J Supercomput (2014) 67:84–103
DOI 10.1007/s11227-013-0988-6

Investigation on runtime partitioning of elastic mobile applications for mobile cloud computing

Muhammad Shiraz · Ejaz Ahmed · Abdullah Gani · Qi Han

Published online: 2 August 2013

Abstract The latest developments in mobile computing technology have increased the computing capabilities of smartphones in terms of storage capacity, features support such as multimodal connectivity, and support for customized user applications. Mobile devices are, however, still intrinsically limited by low bandwidth, computing power, and battery lifetime. Therefore, the computing power of computational clouds is tapped on demand basis for mitigating resources limitations in mobile devices. Mobile cloud computing (MCC) is believed to be able to leverage cloud application processing services for alleviating the computing limitations of smartphones. In MCC, application offloading is implemented as a significant software level solution for sharing the application processing load of smartphones. The challenging aspect of application offloading frameworks is the resources intensive mechanism of runtime profiling and partitioning of elastic mobile applications, which involves additional computing resources utilization on Smart Mobile Devices (SMDs). This paper investigates the overhead of runtime application partitioning on SMD by analyzing additional resources utilization on SMD in the mechanism of runtime application profiling and partitioning. We evaluate the mechanism of runtime application partitioning on SMDs in the SmartSim simulation environment and validate the overhead of runtime application profiling by running prototype application in the real mobile computing environment. Empirical results indicate that additional computing resources are uti-

M. Shiraz (✉) · E. Ahmed · A. Gani · Q. Han
Mobile Cloud Computing Research Lab, Faculty of Computer Science and Information Technology, University of Malaya, Kuala Lumpur, Malaysia
e-mail: muh_shiraz@siswa.um.edu.my

E. Ahmed
e-mail: ejazahmed@ieee.org

A. Gani
e-mail: abdullah@um.edu.my

Q. Han
e-mail: hanqi@siswa.um.edu.my

lized in runtime application profiling and partitioning. Hence, lightweight alternatives with optimal distributed deployment and management mechanism are mandatory for accessing application processing services of computational clouds.

Keywords Mobile cloud computing · Elastic applications · Distributed systems · Application offloading

1 Introduction

The recent developments in mobile computing technology have enriched mobile devices with smart computing capabilities. Users enjoy the computing and communication services of SMDs with the freedom of mobility. SMDs incorporate the computing potentials of PDAs and voice communication capabilities of cellular phones by providing support for new user applications and multimodal connectivity for accessing cellular and data networks. Therefore, SMDs are expected as the leading future computing devices with high user expectations for employing computational intensive applications. Examples of the intensive applications include speech recognition, natural language processing, computer vision and graphics, machine learning, augmented reality, planning and decision making [39]; however, the intrinsic limitations in the wireless access medium and mobile nature of SMDs obstruct the employment of intensive mobile applications [18, 33, 34, 36]. The latest approach to alleviate resources limitations in SMDs is the employment of Cloud Computing (CC) services and resources, which are accessed by using traditional internet technologies [12]. CC implements the vision of computing utility and employs diverse IT business models such as on-demand, pay-as-you-go, and utility computing for the provisioning of computing services [5, 8]. For example, Amazon Web Services (AWS) are utilized to store personal data through its Simple Storage Service (S3) [3], and Elastic Cloud Compute (EC2) is employed for application processing services [5]. Successful practices of CC for stationary computers motivate for leveraging cloud resources and services for SMDs.

MCC envisions software level solutions for addressing the issues of resources limitations in SMDs by leveraging the services and resources of computational clouds on a demand basis. Currently, a number of techniques are endeavored for the augmentation of the computing potentials of SMDs [2, 25]. Application offloading is ascertained as a software level solution for addressing the issue of resources incapacitation in SMDs. The traditional offloading algorithms [10, 11, 17, 35] implement static partitioning or dynamic partitioning algorithms for the distribution of processing load between powerful server nodes and resources constraint SMDs. The critical aspects of offloading frameworks are the runtime application profiling and partitioning, which utilize additional computing resources (CPU, battery power) on SMD. Moreover, the establishment of distributed platform at runtime involves the issues of resources utilization on SMD in cloud server arbitration for partition offloading, application partition migration, and security threats for partition migration [23]. For instance, the profiler mechanism searches for computational intensive components of the application and the solver mechanism separates the intensive components of

the mobile application. The migrator mechanism arbitrates with cloud datacenters for the selection appropriate remote server node in cloud datacenter and transfers the partitions of the application at runtime. Therefore, resources intensive distributed platform is established at runtime. This paper investigates the overhead of application partitioning by analyzing additional resources utilization in runtime application profiling and partitioning on SMD. SmartSim [40] is deployed for evaluating resources utilization in runtime application profiling and partitioning. A prototype application is tested in the Android platform for analyzing the additional cost of runtime application profiling. Empirical results indicate that additional computing resources are utilized in runtime application profiling and partitioning. Hence, lightweight alternatives are mandatory for accessing application processing services of computational clouds. The contribution of the paper lies in analyzing additional resources utilization in application profiling and partitioning of elastic mobile application for mobile cloud computing, which assists in proposing lightweight procedures for computational load in MCC.

The paper is divided into the following sections. Section 2 presents fundamental concepts of cloud computing, mobile cloud computing, and application offloading for MCC. Section 3 reviews current elastic application offloading algorithms. Section 4 discusses methodology used to experiment and evaluate the overhead in runtime application profiling and partitioning. Section 5 presents results and discussion of the analytical findings. Finally, Sect. 6 draws concluding remarks and future directions.

2 Background

This section discusses the fundamental concept of cloud computing and mobile cloud computing and explains the mechanism of application offloading for MCC.

2.1 Mobile cloud computing

Mobile cloud computing is the evolving computing model, which extends the utility computing vision of computational clouds to resources constrained SMDs. Cloud computing is based on the centralization of resources in cloud datacenters and provision of resources on demand basis [22]. Service providers provide services in the form of various service models; Software as a Service (SaaS), Infrastructure as a Service (IaaS), and Platform as a Service (PaaS) [5]. In the SaaS model, computer software in the cloud datacenters is provided access on demand basis. An example of the SaaS includes GoogleDocs [19], which provides free access for spreadsheet and word processing tools. Platform as a Service (PaaS) model enables to rent hardware, operating systems, storage, and network capacity over the Internet. For instance, the Google App Engine provides an application developmental and deployment platform in Google's data centers [16]. The IaaS model of the computational clouds enables consumers to outsource storage, hardware, servers, and networking components. AWS [42] offers infrastructure as a service and software as service, which enable to utilize the virtualize resources and services in cloud datacenters. AWS are utilized to store personal data through its Simple Storage Service (S3) [5].

