J Supercomput (2014) 68:978–995
DOI 10.1007/s11227-013-1076-7

A lightweight active service migration framework for computational offloading in mobile cloud computing

Muhammad Shiraz · Abdullah Gani

Published online: 14 January 2014

Abstract Cloud computing enables access to the widespread services and resources in cloud datacenters for mitigating resource limitations in low-potential client devices. Computational cloud is an attractive platform for computational offloading due to the attributes of scalability and availability of resources. Therefore, mobile cloud computing (MCC) leverages the application processing services of computational clouds for enabling computational-intensive and ubiquitous mobile applications on smart mobile devices (SMDs). Computational offloading frameworks focus on offloading intensive mobile applications at different granularity levels which involve resource-intensive mechanism of application profiling and partitioning at runtime. As a result, the energy consumption cost (ECC) and turnaround time of the application is increased. This paper proposes an active service migration (ASM) framework for computational offloading to cloud datacenters, which employs lightweight procedure for the deployment of runtime distributed platform. The proposed framework employs coarse granularity level and simple developmental and deployment procedures for computational offloading in MCC. ASM is evaluated by benchmarking prototype application on the Android devices in the real MCC environment. It is found that the turnaround time of the application reduces up to 45 % and ECC of the application reduces up to 33 % in ASM-based computational offloading as compared to traditional offloading techniques which shows the lightweight nature of the proposed framework for computational offloading.

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

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

Keywords Mobile cloud computing · Mobile application · Distributed system · Application offloading · Lightweight

1 Introduction

The latest developments in mobile computing technology have changed user preferences for computing. Smartphones have replaced the traditional computing and communication devices as an all-in-one device [1]. It is predicated that by 2015 more than 240 million business customers will be leveraging cloud computing services through mobile devices which will drive revenues of $5.2 billion [2]. Therefore, smart mobile devices (SMDs) are the future computing devices with high user expectations for accessing computational-intensive applications. For example, natural language translators [3,4], speech recognizers [5], optical character recognizers, image processors [6,7], online computational-intensive games, video processing [8] and wearable devices for handicapped people [9]. Such applications require high computing power, memory, and battery power on resource-constrained SMDs [10]. However, SMDs are still low-potential computing devices having limitations in storage capacity, CPU potential and battery lifetime. Therefore, mobile cloud computing (MCC) provides software-level solutions for mitigating resource constraints in SMDs [11].

Cloud computing implements different service provision models for the provision of cloud resources and services to SMDs. For example, a number of online file storage services are available on cloud server for providing off-device storage services; such as, Amazon S3 [12], Google Docs [13], MobileMe [14], and DropBox [15]. Computational clouds augment the computing potentials of client devices including PDAs and smartphones. MCC employs the services and resources of cloud datacenters for enabling off-device storage services [12–15] and accessing the application processing services of cloud server nodes [16,17]. MCC employs a number of augmentation procedures for SMDs, such as screen augmentation, energy augmentation, storage augmentation and application processing augmentation of SMD [11]. From one perspective, the processing potential of SMDs is increasing rapidly. For instance, Galaxy S3 (Samsung Exynos 4412, four cores, ARMv7 Cortex A9) and Galaxy S4 (two models: Samsung Exynos 5 Octa 5410 8 cores, ARMv7 Cortex A15; Quad-core Qualcomm Snapdragon 600, CPU Krait 300). However, powerful processing hardware is energy starving. Therefore, MCC employs computational offloading for enabling complicated and ubiquitous mobile applications. Examples of the recent applications which employ computational offloading include Apple's iCloud and Amazon's Silk. Apple's iCloud [16] provides on-demand access automatically to applications, such as music, photos, apps, calendars, documents. Amazon EC2 and Microsoft Azure host the application store of Apple's iCloud. Similarly, Silk application [17] is a cloud-accelerated "split browser" which resides on both Kindle Fire and EC2. For each web page request, Silk dynamically determines the distribution of computational load between the local SMD and remote Amazon EC2. Silk considers the objective functions of network conditions, page complexity and the location of any cached content. The traditional frameworks for computational offloading [18–20] employ computational offloading at different granularity levels which require additional support from application devel-

opers at design time. Application developers classify the components of the mobile applications as local or remote. Similarly, the loosely coupled and intensive components of the mobile application are tagged as remote, whereas the tightly coupled or slightly intensive components are labeled as local. Such models implement application profiling mechanism to evaluate the feasibility of outsourcing the remotely tagged components of mobile application [21–23]. Therefore, the deployment of distributed application processing platform is time consuming and resource intensive.

We propose a lightweight ASM framework for the distributed processing of intensive mobile application in MCC. ASM framework requires a simple procedure for application development and employs a lightweight mechanism for computational offloading. The model is implemented for outsourcing the running instance(s) of mobile application at service-level granularity to remote server nodes. We evaluate the proposed framework by benchmarking prototype application in the real MCC environment. The significance of proposed framework is validated by comparing the performance of the proposed framework with traditional computational offloading frameworks which employ finer granularity for computational offloading [18–20,24]. Analysis of the results indicates that computational offloading reduces resource utilization (CPU, RAM, battery) on mobile device. Furthermore, by employing ASM framework for computational offloading the energy consumption cost and turnaround time of the application is reduced considerably. For instance, the turnaround time of the application reduces up to 45 % and energy consumption cost (ECC) of the application reduces up to 33 % in ASM-based computational offloading as compared to traditional offloading techniques which shows the lightweight nature of the proposed framework for computational offloading. The paper is structured as follows.

Section 2 discusses the state-of-the-art for computational offloading in MCC. Section 3 explains the architecture, working and features of the proposed framework. Section 4 explains the methodology used for the evaluation of proposed framework. Section 5 discusses results and empirical findings. Finally, Sect. 6 concludes the paper and proposes future directives.

