CloudAware: A Context-adaptive Middleware for Mobile Edge and Cloud Computing Applications

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*—The widespread use of mobile devices such as smartphones and tablets is accompanied by an ever increasing market for mobile applications, including resource demanding services like speech- or face recognition, that formerly were restricted to stationary devices. But as mobile devices remain comparatively limited in terms of resources (e.g., computation, storage and battery life), current approaches for augmentation have advocated the integration of cloud servers as well as other nearby devices to provide scalable computation- and storage resources to mobile end users. A current solution is the so-called computation offloading, which is the most prominent strategy used in Mobile Cloud Computing (MCC) and its successor known as Mobile Edge Computing (MEC). While MCC and MEC are receiving increasing attention, current work is often not able to cope with the quickly and constantly changing context (e.g., intermittent connectivity) of mobile devices. Therefore, this paper presents the evaluation of CloudAware, a context-adaptive mobile middleware for MCC as well as MEC that supports automated context adaptation by linking the distribution features of mobile middleware with context-aware self-adaptation techniques. In particular, we present a system software infrastructure and a data mining process which facilitate the development of elastic, scalable and context-adaptive mobile applications and present their evaluation using real usage data provided by the Nokia Mobile Data Challenge (MDC) dataset.**

I. Introduction

Mobile devices like smartphones, tablets and wearables are continuously replacing stationary devices. Along with this trend the porting of resource-hungry desktop applications and mobile-first applications like augmented reality games have led to a constant demand for higher performance and capabilities. Although some resources like processing power, memory and bandwidth are constantly improving to keep up with the increasing requirements, other features like battery capacity have only experienced moderate improvements of about 10% per year.

To overcome these limitations and to enable even more sophisticated applications to run on mobile devices, external resources have to be integrated into the local execution of mobile applications [1]. But, to ensure an acceptable performance and latency it is not always useful to only rely on centralized resource providers like cloud servers. It is often beneficial to dynamically integrate the surrounding and nearby devices, a concept known as *Fog Computing* [2] or *(Mobile) Edge Computing* [3]. However, these concepts are posing even bigger challenges to the mobile applications' capability to dynamically adapt to the constantly changing execution environment. Just to name a few challenges: The information about the current bandwidth to specific resources and a forecast about their future availability, the current battery level or the probability that an offloaded task will be successfully executed are all part of the applications' and devices' so-called context that needs to be integrated into the distribution strategy. The mentioned concept has led to an increasing demand for software that is able to exploit the potentials of spontaneous interaction and therefore needs to be able to dynamically adapt to the quickly and constantly changing context of mobile ad-hoc scenarios. At present, many of the proposed solutions provide only limited context-awareness and hence adaptation capabilities that moreover take current user preferences like saving energy or speeding up computations into account.

Filling this gap, this paper discusses the potential of self-adaptation in the light of the particular requirements of mobile edge computing. In previous work [4] the idea of a context-adaptive MEC solution has been presented, which is now extended to a self-adaptive mobile middleware named CloudAware that aims at linking existing concepts of mobile middleware with the specific requirements of MCC and MEC. To provide dynamic adaptation through a configuration-free programming model CloudAware employs compositional adaptation and sensor-based reasoning to allow a flexible adaptation to current as well as future context states, that can hardly be foreseen by developers. In particular, we use connectivity- and execution predictions that support the efficiency of the so-called offloading decision in MCC- as well as MEC scenarios. In this way, more generic and flexible scenarios that go beyond than just offloading computations become possible. To realize such scenarios and to support a broad range of mobile applications, CloudAware only relies on the presence of a Java Virtual Machine which enables our prototype to augment Android applications without modifying the underlying mobile operating system. The contribution in this paper can be summarized as follows:

- A flexible architecture for the development of MCC- and MEC applications that provides programming abstractions and distribution transparency features without modifying the underlying mobile operating system.
- An evaluation of the developed CloudAware mobile middleware that is based on realistic application and device usage data provided by the Nokia Mobile Data Challenge campaign.

