Review

Application partitioning algorithms in mobile cloud computing: Taxonomy, review and future directions

Jieyao Liu a,*,1, Ejaz Ahmed a,2, Muhammad Shiraz a,1, Abdullah Gani a,*,1, Rajkumar Buyya b,3, Ahsan Qureshi a,1

a Centre for Mobile Cloud Computing Research (CMCCR), Faculty of Computer Science and Information Technology, University of Malaya, Kuala Lumpur, Malaysia
b Department of Computing and Information Systems, The University of Melbourne, Doug McDonell Building, Parkville Campus, Melbourne, VIC 3010, Australia

ARTICLE INFO

Article history:
Received 1 April 2014
Received in revised form 13 July 2014
Accepted 15 September 2014
Available online 1 October 2014

Keywords:
Application partitioning
Distributed application processing
Algorithm
Mobile cloud computing
Thematic taxonomy

ABSTRACT

Mobile cloud computing (MCC) enables the development of computational intensive mobile applications by leveraging the application processing services of computational clouds. Contemporary distributed application processing frameworks use runtime partitioning of elastic applications in which additional computing resources are occurred in runtime application profiling and partitioning. A number of recent studies have highlighted the different aspects of MCC. Current studies, however, have overlooked into the mechanism of application partitioning for MCC. We consider application partitioning to be an important aspect of dynamic computational offloading and therefore we review the current status of application partitioning algorithms (APAs) to identify the issues and challenges. To the best of our knowledge, this paper is the first to propose a thematic taxonomy for APAs in MCC. The APAs are reviewed comprehensively to qualitatively analyze the implications and critical aspects. Furthermore, the APAs are analyzed based on partitioning granularity, partitioning objective, partitioning model, programming language support, presence of a profiler, allocation decision, analysis technique, and annotation. This paper also highlights the issues and challenges in partitioning of elastic application to assist in selecting appropriate research domains and exploring lightweight techniques of distributed application processing in MCC.

© 2014 Elsevier Ltd. All rights reserved.

Contents

1. Introduction .......................................................... 100
2. Background .......................................................... 100
  2.1. Mobile cloud computing ........................................ 101
  2.2. Elastic applications for mobile cloud computing .......... 101
  2.3. Partitioning of elastic application for distributed application processing in MCC ........................................ 102
3. Application partitioning algorithms for MCC .................. 102
  3.1. Taxonomy of application partitioning algorithms for MCC .......................................................... 102
  3.2. Review of application partitioning algorithms based on the application partitioning models ........................................ 104
    3.2.1. Graph-based application partitioning algorithms ........ 104
    3.2.2. Linear programming-based application partitioning algorithms ........................................ 106
    3.2.3. Hybrid application partitioning algorithms ............ 108

* Corresponding author.
** Principal corresponding author.

E-mail addresses: jieyao@siswa.um.edu.my (J. Liu), ejazahmed@ieee.org (E. Ahmed), muhammad.shiraz@ieee.org (M. Shiraz), abdullahgani@ieee.org (A. Gani), raj@csse.unimelb.edu.au (R. Buyya), ahsanqureshi@siswa.um.edu.my (A. Qureshi).

1 This work is carried out as part of the Mobile Cloud Computing research project funded by the Malaysian Ministry of Higher Education under the University of Malaya High Impact Research Grant with reference UM.C/625/1/HIR/MOE/FCSIT/03.
2 Ejaz Ahmed’s research work is funded by the Bright Spark Unit, University of Malaya, Malaysia.
3 A research in CLOUDS Lab at The University of Melbourne is funded by the Australian Research Council (ARC) under its Discovery and Linkage Projects programs.

http://dx.doi.org/10.1016/j.jnca.2014.09.009
1084-8045/© 2014 Elsevier Ltd. All rights reserved.
1. Introduction

The development of mobile computing and cloud computing technologies has drastically changed the current perspective on distributed computing. Smart Mobile Devices (SMDs) such as smartphones and tablets, and Personal Computers (PCs) have become convenient and essential tools for daily life. However, applications on SMDs are still overshadowed by battery power. Mobile cloud service providers use different service provisioning models such as cloud computing (MCC) to mitigate resource limitations of SMDs by extending the services and resources of rich cloud datacenters. Cloud service providers use different service provisioning models such as Software as a Service (Greschler and Mangan, 2002; Vidyandan, 2007; Wei et al., 2008), Infrastructure as a Service (Bhardwaj et al., 2010; Prodan and Ostermann, 2009), and Platform as a Service (Keller and Rexford, 2010; Lawton, 2008) for providing access to the resources and services of computational clouds. Examples of cloud service providers include Amazon EC2 (http://www.amazon.com/ec2/), Microsoft Azure (http://www.microsoft.com/azure/), and Google AppEngine (http://appengine.google.com). MCC leverages the application processing services of computational clouds to resource-weak client devices. Recently, distributed application processing has been implemented to leverage the limitations of resources on SMDs by outsourcing the application processing load of mobile devices to cloud server nodes entirely (Goyal and Carter, 2004; Liu et al., 2009; Satyanarayan et al., 2009) or partially (Bialek et al., 2004; Cuervo et al., 2010; Giurgiu et al., 2009; Kumar et al., 2012).

Mobile applications need intensive interactions with other capabilities such as Global Positioning System (GPS) and camera, thus, it is also impractical to offload the entire application from SMDs to computational clouds (Pathak et al., 2011). Therefore, elastic applications are partitioned at different levels of granularity for distributed application processing — application partitioning is used to separate the intensive components of the mobile applications which operate independently in the distributed environment. However, runtime partitioning of elastic applications involves additional computing resources to implement runtime application profiling and solving. This makes partitioning of elastic application challenging because the mechanism requires minimal resources utilization and energy consumption cost on mobile devices (Bialek et al., 2004; Cuervo et al., 2010; Giurgiu et al., 2009; Abebe and Ryan, 2011; Park et al., 2014).

A number of recent studies have highlighted the different aspects of MCC (Abolfazli et al., 2014b; Shiraz et al., 2013a,b; Abolfazli et al., 2014a; Sanaei et al., 2014; Gani et al., 2014). Yu et al. (2013) studied application mobility in pervasive computing, which classifies and compares mobile application frameworks along with the four dimensions of design concerns in application migration. Their findings provide a systematic reference for developers to leverage different migration strategies for seamless application mobility. Fernando et al. (2012) reviewed previous researches on MCC and proposed a taxonomy for key issues in MCC — operational, end user, service levels, security, and context awareness issues for accessing cloud services. Similarly, Dinh et al. (2011) highlighted different research domains of MCC and provided an overview of the MCC including the definition, architecture, and applications. Current studies, however, have overlooked into the mechanism of application partitioning for MCC. We consider application partitioning to be an independent aspect of dynamic computational off-loading and therefore we review the current status of application partitioning algorithms (APAs) to identify the issues and challenges. The techniques used are categorized based on thematic taxonomy, and the implications and crucial issues of these techniques are analyzed. The similarities and differences of current APAs are compared based on partitioning granularity, partitioning objective, partitioning model, programming language support, profiler, allocation decision, analysis technique, and annotation.

The key contributions of this paper are the following: (a) proposing a taxonomy for classification of partitioning algorithms for distributed applications processing for mobile cloud computing; and (b) analysis of the current techniques based on relevant parameters to identify the issues which impede optimization goals of distributed application processing in MCC.

The paper is organized into the following sections. Section 2 discusses the fundamental concepts of cloud computing and MCC, elastic applications, and application partitioning for distributed application processing. Section 3 describes the thematic taxonomy of current APAs, reviews current partitioning algorithms based on taxonomy, and the implications and critical aspects of current partitioning algorithms. It also presents an analysis of current APAs based on selected parameters presented in the taxonomy. Section 4 highlights the issues and challenges in partitioning elastic application for MCC. Section 5 focuses on discussion, gap analysis, and future trends. Finally, Section 6 concludes the paper and suggests the directions for future research.

2. Background

This section presents the fundamental concepts of cloud computing and MCC, and explains the theoretical aspects of application partitioning and distributed application processing for MCC.
2.1. Mobile cloud computing

Cloud computing is the centralization of computing resources and services in the powerful datacenters which provide access on demand basis (Armbrust et al., 2010). A wide variety of devices such as PCs and laptops are used to access different types of utility programs, storage, and application development platforms over the Internet, via services offered by cloud computing providers (http://www.amazon.com/ec2/; http://www.microsoft.com/azure; http://appengine.google.com). Examples of utility programs or Software as a Service (SaaS) include Google Apps (http://www.google.com/apps/index1.html), Microsoft Office 365 (http://office365.microsoft.com), and OnLive (http://www.onlive.com/). Google Apps is similar to Microsoft Office 365 as both offer several cloud-based applications with similar functionalities to traditional office suites, which include email, calendar, instant messaging, documents, and conferencing. OnLive is a cloud-based platform which offers gaming services. Amazon EC2 is the central part of Amazon’s cloud computing platform in Amazon Web Services (AWS). EC2 provides virtual machines for the personalized configuration of client users. Windows Azure is another cloud computing platform used for building, deploying, and managing applications through Microsoft’s datacenters. Windows Azure allows users to build their application using different programming languages, tools or frameworks. Examples of application development platforms over Internet or Platform as a Service (PaaS) include Microsoft Azure (http://www.microsoft.com/azure), Google App Engine (http://appengine.google.com), AWS Elastic Beanstalk (http://aws.amazon.com/elasticbeanstalk/), and Heroku (http://www.heroku.com). The cloud providers offer a computing platform that includes an operating system, a programming language execution environment, database, and web server. Developers develop, deploy and execute application on the cloud without having to worry about the cost and complexity of buying and managing the required hardware and software. The advantages of cloud computing include cost savings, higher availability, and easy scalability. The successful implementation of cloud computing for stationary computers has motivated leveraging the on-demand services of computational clouds for mitigating resources limitations in SMDs (Shiraz et al., 2013b).

MCC is a distributed computing model in which both data storage and data processing happen outside of the mobile device. Mobile cloud applications move the computing processes and data storage away from mobile phones into the cloud. MCC has changed the current way of delivering mobile computing and communication services to global mobile users to enjoy seamless global mobile resource sharing and accesses such as electronic commerce (e-commerce) (Yang et al., 2010), mobile learning (m-Learning) (Peters, 2007), MHS (Mobile Healthcare Service) system (Li et al., 2004), E-Ambulatory Healthcare System (Rahbar, 2010) and Gmail (http://mail.google.com). Mobile users, however, are currently using SMDs that have limited processing power as well as resources to execute mobile applications effectively (Shiraz et al., 2013b). On the other hand, MCC offers cloud infrastructure and resources to mobile users by delivering various diverse and scalable service models (Greschler and Mangan, 2002; Vidyanand, 2007; Wei et al., 2008; Bhardwaj et al., 2010; Prodan and Ostermann, 2009; Keller and Rexford, 2010; Lawton, 2008; Yeo et al., 2014; Manvi and Shyam, 2014) to run mobile application with relatively low-end SMDs, regardless of time and location. It is envisaged that MCC will bring new opportunities to mobile users, mobile cloud vendors, and businesses as well as spawn more innovative applications on SMDs (http://www.abiresearch.com/research/product/1005283-mobile-cloud-applications/; http://www.juniperresearch.com/reports/mobile_cloud_applications_and_services).

Figure 1 shows that MCC is enabled by three major components — SMDs (such as smartphone, PDA, and tablet PC); advanced wireless technology (such as Wi-Fi, WiMax, 3G and LTE); and computational cloud (such as Microsoft Azure and AWS).