2 Related work

Computational offloading is endeavored as software-level solution for mitigating low computing potentials of SMDs [25]. In the recent years, a number of cloud-based computational offloading frameworks are proposed which are implemented by outsourcing mobile application at different granularity levels [18–20,24]. The following section reviews recent approaches for computational offloading. CloudClone [26] seamlessly offloads the complete image of the running application to the nearby computer. The framework exploits unique augmentations strategies for different types of applications. VM-based Cloudlets architecture [27] uses the procedure of copying the entire processing environment of the mobile device into remote cloudlet. A cloudlet is a trustable remote computer which provides the services of outsourced processing of application to SMD. A VM-based CloneCloud approach [19] extends the concept of CloudClone [26] to computational clouds. The framework implements thread-level granularity for the partitioning of the mobile application at runtime. The framework

is based on copying the complete application processing environment of SMD to the cloud server node. Mirror server [28] augments smartphones by configuring mirror server in the telecommunication server provider platform. The framework augments smartphones in three distinct ways: security (file scanning), storage (file caching) and computation offloading. Mirror server maintains VM template for each of the different types of mobile devices platform. The VM template for each mobile device is kept with default configurations and a new VM instance is created for offloaded component of the mobile application. Offloading the entire image of the running application involves the overhead of VM cloning, VM instance migration and VM configuration on the remote server node [22].

A cloud-based framework [29] employs application-level process migration for live component migration. Cloud server creates fresh VM instance on demand, and the delegated application resume processing on the newly created VM instance on cloud server node. In [30], a middleware framework is proposed for sharing the application processing dynamically between cloud server and mobile client. The framework implements both static partitioning and dynamic partitioning strategies. MAUI [18] and Think Air [24] exploit dynamic application profiling and partition approach for partition offloading. The framework follows method state migration approach as an alternative of method code migration. Elastic application model [26] is a middleware framework for elastic mobile applications. The framework implements dynamic distributed processing platform at application layer. Recent frameworks for computing offloading [18,20,24] employ method-level granularity for computational offloading. Such frameworks require to explicitly annotate each method of the application either locally or remotely. Locally annotated methods are executed on the mobile device locally, whereas remotely tagged methods are considered for remote processing on the cloud server node. The operating procedure for such frameworks involves application profiling for determining the feasibility of computational task to the cloud server nodes.

Computational offloading involves blocking and non-blocking scenarios of the mobile application during cloud-based processing of intensive partitions of the application. In the blocking scenario, the execution of mobile application is halted until the successful execution of the offloaded task to cloud server node. Once the result of remote execution is returned, the suspended application on local mobile device is resumed and the running states of the mobile application are synchronized. For instance, CloneCloud [26] follows blocking mechanism for implementing thread-level granularity for computational offloading, MAUI [18] and Think Air [24] implements method-level granularity by using blocking mechanism. Similarly, Odessa [31] also implements blocking mechanism for cloud-based processing of intensive tasks of the mobile application. In the non-blocking scenarios, the application on mobile device remains in the running state and the offloaded tasks of the application are executed in parallel. For instance, Orleans [32] and RESTful frameworks employ non-blocking scenario for computational offloading to cloud server node. Therefore, a rigorous synchronization mechanism is required for maintaining the consistency of mobile application in cloud-based application processing [33].

The drawbacks of current computational offloading frameworks [18–20,24,31,32] include refined granularity level for application partitioning, dynamic runtime appli-

cation profiling and partitioning, developers support for annotating the components of mobile application as local and remote [23]. Refined granularity-level application partitioning and computational offloading involves the overhead of maintaining consistency of data states between mobile application running on mobile device and remote server node. The low-level granularity includes object [34], class [35–37], thread [26], method [18] and task [38]. The application profiling mechanism uses the probability measures and predicative techniques for resource allocation to the components of mobile application which involves the factor of inaccuracy [18,24]. The manual annotation of the components of mobile application as local and remote is time consuming. Further, the mechanism of runtime application profiling and partitioning is time consuming, resource intensive and energy starving [21].

3 Proposed framework

We propose a lightweight ASM framework for offloading running instances of the intensive components of mobile application to cloud server node. Refined granularity-level application partitioning and computational offloading [18–20,24,26,34–38] involves the overhead of profiling at finer level granularity, resource-intensive monitoring mechanism for the management of distributed application execution platform and energy starving synchronization mechanism for sustaining consistency of the distributed application execution platform. As a result, the energy consumption cost and turnaround time of the application is increased. Furthermore, partitioning and offloading decision is made on the basis of the resource utilization history of the remotely annotated component of the mobile application [18] and prediction of future demands for computational resources on mobile device [24]. ASM framework reduces the additional cost of runtime computational offloading by employing coarse-level granularity which reduces the overhead of dynamic application profiling and partitioning. Coarse-level granularity is achieved by employing service-level granularity for offloading the component of mobile application. The running service component of the mobile application is offloaded which involves minimum resource utilization in the establishment and management of distributed application processing platform. ASM framework follows two distinct operating procedures for the execution of mobile application. In the primary operating procedure, mobile application is capable of executing on mobile device; whereas, in the secondary operating procedure, mobile application is enabled to switch to the online mode, wherein the intensive service components of the mobile application are offloaded to the cloud server node. Mobile application is enabled to dynamically switch between online and offline mode of the application all through the execution period of the mobile application. Figure 1 shows the architecture of the ASM framework.

ASM framework is composed of configuration manager, preferences manager and synchronizer components. The following section discusses the components of ASM framework.

Configuration manager Configuration manager is responsible for the configuration and operation of the mobile application in two distinct operating modes: offline mode and online mode. In the offline mode, the components of the mobile application are

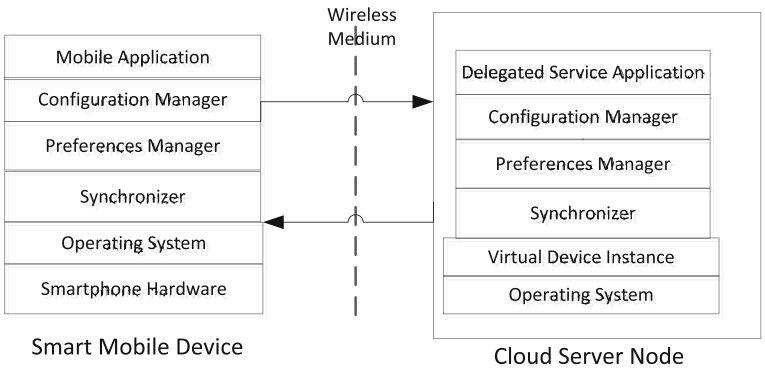

Fig. 1 Architecture of active service migration framework

executed on SMD. The options of offloading application services are enabled in the online mode, wherein the components of mobile application can be offloaded to the remote server node. The configuration manager activates the preferences manager component to save the data states of the running mobile application. Configuration manager on the cloud server node configures the delegated service application on the remote server node. It resumes the running state of the delegated service component of the application by accessing the preferences files from the persistent storage. The configuration manager arbitrates with the remote server node for offloading the selected running service of the mobile application.

