2014 IEEE Fourth International Conference on Big Data and Cloud Computing

Context-Aware Data Prefetching in Mobile Service Environments

Waldemar Hummer, Stefan Schulte, Philipp Hoenisch, Schahram Dustdar
Distributed Systems Group, Vienna University of Technology, Austria
Email: {lastname}@dsg.tuwien.ac.at

***Abstract*—Mobile environments, such as vehicular communication systems (VCSs), are typically subjected to network fluctuations and intermittent downtimes, e.g., if service consumers operate in a tunnel or switch between cells of an ISP. In this work, we present an approach for service and data prefetching from the Cloud, which allows to ensure continuous service delivery and consistent quality of experience (QoE). We leverage the fact that most applications have typical access patterns, for instance streaming, or polling in regular intervals. In our system model, we consider the context under which the consumer is currently executing, including time, location, and projected route (e.g., known from GPS navigation). Based on projections for network quality at future locations, we propose a decision problem for optimizing data prefetching and continuous QoE, and discuss different mechanisms for generating service requests for prefetching. We thoroughly evaluate our approach based on a popular data set of vehicular GPS traces in Switzerland, which we deploy and simulate in a Cloud environment. In our experiments we compare prefetching approaches and address different aspects, including successful and unsuccessful invocations, prefetching hits and misses, as well as age and usage of prefetched results.**

I. Introduction

The rise of Cloud Computing [1] has opened unprecedented possibilities for highly dynamic, scalable, and cost-efficient applications. Today, the worldwide public Cloud services market is worth 131 Billion US dollars in 2013, and the annual growth rate is estimated to be 17.4% from 2011 to 2017[1]. In parallel, smart devices like smartphones are omnipresent and a major driver in the ICT domain: From 2013 to 2018, the compound annual growth rate of mobile data traffic is forecast as 61% and by 2016, traffic from wireless devices will exceed traffic from wired devices. In 2013, smart devices have already generated 62% of total mobile data traffic, and it is estimated that there will be more mobile-connected devices than people on earth by the end of 2014. In 2013 alone, more than 400 million smartphones were added to this pool [3].

Not surprisingly, we see a convergence of mobile and Cloud Computing, which manifests itself (amongst others) in two research fields: First, researchers are investigating how the power of Cloud offerings can be utilized for hybrid environments with smart devices, for instance to offload computation in order to save energy or for speed up purposes [4]. Second, mobile devices are more and more becoming the primary user interface to Cloud services of all kinds, ranging from media streaming to general data services located in the Cloud [5].

The convergence of mobile and Cloud computing leads to both new opportunities and challenges which need to be addressed by the research community. One central problem in mobile Cloud computing is that Cloud services can only be accessed efficiently if the end user's device is connected to a network in a reliable and fault-tolerant manner: consuming Cloud data sources from a mobile device, by nature, is sensitive to interruptions. Even a connectivity loss of a few seconds could lead to a drastic reduction of the quality of experience (QoE). Especially if mobile devices change their position rapidly and/or constantly, e.g., because the user is moving in a vehicle, the QoE may fluctuate heavily.

Hence, it is imperative to prefetch data and buffer it locally in order to mask connection losses or bad connection quality [6]. Prefetching is the technique of querying or gathering any kind of data or service functionality before the moment that they are actually needed or used. It is related to caching, not only because caching and prefetching are often combined, but also since the prefetched data have to be stored in caches or similar modules. Similar to caching, prefetching can be used in different domains and with different goals. Apart from the fact that prefetching may facilitate uninterrupted service consumption in the first place, it is also an approach to achieve lower user-perceived latency [7].

Despite the fact that prefetching is a well-studied and established technique, to the best of our knowledge, there is a lack of approaches explicitly aiming at prefetching data from the Cloud on mobile devices (see Section VI). Hence, in this paper, we conceptualize and implement a corresponding data prefetching solution, taking into account the specific demands of Cloud services and mobile users. The prefetching algorithm which is at the core of our approach relies on context data (most importantly the user's location) in order to provide reliable and personalized prefetching decisions. We will show that context information like the current and future location of the user is not only helpful to achieve prefetching in general, but also helpful to decrease the user-perceived latency.

The remainder of the paper is structured as follows. First, in Section II we introduce an illustrative scenario which motivates the research problem. Section III introduces the assumed system model and details the proposed approach for data prefetching. Section IV discusses implementation details, and the approach is thoroughly evaluated in Section V. In Section VI, we comment on the related work. Finally, Section VII concludes the paper with outlook for future work.

[1] For an overview on different estimates of the size of the Cloud market, we refer to [2].

978-1-4799-6719-3/14 $31.00 © 2014 IEEE
DOI 10.1109/BDCloud.2014.104

IEEE computer society

II. Scenario

We consider a scenario from the road user information system (RUIS) domain, which is developed within the SIMPLI-CITY[2] research project. The project aims at providing a framework for integrating heterogeneous Cloud services which contribute to the users' driving experience. We assume that the RUIS is running on a mobile device, e.g., a car or smartphone.

A. Characteristics of Scenario Services

Table I contains an exemplary list of considered services, along with their key characteristics. Service s_1 provides important updates such as upcoming traffic jams, s_2 performs re-routing if the car gets off track, s_3 shows important landmarks and events in the vicinity of the current location, s_4 performs media streaming (e.g., music streaming like *Spotify*), s_5 allows to fetch Email and instant messages (IM), and s_6 gathers usage statistics and synchronizes updates with the Cloud-based server. Currently, we consider mainly services consumed by (human) end users; in the future we also plan to integrate advanced scenarios with machine-to-machine (M2M) communication, e.g., mobile stream processing applications [8].