2.2. Elastic applications for mobile cloud computing

Distributed application processing is an important software level technique for enabling computationally intensive mobile applications on SMDs (Chun and Maniatis, 2009). A number of augmentation algorithms have been proposed for alleviating the resources limitations of SMDs — energy augmentation (Cuervo et al., 2010; Giurgiu et al., 2009, 2012; Abebe and Ryan, 2012; Chun et al., 2011; Kovachev and Klamma, 2012; Pedrosa et al., 2012; Ra et al., 2012; Wang and Li, 2004; Goraczko et al., 2008), memory augmentation (Abebe and Ryan, 2012; Gu et al., 2003), and application processing augmentation (Cuervo et al., 2010; Abebe and Ryan, 2012; Chu et al., 2004; Chun et al., 2011; Kovachev and Klamma, 2012; Ou et al., 2006; Yang et al., 2012a,b; Newton et al., 2009). Rudenko et al. (1998) introduced the concept of computational outsourcing to conserve mobile device’s energy, and this was subsequently extended to include remote application execution, which they termed as cyber foraging (Satyanarayanan, 2001). In distributed application processing, the resource-intensive components of the

![Fig. 1. Model of mobile cloud computing.](image-url)
mobile application are migrated to remote server nodes, which could either be nearby surrogate (Chun and Maniatis, 2009) or remote cloud server node (Kumar et al., 2012). Distributed application processing enables the execution of computationally intensive applications on SMDs and conserves utilization of the local resources such as memory, battery, and CPU on SMDs.

In recent years, a number of cloud server-based application offloading frameworks have been proposed for outsourcing computational intensive components of the mobile applications to cloud datacenters. Runtime computational offloading involves surrogate discovery (Balan et al., 2002; Ou et al., 2007; Navimipour et al., 2014), resource estimation (Levine et al., 1997; Reddy and Ranjan, 2003; Briand and Wieczorek, 2002), and application partitioning (Bialek et al., 2004; Cuervo et al., 2010; Smit et al., 2012; Tilevich and Smaragdakis, 2006). Computational offloading is performed for achieving different functions including saving energy, reducing turnaround time, conserving memory and CPU of mobile devices (Kumar and Lu, 2010). However, in certain cases, the offloading cost could exceed the conserved local resources because of the additional resources utilization in partitioning of mobile application and the deployment of distributed application execution platform (Sharifi et al., 2011). Hence, it is important to adopt appropriate procedures for the partitioning of elastic applications which involve minimal computing resources utilization in the establishment of distributed platform at runtime. The following section explains the fundamental concepts of elastic application, and the application partitioning mechanism.

2.3. Partitioning of elastic application for distributed application processing in MCC

The applications which are involved in runtime partitioning are known as elastic applications. Application partitioning is a technique of splitting up the application into separate components, while preserving the semantics of the original application (Veda, 2006). Computation offloading uses partitioning of the mobile application to separate the operational logic of a mobile application into distinct partitions, which are capable of operating independently in a distributed environment. The original source application may or may not be designed, implemented and deployed to run on a standalone system. The resulting components, however, are distributed to take advantage of the distributed execution platform. Thus, partitioning of elastic application is a pre-phase of computational offloading in the contemporary computational offloading frameworks for MCC. Nevertheless, both application partitioning and computation offloading are parts of the execution framework mechanism.

The fundamental attributes of elastic applications are as follows (Shiraz et al., 2013b):

- ad hoc platform is created by splitting up the distributed platform between SMDs and cloud at runtime. Application developers determine the local and remote components of the application based on the functionalities and runtime behavior including computation demand, data dependency, and communication need;
- elastic application is dynamically partitioned at runtime into separate components with different granularity levels based on the partitioning algorithms policy;
- partitioning algorithms consider the resources constraints and the application’s execution patterns. Partitions of the elastic application are offloaded adaptively to surrogates for remote execution depending on the different objective functions such as energy saving, processing power, memory storage, and fast execution; and
- elastic application executes seamlessly and transparently on remote surrogates. The transparent distributed processing environment gives the notion that an entire application is being executed on the local SMD.

Figure 2 shows the general flow of operations involving duration of the application partitioning and component offloading for MCC. The elastic application executes on SMD and the application profiling mechanism evaluates computing resources utilization, availability of resources and computing requirements of the mobile application. In a critical situation when there is insufficient resources on the SMD, the application solving-mechanism is activated to separate the computational intensive components of the application at runtime. The SMD negotiates with cloud servers for the selection of an appropriate server node and the intensive partition of the application is outsourced to a 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 the SMD.

3. Application partitioning algorithms for MCC

The current APAs for elastic applications use a number of strategies for separating the intensive components of the mobile application. This section discusses the thematic taxonomy for the classification of current APAs and reviews the algorithms based on the application partitioning model attribute of the taxonomy. It also discusses the advantages and important aspects of current APAs. The APAs are compared based on selected parameters presented in the taxonomy.

3.1. Taxonomy of application partitioning algorithms for MCC

Figure 3 shows the thematic taxonomy for the classification and comparison of APAs. The classification parameters include
The partitioning granularity attribute of an APA indicates the granularity level for partitioning computational-intensive mobile application. Current APAs implement the following granularity levels for application partitioning:

- **Module level partitioning**: The entire module of the application is partitioned and distributed (Yang et al., 2012a; Balan et al., 2007).
- **Method level partitioning**: Partitioning occurs at the method of application (Cuervo et al., 2010);
- **Object level partitioning**: The object of an application is partitioned to prepare for cyber foraging (Tilevich and Smaragdakis, 2006; Niu et al., 2014).
- **Thread level partitioning**: Partitioning occurs at the threads of an application (Chun et al., 2011; Newton et al., 2009).
- **Class level partitioning**: Application is partitioned into classes for offloading (Abebe and Ryan, 2012; Gu et al., 2003; Ou et al., 2006).
- **Task level partitioning**: Application is partitioned according to task (Wang and Li, 2004; Goraczko et al., 2008).
- **Component level partitioning**: Partitioning a group of classes which may or may not be coupled for outsourcing to the remote server (Bialek et al., 2004; Chu et al., 2004; Pedrosa et al., 2012; Ra et al., 2012; Smit et al., 2012; Yang et al., 2012b; Verbelen et al., 2013).
- **Bundle level partitioning**: Groups of Java class of applications are partitioned (Giurgiu et al., 2009, 2012; Kovachev and Klamma, 2012).
- **Allocation-site level partitioning**: Partitioning occurs on the level of allocation site where all the objects at this particular site will be seen as a single unit (Sinha and Kulkarni, 2011).
- **Hybrid level partitioning**: The results of partitioning consist of different granularity (Abebe and Ryan, 2011; Jamwal and Iyer, 2005).

The partitioning objective is represented by the objective function for application partitioning and distributed application processing. The partitioning objectives are achieved by

- **Improving performance**: Partitioning algorithms tend to increase their efficiency and effectiveness for particular measurement(s). These measurements for performance improvement include throughput (Yang et al., 2012b), adaptation time and efficacy (Abebe and Ryan, 2012), algorithm efficiency and cost effectiveness (Ou et al., 2006), latency (Cuervo et al., 2010; Chu et al., 2004; Yang et al., 2012a), execution time (Giurgiu et al., 2012; Kovachev and Klamma, 2012; Niu et al., 2014), as well as CPU workload (Newton et al., 2009).
- **Multi-site offloading**: The application is partitioned and distributed among several remote servers (Sinha and Kulkarni, 2011).
- **Reducing memory constraints**: Partitioning algorithms tend to alleviate the problem of memory restriction on SMDs (Abebe and Ryan, 2012; Gu et al., 2003).
- **Reducing network overhead**: Partitioning algorithms incur low network overhead for computation offloading (Abebe and Ryan, 2011, 2012; Giurgiu et al., 2012; Newton et al., 2009; Verbelen et al., 2013);
- **Reducing programmers burden**: Low input effort of programmers in developing and partitioning application (Cuervo et al., 2010; Chu et al., 2004; Chun et al., 2011; Jamwal and Iyer, 2005; Smit et al., 2012; Tilevich and Smaragdakis, 2006; Balan et al., 2007).
• **Saving energy:** A crucial factor in partitioning algorithms in reducing power consumption while prolonging the battery life (Cuervo et al., 2010; Giurgiu et al., 2009, 2012; Abebe and Ryan, 2012; Chun et al., 2011; Kovachev and Klamka, 2012; Pedrosa et al., 2012; Ra et al., 2012; Wang and Li, 2004; Goraczko et al., 2008; Niu et al., 2014).

• **Updating application dynamically:** The application is partitioned and updated without shutting down the application and remote servers (Bialek et al., 2004).

The **partitioning model** attribute shows the type of partitioning model used for modeling the components of mobile application or their capabilities. Current APAs implement the following models for application partitioning: (a) **Graph-based partitioning model:** APAs abstract the entire partitioning process as a graph (Smit et al., 2012; Wang and Li, 2004; Pedrosa et al., 2012; Giurgiu et al., 2009, 2012; Gu et al., 2003; Ou et al., 2006; Abebe and Ryan, 2011, 2012; Jamwal and Iyer, 2005; Niu et al., 2014; Verbelen et al., 2013). (b) **Linear Programming (LP) model:** APAs formulate linear equations to represent the application partitioning (Yang et al., 2012a; Kovachev and Klamka, 2012; Ra et al., 2012). (c) **Hybrid partitioning model:** APAs combine graph-based model and LP model for implementation of application partitioning (Cuervo et al., 2010; Chun et al., 2011; Goraczko et al., 2008; Newton et al., 2009; Yang et al., 2012b; Sinha and Kulkarni, 2011). Nevertheless, a number of partitioning algorithms do not model the application as LP equation or graph, therefore, such algorithms are classified as **exceptional APAs** (Balan et al., 2007; Chu et al., 2004; Bialek et al., 2004; Tilevich and Smaragdakis, 2006).

**Programming language support** indicates the type of programming language supported by the partitioning approach. Current APAs use the following programming languages: (a) **Single programming language support:** The partitioning approach is restricted to a particular programming environment (Bialek et al., 2004; Giurgiu et al., 2009, 2012; Abebe and Ryan, 2011, 2012; Chu et al., 2004; Gu et al., 2003; Jamwal and Iyer, 2005; Kovachev and Klamka, 2012; Ou et al., 2006; Pedrosa et al., 2012; Sinha and Kulkarni, 2011; Tilevich and Smaragdakis, 2006; Cuervo et al., 2010; Niu et al., 2014; Verbelen et al., 2013). (b) **Multiple programming languages support:** A partitioning approach that supports multiple programming languages (Yang et al., 2012b; Chun et al., 2011; Smit et al., 2012). (c) **Universal programming language support:** The partitioning approach supports a majority of the programming languages (Wang and Li, 2004; Ra et al., 2012; Balan et al., 2007).

The **profiler** attribute indicates the type of profiling used by APAs. Current APAs use the following profilers: (a) **Hardware profiler:** It collects the information relevant to the physical hardware or the SMDs such as CPU, RAM, and battery (Giurgiu et al., 2009, 2012; Chun et al., 2011; Newton et al., 2009; Cuervo et al., 2010; Goraczko et al., 2008; Kovachev and Klamka, 2012; Abebe and Ryan, 2011; Ou et al., 2006; Yang et al., 2012a,b). (b) **Software profiler:** It collects the application information such as accessed data size (send size, receive size, and transfer size), interdependency between modular structures, application behavior, performance cost (execution time, throughput, and latency), and code size (Gu et al., 2003; Cuervo et al., 2010; Giurgiu et al., 2009, 2012; Ou et al., 2006; Yang et al., 2012b; Tilevich and Smaragdakis, 2006; Abebe and Ryan, 2011). (c) **Network profiler:** It gathers information on the network environment, for example, Wi-Fi/3G/4G/Wi-Max connectivity, bandwidth, and data transfer rate (Cuervo et al., 2010; Ou et al., 2006; Yang et al., 2012a,b; Newton et al., 2009; Chun et al., 2011; Gu et al., 2003; Giurgiu et al., 2012; Niu et al., 2014).

