



**HAL**  
open science

## Mobile cloud computing for computation offloading: Issues and challenges

Khadija Akherfi, Micheal Gerndt, Hamid Harroud

► **To cite this version:**

Khadija Akherfi, Micheal Gerndt, Hamid Harroud. Mobile cloud computing for computation offloading: Issues and challenges. Applied Computing and Informatics, 2018, 14 (1), pp.1-16. hal-01767212

**HAL Id: hal-01767212**

**<https://hal.science/hal-01767212>**

Submitted on 15 Apr 2018

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

---

# Mobile cloud computing for computation offloading: Issues and challenges

Khadija Akherfi <sup>a,\*</sup>, Micheal Gerndt <sup>a</sup>, Hamid Harroud <sup>b</sup>

<sup>a</sup> *Technical University of Munich, TUM, Munich, Germany*

<sup>b</sup> *Al Akhawayn University in Ifrane, Ifrane, Morocco*

## KEYWORDS

Cloud computing;  
Mobile cloud computing;  
Computational offloading;  
Algorithms;  
Partitioning

---

**Abstract** Despite the evolution and enhancements that mobile devices have experienced, they are still considered as limited computing devices. Today, users become more demanding and expect to execute computational intensive applications on their smartphone devices. Therefore, Mobile Cloud Computing (MCC) integrates mobile computing and Cloud Computing (CC) in order to extend capabilities of mobile devices using offloading techniques. Computation offloading tackles limitations of Smart Mobile Devices (SMDs) such as limited battery lifetime, limited processing capabilities, and limited storage capacity by offloading the execution and workload to other rich systems with better performance and resources. This paper presents the current offloading frameworks, computation offloading techniques, and analyzes them along with their main critical issues. In addition, it explores different important parameters based on which the frameworks are implemented such as offloading method and level of partitioning. Finally, it summarizes the issues in offloading frameworks in the MCC domain that requires further research.

© 2016 Production and hosting by Elsevier B.V. on behalf of King Saud University. This is an open access article under the CC BY-NC-ND license (<http://creativecommons.org/licenses/by-nc-nd/4.0/>).

---

## Contents

1. Introduction . . . . .	00
2. Concepts and background . . . . .	00
2.1. Cloud computing . . . . .	00
2.2. Mobile cloud computing . . . . .	00

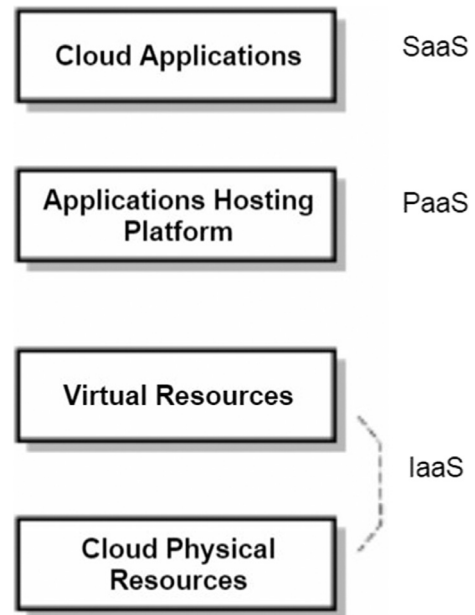
24	2.3. Computation offloading . . . . .	00
25	3. Offloading approaches . . . . .	00
26	3.1. Offloading steps . . . . .	00
27	3.1.1. Application partitioning . . . . .	00
28	3.1.2. Preparation . . . . .	00
29	3.1.3. Offloading decision. . . . .	00
30	3.2. Framework classes . . . . .	00
31	3.3. Framework mechanisms . . . . .	00
32	4. Comparison of offloading frameworks in mobile cloud computing. . . . .	00
33	4.1. CloneCloud . . . . .	00
34	4.2. MAUI. . . . .	00
35	4.3. Cloudlet. . . . .	00
36	4.4. Jade. . . . .	00
37	4.5. Mirror server . . . . .	00
38	4.6. Cuckoo . . . . .	00
39	4.7. Phone2Cloud . . . . .	00
40	4.7.1. A comparative review of some offloading frameworks . . . . .	00
41	5. General issues and challenges in computation offloading for MCC . . . . .	00
42	5.1. Platform diversity . . . . .	00
43	5.2. Security and privacy in mobile cloud applications . . . . .	00
44	5.3. Fault-tolerance and continuous connectivity . . . . .	00
45	5.4. Automatic mechanism. . . . .	00
46	5.5. Offloading economy/cost . . . . .	00
47	5.6. Partition offloading and external data input . . . . .	00
48	6. Conclusion . . . . .	00
49	Acknowledgments . . . . .	00
50	References . . . . .	00

**1. Introduction**

The main goal of CC is to allow IT departments to focus on their businesses and projects instead of just taking care of their data centers and keeping them working [2,18,20]. CC is a new concept that aims to provide computational resources as services in a quick manner, on demand, and paying as per usage. The CC paradigm is presented in three cloud delivery models: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS) as shown in Fig. 1. According to Gartner [3], CC will have in 2016 a Global Compounded Annual Growth Rate (CAGR) of IaaS: 41%, PaaS: 26.6% and SaaS: 17.4%.

Recently, user preferences for computing have changed because of the latest developments and enhancements in mobile computing technologies. Several reports and studies have presented the importance of MCC and its impact on mobile clients and enterprises. For instance, and according to a recent study by ABI Research, more than 240 million business will use cloud services through mobile devices by 2015 and this will push the revenue of the MCC to \$5.2 billion [11]. Moreover, the usage of smartphones has increased rapidly in various domains, including enterprise, management of information systems, gaming, e-learning, entertainment, gaming, and health care. Although the predictions that mobile devices will be dominating the future computing devices, mobile devices along with their applications are still restricted by some limitations such as the battery life, processor potential, and the memory capacity of the SMDs [31]. Nowadays, modern mobile devices have sufficient resources such as fast processors, large memory, and sharp screens. However, it is still

not enough to help with computing intensive tasks such as natural language processing, image recognition, and decision-making. Mobile devices provide less computational power comparing to server computers or regular desktops and computation-intensive tasks put heavy loads on battery power.



**Figure 1** Cloud computing layers.

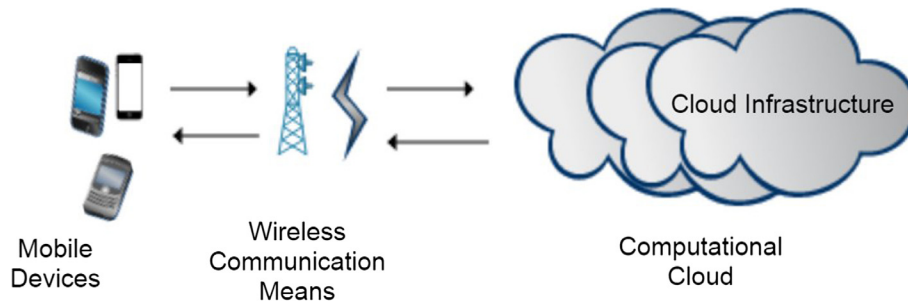


Figure 2 General view of MCC.

Currently, there are several works and research in CC that aim at enhancing the computing capabilities of resource-constrained mobile client devices by providing mobile clients access to cloud infrastructures, software, and computing services. For example, Amazon web services are used to protect and save clients' personal data via their Simple Storage Service (S3) [23]. In addition, there are several frameworks that allow to process data intensive tasks remotely on cloud servers. For instance, the ASM computation offloading framework [6] showed that computation offloading helped to reduce the energy consumption cost of mobile devices by 33%, and the turnaround time of the application by 45% [30].

The following two points highlight our main contribution in this paper.

1. Classifying current computation offloading frameworks. Analyzing them by identifying their approaches and crucial issues.
2. Presenting the related open issues in computation offloading for MCC and challenges that require more investigation and elaboration.

This paper is organized as follows: Section 2 explains the essential background concepts and terminology, including CC, the MCC concept, and computation offloading. Section 3 presents the offloading approaches. A comparison between the different offloading frameworks and their critical issues is discussed in Section 4. Section 5 highlights general issues and challenges in computation offloading for MCC. Finally, Section 6 gives a summary and points to future work.

## 2. Concepts and background

### 2.1. Cloud computing

CC is a new way of providing computing resources and services. It refers to an on-demand infrastructure that allows users to access computing resources anytime from anywhere [25]. CC offers to users and business three main advantages: (1) enormous computing resources available on demand, (2) payment for use as needed and on a short-term basis (storage by the day and release them as needed), and (3) simplified IT management and maintenance capabilities [1]. CC provides clients with different applications as services via the Internet. As examples of public CC we can list Windows Azure and Amazon Web Services (AWS). Windows Azure is an open and flexible cloud platform which provides several services to develop,

deploy and run web applications and services in cloud data centers [33]. AWS, which is considered as an example of a public computing tool, provides users with two models: infrastructure as a service and software as a service. These services allow the user to use virtualized resources in cloud data centers [23]. Computational clouds implement a variety of service models in order to use them in different computing visions [4].

### 2.2. Mobile cloud computing

MCC can be seen as a bridge that fills the gap between the limited computing resources of SMDs and processing requirements of intensive applications on SMDs.