TABLE I
Exemplary Services in a Road User Information System

Service	**Importance**	**Time Criticality**	**Access Pattern**	**(Pre-)Fetching Strategy**
s_1: Traffic Updates	high	high	push / poll	timely updates
s_2: (Re-)Routing	high	medium	on demand	precompute routes
s_3: Vicinity Info	medium	medium	recurrent	pre-load for route
s_4: Media Stream	medium	low	continuous	pre-load & cache
s_5: Mail and IM	medium	high	polling	timely updates
s_6: Stats & Sync	low	low	recurrent	postpone if required

Each service is associated with the level of importance (encoded as high/medium/low), the time criticality (whether the time of service execution has a crucial impact on the delivered functionality), the typical access pattern (e.g., streaming, or polling in regular intervals), as well as possible strategies for prefetching. For instance, s_1 is considered highly important and its information is highly time-critical (e.g., traffic jams); contrarily, s_4 is considered less important but also its time criticality is low, hence it is well-suited for prefetching. Services with high time criticality are generally hard to prefetch entirely. One solution is "timely updates", a strategy where the service is called immediately before an expected network outage, in order to maximize freshness of the data. Other possible strategies include preloading, precomputing, or postponing (e.g., for services with low importance).

B. Network Quality and Required Service Prefetching

Figure 1 illustrates a simplified scenario with three mobile devices (m_1, m_2, m_3) connected via an Internet Service Provider (ISP) to a set of backend services in a Cloud environment. Note that we primarily investigate the communication between devices and the backend services, i.e., car-to-car communication is not in our focus (see Section VI).

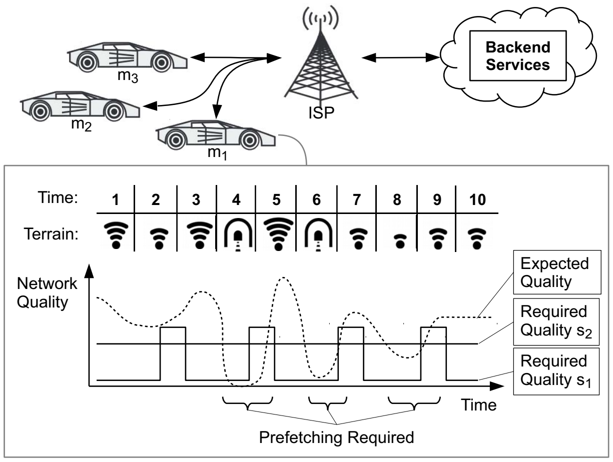

Fig. 1. RUIS Scenario With Fluctuating Network Quality

The lower part of the figure summarizes the terrain, network characteristics, and service usage patterns of device m_1 over time. The terrain information reflects tunnels along the road with typically no network connectivity (time points 4 and 6), as well as different signal strengths (e.g., low signal at time 8, excellent signal at point 5). We assume the current location and planned route of the device as given (using GPS navigation). Given a GPS location, it is possible to determine the expected network quality; these data are either provided by the ISP, or available from a variety of "speedtest" pages on the Web [9].

We observe that there are in particular three points on the time axis where the expected network quality is below the required level (around time points 4, 6, and 8). Since our goal is to guarantee continuous service delivery, these situations are considered as the main motivation for the need for service prefetching. Note, however, that prefetching is not only required on the mobile consumer side, but also plays a role if the context of the Cloud backend services, e.g., to reduce user-perceived latency or hedge against (temporary) downtimes when using external third-party services.

III. Service Prefetching Approach

This section presents our approach for data prefetching in mobile service environments. We first introduce the detailed system model in Section III-A, then outline a decision problem for scheduling of prefetching in Section III-B, and finally discuss different variants for generating and executing prefetching requests in Section III-C.

A. System Model

The core artifacts of the assumed system model are listed in Table II. Where applicable, an example with reference to the scenario in Section II is provided. Note that $\mathcal{P}(X)$ denotes the *powerset* of a given set X. We use the notation $x[i]$ to refer to the ith item of a tuple x, whereas $idx(j, x)$ gives the (one-based) index of the first occurrence of item j in tuple x. Moreover, $X^{\mathbb{N}} := \bigcup_{n \in \mathbb{N}} X^n$ denotes the set of all tuples (with any length) over the set X.

[2] http://www.simpli-city.eu/

TABLE II
SYSTEM MODEL

Symbol	Description	Example
S	Set of services	$S = \{s_1, \ldots, s_6\}$
$R \in S \times I$	Set of concrete service requests from service input domain I	$R = \{r_1 = (s_1,\ loc = (47.1, 10.2)), \ldots\}$
M	Set of mobile service consumers, e.g., devices in vehicles	$M = \{m_1, m_2, m_3\}$
$i : S \rightarrow \mathbb{R}^+$	Importance of a service, from subordinate (0) to critical (1)	$i(s_1) = 1, i(s_2) = 0.7$
$c : S \rightarrow \mathbb{R}^+$	Time criticality of a service, from low (0) to high (1)	$c(s_1) = 1, c(s_2) = 0.8$
C	Domain of consumers' context information, e.g., time/location	-
$T \subset C$	Domain of time	$\{t = 1, \ldots, t = 10\}$
$L \subset C$	Domain of location, e.g., encoded as GPS latitude/longitude	$\{l = (47.1, 10.2), \ldots\}$
$t : C \rightarrow T$	Get time encoded in a context	$t(\{t = 1\}) = 1$
$e : M \times C \rightarrow C^{\mathbb{N}}$	(Predicted) evolution of a consumer's context over time	$e(m_1, \{t = 1\}) = (\{t = 2, ...\}, \{t = 3, ...\}, ...)$
$r : S \times M \times C \rightarrow \mathcal{P}(R)$	Concrete service requests to be issued under certain contexts	$r(s_1, m_1, \{t = 4\}) = \{r_1\}$
$q_a : C \rightarrow \mathbb{R}^+$	Available network quality under a given context	$q_a(\{l = (47.1, 10.2)\}) = 0.0$
$q_r : R \times M \times C \rightarrow \mathbb{R}^+$	Required network quality for a request under a given context	$q_r(r_1, m_1, \{t = 4\}) = 0.6$
$p : R \times M \times C \rightarrow T$	Prefetching time scheduled for a request in a given context	$p(r_1, m_1, \{t = 3\}) = 3$
$q : M \times T \rightarrow R^{\mathbb{N}}$	Queue with currently scheduled requests at a given time	$q(m_1, 3) = (r_1)$

The model contains a set of services (S) which are used by different mobile consumers (M). The set of service requests (R) represents the domain of concrete invocations issued for a service. Functions i and c determine the importance and time criticality of services, respectively; they allow to give precedence to services that are critical with respect to prefetching. Each consumer is associated with a context (C) that changes over time (e.g., future path of a vehicle). The domains of time (T) and location (L) are also encoded in the context. Function e expresses how the context is going to evolve over the (near) future. This prediction is important to make prefetching decisions, discussed later in this section.

