Zentrum für sichere Informationstechnologie – Austria
Secure Information Technology Center – Austria

A-1030 Wien, Seidlgasse 22 / 9
Tel.: (+43 1) 503 19 63–0
Fax: (+43 1) 503 19 63–66

A-8010 Graz, Inffeldgasse 16a
Tel.: (+43 316) 873-5514
Fax: (+43 316) 873-5520

http://www.a-sit.at
E-Mail: office@a-sit.at

DVR: 1035461
ZVR: 948166612
UID: ATU60778947

CLOUD-BASED MOBILE AUGMENTATION SYSTEMS

VERSION 1.2 - 16. FEBRUAR 2015

Andreas Reiter – andreas.reiter@a-sit.at

Zusammenfassung: Seit dem Aufkommen von mobilen Geräten wird an Verfahren gearbeitet, um diese leistungsfähiger zu machen oder über Delegation von Aufgaben leistungsfähiger erscheinen zu lassen.. Eines der ersten Verfahren wird als *Cyber Foraging* bezeichnet, dabei werden ressourcenintensive Aufgaben auf umgebende Geräte ausgelagert. Gebündelt mit dem immer populärer werdenden Cloud Computing ergeben sich daraus neue Möglichkeiten, um einem mobilen Gerät scheinbar unbegrenzte Ressourcen zur Verfügung zu stellen. Dieses Dokument untersucht bestehende aktuelle Auslagerungsverfahren und zeigt deren Vor- und Nachteile auf. Außerdem wird ein besonderer Fokus auf Sicherheitsaspekte dieser Verfahren speziell mit Hinblick auf Cloud Computing gelegt.

Inhaltsverzeichnis

1. Einleitung

Durch immer größere Leistungsfähigkeit bei gleichbleibendem Formfaktor von mobilen Geräten wie Smartphones oder Tablets, werden klassische Heimcomputer und Notebooks im täglichen Leben immer mehr verdrängt. Außerdem wird das Anwendungserlebnis für die Benutzerin bzw. den Benutzer durch zusätzliche Sensoren wie GPS, WiFi, Kameras und andere, im Gegensatz zu stationären Geräten erheblich verbessert und ergänzt.

Dedizierte App-Stores für mobile Geräte bieten alle erdenklichen Anwendungen. Beispielsweise hat sich die Anzahl der angebotenen Anwendungen im Google Play Store von Mai 2012 bis Juli 2013 von 500.000 auf 1.000.000 verdoppelt[1]. Die Anzahl der heruntergeladenen Apps hat sich im selben Zeitraum mehr als verdreifacht[2].

Bevor jeder Haushalt mit eigenen Rechnern ausgestattet war, waren Terminals als Schnittstelle zu Großrechnern bekannt. Hierbei wurde die Rechenleistung unter mehreren Benutzerinnen und Benutzern aufgeteilt, dieser Ansatz verschwand aber für den eigenen Gebrauch mit dem Aufkommen von Computern in jedem Haushalt. In 2001 wurde das Konzept in veränderter Form von Satyanarayanan [1] unter dem Namen *Cyber Foraging* mit dem Fokus auf mobile Geräte wieder aufgegriffen. Darunter werden Technologien zusammengefasst, die die Leistungsfähigkeit von ressourcenschwachen Geräten, unter Zuhilfenahme von anderen Geräten aus der Umgebung, vergrößern. Damit ist es möglich, auf Geräten mit eingeschränkter Leistungsfähigkeit Applikationen mit höheren Anforderungen auszuführen, durch Auslagerung von aufwendigen Programmteilen auf geeignete in der Umgebung liegende Systeme. Selbst wenn die Leistungsfähigkeit von aktuellen mobilen Geräten ausreichend wäre, geht die erhöhte Leistungsaufnahme einher mit einer erhöhten Energieaufnahme, was sich auf die ohnedies meist kurze Akkulaufzeit auswirkt. Wie in den folgenden Kapiteln beschrieben, kann sich ein Auslagern von intensiven Rechenaufgaben auch erheblich in der Performance der Applikationen widerspiegeln. Daraus wurden unterschiedlichste Technologien und Systeme entwickelt.

Dieses Dokument bietet eine Übersicht und Kategorisierung über die relevanten. bestehenden Offloading-Systeme, deren Vorteile und Probleme, Außerdem wird besonderes Augenmerk auf eine Sicherheitsbetrachtung der Systeme gelegt.

2. Hintergrund

Seit der Einführung des Begriffs *Cyber Foraging* in 2001 wurde eine Vielzahl von daran angelehnten Konzepten entwickelt. Außerdem hat Cloud Computing in den letzten Jahren eine solide Stellung eingenommen und kann den Benutzern nahezu unbegrenzte Rechenleistung und Speicherplatz anbieten. Cloud Computing ist demnach ideal dafür geeignet, um ressourcenschwache mobile Geräte zu unterstützen. Dies wird von Applikationsanbietern auch aktiv genutzt, um rechenintensive Aufgaben durchzuführen wie Spracherkennung, zum Anreichern von Applikationen um hilfreiche Informationen, Kartenapplikationen zum Anzeigen von aktuellem Kartenmaterial, optimale Wegesuche und viele andere. Außerdem werden Cloud-Anbieter dafür verwendet um mobile Geräte um zusätzlichen und über Gerätegrenzen hinweg verfügbaren Speicher zu erweitern. Voraussetzung für all diese Lösungen ist eine bestehende und ausreichend schnelle Verbindung zum entsprechenden Cloud-Anbieter. Die Aufteilung, welche Teile eines Programmes lokal und welche in der Cloud ausgeführt werden, ist somit bei derzeit produktiv eingesetzten Systemen statisch und es wird nicht auf Gegebenheiten wie verfügbare Netzwerkbandbreite, Latenz oder ähnliches eingegangen.

Die in diesem Dokument untersuchten Systeme fokussieren darauf, dass einerseits die Aufteilung möglichst dynamisch passiert und andererseits der Aufwand, um eine Applikation mit Cloud Unterstützung ausführbar zu machen, gering gehalten wird. Die Ansätze gehen also weit über traditionelle Ansätze wie simple Web-API Schnittstellen, die fix von der Applikation aufgerufen werden, hinaus.

---

[1] http://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/

[2] http://www.statista.com/statistics/281106/number-of-android-app-downloads-from-google-play/

Die Thematik hat mehrere Dimensionen und somit ergeben sich mehrere Möglichkeiten der Einteilung. Ein komplettes System (wie es in Kapitel 3 erläutert wird) besteht prinzipiell aus mehreren Komponenten:

- Das ressourcenschwache Gerät, das unterstützt werden soll.
- Verschiedene verfügbare Ressourcen, eingeteilt nach ihrer Latenz und Bandbreite zum Endgerät.
- Auslagerungs- und Reintegrationsverfahren, um Teile eines Programms remote ausführen zu können.

2.1. Verwendete Ressourcen

Abolfazli et al. [2] führen eine Klassifizierung der verwendbaren Ressourcen ein. Sie unterscheiden zwischen *Entfernte nicht mobile Cloud Ressourcen (Distant Immobile Clouds), Nahe nicht mobile Ressourcen (Proximate Immobile Computing Entities), Nahe mobile Ressourcen (Proximate Mobile Computing Entities)* und *Hybrid*.

Entfernte nicht mobile Cloud-Ressourcen: Hierbei handelt es sich um Cloud-Ressourcen von großen Providern mit Rechenzentren verteilt um die Welt, aber auch um beispielsweise Private-Cloud-Ressourcen von Firmen und Organisationen. Die Verbindung zu diesen Ressourcen wird meist über das Internet hergestellt und es werden entsprechende Sicherheitsmechanismen eingesetzt, um die Verbindungssicherheit zu gewährleisten. Die Effizienz in entfernten, nicht mobilen Cloud-Ressourcen ist hoch, die Latenz vom mobilen Gerät zum Provider ist, verglichen mit den anderen Typen, ebenfalls hoch und kann somit die Performance der gesamten Applikation schwächen. Wenn also nur eine geringe Bandbreite bei relativ hoher Latenz zur Verfügung steht, sind Cloud-Ressourcen nicht die erste Wahl, sind im Gegenzug aber überall verfügbar. Eine weitere Gruppe von Ressourcen, die ebenfalls zu diesem Ressourcentyp gezählt werden kann, aber nur im weitesten Sinne als *Cloud* bezeichnet werden können, sind Recheneinheiten die beispielsweise vom Benutzer selbst zur Verfügung gestellt werden. Als bestes Beispiel hierfür ist ein von der Benutzerin bzw. dem Benutzer selbst aufgesetztes *ownCloud*[3] System zu nennen. Die Daten werden dadurch zwar auf allen möglichen Geräten zur Verfügung gestellt und synchronisiert, der *ownCloud* Server befindet sich aber unter voller Kontrolle der Benutzerin bzw. des Benutzers.