The remainder of this paper is structured as follows: Section II introduces the foundations of MEC and self-adaptation. Afterwards, Section III describes typical application scenarios, whose general requirements are matched with the related work, presented in Section IV. Subsequently, our *CloudAware* middleware is presented and evaluated in Section V. At the end, we summarize our findings and give prospects for future work in Section VI.

II. Background

In this section we will first describe the concept of MEC and mobile middleware as an enabler for the greater idea of pervasive computing. Hereupon we will further give a definition of self-adaptation in the light of mobile applications and briefly describe its core enablers: computational reflection, dynamic reconfiguration and context awareness.

A. Mobile Edge & Pervasive Computing

While Mobile Cloud Computing tries to push the limits of mobile applications by including centralized resources to e.g. perform computational offloading, Mobile Edge Computing goes further by assigning the major part of remote operations directly to the surrounding infrastructure. These resources, typically located at the logical edges of a network, can include LTE base stations, routers providing shared resources [5] and are often directly connected to the mobile device, as shown in Figure 1. Deploying replicated parts of a mobile application's business logic onto such edge computers can then bring a large benefit to latency-sensitive tasks like streaming or cloud gaming. Hence we consider MEC as a main enabler for the more generic concept of Pervasive Computing, described next.

Pervasive and Ubiquitous Computing aim at integrating computing capabilities into our everyday life. In this course, smart objects sense their environment and communicate and cooperate with each other in order to adapt to their surrounding's needs [6]. The seamless integration of such (generally) resource-poor and possibly mobile devices, which are in most cases located at the logical edge of a network, and their tight cooperation can be achieved by considering the current state of a device, its user and the environment, referred to as the three main categories of the so-called "context". But due to the possible heterogeneity of context some kind of abstraction that provides a standardized execution environment, referred to as a (mobile) middleware, is required. In the remainder of this paper, we will refer to all of the mentioned concepts by MEC only.

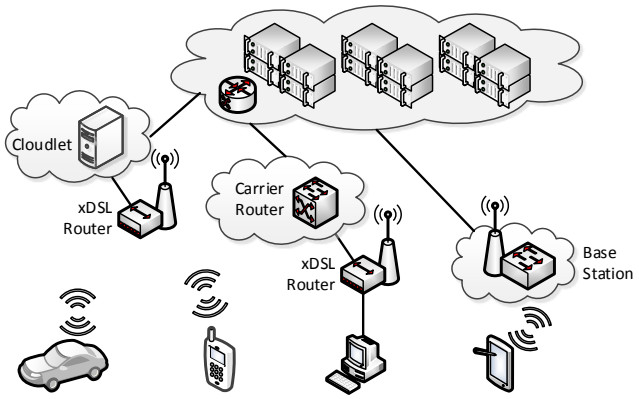

Fig. 1. MEC architecture and use-case [4]

B. Self-Adaptation

We consider an early definition applicable to our perception of self-adaptation as referred by Laddaga in [7]: *"Self-adaptive software evaluates its own behavior and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible."* A well-known example for self-adaptation can be found in the implementation of the Transmission Control Protocol (TCP) of the Internet, which dynamically adapts the packet window size to the current load in the network. Current patterns for self-adaptation can be briefly differentiated in two classes [8]: Parametric adaptation and compositional adaptation. While the adaptation rules in the former are often woven into the business logic, compositional adaptation in contrast follows the paradigm of the separation of concerns. Taking into consideration the simple example of three parameters of which each can have 4 states - already 64 different states are possible and need to be considered by the developer. Considering real-life examples that have way more parameters and states, as also found by [9], it becomes clear that compositional adaptation provides a far more flexible concept to develop self-adaptive applications - whose three main enablers, according to [10], are the separation of concerns, computational reflection, and component-based design. According to [9], self-adaptive systems can be further differentiated by the level of anticipation they provide. If a developer predefines the systems behavior, the degree of anticipation is lower than in the case where no rules are defined and the anticipation to the current context happens dynamically at runtime, referred to as context-awareness. For a detailed definition and a complete survey about the most relevant context modeling approaches, we refer to [11].