The available network quality at a given location is expressed via function q_a. Our primary means for assessment is the data transfer rate of the cellular network, but q_a may also combine aspects such as latency or packet drop rate. As noted in Section II, real(istic) values for q_a can be obtained from publicly available data by providers and users. Additionally, cost aspects can be expressed in q_a, e.g., if data roaming is disabled then q_a drops to 0 as soon as the consumer passes a country border. The required network quality for a service is expressed in function q_r, which is currently derived from user-defined access patterns; our future work involves automatic refinement of q_r using monitoring and data mining techniques.

Finally, function p in the system model defines the scheduled time at which data prefetching should be performed, and q represents a consumer's priority queue of requests that are currently scheduled for prefetching.

Based on the available information encoded in the system model, the core problems are (1) to decide whether and for which points in time prefetching should be scheduled, and (2) to define which concrete service requests should be used to perform prefetching. Problem (1) is discussed in Section III-B, and problem (2) is discussed in Section III-C.

B. Prefetch Scheduling Strategies

The service result availability function ($a : R \times M \times C \rightarrow Bool$) in Equation 1 specifies under which conditions a service result is available under a given context: if either (1) the available network quality is sufficient to make the service invocation at the time it is required, or (2) the service request had been prefetched before.

$$\begin{aligned} a(r_x, m_x, c_x) :=\, & q_a(c_x) > q_r(r_x, m_x, c_x) \vee \\ & \exists c_y \in C : p(r_x, m_x, c_y) = true \end{aligned} \tag{1}$$

Overall, we aim for the goal that all Cloud service data is available whenever requested, expressed in Equation 2. This implies that all requests which are scheduled for a time where the network is unavailable (or the network quality is insufficient) need to be prefetched.

$$\begin{aligned} & \forall_{r_x \in R, m_x \in M, c_x \in C} : \\ & (q_r(r_x, m_x, c_x) > 0) \Longrightarrow (a(r_x, m_x, c_x) = true) \end{aligned} \tag{2}$$

As soft constraints for optimizing the prefetch scheduling, we require that more time-critical services are requested at the latest possible time (Equation 3) and that precedence is given to more important service requests (Equation 4).

$$\sum_{\substack{s_x \in S, m_x \in M, \\ c_x \in C, \\ r_x \in r(s_x, m_x, c_x)}} (t(c_x) - p(r_x, m_x, c_x)) * c(s_x) \rightarrow \min! \tag{3}$$

$$\begin{aligned} & \forall m_x \in M, t_x \in T, r_1 \in q(m_x, t_x), r_2 \in q(m_x, t_x) : \\ & i(r_1[1]) > i(r_2[1]) \Longrightarrow idx(r_1) < idx(r_2) \end{aligned} \tag{4}$$

The prefetching mechanism can be tailored to the importance and the time criticality of Cloud services, based on the level of context information available to make the decision. We distinguish between two basic prefetching strategies, *periodic* and *context-aware*, which are discussed in the following paragraphs.

1) Periodic Prefetching Strategy: The basic type of prefetching is to periodically invoke the target service in fixed or predefined time intervals, denoted t_i. Figure 2 shows a timeline where red crosses (e.g., time points t_5, t_6) indicate failed invocations due to insufficient network connectivity ($q_a < q_r$). Periodic prefetching is not well suited for services with high time criticality ($c(s_x) \rightarrow 1$), because prefetchings may be scheduled too early (i.e., could be scheduled closer to the time when the result is actually needed, see result age in the figure). Moreover, this strategy is sub-optimal with respect to network usage, since it may perform unnecessary prefetch invocations (i.e., results which are never used, see t_1, t_2). Yet, it can be applied when the context evolution (e) is not known in advance, i.e., there is no information about how the context and in particular the network quality are going to evolve in the future (e.g., planned route of the vehicle is not available).

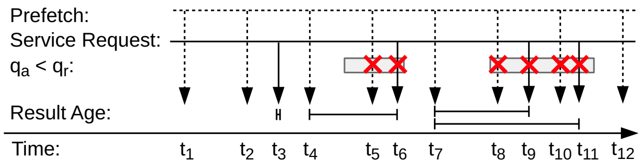

Fig. 2. Periodic Prefetching

2) Context-Aware Prefetching Strategy: With context-aware prefetching, we assume that the evolution of the user context, or at least a relevant portion of it, is known in advance. In future work we plan to automatically derive sophisticated predictions for service usage patterns and contexts evolution, possibly integrating existing work on predicting violations of service level agreements [10]. The future context information is used to reveal upcoming problems in connectivity ($q_a < q_r$), and to prefetch in a timely manner. Figure 3 illustrates that by tendency less requests are required and the result age is reduced. Moreover, note that this strategy allows to create context-specific requests. In contrast to Figure 2, where the same service results are used at t_9 and t_{11}, we now prefetch two results at t_4 and t_5, which anticipate the concrete contexts at time t_6 and t_7, respectively. Details follow in Section III-C.