The **allocation decision** attribute indicates the decision-making policy of the APAs in allocating components in a local or a remote server. Allocation decision is represented as (a) **online**, where the decision is made during runtime (Cuervo et al., 2010; Abebe and Ryan, 2011; Chu et al., 2004; Chun et al., 2011; Gu et al., 2003; Kovachev and Klamka, 2012; Ou et al., 2006; Pedrosa et al., 2012; Sinha and Kulkarni, 2011; Wang and Li, 2004; Yang et al., 2012a,b; Niu et al., 2014); (b) **offline**, where the decision is made before execution (Bialek et al., 2004; Abebe and Ryan, 2012; Jamwal and Iyer, 2005; Ra et al., 2012; Smit et al., 2012; Tilevich and Smaragdakis, 2006; Goraczko et al., 2008; Newton et al., 2009; Balan et al., 2007); and (c) **hybrid**, where part of the decision is done by a programmer-defined specification or static analysis tool, and part of the decision is done during application execution (Giurgiu et al., 2009, 2012; Verbelen et al., 2013).

The **analysis technique** attribute of an APA is the technique used for the identification of dependency relationships between the components of a mobile application. The analysis technique is categorized as (a) **static:** it is implemented at source level or bytecode level using analysis tools (Abebe and Ryan, 2011; Chu et al., 2004; Chun et al., 2011; Pedrosa et al., 2012; Sinha and Kulkarni, 2011; Smit et al., 2012; Newton et al., 2009; Balan et al., 2007); and (b) **dynamic:** it involves the runtime profiling, and studies the running application’s component interaction and subsequent logs (Bialek et al., 2004; Cuervo et al., 2010; Giurgiu et al., 2009, 2012; Abebe and Ryan, 2012; Gu et al., 2003; Jamwal and Iyer, 2005; Kovachev and Klamka, 2012; Ou et al., 2006; Ra et al., 2012; Tilevich and Smaragdakis, 2006; Wang and Li, 2004; Yang et al., 2012a,b; Goraczko et al., 2008; Niu et al., 2014; Verbelen et al., 2013).

The **annotation** attribute of an APA is the syntactic metadata that is added to the application source code. The annotation for current APAs can be categorized as (a) **automatic:** the execution framework implements automatic annotation by using the profiler to collect the relevant information and annotate the relevant component in the application as an indication of availability of partitioning (Giurgiu et al., 2009, 2012; Abebe and Ryan, 2011; Chun et al., 2011; Kovachev and Klamka, 2012; Sinha and Kulkarni, 2011; Yang et al., 2012a,b; Goraczko et al., 2008); and (b) **manual:** annotation is done by the application developers at the design phase, and it involves examining the intensity and scope of the components of the elastic application at design time (Bialek et al., 2004; Cuervo et al., 2010; Chu et al., 2004; Gu et al., 2003; Jamwal and Iyer, 2005; Pedrosa et al., 2012; Ra et al., 2012; Smit et al., 2012; Tilevich and Smaragdakis, 2006; Wang and Li, 2004; Newton et al., 2009; Balan et al., 2007; Verbelen et al., 2013).

The current APAs were reviewed based on application partitioning model used. Figure 3 shows the three types of application partitioning models used by current partitioning algorithms — graph model, LP model and hybrid model. Section 3.3 presents the comparison of current APAs based on the classification parameters presented in the taxonomy. It includes a detailed discussion on the significance and possible values of partitioning granularity, partitioning objective, partitioning model, programming language support, profiler, allocation decision, analysis technique, and annotation.

### 3.2 Review of application partitioning algorithms based on the application partitioning models

This section presents a review of the current APAs and discusses their implications and other important aspects based on the different application partitioning models presented in the taxonomy. Current APAs can be classified into graph-based model, LP-based model, and hybrid-based model.

#### 3.2.1 Graph-based application partitioning algorithms

Generally, the elements of graph — vertices and edges — are used to represent the parameter or context of an application. These include the available resources, data size, communication
overhead, computation cost, and memory cost. APAs adopt the graph model for modeling execution states, cost models, internal dependency, data flow, and control flow. A mobile application is modeled at different granularity levels, which include finer granularity adaptation that results in larger number of vertices with highly complex interaction patterns as compared to class graphs. Partitioning an application graph into a number of disjoint partitions is an important step for identifying the list of classes that are going to be offloaded to remote servers. An appropriate graph model can improve the efficiency of partitioning decision regardless of whether fine-grained or coarse-grained granularity is applied. Basically, each partition satisfies the condition that its weight is less than or equal to the resource availability of the device. Obtaining the optimal partitioning decision in graph-based APAs is a Non-deterministic Polynomial-Complete (NPC) problem (Garey and Johnson, 1979). 

Smit et al. (2012) proposed a novel algorithm for modeling an application as a dependency graph. Using this algorithm, existing applications are partitioned based on low effort input from developers and static code analysis. Application developers are required to annotate the code units to provide cues to the partitioning algorithm. This approach defines a set of five annotations, including Mobile and Immobile annotation, to provide cues on the public cloud versus the private cloud. The annotation language used in this approach is the Java-based Annotation4. It allows application developers to annotate the components of the application within the source code, and this makes the annotations explicit. Bialek et al. (2004) suggested that annotations can also be specified in a configuration file, which does not need any language support. However, it is less visible for maintenance whenever further annotation is needed. Additional tool support is required for the maintenance activity. The benefit is that development of the application is not affected and no manual migration is required.

Parametric program analysis is used to partition the program, whereas optimal partitioning decision is made based on runtime parameter values during execution (Wang and Li, 2004). Mobile application is modeled to task control flow graph. The optimal application partitioning is dependent on unknown execution count and allocated data size values, hence, user annotation is required. Similar to the Worst Case Execution Time (WCET) (Engblom and Ermedahl, 2000), there is no restriction that the user annotations should be constant values. Nevertheless, annotation is expressed as functions of the input parameter vector, whose values vary with different runtime parameter values. The clearly means that the parametric algorithm is useful for determining the necessary user annotations as it treats each unknown as a dummy parameter and it then solves it in the usual way. The programmer has to annotate the dummy parameter that appears in the solution as this can affect the program partitioning decision.

Pedrosa et al. (2012) proposed a graph partitioning algorithm which models the applications as a component graph, and to make the decision on where to execute the partition result before its execution. The use of complexity metrics is proposed to enhance prediction of resources usage by the components of the application. The prediction is made by taking into account relevant properties of individual component input — general purpose and type-specific input. The advantages of such algorithms are that they are modular and easily extensible to new user-defined or application-specific input complexity metrics. Programmers need to specify the test input, complexity metrics, and metric function, which are needed for training of predictors. The algorithm does not over-burden the developers as it is possible to define the metrics on a type-specific basis and share them across many components that accept the same type of data as input. Developers are also able to test and debug the applications on test inputs. It is crucial, however, for the programmers to balance metric utility with overhead, and also to specify the lightweight metric functions.

Giurgiu et al. (2009) proposed a middleware framework which automatically distributes different layers of an application among the devices and the servers while optimizing different parameters such as cost and latency. This approach incorporates a distributed module management capability which automatically and dynamically determines when and which application modules should be partitioned and offloaded. The objective is to achieve optimal performance or incur minimal cost of the overall application during execution. The application is modeled into data flow graph, and AlfredO framework (Rellermeyer et al., 2008) is used for task distribution. The partitioning approach is based on existing technology, thus, new infrastructure is not required. These solutions, however, do not support platform-independent cooperative interaction over an open network. In addition, after moving some partitions of applications from the devices to the cloud, several issues such as privacy and provenance need to be considered in advance. It is also lack of dynamic adaptation process of the computation task between mobile devices and the cloud.

Offloading Inference Engine (OIE) is a Java program-partitioning system for mobile devices, and it is used to distribute Java classes without native states (Gu et al., 2003). The algorithm partitions Java classes into groups by using adapted min-cut heuristic techniques to minimize the component interactions among the partitions. The algorithm partitions a class graph into candidate partition plans according to the edge-weight, and selects the best partition plan by comparing the combination metrics. Since most of the resource constraints are related to vertex-weights or memory consumption, the min-cut heuristic technique could miss better partitioning solutions in its candidate partition plans. The decision criterion is based only upon the memory and does not consider other factors. This approach does not consider partitioning constraints like CloneCloud (Chun et al., 2011), thus, the granularity of partitioning is coarse as it is at class level, and it focuses on static partitioning.

Ou et al. (2006) adopted a multi-cost graph model to partition an application and select an execution location for each component of the application. The decision for application partitioning is based on CPU utilization, bandwidth constraint, memory requirement of the component, and available memory on the mobile device. The algorithm assumes that there is a single mobile device, and a dedicated unconstrained surrogate node to serve as an offloading target. This approach produces better performance and adaptation, but it is expensive to compute an adaptation decision. The algorithm is also inappropriate for applications with a large number of components since it cannot be scaled to the size of the application graph. Besides, using a class level graph model is relatively less effective in yielding object topologies, and consequently more resources are utilized. This leads to performance degradation because of the heavy edges and vertices that have to be maintained by such application graphs.

Giurgiu et al. (2012) proposed an approach that optimally partitions a modular application on-the-fly among the cloud and the mobile device based on the device's CPU load, network conditions, or user inputs. The partitioning approach considers the structure of the application, resource requirements, and device constraints, using a profiler. It is able to reconfigure the current application deployment without interrupting ongoing interaction. Mobile clients are allowed to autonomously decide which configuration to adopt. The main advantage is that the algorithm makes it possible for cloud applications to run on mobile devices in a resource-efficient manner, although the applications were not originally designed for the mobile
devices platform. The applications, however, must be modularized — loosely coupled and tightly cohesive. Additionally, the partitioning approach demands prior accurate measurements of the application execution and data transfer for each specific setup.

A Hybrid Granularity Graph (HGG) has been proposed to reduce the network overhead (Abebe and Ryan, 2011). A vertex of HGG is mapped to a configurable subset of objects of a given class instead of mapping it to a single runtime component (object or class). HGG implements bytecode injection for a profiler to collect information on performance cost, memory utilization, and runtime coupling patterns, which are subsequently used to construct runtime graphs. HGG provides finer level adaptation and more effective object topologies without incurring the computational overheads of an object-level graph (Gu et al., 2003; Ou et al., 2006). It also offers finer granularity when compared to a class graph, and thus, provides more flexibility. HGG is computationally feasible because it is smaller than an object graph (Abebe and Ryan, 2011).

Abebe and Ryan (2012) proposed an abstract class graph-based algorithm, in which the devices maintain graph vertices for components within the memory space, and abstraction vertices called cloud-vertices, for components in remote devices. A multi-level graph partitioning algorithm is a novel graph partitioning heuristic, which reduces network, power supply, memory utilization, and the performance cost of adaptive offloading (Abebe and Ryan, 2012). The algorithm allows the partitioning result on the cloud to be on-loaded back to the client device for utility purpose. Additionally, the generated partitions are more effective, while the remote object coupling and migration costs are also reduced.

Janwal and Iyer (2005) adopted the breakable objects (BoBs) for compile time application partitioning in Java. BoBs are the separable entities of mobile applications. An automated refactoring process is available to make the application components amenable to partitioning. Meta-languages are used to annotate the source code of applications before partitioning. A pre-compiler generates a language-specific code for both local and remote partners. After the elements are loaded, the local and remote calls are initiated based on the prescribed definition. The algorithm mitigates the challenges of interoperability and context-awareness, however, it requires the programmers to modify the source code of applications. Therefore, application developers are required to determine the components of the application, which are appropriate for partitioning.

Niu et al. (2014) proposed Weighted Object Relational Graphs (WORG) to improve static partitioning while avoid high overhead of dynamic partitioning. The WORG combines static analysis and dynamic profiling to optimize the application partitioning. Three optimization methods are proposed in which each of them has different objectives with regard to execution time optimization, energy optimization and combination of both. The significant aspect is that bandwidth is considered as an essential manipulating variable in the proposed method which aims to minimize the trade-off of the energy or time savings against transmission costs and delay in MCC environment. Furthermore, it implements automatic partitioning methods instead of manual annotation in distributed application processing. The critical aspect is that two partitioning algorithms are proposed for two different situations, namely small application and large-scale applications.

