Application Partitioning Approaches for Mobile Cloud Computing: Review, Issues and Challenges

Liu Jie Yao, Ejaz Ahmed, Muhammad Shiraz, Abdullah Gani

Abstract

The execution of mobile application using computational cloud resources is the latest augmentation strategy for resources constraint Smart Mobile Devices (SMDs). Computational offloading requires mobile application to be partitioned during the execution of the application on SMDs. Since an optimal partitioning approach promises optimization of energy savings and performance on SMDs, partitioning of mobile application at runtime is a challenging research perspective. This paper reviews existing Application Partitioning Approaches (APAs) from different domains including Mobile Cloud Computing (MCC). This work is driven by the objective to highlight the issues and challenges associated with the existing APAs in dealing with different context, utilizing the local and cloud resource during partitioning, and augmenting the execution of computation intensive mobile application. We proposes thematic taxonomy of current APAs, reviews current partitioning approaches by using thematic taxonomy, and investigates the implications and critical aspects of current partitioning approaches. The commonalities and deviations in such approaches are analysed based on significant parameters such as context-awareness, granularity level, annotation, and partitioning model. Finally, we put forward the open research issues in application partitioning approaches for MCC that remain to be investigated.

I. INTRODUCTION

Smart Mobile devices (SMDs), such as smartphone and tablet PCs, are becoming an essential tool of human life nowadays[1]. For mobile users, SMDs is the most convenient and essential tools that not only used for communication [2] but also for other purposes such as gaming [3] and working [2]. Furthermore, SMDs are totally free from the boundaries of time and place [2]. For example, in the scenario of emergency, an user can quickly edit and improve his works with Google Drive [4] using Galaxy Note when he is in the subway. Besides, mobile users always have high expectation that the same applications can be supported on their SMDs when they are moving around rather than bringing the relatively heavier laptops or sitting in front of an unmoveable powerful desktop PCs [5, 6]. SMDs have limited resources such as memory, network bandwidth, processing power, batter life, and storage capacity [7, 8]. SMDs are unable to compete against desktop with regard to any type of resource, and especially battery life and network capacity. Consequently, Cloud Computing [9-11] has been incorporated with SMDs to leverage the resource constraint via wireless technology such as Wi-Fi, 3G and LTE. Cloud Computing provides cloud resources and services in on-demand basis to SMDs in various model, such as Software as a Service [12-14], Infrastructure as a Service [15, 16], and Platform as a Service [17, 18].

The heterogeneity of the distributed execution environments such as ubiquitous, pervasive, grid and mobile cloud computing, highlight the issues and challenges for the development and optimal execution of the mobile applications. A mobile application designed for a targeted specific environment will not perform optimally when executed on a different or dynamic environment. In order to address this issue, the latest augmentation strategies for resources constraint SMDs is executing the mobile application using computational cloud resources such as Amazon EC2 [19], Microsoft Azure [20] and Google AppEngine [21] as aforementioned. Computation offloading is a common approach to leverage the severity of resource constraints of SMDs by migrating the mobile application entirely [22-24] or partially [25-27] to some resourceful nearby surrogates such as computational clouds [7]. Current mobile applications need intensive interactions with SMDs' capabilities such as GPS and camera. Hence, it is always impractical to offload the entire application from SMDs to computational cloud [28]. Instead, a mobile application needs to be partitioned prior to offloading. In the mechanism of application offloading, partitioning is employed to separate the functionality of mobile applications into distinct partitions that can operate independently in distributed setting. Since an optimal partitioning approach promises optimization of energy savings and performance on mobile devices [5, 25-27, 29-47], partitioning of elastic mobile application at runtime is a challenging research perspective.

In this paper, we clarify and distinguish "application partitioning" as an independent process from "offloading". We also extensively review different partitioning approaches from different domains such as Grid Computing, Pervasive Computing and Distributed Computing. Besides, we examine how these partitioning approaches from other domains can contribute to the partitioning approaches in MCC domains. Further, this paper presents a thematic taxonomy of current partitioning approaches. Based on the proposed taxonomy, we compare the strengths and weaknesses of approaches among each other critically based on the significant parameter such as granularity level, annotation, and partitioning model. Then, from the result of comparison, we identify the key issues and challenges of APAs and list out the remaining work for further investigation.

This paper presents a comprehensive review on APAs. The rest of the paper is organized as follows. We first explain the fundamental concepts of elastic mobile application in section 2. It discusses the concept of cloud computing, mobile cloud computing, offloading and application partitioning. Section 3 describes thematic taxonomy of current APAs, reviews current partitioning approaches on the basis of taxonomy and investigates the implications and critical aspects of current partitioning approaches. Section 4 presents an analysis of current APAs by comparing the commonalities and deviations based on the significant parameters presented in the taxonomy. Section 5 highlights the issues in current APAs and discusses challenges towards adaptive context-aware partitioning for MCC. Finally, we conclude the paper with future work in section 6. Table 1 shows the list of acronyms used in the paper.

Acronyms	Description
0-1LP	Zero-One Linear Programming
3G	Third Generation
AD	Allocation Decision
AN	Annotation
APAs	Application Partitioning Approaches
AT	Analysis Technique
BoBs	Breakable Objects
CC	Cloud Computing
CPU	Central Processing Unit
GM	Graph Model
HGG	Hybrid Granularity Graph
ILP	Integer Linear Programming
I/O	Input Output
LP	Linear Programming
LTE	Long Term Evaluation
MACS	Mobile Augmentation Cloud Services
MAUI	Mobile Assistance Using Infrastructure
MCC	Mobile Cloud Computing
MILP	Mixed Integer Linear Programming
NIST	National Institute of Standards and Technology
OLIE	Offloading Inference Engine
PC	Personal Computer
PDA	Personal Digital Assistant
PG	Partitioning Granularity
PLS	Programming Language Support
QoS	Quality of Service
RAM	Random Access Memory
RCMCPP	Resource Constrained Multi-user Computation Partitioning Problem
SCPP	Single User Computation Partitioning Problem
SMDs	Smart Mobile Devices
Wi-Fi	Wireless Fidelity
WiMax	Worldwide Interoperability for Microwave Access

Table 1. List of Acronyms

II. BACKGROUND

This section elaborates the concept of cloud computing and mobile cloud computing. Further, it explains the mechanism of offloading and application partitioning

A. Cloud Computing

According to Armbrust et al. [9] and Fox et al. [11], cloud computing does not have a commonly agreed upon definition. Vaquero et al. [48] has studied more than 20 definitions to extract essential characteristics of Cloud. Wang et al. [49] defined that "a computing Cloud is a set of network enabled services, providing scalable, QoS guaranteed, normally personalized, inexpensive computing infrastructures on demand, which could be accessed in a simple and pervasive way". Further, NIST has listed five essential characteristics of cloud computing: 1) on-demand self-service, 2) broad network access, 3) resource pooling, 4) rapid elasticity and 5) measured service [50]. In general, cloud computing is defined as a new paradigm of computing in which dynamically scalable and often virtualized resources are provided as pay-as-use services over the Internet. Users use a wide variety of devices, including PCs and laptops to access different kinds of utility programs, storage, and application development platforms over the Internet, via services offered by cloud computing providers [19-21]. Examples of utility programs or Software as a Service (Saas) are Google Apps [51], Microsoft Office 365 [52] and Onlive [53]. Google Apps is similar to Microsoft Office 365 which features several cloud-based applications with similar functionality to traditional office suites, including email, calendar, instant messaging, documents and conferencing. Onlive is a cloud-based platform which offers gaming with the features to synchronized and stored in remote server via internet. Examples of storage or Infrastructure as a Service (IaaS) are Amazon EC2 [19] and Microsoft Azure [20]. Amazon EC2 is the central part of Amazon's cloud computing platform, Amazon Web Services (AWS). EC2 provides virtual machines for users to run their own applications. Users are able to create, launch, and terminate server instances as needed while pay according to the hour for active server. Similarly, Windows Azure is another cloud computing platform used to build, deploy and manage applications through Microsoft's data centres. Windows Azure allows users to build their application using many different programming languages, tools or frameworks. This makes it possible for developers to integrate their public cloud applications into their existing IT environment. Example of application development platforms over Internet or Platform as a Service (Paas) are Microsoft Azure [20], Google App Engine [21], AWS Elastic Beanstalk [54], and Heroku [55]. These cloud providers offer a computing platform which typically includes operating system, programming language execution environment, database, and web server. Developers can develop, deploy and execute their application on cloud without the cost and complexity of buying and managing the underlying hardware and software layers. Computer and storage resources scale dynamically to match application demand such that users do not have to allocate resources manually [21]. Advantages of the cloud computing technology include cost savings, high availability, and easy scalability.

B. Mobile Cloud Computing

Cloud-based applications and services nowadays are mostly non-real-time services where high performance or fast response may be demanded or preferred, yet, there are no strict constraints on response time. Most of the browser-based applications or dedicated applications on smartphones share these attributes. Real-time systems have strict constraints on response times, for example, voice, text, instant messaging and video communication applications. These applications must function within strict constraints regardless of availability or system load. In order to support today's real-time communication application and services, as well as other purposes, cloud must be extended to deliver Mobile Cloud Services and this is how Mobile Cloud Computing emerged [56]. Figure 1 shows the evolution of computing from mainframe computing to mobile cloud computing.

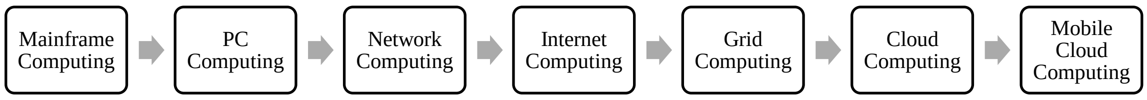

Figure 1. Evolution of Computing

Mobile Cloud Computing (MCC) is a recent practical computing paradigm that extends cloud computing to leverage resources constraints of SMDs [8]. Han et al.

[57] defined MCC as a development of mobile computing, and an extension to cloud computing while Huang et al. [58] argued that MCC cannot be simply illustrated as merging mobile computing and cloud computing technologies. The MCC Forum defines MCC as follows: "MCC at its simplest refers to an infrastructure where both the data storage and the data processing happen outside of the mobile device. Mobile cloud applications move the computing power and data storage away from mobile phones and into the cloud, bringing applications and mobile computing to not just smartphone users but a much broader range of mobile subscribers" [59].

MCC changes the current way of delivering mobile computing and communication services to global mobile users. It is also changing their working and life styles with seamless global mobile resource sharing and accesses. For example, electronic commerce (e-commerce) change the way people trade products or services [60]. E-Commerce on MCC allows buying and selling of products or services over electronic systems such as the Internet or cloud. Another example is mobile learning (m-Learning) which is electronic learning (e-learning) plus mobility. M-Learning enables people learning or studying anywhere and anytime with a portable device or SMDs [61]. There are more and more great, useful and powerful mobile applications such as MHS (mobile healthcare service) system [62], E-Ambulatory Healthcare System [63] and Gmail [64]. Currently, mobile users use SMDs with limited processing power and resources to execute mobile applications [8]. Unlike mobile computing, MCC offers cloud infrastructures and resources to mobile users by delivering various diverse and scalable services model, such as Paas, Iaas, and Saas, to run the mobile application with their relatively low-end SMDs at anytime and anywhere. Researchers believe MCC will bring many exciting new opportunities and enable innovative applications to mobile users, mobile cloud vendors, and businesses [65, 66].

Mobile Cloud Computing is enabled by three major components: SMDs, wireless technology and computational cloud. Figure 2 shows the MCC model. 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) are the enabling technologies of MCC to leverage the resources constraints of SMDs. How does MCC actually augment the capabilities of SMDs and leverage the resources constraints? The answer is outsourcing the computation task to the cloud. The following section will address this issue in details.

C. Computation Offloading

There are many augmentation approaches are developed to alleviate the issues of resources limitations of SMDs [8], such as energy augmentation [26, 27, 30, 33, 34, 37, 39, 40, 45, 67], memory augmentation [30, 35], and application processing augmentation [5, 26, 30, 32, 33, 37, 46, 47, 68]. Generally, augmentation approaches are classified into 2 major categories, namely hardware and software [69]. Outsourcing computation task to cloud or remote execution is a popular software augmentation approaches in MCC for resources constraint of SMDs. The idea of remote execution was introduced in 1998 which aims to conserve mobile device's energy [70]. Then, the concept of remote execution is re-introduced again with the term cyber foraging [71]. In general, the mechanism of outsourcing computational task to remote server is called cyber foraging or offloading [8]. Computation offloading entirely or partially migrate the resource-hungry components of mobile application to nearby resource-rich surrogates such as computational clouds. Computation offloading not only conserves the usage of local resource such as memory, battery, and storage of mobile devices, but also enables execution of computation intensive applications in SMDs. A numbers of researches ascertain the contribution of offloading in augmenting the capabilities of SMDs [11, 26, 27, 33, 72-74].