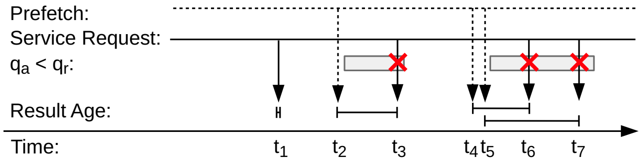

Fig. 3. Context-Aware Prefetching

The high-level scheduling procedure for context-aware prefetching (see Algorithm 1) is discussed in the following.

Algorithm 1 Context-Aware Prefetch Scheduling

```
Require: consumer $m_x \in M$, context $c_{cur} \in C$, timespan $t_p$
 1: $contexts \Leftarrow e(m_x, c_{cur})$ in range before $t(c_{cur}) + t_p$
 2: while $contexts \neq ()$ do
 3:     $c_{next} \Leftarrow contexts[1]$
 4:     $requests \Leftarrow \bigcup_{s_x \in S} r(s_x, m_x, c_{next})$
 5:     $q_{req} \Leftarrow \sum_{r_x \in R} q_r(r_x, m_x, c_{next})$
 6:     if $q_a(c_{next}) < q_{req}$ then
 7:         wait until current time reaches $t(c_{next}) - t_{prefetch}$
 8:         $requests \Leftarrow$ sort $requests$ by importance
 9:         add $requests$ to queue ($q$)
10:     end if
11:     wait until $c_{next}$ becomes current context
12:     $c_{cur} \Leftarrow c_{next}$
13:     $contexts \Leftarrow e(m_x, c_{cur})$ in range before $t(c_{cur}) + t_p$
14: end while
```

Given a service consumer m_x and current context c_{cur}, the algorithm repeatedly projects the next context c_{next} (line 3) within a given future timespan t_p (lines 1 and 13), and generates requests (line 4) to determine the required network quality (line 5). If we anticipate that the quality under context c_{next} will be insufficient (line 6), then we initiate prefetching by sorting the requests by importance (line 8) and adding the requests to the queue (line 9). Note that line 8 satisfies the criterion in Equation 3. Moreover, using an estimate for the duration to execute the requests ($t_{prefetch}$) the prefetching is scheduled for the latest possible time (line 7), which is a useful heuristic to approximate the criterion in Equation 4.

C. Prefetch Request Generation

Based on the scheduling for prefetching (Section III-B), we now discuss different mechanisms for generating prefetching requests. In general, to prefetch a service for time t_2 at time $t_1 < t_2$ requires to anticipate the concrete invocation that will be requested at t_2. We distinguish between constant requests (Section III-C1), template-based requests (Section III-C2), and complex request patterns (Section III-C3).

1) Constant Requests (Polling): Constant requests are independent of (client-side) context information. Figure III-C1 illustrates a periodic request pattern with a constant request to retrieve new email messages.

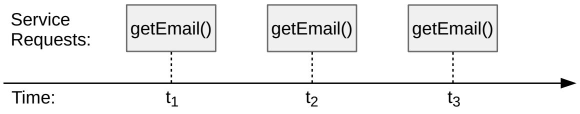

Fig. 4. Polling an Email Service

2) Template-Based Requests: If request messages depend on the context under which they are issued, we use a template-based approach. In Figure 5, the template for the *getTrafficInfo* request contains a placeholder for the current location (indicated with double curly braces "{{}}"), which is replaced by the location parameter (l) of the associated context. This approach allows to prefetch the exact service data which will be required in the future, provided that context-aware prefetching (see Section III-B2) is possible.

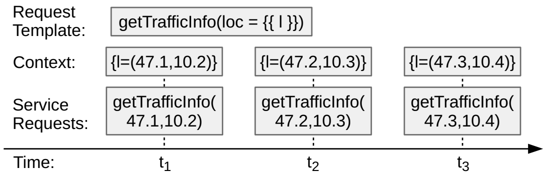
Request Template:
getTrafficInfo(loc = {{ l }})
Context:
{l=(47.1,10.2)}
{l=(47.2,10.3)}
{l=(47.3,10.4)}
Service Requests:
getTrafficInfo(47.1,10.2)
getTrafficInfo(47.2,10.3)
getTrafficInfo(47.3,10.4)
Time:
t1
t2
t3

Fig. 5. Template-Based Requests for a Traffic Info Service

3) Complex Request Patterns: Some service interactions require more complex patterns, which also need to be considered for prefetching. For instance, Figure 6 illustrates the generation of requests for the Media Stream service. The context includes the progress (*progress*) of the currently playing song (*curSong*). Assume that music titles are divided into chunks, and the chunks of the next song (*nextSong*) are requested when the progress of the current song reaches a certain threshold (e.g, at 1% and 51%). To support such complex request patterns, a simple template-based or other declarative approach is not sufficient. Hence, the request generation logic has to be defined programmatically, e.g., using code statements, a rule engine, or similar. We are currently working towards a domain-specific language (DSL) to simplify this task.

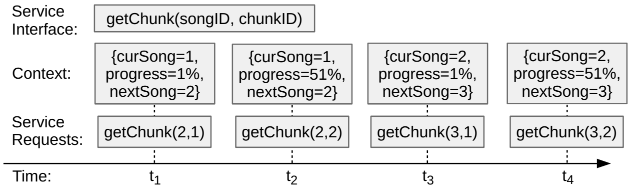
Service Interface:
getChunk(songID, chunkID)
Context:
{curSong=1, progress=1%, nextSong=2}
{curSong=1, progress=51%, nextSong=2}
{curSong=2, progress=1%, nextSong=3}
{curSong=2, progress=51%, nextSong=3}
Service Requests:
getChunk(2,1)
getChunk(2,2)
getChunk(3,1)
getChunk(3,2)
Time:
t1
t2
t3
t4

Fig. 6. Complex Requests for a Media/Music Service

IV. Implementation

We have implemented the presented approach in a Java prototype, which is available open-source[3] (including the evaluation data, see Section V). The coarse-grained architecture is depicted in Figure 7. The implementation is embedded in the OSGi (Open Service Gateway initiative)-based SIMPLICITY runtime environment for mobile Cloud services. The *prefetching manager* is currently integrated as a client-side component; in future, we plan to provide a reliable deployment mechanism [11] to host it as a migratable Cloud service.

