Rameshwar Singh Sikarwar[1],

Dr. Rajeev G. Vishwakarma

A Customizable Mechanism for Computing Offloading in Mobile Cloud Computing

Journal of Electrical Systems

Abstract: - Due to the widespread use of mobile devices, mobile apps are getting more complex in terms of the features they provide to users. This is a direct result of the demand for these applications. The execution of these applications consumes a significant amount of the device's resources, despite the fact that users may use them often. Using mobile cloud computing (MCC), which moves part of the application's burden to the cloud, it is possible to find a solution to this problem. In the beginning, compute offloading could seem to be an excellent method for conserving device resources; nevertheless, if it is carried out in a static manner, the result might end up being the reverse. It is possible that the efficacy of offloading tactics may be diminished by making regular alterations to the computing environment and the resources at the end of the device. Additionally, the quality of service for applications that depend on real-time data could be diminished. This problem is addressed by the authors via the provision of an adaptive compute offloading framework for programs that deal with data streams. This framework separates applications in a dynamic manner and then offloads them according to the cloud resources, network conditions, and device end characteristics. A technique that is provided and that explains the process of the offloading model is also given in the article. Last but not least, the authors provide the results of the simulation as well as an illustration of how the proposed system functions.

Keywords: Customize Code Offloading, Data Stream Applications, Mobile Cloud Computing, Mobile Computation Offloading

1. INTRODUCTION

The objective of the notion of mobile cloud computing, which has recently gained popularity, is to provide mobile clients with improved services without placing an excessive burden on the capacity of their devices or adding unneeded charges for data and energy. Mobile cloud computing has achieved a number of significant achievements, one of which being the resolution of the problem of resource limitations on mobile devices. The concept of Mobile Computational Offloading (MCO), which included exploiting cloud computing characteristics to boost the computational capability of devices with constrained resources, was the driving force behind this development. In the present time, the vast majority of applications are offline, which indicates that data is downloaded from backend systems and resources are located locally. It is possible for web technologies to serve as effective alternatives for internal applications, and online applications provide users with the opportunity to see data whenever it is convenient for them. In every one of these situations, it is essential to have a flexible system that is able to respond to the many changes that occur in the mobile environment in order to get the best potential outcomes. Mobile devices are able to offload to any of the computational infrastructures, such as virtual machines or cloudlets, in order to acquire the desired result (Khanna, 2016). This is done in accordance with the unique needs of an application. Under these circumstances, the most important thing is to maximize the performance of smartphone applications while simultaneously minimizing energy consumption (Baker, 2015). On the other hand, mobile cloud computing functions in a heterogeneous environment, which means that customers utilize a variety of mobile devices and have specific requirements. This necessitates the use of remote execution and dynamic partitioning, both of which may be accomplished by distributing program modules between a mobile device and the server via the utilization of the Alfredo framework (Rellermeyer, 2008). This is analogous to the way that some systems make advantage of R-OSGi (Rellermeyer, 2007) in order to make communication across virtual machines easier. If this is not the case, an application may be broken down into elastic components that can be executed dynamically via the use of weblets (Zhang, 2011). Because weblets are not restricted to a particular programming language, they may be used for a wider range of activities than other types of applications. Migration of processes is yet another essential component that must be there in order to facilitate the seamless transformation of individual activities

[1,2]Department of Computer Science and Engineering
[1]Research Scholar, Dr. A. P. J. Abdul Kalam University, Indore
[2]Research Supervisor and Pro Vice-Chancellor, Dr. A. P. J. Abdul Kalam University, Indore, (M.P.), India
Corresponding Author : Rameshwar Singh Sikarwar
Email : rameshwarsingh@aku.ac.in

without sacrificing performance.

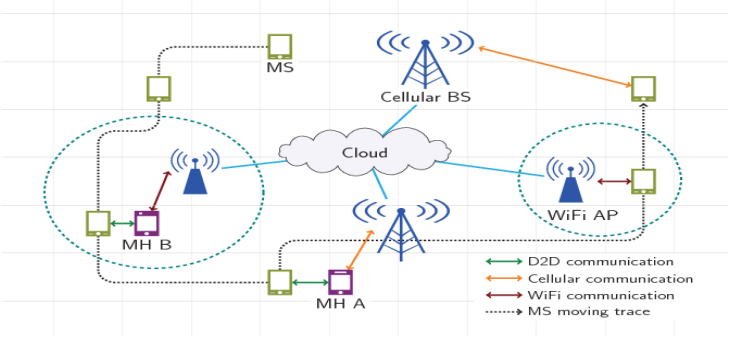

Figure 1 Data offloading in mobile cloud computing

Within the scope of this study, we concentrate on mobile data stream applications and investigate strategies to improve their execution as well as make use of cloud resources in order to execute such applications on mobile devices that have resource constraints. An adaptive method for offloading computations is something that we provide for applications that deal with data streams. The migration granularity that we give is at the thread level, and the approach that we provide is based on the concepts of dynamic code splitting. In the course of our research, we have endeavored to simultaneously reduce the makespan and the amount of energy that is used in order to maximize the efficiency of data stream applications (Baker, 2017).

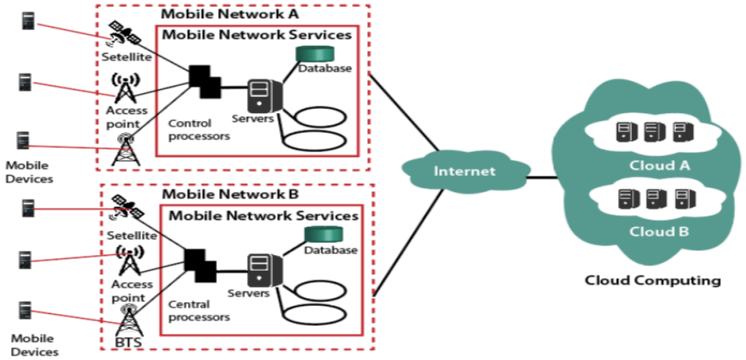

Figure 2 Mobile cloud computing: architecture, applications, and approaches (Dinh 2013)