The Mobile Cloud Computing Forum defines MCC as follows [11]: "Mobile Cloud Computing at its simplest form refers to an infrastructure where both the data storage and the data processing happen outside of the mobile device. Mobile cloud applications move the computing power and data storage away from mobile phones and into the cloud, bringing applications and mobile computing to not just smartphone users but a much broader range of mobile subscribers".

MCC has attracted the attention of business people as a beneficial and useful business solution that minimizes the development and execution costs of mobile applications, allowing mobile users to acquire latest technology conveniently on an on-demand basis.

Fig. 2 shows the general view of MCC which is composed of three main parts: the mobile device, wireless communication means, and a cloud infrastructure that contains data centers. These latter provide storage services, processing, and security mechanisms for both the cloud environment and mobile devices.

### 2.3. Computation offloading

Computation offloading is the task of sending computation intensive application components to a remote server. Recently, a number of computation offloading frameworks have been proposed with several approaches for applications on mobile devices. These applications are partitioned at different granularity levels and the components are sent (offloaded) to remote servers for remote execution in order to extend and enhance the SMD's capabilities. However, the computation offloading mechanisms are still facing several challenges.

In the remaining part of this section, our objective was to give a summary about the MCC offloading research by discussing the following:

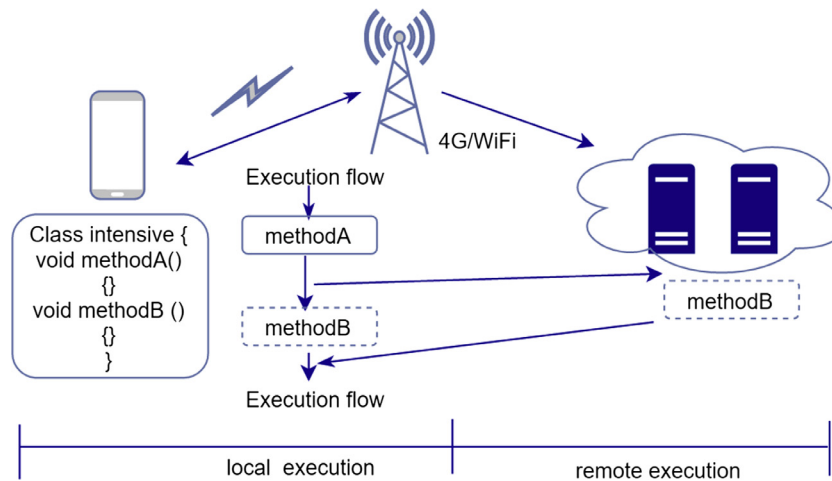


Figure 3 Offloading process overview.

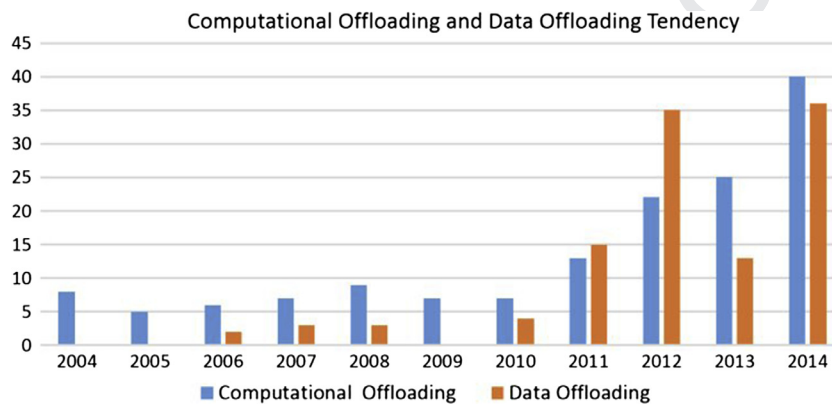


Figure 4 Number of computation offloading and data offloading papers.

- 174 1. Usage scenarios for offloading in MCC.
- 175 2. Techniques being applied in offloading.
- 176 3. A classification of proposed offloading frameworks.

177

178 Computation offloading emerged around 1970s. However,  
 179 its potential has been widely explored only when wireless communication and Internet speed became sufficiently enhanced  
 180 and could support it [37].

181 The potential of mobile offloading mainly depends on the  
 182 mobile network technologies such as cellular and WiFi. They  
 183 determine the viability of mobile offloading. Today, the WiFi  
 184 technology is able to provide high bandwidth connections.  
 185 However, the data transmission using the cellular network  
 186 requires a considerable amount of energy from the mobile  
 187 device as opposed to a WiFi network.

188 Fig. 3 illustrates the environment that supports computation  
 189 offloading. In this overview, the mobile device decides  
 190 to offload method B to a cloud server or a powerful machine.  
 191 The cloud here provides the virtual computation resources to  
 192 run the offloaded components. The powerful machine can be  
 193 a server or cluster in a computing center, or a computing grid,  
 194 or a virtual server in the cloud. Fig. 4<sup>1</sup> shows the number of  
 195 published papers since 2004 citing the word “Offloading”  
 196

197 and “Computation”. Most of the research works tackling data  
 198 offloading have the goal to store data in remote large repositories.  
 199 It can be seen in Fig. 4 that the research work citing “computation  
 200 offloading” and “data offloading” is increasing progressively.  
 201

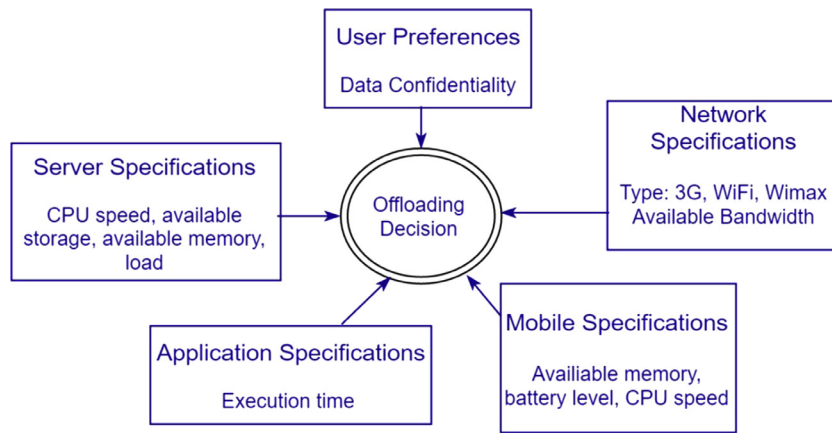
202 Clearly, computation offloading is worthwhile only when  
 203 the local execution (mobile device) consumes more time and  
 204 energy than the offloading overhead. Many factors can impact  
 205 the offloading decision and could influence the offloading  
 206 process.

207 Fig. 5 illustrates these factors which are network specifications,  
 208 mobile specifications, application characteristics, server  
 209 specifications, and user’s preferences.

210 The computation offloading has experienced a remarkable  
 211 improvement which makes it applicable in a wide range of  
 212 domains. Table 1 shows some of these domains.

213 As it is known, the battery life of mobile devices and the  
 214 limited processors’ capabilities remain key limiting factors in  
 215 the design of mobile applications. Today, the demand for  
 216 resource intensive applications such as 3D video games and  
 217 voice recognition is increasing day by day. To close this gap  
 218 between the users demand and the mobile devices limitations,  
 219 research studies have been exploring computation offloading  
 220 in MCC to bring the power of cloud computing to the other-  
 221 wise limited mobile devices capacity.

<sup>1</sup> The number of published works is retrieved from Google Scholar.



**Figure 5** Aspects affecting the offloading decision.

**Table 1** Areas of the research work related to computation offloading.

Domain	Research work	Contribution of some research work
Artificial intelligence based applications	[6,8,16,27,37]	The proposed work aims to reduce computation time on robots by using an offloading technique. The system is designed for real-time moving object recognition and tracing using computation offloading [27]
Graphics and image processing	[6,37,17,35]	The suggested work examines the trade-offs that emerge from executing some of the workload locally and some on remote cloud servers. Extraction and matching are two features that are crucial in image classification. The paper analyzes the possibility of executing the previously mentioned features in a mobile device using different scenarios [17]
Health and social applications	[24,21]	The proposed work presents an extensible module that proactively facilitates the management of processes for web service supported by mobile applications. This module measures power consumption and application performance from the smartphone device. Based on the obtained measurements, the module dispatches image processing jobs locally or remotely [24]
Games applications	[35]	The objective of the presented work was to satisfy Cloud Mobile Gaming communication and computation constraints by using an adaptation technique that ensures a good mobile gaming experience [35]
Mathematics	[7,34]	The work presents a new architecture that addresses the mobile device limitations by using a partial offloading execution from the smartphone to a remote cloud infrastructure hosting smartphone clones [7]
File system and database	[26,7]	The presented work addresses the maximization of the lifetime of mobile devices by developing computational offloading schemes while considering the network status. The paper presents an experimental environment where different profiles and computational moles are evaluated [26]

### 3. Offloading approaches

#### 3.1. Offloading steps

Offloading transfers a compute intensive task from the SMD to a remote server. Offloading is a process that includes basically three steps: application partitioning, preparation, and offloading decision.