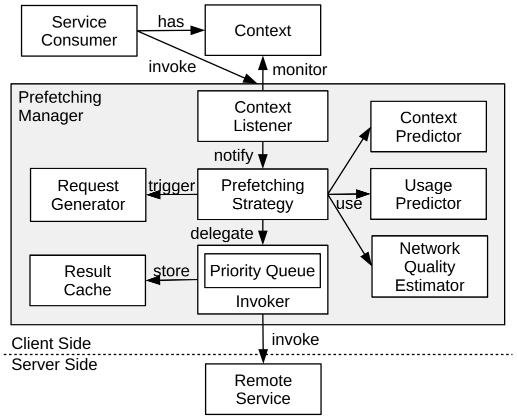

Fig. 7. Prefetching Client Architecture Overview

Each service consumer is associated with a *context*, which holds data such as the current time and location, the future route of the vehicle, or the configuration of running services (e.g., a music title currently playing in the media streaming service). Whenever the context changes, the *context listener* informs the *prefetching strategy*, which makes its decisions based on the *context predictor* (predicting the context evolution function e), the *usage predictor* (how services will be requested in the future) and the *network quality predictor* (which network coverage will be available in the future). The *request generator* creates new service invocation requests, which are executed by the *invoker* and stored in the *result cache*. Later, when a service consumer attempts to invoke a service which is not reachable, the prefetched result is loaded from this cache.

Network quality prediction works on the basis of cellular network coverage maps, which are readily available from most telecommunications providers. Given the predicted future location of a device, an estimation of the connection quality can be determined, depending on the available connection type(s) (e.g., 2G/3G/4G). In the evaluation (see Section V), we require information about network quality in Switzerland. We utilize the mobile network coverage service from telecommunications provider *Swisscom*[4], which is provided as a graphical overlay for Google maps. Since there is no computer-readable API for this service, we automatically fetch and parse the overlay map images to determine the network quality function $q : L \rightarrow \mathbb{R}^+$. In addition, we query the OpenStreetMap API[5] for nodes tagged *"tunnel=yes"*, since we can expect that connectivity is limited or unavailable in some tunnels.

V. Evaluation

To evaluate our approach, we have set up a comprehensive experimentation based on a popular data set of vehicular traces[6], introduced in [12]. The trace data set simulates a multitude of cars driving across Switzerland, based on GPS locations and routes of real road maps.

A. Evaluation Setup

We have extracted traces of 50 moving cars (i.e., mobile devices, $M = \{m_1, \ldots, m_{50}\}$) from the data set, resulting in 13135 data points with different time and location information. Since the accuracy of the data points does not match our requirements (average distance of 79.9 seconds between each two data points), we interpolated the simulation between the data points and move forward in our simulation in steps of 10 seconds, resulting in a total of 109248 time points. Moreover, there are some large gaps in the data (1 hour and more between two points), which we eliminated during preprocessing.

All tests were executed in a private Cloud environment, running *OpenStack* on top of machines with two Intel Xeon quad-core CPUs, 32 GB RAM, Linux kernel 3.0.0-16. Clients (simulating the moving devices) and services (hosting the required functionality) are deployed on distributed hosts. The evaluation occupied our hardware for roughly two hours.

We assume that the clients use the scenario services from Section II with different access patterns. Table III lists for each service the importance, time criticality, and required network quality q_r, measured using the assumed data transfer rate (in kbps). These values are rough approximations, and for simplicity we do not distinguish download and upload speeds.

[3] https://github.com/whummer/service-prefetching

[4] http://scmplc.begasoft.ch/plcapp/pages/gis/netzabdeckung.jsf

[5] http://api.openstreetmap.org/

[6] http://www.lst.inf.ethz.ch/research/ad-hoc/car-traces/

TABLE III
ACCESS PATTERNS AND TRANSFER RATES FOR SCENARIO SERVICES

Service (s_x)	Access Pattern	$i(s_x)$	$c(s_x)$	q_r
s_1: Traffic Updates	fetch every 60 secs	1.0	1.0	50
s_2: (Re-)Routing	initiate every 300 secs	1.0	0.7	75
s_3: Vicinity Info	query when location changes	0.7	0.6	100
s_4: Media Stream	retrieve every 20 secs	0.8	0.3	150
s_5: Mail and IM	sync every 180 secs	0.6	0.9	50
s_6: Stats & Sync	upload every 600 secs	0.2	0.2	100

Figure 8 compares the projected service data usage against the available network quality for an exemplary service consumer $m_1 \in M$. For simplicity, the quality is measured as data rate (kbps): the required quality is derived from the combined values of q_r for all services (see Table III), and the available quality q_a corresponds to the theoretical maximum speed of different cellular connections based on the Swisscom coverage map (we assume 2G=150kbps, 3G/UMTS=384kbps, 3G/HSPA=5.76mbps, 4G=50mbps).

Note that our approach does not rely on these values being entirely accurate; typically it is mainly relevant whether there is any connectivity (e.g., no connectivity around time point 12000 in the figure) or at least 3G/UMTS, because data rates on the high end of the spectrum are hardly reached with the services considered here (note the logarithmic scale on the y-axis). To get more realistic values, regional data from "speed test" pages could be integrated.