2. LITERATURE REVIEW

A multitude of sensors, global positioning system (GPS), internet access, and connectivity to neighboring devices are all characteristics that were not even available in personal computers in the 1990s. Smartphones. Over time, applications have become more complicated, have increased their power usage, and have taken up more space. Creating a smartphone that is both fashionable and has a long battery life is the primary challenge that producers of smartphones must contend with. On the other hand, developers of applications are required to create programs that can calculate more data in a shorter amount of time and with less power. However, it is a substantially more difficult challenge to solve in a mobile device of a smaller size. Mobile cloud computing is an approach that aims to avoid the performance limitations of mobile devices by providing access to clouds that are very rich in resources. (2015) According to Gupta. Network providers and the cloud both stand to profit from the use of MCC, which makes virtualized cloud services accessible to mobile applications via the use of a network. On cloud servers, it is possible to operate mobile apps that need a lot of resources with simplicity. A network, a cloud, and a mobile device are the components that make up the architectural aspects of MCC. There is a network that serves as the communication route between a mobile user and the cloud cloud. Both low bandwidth and network latency have the potential to hamper the purposes of the MCC. When their network has favorable characteristics, a rising number of users are able to reap the benefits of MCC. Increasing the amount of resources available for mobile application execution on cloud platforms is made possible via the exploitation of virtual machines (VMs) and the concept of virtualization. These resources include memory, storage, and the core of the CPU.The act of shifting application code away from the mobile device and running it on the cloud is referred to as mobile computation

offloading, or MCO for short.

In order to accomplish a variety of objectives, mobile devices make use of compute offloading. These objectives include the decrease of turnaround time, the conservation of CPU and memory, and the saving of energy. However, due to the fact that mobile applications need to be partitioned and a distributed application processing platform has to be implemented, it is possible that during the offloading process, more local resources would be required to be used than are saved by the mobile resource conservation. In order to ensure that the deployment of a distributed platform requires the least amount of local resource consumption possible, it is essential that the procedures for runtime partitioning of mobile cloud applications be strategically prepared. In addition to providing information on offloading systems, MCO also conducts an analysis of the effect of offloading. In order to accomplish this goal, the initial stages include partitioning the application code and deciding whether to run it locally on a mobile device or remotely on the cloud. The programmer may divide an application in a static manner, or the code profiler can segment it in a dynamic manner while the application is running. The MCO design makes use of a decision engine to determine which components are to be offloaded by analyzing the resources that are available at the device end. This is done in order to evaluate the possibility of offloading in terms of the availability of the network and the amount of battery life. When it is determined that offloading would result in a reduction in energy use, the process is carried out. As a consequence of this, dynamic application partitioning is advantageous since it enables offloading practices that are more efficient and thoughtful. Several other MCO designs have been suggested in this area, some of which include CloneCloud (Chun, 2011), MAUI (Cuervo, 2010), Cuckoo (Kemp, 2010), and Odessa (Ra, 2011). Image offloading, method offloading, and feature offloading are some of the offloading strategies that are used by these designs. Each of these offloading approaches has its own set of benefits and cons. Adaptive computation offloading, on the other hand, is the process of making judgments based on a real-time assessment of all of the characteristics of the device's end, including the features of the network, the capacity of the battery, and the capabilities of the computer.

In order to function properly, applications designed for mobile devices, particularly data stream applications, need constant contact with remote components, such as cameras, GPS units, and high-speed sensors. Response time, energy conservation, makespan, throughput, and makespan are some examples of performance measures that may be used for data stream applications respectively. Applications like augmented reality (AR) that make use of interactive data streams are computationally costly. This is due to the fact that these applications often interact with sensors and generate a stream of data that has to be processed in a short amount of time. These applications are sensitive to network latency; in order to perform at their best, they need high bandwidth and low latency. These characteristics make it possible for them to easily offload tasks that require a lot of computing power and extend the amount of time that their batteries can last. Wei et al. used a large number of cloudlets in order to solve the problem of network latency that existed between mobile devices and cloudlets, or more precisely, between a mobile device and a cloud server. As a result of the fact that its modules may be parallelized, interactive mobile applications can be used to generate fine-grained segmentation. This will result in an increase in the number of communication channels and, subsequently, the amount of bandwidth that is consumed. Virus scanners, chess games, and photo searches are examples of applications such as data stream applications that are not sensitive to delays caused by the network. Applications for virus scanners need a significant amount of connection since they are required to upload a substantial quantity of data to the cloud. This also makes them a data-intensive application. Several offloading solutions for data stream applications have been presented in research publications pertaining to the topic.

As a consequence of this, these applications are partitioned in order to provide distributed processing at a variety of granularities. Workloads that are partitioned may also be classified as CPU-intensive or I/O-intensive, including further categories. In MCC, activities that require a significant amount of CPU power may be outsourced to other people and independently carried out. Because every data stream application requires a significant amount of computational power, we have to offload the code to the cloud in order to support them on a mobile device that has a restricted amount of resources. Data stream applications, on the other hand, are very sensitive to communication; hence, poor network performance may further increase the cost of communication and cause an increase in energy consumption. Because of this, the use of offloading techniques is required in order to enhance the performance of programs that deal with data streams. Nevertheless, in addition, it is necessary to make appropriate judgments on offloading in real time by regularly analyzing the cost of communication as a function

of bandwidth.

3. RELATED WORK

The use of code segmentation and scheduling was proposed by Zhang (2016) as a method for distributing cloudlet resources among the offloading requirements of several clients. In contrast to other approaches, which presume that there are an infinite number of resources available at the cloud end, this method takes into consideration both the availability of resources and the dynamic allocation of cloud resources. There is a possibility that the execution cost of a method on cloudlet may increase as a result of the competition between several users for the available resources; however, this cost will not alter when the method is done on a mobile device. The code partitioning method is responsible for determining which code should be offloaded in order to speed up execution. This determination is made depending on the workload and the status of the network at the moment. In the case that the required workload is more than the capacity of the cloudlet, the application is not permitted to offload any of its task. They established a mixed integer programming model in order to ensure that the total execution and transmission time for tasks that are sensitive to delays does not exceed the completion time. This was done in order to guarantee that the work would be completed on time. For the purpose of indicating an execution order via the use of depth-first search, the code partition algorithm makes use of call trees. In order to manage cloudlet offload requests, the resource scheduling algorithm will allocate a virtual machine that is suited for the task..