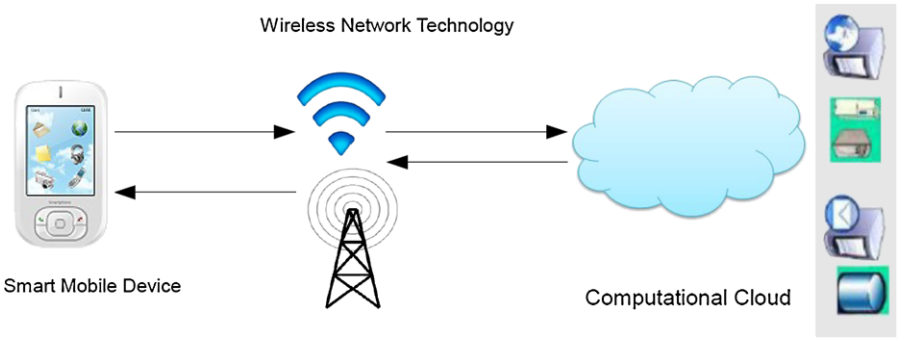

Fig. 1 Model of mobile cloud computing [39]

MCC envisions a distributed computing model, which addresses the issue of resources limitations in SMDs. In this vision, mobile users seamlessly access the services and resources of a computational cloud to obtain the resource benefits at low cost on the move [1]. MCC leverages the storage and application processing services of resources rich and powerful centralized computing datacenters in computational clouds for SMDs [32]. The attributes of centralized management, availability of resources, and scalability of services and on demand access to widespread service on the move are the motivating factors for leveraging cloud services and resources for SMDs. MCC is an attractive computing model for the business persons for the reason of a profitable business option, which reduces the development and execution cost of mobile applications. It enables mobile users to acquire new technology conveniently on a demand basis and on the move [26]. Different strategies are employed for the augmentation of SMDs including screen augmentation, energy augmentation, storage augmentation, and application processing augmentation of SMD [1]. The MCC model is composed of three components: mobile devices, wireless technologies, and computational cloud. Mobile devices use cellular networks (such as 3G, LTE) or data networks (such as Wi-Fi) to access the services of computational cloud in mobile environment. As SMD inherits its nature of mobility, it needs to execute location-aware services, which consume resources and turned it to be a low-powered client. Figure 1 shows the model of MCC wherein the computational cloud is integrated with SMD via wireless network technologies.

The storage services of computational clouds are utilized for augmenting the storage capacity [13], whereas the application processing services are leveraged for augmenting processing capabilities of SMDs [26]. The application processing capabilities of SMDs are augmented by offloading computational intensive components of the mobile applications to cloud datacenters.

2.2 Application offloading for MCC

The mechanism of outsourcing computational load to remote surrogates in the close proximity is called cyber foraging [24]. In MCC, smartphones implement application offloading to utilize the computing power of the cloud. The mechanism of outsourcing computational load to remote server node is called application offloading.

Researchers extend process offloading algorithms for pervasive computing [30], grid computing [27], and cluster computing [7]. In recent years, a number of cloud server based application offloading frameworks are introduced for outsourcing computational intensive components of the mobile applications partially or entirely to cloud datacenters [13, 15, 20, 22, 43]. Mobile applications which are attributed with the features of runtime partitioning are called elastic mobile applications. Elastic applications are partitioned at runtime for the establishment of a distributed processing platform [39]. For instance, Amazon released Silk, which is a cloud-accelerated Web browser [21] whose software resides both on Kindle Fire and EC2. With the web page request, Silk dynamically determines the distribution of processing load between mobile hardware and Amazon EC2. The workload distribution considers different factors such as network conditions, page complexity, and the location of any cached content [6]. The traditional offloading frameworks implement a static or dynamic partitioning approach for the division of application processing load at runtime. In static application partitioning [14], the application is partitioned in a fixed number of partitions either at compile time or runtime. The computational intensive partitions of the applications are outsourced to remote servers for offload processing [39]. The dynamic partitioning approach is implemented to address the issue of dynamic application processing load on SMDs at runtime [13, 15, 43]. In dynamic partitioning, the application is partitioned dynamically at runtime casually or periodically. In casual partitioning, the runtime partitioning mechanism is activated in critical conditions to offload intensive components of mobile application, whereas in periodic partitioning the profiling mechanism evaluates computing resources utilization on SMD periodically for resolving critical condition.

The dynamic partitioning mechanism is employed in two steps, which include application profiling and solving [13]. The mechanism in which the intensive components of the application are identified is called application profiling, whereas the mechanism in which the intensive components are separated from the application for offloading is called application solving. The profiling mechanism evaluates computing resources requirements of mobile application and the availability of resources on SMD. In critical condition (the unavailability of sufficient resources on SMD) elastic mobile application is partitioned and the computational intensive components of the application are offloaded dynamically at runtime. SMDs negotiate with cloud servers for the selection of appropriate server node. At that moment, the intensive partitions of the application are migrated to remote server node for remote processing. Upon successful execution of the remote components of the application, the result is returned to the main application running on SMD.

3 Traditional elastic application frameworks for mobile cloud computing

The application offloading frameworks implement runtime application partitioning in two ways: static partitioning or dynamic partitioning [39]. The static application partitioning involves one time application partitioning mechanism for the distribution of workload between SMD and cloud server node. The intensive components of the application are partitioned and transferred to the remote server node. The distribution of workload between SMDs and cloud server nodes occurs only once. For

example, the primary functionality offloading [17] mechanism involves partitioning and offloading of the intensive components at runtime. The interactive components of the application are configured on SMD, whereas the intensive and none interactive components of the application are offloaded to remote server node. VM based cloudlet [35] involves the overhead of virtual machine deployment and management for component offloading at runtime. In [37, 41], we investigate the impact of VM deployment for application processing. MISCO [14] implements a static partitioning approach for the classification of the application processing load between Map and Reduce functions. Map function is applied on the set of input data that produces <key, value> pairs, which are grouped into a number of partitions. The intermediate results of every partition are passed to a reduce function, which returns the final results. Application developers classify the functionalities of the application as the Map and Reduce functions. Mobile devices serve as worker nodes and are monitored through the centralized master server. The worker nodes provide the services of processing Map and Reduce functions. Static partitioning is a lightweight mechanism for the distribution of workload between SMD and cloud server node. However, it lacks in coping with the dynamic processing load of the mobile device. Therefore, the latest application offloading frameworks implement the dynamic partitioning mechanism.