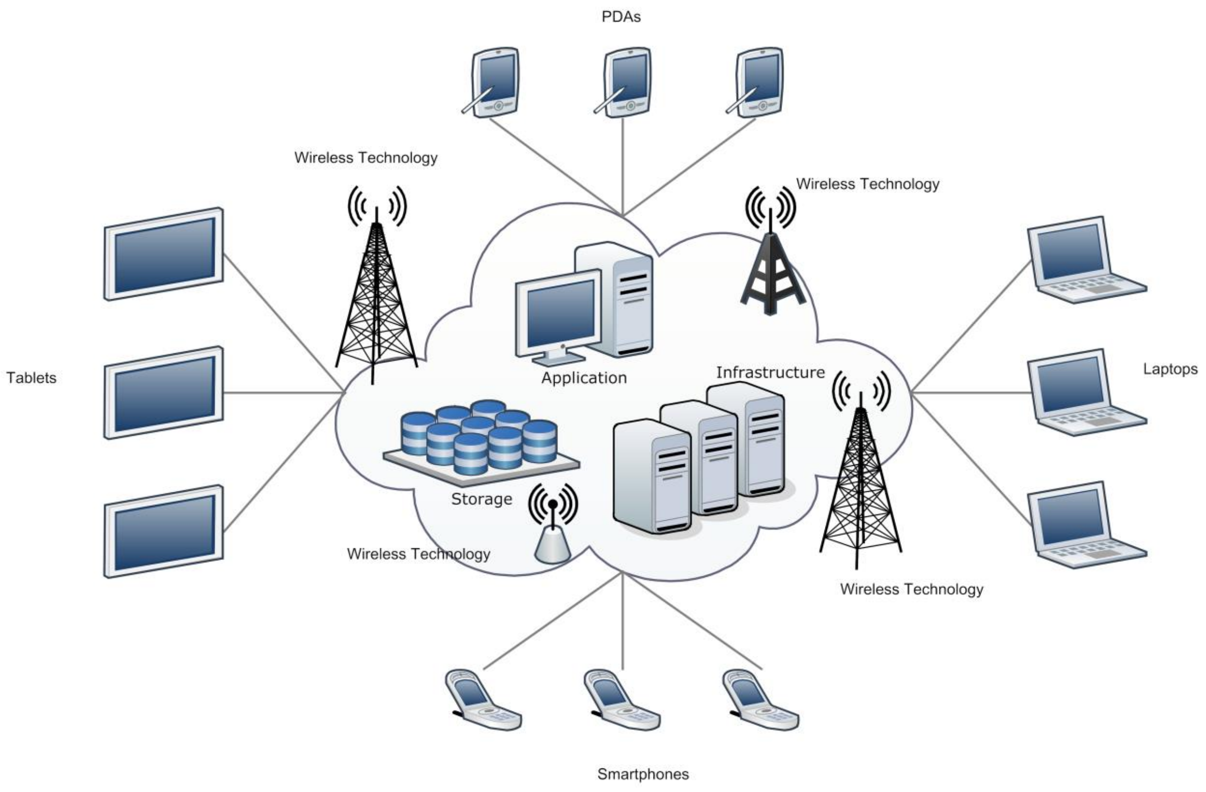

Figure 2. Model of Mobile Cloud Computing

There are many tasks before and after offloading, such as surrogate discovery [75, 76], resource estimation [77-80], and partitioning [25, 26, 42, 43]. These essential tasks of offloading will consume relatively less local resources compared to running entire application locally. However, in some cases, offloading overhead may exceed conserved local resources [81]. It is due to the inefficient execution of pre and post task of offloading including partitioning approach. An optimal partitioning approach promises optimization of computation offloading; therefore, it also promises optimization of energy savings and performance on SMDs.

D. Application Partitioning

Previous works show that remote execution is potential to optimize the energy savings and accelerate the performance for applications running on weak devices [70, 82]. In the mechanism of remote execution, computation offloading employs partitioning to separate the functionality of elastic mobile applications into distinct partitions that can operate independently in distributed setting. Partitioning is the In Veda et al.'s works [83], application partitioning is defined as a technique of splitting up the application into components while preserving the semantics of original application. The original source application may or may not be designed, implemented and deployed to run on a standalone system. However, the resulting components are distributed to take advantage of distributed setting. Thus, we have to clarify that application partitioning is an independent process from offloading. Nevertheless, both application partitioning and computation offloading are parts of the execution framework mechanism. Partitioning applications to take advantage of remote execution was actively researched. For example, MAUI, a popular augmentation approaches, is proposed to reduce developers' burden, improve overall performance, and save energy consumption by automating program partitioning with the combination of code portability, serialization, reflection, and type safety [26]. Application migration [22, 84] and virtual machine migration [24, 85] are another two common approaches other than application partitioning to migrate the execution of a mobile application to the nearby resource-rich surrogates. A mobile application should be featured with the elasticity so that it can be partitioned appropriately.

An elastic mobile application is featured with the ability to be partitioned during run time for the establishment of distributed processing platform. The features of elastic mobile applications are attributed with the following [72, 86, 87]. a) Ad-hoc platform is created by splitting up the distributed platform between SMDs and cloud at run time. Application developers should be able to determine the most appropriate surrogates based on their functionalities and runtime behaviours such as computation demand, data dependency, and communication need, which we believe should be part of high-level design consideration of an elastic application. b) Elastic mobile application should be partitioned into components dynamically at runtime to achieve seamless and transparent migration and offloading. Applications can be partitioned at different granularity level based on the partitioning approaches' policy. c) Partitioning approaches should consider the resources constraints and the application's execution patterns. Partitions of the elastic application are offloaded adaptively to surrogates for remote execution depending upon different objective functions; such as energy saving, processing power, memory storage, and fast execution. d) Elastic mobile application should execute seamlessly and transparently on remote surrogates. All the complexities of remote execution should not being aware from mobile users. This transparent distributed processing environment makes the entire application seems like being executed only in local of SMD. Thus, an elastic mobile application is essential for seamless offloading via application partitioning.

III. APPLICATION PARTITIONING APPROACHES (APAs)

The current APAs are from different domains with commonalities and deviations. This section presents thematic taxonomy for current APAs and reviews the APAs on the basis of partitioning model attributes of the taxonomy. Further, it investigates the advantages and critical aspects of current APAs.

A. Taxonomy of Application Partitioning Approaches

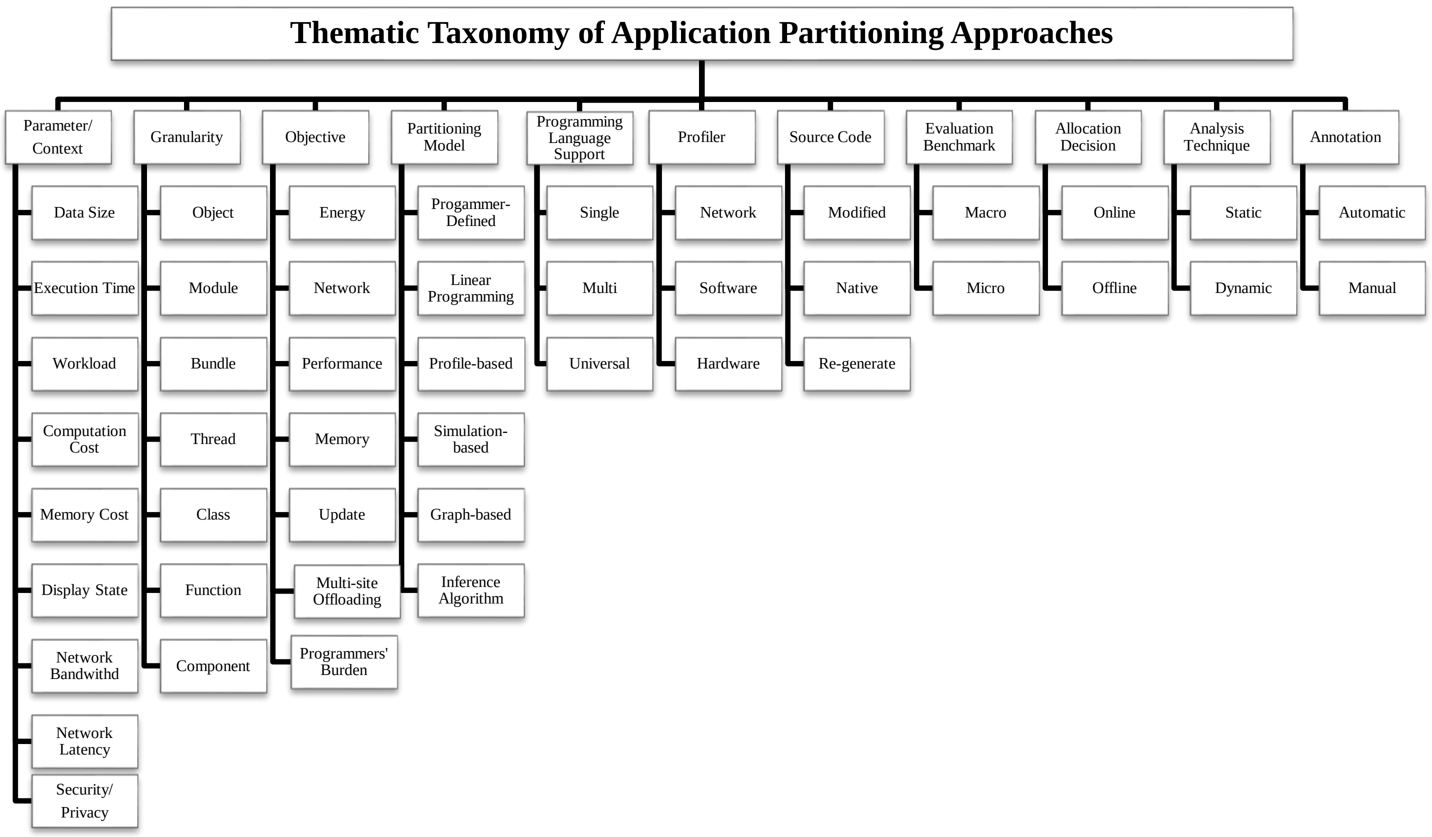

Figure 3. Thematic Taxonomy of Application Partitioning Approaches

Figure 3 shows the thematic taxonomy of APAs, which are categorized according to parameter/context, granularity, partitioning model, objective, programming language support, profiler, source code, evaluation benchmark, allocation decision, analysis technique and annotation. The attribute of partitioning model indicates the main approaches employed for application partitioning. We categorize current APAs on the basis of programmer-defined, integer linear programming, profile-based, simulation-based, graph-based, and inference algorithm. In programmer-defined of APAs, developers or programmers have to examine source code and provide the application-specific information to the frameworks for partitioning [25, 26, 32, 39, 42, 43, 45, 68]. Usually, there are declarative languages such as little language or other specification formats provided for user to annotate the application manually. Linear programming (LP) of APAs is an essential method used for optimizing the graph model of application [26, 33, 37, 41, 44, 46, 47, 67, 68]. LP formulates the application or graph model in linear equation. Then, LP solves the equations based on the constraint defined by developers. LP used in APAs includes Integer Linear Programming (ILP), Zero-One Linear Programming (0-1 LP) and Mixed Integer Linear Programming (MILP). Profile-based of APAs needs a profiler to collect the context-aware information whether on run-time or compile-time. An elastic application needs a profiler to run partitioning. Profiling is very important for estimating the CPU and network requirement for application. Simulation-based of APAs obtains the estimation and calculation of resources needed from result of simulation [40, 68]. Simulation-based approaches derive more accurate results closer to reality. Therefore, applications are able to run optimal partitioning. Graph-based of APAs models the mobile applications in various forms of graph such as consumption graph [27, 34], call graph [26, 42], control flow graph [33], data flow graph [33, 47, 68], and hybrid graph [29]. The elements of graph, namely vertexes and vertices, is used to represent the parameter or context of mobile application, such as granularity, data size, communication overhead, computation cost and memory cost. Graph-based approaches optimize the application partitioning through coarsening by inference algorithm. Inference Algorithm of APAs is typical solution for application partitioning [5, 27, 29-31, 34-36, 38, 39, 41-47]. After modelling the application in graph form and optimizing it with LP, inference algorithm is responsible to partition the graph or application based on the result from LP following the flow of inference algorithm itself. Besides, inference algorithm is used to model the application and optimize the application model.

