Mobile Netw Appl (2015) 20:297–307
DOI 10.1007/s11036-015-0617-0

Establishing A Personal On-Demand Execution Environment for Mobile Cloud Applications

Huijun Wu[1] · Dijiang Huang[1] · Yan Zhu[2]

Published online: 21 May 2015

Abstract A distributed mobile cloud service model called "POEM" is presented to manage the mobile cloud resource and compose mobile cloud applications. POEM provides the following salient features: (a) it considers resource management not only between mobile devices and clouds, but also among mobile devices; (b) it utilizes the entire mobile cloud system as the mobile application running platform, and as a result, the mobile cloud application development is significantly simplified and enriched; and (c) it addresses the interoperability issues among mobile devices and cloud resource providers to allow mobile cloud applications running cross various cloud virtual machines and mobile devices. The proposed POEM solution is demonstrated by using OSGi and XMPP techniques. Our performance evaluations demonstrate that POEM provides a true elastic application running environment for mobile cloud computing.

✉ Huijun Wu
Huijun.Wu@asu.edu

Dijiang Huang
Dijiang.Huang@asu.edu

Yan Zhu
zhuyan@ustb.edu.cn

[1] School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, 699 S Mill Ave, Tempe, AZ 85281, USA

[2] School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing 100083, China

Keywords Mobile application · Cloud computing · Offloading · Service oriented architecture · OSGi · XMPP

1 Introduction

In mobile clouds, mobile devices and cloud resources compose a distributed mobile application running environment, where a mobile application may consume resources from both local and remote resource providers who provide computing, networking, sensing, and storage resource provisioning. Mobile devices can serve as either service consumers or service providers in the mobile cloud, in which the cloud boundaries are extended into the mobile domain [1]. Mobile applications may require a mobile device to interact with other mobile devices and cloud resource providers to achieve desired computing, storing, collaboration, or sensing features.

An ideal mobile cloud application running system should enable mobile devices to easily discover and compose cloud resources for its applications. From mobile resource providers' perspectives, they may not even know what applications are using their resources and who may call their provisioned functions beforehand. In this way, the mobile application design should not be application-oriented; instead, it should be functionality-oriented (or service-oriented). For example, the video function of a mobile device should provide general interfaces that can be called by multiple local or remote functions in the runtime. To achieve this feature, we can consider these Provisioning Functions (PFs) as the fundamental application components in the mobile cloud, which can be composed by mobile cloud service requesters in the runtime. As a result, mobile cloud can significantly reduce the mobile application development overhead and greatly improve the agility

and flexibility to build a personalized mobile cloud computing system that can be customized for each mobile user.

A simple vehicular video sensing example is used to illustrate the above described mobile cloud features. Alice is driving on the road and her smartphone, which is mounted on the front dashboard for navigation, has the basic video capture PF. Bob is driving next to Alice and is running an image processing PF in his phone and wants to utilize more video clips from the neighboring vehicles in order to reconstruct the road situations around his vicinity. Then Bob can consume Alice's video PF to reconstruct the view of the entire road segment captured by their video cameras. Moreover, Bob wants to share his captured video clips to his friend Carol who is managing a traffic monitoring website that posts videos from smartphone users for the public to access the realtime road traffic information. As the result, Bob can share his augmented traffic view to Alice, i.e., through Carol's website. In this mobile application scenario, all participants have them basic PFs: (a) Alice: video capture, (b) Bob: video augment, and (c) Carol: video display. Note that a PF can be called by multiple other PFs for different application purposes, and they altogether can build several mobile cloud applications.

There are several challenges invoked by the above described application scenario. The first challenge is that knowing the status of mobile devices, e.g., online/offline and runtime information (such as battery, computing power, connectivity, etc.), is difficult due to the mobility of mobile users. The second challenge is that knowing the available PFs on each mobile device is not a trivial task. Currently, there is no such a common framework allowing mobile devices for exchanging the available PFs information and running such a system in a distributed environment. The third challenge is to compose PFs crossing various hardware and software platforms, which demands a universal programming and application running environment with little compatibility issues.

To address these challenges, we present a new mobile cloud application running system, which is called POEM (Personal On-demand execution Environment for Mobilecloud computing), as shown in Fig. 1. POEM treats each mobile device as a PF provider. In addition, POEM is designed based on the mobile cloud framework, where a dedicated Virtual Machine (VM) is assigned to each mobile device providing computing and storage support. Moreover, PFs can be offloaded/migrated from a mobile device to its assigned VM. Thus, the VM can not only run mobile devices' PFs (i.e., as shadows), but also can run extended PFs that mobile devices may not have the capacity to execute. Thus, we also call the VM in the POEM framework as ESSI (Extended Semi-Shadow Image). Collectively, the PFs provided by a mobile device X and its corresponding $ESSI_X$ is denoted as $\{PF\}_X$. POEM regards both mobile

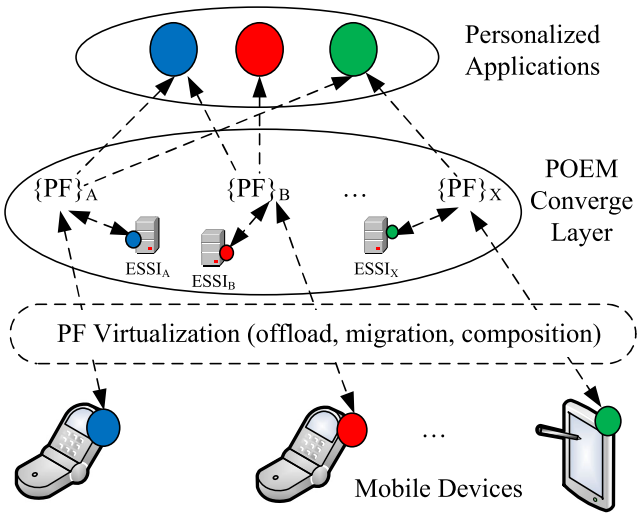


Fig. 1 Overview of POEM system

devices and their dedicated ESSIs as PF providers. As a result, the mobile user's applications can be composed by PFs from local PFs (may be offloaded/migrated to its dedicated ESSI) and/or remote PFs (may run on remote mobile devices or their dedicated ESSIs).