III. Application Characteristics

To further illustrate the idea of MEC, an application scenario is exemplified in the following that serves as a base for the evaluation presented in section V Subsequently, the generalized main characteristic of MEC applications are presented.

Image Processing: As all current smartphones have a built-in camera, mobile image processing gains more attraction. And with the advent of high-resolution cameras (e.g. Nokia's Lumia with 41 Megapixel) respective tasks becomes even more challenging. Moreover, also professionals, journalists for example, use mobile phone cameras to take snapshots, which subsequently undergo typical processing steps like noise reduction, color enhancements, object extraction, masking, etc. before being published. Finally, more complex tasks like image stitching to create 360 degree panorama images or face recognition raise the bar even higher. Imagine you are on a trip abroad and you need to process some images and send them back to your office or your friends. Mobile communication is costly, but your smartphone's battery should still last a couple of hours. Depending on your preferences, you could either substitute communication with processing or vice versa, using an MEC image processing app.

Further common application scenarios share the exemplified characteristics: 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* (that depict more powerful resource compared to the mobile device) at the network's edge where they are processed and results are finally sent back to the initiating device.

IV. Related Work

A lot of research on context-aware software has been carried out to find mechanisms that best support dynamic adaptation for specific use-cases. A current trend that follows the separation of concerns principle, is aspect oriented programming (AOP) [12] was designed to provide a separation of concerns, as advocated by many researchers in the field [13]. AOP separates the business logic of an application from additional features (like context-aware behavior) by adding additional behavior to existing code without modifying the code itself. This is often helpful in order to dynamically react to changes (and for example replace parts of the code to change the behavior). This way, the concerns of context-awareness and self-adaptation are treated independently of each other. As the developers may now focus on individual aspects of their code, development and maintenance tasks are rendered easier, faster, and more cost efficient [13]. As an evolution and directly tailored to the domain of self-adaptation, context oriented programming (COP) [13] has been proposed as an extension to AOP by providing additional features concerning the separation of the definition of adaptations from their composition. Nevertheless AOP and COP require the developer to learn a different language, which we do not consider easy and quick in terms of a learning curve. The COP principle has been implemented as an extension to many programming languages (i.e., ContextJ (as an extension to Java), or ContextPy (as an extension to Python)), a complete list of all COP variants is found at [13].

Notable works that are similar to CloudAware have been presented by Preuveneers and Barbers [14]. They present a context-aware middleware for mobile devices that is component-based and self-adaptive. Nevertheless, there are no programming-level concepts that ease the development. Another prominent approach that aims at providing a self-adaptive middleware is presented by Geihs et al. In MADAM [15], and later in MUSIC [16], applications are assembled through a component composition process and context-awareness is achieved by exchanging the components' implementations with others having the same functional behavior. Still, adaptation rules need to be defined by the developer (e.g. using annotations). Further well-known solutions include PACE [17], SOCAM [18] and CAMPUS [19] whereas the last one provides the most current approach of an automatized reasoning by using the applications' context. However, it can be concluded that in most context-aware systems, the adaptation logic of an application is implemented at the time of development. Such an approach has limited flexibility and poses a high burden for developers as it is almost impossible to foresee all conceivable context states, especially in mobile environments with a quickly changing context.

For the sake of completeness we also highlight some prominent solutions from the domain of MCC and MEC and their respective adaptation features. For a complete review of related solutions in this field we refer to [20] and [1]. From the MEC perspective, classical approaches like *CloneCloud* [21], *MAUI* [22], *Thinkair* [23] and *Cuckoo* [24] have tried to adapt to the environment 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 shown 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 like *IC-Cloud* [25] and *ToGo* [26] have been proposed that take into consideration the effects of intermittent connectivity, but only provide adaptation features regarding the connectivity of the mobile device, which is an important but still a small part of the applications', devices' and users' context.

