Computing at the Mobile Edge: Designing Elastic Android Applications for Computation Offloading

Gabriel Orsini, Dirk Bade, Winfried Lamersdorf
Distributed Systems Group
Department of Computer Science
University of Hamburg, Germany
Email: [orsini,bade,lamersd]@informatik.uni-hamburg.de

***Abstract*—Current centralized cloud datacenters provide scalable computation- and storage resources in a virtualized infrastructure and employ a use-based "pay-as-you-go" model. But current mobile devices and their resource-hungry applications (e.g., speech- or face recognition) demand for these resources on the spot, though a mobile device's intrinsic characteristic is its limited availability of resources (e.g., CPU, storage, bandwidth, energy). Thus, *mobile cloud computing* (MCC) was introduced to overcome these limitations by transparently making accessible the apparently infinite cloud resources to the mobile devices and by allowing mobile applications to (elastically) expand into the cloud. However, MCC often relies on a stable and fast connection to the mobile devices' surrogate in the cloud, which is a rare case in mobile scenarios. Moreover, the increased latency and the limited bandwidth prevent the use of real-time applications like, e.g. cloud gaming.**
Instead, *mobile edge computing* (MEC) or *fog computing* tries to provide the necessary resources at the logical edge of the network by including infrastructure components to create ad-hoc mobile clouds. However, this approach requires the replication and management of the applications' business logic in an untrusted, unreliable and constantly changing environment. Consequently, this paper presents a novel approach to allow mobile app developers to easily benefit from the features of MEC. In particular, we present a programming model and framework that directly fit the common app developers' mindset to design elastic and scalable edge-based mobile applications.

I. Introduction

For the first time in 2011, smartphones overtook traditional PCs in terms of shipped units [1]. Along with this trend, the constantly increasing use of mobile devices has led to the porting of resource-hungry desktop applications and the development of mobile-first applications that make use of the mobile devices' special capabilities (e.g., augmented reality apps). However, in addition to limited interaction capabilities, mobile devices lack computational power, storage capacity, energy and they suffer from a network interface with low bandwidth, high latency and intermittent connectivity. To overcome these obstacles and to allow even more sophisticated applications being used by mobile users, external resources have to be woven into the local execution of mobile applications [2].

But still, the limited bandwidth and the high latency can have a significant impact on the usability and the user experience. Hence, to ensure an acceptable performance of widely distributed mobile applications, it is not always possible to rely on centralized cloud resources as the single backend. *Fog computing* [4], a term coined by Cisco Systems and also known as *(mobile) edge computing* [5], *mist computing* [6] or under the concept of *cloudlets* [7], relies on the assumption that it is impractical or even impossible to always send all data across the whole internet from the mobile devices to the cloud service provider. Accordingly, fog computing aims at providing resources like computation power, storage and business logic at the logical edge of the network and through a more geographically distributed platform, rather than at centralized spots, as it happens nowadays through e.g. Amazon AWS and Microsoft Azure cloud datacenters. A prominent example is Cisco's *IOx* operating system that allows to run third-party virtual machine-based services directly on infrastructure components like routers (see. Fig. 1), located at the edge of the network. This enables to offer new mobile services like mobile gaming [8] or augmented reality [9] and further resource-demanding applications on current mobile devices.

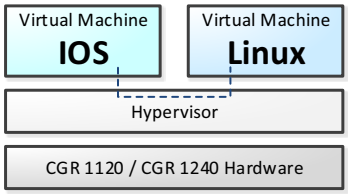

Fig. 1. Cisco IOx architecture [3]

In previous works, we introduced our idea of a context-adaptive MCC framework [10]. This paper contributes with the extension of the framework with elasticity features that allow the development of both, cloud- and edge-augmented mobile applications, by focusing on a holistic approach between computation offloading and context adaptation that eases the development of MCC/MEC applications by offering a flexible architecture that provides programming abstractions, multi-level distribution transparency and context adaptation features without modifying the underlying mobile operating system.

The remainder of this paper is structured as follows: Section II further clarifies the different characteristics of mobile cloud- and mobile edge computing and related paradigms. Afterwards, application characteristics are exemplified in Section III whose general requirements and challenges are explained on the background of related work in Section IV. Finally, our *CloudAware* framework is presented in Section V. At the end, we summarize our findings and give prospects for open challenges in Section VI.

978-0-7695-5662-8/15 $31.00 © 2015 IEEE
DOI 10.1109/WMNC.2015.10

II. Background

In this chapter we will further refine the definition of the different MCC paradigms, its synonyms, as well as related concepts and outline their differences. Afterwards, a typical MCC architecture for computation offloading and their main components are presented.
As Figure 2 indicates, the several terms for MCC like fog computing, mist computing and MEC have been coined to describe a concept, in which devices with limited capabilities incorporate the resources of other (nearby) devices. We conclude that the main reason for the multitude of terms relies on the fact, that the edge of the network is not clearly defined. Whether it is the mobile device itself, a proximate LTE base station or the mobile network operators' core network. In the following we will differentiate the two most distant concepts, which are the mobile cloud computing (MCC) and the (mobile) edge computing (MEC), integrate the remaining definitions and conclude with a definition of the currently most prominent term: fog computing.