To demonstrate the proposed POEM solutions, we implemented a pilot POEM system based on OSGi [2] and XMPP [3] techniques. In summary, the contributions of the presented research is highlighted as follows:

- *Social mobile cloud computing:* POEM solution enables mobile cloud application to utilize social network power, i.e., in addition to the discovered PFs through the mobile cloud system, mobile user can establish mobile cloud applications through their trusted social connections. In this way, POEM applications not only can use the resource in cloud by offloading resource intensive components but also can use services provided from their social connections.
- *Versatile and personalized application offloading, migration, and composition:* POEM maintains available mobile cloud resource and allows users choosing a mobile cloud application by using different approaches (offloading, migration, and composition) based on the available system resources and their personalized application requirements.

The paper is organized as follows. Section 2 introduces related work and compares POEM to them. Section 3 describes systems and models. Section 4 discusses POEM design and implementation, respectively. Section 5 presents evaluation results. Section 6 discusses the application of POEM platform in vehicle cloud scenario. Finally, Section 7 concludes the paper.

2 Related work

Most of the research focuses either on mobile task partition and composition or on offloading techniques. μCloud [4] describes a framework for mobile cloud application composition from heterogeneous software components. μCloud presents mobile cloud application as a directed graph whose nodes are categorized as mobile, cloud, and hybrid. μCloud is a static mobile cloud application model, which requires a lot of work for programmers to partition application and decides which partition runs on which part of the cloud. eXCloud [5] focuses on offloading and migration: it migrates Java Virtual Machine (JVM) runtime to cloud. However it migrates only the top portion of runtime stack, rather than the whole virtual machine, to cloud using Stack On Demand (SOD) [6] migration technique. eXCloud triggers offloading processes according to local resource availability by capturing resource exception. So performance and energy gain are not optimized. CloneCloud [7] maintains a clone of mobile device in the cloud. The mobile device and its clone in cloud is synchronized. The process running on phone can be set to sleep state and transfer the execution state to its clone. When finishing execution on clone, the process states are transferred back to mobile device and integrated into the original process. CloneCloud can deal with dynamic offloading, however it requires synchronization between mobile device and cloud, which is not always satisfied in mobile cloud application scenario due to unstable mobile connection to cloud.

Some frameworks generate two versions of application for mobile device and cloud separately. MAUI [8] and ThinkAir [9] use similar offloading technique and both have their own decision making algorithms. They require the programmer to mark the offloadable method and they generate two versions of an application: one is for mobile execution and the other is for cloud. The offloading decision is made according to execution history and energy consumption. ThinkAir in addition considers high availability in unstable connection scenario. Cuckoo [10] focuses on offloading technique. Cuckoo generates local and remote version of an Android Service component, which is similar to MAUI and ThinkAir. Cuckoo requires programmer support to build the final application and it's offloading decision making algorithm is static.

Some frameworks apply loose couple between mobile devices and clouds. Zhang et al. [11] proposed a web based mobile cloud application model. It defines *weblet* as independent compute unit that provides web service. *Weblet* can be migrated based on the decision of its cost model. Zhang et al. model restricts application structure to User Interface (UI), *weblet* and manifest, which force application components to communicate through web service. Satyanarayanan et al. [12] proposed *cloudlet* solution for offloading. The *cloudlet* is virtual machine that is near the mobile phone and that is usually connected to WiFi access point. The offloading is achieved through virtual machine migration, which increases migration load and leads to high latency. Besides the above work, [13–17] discussed the mobile cloud system in specific areas.

3 Systems and models

This section presents application and execution model of the presented POEM system.

3.1 Application model

POEM is implemented based on OSGi framework [2] that is a general purpose, secure, and managed Java framework that supports the deployment of extensible and downloadable applications [2]. Due to the popularity of Java, OSGi framework is compatible with major operating systems for both desktop and mobile device systems. The framework is stacked in layers: from bottom-up, module layer, life cycle layer, and service layer. The framework defines a unit of modularization, called a bundle, i.e., "PF" in the POEM. In the later descriptions, we do not differentiate the terms bundle and PF. In POEM, a PF is comprised of Java classes and other resources, and is deployed as a Java ARchive (JAR) file. PF sits on the top of stacked layers and interacts with them through PF context. Module layer and life cycle layer handle PF installation and activation. PF can be installed/uninstalled and started/stopped. Service layer has a service registry and handles service publication and discovery. A service is a normal Java object that is registered under one or more Java interfaces with the service registry. PFs can register services, search for them, or receive notifications when services' states change. When PF is installed, the framework must cache the PF JAR file. A SERVICE_RANKING property may be specified when a service is being registered. The service with the highest ranking is returned when the framework receives service query. Before the service is consumed, it may become a stale reference. Service tracker is usually used for service consumer to prevent stale reference by obtaining reference when consumption happens. Besides local service activities, a distribution provider can export service to another framework by creating end point or import service from another framework by creating proxy for service composition, and then registering the proxy as an imported service.

POEM models a mobile cloud application as a set of PFs. The PF may provide class definitions and host services that implements PFs. POEM does not differentiate PFs on mobile devices and PFs on their ESSIs as the PFs may be migrated from mobile side to cloud side or vice versa

without any modification. The uniform PF format make PFs reusable and reduces develops' workload by avoiding developing separate PFs for specific platform. The application may use services provided by local or remote PFs. The application can migrate PFs between mobile devices and ESSIs without disrupting the other active PFs.

