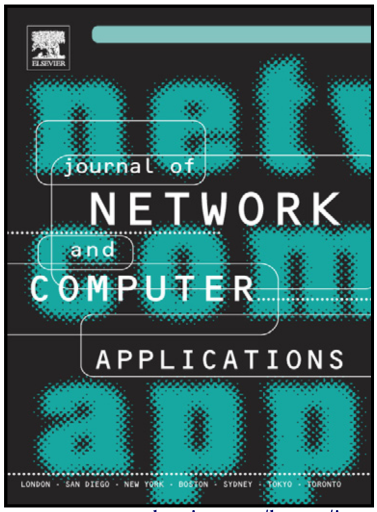
Author's Accepted Manuscript



www.elsevier.com/locate/jnca

A Survey of Computation Offloading Strategies for Performance Improvement of Applications Running on Mobile Devices

Minhaj Ahmad Khan

PII: S1084-8045(15)00132-0
DOI: http://dx.doi.org/10.1016/j.jnca.2015.05.018
Reference: YJNCA1414

To appear in: *Journal of Network and Computer Applications*

Received date: 6 August 2014
Revised date: 1 May 2015
Accepted date: 25 May 2015

Cite this article as: Minhaj Ahmad Khan, A Survey of Computation Offloading Strategies for Performance Improvement of Applications Running on Mobile Devices, *Journal of Network and Computer Applications,* http://dx.doi.org/10.1016/j.jnca.2015.05.018

A Survey of Computation Offloading Strategies for Performance Improvement of Applications Running on Mobile Devices

Minhaj Ahmad Khan

Bahauddin Zakariya University Multan, Pakistan.

Abstract

Handheld mobile devices have evolved from simple voice communication devices to general purpose devices capable of executing complex applications. Despite this evolution, the applications executing on the mobile devices suffer due to their constrained resources. The constraints such as limited battery lifetime, limited storage and processing capabilities produce an adverse impact on the performance of applications executing on the mobile devices.

Computation offloading addresses the issue of limited resources by transferring the computation workload to other systems having better resources. It may be oriented towards extending battery lifetime, enhancing storage capacity or improving the performance of an application. In this paper, we perform a survey of the computation offloading strategies correlated with performance improvement for an application. We categorize these approaches in terms of their workload distribution and offloading decisions. We also describe the evolution of the computation offloading based environment as well as a categorization of application partitioning mechanisms adopted in various contributions. Furthermore, we present a parameter-wise comparison of automated frameworks, the application domains that benefit from computation offloading and the future challenges impeding the evolution of computation offloading.

Keywords:
Computation Offloading, Mobile Computing, Performance Improvement, Mobile Cloud Computing, Cyberaging

1. Introduction

With the advent of smartphone technologies, the mobile devices have become ubiquitous. These devices are no longer constrained to providing only communication services. Instead, these devices are capable of executing applications with diverse requirements. The processing required by these applications may range from simple mathematical computations performed by a calculator to a very complex voice recognition system.

The execution of complex applications requires the mobile devices to possess powerful resources. The scarcity of these resources has adverse effects on the ever-growing usage of the mobile devices. For instance, the statistics according to *StatCounter* show that about 30.66%

Email address: `mik@bzu.edu.pk` (Minhaj Ahmad Khan)

of the platforms used for web browsing are the mobile systems (smartphones/tablets) [1]. Consequently, the mobile market plays a significant role in e-commerce and sales growth. This role is however diminished by the fact that the mobile systems have limited energy and power resources. Although there have been efforts to incorporate high performance multiple core processors in smartphones, the gap b/w the existing and the required resources continues to grow. In this context, the computation offloading is a mechanism that enables us to bridge the gap by making intensive computations execute on large systems having sufficient resources as required by the application. This not only makes a resource constrained mobile system seem like a high-end powerful machine, but also enables to perfectly utilize the existing resources.

The computation offloading is not a novel idea as it has evolved from various paradigms incorporating distributed computing [2, 3, 4, 5]. The performance improvement of an application is achieved by partitioning it into several subprograms each of which may be assigned to a different processor for execution. Each processor makes use of its own memory and/or shares the memory with other processors to perform computations in parallel. Subsequently, the results are returned to the processor controlling the overall execution.

A cloud computing platform is also based on the intuition of distributed computing and offers the compute services through a Service Level Agreement (SLA) on a large network usually the internet. It differs from other computing paradigms since an assurance regarding availability of services is provided to the users. The Mobile Cloud Computing (MCC) therefore refers to provision of services through a cloud to mobile devices that are characterized with limited resources [2, 3, 4, 5, 6, 7, 8]. The computation of a mobile application may be offloaded to another resource-rich system termed as *surrogate*. Such kind of computation offloading not only mitigates the issue of limited resources of mobile devices but also enables to harness the processing power of high-end machines that will otherwise be idle [9, 10, 11, 12, 13, 14, 15].

In this paper, we perform a comprehensive survey of the computation offloading strategies impacting the performance of the applications executing on mobile devices. Although, the computation offloading has also been aimed at saving energy required for executing an application [16, 17, 18, 19, 20, 21, 22, 23, 24, 25], but in this paper, we mainly consider the contributions which impact the execution performance (computation speed) of applications running on mobile devices. The survey encompasses the research work for computation offloading arranged in terms of multiple aspects including the taxonomy, strategies, evolution pattern and relevant application domains. We also present a categorization of partitioning approaches adopted in different contributions and a parameter-wise comparison of main offloading frameworks. We also discuss main issues related to computation offloading and suggest possible approaches to address these issues effectively.

The rest of the paper is organized as follows. Section 2 describes the offloading taxonomy in terms of architectures and criteria for its effectiveness. The evolution of offloading and wireless technologies is described in Section 3. The offloading approaches and contributions aimed at performance improvement are surveyed in Section 4. A categorization of partitioning approaches used in computation offloading is given in Section 5. A parameter-wise comparison of the automated computation offloading frameworks is described in Section 6, whereas the applications benefiting from computation offloading are discussed in Section 7. The main issues related to an effective implementation of computation offloading are discussed in Section 8 together with their solutions before concluding at Section 9.

2. Offloading Taxonomy: Architectures and Effectiveness

Many clients such as mobile phones or low power laptops require computation to be offloaded to powerful server machines. The decision of offloading may not always be beneficial to leverage the performance or energy requirements as a significant overhead is involved while offloading computations. This section describes succinctly the general architectures for which offloading may be required and the parameters that impact its effectiveness.

2.1. Computation Offloading Architectures

In an environment supporting computation offloading, the users with mobile devices are connected to a high performance server in different ways. The simplest form of this connection is made through Wi-Fi based networks that connect mobile devices to other machines using wireless routers as shown in Figure 1. The wireless router not only connects devices to a local network but also may be connected to a DSL device thereby providing connections to remote servers through internet.

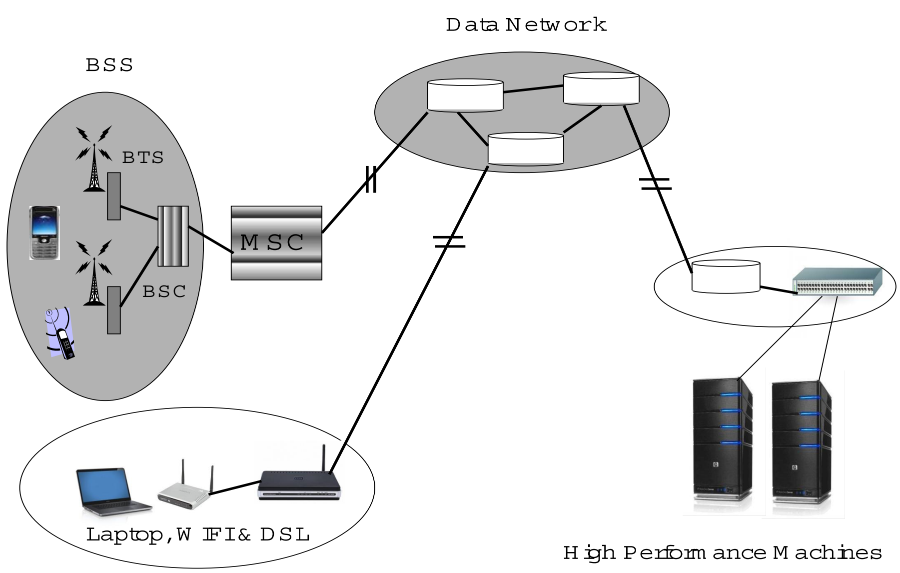


Figure 1: Offloading architecture

Similarly, in a more complex form, the users with mobile devices first connect to a wireless network through devices such as Base Transceiver Station (BTS), Base Station Controller (BSC), and Mobile Switching Center (MSC) to transfer data to public data networks. The communication data is then transferred through gateways to any local network on which the high performance machines are hosted.

After establishing a connection with the high performance machines, the mobile devices may perform a lookup operation to search for services that may be provided by the high performance server machines. This may also be termed as the first operation initiated by the application. The application may however opt to perform the lookup operation at a later time during execution depending upon the time at which the offloading decision is made and the requirement of the application. The client machines in these environments are usually low power mobile devices, and consequently, the computation offloading strategies take into account the cost/benefit analysis in terms of the execution time and energy requirements. The server machines are mostly the high-end standalone servers, or machines connected to form a grid, cluster, cloud or a combination of these. The computers in a grid are loosely coupled, whereas those in a cluster are tightly coupled with highly efficient interconnection interfaces such as *Myrinet*. A cloud system, in contrast, uses virtualization to enable multiple operating systems so that remote users can access services offered by the cloud platform.

2.2. Trade-offs for Offloading Decisions

For minimization of execution time and reduction of energy, the computation offloading from a mobile device to a server machine is performed by applying a specific criteria to ensure that the offloading will be beneficial [26, 27, 28, 29, 30, 31, 32, 33]. The required criteria takes into account several parameters as elaborated below.