##### 3.1.1. Application partitioning

The first step is application partitioning which is very important for computation offloading. It divides the application into offloadable and non-offloadable components meaning which components to retain on the mobile device and which to migrate to the cloud server. The decision whether a component is offloadable can be taken based on different information. The

programmer can annotate application components for example through a special API as offloadable. Compute intensive parts that are candidates for offloading can be identified also by source code analysis in combination with performance prediction or via application profiling. If the partitioning is done at design time, both techniques have a limited accuracy since they do not take the real execution context into account, when the application is run.

##### 3.1.2. Preparation

The preparation step performs all actions required for offloadable components to enable their use in mobile applications. This includes the selection of a remote server, the transfer and installation of the code, as the start of proxy processes that receive and execute tasks on behalf of the SMD. Besides the transfer of the code, also data might be transferred to prepare for the remote execution.

222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234

235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250

### 251 3.1.3. Offloading decision

252 The offloading decision is the final step before remote execu-  
253 tion is started for offloadable components. Whether an  
254 installed remote component is used in the SMD application  
255 or not depends typically on the execution context. If the deci-  
256 sion is taken at runtime, more precise information is available,  
257 for example, the SMD might even not have a wireless connec-  
258 tion or the energy consumption for transferring the data for  
259 the remote execution might simply be too high. Whenever  
260 the situation changes, the offloading can be adapted. Such a  
261 runtime decision induces some overhead that typically is not  
262 present if the decision is taken at design time.

### 263 3.2. Framework classes

264 According to when the decision is taken to offload computation  
265 on a remote server, we can distinguish two types of offloading  
266 frameworks. The first class is static offloading frameworks.  
267 Here all the presented steps are performed at design time,  
268 before the application is started on the mobile device. The other  
269 classes are dynamic offloading frameworks. In those frame-  
270 works, at least the final decision whether to offload a computa-  
271 tion is taken at runtime. The other two steps can be executed at  
272 design or runtime. For example, a framework that is based on  
273 user annotations of offloadable components and on pre-  
274 installation of the components on a remote server will be called  
275 a dynamic offloading framework, if it decides at runtime  
276 whether it is better to offload computation or not.

### 277 3.3. Framework mechanisms

278 Although there are several offloading mechanisms available  
279 for offloading computation intensive components of mobile  
280 applications to the cloud, we can classify these mechanisms  
281 into two broad categories:

- 282 1. Frameworks based on virtual machine cloning.
- 283 2. Frameworks based on code offloading.

284  
285 Frameworks based on code offloading offload intensive  
286 application components by invoking a remote procedure call  
287 (RPC) using annotations, special compilation or binary modifi-  
288 cation. Whereas in virtual machine cloning, the mobile device's  
289 full image is captured and stored on the cloud server. During  
290 offloading, the mobile's execution is suspended and  
291 transferred to the VM clone in the cloud.

## 292 4. Comparison of offloading frameworks in mobile cloud 293 computing

294 This section introduces different existing offloading frame-  
295 works. For each of the frameworks we identify the approaches  
296 used in the three steps introduced in the previous section. At  
297 the end of the section, the different frameworks will be com-  
298 pared with respect to their most important properties.

### 299 4.1. CloneCloud

300 Chun et al. present in [6] the CloneCloud framework which  
301 aims at improving the battery life and performance on the

mobile device by offloading intensive components to cloud  
302 servers.

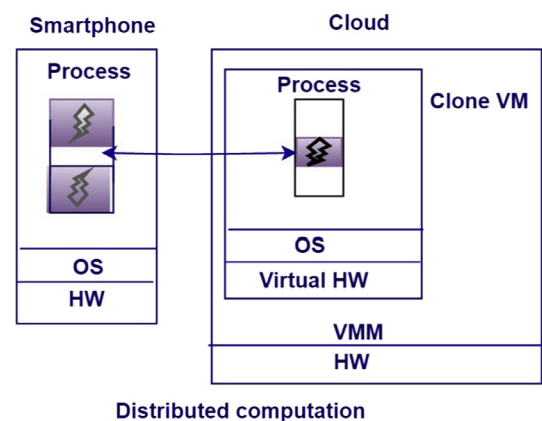
303  
304 The partitioning step in this framework combines static  
305 program analysis with program profiling to produce a set of  
306 offloadable components while meeting some constraints, such  
307 as methods that use mobile sensors should be executed locally.  
308 The framework uses thread level granularity for partitioning of  
309 applications. The role of static analysis is to discover con-  
310 straints on possible migration points while profiling aims to  
311 build a cost model for offloading and execution. Partitioning  
312 and integration of the applications are performed at the appli-  
313 cation level.

314 As a preparation step, a duplicate of the smartphone's soft-  
315 ware is captured and stored within a cloud server.

316 At runtime, the offloading decision is taken and threads are  
317 migrated from the mobile device to the clone in the cloud. In  
318 CloneCloud, threads must be suspended, all states of the  
319 threads must be transferred to the server, and then threads  
320 resume on the server in order to offload computation. The  
321 framework is based on VM instance migration to the cloud  
322 server. Fig. 6 shows the CloneCloud execution model. Initially,  
323 a duplicate of the smartphone's software is created in the  
324 cloud. The state of the smartphone and the clone is synchron-  
325 ized periodically or on-demand. After the execution of off-  
326 loaded components, results from the execution on the clone  
327 are reintegrated back into the smartphone state. CloneCloud  
328 employs dynamic offloading.

329 The objective of the distributed execution mechanism in  
330 CloneCloud was to implement a specific partition of a given  
331 application process executing inside an application-layer vir-  
332 tual machine.

333 In CloneCloud framework, the distributed execution goes  
334 through several steps as follows. When the user tries to run a  
335 partitioned application, the framework looks in a database  
336 of pre-computed partitions for current execution conditions  
337 (such as available network bandwidth and cloud resources).  
338 The result of the verification is a partition configuration file.  
339 The partition is loaded by the application binary which instru-  
340 ments the selected methods with migration. On the mobile  
341 device, once the execution of the process reaches a migration  
342 point, the running thread is suspended and its state is wrapped  
343 and shipped to a synchronized clone. In this clone, the thread  
344 state is instantiated into a new thread with the same stack and



Distributed computation

Figure 6 CloneCloud execution model (adapted from [6]).

345 reachable heap objects, and then resumed. On the cloud server,  
 346 when the migrated thread reaches a re-integration point, it is  
 347 suspended, packaged, and then shipped back to the mobile  
 348 device. Finally, the received packaged thread is merged into  
 349 the state of the original process in the mobile device.

350 To evaluate the prototype, the authors implemented three  
 351 applications (virus scanner, image search, and privacy-  
 352 preserving targeted advertising). All measurements are the  
 353 average of five executions. Phone, CloneCloud with WiFi,  
 354 and CloneCloud with 3G are the used environments.

355 A clear tendency is that larger workloads benefit from  
 356 offloading because of amortization of the migration cost over  
 357 a larger computation [6].

#### 358 4.2. MAUI

359 MAUI [8] is a framework that considers energy saving on  
 360 smartphones as the main objective function for the offloading  
 361 process. MAUI is a highly dynamic offloading framework  
 362 because of a continuous profiling process. The framework  
 363 hides the complexity of a remote execution from the mobile  
 364 user and gives the impression as if the entire application is  
 365 being executed on the mobile device.

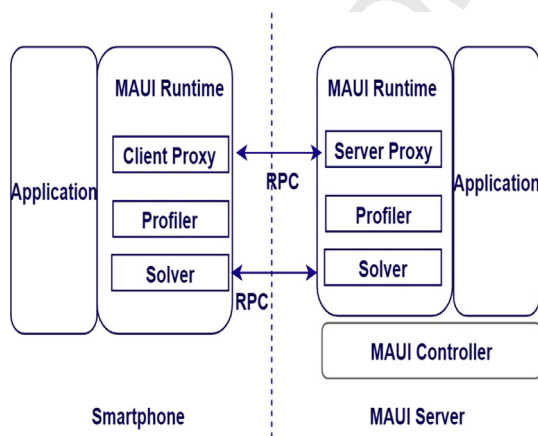
366 In MAUI, partitioning is done based on developer annota-  
 367 tions to specify which components can be executed remotely  
 368 and which cannot.

369 In the preparation step, two requirements should be met:  
 370 (1) application binaries must be in both mobile and server sides  
 371 and (2) proxies, profilers and solvers must be installed on both  
 372 the mobile device and server sides.

373 At the beginning, MAUI profiler measures the device char-  
 374 acteristics. Then, it keeps monitoring the program and net-  
 375 work characteristics during the whole execution time because  
 376 these characteristics can often change and any old or inaccur-  
 377 ate measurement may lead MAUI to make the wrong  
 378 decision.

379 The offloading decision is taken at runtime. The framework  
 380 chooses which components should be remotely executed  
 381 according to the decision of the MAUI solver. The decision  
 382 is based upon the input of the MAUI profiler.

383 Fig. 7 shows the MAUI architecture. On the smartphone,  
 384 the framework consists of the following components: a proxy,  
 385 a profiler, and a solver. Each time a method is called, the



386 **Figure 7** High-level view of MAUI's architecture (adapted from  
 387 [8]).

386 MAUI profiler evaluates it for its energy-saving potential  
 387 and profiles the device and the network to obtain the status  
 388 information. Then, the MAUI solver works on the results pro-  
 389 vided by the profiler and determines the destination where the  
 390 method will be remotely executed; the proxy is responsible for  
 391 control and data transfer between the server and the smart-  
 392 phone. On the server side, the profiler and the server proxy  
 393 perform similar roles as their client-side counterparts. The  
 394 MAUI controller is responsible for authentication and  
 395 resource allocation for incoming requests [8].