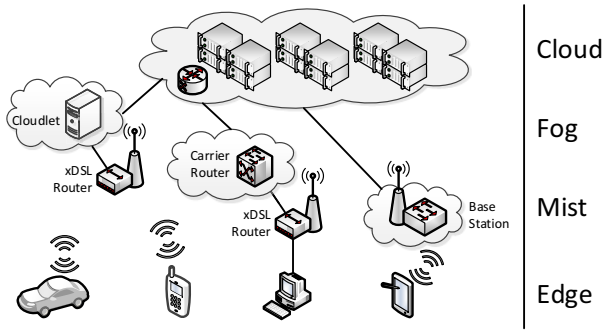

Fig. 2. Different perceptions of (mobile) edge computing

A. Mobile Cloud Computing

Mobile cloud computing tries to push the boundaries of mobile applications by offering centralized resources to augment mobile devices. According to [2] it is defined as the integration of cloud computing into the mobile environment to overcome obstacles related to performance (e.g., battery life, storage and bandwidth), environment (e.g., heterogeneity, scalability and availability), and security (e.g., reliability and privacy).

While this first generation of cloud-augmented mobile applications was highly domain-specific and relies on a stable connection to specific cloud servers, recent work [11]–[14] has aimed to tackle this problem and generalize this approach to allow more mobile applications to participate in MCC without having to deal with the specific details of the underlying infrastructure. Here, instead of relying on conventional service calls, MCC applications mostly adopt the concept of code offloading by migrating certain parts (code and state) of a mobile application into the infrastructure or onto other nearby devices, so-called cloudlets (see Section I), and collecting the results once they are computed. However, in current MCC solutions the elasticity, i.e. the ability to integrate surrogates and parallelize tasks, is often limited to a single specific surrogate. Moreover, context adaptation features to handle the effects of intermittent connectivity and providing mechanisms for disconnected operation are often not considered. The common denominator of many approaches and research efforts in the domain of MCC is offloading computations rather than data. In the following we concentrate on computation offloading, too, but not solely.

B. (Mobile) Edge Computing

As an evolution to mostly centralized resource augmentation strategies like MCC, mobile edge computing or simply egde computing moves further in terms of decentralization. While MCC was the first step of connecting mobile devices with centralized resources, MEC advances and tries to move the major part of remote operations from central resources directly into the surrounding infrastructure and so includes the logical extremes of a network. To do so, it replicates parts of an application's business logic onto nearby devices. Coined in 2004 by Akamai [15] it was first used to describe the topology of their content delivery networks that was used to cache often-requested contents at the logical edges of the network, but today the definition is used to provide more complex services by using cloud computing principles in a pay-as-you-go manner. Current ready-to-market solutions include switches like the Cisco IOx operating system [3] that already provides computation and storage in virtual machines at the logical edge of the network or the Nokia Siemens Networks' intelligent base stations [16] that allow the integration of lightweight third-party software components that extend the base stations' core functionalities with application-specific features.

To summarize, the benefits of edge computing are its idea to move load from the centralized resources, but even more important to avoid bottlenecks and single points of failure. MEC aims at reducing latency by shifting computational efforts from the centralized cloud to the mobile edge. Hence, providing a continuous availability and reducing the latency to allow real-time applications like cloud gaming. From a topology point of view it can be seen as an ad-hoc or peer-to-peer interaction between devices, located close to each other in terms of network metrics. A similar concept, to be located somewhere between MCC and MEC is the so-called fog computing, defined by [17] as: *"[...] a highly virtualized platform that provides compute, storage, and networking services between end devices and traditional cloud computing data centers, typically, but not exclusively located at the edge of network."*

Introducing the concept of an intermediate layer between mobile devices and cloud resources, MEC and fog computing are expected to further improve latency and execution speed. Even if both, fog computing and MEC might appear similar to grid computing, the difference is that the distribution logic is not explicitly woven into the specific application, but implicitly handled by an underlying infrastructure layer. However, whenever computations need to be performed quickly and locally, they are candidates for fog computing.

Both, fog computing and its even younger brother mist computing, also coined by Cisco and located somewhere between the fog and the edge, extend the classical client-server architecture to a more peer-to-peer based approach, similar or equal to MEC [18]. In the remainder of this paper, we will refer to all of the mentioned concepts by MEC only.

III. Application Characteristics

To further illustrate the idea of MEC, a generalized application scenario is exemplified and the main characteristic of MEC applications are presented: The device the user interacts with has resource constraints, but the tasks that should be carried out require either computational power (and energy), loads of data, network bandwidth or additional resources. Depending on the context, the tasks can be offloaded to surrogates at the network's edge where they are processed and results are finally sent back to the initiating device.

The primary objectives for using edge nodes as intermediaries between mobile devices and clouds are:

- Provide services or deliver content close to the user's access point (LTE or WiFi)
- Process or filter large amounts of data before they are transferred
- Speed up geo-distributed applications like sensor networks
- Perform distributed large-scale analysis of real-time data
- Allow latency-sensitive applications like cloud gaming and real-time video analytics

IV. Related Work

Several solutions have been proposed for the challenges of MEC. Most of them try to either see the problem as a software engineering- or a networking problem, which is why related work exists in several domains and will be classified accordingly. Further related work can also be found in the domains of mobility- and connectivity prediction like in mobile ad-hoc networking (MANET) or vehicular ad-hoc networking (VANET). For complete surveys, we refer to [18]–[22]. In the following, we will focus the most promising and current work only. Furthermore we omit emerging software-defined networking concepts as they only allow to inject routing logic into the network hardware, but no generic application logic. According to [23], neither an elastic resource, nor a programming model for general purpose applications is available.