For minimizing execution time, let O_r be the overhead of runtime activities including the time for data transfer and the time for offloading code, i.e.,

$$O_r = T_d + T_o, \tag{1}$$

where, T_d is the time for data transfer and T_o is the time taken for offloading code (performing offloading decision, partitioning and the code transfer). Let T_s be the time to execute code on the server machine and T_m be the time to execute code on the mobile device. The computation offloading is considered effective for minimization of execution time, if we have,

$$T_s + O_r < T_m. \tag{2}$$

Similarly, for energy reduction, let E_d represent the energy for data transfer and E_o represent the energy required for offloading. Let E_m represent the energy required for execution of entire application on the mobile device and E_r be the energy required for runtime activities. The computation offloading is effective for reducing requirements if

$$E_r < E_m, \tag{3}$$

where E_r is represented as

$$E_r = E_d + E_o. \tag{4}$$

3. Evolution of Offloading and Wireless Technology

The term ”offloading” has been used widely since year 1995. Its usage has evolved together with the evolution of distributed and parallel computing paradigms. Figure 2 shows the number of publications each year[1] citing the term offloading.

Similarly, the research work referring to the terms ”data offloading” and ”computation offloading” is also increasing gradually, as shown in Figure 3. Most of the data offloading systems aim at storage of data to remote servers with large storage repositories. One of the objectives of the recently evolved Mobile Cloud Computing (MCC) is to provide storage facilities to the users. The synchronization of data with that existing on the cloud storage repository is also provided by

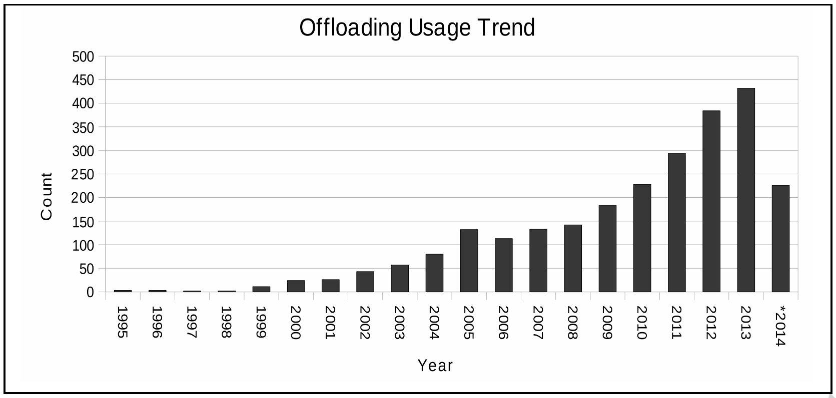


Figure 2: Offloading Usage Trend

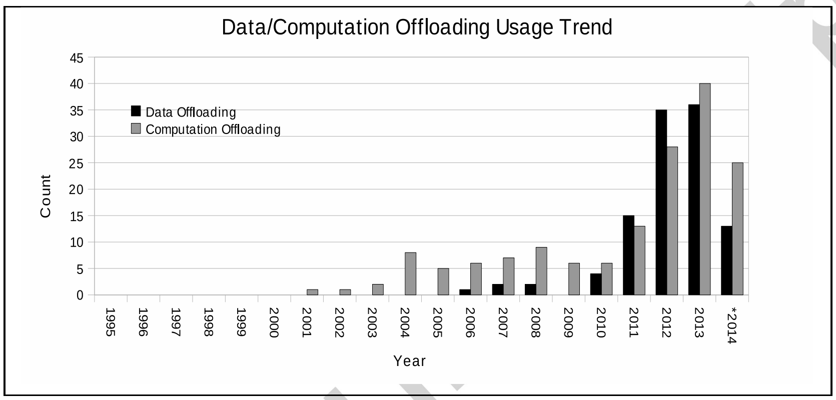


Figure 3: Data and Computation Offloading Usage Trend

MCC. Similar to data offloading, the computation offloading has also evolved to be incorporated in MCC. In general, it aims at energy minimization and performance improvement.

Figure 4 shows a quantitative and chronological evolution of several parameters related to wireless technology. The smartphones have evolved to contain multi-core based processors. Similarly, with the implementation of 3G and 4G based networks, the wireless technology is now able to offer more bandwidth than the previous generations. The orientation of offloading research has evolved from defining manual mechanisms to automated transparent offloading mechanisms. The energy requirements (Joules) as given in [34] for 50 KB data transfer (download with intervals of 20 seconds) through GSM, 3G and Wi-Fi are also shown. The Wi-Fi based data transmission requires the highest amount of energy.

4. Offloading Architectures and Approaches

We categorize computation offloading approaches into static and dynamic depending upon the time at which the decision of offloading takes place.

[1]Statistics obtained from the ACM Digital Library for duration up to July 2014

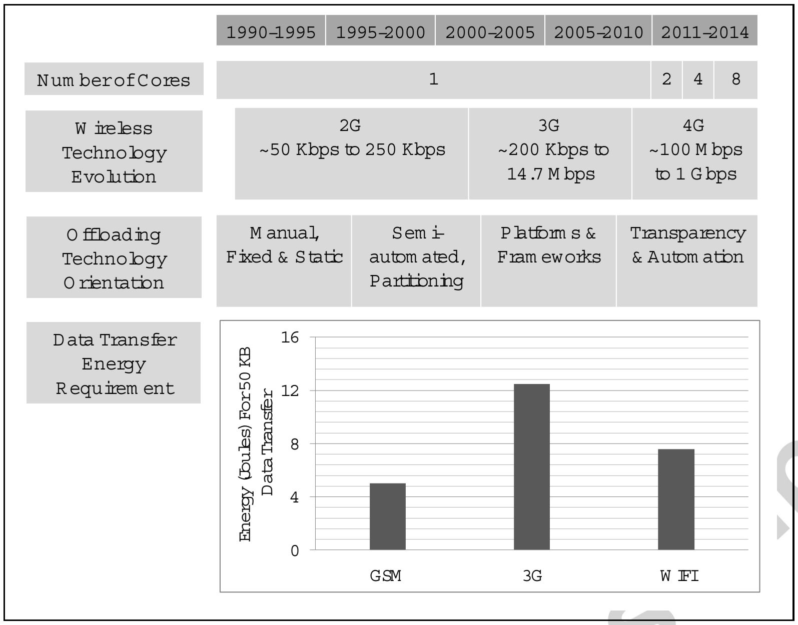


Figure 4: Evolution of wireless technology

112 *4.1. Static Offloading*

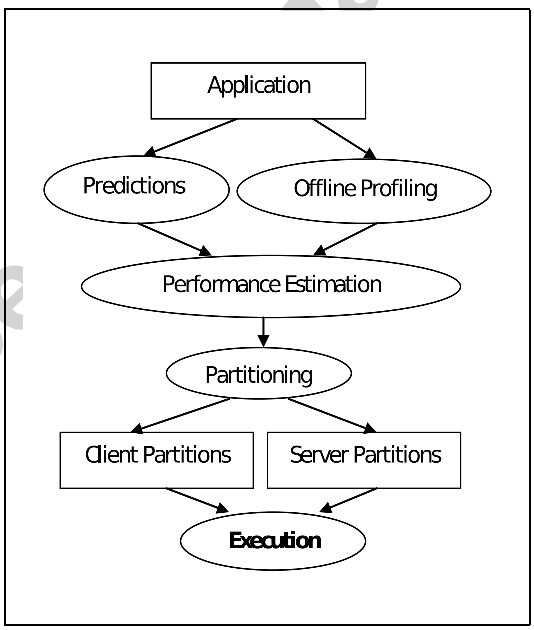


Figure 5: Static Offloading Mechanism

As shown in Figure 5, the static offloading approach makes use of performance prediction models or offline profiling to estimate the performance [26, 27, 35, 36, 37, 38]. The application is then partitioned into client and server partitions which may subsequently be executed.

A comparison of different static offloading strategies is shown in Table 1. The comparison is performed in terms of core components (the basic component on which processing takes place), the parameters considered for offloading decision, the offloading approach and the benchmarks for which the strategy is shown to be beneficial.

The approach suggested in [26] first generates a cost graph for the application. The cost graph takes into account the computation time and the data to be transferred. The suggested approach then distributes the program into client and server subtasks. The data communication among the tasks being executed by hosts takes place using the primitives of *push* and *pull*. The primitives correspond to sending and receiving the modified data. The application is modelled to produce the cost graphs representing energy consumption and data communication. The sum of both these parameters is minimized by suggesting a branch-and-bound algorithm and a pruning heuristic that reduces the search space to provide a near-optimal solution. The suggested approach produces a significant improvement in execution time and energy consumption for benchmarks from Mediabench suite and *gnugo* game.

An adaptive approach presented in [27] performs computation offloading by using an initial profile obtained by executing the program. If the program does not run to completion within a specified timeout, the offloading takes place and the rest of the computations are performed on some server. The minimum time required for executing the code on the mobile system is computed using the energy consumption on the local mobile processor. With the reduced energy consumption, a significant improvement in the performance is achieved for image processing benchmarks.

A framework called *Roam* which may be used for offloading of applications is suggested in [35]. The framework enables partitioning of an application into several components that may then be migrated to any other platform. This architecture supports heterogeneity in that the application components may be migrated to another system having a different execution environment. The approach of application offloading incorporates adaptation of three different types. The first one, *dynamic instantiation* based adaptation, partitions an application into several device dependent components. Each component has implementation for multiple platforms. The approach then takes into account the capabilities of the target system in order to select the components to be migrated. The second type, *offloading computation*, makes the applications use distributed resources by offloading components to remote servers. It is mainly required for offloading the application logic based code. The third type *trasformation* makes the user interface components compatible with the target device at runtime. The decision of partitioning is however static and is made at the time of designing the application.

The application partitioning algorithm suggested in [39] divides the application into two main parts. The first part contains the partition that can not be offloaded and will execute on the mobile device locally. The second part contains *k* partitions that can be offloaded to surrogates. The partitions are formed by modelling the computation and communication costs of the application components as a dynamic multi-cost graph. A special tightest and lightest vertex solution algorithm is then used to select a vertex in a partition. The algorithm considers the edge weights and vertex weights for partitioning. On the IBM laptop X31 and using two desktop PCs as surrogates, the application partitioning is shown to improve the performance for PI calculation, MP4 player and MP4 audio/video generation benchmarks.