To conclude, 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 CloudAware mobile middleware, a holistic approach to tackle the challenges of both domains: computation offloading and context adaptation.

V. CloudAware

CloudAware differs from previous or similar approaches in the domain of MEC and context-adaptive mobile middleware 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 a 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, or prominent MCC approaches are often not suitable as they are not able to either 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 has been described in previous work [4] and will in the following only be summarized, while the focus of this work is the evaluation of the prototype of the CloudAware mobile middleware that is evaluated using real data from the Nokia MDC dataset, introduced afterwards.

A. CloudAware Mobile Middleware

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 uses the principle of "active components" a concept developed together with the *Jadex* [27] middleware. Jadex not only ships with several tools to ease the development (like debugging, message inspection, simulation), but also supports - among other features - diverse asynchronous communication styles (remote method calls, transparent service invocations, message passing and interaction protocols) as well as secure messaging, an efficient binary message encoding, the creation of ad-hoc networks, service discovery and NAT traversal over relay servers to bypass firewalls. Moreover, the Jadex middleware runs on desktop computers and servers as well as on a broad range of mobile devices, including the Android platform as it requires no modification of the mobile devices' operating system, but just relies on the presence of a Java Virtual Machine that is available out of the box for almost any device.

While carrying out our design decisions we rely on the simple principle to build as much as possible upon widespread commodity software, to leverage the reliability that comes through extensive testing. Hence, we employ standard Java and Android technology, extended by the Jadex runtime environment that we use to implement the functionalities required in CloudAware to perform MEC. For example, to separate front-end user interaction from the offloadable back-end tasks, we make use of Android's service concept by using the AIDL (Android Interface Definition Language) specification. This allows developers to easily benefit from the offloading capabilities, while changes to their present applications remain small and are limited to the back-end of the application. Furthermore, this allows the coexistence of regular and cloud-augmented applications as they participate in the standard Android mechanisms like resource scheduling and power management. Hence, no modification of the Android operating system is necessary, enabling the majority of current smartphones to directly participate from the benefits of CloudAware.

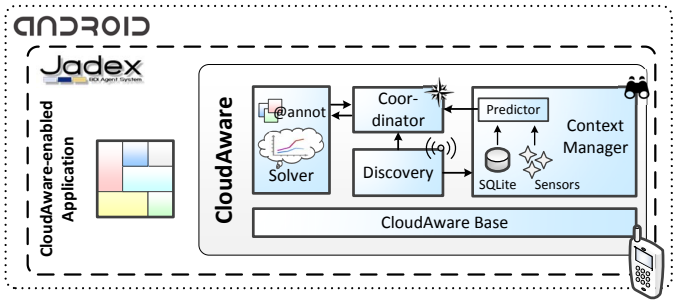

Fig. 2. CloudAware: Execution Platform and Architecture [28]

Figure 2 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.

C. Nokia MDC Dataset

In January 2009, the Nokia Research Center Lausanne (NRC), the Idiap Research Institute, and the EPFL initiated the creation of a large-scale mobile data research. This included the design and implementation of the Lausanne Data Collection Campaign (LDCC), an effort to collect sensor data from smartphones created by almost 200 volunteers in the Lake Geneva region over 18 months [29]. To our knowledge it is the largest dataset that contains information about mobile devices as well as application usage statistics, which is why we chose the Nokia MDC dataset to derive the following information as input to a simulation with the CloudAware mobile middleware:

- GSM/WiFi/Bluetooth state (on/off), discovered MAC addresses and GSM cells, signal strength of WiFi as well as GSM cells, extended with our own measurements to get an assumption about the available bandwidth.
- General information about the mobile device itself: time since the last user interaction, silent mode switch, charging state, battery level, free memory.
- Date, time, location, calendar events, average and predicted remaining duration of stay at the current location.
- Reasoned attributes: remaining duration of stay at the same WiFi AP or GSM cell, user is at home/work, traveling, moving, resting.
- Application usage data: The applications the user interacts with and the specific screens of these applications.