Verbelen et al. (2013) proposed a hybrid partitioning algorithm for the allocation of components to server in the computational cloud while minimizing required bandwidth. This work is further extended by including dynamic runtime adaptation to the framework and a programming model based on annotations which aims to minimize the programmers’ burden (Verbelen et al., 2014). Simulated annealing (SA) algorithm and multilevel graph partitioning (MLKP) are combined into a hybrid partitioning algorithm to address the issue of the randomness of SA. Therefore, only at the coarsest level in MLKP, the partition is first refined with a few runs of SA. The significant aspect is that a better solution can be found with SA at the coarsest level in shorter time while the extra computation cost remains relatively low. Instead of creating balanced partitions, the hybrid partitioning algorithm tends to make sure that no cloud server gets overloaded. The critical aspect is that the paper considers single user mobile computing scenarios, thus implicitly assuming a perfect isolation in case of concurrent users. Furthermore, it does not consider the execution time metric since it only focus on applications on the cloud.

Figure 4(a) shows the abstract level flowchart of graph-based model application partitioning. In graph modeling, the first step is to check whether annotation is needed. If annotation is necessary, the programmer does the annotation manually. If annotation is not necessary, the system will continue to check whether the profiler is available in the application. If it is available, it will gather the relevant information needed by the application, otherwise, it will go to next step for graph modeling. The annotation made by the programmer as well as the profiling result are used during graph modeling. Following that, one or more algorithms are used to optimize the graph model. Finally, the result of optimization is made available to the inference algorithm such as the solver to make a decision on the partitioning.

In conclusion, graph-based APAs minimize the input effort or dispense with the need to make annotation. Besides, graph-based APAs are also computationally feasible as finer level adaptation is offered for minimizing the interactions between the component and the partition. Graph-based APAs also decrease the coupling effect and migration cost for distributed application processing. These annotations, however, are less visible for maintenance. An efficient manual annotation technique requires programmers effort to balance the metric utility and specify lightweight metric function and also to annotate the dummy parameter. In addition, graph-based APAs are not suitable for applications with a large number of components due to the high resource overhead which can decrease performance. The performance of graph-based APAs is strictly tied to the behavior of an application, for example, whether the application is modularized or not modularized. Graph-based APAs might not offer the best partitioning solution. Table 1 shows a summary of the strengths and weaknesses of graph-based APAs based on graph types. Class graph is implemented by two out of ten graph-based APAs — Distributed Abstract Class Graph (Abebe and Ryan, 2012) and OLIE (Gu et al., 2003) — to reduce the coupling effect. Class graph is observed as the most usual implemented graph type by researchers.

3.2.2. Linear programming-based application partitioning algorithms

Linear programming (LP) is a mathematical method for determining a way to achieve the best result such as maximum profit or lowest cost with a list of requirements which are represented as a linear equation in a mathematical model. Formally, LP is defined as a technique for the optimization of a linear objective function, subject to linear equality and linear inequality constraints, and it is useful for solving worst case problems (Gass, 2003). The advantage of implementing LP is that it always produces optimal results for a particular objective function. Solving LP problems, however, often requires a lot of computation time (Niemann and Marwedel, 1997).

Yang et al. (2012a) investigated the application partitioning problems from the perspective of the application provider by evaluating the constraints on the cloud resources. The partitioning logic is applied to the interactive mobile cloud applications which receive input from the sensors of the SMDs. Initially, the Single-user Computation Partitioning Problem (SCPP) is considered and the Resource Constrained Multi-user Computation Partitioning Problem (RCMCP) is then derived from SCPP. RCMCP is formulated as a
MILP problem in order to minimize the total execution time. The proposed partitioning algorithms are beneficial for the application provider to provide optimal application performance when faced with unpredictable number of users. Application providers can apply the Performance-Resource-Load (PRL) model which is obtained by algorithms in the design of a resource provisioning mechanism to achieve cost-efficient utilization of the cloud resources. The PRL model provides an optimal trade-off between

![Flowchart of application partitioning algorithms for MCC](image)

Fig. 4. A generic flowchart of application partitioning algorithms for MCC. (a) Graph-based, (b) LP-based, and (c) hybrid-based.

<table>
<thead>
<tr>
<th>APAs</th>
<th>Graph type</th>
<th>Strengths</th>
<th>Weaknesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partitioning Application (Smit et al., 2012)</td>
<td>Dependency</td>
<td>Does not affect application development</td>
<td>Less visible for maintenance task</td>
</tr>
<tr>
<td>Parametric Analysis (Wang and Li, 2004)</td>
<td>Task control</td>
<td>Does not require manual migration</td>
<td>Demands programmers effort for dummy parameter annotation</td>
</tr>
<tr>
<td>Complexity Prediction (Pedrosa et al., 2012)</td>
<td>Component</td>
<td>Useful for determining necessary user annotation</td>
<td>Needs programmers to balance metric utility and specify lightweight metric function</td>
</tr>
<tr>
<td>Calling the Cloud (Giurgiu et al., 2009)</td>
<td>Data flow</td>
<td>Demands low input effort from programmers</td>
<td>Does not support platform-independent cooperative interaction</td>
</tr>
<tr>
<td>OLIE (Gu et al., 2003)</td>
<td>Class</td>
<td>Does not need new infrastructure</td>
<td>Lacks of privacy, provenance and dynamic adaptation</td>
</tr>
<tr>
<td>Adaptive Multi-Constrained Partitioning (Ou et al., 2006)</td>
<td>Multi-cost</td>
<td>Provides better performance and effective adaptation</td>
<td>Might not offer the best partitioning solutions</td>
</tr>
<tr>
<td>Dynamic Software Deployment (Giurgiu et al., 2012)</td>
<td>Consumption graph</td>
<td>Makes it possible for an application to run in a resource-efficient manner</td>
<td>Needs modularized applications</td>
</tr>
<tr>
<td>HGG (Abebe and Ryan, 2011)</td>
<td>Hybrid granularity</td>
<td>Provides finer level adaptation and more effective object topologies</td>
<td>Demands prior accurate measurements</td>
</tr>
<tr>
<td>Distributed Abstract Class Graph (Abebe and Ryan, 2012)</td>
<td>Class</td>
<td>Computationally feasible</td>
<td>Only allowed to migrate partition to the same device</td>
</tr>
<tr>
<td>Automated Refactoring (Jamwal and Iyer, 2005)</td>
<td>Internal dependency</td>
<td>Allows cloud partitioning result to be on-loaded back to device</td>
<td>Depends on the behavior of an application</td>
</tr>
<tr>
<td>WORG (Niu et al., 2014)</td>
<td>Object relational</td>
<td>Has better interoperability and context-awareness</td>
<td>Demands more effort from application developers</td>
</tr>
<tr>
<td>Graph Partitioning (Verbelen et al., 2013)</td>
<td>Component</td>
<td>Optimal bandwidth-adaptive partitioning</td>
<td>Applied to specific situations</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Higher probability to reach global optimum</td>
<td>Demands annotation input from application developers</td>
</tr>
</tbody>
</table>
the application performance and the cost of cloud resources. The main benefit of the PRL model is that it saves the providers' operational cost.

A Mobile Augmentation Cloud Services (MACS) middleware had been proposed in order to reduce local execution time and battery power consumption (Kovachev and Klamma, 2012). MACS enables adaptive extension of Android application execution from a mobile client into the cloud. It is also responsible for the heavy lifting of adaptive application partitioning, resource monitoring, and computation offloading. These elastic applications run as usual mobile applications, but they can also use remote computing resources transparently. MACS also supports offloading of multiple Android services in one application through the allocation determination from the resource monitor. MACS uses a dynamic partitioning scheme, and lightweight as extra profiling. Resource monitoring is performed for adaptive partitioning decision during runtime.

Ra et al. (2012) stated that pre-determining a library of methods for the low processor is energy efficient in practice. This approach investigates the partitioning of applications across the two processors from an energy efficiency perspective. The algorithm provides a method for analyzing the application components to determine the most efficient placement. Partitioning guidelines and proper runtime design principles are provided to enable continuous sensing applications. It uses a simulation-based approach to obtain accurate results which are more realistic for application partitioning. Even though additional tasks are imposed on the low processor, it requires dynamic job scheduling technique, which is based on accurate resource monitoring of low processor at runtime.

LP models and formulates the application as a mathematical optimization problem where some or all of the variables are restricted to be integers. In most cases, LP is used to optimize the formulated equations which represent the mobile application as a call graph for solving graph optimization using 0-1LP optimization (Chun et al., 2011). It examines the process binary on both a target smartphone and the cloud. The database is used to determine which parts should be offloaded to the cloud. CloneCloud models the application as a control flow graph before ILP optimization (Chun et al., 2011). It examines the process using an offline static analysis of different running conditions of the process binary on both a target smartphone and the cloud. The results of the analysis are used to build a database for pre-computed partitions of the binary. The database is used to determine which parts should be offloaded to the cloud. CloneCloud is effective for static analysis of Java code to dynamically partition applications. It is, however, restricted to the input/environmental conditions during offline pre-processing. It needs to run the analysis again for every newly built application. The applications require programmers’ support for annotating the components, thus, it is difficult for ordinary users to customize the application functionality.

3.2.3. Hybrid application partitioning algorithms

The hybrid APAs incorporate the combined features from both the graph-based and LP-based application partitioning techniques. They tend to extract the important features of graph-based APAs and LP-based APAs in order to improve the performance of APAs.

Mobile Assistance Using Infrastructure (MAUI) represents the application as a call graph for solving graph optimization using 0-1LP for application partitioning (Cuervo et al., 2010). It provides fine-grained code offloading to optimize energy utilization with minimal burden to the programmer. MAUI enables programmers to produce an initial partitioning of application via annotating the local and remote components of the application. An important feature of MAUI is that it utilizes managed codes to reduce the burden on the programmers in dealing with program partitioning, and it also maximizes energy-saving during computation offloading.

CloneCloud models the application as a control flow graph before ILP optimization (Chun et al., 2011). It examines the process using an offline static analysis of different running conditions of the process binary on both a target smartphone and the cloud. The results of the analysis are used to build a database for pre-computed partitions of the binary. The database is used to determine which parts should be offloaded to the cloud. CloneCloud is effective for static analysis of Java code to dynamically partition applications. It is, however, restricted to the input/environmental conditions during offline pre-processing. It needs to run the analysis again for every newly built application. The applications require programmers’ support for annotating the components, thus, it is difficult for ordinary users to customize the application functionality.

Goraczko et al. (2008) proposed an energy-optimal software partitioning algorithm for heterogeneous multiprocessor systems. The algorithm incorporates resource model by considering the time and energy overheads of runtime mode switching. It optimizes the software partitioning during compilation by formulating it as an ILP problem, while it addresses the problem with task dependencies. The algorithm is targeted at loosely coupled multiprocessor systems. It ignores the latency and energy consumption in communication, therefore, it is an inefficient heuristic for fast computation in task mapping.

Wishbone is a profile-based APA which represents an application as a data flow graph of operators, in order to execute on multiple and heterogeneous devices in a sensor network (Newton et al., 2009). It is primarily concerned with high-rate data processing applications which statically minimize a combination of network bandwidth and CPU load during compilation by solving an ILP problem. Wishbone, however, uses WaveScript to describe...
the interconnection of the application components, which are implemented using a lower-level, device-specific language. Hence, such programming model is not robust enough for the dynamics of sensor network environments, regardless of the efficient use of the limited network resources. The algorithm does not have the flexibility for re-engineering of the sensor application in runtime failure. The service model of Wishbone is not adapted to the ever-changing processing and energy resources demands of the nodes, and it also does not consider the battery level of the device. Yang et al. (2012b) modeled the partitioning of data stream application by using a data flow graph. The genetic algorithm is used to maximize the throughput of the application. It is different from existing algorithms because it allows dynamic partitioning for a single user and supports sharing of computation requirements among multiple users in the cloud to achieve efficient utilization of the underlying cloud resources. The framework has better scalability because it is designed on the elastic cloud fabrics. The partitioning algorithm is implemented on the cloud side and the powerful computing resource available ensures that the genetic algorithm is likely to create a global optimal partition. However, this partitioning approach is not suitable for the batch computation system where the analysis of large data sets is necessary.

