zu Nutzen macht.

Im nachfolgenden Abschnitt werden Ergebnisse aus verschiedenen Studien mit unterschiedlichen Testszenarien und Übertragungstechnologien präsentiert [3, 13, 15, 11]. In einer 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.

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 (zum Beispiel die Cloud) über Netzwerkübertragung. Dafür werden die entsprechenden 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 berechneten 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.

Die typische Architektur von Computation Offloading Systemen, bestehend aus drei Hauptkomponenten (Profiler, Kostenmodell und Partitionierung), zeigt Abbildung 1.

Profiler sammeln wertvolle Informationen über die CPU-Auslastung, Akkulaufzeit und Bandbreite des Smartphones und dessen Netzwerkumgebung [21]. 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 bzw. 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 [21]. 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 [21].

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 [15]. Ebenso bietet sie den Vorteil des Multitaskings und kann damit mehrere Applikationen und Prozesse simultan laufen lassen [12]. Die Popularität der Cloud liegt unter anderem auch daran, dass sie relativ kostengünstig ist [21].

Trade-off

Es ist wichtig anzumerken, dass Computation Offloading eine opportunistische Möglichkeit zur Erstellung von energieeffizienten Applikationen ist, keinesfalls eine Pflicht für zukünftige Programme darstellt. Das liegt insbesondere an den zusätzlichen Energie- und Netzwerkkosten für die Datenübermittlung, die natürlich bei der Entscheidung für die Auslagerung bedacht werden müssen. [21, 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 überhaupt sinnvoll ist. Dieser Trade-off lässt sich mathematisch wie folgt definieren [13]:

$$E_{trade} = E_{lokal} - E_{offload} > 0$$

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

Da sich Energie über die durchschnittliche Leistung über einen bestimmten Zeitraum bestimmt ($E = P \times T$), lässt sich E_{lokal} weiter auflösen. Sei C dafür die Komplexität der Aufgabe (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 die Cloud E_{senden}, zum Warten auf die Beendigung der Berechnung in der Cloud E_{warten} und zum Empfangen des Ergebnisses $E_{empfangen}$ zusammen [13]:

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

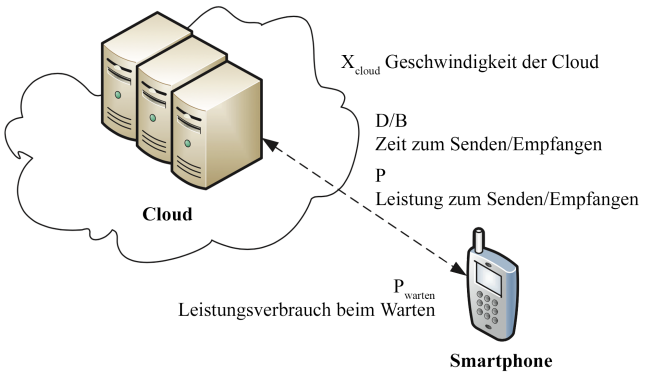

Figure 2. Energiekosten eines Offloading-Prozesses nach [21]

Diese Energien lassen sich erneut mit Hilfe der Beziehung zwischen Leistung und Zeit zum Energieverbrauch 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 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 die aktuellen Bandbreiten B_s, B_e berechnen [13]. Die Netzwerklatenzen seien hier mit in der Bandbreite berücksichtigt. Es ergibt sich damit:

$$E_{trade} = \frac{P_{lokal} \times C}{X_{lokal}} - \frac{P_w \times C}{X_{cloud}} - \frac{P_s \times D_s}{B_s} - \frac{P_e \times D_e}{B_e} > 0$$

$$\Leftrightarrow \quad E_{trade} = C\left(\frac{P_{lokal}}{X_{lokal}} - \frac{P_w}{X_{cloud}}\right) - P_s\frac{D_s}{B_s} - P_e\frac{D_e}{B_e} > 0$$

$P_{lokal}/X_{lokal} - P_w/X_{cloud}$, 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

Die Wahlmöglichkeit, welche Bereiche einer Applikationen geeignete Kandidaten für die Auslagerung darstellen, kann entweder statisch oder dynamisch von Statten gehen. 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 und kann deshalb im späteren Verlauf nur schwer rückgängig gemacht werden. Das hat wiederum 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, zum Beispiel 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 den Nachteil, dass die Parameter zur Entscheidungsfindung über den gesamten Zeitraum der Applikation überwacht werden müssen. Die Überwachung dieser Parameter ist aber mittlerweile relativ kostengünstig, so dass sich der Mehraufwand dafür in Grenzen hält [11]. Auf der anderen Seite sind der Profiler und das Kostenmodell 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 Computation Offloading Systemen 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 eine Menge Arbeit abnehmen.

