vStore: A Context-Aware Framework for Mobile Micro-Storage at the Edge

Julien Gedeon[1], Nicolás Himmelmann[1], Patrick Felka[2], Fabian Herrlich[1], Michael Stein[1], and Max Mühlhäuser[1]

[1] Telecooperation Lab
Department of Computer Science
Technische Universität Darmstadt, Germany
gedeon@tk.tu-darmstadt.de
https://www.tk.informatik.tu-darmstadt.de
[2] Institute for Information Systems
Faculty of Economics and Business Administration
Goethe-Universität Frankfurt am Main, Germany
felka@wiwi.uni-frankfurt.de
https://www.wiim.uni-frankfurt.de

Summary. The way mobile users store and share their data today is completely decoupled from their current usage context and actual intentions. Furthermore, the paradigm of cloud computing, where all data is placed in distant cloud data centers is seldom questioned. As a result, we are faced with congested networks and high latencies when retrieving data stored at distant locations. The emergence of edge computing provides an opportunity to overcome this issue. In this paper, we present *vStore*, a framework that provides the capabilities for context-aware micro-storage. The framework is targeted at mobile users and leverages small-scale, decentralized storage nodes at the extreme edge of the network. The decision where to store data is made based on rules that can either be pushed globally to the framework or created individually by users. We motivate our approach with different use cases, one of which is the sharing of data at events where cellular networks tend to be congested. To demonstrate the feasibility of our approach, we implement a demo application on the Android platform, leveraging storage nodes placed at different locations in a major city. By conducting a field trial, we demonstrate the key functionalities of *vStore* and report on first usage insights.

Key words: mobile storage, edge computing, fog computing, context-awareness

1 Introduction

Smartphones today have long surpassed their predecessors in terms of computing power, sensory capabilities and application diversity. Mobile phones nowadays feature a variety of different applications. Data captured by or sent to those

applications is usually stored at a distant server, i.e., in cloud computing infrastructures. In any case, this location is predetermined by that particular application. Furthermore, we see a plethora of different applications that serve the same or similar purpose (e.g., Dropbox, Google Drive and OneDrive for cloud-based data storage). Although users sometimes use different services to store the same data, the different applications remain isolated from one another and therefore hinder the sharing of data across them.

Recently, Cisco predicted that by the year 2020, 70 percent of the world population (i.e., 5.5 billion people) will be mobile users[1]. According to their predictions, 72 percent of those will use so-called smart devices, generating a traffic of over 30 exabytes every month. It is fair to assume that this will lead to highly congested core networks. This in part can be avoided if mobile data is stored closer to where it actually is retrieved. More generally, the way mobile data is stored and accessed today is completely decoupled from how the data is actually used and what the current usage contexts of users and their intentions are. Therefore, from the current state of the art, we can derive the following drawbacks and limitations:

1. High bandwidth utilization in the core network: Despite often being retrieved only in a locally restricted area, all data is first sent to the cloud, thus creating high bandwidth utilization and possible bottlenecks in the core network. This is going to worsen as more large-volume data, such as video, will be generated in the future.
2. High latency when retrieving data from a distant cloud. For data such as video, this has a direct impact on the perceived quality of service and therefore is undesirable.
3. No efficient sharing of data across different applications and users.

In this paper, we introduce *vStore* (virtual store), a framework that abstracts from concrete storage locations and—based on the current usage context and intentions of the user—chooses the most suitable storage location. From a networking point of view, *vStore* reduces the bandwidth utilization in the core network and the latency when retrieving nearby copies of requested data. From the user perspective, *vStore* provides context-awareness and facilitates the sharing and reuse of data across locations and applications. Furthermore, *vStore* enables network operators and businesses to provide better quality of experience for their customers by providing proximate cloudlet storage. Our novel framework takes into account the following when deciding on where to store data:

– **Type of data**, such as photo, video, contacts, etc.
– **Usage context** as provided by the mobile device. The context includes for example time, location, ambient noise level and network conditions.
– **User intention**, such as private use or sharing of data.

Instead of solely relying on either local (i.e., on the mobile device itself) or cloud-based storage, we also consider the use of cloudlets [4], small-scale micro-data

[1] https://newsroom.cisco.com/press-release-content?articleId=1741352

centers located at the edge of the network that can be leveraged for computations. In the future, cloudlets on various kinds of network nodes, such as WiFi gateways, cellular base stations or middleboxes, could be used for storing small pieces of data (e.g., a photo taken at an event or point of interest). With *vStore*, we consider the heterogeneity of those network nodes in terms of the bandwidth and latency they can provide in order to optimize the placement decision. To this end, we implement our framework for Android devices and deploy storage nodes in a major city to demonstrate the feasibility of our approach. To the best of our knowledge, this is the first framework that provides the functionality to abstract storage locations and enables to perform storage decisions based on rules that take into account the current context of the user and heterogeneous edge infrastructures.

The remainder of this paper is organized as follows: In Section 2, we give an overview of the research topic and related work. Section 3 presents our idea of context-aware storage at the edge. We describe the design of our system and its components in Section 4. The results of our case study are presented in Section 5. Finally, we conclude the paper and given an outlook on future work in Section 6.

2 Background & Related Work

Before describing our approach to micro-storage at the edge, this section introduces the underlying concepts and reviews previous work related to ours.

2.1 Shifting the Focus from Cloud to Edge

While cloud computing has been the prevailing way for offloading data and computations [1, 2, 3], recent research has begun to identify the drawbacks of this approach [6, 7]. Based on this, we can observe a shift from the cloud towards the edge of the network [5]. Known as fog computing [8, 6] or edge computing [10, 11], this new paradigm makes use of resources close to the users and their data. This includes leveraging opportunistic devices present in one-hop distance, such as standard WiFi routers [12, 13]. Compared to cloud computing, the benefits of this approach include saving bandwidth in the core network and reduced latency. The concept of cloudlets [4] is promising to provide lightweight virtualization of applications for their deployment on resource-constrained devices. We argue that by applying this concept for the storage of data, we can reduce the amount of data stored in distant cloud infrastructures and at the same time provide better quality of service to mobile end users.