As mentioned earlier, the development of MCC applications performing computation offloading, is often complex and requires proper support to ease the development. Models like inter-process communication, remote method invocation or service invocation cannot be employed right away to the domain of mobile cloud computing [24]. Classical MCC aproaches like *CloneCloud* [12], *MAUI* [11], *Thinkair* [13] and *Cuckoo* [14] have tried to solve this issue by focusing on the offloading onto centralized surrogates, either on the granularity of a thread (CloneCloud), a method (MAUI and Thinkair) or a component (Cuckoo). Even if all of them have showed that offloading can be beneficial in terms of speedup or energy savings, this only holds for cases in which little synchronization of a shared state between the mobile device and the surrogate is required. As an evolution to the mentioned solutions in the domain of classical MCC other solutions have been proposed that take the effects of intermittent connectivity into consideration, the most promising ones will be described next:

Serendipity [25] is a framework that is able to distribute computation tasks among other nearby mobile devices to speed up computation or to save energy. A mobile device may execute tasks locally or remotely on available surrogates based on the selected optimization criterion. Hereby, offloading is planned, using different algorithms, depending on the current reliability of the mobile network connection: In more stable scenarios a control channel is used to coordinate the offloading tasks, whereas, when the future connectivity cannot be estimated, more fault-tolerant and robust mechanisms come into play. Similarly, *IC-Cloud* [26] focuses on the challenge of dynamically offloading computation tasks to surrogates, by taking into consideration that the necessary code and data can be delivered and the results received in time before the next link failure is likely to happen. Its offloading algorithms are designed to handle different levels of intermittent connectivity, from those in which the connectivity is predictable to those in which the next link failure cannot be foreseen. As an evolution of the two aforementioned ones, *COSMOS* [27] provides computation offloading as a generic service on the Android platform that incorporates the different connectivity states.

Similar, *ToGo* [28] has been proposed as a framework that predicts the length of stays at certain WiFi hotspots to perform what the authors call "preemptive offloading". For this purpose, a hidden Markov model is trained to predict the probability of the remaining time that a user is connected to a certain hotspot. Using this information, computation tasks are offloaded, if the expected waiting period is less than remaining connection time. Furthermore, the authors introduce a safe-point concept between the surrogate and the mobile device that allows to re-use partial results of the remote execution. After a link failure, the mobile device uses the previously received safe-point to continue execution. Another approach is presented by [29], where one waits for a re-connection to receive the results of offloading. But both solutions do not appear to be optimal in terms of efficiency or speed gains.

A first approach, that is directly tailored to the domain of fog computing, is Mobile Fog [23]. The authors propose a high-level programming model that they envision for future internet applications which are geospatially distributed, large-scale, and latency-sensitive. They propose to address the specific challenges of heterogeneous devices by providing suitable event handlers and an application programming interface.

But the last-mentioned problem is not only relevant in the domain of fog computing, although it is pushed to an extreme here, but has as well tried to be solved in related domains like connectivity prediction and location prediction, as the connectivity of a mobile device is most dependent on its physical location. Two prominent approaches that focus on the aspect of connectivity prediction are presented next: *BreadCrumbs* [30] calculates the future connectivity to WiFi hotspots based on a model of the environment. Recorded information is used to generate user-based models which are then applied to schedule network usage based on connectivity forecasts. BreadCrumbs relies on the fingerprinting of hotspots that is combined with GPS data to predict a mobile user's bandwidth. In *Wiffler* [31] a similar approach is employed for the domain of VANETs to impove the bandwidth and reduce the 3G usage by offloading communication onto WiFi.

Here, the main problem is the extremely short duration of each WiFi connection. Focusing the aspect of the location prediction even further, in *NextPlace* [32] a non-linear method is employed to predict the time and duration of a user's next visit to one of his significant places. Their method identifies patterns in a users' mobility history that are similar to his recent movements in order to predict his behavior. Similar, Anagnostopopulus et al. [33] employ supervised learning to perform a classification of trajectories which is then used to predict the future location of mobile users. *PnLUM* [34] provides a soft classifier for predicting a user's future location. Instead of one complex model for capturing user mobility patterns, the authors use multiple models, each focusing only on a certain aspect of location prediction such as the likelihood of location transitions.

Summarizing the previous findings, we conclude that several solutions have been proposed to contribute to the field of MEC by addressing the aforementioned requirements. However, there is no ready-to-use solution, as none of the current solutions is able to address all requirements. As a consequence, we present the architecture of the CloudAware framework, a holistic approach to integrate the challenges of both domains: computation offloading and context adaptation.

