Help Your Mobile Applications with Fog Computing

Mohammed A. Hassan, Mengbai Xiao, Qi Wei and Songqing Chen

mohammeh@netapp.com, NetApp Inc.
{mxiao3,qwei2,sqchen}@gmu.edu, Department of Computer Science, George Mason University

***Abstract*—**

Cloud computing has paved a way for resource-constrained mobile devices to speed up their computing tasks and to expand their storage capacity. However, cloud computing is not necessary a panacea for all mobile applications. The high network latency to cloud data centers may not be ideal for delay-sensitive applications while storing everything on public clouds risks users' security and privacy. In this paper, we discuss two preliminary ideas, one for mobile application offloading and the other for mobile storage expansion, by leveraging the edge intelligence offered by fog computing to help mobile applications. Preliminary experiments conducted based on implemented prototypes show that fog computing can provide an effective and sometimes better alternative to help mobile applications.

I. Introduction

The increasing popularity of the mobile devices is attracting more and more application developers to develop computation- and data-intensive applications for mobile devices. But mobile devices are inherently constrained by the limited on-device resources, including limited computing power, storage space, and the battery power supply. To address these limitations, a lot of efforts have been made to improve mobile devices' capability with cloud computing via application offloading and storage augmentation [18], [19], [29], [27], [15], [21], [20], [25], [11], [28], [31], [24].

However, cloud computing is not necessary a panacea for resource-constrained mobile devices. Day to day mobile applications can be data-intensive and the network latency between a mobile device and a cloud data center can significantly affect the performance of offloaded computation from a mobile device, leading to un-desired results. Some prior studies [30], [22], [24] show that due to the high network latency and the slow network bandwidth, it may not always be suitable to offload computation to the cloud. But this decision is not straightforward. Consider the varying nature of the environment (e.g., network dynamics), the decision has to be made dynamically and adaptively about which part(s) of the application should be offloaded (even whether to be offloaded at all). With the availability of nearby resources via fog computing, there are more choices and thus the decision process is more complex. Some prior frameworks [29], [15], [27], [19] mainly focused on how the computation can be offloaded while treating the decision process (whether and where to offload) lightly.

On the other hand, currently cloud computing plays an important role in expanding storage capacity for mobile devices as today mobile devices often have very limited storage. For example, a typical iPhone or Samsung Galaxy S5 may have 16 to 32 GB of storage. To increase the storage, there are a lot of third party cloud storage services available for mobile devices, such as `Dropbox` [4], `Google Docs` [7], `Amazon s3` [2], `Windows SkyDrive` [10] and `SME Storage` [11]. Although such cloud storage services always aim to provide 7/24 access to users, there are several limitations. First, their storage servers are behind public network where the performance could be an issue [22], [24]. Moreover, these third party cloud service providers are still prune to the single point of failure [8], [5], [1]. Most importantly, storing all user data on public cloud storage risks users' security and privacy, which is an increasing concern of mobile users as today mobile devices tend to contain more and more sensitive user information.

In this paper, we set to explore the edge intelligence of fog computing to deal with these limitations. For this purpose, we present some of our ideas in utilizing nearby resources to help mobile applications. To efficiently speed up mobile computation, we propose to take into account all the available resources, particularly their runtime configurations (e.g., the network latency and the bandwidth between the mobile device and the server, the size of the overhead data, etc.) and dynamically choose partition(s) of the application for offloading. To expand the mobile device storage, we propose to leverage the users personal devices for nearby-accesses and better security and privacy.

To demonstrate the effectiveness of our proposed approaches, we have built prototypes of our proposed ideas and conducted some preliminary experiments accordingly. Experimental results show that better results could be achieved via our proposed approaches.

II. Adaptive Computation Offloading

Today, mobile devices are pervasive and common users rely more and more on their mobile devices than the desktop counterpart. Corresponding to this trend, mobile applications are fast growing, ranging from simple applications to more and more complicated computation- and data-intensive applications. Yet the gap of the on-device resources of a mobile device and those on a traditional desktop or laptop computer is still big. In addition, mobile devices are fundamentally constrained by limited battery power supply. To deal with these constraints, plenty of research [30], [15], [27], [21], [18], [19], [29], [32] has been conducted to offload the computation to the power computing resources, e.g., the cloud today. However, cloud computing is not necessary a panacea to augment computation for resource constrained mobile devices [22]. For example, a simple face recognition application like Picaso [9] may take up to six seconds to find a match when it is offloaded to a cloud, which is even slower than executing the computation

978-1-4673-7392-0/15/$31.00

on device locally. Such overhead mainly comes from the network conditions (e.g., the network bandwidth and latency). Naturally, nearby resources through fog computing [17] may provide better performance by bringing the computation at arm's reach.