D. Generic Context Adaptation Process

One of CloudAwares main features are context adaptation features that have been described in previous work [30] as the *Generic Context Adaptation* (GCA) process and will in the following be briefly summarized. GCA has been designed as a lightweight data mining process that is tailored to forecast arbitrary context attributes out of a provided feature set. More formally, GCA represents a data mining process that provides three types of predictions: A binary classifier (e.g. for predicting the future availability of a WiFi), a multi-class classifier (e.g. to predict a bandwidth range) and a regression mode to forecast real-valued attributes like the execution time of an offloaded task. This way, historical data that has been collected by the mobile device can be used to forecast a future value of a certain context attribute.

To perform a prediction the GCA process needs to be provided with the historical input data in the structure that is used in the Nokia MDC dataset. It is furthermore required to choose the context attribute to be predicted, the context interval (that is required to assign different measurements to a specific time interval) and the forecast horizons (how many

time intervals into the future) to initiate the learning phase. The resulting predictive model can then directly be applied to forecast the chosen context attribute. The data used in the GCA process is mainly the same as in the MDC and is just converted to a representation that is suitable to be presented to machine-learning algorithms.

To be used together with CloudAware, the GCA process is designed to be executed on a mobile device. As the preprocessing is performed in SQL and the application of the data mining models is performed in Java, the process is able to run on any (mobile) that provides a Java virtual machine. Nevertheless, the training of the data mining models is computationally intensive and is currently intended to run on more powerful cloud resources, whereupon the trained model is transferred to the mobile device, to be used for predictions. We currently use an interval of 5 minutes to collect the required sensor data to not drain the battery too much.

E. Evaluation

The evaluation is performed by picking 20 users that provided data for at least 18 months from the Nokia MDC dataset and by using the provided context attributes to design an event based simulation that considers the connectivity (i.e. latency, intermittent connectivity and round trip times) of a mobile device as well as the battery drain, charging state, the limited computation power of a mobile device as well as the energy that is required to compute and to send or receive a specific amount of data via a specific interface (WiFi or GSM). The developed sample application is inspired by the application scenario mentioned in section III and applies different image filters. It produces 20 different types of tasks. For each task the execution time, its variance, as well as input and output sizes (i.e. the amount of data that needs to be transferred in case of an offloading) have been measured and linked to real application events, that have been recorded in the MDC dataset.

Simulation of the infrastructure: The used infrastructure consists of a Samsung Galaxy S5, representing the mobile device. The devices battery level found in the MDC dataset is considered a baseline, while the prototyped sample application generates an additional battery drain, therefore all tasks could only successfully executed if the mobile device would be connected to an energy source at all times.

The cloud is simulated by measurements from and to a virtual server running at a German cloud service provider. To reflect the MEC scenario we furthermore assume cloudlets with the performance of an up-to-date desktop computer to be available at the three most frequently visited locations of each user. Furthermore, due to their limited resources, these cloudlets are assumed to be overloaded at certain times of the day, resulting in longer execution times or even timeouts that lead to unsuccessful offloading.

Scheduling and optimization goal: For each of the offloadable tasks, it is decided whether to offload this task to a cloud-server or a cloudlet by predicting its probability to be executed successfully, meaning that the result is returned to the mobile device. Therefore we consider all tasks equally important and therefore execute them by their invocation time, with the exception that already computed tasks may return