V. CloudAware

CloudAware differs from previous or similar approaches in the domain of MEC by its primary design goal to support ad-hoc and short-time interaction with not only centralized resources but also nearby devices. This idea is extended by the secondary design goal, which is to provide an uninterrupted availability of the mobile application even if no surrogates are available or the connection gets interrupted by using the mobile device as a fallback. It remains the primary instance to hold the mobile applications' relevant state. To achieve this type of spontaneous interaction, classical client/server solutions, service composition/orchestration, or prominent MCC approaches are often not suitable as they are not able to cope with the requirements of the ad-hoc interaction or as they are not as lightweight enough to meet the limited resources of a mobile device. How CloudAware faces these restrictions and which general assumptions motivate specific design decisions, will be highlighted in the following.

A. Main Challenges and General Design Decisions

The Offloading Granularity: In our previous work [20] we analyzed different categories of approaches that allow computation offloading. Our findings regarding a suitable offloading granularity for MEC scenarios are as follows: The most promising MCC solutions can be broadly classified in two categories with respect to their offloading granularity: The first relies on the concept of virtual machines and provides almost full execution transparency by moving either the thread [12] or the method call [11] to a more powerful surrogate in the cloud. The main benefit of these approaches is their almost automatic offloading that requires only little or even no modification of the mobile applications' source code and can hence be considered "hands-free". But this high level of distribution transparency bears several drawbacks, as often higher synchronization efforts are necessary, both for the initial setup of the heavyweight virtual machine image and to preserve a consistent state between the two devices. Moreover, performance gains by the parallelization of tasks are often not achievable when the developer's expert knowledge is not captured appropriately, which is also stated by Porras et al. [24] and Flinn [35].

To capture the expert knowledge, the second category of MCC solutions relies on a more explicit modeling of the applications' dependencies by providing conceptual building blocks, e.g. components, to ease the partitioning and to parallelize tasks. This way, an initially linear program can e.g. be easily refactored to a number of parallel tasks [24]. With regard to the aforementioned requirements of MEC, we chose self-contained components as the offloading granularity, as they have proven by Giurgiu et al. [36] and with μ*Cloud* [37] to be a viable concept in scenarios with intermittent connectivity where a loose coupling with only little synchronization efforts is required. Following this approach, object-oriented programming still remains the first choice to develop mobile applications and well integrates into the common development process and hence the typical developer's skills and mindset, as they represent a well-known engineering paradigm. In contrast to service orchestration approaches, this approach enables the developer to just "write his code" as usual.

The Network Model: Mitigating the effects of intermittent connectivity can either be seen as a networking or as a software engineering problem. Even if there is a current trend to virtualize networks and their services, providing edge computing as a generic network service would require large standardization efforts, but standards are currently not available and early standardization efforts are just emerging [5]. Moreover, employing MEC and its transparent execution of networked applications is almost impossible without proper kernel and networking layer support and access to it, but network carriers and mobile telecom providers do not allow to access this information. Furthermore, the TCP protocol is not designed to support seamless handovers between intermittently connected devices at the edge of the network. To foster the practical relevance of CloudAware, our approach is based on the end-to-end architecture between mobile devices and surrogates and is independent from the internal architecture of cellular networks. Therefore, the assumed network model consists of a primary mobile device and a set of further mobile as well as fixed nodes that compose a temporary network by relying on different connections/protocols like LTE, Wifi or Bluetooth.

Context Adaptation: The main challenge in MCC is considered the offloading decision [35]. This problem is mainly related to the availability of other devices that qualify as surrogates. In MEC, this problem is further aggravated by the fact that the number and type of available devices changes frequently. Therefore, the offloading decision needs to incorporate various aspects, most important the mobility pattern of the user and the access medium that is used to offload a task as these are the main aspects that decide about the bandwidth and latency to the available surrogates [38]. Therefore, one of CloudAware's main features is the connectivity prediction, that enables the framework to decide whether the offloading to a specific surrogate will be successful. Similar problems like activity recognition or -prediction, connectivity- and link quality estimation and mobility- and location prediction via

mobility models have been examined thoroughly, since current mobile devices are able to acquire the applications', devices' and users' context through their physical (e. g. GPS or Wifi signal strength) and logical (e. g. call- and sms logs) sensors. Nevertheless, many of these approaches require long training phases, are not suitable for the limited resources of mobile devices and most of them focus on either specific datasets, sensors or algorithms, restricting their use to a rather limited set of scenarios. In contrast, we aim to develop a generic process to evaluate context information of different MEC applications to improve their offloading decision. We especially try to cover a broad range of scenarios and different subsets of available sensors, supporting the offloading decision with connectivity predictions even in situations suffering from the fact that only little, vague or no information is available to predict the future context.

Design Goals: Concluding the previous explanations and resuming the general objectives of MEC applications that have been worked out in Section II, the specific design goals of CloudAware can be summarized by the following key objectives:

- Speed up computation through parallelization
- Save energy or bandwidth by offloading computations
- Enable offloading for diverse mobility scenarios

This way, we are enabling (future) complex applications on resource-constrained mobile devices by dynamically adapting their execution to changing conditions in their physical and logical environment.

B. Runtime Environment

CloudAware relies on the "active components" middleware *Jadex* [39]. The active components paradigm, a combination of agent-, service-, and component-oriented engineering perspectives, supports different ways to realize CloudAware's constitutional components. The enabling execution environment Jadex has been chosen because it ideally complies with general requirements like distribution, concurrency and non-functional aspects of CloudAware's targeted application scenarios. Moreover, the Jadex middleware runs on desktop computers and servers as well as on a broad range of mobile devices, including the Android platform.