Dynamic partitioning of the intensive mobile application is a robust technique for solving the critical condition of resources limitations repeatedly on SMD [39]. The traditional dynamic partitioning approaches evaluate the statistics of resources utilization on SMD and execution requirements of the application at runtime. In [15], a middleware framework is proposed for the dynamic distribution of application processing load between SMD and the cloud server node. The framework deploys the application partitioning in the optimal mode and dynamically determines the execution location for modules of the mobile application. The partioning mechanism determines the intensive components and splits the application in modules on the basis of its behavior. The framework implements the K-Step algorithm for dynamic partition at runtime, whereas the ALL algorithm is employed for static partitioning of the application. In order to reduce search space, the framework implements a preprocessing mechanism on the consumption graph. Preprocessing separates local and remote bundles of the application. It searches for the application modules, which can result in high cost in offloading and for that reason are not feasible for offloading. The framework distributes workload as per the statistics of resources available on SMD. The framework determines the optimal solution for the optimization problem in order to optimize different objective functions, such as interaction time, communication cost, and memory consumption. The runtime partitioning strategy, however, requires additional computing resources in the dynamic application partitioning mechanism. The distributed platform in AIDE [29] is composed of computing devices in the locality of mobile devices; SMD access, a preconfigured surrogate server, which maintains information of the volunteer server nodes. The runtime profiling mechanism of the framework is activated dynamically, which implements class level granularity for partitioning of mobile application. The application profiler determines the feasibility of partition offloading on the basis of the execution history of the application and prediction of the future resources required for the application. AIDE provides a transparent distributed application deployment framework for mobile applications and provides

the notion of application being processed on mobile device locally. However, the runtime partitioning of the application requires additional computing resources utilization for the establishment of a distributed platform.

Mobile Assistance Using Infrastructure (MAUI) [13] focuses on energy saving for SMD. Application developers identify the local and remote components of the application at design time. The MAUI profiler determines the feasibility of a remotely annotated method for offload processing. Each time a method is called, the profiler component assesses it for energy saving, which utilizes additional computing resources (CPU, energy) on SMD. The MAUI solver operates on the input provided by application profiler. It determines the destination of execution for the method annotated as remote. The framework implements application proxies on SMD and the cloud server node. MAUI generates a wrapper for each method marked as remote at compile time. The wrapper method is created with two changes: one additional input argument, and one additional return type. Input argument is required for the state transfer of smartphone to MAUI server through client application proxy. The additional return value is used to transfer the application state back from the server to the smart mobile device using server proxy. State of the method is transferred in serialized form. MAUI implements application level partitioning for outsourcing computational load of SMD. However, the mechanism of runtime application profiling and solving at runtime involves additional computing resources utilization for application partitioning. Development of the applications on the basis of MAUI requires additional developmental efforts for annotating the execution pattern of each individual method the application. MAUI deploys full proxies of the application on both SMD and the cloud datacenter. MAUI involves the overhead of dynamic application partitioning, component migration, and reintegration.

CloneCloud [11] involves partitioning and reintegration of the application at the application level. The partitioning phase of the framework includes static analysis, dynamic application profiling, and optimization solution. A preprocess migratory thread is implemented on mobile devices to assist in the partitioning and reintegration of the thread states. The elastic application model [43] provides a middleware framework for mobile applications. Application is dynamically partitioned into weblets, which are migrated dynamically to cloud server node. The framework implements different elastic patterns for the replication of weblets on the remote cloud. It considers different parameters for offloading of the weblets; such as status of the mobile device, cloud, application performance measures and user preferences, which comprise power saving mode, high speed mode, low cost mode, and offload mode. The framework considers an optimal cost model for the execution configuration of the weblets, which considers different costing factors such as power consumption, monetary cost, performance attributes, and security and privacy. The framework implements a resources intensive mechanism for runtime application partitioning and the migration of weblets between SMD and remote cloud nodes. It includes additional resources utilization on SMD in the process of application profiling, dynamic runtime partitioning, weblets migration and reintegration, and continuous synchronization with the cloud server node for the entire duration of application processing.

Traditional computational offloading frameworks for MCC involve the issues of additional resources utilization on SMD in the establishment of distributed plat-

form [39]. For instance, the profiler mechanism searches for computational intensive components of the application and the solver mechanism separates the intensive components of the mobile application. The migrator mechanism arbitrates with cloud datacenters for the selection appropriate remote server node in the cloud datacenter and transfers the partitions of the application at runtime. Therefore, resources intensive distributed platform is established at runtime.

4 Methodology

Experimental setup The overhead of runtime application profiling and partitioning is investigated in simulation and in the real mobile computing environment. SmartSim [40] is employed for the evaluation of the overhead of runtime application partitioning on SMDs. SmartSim is a simulation tool that models the application processing capabilities of the mobile device and mobile applications. It is deployed for the evaluation of resources utilization by mobile application on SMD. The resources provision algorithm schedule computing resources of the SMD for the processing of application or application components.

The overhead of runtime application profiling is further analyzed by benchmarking the prototype application for Android devices in the real mobile cloud computing environment. The experimental setup is composed of cloud server node, Wi-Fi wireless network, and Samsung Galaxy SII mobile device. The virtual device instance is employed on the server machine for the execution of offloaded application at runtime. The mobile device accesses the wireless network via a Wi-Fi wireless network connection of radio type 802.11g, with the available physical layer data rates of 54 Mbps. A Java based Android software development toolkit (Android SDK) is deployed for the development of the prototype application. Power Tutor tool [31] is used for the measurement of battery power consumption in distributed application processing.

Mobile application The simulated mobile application is composed of a finite set of components with a specific processing intensity (CPU) and memory allocation requirement (RAM). Mobile application is composed of 10 components with different computational intensities. The overhead of runtime application partitioning is analyzed by evaluating the partitioning mechanism of simulated mobile application with varying processing intensities in two different scenarios. In scenario I, the application is composed of none intensive operations and all the operations of the application are executed on SMD, whereas in scenario II represents the application is composed of intensive operations wherein SMD is not able to fulfill resources requirements of the intensive mobile application. Therefore, the execution of the application employs runtime optimization to resolve the critical condition and separate the intensive components of the applications. Application partitioning is a CPU intensive and time consuming mechanism. Therefore, we analyze the percent additional CPU utilization and percent additional time taken in separating the intensive components of the application. We use wall clock time for evaluating the execution time of the application in the simulation environment. Wall clock time is the time taken from submission of the application until the completion of the execution [9].

A prototype application is developed by using Service Oriented Architecture (SOA) of Android applications. The application tested the Android device for the evaluation of the additional overhead of runtime profiling mechanism on SMD. The prototype application is composed of three service components. (a) The sorting service component implements the logic of bubble sort for sorting linear list of integer type values. The sorting logic of the application is tested with 30 different computational intensities (11000–40000). (b) The matrix multiplication service of the application implements the logic of computing the product of 2-D array of integer type values. Matrix multiplication logic of the application is tested with 30 different computational intensities by varying the length of the 2-D array between $160 * 160$ and $450 * 450$. (c) The power compute service of the application implements the logic of computing $b\hat{}e$, whereas b is the base and e is the exponent. The power compute logic of the application is tested for 30 different computational intensities by varying the exponent between 1000000 and 200000000. Empirical data are collected by sampling all computational intensities of the application in 30 different experiments and the value of sample mean is signified with 99 % confidence interval for the sample space of 30 values in each experiment. The measurement parameter includes percent CPU utilization, Execution Time (ET) in Milliseconds (ms), percent RAM allocation and Energy consumption Cost (EC) in units of Joules (J).