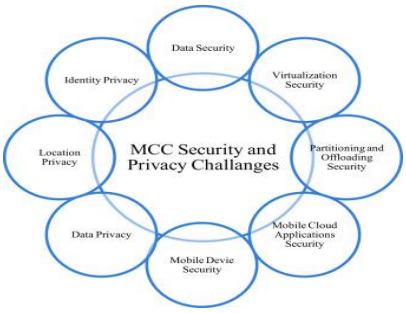

Figure 3. Security and privacy challenges in mobile cloud computing

Through the use of the One Time Offload Property and the Optimal Substructure Property, it is possible to ascertain the optimal places for offloading and integration, as well as to ensure that all methods share the same integration point. The completion ratio for major works is increased by the combined code split and resource allocation technique, despite the fact that the completion ratio for smaller projects is very low. (Niu, 2013) described the multisite offloading strategy in their research article. They used the Markov decision process and mathematical formulas to calculate the process's time and energy consumption. This was done in order to indicate the savings that were achieved throughout the course of the operation in comparison to a single site.

This may assist guarantee that the mobile user obtains better service without taking up all of his device's resources. This can be accomplished by picking the suitable algorithm and making the appropriate choice for each action that involves mobile cloud computing. In the context of this study, the environment is a mobile device equipped with a user interface module. In order to properly execute an application, it is necessary to offload data-intensive (DI) or computation-intensive (CI) tasks to many heterogeneous servers. Additionally, the network bandwidth between these servers is arbitrarily variable. For the purpose of this investigation, a mathematical model is used to compute the amount of energy that a program consumes using both static and dynamic profiling. For the purpose of determining how to make judgments in least-cost short route and delay-constrained scenarios, a paradigm that is known as the Markov Decision Process (MDP) is presented. An EMOP (Energy Efficient Multisite Offloading policy) algorithm is described as the last step in the process of selecting the most effective offloading techniques. A large number of simulations were carried out in order to evaluate the efficiency of the recently created EMOP algorithm. The results of these simulations were compared to the results of an offloading method that was carried out at a single site. The conditions that were used in the simulation included three offloading sites inside a heterogeneous system, as well as one server located close to the mobile site. In second place is a database server, while third place goes to a cloud server and a web server.

It was expected that there was a two-state dynamic channel between the offloading sites and mobile devices, and that there was ten times greater bandwidth between the servers and the mobile devices. For the purpose of CI simulation, a software

that solves Sudoku or N-queens puzzles was used, but for DI, a virus scan application was utilized. In the study, it was discovered that lower node apps utilized less energy when servers were located in close proximity to the mobile site. On the other hand, when they used greater nodes, the cloud servers consumed less energy than the local servers themselves. When compared to single-site operation, multisite offloading achieved via the use of an EMOP algorithm may result in energy savings of up to 30.8%. When it comes to this particular occurrence, there is also less of a wait in time. In spite of this, the algorithm's energy consumption rises since it uses the strategy with the shortest possible delay in time. Depending on the kind of application (CI or DI), the EMOP algorithm chose the servers that would be most efficient in terms of the amount of energy that they used. Additionally, the simulations demonstrated that the one-time unloading feature is supported by at least one migration that takes place between sites as well as between mobile and the site for the mobile device. A multisite offloading method that takes into consideration the varied configurations and capabilities of server machines is proposed in this paper as a means of reducing energy consumption and maximizing energy savings. In order to demonstrate the improved cost-effectiveness of multisite offloading in comparison to single site execution, the application was used to monitor data offloading.

2011. Chun (Chun) provides a solution for using cloud execution to enhance the experience of using a smartphone without requiring the hardware to be changed and which also has a larger battery reserve. The intention is to facilitate the execution of the user's preferred program on the mobile device, while a remote desktop phone clone will be responsible for managing complex tasks. The following are some of the ways in which the user benefits from this: 1) The complicated and time-consuming processes are calculated without causing the phone to become sluggish or draining its battery. 2) The incorporation of cloud computing has made it possible for a smartphone with a limited amount of hardware to perform the same functions as desktop computers to which it is connected. 3) Even in the case that the user's smartphone is misplaced, he will not lose any of his data since a copy of the device is already accessible. For the purpose of accomplishing all of this enhanced performance, the following processes are used.

1) **Functionality outsourcing for primary functions**: the user interface and minor operations remain on the smartphone, while distant desktops handle only the intricate and time-consuming calculation activities. This is accomplished by automatically dividing the procedure, and the service is delivered by a cloud computing server.

2) **Background Augmentation**: This method is applied to face analysis in photos, file indexing, and virus scanning. Here, the entire procedure is run on a distant computer, and when the outcomes are vailable, it is repeated. The procedure is transmitted to the phone once it is turned on, and it continues to work even when the smartphone is turned off.

3) **Mainline Augmentation**: This procedure falls between background augmentation and major function. mostly utilized for debugging, fault tolerance, and the detection of private data leaks.

4) **Hardware Augmentation**: This technique digitally connects a smartphone to a powerful desktop computer to make up for its hardware limitations. The results of the tests indicate that the debugging application ran 11.8 times faster on the desktop than it did on the smartphone for the sole purpose of execution.

5) **Amplification by the use of multiplicity:** This type of augmentation involves analyzing an application more than once by running several photos through a system and investigating every conceivable result to provide the application options. By copying the phone and moving the calculation to distant servers, heterogeneous computing systems are used to achieve the augmentation.

Creating the clone on the desktop, often syncing it with compressed data packages, executing programs in the clone either automatically or upon request, and then reproducing the results back into the phone are all steps involved in the process. Specifically, this process is comprised of three stages: While the Controller is responsible for performing calculations, the Replicator is in charge of synchronization. The cloned Augmenter is in charge of the local execution. When a procedure on a smartphone is initiated, the application enters a state of sleep, and the clone does the computation, therefore copying the results on the phone. As soon as that occurs, the program reawakens and begins to carry out the user interface once more. The two primary challenges that the system faced were figuring out how and when to do the compute transformation, as well as synchronizing the clone with the smartphone, all while using the least amount of memory and maximizing the amount of battery life. To ensure that the primary and clone processes are carried out in a coordinated manner, it is required to create separate methods. Having trouble trusting others is another big obstacle that must be overcome. When it comes to the functionality of this system, it is necessary to have a sizable cloud computing network as well as a substantial number of virtual machines (VMs). The establishment of the required infrastructure

would be expensive; hence, the only alternative option that is feasible will be to transfer sensitive personal information to kiosks that are privately managed. Nevertheless, it is feasible provided that stringent safety precautions are adhered to.

In 2011, Wang presented a comprehensive review of the virtual machine substrate and substrate pool, as well as an updated concept of the virtual machine lifetime. According to their definition, a substrate is an instance that is static, reusable, and may be triggered whenever it is required. VM substrates are created by reducing the size of virtual machines (VMs) while using the fewest resources possible, such as memory and CPU. Following the completion of just the most essential execution stages, these virtual machine substrates are placed in memory and secured there. The capacity of a virtual machine with a capacity of 2 gigabytes may be decreased to tens of megabytes. Additionally, these substrates for virtual machines are transferred to the substrate pool. It is possible to restore the capacity of virtual machine substrates and activate them when stateful virtual machines are restarted without first having to restart the computer. On a computer that has four gigabytes of random access memory (RAM), there might be as many as one hundred substrates.