The multi-site and fine-grained partitioning algorithm focuses on offloading data-centric applications such as the image-matching application, which requires an application to be partitioned across multiple locations (Sinha and Kulkarni, 2011). The mobile device contains at least the user-facing modules such as the user interface, and one or more servers, which are used for computation offloading or for co-locate computation with data in order to reduce communication costs. The framework allows finer offloading decision to be made for multi-site data-centric offloading as the partitioning granularity is on the allocation-site level. However, it forces all such objects to be offloaded to the same site, even though these objects are part of different data structures, and this could result in unnecessary overheads of the partitioning approach.

The generic sequence of operations for hybrid-based model (Yang et al., 2012b; Chun et al., 2011; Goraczko et al., 2008; Cuervo et al., 2010; Sinha and Kulkarni, 2011; Newton et al., 2009) is a combination of graph-based model (Abebe and Ryan, 2012, 2011) and LP-based model (Kovacev and Klamma, 2012; Ra et al., 2012; Yang et al., 2012a). In hybrid modeling, the first step is to determine whether annotation is needed. If it is needed, it means that the programmers have provided the specifications which contain the annotation and information related to partitioning. If no annotation is necessary, it continues to check whether there is profiler in the application. If a profiler is present, relevant information will be gathered to carry out the application partitioning. If there is no profiler, it then formulates the application optimization problem into a LP equation. The annotation made by the programmer and the profiling result are useful in formulating the LP equation. LP techniques such as Integer Linear Programming (ILP), Zero–One Linear Programming (0–1LP), and Mixed Integer Linear Programming (MILP), are implemented to solve the formulated optimization problem. Finally, the optimization result is provided to the inference algorithm to make a decision on the partitioning. Figure 4(c) shows an abstract level flowchart of hybrid-based model application partitioning.

Unlike graph-based APAs and LP-based APAs, hybrid APAs do not have the same strengths and weaknesses as the other algorithms. For example, some hybrid APAs incur low overhead in the analysis technique, while others incur unnecessary overheads. Besides, some make online allocation decision while others need offline pre-processing. Nevertheless, hybrid APAs are targeted at single specific type of computation system, for example, loosely coupled multiprocessor system and data stream system. Table 3 shows a summary of hybrid APAs based on graph type, LP type, together with their strengths and weaknesses. The combination of ILP with data flow graph seems to be the most ideal hybrid APAs. This is because there are two out of six APAs from Table 3 implementing this combination.

Some partitioning algorithms do not model the application as LP equation or graph, and such algorithms are classified as exceptional APAs.

### 3.2.4. Exceptional application partitioning algorithms

Exceptional APAs do not model the application into a graph or LP equation. These exceptional APAs emphasize either manual or automatic annotation before performing application partitioning. Also, they demand relatively higher level of input effort from application developers to produce efficient partitioning solutions. Nevertheless, these algorithms, similar to the other types of APAs, yield optimal results.

Cyber foraging solution (Balan et al., 2007) is proposed with an adaptive runtime system based on the little languages Vivendi (Bentley, 1986). Programmers examine the source code of application...
and create a tactics file, called Vivendi, which contains the function prototype of each procedure which can potentially be executed remotely, and specifies how these procedures are combined to produce the partition result. The application developers need to create a tactics file for each application, and this is tedious and prone to human errors.

Roam offers a novel application framework for developers to build resource-aware seamless applications which can be adapted to heterogeneous devices (Chu et al., 2004). When designing the application, developers concentrate particularly on how to partition an application into separate components. Also, it is essential for developers to annotate each component, as to whether to provide multiple device-dependent implementations (M-DD), a single device-dependent implementation (S-DD), or a single device-independent implementation (S-DI). The existing SGUI toolkit for transforming S-DI components is difficult to use for customizing a device-independent representation for a particular device. To perform device-specific customization, the developers need to add both device-specific and application-specific transformation rules.

Bialek et al. (2004) proposed a novel approach for creating dynamically updatable Java applications, based on the concept of partitioning applications of J-Orchestra into units of dynamic updates (Tilevich and Smaragdakis, 2006). A fully developed application is partitioned in accordance to the partitioning requirement specification. The specification includes information on the number of partitions to create and what classes each partition must include. Application developers are responsible for defining the specification in a single configuration file which lists all the partitions and the modules that belong to them. Each partition needs to be labeled with unique partition name, version number, and the list of classes that belong to the partition. This approach enables transparent partitioning and dynamic update of the applications. It simplifies application updates as well as allows optimization of the application and updates performance through the careful choice of partitions.

J-Orchestra provides automatic partitioning for Java programs. The Java applications are taken as input in bytecode format and partitioned into distributed applications which can be executed on distinct Java Virtual Machines (Tilevich and Smaragdakis, 2006). J-Orchestra uses bytecode rewriting to replace method calls with remote method calls as well as direct object references with proxy references. It requires low-effort input from programmers to clearly specify the network location of various hardware and software resources and the corresponding application classes. Annotation mistakes can lead to an inefficient or an incorrect distributed application, thus, J-Orchestra provides two tools — a profiler and a classifier — in order to ensure correct and efficient partitioning. Compared with previous techniques on automatic partitioning, J-Orchestra has the advantages on generality, flexibility, and degree of automation.

Similar to hybrid APAs, exceptional APAs are also not able to share the similarities and deviations among the algorithms. Table 4 shows a summary of the strengths and weaknesses of the exceptional APAs.

### 3.3. Comparison of application partitioning algorithms for MCC by using thematic taxonomy

Table 5 shows a comparison of APAs for MCC based on the parameters presented in the taxonomy (presented in Fig. 3). The Partitioning Granularity (PG) attribute of an APA indicates the granularity level of partitioning for computational-intensive mobile application (Table 5). The granularity level of the application partitioning affects the complexity of distributed application processing and the overhead of resources utilization in runtime component migration (Shiraz et al., 2013b). Refined level granularity requires a highly intensive monitoring mechanism on the SMD at runtime as well as intensive synchronization mechanism between the SMD and the remote servers (Cuervo et al., 2010; Abebe and Ryan, 2012; Chun et al., 2011; Gu et al., 2003; Ou et al., 2006; Tilevich and Smaragdakis, 2006; Newton et al., 2009; Kosta et al., 2012). On the other hand, the coarser level of partition granularity will result in simple offloading mechanism (Giurgiu et al., 2009, 2012; Kovachev and Klamma, 2012; Yang et al., 2012a; Newton et al., 2012b; Balan et al., 2007). Nevertheless, coarser level of granularity causes increase in data transmission overhead between the SMD and the remote servers. For example, the framework for the migration of running instances of the mobile application at runtime involves the cost of additional resources utilization in transmitting the application binary file, and the data file of the running instance of the application (Hung et al., 2012). The overhead of coarse granularity level component migration increases the energy consumption cost and the turnaround time of the mobile application in cloud-based application processing. The table shows that the most implemented granularity level of partitioning is component.

The Partitioning Model (PM) attribute shows the type of partitioning model used for modeling the components or capabilities of mobile application (refer to Table 5). Three different types of application partitioning models are used in current partitioning algorithms — graph model, LP model, and hybrid model. Current APAs implement the following graph models for application partitioning: (a) class graph: describes the structure of a mobile application by showing the classes’ dependency (Giurgiu et al., 2009; Gu et al., 2003); (b) data flow graph: represents data dependencies between a number of operations (Giurgiu et al., 2009; Yang et al., 2012b; Newton et al., 2009); (c) hybrid granularity graph (HGG): represents its vertices in different levels of granularity including object, class, and especially, a configurable subset of objects of a given class (Abebe and Ryan, 2011); (d) multi-cost graph: indicates

### Table 4

<table>
<thead>
<tr>
<th>APAs</th>
<th>Strengths</th>
<th>Weaknesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simplifying Cyber Foraging (Balan et al., 2007)</td>
<td>Easy to learn to create tactics file</td>
<td>Needs to create tactics file once for each application</td>
</tr>
<tr>
<td>Roam (Chu et al., 2004)</td>
<td>Language independent</td>
<td>Tedious and prone to human errors</td>
</tr>
<tr>
<td>Dynamic Updates (Bialek et al., 2004)</td>
<td>Allows migration among heterogeneous devices</td>
<td>Difficult to customize a device-independent representation for a particular device</td>
</tr>
<tr>
<td>J-Orchestra (Tilevich and Smaragdakis, 2006)</td>
<td>Enables transparent partitioning</td>
<td>Needs partitioning guidelines</td>
</tr>
<tr>
<td></td>
<td>Simplifies and optimizes application updates</td>
<td>Needs more efficient distribution middleware</td>
</tr>
<tr>
<td></td>
<td>Requires low effort input from programmers</td>
<td>Lack of automation to infer detailed object migration strategies in response to synchronous events</td>
</tr>
</tbody>
</table>

Table 4 Strengths and weaknesses of exceptional application partitioning algorithms.
that there are several cost types on the edges of a graph (Ou et al., 2006); (e) internal dependency graph: used particularly for understanding various dependencies between fields and methods, and among methods in a BoB class (Jamwal and Iyer, 2005); (f) consumption graph: represents the application specification which captures both the application structure and the gathered statistics (Giurgiu et al., 2012); (g) object interaction graph: represents each allocation-site in application as the graph node (Sinha and Kulkarni, 2011); (h) through a mobile application during its execution (Chun et al., 2011); (i) flow graph: a program control flow graph whose nodes represent tasks (Wang and Li, 2004); (j) control flow graph: indicates all paths that might be traversed through a mobile application during its execution (Chun et al., 2011); (k) call graph: a directed graph that represents calling relationships between methods in a mobile application (Cuervo et al., 2010); (l) dependency graph: a directed graph representing dependencies among several nodes (Smit et al., 2012); (m) task graph: similar to a data flow graph, except that the tasks in a task graph represent larger units of functionality (Goraczko et al., 2008); (n) object relation graph: which can precisely extract the real structure of applications with low complexity (Niu et al., 2014); (n) Component graph: similar to class graph which describe the dependencies of each application component with the cloud servers (Verbelen et al., 2013).

Similarly, current APAs use the following LP equations:

(a) Integer Linear Programming (ILP) — wherein all the variables are integers only — is commonly used in APAs (Cuervo et al., 2010; Chun et al., 2011; Kovachev and Klamma, 2012; Verbelen et al., 2012; Yang et al., 2012b; Goraczko et al., 2008; Newton et al., 2009). ILP is usually implemented in practical situations especially those with bounded variables which is NP-hard; (b) Zero–One Linear Programming (0–1LP) involves problems in which the variables are either 0 or 1 (Sinha and Kulkarni, 2011). The problem of 0–1LP is also classified as NP-hard; (c) Mixed Integer Linear Programming (MILP) involves problems in which only some of the variables are restricted to be integers (Yang et al., 2012a). Since MILP problems are even more general than ILP problems, MILP is generally also classified as NP-hard. The Linear Programming (LP) attribute of an APA shows that the application is represented as an LP equation. A number of partitioning algorithms do not model the application as an LP equation or graph, therefore, such algorithms are classified as exceptional APAs. The table shows that the most implemented partitioning model is graph model which is implemented by ten (10) APAs. Section 3.2 provides a detailed discussion on the PM attribute and its possible values.

With the rapid development in information and communication technology, there is increasing expectations for platform-neutral programming languages. Application developers demand language-independent partitioning approach that does not restrict them to a single programming environment. The Programming Language Support (PLS) attribute indicates the type of programming language supported by current APAs (refer to Table 5). Most APAs support single programming environment, usually Java (Bialek et al., 2004; Giurgiu et al., 2009, 2012; Abebe and Ryan, 2011, 2012; Chun et al., 2004; Gu et al., 2003; Jamwal and Iyer, 2005; Kovachev and Klamma, 2012; Ou et al., 2006; Pedrosa et al., 2012; Sinha and Kulkarni, 2011; Tilevich and Smaragdakis, 2006; Niu et al., 2014; Verbelen et al., 2013) or .Net (Cuervo et al., 2010). There are, however, partitioning approaches which support multiple programming languages. These include Data Stream Application, which supports flow-based programming, which includes C++, C# and Java (Yang et al., 2012b); CloneCloud supports Virtual Machine (VM) programming environment including Java VM, Dalvik VM, and Microsoft .Net (Chun et al., 2011); and Partitioning Application supports Java and PHP (Smit et al., 2012). There are some partitioning approaches which support a wide variety of programming languages such as Parametric Analysis which supports GCC programming that includes C, C++, Objective-C, Objective-C++, Fortran, Java, Ada, and Go (Wang and Li, 2004); Improving Energy Efficiency which supports major mobile phone platforms such as Windows Phone OS, Android, and iOS (Ra et al., 2012); and Simplifying Cyber Foraging which supports C, C++, Java, Tcl/Tk and Ada (Balan et al., 2007). The table shows that most APAs focus on single programming environment instead of multiple programming environment.