Preferences manager Mobile applications are associated with preferences manager which provides access to the data files of the application (preferences file). Mobile application uses the preferences file to write and read the data states during the activation and de-activation of mobile application. In active service offloading process, the preferences manager component is activated to save the data states of the running service. Preferences manager saves the data states to the persistent medium. The role of preferences manager is to provide access to the preferences of the mobile application. The preferences manager components copies the preferences file to the external storage device which is directly accessible for the synchronizer component. The preferences manager component of the server node is responsible for providing access to the data files which are downloaded with the delegated service application. Similarly, whenever the service application completes execution on the remote server node, preferences manager associated the virtual device instance on the cloud server node saves the final results to the preferences file. Synchronization manager component accesses the preferences for the exchange of data files between SMD and remote server node.

Synchronizer The synchronizer component of the framework is responsible for the synchronization of transmission between SMD and remote server node. Whenever the states of the application are saved on the persisted medium, the synchronizer component is activated to offload the service application to remote server node. The configuration manager component searches for the configuration file of the service application on mobile device. Whenever, the configuration of the service application is validated, the synchronizer component is activated to outsource the configuration

files to remote server node. Synchronizer component of the remote server node is activated to receive the delegated service application. Whenever, the configuration file of the delegated service application is received successfully on the remote server node, the configuration manager component of the server node is activated to configure the delegated service application and resume the running states from the preferences file.

The synchronizer component arbitrates with the cloud datacenter for the selection of remote server node. A fresh virtual device instance is created on the server node for the execution of delegated service application. At that instance, the configuration manager on the mobile device saves the running states of the service application by activating the preferences manager and terminates the selected service to release the systems resource occupied by the selected intensive service. The synchronizer component offloads the service application to remote service node. The synchronizer component is also responsible for the uploading and downloading of preferences files between SMD and remote server node. The communication between application running on the SMD and remote server takes place in the form of XML file. The preferences manager component of the framework is responsible for the migration of XML file. We develop our own protocol for the migration of preferences file. However, preferences manger component of ASM conceals such details and provides a transparent environment for the distributed processing of mobile application in MCC.

The configuration manager component on the cloud server node configures the delegated service application and resumes the running states of the service application in the guest virtual device instance created on the server node. The synchronizer components on both the SMD and cloud server node communicate for the exchange of configuration and data files. On successful execution of the service application components of the mobile application on SMD, results of the mobile application are saved in the preferences file and returned to the SMD. Figure 2 shows the flowchart of the operation modes of ASM.

A challenging aspect of MCC is the diversity in operating system platforms and application frameworks [39]. For instance, Android, iOS, Windows Mobile and Symbian. Therefore, homogenous solution for the heterogeneous operating system platforms is a challenging perspective for the application in MCC [40]. Android operating system uses services oriented architecture (SOA) for applications. Therefore, ASM framework employs SOA of the mobile application and is compatible with all mobile operating system platforms which use SOA for mobile applications. Each Android application is associated with five types of files on SMD: application package, configuration files, database files, cache buffers and preferences file. The application package file is installed in \data\app folder with the name applicationpackagename.apk, whereas the data preferences files are created in \data\data\packagename\sharedpreferences folder with the default name SharedPreferencesfile.xml. In the same way, cache files are stored in \data\data\packagename\cache folder. It is possible that a mobile application does not require the configuration or data file. For example, Youtube application does not require configuration and data files, Gmail application requires only the application package file to be installed on the SMD, Android MMS application requires application package and configuration files, Contact Provider application requires application package and database files only [32]. Application preferences file is not included in

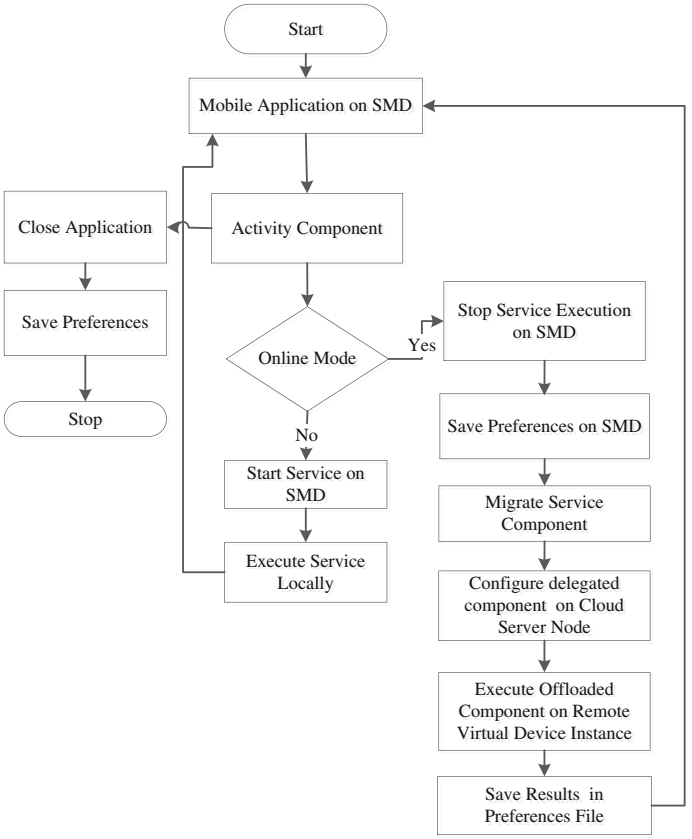

Fig. 2 Flow chart for the operation modes of ASM operation

the package file (.apk). Therefore, a newly installed application always lacks in preferences file. The preferences manager component of ASM is responsible for uploading and downloading preferences files by activating the synchronizer component (on both SMD and the remote guest environment). The important aspect of ASM is that the service application package (.apk file) is transferred only once to the remote server node. However, configuration and data files require repeated transmission for each instance of remote execution of the service application. It means that at the first instance the entire service application package file and the other related files are transferred to cloud server node. However, if the same service is required to be executed again on the cloud server node, it does not require the application package to be migrated repeatedly. Instead, later instances of remote service execution require to upload the configuration and data files in order to synchronize the execution of service application on the remote server node.