Nahe nicht mobile Ressourcen: An öffentlichen Plätzen werden Computer eingesetzt, um beispielsweise Werbetafeln anzuzeigen, Musik abzuspielen oder werden als einfache Eingabegeräte verwendet und sind meist bei Weitem nicht ausgelastet. Sie verfügen meist über eine breitbandige Internetanbindung und über eine schnelle Funkanbindung. Diese freien Ressourcen könnten von mobilen Geräten in der Umgebung verwendet werden, um ressourcenintensive Aufgaben auszulagern. Ein weiterer Ansatz, der sich in derselben Kategorie ansiedelt, basiert auf sogenannten *Cloudlets* [3], die auch *Cloud-in-the-Box* genannt werden. Bei diesem Ansatz werden dedizierte Recheneinheiten (*Cloudlets*) an öffentlichen Plätzen, Flughäfen, Bahnhöfen oder Cafés bereitgestellt und verfügen über eine entsprechend schnelle Internetanbindung. Diese können von mobilen Geräten zur Auslagerung von Aufgaben verwendet werden, oder dienen als Gateway zu Cloud-Providern. Ein Problem mit diesen Ansätzen könnte die Vertrauensstellung der Benutzerin bzw. des Benutzers des mobilen Geräts zum Betreiber der Recheneinheiten darstellen. Dieser hätte ohne weitere Sicherheitsmaßnahmen die Möglichkeit, die Benutzerin bzw. den Benutzer auszuspähen und könnte eventuell an private, personenbezogene oder vertrauliche Daten kommen.

Nahe mobile Ressourcen: Eine weitere Möglichkeit besteht darin, Aufgaben auf in der Nähe gelegene mobile Geräte auszulagern wie beispielsweise Smartphones, Tablets oder Notebooks. Idealerweise besitzen die verwendeten Geräte ein ähnliches Bewegungsmuster, sodass sie möglichst lange verfügbar und im Umkreis der Benutzerin bzw. des Benutzers sind. Die Sicherheitsbedenken, wie sie im vorherigen Absatz beschrieben wurden, gelten auch hier unverändert, da die verwendeten mobilen Geräte unter der Kontrolle von anderen Personen sind und eventuell mit Malware oder ähnlichem infiziert sein könnten.

[3] https://owncloud.org/

Hybrid: In der Praxis muss sich ein System immer an die Gegebenheiten anpassen, und es werden nicht immer alle Anforderungen von nahe gelegenen und vertrauenswürdigen Recheneinheiten abgedeckt werden können.

Abbildung 1 zeigt eine Übersicht über die bisher beschriebenen Ressourcentypen zur Auslagerung von ressourcenintensiven Aufgaben.

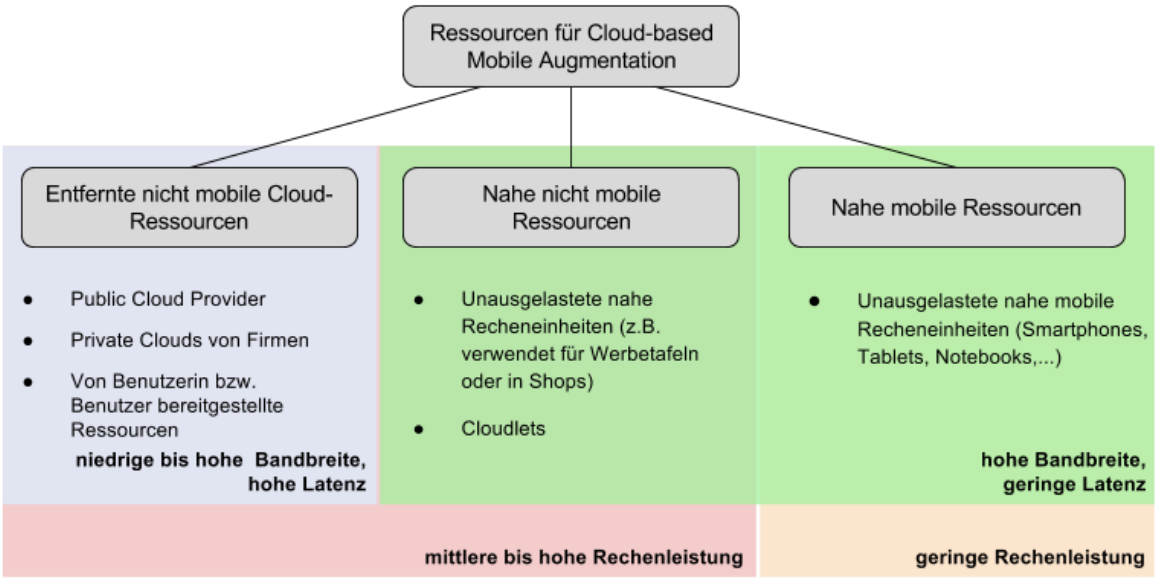

Abbildung 1 - Übersicht über mögliche Ressourcen zur Aufgaben Auslagerung

Die in Kapitel 3 beschriebenen Systeme können je nach Typ der verwendeten Ressource weiter eingeordnet werden und besitzen mit den zuvor eingeführten Begriffen Synergien:

Cloud Computing ist eine serviceorientierte Herangehensweise, um Benutzerinnen und Benutzern nach dem pay-as-you-go Verfahren IT-Dienstleistungen anzubieten. Diese werden im Allgemeinen auf drei Ebenen angeboten: Infrastructure-as-a-Service (IaaS) bietet virtualisierte Hardware Platformen, Platform-as-a-Service (PaaS) bietet Benutzerinnen und Benutzern fertig einsetzbare Laufzeitumgebungen zur eigenen Serviceentwicklung und Software-as-a-Service (SaaS) bietet der Benutzerin bzw. dem Benutzer fertige Services für einen definierten Zweck. Bekannte Cloud Computing Anbieter sind beispielsweise Google AppEngine[4], Microsoft Azure[5] und Amazon Web Services[6]. Der Vorteil von Cloud Computing gegenüber herkömmlichen IT Infrastrukturen liegt auf der Hand. Serviceanbieter können ohne große Investitionskosten und ohne Vorlaufzeit neue Ressourcen hinzufügen und damit Auslastungsspitzen abfangen. Im Kontext von Cloud-based Mobile Augmentation (CMA) wird Cloud Computing dazu eingesetzt, um von mobilen Geräten aus Services zu verwenden, um die Leistungsfähigkeit des mobilen Gerätes scheinbar zu steigern. Dieses Model deckt sich mit dem von Flores et al. [4] beschriebenem *Delegation* Model, in dem Geräte durch die Verwendung einer Service-orientierten Struktur entlastet werden. Problem dabei ist, dass die Aufteilung lokal bzw. remote ausgeführter Teile fixiert ist, und somit auf aktuelle Gegebenheiten (wie beispielsweise hohe Latenzzeiten) nicht eingegangen werden kann. Außerdem ist der Migrationsaufwand, um eine Applikation teils lokal und teils remote auszuführen, generell damit verbunden, auszulagernde Teile für die Ausführung in der Cloud neu zu implementieren. Cloud Computing ist also laut der zuvor eingeführten Kategorisierung der Gruppe der *entfernten nicht mobilen Cloud Ressourcen* zuzuordnen.

[4] https://appengine.google.com/
[5] https://azure.microsoft.com
[6] http://aws.amazon.com

Mobile Cloud Computing (MCC) ist ein weiterer Schritt von *Cloud Computing* hin zu mobilen Geräten und deren Erweiterung der Funktionalitäten. Shiraz et al. [5] definieren *Mobile Cloud Computing* als ein neues Paradigma im Kontext von *Utility Computing*, das den Ansatz auf Geräte mit eingeschränkten Ressourcen ausweitet. *Mobile Cloud Computing* verwendet also verschiedene Ansätze und Komponenten, die im weiteren Verlauf dieses Dokuments beschrieben werden, um ressourcenintensive Aufgaben auszulagern. Dazu werden wie in Abbildung 2 dargestellt je nach Verfügbarkeit verschiedene Verbindungstypen herangezogen. MCC umfasst nicht nur die Auslagerung von Aufgaben (sprich also die Vortäuschung von Rechenleistung) sondern auch beispielsweise die Erweiterung um zusätzlichen Speicher.