5 Results and discussion

Current computational offloading frameworks [4, 10, 11, 13, 15, 17, 20, 22, 28, 35, 43] establish distributed application processing platform at runtime which utilizes computing resources of the mobile devices for the entire duration of distributed application processing. Such approaches focus on what components of the application to offload and where to offload the computational intensive components of the applications. Current frameworks lack of considering the overhead of runtime distributed application deployment on SMD. The overhead comprises computing resources (CPU, memory, battery) exploitation in application partitioning and component offloading dynamically at runtime.

5.1 Analysis of runtime application partitioning

The overhead of runtime application partitioning is analyzed by evaluating the simulated mobile application in two different scenarios. The computational intensity of mobile application is varied in eight different experiments. The profiler mechanism is activated in the critical situations, wherein insufficient resources exist on the SMD for the execution of mobile application. Profiler identifies the intensive components at runtime and activates the solver mechanism to separate the intensive components of the mobile application. The process continues until the mobile device becomes capable to execute the application locally. The profiler mechanism identifies the intensive components of the application to resolve the critical condition.

We evaluate percent CPU utilization and Execution Time (ET) of the application on SMD in scenarios I and II (as discussed Sect. 4). Figure 2 compares CPU utilization on SMD in scenarios I and II, wherein operations/components of the application

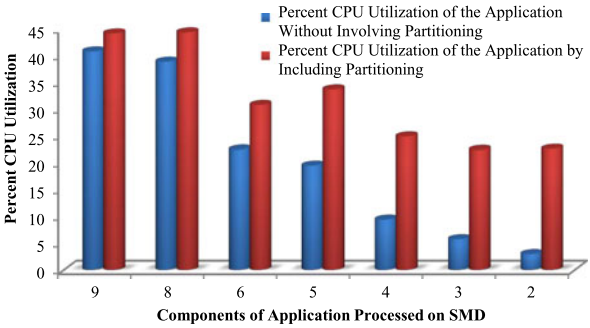

Fig. 2 Comparison of percent CPU utilization in scenarios I and II

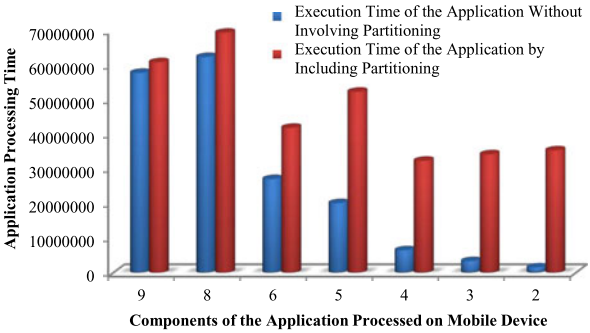

Fig. 3 Comparison of application execution time in scenarios I and II

with the same processing intensity are executed on SMD. It is found that scenario II involves partitioning of the intensive components application at runtime to resolve the critical condition, which increases CPU utilization. It is examined that the execution of nine components of the application with processing intensity 326 MI utilizes 40.8 % CPU in scenario I and 44.8 % CPU in scenario II. Similarly, the execution of five components of the application with processing intensity 157 MI utilizes 19.6 % CPU in scenario I and 33.7 % CPU in scenario II. In the same way, the execution of two components of the application with the processing intensity 23.6 MI utilizes 3 % CPU in scenario I and 22.6 %CPU in scenario II. It shows that in scenario II, creating a single partition, which has processing intensity 520 MI increases CPU utilization 3.4 %, creating five partitions of the application, which have processing intensity 2185 MI increases CPU utilization 14.2 % and creating eight partitions of the application, which have processing intensity 3039 MI increases CPU utilization 19.7 %.

Figure 3 compares the ET of the application in scenario I and scenario II. In both, the scenarios operations with the same processing intensity are executed on SMD. It is examined that the ET of the application increases in scenario II for the reason of partitioning the application at runtime. It is found that execution of nine components of the application with processing intensity 326 MI takes 57830107 wall clock time in scenario I and 60907932 wall clock time in scenario II. Similarly, the execution of five components of the application with processing intensity 157 MI takes 20099920 wall clock time in scenario I and 52393007 wall clock time in scenario II. In the same way, the execution of two components of the application with the processing intensity

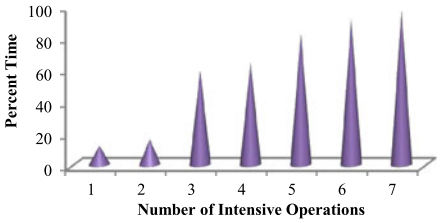

Fig. 4 Additional time taken in partitioning intensive component of the application

23.6 takes 1498245 wall clock time in scenario I and 35395345 wall clock time in scenario II. It shows that creating a single partition, which has processing intensity 520 MI increases the ET 5 %, creating five partitions of the application, which have processing intensity 2185 MI increases ET 61.6 % and creating eight partitions of the application, which have processing intensity 3039 MI increases ET 95.5 %.

Investigation of the application processing in scenarios I and II shows that runtime partitioning of the elastic mobile application requires additional computing resources on SMD. It is observed that the average CPU utilization of the SMD increases 10.4 % and an average ET of the application increases 54 % in scenario II for the processing application composed of 2–10 operations, which shows the overhead of runtime application partitioning.

In another example, it is found that the number of partitions created for an application depends on the computational intensity of the application. For instance, in the first experiment, the computational intensity of the application is kept low as a result the entire application is executed on the mobile device. Therefore, the overhead of runtime partitioning is found zero. In the second experiment, the total processing intensity of the application is 520 MI, wherein the profiler identifies a single highly intensive component of the application to resolve the critical condition and enable mobile application to be processed locally on mobile device. The remaining nine components of the application are executed on local mobile device. Similarly, the partitioning mechanism separates five intensive components of the application with processing length 2185 MI, whereas the remaining five components of the application are executed on local mobile device. In the same way, in another experiment eight intensive partitions are separated from the application of processing length 3039 MI, whereas a single component of the application is executed on the mobile device. It is examined that additional computing resources are utilized on SMD for each instance of application partitioning at runtime.