2.2 Context-Awareness

We aim to build a framework that makes storage decisions based on rules that take into account the current context of the user. Context is any information that characterizes a current situation [14] and according to Abowd [15], a system is

context-aware if it uses context to provide relevant information and/or services to its users. In our system, contextual information should influence the storage decision. Examples of relevant context include from where the user retrieves the content, where one is located or what the network conditions are alike. In general, we can distinguish between low-level context (i.e. raw and unprocessed sensor data) and high-level context that is inferred from (often multiple) low-level context information.

As mobile phones today feature a multitude of built-in sensors, they are able to capture diverse contextual information. The most prominent contextual information is the location. However, it is easy to see how we can extend this to more sophisticated context. Especially fusing data from *hard sensors* (e.g., a GPS receiver or microphone) with data from *soft sensors* (e.g. one's calendar entries) can generate meaningful higher-level context. As an example, let's assume a user is located at a certain geo-coordinate. Adding a list of point-of-interests, we might derive that he or she is at a sports stadium. Further addition of microphone readings then might derive whether a sporting event is currently in progress. We will later describe how *vStore* uses this kind of context information to make storage decisions.

2.3 Mobile Storage

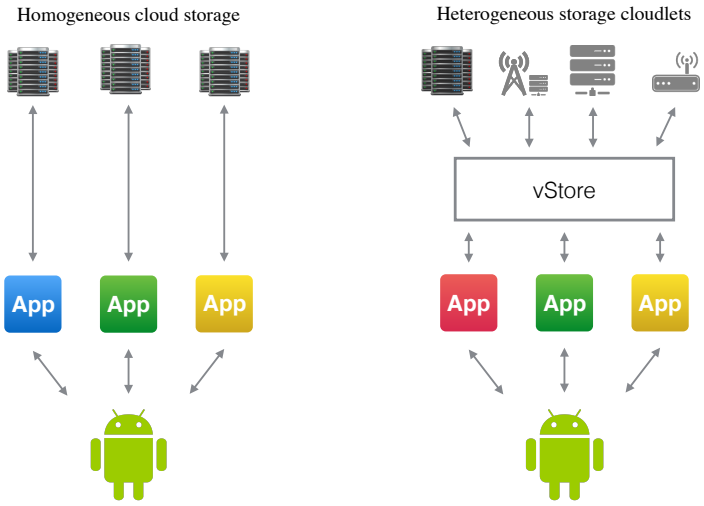
While offloading computations closer to the edge of the network has been studied previously, the possibility to store data at the edge has seldom been examined. Some previous works have proposed to complement cloud storage with an additional layer at the edge of the network. The decision where to store the data is often based on location alone [22], or data is synchronized with cloud storage infrastructures [21]. Using cloudlets for storage in a peer-to-peer fashion has been proposed by Yang et al. [24]. Other than location, network information and usage patterns of files have been taken into account to make storage decisions [25, 26]. In our work we do not limit ourselves to these but provide a general framework that operates on rules, which can incorporate whatever contextual information can be gathered by the devices.

Several approaches have been proposed for caching data, either in a hierarchical way [18] or collaboratively determined by content popularity [19]. Other work combines caching with prefetching strategies based on predicted mobility [20]. By definition, caching is non-persistent and in our cases, we need higher retention times of the data (e.g., to enable sharing).

3 Context-Aware Storage at the Edge

We propose a novel approach to provide context-aware micro-storage to mobile users. Figure 1 compares our approach with the traditional way of having application-specific cloud resources for data storage (Figure 1(a)). Contrary to that, we propose *vStore*, which makes decisions where to store data across

application domains and takes into account heterogeneous storage nodes (Figure 1(b)). Furthermore, our system takes into account contextual information to make the storage decision. We envision existing applications to use the framework as a middleware in order to facilitate the exchange of data between applications.



(a) Traditional application-specific cloud storage

(b) Our proposed approach

Fig. 1: Comparison of approaches

3.1 Use Cases

To further motivate the need for *vStore*, we describe three use cases that benefit from our approach. We base our observations partly on the results of a survey we conducted that had a total of 51 participants. Participants were aged 16–40 and mostly students and researchers.

Sharing data at an event Especially during large-scale events, cellular networks are often congested [16]. A prominent example is football matches. Figure 2 shows measurements of the available cellular bandwidth during a match at the Commerzbank Arena, a stadium in Frankfurt, Germany with a capacity of 51,500 spectators. Compared to the average bandwidth available in the stadium when no match takes place, we can clearly see that the network quality decreases tremendously. At some distinct events, such as goals occurring in the match, the network collapses almost entirely. In such cases, edge cloudlets (that for instance are deployed on several WiFi access points) can be useful to provide users with storage services. Besides the obvious use case of storing data in the cloud for

later use or sharing with people not present at the event, a more interesting use case for edge storage arises when data is to be shared among people present at the very same event. This type of sharing has been examined before in the context of video streaming [17] but not in consideration of infrastructural support of edge computing architectures. In our survey, over 50 percent of the participants stated that they at least occasionally share data such as pictures at an event; close to 20 percent of those almost exclusively with other people attending the same event. Only 4 percent of our participants have never experienced congested connections during events.