<table>
<thead>
<tr>
<th>APAs</th>
<th>PG</th>
<th>PM</th>
<th>PLS</th>
<th>PR</th>
<th>AD</th>
<th>AT</th>
<th>AN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Stream Application (Yang et al., 2012b)</td>
<td>Component</td>
<td>Hybrid</td>
<td>Multiple</td>
<td>Network</td>
<td>Online</td>
<td>Dynamic</td>
<td>Automatic</td>
</tr>
<tr>
<td>HGG (Abebe and Ryan, 2011)</td>
<td>Hybrid</td>
<td>Graph</td>
<td>Single</td>
<td>Software,</td>
<td>Offline</td>
<td>Dynamic</td>
<td>N/A</td>
</tr>
<tr>
<td>Distributed Abstract Class Graph (Abebe and Ryan, 2012)</td>
<td>Class</td>
<td>Graph</td>
<td>Single</td>
<td>N/A</td>
<td>Offline</td>
<td>Dynamic</td>
<td>N/A</td>
</tr>
<tr>
<td>OLIE (Gu et al., 2003)</td>
<td>Class</td>
<td>Graph</td>
<td>Single</td>
<td>All</td>
<td>Online</td>
<td>Dynamic</td>
<td>N/A</td>
</tr>
<tr>
<td>Adaptive Multi-Constrained Partitioning (Ou et al., 2006)</td>
<td>Class</td>
<td>Graph</td>
<td>Single</td>
<td>N/A</td>
<td>Offline</td>
<td>Dynamic</td>
<td>Manual</td>
</tr>
<tr>
<td>Automated Batching (Jamwal and Iyer, 2005)</td>
<td>Hybrid</td>
<td>Graph</td>
<td>Single</td>
<td>All</td>
<td>Hybrid</td>
<td>Dynamic</td>
<td>Automatic</td>
</tr>
<tr>
<td>Dynamic Software Deployment (Giurgiu et al., 2012)</td>
<td>Bundle</td>
<td>Graph</td>
<td>Single</td>
<td>All</td>
<td>Hybrid</td>
<td>Dynamic</td>
<td>Automatic</td>
</tr>
<tr>
<td>Parametric Analysis (Wang and Li, 2004)</td>
<td>Task</td>
<td>Graph</td>
<td>Universal</td>
<td>N/A</td>
<td>Online</td>
<td>Dynamic</td>
<td>Manual</td>
</tr>
<tr>
<td>RCMCPP (Yang et al., 2012a)</td>
<td>Module</td>
<td>LP</td>
<td>N/A</td>
<td>Network,</td>
<td>Online</td>
<td>Dynamic</td>
<td>Automatic</td>
</tr>
<tr>
<td>Multi-site Computation Offloading (Sinha and Kulkarni, 2011)</td>
<td>Allocation-site</td>
<td>Hybrid</td>
<td>Single</td>
<td>N/A</td>
<td>Online</td>
<td>Dynamic</td>
<td>Automatic</td>
</tr>
<tr>
<td>Calling the Cloud (Giurgiu et al., 2009)</td>
<td>Bundle</td>
<td>Graph</td>
<td>Single</td>
<td>Software,</td>
<td>Hybrid</td>
<td>Dynamic</td>
<td>Automatic</td>
</tr>
<tr>
<td>CloneCloud (Chun et al., 2011)</td>
<td>Thread</td>
<td>Hybrid</td>
<td>Multiple</td>
<td>Network,</td>
<td>Online</td>
<td>Statis</td>
<td>Automatic</td>
</tr>
<tr>
<td>MACS (Kovachev and Klamma, 2012)</td>
<td>Bundle</td>
<td>LP</td>
<td>Single</td>
<td>Software,</td>
<td>Online</td>
<td>Dynamic</td>
<td>Automatic</td>
</tr>
<tr>
<td>Improving Energy Efficiency (Ra et al., 2012)</td>
<td>Component</td>
<td>LP</td>
<td>Universal</td>
<td>N/A</td>
<td>Offline</td>
<td>Dynamic</td>
<td>Manual</td>
</tr>
<tr>
<td>MAUI (Cuervo et al., 2010)</td>
<td>Method</td>
<td>Hybrid</td>
<td>Single</td>
<td>All</td>
<td>Online</td>
<td>Dynamic</td>
<td>Manual</td>
</tr>
<tr>
<td>Partitioning Application (Smit et al., 2012)</td>
<td>Component</td>
<td>Graph</td>
<td>Multiple</td>
<td>N/A</td>
<td>Offline</td>
<td>Static</td>
<td>Manual</td>
</tr>
<tr>
<td>Dynamic Updates (Bialek et al., 2004)</td>
<td>Component</td>
<td>Graph</td>
<td>Exceptional</td>
<td>N/A</td>
<td>Offline</td>
<td>Dynamic</td>
<td>Manual</td>
</tr>
<tr>
<td>Roam (Chu et al., 2004)</td>
<td>Component</td>
<td>Graph</td>
<td>Exceptional</td>
<td>N/A</td>
<td>Offline</td>
<td>Static</td>
<td>Manual</td>
</tr>
<tr>
<td>Complexity Prediction (Pedrosa et al., 2012)</td>
<td>Component</td>
<td>Graph</td>
<td>Single</td>
<td>N/A</td>
<td>Online</td>
<td>Static</td>
<td>Manual</td>
</tr>
<tr>
<td>Simplifying Cyber Foraging (Balan et al., 2007)</td>
<td>Module</td>
<td>Exceptional</td>
<td>Universal</td>
<td>N/A</td>
<td>Offline</td>
<td>Static</td>
<td>Manual</td>
</tr>
<tr>
<td>Energy-Optimal Software Partitioning (Goraczko et al., 2008)</td>
<td>Task</td>
<td>Hybrid</td>
<td>N/A</td>
<td>Software,</td>
<td>Offline</td>
<td>Dynamic</td>
<td>Automatic</td>
</tr>
<tr>
<td>Wishbone (Newton et al., 2009)</td>
<td>Thread</td>
<td>Hybrid</td>
<td>Multiple</td>
<td>Network,</td>
<td>Offline</td>
<td>Static</td>
<td>Manual</td>
</tr>
<tr>
<td>WORB (Niu et al., 2014)</td>
<td>Object</td>
<td>Graph</td>
<td>Single</td>
<td>Network</td>
<td>Online</td>
<td>Static</td>
<td>N/A</td>
</tr>
<tr>
<td>Graph Partitioning (Verbelen et al., 2013)</td>
<td>Graph</td>
<td>Single</td>
<td>N/A</td>
<td>Hybrid</td>
<td>Dynamic</td>
<td>Manual</td>
<td>Automatic</td>
</tr>
</tbody>
</table>

The Profiler (PR) attribute indicates the type of profiling used by APA (refer to Table 5). The profiling mechanism is a form of dynamic program analysis to measure, for example, the memory or the time complexity of an application, the usage of particular instructions, or frequency and duration of function calls. Profiling information is commonly used in optimizing application partitioning, especially in MCC. Profiling is achieved by instrumenting either the program source code or its binary executable form using a tool called a profiler. APAs which implement hardware profiler are discussed in this paper. Dynamic Software Deployment (Giurgiu et al., 2012), CloneCloud (Chun et al., 2011) and Wishbone (Newton et al., 2009) focus on collecting CPU information as they aim at developing lightweight algorithms. On the other hand, energy saving is given the top priority by some APAs (Cuervo et al., 2010; Goraczko et al., 2008; Kovachev and Klamma, 2012) as they need longer battery life. Similarly, memory status is profiled for these algorithms (Giurgiu et al., 2009; Abebe and Ryan, 2011; Ou et al., 2006) as they emphasize on RAM utilization. RCMCPP collects all the hardware information about the device or SMDs including CPU, RAM and battery life (Yang et al., 2012a).

The following section discusses APAs which implement software profiler. OLIE (Gu et al., 2003) and MAUI (Cuervo et al., 2010) profile the program behavior into their execution framework. Similarly, the size of the accessed data and the code size are profiled in these partitioning algorithms (Giurgiu et al., 2009; Ou et al., 2006; Yang et al., 2012b). Besides, Dynamic Software Deployment (Giurgiu et al., 2012) and J-Orchestra (Tilevich and Smaragdakis, 2006) concentrate on the interdependency of classes and the modular structure of the application. The HGG (Abebe and Ryan, 2011) profiles the performance cost for optimal partitioning decision. Finally, implementation of network profiler is discussed.

From studied algorithms (Cuervo et al., 2010; Ou et al., 2006; Yang et al., 2012a,b; Newton et al., 2009), we observed that profiling bandwidth condition is essential. CloneCloud (Chun et al., 2011) profiles the transfer rate between SMDs and clouds. OLIE (Gu et al., 2003) profiles the network latency. WORG (Niu et al., 2014) profiles the execution time for data transfer, and Dynamic Software Deployment (Giurgiu et al., 2012) profiles the type of network. However, there are partitioning algorithms which do not implement profiler in their execution framework (Bialek et al., 2004; Abebe and Ryan, 2012; Chu et al., 2004; Jamwal and Iyer, 2005; Pedrosa et al., 2012; Ra et al., 2012; Sinha and Kulkarni, 2011; Smit et al., 2012; Wang and Li, 2004; Balan et al., 2007). Most of the partitioning algorithms which have been reviewed implement a profiler to gather information from at least two among three categories (hardware, software, network) — hardware with software (Giurgiu et al., 2009; Abebe and Ryan, 2011; Kovachev and Klamma, 2012; Goraczko et al., 2008), hardware with network (Chun et al., 2011; Yang et al., 2012a; Newton et al., 2009), or software with network (Gu et al., 2003). Nevertheless, other partitioning algorithm profilers collect the information from all three categories (Cuervo et al., 2010; Giurgiu et al., 2012; Ou et al., 2006). On the other hand, J-Orchestra (Tilevich and Smaragdakis, 2006) uses only a software profiler, while Data Stream Application (Yang et al., 2012b) and WORG (Niu et al., 2014) implements only a network profiler. The table shows that all types of profilers are used by most of the APAs.

The Allocation Decision (AD) attribute of an APA indicates its decision-making policy for allocating components in a local or a remote server (refer to Table 5). At runtime, the AD results in relatively more optimal solution compared to offline AD. However, the online AD incurs computational overhead on the SMD because additional computing resources of SMDs are utilized. Offline AD, on the other hand, incurs lower overhead of the SMDs, but it produces less optimal partitioning decisions. The hybrid AD provides the solution to this issue of balancing the pros and cons of online/offline ADs. The table shows that most of the APAs use online AD and offline AD.

The Analysis Technique (AT) attribute of an APA represents the technique used in identifying the dependency relationships between components in mobile applications (refer to Table 5). Static AT is an easy-to-use method as it uses the existing libraries for source code analysis. Source code level analysis is easier to understand and run, whereas bytecode level analysis involves analyzing system classes which are difficult to understand. Static AT is lightweight and incurs minimal overhead on the SMDs in application partitioning. It requires input from programmers with different levels of effort. Dynamic AT requires minimal effort from the programmers for analyzing the mobile application. Compared to static AT, dynamic AT involves thorough analysis for gathering context information which is used for making optimal partitioning decision. The table shows that most of the APAs implement dynamic AT.