It is found that the overhead of runtime application profiling depends on two factors, which include the number of partitions created and the number of components in the mobile application. Therefore, larger instances of application partitioning at runtime utilizes higher computing resources for a larger period of time and, therefore, results in larger additional overhead on SMD. The mechanism of runtime application partitioning is time consuming and, therefore, increases the execution time of the application. Figure 4 shows the additional time taken in runtime partitioning of the mobile application. The runtime partitioning of single component of mobile application with processing intensity of 520 MI takes 12 % additional time of the total application processing time on SMD. The partitioning of four intensive components with the processing intensity 1770 MI takes 59 % additional time of the total application processing time on the mobile device. Similarly, partitioning of eight components

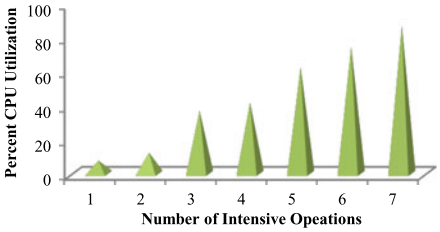

Fig. 5 Additional CPU utilization in partitioning intensive components of the application

with processing intensity 3039 MI takes 97 % additional time of the total application processing time on SMD. It shows that partitioning of larger number of intensive components results in higher resources utilization on SMD for a longer period of time. The increase in execution time of the application is for the reason that each instance of application partitioning involves additional time taken in profiler and solver activation for the identification and separation of the highly intensive component of the application at that specific instant. For instance, additional partitioning time in creating two partitions is 25 % higher as compared to creating a single partition as it involves the activation of partitioning mechanism two times. Similarly, the additional partitioning time in creating eight partitions is 83.5 % higher as compared to creating two partitions, as it involves the activation of partitioning mechanism eight times.

Figure 5 shows the additional CPU utilization in runtime partitioning of elastic mobile application. It is found that creating a single partition involves additional 27 MIPS() utilization in application partitioning, which constitutes 7.6 % of the total CPU utilization in application processing on the local mobile device. Similarly, partitioning six components of the application involves additional 124 MIPS utilization in application partitioning which constitutes 62.3 % allocation of the total CPU utilization for application processing. In the same way, creating eight partitions involves additional 157 MIPS utilization, which constitutes 87 % of the total CPU utilization for application processing on the mobile device. Analysis of the results show that the duration of CPU utilization is 83.5 % longer for separating eight intensive components as compared to separating a single intensive component of the application at runtime. It indicates that the mechanism of runtime application profiling and solving utilizes a higher percentage of CPU for a longer period of time in resolving larger instances of critical conditions.

The total overhead of runtime application partitioning is the sum of the resources utilization in application profiling and application solving. On the average, the profiler uses an additional 3.1 MIPS for profiling (1–8) components of the application at runtime and the solver uses additional 83.2 MIPS for solving (1–8) components of intensity level (520–3039 MIPS). Similarly, runtime application partitioning takes additional 52.6 % time and 40.4 % CPU of the application processing for partitioning (1–8) components of intensity level 520–3039 MIPS, which indicates the additional processing overhead on SMD in the deployment of runtime application partitioning.

5.2 Analysis of runtime application profiling

Profiling is employed for automatically partitioning elastic mobile application and partition offloading. For instance, application profiling mechanism dynamically evaluates availability of resources (CPU, RAM) on the mobile device and computational

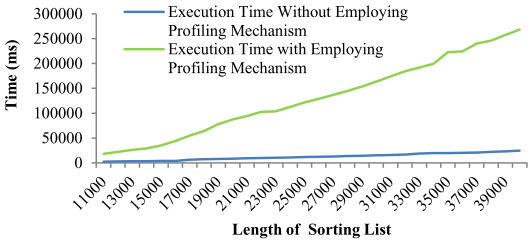

Fig. 6 Sort service execution time on SMD without employing profiling and by including profiling mechanism

requirements of mobile application. Similarly, the network profiling mechanism determines accessibility of network and quality of signal strengths while accessing the wireless access medium in MCC. Energy profiling mechanism examines utilization of battery power during the processing of mobile application. However, the implementation of runtime profiling mechanism is resources intensive, energy starving and time consuming. In this section, we analyze the impact of the application profiling mechanism on the execution time and energy consumption cost of the mobile application for Android devices. The components of mobile application are executed without including runtime profiling and by including runtime profiling mechanism.

Figure 6 shows the comparison the Execution Time (ET) of sorting service execution on the mobile device in two different scenarios, i.e., without profiling and with profiling. Sort service execution time is evaluated for 30 different computational intensities by varying the length of sorting operation between 11000–40000 values. It is examined that the ET on the local SMD is smaller for sorting operation in the scenario of without employing runtime profiling as compared to performing sorting operation with including profiling mechanism. For instance, the ET for sorting operation without employing profiling is found 2438 ms for list length 11000 values, 11647 ms for list length 25000 values, and 24468 ms for list length 40000 values. Whereas the ET for sorting operation with employing the runtime profiling mechanism is found 18318 ms for list length 11000 values, 121600 ms for list length 25000, and 268409 ms for list length 40000 values. It shows that by including the runtime profiling mechanism the value of ET increases 86.7 % for sorting list of 11000 values, 90.4 % for sorting list of 25000 values and 90.8 % for sorting list of 40000 values. The overall increase in the ET of sorting service by including runtime profiling in the sorting operation is found 90 % for sorting list length 11000–40000 in 900 experiments.

Figure 7 shows the comparison the Execution Time (ET) of matrix multiplication service execution on the mobile device in two different scenarios. Matrix multiplication service execution time is evaluated for 30 different computational intensities by varying the length of matrix 2-D arrays between $160 * 160$–$450 * 450$ values. It is examined that the ET on the local SMD is smaller for matrix multiplication operation in the scenario of without employing runtime profiling as compared to performing matrix multiplication operation by including the profiling mechanism. For instance, the ET for matrix multiplication operation without employing profiling is found 359 ms for matrices of length $160 * 160$, 1951 ms for matrices of length $290 * 290$ and 8248 ms for matrices of length $450 * 450$, whereas the ET for matrix multiplication operation by including profiling mechanism is found 960 ms for matrices of length

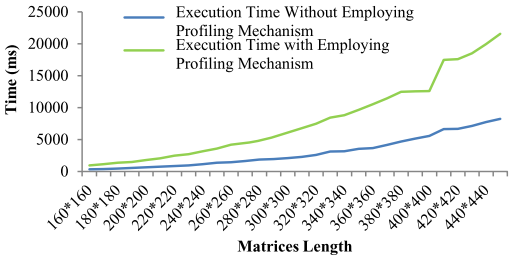

Fig. 7 Matrix multiplication service execution time on SMD without employing profiling and by including profiling mechanism

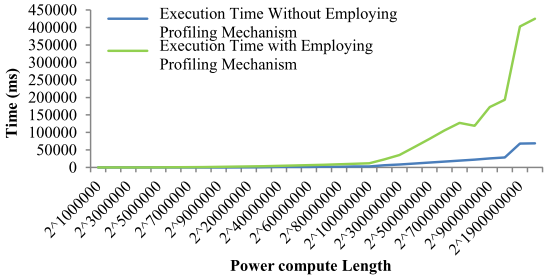

Fig. 8 Power compute service execution time on SMD without employing profiling and by including profiling mechanism

160 ∗ 160, 5445 ms for matrices of length 290 ∗ 290 and 21590 ms for matrices of length 450 ∗ 450. It shows that by including the runtime profiling mechanism the value of ET for matrix multiplication operation increases 62.6 % for multiplying matrices of length 160 ∗ 160, 64.1 % for multiplying matrices of length 290 ∗ 290 and 61.8 % for multiplying matrices of length 450 ∗ 450. The overall increase in the ET by including runtime profiling in the matrix multiplication service execution is found 63 % for 30 different intensities of the matrix multiplication operation.