Additionally, the size of the substrate is influenced by the status of an application. Administrators have the ability to build substrates for virtual machines via live checkpointing or intrusive shrinking carried out at the application level. The virtual machine (VM) to virtual machine (VM) substrate arrangement requires the fewest possible computer processors, memory footprints, unconnected network cards, and disk states. Live checkpointing, in which the size of the checkpoint is equal to the amount of the virtual machine's memory, is performed at the system level for the purposes of virtual machine migration, fault tolerance, and debugging. It also assists in saving the running states of the CPU, memory, and disk in the core. Both of these approaches are used in order to minimize the amount of memory that is included inside the VM substrate. Rapid deployment of several virtual machines (VMs) is possible via the deployment of the VM substrate at the same time.

This architecture, which was described by Chun (2011), is a flexible one that makes it easier to execute mobile apps in a distributed manner. Certain threads of an executable are transferred from a mobile device to a cloud-based device clone using the CloneCloud paradigm. For instance, the threads that are chosen at method entry and exit points are transferred from the mobile device to the cloud-based device clone. In contrast to MAUI, CloneCloud does away with the need for source annotations, which frees the programmer from the burden of having to deal with annotations (Cuervo, 2010). Applications are regularly partitioned using a wide variety of objective functions and situations, which results in a large number of partitions. A dynamic profiler and a static program analyzer are used in the process of partitioning in order to initiate the creation of a partition. Static analysis is performed on the binary representation of the application, which identifies migration and re-integration locations, which are often called method entry and exit points. Building profiler trees is accomplished by dynamic profilers via the process of profiling binary files with a variety of inputs and executing the partitions on a variety of platforms, such as clones and cellphones. It is the responsibility of the Optimization Solver to determine which smartphone section may be transferred to a clone in order to achieve the lowest feasible execution cost.

After assigning a value to the decision variable R(m) = 1, the partitioner then proceeds to insert migration and reintegration at the entry and exit procedures. The calls to the methods that produce the result R(m) = 1 are all relocated. The migration technique made available by CloneCloud allows for thread migration, however it does not provide native or non-virtualized feature transfer. A native per-process migrator, a database of partitions, and a per-node manager are all components that are used by CloneCloud Migrator. The process of thread migration involves putting an end to the execution of the thread that has to be migrated, collecting the relevant data objects, and then transferring the state to the node manager. The captured state of a thread is uploaded to the device that is meant to be the clone. Next, the node management will send it on to the migrater for further processing. In order to facilitate the process of state merging, object mapping tables are used. This is due to the fact that ultimately, the thread will need to be reintegrated with its executable code on a mobile device.

According to Zhao (2010), a mirroring approach was used, in which a mirror was kept in the computer infrastructure of the telecom network for each smartphone. Considering that the smartphone and its mirror are synchronized, it is possible to utilize some applications from the smartphone, such as virus scanners, on the mirror instantly. Additionally, the synchronization module syn-client, which is a part of the smartphone operating system, is responsible for collecting data from the user and synchronizing it with a mirror server. It is possible to

update the mirrors by using the Traffic Monitor module and the Synchronization module that are located on the Mirror Server. Mirror servers only need one template for each model, which is one of the numerous advantages of the mirroring process. This is due to the fact that identical smartphone models have the same hardware requirements, as well as the same operating system and factory default settings. Similar to virtual machines (VMs) that are constructed using the factory default template, mirrors on the mirror server are also known as mirrors. For those who are using a smartphone for the very first time, it is possible to finish all updates in a short amount of time at the mirror by offering valuable guidance. In the course of the synchronization process, the smartphone is kept in a frozen state, which means that the user is not permitted to take any actions. This occurs while the software specs and user data are being transferred from the smartphone to the mirror. In accordance with the framework that has been proposed, synchronization is feasible provided we provide a smartphone and its mirror with similar inputs in the same sequence. In the context of a 3G network, the framework is designed to be used, and it is anticipated that smartphones would be connected only over the 3G network, rather than through Bluetooth or Wi-Fi. As a consequence of the findings, virus scanning applications will have a triple advantage in terms of energy consumption, task sharing, and faster execution. It is possible to use this framework for applications that cache data as well as for uploading and downloading large amounts of data via bulk.

When doing research on peer-to-peer networks, Begum (2010) used dynamic process migration as another method. To reduce the amount of time required for the process migration in the dynamic network, they developed a model. The load balancing across peer nodes and the movement of processes in a homogeneous manner were their primary concerns. In order to do this, DHT is employed to link nodes, which are resources, and to provide assistance to a mobile node in identifying a destination node, which allows tasks to be transmitted to the destination node. The Base Station (BS), which is responsible for managing the distribution of applications to peer mobile nodes, is designated as having super-peer status. The capacity and processing load of each node must be taken into consideration in order to determine whether or not the processes are given an allocation. The super-peer facilitates coordination and service for all mobile devices that are included inside the cell.

Every single cluster node is equipped with a hash table, and the DHT first generates a max-heap consisting of cluster nodes before using a selection method to ascertain which node has the most amount of available capacity. The required amount of time for nodes to depart and join the cluster is O(log n), while the amount of time required for the destination node to search is O(1). This allows for the smallest amount of disruption possible. When a mobile node leaves the cluster, it alerts the base station (BS) to transmit the process state to the destination node via the BS by means of the BS. By using this heap-based DHT technique, the timeframes required for the selection of the destination and the transfer of the process state are reduced.

An experiment was carried out by Goyal (2004) to determine the utility of cloud computing for personal digital assistants (PDAs) in terms of the amount of energy used and the amount of time required to finish a task. A speech recognition program that requires a lot of computing power and a data mining application that requires a lot of bandwidth were brought together for the purpose of performance comparison. The Sharp Zaurus PDA was used as the client for both of the experiments, while the RedHat Linux PC was used as the server or host for both of them. In its whole, the system is referred to as foraging, and the clients are personal digital assistants (PDAs), mobile phones, or tabs that make use of remote desktops as stand-ins. The system is comprised of a personal digital assistant (client), any local area network (LAN), wireless network (WLAN), or internet network, a desktop computer (which is referred to as a surrogate in this context), and connecting devices. A client and a surrogate are both components of a system that allows the client to locate the surrogate by using a service location protocol or a notation that is similar to XML on a service discovery server. An Internet Protocol (IP) address and a port number are the responses that the server gives in response to a request for service discovery. After establishing a connection with the client, the server will verify the client's identification and then proceed to initialize a root disk and root image that have been pre-allocated. This will initiate the launch of a new virtual server for the client. The client for the appropriate virtual server then gets the IP address from the surrogate manager when this step has been completed.