Applikationen

Es gibt eine unzählige Anzahl von Applikationen, die in verschiedener Art und Weise für Computation Offloading in Frage kommen. Generell ziehen Applikationen mit komplexen Methoden und wenig Datenaustausch den größten Nutzen aus der Auslagerung. Als Beispiel dafür sind zum Beispiel Texteditoren zu nennen, die eine geringe Menge von Daten D_s (den Text) verschicken und darauf beispielsweise eine Rechtschreibüberprüfung in der Cloud ausführen [12]. Als weiteres Beispiel für geringen Datenaustausch sei hier eine Schach-KI zu nennen. Ihre Entscheidung zur Auslagerung hängt zusätzlich von der Fähigkeitsstärke des Spielers zusammen. Um einen guten Spieler zu schlagen, muss die künstliche Intelligenz eine Mehrzahl an Berechnungen (steigende Komplexität) durchführen und folglich wird Computation Offloading gegen erfahrene Spieler profitabler [12].

Applikationen wie Videobearbeitung oder Objekterkennung transferieren große Datenmengen zur Cloud und die Einsparung von Energie hängt damit größtenteils von der Bandbreite B_s ab. Bezieht man sich auf Videos oder Bilder im Internet (zum Beispiel YouTube), so ist Computation Offloading natürlich effizienter [12].

Weitere Typen von Applikationen sind zum Beispiel Spracherkennungs- [11], Augmented Reality- [21] oder Datenanalyse-Programme für Gesundheit und Fitness [5] sowie Virusscanner [11].

BISHERIGE ARBEITEN

Der Energieverbrauch einer Applikation hängt stark davon ab, auf welchem System und auf welcher Hardware diese ausgeführt wird [14]. Eine vor kurzem erfolgte Studie zeigt auf, dass mehr als 10.000 mögliche Kombinationen bezüglich Hardware und Software für Android-Applikationen existieren [17]. Es gibt natürlich Kombinationen, die allgemein üblicher sind als andere, aber für Entwickler sind es dennoch zu viele, um diese zu berücksichtigen [14]. Es bietet sich demnach an, Systeme bzw. Frameworks zu nutzen, die einem diesen Entscheidungsprozess abnehmen.

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 zum Beispiel *ThinkAir* [11], *MAUI* [6], *CloneCloud* [5], *Cuckoo* [10], *MOCA* [2] und *SmartDiet* [20]. Diese bieten unterschiedliche Ansätze und Lösungen zur Verbesserung des Computation Offloadings und adressieren unter anderem auch Thematiken wie Transparenz und Sicherheit für den Benutzer [12].

MAUI unterstützt beispielsweise wie viele andere existierende Frameworks die Partitionierung der Applikationen über die Annotierung von Methoden [6]. Methoden, die potenziell ausgelagert werden sollen, werden somit als solche gekennzeichnet. 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 hinzuzu einem Applikationsklon in der Cloud [5]. Es bestimmt diesen Zeitpunkt mit Hilfe einer statischen Analyse. Die Threads werden 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]. ThinkAir ist damit ein dynamisches System und kann folglich schnell auf Veränderungen der Umgebung reagieren. Mittels eines einfachen Interfaces ist es für Entwickler einfach zu benutzen und mindert damit gleichzeitig die Gefahr, dass Framework sinnwidrig zu benutzen. 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