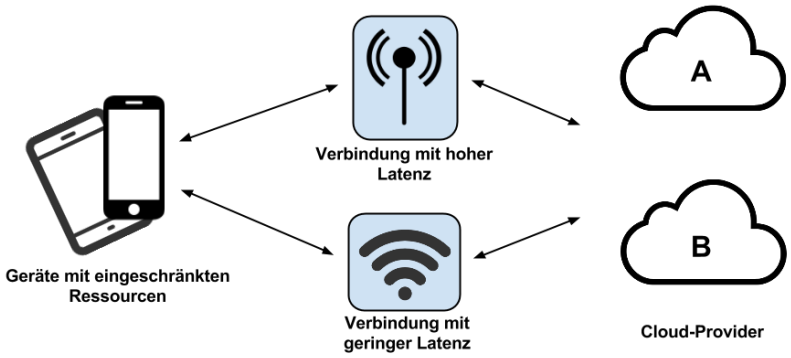

Abbildung 2 - Mobile Cloud Computing Komponenten

Surrogate Computing oder *Cyber Foraging* [1] bezeichnet ähnlich wie MCC das Erweitern der Leistungsfähigkeit von Geräten durch Auslagern von Aktivitäten. Hierbei wird die Palette an verwendeten Geräten erhöht durch die Berücksichtigung von sich in der Umgebung befindlichen Geräten mit freien Ressourcen. Dabei kann es sich um stationäre Rechner, Notebooks oder Cloudlets [3], aber auch um andere mobile Geräte handeln. Nach einer Legitimation dieses Ansatzes muss nicht lange gesucht werden. Laut Satyanarayanan et al. [3] sind bereits niedrige (für WAN-Netzwerke) Latenzzeiten von 33 ms ein Problem für Benutzerinnen und Benutzer, und lässt die Anwendungserfahrung sinken. Im Test wurde eine Applikation lokal und über verschiedene Remotedesktopanwendungen ausgeführt. Entweder die Bildwiederholfrequenz erlitt drastische Einbrüche, oder die Anwendungen reagierten, zumindest im Vergleich zur lokalen Ausführung, nur sehr langsam. Ein Ausweg daraus wäre die Verbesserung der Latenzzeiten für WAN-Netzwerke. In letzter Zeit gehen Verbesserungen aber eher in Richtung gesteigerter Bandbreite. Abbildung 3 zeigt das erweiterte Komponentenmodell von MCC. Einerseits können ressourceneingeschränkte Geräte Aufgaben an andere Geräte (wie beispielsweise Notebooks, Smartphones, Tablets oder stationäre Rechner) aus der Umgebung auslagern. Dabei sollte darauf geachtet werden, dass Geräte mit ähnlichen Bewegungsmustern gewählt werden, damit sie möglichst lange verfügbar sind, da natürlich jedes Gerät eine gewisse Vorlaufzeit benötigt, bis es einsatzbereit ist. Es können aber auch sich in der Gegend befindliche *Cloudlets* gewählt werden, die für sich wiederum eine Verbindung zu Cloud-Providern haben können.

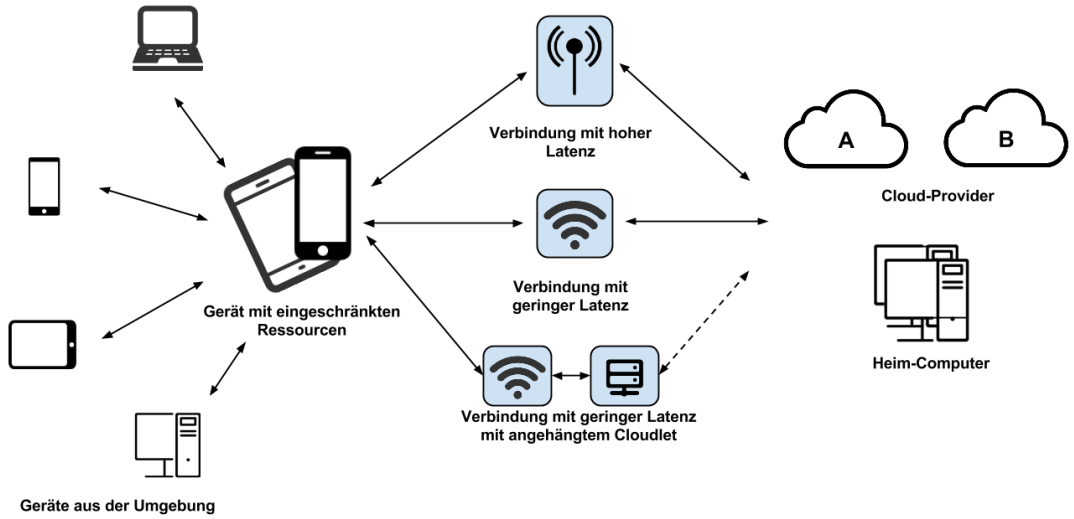

Abbildung 3 - Cyber Foraging Komponenten

2.2. Auslagerungs- und Reintegrationsverfahren

Das Auslagerungs- und Reintegrationsverfahren ist eine Kernfunktionalität in einem *Cloud-based Mobile Augmentation* System. Sie setzt sich prinzipiell aus zwei Komponenten zusammen: Einem Partitionierungs- bzw. Segmentierungsalgorithmus und einer Auslagerungs- bzw. Offloading Komponente.
Der Partitionierungsalgorithmus ist dafür verantwortlich, eine Applikation in Teile, die ausgelagert werden können, und Teile, die lokal ausgeführt werden müssen, aufzuteilen. Lokal ausgeführt wird beispielsweise jegliche Interaktion mit der Benutzeroberfläche, und Interaktionen mit am Gerät verbauten Sensoren wie GPS, Beschleunigungssensoren und anderen. In weiterer Folge muss außerdem eine Abschätzung ermittelt werden, wie effizient unter den aktuellen Gegebenheiten eine Auslagerung ist. Die Partitionierung kann statisch zur Compilezeit, dynamisch zur Laufzeit oder aus einem Mix aus beiden Verfahren stattfinden. In der Praxis wird meistens die letztere Methode verwendet werden indem von der Entwicklerin bzw. vom Entwickler beispielsweise eine grobe statische Vorpartitionierung vorgegeben wird. Die Vorpartitionierung kann z.B. durch markieren von Gruppen von Klassen erfolgen die ausgelagert werden können, oder basiert auf der Auslagerung von parallelen Ausführungssträngen. Außerdem wird dynamisch zur Laufzeit analysiert, welche Teile unter den aktuellen Bedingungen (verfügbare Bandbreite, Latenz, Energieverbrauch, verfügbare eigene Rechenleistung) derzeit am günstigsten für eine Auslagerung in Frage kommen. Damit endet die Zuständigkeit des Partitionierungsalgorithmus, alle weiteren Schritte werden von der Offloading-Komponente durchgeführt.
Die Offloading-Komponente führt die Auslagerungen vom lokalen Gerät auf die remote Ressource durch. Dafür gibt es mehrere Ansätze mit unterschiedlichem Granularitätslevels.

Virtual Machine (VM) basierte Ansätze wie in [6] zielen darauf ab, dem Gerät einen möglichst ähnlichen und kompatiblen virtuellen Rechner zur Verfügung zu stellen. Darauf können unveränderte Applikationen des Geräts ausgeführt werden, was einerseits den Entwicklungsaufwand senkt, und andererseits schnell für eine Vielzahl an Applikationen verfügbar sein kann. Die Nachteile dieses Ansatzes liegen allerdings auf der Hand.

- Es müssten für viele erdenkliche Kombinationen an Hardware und Betriebssystem vorgefertigte virtuelle Images bereitgestellt werden.
- Muss eine neue virtuelle Maschine gestartet werden, ergibt sich eine erhebliche Verzögerung. Auch wenn sich diese nur im kleineren Sekundenbereich befindet, ergibt sich für den Benutzer eine deutlich merkbare Verzögerung.
- Die gesamten Applikationen müssen dem Ressourcenprovider zur Verfügung stehen oder müssen bei Bedarf hochgeladen werden.

- Durch den Einsatz von kompletten virtuellen Abbildern kommt es zu erheblichem Mehraufwand an Speicher und Prozessorleistung, der den der eigentlichen Applikation sogar übersteigen könnte.