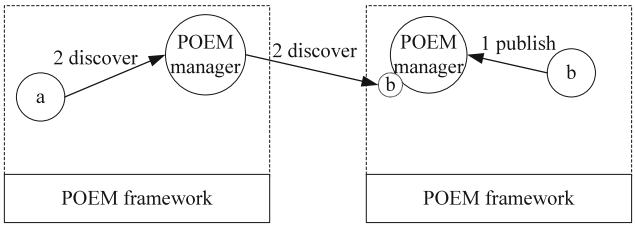
POEM achieves social feature through an implemented XMPP [3] system within the MobiCloud system [18]. The availability information of the system resources and mobile devices is maintained through a decentralized client-server architecture, where every mobile cloud entity needs an address called a JabberID (JID). JID is presented in the form of user@domain/resource. Domain represents the XMPP service provider, user represents virtual identity in the domain, and resource identifies connection to an XMPP server. Three basic status services are achieved through XMPP: message, presence, and info/query(or iq). POEM's service discovery protocol provides two discovery methods: one enables discovering information about an entity; and the other enables to discover the items associated with an entity.

In POEM, each entity, i.e., a mobile device or an ESSI, runs an OSGi framework, which is identified uniquely by its JID. One POEM entity discovers services hosted by his/her friends through XMPP service discovery protocol and XMPP publish-subscribe protocol. Mobile applications offload PFs to ESSIs through XMPP file transfer protocol, and the data exchange with remote application in POEM is through XMPP *iq* communication.

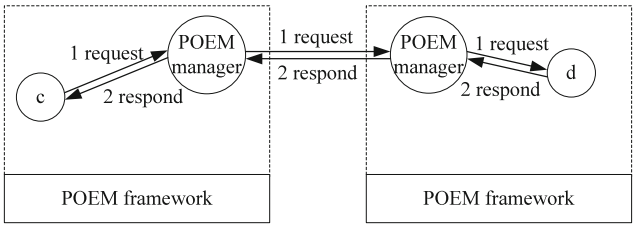
3.2 Execution model

According to previous application scenario, there are three fundamental execution patterns in POEM, as shown in Fig. 2. The first pattern describes how one PF discovers remote available PFs, which is shown in Fig. 2a. PF *b* hosts a service and it publishes the service through local POEM for remote PF to discover. Then PF *a* can discover the published service on remote side with local POEM PF's help. One prerequisite for *a* to discover and use service of *b* is that they are mutual friends, in other words they in each other's contact list. PF *a* does not know that PF *b* is running on remote side because POEM pretends that *b* is running locally. Thus, a programmer does not need special treatments in coding when developing PF *a*.

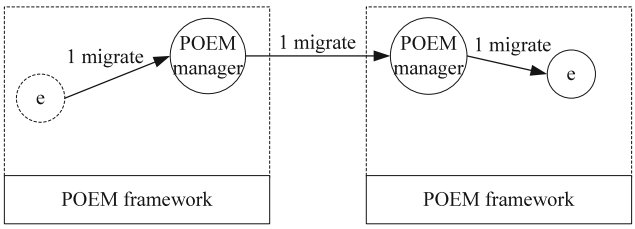
The second pattern presents how an application recruit a service provided by a remote PF, which is shown in Fig. 2b. The PF *c* sends method invocation parameters, which are transferred by the POEM on local side and then on remote side, to the destination PF *d*. Then, the service result returns along the reverse route from *d* back to *c*. PF *c* also regards it is calling a local target *d* due to POEM transparent transfer, and *d* also thinks local *c* is calling it.



(a) PF publishing and discovery pattern.



(b) PF composition pattern.



(c) PF offloading pattern.

Fig. 2 Execution patterns in POEM

The third pattern presents how one PF migrates to a remote entity. A POEM PF initializes the migration process. There are two types of migrations: pull and push. In pull migration, the POEM PF on the right side sends request to left side POEM PF, and then the later fetches and transfers the target PF *e* to the right side. In push migration, the POEM PF on the left side transfers PF *e* to the remote side. The source keeps the PF *e* active during transfer to provide the failsafe when the transferring is not successful.

4 POEM design and implementation

Figure 3 illustrates the overall design of POEM system. The POEM Manager monitors local services, tracks service state change, maintains local PF repository and responds to remote service queries. Its networking component also maintains XMPP connections to XMPP peers that provides the communication and signaling infrastructure among mobile devices and their ESSIs. The POEM composition component creates local proxy for remote service provider that responds to service request by transferring the request to the remote PF, and then getting the result to the local PFs. Based on a systematic decision model, POEM initiates

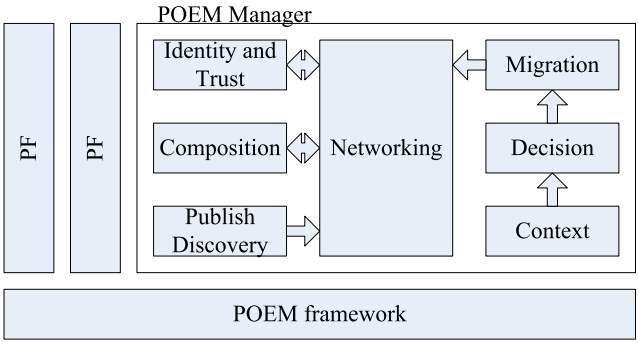


Fig. 3 POEM design and components

the migration operations for PF offloading. In the following sections, we describe each component within the POEM framework.

4.1 Distributed POEM service platform

POEM's networking and signaling system is deployed based on XMPP approaches. The communication between POEM entities (i.e., mobile devices and ESSIs) is full duplex compared to half duplex HTTP approach deployed by many web-based service frameworks. In a distributed execution environment, any entity can be both a client and a server at the same time, which is different from web-based service models where clients and servers are explicitly defined. Moreover, POEM inherits the XMPP trust and identity management framework, where every POEM entity is authenticated when joining the system and data transferred are also protected through cryptographic approaches. As a result, the PF offloading and PF compositions can utilize the XMPP trust management framework with fine-grained access control capabilities. Furthermore, POEM entities need to provide their presence information to indicate its availability information in real-time, which is as well used to indicate their service status. Finally, POEM must provide the support for secure file transfer, service discovery, and service composition, in which the XMPP provides the fundamental support to realize these features.