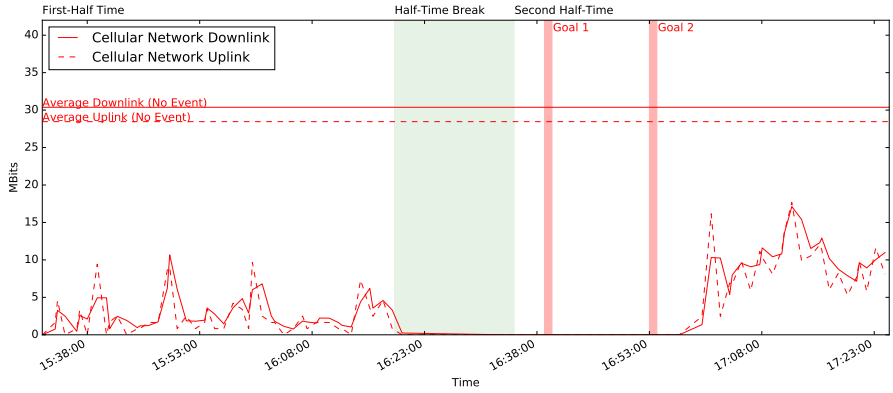
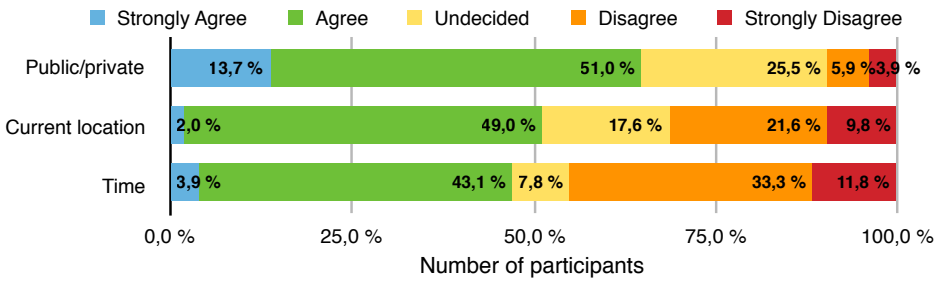


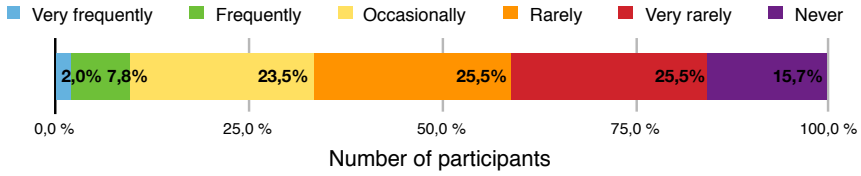
Fig. 2: Cellular bandwidth during a football match

Context-aware storage across applications In our survey, we questioned participants whether the storage services they choose to use depend on (i) whether the data is intended for private or public use, (ii) their current location and (iii) the date and time of data capture. The results of those questions are depicted in Figure 3(a). We can clearly observe that the majority of users bases the decision on where to store their data to a great extent on these three contextual properties. Thus, we will consider them among other properties in *vStore* to make context-aware storage decisions. Furthermore, some users upload the same data to more than one storage service (Figure 3(b)). With the introduction of *vStore*, we aim to provide a unified way to make this decision for the user.

Getting suggestions for data related to one's current context When at a certain location or when performing a certain activity, users often search for information related to that specific context. With the capability to query our framework for data that is similar to one's usage context, we can provide users with such kind of information. Coming back to the example of an event, over 78 percent of our surveyed participants at least sometimes retrieve data related to an event they attend.



(a) Usage of services depending on contextual properties



(b) Upload of the same data to multiple storage services

Fig. 3: Survey results about the usage of multiple storage services

3.2 Problem Definition and Requirements

From the use cases motivated above, we define the problem we want to tackle as follows: Given data that mobile users want to store and contextual information, find the most suitable storage location. In order to provide context-aware micro-storage as we envisioned, the system should fulfill the following requirements: (i) storage location agnosticism, (ii) openness to extensions and third-party applications, and (iii) extensibility to implement new rules for storage decisions. In the next section, we will describe the design of our system in detail and how it fulfills these requirements.

4 System Design and Implementation

In this Section, we describe the design of our system and its individual components. Figure 4 shows a high-level overview of our system. In the following subsections, we will explain in detail the most important parts of our system, including a demo application that makes use of *vStore* on Android phones.

4.1 Virtual Storage Framework

The storage framework is the main contribution of this paper. It provides interfaces to applications in order to store and retrieve data while abstracting from a concrete storage location. The framework collects current contextual information, maintains a list of available storage nodes and—based on a set of

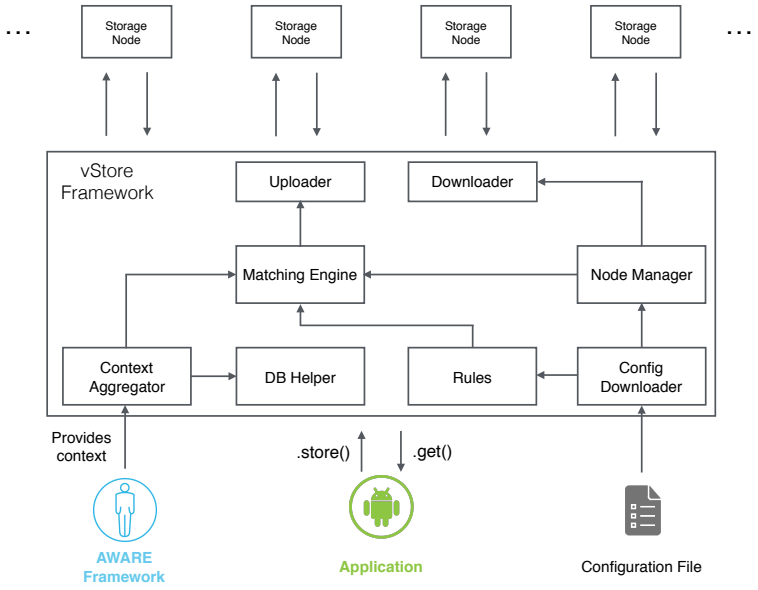


Fig. 4: System architecture

rules—makes the decision where to store the data. For each data item to be stored, a unique identifier is generated that is later used to retrieve specific data across storage nodes.