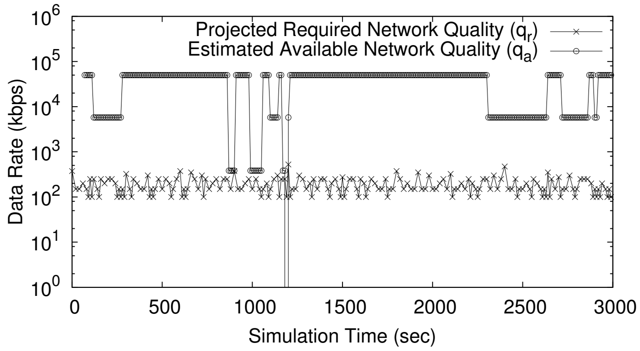

Fig. 8. Exemplary Data Usage and Availability for Consumer m_1

B. Evaluation Results

During execution of the experiment, we simulated network downtimes (based on the Swisscom coverage map and OpenStreetMap data), and the service invoker reported whenever a service request could not be sent. Figure 9 reports the aggregated numbers for service invocations which were possible when requested ($q_a >= q_r$) and service requests which required prefetching ($q_a < q_r$). We observe that most requests can be performed normally, but around 1000 of 40000 requests required prefetching. Note, however, that this value may be higher in a real setting if there are intermittent downtimes caused by fluctuations in the cellular network.

Next, we investigate the freshness of data, i.e., the age of results caused by service prefetching. Figure 10 depicts a box plot with the age of prefetched results, aggregated over periods

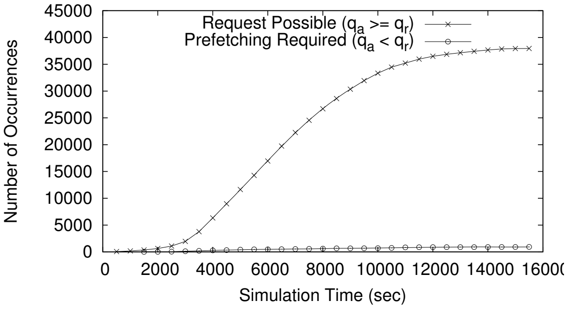

Fig. 9. Sufficient Network Quality versus Prefetching Required

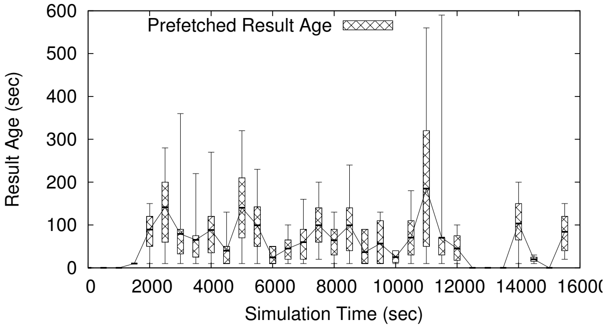

Fig. 10. Age of Results with Context-Based Prefetching

of 500 seconds along the timeline in the graph. Evidently, the longer a car is offline, the older the results. The longest streaks of offline activity are due to cars leaving the Swiss country territory where Swisscom has no network coverage (result age = 560 seconds), and driving through the 17km long *Gotthard* road tunnel (result age = 590 seconds).

Related to the age of results, we evaluate the accuracy of prefetching in terms of whether results are available or not. Similar to misses in caching, we speak of a *prefetching miss* if a service request $r_x \in R$ cannot be issued ($q_a < q_r$) and there is no prefetched result available. Prefetch misses can be either caused by unanticipated events (e.g., temporary network outage), or if the algorithm does not consider contexts sufficiently long into the future (timespan t_p, see Algorithm 1). Figure 11 analyzes the prefetch misses for different times of

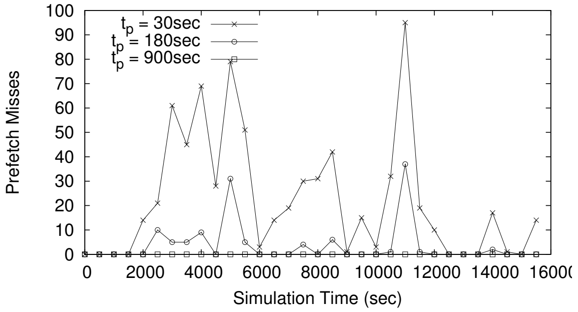

Fig. 11. Prefetch Misses for Different Prediction Timespans (t_p)

t_p (30, 180, and 900 seconds). While there is a considerable amount of misses for $t_p = 30$, there are effectively no prefetch misses for $t_p = 900$, because this value is greater than the longest period that requires prefetching in our scenario (driving through Gotthard tunnel, 682 seconds).

Finally, we take a closer look at comparing periodic with context-based prefetching, considering the result age. Figure 12 shows the accumulated sum of all result ages over time, for different prefetching strategies (each projecting $t_p = 900$ seconds into the future). The lowest numbers are achieved when using context-based prefetching. For periodic prefetching, the results depend on the update interval (t_i, see Section III-B1).

However, the benefits (result age) of low update intervals (t_i) in periodic prefetching are offset by the disadvantage of unused results, due to the fact that results are repeatedly prefetched but only the most recent result is actually used by the consumer. As illustrated in Figure 13, unused results are considerably higher (max. 84 versus max. 460) if the update interval is reduced (450 vs. 90).

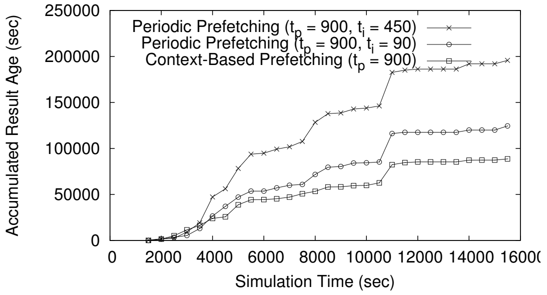

Fig. 12. Result Age for Periodic and Context-Based Prefetching

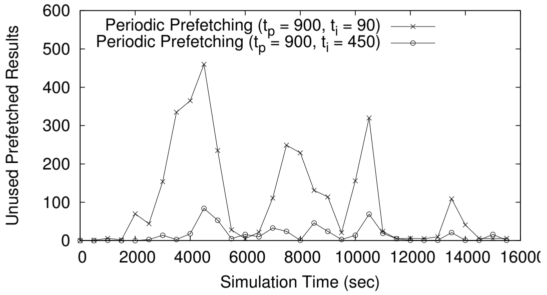

Fig. 13. Unused Prefetched Results for Different Update Intervals

VI. Related Work

Prefetching has been observed in different areas of computer science for a long time, e.g., file systems or databases [13], media streaming [14], [15], or WWW latency [16], [17], [18]. However, to the best of our knowledge, research on data prefetching for mobile applications as discussed in this paper is still in its infancy and the number of sophisticated approaches is rather small [6]. Notably, we are not aware of any approach which addresses data from the Cloud to mobile devices.