Annotation (AN) attribute of an APA represents the syntactic metadata that is added to the application source code (refer to Table 5). Programmers annotate the components of the elastic application at different granularity levels, such as object, classes, methods, variables, parameters, and modules. The annotation of the component of the mobile application is used as an input for the application profiler to make the decision on application partitioning at runtime. Automatic annotation greatly reduces the efforts of application developers in annotating the intensive components of the application at the design stage. In addition, automatic annotation contributes to the seamless execution of elastic application in the distributed MCC environment. However, automatic annotation incurs additional resources utilization cost on the local mobile device. On the other hand, manual annotation by programmers demands relatively higher input effort for examining the intensity and scope of the components of the mobile application at the design stage. Nevertheless, programmers are able to annotate the application manually to achieve optimal application partitioning if the programmers have good knowledge of the particular application. The table shows that most of the APAs need manual annotation.

The Not Applicable (N/A) value in Table 5 indicates that the APA does not implement the corresponding attribute. For example, Distributed Abstract Class Graph (Abebe and Ryan, 2012) and Automated Refactoring (Jamwal and Iyer, 2005) do not use the profiling mechanism, therefore, the value of the PR attribute is represented as ‘N/A’ in the corresponding cell of the table.

Table 6 shows a comparison of current APAs based on their partitioning objectives. Some existing partitioning algorithms focus on a single objective function to perform application partitioning (Bialek et al., 2004; Giurgiu et al., 2009; Abebe and Ryan, 2011; Gu et al., 2003; Jamwal and Iyer, 2005; Ou et al., 2006; Pedrosa et al., 2012; Ra et al., 2012; Sinha and Kulkarni, 2011; Smit et al., 2012; Tilevich and Smaragdakis, 2006; Wang and Li, 2004; Yang et al., 2012a,b; Goraczko et al., 2008; Balan et al., 2007). For instance, HGG (Abebe and Ryan, 2011) attempts to reduce the network overhead while Parametric Analysis (Wang and Li, 2004), Calling the Cloud (Giurgiu et al., 2009), Improving Energy Efficiency (Ra et al., 2012), Complexity Prediction (Pedrosa et al., 2012), and Energy-Optimal Software Partitioning (Goraczko et al., 2008) focus on energy-saving. On the other hand, Data Stream Application (Yang et al., 2012b), Adaptive Multi-Constrained Partitioning (Ou et al., 2006), and RCMCPP (Yang et al., 2012a) focus on improving performance. A single objective partitioning is always relatively easier to be implemented. This approach is usually better than multiple objective partitioning approach if only a single parameter or attribute is used for comparison. Nevertheless, as technology advances, partitioning algorithms tend to be more adaptive and context-aware. Many algorithms set multiple objectives for application partitioning and component offloading (Cuervo et al., 2010; Abebe and Ryan, 2012; Chu et al., 2004; Chun et al., 2011; Giurgiu et al., 2012; Kovachev and Klamma, 2012; Newton et al., 2009;
Table 6
Comparison of application partitioning algorithms based on the partitioning objective.

<table>
<thead>
<tr>
<th>APAs</th>
<th>RNO</th>
<th>SE</th>
<th>IP</th>
<th>RMC</th>
<th>RPB</th>
<th>UAD</th>
<th>MO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Stream Application</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>HGG (Abebe and Ryan, 2011)</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Distributed Abstract Class Graph (Abebe and Ryan, 2012)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>OLIE (Giurgiu et al., 2003)</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Adaptive Multi-Constrained Partitioning (Ou et al., 2006)</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Automated Refactoring (Jawad and Iyer, 2005)</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Dynamic Software Deployment (Giurgiu et al., 2012)</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>J-Orchestra (Tilevich and Smaragdakis, 2006)</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Parametric Analysis (Wang and Li, 2004)</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>RCMCPP (Yang et al., 2013)</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Multi-site Computation Offloading (Sinha and Kulkarni, 2011)</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Calling the Cloud (Giurgiu et al., 2009)</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>CloneCloud (Chun et al., 2011)</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>MACS (Kovachev and Klamma, 2012)</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Improving Energy Efficiency (Ra et al., 2012)</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>MAUI (Cuervo et al., 2010)</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Partitioning Application (Smit et al., 2012)</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Dynamic Updates (Bialek et al., 2004)</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Roam (Chu et al., 2004)</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Complexity Prediction (Pedrosa et al., 2012)</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Simplifying Cyber Foraging (Balan et al., 2007)</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Energy-Optimal Software Partitioning (Goraczko et al., 2008)</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Wishbone (Newton et al., 2009)</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>WORG (Niu et al., 2014)</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Graph Partitioning (Verbelen et al., 2013)</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>


For instance, Distributed Abstract Class Graph (Abebe and Ryan, 2012) aims at reducing network overhead, saving energy, improving performance, and reducing memory constraints. Besides, Roam (Chu et al., 2004) is aimed at improving performance, and reducing the programmers’ burden. On the other hand, Dynamic Software Deployment (Giurgiu et al., 2012) is aimed at reducing network overhead and saving energy. WORG (Niu et al., 2014) concerns about energy saving and improving the performance by minimizing the execution time. In Table 6, the ‘Yes’ value of the attribute shows that the APA implements the corresponding objective function, whereas the ‘No’ value of the attribute shows that partitioning algorithm does not implement the corresponding objective function. Table 6 shows that most of the APAs focus on saving SMDs’ energy and improving the performance of application.

4. Issues and challenges for partitioning of elastic applications in MCC

This section highlights the issues and challenges in partitioning of elastic applications for MCC, which will help in selecting the appropriate domain for future research and exploring lightweight techniques for distributed application processing in MCC.

4.1. Synchronization in distributed deployment of elastic application for MCC

In distributed application processing, elastic application is partitioned and the intensive components of the application are offloaded to cloud server nodes (Cuervo et al., 2010; Giurgiu et al., 2009; Chun et al., 2011; Ou et al., 2006; Tilevich and Smaragdakis, 2006). As a result, a distributed application processing platform is established and the application processing load of mobile devices is outsourced to the cloud server. Some APAs currently implement a mechanism for distributed processing of intensive mobile applications in which the execution on the mobile application is blocked unless the results are returned from the remote server node (Cuervo et al., 2010; Giurgiu et al., 2009; Chun et al., 2011). An intensive synchronization mechanism is required to ensure the consistency of application execution in the distributed environment. Similarly, ThinkAir proposed an application execution framework by implementing simultaneous execution of the offloaded partitions of the application to the cloud server node (Kosta et al., 2012). It is challenging to implement simultaneous execution of mobile applications because parallel distributed processing of mobile application necessitates close synchronization between the mobile device and the remote server node.

Similarly, it is crucial to ensure regular semantics for the distributed processing of mobile application which is partitioned at runtime (Tilevich and Smaragdakis, 2006; Wang and Li, 2004; Fahringer and Jugravu, 2005). The issue of self-deadlocks must also be seriously addressed because the thread identity is not maintained when the flow of control is moved over the network. For example, we assume that thread t1 represents the current head segment and owns the monitor of method first(). Self-deadlock occurs if t1 becomes inactive, and calls another remote method second() which creates a new head segment t2. If the distributed thread t3 enters the method first2() which is synchronized again on the same object, then a deadlock will occur. This thread t1 is currently waiting for its successor segment t2 to return from the remote invocation. However, this segment never returns because the current head of the distributed thread t3 applies for the monitor owned by t1. The deadlocks can be prevented by using a standardized Inter Process Communication Mechanism (IPC) such as Remote Method Invocation (RMI) or Remote Procedure Call (RPC) rather than finer component migration at runtime. However, it is challenging to use RPC and RMI in the distributed MCC environment.

During dynamic class update of an application, inheritance across partitions is introduced (Bialek et al., 2004). When the parent and the child classes are separated, the child object instance requires the parent object to be instantiated in its partition. J-Orchestra addresses this issue in such a way that an update of the parent class will also be directly reflected on all the derived classes, but the partitions of the application which are...
delegated to remote server node will require the parent class to be instantiated before the child class (Tilevich and Smaragdakis, 2006). The phenomenon becomes complicated when an abstract parent class cannot be instantiated, directly. Current APAs focus on the optimization of partitioning mechanism, but the consistency of distributed partitions is still an addressable issue (Abebe and Ryan, 2012; Gu et al., 2003; Kovachev and Klamma, 2012). A distributed synchronization mechanism requires the maintenance of identical semantics for all the partitions of the applications. A synchronization mechanism is used for exchanging the active state information of the mobile application between the mobile device and the cloud server node (Hung et al., 2012). However, to achieve continuous synchronization, the mobile device must be kept in the active state, in which case, the wireless medium is accessed continuously, thus making this an energy-starving mechanism (Tilevich and Smaragdakis, 2006). Hence, it is a challenge to use an optimal synchronization mechanism for ensuring consistency in distributed partitions of the mobile applications.

4.2. Appropriate application partition granularity

The granularity of application partitioning for distributed application processing is important for an optimal partitioning algorithm design. The granularity level of the application partitioning can affect the complexity of distributed application processing and the resources utilization overheads in runtime component migration (Shiraz et al., 2013b). Finer partitioning granularity requires a highly intensive monitoring mechanism on the SMD at runtime as well as an intensive synchronization mechanism between the SMD and the remote servers (Cuervo et al., 2010; Abebe and Ryan, 2012; Chun et al., 2011; Gu et al., 2003; Ou et al., 2006; Tilevich and Smaragdakis, 2006; Newton et al., 2009; Kosta et al., 2012). Consistency is another concern for finer level of granularity in the distributed execution of mobile application. The object level makes the decision to offload all objects of a particular class (Tilevich and Smaragdakis, 2006; Sinha and Kulkarni, 2011). Using objects as the granularity of partitioning presents problems. Firstly, it is difficult to represent each application as an object graph for partitioning because the number of objects is potentially unbounded, and moreover, indeterminate statically. Secondly, even if a well-suited object graph that is amenable to partitioning is given, it is always unclear on how to generate the code to support distributed application processing.

A number of issues are also associated with the different granularity levels of application partitioning such as compatibility, object identity, class unloading, state transfer, and performance overhead. Finer level of partitioning involves overhead for ensuring the consistency of an application before and after partitioning. Besides, it is necessary to verify and validate the object identity and the compatibility for security and performance purposes. Dynamic update of applications should not compromise the correctness of security and semantics as well as stability of the updated applications. Whenever the bytecode of a class is replaced dynamically by a new bytecode, all program entities that refer either to this class or new bytecode, all program entities that refer either to this class or any of its instances have to be updated accordingly. Therefore, sustaining the consistency of each component of the application is challenging in elastic applications. Classes loaded by a different class loader cannot refer to each other unless they are loaded by the same parent class loader (Bialek et al., 2004). This leads to the creation of a hierarchy of class loaders, and this is not desirable for long-term applications whose structures might change over time. It is possible to introduce cyclic class dependencies. In addition, dynamic update requires some level of indirection so that method calls are directed to the correct objects and classes. As this level of indirection is required for every updatable class, some runtime performance degradation will occur for all updatable classes. Hence, it is challenging to implement optimal granularity level for application partitioning which involves minimal communication overhead for component offloading and lightweight mechanism for the establishment and management of distributed platforms.

4.3. High software quality requirements for application partitioning

Current partitioning algorithms are designed to offload intensive parts of a mobile application from the mobile device to the cloud, thereby prolonging battery life and enhancing performance. Distributed application processing frameworks use different objective functions for partitioning of elastic applications. Such frameworks focus on which component of the application to partition, and when to offload application partitions for remote execution (Giurgiu et al., 2012). The decision to perform application partitioning involves selecting appropriate partitioning granularity, as well as addressing application optimization problems. Partitioning of elastic/adaptive applications requires continuous profiling mechanism which is time critical, energy consuming, and resources intensive (Shiraz et al., 2013b; Cuervo et al., 2010; Giurgiu et al., 2009, 2012; Abebe and Ryan, 2011; Chun et al., 2011; Gu et al., 2003; Kovachev and Klamma, 2012; Ou et al., 2006; Tilevich and Smaragdakis, 2006; Yang et al., 2012a,b; Goraczko et al., 2008; Newton et al., 2009).