4.1.1 POEM service discovery and publishing

POEM service discovery is designed based on XMPP service discovery protocol and XMPP publish-subscribe extension. A PF may reside on a mobile device or its corresponding ESSI. The ESSI takes the responsibility to represent the mobile user for any PF related operations and the mobile device POEM Manager can frequently update its available PFs information to the ESSI. In this way, the main POEM service discovery, migration, and composition operations will not be flooded to end mobile devices. The ESSI POEM Manager also maintains the mobile device availability information and provides its reachability information to its trusted POEM peers. When the ESSI POEM Manager receives the service discovery message, it replies with its available PFs with the available remote service interfaces.

POEM Manager also monitors local service changes and notifies its friends. This is done through a publishing procedure. POEM Manager first registers a publish node (i.e., a virtual node in the XMPP server) under its JID. Thus, when local service status changes, POEM Manager can post the notice on its publish node and its friends get notified and update their PFs availability database.

4.1.2 POEM service composition

When POEM discovers service provided by remote POEM entities, it tries to create a proxy for that service so that remote PF can be used locally. POEM uses Java dynamic proxy technique to create proxy. Dynamic proxy requires that the target interface's Class instance must exist. To have remote service interface's Class instance in local OSGi framework instance, POEM fetches PF JAR file corresponding to the target service from remote POEM framework. POEM Manager installs the PF, and then the target Class instance is available and proxy generation is done.

POEM uses JavaScript Object Notation (JSON) over XMPP for service composition because JSON is lightweight and has abundant expression ability. The service proxy generated by POEM Manager captures local service requests that are then converted into JSON requests. Then the JSON request is sent to XMPP channel to the destination. The destination POEM Manager receives the JSON request and translates it to method invocation on service provider's object. It then returns the result in form of JSON back to the source POEM Manager. Then the JSON response is decoded and returned to calling object.

4.2 PF offloading

When application decides to offload a service provider object and migrate it to cloud, POEM Manager chooses to send the object's byte code to cloud and start the object from byte code. How to choose POEM PFs to be migration is based on several conditions described as follows: First, thread migration solution is not adopted because some objects that exist in the same thread have to run on mobile device, such as user interfaces and sensors. Second, an application usually wants to migrate only the compute intensive operations rather than the whole thread. Third, object state is not maintained because the insight private details of

the object to be migrated cannot be fetched due to Java security management. Our recent practice suggests that service implementation should be stateless, so that the object states will not bother POEM like Representational State Transfer (REST) does [19].

4.2.1 Migration

The service provider object offloading process follows a three-step approach: First, the target PF JAR file is transferred to ESSI and started. Then, a proxy object is created to intercept and capture service request to remote target service. Finally, the PF containing target service provider object is stopped.

The migration happens according to the migration decision module command. POEM constructs the migration decision module as plug-in framework. The previous work [20–22] on decisions can be applied. User can develop his own migration decision strategy plug-ins and install the strategy bundle into POEM, which not only provides the flexibility for user customized migration strategy but also scales the POEM intelligence.

4.2.2 PF isolation

The migrated PFs are running in the surrogate POEM framework for providing service for its origination. These PFs may interact with the POEM framework and interrupt the PFs that belong to surrogate host. The PF isolation is required to protect the surrogate POEM framework and cease the potential attack from the migrated PF.

The POEM manager initializes a separate PF container for each friend who wants to offload his PF. The PF container is duplication of the surrogate host POEM framework. The only difference is that this nested PF container is empty and dedicate for the corresponding friend. The friend identity is stored and managed by identity manager. The surrogate host defines the accepted PF policies that are enforced by policy manager.

4.2.3 Connection failsafe

The connection between mobile device and cloud is usually not stable as mobile device moves. When the connection is lost, POEM Manager restarts the PF that has been stopped in offloading process. The recovery process has the following two steps: First, the target PF is started. Then, the proxy service is unregistered and the proxy object is destroyed. The first step prepares for receiving service request. The second step destroys proxy, which makes the target service provider object be the first in the ranking order to receive service request.

4.3 POEM manager implementation

POEM Manager consists of several objects as shown in Fig. 4. They are categorized as three sets - XMPP connection and related listeners, PF context and related listeners, and proxy and migration management. The three object sets represent three POEM functional sets: XMPP connection set represents remote POEM framework; PF context set represents local POEM framework; and proxy and migration management represent core POEM logic and operation that connect the other two parts.

4.4 Seamless offloading

POEM Manager registers a service with an Java interface that contains a method to do service migration. Service migration involves two framework instances that are source framework and destination framework. The offloading process can be illustrated using the following application scenario. The source is device 1 and the destination is an ESSI. The migration method is called on device 1. Service name and destination XMPP identity are passed to the migration method. The migration process consists of five steps as follow. First, a migration notice is sent by device 1 to the ESSI. Along with the migration notice, the PF JAR file that owns the indicated service is transferred from device 1 to the ESSI. Second, POEM Manager in the ESSI starts the PF. When PF is running, services including the indicated service

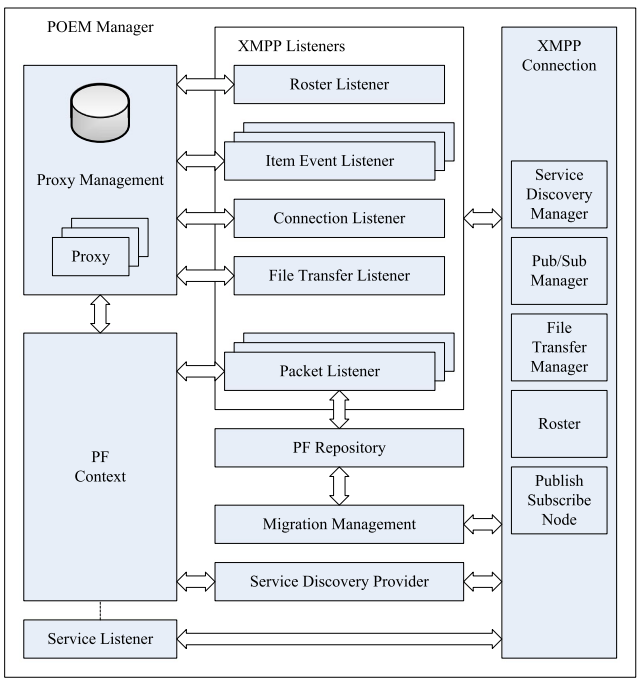


Fig. 4 POEM manager details

are registered. Third, POEM Manager in the ESSI is notified with service changes in last step. it unregister existing proxy under the same service name. Then it publishes the new services to the ESSI's publish node. At this point, both sides have the running PF that provides services to local PFs. Fourth, POEM Manager on device 1 is notified due to the publishing in last step. it creates the proxy for the published services with a higher ranking. Then it stops the local PF. At this point, the PFs on device 1 are consuming services provided by the ESSI. The sequence diagram of migration process is shown in Fig. 5.