Today nearby computing resources are pervasive, such as small wifi and routers deployed at home, offices, and public places. To facilitate the utilization of such edge resources, it is ideal to join them together collaboratively, e.g., via fog networking. Users can collaborate with each-other [22] to offer computation offloading functions. In this section, we do not focus on the formation of such network. Instead, we focus on how to make proper offloading (e.g., where and what to offload) once such resources are available so that one can explore the potentials of such offloading.

Consider that sometimes the network bandwidth and the latency can deteriorate the performance of the offloaded tasks, a careful decision needs to be made upon whether and where to offload the computation-intensive task, if necessary. In the cloud computing scenario, some prior research [27], [19], [32], [18] has adopted threshold or linear regression models, which often results in static decisions. Without thorough consideration of the dynamics of the available resources, such simpler models may fail to make correct decisions as shown in POMAC [25]. With the availability of nearby resources, the options are much richer and such a decision process becomes more dynamic. Some intelligence is necessary to help mobile devices to make right decisions. We propose the following framework for this purpose.

A. Problem Formulation

To make a proper offloading decision, we need to actively monitor the system resources. Based on the resource availability and dynamics, we may predict the performance of different tasks of the application on different computing facilities. Once we can figure out which tasks can benefit from offloading most, we can offload these tasks.

TABLE I: Experimental Environment Setup

Simulated Network	Network Bandwidth (Kbps)	CPU (GHz)	Memory (MB)	Network Latency (ms)
Fog 1	100000	1	1024	20
Fog 2	30000	1	2048	20
Fog 3	25000	2	2048	50
Cloud 1	5000	2	2048	75
Cloud 2	500	2	2048	200

The same task of an application may have different performance result on different computing facilities. To empirically study this, we conduct experiments with the face recognition method of Picaso [9] under different environments. We set up a total of five different environments, three of them emulate fog and two of them emulate cloud. The clouds have relatively worse bandwidth and latency configuration but better CPU and memory configurations compared to those of the fog, as cloud is usually accessed through 4G or 3G, while fog devices are usually reachable through wifi or WLAN. Table I summarizes the different parameters for different environments.

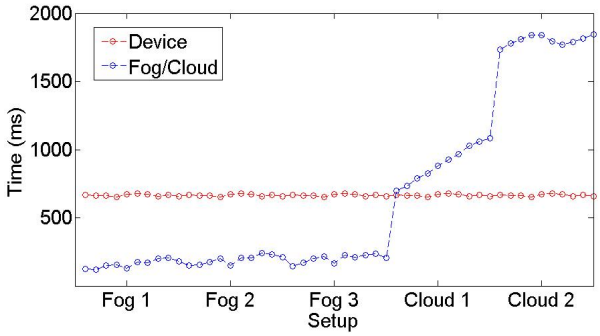

Fig. 1: The Response Time of Picaso is better in nearby resources

Figure 1 shows the response time of 10 experiments of Picaso in different environments. The figure shows that when the computation is offloaded to the cloud, the response time increases drastically even though the cloud has better computing power and/or memory in some cases. Sometimes it may be worthwhile even not to offload computation to the cloud as it takes more time than that of on-device execution.

Such results hint us two things. First, we need to find out carefully which parameters impact the offloading performance (including bandwidth, latency, server-side CPU and memory, etc.). Second, we need to accurately predict the offloadable task's performance based on these factors.

To measure the bandwidth and latency between the mobile device and the potential offloading target, we may periodically ping the fog/cloud server and measure them using a number of existing methods. We can also find the available CPU and memory information from the fog and cloud server. Our study shows that we can predict these values with better accuracy using moving average [23].