Klassen- oder *Methoden-basierte* Ansätze wie in [7] sind einen Granularitätslevel höher und konzentrieren sich auf gesamte Klassen oder Methoden einer Applikation. Es wird dabei vor dem Aufruf der entsprechenden Klasse entschieden, ob Methoden lokal oder remote ausgeführt werden. Je nach verwendeter Programmiersprache der Applikation ist dadurch im Gegensatz zum VM-basierten Ansatz keine idente Ausführungsumgebung erforderlich. Bei Plattformunabhängigen Programmiersprachen wird der Status der Applikation bzw. der Status des Ausführungsstranges auf das Ausführungsziel repliziert und dort die Ausführung fortgesetzt. Dieser Ansatz benötigt potentiell eine größere Unterstützung vom Entwickler, eliminiert aber die bei der VM-basierten Variante induzierten Verzögerungen aufgrund des Overheads zum Starten und Einrichten der virtuellen Maschine. Es müssen zwar auch wie beim VM-basierten Ansatz zumindest Teile des Programms und Daten übertragen werden, es besteht aber eher die Möglichkeit durch intelligente Extraktion der betroffenen Programmteile die Größe dieser Daten gering zu halten.

Objekt-basierte Ansätze wie von Sinha et al. [8] beschrieben. Hier wird ein konkretes Beispiel einer Fotomanipulationssoftware herangezogen. Dabei werden von einer Klasse verschiedenste Operationen (beispielsweise Effekte) angewandt. Durch einen Objekt-basierten *Offloading* Ansatz können Instanzen desselben Objekts durch voraussagen der benötigten Rechenleistung auf unterschiedliche, geeignete Ressourcen ausgelagert werden. Im Prinzip ist dieser Ansatz dem Klassen- oder Methoden-basierten Ansatz sehr ähnlich, verfügt aber über weitere und tiefer gehende Analysemethoden.

Im folgenden Kapitel 3 wird eine Übersicht über bestehende Offloading Systeme geboten.

3. Bestehende Offloading Systeme

Dieses Kapitel bietet eine Übersicht über bestehende Offloading Systeme und bewertet diese anhand einer Zuordnung zu den in den vorherigen Kapiteln identifizierten Verfahren und Gruppierungen.
Im Allgemeinen können Offloading Systeme über die Komponenten, wie in Abbildung 4 schematisch dargestellt, beschrieben werden wobei nicht jede Komponente für jedes Framework von Relevanz ist.

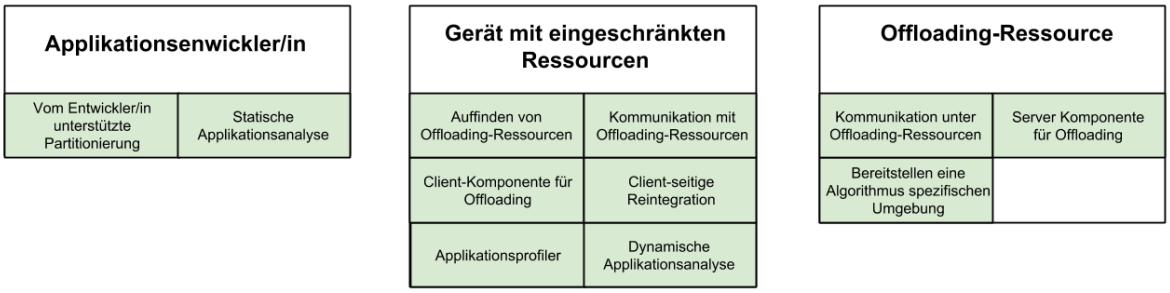

Abbildung 4 - Offloading-System Komponenten

Die Applikationsentwicklerin bzw. der Applikationsentwickler kann, wenn unterstützt, den statischen Vorpartitionierungsprozess unterstützen wie es beispielsweise in Kapitel 3.3 beschrieben wird. Zusätzlich kann zur Kompilierungszeit bzw. während des Entwicklungsprozesses ein statisches Analysetool verwendet werden, das den Entwickler bei der Identifikation von Programmteilen, die lokal bzw. remote ausgeführt werden können, unterstützt.

Das Gerät auf dem eine ressourcenintensive Applikation ausgeführt wird, kann hingegen über eine dynamische Applikationsanalysekomponente verfügen. Diese arbeitet auf Basis von Daten des *Application Profiler* und der *Offloading resource discovery* Komponente. Der *Application Profiler* überwacht die Ausführung des Programms und kann Zeitabschätzungen über die Ausführungsdauer oder Speicherabschätzungen für gewünschte Methoden und Klassen liefern. Die *Offloading resource discovery* Komponente steht in Kontakt mit verfügbaren *Offloading Ressourcen* so dass bei Bedarf schnell eine Ressource bereitgestellt werden kann. Die Kommunikation mit der Ressource erfolgt über die *Offloading resource communication* Komponente und kann beispielsweise über simple Web Schnittstellen erfolgen, aber auch über komplexere Systeme wie peer-to-peer Netzwerke. Die zentralen Komponenten die sich auf das Gerät aber auch auf die verwendete Ressource aufteilen sind die Offloading- und Reintegrations-Komponenten. Sie führen die eigentliche Migration des auszuführenden Programmteils auf die remote Ressource durch und integrieren das Ergebnis sobald es von der Ressource geliefert wird.

Die Offloading Ressource verfügt über den oben beschriebenen Serverteil der Offloading Komponente, um einerseits eventuell Programme oder Fragmente von Programmen vom Client abzurufen und andererseits diese auszuführen und das Ergebnis dem Client zur Verfügung zu stellen. Eine weitere zentrale Komponente der Offloading Ressource ist außerdem die Komponente zur Bereitstellung der Ausführungsumgebung wie beispielsweise kompatible Virtual-Machine-Abbilder oder kompatible Ausführungsumgebungen auf anderen Ebenen. Die Komponenten der Offloading Ressource können schwer generalisiert werden da sie sehr stark von der jeweiligen Implementierung abhängen.

In den folgenden Unterkapiteln wird eine Liste von relevanten Offloading Frameworks präsentiert. Relevante Frameworks definieren sich dadurch, dass die damit betriebenen Anwendungen nicht auf spezielle Fälle reduziert werden, sondern, dass sie generell für ein breites Spektrum an verschiedensten Anwendungen einsetzbar sind.

3.1. AlfredO

AlfredO [9] basiert auf *R-OSGi* [10], eine Middleware die die Ideen und Gedanken von OSGi [11] übernimmt. OSGi wurde entwickelt, um Java Anwendungen einfach und effizient in lose gekoppelte Module aufteilen zu können. R-OSGi setzt dies fort und ermöglicht es, OSGi Module verteilt auf mehreren Systemen auszuführen. Die Vision bzw. das Ziel von *AlfredO* ist es, Mobiltelefone zu universellen Schnittstellen für umgebende Geräte werden zu lassen. Dabei sollen die Alleinstellungsmerkmale von mobilen Geräten nicht verloren gehen, wie beispielsweise Touchscreens, Kameras und andere Sensoren. Um die Sicherheit der mobilen Geräte nicht zu gefährden und die gesteckten Ziele zu erreichen führt *AlfredO* drei Mechanismen ein:

- Mittels eines *Service-basierten Softwareverteilungsmodells* (vergleichbar mit SaaS) können Mechanismen die von R-OSGi bereitgestellt werden verwendet werden um Software zu verteilen bzw. es können remote Schnittstellen angefordert werden, um ein Service am Zielgerät auszuführen.
- AlfredO definiert eine *Multi-tier Service Architektur* bestehend aus der Präsentationsebene, der Logikebene und der Datenebene. Die Ausführung kann dynamisch auf das Mobiles- bzw. das Zielgerät verteilt werden.
- Die *Geräte und Formfaktor unabhängige Präsentationsschicht* ermöglicht es, die Vorteile der einzelnen Geräte optimal zu nutzen, indem nur Beschreibungen der Benutzeroberfläche versendet werden und jedes mobile Endgerät assembliert sich eine eigene, auf das Gerät angepasste Benutzeroberfläche.

AlfredO verfügt außerdem über einen nur sehr kleinen Footprint am mobilen Gerät von minimal 290 kBytes, wobei dieser stark abhängig von bereitgestellten Komponenten wie verschiedenen Render Komponenten ist.
Da der Ansatz von AlfredO auf R-OSGi bzw. OSGi basiert, ist die Anwendung auf Geräte limitiert, auf denen einerseits eine Java Virtual Machine verfügbar ist und andererseits die

Konzepte von R-OSGi unterstützt werden. Ersteres stellt kein Problem dar, da Java auf einer Vielzahl von Geräten und Plattformen[12] verfügbar ist, aber nur mit eingeschränktem Funktionsumfang, und somit für OSGi jede Plattform nochmals evaluiert werden müsste. Außerdem muss bei dem von AlfredO verfolgten Konzept der Softwareentwickler bereits sehr stark auf diese Trennung eingehen, eine dynamische Partitionierung ist dadurch nur schwer möglich.