Besides device 1 and the ESSI, a third framework instance on device 2 is using the service being migrated. When POEM Manager in the ESSI signals the new service, POEM Manager on device 2 creates proxy for the new service with a higher ranking as device 1 does. When POEM Manager in the ESSI signals the service recycling, POEM Manager on device 2 recycles the proxy for that service. Other PFs on device 2 are not disturbed during the process.

5 Performance evaluation

This section describes POEM performance evaluation through both macro-benchmarks and micro-benchmarks. Then migration evaluation is then followed.

5.1 Methodology

The POEM Manager is implemented on Felix [23] OSGi implementation version 4.0.3. Mobile application that contains a Felix OSGi framework instance that hosts POEM Manager runs on Android Motorola phone A855. The phone's parameters are 600MHz CPU and 256M memory. The Android version is 2.2.3. The virtual machine is with 1GHZ CPU and 512M memory, which runs Ubuntu 11.10.

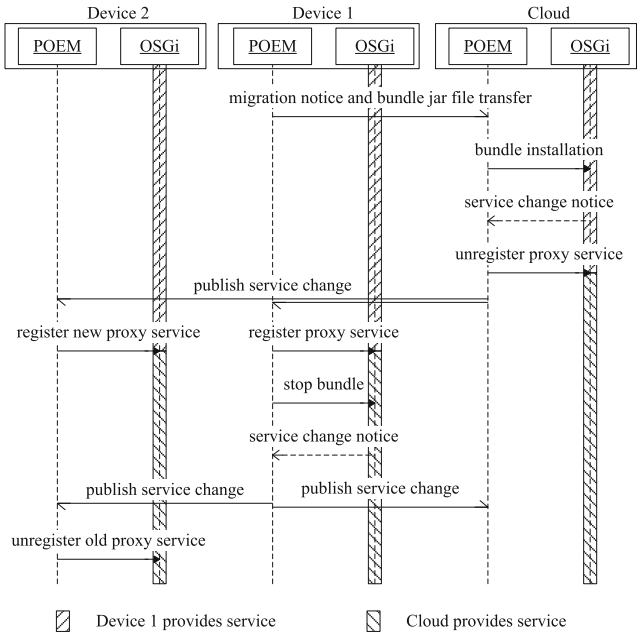


Fig. 5 PF migration sequence

Four applications are used to evaluate the POEM performance. They are Fibonacci sequence generator, N-Queens puzzle, nested loop and permutation generator. The Fibonacci application generates Fibonacci sequence in a recursive manner. Its time complexity is $O(2^n)$ and its stack usage is high due to recursive algorithm. The N-Queens application calculates all solutions for input chessboard size. Its time complexity is $O(n^2)$ and its stack usage is also high due to recursive algorithm. The nested loop application contains a six layer loop which leads to time complexity $O(n^6)$. The permutation application's time complexity is $O(n!)$ and uses little memory. Experiment result is obtained by running the application 50 times for every scenario and averaged. Between two consecutive executions there is a pause of 1 second.

The experiments are run under two scenarios:

- Phone: Applications are run only in phone.
- WiFi: Phone is connected to the ESSI through WiFi.

The WiFi connection has averaged latency of 70 ms, download bandwidth of 7 Mbps, and upload bandwidth of 0.9 Mbps. Ping is used to report the average latency from the phone to the ESSI, and Xtremelabs Speedtest, downloaded from Android market, is used to measure download and upload bandwidth.

5.2 Macro-benchmarks

For typical input parameter values, four applications are run on phone and in the ESSI separately. The application running time is recorded in Table 1. By subtracting time on phone and in the ESSI, the max speed up is put in the last column of the table. However, the max speed up is seldom achieved due to cost of communication and proxy. This cost changes little while offloading benefit changes much, so there should be some point when the benefit of offloading surpasses its cost giving application net gain.

Fibonacci application takes a sequence index number and calculates the corresponding number in the Fibonacci sequence. Figure 6a shows execution time of Fibonacci application. The intersection of execution time on phone and WiFi offloading is the Boundary input value (BIV) [9] that shows the offloading benefit starting point. N-Queens application takes chess board size and calculates all solutions and return solution number. Figure 6b shows execution time of N-Queens application. The execution time on phone rises dramatically as the chessboard size increases one scale. Offloading offers benefit after chess size is larger than 10.