396 The authors presents different experiments in order to com-  
 397 pare the energy consumption of running three applications(-  
 398 face recognition, chess, and video) standalone on the  
 399 smartphone versus using MAUI for remote execution to ser-  
 400 vers that are located elsewhere. The face recognition applica-  
 401 tion can achieve strong energy savings when using MAUI.  
 402 On the one hand, the results of the conducted experiments  
 403 showed that the energy consumed when offloading code using  
 404 3G is 2.5 times higher than offloading code to a close server.  
 405 On the other hand, the energy savings for both video and chess  
 406 game are less strong but they are still important; when offload-  
 407 ing to a close server, MAUI saves 45% for chess and 27%  
 408 energy for the video game.

#### 409 4.3. Cloudlet

410 Offloading to the cloud is not always a solution, because of the  
 411 high WAN latencies, mainly for applications with real-time  
 412 restrictions. Thus the cloud has to be moved closer to the  
 413 mobile user in the form of cloudlets.

414 Satyanarayanan et al. suggest in [29] a VM based cloudlet  
 415 framework. A cloudlet can be defined as a hosting environ-  
 416 ment for offloaded tasks that is deployed to remote resources,  
 417 as different as individual servers or parallel systems. Cloudlets  
 418 are virtual-machine (VM) based on support scalability, mobil-  
 419 ity, and elasticity. They are located in single-hop nearness to  
 420 mobile devices.

421 In the preparation step, the framework requires the cloning  
 422 of the mobile device application processing environment to a  
 423 remote host. It offloads the entire application using VM as  
 424 the offloading mechanism and more precisely it uses a tech-  
 425 nique called dynamic VM synthesis. The VM would encapsu-  
 426 late and separate the guest software from the cloudlet's host  
 427 software. The mobile device serves as a thin client providing  
 428 only the user interface, whereas the actual application process-  
 429 ing is performed on the cloudlet infrastructure.

430 Device mobility is the main critical issue for mobile users on  
 431 the move while connected to cloudlets.

432 As Fig. 8 illustrates, cloudlets are widely distributed Inter-  
 433 net infrastructure components whose storage resources and  
 434 computing cycles can be exploited by nearby mobile devices  
 435 while avoiding the long latency which is available for accessing  
 436 distant cloud resources. These cloudlets would be situated in  
 437 common areas, such as coffee shops, so that mobile devices  
 438 can connect and work as a thin client.

439 Fig. 9 depicts the cloudlet architecture. Cloudlets are dis-  
 440 coverable, and located in single-hop proximity of mobile  
 441 devices. The main elements of the architecture are Cloudlet  
 442 Host and Mobile Client. A Discovery Service is a component  
 443 running in the cloudlet host and publishes cloudlet metadata.  
 444 The cloudlet metadata (e.g. IP address and port to connect



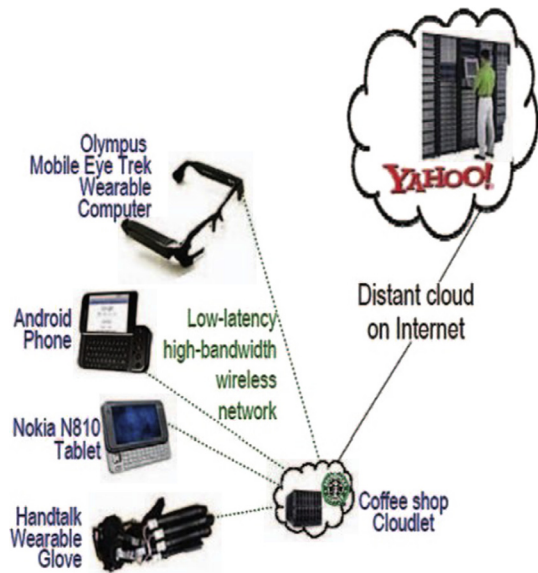


Figure 8 Cloudlet illustration adapted from [29].

to the cloudlet) are used by the mobile client to specify the suitable cloudlet for computation offloading. Once the cloudlet is determined for offload, the mobile client sends the application code and the application metadata to the cloudlet server. The cloudlet server deploys the application code in the guest VM. Once the deployment is done, the execution of the application is launched.

We can take a scenario where the user must execute a computation intensive application. At runtime, the application discovers a nearby cloudlet and offloads the computation intensive mobile application [15]. However, because of loss of network connectivity, the mobile application can find a different cloudlet and run the application in a short time.

#### 4.4. Jade

Sharing the same concern but from a different perspective, Qian et al. present in [28] a system that monitors application and device status and that automatically decides where the code should be executed. The goal of Jade was to maximize the benefits of energy-aware computation offloading for mobile applications while minimizing the burden on developers to build such an application.

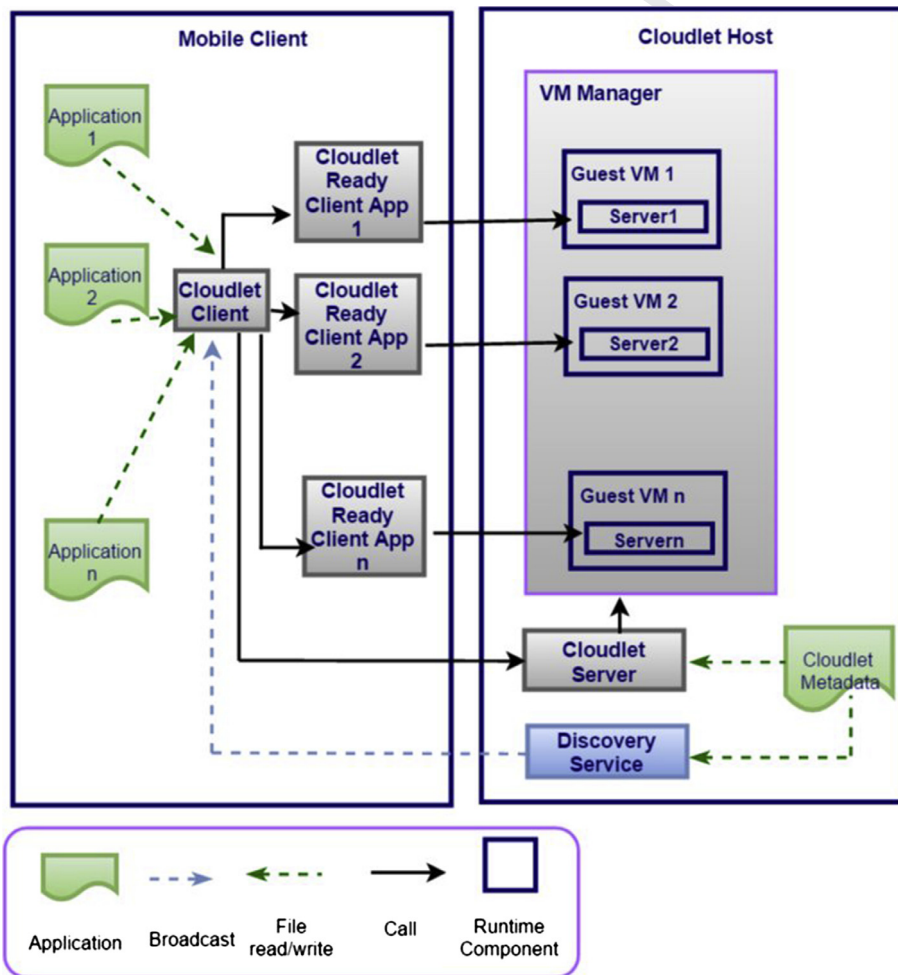


Figure 9 Cloudlet architecture (adapted from [22]).

466 During partitioning, applications are partitioned at the  
467 class level in Jade based on the collected information.

468 As a preparation, the system checks the application and  
469 device status by monitoring the communication costs, work  
470 load variation, and energy status. The framework provides a  
471 sophisticated programming model with a full set of APIs, so  
472 developers have total control on: how the application is parti-  
473 tioned, and how remote code interacts with local code.

474 The offloading decision is taken at runtime to decide where  
475 the code should be executed. Jade supports two types of servers:  
476 (1) Android servers and (2) Non-Android servers running  
477 operating systems such as Windows and Linux. Non-Android  
478 servers must have Java installed in order to support Jade.  
479 Jade's runtime engine runs as a Java program on a non-  
480 Android server.

481 Jade can dynamically change its offloading decision accord-  
482 ing to the device status and thus efficiently reduce energy con-  
483 sumption of mobile devices.

484 Fig. 10 presents an overview of the Jade framework. The  
485 mobile device that offloads computation is called the client.  
486 The device that executes the offloaded code is called the server.  
487 Mobile applications contain remote tasks which can be off-  
488 loaded to the server. The Jade runtime engine automatically  
489 decides where to execute remote tasks and initiates distributed  
490 execution.