Current APAs employ two different strategies for analysis and identify closely interacting components so that they can be co-located. Analysis technique involves program analysis to identify dependency relationships between components. The analysis can be static which may be at source level or byte code level. Alternatively, profiling is used to investigate the component interactions and subsequent logs. Allocation decision is the decision of application to allocate the component before the execution of application starts. Offline allocation decision is decided and specified in some configuration file or in the form of annotations within the mobile application. On the other hand, online allocation decision is postponed to run-time. The attribute of programming language support indicates whether the partitioning approach is specifically designed for only one programming language or more. Currently, programming language support is provided for single programming language, multi programming language, or universal to all programming language. Annotation is another essential attributes in APAs. Annotation can be done automatically by using profiling technique. However, some circumstances need developers or programmer to annotate the network-specific, tactics-specific, and application-specific information for partitioning. The attribute of partitioning granularity indicates the granularity level at which application is partitioned. Current APAs partition intensive components of the mobile application at different granularity level such as object, module, class, thread and function. Current APAs employ profiler to collect the information for partitioning purpose. Profiler can collect the information from hardware, software, and network. During the partitioning, mobile application source code may be modified or re-generated. In some approaches, native application source code will be partitioned without any modification on any part of source code. The objective function attribute indicates the primary objective of a framework for application offloading. Current APAs aim for a number of objective functions such as optimizing energy saving on SMD, minimize the network latency, increase the speed of execution, alleviate the memory constraints, or dynamically update the mobile application. In the partitioning of elastic application, parameter or context is a significant attribute to be considered. Current context-aware parameter includes data size, execution time, workload, computation cost, memory cost, display state, network bandwidth, network latency and security or privacy. Evaluation benchmark represents evaluation method to be used to evaluate the performance of partitioning approaches. Mircobenchmarks is useful for understanding component use trade-offs. Microbenchmarks do not capture application or user behaviours while macrobenchmarks measure the overall performance and the scalability of the mobile applications.

B. Review on Application Partitioning Approaches by Using Thematic Taxonomy

A classification of application partitioning approaches by using their attributes is shown in Figure 3. This section analyses current application partitioning approaches and investigates their implications and critical aspects on the basis of different models, namely graph-based model, linear programming-based model, and hybrid-based model.

1) Graph-based model: Michael et al. [42] proposes a novel approach for producing partitioned applications by modelling the application as dependency graph and then automatically partitioning existing applications on the basis of low-effort input from developers and static code analysis. Application developers have to annotate the code units to provide cues to the partitioning algorithm. They need only annotate the code whichever they think there is necessary. For example, in an online payment application, nothing except security-related code unit need to be annotated. This approach defines a set of five annotations, including Mobile and Immobile to provide cues on public cloud versus private cloud. The annotation language used in this approach is Annotation which is Java-based. Annotations benefits developers to annotate from within the source code, thus makes the annotations explicit when developing or maintaining the source code. Similar to Bialek et al.'s work [25], annotations can also be specified in a configuration files, which does not need any language support. However, it is less visible during maintenance tasks. The benefit is that development of the application is not affected and no manual migration effort is required.

Wang et al. [45] presents a parametric program analysis to partition the program where optimal partitioning decision is made based on run-time parameter values during execution. Similar to other graph-based partitioning approaches, it models the application in to task control flow graph. The optimal application partitioning is depends on unknown execution count and allocated data size values, thus, user annotations are required. Similar to worst case execution time (WCET) [88], there is no limitation for user annotations to constant values but annotations is expressed as functions of input parameter vector, whose values vary with different run time parameter values. The critical aspect is that parametric algorithm is useful for determining the necessary user annotations as it treat each unknown as a dummy parameter and then solve it in the normal way. Programmer only requires annotating the dummy parameter that appears in the solution which will affect the program partitioning decision. The user input effort for annotations of this partitioning approach is low.

In other approach which modelling the applications as a component graph, the decision of where to execute the partition result must be made before its execution [39]. It proposes the use of complexity metrics that enhance predictions of application's component's resource usage by taking into account relevant properties of each component's input including general purpose and type-specific. The benefits of this proposed system is that it is 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 intended for training of predictors. Obviously, it does not put a large burden on developers since the metrics can likely be defined on a type-specific basis and shared across many components that accept the same type of data as input. Furthermore, developers already test and debug their applications on test inputs. The critical aspect is that the programmers need to balance metric utility with overhead and specify lightweight metric functions.

Giurgiu et al. [27] develop a middleware that can automatically distribute different layers of an application between the device and the server while optimizing several parameters such as cost and latency. In this approach, there is a distributed module management which automatically and dynamically determines when and which application modules should be partitioned and offloaded. The objective is to achieve the optimal performance or the minimal cost of the overall application during the entire duration of execution. Giurgiu et al. model the application into data flow graph and utilize the AlfredO [89] framework for distribution. The critical aspect is that this partitioning approach builds on existing technology, thus, does not require new infrastructures. However, these solutions do not support platform-independent cooperative interaction over an open network. In addition, after moving some partitions of applications from devices to the cloud, several issues need to be considered in advance such as privacy and provenance. It is also lacks of dynamic adaptation of the computation between mobile devices and the cloud. Further, their solutions only focus on the client side and have assumed the cloud's resources to be infinite which is obviously impractical in real world.

There are also Java program partitioning systems for mobile devices, whose limitation is that only Java classes without native state is placed remotely [35]. The general approach is to partition Java classes into groups using adapted MINCUT heuristic algorithms to minimize the component interactions between partitions. The algorithm partitions a class graph into candidate partition plans according to the edge-weight, and then selects the best partition plan by using a combination metric comparison. Since most of the resource constraints are related to vertex-weights or memory consumption, there is a possibility that Gu's min-cut heuristic may miss some better partitioning solutions in its candidate partition plans. Besides, the decision criterion is based only upon the memory, not considering multiple factors. This approach does not consider partitioning constraints like CloneCloud [33], the granularity of partitioning is coarse since it is at class level, and it focuses on static partitioning.

Adaptive Offloading techniques [5] works on adaptation for sequential applications specifically targeting mobile environments. Ou et al. have used a multi-cost graph model to partition the application and select an execution location for every part. This approach not only partition applications according to the required memory of each part and available memory on every location, it also take into account CPU and bandwidth constraints, too. This runtime partitioning approach generally assumes the existence of a single mobile device and a dedicated unconstrained surrogate node to serve as an offloading target. This approach was shown to provide both better performance and efficacious adaptation. However, computing an adaptation decision is expensive. Further, it is not scalable to the size of the application graph, thus, unsuitable for applications with a large number of components. Besides, using class level graph model yields object topologies of relatively lesser efficacy which consequently incur more resource usage. Consequently, it causes performance degradation as a result of the heavy edges and vertices maintained by such application graphs. Furthermore, some partitions of applications are not transferable to the surrogate, such as codes that interact with I/O devices.

Giurgiu et al. [34] propose an approach that optimally partitions a modular application on-the-fly between the cloud and the mobile device according to the device's CPU load, network conditions, or user inputs. The critical aspect of this approach is that it tends to make cloud applications not originally designed for mobile platforms capable of running on mobile devices in a resource-efficient manner. Its partitioning approach considers an application's structure, resource requirements and device constraints via profiler and model the application into consumption graph to identify its best mobile-cloud partition and adjust it online. Further, it is able to reconfigure the current application deployment without interrupting ongoing interaction. It also allows clients to autonomously decide which configuration to adopt once the user inputs have been submitted to the application. However, it needs the application applications are modularized, namely loose coupling and high functionality cohesion as not all existing application is well-modularized. Besides, this partitioning approach demands prior accurate measurements of the application execution and data transfers for each specific setup.

A Hybrid Granularity Graph (HGG) for representing an application’s runtime is proposed by Abebe et al. [29] to reduce the network overhead. In this approach, instead of mapping to one runtime component, namely object or class, a vertex rather mapping to a configurable subset of objects of a given class. It also implement byte code injection for profiler to collect information of performance cost and memory utilization as well as runtime coupling patterns, which were used to construct runtime graphs. The critical aspect is that this approach can provide finer level adaptation and more efficacious object topologies without incurring the computational overheads of an object-level graph [5, 35]. HGG offers finer granularity than a class graph does which results in more flexibility. Further, HGG remains smaller than object graph and consequently more computationally feasible.

Another novel distributed graph representation, namely abstract class graph, is proposed by Abebe et al. [30]. In this graph model, devices maintain graph vertices only for components within their memory space, and abstraction vertices called cloud-vertices for components in remote devices. Abebe et al. also propose a novel graph partitioning heuristic, namely multilevel graph partitioning algorithm, to reduce network, power, and memory utilization as well as the performance cost of adaptive offloading. The critical aspect is that this approach allows the partition result on the cloud to be on-loaded back to the client device for utility purpose. Additionally, the efficacy of the generated partitions is also improved as remote object coupling and migration costs are reduced. The issue here is that it is not yet tested in real world applications.

The idea of breakable objects (BoBs) for compile time application partitioning in Java is introduced by Jamwal et al. [36]. BoBs are the entities in an application that can be split easily. This paper present an automated refactoring process that makes application components should be amendable to partitioning. Jamwal et al. introduce meta-languages to annotate the source code of applications before partitioning. A pre-compiler then generates a language specific code for both local and remote partners. After the elements are loaded, the execution proceeds to perform local and remote calls according to the prescribed definition. The critical aspect of this approach is that it alleviates the challenges of inter-operability and context-awareness. However, it requires programmers to modify the source code of applications which is not always available. Besides, programmers need knowledge about the applications to determine where it is necessary to perform the changes for the partitioning.

Graph-based of APAs models the mobile applications in various forms of graph such as consumption graph, call graph, control flow graph, data flow graph, and hybrid graph. The elements of graph, namely vertexes and vertices, is used to represent the parameter or context of mobile application, such as granularity, data size, communication overhead, computation cost and memory cost. Graph-based approaches optimize the application partitioning through coarsening by inference algorithm. The following section describes the generic sequence of operations for graph-based model application partitioning. First step is to check whether annotation is needed. If yes, it may need the manual annotation from programmer. If no, it will continue to check whether the profiler is available in the application. If yes, the profiler will gather the relevant information needed by the application. If no, it will go on to next step for graph modelling. Annotation by programmer and profiling result may be used during graph modelling. After that, one or more algorithms are implemented to optimize the graph model. Finally, the result of optimization is provided to inference algorithm such as solver to decide and perform the partitioning. Figure 4a shows abstract level flowchart of graph-based model application partitioning.

2) Linear programming-based model: Yang et al. [46] consider, for the first time, the computation partitioning problem from the application provider's standpoint by taking into account the constraint on the cloud resources. They target the partitioning of interactive mobile cloud applications which always take input from sensor of SMDs. First, they describe about single user computation partitioning problem (SCPP) and then they derive the resource constrained multi-user computation partitioning problem (RCMCPP) from SCPP. They formulate RCMPP as a MILP problem which aim to minimize the total execution time. The critical aspect is that the proposed partitioning algorithms are beneficial for the application provider to provide optimal application performance when faced with unpredictable number of users. Also, application providers can apply the Performance-Resource-Load model obtained by algorithms in the design of resource provisioning mechanism, with the objective to minimize their operational cost.

Mobile Augmentation Cloud Services (MACS) [37] middleware is proposed by Kovachev et al. in order to reduce local execution time and then reduce battery power consumption. MACS enable adaptive extension of Android application execution from a mobile client into the cloud. The critical aspect is that the middleware is responsible for the heavy lifting of adaptive application partitioning, resource monitoring and computation offloading. These elastic mobile applications can run as usual mobile application, but they can also use remote computing resources transparently. MACS also support offloading of multiple Android services in one application through the allocation determination from resource monitor. MACS's partitioning scheme is dynamic and lightweight as MACS do extra profiling and resource monitoring for adaptive partitioning decision during run time.