Context aggregator The task of the context aggregator is to collect the different kinds of contextual information. In Figure 5, the architecture of this aggregator is summarized. To gather contextual information from the mobile phones, we rely on three providers of such information: First, we make use of AWARE[2], an open source framework for context instrumentation on Android phones. Second, the Google Places API provides a list of places that surround the user, their type and the likelihood of the users being located at those places. Third, the Android Connectivity API provides information about the network connectivity of the device. In detail, we collect the following contextual information:

- Location: A plugin for AWARE provides location information using the Google Fused Location API.
- Places: Each time a new location is available, we query the Places API for a new list of places. We group the large amount of place types provided by this API into three groups, namely points of interest (POI), event (such as stadiums, city halls and night clubs) and social (such as restaurants, cafes and bars).

[2] https://www.awareframework.com

– Noise: The ambient noise level is measured by an AWARE plugin through the phone's microphone. By configuring a threshold, we can determine if the current environment should be considered as loud or silent.
– Activity: The user activity is provided by another plugin that internally uses the Google Awareness API to identify the user's current activity (e.g., still, driving or walking).
– Network: We use Android's *ConnectivityManager* and *TelephonyManager* to fetch details about the user's current connectivity situation (e.g. to what kind of network the user is currently connected to).
– The time and date as reported by the phone's operating system.

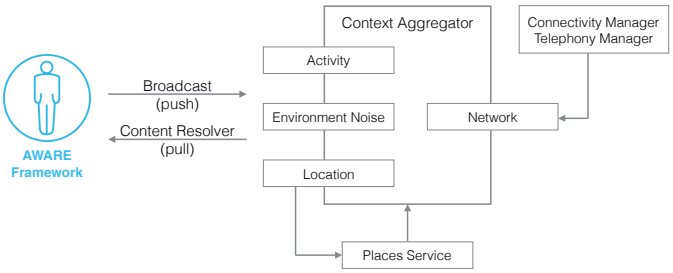


Fig. 5: Context aggregator

Node Manager The node manager maintains a list of all available storage nodes. When storage nodes are added to the framework, their type, location and bandwidth need to be specified. Available nodes can then be queried according to these properties. Before a node is stored in the internal database of *vStore*, the node manager checks if the node is reachable. Node information can be updated and deleted through an API.

Rules In our framework, rules are used to make the storage decision and are evaluated by the matching engine, as described later. In *vStore*, rules can either be defined globally or created individually by the users. A rule specifies certain conditions that have to be fulfilled in order for that rule to be triggered. We specify our rules to consist of three parts:

– Metadata properties: These denote for which MIME type and file size the rule should be taken into account during the matching process.
– Context triggers: These properties determine under which contextual conditions a rule is triggered. Any of the aforementioned contextual information can be specified here. All of the configured contextual properties have to match the context that is given at the time of evaluating the rule.
– Decision layers: The decision layers determine which storage nodes are chosen. A rule can consist of several layers where each layer represents a possible decision outcome. A layer can either be configured to match storage nodes that

are of a certain type or that match certain constraints such as bandwidth or distance to the user. A decision layer can also point to one specific storage node. In this case, the file will be stored on this particular node. The decision layers are evaluated only if the first two tiers of the rule (i.e. metadata properties and context triggers) are fulfilled.

The way we define rules follows the *Event Condition Action* (ECA) paradigm. Mapped to our implementation, the event is the request to store a file, the conditions are the triggers that have to be fulfilled, and the action is the execution of a decision layer and, thus, the storage of a file on a storage node.

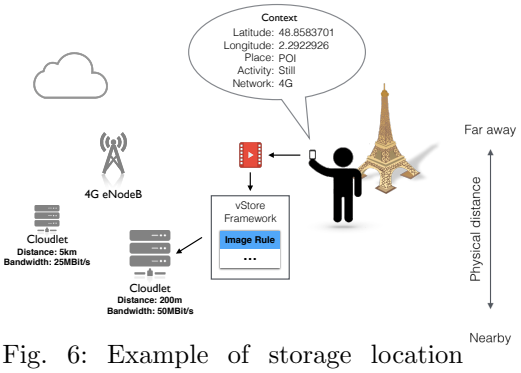


Fig. 6: Example of storage location matching

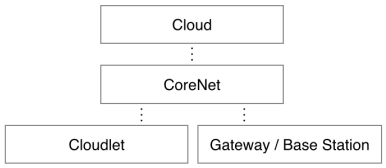


Fig. 7: Storage node hierarchy

Matching engine The matching engine is the main part of the framework. It makes the decision where to store data, given user data, contextual information, a set of available storage nodes and rules as input. The matching process consists of two steps: First, all rules are evaluated with respect to the configured trigger conditions and the type of data. We only consider rules where these two parts match the input. For instance, a rule that only applied to image files would not be evaluated further if the data the users wants to store is a document. Similarly, for the contextual information, let's assume the rule specifies that it only applies to files of a certain size and is restricted to a specific location or within a range from a point of interest. If these properties do not match with the file that is to be stored, this particular rule is discarded. In the second step of the matching process, for each of the remaining rules that satisfy the metadata and context triggers of the input data, a score $s, 0 \leq s \leq 100$ is computed to determine the most detailed rule, meaning the rule that incorporates the most contextual information. For instance, a rule that triggers within 150 meters of a point of interest would be assigned a higher score than one triggering within 500 meters. We apply different weights $\alpha \in \{0.1, 0.15, 0.2\}$ to put more emphasis on contextual information that is based on location. The rule with the highest score is then chosen, and according to its decision layers (see previous section), a storage node is then selected. In cases where the decision layers do not lead