ThinkAir bietet eine einfach zu benutzende Bibliothek für die potenzielle Auslagerung von Methoden für Android-Applikationen. Jede Methode, die es wert sein könnte, ausgelagert zu werden, wird vom Entwickler als solche mit `@Remote` annotiert. 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 dabei abhängig von den gesammelten Umgebungsdaten der Profiler sowie von vergangenen Entscheidungen und Informationen zur Ausführung dieser Methode. Gibt es noch keinen vorangegangen Aufruf dieser Methode, basiert die Entscheidung des `ExecutionControllers` nur auf den Umgebungsparametern der Profiler. Gibt es beispielsweise eine signalstarke WiFi-Verbindung, so wird die Methode aller Voraussicht nach ausgelagert. Auf einer signalschwachen Verbindung wird die Methode eher lokal ausgeführt. Beim Aufruf sammeln die Profiler dann Daten über Ausführungszeit und Energieverbrauch und trainieren damit den `ExecutionController` auf bessere Entscheidungsfindung. Wird eine Methode ausgelagert und die Verbindung schlägt aus unbekannten Gründen fehl, wird die Methode lokal ausgeführt. Damit ist sichergestellt, dass die Benutzung der Applikation auch bei Verbindungsabbrüchen weiter funktioniert.

Applikationsserver

Der Applikationsserver kümmert sich auf Seiten der Cloud um den auszulagernden Quelltext und dessen Ausführung. Ein `ClientHandler` registriert jede neue Client-Verbindung sobald sich ein neuer ThinkAir-`ExecutionController` mit der Cloud verbindet. Ist die Applikationssoftware der Cloud unbekannt, lädt der `ClientHandler` den Quelltext vom Smartphone und kümmert sich automatisch um die Einbindung aller nötigen nativen Bibliotheken. Ebenso antwortet der `ClientHandler` auf alle Ping-Nachrichten für die Bestimmung der aktiven Verbindungslatenz.

Wirft der Applikationsserver beim Berechnen eine `OutOfMemory`-Exception, wird die Methode nicht lokal auf dem Smartphone neugestartet. Stattdessen kümmert sich der Applikationsserver selbstständig um die Umschichtung der Aufgabe auf eine andere leistungsstärkere virtuelle Maschine. Ebenso kann der Client explizit nach mehr Leistung fragen.

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

Profiler

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

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

Um den Energieverbrauch einzelner Methoden dynamisch zur Laufzeit zu ermitteln, macht sich ThinkAir das *PowerTutor*-Modell zu Nutze [22]. PowerTutor ist ein Energiemodell zur Abschätzung der Energiekosten von CPU, LCD, GPS, WiFi und 3G der Smartphones HTC Dream und HTC Magic. ThinkAir kann über dieses Modell mit einer getesteten maximalen Fehlerrate von 6,27% über verschiedener Geräte den Energieverbrauch einzelner Methoden vorhersagen [11].

TESTERGEBNISSE

Studie [3] beschäftigt sich mit der Netzwerkverfügbarkeit und dem mobilen Datenverkehr von Smartphones in einem Testumfeld von 11 Endnutzern. Sie will damit die Realisierbarkeit von Computation Offloading im Alltag überprüfen. Zur Messung ist auf den Smartphones eine Messwerterfassungs-Applikation installiert.

Unteranderem zeigt die Studie die durchschnittliche tägliche Zeit über einer Woche in Prozent an, in der Benutzer entweder mit 3G/2G oder mit WiFi zum Internet verbunden sind (vgl. Abbildung 3). Es lässt sich feststellen, dass die tägliche Konnektivität über der ganzen Woche mit einer Übertragungstechnologie mit 90% sehr hoch ist [3]. Die Zeiten der Verbindungsabbrüche lassen sich durch signallose Gebiete (zum Beispiel U-Bahnen) oder durch das Ausschalten des Smartphones erklären [3]. Die Zeiten der Konnektivität der unterschiedlichen Technologien (WiFi, 3G, 2G) über der

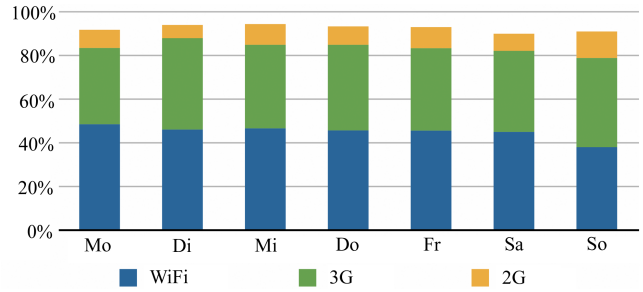

Figure 3. Durchschnittliche tägliche Konnektivität für unterschiedliche Übertragungstechnologien [3]

Woche 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.