To predict the performance of the offloaded methods, some prior research [27], [21] claims that a threshold based offloading policy may help. Some other schemes [19], [18] use history based information or linear regression based policies, which are oversimplified, failing to capture the complex relationship between different parameters, as shown in POMAC [25]. As a result, we suggest high dimensional models, such as Multilayer Perceptron (MLP), to predict the performance of different tasks in different environments and thus make the proper offloading decision. Extending from the same basic idea, with the availability of different nearby resources as well, we can monitor the different environmental parameters and response time and train the Multilayer Perceptron model. At runtime, we monitor the system environment continuously and predict the performance of different computation-intensive parts of the application in different environment (the fog, or the cloud, or even the mobile device itself). Based on these predicted values, we partition the application and make the best offloading decision, which we explain below.

To decide which task(s) of an application is appropriate for offloading, we formulate the problem as a constrained graph partitioning problem to maximize the benefit by offloading. In this work, we consider the candidate tasks at the method level. That is, we have a list of methods M which we want

to offload, and each method $m \in M$ has the following members: i) the execution cost on the mobile device, denoted as m_m, ii) the offloading (the parameters) and execution cost on server (fog or cloud), denoted as m_s, iii) the list of all the variables reachable from this method, denoted as RV (Reachable Variables), iv) the list of all the methods reachable from this method, denoted as RM (Reachable Methods),

Denote the method call graph as MCG. Each node of MCG represents a method and an edge e_{xy} represents whether a method x is invoking another method y. Assume that the execution cost of a method m in the mobile device is m_m and on the server (fog or cloud, without considering the parameters sending cost) m_{swp}. In addition, the cost of invoking a method m from method x is denoted as e_{xm}, and the cost for accessing a variable (global or class) v is e_{mv}. So the cost of executing a method on the server (including the overhead) is:

$$m_s = m_{swp} + \sum_{\forall x \in \{M-m\}} e_{xm} + \sum_{\forall x \in RV} e_{mx}. \quad (1)$$

And the cost saved by offloading a method is

$$b_m = m_m - m_s. \quad (2)$$

Thus, our goal is to partition the application's MCG into two partitions S and T and offload all the methods $m \in S$ to the server side such that:

$$Execution\ Time = \sum_{\forall m \in T} m_m + \sum_{\forall m \in S} m_s \quad (3)$$

is minimized. Note that we predict m_m and m_s for different setups (fog or cloud) mentioned in equation 1 by Multilayer Perceptrion. After calculating the minimized time for mobile device, cloud, and fog, we may choose the optimal method(s) for offloading.

B. Preliminary Evaluation

We implemented a prototype in Android OS with the above discussed decision making process and candidate method selection process. We trap method calls in application VM of a modified Android VM. We trap the frame page and the stack pointer when one method invokes the offloadable method, and instead of executing the method locally, we offload them to the cloud or the fog server. Such a modified Android VM allows the methods to be trapped and offloaded in the application VM level, so there is no need to modify the code. Thus, we can offload the third party applications in a transparent manner.

We present our preliminary evaluation of two different application Picaso [9] (Android Face Recognition app) and DroidSlator [3] (Android Dictionary app) in different environmental setup as shown in Table I. Both of these applications are data- and computation-intensive and they are popular types of application.

We have conducted ten experiments in each of the environmental setup. Figure 2 and 3 show the average response time and the energy consumption of the applications in different environments. In these experiments we predict the applications' performance in different environments using MLP and then partition the applications. In these figures, *OnDevice* means

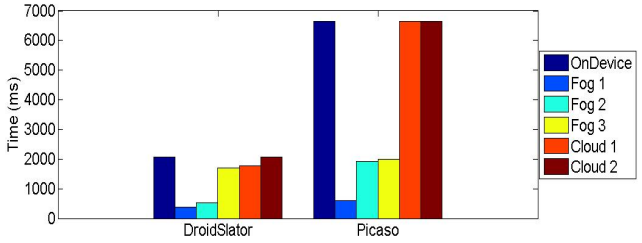

Fig. 2: Response time of the applications in different environment

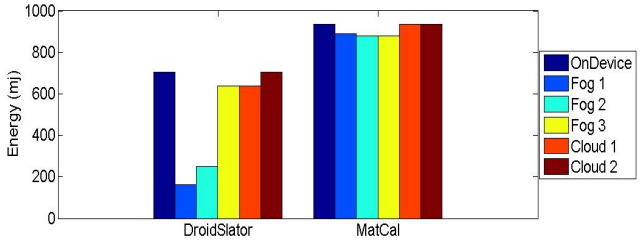

Fig. 3: Energy consumption of the applications in different environments

running on the mobile device, a Google nexus one with 1 GHz CPU and 512 MB of RAM. Note that the training and classification overhead of the MLP has been accounted for. As shown in the figures, offloading to the nearby fog computing servers outperforms offloading to the cloud and execution on the local device, in terms of both the response time and the corresponding energy consumption. For example, fog computing sometimes can save upto 5.5 time response time and energy consumption, while for cloud computing we can save upto 1.3 times.