On this server, a virtual service manager is responsible for managing all client requests. For example, the execution of software may be accomplished by just providing the URL of the program to the server. When the service is terminated, cleanup of the virtual server may be accomplished by reinstalling the root file. Modifications made by clients are undone in order to protect the system and the privacy of users who come after others. In the event

that a client uses this system more than once or on a home network, it is possible to build up a customized partition that meets the necessary parameters. During the course of the same operation, the Sphinx speech recognition experimental setup was used to study the changes in power consumption and processing time that occurred in two unique circumstances. When the process was carried out on a personal digital assistant (PDA), it took sixty times more energy to finish and used more than ninety-five percent of the central processing unit (CPU), with occasional spikes reaching 98.8 percent.

Using a surrogate system reduces the amount of CPU time used to between 0.3 and 5%, and it also reduces the amount of memory overhead to 1.1%. Experiments of a similar kind were carried out for data mining operations that needed a significant amount of bandwidth; however, testing a personal digital assistant took just a fourth of the time and a quarter of the battery life. Additionally, when carrying out these actions, the personal digital assistant (PDA) device either becomes unable to run any other programs or hangs up permanently. At the same time, a cost-effective solution is being researched in order to address the security and scrutiny problem. This solution will enable consumers to entrust surrogates with their data while the surrogates protect their hardware from harmful attacks (Arora, 2017).

4. PROPOSED WORK

A significant number of researchers and developers have developed an interest in offloading mobile applications to the cloud over the course of the last few years. This interest has enabled them to propose architectures and strategies that simplify the process of code offloading (Alam, 2016). Implementing the concept of mobile compute offloading may be done in two different ways: either via static code segmentation or through dynamic code segmentation. Static partitioning requires an application developer to actively divide a program in order to run it remotely, in contrast to dynamic partitioning, which enables an application to partition itself dynamically without the need for human involvement depending on the variables of the execution environment.

As far as mobile computation offloading is concerned, the components of an application may be classified into three distinct groups: those that are required to be executed locally, those that are required to be executed in the cloud, and those that are capable of being executed both locally and remotely (Khanna, 2016). The implication of this is that the same application component could be able to operate locally or might need to be offloaded, depending on the resources that are available at the device end. Taking into consideration the efficiency of the offloading model is an essential component to take into account when addressing the granularity of compute offloading migration (Tao, 2015). The granularity of the information will decrease, which will result in an increase in the accuracy of detecting the application's execution pattern. Within this part, we provide an adaptive compute offloading strategy that may be used for applications that include data streams. The migration granularity that we give is at the thread level, and the approach that we provide is based on the concepts of dynamic code splitting. There are other categories for the application threads, which include tasks that need a lot of CPU and I/O resources. In the next section, an application thread shall be referred to as a "task" throughout the whole section. Because they need spontaneous responses from the user and use a limited number of resources, activities that are labor-intensive in terms of input and output are not taken into consideration by our method for offloading. The system will offload operations that need a significant amount of CPU time depending on the size of the job, the number of instructions, the execution pattern, the length of the execution, and the device end factors. Our model's key objective will be to ensure that both energy saving and performance enhancement in terms of makespan are achieved over its lifetime.

5. APPLICATION MODEL

The application model of the technology that we have provided would be the topic of discussion in this paragraph. In order to provide an explanation of how our offloading system operates, the model is composed of eight separate components that have a relationship with one another. In Figure 1, the application model is shown in its working state.The Device Profiler is the component that is accountable for dividing a program into a number of different activities. Whether a task requires a significant amount of CPU or I/O, it is differentiated from other tasks. In addition to this, the device profiler will demonstrate the manner in which a certain group of tasks are carried out. It is possible for a job to be independent of other activities or dependent on them; yet, the execution of a particular set of dependent tasks may be distinguished from the execution of another set of dependent tasks. When it comes to unloading many operations at the same time, the categorization of this type that the device profiler provides proves to be pretty beneficial. In order to assist the system in determining whether a work is CPU-intensive or I/O-intensive, the device profiler provides specific annotations that are unique to the job.

Analyzer of Applications: When it comes to the amount of battery life and amount of time it takes to complete a task, this tool's role is to determine how much it will cost to carry out a work either locally or remotely. The application execution cost is comprised of two fundamental components: the costs of computation and the costs of communication. In the context of mobile devices, the term "communication cost" refers to the expense incurred when task data is sent from the device to the cloud.Because of this, the application analyzer and the compression unit are connected. This is because the compression unit is responsible for providing the exact value of the data that has to be sent. In order to determine the execution cost, the application analyzer takes into account a number of criteria, such as the number of instructions that need to be carried out, the bandwidth of the network connection, the size of the job at hand, and the processing capability of both the mobile device and the cloud computing environments.

The Unit of Compression: This process compresses the task data, which includes both the code data and the computation data, before offloading. It is guaranteed by the compression unit that the compression ratio will fall somewhere in the region of 10 to 15. Additional information on the methods of data compression may be found in the section under "Mathematical Model";

Engine for Making Decisions: The application analyzer and the device profiler are the sources of inputs that it receives. To assess whether or not a work has to be offloaded, it performs certain mathematical calculations depending on the inputs and then makes a determination. Moreover, it tells the task scheduler of the decision that it has made;