to a feasible solution, for instance because no nodes that fit the constraints are available, the rule with the second highest score is chosen and evaluated next. Figure 6 shows an example of a storage decision, in which a user takes a photo near a point of interest. A rule triggers because the user is currently still and not in motion, is connected to a 4G cellular network and at a point of interest. In this example, this rule (labeled *Image Rule* in the figure) is the one with the highest score, and in its decision layer, a cloudlet with a maximum distance of 200 meters and at least 50 MBit/s of bandwidth is chosen to store the photograph.

4.2 Storage Nodes

Storage nodes are the devices that are available to store the data. In a real-world deployment, a storage node could be hosted on a variety of devices, either close-by or distant to the user. To take into account this heterogeneity, we define different types of storage nodes as outlined in Figure 7. Besides cloud nodes, we consider cloudlets, gateway nodes and nodes in the core net. The latter could be represented by network layer middleboxes, which could have additional capabilities to store data as it passes through those devices. Gateway nodes on the other hand are devices to which users have a direct wireless connection to, such as WiFi access points or cellular base stations.

In addition, we also consider private clouds as a type of storage nodes, for instance systems such as ownCloud [3] that are owned by end users themselves. Including this kind of nodes allows to define storage rules for the storage of private data, i.e., data that is not shared among different users of the framework.

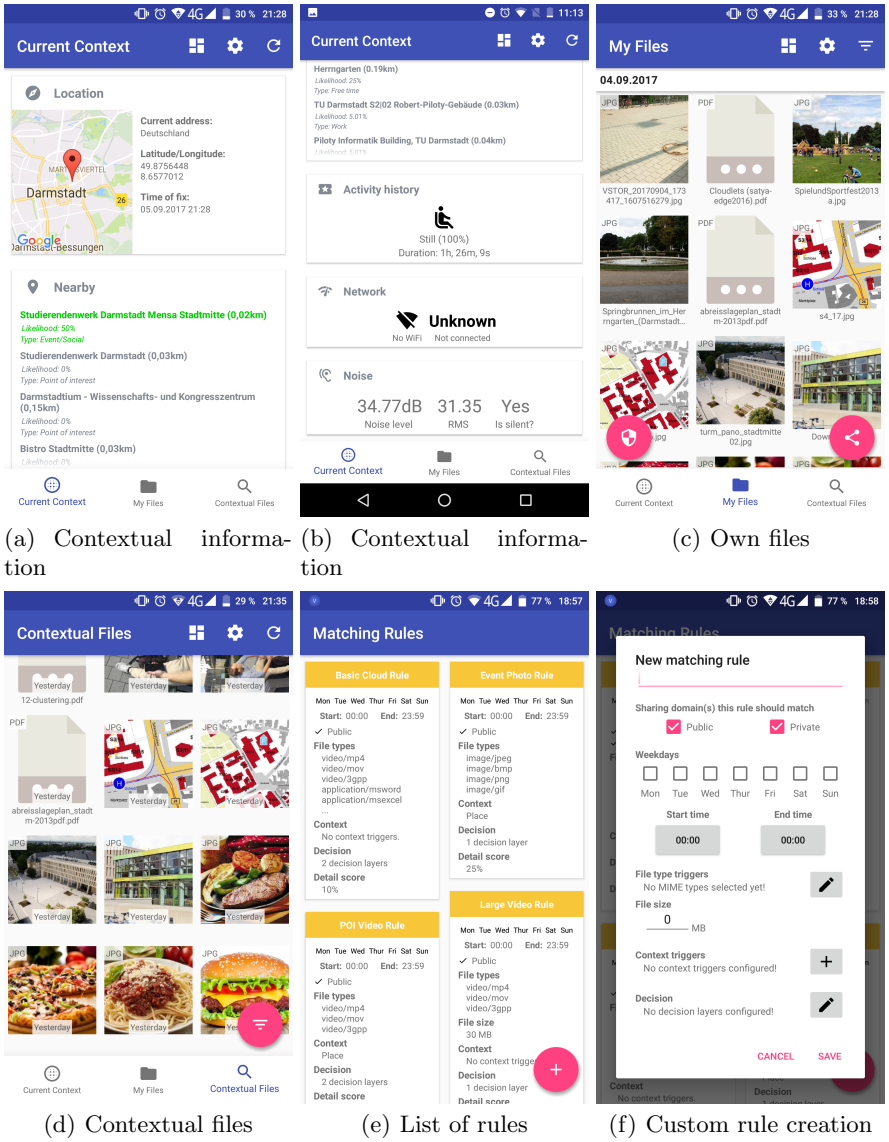
4.3 Configuration

The *vStore* framework can be configured externally. This mainly serves two purposes: (i) initially retrieving available storage nodes and (ii) including global rules for the placement decision. Defining global rules that are available on all devices is important for users who do not wish to specify custom rules. This ensures that at least some basic storage decisions can be made. To this end, the framework relies on a central configuration file that is regularly retrieved and updates available storage nodes and rules. However, in the future, we envision the configuration of the framework to be managed in a distributed way. This would for instance enable users who have the same or similar context to share custom rules they have defined.