C. Challenges

Although fog computing can offer better help to mobile applications, there are some challenges to overcome before we can implement this in practice. For example, cloud resources are almost available 7/24. To provide comparable services, fog computing needs to provide similar resource availability. Although wifis/routers may be leveraged, the reliability needs to be improved. In addition, leveraging the private wifi/router resources also needs to deal with the associated security and privacy concerns, of both the owners and the users. Private networks, such as the one built for vUPS [24], may offer computation at home or office to the users in a private and secure manner to avoid this issue.

III. Mobile Storage Expansion

Despite the reducing price of storage, today mobile devices often still have very limited storage (a typical iPhone or Samsung Galaxy typically have 16 to 32 GB of storage) even when compared with a low-end desktop computer. As a result, cloud based storage services, such as `Dropbox` [4], `Google Docs` [7], `Amazon s3` [2], `Windows SkyDrive` [10] and `SME Storage` [11] are very popular.

These cloud services are often accessible via public networks and they are not at arm's reach for mobile users. The network latency can play an important role when accessing the data stored there [24]. Moreover, these solutions are inherently prune to the single point of failure. Furthermore, in the past years, we have seen security breaches, such as Mark Zuckerberg's pictures leak incident in Facebook [8], DropBox account breach with wrong passwords [5], Amazon's data center failure in 2011 [1], etc. As a result, some prior research [11], [28], [31], [24] has been conducted to investigate how to include the user's personal storage to expand storage capacity for mobile devices. In this section, we discuss our effort along this direction by utilizing the personal storage via fog computing to augment storage capacity for mobile devices in a safe and efficient manner.

A. Design and Implementation

Compared to the massive amount of data that demand cloud storage, a typical mobile user's data is often not comparable in the amount and often can be stored in all the personal storage combined together [16]. Motivated by this observation, we propose to integrate all the personal storage space of a user (her laptops, home and office computers, etc.) together to build-up a distributed storage service, backed up via fog networking. That is, an average user today often owns or controls multiple computing devices. While a mobile device often has limited storage space, a desktop computer at work or at home often has rich storage space. If these personal storage space can be seamlessly integrated together, the total space is often more than the demand of a user [16]. To utilize such a service, a user can register any of her devices, e.g., via some lookup directory, which can be hosted in a cloud such as Amazon EC2. Note that from the bootstrapping perspective, there are lots of other alternatives.

One main challenge of such a storage system is its accessibility from anywhere. An intuitive solution is to replicate every piece of data based on the number of participating devices so that there is a copy of the same data on any device. This certainly does not work as mobile devices have storage constraints. Therefore, we propose to distribute the data among her different storage space with backup provision. For better availability, we keep the metafile in a public cloud server so that they can always be accessed, while the actual data is stored only in one's personal devices. In this way, a user can first contact the cloud server, find the device where the desired data is stored and then can fetch the data accordingly. The $< data, location >$ mapping can also be kept in the public cloud for 7/24 accesses. In this approach, the data are stored on one's own storage space, and thus coming with less security and privacy concerns than those with public clouds. Such an idea has been demonstrated by vUPS [24]. In such a distributed storage system, if a user always fetches data from her another remote device, it could be expensive as well. To provide better services, we can trace the user access pattern and place the data closer to where it is accessed most frequently.

For this purpose, we further propose a near-optimal data location policy for this geologically distributed file system. To build such a policy, we have collected file access frequencies with respect to each location. For each file, we collect the operation type, time, and location, the amount of data associated with each operation, the types of operations, etc. We cluster the files based on their operation location, discover data access and operation types motifs, and forecast future data accesses. Accordingly, we can set the data placement policy.

We have utilized linear programming to label the files against each location. Our goal is to minimize the overall communication cost (latency) between the locations maintaining the constraint that the accumulative size of the files at a location can not exceed its physical capacity. After we find the location for each file, we generate the training data set by plotting the number of accesses from each location for each file. To find the location of a new file, we first collect information for that new file from the so-far access frequencies. Then we utilize different classifiers to find the best location for the new files.