Table 1 Execution time for phone and cloud, and the potential max speed up by offloading

Case	Input	Phone (ms)	Cloud (ms)	Max speed up (ms)
Fibonacci	26	59.25	2	57.25
	27	99.5	3.05	96.45
	28	156.75	5	151.75
	29	251	7.65	243.35
	30	408.25	12	396.25
N-Queens	8	11	1.1	9.9
	9	39.75	3.05	36.7
	10	222.75	12.2	210.55
	11	1593.5	64.4	1529.1
	12	9630.25	377.2	9253.05
Nested loop	14	157	15.05	141.95
	15	332	21.55	310.45
	16	276.75	28.6	248.15
	17	392.5	39.85	352.65
	18	560.25	54.35	505.9
Permutation	5	1.25	0.25	1
	6	1	0.25	0.75
	7	6.5	0.4	6.1
	8	49.25	2.05	47.2
	9	1124.75	12.1	1114.65

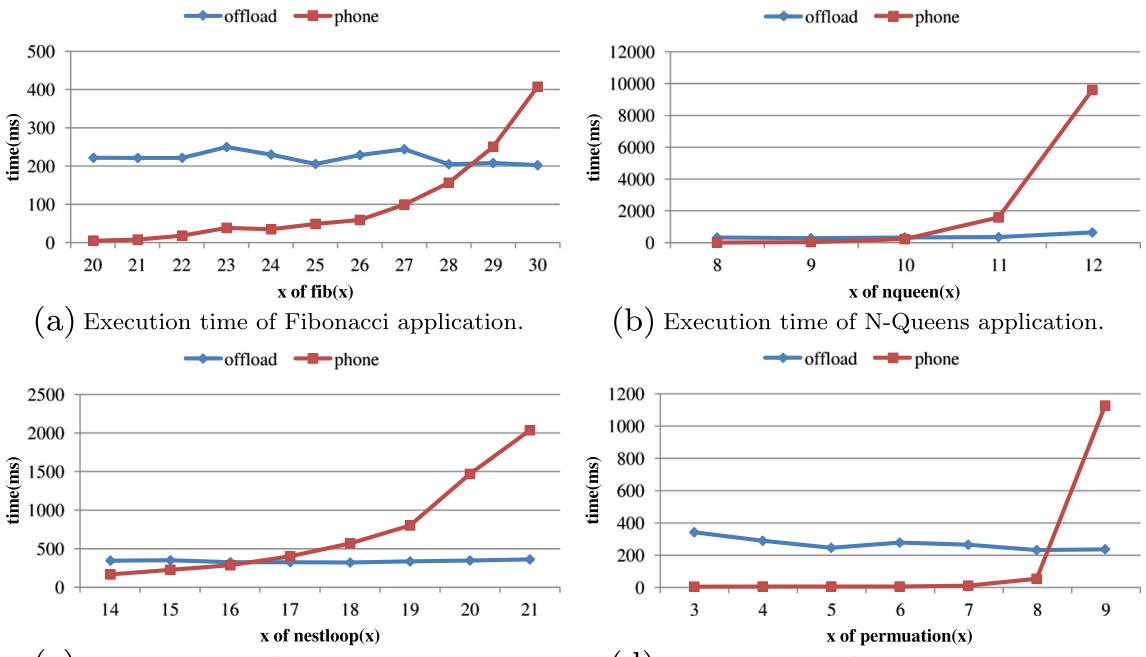
Nested loop application takes loop times and execute loop without memory operation. The execution time on phone is convex, which means it is less than exponential increase compared to the above two applications that requires both computing and storage. The execution time of offloading increases slowly. The Permutation application takes a max number N and returns count of prime number within the range (1,N). The prime number searching algorithm used is Permutation algorithm. The execution time increases on phone, however the execution time for offloading approach almost remains same.

The offloading line of four applications is increasing slowly compared to phone line. As the phone line starts from a low point, which indicates the application runs fast when input is small, the offloading line and phone line intersects finally. Comparing offloading line and the ESSI execution time column in Table 1, the slow increase is reasonable due to execution time increase slowing in the ESSI as well. Besides, the starting point of offloading line is higher than phone line, so there must be cost for remote method invocation.

5.3 Micro-benchmarks

This experiment measures service invocation time. This time is measured on phone where is service consumer side. The remote service consuming time consists of three parts: marshaling time of both consumer and provider sides, network transfer time and actual execution time. The result is shown in Figure 7.

Figure 7 shows time against different input parameters. From the table, the actual execution time is similar to the execution in the ESSI of column the ESSI in Table 1. At



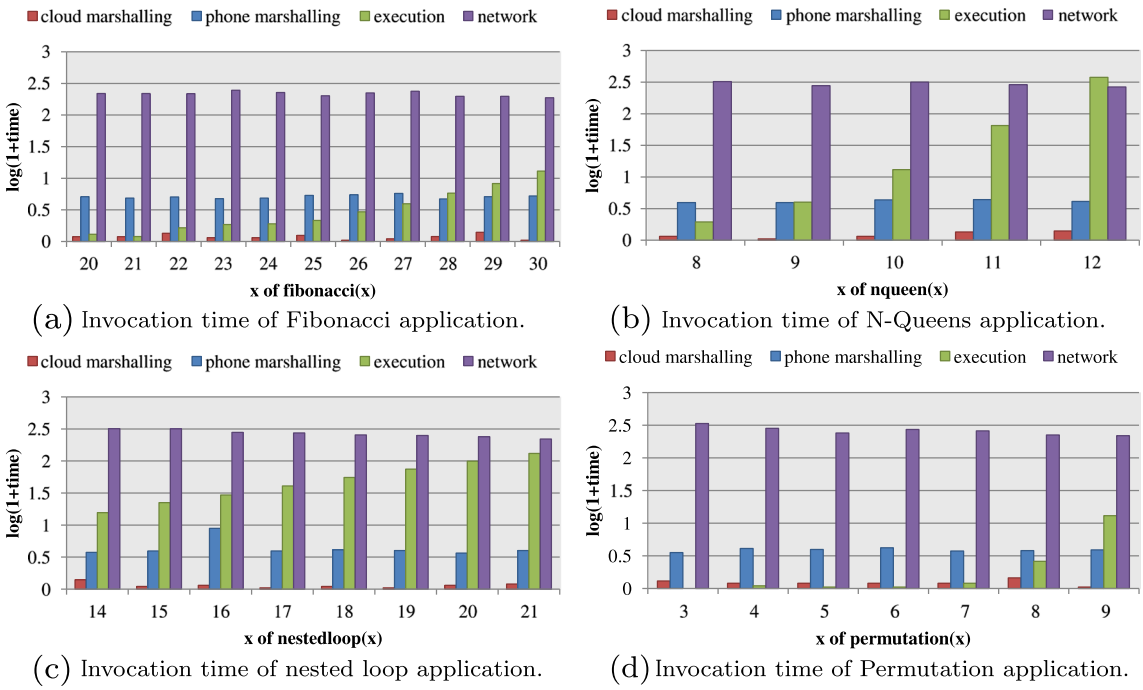
(a) Execution time of Fibonacci application.