Figure 8 shows the comparison the Execution Time (ET) of power compute service execution on the mobile device without employing profiling and with including profiling mechanism. The power compute service execution time is evaluated for 30 different computational intensities of power compute operation, i.e., 2ˆ1000000–2ˆ2000000000. It is examined that the ET on the local SMD is higher for power compute operation in the scenario of employing runtime profiling as compared to performing power compute operation without including the profiling mechanism. For instance, the ET for the power compute operation without employing profiling is found 51 ms for computing 2ˆ1000000, 1501 ms for computing 2ˆ50000000, and 69044 ms for computing 2ˆ2000000000, whereas the ET for the power compute operation with employing profiling is found 160 ms for computing 2ˆ1000000, 5885 ms for computing 2ˆ50000000, and 425382 ms for computing 2ˆ2000000000. It shows that by including the runtime profiling mechanism the value of ET for power compute operation increases 68 % for computing 2ˆ1000000, 74.5 % for computing 2ˆ50000000, and 83.8 % for computing 2ˆ2000000000. The overall increase in the ET by including runtime profiling in the power compute service execution is found 76.8 % for 30 different intensities of the power compute operation.

Figure 9 shows the comparison the Energy consumption Cost (EC) of sorting service execution on the mobile device by employing profiling and without employing the runtime profiling mechanism. The EC of sort service execution is evaluated for

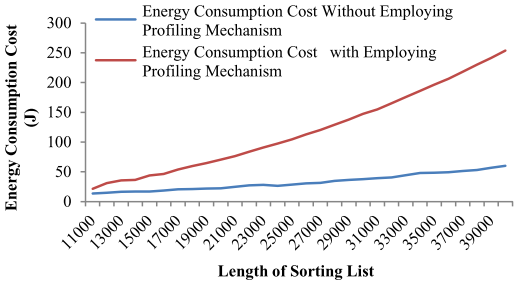

Fig. 9 Energy consumption cost of sort service execution on SMD without employing profiling and by including profiling mechanism

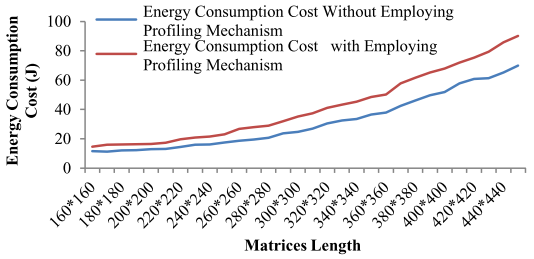

Fig. 10 Energy consumption cost of matrix multiplication service execution without employing profiling and by including profiling mechanism

different intensities of sorting operation (11000–40000). It is found that the runtime profiling mechanism increases the EC of application. For instance, EC for sorting operation without employing profiling is found: 13.6 J for list length 11000 values, 28.6 J for list length 25000 values, and 60.1 J for list length 40000 values, whereas the EC for sorting operation with employing runtime profiling mechanism is found 21.6 J for list length 11000 values, 104.7 J for list length 25000, and 253.6 J for list length 40000 values. The inclusion of runtime profiling mechanism increases energy consumption cost on mobile device as 37 % for sorting list of 11000 values, 73 % for sorting list of 25000 values, and 76 % for sorting list of 40000 values. The overall increase in the EC of sorting service by including runtime profiling in the sorting operation is found 68.4 % for sorting list length 11000–40000.

Figure 10 shows the increasing trend EC of matrix multiplication service execution on mobile device and the comparison of EC for 30 different intensities of matrix multiplication operation. The EC of the application depends on the intensity of operation being performed. It is found that higher intensive operations execute for a longer period of time and are energy starving. For instance, the EC of matrix multiplication service execution without employing profiling mechanism is examined 11.5 J for multiplying matrices of size $160 * 160$, 24.7 J for multiplying matrices of size $300 * 300$, and 69.9 J for multiplying matrices of size $450 * 450$. It shows the EC of matrix multiplication operation for matrices of size $450 * 450$ is 83.5 % larger as compared to multiplying matrices of length $160 * 160$, whereas the EC of matrix multiplication service execution with the inclusion of runtime profiling is found 14.6 J for multiplying matrices of size $160 * 160$, 35.2 J for multiplying matrices of size $300 * 300$, and 90.2 J for multiplying matrices of size $450 * 450$. It shows that the additional cost of application profiling increases for higher intensities of matrix multiplication operation. For instance, the increase in EC is found 21.2 % for multiplying matrices of length $160 * 160$, 29.9 % for multiplying matrices of length $300 * 300$,

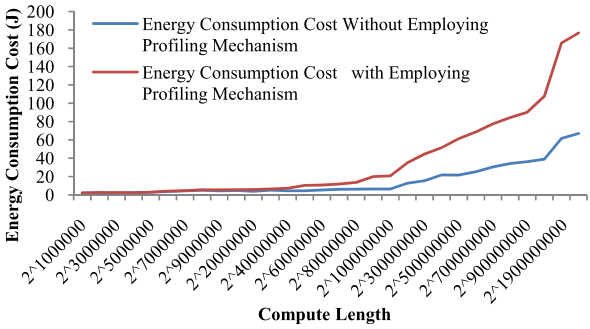

Fig. 11 Energy consumption cost of power compute service execution without employing profiling and by including profiling mechanism

and 22.5 % for multiplying $450 * 450$. The overall increase in the EC by including runtime profiling in the matrix multiplication service execution is found 25.1 % for 30 different intensities of the matrix multiplication operation.