Ra et al. [40] suggests that the pre-determining a library of methods for the low processor (LP) is the most energy efficient one in practice. Besides, it also offers additional advantages in terms of ease of use for developers. Unlike aforementioned approaches, this approach investigates the partitioning of applications across the two processors from an energy efficiency perspective. Ra et al. provide a methodology to analyse the application components to determine the most efficient placement. The critical aspect is that partitioning guidelines as well as proper runtime design principles are provided to enable continuous sensing applications. It uses simulation-based approach to derive more accurate results closer to reality for application partitioning. Even though additional tasks is placed on the LP, there is need of careful dynamic job scheduling based on accurate resource monitoring for LP at runtime for such tasks.

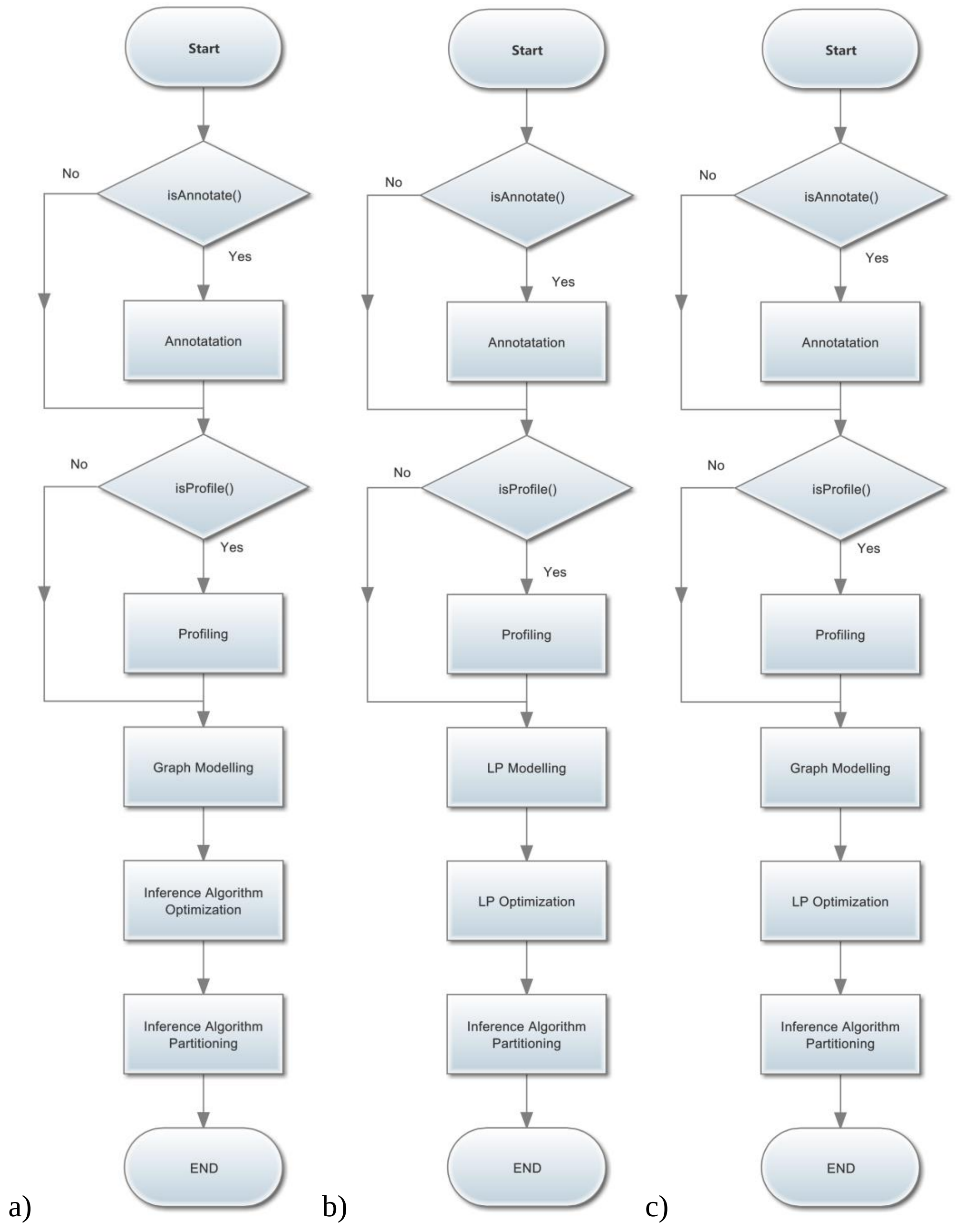

Figure 4. Flowchart Diagram for a) Graph-based b) LP-based c) Hybrid-based

Linear Programming (LP) can model and formulate the application as a mathematical optimization problem where some or all of the variables are restricted to be integers. In most cases, LP will be used to optimize the formulated problems which represent the mobile application. The following section describes the generic sequence of operations for LP-based model application partitioning. Similar to graph-based model application partitioning, the first step is to check whether annotation is needed. If yes, it may need programmer to specify the annotation manually. If no, it will continue to check whether the profiler function is implemented in the application. If yes, the profiler will gather the relevant information which requested the application. If no, it will go on to next step for formulating the LP problem. Annotation by programmer and profiling result may be useful in assisting formulation of LP problem. After that, linear programming techniques, namely, 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 result of optimization is provided to inference algorithm to decide and perform the partitioning. Figure 4b shows abstract level flowchart of LP-based model application partitioning.

3) *Hybrid-based model:* MAUI [26] is a system which provides fine-grained code offloading to optimize energy savings with minimal burden on the programmer. MAUI provides a programming environment where enabling programmers to produce an initial partitioning of application via annotating which methods of application is offloaded for remote execution. The programmers require annotating methods and classes as remoteable which indicate that the MAUI runtime should consider offloading to a remote server. Therefore, programmers do not need to guess about whether or not a particular method makes sense to offload. Annotating as local may lead to fewer annotations, yet, it favours performance over application correctness. Instead, mistakes suck as carelessness of labelling in MAUI approach only affect the program's performance and not its correctness. MAUI promise great reliability.

CloneCloud [33] tries to analysis a given process of application whose execution on the cloud. It would make the overall process execution faster. It examines the process with an offline static analysis of different running conditions of the process binary on both a target smartphone and the cloud. Then, result of analysis is used to build a database which storing pre-computed partitions of the binary. This database is used to determine which parts should be offloaded to the cloud. The critical aspect is that it proves the effectiveness of static analysis of Java code to dynamically partition applications. However, CloneCloud only considers limited input/environmental conditions in the offline pre-processing. Further, it needs to run the analysis again for every new application built. Since the concept of function reusability and loose coupling are not considered, software composition is not addressed. Besides, it is not easy for layman users to customize application functionality as the application needs to be programmatically modified.

An energy-optimal software partitioning scheme for heterogeneous multiprocessor systems is presented by Goraczko et al. [67] incorporating a resource model considering the time and energy overhead of run-time mode switching. It optimizes the software partitioning at compile-time by formulating it as an Integer Linear Programming (ILP) problem while addressing the problem with task dependencies. This approach only targets platforms are loosely coupled multiprocessor systems. It ignores the communication latency and energy. In addition, it is an inefficient heuristic for fast computation of task mapping.

Wishbone [68] uses a profile-based approach to partition applications, which specified as a data-flow graph of operators, in order to execute on multiple and heterogeneous devices in a sensor network. Wishbone is primarily concerned with high-rate data processing applications. It tends to statically minimize a combination of network bandwidth and CPU load during compile time by solving an integer linear programming problem. On the other hand, WaveScript is used by Wishbone to describe the interconnection of application components, which is implemented in a lower-level, device-specific language. However, such programming model is not robust enough for the dynamics of sensor network environments, regardless of efficient use of the limited network resources. Besides, there is no provisioning for flexible re-engineering of the sensor application at runtime if there is any failures such as the service model is not adapted to the ever-changing processing and energy resources of the nodes. Also, Wishbone does not take account on battery level of the device which is not favorable by users.

Yang et al. [47] focus on the partitioning of data stream application, and use a dataflow graph to model the application. The genetic algorithm is used to maximize the throughput of the application. Different from existing works, this framework not only allows the dynamic partitioning for a single user but also supports the sharing of computation instances among multiple users in the cloud to achieve efficient utilization of the underlying cloud resources. Meanwhile, the framework has better scalability because it is designed on the elastic cloud fabrics. The critical aspect of this approach is that it can run the partitioning algorithm on the cloud side. In addition, the powerful computing resource at cloud side ensures that the genetic algorithm is more likely to converge to the global optimal partition. However, this partitioning approach is not suitable for the batch computation system where the analysis of large data sets is necessary. However, it will cost high overhead of cloud resources while increasing the application provider's operational.

Sinha et al. [41] argue that data-centric offloading applications, such as image-matching application, require that an application be partitioned across multiple locations, namely, the mobile device, which must contain at least the user-facing modules such as the user interface, and one or more servers, which is used for computation offloading or to co-locate computation with data in order to reduce communication costs. This paper presents a multi-site, fine-grained partitioning approach. The critical aspect is that it allows finer offloading decision to be made for multi-site data-centric offloading as the partitioning granularity is on allocation-site level. However, it would force all such objects to be offloaded to the same site, even though they might be part of different data structures which cost unnecessary overhead to the partitioning approach.

The following section describes the generic sequence of operations for hybrid-based model application partitioning. Annotation function will be identified whether is needed or not. If yes, programmer have provide the specification which contain the annotation and information related to partitioning. If no, it will continue to check whether there is profiler in the application. If yes, the profiler will gather the relevant information which is needed to carry out the application partitioning. If no, it will go on to next step for formulating application optimization problem into LP equation. Annotation by programmer and profiling result assist the formulation of LP equation. After that, linear programming techniques, namely, 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 result of optimization is provided to inference algorithm to decide and perform the partitioning. Figure 4c shows abstract level flowchart of hybrid-based model application partitioning.

4) Execption: In [90], Balan et al. proposed a mobile-computing-specific cyber-foraging solution with an adaptive runtime system based on the well-known approach of little languages [91], Vivendi. Vivendi is useful for expressing application-specific information such as tactics and fidelities. Programmers first need to examine the source code of application and creates a Vivendi file called “tactics file.” The tactics file contains the function prototype of each procedure which is potential for remote execution, and specifies how these procedures is combined to produce a partition result. The critical aspect of such approach is that programmers or developers need to create a tactics file once for each application. They also have to spend a lot of time to find relevant fidelities and thus increase the burdens of them. Even though finding fidelities could be easy, checklist is provided for checking and ensuring to complete all necessary steps of this approach. However, it is unnecessary to make any changes when deploying same application in a new mobile device.

Roam [32] offers a novel application framework for developers to build resource-aware seamless applications which are capable to adapt to heterogeneous devices. During designing application, developers have to decide on how to partition particularly an application into separate components. Also, it is essential for developers to annotate each component, whether to provide multiple device-dependent implementations (M-DD), a single device-dependent implementation (S-DD), or a single device-independent presentation (S-DI). Existing SGUI toolkit for transforming S-DI components is difficult to customize a device-independent representation for a particular device. Device-specific customization needs developers to add both device-specific and application-specific transformation rules. Once the developers change the device-independent model, they may also require updating these transformation rules that are affected by the change.

Bialek et al. [25] propose a novel approach to creating dynamically updatable Java applications based on the concept of partitioning applications of J-Orchestra[43] into units of dynamic updates. A fully developed application is partitioned using the provided partitioning specification. The specification includes information about number of partitions to create and what classes each partition includes. Application developers are responsible to define the specification in a single configuration file which listing all the partitions and the modules that belong to them. Each partition needs to be labelled with unique names with partition name, version number and the list of classes that should belong to the partition. This approach enables transparent partitioning and dynamic update of applications. It simplifies application updates as well as allows to optimize the application's and update's performance through critical choosing of partitions.

J-Orchestra [43] is a system provides automatic partitioning for Java programs. J-Orchestra takes Java applications as input in bytecode format and then partitions them into distributed applications, executing on distinct Java Virtual Machines. J-Orchestra uses bytecode rewriting to replace method calls with remote method calls as well as direct object references with proxy references. It demands low-effort input from programmers such that they only have to specify the network location of various hardware and software resources and their corresponding application classes. Since an annotation mistakes may yield an inefficient or incorrect distributed application, J-Orchestra provides two tools, namely a profiler and a classifier, in order to ensure a correct and efficient partitioning. In comparison with previous work on automatic partitioning, J-Orchestra has great advantages on generality, flexibility, and degree of automation.

IV. COMPARIONS OF APPLICATION PARTITIONING APPROACHES BY USING THEMATIC TAXONOMY

This section presents comparison of current application partitioning approaches on the basis of taxonomy shown in Figure 3. It investigates the commonalities and deviations in the current partitioning approaches using the thematic taxonomy. The comparison parameters considered are: Partitioning Granularity (PG), Graph Model (GM), Programming Language Support (PLS), Allocation Decision (AD), Analysis Technique (AT), Annotation (AN), Partitioning Objective, and Profiler.