3.2. Misco

Misco [13] verwendet das *Map-Reduce* Verfahren, um Aufgaben abzuarbeiten. Unter dem *Map-Reduce* Verfahren versteht man Aufgaben, für die eine hohe Rechenleistung benötigt wird, aufzuteilen und hochparallelisiert abzuarbeiten. Dazu werden aus großen Aufgaben kleinere erzeugt die parallel ausgeführt werden können. Es werden dabei zwei Schritte durchgeführt. Im ersten Schritt, der *Map* Phase, wird eine Funktion auf ein Set von Eingabedaten angewandt, und so ein Zwischenergebnis berechnet. Im zweiten Schritt, der *Reduce* Phase, werden gruppierte Zwischenergebnisse kombiniert und zu den Endergebnissen verarbeitet. Abbildung 5 zeigt eine Übersicht der Komponenten des Misco Frameworks. Es besteht aus einem Server der über das Benutzerinterface Aufgaben entgegennimmt, und aus *Worker*-Geräten die, sobald freie Ressourcen verfügbar sind, vom Server neue Aufgaben abrufen. Ein *Worker* erhält jeweils nur einen *Map*- oder *Reduce* Task, arbeitet diesen ab und liefert das Ergebnis an den Server. Als *Worker* können in diesem System unter anderem auch mobile Geräte verwendet werden. Neue Aufgaben werden vom Benutzer über eine Web-Schnittstelle eingegeben.

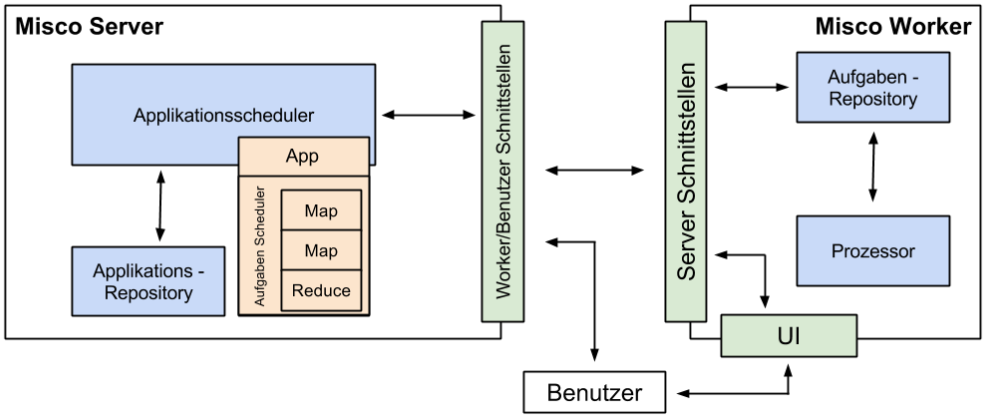

Abbildung 5 - Misco Komponenten[13]

Das Konzept und die Arbeitsweise von Misco sind nur bedingt mit den anderen hier beschriebenen Frameworks vergleichbar, trotzdem bietet es eine interessante Herangehensweise durch Aufteilung der Rechenleistung.
Die Schwachstelle im Misco-System ist schnell ausgemacht. Es gibt einen zentralen Punkt (der Misco Server) der die komplette Verteilung der Aufgaben innehat. Fällt diese Komponente aus, kommt das gesamte System zum Erliegen. Das System wurde in Python entwickelt und bietet somit eine relativ breite Kompatibilität mit bestehenden Systemen. Trotzdem stellt sich die Frage, wie sehr es sich lohnen würde, vorhandene (mobile) Applikationen auf das Misco System zu portieren, da Python auf mobilen Geräten eher ein Schattendasein führt.

3.3. MAUI

MAUI [7] ist ein Ansatz, der auf eine möglichst stromsparende Ausführung von Applikationen abzielt, um eine Erhöhung der Akkulaufzeit von mobilen Geräten zu erreichen. Prinzipiell gibt es, wie in Kapitel 2 beschrieben, die Ansätze die Benutzerin bzw. den Benutzer die Partitionierung einer Applikation durchführen zu lassen, oder komplette Systemabbilder auszulagern, was den Entwickler entlastet und automatisiert die Programmzustände migriert. MAUI kombiniert beide Möglichkeiten durch die Einschränkung auf *Managed Code*

Environments (im speziellen Fall auf Microsoft .NET [14]). MAUI operiert auf dem Methodenlevel und bietet somit einen hohen Granularitätslevel. Es werden die auslagerbaren Methoden markiert, was im Gegenzug aber nicht zwingend bedeutet, dass diese Methoden bei jedem Aufruf auch ausgelagert werden. Es wird ein Optimierungsframework verwendet, welches diese Entscheidung auf Basis von Faktoren wie verfügbare Konnektivität (3G, Wifi), Round-Trip-Time (RTT) und Transferaufwand um den aktuellen Stand der Applikation auf das Offloading Ziel zu laden, in Betracht zieht und ein Optimierungsproblem formuliert und löst. Die markierten Methoden können als von der Entwicklerin bzw. vom Entwickler vorgeschlagene initiale Applikationspartitionierung angesehen werden. Von der Entwicklerin bzw. vom Entwickler muss dabei nur berücksichtigt werden, dass Methoden die zur Interaktion mit der Benutzerschnittstelle oder mit am mobilen Gerät verfügbaren Ressourcen nicht remote ausgeführt werden können. Sollte die Verbindung zum Offloading Server verloren gehen, führt MAUI die Methode erneut lokal aus, was somit nur zu geringen Geschwindigkeitseinbußen führt, aber nicht zum kompletten Fehlschlagen der Applikation.
MAUI verwendet ein *Profiling-* und *Solver-Model,* dabei werden die Programmabläufe vom *Profiler* kontinuierlich überwacht und Änderungen in den Gegebenheiten aufgezeichnet. Diese Daten dienen als Eingabeparameter für den *Solver* der entscheidet welche Methoden ausgelagert werden.
Das Interessante an dem Ansatz, der von MAUI verwendet wird, ist die Einfachheit mit der Entwicklerinnen bzw. Entwickler Applikationen für dieses System entwickeln können bzw. auf dieses System migrieren können. Mit etwas Aufwand und unter Berücksichtigung der Abläufe im MAUI Framework können Entwicklerinnen bzw. Entwickler durch Umstrukturierung von Applikationen noch erheblich höhere Performancegewinne erzielen.
MAUI setzt auf *Managed* Programmiersprachen (wie das Microsoft .NET Framework) um ein möglichst breites Einsatzgebiet zu erreichen. Leider spielt das .NET Framework auf mobilen Geräten aber eher eine Nebenrolle. Laut IDC Bericht [15] liegt der Marktanteil von Windows Phone im zweiten Quartal 2014 bei 2,5%. Es gibt zwar auch Möglichkeiten, um .NET Applikationen auf Android bzw. iOS auszuführen. Das ist derzeit aber nicht der Standard und liegt wohl auch nicht im Interesse des jeweiligen Plattformherstellers. Der MAUI Ansatz ist, wie auch von den Autoren angemerkt, nicht auf das .NET Framework beschränkt, und kann auf andere *Managed Code* Umgebungen übertragen werden. Trotzdem handelt es sich dann um zwei Systeme die nicht miteinander kompatibel sind, und Ressourcen untereinander nicht austauschen können.