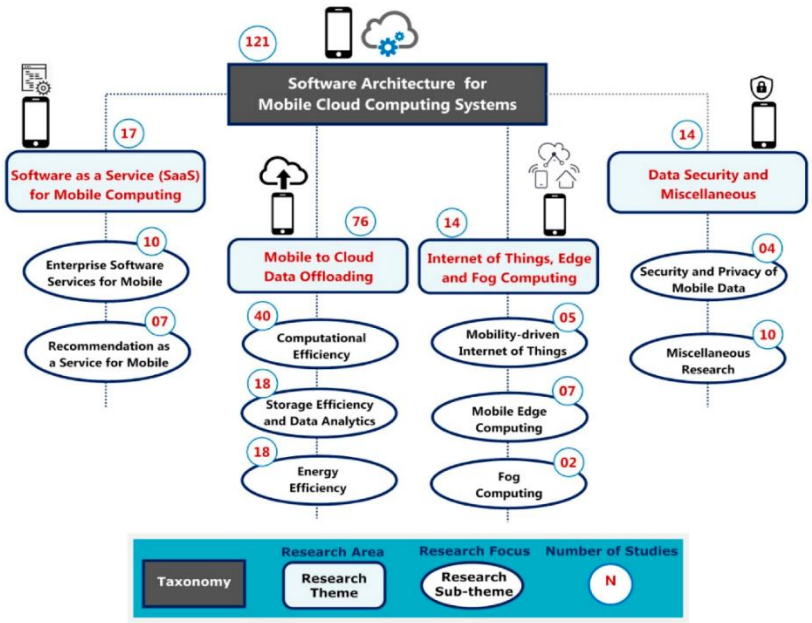

Figure 4. Software Architecture for Mobile Cloud Computing Systems

Task Scheduler is a software that is used to plan out the execution of a certain task or a set of actions that are connected to one another. Following the receipt of information from the decision engine, it places the tasks in the proper queue. Furthermore, it engages in information exchange with the application analyzer in order to acquire knowledge on the execution of a task. When the device end parameters are taken into consideration, the task scheduler has the ability to alter the execution pattern for many sets of tasks. It is possible for it to combine a number of tasks that, at first glance, would not seem to be connected due to the execution cost and offloading factor of each of them. One of the key objectives of the task scheduler is to ensure that tasks are carried out efficiently while using the least amount of resources possible. It is the responsibility of the task scheduler to provide one-of-a-kind annotations in order to enable the system to identify the execution of a task based on its execution location.

Administrator of Synchronization: It is the responsibility of the synchronization manager to identify integration points for action sequences that are carried out in different locations. It does this by collecting the output data from a number of activities that are carried out in a variety of locations and then using that data as an input for another operation. It engages in conversation with the task scheduler in order to ascertain the location and pattern of execution of a procedure.

Additionally, it connects with the application analyzer in order to ascertain the dependencies that a task has with other tasks. The responsibility of ensuring that no work is stalled and that all jobs finish at the same time falls on the shoulders of the synchronization manager.

The term "Resource Manager" refers to an organization that operates inside the cloud. According to Liu, Ahmed, Shiraz, Gani, Buyya, and Qureshi (2015), it is responsible for allocating and supervising each and every resource that is made available to the user in order for them to be able to execute their application. Through interaction with the device Monitoring Agent and dynamically raising or lowering the availability of cloud resources, resource management is what maintains the execution cost stability, even in circumstances where there are noticeable variations (decreases) in the mobile end resources. This is accomplished by engaging with the device Monitoring Agent. Furthermore, it engages in information exchange with the Cloud Monitoring Agent in order to ascertain the threshold value for the cloud resources that may be allocated to a particular mobile user;

The Agent of Monitoring: It is the responsibility of the monitoring agent to keep a close check on the resources that are accessible in both forms of computer environments. Our proposed architecture is comprised of two separate monitoring agents, one of which is located at the side of the mobile device, and the other is located at the cloud end. Constant communication takes place between the two agents responsible for monitoring. The monitoring agent of the mobile device transmits information to its counterpart at the cloud end on the parameters of the device end. This source provides the resource management with extra information, which is then used to modify resource allocation in order to maintain a constant total execution cost**.**

6. PROPOSED APPROACH FOR TASK SCHEDULING MODEL

The approach that has been offered (GMOPSO-BFO) is based on a hybrid strategy that combines the strategies of particle swarm optimization (PSO) and bacterium foraging optimization (BFO). Despite the fact that the BFO performs most effectively using local search resources, the PSO technique performs very well while looking for the answer worldwide. When these two strategies are combined, the result is a more globally and locally optimum solution with a faster convergence time. The size of the job and the characteristics of the virtual machine affect how long a given task takes to complete in this study. The following are some fundamental definitions related to task scheduling on mobile devices: Assume the following:

a) A job of the application tasks T = {T1, T2…, Tx};

b) A collection of n virtual machines V = {V1, V2, V3…, Vn}

c) Ti and Tj are any two jobs, and E is the collection of links between them.

d) The data center's collection of physical machines (PMs) = (PM1, PM2, PM3..., PMn)

The research operates on the presumption that the cloud service provider has an adequate quantity of computing power. The processing units (CPU), random access memory (RAM), and networking capabilities of different virtual machines are all entirely different from one another. V virtual machines are installed on the actual computers. Once the job is near, the data center brokers allocate the machine to it after keeping an eye on all the resources that are accessible. Every work that needs processing resources must wait in line, and tasks are scheduled to run on the system according to a task scheduling scheme.

7. PERFORMANCE MATRICES

In this part, we will discuss the primary components that were used as performance matrices for the model that we have proposed:

High Performance: It is important to note that throughput and makespan are the most important performance variables for applications that deal with data streams. On the other hand, throughput is a measurement of the number of tasks that are completed in a given length of time, while makespan is the amount of time required to complete a single task. As a result of its ability to reduce response time and enable the processing of more frames in a single second, minimal makespan in MCC is an essential component for high-performance applications such as face recognition systems that are dependent on gesture communication.

Energy Optimization: Mobile cloud computing provides a technique of lowering the amount of energy that is used by processing by applying the compute offloading approach. Considering that augmented reality (AR) applications need a

significant amount of computer power, they rapidly degrade batteries. There are situations in which the aim of energy saving cannot always be accomplished by simply moving compute-intensive tasks to the cloud, especially in situations where there is a restricted amount of network bandwidth (Calheiros, 2011). Therefore, it is predicted that MCC will include energy optimization measures in order to facilitate the creation of a balance between the costs of transmission and processing energy in order to increase the life of the battery;

Improved Granularity: MCC applications need to be partitioned in order to achieve improved granularity. This is done so that only the code that uses a significant amount of resources may be offloaded for remote execution. Because of effective segmentation, it is feasible to send the least amount of data possible while yet achieving the highest potential level of performance. Migration granularity is an important aspect that plays a significant role in influencing the efficiency of an offloading strategy. The granularity of the information will decrease, which will result in an increase in the accuracy of recognizing the application's execution pattern;

Dynamic Execution: In an MCC scenario, the network state and device-end properties are constantly shifting and dynamic. This characteristic is referred to as dynamic execution. Performing dynamic profiling of the features of end-user devices and networks is something that has to be done in order to produce the greatest offloading choice possible. When selecting an offloading mechanism, it is important to take into consideration the trade-off between the expenses of processing and communication. In order to accomplish cost-effectiveness, we have used offloading and splitting, as well as designed dynamic execution for all of the operations that are included in our recommended model.