Studien [13, 15] analysieren die Energieeffizienz durch Computation Offloading anhand der Trade-off-Ungleichung in Kapitel 2 am Beispiel eines Nokia N900. Dafür vernachlässigen sie die entstehenden Energiekosten des mobilen Geräts beim Warten auf die Beendigung der Berechnung der ausgelagerten Funktion. Die Studien nehmen für ihre Testzwecke an, dass die entsprechende Cloud-Plattform die Aufgabe schnell lösen kann und das Handy wenig Energie beim Warten verbraucht [13]. Man erhält dadurch eine 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 dabei 0.9W Leistung pro Sekunde P_{lokal}. Studie [15] verifiziert diese Ergebnisse. 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 = 600kB 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) [15]. 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 [15]. Unter besten Umständen müssen damit für eine nutzbringende Auslagerung ungefähr eine Millionen Befehle in der Berechnung ausgeführt werden, wenn ein Megabyte an Daten übertragen wird [15]. Dieses Ergebnis deckt sich mit 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 komplexen Berechnungen können ausgelagert werden

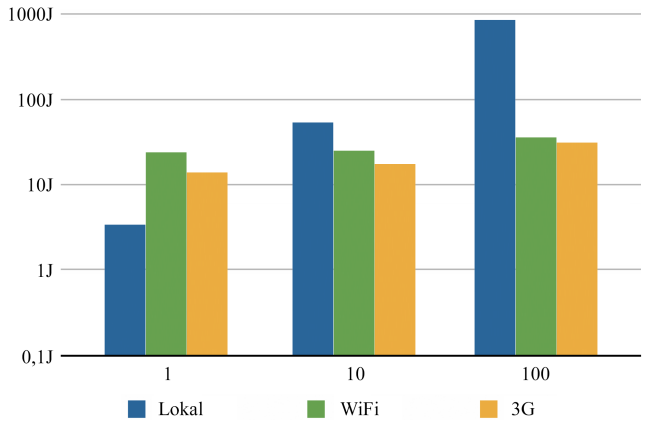

Figure 4. Energieverbrauch bei der Benutzung der unterschiedlichen Übertragungstechnologien für die Gesichtserkennungs-Applikation nach [11]

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

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

Gesichtserkennung

Die Applikation zählt die Anzahl der Gesichter in einem Bild mit Hilfe der Android API `FaceDetector`. Damit ist die Applikation hochgradig für Android optimiert und sollte daher relativ schnell auf dem Smartphone ihre Arbeit verrichten [11]. Der Test wird mit einer variablen Anzahl von zu überprüfenden Fotos durchgeführt (1, 10 und 100). Dabei liegen die Fotos bereits 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 gebraucht würden [11]. Der Unterschied zwischen den Übertragungstechnologien WiFi und 3G ist in diesem Beispiel zu vernachlässigen, da keine Daten an die Cloud übermittelt werden müssen.

Virusscanner

Der Virusscanner ermittelt über einer lokalen beziehungsweise in der Cloud liegenden Datenbank mit 1000 enthaltenden Virussignaturen in einem Dateisystem die Anzahl der gefundenen Viren. Als Testszenario wurden 3500 Daten durchsucht mit einer Gesamtgröße von 10MB [11]. Damit müssen im Gegensatz zur Gesichtserkennung in diesem Fall große Datenmengen an die Cloud ausgelagert werden. Es zeigt sich aber, dass sich die Auslagerung lohnt. 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, 15]), dass die Benutzung von WiFi knapp weniger energieeffizient ist als von 3G [11]. Ähnliche Ergebnisse zeigte auch die Gesichtserkennung.