3.4. CloneCloud

CloneCloud [6] ist ein Ansatz, um Aufgaben in Klone der eigenen Plattform (also in Virtuelle Maschinen mit möglichst ähnlichen Eigenschaften) auszulagern mit dem Ziel die Ausführungsgeschwindigkeit bzw. die Kosten zu minimieren. CloneCloud kann auf unveränderten Programme angewendet werden, und benötigt nicht wie beispielsweise MAUI aus Kapitel 3.3 spezielle Annotationen oder Markierungen. Dazu wird eine Kombination aus statischer Partitionierung und dynamischem Profiling verwendet.
Mittels statischer Analyse werden gültige Partitionen bestehend aus Migrations- und Reintegrationspunkten gesucht und für eine spätere Verwendung gespeichert. Dabei müssen drei fundamentale Eigenschaften erfüllt werden: (1) Methoden die Geräteabhängige Funktionalitäten verwenden, müssen auch am jeweiligen Gerät ausgeführt werden, (2) Methoden, die auf nativen Status zugreifen müssen, immer am selben Gerät (bzw. immer in der virtuellen Maschine) ausgeführt werden und (3) verschachtelte Auslagerungsoperationen sind nicht erlaubt.
Das dynamische Profiling generiert eine Kostenabschätzung für die Applikation unter unterschiedliche Ausführungsgegebenheiten durch das Aufrufen der Applikation mit zufällig generierten Eingabedaten. Derzeit wird eine Metrik bestehend aus Ausführungszeit und verbrauchter Energie verwendet.
In weiterer Folge entscheidet der *Solver* aufgrund der zur Verfügung stehenden Daten, ob die Ausführung einer Methode ausgelagert wird oder ob sie lokal ausgeführt wird.
Der von Chun et al. [6] vorgestellte Prototyp basiert auf Android und ist somit theoretisch in der Lage, jede beliebige auf Android lauffähige Java Applikation auszulagern. Die Migrations- und Reintegrationspunkte werden in die Applikationen mittels speziell eingeführter Java-VM

Instruktionen gekennzeichnet. Es ist deshalb erforderlich, jedes Endgerät und jede Zielressource mit modifizierten Android Versionen bzw. modifizierten Java-VMs auszustatten. Dies bedeutet im Gegenzug, dass keine dieser analysierten und mit Migrations- und Reintegrationspunkten versehenen Applikationen mehr auf unmodifizierten Dalvik- (bzw. Java-) VMs lauffähig ist. Da nach wie vor die auf aktuellen Android Geräten verwendeten Versionen sehr breit gestreut zwischen Version 2.2 und 4.4 sind [16], scheint es sehr unwahrscheinlich dass diese Technologie ein breites Anwendungspublikum bzw. eine weite Verbreitung findet.

3.5. ThinkAir

ThinkAir [17] greift die Konzepte von MAUI [7] und CloneCloud [6] auf und erweitert sie um Skalierbarkeit und um eine online Methodenbasierte-Auslagerung. ThinkAir benutzt eine dynamisch Herangehensweise, um Ressourcen bei Bedarf anzufordern und nutzt die Parallelisierbarkeit von Aufgaben, durch dynamisches starten bzw. beenden von virtuellen Maschinen, um die Ausführungszeit zu verringern.
ThinkAir wurde rund um vier Designziele gestaltet:

1. *Dynamische Anpassung an sich verändernde Umgebungen:* Da Offloading Frameworks hauptsächlich auf mobile Geräte abzielen und diese sich in einer ständig verändernden Umgebung befinden ist es wichtig, dass sich ein Framework dynamisch an neue Gegebenheiten anpassen kann
2. *Leichte Verwendbarkeit für Entwickler:* Die Einstiegshürde für Softwareentwickler muss möglichst niedrig gehalten werden, um eine möglichst große Verbreitung zu erzielen. Das Framework muss für weniger erfahrene und erfahrenere Benutzerinnen und Benutzer verwendbar sein.
3. *Performancegewinne durch Cloud Computing:* Das oberste Ziel von ThinkAir ist es, die Applikationsperformance zu verbessern und die Energieaufnahme zu optimieren.
4. *Dynamische Skalierbarkeit der Rechenleistung:* Verschiedene Anwendungen bzw. verschiedene Anwenderinnen bzw. Anwender haben unterschiedliche Bedürfnisse. Ein Framework muss sich diesen Anforderungen anpassen können.

Die Kombination von MAUI und CloneCloud äußert sich dadurch, dass einerseits auslagerbare Methoden ähnlich wie in MAUI annotiert werden, somit eine Vorpartitionierung stattfindet, und andererseits, es wie bei CloneCloud einen Kompilierungsschritt gibt, in dem der Programmcode für die Remoteausführung generiert wird.
Der ThinkAir-Prototyp wurde für Java auf Android-basis entwickelt. Die Ausführung von ausgelagerten Applikationen findet auf je nach Ressourcenanforderungen gestarteten virtuellen Maschinen statt.
Der Ansatz von ThinkAir ist durch die Kombination von einer MAUI inspirierten Vorpartitionierung und einer dynamisch, von CloneCloud inspirierten Ausführungsschicht vielversprechend. Es wird jedoch, wie in Kapitel 3.4 beschrieben, nicht der Overhead der durch die Bereitstellung von virtuellen Maschinen, die möglichst kompatibel zum mobilen Gerät sind, eliminiert oder die im Verhältnis hohe und für die Benutzererfahrung erhebliche Zeit bis neue virtuelle Maschinen bereitgestellt werden können.

3.6. Cuckoo

Cuckoo [18] ist ebenfalls ein auf Android basierendes Offloading-Framework, welches im Gegensatz zu ThinkAir und CloneCloud leicht abgeänderte Ziele verfolgt:

- Als oberstes Ziel steht die Einfachheit für die Entwicklerin bzw. den Entwickler. Es wird deshalb eine vollständige Integration in den Entwicklungsablauf, inklusive Integration in Entwicklertools bereitgestellt.
- Ein weiteres Ziel ist die Einfachheit auf Seiten der Ressourcenprovider. Cuckoo geht nicht von einer vollständigen Android virtuellen Maschine aus, sondern begnügt sich mit einer Java-VM die ohne größeren Aufwand bereitgestellt werden kann.

Außerdem verfolgt Cuckoo das Konzept, dass der auf dem Gerät laufende Programmteil nicht zwingend der selbe sein muss wie der Programmteil auf der remote Ressource. Dazu werden von der Entwicklungsumgebung Interfaces generiert die für eine Remoteausführung implementiert werden müssen. Dieser Code kann entweder von der lokalen Implementation übernommen werden, oder aber komplett unabhängig davon implementiert werden. Dabei können beispielsweise Funktionalitäten verwendet werden, die am mobilen Gerät nicht unterstützt werden, durch die vollwertige Java-VM auf der Auslagerungsressource aber abgedeckt sind.
Die Möglichkeit der getrennten Implementierung scheint eine Ressourceneffiziente Variante zu sein um Aufgaben auszulagern, erfüllt aber nicht das Streben nach einem möglichst einfachen Weg um bestehende Applikationen zu migrieren. Es ist von der Entwicklerin bzw. vom Entwickler immer erheblicher Einsatz gefordert, bis hin zur kompletten Umstrukturierung der Applikation.

3.7. COSMOS

COSMOS [19] ist ein *Offloading* System mit dem Ziel *Offloading-as-a-Service* bereitzustellen und so die entstehenden Kosten für die Auslagerungsoperationen so gering wie möglich zu halten. COSMOS basiert ebenfalls auf Android, auf den Ressourcen werden Android x86 Abbilder verwendet.
Das Problem anderer Systeme ist, dass eine Virtuelle Maschine immer exklusiv einem mobilen Gerät zugeordnet ist. COSMOS erweitert das System bestehend aus Cloud-Ressource und mobilem Gerät um eine weitere Komponente, den COSMOS Master. Der COSMOS Master überwacht alle Ressourcen und reagiert frühzeitig auf Auslastungsspitzen durch Starten von neuen virtuellen Maschinen bzw. Beenden von virtuelle Maschinen um die Kosten gering zu halten. Zur Auswahl der Auslagerungsressource werden drei Möglichkeiten in Betracht gezogen: (1) Der Task wird direkt an den COSMOS Master gesendet, der diesen dann an einen freien COSMOS Server weiterleitet, (2) Der Client sucht sich selbst einen COSMOS Server mit freien Ressourcen, (3) Der COSMOS Master stellt dem Client einen COSMOS Server mit freien Ressourcen zur Verfügung.
Auf den Offloading Algorithmus an sich wird in [19] nicht näher eingegangen. Die beschriebenen Konzepte eliminieren allerdings das in den letzten Kapiteln angesprochene Problem einer langen Wartezeit durch starten von virtuellen Maschinen, da der COSMOS Master immer überwacht das ausreichend freie Ressourcen verfügbar sind.

3.8. Cloud Offloading für Web Applikationen

Der von Hwang et al. [20] beschriebene Ansatz zum Auslagern von Web Applikationen basiert auf der in HTML5 eingeführten Web Worker Spezifikation [21]. Web Worker sind quasi parallele Ausführungsstränge, die aber im Gegensatz zu anderen Programmiersprachen sehr entkoppelt vom Hauptausführungsstrang abgearbeitet werden, und mit diesem nur über *PostMessage* kommunizieren können.
Es bietet sich deshalb an, diese separaten Ausführungsstränge, über Kommunikation mit Cloud-Ressourcen mittels Web Sockets auszulagern, und das Ergebnis dem Hauptausführungsstrang wieder zu melden.
Das Verfahren scheint vielversprechend zu sein, da es nicht auf einzelne Plattformen eingeschränkt ist. Es wird aber noch weiteren Forschungsaufwand benötigen, da vermutlich die Granularität wenn rein nur Web Worker ausgelagert werden zu gering sein wird.