Increased Parallelism and Elasticity respectively: In the context of application development, elasticity is defined as the capacity to dynamically alter the execution of application code at either the local or distant end in response to shifting environmental conditions. For example, in our mode, a project is only offloaded if it needs a significant amount of processing, appropriate network bandwidth, and most importantly, the availability of resources at the cloud's end (Khanna, 2015). On the other hand, if there is a low demand for the central processing unit (CPU) at the device end, as well as a low bandwidth and cloud resource availability, the task will be carried out locally on the mobile device. According to Yang (2013), data stream applications that are operated in a distributed way create parallelism, which ultimately results in an improvement in performance levels.

As a result of the fact that the service provider leases cloud resources in mobile cloud computing, the cost of the resources is an essential consideration that must be taken into account while developing the system. The quantity of cloud resources that the application provider leases to the host application is included in the operational expenses. Through the development of a dynamic offloading model, it is feasible to successfully achieve low operating costs while simultaneously making effective use of cloud resources and accomplishing energy conservation objectives.

8. CONCLUSION

The purpose of this research is to offer an adaptive offloading strategy for applications that deal with mobile data streams. Using our technology, it is feasible to do unloading in real time. An method that separates code at the thread level and subsequently conducts dynamic offloading has been developed by our team. There were other categories for the application threads, which included tasks that required a lot of input and output from the computer. Due to the fact that they need spontaneous responses from the end user and use a limited amount of resources, activities that are I/O-intensive were not taken into account for offloading by our system. When selecting CPU-intensive tasks for offloading, the size of the job, the number of instructions, the length of time it took to execute, and the device end variables were all taken into account. The approach that we have proposed is intended to enhance the performance of data stream applications when factors such as computational load and available bandwidth are altered, with a particular emphasis on makespan and energy consumption. It has been shown via the results of simulated experiments that performance is not affected by the restricted resources that are available. It is possible to extend this work for applications that need a significant amount of data stream. It is our intention to designate one more action as one that calls for a significant amount of data. It is our intention to include the concept of background augmentation into data stream applications at some point in the future. In addition to this, we would want to create a three-dimensional vector space model that takes into account throughput, RAM utilization, and energy consumption. With the assistance of the produced vector space model, we were able to guarantee a compromise between all three aspects and provide the decision engine with the ability to make a more efficient offloading decision. At the end of the day, we observe a system that is capable of optimizing throughput while also attaining minimal makespan and energy use..

9. REFERENCES

[1] Abebe, E., & Ryan, C. (2012). Adaptive application offloading using distributed abstract class graphs in mobile environments. *Journal of Systems and Software*, *85*(12), 2755–2769. doi:10.1016/j.jss.2012.05.091

[2] Alam, S., Dewangan, K., Sinharay, A., & Ghose, A. (2016, June). Mobile sensing framework for task partitioning between cloud and edge device for improved performance. In *2016 IEEE Symposium on Computers and Communication (ISCC)* (pp. 379-384). IEEE. doi:10.1109/ISCC.2016.7543769

[3] Arora, A., Khanna, A., Rastogi, A., & Agarwal, A. (2017, January). Cloud security ecosystem for data security and privacy. In *2017 7th International Conference on Cloud Computing, Data Science & Engineering-Confluence* (pp. 288-292). IEEE. doi:10.1109/CONFLUENCE.2017.7943164

[4] Baker, T., Al-Dawsari, B., Tawfik, H., Reid, D., & Ngoko, Y. (2015). GreeDi: An energy efficient routing algorithm for big data on cloud. *Ad Hoc Networks*, *35*, 83–96. doi:10.1016/j.adhoc.2015.06.008

[5] Baker, T., Asim, M., Tawfik, H., Aldawsari, B., & Buyya, R. (2017). An energy-aware service composition algorithm for multiple cloud-based IoT applications. *Journal of Network and Computer Applications*, *89*, 96–108. doi:10.1016/j.jnca.2017.03.008

[6] Begum, Y. M., & Mohamed, M. M. (2010, April). A DHT-based process migration policy for mobile clusters. In *2010 Seventh International Conference on Information Technology: New Generations (ITNG)* (pp. 934-938). IEEE.

[7] Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., & Buyya, R. (2011). CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software, Practice & Experience*, *41*(1), 23–50. doi:10.1002/spe.995

[8] Chun, B. G., Ihm, S., Maniatis, P., Naik, M., & Patti, A. (2011, April). Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems* (pp. 301-314). ACM. doi:10.1145/1966445.1966473

[9] Cuervo, E., Balasubramanian, A., Cho, D. K., Wolman, A., Saroiu, S., Chandra, R., & Bahl, P. (2010, June). MAUI: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services* (pp. 49-62). ACM. doi:10.1145/1814433.1814441

[10] Dewangan, B. K., Agarwal, A., & Pasricha, A. (2016, October). Credential and security issues of cloud service models. In *2016 2nd International Conference on Next Generation Computing Technologies (NGCT)* (pp. 888-892). IEEE. doi:10.1109/NGCT.2016.7877536

[11] Giurgiu, I., Riva, O., & Alonso, G. (2012, December). Dynamic software deployment from clouds to mobile devices. In *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing* (pp. 394-414). Springer. doi:10.1007/978-3-642-35170-9_20

[12] Goyal, S., & Carter, J. (2004, December). A lightweight secure cyber foraging infrastructure for resource-constrained devices. In *Sixth IEEE Workshop on Mobile Computing Systems and Applications, WMCSA 2004* (pp. 186-195). IEEE. doi:10.1109/MCSA.2004.2

[13] Gupta, N., & Agarwal, A. (2015, March). Context aware mobile cloud computing: review. In *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)* (pp. 1061-1065). IEEE.

[14] Kemp, R., Palmer, N., Kielmann, T., & Bal, H. (2010, October). Cuckoo: a computation offloading framework for smartphones. In *International Conference on Mobile Computing, Applications, and Services* (pp. 59-79). Springer.