DISKUSSION

Wie die Testergebnisse des vorangegangenen Kapitels zeigen, bietet Computation Offloading nicht nur die Möglichkeit der Leistungssteigerung, sondern auch die Einsparung von Energie auf Smartphones bei bestimmten Anwendungsszenarien. Dabei ist die Einsparung der Energie auch davon abhängig, wie oft das Smartphone mit der Cloud kommuniziert. Es ist 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. Wird der Netzwerkprozess eine Qual für den Benutzer, so werden sich diese höchstwahrscheinlich von mobilen Cloud-Systemen abwenden [3]. Computation Offloading-Systeme benötigen sorgsame Planung, um sichtbar lange Latenzzeiten für den Benutzer zu vermeiden [15]. Allerdings schmälern eine durchschnittliche gute Konnektivität von 85% am Tag 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 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. Zum Beispiel können Benutzerdaten in der Cloud gespeichert werden oder Inhalte über mehrere Benutzer geteilt werden [15]. Das hat zur Folge, dass für viele Anforderungen die entsprechenden Daten bereits in der Cloud gespeichert sind. Damit reduziert sich der Netzwerktransfer beim Computation Offloading auf ein Minimum. Lediglich die Aufforderung zur Berechnung der Aufgabe muss der Cloud mitgeteilt werden [15]. Bereits heute lagern viele Benutzer ihre Daten an die Cloud aus. Damit wird ihnen eine Sicherheit ihrer Daten garantiert und diese werden verschiedenen Geräten einfach zugänglich gemacht.

Computation Offloading bietet weiterhin den Vorteil, Berechnungen, die sich auf eine Datenbank beziehen, leicht und oh-

ne das Wissen vom Benutzer zu erweitern ohne das dieser durch nervige Updates der Applikation gestört wird.

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 doch lieber lokal auf dem Smartphone laufen lassen will. Bei der heutigen Angst der ständigen Überwachung ist auch Sicherheit ein wichtiges Thema [12]. Ein Dritter kann möglicherweise die Daten bei der Übertragung an die Cloud abfangen. Mit Hilfe von Kryptografie und dem Aufbau von sicheren Verbindungen kann diesem zwar entgegengewirkt werden, es mindert aber 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 zusätzliche Kosten aufwenden. Rein aus energieeffizienten Gründen ist es verständlich, wenn Entwickler auf das Computation Offloading verzichten. So sollte neben dem Gewinn von Energieeinsparung ebenso der Gewinn an Performance beim Computation Offloading im Vordergrund stehen. Es scheint schwierig, Entwickler von der Idee des Computation Offloadings zu überzeugen, nur weil Benutzer nicht gerne ihr Smartphone laden. Ebenso müssen bei unterschiedlichen App-Versionen möglicherweise verschiedene Smartphone-Klone in der Cloud betrieben werden.

In [11] wird auch auf mögliche Schwächen im ThinkAir-Framework eingegangen. So 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 das Überprüfen auf Veränderungen einen gewissen Mehraufwand, der die Trade-off-Ungleichung verschiebt [11]. Daraufhin wird auch auf die Sicherheits-Problematik im ThinkAir-Framework eingegangen. Ein Vorschlag dafür ist ein Authentifizierungsmechanismus, um eine Verbindung über einen Shared-Secret-Service zu registrieren [11]. Für die Übermittlung von privaten Daten, wie dem Standort oder den Profildaten des Benutzers, sollen diese vor Dritten in einer zukünftigen ThinkAir-Version über die `SecureRemotable`-Klasse abgesichert werden und dem Benutzer damit die Last abnehmen, 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 [15]. Die mobile Breitbandabdeckung zum Beispiel 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]. Dadurch kann die Ausführungszeit einer Applikation und dessen Energiekosten weiter gespart und die Benutzerfreundlichkeit der App gesteigert werden. 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 [15]. Frameworks wie ThinkAir müssen sich auch heute noch mit Schätzungen vom Energieverbrauch einzelner Methoden zufriedengeben und implizieren somit natürlich eine gewisse Fehlertoleranz bei der Entscheidung zur Auslagerung [11]. Ebenso können feinkörnigere Energie-Profiler Entwickler zu energieeffizienteren Applikationen motivieren. Energie-Bugs, welche unerwünschterweise viel Energie verbrauchen, können damit besser ermittelt werden.

Die zur Verfügung stehenden Ressourcen in der Cloud sind unvorstellbar und bieten Spielraum und Möglichkeiten für gänzlich neue und innovative Applikationen [15]. Smartphone-Apps mit umfangreichen Berechnungen werden dadurch erst wirklich möglich. So wird zum Beispiel die Datenmenge auf Smartphones auf Grund von einer Anreicherung von Fotos und Videos immer komplexer. Es reicht in der heutigen Zeit nicht mehr aus, diese nur über einen Dateinamen, ein Datum und einen Pfad zu verwalten. Ebenso wollen Benutzer aber nicht jede einzelne Datei über eine Menge von Schlüsselwörtern näher beschreiben. Das bietet enormen Raum und Wachstum für Erkennungs- und Datenverwaltungstechnologien, die auf Grund ihrer intensiven Berechnungen nur über Computation Offloading umsetzbar sind [15]. Es wird spannend, zu sehen, welche neuen Applikationen über Computation Offloading und der Cloud realisierbar werden [15].