A prototype platform *AIDE* suggested in [40] makes use of three modules for profiling the

Framework Contribution	Core Component	Parameters	Offloading Approach	Candidate Applications
[26]	Cost graph	Computation and data transfer time	Static	Mediabench & *gnugo*
[27]	Execution profile	Energy consumption and time required for execution	Static	Image processing
[35]	Application components	Components categorization	Static	General applications
[39]	Multi-cost graph	Computation and communication costs	Static	Audio and video applications
[40]	Execution profile	Communication cost and connectivity of nodes	Static	Text editor, Biomer and Voxel
[41]	Java bytecode	Configuration based	Static	SciMark benchmark
[42]	Jobs	Power consumption for execution and data transfer	Static	General applications
[31]	Control flow graph	Execution, communication, scheduling and bookkeeping costs	Static	Image processing, speech recognition and compression
[43]	3D rendering	Communication and computation costs	Static	Games processing
[44]	Mobile phone sensor samples	Energy, latency and data traffic	Static	Social behavior
[45]	Functions based modules	Configuration based	Static	Natural language, speech processing and computer vision
[46]	Execution profiles	Computation cost and migration cost	Static	Virus scanning and image search
[47]	Analytical model	Surrogates coverage	Static	General applications
[48]	Performance history	Prediction errors	Static	General applications
[32]	Object relation graph	bandwidth, execution cost and data transfer	static	Dacapo benchmark

Table 1: Comparison of static offloading strategies

application execution, partitioning and migration of code. Initially, a Java application is partitioned by providing a set of min-cut partitioning. All the partitions are then evaluated by placing one node in first partition and all others in second partition. The nodes of second partition having the highest connectivity are moved to first partition iteratively. Subsequently, the minimum cut represents partitioning with the lowest inter-partition weight with respect to the communication cost between two partitions. For a diverse set of benchmarks including the *JavaNote* (text editor), *Biomer* (molecular editor) and *Voxel* (fractal landscape), the *AIDE* platform is shown to reduce the execution time significantly.

The framework *DiET* [41] is able to make modification to Java bytecode to support offloading of methods. The mobile users request to execute an application available through service providers. The client part of the application is downloaded to the mobile device. The complex computation based methods are modified with remote procedure calls in the client part. The server reads the requests and executes the code. Moreover, the automated offloading mechanism is portable and requires no special JVM dependent instructions. For the SciMark benchmark, the suggested approach is able to produce up to 59% of speedup for *MonteCarlo* integration method.

In [42], the authors target offloading in a wireless network from a mobile device to the mobile support station (MSS). It estimates the power consumption by the CPU in case of local execution and power consumption for data/results transfer to/from the remote server together with the response time for executing on the local machine and the MSS. If it is found beneficial to use the MSS, the jobs are offloaded. Consequently, there is a significant improvement in response time for execution of different jobs offloaded to the MSS.

The strategy proposed in [31] implements computation offloading by partitioning the code in client and server parts. A polynomial time algorithm is suggested to achieve optimal partitioning of code for a given set of input data. For a program, a control flow graph is built where each vertex is a basic block and each edge represents dependencies. A point-to analysis is then performed to identify the memory addresses or locations during data transfer. For distribution, various constraints are used to ensure data consistency. A cost analysis that takes into account the costs required for execution, scheduling, bookkeeping and communication is used to model the problem as a minimization problem. The problem is then represented as the min-cut network flow problem and is solved using an option-clustering heuristic. On an IPAQ 3970, and a Pentium-IV based server, the suggested offloading approach is able to reduce execution time for photo processing, graphics compression/de-compression, speech recognition and graph drawing benchmarks.

In [43], an approach for adapting the rendering settings for games in a mobile cloud is described. A static analysis is initially performed to select optimal settings for 3D rendering. These settings correspond to different adaptation levels where each level is associated with a total of communication and computation costs. During execution, an algorithm works to adjust the rendering settings in conformance with the existing communication and computation costs. For the game *PlaneShift* being played on a netbook, and using game server having GPU, the experimental results show an improvement in the performance in terms of the Game Mean Opinion Score (GMOS) corresponding to the gaming user experience.

A mobile phone based framework to capture the users' social behavior in a working environment is specified in [44]. The quantitative information such as the most sociable person in the environment and the number of interactions between two users have been useful for increasing productivity of organizations. To obtain such information, the mobile phone sensors are used to capture the behavior. The sensors sample the data at a specific rate. The samples are then processed to infer the required information. Due to the limited capability of the mobile devices, the processing is distributed among several devices. The decision of performing the computation locally or remotely is made by considering the parameters of energy, latency and data traffic. The overall task with these parameters is first divided into subtasks and a configuration for processing the task is found using the multi-criteria decision theory. With a Nokia 6120 mobile phone as a client and an Intel Xeon based server, the suggested approach is efficiently able to process the data and infer the required information.

An approach to partition the application for offloading using a language *Vivendi* is suggested in [45]. The language *Vivendi* is developed to describe the relevant specification of the application whose computation is to be offloaded. A file in the *Vivendi* language may contain the prototypes of functions that can be executed remotely. The next part of the approach incorporates Chroma [49] to monitor resources and predict the behavior. Subsequently, the stubs may be generated using the *Vivendi* stub generator and all function calls at corresponding points are replaced by calls to stubs. All the modules are then compiled and linked to generate an executable application. The suggested approach is able to support offloading for diverse applications including the natural language, speech and computer vision based applications.

The framework *CloneCloud* [46] facilitates the execution of a mobile application on the cloud. The *CloneCloud* initially partitions the application to make its parts execute on the mobile device and the cloud servers. A static offline analysis is performed to identify the partition. A dynamic profiler then generates profiles corresponding to different inputs. Consequently, a profile tree representing the execution traces is constructed. For each call of code, the computation cost and the migration cost in the case of local, remote or hybrid execution are computed. The optimization problem is then solved by minimizing these costs using an integer linear programming (ILP) solver. On an Android phone used as a client, and an Intel Xeon based server running mobile clones, the experimental results of clone execution show up to 20 times speedup for the applications including the virus scanning, image search and behavior profiling.

In [47], an analytical model is presented for analyzing the performance of offloading systems. The model takes into account the distribution of surrogates and shows that in the areas well covered by surrogates, the offloading may result in speedup in the performance. In contrast, the areas with less coverage of surrogates, the offloading does not improve the performance.

The framework *NWSLite* [48] is used for predicting the costs of location and remote execution. Its prediction model uses a non-parametric approach. The *NWSLite* framework incorporates a large number of models each with different parameterization. It forecasts measurements based on the performance history. The predictors are ranked with respect to the prediction errors and the best prediction model having the smallest prediction error. The *NWSLite* prediction models are executed in parallel thereby making it more efficient than the previously suggested LSQ [50] and RPF [51].

The authors in [32] aim at improving the execution performance by using the branch-and-bound and min-cut based approaches for partitioning mobile applications. It works by performing a static analysis & profiling, followed by the generation of a weighted object relation graph (WORG), which is used to represent the objects and relations between objects. The bandwidth parameter is then used together with the WORG to partition an application into client and server parts. The branch-and-bound based algorithm produces optimal partitioning results for small applications, whereas, the min-cut based approach works for large applications. Using a ThinkPad notebook for customized and the Dacapo suite benchmarks, the branch-and-bound and the min-cut based approaches produce speedups of 44.17% and 37.44%, respectively.

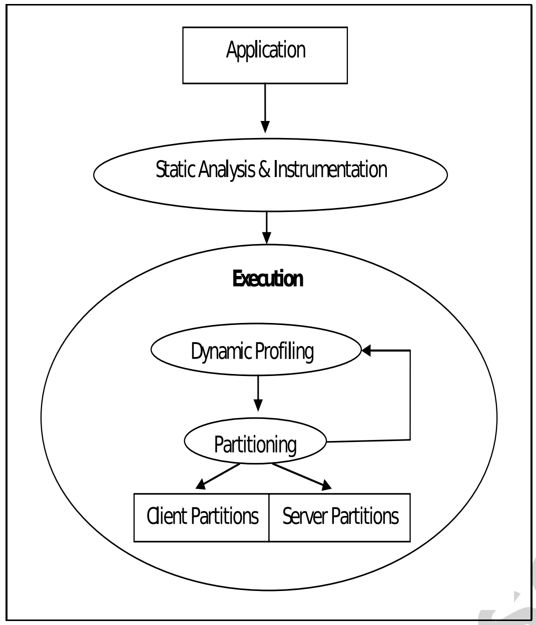


Figure 6: Dynamic Offloading Mechanism

4.2. Dynamic Offloading

As shown in Figure 6, the dynamic offloading strategies initially perform static analysis of the code and instrumentation in order to perform dynamic/online profiling during execution [52, 53, 54, 55, 56]. Based on the information obtained from dynamic profiling, the application is partitioned into client and server partitions. The execution then continues with the updated configuration.

A comparison of different dynamic offloading strategies is shown in Table 2. The comparison is performed in terms of core components, the parameters considered for offloading decision, the offloading approach and the benchmarks for which the strategy is shown to be beneficial.

In [52], the authors suggest to perform compression and de-compression operations simultaneously during computation offloading. For any application requiring the data to be transferred, it reduces the penalty of data transfer. Consequently, the application performance improves if the benefit produced by the data compression (in terms of the reduced number of packets) is higher than the overall cost of data compression and de-compression. The suggested approach is shown to be effective for making decision of Java code to be compiled and executed on remote server or locally.

With the notion of augmented execution, an application may be executed on some clones of a smartphone [53]. The runtime engine offloads the computation in a seamless way to another system that contains a clone of the entire system image. Consequently, the results may be integrated back to the smartphone. A special case of multiplicity based augmentation is presented that could work for performance improvement of data parallel applications. It requires multiple clones of the smartphone image. Similarly, a hardware based augmented execution is shown to improve the performance of scanning the file system.