(b) Execution time of N-Queens application.

(c) Execution time of nested loop application.

(d) Execution time of Permutation application.

Fig. 6 Execution time of four applications for both offloading and local execution scenarios



(a) Invocation time of Fibonacci application.
(b) Invocation time of N-Queens application.
(c) Invocation time of nested loop application.
(d) Invocation time of Permutation application.

Fig. 7 Time cost of each steps in remote service invocation process

the beginning, execution time is nearly zero. The execution time increases along with input parameter value increases. Figure 7 shows that marshaling time is relatively small compared to network delay. Figure 7 also shows that the main cost for remote method invocation is network delay around BIV point. And marshaling time and network time against different input parameters are approximately identical. The marshaling and network cost decides the start points of offloading line in Fig. 6a–d. And execution time decides the trend of those offloading line. If the network delay or the marshaling is reduced in some situation, the offloading line will drop and then BIV point will go to left, which means the range of benefit increase and application components are supposed to be offloaded to the ESSI. In another perspective, if component's ratio of computation cost to network cost increases, it is better to offload that component to the ESSI.

Besides service invocation time, the proxy generation time is also measured. The proxy generation time indicates POEM initialization time, which is paid once at starting POEM Manager.

5.4 PF migration

This experiment measures PF migration time. PF migration time period starts when service migration command is issued and ends when proxy for migrated service is available. The result is in Table 2 which shows that the migration time is nearly same for the tested four applications. This is reasonable because the migration time is mainly the time of transferring PF bundles on the network and these four PF bundle sizes are similar.

6 Case study

The POEM framework is used in the MIDAS [24] project, which develops the system that proactively manages the interacting traffic demand and the available transportation supply. This project demonstrates the synergistic use of a cyber-physical infrastructure consisting of smart-phone devices; cloud computing, wireless communication, and intelligent transportation systems to manage vehicles in the complex urban network – through the use of traffic controls, route advisories and road pricing – to jointly optimize drivers' mobility and the sustainability goals of reducing energy usage and improving air quality. A key element of MIDAS is the data collection and display device PICT that collects each participating driver's vehicle position, forward

Table 2 PF migration time cost

Cases	Migration time (ms)
Fibonacci	272
N-Queens	335
Nested loop	290
Permutation	304

images from the vehicle's dashboard, and communication time stamps, and then displays visualizations of predicted queues ahead, relevant road prices, and route advisories.

6.1 Setup

The same smartphone Motorola A855 in previous evaluation is used as PICT device. The smartphone captures the road image which is processed by calling the image process PF exposed by one VM in the cloud as shown in Fig. 8. The smartphone communicates with the cloud through WiFi while the other network configuration is the same as previous evaluation.

The image process part is implemented based on OpenCV [25] library. We used the bilateral filter as the example PF for image processing. Bilateral filter can reduce unwanted noise very well while keeping edges fairly sharp. One of the parameters to the bilateral filter is the filter size that is the diameter of each pixel neighborhood that is used during filtering. We run the application against different filter size and try to find the BIV for this application. The test image size is around 45 KB and it is with 800 × 600 pixels.

6.2 Evaluation

We measure the time cost of three indicators in this scenario as shown in Fig. 9. The 'phone' series indicates the time cost without remote service composition, which means all the computation happens on the smartphone locally. The Fig. 9 shows the 'phone' series grows drastically from 509 milliseconds to 17,847 milliseconds on average along with the filter size increases from 2 to 20. Since the VM in the cloud is much more powerful than the smartphone, the time cost

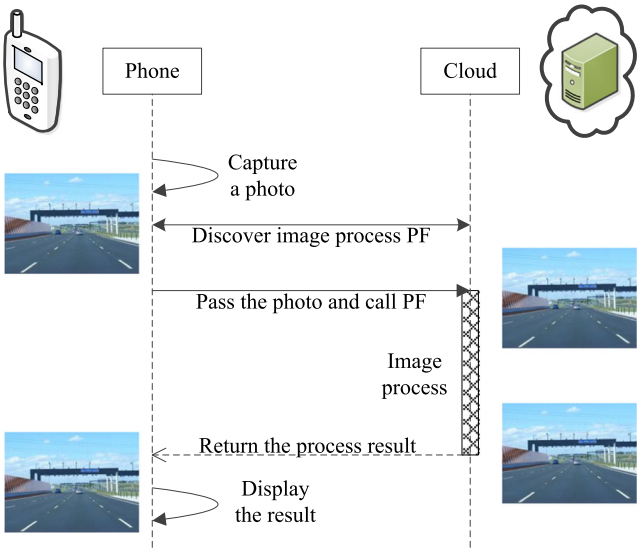


Fig. 8 Image process application case scenario

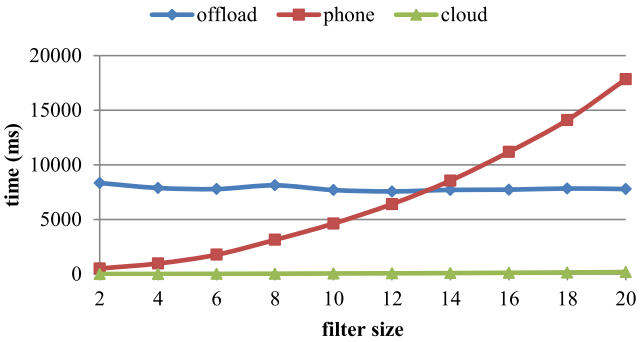


Fig. 9 Execution time of image process application for local, cloud and offloading execution scenarios

on the same filter size parameter is much less than the time spent for phone local execution. The 'cloud' series shows the time spend for the image process in the VM, which grow from 5 milliseconds to 183 milliseconds while the filter size is from 2 to 20. Compared to the 'phone' series, the 'cloud' series is almost a horizontal line. The 'offload' series measures the total time spent for calling remote image process PF, including the time indicated by 'cloud'. The area between the 'offload' series and the 'cloud' series presents the time spent on the network, which is around 7.8 seconds in the evaluation configuration for the test image. The 'offload' series actually adds the image transfer time on to the 'cloud' series and the major time spent for the image process application is for image transfer. The Fig. 9 shows the BIV point is around 13. If the application requires more obvious filter result by altering the filter size greater than the BIV, the application had better offload the image process to the cloud to gain time benefit.