Table II summarizes the input parameters and the symbols used.

TABLE II: Parameter List

Parameter	meaning
n	Number of locations (home, office, school, etc.)
f	Number of files
$C_{i,j}$	Cost of unit data transfer between location i and j
P_i	Disk capacity of location i
S_i	Size of file i
$R_{i,j}$	Number of access of file i from location j
$X_{i,j}$	File i is placed in location j

Our objective here is to find the allocation matrix $\forall i, j \in n \;\; X_{i,j}$ so that the following equation is minimized:

$$min \sum_{i \in f} \sum_{j,k \in n} S_i C_{j,k} R_{f,j} X_{i,k} \tag{4}$$

subject to

$$\sum_{i \in f} X_{i,k} S_i \leq P_k \tag{5}$$

$$X_{i,k} \in \{0, 1\} \tag{6}$$

This problem is NP-Hard [14], so we relax Equation 6 and map this problem to fractional linear programming. From this fractional solution, we find the approximation by placing the file i to the location with highest value $X_{i,j}$. We use these optimal locations for some set of files as the training set, from which we find the optimal locations of new files via classifiers.

B. Preliminary Evaluation

We have implemented a prototype on Android so that users and applications can utilize our file system. To provide a unified view, we connect the user devices with HTML5 and WebSockets together with Amazon EC2. Whenever a user device comes up, it joins via the directory service hosted on EC2. Any new file created or relocated needs to update its location in the EC2 server using $< data, location >$ pair. We thus build a layer on top of the local file systems of one's participating devices. We store the unified file systems data in the local storage of the participating devices. From this $< data, location >$ metadata, we can locate the actual location

of the files. The actual transfer of the file takes places over the network. The placement of the data is calculated by our placement algorithm (fractional linear programming) discussed above for better availability and low latency.

We also implemented our file system mounting in the Android native function call level. The Android applications use the Java APIs, which use the Dalvik native calls to make system call for file operations. We trap the file system I/O and check whether the associated file is a local file or a remote file. In case of a local file, we call the standard system calls to perform the operation. Otherwise, the operation is redirected to the remote interface. We implemented the standard file operations, including `open`, `close`, `read`, `write`, etc. Upon opening a remote file, our system provides a virtual file handler to track subsequent operations associated with this file. Whenever the application performs any operations on the tracked file, our file system native module redirects the operations to the the APIs for remote communication. In this way, the remoteness is transparent to the application, requiring no modification to its source code.

We have evaluated our implementation with Yahoo Web-Scope cluster data [13]. This user trace includes different file access frequencies for different IPs. We have collected the probability distribution about the size of the files and their access frequencies. We have generated 1000 files and 10 locations according to the probability distribution. We could not collect the data about the bandwidth between the clusters (locations) and the capacity of them, so we followed a uniform distribution to generate those values.

We have clustered the files to locations by utilizing linear programming in `Matlab` and generated the dataset based on the location, file item, and frequency pattern. From the dataset, we use 2/3 as the training data and evaluate using the rest. Table III shows some preliminary results.

TABLE III: Accuracy of different classifiers

Name of the classifier	Accuracy
Naive Bayes	90.09%
Linear Regression	90.80%
SVM	91.00%
Decision Tree	93.50%
MLP	94.90%

Table III shows that all classifiers have high accuracy, which promises better performance. We have classified the dataset using standard `Weka` [12] library. Thus finding the optimal location for the files results in better throughput for file operations.

TABLE IV: Throughput for Local, Cloud, and Fog storage

Setup	Mbps
Local Storage	1.027
DropBox	2.83
Fog Storage	4.73

We use different benchmarks to evaluate our proposed framework and the file location policy. Due to page limit, here we mainly present the impact of network speed on file read operations based on an emulated user trace. In the local setting, we store all the files in the local sdcard of Android device. We also store the file in Dropbox for comparisons. In the distributed storage backed up by fog computing, represented by *Fog Storage* in the table, we place the files according to our location policy based on the user trace. Table IV shows that the read throughput is the best with the distributed storage service, denoted as fog storage in the table.