491 Fig. 11 presents the Jade framework architecture. In order  
492 to offload a computation, the system handles the following  
493 tasks:

- 494 • Profiling: In order to make correct offloading decisions, the  
495 framework should have updated information concerning  
496 the status of the application and the device. Application  
497 profiling is the process of collecting information about pro-  
498 grams, such as energy consumption, data size, execution  
499 time, and memory usage. Similarly, device profiling collects  
500 information about devices status, such as battery level,  
501 CPU usage, and wireless connection.
- 502 • Communication: To offload code from the mobile client to  
503 the server, the system should be able to (1) connect to the  
504 other server; (2) coordinate with the remote server for off-

505 loaded components; (3) send data between the mobile  
506 device and the remote server; (4) follow status of remote  
507 execution; and (5) save information related to all connected  
508 devices (e.g., connection speed, hardware configuration).

- 509 • Optimization: The framework determines an optimized  
510 offloading approach to reduce energy consumption and  
511 enhance application's performance.

512 The Jade framework automatically transforms application  
513 execution on one mobile device into a remote execution opti-  
514 mized for wireless connection, power usage, and server  
515 capabilities.

516 In order to check the amount of energy that Jade can save  
517 for mobile device, authors run face detection application on a  
518 mobile device. The application performs face detection on 50  
519 pictures with size of each less than 200 KB. Results showed  
520 that Jade reduces the power consumption for face detection  
521 application. Average power consumption was decreased by  
522 34%.

#### 524 4.5. Mirror server

525 Zhao et al. present in [38] the mirror server framework that  
526 uses Telecommunication Service Provider (TSP) based remote  
527 services. A TSP is a type of communication service provider  
528 which provides voice communication services such as landline  
529 telephone services. Mirror server extends capabilities of smart-  
530 phones by providing three different types of services: computa-  
531 tion offloading, security, and storage. Mirror server is a  
532 powerful server which retains VM templates for different  
533 mobile device platforms.

534 This framework does not require a partitioning as the entire  
535 application is offloaded.

536 In the preparation step, a new VM instance is created. This  
537 VM is called mobile mirror and the mirror server takes care of  
538 managing and deploying the mobile mirrors on a computing  
539 infrastructure in the telecom network. Applications are execu-  
540 ted in the mirror VM instances and results are returned to  
541 the SMD. The framework employs an optimized mechanism  
542 for offloading.

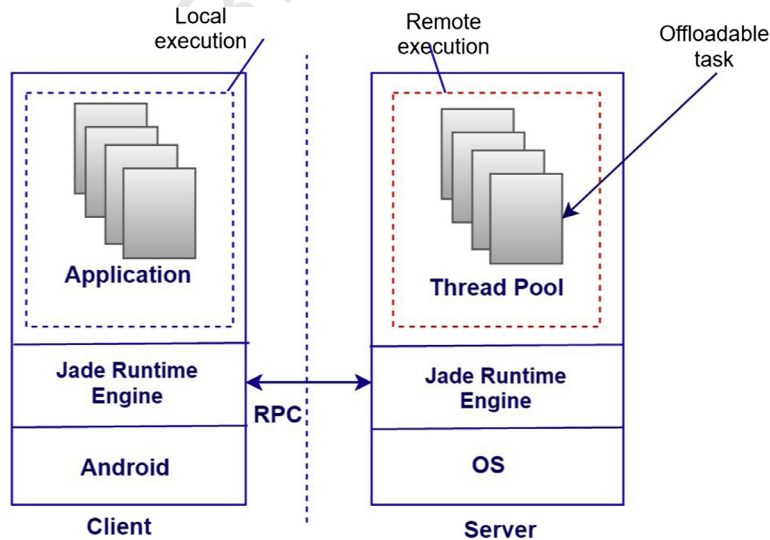


Figure 10 Jade overview (adapted from [29]).

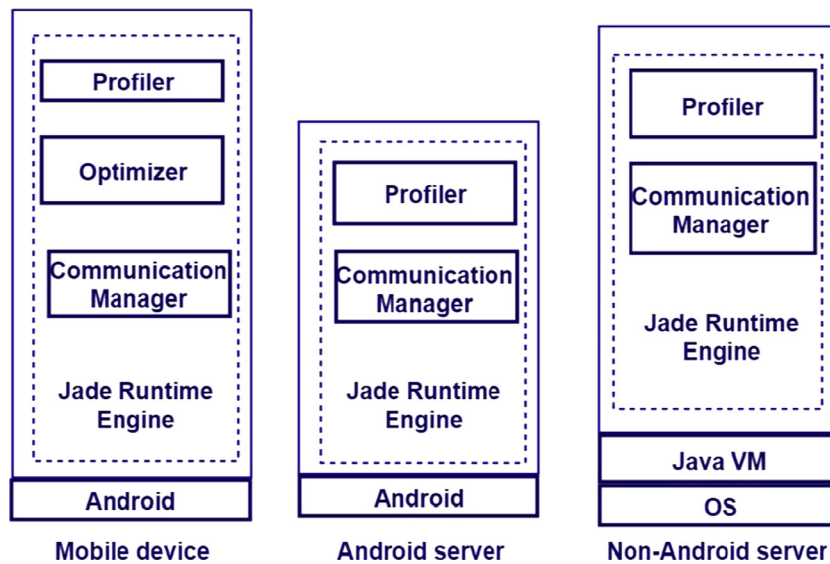


Figure 11 Jade architecture (adapted from [28]).

543 The Mirror Server architecture is presented in Fig. 12. On  
 544 the SMD side, a client synchronization module (Syn-Client)  
 545 is deployed within the SMD operating system (OS) to collect  
 546 SMD input data and transmit them to the mirror server for  
 547 synchronization. On the server side, in order to keep mirrors  
 548 and smartphones synchronized, the synchronization module  
 549 (Syn-Server) updates mirrors according to the data provided  
 550 by Syn-Client and the Traffic Monitor module which monitors  
 551 network traffic between the smartphone and the IP network.

552 The main critical issue is that mirror servers are not  
 553 designed for data processing and because of that only limited  
 554 services (i.e. file caching, file scanning) can be provided com-  
 555 pared to the variety of services in cloud data centers.

556 In the proposed framework, antivirus scanner application  
 557 can be deployed as a service on the mirror server, and the  
 558 application can access the file system on mirrors. The benefits  
 559 of sending antivirus scanner to the mirrors are significant:

- 560 1. It saves battery power on smartphones.
- 561 2. Scanning will not affect the common usages of smartphone  
 562 devices since the CPU and I/O intensive workload are  
 563 moved to the mirror server.
- 564 3. Running the scan on mirror will be much faster than that  
 565 on the phone due to its limited resources.

#### 567 4.6. Cuckoo

568 Kemp et al. present in [19] the Cuckoo framework for compu-  
 569 tation offloading for smartphones. This framework offloads  
 570 mobile device applications onto a cloud server using a Java  
 571 stub model. Cuckoo's objectives were to enhance mobile's per-  
 572 formance and reduce battery usage. The framework integrates  
 573 the Eclipse development tool with the open source Android  
 574 framework.

575 In the partitioning step, Cuckoo takes advantage of the  
 576 existing activity model in Android which makes the separation  
 577 between the intensive and non-intensive components of the  
 578 application. This activity presents a graphical user interface

579 to the user, and is able to bind to services. The framework  
 580 can offload intensive components to any resource running a  
 581 Java Virtual machine (JVM).

582 As a preparation, the framework requires the developer to  
 583 write offloadable methods twice - one for local computations  
 584 and one for remote computations. For this purpose, a pro-  
 585 gramming model is made available to application developers.  
 586 This programming model is used for dropped connection, sup-  
 587 ports local and remote execution, and combines all codes in a  
 588 single package so the user will have a compatible remote  
 589 implementation.

590 Cuckoo is a dynamic offloading framework as it takes the  
 591 offloading decision at runtime and offloads the well-defined  
 592 components of the application. In case the remote resources  
 593 are not reachable (i.e. network connection is not available)  
 594 then the application can be executed on local resources (the  
 595 mobile device).

#### 596 4.7. Phone2Cloud

597 Xia et al. present in [36] a computation offloading framework  
 598 called Phone2Cloud. The objective was to improve energy effi-  
 599 ciency of smartphones and improve the application's perfor-  
 600 mance. Unlike the previous frameworks, authors focus on  
 601 conducting a fully quantitative analysis on energy saving of  
 602 the system by conducting application experiments and scenario  
 603 experiments.

604 Phone2Cloud is a semi-automatic offloading framework. In  
 605 order to run applications on the cloud and receive the results,  
 606 applications need to be manually modified during preparation  
 607 step to make it possible to be executed on cloud servers.