In [54], the application partitioning is performed through a parametric analysis of the com-

Framework Contribution	Core Component	Parameters	Offloading Approach	Candidate Applications
[52]	Application code	Data transfer	Dynamic	Compilation and execution of Java code
[53]	System clones	Fixed configuration	Dynamic	Data parallel applications and file system scanning
[54]	Application graph	Computation, communication, registration and scheduling costs	Dynamic	FFT, encode and decode benchmarks
[57]	Application graph	Graph dependencies, network traffic, call delay and memory sizes	Dynamic	Image and text editors
[58]	Application code	Fixed configuration	Dynamic	Speech synthesis and MS-PowerPoint
[59]	Execution profile	Class usage and frequency	Dynamic	General applications
[60]	Multi-cost graph	Communication cost and class weight	Dynamic	Text recognition and translation
[29]	Real-time constraints	Network bandwidth and server speed	Dynamic	Real-time surveillance
[30]	Application profile	Energy consumption, bandwidth and latency	Dynamic	Face recognition and games
[61]	Application code	Safety for remote execution	Dynamic	Face recognition
[62]	Application profile	Application fidelities	Dynamic	General applications
[63]	Lookup service	Lookup latency	Dynamic	General applications
[28]	Estimation model	Network bandwidth and execution costs	Dynamic	General applications

Table 2: Comparison of dynamic offloading strategies

putation and communication costs. The problem of finding optimal partitioning is modelled as the min-cut network flow problem. The modules of the application distributed on the mobile device of the server depending upon the current value of runtime parameters. A program is first divided into modules or tasks that are executed on the server or the mobile device exclusively. A cost analysis then takes into account the computation, communication, task scheduling, and data registration costs and formulates the optimal partitioning as a single-source single-sink min-cut network flow problem. Using the mobile client HP IPAQ 3970, and a server machine having P4 processor, the results show that an effective partitioning significantly impacts the performance of several applications such as FFT, *encode* and *decode* from Mediabench and Minbench benchmarks.

An architecture of an inference engine is proposed in [57] for deciding the time of offloading and the application partition to be offloaded. The inference engine employs a fuzzy model and is implemented in the *AIDE* framework [40]. Each class of a Java application is represented as a node in a weighted graph. Each class is annotated with a flag describing whether or not the class may be offloaded to a server. The inference engine uses a min-cut based algorithm to find all 2-way cuts of the weighted graph. The nodes in the graph that may not be migrated to the surrogate are merged in the partition which will be executed on the mobile device. The other nodes are merged taking into account the dependencies and the metrics of network traffic, function call delay and memory size. The experiments performed for evaluation of an image editor, a text editor and a molecular editor show that the suggested approach minimizes the traffic requirements while working with a very small offloading overhead.

An automated approach of partitioning a Java application for remote execution is presented in [58]. A platform called *J-orchestra* is developed to perform replacement of the object code i.e. bytecode of method calls with the remote invocation. It divides an application into a client-server based model whose most of the I/O operations are performed on the client machine and the rest of the execution takes place on the server machine. With an iPAQ PDA, the J-orchestra has been shown to automatically distribute applications such as speech synthesis and MS PowerPoint.

The approach presented in [59] provides an adaptable offloading mechanism based on the application's execution behavior. A history of the execution pattern is maintained and is later used for making offloading decision. The static offloading policy offloads the most used classes, whereas the dynamic offloading moves only the invoked classes. The decision of offloading, i.e. *static*, *dynamic*, *no action*, or *profile* is made for each resource. Subsequently, the most common decision is opted for implementation. On PDAs, the offloading approach makes the application execute faster than local execution and is beneficial for applications with large execution times.

An offloading service for mobile handsets which may be used during mobility is presented in [60]. Initially, the resource information is collected and is followed by partitioning of application execution on the local system and the surrogate. The discovery of a suitable surrogate is made using the instantiation of classes for remote execution. The instrumented classes are then offloaded to the surrogates. The application partitioning uses a multi-cost graph, each of whose vertices is a class. The problem of graph partitioning is then solved by using a $k + 1$ partitioning algorithm. The proposed algorithm takes into account the weight of one class together with the weights of one-hop weights while minimizing the communication cost. On an HP iPAQ PDA, the suggested approach is applied to the *autoTranslator* software to recognize text in German language and translate it to English. The approach performs 3 to 5 times better than the randomly selected and the highest transfer rate based algorithms.

In [29], an approach for object recognition and tracing is presented, which may be used in the real-time surveillance systems. The approach performs computation offloading on the basis of real-time constraints. These constraints use various ranges of network bandwidth and server speed to make the offloading decision of executing code locally on a robot or remotely on a server.

The *MAUI* framework [30] supports fine-grained offloading of code in an automated way. To accomplish the portability of applications, two versions are created corresponding to execution on the mobile phone and the server. The *MAUI* architecture contains decision engine, proxy and profiler on both client and the server. The server part also contains the coordinator component to create an instance of the partitioned application. Initially, the methods to be offloaded are annotated by the programmer. These methods are identified by *MAUI* through Reflection API. Subsequently, the state of the application required for transfer or return to/from the server is identified. The *MAUI* profile provides feedback regarding energy consumption, bandwidth and latency etc. to the *MAUI* solver that in turn decides whether or not the code should be offloaded to the server. The solver models it as an optimization problem for minimizing the energy consumption subject to various latency constraints. Using *MAUI*, the code offloading for face recognition, video game and chess game is shown to improve the execution time.

The application partitioning by performing code analysis is suggested in [61]. The subtasks that are safe for remote execution are first identified. Subsequently, an analysis is performed to estimate the actual gains after offloading. Finally, two versions corresponding to execution on local and remote machine are generated. The suggested approach is implemented in the SUIF2 compiler [64], and is able to achieve almost 13 times and 15 times speedup in the performance of face recognition code on Skiff and iPAQ mobile appliances.

In [62], the architecture of a framework *Spectra* is presented. The *Spectra* framework does not require the application to describe the resources to be used, instead, it can predict the application behavior for future execution. It is implemented as part of the Aura framework [65] and uses the application fidelities as parameters to decide to perform execution on local and remote machines exclusively or hybridly. The CPU availability, network bandwidth, battery energy and data access costs are estimated by monitors to predict the application behavior. The *Spectra* framework then selects the best location and fidelity for application execution while taking as input the application description and the application behavior parameters. Using a Pocket PC with an SA-1100 processor as a client and an IBM T20 Laptop as a server, the *Spectra* framework is shown to select the best option for local, remote or hybrid execution.

In [63], two strategies of service discovery for offloading applications are presented. These strategies are based on flooding and unicasting. Every device is represented by a node and is associated to a lookup server that is used to store service description. When a service is required by a node, a service lookup is performed. The scope of the search (in terms of the area) for the server machine is increased gradually if no response is received from the lookup server. With flooding, the lookup message is broadcast, in contrast to unicast, which is useful for large environments. The experimental results show that the service discovery based approach for cyberaging applications is able to reduce the latency of the service lookup operation.

An approach for deciding offloading between the local and the remote system by making use of the bandwidth parameter is provided in [28]. The problem of estimating the local and remote execution costs is modelled as a statistical decision problem. The remote execution cost is computed as a function of the bandwidth available for transfer of data between the local and the remote systems. The Bayesian approach is then used to solve the problem and make the prediction regarding the offloading decision.

5. Application Partitioning For Computation Offloading

Together with the evolution of wireless technology, the research in the field of computation offloading has also evolved vigorously. As discussed earlier, an effective computation offloading technique may significantly impact the performance. The computation offloading incorporates various steps and analyses to ensure performance gain. One of the major steps used in computation offloading is application partitioning which distributes code for local and remote execution. The application partitioning may be categorized into static (application specific, framework based and offline profile based) and dynamic as shown in Table 3, and elaborated in this section.

Partitioning Category	References
Application specific static partitioning	[66], [67], [68], [69], [70], [71], [72], [73], [74], [75], [76], [77], [78]
Framework/API based static partitioning	[79], [80], [81], [82], [83], [84], [85], [86], [87], [88], [89], [90], [91]
Offline profile based static partitioning	[32], [92], [33], [93], [94], [95], [96], [97], [37], [98], [38], [99]
Dynamic partitioning	[100], [55], [56], [101], [102], [103], [104], [105], [106], [107], [100], [108], [109]

Table 3: Comparison of partitioning approaches adopted for computation offloading

5.1. *Static Partitioning*

For offloading computation to a remote machine, a static partitioning approach is adopted when an application's code modules are fixed to be executed on local or remote machines. The static partitioning may be implemented through an application specific, a framework based or an offline profile based strategy.

For a few partitioning strategies [66, 67, 68, 69], the parts of an application (such as AES encryption, image processing, multimedia services and Javascript code) are pre-defined to be executed on local or remote machines. These strategies set the portions of code depending upon the application. Similarly, for offloading strategies suggested in [70] and [71], the partitioning works for health related applications and frame-based tasks, respectively, whereas in [72, 73], various performance parameters are used to fix application based partitioning. The approach given in [74] uses mathematical model for improving face detection. For GPS services, the application specific partitioning uses signal processing stages and navigation methods [75], whereas for mobile games, fixed partitioning is adopted [76]. For surveillance system, a hierarchical partitioning approach is given in [78].

For some framework based strategies [81, 79, 85, 86, 88, 89, 87], the fixed partitioning mechanism is usually driven by programmers. In [90], an operating system to support distributed execution of java bytecode through static partitioning is described. The partitioning requires programmer annotations to decide the portions of code to be distributed. Similarly, the frameworks with fixed partitioning for collaborative or coalition based execution [91, 80] are also proposed. Different API functions to support offloading are suggested in [82]. The framework proposed in [83] requires the developer to annotate classes which must be offloaded. The offloading approach in [84] partitions the application into user interface and computation based components through the proposed framework.

The offline profile based static partitioning uses a set of parameters and evaluates them before actually executing the application. The application partitioning approach given in [32] uses branch-and-bound and min-cut based algorithms together with the bandwith parameter. Similarly, the genetic and machine-learning based approaches are suggested in [92, 96, 93] which take into account the resource status, network parameters and data to be transferred. In [33], the operations of a web service are profiled to generate a resource consumption profile which is subsequently used for performing computation offloading. For executing Javascript code, a profiler and a points-to analysis are suggested for helping developers to decide the portions of code to be offloaded [94]. The approach in [95] maps application partitioning as a minimization problem while taking into account performance estimate and communication cost. Similarly, a dynamic programming based algorithm [97] uses the estimated execution time for offloading tasks which satisfy a specific set of constraints. Other approaches adopted in [37, 98, 38, 99] also make use of similar parameters and conditions for partitioning applications for computation offloading.

5.2. Dynamic Partitioning

Many offloading strategies are able to adapt partitioning of code dynamically by taking into account several parameters [101, 102, 104]. These parameters are evaluated using profiling and performance prediction based mechanisms which manifest the possible behavior of an application. To profile execution of an application, the code is first instrumented and then analyzed for performance prediction.