A. Significance of Comparison Parameters

The significance of comparison parameters are clarified and justified in this subsection. The first parameter is Partitioning Granularity (PG) with attributes such as object, class, component, bundle, thread, method and task. PG indicates the level of granularity of partitioning of mobile application. A finer level of PG is optimal, yet, it requires highly intensive monitoring mechanism on SMD during execution. The second parameter is Graph Model (GM). Its attributes include various type of graph such as data flow graph, hybrid granularity graph, class graph, multi-cost graph, internal dependency graph, consumption graph, task control flow graph, object interaction graph, control flow graph, call graph, dependency graph, component graph, and task graph. Each type of GMs represents mobile application's components or capabilities such as execution states, cost models, internal dependency, data flow, and also control flow. An appropriate GM results in improving the efficiency of partitioning decision regardless of applying fine-grained or coarse-grained granularity. The third parameter is Programming Language Support (PLS). PLS indicates whether a partitioning approach supporting a single language (specific to particular programming language), multiple languages (supporting more than one but not all), or universal to any language (applied in any programming language). Some approaches maybe optimal while they may be restricted to a particular programming language such as JAVA. A universal PLS is highly demanded due to the heterogeneity of the distributed execution environments nowadays especially different platform such as iOS which using Objective C programming language and Android which using Java programming language. The next parameter is Allocation Decision (AD). AD can be online or offline. Online AD indicates that it make the partitioning decision during run time which results in relatively optimal solution. However, online AD will incur intensive overhead during execution which also spending battery life of SMD faster. Offline AD, on the other hand, incurs lower overhead of SMDs while produces less optimal partitioning decisions. Besides, Analysis Technique (AT) is also another crucial parameter for comparison. AT involves program analysis to identify dependency relationships between components in mobile application. Similar to AD, AT can be static or dynamic. A static AT is run at source level or byte code level by analysis tools. Static AT will not incur intensive overhead on SMDs while demand input from programmers with different level of effort. Alternatively, profiling can be used as dynamic AT to investigate the component interactions and subsequent logs. Dynamic AT does not require or require very less effort from programmers while analyse the mobile application more thoroughly in order to obtain more information which is essential for optimal partitioning. Annotation (AN) is the next important comparison parameter. An automatic annotation is done by the execution framework itself by employing the profiler to collect the relevant information, then, annotate the relevant component in application as the indication of availability of partitioning. Automatic annotation greatly reduces the burdens of programmers as it let the execution frameworks do the job. Seamless application demands automatic AN, but it will also increase the workload of SMDs which cause an issue to battery life. Manual annotation is done by programmers or developers by examining the source code before execution. Although manual annotation demand relatively higher level of programmers' effort, it is usually good for energy saving while resulting in optimal partitioning. Then, we also consider Partitioning Objective for comparison. There are different objectives is considered for current partitioning approaches including maximizing throughput, reducing network overhead, saving energy, increasing performance, reducing memory constraint, reducing programmers' burden, updating mobile application, offloading of multisite, reducing CPU workload, and reducing latency. Currently, most of the partitioning approaches concern for one or two partitioning objectives. There is no perfect partitioning approach which covers all aforementioned partitioning objectives. If a partitioning approaches focus on a single particular objective, it will cause another issue emerged, thus, it will not be an optimal solution in real world application. Profiler is another important parameter for comparison. Profiler collects various type of information from hardware (CPU, RAM, battery life), software (accessed data size, modular structure, application behaviour, performance cost, code size), and network (wireless connectivity, bandwidth). These information enable execution framework decide optimal partitioning decision.

B. Comparison of Partitioning Approaches

Approaches	PG	GM	PLS	AD	AT	AN
Data Stream Application [47]	component	data flow	multiple	online	dynamic	automatic
Hybrid Granularity Graph [29]	hybrid	hybrid granularity	single	online	static	automatic
Distributed Abstract Class Graph [30]	class	class	single	offline	dynamic	N/A
OLIE [35]	class	class	single	online	dynamic	manual
Adaptive Multi-Constraint Partitioning [5]	class	multi-cost	single	online	dynamic	N/A
Automated Refactoring [36]	hybrid	Internal dependency	single	offline	dynamic	manual
Dynamic Software Deployment [34]	bundle	consumption	single	hybrid	dynamic	automatic
J-Orchestra [43]	object	N/A	single	offline	dynamic	manual
Parametric Analysis [45]	task	task control flow	universal	online	dynamic	manual
RCMPP [46]	module	N/A	N/A	online	dynamic	automatic
Multi-site Computation Offloading [41]	allocation-site	object interaction	single	online	static	automatic
Calling the Cloud [27]	bundle	data flow	multiple	hybrid	dynamic	automatic
CloneCloud [33]	thread	control flow	multiple	online	static	automatic
MACS [37]	bundle	N/A	single	online	dynamic	automatic
Improving Energy Efficiency [40]	component	N/A	universal	offline	dynamic	manual
MAUI [26]	method	call	single	online	dynamic	manual
Partitioning Application [42]	component	dependency	multiple	offline	static	manual
Dynamic Updates [25]	component	N/A	single	offline	dynamic	manual
Roam [32]	component	N/A	single	online	static	manual
Complexity Prediction [39]	component	N/A	single	online	static	manual
Simplifying Cyber Foraging [90]	module	task	universal	offline	static	manual
Energy-Optimal Software Partitioning [67]	task	task	N/A	offline	dynamic	automatic
Wishbone [68]	thread	data flow	multiple	offline	static	manual

Table 2. General Comparison of Application Partitioning Approaches

Table 2 show the general comparison of partitioning approaches. Partitioning Granularity (PG) indicates the granularity level of partitioning for computational intensive mobile application. Refined level granularity requires highly intensive monitoring mechanism on SMD at runtime as well as intensive synchronization mechanism between SMD and remote servers. Issue of consistency is another great concern for finer level of PG in the distributed execution of mobile application. On the other hand, the coarser level of PG results in simple offloading mechanism. Nevertheless, coarser level of PG causes increasing in data transmission overhead between SMD and remote servers. Current APAs implement the following granularity levels for application partitioning. a) Module level partitioning indicates that entire module of the application is partitioned and distributed [46, 90]. b) Method level partitioning indicates that partitioning occurs at the method of application[26]. c) Object level partitioning indicates that entire object of application is partitioned for the preparation of cyber foraging [43]. d) Thread level partitioning indicates partitioning occurs at threads of application [33, 68]. e) Class level partitioning indicates application is partitioned into classes for offloading [5, 30, 35]. f) Task level partitioning indicates the application is partitioned according to task [45, 67]. g) Component level partitioning indicates the partitioning a group of classes for outsourcing to the remote server [25, 32, 39, 40, 42, 47]. h) Bundle level partitioning indicates groups of Java class of application are partitioned [27, 34, 37]. i) Allocation-site level partitioning indicates the partitioning occurs on the level of allocation site where all the objects in this particular site will be seen as a single unit [41]. j) Hybrid level partitioning indicates results of partitioning consist of different granularity [29, 36].

Graph Model (GM) represents the application as graph so that the partitioning problem is solved using graph-based approaches. Different GM models the application in different way. Generally, the elements of graph, namely vertexes and vertices, is used to represent the parameter or context of application, such as available resources, data size, communication overhead, computation cost and memory cost. Fine granularity adaptation always produces larger number of vertices with more complex interaction patterns than class graphs. Partitioning an application graph into a number of disjoint partitions is the crucial step to identify the list of classes that are going to be offloaded to remote servers. Basically, each partition must satisfy 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 approaches is an NP-Complete problem [92]. Current APAs implement the following GM for application partitioning: a) Class graph [27, 35]. b) Data flow graph [27, 47, 68]. c) Hybrid granularity graph [29]. d) Multi-cost graph [5]. e) Internal dependency graph [36]. f) Consumption graph [34]. g) Object interaction graph [41]. h) Task control flow graph [45]. i) Control flow graph [33]. j) Call graph [26]. k) Dependency graph [42]. l) Task graph [67, 90]. However, there are partitioning approaches do not model the applications as a graph [25, 32, 37, 39, 40, 43, 46].

Programming Language Support (PLS) indicates the type of programming language supported by current APAs. Existing APAs' PLS are as following. a) Single PLS indicates a partitioning approach is restricted to a particular programming environment, usually is Java [5, 25, 29, 30, 32, 34-37, 39, 41, 43] and .Net [26]. b) Multiple PLS indicates a partitioning approach can support more than one programming language. Data stream application [47] supports flow-based programming which includes C++, C# and Java; calling the cloud [27] support OSGi and Java; CloneCloud [33] supports virtual machine programming environment including Java VM, DalvikVM, and Microsoft .Net; partitioning application [42] supports Java and PHP. c) For Universal PLS, we consider a partitioning approach which is able to support almost all majority programming languages. Parametric analysis [45] support GCC programming which includes C, C++, Objective-C, Objective-C++, Fortran, Java, Ada, and Go; improving energy efficiency [40] supports major mobile phone platform such as Windows Phone OS, Android and iOS;

Simplifying Cyber Foraging [90] supports C, C++, Java, Tcl/Tk and Ada. However, we could not identify the programming environment for RCMCPP[46] and energy-optimal software partitioning [67]. As the technology advances on double, there must be more and more new programming languages are created, thus, programmers demands a partitioning approach which is not restricted to a single programming environment only.

Allocation Decision (AD) indicates that the decision making of partitioning approaches to allocate component in local or remote server respectively. AD can be a) online [5, 26, 29, 32, 33, 35, 37, 39, 41, 45-47], which the decision is make during runtime, b) offline [25, 30, 36, 40, 42, 43, 67, 68, 90], which the decision is make before execution, or c) hybrid [27, 34], which part of the decision is done by programmer-defined specification or static analysis tool and part of the decision is done during application execution. AD during runtime surely results in relatively optimal solution compared to offline AD. However, online AD will incur intensive overhead during execution which also spending battery life of SMD faster. Offline AD, on the other hand, incurs lower overhead of SMDs while produces less optimal partitioning decisions. To address this issue, hybrid AD is the solution to balance the pros and cons of online/offline AD. For example,

Analysis Technique (AT) represents a form program analysis to identify dependency relationships between components in mobile application. AT can be a) static [29, 32, 33, 39, 41, 42, 68, 90], which is run at source level or byte code level by analysis tools, or b) dynamic [5, 25-27, 30, 34-37, 40, 43, 45-47, 67], which profiles and studies the running application's component interaction and subsequent logs. Static AT is an easy-to-use method as it can use the existing libraries for source code analysis. Source code level analysis is easier to understand and run whereas byte code level analysis can involve analysing system classes which are technically hard to understand. Static AT will not incur intensive overhead on SMDs while demand input from programmers with different level of effort. On the other hand, dynamic AT does not require or require very less effort from programmers for analysing the mobile application. Compared to static AT, dynamic AT does more thorough analysis in order to obtain more information which is essential for optimal partitioning decision.

Annotation (AN) is a form of syntactic metadata that is added to application source code. Programmers can annotate any granularity level of components of application such as object, classes, methods, variables, parameters and modules. These annotations indicate which part of application should be partitioned when generated by the compiler during execution. Annotation can be a) automatic [27, 29, 33, 34, 37, 41, 46, 47, 67], which is done by the execution framework itself by employing the profiler to collect the relevant information, then, annotate the relevant component in application as the indication of availability of partitioning, or b) manual [25, 26, 32, 35, 36, 39, 40, 42, 43, 45, 68, 90], which is done by programmers or developers by examining the source code before execution. Automatic annotation greatly reduces the burdens of programmers as it let the execution frameworks do the job. Seamless application demands automatic annotation but it will also increase the workload of SMDs which cause an issue to battery life. Although manual annotation demand relatively higher level of programmers' effort, it is usually good for energy saving while resulting in optimal partitioning in acceptable level. Obviously, annotation brings advantages to optimal partitioning of application, yet, there are partitioning approaches without any annotation [5, 30].