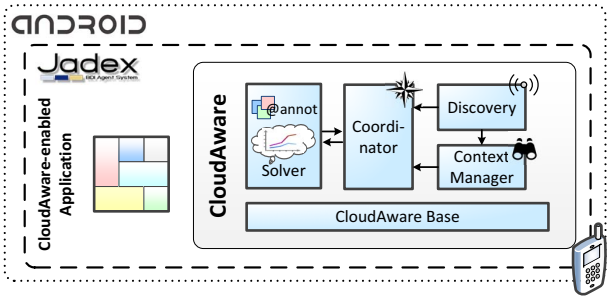

Fig. 3. CloudAware: Execution Platform and Architecture

Figure 3 depicts a bird's eye view on the resulting execution platform on mobile devices. As already mentioned, Jadex and hence CloudAware support the Android platform. While Jadex runs in an Android process, the CloudAware components are executed in several independent threads. Thereby, we can easily distribute load on multiple processor cores, while Jadex's asynchronous way of execution prevents deadlocks. Finally, MCC applications designed for CloudAware are running encapsulated in specialized CloudAware components, hence developers may just follow an object-oriented approach to implement their applications using the CloudAware API. In the following, we first introduce the wrapper components types that the developer has to deal with and afterwards detail the main services that CloudAware relies on.

C. Component Types

Ideally, the developer would just need to implement the business logic of his application and the underlying edge infrastructure would take care of an ideal partitioning, distribution, grouping, task assignment and scheduling. In reality, this is hardly feasible and often beneficial to let the developer group the business logic into self-contained components. This is why we encourage the developer to limit the interaction alternatives by a small rule set and three predefined component types that address the most common requirements we identified in existing sample applications:

The Stateless Component: Using Java-based components has several advantages, because they are very lightweight, easily deployable and because of the fact that they only require a Java virtual machine to be executed. A stateless (SL) component is mainly used for simple offloading tasks. It is possible to hold duplicates on different surrogates and decide, based on the specific offloading request and the current connectivity metrics like bandwidth, latency and the expected runtime together with a connectivity prediction for the specific surrogate which instance to use.
The SL component is suitable whenever the same global state can be recreated after a loss of connection by delegating the last service call to another instance of the same component, which is CloudAware's error handling for SL service calls. While implementing sample applications we found most of the components being stateless and we argue that they should be used whenever possible, as they allow a higher degree of freedom in the offloading decision.

The Stateful Component: Stateful (SF) components can be considered the special case of SL components. They come into play when a state needs to be preserved over multiple service calls. Here, a particular component type may either exist multiple times or can be restricted to a single running instance by a *singleton*-annotation. Depending on this annotation, in case of a connection loss, one either waits for a reconnection or the service call is redirected to another available copy. Therefore, SF singletons should be used carefully or be restricted to run on the user's primary mobile device itself via the *local*-annotation so that executability is ensured at all times.
Even if SF components are only created on surrogates which are likely to be reliable and reachable (this information is generated by the Context Manager, described later in this chapter), a connection loss needs to be handled appropriately. For SF components that implement the singleton pattern there is a further requirement: As long-running tasks might result

in a scenario where the connectivity to the formerly well-reachable SF component falls below a certain limit, a relocation can be necessary. As a simple object synchronization is not always possible (e.g., open database connections cannot be serialized) and often not efficient as well (e.g., often not every dependency needs to be transferred), the SF component that implements the singleton pattern requires a *migrate()*-method to be implemented to move the relevant state to a new instance on a different surrogate. During transfer, the CloudAware framework ensures that service calls are deferred. This allows to implement a broad range of business logic that can handle periods of disconnected operation without further modification. The SF component can as well be used to share a common state (e.g., shared information) among a set of users or provide a decentralized management among its users without a connection to the Internet (e.g., disaster recovery and the use of mobile base stations).

The Data Access Object: The third component type is the Data Access Object (DAO) that provides an efficient way of virtualized access to the local resources (e.g. sensors or storage) of the mobile device to even allow the offloading of SL- and SF components that rely on specific dependencies or capabilities of the mobile device.

D. CloudAware Components

The CloudAware framework as presented in Figure 3 consists of a runtime environment providing several components to allow optimized offloading decisions. Among these are a *Coordinator* to provide synchronization, a *Discovery Service*, a *Context Manager* providing a (future) connectivity status and a *Partitioner* as well as a *Solver* in order to create optimal offloading strategies. In the following, the main responsibilities of the components will be described briefly:

Discovery Service: It is the purpose of the discovery service to integrate the different interfaces (i.e. LTE, Wifi or Bluetooth) by spanning an overlay network. This overlay network is augmented with cost metrics like the current connection quality (i.e. latency and bandwidth) and further information about the available surrogates, their resources and current workload.