The explicit configuration of services on the cloud server node and invoking such services by passing parameters is a simple approach; however, relying on the preconfigured services of the cloud server nodes lead to the problem of dependency on the

centralized services and reduces offline usability and adversely affects rich user experiences [41,42]. Similarly, it leads to the employment of thin client applications like traditional web and email applications, wherein the processing logic of the application is hosted on the remote server nodes and client applications provide user interface. Furthermore, such approach is not compatible with the design of existing mobile applications. The architecture of existing applications needs to be changed in order to adopt the client/server architecture. ASM framework employs offloading the selected intensive component at runtime and proposes two distinct operating modes (offline mode and online mode) for the execution of mobile application. Therefore, the mechanism of runtime component offloading requires outsourcing the binary code of the component being offloaded which is configured on the remote server node. ASM employs simple developmental procedure. Unlike the traditional elastic application offloading models [18–20,24], ASM does not restrict application developers to classify and annotate the application components as local or remote at finer granularity level. ASM involves entire service-level granularity for the application offloading, which reduces the overhead associated with finer level granularity nature of traditional application offloading frameworks. It eradicates the overhead of runtime application profiling and solving [21]. The framework focuses on the user preferences for offloading the intensive components of the mobile application at runtime. Mobile users are provided full control over the execution mode of the mobile application. In the offline mode all the components of mobile application are executed locally on SMD, whereas in the online mode the intensive components of mobile application are offloaded dynamically at runtime. The dual operation modes of ASM provide robustness to the mobile applications. The applications are capable of operating with full functionalities in the situations of unavailability of remote services. ASM masks the complexities of computational offloading from mobile users by providing the notion as entire components of the mobile application are executed locally on SMD.

4 Methodology

We evaluate the proposed framework by benchmarking prototype application for Android devices in the real mobile cloud computing environment. The experimental setup is composed of server node which runs instances of the Android virtual device (AVD), Wi-Fi wireless network and Samsung Galaxy SII mobile device. The Android AVD is employed on the server machine for the execution of offloaded service component of the application at runtime. Mobile devices access the wireless network via Wi-Fi wireless network connection of radio type 802.11g, with the available physical layer data rates of 54 Mbps. Java-based Android software development toolkit (Android SDK) is deployed for the development of the prototype application. Monitoring tools such as Android debug bridge and Dalvik debug monitor system are used for the measurement of resource utilization (CPU and RAM), whereas Power Tutor tool [43] is used for the measurement of battery power consumption on SMD in distributed application processing.

The prototype application is composed of three computational-intensive service components and a single activity component. The service components implement

the computational logic of the application, whereas the activity component provides graphical user interface (GUI) for interacting with the mobile application. The computational logic of the application includes three service components. Each service component of the application is evaluated with 30 different computational intensities. (a) The sorting service component implements the logic of bubble sort for sorting liner list of integer type values. The sorting logic of the application is evaluated for varying computational intensities (11,000–40,000) of the sorting operation. (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 evaluated with 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 bˆe, whereas b is the base and e is the exponent. The power compute logic of the application is evaluated for varying power computing intensities 2ˆ1,000,000–2ˆ200,000,000.

Measurement parameters for the evaluation of ASM framework include turnaround time (TT) in milliseconds (ms) and ECC in Joules (J) of the mobile application, RAM (MB) allocation and % CPU utilization. Data are collected by evaluating the prototype application in different scenarios. The significance of ASM framework is determined by evaluating application execution on local mobile device and cloud server node. The measurement parameters in this scenario include CPU utilization and RAM allocation on SMD. The lightweight nature of ASM framework is validated by employing computational offloading in two different scenarios. In the first scenario, ASM framework is implemented by offloading running instances of the service components of the application. In the second scenario, the intensive components of the mobile application are offloaded to the cloud server node by implementing the traditional computational offloading technique which uses runtime application profiling [18–20,24]. Experimentation is repeated 30 times for either computational intensity to signify the interval estimate of each experiment with 99 % confidence interval.

5 Results and discussion

The turnaround time of the application is an important parameter in remote processing of intensive mobile applications. It shows the total execution time of the mobile application in the distributed MCC environment. Distributed processing of mobile application involves delays at different stages of computational offloading. The turnaround time of the application includes: (1) the time taken in saving the data states of the running instance of the component of the mobile application which is being offloaded; (2) time taken in transferring application binary code to the remote server; (3) time taken in downloading the delegated application binary code to the remote virtual machine on the cloud server node; (4) time taken in uploading the preferences (data states file) of the mobile application to remote server node; (5) time required for resuming the running state of the mobile application on the remote server node; (6) time taken in processing the application on remote machine; and (7) time taken in returning result file to the mobile device. It is observed that the TT of the components offloaded at runtime varies on the basis of two factors. (1) The processing time of the

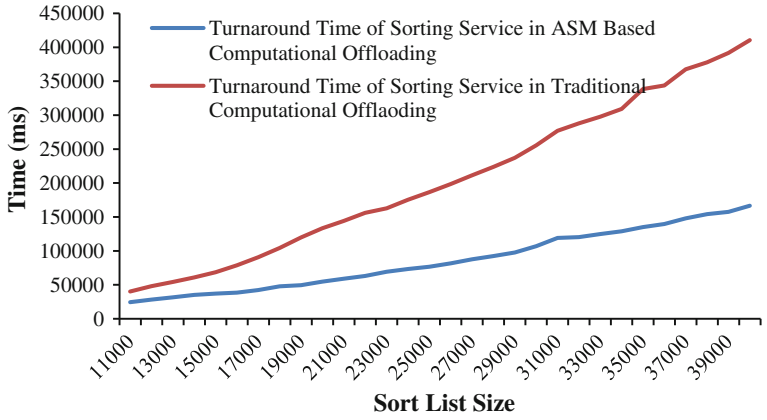

Fig. 3 Comparison of turnaround time of sorting service execution in ASM and traditional computational offloading

offloaded component that depends on the computational length of the offloaded component. (2) The data transmission time between the local and remote machine, which depends on the size of data transmission between local mobile device and remote machine. Hence, the TT value is the total time taken in offloaded processing of the component of mobile application, which is the sum of the application processing time on the remote virtual device and timing cost of runtime component offloading.