Current algorithms for application partitioning do not consider the changes in the network bandwidth or latency, sudden increase of the CPU load on the mobile device, and variations in the user’s inputs during interactions. On the other hand, running the applications remotely is affected by the network bandwidth, which might result in poor user experiences (Cuervo et al., 2010; Giurgiu et al., 2012; Smit et al., 2012). The distributed execution platform of elastic application for MCC necessitates lightweight and fault-tolerant algorithms for application partitioning and distributed application processing. Hence, high software quality is required for addressing the aforementioned issue, which can impact the optimization of the partitioning algorithm, and rich user experience in the distributed deployment of elastic application for MCC.

4.4. Developers support for application partitioning

Current software engineering practices for application development require minimum efforts from application developers. This has contributed greatly to the rapid development of software, and reduced the developmental time and cost. However, current APAs require involvement of the application developers for annotating the scope and intensity of the elastic application (Bialek et al., 2004; Cuervo et al., 2010; Chu et al., 2004; Gu et al., 2003; Jamwal and Iyer, 2005; Pedrosa et al., 2012; Ra et al., 2012; Smit et al., 2012; Tilevich and Smaragdakis, 2006; Wang and Li, 2004; Newton et al., 2009; Balan et al., 2007). Programmers annotate the components of the elastic application as a local or a remote component at different granularity levels (Cuervo et al., 2010; Kosta et al., 2012). The annotation is used as an input for the application profiler to make decision on application partitioning at runtime. Manual annotation of the application is implemented by the application developers at the design stage, and this involves examining the intensity and scope of the components of the elastic application. Hence, it is challenging to automate the application partitioning mechanism by eliminating the additional effort to annotate the components at different granularity levels. For instance, ThinkAir provides a library together with the compiler support to make the programmer’s job more straightforward (Kosta et al., 2012). The compiler annotates any method being considered for offloading with @Remote. The ExecutionController detects remotely tagged methods and handles associated profiling, decision-making and communication with the application server.
without the developer having to be aware of the details of partitioning and distributed application processing. However, effort is still required to classify the intensive components of the applications in a formal syntax in the source file, which can be detected by the compiler to generate the necessary remotable method wrappers. Therefore, it is challenging to minimize the developmental efforts for the classification of the components of the application as local or remote. The issue is even more challenging for finer granularity level partitioning as compared to coarse granularity level partitioning.

4.5. Diverse design objective for application partitioning

Computation offloading is implemented as a significant application layer solution for enabling computationally intensive mobile applications on SMDs. Using the offloading technique, the main problem encountered is in partitioning the computational load of the intensive mobile application between the mobile device and the cloud. Computational offloading frameworks consider several objective functions on mobile devices for making the decision on application partitioning and component offloading (Kumar et al., 2012). As shown in Table 6, current APAs are designed to reduce network overhead (Abebe and Ryan, 2011, 2012; Giurgiu et al., 2012; Newton et al., 2009), save energy (Abebe and Ryan, 2012; Giurgiu et al., 2009, 2012; Wang and Li, 2004; Chun et al., 2011; Ra et al., 2012; Cuervo et al., 2010; Kovachev and Klamma, 2012; Pedrosa et al., 2012; Goraczko et al., 2008), as well as improve performance (Yang et al., 2012a,b; Abebe and Ryan, 2012; Ou et al., 2006; Chun et al., 2011; Kovachev and Klamma, 2012; Cuervo et al., 2010; Chu et al., 2004; Newton et al., 2009). However, many APAs do not consider the application partitioning problem from the perspective of the service provider (Yang et al., 2012b). For instance, Amazon released a cloud-accelerated web browser, known as Silk, which is a split browser and resides on both Kindle Fire and EC2. Silk dynamically partitions the application and determines the distribution of computational load between the SMD and the remote Amazon EC2. It also considers the objective functions of the network conditions, page complexity, and the location of any cached content. However, a challenging aspect of such applications is providing uniform services and ensuring the QoS for a large number of mobile users. The traditional computational offloading algorithms focus on single-site offloading, where an application is divided between the mobile device and a single remote server (Bialek et al., 2004; Cuervo et al., 2010; Giurgiu et al., 2009, 2012; Abebe and Ryan, 2011, 2012; Chu et al., 2004; Chun et al., 2011; Gu et al., 2003; Jamwal and Iyer, 2005; Kovachev and Klamma, 2012; Ou et al., 2006; Pedrosa et al., 2012; Ra et al., 2012; Smit et al., 2012; Tilevich and Smaragdakis, 2006; Wang and Li, 2004; Yang et al., 2012a,b; Goraczko et al., 2008; Newton et al., 2009; Balan et al., 2007). However, the contemporary frameworks for computational offloading implement multi-site offloading to ensure seamless application execution by reducing the turnaround time of the application (Sinha and Kulkarni, 2011; Kosta et al., 2012). Thus, it further complicates the decision on application partitioning and component offloading. It is challenging to design algorithms with diverse objective functions for application partitioning and component offloading such as offline usability, rich user experiences, battery power conservation and seamless application execution.

4.6. Availability of remote services in MCC

The mobile nature of SMDs demands the availability of services even when there is disruption of remote services. The wide variations and dynamic changes in network conditions and local resources availability must also be addressed. Whereas the APAs identify the intensive components and create application partitions at runtime (Chu et al., 2004; Giurgiu et al., 2012; Kovachev and Klamma, 2012; Sinha and Kulkarni, 2011; Smit et al., 2012; Yang et al., 2012a,b). It is challenging to make the decision to determine the elastic application components, which need to be offloaded at runtime, and find the appropriate destination server node. Such algorithms require the selection of the remote server node on an ad hoc basis. This is a resource-intensive effort, and it can adversely affect the turnaround time of the offloaded partitions of the application (Giurgiu et al., 2012).

The cloud server is required to handle the unpredictable and varying loads from multiple mobile client applications. Different cloud computing services provide different privacy guarantees, I/O performance, CPU power, or network latency (Smit et al., 2012). In addition, the components of an application can have different privacy/security demands, I/O needs, CPU requirements, or latency expectations. Even though existing APAs enable adaptive execution of the mobile application between the mobile devices and the server, such algorithms do not provide any solution on how to utilize the elastic resources in the clouds to make the applications scalable in situations where a large number of mobile users need to be served. Because scheduling the computation across the client and clouds is not flexible and does not have an adaptive mechanism, other efforts in facilitating large-scale cloud applications are not appropriate in the MCC applications.

5. Discussion, gap analysis, future trends

This paper distinguishes application partitioning as an independent aspect of the dynamic computational offloading technique for MCC. It reviewed the state-of-the-art APAs in distributed application processing to identify the main issues and challenges. Current application partitioning techniques, categorized based on thematic taxonomy, and other important aspects are evaluated. We compared the similarities and differences among the current APAs based on relevant parameters. Current APAs adopt different models for modeling elastic applications such as graph-based model, the LP optimization problem for application partitioning, and the hybrid model which adopts the graph model combined with LP-based modeling. Some algorithms are restricted to the use of a single programming language, whereas others use multi-programming language support. APAs incorporate different types of profilers, which include network, software, and hardware profilers. Some APAs have features for online allocation decision, while others incorporate offline allocation decision. The static analysis technique is used in some algorithms, whereas others use dynamic analysis technique for application partitioning. Some algorithms use automatic annotation for defining the scope of the components of the mobile application, whereas others still need manual annotation by application developers. APAs have been developed to achieve diverse objectives such as reducing network overhead, saving energy, improving performance, reducing memory constraints, reducing programmers’ burden, dynamically updating application, and multi-site offloading.

For graph-based APAs, the annotations are less visible for maintenance. An efficient manual annotation technique requires programmers effort to balance the metric utility and specify lightweight metric function. In addition, graph-based APAs are not suitable for applications with a large number of components due to the high resource overhead which can decrease performance. Besides, the performance of graph-based APAs is strictly tied to the behavior of an application, for example, whether the application is modularized or not modularized. Hence, graph-based APAs might not offer the best partitioning solution. On the other hand, LP-based APAs need dynamic scheduling technique to produce optimal partition results. The dynamic scheduling...
technique demands high overheads for extra profiling and resource monitoring which further overuse the available resources on SMDs. Both hybrid APAs and exceptional APAs are not guaranteed to produce the optimal partitioning solution in most cases. This is because hybrid APAs target at single specific type of computation system, for example, loosely coupled multiprocessor system and data stream system while exceptional APAs emphasize either manual or automatic annotation before performing application partitioning which demand relatively higher level of input effort from application developers.

Current APAs focus on separating the intensive components of the application while profiling the availability of resources on local mobile device and do not consider the availability of services on the computational clouds. The computational offloading techniques do not incorporate a fault-tolerant mechanism to cope with partition loss. Similarly, the dynamic application profiling mechanism in application partitioning is time-consuming, energy starving, and resources intensive. The limited resources in SMDs necessitate minimal utilization of resources in application partitioning and distributed application processing for MCC. Similarly, the mobile nature of SMDs and the intrinsic limitations in the wireless access medium of MCC require adaptive and context-aware APAs. Application adaptation aims to enable mobile applications to cope with the varying execution environments and allows application to adaptively (re)adjust the placement of its partitions at runtime while context-awareness deals with linking changes in the environment with computer systems. The contexts include data size, execution time, workload, computation cost, memory cost, display cost, network bandwidth, and network latency. Hence, adaptive and context-aware APAs with minimal resource utilization on SMDs are required for implementing optimal distributed application processing in MCC.

6. Conclusion and future directions

In this paper, we have studied, characterized, and categorized several aspects of APAs in MCC. APAs are essential for an efficient application offloading from application on SMD to cloud server due to the impracticality of executing the whole application on SMDs or on cloud servers. The latest version of SMDs is still constrained by the intrinsic limitation while current APAs need to interact with the SMDs’ capabilities such as GPS and camera. APAs enable the elastic applications optimally utilizing the resources available on SMDs while enhancing the performance of the application. We have enumerated several characteristics where the partitioning model of APAs shares similarities with, and are different from, each other — graph-based APAs, LP-based APAs, hybrid APAs, and exceptional APAs.

We have developed taxonomies for MCC to classify the common partitioning approaches and to provide a basis for comparison of APAs. We then reviewed and compared 26 APAs, and categorized them according to the respective taxonomies. In doing so, we have gained an insight into the partitioning models, strategies, and practices that are currently implemented by researchers in MCC. Furthermore, we are also able to discover the shortcomings and identify gaps in the current APAs through the characterization. These indicate that the future directions can be taken by researchers in this area. Thus, this paper not only provides a comprehensive classification framework and serves as a tool for understanding APAs, but also serves as a reference for future works.

In conclusion, APAs are being adopted widely to alleviate the resource limitations for optimal execution of elastic application on SMDs in MCC. Nevertheless, more further works need to be undertaken in terms of lightness, adaptability, and context-awareness. Some APAs can truly become the preferred solution for distributed application processing in MCC. Addressing these problems creates the potential for APAs to evolve and become lightweight and thus, creating the next generation of APAs for enabling mobile application to extract maximum utility out of the volumes of available resources on SMD.

References


Amazon, Amazon Elastic Compute Cloud (EC2); 2013. URL: (http://www.amazon.com/ec2/).

Amazon, Amazon Web Services Elastic Beanstalk; 2013. URL (http://aws.amazon.com/elasticbeanstalk/).


Chu H, Song H, Wong C, Kurakake S, Katagiri M, Roam: a seamless application loading from application on SMD to cloud server.

Chun B-G, Maniatis P. Augmented smartphone applications through clone cloud execution. In: Proceedings of the eighth workshop on hot topics in operating systems (HotOS’09), Monte Verita, Switzerland. USENIX Association Berkeley; 2009. p. 8–16.


Google, Google Apps; 2013. URL: (http://www.google.com/apps/index1.html).

Google, Google Drive; 2013. URL: (http://www.google.com/drive).