Partitioner: Partitioning mainly requires to estimate which parts of the global application state will need to be synchronized during the offloading, before parts of a mobile service can be executed on a surrogate. While some solutions rely on strong synchronization between a mobile device and cloud servers, others completely disregard this topic and leave it to the developer. We believe that the solution lies somewhere in between and that a good solution should combine the best of the two, depending on the current context. Because of this, we introduced the concept of self-contained wrapper components (SL, SF, and DAO). But this just solves part of the problem, as it remains the task of the Partitioner to decide about a grouping of components, which is at runtime used by the Solver to decide about the allocation strategy of components and the redirection of service calls – based on the expected interaction (e.g., method calls among each other and the amount of data exchanged). This process happens during the first executions and is repeated if the current measurements significantly deviate from the initially measured averages. Using these metrics, it will be decided by the Solver which components should reside on the same surrogate and how service calls should be made to and among them. Furthermore, these cost metrics help to decide whether to duplicate a shared application or assign it to one partition. Hence, it is the task of the developer to build self-contained units and let CloudAware group them into partitions with high cohesion.

Solver: Now that there is a set of components, enhanced by the Partitioner with metrics that indicate their interaction at runtime, it is the task of the Solver to decide for each service call which instances on what surrogates to use for offloading. This happens by calculating the cost that is derived from one of the following optimization targets: i) speed up computation, ii) save energy, iii) save bandwidth, iv) low latency, or v) use a balanced mode.
As mentioned in the design goals, it is CloudAware's main focus to support a broad range of mobility scenarios and heterogeneous devices to act as surrogates. This requires to design a robust and somehow elementary offloading logic, keeping in mind that devices that participate in the MEC scenario are constantly entering and leaving the temporary offloading space. To anticipate this aspect, complex dependencies between the devices should be avoided while still maintaining a high resource utilization. The topmost goal is therefore, to keep the dependency graph that exists in the temporary network of surrogates as small as possible with respect to the number of nodes and edges in order to be robust to connectivity losses. In CloudAware this task is implemented by the following rule: A mobile device can connect to multiple surrogates, but surrogates may only re-issue service calls if their connection is significantly more stable than to the mobile device.
The quality of the offloading decision is mainly limited by two factors: Solving an optimization problem that delivers the optimal allocation strategy for components to surrogates and the awareness about the current and future networks metrics. But as service calls vary in their runtimes and data usage based on the parameters passed with the service call, and as the complete confidence about the future network topology and metrics in mobile scenarios are unrealistic, trade-offs have to be made. As complete confidence is certainly not available anyway and solving an optimization problem on a mobile device is a computationally intensive task itself, that can easily overcompensate the expected savings [11], CloudAware uses an efficient minimum-effort strategy that always considers the expected savings by employing the following basic ruleset based on the average runtime (the chosen boundaries reflect the experiments with different sample applications and will be subject to adjustments in our future work):

- runtime less than 30 secs.: simple re-execution of the last successful offloading in terms of conformity to the optimization goal
- runtime less than 5 mins.: offloading strategy based on past execution metrics and reasonably up-to-date connectivity prediction
- longer runtime: new connectivity prediction for the timespan of the expected runtime and well-evaluated offloading strategy (amount of state required to be transfered)

This approach reflects the different gains, that result from

the dynamics of the application behavior and the fact that it is not required to forecast a timespan that excesses the expected runtime of the service call.

Context Manager: It is often argued that the consideration of the quickly changing context of mobile applications has a high potential on the applications service quality but requires the application to become context-aware [2]. Especially, to really benefit from their surrogates, MEC applications performing computation offloading need a certain awareness about the availability of their surrogates. More precisely, it is necessary for an MEC application to know about the connectivity and link quality to their surrogates, requiring information about the current and a prediction about the future connectivity states to them. But the classification of surrogates changes quite often, even during a single method call. This is why the Context Manager periodically re-evaluates the current quality of a surrogate. This re-evaluation poses a significant challenge, as in terms of providing a qualified decision whether and where to offload, a forecast of the device's connectivity and the bandwidth is required. Forecasting the mobile device's connectivity itself is complicated as it not only relies on the physical location but on many other factors, as for example other nearby devices using the shared medium WiFi as well. To perform this prediction we are currently developing a generic forecasting process that is able to cover a broad range of scenarios, even those where only limited information is available. We currently assume the following information, including a reasonably long history, to be relevant in this case:

- date, time, location, calendar events, charging y/n
- LTE/WiFi/Bluetooth on/off: bandwidth, ping, signal strength, discovered MAC addresses and GSM cells
- reasoned attributes: remaining time to stay at the same location, at the same WiFi AP or GSM cell, user is at home/work, traveling, moving, resting

This information is then used to predict how well a surrogate will be reachable, which is, among other things, the decision criterion whether it qualifies for the placement of components and the invocation of service calls. As soon as the connectivity is likely to change, this information needs to be re-evaluated. Deciding about how often to perform this energy-wise expensive task, is part of our future work.

Coordinator: Finally, it is the task of the Coordinator to merge available information regarding the available surrogates, the connectivity prediction and the Solver to execute a service call, based on the chosen optimization target (e.g., saving energy). Furthermore, the Coordinator is responsible for the error management. In situations where the offloading is likely to be unsuccessful or excesses the requested service level (e.g. "speed up computation") the service call is redirected to another surrogate or executed locally. Whereas on other service levels (e.g. "save energy") the amount of required energy is used to calculate the value (i.e. in terms of energy) of the result on the remote surrogate and whether it is beneficial to wait for a reconnect to receive the result. Additionally, the Coordinator takes care of the privacy issues by just allowing service calls to remote components that do not carry the "local"-annotation.