In a seminal approach, Cao presents power-aware proactive caching for mobile devices, which is actually prefetching [19]. However, the author assumes a broadcasting scenario, which is not in line with the system model applied here. Also, the location of mobile users does not influence the prefetching.

Schreiber et al. propose a prefetching mechanism for mobile access to services hosted in Service-oriented Architectures (SOA) [20]. While in our work prefetching is controlled by the client, they apply a proxy server-based approach. Interestingly, prefetching is done using a piggybacking approach. This minimizes additional connections between proxy and client and therefore enhanced power efficiency. The actual prefetching control is based on a sequence prediction algorithm and therefore best-suited for known workflows. There are further examples which apply a dual prefetching approach for Web services, i.e., on the client *and* server/proxy side, e.g., [21], [22], [23]. These approaches do not come into question for the scenario at hand, since we assume that the proxy/server side is not available at certain points of time and therefore prefetching has to be done at the client side.

Parate et al. [24] provide a prefetching approach for mobile apps, based on estimations of the app(s) to be used next. Depending on the prediction, data for these apps are prefetched. Context information like the future location of the user, which we use to derive the actual need for prefetching, is not explicitly taken into account. However, the authors apply coarse-grained location information, e.g., “at home”, “workplace”.

Higgins et al. [6] present the *Informed Mobile Prefetching* (IMP) approach, which is based on the three dimensions performance, energy usage, and data consumption. In contrast to our work, their focus is mostly on performance (in terms of user-perceived latency) optimization for devices with limited power. Correspondingly, the need for prefetching in the IMP system model results from a latency gain, while in our work, we focus on actual data provision as the primary goal of prefetching. Moreover, IMP assumes that exact predictive values for network bandwidth, latency etc. are available, whereas we mainly build on (GPS) location information to predict contexts with low or zero connectivity (e.g., tunnels). Nevertheless, IMP comes closest to the work at hand.

In the field of Internet access from cars, the research focus is primarily on improvement of vehicular WiFi access, e.g., [25], or between cars in *Vehicular Ad Hoc Networks* (VANets), e.g., [26], [27]. In contrast, data prefetching has only been applied rarely: Siris and Kalyvas introduce a solution to prefetch data on roadside WiFi hotspots for later downloads by passing vehicles [28]. In this work, the authors observe the problem from a different angle, i.e., the one of an infrastructure provider who wants to reduce costs by minimizing 3G/4G traffic. Prefetching is done based on user requests from the mobile device: After a request has been received, the data is prefetched on WiFi hotspots that the user will encounter in the near future. Wu et al. take into account prefetching in a very specific case, i.e., cooperative media streaming in mobile environments [29]. Similar to [28], prefetching is done at (WiFi-based) streaming access points the user may likely encounter in the near future. Both approaches use location information to estimate the need for prefetching.

While the number of context-aware prefetching approaches is rather small, it should be noted that context awareness has been a major topic in Service-oriented Computing in recent years [30]. In the field of mobile services, context awareness has been primarily applied to adapt service protocols based on the context (e.g., [7]) or to personalize the outcome of a service based on the information needs of the user (e.g., [31]). Such solutions are orthogonal to the work at hand and could be combined in the future to further improve the user experience.

VII. Conclusion

Whereas prefetching has been elaborated in other areas in the past, this work is among the first to study prefetching on the level of Cloud-based mobile services. Our context-based approach anticipates foreseeable service delivery problems and prefetches results in a timely manner, in order to deliver consistent Quality of Experience (QoE). The paper discusses and thoroughly evaluates periodic prefetching (useful if context predictions are limited) and context-based prefetching (ideal to optimize data freshness and network usage). Based on an illustrative scenario, we demonstrate how concrete prefetching requests are generated, from simple polling, to template-based requests and complex request patterns. Our evaluation, which combines realistic GPS car traces with cellular network coverage maps and OpenStreetMap data, provides an in-depth analysis of the experimentation data and reveals the strengths and weaknesses of different prefetching variants.

Some limitations remain, which we tackle as part of our future work. In our extended evaluations, we strive to obtain more detailed insights concerning prefetching under faults [32] (e.g., intermittent network outages) or other irregularities (e.g., deviation from the projected contexts). In addition, the current approach does not distinguish in detail between different data types (e.g., volatile or stable) and does not take into account that users may share data among different application contexts, which may open opportunities for further optimization. Moreover, we aim to apply data prefetching for event-based applications within the Internet of Things (IoT). One of the core challenges in this field is related to optimal positioning [33] of prefetching services among multiple interacting devices.

Acknowledgements

This work is partially supported by the European Union within the SIMPLI-CITY FP7-ICT project (Grant agreement no. 318201).

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin *et al.*, "A view of cloud computing," *Comm. of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[2] D. Castro, *How Much will PRISM Cost the U.S. Cloud Computing Industry?* The Inf. Techn. & Innov. Foundation (ITIF), August 2013.

[3] "Cisco Visual Networking Index: Forecast and Methodology," 2014.

[4] N. Fernando, S. Loke, and W. Rahayu, "Mobile cloud computing: A survey," *Future Gen. Comp. Syst.*, vol. 29, no. 1, pp. 84–106, 2013.

[5] I. Giurgiu, O. Riva, D. Juric, I. Krivulev, and G. Alonso, "Calling the Cloud: Enabling Mobile Phones as Interfaces to Cloud Applications," in *10th Int. ACM/IFIP/USENIX Middleware Conf.*, 2009, pp. 83–102.