608 The offloading decision is based on a static analysis while  
 609 considering user's delay-tolerance threshold.

610 For delay tolerant applications, the framework uses a simple  
 611 model to expect WiFi connectivity.

612 The threshold is defined based on predictions to delay  
 613 transfers in order to offload more data on WiFi while respect-  
 614 ing the application's tolerance threshold [42]. The framework  
 615 will wait for WiFi (only if 4G savings are expected within

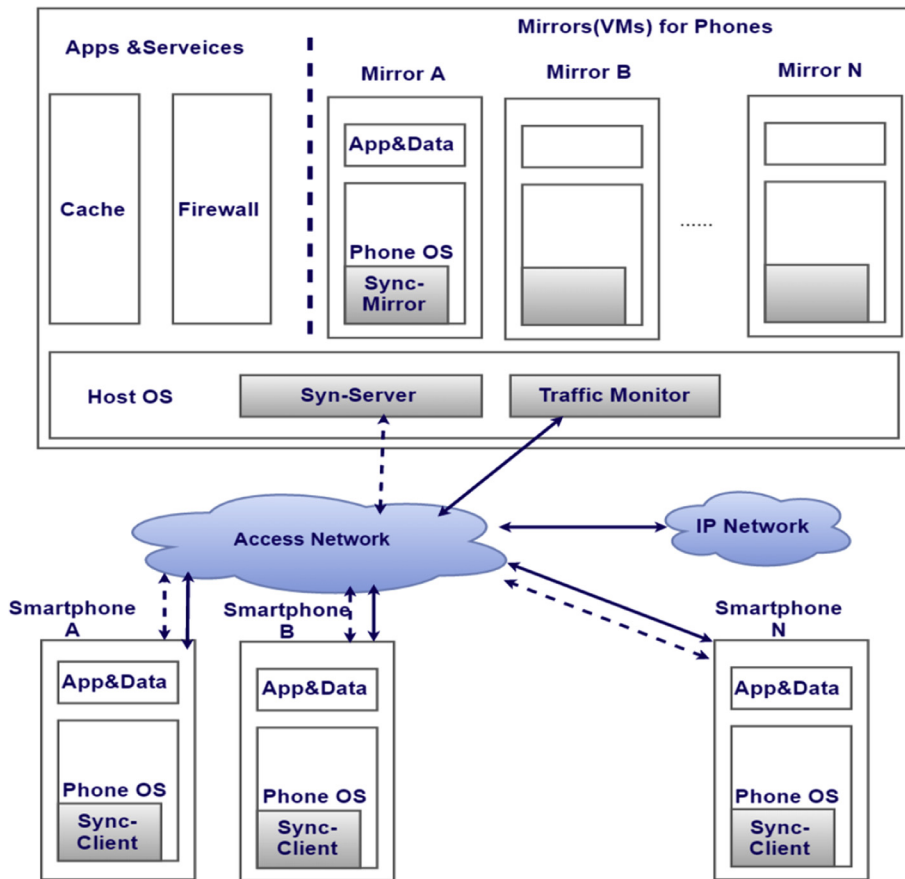


Figure 12 Mirror server architecture (adapted from [38]).

the application's delay tolerance) to become available, rather than sending data immediately.

The framework can offload the whole or part of an application to a cloud server. The prototype of Phone2Cloud is implemented for Android and Hadoop environment (to serve as a cloud). It consists of several components, including an offloading decision engine, a local execution manager, a bandwidth monitor, a resource monitor, an execution time predictor, a remote execution manager, and an offloading proxy that links the offloading decision engine to remote execution manager.

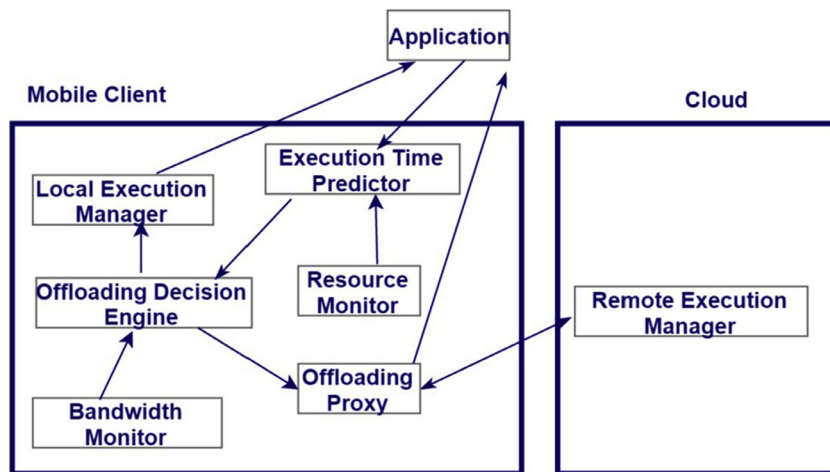
The decision engine is built in order to analyze the power consumption due to offloading. Before execution, two types of comparisons are made:

- (1) The average execution time of the application running on the SMD is compared with the user's delay-tolerance threshold.
- (2) The power consumption to run the application on the SMD is calculated and compared with power consumption required to run the same application on the cloud.

First, average execution time and user's delay-tolerance threshold are compared. If user's delay-tolerance threshold is smaller than average execution time then the application is offloaded to the cloud. Otherwise, the decision engine checks whether power consumption to run the application on the cloud is greater than power consumption to run the application on the SMD. If it is the case, the application is executed locally. Otherwise, the application is offloaded to the cloud.

An illustration of the architecture of Phone2Cloud is provided in Fig. 13. Phone2Cloud is composed of seven key components.

- Execution time predictor: It is one of the key components in Phone2Cloud. It is used to predict average execution time of an entire application on a mobile device.
- Bandwidth monitor and resource monitor: Bandwidth monitor is used to monitor current bandwidth usage of the network while resource monitor takes care of monitoring CPU workload and other resources. The two monitors serve the offloading decision engine and the execution time predictor separately.
- Offloading proxy: It sends necessary input data to the remote execution manager, receives the results returned by the remote execution manager, and sends back the results to the application.
- Offloading decision engine: It is the core component of Phone2Cloud. Offloading decision engine decides whether to offload the application's components from the mobile device to the cloud server. When it decides to run the application locally, it invokes local execution manager to execute the application. Otherwise, it invokes the offloading proxy to offload computation to the cloud.
- Local execution manager and remote execution manager: The local execution manager is designed to execute the application locally on the SMD. It calls the SMD's operating system, like Android OS, to execute the application. When the remote execution manager receives the required



input data from the offloading proxy, it runs the offloaded computation on the cloud, and returns results to the offloading proxy.

The authors examine how the energy consumption and execution time of applications will be affected. The evaluated applications are word count, path finder, and sort application. The framework saves energy and improves applications' performance and users' experience of smartphones.

For instance, face finder application costs more energy on smartphone than on a cloud server and the difference between the two costs gets bigger as input grows. The reason behind this is because data transmission costs less energy than running the application locally. Moreover, the energy consumption in smartphones grows faster than that on the cloud server. Thus, face finder application should be offloaded to cloud.

#### 4.7.1. A comparative review of some offloading frameworks

Having reviewed different existing computation offloading frameworks along with their main characteristics, Table 2 presents an overall view about these frameworks and classifies them based on the following attributes:

- Preparation: Any necessary preparations before offloading.
- Partitioning: Partitioning supported or not.
- Decision: Dynamic or static.
- Offloading Mechanism: Mechanism used to offload intensive computations.
- Granularity Level: Granularity Level (i.e. class, method, thread).
- Annotation: Automation of partitioning process (Automatic or manual).
- Contribution: Solved problems?

Annotation is one of the important attributes in partitioning step. It can be seen as a metadata added to the source code. The current partitioning algorithms used in the offloading frameworks can be categorized as (a) automatic and (b) manual.

In automatic annotation, the offloading framework implements automatic annotation by using the profiler to collect

the necessary information and annotate the relevant component in the application as an indication of availability of partitioning [6,7].

Manual annotation is performed by the programmers at the design phase. It requires examining the scope of the components of the application at design time. Programmers annotate the components of the application at different granularity such as classes and methods [8,32].

We can notice that some frameworks offload the entire application while other frameworks split the application into its components. Concerning the offloading mechanism, some frameworks encapsulate the offloaded components into a VM or create a VM with exactly the same hardware/software specifications. Other frameworks focus on the code mechanism to offload intensive components. For the annotation attribute, some frameworks use manual annotation while others use automatic one except Phone2Cloud framework which follows a semi-automatic way. Decision offloading is the main attribute of the different offloading frameworks. Some frameworks take the offloading decision at runtime based on a program profiling and program analysis while others take the decision during design or compile time using programmers' annotations and some estimations. A static offloading decision could not adapt to fluctuating network conditions efficiently and depends on programmers' decision. A dynamic offloading decision incurs overhead as it is continuously performed to obtain the latest information.

It can be seen from the presented frameworks that they use different approaches to offload intensive tasks to remote cloud servers. However, none of them use or adopt containers technology such as Linux Containers (LXC). LXC is attracting researchers these days as a lightweight alternative to full machine virtualization such as the common known hypervisors such as KVM or Xen. Recently, research suggests that applications running in containers can achieve approximately same speed in memory, processing and also network throughput as if they were running on a physical machine [41]. LXC is considered as an OS level virtualization where each container has its own environment called a namespace where specific processes are running and isolated from the rest of the system. The usage of containers instead of VM will be a good idea since it is lighter than VM.