E. Use Case

A future application scenario can be envisioned by Jadex platforms that run parallel to other virtual machines on nearby devices to enable cloud gaming or other resource-intensive tasks. In practice, this means that Docker [40] containers can be used to bundle the required code fragments as well as the residual parts (e.g., libraries) of the applications' environment. This allows an easy build process and a quick deployment of lightweight application bundles as duplicate objects only need to be stored and copied to every surrogate only once to provide a minimal but self-contained execution environment. These bundles can then be deployed to up-to-date routers and base stations (see Section II) to allow MEC applications to elastically expand onto edge devices by accessing a component's copy on a surrogate as shown in Figure 4 while the offloading decision and its dependent tasks are handled by the CloudAware framework. This scenario allows low latency tasks like cloud gaming to benefit from surrogates located closer to the edge of the network.

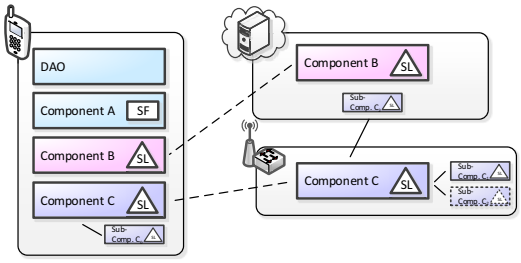

Fig. 4. CloudAware: Migration Strategies

VI. Conclusion

Edge computing enables to offload computation-intensive tasks from a user's mobile device to edge servers in order to speed up the execution, to make applications less dependent from the devices capabilities and the availability of centralized resources as well as to increase the overall user experience. But, there is neither an MEC framework, nor a cloudlet infrastructure available in the market yet, which we believe, is due to the lack of proper development support, insufficient context adaptation and the absence of "the app" relying on cloud augmentation.

Therefore, we presented CloudAware as a holistic approach to bond computation offloading and context adaptation. Following a component-based approach, expert knowledge of developers is used for guiding offloading decisions. Moreover, CloudAware is especially construed to deal with the effects of intermittent connectivity and to address the lack of proper context adaptation. In contrast to other related work in this area, the specific focus of this work is on computation offloading techniques that fit into the common development process of mobile applications as well as the developers' mindset.

Our prospects for the future are to provide a full evaluation in a realistic testbed. To show the wide applicability of our approach, we plan to simulate the offloading based on one of the biggest publicly available datasets containing trajectories and network availability metrics, namely the Nokia MDC

dataset [41]. Further open issues include caching concepts, safe-pointing and security features.

REFERENCES

[1] Canalys, "Smart phones overtake client pcs in 2011 — canalys," http://www.canalys.com/newsroom/smart-phones-overtake-client-pcs-2011, accessed 03.06.2015.

[2] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless Comm. and Mobile Computing*, vol. 13, no. 18, pp. 1587–1611, 2011.

[3] Cisco Systems Inc., "Cisco iox," https://developer.cisco.com/site/iox/, accessed 03.06.2015.

[4] Cisco Systems Inc, "Fog computing, ecosystem, architecture and applications - research at cisco," http://www.cisco.com/web/about/ac50/ac207/crc_new/university/RFP/rfp13078.html, 2013, accessed 03.06.2015.

[5] ETSI, "Mobile edge computing," http://www.etsi.org/technologies-clusters/technologies/mobile-edge-computing, 2015, accessed 03.06.2015.

[6] Cisco Systems Inc., "Cisco pushes iot analytics to the extreme edge with mist computing - rethink," http://rethinkresearch.biz/articles/cisco-pushes-iot-analytics-extreme-edge-mist-computing-2/, accessed 03.06.2015.

[7] M. Satyanarayanan, V. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *Pervasive Computing, IEEE*, vol. 8, no. 4, 2009.

[8] S. Choy, B. Wong, G. Simon, and C. Rosenberg, "A hybrid edge-cloud architecture for reducing on-demand gaming latency," *Multimedia Systems*, vol. 20, no. 5, 2014.

[9] Microsoft Inc., "Irides: Attaining quality, responsiveness and mobility for virtual reality head-mounted displays - microsoft research," http://research.microsoft.com/apps/video/default.aspx?id=246323, 2015, accessed 03.06.2015.

[10] G. Orsini, D. Bade, and W. Lamersdorf, "Cloudaware: Towards context-adaptive mobile cloud computing," in *IFIP/IEEE IM 2015: 7th Intern. Workshop on Management of the Future Internet (ManFI)*, 2015.

[11] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: Making smartphones last longer with code offload," in *ACM MobiSys 2010*, 2010.

[12] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "CloneCloud: elastic execution between mobile device and cloud," in *Proceedings of the 6. European Conference on Computer Systems*, 2011, pp. 301–314.

[13] S. Kosta, A. Aucinas, H. Pan, R. Mortier, and Z. Xinwen, "ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *IEEE Proc. INFOCOM*, 2012.

[14] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo: A computation offloading framework for smartphones," in *MobiCASE*, 2010.

[15] A. Davis, J. Parikh, and W. E. Weihl, "Edgecomputing: Extending enterprise applications to the edge of the internet," in *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, 2004.

[16] Nokia Siemens Networks, "Intelligent base stations," http://networks.nokia.com/sites/default/files/document/liquid_applications_wp.pdf, 2015, accessed 03.06.2015.