4.4 Demo Application

As a case study and to demonstrate the integration of the framework on a mobile platform, we developed a demo application on Android phones. With this application, users are able to (i) store their data on a storage node determined by the matching engine of the framework, (ii) mark this data as either private or public,

[3] https://owncloud.org



(a) Contextual information (b) Contextual information (c) Own files

(d) Contextual files (e) List of rules (f) Custom rule creation

Fig. 8: Screenshots of the demo application

(iii) create custom storage mapping rules that are then used by the mapping engine and (iv) get context-related data from other users. The application's main screen shows a summary of all current contextual information available (Figures 8(a) and 8(b)). Users can view their own files (Figure 8(c)) and are able to upload files on this screen. This can be done for public files (right pink button) and private files (left pink button). To demonstrate the framework's ability to

retrieve files based on context, users can retrieve files with a similar context to their current one (Figure 8(d)). Furthermore, the app shows all currently active storage rules (Figure 8(e)) and allows for the creation of custom rules by the user (Figure 8(f)).

We will use this application for our user study to conduct preliminary experiments on storage rules in a heterogeneous city-scale environment. Details of this evaluation are described in the next section.

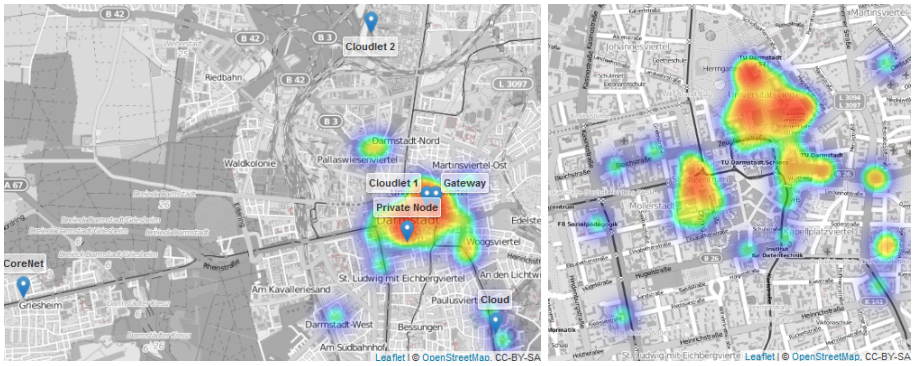
5 Preliminary Results

In this section, we report on preliminary experiments we conducted using the demo application we described in Section 4.4. We show the feasibility of our approach by deploying several storage nodes in a major city and conduct a user study by defining sample rules and evaluating the resulting storage decisions that *vStore* made. We conclude the section by providing a discussion about our findings.

5.1 Experimental Setup

We deployed a total of six storage nodes in the area of Darmstadt, Germany. The maps shown in Figure 9 visualize our deployment. Figure 9(a) shows an overview of the area with the location of the storage nodes and Figure 9(b) zooms in on the city center with a heatmap depicting where most of the data was captured. For a quick deployment, we use Raspberry Pis running mongoDB as data storage. A NodeJS server implements the storage service and acts as an interface between the data storage and the *vStore* framework. To simulate different types of storage nodes we would have in a large-scale deployment, we set different node types in our system: two cloudlets, one gateway, one cloud node, one core net node and one private cloud.

We defined several global rules that were pushed to the phones in our field trial of *vStore*. They are described in Table 1. The *POI Photo Rule* will be executed when the user is detected to be close to a point of interest and takes a photo. It matches any file size, day and time, and only applies to photos the user wants to share. With the configured context, a detail score of 25% is reached. The decision is divided into two layers. The first layer tries to save the file on a gateway if one is available within 5km from the POI. If none is available, it tries to find a cloudlet in a radius of 20km. The *Social Photo Rule* will be executed in places that we consider to be social according to our places context. Here, the same conditions apply as for the POI Photo Rule. We define two rules to be able to evaluate them separately, according to the different place context. The *Driving Rule* takes into account the activity context, as reported by the context aggregator. If the user's current activity is driving, it does not mean that he is driving the car himself. This rule would also apply in trains or taxis. Any file uploaded in this context will not be stored on nearby cloudlets since he



(a) Overview (b) City center

Fig. 9: Node locations and usage heatmaps

might only drive by a nearby POI without the intention of sharing or retrieving related data. The *Event Photo Rule* applies in the context of a place that is of the type event and a noise level of at least 20dB. We determined the value for this empirically by evaluating the behavior of the AWARE Noise plugin, even though this value seems highly inaccurate. If this context is given, the rule will store shared photos on a cloudlet within a radius of 30km. The *Basic Cloud Rule* will be used as a fallback due to the low detail score, should no other rule yield a result. It then checks if a core net node with a bandwidth of 10GBit/s is available to upload the data. If this is not the case, the file will be stored in the cloud. Finally, to also evaluate the storage of private files, we created a *Basic Private Rule*. This rule matches everything the user wants to store for private use.

We distributed the demo application to six participants and configured the framework with the aforementioned rules. The participants were asked to use the application to capture various kinds of data (e.g. photos, videos, contacts) and store them using the demo application described in Section 4.4. The map in Figure 9 shows on a heatmap where users were most active.

5.2 Usage Patterns and Storage Decisions

We now look at how users used their devices, i.e., which types of data they stored and which storage decisions were made based on the rules we defined. The bottom row of Table 1 shows how many times each rule was triggered. We can observe that the *Basic Cloud Rule* was triggered the most, however, data was stored on cloud nodes only for 29.3% of all data. This is because the cloud rule has a very low detail score. In many cases, other rules that relate to the user's location or define a proximity to a point of interest, have a more detailed score and therefore those are the ones that determine the placement. We can think of the cloud rule as a fallback, in case there is no most likely place (e.g., when we are not sure where the user is).