Table 3 presents the comparisons of currents APAs on the basis of partitioning objectives. There are many previous partitioning approaches target on one particular partitioning objective [5, 25, 27, 29, 35, 36, 39-43, 45-47, 67, 90]. However, as the available technology advances, partitioning approaches tend to be more adaptive and context-aware. Thus, there are partitioning approaches want to achieve more than one single objective in their execution framework [26, 30, 32-34, 37, 68]. The common partitioning objectives are as follow. a) Reducing network overhead indicates partitioning approaches will utilize low overhead of network for computation offloading [29, 30, 34, 68]. b) Energy saving is always a great concern of partitioning approaches to reduce the power consumption while prolong the battery life [26, 27, 30, 33, 34, 37, 39, 40, 45, 67]. c) Improving performance indicates partitioning approaches tend to increase its efficiency and effectiveness for particular measurement(s). These measurements for performance improvement includes throughput [47], adaption's time and efficacy [30], algorithm's efficiency and cost effectiveness [5], latency [26, 32, 46], execution time [34, 37], as well as CPU workload [68]. d) Reducing memory constraint indicates partitioning approaches tend to alleviate the memory restriction on SMDs [30, 35]. e) Reducing programmers' burden concerns on low input effort of programmers in developing and partitioning application [26, 32, 33, 36, 42, 43, 90]. f)Updating application dynamically indicates the application is partitioned and updated without shutting down the application and remote servers [25]. g) Multi-site offloading indicates the application is partitioned and distributed among sever remote servers [41].

Table 4 shows the comparison of current APAs on their profiler. As aforementioned, profiling is a form of dynamic program analysis that measures, for example, the memory or time complexity of an application, the usage of particular instructions, or frequency and duration of function calls. Profiling information is common to be used in optimizing the 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. We categorized the profiler in to three main categories, namely, hardware, software, and network. The following will compare partitioning approaches among themselves on the basis of profiler's categories. a) Hardware profiler collects the information relevant to physical hardware or SMDs such as CPU, RAM, and battery life. Dynamic software deployment [34], CloneCloud [33] and Wishbone [68] focus on collecting CPU information as they would like to make their approaches lightweight. Energy saving is given the top priority in [26], [67] and [37] as they always instrumenting the application battery's life. Memory status is profiled for these approaches [5, 27, 29]. RCMCPP [46] collect all the information about the device or SMDs including CPU, RAM and battery life. b) Software profiler is responsible to collect the application information, namely, accessed data size (send size, receive size, and transfer size), interdependency between modular structure, application behaviour, performance cost (execution time, throughput, and latency), and code size. OLIE [35] and MAUI [26] profiling the program behaviour into their execution framework. Size of accessed data and code size is profiled in the partitioning approaches of [5, 27, 47]. Dynamic software deployment [34] and J-Orchestra [43] concern about the interdependency of classes and the modular structure of application. HGG [29], on the other hand, profiles the performance cost for optimal partitioning decision. c) Network profiler focuses on information gathering of network environment condition, for example, Wi-Fi/3G/4G/Wi-Max connectivity, bandwidth, and transferred rate. From studied approaches [5, 26, 46, 47, 68], we observed that bandwidth condition is essential for network profiler including CloneCloud [33] which also observes transfer rate between SMDs and clouds, OLIE [35] which also observes the network latency, as well as dynamic software deployment [34] identify the type of network. However, there are partitioning approaches which do not implement profiler in their execution framework [25, 30, 32, 36, 39-42, 45, 90]. Most of the studied partitioning approaches which implementing profiler to gather information from at least two categories among hardware with software [27, 29, 37, 67], hardware with network [33, 46, 68], or software with network [35]. There are also profilers of partitioning approaches collect the information from all of the three categories [5, 26, 34]. Only software profiler is implemented J-Ochestra [43] and only network profiler is implemented in Data Stream Application [47].

Approaches	**reducing network overhead**	**saving energy**	**improving performance**	**reducing memory constraint**	**reducing programmers' burden**	**updating application dynamically**	**multisite offloading**
Data Stream Application [47]	No	No	Yes	No	No	No	No
Hybrid Granularity Graph [29]	Yes	No	No	No	No	No	No
Distributed Abstract Class Graph [30]	Yes	Yes	Yes	Yes	No	No	No
OLIE [35]	No	No	No	Yes	No	No	No
Adaptive Multi-Constraint Partitioning [5]	No	No	Yes	No	No	No	No
Automated Refactoring [36]	No	No	No	No	Yes	No	No
Dynamic Software Deployment [34]	Yes	Yes	No	No	No	No	No
J-Orchestra [43]	No	No	No	No	Yes	No	No
Parametric Analysis [45]	No	Yes	No	No	No	No	No
RCMPP [46]	No	No	Yes	No	No	No	No
Multi-site Computation Offloading [41]	No	No	No	No	No	No	Yes
Calling the Cloud [27]	No	Yes	No	No	No	No	No
CloneCloud [33]	No	Yes	Yes	No	Yes	No	No
MACS [37]	No	Yes	Yes	No	No	No	No
Improving Energy Efficiency [40]	No	Yes	No	No	No	No	No
MAUI [26]	No	Yes	Yes	No	Yes	No	No
Partitioning Application [42]	No	No	No	No	Yes	No	No
Dynamic Updates [25]	No	No	No	No	No	Yes	No
Roam [32]	No	No	Yes	No	Yes	No	No
Complexity Prediction [39]	No	Yes	No	No	No	No	No
Simplifying Cyber Foraging [90]	No	No	No	No	Yes	No	No
Energy-Optimal Software Partitioning [67]	No	Yes	No	No	No	No	No
Wishbone [68]	Yes	No	Yes	No	No	No	No

Table 3. Comparison of Partitioning Objectives

Approaches	Hardware	Software	Network
Data Stream Application [47]	No	No	Yes
Hybrid Granularity Graph [29]	Yes	Yes	No
Distributed Abstract Class Graph [30]	No	No	No
OLIE [35]	No	Yes	Yes
Adaptive Multi-Constraint Partitioning [5]	Yes	Yes	Yes
Automated Refactoring [36]	No	No	No
Dynamic Software Deployment [34]	Yes	Yes	Yes
J-Orchestra [43]	No	Yes	No
Parametric Analysis [45]	No	No	No
RCMPP [46]	Yes	No	Yes
Multi-site Computation Offloading [41]	No	No	No
Calling the Cloud [27]	Yes	Yes	No
CloneCloud [33]	Yes	No	Yes
MACS [37]	Yes	Yes	No
Improving Energy Efficiency [40]	No	No	No
MAUI [26]	Yes	Yes	Yes
Partitioning Application [42]	No	No	No
Dynamic Updates [25]	No	No	No
Roam [32]	No	No	No
Complexity Prediction [39]	No	No	No
Simplifying Cyber Foraging [90]	No	No	No
Energy-Optimal Software Partitioning [67]	Yes	Yes	No
Wishbone [68]	Yes	No	Yes

Table 4. Comparison of Profilers.

V. ISSUES AND CHALLENGES FOR APPLICATION PARTITIONING

Table 5 summarizes challenges to current APAs and open research issues in mobile cloud application partitioning. Issues are about the unresolved problems in current APAs whereas challenges are the issues of research in application partitioning for MCC that remain to be addressed. The following section discusses issues in current partitioning approach and identifies challenges to the cloud-based mobile application partitioning.

A. Diverse Nature of Partitioning

The handling of synchronization is a critical issue in guaranteeing regular semantics for a partitioned application [43, 45, 93]. Most of the programming languages such as Java have no support for remote synchronization. Nevertheless, it is possible to build a distributed synchronization mechanism that will guarantee semantics identical to regular programming language for all partitioned applications. However, such a mechanism will likely be complex and inefficient. Another issues with synchronization is the possibility of self-deadlocks if thread identity is not maintained when the flow of control moves over the network. Besides, during application partitioning, there is need of inheritance across partitions [25]. When parent and child classes are separated, a child object instance requires the parent object to be instantiated in its partition. Even though J-Orchestra [43] does try to address this issue that an update of a parent class is directly reflected in all its derived classes, cross partition delegations require the parent class to be instantiated before the child class, which is not possible if the parent class is abstract. Besides, there are partitioning approaches may create overlapping partitions as they allows for classes belonging to more than one partition [25]. It is necessary to meet specific performance requirements. For example, local method invocations do not suffer from any run-time overhead. Also, having the same classes with different versions in different partitions are problematic which may result in update cannot be performed. Despite of dynamic partitioning, static partitioning need to extend the presented model with a support for repartitioning, nested partitions, and class migration across partitions. Therefore, semi-dynamic partitioning or nested partitioning is suggested to be linked with a dynamic performance analysis tool to automatically repartition the application into the optimal partitioning configuration that best addresses the application's needs.

	Challenges to Current APAs	**Open Issues in Application Partitioning for MCC**
Wide variations and dynamic changes in network condition	Yes	Yes
Local resource availability		Yes
Device heterogeneity	Yes	Yes
Heterogeneity in cloud computing services		Yes
Multi-site offloading	Yes	Yes
Dynamic update issue (compatibility, object identity, class unloading, state transfer, and performance overhead.)	Yes	Yes
Degeneracy problem	Yes	
Cloud resources' competence by users		Yes
Programmer burdens	Yes	Yes
Rapidly modifying large, unfamiliar applications for cyber foraging	Yes	
Programming language support	Yes	Yes
Design and implementation of application partitioning	Yes	
Power-save mode can hurt the overall energy consumption when the latency to server is slow	Yes	Yes
Cost of profiling is not negligible and impact overall performance	Yes	Yes
Profiler garbage Collection	Yes	Yes
Security and privacy	Yes	Yes
Fault tolerance		Yes
Usability issues	Yes	Yes
Partition synchronization	Yes	Yes
Partitioning granularity	Yes	Yes
Overlapping partition	Yes	
Inheritance across partition	Yes	
Repartitioning, nested partitions, class migration across partitions for static partitioning	Yes	Yes

Table 5. Issues and Challenges in Application Partitioning Approaches

B. High Software Quality Requirements for Partitioning

Security and privacy is always the great concern for MCC as it involves data transmission between SMDs and cloud servers over network of wireless technology [8]. Coarser level of partitioning granularity results in greater security threats of the outsourcing components of mobile application. For example, offloading of partitioning results in module or component level is more vulnerable to network threats as compared to thread or method level of partitioning results. Remote execution and remote communication will inevitably increase the risk of application failure. Consequently, such an effective and efficient fault-tolerant mechanism is highly needed for partitioning approaches for offloading [5, 27, 42]. Existing works always place the issue of security and privacy as the future work which remain to be addressed. This practice has cause a rising in security problem. On the other hand, running the applications remotely is limited by the network bandwidth and often raises usability issues due to varying latency [26, 34, 42]. Thus, recent research efforts have proposed to offload parts of an application from the mobile device to the cloud, thereby demonstrating important gains in battery life and performance. Computation offloading raises two important issues, namely, what and when to migrate for remote execution [34]. Making partitioning decisions always involves selecting appropriate partitioning granularity and addressing application optimization problems. Beside, most of the existing works also ignore the changes in the network bandwidth or latency, sudden increases of the CPU load on the mobile device, and variations in the user's inputs during interactions. These incidents can dramatically impact the performance and responsiveness of most applications. Hence, it is very important for the researchers to work on these aforementioned issues as it will greatly impact on software quality and user experience of MCC application.

C. Availability and Heterogeneity of Execution Environment

Along with the remarkable benefits brought by cloud computing, there is significant complexity involved in ensuring that mobile applications and associated data perform and adjust whenever they need to achieve efficient operation under variable loads [32, 34, 37, 41, 42, 47]. Specifically, there are two challenges to address. First, on the SMDs side, we need to handle the wide variations and dynamic changes in network conditions and local resource availability. In order to achieve high and optimal performance of SMDs, the decision of which units of computation should be moved to the cloud and it has to be made adaptive to the changes in mobile environments. Second, for the cloud server side, we need to handle the unpredictable and varying load from multiple mobile clients of the application. Different cloud computing services provide different privacy guarantees, I/O performance, CPU power, or network latency. Further, the components of an application might have different privacy/security demands, I/O needs, CPU requirements, or latency expectations. To the best of our studies, there is no comprehensive treatment of these problems. Even though existing computation partitioning mechanisms enable adaptive execution of mobile application between mobile devices and the server, these approaches are only suitable in traditional mobile Internet computing. Also, such approaches do not give any solution on how to utilize the elastic resources in clouds to make the applications scalable in cases of serving a large number of mobile users. Other efforts in facilitating large scale cloud applications do not fit well in the MCC applications because they do not provide a flexible and adaptive mechanism to schedule the computation across the client and clouds.