In [100], a programming model with an event-driven approach for providing elastic execution of applications is suggested. Its dynamic migration mechanism distributes the execution among multiple nodes depending upon the workload requirements. A framework for dynamically adapting execution on a collection of smartphones is suggested in [55]. Similarly, the authors in [56] propose dynamic partitioning using genetic algorithm for mobile data streams. The approach proposed in [103] initially detects movable classes and then offloads by profiling classes during execution. In [105, 106], the partitioning is mapped to min-cut problem, whereas, a few components are replicated for minimizing component migration at runtime. Other offloading frameworks and mechanisms [107, 100, 108, 109] use online profiles while considering various parameters for performing code partitioning dynamically.

6. Comparison of Offloading Frameworks

Table 4 describes a comparison of the automated offloading frameworks in terms of the parameters of automation, optimization problem solving, replication granularity, fine-grained offloading and native method call support. For automation, the frameworks *CloneCloud*, *Spectra*, *Roam* and *J-Orchestra* provide offloading in a highly automated manner. This requires less interaction of the programmer as compared to those having low automated offloading support. Similarly, the frameworks *CloneCloud*, *AIDE*, and *J-Orchestra* solve the optimization problem in a highly asynchronous manner with regards to execution of the application. The replication granularity refers to the main component that is replicated or transferred for remote execution. The fine-grained component support is provided in the *CloneCloud* and *MAUI* frameworks. Moreover, a few frameworks including the *CloneCloud*, framework in [110], *AIDE* and *J-Orchestra* also support native method calls.

A comparison of the working mechanism in terms of the analysis performed, dynamic profiling, late binding and trusted execution of the automated frameworks is given in Table 5. All the frameworks make use of a static analysis which is performed before execution of the application. The frameworks *CloneCloud*, *MAUI*, *Roam* and *AIDE* incorporate dynamic profiling to obtain

Framework	Automation	Optimization Problem Solving	Replication Granularity	Fine-grained	Native Method Call
CloneCloud [46]	High	Highly Asynchronous	Partial Threads	Yes	Yes
MAUI [30]	Low	Low Asynchronous	Low-level (fine-grained)	Yes	No
SociableSense [44]	Low	Asynchronous	Module-level	No	No
Spectra [62]	High	Asynchronous	Task-level	No	No
Framework in [110]	Medium	Asynchronous	Components	No	Yes
Roam [35]	High	Asynchronous	Component/Roamlet	No	No
AIDE [40]	Medium	Highly Asynchronous	Class	No	Yes
DiET [41]	Medium	Asynchronous	Class methods	No	No
J-Orchestra [58]	High	Highly Asynchronous	Class methods	No	Yes

Table 4: Comparison of the automation, optimization problem solving, replication granularity, fine-grained and native method call support based characteristics of the offloading frameworks

Framework	Static Analysis	Dynamic Profiling	Late binding (offloading)	Trusted execution
CloneCloud [46]	Yes	Yes	Yes	No
MAUI [30]	Yes	Yes	Yes	No
SociableSense [44]	Yes	No	Yes	No
Spectra [62]	Yes	No	No	No
Framework in [110]	Yes	No	Yes	No
Roam [35]	Yes	Yes	Yes	No
AIDE [40]	Yes	Yes	Yes	No
DiET [41]	Yes	No	No	No
J-Orchestra [58]	Yes	No	No	No

Table 5: Comparison of the static analysis, dynamic profiling, late binding and trusted execution based characteristics of the offloading frameworks

Framework	Applications	Trade-off Parameters	Optimization Strategy	Dynamic Adaptation Strategy
CloneCloud [46]	Scientific	Execution speed, energy and data transfer	Integer Linear Programming (ILP)	Profile tree based
MAUI [30]	Scientific	Energy & execution speed with data transfer	0-1 ILP	Call graph based
SociableSense [44]	Social Interaction	Accuracy, energy, latency and data traffic	Multi-criteria decision theory	Learning based
Spectra [62]	Voice recognition	Latency, battery life and fidelity	Fidelity solver	None
Framework in [110]	Language translation & Character recognition	Response time, communication, CPU and memory	$(k+1)$ partitioning algorithm	Speedup based
Roam [35]	Games & Graphics	Capabilities of target devices and user interface design	Component-based partitioning	Target device capabilities based mechanism
AIDE [40]	Image and text processing	Processor load, memory and communication	Min-cut based heuristic	Execution graph based
DiET [41]	Mathematical applications	User directives based	User configuration based	User configuration based
J-Orchestra [58]	General applications	Input/output, disk processing and GUI	User directives based parameters of I/O usage	None

Table 6: Comparison of the applications, trade-off parameters, optimization and dynamic adaptation mechanisms of the offloading frameworks

information during execution of the application and perform adaptation accordingly. The late binding for offloading refers to the offloading implemented at a later time during execution of the application. It is performed by the *CloneCloud*, *MAUI*, *SociableSense*, [110], *Roam* and *AIDE* frameworks. Currently, none of these frameworks ensures a trusted execution to provide secure, reliable and authenticated access for offloaded applications.

Table 6 provides a comparison of the offloading frameworks in terms of their applications, trade-off parameters, optimization and dynamic adaptation strategies. The *CloneCloud*, *MAUI*, *DiET* and *J-Orchestra* are useful for general scientific applications, whereas the frameworks *Roam* and *AIDE* are shown to be effective for image and graphics processing. Similarly, the framework in [110] and Spectra are shown to work on voice and character recognition based applications. The *SociableSense* is specific for applications requiring processing on social interaction in an organization. The trade-off parameters are the elements considered while optimizing the offloading decision. In general, most of the frameworks use the execution time, energy consumption and communication overhead as the main trade-off parameters. While optimizing the decision problem, different heuristics based on the min-cut, *k+1* partitioning, and integer linear programming (ILP) are used in most of the offloading frameworks. The frameworks also require dynamic adaptation for offloading decisions during execution of the application. The *CloneCloud*, *MAUI* and *AIDE* frameworks use execution pattern for runtime adaptation. Similarly, the framework in [110] performs adaptation using the speedup obtained through offloading. The *Roam* framework uses the target device platform based runtime adaptation, whereas the

Multimedia	[26], [54], [39]
Games	[26], [35], [30], [43]
Graphics and image processing	[27], [52], [48], [40], [57], [60], [31], [43], [45], [46]
Mathematical computations	[54], [52], [53], [39], [41], [31]
Artificial Intelligence based applications	[52], [58], [60], [29], [30], [61], [31], [62], [45], [46]
Health & Social applications	[111], [112], [44]
Database, file system or GPS Processing	[52], [53], [82]

Table 7: Domain-wise categorization of the research work related to computation offloading

DiET framework requires user configuration for runtime adaptation.

7. Application Domains Benefiting From Offloading

The computation offloading has proved to be beneficial for a large number of applications lying in several domains. A domain-wise categorization of research work is shown in Table 7. A large part of the research work has targeted the applications lying in the domains of mathematics and graphics/image processing. Likewise, the games and multimedia based applications are also targeted and their number continues to grow together with the evolution of wireless technology. The applications related to Artificial Intelligence and social behavior are also being offloaded as they involve complex learning based computations. The applications with database processing, file system and GPS processing have also been implemented through offloading to improve their performance.

8. Current Challenges For Effective Computation Offloading

Despite the long term evolution of the offloading techniques, several issues are yet to be resolved. The most challenging issues including partitioning, automated transparency & portability, security, and application requirements are discussed below together with their possible solutions.

8.1. Partitioning

The computation offloading requires the application code to be partitioned into client and server parts for local and remote execution, respectively. The partitioning takes into account several parameters including costs of data transfer and computation time, however the optimal partitioning is an NP-complete problem. Consequently, different heuristics with fixed constraints are employed in the partitioning strategies.

For an effective offloading implementation, the partitioning problem needs to be solved in a quasi-automated manner requiring directives from the programmer as well as automated distribution of modules. In this regard, the scheduling techniques for heterogeneous systems [113, 114, 115] may be incorporated to minimize the total execution time.

8.2. Automated Transparency & Portability

The frameworks implemented for computation offloading yet lack the automated transparency so that the surrounding environment is detected and the computation offloading takes place in a seamless manner [12, 4, 11, 100, 84]. This is a complex task as it requires an implementation of a standard protocol that will perform lookup services and other functionalities depending upon the environment while taking into account its constraints. An implementation of the standard protocol for a diverse collection of devices and environments will render it portability as well.

8.3. Security

With computations being offloaded to remote machines/servers, the security of data and environment for the remote systems needs to be ensured [116, 4, 7, 14, 117, 77]. This requires restraining the types of operations that may be offloaded for remote execution. A limited set of permissible operations may be provided by implementing a virtual machine and making the remote component execute in the environment provided by the virtual machine [118]. Moreover, different authorization and authentication mechanisms may be incorporated in order to ensure security of data on the cloud [119, 120].

8.4. Application Requirements

The applications being executed on mobile devices are not only growing in size but also in terms of complex operations. The widely used multimedia applications including the VoIP, online streaming, and video/audio chat require the mobile devices to improve the energy requirements, graphics rendering and the execution time. Moreover, these applications require real-time processing. Consequently, it is not possible to offload all the modules remotely. In this regard, the caching techniques and implementation of a specialized hardware such as a Digital Signal Processor (DSP) [121] or a System-on-Chip (SoC) [122] may be beneficial for an effective offloading.

9. Conclusion

This paper presents a comprehensive survey of the research work conducted on computation offloading which aims at performance improvement of applications executing on the resource constrained mobile devices. The limited resources of mobile devices require the intensive computations to be offloaded in order to mitigate the issues of slow execution and low energy. Some of the offloading strategies work in a fixed static manner while others are able to perform offloading in accordance with the dynamic behavior of the application. We perform a comparative analysis of these strategies as well as the automated frameworks implemented to support computation offloading.

We also survey the evolution of mobile technologies and also compare different partitioning mechanisms used for distributing code between local and remote machines. The research work is also categorized in terms of the application domains for which the computation offloading is shown to be effective. Moreover, the main issues related to computation offloading: partitioning, automated transparency & portability, security, and application requirements are discussed, and their possible solutions are also proposed.