The TT in offloaded processing of the sorting service, matrix multiplication service and power compute service component of the mobile application is evaluated with 30 different computational intensities. The TT of the offloaded component increases with the increases in the computational intensity of the offloaded component. For instance, in ASM-based computational offloading the TT of sorting service component is found 24,331 ms for list size 11,000 values and 166,457 ms for list size 40,000 values. It shows that the TT is 85.4 % higher for sorting list of 40,000 values as compared to sorting list of 11,000 values. Whereas, in traditional computational offloading which uses profiling mechanism, the TT of the sorting service is examined 40,211 ms for list size 11,000 values and 410,398 ms for list size 40,000 values which shows 90 % increase in TT for list of 40,000 values as compared to the sorting list of 11,000 values. Figure 3 shows the comparison of TT of the sorting service in two different scenarios. The comparison of TT for sorting operation in ASM-based computational offloading and traditional computational offloading technique shows reduction in TT of sorting service in ASM-based offloading. For instance the TT for sorting operation reduces 39 % for list size 11,000, 59 % for list size 20,000, 58 % for list size 30,000 and 60 % for list size 40,000. The overall reduction in TT of sorting service execution is 55.3 % in employing ASM framework-based computational offloading.

In ASM-based computational offloading the TT of matrix multiplication service component is found 16,431 ms for matrices size 160 $\times$ 160 and 262,697 ms for matrix size 40,000 values. It shows that the TT is 93.4 % higher for multiplying matrices of 450 $\times$ 450 values as compared to matrices of 160 $\times$ 160 values. Whereas, in traditional computational offloading which uses profiling mechanism, the TT of

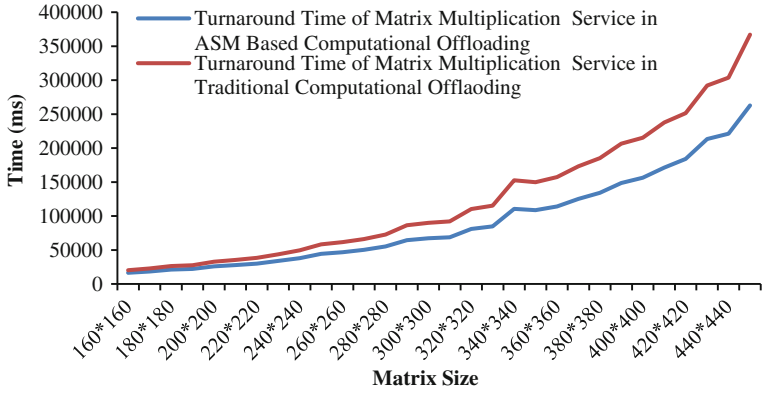

Fig. 4 Comparison of turnaround time of matrix multiplication service execution in ASM and traditional computational offloading

matrix multiplication service is examined 20,326 ms for matrices size 160 × 160 and 367,077 ms for matrices size 450 × 450 which shows 90 % increase in multiplying matrices of 450 × 450 size as compared to multiplying matrices 160 × 160 size. Figure 4 shows the comparison of TT of the matrix multiplication service in traditional and ASM-based computational offloading. It is examined that TT for matrix multiplication operation in ASM-based computational offloading decreases as compared to traditional computational offloading technique. For instance, the TT of matrix multiplication operation decreases 19 % for matrix size 160 × 160, 24 % for matrix size 260 × 260, 27 % for 360 × 360 matrix size and 28 % for matrix size 450 × 450. The overall reduction in TT of matrix multiplication service execution is 24.7 % in employing ASM framework-based computational offloading.

In ASM-based computational offloading the TT of power compute service component is found 7,175 ms for computing 2ˆ1,000,000 and 265,724 ms for computing 2ˆ2,000,000,000. It shows that the TT is 93.4 % higher for computing 2ˆ2,000,000,000 as compared to computing 2ˆ1,000,000. Whereas, in traditional computational offloading the TT of matrix multiplication service is examined 7,284 ms for computing 2ˆ1,000,000 and 622,062 ms for computing 2ˆ2,000,000,000 which shows 98.8 % increase in TT for computing 2ˆ2,000,000,000 as compared to 2ˆ1,000,000. Figure 5 shows the comparison of TT of the power compute service in traditional and ASM-based computational offloading. It is examined that TT for power computing operation in ASM-based computational offloading decreases as compared to traditional computational offloading technique. For instance, the TT of power compute operation decreases 15 % for computing 2ˆ1,000,000 and 91 % for computing 2ˆ9,000,000.

The energy consumption cost (ECC) includes energy consumed in runtime component migration, energy consumed in saving the data states of running instance of the mobile application, energy consumed in uploading the data file to remote server node and energy consumed in returning the resultant data files to local mobile device. ECC is evaluated with 30 different computational intensities in offloaded processing of the sorting service, matrix multiplication service and power compute service component

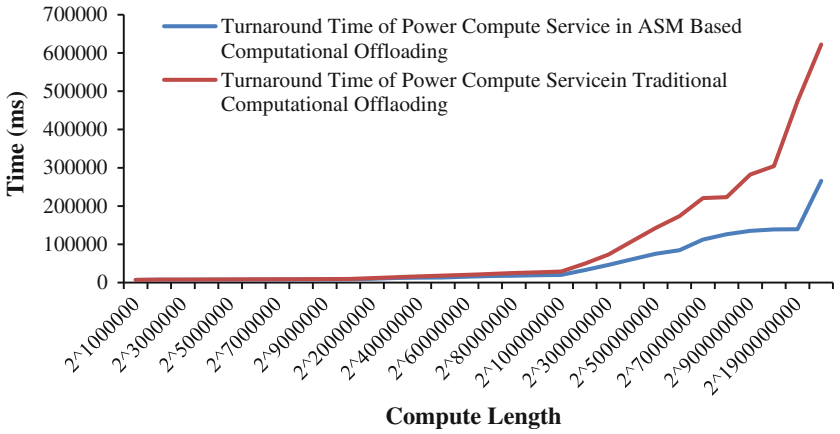

Fig. 5 Comparison of turnaround time of power compute service execution in ASM and traditional computational offloading

of the mobile application. The ECC of the offloaded component increases with the increases in the computational intensity of the offloaded component. In ASM-based computational offloading the ECC of sorting service component is found 49.8 J for list size 11,000 values and 201.4 J for list size 40,000 values. It shows that the ECC is 75.3 % higher for sorting list of 40,000 values as compared to sorting list of 11,000 values.