[6] B. D. Higgins, J. Flinn, T. J. Giuli, B. Noble, C. Peplin, and D. Watson, "Informed mobile prefetching," in *10th Int. Conf. on Mobile Systems, Applications, and Services (MobiSys)*, 2012, pp. 155–168.

[7] A. Papageorgiou, A. Miede, S. Schulte, D. Schuller, and R. Steinmetz, "Decision support for web service adaptation," *Pervasive and Mobile Computing*, vol. 12, pp. 197–213, 2014.

[8] W. Hummer, B. Satzger, and S. Dustdar, "Elastic Stream Processing in the Cloud," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 3, no. 5, pp. 333–345, 2013.

[9] S. Wallsten, "Understanding international broadband comparisons," *Available at SSRN 1136831*, 2008.

[10] P. Leitner, J. Ferner, W. Hummer, and S. Dustdar, "Data-driven and automated prediction of service level agreement violations in service compositions," *Distributed and Parallel Databases*, vol. 31, no. 3, 2013.

[11] W. Hummer, F. Rosenberg, F. Oliveira, and T. Eilam, "Testing Idempotence for Infrastructure as Code," in *Middleware Conference*, 2013.

[12] V. Naumov, R. Baumann, and T. Gross, "An evaluation of inter-vehicle ad hoc networks based on realistic vehicular traces," in *7th ACM Int. Symp. on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2006.

[13] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka, "Informed prefetching and caching," in *15th ACM Symp. on Operating Systems Principles (SOSP)*, 1995, pp. 79–95.

[14] F. H. P. Fitzek and M. Reisslein, "A prefetching protocol for continuous media streaming in wireless environments," *IEEE J. on Selected Areas in Comm.*, vol. 19, no. 10, pp. 2015–2028, 2001.

[15] S. Bagchi, "A fuzzy algorithm for dynamically adaptive multimedia streaming," *ACM Trans. on Multimedia Computing*, vol. 7, no. 2, pp. 11:1–11:26, 2011.

[16] V. N. Padmanabhan and J. C. Mogul, "Using predictive prefetching to improve world wide web latency," *SIGCOMM Computing Communication Review*, vol. 26, no. 3, pp. 22–36, 1996.

[17] Z. Jiang and L. Kleinrock, "Web Prefetching in a Mobile Environment," *IEEE Personal Communications*, vol. 5, no. 5, pp. 25–34, 1998.

[18] J. Domènech, A. Pont, J. Sahuquillo, and J. A. Gil, "A user-focused evaluation of web prefetching algorithms." *Computer Communications*, vol. 30, no. 10, pp. 2213–2224, 2007.

[19] G. Cao, "Proactive power-aware cache management for mobile computing systems," *IEEE Trans. on Computers*, vol. 51, no. 7, 2002.

[20] D. Schreiber, A. Göb, E. Aitenbichler, and M. Mühlhäuser, "Reducing user perceived latency with a proactive prefetching middleware for mobile SOA access," *Int. J. of Web Service Res.*, vol. 8, no. 1, 2011.

[21] X. Liu and R. Deters, "An Efficient Dual Caching Strategy for Web Service-Enabled PDAs," in *ACM Symp. on Applied Computing*, 2007.

[22] M. S. Qaiser, P. Bodorik, and D. N. Jutla, "Differential Caches for Web Services in Mobile Environments." in *IEEE Int. Conf. on Web Services (ICWS)*, 2011, pp. 644–651.

[23] N. D. R. Armstrong and P. A. S. Ward, "Just-In-Time Push Prefetching: Accelerating the Mobile Web," in *27th IEEE Int. Conf. on Advanced Information Networking and Applications (AINA)*, 2013, pp. 1064–1071.

[24] A. Parate, M. Böhmer, D. Chu, D. Ganesan, and B. M. Marlin, "Practical prediction and prefetch for faster access to applications on mobile phones," in *ACM UbiComp'13 Conference*, 2013, pp. 275–284.

[25] J. Eriksson, H. Balakrishnan, and S. Madden, "Cabernet: vehicular content delivery using wifi," in *14th Ann. Int. Conf. on Mobile Computing and Networking (MOBICOM)*. ACM, 2008, pp. 199–210.

[26] H. Hartenstein and K. Laberteaux, "A tutorial survey on vehicular ad hoc networks," *IEEE Communications*, vol. 46, no. 6, pp. 164–171, 2008.

[27] W. Chen, R. K. Guha, T. J. Kwon, J. Lee, and Y. Hsu, "A survey and challenges in routing and data dissemination in vehicular ad hoc networks." *Wireless Comm. and Mobile Comp.*, vol. 11, no. 7, 2011.

[28] V. A. Siris and D. Kalyvas, "Enhancing mobile data offloading with mobility prediction and prefetching," *Mobile Computing and Communications Review*, vol. 17, no. 1, pp. 22–29, 2013.

[29] S. Wu, J. Hsu, and C.-M. Chen, "Headlight Prefetching and Dynamic Chaining for Cooperative Media Streaming in Mobile Environments," *IEEE Trans. on Mobile Computing*, vol. 8, pp. 173–187, 2009.

[30] H. L. Truong and S. Dustdar, "A Survey on Context-aware Web Service Systems," *Int. J. of Web Information Systems*, vol. 5, no. 1, 2009.

[31] D. Gavalas and M. Kenteris, "A web-based pervasive recommendation system for mobile tourist guides," *Personal and Ubiquitous Computing*, vol. 15, no. 7, pp. 759–770, 2011.

[32] W. Hummer, C. Inzinger, P. Leitner, B. Satzger, and S. Dustdar, "Deriving a unified fault taxonomy for event-based systems," in *6th Int. Conf. on Distributed Event-Based Systems*, 2012, pp. 167–178.

[33] W. Hummer, P. Leitner, B. Satzger, and S. Dustdar, "Dynamic Migration of Processing Elements for Optimized Query Execution in Event-based Systems," in *OnTheMove Conferences*, 2011, pp. 451–468.