Figure 11 shows the increasing trend EC of power compute service execution on the mobile device and the comparison of EC. The EC of power compute operation is evaluated with 30 different intensities of power compute operation in two different scenarios. It is found that EC increases for higher intensities of power compute operation in either scenario. For instance, the EC with of power compute service execution without employing profiling mechanism is examined 2.2 J for computing 2^1000000, 4.5 J for computing 2^50000000, and 67 J for computing 2^2000000000. It shows the EC of computing 2^2000000000 is 40 % larger as compared to computing 2^1000000. The EC of power compute service execution with the inclusion of runtime profiling is found 2.2 J for computing 2^1000000, 10.3 J for computing 2^50000000, and 176.7 J for computing 2^2000000000. Analysis of the results shows that the additional overhead of runtime profiling is negligible for smaller intensities of power compute service execution. For instance, the EC is found 2.2 J for computing 2^1000000 with including and excluding runtime profiling mechanism. However, the additional cost of application profiling increases for higher intensities of power compute operation. For instance, the increase in EC is found 11.5 % for computing 2^2000000, 39.1 % for computing 2^40000000, and 62.5 % for computing 2^2000000000. The overall increase in the EC by including runtime profiling in the power compute service execution is found 40 % for 30 different intensities of the matrix multiplication operation.

In runtime, application profiling additional RAM is allocated for maintaining state information of the running application. It is observed that the mechanism of runtime application profiling is allocated additional 8192 KB RAM for maintaining a temporary trace file in all instances of the experimentation. The prototype application is evaluated for offloading the running instance of the application component to the remote server node. At one instance, all the three services are executed locally on SMD, whereas in another instance the sorting service is offloaded to the remote server node. We evaluate computational load on SMD, and the impact of runtime component offloading on the resources allocation and execution time of the locally executing components. It is examined that offloading the intensive service at runtime increases system load and the resources utilization on SMD and adversely affects the execution time of the locally executing components of the mobile application. Analysis shows

that the CPU allocation to local services increases on average 8 % and the execution time of the locally executing services reduces on average 32.6 %. The increase in the local services execution time on SMD is for the reason of additional overhead of active service migration to the remote server node at runtime. Further, analysis indicates that in normal scenarios the operating system average CPU utilization remains is 5.6 % for the entire duration of application processing on the SMD. During the service outsourcing process, the average CPU utilization increases to 9.8 %, which shows 42.8 % increase in the CPU utilization during the runtime component offloading process. Hence, the runtime service migration mechanism increases the system CPU utilization up to 76.3 %.

The deployment of cloud based application processing for mobile application involves the following issues. (1) The traditional elastic mobile application frameworks [13, 43] require the classification of the components of mobile application at design time. Application developers are bound to annotate the individual components of application as local or remote. (2) The configuration of ad-hoc distributed platform dynamically at runtime is resources intensive and time consuming. SMDs arbitrate with the cloud datacenters for the selection of appropriate remote server node for each instance of application partition/component offloading. The unavailability of predefined server component of the application sets up a resources intensive distributed processing environment. (3) Traditional approaches for application offloading employ runtime profiling, and solving mechanism on SMDs to evaluate application processing requirements and the availability of computing resources on SMD. Therefore, the deployment of a distributed platform is resources intensive. (4) Current offloading frameworks require dynamic management of the distributed execution environment all through the remote processing of the mobile application. As a result, the computing resources on SMD are consumed for the entire duration of dynamic distributed platform. (5) The dynamic migration of intensive components of the application increase communication overhead. Transferring data over the cellular network or Wi-Fi is a power starving mechanism and exceeds the energy consumed by data processing on SMD. (6) The management of the runtime distributed platform requires continuous synchronization between SMD and the remote server node. The implementation of uninterrupted synchronization mechanism in the wireless network medium is resources starving and time consuming mechanism. In [38], we propose a distributed application model for intensive mobile application. The framework reduces the instances of runtime application partitioning by implementing the traditional client/server architecture for the design time classification of application processing load. It focuses on the configuration of explicitly configured server in the cloud data center, which is accessed on demand basis for mobile devices.

6 Conclusion and future work

The paper investigates the overhead of runtime profiling and partitioning in simulation and a real mobile computing environment. The mechanism of runtime partitioning and offloading of the intensive components of the mobile applications is a resource intensive mechanism. Empirical analysis indicates that the additional resources utilization on SMD depends on two factors; (1) the instances of runtime

application partitioning and offloading, and (2) the computational intensity of the application partitions. We conclude the runtime application profiling and partitioning of the traditional elastic mobile applications results in the additional overhead for the deployment and management of distributed platform. The mechanism of dynamic application offloading at runtime involves complications in arbitration with cloud servers for the selection of appropriate remote node, dynamic assessment of the resources utilization and availability of resources on SMDs, runtime profiling and solving, application migration and reintegration, and continuous synchronization with cloud servers for the entire duration of the distributed platform, since the traditional offloading frameworks lack in the consideration of the intensity of runtime distributed deployment on SMD.

The intrinsic limitations associated with the mobile devices demand for optimal, and lightweight procedures in cloud based application processing. Lightweight procedures will result in minimum computing resources utilization (CPU, RAM), energy consumption, and reduce application processing time. We aim for the incorporation of client/server model with the elastic attributes of the traditional offloading techniques for reducing the instances of runtime application partitioning on SMD. Computational clouds provide the SaaS model for the employment of preconfigured services on the cloud server node. The proposed framework is aimed to employ simple developmental and lightweight deployment procedures for utilizing the application processing services of computational clouds on a demand basis.

Acknowledgements This work is part of the Mobile Cloud Computing research project at the Mobile Cloud Computing Research Lab at the Department of Computer Systems and Technology, Faculty of Computer Science and Information Technology, University of Malaya, Malaysia. The project is funded by the Malaysian Ministry of Higher Education under the University of Malaya High Impact Research Grant with reference UM.C/HIR/MOHE/FCSIT/03.

References

1. Abolfazli S, Sanaei Z, Gani A (2012) Mobile cloud computing: a review on smartphone augmentation approaches. In: Proceedings of the 1st international conference on computing, information systems and communications, Singapore
2. Abolfazli S, Sanaei Z, Gani A, Buyya R (2013) Cloud-based augmentation for mobile devices: motivation, taxonomies, and open issues. IEEE Commun Surv Tutor. doi:10.1109/SURV.2013.070813.00285
3. Amazon S3 (2011) http://status.aws.amazon.com/s3-20080720.html. Accessed on 20th July 2011
4. Apple—iCloud (2013) www.apple.com/icloud/. Accessed on 1st January 2013
5. Armbrust M, Fox A, Grifth A, Joseph DA, Katz HR, Konwinski A, Lee G, Patterson AD (2009) In: Rabkin A, Stoica A, Zaharia M (eds) Above the clouds: a Berkeley view of cloud computing. Electrical Engineering and Computer Sciences University of California at Berkeley, Berkeley
6. Bahl P, Han YR, Li EL, Satyanarayanan M (2012) Advancing the state of mobile cloud computing. In: MCS'12, Low Wood Bay, Lake District, UK, 25 June 2012
7. Begum Y, Mohamed M (2010) A DHT-based process migration policy for mobile clusters. In: 7th international conference on information technology, Las Vegas, pp 934–938
8. Buyya R, Yeo CS, Venugopal S, Broberg J, Brandic I (2009) Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. Future Gener Comput Syst 25(6):599–616
9. Calheiros RN, Ranjan R, Beloglazov A, De Rose CAF, Buyya R (2011) CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw Pract Exp 41(1):23–50