	POI Photo Rule	Social Photo Rule	Driving Rule	Event Photo Rule	Basic Cloud Rule	Basic Private Rule
Context	Place: POI	Place: Social	Activity: Driving	Place: Event, Noise: > -20dB	None	None
File Size	Any	Any	Any	Any	Any	Any
File Types	JPG, BMP, PNG, GIF	JPG, BMP, PNG, GIF	Any	JPG, BMP, PNG, GIF	Any	Any
Sharing Domain	Public	Public	Public	Public	Public	Private
Weekdays	Mon-Sun	Mon-Sun	Mon-Sun	Mon-Sun	Mon-Sun	Mon-Sun
Time	Any	Any	Any	Any	Any	Any
Decision Layers	**Layer 1** Gateway ≤ 5km **Layer 2** Cloudlet ≤ 10km	**Layer 1** Cloudlet ≤ 5km **Layer 2** Cloudlet ≤ 10km	**Layer 1** Cloud	**Layer 1** Cloudlet ≤ 30km	**Layer 1** CoreNet with ↑ 10GBit/s, ↓ 10GBit/s **Layer 2** Cloud	**Layer 1** Private Node
Detail Score	25 %	25 %	20 %	35 %	10 %	10 %
Times executed	36	34	18	5	47	5

Table 1: Placement rules

The resulting placement decisions for the different file types that were used during our study are shown in Table 2. In total, users stored 178 files using *vStore*, most of which were photos. Out of those, 35.9% were stored on cloudlets, 19.3% on gateway nodes and 2.7% on core net nodes. These numbers confirm the benefits that can be obtained in future edge computing environments. In contrast to this, without *vStore*, users would likely have all their photos uploaded to distant cloud infrastructures. The table furthermore depicts the sharing ratio for each type of data, i.e., whether users marked the data to be publicly shared on the storage nodes or for their private use. From the results, we can observe that this heavily depends on the data type. While users were willing to share over 80 percent of their images, for more sensitive information such as contacts this number drops down close to 3 percent. With the set of rules we defined, we are able to capture the user's intention, as the sharing domain influences the placement decision *vStore* makes.

	Node Type								
	Gateway	Cloudlet 1	Cloudlet 2	CoreNet	Cloud	Private Node	Phone	Total	Sharing Ratio
Images	28	35	17	4	43	4	14	145	81.46%
Videos	3	6	1	1	3	0	3	17	9.55 %
Documents	0	1	0	2	7	1	0	11	6.18 %
Contacts	2	0	0	0	3	0	0	5	2.81 %

Table 2: Placement results

5.3 Discussion

With our preliminary experiments outlined in this section, we were able to show how we can make context-aware storage decisions that include heterogeneous storage nodes by using rule-based matching. However, the accuracy of contextual descriptions remains an issue. For instance, files were sometimes saved using a

wrong context, due to the fact that the context is not updated in real time. Keeping an accurate context on a mobile phone is always a trade-off between accuracy and energy consumption. In addition, much work still needs to be done in order to correctly recognize higher-level context. However, our user study motivated the use cases of using cloudlets, especially if they are located at the edge of the network and close-by to mobile users. This is especially true in the context of sharing data locally. For this use case, *vStore* offers the possibility to define rules that are triggered when a user is at a certain location or point of interest. As outlined in Section 3.1, many people today share data at events, some of them even with people present at the same event. For the future, we envision storage cloudlets to be deployed throughout city areas, some of which are co-located at the radio access network or act as gateway nodes themselves (e.g., WiFi hotspots during events).

Of course, appropriate rules are required to make the framework beneficial in practical use. We enable users to define custom rules that they can represent their usage intentions with. In addition to custom rules, the framework allows for global rules to be configured. In our field trial, we could see that even with just a basic set of global rules, these were often executed when making the placement decisions. In future use of the system, infrastructure providers could set these global rules, e.g., to specify local cloudlets on gateway nodes when regular networks are overloaded.

6 Conclusion and Future Work

In this paper, we presented *vStore* (virtual store), a framework that enables micro-storage at the edge of the network and abstracts from predefined storage locations for data captured by mobile users. This enables (i) decoupling of storage locations from specific cloud infrastructures and therefore facilitates the exchange of data across applications and (ii) leveraging small-scale cloudlets at the edge of the network to provide better quality of service to mobile users. An example use case for the latter is the sharing of data when cellular networks are congested, e.g., during large-scale events. *vStore* allows different stakeholders (e.g., mobile users or infrastructure providers) to define custom rules that are evaluated when making the decision where to store the data. We showed the viability of our system with a user study using a demo application that users could use to capture and upload data. Furthermore, users were able to retrieve related data related to their current usage context. We deployed different storage nodes in a major city and through the implementation of example decision rules we were able to show how this framework can complement existing cloud-based storage infrastructures.

vStore provides an extensible framework we encourage other researchers to use in order to test new approaches to decide storage locations. The rule framework allows for custom definition and evaluation of decision algorithms. We especially envision the emerging research on machine learning to be able to

contribute interesting insights on this. In future work, we will investigate optimizations on the network layer as well as suitable replication mechanisms that are required for a larger-scale deployment of our system. Furthermore, we plan to include mechanisms to include the dynamic discovery of storage nodes into the framework.

Acknowledgements

This work has been co-funded by the German Federal Ministry for Education and Research (BMBF, Software Campus project DynamicINP, grant no. 01IS12054) and by the German Research Foundation (DFG) as part of the Collaborative Research Center (CRC) 1053 - MAKI.