**Table 2** A comparative review of some offloading framework.

Framework	Partitioning	Preparation	Decision	Offloading mechanism	Contribution	Granularity level	Automation	Year
VM Cloudlet [29]	The entire migrating image of the running application is offloaded to the designated remote server while the mobile device provides a user interface and serves as a thin client	It requires the cloning of the mobile device application processing environment to a remote host	Not available	VM: the mobile device transmits all the states of the application to the cloudlet, which applies it to the base VM to launch and execute the VM	Cloudlet-based resource-rich mobile computing	Entire App	Not available	2009
Phone2Cloud [36]	The application can be partitioned or entirely offloaded	Applications need to be manually modified in order to be executed on cloud servers	Static: the offloading decision is based on user's delay-tolerance threshold and static analysis	Code: the remote execution manager gets required input data, it executes offloading computation on the cloud server, and sends back results to the offloading proxy	Enhancement of the application's performance and improvement of energy efficiency of smartphones	Part/Entire App	Semi-automatic	2009
MAUI [8]	Annotate each individual method as local or remote	It creates two versions of a mobile application (for mobile device and cloud). It uses programming reflection to identify which methods are marked offloadable or not	Dynamic: decision is based upon the input of the MAUI profiler and MAUI solver	Code: MAUI does not support executing only portions of a method remotely	Energy-aware code offload	Method	Manual	2010
Mirror Server [38]	The framework does not require a partitioning so the entire application is offloaded	It creates a mirror for a smartphone	Not available	VM: During the copying process, no operation from user is authorized	Reduce the workload and increase the resources of smartphones in a virtual manner	Entire App	Not available	2010
Cuckoo [19]	Partitioning is made based on the existing activity model in Android. The graphical components remain on the mobile device while the services can be offloaded	Destination running a Java VM	Dynamic: method invocations to services are received and Cuckoo framework will then decide whether to offload it or not while checking the availability of the remote resources	Code: the framework receives method calls and evaluates whether to offload the method using heuristics information	Simplifying the development of smartphone applications while benefiting from computation offloading	Method	Manual	2010
CloneCloud [6]	The partitioning is made based on static program analysis and program profiling	A duplicate of mobile device's software stored on the cloud server	Dynamic: threads are migrated from the mobile device to the clone in the cloud	VM: offloaded components of an application are running inside a virtual machine	Elastic execution between mobile devices and clouds while adapting the application partitioning	Thread	Automatic	2011
Jade [28]	An application is partitioned at the class level in Jade. A class must implement one of two interfaces to be offloadable	It checks the application and device status by monitoring the communication costs, work load variation, and energy status	Dynamic: have updated information concerning the status of the application and the device	Code: an offloaded object can be executed on the remote server	An energy-aware computation offloading system	Class	Automatic	2015

753	<b>5. General issues and challenges in computation offloading for MCC</b>	
754		
755	The selected issues are presented from three perspectives: the resource-intensive structures of the existing frameworks, the security perspective, and the optimal offloading platform.	
756		
757		
758	<i>5.1. Platform diversity</i>	
759	One of the challenges in the current computation offloading frameworks is the diversity and heterogeneity of smartphone architectures and operating systems. This diversity is seen in the following example: MAUI [8] is an offloading framework which is applicable for the .Net framework whereas Mirror Server [38] is a framework which is compatible with the Android platform. A consistent access to cloud services is expected wherein SMDs are enabled to access cloud computing services regardless of the installed operating system or the used hardware. A standardized offloading framework for different smartphone platforms is still a challenging issue in the MCC field.	
760		
761		
762		
763		
764		
765		
766		
767		
768		
769		
770		
771	<i>5.2. Security and privacy in mobile cloud applications</i>	
772	Security of data transmission is an important concern in cloud based application processing. Security and privacy are two crucial concepts that need to be maintained during the offloading process. These concepts can be addressed from different angles: (1) Mobile device, (2) cloud data centers, and (3) during data transmission over the network. Besides all the technologies, there is a great increase in the variety of sophisticated attacks on mobile phones which are the main targets for attackers. Regarding the security in the cloud data centers, threats are basically related to the transmission of data between the different nodes over the network. Thus, high levels of security are expected by both the mobile clients and the cloud providers. In the current frameworks [10,12], binary transfer of the application code at runtime is continually subjected to security threats. Despite the available solutions, strong measures and a secure environment are required for the three entities of MCC model.	
773		
774		
775		
776		
777		
778		
779		
780		
781		
782		
783		
784		
785		
786		
787		
788		
789	In [39], the authors focus on optimizing tasks and computations, and they explore secure offloading of applicable linear programming (LP) computations. In this paper, authors build their work based on the decomposition of the LP computation offloading into public LP solvers running on the cloud and private LP parameters owned by the customer. To achieve an efficient and validate results, the authors focus on the fundamental duality theorem of LP computation and come up with the essential conditions that must satisfied by correct results. Bugiel et al. present in [40] an architecture for secure outsourcing of data and arbitrary computations to an untrusted commodity cloud. The architecture proposed in their approach consists of two clouds (twins): a trusted cloud and a commodity cloud.	
790		
791		
792		
793		
794		
795		
796		
797		
798		
799		
800		
801		
802		
803	The computations are divided in such a way that the trusted cloud is mainly used for critical operations, whereas requests to the offloaded data are processed in parallel by the fast commodity cloud on encrypted data.	
804		
805		
806		
807	However, the idea of dividing operations and handling them by different clouds lead to different difficulties.	
808		
	For instance, the deployment and maintenance of this architecture of cloud will need clear modifications in the main infrastructure.	
	The security threat is advancing in a quick manner more than we can keep up with it. Security techniques need to enhance and progress constantly to meet new changes and new offered services. Thus, it is no longer possible to define a security system that would solve all the security threats at once.	
	<i>5.3. Fault-tolerance and continuous connectivity</i>	
	In MCC, mobility is one of the most important attributes of SMDs. This is because freedom of movement and autonomy of communication during the consumption of mobile cloud services, are crucial criteria for users' satisfaction. However, there are some constraints that prevent the achievement of seamless connectivity and uninterrupted access to cloud services while on the move. As mobile users move, data exchange rates and network bandwidth may vary. Moreover, users may lose their connection while sending or receiving data; therefore, offloading approaches should be provided with suitable fault-tolerant strategies in order to resend the lost components, minimize the response time, and reduce the energy consumption of mobile devices. It should be noted that the guarantee of a successful execution of offloaded applications is very crucial for mobile users.	
	<i>5.4. Automatic mechanism</i>	
	The available computation offloading frameworks still need to be automated. This will help the offloading process to be performed in a seamless fashion while discovering the surrounded environment [5,9,14]. The achievement of such automation is not an easy task as it needs the implementation of a protocol dedicated to finding and discovering services depending on the current context and its constraints.	
	<i>5.5. Offloading economy cost</i>	
	Using cloud infrastructure resources imposes financial charges on the end-users, who are required to pay according to the Service Level Agreement (SLA) agreed on with the cloud vendor serving them. Generally, the operations of content offloading and data transfer between cloud providers incur additional costs on end-users. Therefore, economic factors should be taken into consideration while making the offloading decisions.	
	<i>5.6. Partition offloading and external data input</i>	
	At runtime, it is challenging to decide which application components need to be offloaded and to find the suitable server for that. Algorithms answering this problem need resource-intensive effort, which can affect the execution time of the off-loaded partitions of the application [13].	
	Although existing application partitioning algorithms allow an adaptive execution of the application between the mobile devices and the cloud servers, they still do not provide any solution on how to utilize and benefit from the elastic resources in the clouds. This is specifically needed in order to make the	

**Table 3** Some challenges and open issues in offloading frameworks for MCC.

	Open Issues in offloading frameworks	Challenges to available offloading frameworks
Access a distributed platform transparently	✓	✓
Continuous connectivity to cloud servers	–	✓
Diversity of operating systems in mobile devices along with the variety of their architectures	✓	✓
Provide an effective execution of a process remotely and returns result to mobile device	✓	✓

applications scalable when a large number of mobile users need to be served and when the application requires input data that are stored in other remote servers.

Table 3 recapitulates the main challenges to current offloading frameworks and open research issues in MCC. The challenges indicate the issues in the computation offloading frameworks in MCC that still require more elaboration and thorough study, while the open issues specify unresolved problems in current offloading frameworks.

## 6. Conclusion

This paper discusses three main concepts: (1) cloud computing, (2) mobile cloud computing, and (3) computation offloading. More specifically, it presents existing frameworks for computation offloading along with the various techniques used to enhance the capabilities of smartphone devices based on the available cloud resources. The paper investigates the different issues in current offloading frameworks and highlights challenges that still obstruct these frameworks in MCC. Moreover, the paper shows the different approaches that are used by the frameworks to achieve offloading. Some of these approaches use static offloading while others employ dynamic offloading. Even though there exist a variety of approaches, all of them target the same objective which is the improvement of the smartphone device capabilities by saving energy, reducing response time, or minimizing the execution cost.

We notice that current offloading frameworks are still facing some challenges and difficulties. For instance, lack of standard architectures. This shortage leads to more complications while developing and managing a proposed framework. Finally, it is important to come up with a lightweight paradigm or model that will help to overcome the difficulties and minimizing efforts while developing, deploying, and managing an offloading framework.

We believe that exploring other alternatives, such as introducing a middleware based architecture using an optimizing offloading algorithm, could help better the available frameworks and provide more efficient and more flexible solutions to the MCC users.

## Acknowledgments

This work has been funded by the Schlumberger Foundation Faculty for the Future and Technical University of Munich.