[17] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *In Proceedings of the 1. Edition of the MCC Workshop on Mobile Cloud Computing*, 2012, pp. 13–16.

[18] S. Yi, C. Li, and Q. Li, "A survey of fog computing: Concepts, applications and numbers," in *Mobidata'15 Workshop Subm.*, 2015.

[19] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: A survey," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 84–106, 2013.

[20] G. Orsini, D. Bade, and W. Lamersdorf, "Context-aware computation offloading for mobile cloud computing: Requirements analysis, survey and design guideline," in *Proc. of The 12th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2015)*, 2015.

[21] K. S. K. O. Sharifi, Mohsen, "A survey and taxonomy of cyber foraging of mobile devices," *IEEE Communications Surveys & Tutorials*, vol. 14, no. 4, pp. 1232–1243, 2012.

[22] M. Shiraz and A. Gani, "A lightweight active service migration framework for computational offloading in mobile cloud computing," *Journal of Supercomputing*, vol. 68, no. 2, pp. 978–995, May 2014.

[23] K. Hong, D. Lillethun, U. Ramachandran, B. Ottenwälder, and B. Koldehofe, "Mobile fog: A programming model for large–scale applications on the internet of things," in *Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing*. ACM, 2013.

[24] J. Porras, O. Riva, and M. D. Kristensen, "Dynamic resource management and cyber foraging," in *Middleware for Network Eccentric and Mobile Applications*. Springer, 2009, pp. 349–368.

[25] C. Shi, V. Lakafosis, M. H. Ammar, and E. W. Zegura, "Serendipity: Enabling remote computing among intermittently connected mobile devices," in *Proceedings of the thirteenth ACM international symposium on Mobile Ad Hoc Networking and Computing*. ACM Press, 2012.

[26] C. Shi, P. Pandurangan, K. Ni, J. Yang, M. Ammar, M. Naik, and E. Zegura, "IC-Cloud: Computation offloading to an intermittently-connected cloud," Georgia Institute of Technology, Tech. Rep. GT-CS-13-01, 2013.

[27] C. Shi, K. Habak, P. Pandurangan, M. Ammar, M. Naik, and E. Zegura, "Cosmos: Computation offloading as a service for mobile devices," in *Proceedings of the 15th ACM international symposium on Mobile ad hoc networking and computing*, 2014, pp. 287–296.

[28] F. Berg, F. Dürr, and K. Rothermel, "Optimal predictive code offloading," in *Proceedings of the 11th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, 2014.

[29] K. Lee and I. Shin, "User mobility-aware decision making for mobile computation offloading," in *IEEE 1st International Conference on Cyber-Physical Systems, Networks, and Applications (CPSNA)*, 2013.

[30] A. Nicholson and B. D. Noble, "Breadcrumbs: Forecasting mobile connectivity," in *Proceedings of the 14th ACM international conference on Mobile computing and networking*, 2008.

[31] A. Balasubramanian, R. Mahajan, and A. Venkataramani, "Augmenting mobile 3g using wifi: Measurement, system design, and implementation," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 2010.

[32] S. Scellato, M. Musolesi, C. Mascolo, V. Latora, and Andrew T. Campbell, "Nextplace: A spatio-temporal prediction framework for pervasive systems," *Pervasive Computing, IEEE*, 2011.

[33] T. Anagnostopoulos, C. Anagnostopoulos, and S. Hadjiefthymiades, "Mobility prediction based on machine learning," in *12th IEEE International Conference on Mobile Data Management*, 2011, pp. 27–30.

[34] T. Nguyen, H.-T. Cheng, P. Wu, S. Buthpitiya, and Y. Zhang, "Pnlum : System for prediction of next location for users with mobility," *Nokia Mobile Data Challenge Workshop*, vol. 2, no. 2, 2012.

[35] J. Flinn, *Cyber Foraging: Bridging Mobile and Cloud Computing*. Morgan & Claypool, 2012.

[36] I. Giurgiu, O. Riva, and G. Alonso, "Dynamic software deployment from clouds to mobile devices," in *Middleware*, ser. LNCS, vol. 7662. Springer, 2012, pp. 394–414.

[37] V. March, Y. Gu, E. Leonardi, G. Goh, M. Kirchberg, and B. S. Lee, "μcloud: Towards a new paradigm of rich mobile applications," *Procedia Computer Science*, vol. 5, no. 0, pp. 618 – 624, 2011.

[38] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Bitrate and video quality planning for mobile streaming scenarios using a gps-based bandwidth lookup service," in *2011 IEEE International Conference on Multimedia and Expo (ICME)*, 2011.

[39] A. Pokahr and L. Braubach, "The active components approach for distributed systems development," *International Journal of Parallel, Emergent and Distributed Systems*, 2013.

[40] I. Docker, "Docker - build, ship, and run any app, anywhere." [Online]. Available: https://www.docker.com/

[41] J. K. Laurila, D. Gatica-Perez, I. Aad, J. Blom, O. Bornet, Do, Trinh Minh Tri, O. Dousse, J. Eberle, and M. Miettinen, "From big smartphone data to worldwide research: The mobile data challenge," *Pervasive and Mobile Computing*, vol. 9, no. 6, pp. 752–771, 2013.