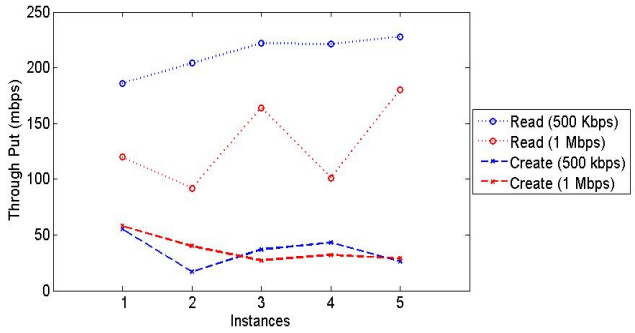

Fig. 4: The bandwidth has dominating affect on the file system's performance

Previously we have noted that the network is a dominating factor for such a file system. To empirically investigate its impact, we evaluate the response time for 5 trials with different network speed for file read or create operations. The results are presented in Figure 4. The results show that the network speed indeed has significant impact on the file operations, particularly for file read operations. In such cases, a nearby file server such as the one offered via fog networking can provide better performance.

C. Challenges

Similar to the fog computing resources for mobile application offloading, to practically employ such a distributed storage service, there are several challenges to overcome. For example, the service availability is still a challenge as a public cloud storage service often is available 7/24 while a user may power on and off her devices at different times. With a typical disk failure rate (4% [6]), keeping a proper number of replicates can alleviate this problem. For example, making one copy (of each data file) can lead to 99% availability in such a service [24]. This availability is comparable to Amazon services (with a failing rate 0.1-0.5% [6]). Security is another challenge as such a storage service requires remote access to different devices. Ad-hoc solutions could help with these issues while an open and comprehensive network protocol suite (e.g., via fog networking) could be more desirable.

IV. Related Work

Previous research [18], [30], [19], [29], [15], [21] has investigated how to partition mobile applications and offload computing-intensive tasks to the more powerful counterparts such as clouds. Balan et al. [15] focused on how easily applications can be modified for offloading. Odessa [29] showed that offloading can improve the response time for streaming and pipelined applications by three times. Thinkair [26] provides API for computation offloading which requires special compilation. Some other research [19], [27] focused on

application level partitioning. While making the offloading decision, MAUI [19] adopted a linear model to make the offloading decision, while Odessa [29] leveraged previous execution statistics and used the index of the ratio-of-benefit to make the offloading decision. Young et al. [27] proposed a static threshold to make offloading decisions. These works mainly focused on how efficiently applications can be offloaded. However, these works mainly tried to prove that cloud computing can improve the performance, which is hard for real world applications [21], [25] considering the overhead over the network. However, in our preliminary evaluation, we have shown that cloud computing is not a conclusive solution for mobile computation offloading. We also propose a framework to offload computation to either fog or cloud based on the available resources and dynamic conditions. Some preliminary results shows that most of the time, fog computing is better than cloud computing.

On the other hand, the previous work [11], [28], [31] related to the hybrid cloud storage for mobile devices was mainly motivated by data privacy issue. We are along the same direction to expand storage capacity of mobile devices via personal storage owned by the same user.

V. Conclusion and Future Work

Mobile devices are constrained by limited computing power and storage space. To address these limits, cloud computing and storage has been playing a critical role. However, cloud is not necessary the best solution all the time for all the mobile applications and data. In this paper, we explore the potentials of fog computing for mobile application offloading and storage expansion. For both of these applications, our experimental results show that indeed nearby resources may offer better performance.

To facilitate the utilization of nearby resources, for both computation offloading and data storage, there are a lot of challenges to be overcome. Not exclusively, the service availability, the security and privacy, the resource discovery, the service model, etc. are the remaining challenges to be overcome. Fog networking could shed some light on these issues.

VI. Acknowledgement

This work was mainly done when the first author was with George Mason University. This work is partially supported by National Science Foundation (NSF) under grant CNS-1117300.

References

[1] Amazon EC2 outage. http://www.informationweek.com/news/cloud-computing/infrastructure/229402054.

[2] Amazon S3. http://aws.amazon.com/s3/.

[3] Droidslator. http://code.google.com/p/droidslator/.

[4] Drop Box. http://www.dropbox.com/.

[5] DropBox security breach. http://www.newsy.com/videos/dropbox-security-glitch-leaves-users-accounts-unprotected/.

[6] EC2 Failure Rate. http://aws.amazon.com/ebs/.