## References

- [1] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al, A view of cloud computing, *Commun. ACM* 53 (4) (2010) 50–58.
- [2] R. Barga, D. Gannon, D. Reed, The client and the cloud: democratizing research computing, *IEEE Internet Comput.* 15 (1) (2011) 72.
- [3] B. Butler, Gartner: Cloud Putting Crimp in Traditional Software, *Hardware Sales, Networkworld*, 2012.
- [4] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, I. Brandic, Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility, *Future Generat. Comput. Syst.* 25 (6) (2009) 599–616.
- [5] W.-C. Chuang, B. Sang, S. Yoo, R. Gu, M. Kulkarni, C. Killian, Eventwave: programming model and runtime support for tightly-coupled elastic cloud applications, in: *Proceedings of the 4th annual Symposium on Cloud Computing, ACM*, 2013, p. 21.
- [6] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, A. Patti, CloneCloud: elastic execution between mobile device and cloud, in: *Proceedings of the Sixth Conference on Computer Systems, ACM*, 2011, pp. 301–314.
- [7] B.-G. Chun, P. Maniatis, Augmented smartphone applications through clone cloud execution, *HotOS 9* (2009) 8–11.
- [8] E. Cuervo, A. Balasubramanian, D.-K. Cho, A. Wolman, S. Saroiu, R. Chandra, P. Bahl, Maui: making smartphones last longer with code offload, in: *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services, ACM*, 2010, pp. 49–62.
- [9] Y. Cui, X. Ma, H. Wang, I. Stojmenovic, J. Liu, A survey of energy efficient wireless transmission and modeling in mobile cloud computing, *Mobile Networks and Applications* 18 (1) (2013) 148–155.
- [10] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, *Commun. ACM* 51 (1) (2008) 107–113.
- [11] H.T. Dinh, C. Lee, D. Niyato, P. Wang, A survey of mobile cloud computing: architecture, applications, and approaches, *Wireless Commun. Mobile Comput.* 13 (18) (2013) 1587–1611.
- [12] A. Dou, V. Kalogeraki, D. Gunopulos, T. Mielikainen, V.H. Tuulos, Misco: a mapreduce framework for mobile systems, in: *Proceedings of the 3rd International Conference on Pervasive Technologies Related to Assistive Environments, ACM*, 2010, p. 32.
- [13] I. Giurgiu, O. Riva, G. Alonso, Dynamic software deployment from clouds to mobile devices, in: *Middleware 2012, ACM*, 2012, pp. 394–414.
- [14] M.S. Gordon, D.A. Jamshidi, S. Mahlke, Z.M. Mao, X. Chen, Comet: Code offload by migrating execution transparently, in: *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, 2012, pp. 93–106.
- [15] K. Ha, G. Lewis, S. Simanta, M. Satyanarayanan, Cloud Offload in Hostile Environments, Technical Report, DTIC Document, 2011.
- [16] S. Hakak, S.A. Latif, G. Amin, A review on mobile cloud computing and issues in it, *Int. J. Comput. Appl.* 75 (11) (2013).
- [17] J. Hauswald, T. Manville, Q. Zheng, R. Dreslinski, C. Chakrabarti, T. Mudge, A hybrid approach to offloading mobile image classification, in: *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014 IEEE, IEEE, 2014, pp. 8375–8379.



- 964 [18] A. Huth, J. Cebula, *The Basics of Cloud Computing*, United  
965 States Computer, 2011. 1005
- 966 [19] R. Kemp, N. Palmer, T. Kielmann, H. Bal, Cuckoo: a  
967 computation offloading framework for smartphones, in:  
968 *Mobile Computing, Applications, and Services*, Springer, 2010,  
969 pp. 59–79. 1006
- 970 [20] ZEKRIFA, Djabeur Mohamed Seifeddine et LEE, Seong,  
971 *Information System Security and Performance Modeling and Simulation*  
972 *for Future Mobile Networks*.  
973 *Mathematical and Computer Modelling*, 2012. 1007
- 974 [21] S. Kundu, J. Mukherjee, A.K. Majumdar, B. Majumdar, S.S.  
975 G.A. Lewis, S. Echeverría, S. Simanta, B. Bradshaw, J. Root,  
976 Cloudlet-based cyber-foraging for mobile systems in resource  
977 constrained edge environments, in: *Companion Proceedings of*  
978 *the 36th International Conference on Software Engineering*,  
979 ACM, 2014, pp. 412–415. 1008
- 980 [22] S. Mathew, *Overview of Amazon Web Services*, Amazon  
981 Whitepapers, 2014. 1009
- 982 [23] J. Matthews, M. Chang, Z. Feng, R. Srinivas, M. Gerla,  
983 Powersense: power aware dengue diagnosis on mobile phones,  
984 in: *Proceedings of the First ACM Workshop on Mobile*  
985 *Systems, Applications, and Services for Healthcare*, ACM,  
986 2011, p. 6. 1010
- 987 [24] P. Mell, T. Grance, *The Nist Definition of Cloud Computing*,  
988 2011. 1011
- 989 [25] A. Mtibaa, A. Fahim, K.A. Harras, M.H. Ammar, Towards  
990 resource sharing in mobile device clouds: power balancing  
991 across mobile devices, *ACM SIGCOMM Comput. Commun.*  
992 *Rev.* 43 (4) (2013) 51–56. 1012
- 993 [26] Y. Nimmagadda, K. Kumar, Y.-H. Lu, C.G. Lee, Real-time  
994 moving object recognition and tracking using computation  
995 offloading, in: *International Conference on Intelligent Robots*  
996 *and Systems (IROS)*, 2010 IEEE/RSJ, IEEE, 2010, pp. 2449–  
997 2455. 1013
- 998 [27] H. Qian, D. Andresen, Jade: reducing energy consumption of  
999 android app, *Int. J. Network. Distrib. Comput (IJNDC)* 3 (3)  
1000 (2015) 150–158 (Atlantis Press). 1014
- 1001 [28] M. Satyanarayanan, P. Bahl, R. Caceres, N. Davies, The case  
1002 for vm-based cloudlets in mobile computing, *IEEE Pervasive*  
1003 *Comput.* 8 (4) (2009) 14–23. 1015
- 1004 [29] M. Shiraz, E. Ahmed, A. Gani, Q. Han, Investigation on  
1005 runtime partitioning of elastic mobile applications for mobile  
1006 cloud computing, *J. Supercomput.* 67 (1) (2014) 84–103. 1007
- 1008 [30] M. Shiraz, A. Gani, R.H. Khokhar, R. Buyya, A review on  
1009 distributed application processing frameworks in smart mobile  
1010 devices for mobile cloud computing, *IEEE Commun. Surv.*  
1011 *Tutorials* 15 (3) (2013) 1294–1313. 1012
- 1012 [31] M. Smit, M. Shtern, B. Simmons, M. Litoiu, Partitioning  
1013 applications for hybrid and federated clouds, in: *Proceedings of*  
1014 *the 2012 Conference of the Center for Advanced Studies on*  
1015 *Collaborative Research*, IBM Corp., 2012, pp. 27–41. 1013
- 1016 [32] M. Tulloch, *Introducing Windows Azure for IT Professionals*,  
1017 Microsoft Press, 2013. 1014
- 1018 [33] C. Wang, Z. Li, Parametric analysis for adaptive computation  
1019 offloading, *ACM SIGPLAN Notices*, vol. 39, ACM, 2004, pp.  
1020 119–130. 1015
- 1021 [34] S. Wang, S. Dey, Rendering adaptation to address  
1022 communication and computation constraints in cloud mobile  
1023 gaming, in: *Global Telecommunications Conference*  
1024 *(GLOBECOM2010)*, 2010 IEEE, IEEE, 2010, pp. 1–6. 1016
- 1025 [35] F. Xia, F. Ding, J. Li, X. Kong, L.T. Yang, J. Ma, Phone2cloud:  
1026 exploiting computation offloading for energy saving on  
1027 smartphones in mobile cloud computing, *Inform. Syst. Front.*  
1028 16 (1) (2014) 95–111. 1017
- 1029 [36] K. Yang, S. Ou, H.-H. Chen, On effective offloading services for  
1030 resource-constrained mobile devices running heavier mobile  
1031 internet applications, *Commun. Mag. IEEE* 46 (1) (2008) 56–63. 1018
- 1032 [37] B. Zhao, Z. Xu, C. Chi, S. Zhu, G. Cao, Mirroring smartphones  
1033 for good: a feasibility study, in: *Mobile and Ubiquitous Systems:*  
1034 *Computing, Networking, and Services*, Springer, 2010, pp. 26–  
1035 38. 1019
- 1036 [38] C. Wang, K. Ren, J. Wang, Secure and practical outsourcing of  
1037 linear programming in cloud computing, in: *INFOCOM*, 2011  
1038 *Proceedings IEEE*, IEEE, 2011, pp. 820–828. 1020
- 1039 [39] S. Bugiel, S. Nurnberger, A. Sadeghi, T. Schneider, Twin clouds:  
1040 an architecture for secure cloud computing, in: *Workshop on*  
1041 *Cryptography and Security in Clouds (WCSC 2011)*, 2011. 1021
- 1042 [40] D. Bernstein, Containers and cloud: from LXC to Docker to  
1043 Kubernetes, in: *IEEE Cloud Computing 1.3*, IEEE, 2014, pp.  
1044 81–84. 1022
- 1045 [41] A. Balasubramanian, R. Mahajan, A. Venkataramani,  
1046 Augmenting mobile 3G using WiFi, in: *Proceedings ACM*  
1047 *MobiSys'10*, Chicago, 2010, pp. 209–222. 1023
- 1048 1024