[15] Khanna, A. (2015, September). RAS: A novel approach for dynamic resource allocation. In *2015 1st International Conference on Next Generation Computing Technologies (NGCT)* (pp. 25-29). IEEE. doi:10.1109/NGCT.2015.7375076

[16] Khanna, A., Kero, A., & Kumar, D. (2016, October). Mobile cloud computing architecture for computation offloading. In *2016 2nd International Conference on Next Generation Computing Technologies (NGCT)* (pp. 639-643). IEEE. doi:10.1109/NGCT.2016.7877490

[17] Kumar, K., & Lu, Y. H. (2010). Cloud computing for mobile users: Can offloading computation save energy?

[18] *Computer*, *43*(4), 51–56. doi:10.1109/MC.2010.98

[19] Liu, J., Ahmed, E., Shiraz, M., Gani, A., Buyya, R., & Qureshi, A. (2015). Application partitioning algorithms in mobile cloud computing: Taxonomy, review and future directions. *Journal of Network and Computer Applications*, *48*, 99–117. doi:10.1016/j.jnca.2014.09.009

[20] Liu, J., Zhao, T., Zhou, S., Cheng, Y., & Niu, Z. (2014). CONCERT: A cloud-based architecture for next-generation cellular systems. *IEEE Wireless Communications*, *21*(6), 14–22. doi:10.1109/MWC.2014.7000967

[21] Meilander, D., Glinka, F., Gorlatch, S., Lin, L., Zhang, W., & Liao, X. (2014, April). Bringing mobile online games to clouds. In *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on* (pp. 340-345). IEEE. doi:10.1109/INFCOMW.2014.6849255

[22] Niu, R., Song, W., & Liu, Y. (2013). An energy-efficient multisite offloading algorithm for mobile devices.

[23] *International Journal of Distributed Sensor Networks.*

[24] Nusrat Pasha, D., Agarwal, A., & Rastogi, R. (2014). Round Robin Approach for VM Load Balancing Algorithm in Cloud Computing Environment. *International Journal*, *4*(5).

[25] Ra, M. R., Priyantha, B., Kansal, A., & Liu, J. (2012, September). Improving energy efficiency of personal sensing applications with heterogeneous multi-processors. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing* (pp. 1-10). ACM. doi:10.1145/2370216.2370218

[26] Ra, M. R., Sheth, A., Mummert, L., Pillai, P., Wetherall, D., & Govindan, R. (2011, June). Odessa: enabling interactive perception applications on mobile devices. In *Proceedings of the 9th international confereF]nce on Mobile systems, applications, and services* (pp. 43-56). ACM. doi:10.1145/1999995.2000000

[27] Rellermeyer, J. S., Alonso, G., & Roscoe, T. (2007, November). R-OSGi: distributed applications through software modularization. *In Proceedings of the ACM/IFIP/USENIX 2007 International Conference on Middleware* (pp. 1-20). New York: Springer-Verlag, Inc.

[28] Rellermeyer, J. S., Riva, O., & Alonso, G. (2008, December). AlfredO: an architecture for flexible interaction with electronic devices. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware* (pp. 22-41). New York: Springer-Verlag, Inc. doi:10.1007/978-3-540-89856-6_2

[29] Rudenko, A., Reiher, P., Popek, G. J., & Kuenning, G. H. (1998). Saving portable computer battery power through remote process execution. *Mobile Computing and Communications Review*, *2*(1), 19–26. doi:10.1145/584007.584008

[30] Sinha, K., & Kulkarni, M. (2011, May). Techniques for fine-grained, multi-site computation offloading. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (pp. 184-194). IEEE Computer Society. doi:10.1109/CCGrid.2011.69

[31] Sirohi, P., & Agarwal, A. (2015, September). Cloud computing data storage security framework relating to data integrity, privacy and trust. In *2015 1st International Conference on Next Generation Computing Technologies (NGCT)* (pp. 115-118). IEEE. doi:10.1109/NGCT.2015.7375094

[32] Smit, M., Shtern, M., Simmons, B., & Litoiu, M. (2012, November). Partitioning applications for hybrid and federated clouds. In *Proceedings of the 2012 Conference of the Center for Advanced Studies on Collaborative Research* (pp. 27-41). IBM Corp.

[33] Tao, Y., Zhang, Y., & Ji, Y. (2015, March). Efficient Computation Offloading Strategies for Mobile Cloud Computing. In *2015 IEEE 29th International Conference on Advanced Information Networking and Applications (AINA)* (pp. 626-633). IEEE. doi:10.1109/AINA.2015.246

[34] Tomar, R., Khanna, A., Bansal, A., & Fore, V. (2018). An Architectural View Towards Autonomic Cloud Computing. In *Data Engineering and Intelligent Computing* (pp. 573–582). Singapore: Springer. doi:10.1007/978-981-10-3223-3_55

[35] Verbelen, T., Stevens, T., De Turck, F., & Dhoedt, B. (2013). Graph partitioning algorithms for optimizing software deployment in mobile cloud computing. *Future Generation Computer Systems*, *29*(2), 451–459.

doi:10.1016/j.future.2012.07.003

[36] Wang, C., & Li, Z. (2004, June). Parametric analysis for adaptive computation offloading. *ACM SIGPLAN Notices*, *39*(6), 119–130. doi:10.1145/996893.996857

[37] Wang, K., Rao, J., & Xu, C. Z. (2011, March). Rethink the virtual machine template. *ACM SIGPLAN Notices*, *46*(7), 39–50. doi:10.1145/2007477.1952690

[38] Yang, L., Cao, J., Yuan, Y., Li, T., Han, A., & Chan, A. (2013). A framework for partitioning and execution of data stream applications in mobile cloud computing. *Performance Evaluation Review*, *40*(4), 23–32. doi:10.1145/2479942.2479946

[39] Zhang, X., Kunjithapatham, A., Jeong, S., & Gibbs, S. (2011). Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing. *Mobile Networks and Applications*, *16*(3), 270–284. doi:10.1007/s11036-011-0305-7

[40] Zhang, Y., Huang, G., Liu, X., Zhang, W., Mei, H., & Yang, S. (2012, October). Refactoring android java code for on-demand computation offloading. *ACM SIGPLAN Notices*, *47*(10), 233–248. doi:10.1145/2398857.2384634

[41] Zhang, Y., Yan, J., & Fu, X. (2016, April). Reservation-based resource scheduling and code partition in mobile cloud computing. In *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)* (pp. 962-967). IEEE.

[42] Zhao, B., Xu, Z., Chi, C., Zhu, S., & Cao, G. (2010, December). Mirroring smartphones for good: A feasibility study. In International Conference on Mobile and Ubiquitous Systems: *Computing, Networking, and Services (pp. 26-38). Springer.*