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. Willeys and Sons, USA.

2. Arijit Banerjee, Xu Chen, Jeffrey Erman, Vijay Gopalakrishnan, Seungjoon Lee, and Jacobus Van Der Merwe. 2013. MOCA: A Lightweight Mobile Cloud Offloading Architecture. In *Proceedings of the Eighth ACM International Workshop on Mobility in the Evolving Internet Architecture (MobiArch '13)*. ACM, New York, NY, USA, 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 *INFOCOM, 2013 Proceedings IEEE*. 1285–1293.
4. Aaron Carroll and Gernot Heiser. 2010. An Analysis of Power Consumption in a Smartphone. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference (USENIXATC'10)*. USENIX Association, Berkeley, CA, USA, 21–21.
5. Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. 2011. CloneCloud: Elastic Execution Between Mobile Device and Cloud. In *Proceedings of the Sixth Conference on Computer Systems (EuroSys '11)*. ACM, New York, NY, USA, 301–314.
6. Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. 2010. MAUI: Making Smartphones Last Longer with Code Offload. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys '10)*. ACM, New York, NY, USA, 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. Shamim 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. Roelof Kemp, Nicholas Palmer, Thilo Kielmann, and Henri Bal. 2012. Cuckoo: A Computation Offloading Framework for Smartphones. In *Mobile Computing, Applications, and Services*, Martin Gris and Guang Yang (Eds.). Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Vol. 76. Springer Berlin Heidelberg, 59–79.
11. S. Kosta, A. Aucinas, Pan Hui, R. Mortier, and Xinwen Zhang. 2012. ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *INFOCOM, 2012 Proceedings IEEE*. 945–953.
12. Karthik Kumar, Jibang Liu, Yung-Hsiang Lu, and Bharat 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 *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2011 IEEE International Conference on*. 117–122.
14. Irene Manotas, Lori Pollock, and James Clause. 2014. SEEDS: A Software Engineer's Energy-optimization Decision Support Framework. In *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*. ACM, New York, NY, USA, 503–514.
15. Antti P. Miettinen and Jukka K. Nurminen. 2010. Energy Efficiency of Mobile Clients in Cloud Computing. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing (HotCloud'10)*. USENIX Association, Berkeley, CA, USA, 4–4.
16. Y. Neuvo. 2004. Cellular phones as embedded systems. In *Solid-State Circuits Conference, 2004. Digest of Technical Papers. ISSCC. 2004 IEEE International*. 32–37 Vol.1.
17. OpenSignal. 2013. Android fragmentation visualized. (2013). http://opensignal.com/reports/fragmentation-2013/
18. Abhinav Pathak, Y. Charlie Hu, and Ming Zhang. 2012. Where is the Energy Spent Inside My App?: Fine Grained Energy Accounting on Smartphones with Eprof. In *Proceedings of the 7th ACM European Conference on Computer Systems (EuroSys '12)*. ACM, New York, NY, USA, 29–42.
19. S. Robinson. 2009. *Cellphone Energy Gap: Desperately Seeking Solutions*. Technical Report. Strategy Analytics.
20. Aki Saarinen, Matti Siekkinen, Yu Xiao, Jukka K. Nurminen, and Pan Hui. 2011. SmartDiet: Can Offloading Save Energy for Popular Apps? (2011).
21. Huaming Wu, Qiushi Wang, and K. Wolter. 2013. Tradeoff between performance improvement and energy saving in mobile cloud offloading systems. In *Communications Workshops (ICC), 2013 IEEE International Conference on*. 728–732.
22. Lide Zhang, Birjodh Tiwana, Zhiyun Qian, Zhaoguang Wang, Robert P. Dick, Zhuoqing Morley Mao, and Lei Yang. 2010. Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones. In *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES/ISSS '10)*. ACM, New York, NY, USA, 105–114.