In traditional computational offloading, the ECC of the sorting service is examined 50.3 J for list size 11,000 values and 386.4 J for list size 40,000 values, which shows 87 % increase in the ECC for list of 40,000 values as compared to the sorting list of 11,000 values. Figure 6 shows the comparison of ECC of the sorting service ASM-based and traditional computational offloading which shows reduction in the ECC of sorting service in ASM-based offloading. For instance, the ECC for sorting operating reduces 1 % for list size 11,000, 34.4 % for list size 20,000, 44.8 % for list size 30,000 and 47.9 % for list size 40,000. The overall reduction in ECC of sorting service execution is 36 % in employing ASM framework-based computational offloading.

The ECC of matrix multiplication service component by employing ASM-based computational offloading is found 40 J for matrices size 160 × 160 and 131.7 J for matrix size 40,000 values. It shows that the ECC is 69.6 % higher for multiplying matrices of 450 × 450 size as compared to matrices of 160 × 160 size. Whereas, in traditional computational offloading, the ECC of matrix multiplication service is examined 43.1 J for matrices of 160 × 160 size and 152 J for matrices 450 × 450 size, which shows 71.6 % increase in the ECC for higher computational intensity (450 × 450 size) as compared to lower computational intensity (160 × 160) of matrix multiplication operation. Figure 7 shows the comparison of ECC of the matrix multiplication service in traditional and ASM-based computational offloading. It shows that the ECC for matrix multiplication operation in ASM-based computational offloading decreases as compared to traditional computational offloading technique. For example, the ECC of matrix multiplication operation decreases 7.2 % for matrix size 160 × 160, 12.2 %

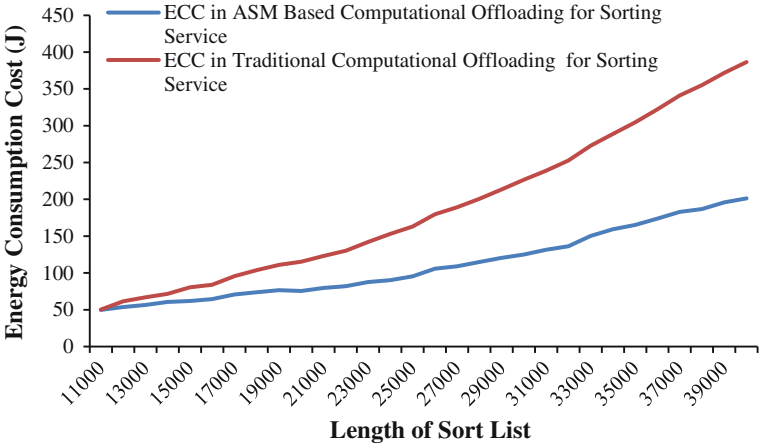

Fig. 6 Comparison of energy consumption cost of sorting service execution in ASM and traditional computational offloading

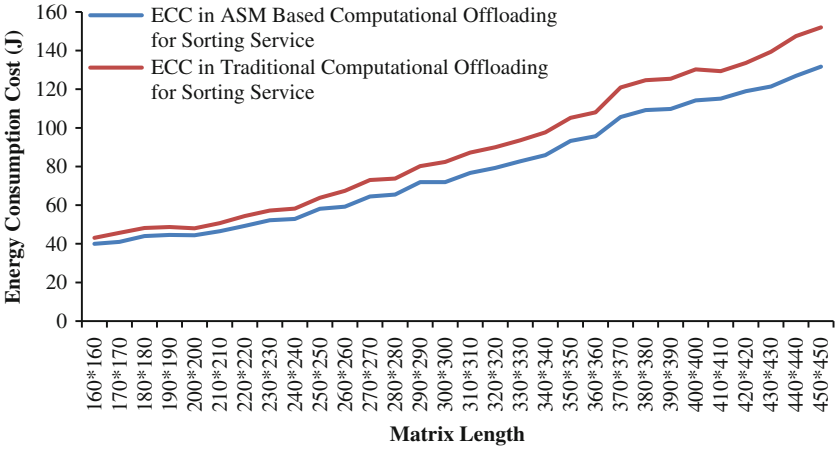

Fig. 7 Comparison of energy consumption cost of matrix multiplication service execution in ASM and traditional computational offloading

for matrix size 260 × 260, 11.4 % for 360 × 360 matrix size and 13.4 % for matrix size 450 × 450. The overall reduction in ECC of matrix multiplication service execution is 10.9 % in employing ASM framework-based computational offloading.

The ECC of power compute service in ASM-based computational offloading is found 5.4 J for computing 2^1,000,000 and 351 J for computing 2^2,000,000,000. It shows that the ECC is 98.5 % higher for computing 2^2,000,000,000 as compared to computing 2^1,000,000. Whereas, in traditional computational offloading the ECC of matrix multiplication service is examined 5.4 J for computing 2^1,000,000 and 460.7 J for computing 2^2,000,000,000 which shows 98.8 % increase in ECC for computing 2^2,000,000,000 as compared to 2^1,000,000. Figure 8 shows the comparison of ECC of the power compute service in traditional and ASM-based computational