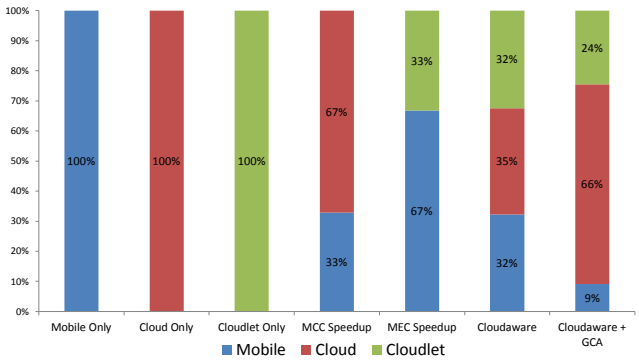

Fig. 3. Offloading targets for tasks

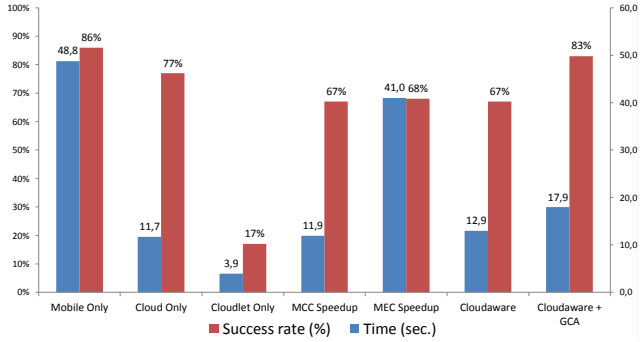

Fig. 4. Success rates and execution times

to the mobile device with a higher priority. The offloading decision is furthermore influenced by choosing the alternative with the minimum invocation time and if equal the lowest energy consumption.

Description of evaluated scenarios: Mobile Only: All tasks are being executed on the mobile device, the success rate is 86% as an empty battery state prevented to execute all tasks. **Cloud Only and Cloudlet Only:** All tasks are being executed on the respective surrogate. **MCC Speedup and MEC Speedup:** Tasks are being assigned to the respective surrogate or the mobile device based on the aforementioned scheduling strategy. **CloudAware:** Tasks are being assigned to the the cloud, the cloudlet or the mobile device based on the aforementioned scheduling strategy. **CloudAware supported by GCA:** Same as CloudAware but the context manager of the CloudAware middleware provides predictions about the probability for a specific task to be successfully executed on a surrogate and furthermore predicts the expected time to execute. This information is used to decide about the offloading target.

TABLE I. SIMULATION RESULTS

	Energy (kJ)	Transfer (GB)	Time (s)	Success (%)
Mobile Only	4928	0,0	48,8	86
Cloud Only	7474	17,1	11,7	77
Cloudlet Only	2483	6,0	3,9	17
MCC Speedup	7018	12,8	11,9	67
MEC Speedup	5516	5,7	41,0	68
CloudAware	6300	11,3	12,9	67
CloudAware + GCA	6003	13,0	17,9	83

Discussion: Table 1 reflects the results of our simulation, together with Figure 3 and 4 it can be seen that already the baseline version of CloudAware provides an speedup of 276% while executing 67% of the tasks. Using the GCA-supported version of CloudAware further reduces the average energy consumption about 15% while it is the only scenario maintaining a similar success rate (83 %) compared with local execution.

VI. Conclusion

Despite context-awareness in mobile middleware being a well-investigated feature, available solutions are often domain specific. Furthermore, the requirement for additional configuration prevents their immediate use in the domain of MEC where a quick, efficient and dynamic context-adaptation is necessary in order to even anticipate future context scenarios that are unforeseeable by the developer.

Consequently, we presented CloudAware as a holistic approach to bond computation offloading and context adaptation, where we combine the requirements and benefits of mobile clouds together with the broader and more generic capabilities of mobile middleware, while relying on a strictly generic model and employing compositional adaptation enabling a highly automated adaptation process. We evaluated the CloudAware mobile middleware based on a real world smartphone application and realistic context data provided by the Nokia MDC dataset and demonstrated that this scenario can benefit to a considerable degree from anticipation of future context states. While a plethora of work on designing context aware mobile services exists, the goal of this paper was the demonstration of a generic approach to support the most relevant mobile-cloud-interation scenarios through context-adaptation.

Acknowledgment: Parts of the research in this paper used the MDC Database made available by Idiap Research Institute.

References

[1] 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.

[2] Cisco Systems Inc, "Fog computing, ecosystem, architecture and applications - research at cisco," http://research.cisco.com/research\#rfp-2013078, 2013, accessed 05.04.2016.