4. Sicherheitsbetrachtung

Allgemeine oder klassische Sicherheitsbedenken verursacht durch beispielsweise Malware, schwache Netzwerksicherheit, fehlerhafte Applikationen, falsch konfigurierte Applikationen und viele andere treffen auch genauso auf Cloud Computing zu.
Die Unterschiede von Cloud Computing zu klassischen IT Infrastrukturen sind laut Ryan [22]:

1. Cloud-Ressourcen werden unter vielen Benutzerinnen und Benutzern aufgeteilt, jede Benutzerin und jeder Benutzer könnte ein Angreifer sein.

2. In der Cloud gespeicherte Daten können auch über unsichere Protokolle bzw. über öffentliche Netzwerke abgefragt werden.
3. Die volle Verantwortung über die Datenintegrität liegt beim Cloud-Provider.
4. Der Cloud-Provider und seine Subunternehmer können alle gespeicherten Daten einsehen.

Unter der Annahme, dass ein Cloud-Provider rein als Datenspeicher verwendet wird, kann Punkt 1 durch eine entsprechende Isolierung und Punkt 2 durch strenge Policies auf Seiten des Providers entschärft werden. Die Wahrung der in Punkt 3 erwähnten Datenintegrität, kann entweder von der Benutzerin bzw. dem Benutzer durch den Einsatz von verschiedenen Cloud-Providern, oder auf Seiten des Cloud-Providers durch das Erstellen von Backups in unterschiedlichen (geographisch getrennten) Datenzentren erreicht bzw. verbessert werden. Für Punkt 4 gibt es Lösungsansätze wie beispielsweise *Fully homomorphic encryption* oder es werden generell nur verschlüsselte Daten abgelegt. Auf das Konzept der *Fully homomorphic encryption* wird hier nicht weiter eingegangen, es ist aber noch nicht so weit fortgeschritten, dass es für allgemeine Fälle effizient eingesetzt werden kann. Der zweite Ansatz, generell nur verschlüsselte Daten in der Cloud abzulegen funktioniert so lange, bis Teile davon auch in der Cloud weiterverarbeitet werden sollen. Dazu muss dann entweder der zur Entschlüsselung benötigte Schlüssel dem Service bereitgestellt werden (was gleichbedeutend mit der Ablage von unverschlüsselten Daten ist) oder die Daten gleich unverschlüsselt abgelegt werden.

In [23] werden noch weitere Sicherheitsbedenken bezüglich Cloud-Services geäußert, die hier kurz zusammengefasst werden:

- *Speicherort der Daten:* Da große Cloud-Provider meist international agieren, haben Kundinnen und Kunden generell keinen Einfluss darauf, wo ihre Daten gespeichert oder repliziert werden. Der Provider wird die für ihn günstigste Konstellation auswählen. Dies könnte zu rechtlichen Problemen führen wenn beispielsweise Daten bestimmte geographische Bereiche nicht verlassen dürfen, vom Cloud-Provider aber beispielsweise weltweit verteilt werden.
- *Mehrere Benutzer verwenden dasselbe Service:* Generell sind Cloud-Services darauf ausgelegt, mehrere Benutzerinnen und Benutzer bzw. Benutzergruppen zu bedienen. Als Benutzerin oder Benutzer muss man daher darauf vertrauen, dass der Provider entsprechende Sicherheitsmaßnahmen einsetzt, um diese voneinander zu isolieren.
- *Monitoring und Logdateien:* Mit der zunehmenden Migration von Applikationen in die Cloud wird es für Benutzerinnen und Benutzer auch immer wichtiger, detaillierte Logausgaben zu erhalten. Diese Logausgaben könnten sensitive Daten enthalten.
- *Cloud Standardisierung:* Für einzelne Teilbereiche sind zwar Standards mit Cloud Computing Tangente verfügbar, generell unterstützt jeder Provider aber nur Applikationen, die dediziert für diesen einen Provider entwickelt worden sind, und es ist ein erheblicher Aufwand notwendig, um Applikationen von einem Cloud-Provider zu einem anderen zu transferieren (Vendor-lock-in).

Im Endeffekt läuft es darauf hinaus, dass mit dem Cloud-Provider ein Vertrauensverhältnis eingegangen werden muss, wenn man die vollen Vorteile des Cloud Computing verwenden möchte.
Betrachtet man nun Cloud-Provider nicht nur als Datenspeicher, sondern in einem *Cloud-based Mobile Augmentation (CMA)* Kontext, so fallen die oben erwähnten Maßnahmen zur Wahrung der Privatsphäre gänzlich weg. Dem Cloud-Provider muss der ausführbare Code und der dazu notwendige Kontext (Status der Applikation, Methodenparameter, Klassenvariable,…) zur Verfügung stehen. Daraus lassen sich in einem *Cloud-based Mobile Augmentation* Kontext folgende Bedrohungen für Benutzerinnen und Benutzer, aber auch für den Cloud-Provider ableiten:

- *Leak von persönlichen Daten:* Wie bereits im Datenspeicher Beispiel oben angeführt muss mit dem Cloud-Provider ein Vertrauensverhältnis eingegangen werden. Selbst wenn in der auszuführenden Applikation codiert ist, dass es sich hierbei um persönliche Daten handelt, hat man als Benutzerin bzw. Benutzer keine Kontrolle

darüber, was der Cloud-Provider damit tut, ob sie zwischengespeichert werden oder wo und in welchem Rechenzentrum sie verarbeitet werden. Abbildung 6 illustriert den Vorgang. Hierbei kann die Benutzerin bzw. der Benutzer keine Notiz davon nehmen, welche Aktionen mit den Daten bzw. welche Aktionen generell in der Cloud ausgeführt werden.

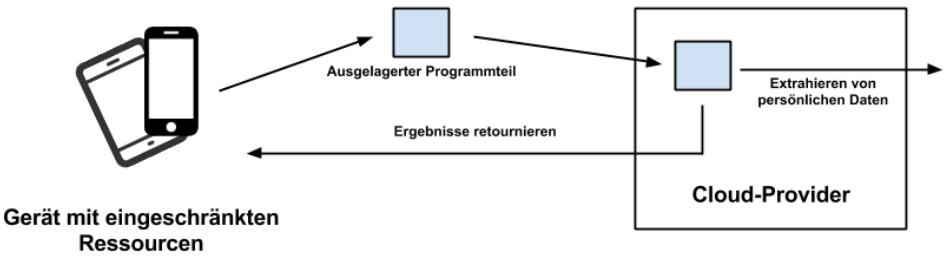

Abbildung 6 - Leak von persönlichen Daten

- *Der Cloud-Provider führt potentiell schadhaften Code aus:* In einem CMA System erhält der Cloud-Provider von den Benutzerinnen bzw. Benutzern Programme oder Programmteile, die unverändert ausgeführt werden müssen und das Ergebnis zurückgeliefert werden muss. Der Cloud-Provider kann natürlich Überprüfungen durchführen und bestimmen, ob erhaltene Programmteile schadhaften Code beinhalten oder nicht. Eine absolute Sicherheit kann aber, wie auch heute bei Webapplikationen mit dynamischen Inhalten nicht gegeben werden. Die Gefahr bzw. das Potential bei CMA Systemen, und somit der Anreiz schadhaften Code auszuführen, könnte aber als erheblich höher eingestuft werden, da man im Allgemeinen nicht auf Web-basierte Sprachen und damit auf deren Funktionalitäten beschränkt ist. Um dem entgegenzuwirken, könnte der Cloud-Provider beispielsweise nur vertrauenswürdige Programmteile ausführen bzw. Programmteile von vertrauenswürdigen Benutzerinnen und Benutzern. Damit wird ein System aber erheblich eingeschränkt, und es wäre eine Datenbank von einem vertrauenswürdigen Herausgeber notwendig in der alle entsprechenden Programme bzw. Programmteile indiziert sind.
- *Der Cloud-Provider führt nicht den gewünschten Code aus:* Dieser Punkt weist Synergien mit den beiden Vorhergehenden auf. Wie in Abbildung 7 dargestellt, führt der Cloud-Provider nicht den gewünschten Code aus und liefert eventuell andere Ergebnisse, als würde das Programm lokal ausgeführt werden. Außerdem könnte die Methode um Analysemethoden zum Auswerten der übergebenen Daten erweitert werden.