[1] StatCounter, Comparison from 2011 to 2014 — StatCounter – Global Stats (2014).
URL http://gs.statcounter.com/#desktop+mobile+tablet-comparison-ww-yearly-2011-2014

[2] H. T. Dinh, C. Lee, D. Niyato, P. Wang, A survey of mobile cloud computing: architecture, applications, and approaches, Wireless Communications and Mobile Computing 13 (18) (2013) 1587–1611. doi:10.1002/wcm.1203. URL `http://dx.doi.org/10.1002/wcm.1203`

[3] K. Kumar, Y.-H. Lu, Cloud computing for mobile users: Can offloading computation save energy?, Computer 43 (4) (2010) 51–56.

[4] Z. Sanaei, S. Abolfazli, A. Gani, R. Buyya, Heterogeneity in mobile cloud computing: Taxonomy and open challenges, Communications Surveys Tutorials, IEEE 16 (1) (2014) 369–392. doi:10.1109/SURV.2013.050113.00090.

[5] R. Fontana, K. Mparmpopoulou, P. Grosso, K. Van der Veldt, Monitoring greenclouds: Evaluating the trade-off between performance and energy consumption in das-4 (2013).

[6] A. Juntunen, M. Kemppainen, S. Luukkainen, Mobile computation offloading - factors affecting technology evolution, in: 2012 International Conference on Mobile Business (ICMB 2012); Delft, The Netherlands; 21-22 June, 2012, 2012, p. 12.

[7] A. u. R. Khan, M. Othman, S. A. Madani, S. U. Khan, A survey of mobile cloud computing application models, Communications Surveys Tutorials, IEEE 16 (1) (2014) 393–413.

[8] A. Berl, E. Gelenbe, M. Di Girolamo, G. Giuliani, H. De Meer, M. Q. Dang, K. Pentikousis, Energy-efficient cloud computing, Comput. J. 53 (7) (2010) 1045–1051. doi:10.1093/comjnl/bxp080. URL `http://dx.doi.org/10.1093/comjnl/bxp080`

[9] M. Barbera, S. Kosta, A. Mei, J. Stefa, To offload or not to offload? the bandwidth and energy costs of mobile cloud computing, in: INFOCOM, 2013 Proceedings IEEE, 2013, pp. 1285–1293. doi:10.1109/INFCOM.2013.6566921.

[10] S. Ou, K. Yang, A. Liotta, L. Hu, Performance analysis of offloading systems in mobile wireless environments, in: Communications, 2007. ICC '07. IEEE International Conference on, 2007, pp. 1821–1826. doi:10.1109/ICC.2007.304.

[11] Y. Cui, X. Ma, H. Wang, I. Stojmenovic, J. Liu, A survey of energy efficient wireless transmission and modeling in mobile cloud computing, Mob. Netw. Appl. 18 (1) (2013) 148–155. doi:10.1007/s11036-012-0370-6. URL `http://dx.doi.org/10.1007/s11036-012-0370-6`

[12] Z. Sanaei, S. Abolfazli, A. Gani, R. H. Khokhar, Tripod of requirements in horizontal heterogeneous mobile cloud computing, CoRR abs/1205.3247.

[13] A. P. Miettinen, V. Hirvisalo, Energy-efficient parallel software for mobile hand-held devices, in: Proceedings of the First USENIX Conference on Hot Topics in Parallelism, HotPar'09, USENIX Association, Berkeley, CA, USA, 2009, pp. 12–12.

[14] K. Kumar, J. Liu, Y.-H. Lu, B. Bhargava, A survey of computation offloading for mobile systems, Mob. Netw. Appl. 18 (1) (2013) 129–140.

[15] M. Satyanarayanan, P. Bahl, R. Caceres, N. Davies, The case for vm-based cloudlets in mobile computing, Pervasive Computing, IEEE 8 (4) (2009) 14–23.

[16] Y. Lu, B. Zhou, L.-C. Tung, M. Gerla, A. Ramesh, L. Nagaraja, Energy-efficient content retrieval in mobile cloud, in: Proceedings of the Second ACM SIGCOMM Workshop on Mobile Cloud Computing, MCC '13, ACM, New York, NY, USA, 2013, pp. 21–26.

[17] Y.-J. Hong, K. Kumar, Y.-H. Lu, Energy efficient content-based image retrieval for mobile systems, in: Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on, 2009, pp. 1673–1676.

[18] Y. Wen, W. Zhang, H. Luo, Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones, in: INFOCOM, 2012 Proceedings IEEE, 2012, pp. 2716–2720.

[19] A. Rudenko, P. Reiher, G. J. Popek, G. H. Kuenning, Saving portable computer battery power through remote process execution, SIGMOBILE Mob. Comput. Commun. Rev. 2 (1) (1998) 19–26. doi:10.1145/584007.584008. URL `http://doi.acm.org/10.1145/584007.584008`

[20] J. Nurminen, Parallel connections and their effect on the battery consumption of a mobile phone, in: Consumer Communications and Networking Conference (CCNC), 2010 7th IEEE, 2010, pp. 1–5.

[21] Y. Nimmagadda, K. Kumar, Y.-H. Lu, Energy-efficient image compression in mobile devices for wireless transmission, in: Proceedings of the 2009 IEEE International Conference on Multimedia and Expo, ICME'09, IEEE Press, Piscataway, NJ, USA, 2009, pp. 1278–1281.

[22] A. P. Miettinen, J. K. Nurminen, Energy efficiency of mobile clients in cloud computing, in: Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing, HotCloud'10, USENIX Association, Berkeley, CA, USA, 2010, pp. 4–4.

[23] R. N. Mayo, P. Ranganathan, Energy consumption in mobile devices: Why future systems need requirements-aware energy scale-down, in: Proceedings of the Third International Conference on Power - Aware Computer Systems, PACS'03, Springer-Verlag, Berlin, Heidelberg, 2004, pp. 26–40.

[24] K. Sinha, M. Kulkarni, Techniques for fine-grained, multi-site computation offloading, in: Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID '11, IEEE Computer Society, Washington, DC, USA, 2011, pp. 184–194. doi:10.1109/CCGrid.2011.69. URL `http://dx.doi.org/10.1109/CCGrid.2011.69`

[25] Y. Ge, Y. Zhang, Q. Qiu, Y.-H. Lu, A game theoretic resource allocation for overall energy minimization in mobile cloud computing system, in: Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design, ISLPED '12, ACM, New York, NY, USA, 2012, pp. 279–284.

[26] Z. Li, C. Wang, R. Xu, Computation offloading to save energy on handheld devices: A partition scheme, in: Proceedings of the 2001 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, CASES '01, ACM, New York, NY, USA, 2001, pp. 238–246.

[27] C. Xian, Y.-H. Lu, Z. Li, Adaptive computation offloading for energy conservation on battery-powered systems, in: Parallel and Distributed Systems, 2007 International Conference on, Vol. 2, 2007, pp. 1–8.

[28] R. Wolski, S. Gurun, C. Krintz, D. Nurmi, Using bandwidth data to make computation offloading decisions, in: Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on, 2008, pp. 1–8. doi:10.1109/IPDPS.2008.4536215.

[29] Y. Nimmagadda, K. Kumar, Y.-H. Lu, C. S. G. Lee, Real-time moving object recognition and tracking using computation offloading, in: Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on, 2010, pp. 2449–2455.

[30] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, P. Bahl, MAUI: making smartphones last longer with code offload, in: Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services, MobiSys '10, ACM, New York, NY, USA, 2010, pp. 49–62. doi:10.1145/1814433.1814441.
URL http://doi.acm.org/10.1145/1814433.1814441

[31] C. Wang, Z. Li, A computation offloading scheme on handheld devices, J. Parallel Distrib. Comput. 64 (6) (2004) 740–746. doi:10.1016/j.jpdc.2003.10.005.
URL http://dx.doi.org/10.1016/j.jpdc.2003.10.005

[32] J. Niu, W. Song, M. Atiquzzaman, Bandwidth-adaptive partitioning for distributed execution optimization of mobile applications, J. Netw. Comput. Appl. 37 (2014) 334–347. doi:10.1016/j.jnca.2013.03.007.
URL http://dx.doi.org/10.1016/j.jnca.2013.03.007

[33] K. Elgazzar, P. Martin, H. S. Hassanein, Empowering mobile service provisioning through cloud assistance, in: Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing, UCC '13, IEEE Computer Society, Washington, DC, USA, 2013, pp. 9–16.

[34] N. Balasubramanian, A. Balasubramanian, A. Venkataramani, Energy consumption in mobile phones: A measurement study and implications for network applications, in: Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference, IMC '09, ACM, New York, NY, USA, 2009, pp. 280–293.

[35] H.-h. Chu, H. Song, C. Wong, S. Kurakake, M. Katagiri, ROAM, a seamless application framework, J. Syst. Softw. 69 (3) (2004) 209–226.

[36] Y. Ding, Z. Li, A compiler analysis of interprocedural data communication, in: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing, SC '03, ACM, New York, NY, USA, 2003, pp. 11–.

[37] S. Gurun, R. Wolski, C. Krintz, D. Nurmi, On the efficacy of computation offloading decision-making strategies, Int. J. High Perform. Comput. Appl. 22 (4) (2008) 460–479.

[38] S. Ou, K. Yang, J. Zhang, An effective offloading middleware for pervasive services on mobile devices, Pervasive Mob. Comput. 3 (4) (2007) 362–385.

[39] S. Ou, K. Yang, A. 'Liotta, An adaptive multi-constraint partitioning algorithm for offloading in pervasive systems, in: Proceedings of the Fourth Annual IEEE International Conference on Pervasive Computing and Communications, PERCOM '06, IEEE Computer Society, Washington, DC, USA, 2006, pp. 116–125.

[40] A. Messer, I. Greenberg, P. Bernadat, D. Milojicic, D. Chen, T. J. Giuli, X. Gu, Towards a distributed platform for resource-constrained devices, in: Proceedings of the 22 Nd International Conference on Distributed Computing Systems (ICDCS'02), ICDCS '02, IEEE Computer Society, Washington, DC, USA, 2002, pp. 43–51.

[41] H. Rim, S. Kim, Y. Kim, H. Han, Transparent method offloading for slim execution, in: Wireless Pervasive Computing, 2006 1st International Symposium on, 2006, pp. 1–6.

[42] M. Othman, S. Hailes, Power conservation strategy for mobile computers using load sharing, SIGMOBILE Mob. Comput. Commun. Rev. 2 (1) (1998) 44–51. doi:10.1145/584007.584011.
URL http://doi.acm.org/10.1145/584007.584011

[43] S. Wang, S. Dey, Rendering adaptation to address communication and computation constraints in cloud mobile gaming, in: Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE, 2010, pp. 1–6.