D. Necessity of Programming Effort for Development of Partitioning

Some existing applications may have been ported to older devices, but others may not. In order to attract new corporate customers, the software vendor must help them rapidly port their critical applications. Thus, vendor always prefers lower programming capability for these efforts due to the economic factor. This leads to the central challenge of MCC. It is very important for new-experienced software developers rapidly modify large, unfamiliar applications for MCC [90]. Source code application is always not available and hard to obtain. Even if the source code is available, understanding a large body of source code is time-consuming task. Therefore, researchers' design must help developers rapidly identify the relevant parts of an unfamiliar code base and then help them easily create the necessary modifications for application partitioning in remote execution framework. It is very rare that a new application may be written from scratch for the new device since the hardware capabilities of devices are improved on double. Besides, some of the partitioning approaches need developers to provide inputs such as annotation and specification [25, 26, 32, 35, 36, 39, 40, 42, 43, 45, 68, 90]. It is certainly unfavourable by the developers as this kind of task is massive.

E. Diverse Design Objective of Partitioning

Computation offloading is considered as a critical technique to improve the responsiveness of compute-intensive interactive applications in MCC. By using the offloading technique, a fundamental problem is to partition the computations involved in the application between the mobile device and cloud. However, there are very less number of approaches study the computations partitioning problems from the application provider's standpoint by taking into account the constraint on the cloud resources [47]. This constraint is due to two practical facts: the resource leased by the application provider is not unlimited because of the operational cost, and the applications on cloud are always faced with unpredictable number of users [47]. In such cases, the users' partitioning results are dependent with each other because of their competence for the cloud resources. For example, one user's decision on whether to offload the task not only depends on its saved computational cost and communication overhead, but also depends on how many other users offload the tasks into cloud. The number of users who offload the tasks onto cloud represents the load on the cloud. If the load is high, the time spent in waiting for available cloud resources may sacrifice the benefit of offloading. MCC requires a unified schedule of all the users' computations onto their mobile devices and cloud resources. Another recent issue for MCC is multi-site offloading [41]. Most previous work focuses on single-site offloading, where an application is divided between the mobile device and a single remote server. Such offloading would incur far more communication costs. Algorithms must be developed to partition a program between multiple possible execution sites while these partitioning algorithms must account for differences in capabilities between remote. Offloading a module to the wrong site can result in less efficient execution. The granularity of partition for offloading must also be chosen appropriately. For example, some prior works perform offloading at the object level, making the decision to offload for all objects of a particular class. Using objects as the granularity of partitioning presents problems. First, it is impossible to represent each application as an object graph for partitioning. It is because the number of objects is potentially unbounded and, moreover, indeterminable statically. Second, even though a well-suited object graph that is amenable to partitioning is given, it is always unclear for how to generate code to support offloading. Besides, several issues and challenges are introduced that need to be addressed in any updatable system regardless of what technique is used [25]. These problems include compatibility, object identity, class unloading, state transfer, and performance overhead. Dynamic updates of applications must not compromise correctness of security and semantic as well as stability of the updated applications. When we dynamically replace the byte code of a class by new byte code, we also must make sure that all instances of this class are updated accordingly. Furthermore, all program entities that refer either to this class or any of its instances have to be updated as well as they now refer to the new versions and not the old one. Besides, classes loaded by a different class loader cannot refer to one other unless they are loaded by the same parent class loader. This may lead to the creation of a hierarchy of class loaders, which is not desirable for long term applications whose structure changes over time. Furthermore, there is the possibility that cyclic class dependencies are introduced. Besides, the support for dynamic updates requires some level of indirection so that method calls are directed to the correct objects and classes, respectively. As this level of indirection is required for every updatable class, some run-time performance degradation will occur for all updatable classes.

F. Functional Complexity of Partitioning

Functional complexity of partitioning is another challenge and issue for MCC. As aforementioned, many of partitioning approaches aim to save energy [26, 27, 30, 33, 34, 37, 39, 40, 45, 67]. However, the great concerns is that using power-save mode during computation offloading can negatively impact the overall energy consumption of the mobile application when the latency to the clouds is slow [26]. Another unforeseen challenge is related to using profiling technique for estimating the energy savings of code offload. Profiling the state transfer overhead of a method whenever it is called can provide the latest and accurate estimation, yet, it can hurt the overall application's performance as the cost of this profiling is not negligible. Besides, there are results of experiment show that more information is needed to be processed or profiled in order to make adaptation decisions [94]. This information is only temporarily stored or used during decision making. However, much or some of this information has not been discarded prior to the measurement of memory usage which causes unnecessary high overhead to the execution framework. Instead, garbage collection should be explicitly requested prior to each measurement or profiling. Garbage collection should be considered by researchers in their proposed frameworks.

VI. CONCLUSION AND FUTURE DIRECTIONS

This paper reviews and presents the thematic taxonomy of existing APAs. It analyses the APAs by highlighting the commonalties and deviations on the basis of significant parameters. It studies current issues in APAs and highlights challenges for achieving optimal and lightweight partitioning for MCC application. Current APAs model applications in diverse models. Several approaches model applications into graph-based model; some approaches formulate the LP optimization problem for application partitioning; others implement graph-based together with LP-based modelling for application. Some of approaches restrict to single programming language support; others are implemented with different programming environment. Variant types of profilers are implemented: network, software, and hardware. Several approaches employ online allocation decision; others prefer offline allocation decision. Static analysis technique is used by some approaches; other exercises dynamic analysis technique. Some APAs require annotation for optimal partitioning. Several approaches annotate application automatically; others need manual annotation from programmer and developer. Diverse objective functions are consider: reducing network overhead, saving energy, improving performance, reducing memory constraints, reducing programmers' burden, updating application dynamically, and multi-site offloading.

Abebe et al.'s partitioning approaches is the best among 23 approaches that we have reviewed. We consider energy-saving as the main concern and top priority in optimal execution of MCC application. This approach decrease the power consumption of SMDs while it also covers other essential objectives for to improve the QoS for mobile users including reduce network, power and memory utilization as well as the performance cost of offloading. Abebe et al. represent the application into abstract class graph which allows the partition result on the cloud to be on-loaded back to the client devices for utility. This approach reduces partitioning overhead during adaptation as it does not need to store and update complete application graphs on each device. We conclude that this approach is the best for MCC environment. On the other hand, OLIE is the most unsuitable partitioning approach for optimal execution in MCC application. OLIE ignores many partitioning objectives especially energy-saving while focus only on alleviating the memory constraint of SMDs which is relatively unfavourable by mobile users compare to other partitioning objectives. Similar to Abebe et al.'s approach, OLIE models the application into class graph, yet, Abebe et al.'s approach models both application and cloud components which allow better partitioning efficacy. Besides, OLIE needs developers to provide decision-making rules to support application-specific adaption whereas Abebe et al's approach does not need any manual annotation which reduce programmers' burden.

Even though application partitioning is an old concept which was introduce in 1993, it has been implemented in different computing domains such as Pervasive Computing, Grid Computing, and Cloud Computing. Application partitioning is a promising concept, significantly useful for computation offloading. Computation offloading employs application partitioning to address the resources-constrained issues of SMDs by separating the functionality of mobile applications into distinct partitions that can operate independently in distributed setting and then offload to the remote surrogates. Current APAs focus on the user experience without considering security and privacy in higher priority where researchers always leave it for future work. Hence, this issue results in leaking of user data in the application. Also, current APAs do not see the computation offloading especially application partitioning from the view of application providers which results in the resources intensive management overheads on computing clouds for the entire duration of remote execution. Current APAs exploit local resources on SMDs as well as computing resources in cloud servers for decision making of partitioning by application profiling on resources consumption of SMDs. Therefore, we conclude that current APAs employ heavyweight procedures for distributed application deployment and management. The heterogeneity of the distributed execution environments in MCC and the poverty of SMDs' resources demand an adaptive, context-aware partitioning approach for optimal remote execution in order to enhance the user experience.

Acknowledgement

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/HIR/MOHE/FCSIT/03.

References

[1] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless Communications and Mobile Computing,* pp. 1-38, 2011.

[2] M. Prgomet, A. Georgiou, and J. I. Westbrook, "The impact of mobile handheld technology on hospital physicians' work practices and patient care: a systematic review," *Journal of the American Medical Informatics Association,* vol. 16, pp. 792-801, 2009.

[3] J. O. Soh and B. C. Tan, "Mobile gaming," *Communications of the ACM,* vol. 51, pp. 35-39, 2008.

[4] *Google Drive*. Available: http://drive.google.com

[5] S. Ou, K. Yang, and A. Liotta, "An adaptive multi-constraint partitioning algorithm for offloading in pervasive systems," in *Pervasive Computing and Communications, 2006. PerCom 2006. Fourth Annual IEEE International Conference on*, 2006, pp. 116 - 125.

[6] E. Abebe and C. Ryan, "A Hybrid Granularity Graph for Improving Adaptive Application Partitioning Efficacy in Mobile Computing Environments," *2011 10th Ieee International Symposium on Network Computing and Applications (Nca),* 2011.

[7] K. Kumar, J. Liu, Y. H. Lu, and B. Bhargava, "A Survey of Computation Offloading for Mobile Systems," *Mobile Networks and Applications,* pp. 1-12, 2012.

[8] M. Shiraz, A. Gani, R. Khokhar, and R. Buyya, "A Review on Distributed Application Processing Frameworks in Smart Mobile Devices for Mobile Cloud Computing," *Communications Surveys & Tutorials,* pp. 1-20, 2012.

[9] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski*, et al.*, "A view of cloud computing," *Communications of the ACM,* vol. 53, pp. 50-58, 2010.

[10] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation computer systems,* vol. 25, pp. 599-616, 2009.

[11] A. Fox and R. Griffith, "Above the clouds: A Berkeley view of cloud computing," University of California, Berkeley, Technical Report2009.

[12] D. Greschler and T. Mangan, "Networking lessons in delivering 'Software as a Service'—part I," *International Journal of Network Management,* vol. 12, pp. 317-321, 2002.

[13] C. Vidyanand, "Software as a Service: Implications for Investment in Software Development," in *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, 2007, pp. 209-219.
[14] S. Wei, Z. Xin, G. Chang Jie, S. Pei, and S. Hui, "Software as a Service: Configuration and Customization Perspectives," in *Congress on Services Part II, 2008. SERVICES-2. IEEE*, 2008, pp. 18-25.
[15] S. Bhardwaj, L. Jain, and S. Jain, "Cloud computing: A study of infrastructure as a service (IAAS)," *International Journal of engineering and information Technology,* vol. 2, pp. 60-63, 2010.
[16] R. Prodan and S. Ostermann, "A survey and taxonomy of infrastructure as a service and web hosting cloud providers," in *Grid Computing, 2009 10th IEEE/ACM International Conference on*, 2009, pp. 17-25.
[17] E. Keller and J. Rexford, "The “Platform as a service” model for networking," *INM/WREN,* vol. 10, pp. 95-108, 2010.
[18] G. Lawton, "Developing software online with platform-as-a-service technology," *Computer,* vol. 41, pp. 13-15, 2008.
[19] *Amazon Elastic Compute Cloud (EC2)*. Available: http://www.amazon.com/ec2/
[20] *Microsoft Azure*. Available: http://www.microsoft.com/azure/
[21] *Google AppEngine*. Available: http://appengine.google.com
[22] S. Goyal and J. Carter, "A lightweight secure cyber foraging infrastructure for resource-constrained devices," in *Mobile Computing Systems and Applications, 2004. WMCSA 2004. Sixth IEEE Workshop on*, 2004, pp. 186-195.
[23] Q. Liu, X. Jian, J. Hu, H. Zhao, and S. Zhang, "An optimized solution for mobile environment using mobile cloud computing," in *Wireless Communications, Networking and Mobile Computing, 2009. WiCom'09. 5th International Conference on*, 2009, pp. 1-5.
[24] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *Pervasive Computing, IEEE,* vol. 8, pp. 14-23, 2009.
[25] R. Bialek, E. Jul, J. G. Schneider, and Y. Jin, "Partitioning of Java applications to support dynamic updates," in *Software Engineering Conference, 2004. 11th Asia-Pacific*, 2004, pp. 616-623.
[26] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra*, et al.*, "MAUI: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, 2010, pp. 49-62.
[27] I. Giurgiu, O. Riva, D. Juric, I. Krivulev, and G. Alonso, "Calling the cloud: Enabling mobile phones as interfaces to cloud applications," *Middleware 2009,* pp. 83-102, 2009.
[28] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y. M. Wang, "Enabling automatic offloading of resource-intensive smartphone applications," Purdue University, Technical Report2011.
[29] E. Abebe and C. Ryan, "A Hybrid Granularity Graph for Improving Adaptive Application Partitioning Efficacy in Mobile Computing Environments," in *Network Computing and Applications (NCA), 2011 10th IEEE International Symposium on*, 2011, pp. 59-66.
[30] E. Abebe and C. Ryan, "Adaptive application offloading using distributed abstract class graphs in mobile environments," *Journal of Systems and Software,* vol. 85, pp. 2755-2769, 12// 2012.
[31] D. Bi-Ru and I. C. Lin, "Efficient Map/Reduce-Based DBSCAN Algorithm with Optimized Data Partition," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, 2012, pp. 59-66.