[7] Google Docs. www.docs.google.com.

[8] Mark Zuckerbergs pictures leaked. http://www.nydailynews.com/news/national/oops-mark-zuckerberg-pictures-leaked-facebook-security-flaw-article-1.988026?localLinksEnabled=false.

[9] Picaso. http://code.google.com/p/picaso-eigenfaces/.

[10] Sky Drive. http://explore.live.com/skydrive.

[11] SME Storage. http://smestorage.com/.

[12] Weka. http://www.cs.waikato.ac.nz/ml/weka/.

[13] Yahoo Webscope. http://webscope.sandbox.yahoo.com/.

[14] David Applegate, Aaron Archer, Vijay Gopalakrishnan, Seungjoon Lee, and KK Ramakrishnan. Optimal content placement for a large-scale vod system. In *Proceedings of the 6th International COnference*, page 4. ACM, 2010.

[15] R. K. Balan, D. Gergle, M. Satyanarayanan, and J. Herbsleb. Simplifying cyber foraging for mobile devices. In *Proc. of Mobisys*, San Juan, Puerto Rico, June 2007.

[16] William J. Bolosky, John R. Douceur, David Ely, , and Marvin Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs. In *Sigmetrics*, 2000.

[17] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012.

[18] B.G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proc. of EuroSys*, pages 301–314, 2011.

[19] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. MAUI: Making smartphones last longer with code offload. In *Proc. of MobiSys*, San Francisco, CA, USA, June 2010.

[20] Ioana Giurgiu, Oriana Riva, and Gustavo Alonso. Dynamic software deployment from clouds to mobile devices. In *Middleware 2012*, pages 394–414. Springer, 2012.

[21] Mark S Gordon, D Anoushe Jamshidi, Scott Mahlke, Z Morley Mao, and Xu Chen. Comet: code offload by migrating execution transparently. In *OSDI*, 2012.

[22] M. A. Hassan and S. Chen. An investigation of different computing sources for mobile application outsourcing on the road. In *Proc. of Mobilware*, June 2011.

[23] Mohammed Hassan, Qi Wei, and Songqing Chen. Studying threshold based policy for computation offloading. In *Technical Report, Dept. of Computer Science, George Mason University, January 2015. http://www.cs.gmu.edu/∼sqchen/open-access/fast-tr.pdf*.

[24] Mohammed A Hassan, Kshitiz Bhattarai, and Songqing Chen. vups: Virtually unifying personal storage for fast and pervasive data accesses. In *Mobile Computing, Applications, and Services*, pages 186–204. Springer, 2013.

[25] Mohammed A. Hassan, Kshitiz Bhattarai, Qi Wei, and Songqing Chen. Pomac: Properly offloading mobile applications to clouds. In *6th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 14)*, Philadelphia, PA, June 2014. USENIX Association.

[26] Sokol Kosta, Andrius Aucinas, Pan Hui, Richard Mortier, and Xinwen Zhang. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *INFOCOM, 2012 Proceedings IEEE*, pages 945–953. IEEE, 2012.

[27] Y. W. Kwon and E. Tilevich. Power-efficient and fault-tolerant distributed mobile execution. In *Proc. of ICDCS*, 2012.

[28] Michelle L. Mazurek, Eno Thereska, Dinan Gunawardena, Richard Harper, , and James Scott. Zzfs: A hybrid device and cloud file system for spontaneous users. In *FAST*, 2012.

[29] M.R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan. Odessa: enabling interactive perception applications on mobile devices. In *Proc. of Mobisys*, pages 43–56. ACM, 2011.

[30] M. Satyanarayanan, P. Bahl, R. Caceres, and N.l Davies. The case for VM-based cloudlets in mobile computing. In *IEEE Pervasive Computing*, volume 8(4), October 2009.

[31] Jacob Strauss, Justin Mazzola Paluska, Bryan Ford, Chris Lesniewski-Laas, Robert Morris, and Frans Kaashoek. Eyo: Device-transparent personal storage. In *USENIX Technical Conference*, 2011.

[32] Ying Zhang, Gang Huang, Xuanzhe Liu, Wei Zhang, Hong Mei, and Shunxiang Yang. Refactoring android java code for on-demand computation offloading. In *ACM SIGPLAN Notices*, volume 47, pages 233–248. ACM, 2012.