[44] K. K. Rachuri, C. Mascolo, M. Musolesi, P. J. Rentfrow, Sociablesense: Exploring the trade-offs of adaptive sampling and computation offloading for social sensing, in: Proceedings of the 17th Annual International Conference on Mobile Computing and Networking, MobiCom '11, ACM, New York, NY, USA, 2011, pp. 73–84.

[45] R. K. Balan, D. Gergle, M. Satyanarayanan, J. Herbsleb, Simplifying cyber foraging for mobile devices, in: Proceedings of the 5th International Conference on Mobile Systems, Applications and Services, MobiSys '07, ACM, New York, NY, USA, 2007, pp. 272–285.

[46] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, A. Patti, CloneCloud: elastic execution between mobile device and

cloud, in: Proceedings of the Sixth Conference on Computer Systems, EuroSys '11, ACM, New York, NY, USA, 2011, pp. 301–314.

[47] S. Ou, K. Yang, A. Liotta, L. Hu, Performance analysis of offloading systems in mobile wireless environments, in: Communications, 2007. ICC '07. IEEE International Conference on, 2007, pp. 1821–1826.

[48] S. Gurun, C. Krintz, R. Wolski, NWSLite: a light-weight prediction utility for mobile devices, in: Proceedings of the 2Nd International Conference on Mobile Systems, Applications, and Services, MobiSys '04, ACM, New York, NY, USA, 2004, pp. 2–11.

[49] R. K. Balan, M. Satyanarayanan, S. Y. Park, T. Okoshi, Tactics-based remote execution for mobile computing, in: Proceedings of the 1st International Conference on Mobile Systems, Applications and Services, MobiSys '03, ACM, New York, NY, USA, 2003, pp. 273–286.

[50] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, K. R. Walker, Agile application-aware adaptation for mobility, SIGOPS Oper. Syst. Rev. 31 (5) (1997) 276–287.

[51] A. Rudenko, P. Reiher, G. J. Popek, G. H. Kuenning, The remote processing framework for portable computer power saving, in: Proceedings of the 1999 ACM Symposium on Applied Computing, SAC '99, ACM, New York, NY, USA, 1999, pp. 365–372.

[52] G. Chen, B.-T. Kang, M. Kandemir, N. Vijaykrishnan, M. Irwin, R. Chandramouli, Studying energy trade offs in offloading computation/compilation in java-enabled mobile devices, Parallel and Distributed Systems, IEEE Transactions on 15 (9) (2004) 795–809.

[53] B.-G. Chun, P. Maniatis, Augmented smartphone applications through clone cloud execution, in: Proceedings of the 12th Conference on Hot Topics in Operating Systems, HotOS'09, USENIX Association, Berkeley, CA, USA, 2009, pp. 8–8.

[54] C. Wang, Z. Li, Parametric analysis for adaptive computation offloading, SIGPLAN Not. 39 (6) (2004) 119–130.

[55] R.-C. Marin, Hybrid contextual cloud in ubiquitous platforms comprising of smartphones, Int. J. Intell. Syst. Technol. Appl. 12 (1) (2013) 4–17.

[56] L. Yang, J. Cao, Y. Yuan, T. Li, A. Han, A. Chan, A framework for partitioning and execution of data stream applications in mobile cloud computing, SIGMETRICS Perform. Eval. Rev. 40 (4) (2013) 23–32.

[57] X. Gu, K. Nahrstedt, A. Messer, I. Greenberg, D. Milojicic, Adaptive offloading inference for delivering applications in environments, in: Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, PERCOM '03, IEEE Computer Society, Washington, DC, USA, 2003, pp. 107–.

[58] E. Tilevich, Y. Smaragdakis, J-Orchestra: automatic java application partitioning., in: B. Magnusson (Ed.), ECOOP, Vol. 2374 of Lecture Notes in Computer Science, Springer, 2002, pp. 178–204.

[59] G. Huerta-Canepa, D. Lee, An adaptable application offloading scheme based on application behavior, in: Advanced Information Networking and Applications - Workshops, 2008. AINAW 2008. 22nd International Conference on, 2008, pp. 387–392.

[60] K. Yang, S. Ou, H.-H. Chen, On effective offloading services for resource-constrained mobile devices running heavier mobile internet applications, Communications Magazine, IEEE 46 (1) (2008) 56–63.

[61] U. Kremer, J. Hicks, J. Rehg, A compilation framework for power and energy management on mobile computers, in: Proceedings of the 14th International Conference on Languages and Compilers for Parallel Computing, LCPC'01, Springer-Verlag, Berlin, Heidelberg, 2003, pp. 115–131.

[62] J. Flinn, D. Narayanan, M. Satyanarayanan, Self-tuned remote execution for pervasive computing, in: Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on, 2001, pp. 61–66.

[63] S. Sivavakeesar, O. Gonzalez, G. Pavlou, Service discovery strategies in ubiquitous communication environments, Communications Magazine, IEEE 44 (9) (2006) 106–113. doi:10.1109/MCOM.2006.1705986.

[64] G. Aigner, A. Diwan, D. Heine, M. Lam, D. Moore, B. Murphy, C. Sapuntzakis, The suif2 compiler infrastructure (2000).

[65] J. a. P. Sousa, D. Garlan, Aura: An architectural framework for user mobility in ubiquitous computing environments, in: Proceedings of the IFIP 17th World Computer Congress - TC2 Stream / 3rd IEEE/IFIP Conference on Software Architecture: System Design, Development and Maintenance, WICSA 3, Kluwer, B.V., Deventer, The Netherlands, The Netherlands, 2002, pp. 29–43.
URL http://dl.acm.org/citation.cfm?id=646546.693943

[66] H. Jo, S.-T. Hong, J.-W. Chang, D. H. Choi, Offloading data encryption to gpu in database systems, J. Supercomput. 69 (1) (2014) 375–394.

[67] W. Liu, J.-J. Chen, A. Toma, T.-W. Kuo, Q. Deng, Computation offloading by using timing unreliable components in real-time systems, in: Proceedings of the The 51st Annual Design Automation Conference on Design Automation Conference, DAC '14, ACM, New York, NY, USA, 2014, pp. 39:1–39:6.

[68] D. Kovachev, Y. Cao, R. Klamma, Building mobile multimedia services: A hybrid cloud computing approach, Multimedia Tools Appl. 70 (2) (2014) 977–1005.

[69] S. Park, Q. Chen, H. Han, H. Y. Yeom, Design and evaluation of mobile offloading system for web-centric devices, J. Netw. Comput. Appl. 40 (2014) 105–115. doi:10.1016/j.jnca.2013.08.006.

URL http://dx.doi.org/10.1016/j.jnca.2013.08.006

[70] R. K. Lomotey, R. Deters, Reliable services composition for mobile consumption in mhealth, in: Proceedings of the Fifth International Conference on Management of Emergent Digital EcoSystems, MEDES ’13, ACM, New York, NY, USA, 2013, pp. 182–186.

[71] A. Toma, J.-J. Chen, Computation offloading for frame-based real-time tasks with resource reservation servers, in: Proceedings of the 2013 25th Euromicro Conference on Real-Time Systems, ECRTS ’13, IEEE Computer Society, Washington, DC, USA, 2013, pp. 103–112.

[72] H. Eom, P. St Juste, R. Figueiredo, O. Tickoo, R. Illikkal, R. Iyer, Snarf: A social networking-inspired accelerator remoting framework, in: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, MCC ’12, ACM, New York, NY, USA, 2012, pp. 29–34.

[73] D. Kovachev, T. Yu, R. Klamma, Adaptive computation offloading from mobile devices into the cloud, in: Proceedings of the 2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications, ISPA ’12, IEEE Computer Society, Washington, DC, USA, 2012, pp. 784–791.

[74] S. Imai, C. A. Varela, Light-weight adaptive task offloading from smartphones to nearby computational resources, in: Proceedings of the 2011 ACM Symposium on Research in Applied Computation, RACS ’11, ACM, New York, NY, USA, 2011, pp. 146–152.

[75] H. S. Ramos, T. Zhang, J. Liu, N. B. Priyantha, A. Kansal, Leap: A low energy assisted gps for trajectory-based services, in: Proceedings of the 13th International Conference on Ubiquitous Computing, UbiComp ’11, ACM, New York, NY, USA, 2011, pp. 335–344.

[76] J. Liu, K. Kumar, Y.-H. Lu, Tradeoff between energy savings and privacy protection in computation offloading, in: Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design, ISLPED ’10, ACM, New York, NY, USA, 2010, pp. 213–218.

[77] X. Zhang, J. Schiffman, S. Gibbs, A. Kunjithapatham, S. Jeong, Securing elastic applications on mobile devices for cloud computing, in: Proceedings of the 2009 ACM Workshop on Cloud Computing Security, CCSW ’09, ACM, New York, NY, USA, 2009, pp. 127–134.

[78] V. Tsiatsis, R. Kumar, M. B. Srivastava, Computation hierarchy for in-network processing, Mob. Netw. Appl. 10 (4) (2005) 505–518.

[79] F. Xia, F. Ding, J. Li, X. Kong, L. T. Yang, J. Ma, Phone2cloud: Exploiting computation offloading for energy saving on smartphones in mobile cloud computing, Information Systems Frontiers 16 (1) (2014) 95–111.

[80] S. Kurkovsky, Bhagyavati, A. Ray, A collaborative problem-solving framework for mobile devices, in: Proceedings of the 42Nd Annual Southeast Regional Conference, ACM-SE 42, ACM, New York, NY, USA, 2004, pp. 5–10.

[81] A. Gangil, S. K. Dhurandher, M. S. Obaidat, V. Singh, S. Bhatia, Moclo: A cloud framework for mobile devices, in: Proceedings of the 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing, GREENCOM-ITHINGS-CPSCOM ’13, IEEE Computer Society, Washington, DC, USA, 2013, pp. 632–637.

[82] A. Mtibaa, A. Fahim, K. A. Harras, M. H. Ammar, Towards resource sharing in mobile device clouds: Power balancing across mobile devices, SIGCOMM Comput. Commun. Rev. 43 (4) (2013) 51–56.