[32] H. Chu, H. Song, C. Wong, S. Kurakake, and M. Katagiri, "Roam, a seamless application framework," *Journal of Systems and Software,* vol. 69, pp. 209-226, 2004.

[33] B. G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems*, 2011, pp. 301-314.

[34] I. Giurgiu, O. Riva, and G. Alonso, "Dynamic Software Deployment from Clouds to Mobile Devices," *Middleware 2012,* pp. 394-414, 2012.

[35] X. Gu, K. Nahrstedt, A. Messer, I. Greenberg, and D. Milojicic, "Adaptive offloading inference for delivering applications in pervasive computing environments," in *Pervasive Computing and Communications, 2003.(PerCom 2003). Proceedings of the First IEEE International Conference on*, 2003, pp. 107-114.

[36] V. Jamwal and S. Iyer, "Automated refactoring of objects for application partitioning," in *Software Engineering Conference, 2005. APSEC'05. 12th Asia-Pacific*, 2005, pp. 671-678.

[37] D. Kovachev and R. Klamma, "Framework for Computation Offloading in Mobile Cloud Computing," *International Jorunal of Interactive Multimedia and Artificial Intelligence},* vol. 1, pp. 6-15, 2012.

[38] S. B. Musunoori and G. Horn, "Intelligent ant-based solution to the application service partitioning problem in a grid environment," in *Intelligent Systems Design and Applications, 2006. ISDA'06. Sixth International Conference on*, 2006, pp. 416-424.

[39] L. D. Pedrosa, N. Kothari, R. Govindan, J. Vaughan, and T. Millstein, "The Case for Complexity Prediction in Automatic Partitioning of Cloud-enabled Mobile Applications," *Small,* vol. 20, 2012.

[40] M. R. Ra, B. Priyantha, A. Kansal, and J. Liu, "Improving Energy Efficiency of Personal Sensing Applications with Heterogeneous Multi-Processors," in *The 14th International Conference on Ubiquitous Computing (Ubicomp 2012)*, 2012, pp. 1-10.

[41] K. Sinha and M. Kulkarni, "Techniques for fine-grained, multi-site computation offloading," in *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, 2011, pp. 184-194.

[42] M. Smit, M. Shtern, B. Simmons, and M. Litoiu, "Partitioning Applications for Hybrid and Federated Clouds," in *Proceedings of the 2012 Conference of the Center for Advanced Studies on Collaborative Research*, 2012, pp. 27-41.

[43] E. Tilevich and Y. Smaragdakis, "J-orchestra: Automatic java application partitioning," *ECOOP 2002—Object-Oriented Programming,* pp. 178-204, 2006.

[44] T. Verbelen, T. Stevens, F. De Turck, and B. Dhoedt, "Graph partitioning algorithms for optimizing software deployment in mobile cloud computing," *Future Generation Computer Systems,* pp. 451-459, 2012.

[45] C. Wang and Z. Li, "Parametric analysis for adaptive computation offloading," *ACM SIGPLAN Notices,* vol. 39, pp. 119-130, 2004.

[46] L. Yang, J. Cao, and H. Cheng, "Resource Constrained Multi-user Computation Partitioning for Interactive Mobile Cloud Applications," Hong Kong Polytechnical University, Technical Report2012.

[47] L. Yang, J. Cao, S. Tang, T. Li, and A. T. S. Chan, "A Framework for Partitioning and Execution of Data Stream Applications in Mobile Cloud Computing," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, 2012, pp. 794-802.

[48] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *ACM SIGCOMM Computer Communication Review,* vol. 39, pp. 50-55, 2008.

[49] L. Wang, G. Von Laszewski, A. Younge, X. He, M. Kunze, J. Tao*, et al.*, "Cloud computing: a perspective study," *New Generation Computing,* vol. 28, pp. 137-146, 2010.

[50] (2010). *Cloud Computing: Silver Lining or Storm Ahead (Vol 13 No 2 Spring ed.)*. Available: http://iac.dtic.mil/iatac

[51] *Google Apps*. Available: http://www.google.com/apps/index1.html

[52] *Microsoft Office 365*. Available: http://office365.microsoft.com

[53] *Onlive*. Available: http://www.onlive.com/

[54] *Amazon Web Services Elastic Beanstalk*. Available: http://aws.amazon.com/elasticbeanstalk/

[55] *Heroku*. Available: http://www.heroku.com

[56] L. Guan, X. Ke, M. Song, and J. Song, "A survey of research on mobile cloud computing," in *Computer and Information Science (ICIS), 2011 IEEE/ACIS 10th International Conference on*, 2011, pp. 387-392.

[57] Q. Han and A. Gani, "Research on mobile cloud computing: Review, trend and perspectives," in *Digital Information and Communication Technology and it's Applications (DICTAP), 2012 Second International Conference on*, 2012, pp. 195-202.

[58] D. Huang, "Mobile cloud computing," *IEEE COMSOC Multimedia Communications Technical Committee (MMTC) E-Letter,* 2011.

[59] *Mobile Cloud Computing Forum*. Available: http://www.mobilecloudcomputingforum.com

[60] X. Yang, T. Pan, and J. Shen, "On 3G mobile e-commerce platform based on cloud computing," in *Ubi-media Computing (U-Media), 2010 3rd IEEE International Conference on*, 2010, pp. 198-201.

[61] K. Peters, "m-Learning: Positioning educators for a mobile, connected future," *Mobile Learning,* pp. 1-17, 2009.

[62] C. J. Li, L. Liu, S. Z. Chen, C. C. Wu, C. H. Huang, and X. M. Chen, "Mobile healthcare service system using RFID," in *Networking, Sensing and Control, 2004 IEEE International Conference on*, 2004, pp. 1014-1019.

[63] A. Rahbar, "An E-Ambulatory Healthcare System Using Mobile Network," in *Information Technology: New Generations (ITNG), 2010 Seventh International Conference on*, 2010, pp. 1269-1273.

[64] *GMail*. Available: http://mail.google.com

[65] *Mobile Cloud Applications. ABI Research Report.* . Available: http://www.abiresearch.com/research/1003385Mobile+Cloud+Computing

[66] J. Research. (Jan 2010). *Mobile cloud application & services* [Web]. Available: http://www.juniperresearch.com/reports/mobile-cloud-applications-and-services

[67] M. Goraczko, J. Liu, D. Lymberopoulos, S. Matic, B. Priyantha, and F. Zhao, "Energy-optimal software partitioning in heterogeneous multiprocessor embedded systems," in *Proceedings of the 45th annual Design Automation Conference*, 2008, pp. 191-196.

[68] R. Newton, S. Toledo, L. Girod, H. Balakrishnan, and S. Madden, "Wishbone: Profile-based partitioning for sensornet applications," in *Proceedings of the 6th USENIX symposium on Networked systems design and implementation (NSDI), Boston, MA*, 2009, pp. 395-408.

[69] S. Abolfazli, Z. Sanaei, and A. Gani, "Mobile Cloud Computing: A Review on Smartphone Augmentation Approaches," *CoRR,* vol. abs/1205.0451, 2012.

[70] A. Rudenko, P. Reiher, G. J. Popek, and G. H. Kuenning, "Saving portable computer battery power through remote process execution," *ACM SIGMOBILE Mobile Computing and Communications Review,* vol. 2, pp. 19-26, 1998.

[71] M. Satyanarayanan, "Pervasive computing: Vision and challenges," *Personal Communications, IEEE,* vol. 8, pp. 10-17, 2001.

[72] X. Zhang, A. Kunjithapatham, S. Jeong, and S. Gibbs, "Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing," *Mobile Networks and Applications,* vol. 16, pp. 270-284, 2011.

[73] K. Kumar and Y. H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?," *Computer,* vol. 43, pp. 51-56, 2010.

[74] C. Li and L. Li, "Energy constrained resource allocation optimization for mobile grids," *Journal of Parallel and Distributed Computing,* vol. 70, pp. 245-258, 2010.

[75] R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, and H. I. Yang, "The case for cyber foraging," in *Proceedings of the 10th workshop on ACM SIGOPS European workshop*, 2002, pp. 87-92.

[76] S. Ou, K. Yang, and L. Hu, "Cross: A combined routing and surrogate selection algorithm for pervasive service offloading in mobile ad hoc environments," in *Global Telecommunications Conference, 2007. GLOBECOM'07. IEEE*, 2007, pp. 720-725.

[77] D. A. Levine, I. F. Akyildiz, and M. Naghshineh, "A resource estimation and call admission algorithm for wireless multimedia networks using the shadow cluster concept," *Networking, IEEE/ACM Transactions on,* vol. 5, pp. 1-12, 1997.

[78] G. Karner, "Resource estimation for objectory projects," Master, Objective Systems SF AB, 1993.

[79] K. Reddy and M. Ranjan, "Solar resource estimation using artificial neural networks and comparison with other correlation models," *Energy Conversion and Management,* vol. 44, pp. 2519-2530, 2003.

[80] L. C. Briand and I. Wieczorek, "Resource estimation in software engineering," *Encyclopedia of Software engineering,* pp. 1160-1196, 2002.

[81] M. Sharifi, S. Kafaie, and O. Kashefi, "A survey and taxonomy of cyber foraging of mobile devices," *Communications Surveys & Tutorials,* pp. 1-12, 2011.

[82] B. G. Chun and P. Maniatis, "Augmented smartphone applications through clone cloud execution," in *Proc. of the 8th Workshop on Hot Topics in Operating Systems (HotOS), Monte Verita, Switzerland*, 2009, p. 8.

[83] A. Veda, "Application Partitioning-A Dynamic, Runtime, Object-level Approach," Indian Institute of Technology Bombay, 2006.

[84] A. Dou, V. Kalogeraki, D. Gunopulos, T. Mielikainen, and V. H. Tuulos, "Misco: a MapReduce framework for mobile systems," in *Proceedings of the 3rd International Conference on PErvasive Technologies Related to Assistive Environments*, 2010, p. 32.

[85] B. Zhao, Z. Xu, C. Chi, S. Zhu, and G. Cao, "Mirroring smartphones for good: A feasibility study," *Mobile and Ubiquitous Systems: Computing, Networking, and Services,* pp. 26-38, 2012.

[86] A. Messer, I. Greenberg, P. Bernadat, D. Milojicic, D. Chen, T. Giuli*, et al.*, "Towards a distributed platform for resource-constrained devices," in *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, 2002, pp. 43-51.

[87] X. Zhang, J. Schiffman, S. Gibbs, A. Kunjithapatham, and S. Jeong, "Securing elastic applications on mobile devices for cloud computing," in *Proceedings of the 2009 ACM workshop on Cloud computing security*, 2009, pp. 127-134.

[88] J. Engblom and A. Ermedahl, "Modeling complex flows for worst-case execution time analysis," in *Real-Time Systems Symposium, 2000. Proceedings. The 21st IEEE*, 2000, pp. 163-174.

[89] J. S. Rellermeyer, O. Riva, and G. Alonso, "AlfredO: an architecture for flexible interaction with electronic devices," in *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, 2008, pp. 22-41.

[90] R. K. Balan, D. Gergle, M. Satyanarayanan, and J. Herbsleb, "Simplifying cyber foraging for mobile devices," in *Proceedings of the 5th international conference on Mobile systems, applications and services*, 2007, pp. 272-285.

[91] J. Bentley, "Programming pearls: little languages," *Communications of the ACM,* vol. 29, pp. 711-721, 1986.

[92] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, 1979.

[93] T. Fahringer and A. Jugravu, "JavaSymphony: A new programming paradigm to control and synchronize locality, parallelism and load balancing for parallel and distributed computing," *Concurrency and Computation: Practice and Experience,* vol. 17, pp. 1005-1025, 2005.

[94] H. Gani, "Transparent and adaptive application partitioning using mobile objects," PhD Thesis, 2010.