10. Chun BG, Maniatis P (2009) Augmented smartphone applications through clone cloud execution. Intel Research Berkeley, Berkeley
11. Chun BG, Ihm S, Maniatis P, Naik M, Patti A (2011) CloneCloud: elastic execution between mobile device and cloud. In: EuroSys'11, Salzburg, Austria, 10–13 April 2011. ACM, New York
12. Cloud computing (2011) http://en.wikipedia.org/wiki/Cloudcomputing. Accessed on 16th June 2011
13. Cuervo E, Balasubramanian A, Cho KD, Wolman A, Saroiu S, Chandra R, Bahlx P (2010) MAUI: making smartphones last longer with code offload. In: MobiSys'10, San Francisco, California, USA, 15–18 June 2010
14. Dou KV, Gunopulos D, Mielikainen T, Tuulos HV (2010) Misco: a MapReduce framework for mobile systems. In: PETRA'10, Samos, Greece, 23–25 June 2010, ACM, New York
15. Giurgiu RO, Juric D, Krivulev I, Alonso G (2009) Calling the cloud: enabling mobile phones as interfaces to cloud applications. In: Middleware'09 proceedings of the ACM/IFIP/USENIX 10th international conference on middleware. Springer, Berlin, pp 83–102
16. Google App Engine (2012) https://developers.google.com/appengine/docs/whatisgoogleappengine. Accessed on 30 September 2012
17. Goyal S, Carter J (2004) A lightweight secure cyber foraging infrastructure for resource-constrained devices. In: WMCSA 2004 sixth IEEE workshop, 2–3 Dec 2004. IEEE, New York
18. Hoang TD, Chonho L, Dusit N, Ping WA (2013) Survey of mobile cloud computing: architecture, applications, and approaches. Wirel Commun Mob Comput. doi:10.1002/wcm.1203
19. https://docs.google.com/. Accessed on 30 September 2012
20. Hung HS, Shih SC, Shieh PJ, Lee PC, Huang HY (2012) Executing mobile applications on the cloud: framework and issues. Comput Math Appl 63(2):573–587
21. Introducing Amazon Silk (2013) http://amazonsilk.wordpress.com/2011/09/28/introducing-amazon-silk/. Accessed on 1st January 2013
22. Iyer R, Srinivasan S, Tickoo O, Fang Z, Illikkal R, Zhang S, Chadha V, Stillwell MP, Lee E (2011) CogniServe: heterogeneous server architecture for large-scale recognition. IEEE MICRO 31(3):20–31
23. Khan NA, Kiah MLM, Khan SU, Madani SA (2013) Towards secure mobile cloud computing: a survey. Future Gener Comput Syst 29(5):1278–1299
24. Khan NA, Mat Kiah ML, Madani SA, Khan A, Ali M (2013) Enhanced dynamic credential generation scheme for protection of user identity in mobile-cloud computing. J Supercomput doi:10.1007/s11227-013-0967-y
25. Kumar K, Lu HY (2010) Cloud computing for mobile users: can offloading computation save energy? Computer 43(4):51–56
26. Kumar K, Lu HY (2010) Cloud computing for mobile users: can offloading computation save energy? Computer 43(4):51–56
27. Li C, Li L (2010) Energy constrained resource allocation optimization for mobile grids. J Parallel Distrib Comput 70(3):245–258
28. Liu J, Kumar K, Lu HY (2010) Tradeoff between energy savings and privacy protection in computation offloading, ISLPED'10, Austin, Texas, USA, 18–20 August 2010. ACM, New York
29. Messer I, Greenberg P, Bernadat D, Milojicic D, Chen T, Giuli J, Gu X (2002) Towards a distributed platform for resource-constrained devices Hewlett–Packard Company
30. Oh LS, Lee E (2006) An adaptive mobile system using mobile grid computing in wireless network. In: International conference on computational science and its applications (ICCSA 2006), Glasgow, UK, pp 49–57
31. PowerTutor (2012) http://ziyang.eecs.umich.edu/projects/powertutor/. Accessed on 15th April 2012
32. Sanaei Z, Abolfazli S, Gani A, Buyya R (2013) Heterogeneity in mobile cloud computing: taxonomy and open challenges. IEEE Commun Surv Tutor. doi:10.1109/SURV.2013.050113.00090
33. Satyanarayanan M (1996) Fundamental challenges in mobile computing. In: Proceedings of the 5th annual ACM symposium on principles of distributed computing, pp 1–7, May 1996
34. Satyanarayanan M (2010) Mobile computing: the next decade. In: Proceedings of the 1st ACM workshop on mobile cloud computing & services: social networks and beyond (MCS)
35. 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
36. Sharifi M, Kafaie S, Kashefi O (2012) A survey and taxonomy of cyber foraging of mobile devices. IEEE Commun Surv Tutor 14(4):1–12
37. Shiraz M, Gani A (2012) Mobile cloud computing: critical analysis of application deployment in virtual machines. In: ICICN 2012 IPCSIT, vol XX, 25–28 Feb 2012. IACSIT Press, Singapore

38. Shiraz M, Gani A, Khokhar RH (2012) Towards lightweight distributed applications in mobile cloud computing. In: Proceedings of 2012 IEEE international conference on computer science and automation engineering (CSAE 2012), China, 25–27 May 2012
39. Shiraz M, Gani A, Rashid HK, Buyya R (2012) A review on distributed application processing frameworks in smart mobile devices for mobile cloud computing. IEEE Commun Surv Tutor. doi:10.1109/SURV.2012.111412.00045
40. Shiraz M, Gani A, Rashid HK, Ejaz A (2012) An extendable simulation framework for modeling application processing potentials of smart mobile devices for mobile cloud computing. In: Frontiers of information technology 2012, Islamabad, Pakistan, 19–21 December 2012, accepted
41. Shiraz M, Abolfazli S, Sanaei Z, Gani A (2013) A study on virtual machine deployment for application outsourcing in mobile cloud computing. J Supercomput 63(3):946–964
42. What is AWS (2012) http://aws.amazon.com/. Accessed on 15th September 2012
43. Zhang X, Kunjithapatham A, Jeong S, Gibbs S (2011) Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing. Mob Netw Appl 16(3):270–285