[83] T. Verbelen, P. Simoens, F. De Turck, B. Dhoedt, Aiolos: Middleware for improving mobile application performance through cyber foraging, J. Syst. Softw. 85 (11) (2012) 2629–2639.

[84] M. S. Gordon, D. A. Jamshidi, S. Mahlke, Z. M. Mao, X. Chen, Comet: Code offload by migrating execution transparently, in: Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation, OSDI’12, USENIX Association, Berkeley, CA, USA, 2012, pp. 93–106.

[85] A. Saarinen, M. Siekkinen, Y. Xiao, J. K. Nurminen, M. Kemppainen, P. Hui, Smartdiet: Offloading popular apps to save energy, SIGCOMM Comput. Commun. Rev. 42 (4) (2012) 297–298.

[86] X. Zhang, A. Kunjithapatham, S. Jeong, S. Gibbs, Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing, Mob. Netw. Appl. 16 (3) (2011) 270–284.

[87] A. Saarinen, M. Siekkinen, Y. Xiao, J. K. Nurminen, M. Kemppainen, P. Hui, Can offloading save energy for popular apps?, in: Proceedings of the Seventh ACM International Workshop on Mobility in the Evolving Internet Architecture, MobiArch ’12, ACM, New York, NY, USA, 2012, pp. 3–10.

[88] J. a. N. Silva, L. Veiga, P. Ferreira, Spade: Scheduler for parallel and distributed execution from mobile devices, in: Proceedings of the 6th International Workshop on Middleware for Pervasive and Ad-hoc Computing, MPAC ’08, ACM, New York, NY, USA, 2008, pp. 25–30.

[89] Y. Ni, U. Kremer, A. Stere, L. Iftode, Programming ad-hoc networks of mobile and resource-constrained devices, SIGPLAN Not. 40 (6) (2005) 249–260.

[90] H. Liu, T. Roeder, K. Walsh, R. Barr, E. G. Sirer, Design and implementation of a single system image operating system for ad hoc networks, in: Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services, MobiSys ’05, ACM, New York, NY, USA, 2005, pp. 149–162.

[91] L. Nogueira, L. M. Pinho, Dynamic qos-aware coalition formation, in: Proceedings of the 19th IEEE Interna-

tional Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 2 - Volume 03, IPDPS '05, IEEE Computer Society, Washington, DC, USA, 2005, pp. 135.1–.

[92] P. Balakrishnan, C.-K. Tham, Energy-efficient mapping and scheduling of task interaction graphs for code offloading in mobile cloud computing, in: Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing, UCC '13, IEEE Computer Society, Washington, DC, USA, 2013, pp. 34–41.

[93] H. Eom, P. S. Juste, R. Figueiredo, O. Tickoo, R. Illikkal, R. Iyer, Machine learning-based runtime scheduler for mobile offloading framework, in: Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing, UCC '13, IEEE Computer Society, Washington, DC, USA, 2013, pp. 17–25.

[94] X. Wang, X. Liu, G. Huang, Y. Liu, Appmobicloud: Improving mobile web applications by mobile-cloud convergence, in: Proceedings of the 5th Asia-Pacific Symposium on Internetware, Internetware '13, ACM, New York, NY, USA, 2013, pp. 14:1–14:10.

[95] J. H. Ahnn, M. Potkonjak, mHealthMon: Toward energy-efficient and distributed mobile health monitoring using parallel offloading, J. Med. Syst. 37 (5) (2013) 1–11.

[96] G. Folino, F. S. Pisani, A framework for modeling automatic offloading of mobile applications using genetic programming, in: Proceedings of the 16th European Conference on Applications of Evolutionary Computation, EvoApplications'13, Springer-Verlag, Berlin, Heidelberg, 2013, pp. 62–71.

[97] A. Toma, J.-J. Chen, Computation offloading for real-time systems, in: Proceedings of the 28th Annual ACM Symposium on Applied Computing, SAC '13, ACM, New York, NY, USA, 2013, pp. 1650–1651.

[98] S. Han, S. Zhang, J. Cao, Y. Wen, Y. Zhang, A resource aware software partitioning algorithm based on mobility constraints in pervasive grid environments, Future Gener. Comput. Syst. 24 (6) (2008) 512–529.

[99] Z. Li, C. Wang, R. Xu, Task allocation for distributed multimedia processing on wirelessly networked handheld devices, in: Proceedings of the 16th International Parallel and Distributed Processing Symposium, IPDPS '02, IEEE Computer Society, Washington, DC, USA, 2002, pp. 312–.

[100] W.-C. Chuang, B. Sang, S. Yoo, R. Gu, M. Kulkarni, C. Killian, Eventwave: Programming model and runtime support for tightly-coupled elastic cloud applications, in: Proceedings of the 4th Annual Symposium on Cloud Computing, SOCC '13, ACM, New York, NY, USA, 2013, pp. 21:1–21:16.

[101] I. Giurgiu, O. Riva, G. Alonso, Dynamic software deployment from clouds to mobile devices, in: Proceedings of the 13th International Middleware Conference, Middleware '12, Springer-Verlag New York, Inc., New York, NY, USA, 2012, pp. 394–414.

[102] E. Abebe, C. Ryan, Adaptive application offloading using distributed abstract class graphs in mobile environments, J. Syst. Softw. 85 (12) (2012) 2755–2769.

[103] Y. Zhang, G. Huang, X. Liu, W. Zhang, H. Mei, S. Yang, Refactoring android java code for on-demand computation offloading, SIGPLAN Not. 47 (10) (2012) 233–248.

[104] B. Gao, L. He, L. Liu, K. Li, S. A. Jarvis, From mobiles to clouds: Developing energy-aware offloading strategies for workflows, in: Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing, GRID '12, IEEE Computer Society, Washington, DC, USA, 2012, pp. 139–146.

[105] S. Han, S. Zhang, Y. Zhang, Energy saving of mobile devices based on component migration and replication in pervasive computing, in: Proceedings of the Third International Conference on Ubiquitous Intelligence and Computing, UIC'06, Springer-Verlag, Berlin, Heidelberg, 2006, pp. 637–647.

[106] S. Han, S. Zhang, Y. Zhang, A generic software partitioning algorithm for pervasive computing, in: Proceedings of the First International Conference on Wireless Algorithms, Systems, and Applications, WASA'06, Springer-Verlag, Berlin, Heidelberg, 2006, pp. 57–68.

[107] H. Cai, W. Zhang, Y. Zhang, G. Huang, Sm@rt offloader: Supporting adaptive computation offloading for android applications, in: Proceedings Demo & Poster Track of ACM/IFIP/USENIX International Middleware Conference, MiddlewareDPT '13, ACM, New York, NY, USA, 2013, pp. 3:1–3:2.

[108] S. Trifunovic, A. Picu, T. Hossmann, K. A. Hummel, Adaptive role switching for fair and efficient battery usage in device-to-device communication, SIGMOBILE Mob. Comput. Commun. Rev. 18 (1) (2014) 25–36.

[109] M. Shiraz, E. Ahmed, A. Gani, Q. Han, Investigation on runtime partitioning of elastic mobile applications for mobile cloud computing, J. Supercomput. 67 (1) (2014) 84–103.

[110] K. Yang, S. Ou, H.-H. Chen, On effective offloading services for resource-constrained mobile devices running heavier mobile internet applications, Communications Magazine, IEEE 46 (1) (2008) 56–63.

[111] J. Matthews, M. Chang, Z. Feng, R. Srinivas, M. Gerla, Powersense: Power aware dengue diagnosis on mobile phones, in: Proceedings of the First ACM Workshop on Mobile Systems, Applications, and Services for Healthcare, mHealthSys '11, ACM, New York, NY, USA, 2011, pp. 6:1–6:6.

[112] S. Kundu, J. Mukherjee, A. K. Majumdar, B. Majumdar, S. Sekhar Ray, Algorithms and heuristics for efficient medical information display in pda, Comput. Biol. Med. 37 (9) (2007) 1272–1282.

[113] G. C. Sih, E. A. Lee, A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures, IEEE Trans. Parallel Distrib. Syst. 4 (2) (1993) 175–187.

[114] M. A. Khan, Scheduling for heterogeneous systems using constrained critical paths, Parallel Computing 38 (45)

(2012) 175 – 193.
URL http://www.sciencedirect.com/science/article/pii/S0167819112000105

[115] H. Topcuouglu, S. Hariri, M.-y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Trans. Parallel Distrib. Syst. 13 (3) (2002) 260–274.

[116] J. R. V. Winkler, Securing the Cloud: Cloud Computer Security Techniques and Tactics, Syngress Publishing, 2011.

[117] A. U. Khan, M. Othman, M. Ali, A. N. Khan, S. A. Madani, Pirax: Framework for application piracy control in mobile cloud environment, J. Supercomput. 68 (2) (2014) 753–776.

[118] I. Goldberg, D. Wagner, R. Thomas, E. A. Brewer, A secure environment for untrusted helper applications confining the wily hacker, in: Proceedings of the 6th Conference on USENIX Security Symposium, Focusing on Applications of Cryptography - Volume 6, SSYM'96, USENIX Association, Berkeley, CA, USA, 1996, pp. 1–1.

[119] U. Khalid, A. Ghafoor, M. Irum, M. A. Shibli, Cloud based secure and privacy enhanced authentication & authorization protocol, Procedia Computer Science 22 (0) (2013) 680 – 688, 17th International Conference in Knowledge Based and Intelligent Information and Engineering Systems - {KES2013}.
URL http://www.sciencedirect.com/science/article/pii/S1877050913009423

[120] N. Antonopoulos, L. Gillam, Cloud Computing: Principles, Systems and Applications, 1st Edition, Springer Publishing Company, Incorporated, 2010.

[121] L.-C. Lin, T.-J. Lin, C.-C. Lee, C.-M. Chao, S.-K. Chen, C.-H. Liu, P.-C. Hsiao, C.-W. Liu, C.-W. Jen, A novel programmable digital signal processor for multimedia applications, in: Circuits and Systems, 2004. Proceedings. The 2004 IEEE Asia-Pacific Conference on, Vol. 1, 2004, pp. 121–124 vol.1. doi:10.1109/APCCAS.2004.1412707.

[122] P. Holzenspies, G. J. M. Smit, J. Kuper, Mapping streaming applications on a reconfigurable MPSoC platform at run-time, in: System-on-Chip, 2007 International Symposium on, 2007, pp. 1–4.