6.3 Potential improvement

The above evaluation incorporates one mobile device, however there are thousands of vehicles in the urban area. If the image process PF is hosted on one POEM framework in one VM, the remote PF composition requests may quickly consume all the resources on the VM. Possible solution is to separate the data collection task to multiple VMs and define the limit for each POEM framework. In this way, the work load is distributed to a bunch of VMs and the system can scale up by adding more VMs.

7 Conclusion

This paper proposes a novel application running platform for mobile cloud computing that allow mobile users to offload and compose mobile cloud application with little management overhead. The implementation is based on

OSGi platform and XMPP protocols. The proposed service platform handles service migration, service discovery and service composition seamlessly in a transparent fashion. The evaluation shows the proposed service platform is flexible and efficient. The future work on POEM is to improve security and privacy control of the POEM system. Moreover, the service discover should incorporate more social network features to make the discovery scalable and customizable.

Acknowledgment The authors would like to thank NSF CPS #1239396 grant to support the research on the MIDAS project.

References

1. Huang D, Xing T, Wu H (2013) Mobile cloud computing service models: a user-centric approach. IEEE Netw 27(5):6–11
2. OSGi Alliance. OSGi Core Release 5, March 2012. http://www.osgi.org/Release5/HomePage
3. Extensible Messaging and Presence Protocol (XMPP), available at http://xmpp.org/. Open Source
4. March V, Gu Y, Leonardi E, Goh G, Kirchberg M, Lee BS (2011) ucloud: Towards a new paradigm of rich mobile applications. In: 8th international conference on mobile web information systems (MobiWIS)
5. Ma RKK, Lam KT, Wang CL (2011) excloud: Transparent runtime support for scaling mobile applications in cloud. In: International conference on cloud and service computing (CSC). IEEE, pp 103–110
6. Ma RKK, Lam KT, Wang CL, Zhang C (2010) A stack-on-demand execution model for elastic computing. In: Proceedings of the 39th International Conference on Parallel Processing (ICPP 2010), pp 208–217
7. Chun BG, Ihm S, Maniatis P, Naik M, Patti A (2011) Clonecloud: Elastic execution between mobile device and cloud. In: Proceedings of the 6th conference on computer systems. ACM, pp 301–314
8. Cuervo E, Balasubramanian A, Cho D, Wolman A, Saroiu S, Chandra R, Bahl P (2010) Maui: making smartphones last longer with code offload. In: Proceedings of the 8th international conference on Mobile systems, applications, and services. ACM, pp 49–62
9. Kosta S, Aucinas A, Hui P, Mortier R, Zhang X (2012) Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In: 2012 Proceedings IEEE INFOCOM, pp 945–953
10. Kemp R, Palmer N, Kielmann T, Bal H (2012) Cuckoo: a computation offloading framework for smartphones. In: Mobile Computing, Applications, and Services, pp 59–79
11. Zhang X, Jeong S, Kunjithapatham A, Gibbs S (2010) Towards an elastic application model for augmenting computing capabilities of mobile platforms. In: Mobile wireless middleware, operating systems, and applications, pp 161–174
12. Satyanarayanan M, Bahl P, Caceres R, Davies N (2009) The case for vm-based cloudlets in mobile computing. IEEE Pervasive Comput 8(4):14–23
13. Chen M, Zhang Y, Li Y, Mao S, Leung V (2015) Emc: Emotion-aware mobile cloud computing in 5g. IEEE Netw 29(2):32–38
14. Chakareski J (2013) Adaptive multiview video streaming: challenges and opportunities. IEEE Commun Mag 51(5):94–100
15. Corradi A, Fanelli M, Foschini L (2014) Vm consolidation: a real case based on openstack cloud. Futur Gener Comput Syst 32:118–127
16. Gerla M (2012) Vehicular cloud computing. In: 2012 The 11th annual mediterranean ad hoc networking workshop (Med-Hoc-Net), pp 152–155
17. Hsu C-Y, Yang C-S, Yu L-C, Lin C-F, Yao H-H, Chen D-Y, Robert Lai K, Chang P-C (2014) Development of a cloud-based service framework for energy conservation in a sustainable intelligent transportation system. Int J Prod Econ 164:454–461
18. Huang D, Zhang X, Kang M, Luo J (2010) Mobicloud: Building secure cloud framework for mobile computing and communication. In: 5th IEEE international symposium on service oriented system engineering (SOSE), pp 27–34
19. Fielding RT, Taylor RN (2002) Principled design of the modern web architecture. ACM Trans Internet Technol (TOIT) 2(2):115–150
20. Wu H, Huang D, Bouzefrane S (2013) Making offloading decisions resistant to network unavailability for mobile cloud collaboration. In: 9th international conference on collaborative computing: networking, applications and worksharing (Collaboratecom). IEEE, pp 168–177
21. Wu H, Huang D (2014) Modeling multi-factor multi-site risk-based offloading for mobile cloud computing. In: 10th international conference on network and service management (CNSM). IEEE, pp 230–235
22. Wu H, Huang D (2015) Mosec: Mobile-cloud service composition. In: 3rd international conference on mobile cloud computing, services, and engineering (MobileCloud). IEEE
23. Apache Felix. http://felix.apache.org/index.html
24. A cyber physical system for proactive traffic management to enhance mobility and sustainability. https://mobile.mobicloud.asu.edu/poem/midas-cps
25. OpenCV library. http://opencv.org/