[3] ETSI, "Mobile edge computing," http://www.etsi.org/technologies-clusters/technologies/mobile-edge-computing, 2015, accessed 05.04.2016.

[4] G. Orsini, D. Bade, and W. Lamersdorf, "Computing at the mobile edge: Designing elastic android applications for computation offloading," in *8th Joint IFIP Wireless and Mobile Networking Conference (WMNC)*. IEEE Explore Washington/DC, USA, 10 2015, p. 8.

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

[6] M. Satyanarayanan, "Pervasive computing: vision and challenges," *Personal Communications, IEEE*, vol. 8, no. 4, pp. 10–17, 2001.

[7] R. Laddaga and P. Robertson, "Self adaptive software: A position paper," in *SELF-STAR: International Workshop on Self-* Properties in Complex Information Systems*, vol. 31. Citeseer, 2004, p. 19.

[8] K. Kakousis, N. Paspallis, and G. A. Papadopoulos, "A survey of software adaptation in mobile and ubiquitous computing," *Enterprise Information Systems*, vol. 4, no. 4, pp. 355–389, 2010.

[9] K. Geihs, "Selbst-adaptive software," *Informatik-Spektrum*, vol. 31, no. 2, pp. 133–145, 2008.

[10] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. Cheng, "Composing adaptive software," *Computer*, 2004.

[11] T. Strang and C. Linnhoff-Popien, "A context modeling survey," in *Workshop Proceedings*, 2004.

[12] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin, "Aspect-oriented programming," in *ECOOP'97Object-oriented programming*. Springer, 1997.

[13] M. Appeltauer, R. Hirschfeld, M. Haupt, J. Lincke, and M. Perscheid, "A comparison of context-oriented programming languages," in *International Workshop on Context-Oriented Programming*. ACM, 2009.

[14] D. Preuveneers and Y. Berbers, "Towards context-aware and resource-driven self-adaptation for mobile handheld applications," in *Proceedings of the 2007 ACM symposium on Applied computing*. ACM, 2007.

[15] M. Mikalsen, N. Paspallis, J. Floch, E. Stav, G. A. Papadopoulos, and A. Chimaris, "Distributed context management in a mobility and adaptation enabling middleware (madam)," in *Proceedings of the 2006 ACM symposium on Applied computing*. ACM, 2006, pp. 733–734.

[16] R. Rouvoy, P. Barone, Y. Ding, F. Eliassen, S. Hallsteinsen, J. Lorenzo, A. Mamelli, and U. Scholz, "Music: Middleware support for self-adaptation in ubiquitous and service-oriented environments," in *Software engineering for self-adaptive systems*. Springer, 2009.

[17] K. Henricksen, J. Indulska, T. McFadden, and S. Balasubramaniam, "Middleware for distributed context-aware systems," in *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*. Springer, 2005, pp. 846–863.

[18] T. Gu, H. K. Pung, and D. Q. Zhang, "A middleware for building context-aware mobile services," in *Vehicular Technology Conference, 2004. VTC 2004-Spring. 2004 IEEE 59th*, vol. 5. IEEE, 2004.

[19] E. J. Wei and A. T. Chan, "Campus: A middleware for automated context-aware adaptation decision making at run time," *Pervasive and Mobile Computing*, vol. 9, no. 1, pp. 35–56, 2013.

[20] 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.

[21] 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.

[22] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 2010, pp. 49–62.

[23] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2012, pp. 945–953.

[24] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo: a computation offloading framework for smartphones," in *Mobile Computing, Applications, and Services*. Springer, 2010, pp. 59–79.

[25] 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.

[26] 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.

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

[28] 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.

[29] 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.

[30] G. Orsini, D. Bade, and W. Lamersdorf, "Generic context adaptation for mobile cloud computing environments," in *Proceedings of The 13th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2016)*, ser. Procedia Computer Science. Elsevier Science, 8 2016.