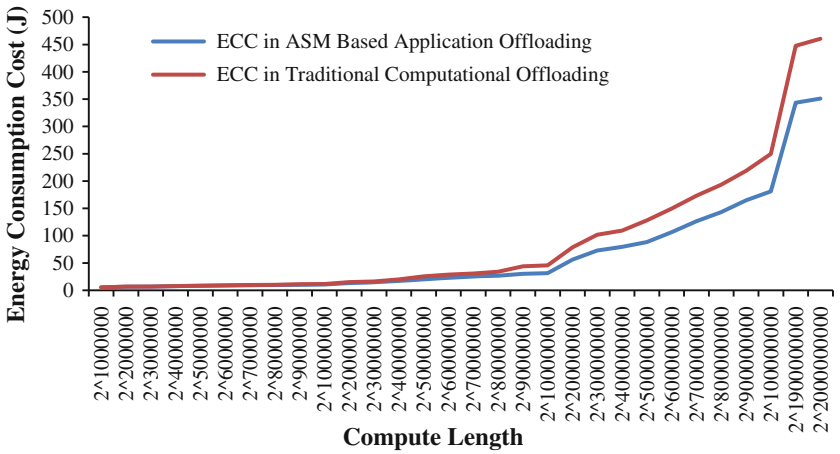

Fig. 8 Comparison of energy consumption cost of power compute service execution in ASM and traditional computational offloading

offloading. It is examined that ECC for power computing operation in ASM-based computational offloading decreases as compared to traditional computational offloading technique. For instance, the ECC of power compute operation decreases 4 % for computing 2^2,000,000, 12 % for computing 2^9,000,000 and 24 % for computing 2^2,000,000,000.

The significance of ASM framework is determined by evaluating application execution on local mobile device and cloud server node. The measurement parameter in this scenario includes CPU utilization and RAM allocation on SMD. RAM allocation on the mobile device increases by increasing the computational length of linear list for sorting operation and 2-D array for matrix multiplication operation. It is found that 10.148 MB memory is allocated for sorting list of 11,000 values, whereas 10.265 MB memory is allocated for sorting list of 40,000 values, which shows that RAM allocation increases 1.4 % for sorting list of 40,000 values as compared to sorting list of 110,000 values. Similarly, 2.78 MB memory is allocated for multiplying matrices of size 160 × 160 size and 22.8 MB memory is allocated for matrices of size 450 × 450, which shows 87.8 % increase for multiplying larger size matrices (450 × 450) as compared to multiplying small size matrices (160 × 160). The allocation of memory for power compute operation on the mobile device is found 10.11 MB for varying computational intensities (2^100,000–2^2, 000,000,000).

In the ASM framework-based computational offloading the entire logic of the intensive components of the application is executed on the remote server node. It is found that by offloading the component of mobile application to the cloud server node, RAM allocation on the local mobile device is reduced 74.5 % in offloading sorting service, 42.2 % in accessing matrix multiplication service and 98 % for power compute service component of the mobile application. Therefore, RAM allocation is saved up to 100 % by computational offloading to cloud server node. The comparison of RAM allocation in local and ASM-based application execution signifies the usefulness of offloading computational load to the cloud server node.

The execution of application on local mobile devices results in high CPU utilization for a longer period of time as compared to accessing the application processing services of cloud server node. It is examined that the average CPU utilization for executing sorting service on local mobile device is 48.67 % of the total CPU utilization on local mobile device for 17,427 ms duration. The average CPU utilization for executing matrix multiplication service on local mobile device is 45.46 % of the total CPU utilization on local mobile device for 31,190 ms duration. The average CPU utilization for executing power compute service on local mobile device is 48 % of the total CPU utilization on local mobile device. Since, in computational offloading application processing load is outsourced to remote server nodes, energy consumption cost of application processing on the local device is reduced up to 95 %. It is examined that the allocation of RAM on mobile device reduces up to 72 % and the duration of CPU utilization reduces up to 99 % by computational offloading to cloud server node. Hence, computational offloading reduces application processing load on local device which results in minimizing resource utilization on SMD (RAM, CPU) and decreases energy consumption on SMD. However, it is found that the execution cost of runtime computational offloading remains high for offloading smaller computational load to the cloud server node. It is for the reason of additional delays incurred during the configuration of distributed application processing platform at runtime. Furthermore, it is examined that for all instances of active services offloading by using ASM, the CPU utilization for operating system increases up to 3 % on the Android virtual device which shows additional load on the mobile device in component offloading. However, for the physical mobile device the increase in CPU utilization is found 0 % during service migration at runtime.

6 Conclusion and future work

ASM is a lightweight framework for offloading the intensive components of mobile application to computational clouds. The proposed framework employs simple developmental and deployment procedures for the cloud-based application processing. Analysis shows the usefulness of employing ASM framework for computational offloading in mobile cloud computing. It is found that the allocation of RAM on mobile device reduces up to 72 % and the duration of CPU utilization reduces up to 99 % by computational offloading to cloud server node. Similarly, the turnaround time of the application reduces 45 % and energy consumption cost of the application reduces up to 33 % in ASM-based computational offloading as compared to traditional offloading technique. The significance of active services migration to cloud server nodes is twofold. First, service offloading reduces the computational load on mobile device which results in saving computing resources (RAM, CPU) and energy consumption on SMD. Second, it makes the computing resources available for slightly intensive components of the application which run locally. As a result, the TT of the components of the application executed on the SMD is locally decreased.

However, migration of the active services at runtime involves the additional complications in the establishment and management of distributed platform at runtime. Analysis indicates the additional overhead of application offloading at runtime. Simi-

larly, outsourcing active states of the service application is subjected to network security threats. Therefore, it is necessary to adopt alternative procedures with minimum complexities and reduced security threats. The merger of distributed application architecture with the runtime outsourcing seems to be an appropriate alternative approach for the computational-intensive mobile applications. Our future work focuses on this specific aspect of distributed processing of intensive mobile applications in MCC.