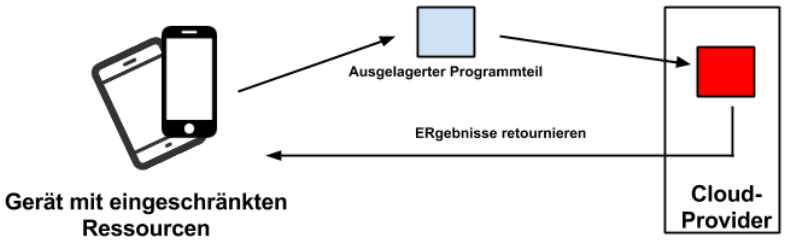

Abbildung 7 – Cloud-Provider führt unerwünschten Code aus

- *Öffnen von Sicherheitslücken am Cloud-Provider:* Dadurch, dass in der Cloud von mobilen Geräten beliebiger Code ausgeführt werden kann, könnten schadhafte Programme die Cloud-Infrastruktur kompromittieren und beispielsweise nicht autorisierten Dritten Zugriff gewähren.

- *Auswirkungen von einer Recheneinheit auf eine andere:* Je nach Verfahren, das vom CMA System verwendet wird, besteht von Haus aus eine schwächere bis stärkere Isolierung unter den verschiedenen Benutzerinnen bzw. Benutzern. Wie in Kapitel 3.4 beschrieben verwendet *CloneCloud* für jede Benutzerin und jeden Benutzer eine eigene virtuelle Maschine, und verfügt somit über eine hohe Isolation, und es müsste im VM Hypervisor eine Lücke gefunden werden, um auf Daten anderer Benutzer zugreifen zu können. Im Fall von COSMOS, wie in Kapitel 3.7 beschrieben, werden Virtuelle Maschinen unter Benutzern geteilt, um die Kosten für Cloud-Ressourcen zu minimieren. Hier besteht die einzige Isolation zwischen den Benutzern in unterschiedlichen Prozessen, die möglicherweise sogar auf derselben Java VM ausgeführt werden.

5. Fazit und Ausblick

Das vorliegende Dokument bietet einen Überblick über bestehende *Cloud-based Mobile Augmentation* Verfahren und die dabei eingesetzten Technologien. Die Verfahren können je nach ihrer Granularität beim Auslagerungsprozess und der verwendeten Ressourcen eingeteilt werden. Eine überwiegende Mehrheit der vorhandenen Implementationen setzt auf virtuelle Klone der mobilen Geräte in der Cloud (oder vergleichbare Konzepte) obwohl dadurch ein nicht unerheblicher Overhead in Bezug auf Ressourcenverbrauch und Zeit bis das Auslagerungsframework bereit ist entsteht. Bezüglich der zur Auslagerung verwendeten Ressourcen wird in den meisten Fällen ausschließlich auf Cloud-Ressourcen zurückgegriffen, was mit der Verwendung von virtuellen Maschinen einhergeht. Die Benutzung und Bündelung von nahegelegenen Ressourcen mit niedrigen Latenzzeiten wird außer von Misco und AlfredO nicht ernsthaft in Betracht gezogen, sondern es werden zu einem großen Teil Wege gesucht um die weit entfernten Ressourcen effizient zu erreichen, durch Zugriff über WiFi oder durch Zwischenschalten von Cloudlets. Durch die fortschreitende Verfügbarkeit von Ressourcen in vielen Gebieten würde es aber durchaus Sinn ergeben diese durch geeignete Verfahren zu koppeln und zu einem gewissen Grad Benutzerinnen und Benutzern zur Verfügung zu stellen, da einerseits nicht überall schnelle Funkverbindungen zur Verfügung stehen, und andererseits beispielsweise im Ausland durch hohe Daten Roaming Gebühren auf die Nutzung entsprechender Services eher verzichtet wird.

Auffallend ist, dass sich keines der Frameworks mit Sicherheitsaspekten oder entstehenden Sicherheitsproblemen beschäftigt. Dieses Dokument identifiziert Sicherheitsprobleme und bietet, wenn vorhanden, Lösungsvorschläge an. Trotzdem besteht ein klarer Bedarf für Ansätze die sich den oben erwähnten Sicherheitsaspekten widmen.

6. Referenzen

[1] M. Satyanarayanan, “Pervasice Computing: Vision and Challenges,” pp. 10–17, 2001.

[2] S. Abolfazli, Z. Sanaei, E. Ahmed, A. Gani, and R. Buyya, “Cloud-Based Augmentation for Mobile Devices: Motivation, Taxonomies, and Open Challenges,” *IEEE Commun. Surv. Tutorials*, vol. 16, no. 1, pp. 337–368, 2014.

[3] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, “The Case for VM-Based Cloudlets in Mobile Computing,” *Pervasive Comput. IEEE*, vol. 8, pp. 14–23, 2009.

[4] H. Flores and S. N. Srirama, “Mobile Cloud Middleware,” *J. Syst. Softw.*, vol. 92, pp. 82–94, Jun. 2014.

[5] M. Shiraz, A. Gani, R. H. Khokhar, and R. Buyya, “A Review on Distributed Application Processing Frameworks in Smart Mobile Devices for Mobile Cloud Computing,” vol. 15, no. 3, pp. 1294–1313, 2013.

[6] B. Chun, S. Ihm, and P. Maniatis, “Clonecloud: elastic execution between mobile device and cloud,” *Proc. sixth …*, pp. 301–314, 2011.

[7] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Stefan, R. Chandra, and B. Paramvir, “MAUI : Making Smartphones Last Longer with Code Offload,” *Energy*, vol. 17, pp. 49–62, 2010.

[8] K. Sinha and M. Kulkarni, “Techniques for Fine-Grained, Multi-site Computation Offloading,” *2011 11th IEEE/ACM Int. Symp. Clust. Cloud Grid Comput.*, pp. 184–194, May 2011.

[9] J. S. Rellermeyer, O. Riva, and G. Alonso, “AlfredO : An Architecture for Flexible Interaction with Electronic Devices,” pp. 22–41, 2008.

[10] J. S. Rellermeyer, G. Alonso, and T. Roscoe, “R-OSGi : Distributed Applications Through Software Modularization,” pp. 1–20, 2007.

[11] OSGi Alliance, *OSGi Service Platform, Core Specification, Release 4, Version 4.3*. 2011, pp. 1–352.

[12] “Java Verified - Table of Supported Devices,” 2014. [Online]. Available: http://javaverified.com/device_matrix. [Accessed: 08-Oct-2014].

[13] A. Dou, “Misco : A MapReduce Framework for Mobile Systems ∗ ,” 2010.

[14] Microsoft, “Overview of the .NET Framework.” [Online]. Available: http://msdn.microsoft.com/en-us/library/zw4w595w%28v=vs.110%29.aspx.

[15] IDC, “Smartphone OS Market Share, Q2 2014.” [Online]. Available: http://www.idc.com/prodserv/smartphone-os-market-share.jsp.

[16] “Android Dashboards.” [Online]. Available: https://developer.android.com/about/dashboards/index.html. [Accessed: 08-Oct-2014].

[17] S. Kosta, A. Aucinas, and R. Mortier, “ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading,” *2012 Proc. IEEE INFOCOM*, pp. 945–953, Mar. 2012.

[18] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, “Cuckoo: a computation offloading framework for smartphones,” *Mob. Comput. Appl. …*, pp. 59–79, 2012.

[19] C. Shi, K. Habak, P. Pandurangan, M. Ammar, M. Naik, and E. Zegura, “COSMOS : Computation Offloading as a Service for Mobile Devices,” 2014.

[20] I. Hwang and J. Ham, “Cloud Offloading Method for Web Applications,” *2014 2nd IEEE Int. Conf. Mob. Cloud Comput. Serv. Eng.*, pp. 246–247, Apr. 2014.

[21] “Web Worker Specification.” [Online]. Available: http://www.w3.org/TR/workers/. [Accessed: 09-Oct-2014].

[22] M. D. Ryan, “Cloud computing security: The scientific challenge, and a survey of solutions,” *J. Syst. Softw.*, vol. 86, no. 9, pp. 2263–2268, Sep. 2013.

[23] C. Rong, S. T. Nguyen, and M. G. Jaatun, “Beyond lightning: A survey on security challenges in cloud computing,” *Comput. Electr. Eng.*, vol. 39, no. 1, pp. 47–54, Jan. 2012.