References

1. Y. Jadeja, and K. Modi. Cloud Computing - Concepts, Architecture and Challenges. In *International Conference on Computing, Electronics and Electrical Technologies (ICCEET),* pp. 877-880, IEEE, 2012.
2. B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. Clonecloud: Elastic execution between mobile device and cloud. In *6th Conference on Computer Systems, ser. EuroSys,* pp. 301314, ACM, 2011.
3. H. Flores, P. Hui, S. Tarkoma, Y. Li, S. Srirama, and R. Buyya. Mobile code offloading: From concept to practice and beyond. *IEEE Communications Magazine,* vol. 53, no. 3, pp. 8088, IEEE, 2015.
4. M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for VM-based cloudlets in mobile computing. *IEEE Pervasive Computing,* vol. 8, no. 4, pp. 1423, IEEE, 2009.
5. M. Satyanarayanan. The Emergence of Edge Computing. *IEEE Computer Magazine,* vol. 50, no. 1, pp. 3039, IEEE, 2017.
6. S. Yi, C. Li, and Q. Li. A Survey of Fog Computing: Concepts, Applications and Issues. In *Workshop on Mobile Big Data (Mobidata)*, pp. 3742, ACM, 2015.
7. S. Yi, Z. Hao, Z. Qin, and Q. Li. Fog Computing: Platform and Applications. In *3rd IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*, pp. 7378, IEEE; 2015.
8. F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog Computing and its Role in the Internet of Things. In *1st Edition of the MCC Workshop on Mobile Cloud Computing*, pp. 13-16, ACM, 2012.
9. M. Baguena, G. Samaras, A. Pamboris, M. Sichitiu, P. Pietzuch, and P. Manzoni. Towards Enabling Hyper-Responsive Mobile Apps through Network Edge Assistance. In *13th IEEE Annual Consumer Communications and Networking Conference (CCNC)*, pp. 399-404, IEEE, 2016.
10. A. Chandra, J. Weissman, and B. Heintz. Decentralized edge clouds. *IEEE Internet Computing*, vol. 17, no. 5, pp. 7073, IEEE, 2013.
11. M. Beck, M. Werner, S. Feld, and T. Schimper. Mobile Edge Computing: A Taxonomy. In *6th International Conference on Advances in Future Internet (AFIN)*, pp. 48-54, IARIA, 2014.

12. C. Meurisch, A. Seeliger, B. Schmidt, I. Schweizer, F. Kaup, and M. Mühlhäuser. Upgrading wireless home routers for enabling large-scale deployment of cloudlets. In *International Conference on Mobile Computing, Applications, and Services (MobiCASE)*, pp. 1229, Springer, 2015.
13. J. Gedeon, C. Meurisch, D. Bhat, M. Stein, L. Wang, and M. Mühlhäuser. Router-based Brokering for Surrogate Discovery in Edge Computing. In *International Workshop on Hot Topics in Planet-Scale Mobile Computing and Online Social Networking (HotPOST)*, IEEE, 2017.
14. A. Dey. Understanding and using context. *Personal and ubiquitous computing*, vol. 5, pp. 4-7, Springer, 2001.
15. G. Abowd, A. Dey, P. Brown, N. Davies, M. Smith, and P. Steggles. Towards a better understanding of context and context-awareness. In *Handheld and ubiquitous computing,* pp. 304-307, Springer, 1999.
16. A. Frömmgen, J. Heuschkel, P. Jahnke, F. Cuozzo, I. Schweizer, P. Eugster, M. Mühlhäuser, and A. Buchmann. Crowdsourcing Measurements of Mobile Network Performance and Mobility During a Large Scale Event. In *International Conference on Passive and Active Network Measurement (PAM),* pp. 70-82, Springer, 2016.
17. N. Dezfuli, J. Huber, S. Olberding, and M. Mühlhäuser. CoStream: in-situ co-construction of shared experiences through mobile video sharing during live events. In *CHI Extended Abstracts on Human Factors in Computing Systems,* pp. 2477-2482, ACM, 2012.
18. F. Duro, J. Blas, D. Higuero, O. Perez, and J. Carretero. CoSMiC: A Hierarchical Cloudlet-Based Storage Architecture for Mobile Clouds. *Simulation Modelling Practice and Theory*, vol. 50, pp. 3-19, Elsevier, 2014.
19. Z. Cao, and P. Papadimitriou. Collaborative Content Caching in Wireless Edge with SDN. In *1st Workshop on Content Caching and Delivery in Wireless Networks (CCDWN)*, ACM, 2016.
20. F. Zhang, C. Xu, Y. Zhang, K. Ramakrishnan, S. Mukherjee, R. Yates, and N. Thu. EdgeBuffer: Caching and Prefetching Content at the Edge in the MobilityFirst Future Internet Architecture. In *World of wireless, mobile and multimedia networks (WoWMoM)*, pp. 1-9, IEEE, 2015.
21. Z. Hao, and Q. Li. EdgeStore: Integrating Edge Computing into Cloud-Based Storage Systems. In *Symposium on Edge Computing*, pp. 115-116, IEEE/ACM, 2016.
22. P. Stuedi, I. Mohomed, and D. Terry. Wherestore: Location-based data storage for mobile devices interacting with the cloud. In *1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*. ACM, 2010.
23. C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos. Context Aware Computing for The Internet of Things: A Survey. *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 420435, IEEE, 2014.
24. Z. Yang, B. Zhao, Y. Xing, S. Ding, F. Xiao, Y. Dai. AmazingStore: Available, Low-Cost Online Storage Service Using Cloudlets. In *9th International Workshop on Peer-to-Peer Systems*, USENIX, 2010.
25. S. Bazarbayev, M. Hiltunen, K. Joshi, W. Sanders, R. Schlichting. PSCloud: A Durable Context-Aware Personal Storage Cloud. In *9th Workshop on Hot Topics in Dependable Systems*, ACM, 2013.
26. D. Han, Y. Yan, T. Shu, L. Yang, S. Cui. Cognitive Context-aware Distributed Storage Optimization in Mobile Cloud Computing: A Stable Matching based Approach. In *37th International Conference on Distributed Computing Systems (ICDCS)*, IEEE, 2017.