Acknowledgments This work is part of the Mobile Cloud Computing research project at the Mobile Cloud Computing Research Laboratory 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. Shiraz M, Whaiduzzaman M, Gani A (2013) A study on anatomy of smartphone. J Comput Commun Collab 1(1):24–31
2. ABI Research (2012) http://www.abiresearch.com/. Accessed 21 August 2012
3. Flinn J, Park S, Satyanarayanan M (2002) Balancing performance energy, and quality in pervasive computing. In: 22nd international conference on distributed computing systems (ICDCS02), Austria, Vienna, pp 217–226
4. Kristensen DM (2007) Enabling cyber foraging for mobile devices. In: 5th MiNEMA workshop, Magdeburg, Germany, pp 32–36
5. Su YY, Flinn J (2005) Slingshot: deploying state-full services in wireless hotspots. In: 3rd international conference on mobile systems, applications, and services, New York, pp 79–92
6. Kristensen DM, Bouvin ON (2008) Developing cyber foraging applications for portable devices. In: 2nd IEEE international interdisciplinary conference on portable information devices, Garmisch-Partenkirchen, Germany, pp 1–6
7. Porras J, Riva O, Kristensen DM (2009) Dynamic resource management and cyber foraging, vol 16. Springer, Berlin
8. Chun B, Maniatis P (2009) Augmented smartphone applications through clone cloud execution. In: 12th workshop on hot topics in operating systems (HotOS), Monte Verita, Switzerland
9. Satyanarayanan M, Bahl P, Ceres R, Davies N (2009) The case for VM-based cloudlets in mobile computing. IEEE Pervasive Comput 8(4):14–23
10. Mohsen S, Somayeh K, Omid KA (2012) Survey and taxonomy of cyber foraging of mobile devices. IEEE Commun Surv Tutor 14(4):1232–1243
11. Abolfazli S, Sanaei Z, Ahmed E, 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
12. Amazon S3 (2012) http://status.aws.amazon.com/s3-20080720.html. Accessed 20 July 2012
13. Google Docs (2012) http://docs.google.com. Accessed 15 July 2012
14. MobileMe (2012) http://en.wikipedia.org/wiki/MobileMe. Accessed 15 June 2012
15. Dropbox (2012) http://www.dropbox.com. Accessed 15 July 2012
16. Apple-iCloud (2013) http://www.apple.com/icloud/. Accessed 1 January 2013
17. Introducing Amazon Silk (2013) http://amazonsilk.wordpress.com/2011/09/28/introducing-amazon-silk/. Accessed 1 January 2013
18. Cuervo E, Balasubramanian A et al (2010) MAUI: making smartphones last longer with code offload. In: MobiSys'10, San Francisco, pp 15–18
19. Chun GB, Ihm S, Maniatis P, Naik M, Patti A (2011) CloneCloud: elastic execution between mobile device and cloud. In: EuroSys'11, Salzburg, Austria, pp 10–13
20. 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. Mobile Netw Appl 16(3):270–285

21. Shiraz M, Ahmed E, Gani A, Han Q (2013) Investigation on runtime partitioning of elastic mobile applications for mobile cloud computing. J Supercomput. doi:10.1007/s11227-013-0988-6
22. Shiraz M, Abolfazli S, Sanaei Z, Gani A (2012) A study on virtual machine deployment for application outsourcing in mobile cloud computing. J Supercomput 63(3):946–964
23. Shiraz M, Gani A, Khokhar HR, Buyya R (2013) A review on distributed application processing frameworks in smart mobile devices for mobile cloud computing. Commun Surv Tutor 15(3):1294–1313
24. 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: Proceedings of the IEEE INFOCOM, pp 945–953
25. Shiraz M, Gani A, Rashid KH (2012) Towards lightweight distributed applications in mobile cloud computing. In: Proceedings of the IEEE international conference on computer science and automation engineering (CSAE 2012), China
26. Chun GB, Maniatis P (2009) Augmented smartphone applications through CloneCloud execution. Intel Research, Berkeley
27. Satyanarayanan M, Bahl P, Caceres R (2009) The case for VM-based cloudlets in mobile computing. IEEE Pervasive Comput 8(4):12–23
28. Zao B, Xu Z, Chi C, Zhu S, Cao G (2011) Mirroring smartphones for good: a feasibility study. ZTE Commun 9:13–18
29. 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
30. Giurgiu I, Riva O, Juric D, Krivulev I, Alonso G (2009) Calling the cloud: enabling mobile phones as interfaces to cloud applications. In: Proceedings of the ACM/IFIP/USENIX 10th international conference on Middleware Urbana Champaign (Middleware'09), Illinois, USA
31. Ra RM, Sheth A, Mummert L, Pillai P, Wetherall D, Govindan R (2001) Odessa: enabling interactive perception applications on mobile devices. In: MobiSys '11, pp 43–56
32. Bykov S, Geller A, Kliot G, Larus RJ, Pandya R, Thelin J (2011) Orleans: cloud computing for everyone. In: Proceedings of the 2nd ACM symposium on cloud computing, SOCC '11, pp 1–16
33. Bahl P, Han YR, Li EL, Satyanarayanan M (2012) Advancing the state of mobile cloud computing. In: MCS '12, Lake District, UK
34. Tilevich E, Smaragdakis Y (2006) J-orchestra: automatic java application partitioning. In: ECOOP 2002—object-oriented programming, pp 178–204
35. Pedrosa DL, Kothari N, Govindan R, Vaughan J, Millstein T (2012) The case for complexity prediction in automatic partitioning of cloud-enabled mobile applications. In: Computer Science Technical Report, University of Southern California, Los Angeles
36. Gu X, Nahrstedt K, Messer A, Greenberg I, Milojicic D (2003) Adaptive offloading inference for delivering applications in pervasive computing environments. In: Proceedings of the 1st IEEE international conference on pervasive computing and communications (PerCom 2003), pp 107–114
37. Ou S, Yang K, Liotta A (2006) An adaptive multi-constraint partitioning algorithm for offloading in pervasive systems. In: 4th annual IEEE international conference on pervasive computing and communications (PerCom 2006), pp 116–125
38. Goraczko M, Liu J, Lymberopoulos D, Matic S, Priyantha B, Zhao F (2008) Energy-optimal software partitioning in heterogeneous multiprocessor embedded systems. In: Proceedings of the 45th annual design automation conference, pp 191–196
39. Shiraz M, Gani A, Khokhar HR, Ahmed E (2012) An extendable simulation framework for modeling application processing potentials of smart mobile devices for mobile cloud computing. In: Proceedings of frontiers of information technology, Pakistan, pp 19–21
40. 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 (in press)
41. Abolfazli S, Sanaei Z, Gani A, Xia F, Yang TL (2013) Rich mobile applications: genesis, taxonomy, and open issues. J Netw Comput Appl. doi:10.1016/j.jnca.2013.09.009 (in press)
42. Whaiduzzaman M, Sookhak M, Gani A, Buyya R (2013) A survey on vehicular cloud computing. J Netw Comput Appl. doi:10.1016/j.jnca.2013.08.004 (in press)
43. PowerTutor (2012) http://ziyang.eecs.umich.edu/projects/powertutor/. Accessed 20 August 2012