An Efficient Agent Based Offloading Decision Maker For Mobile Cloud Computing

SHANTHI THANGAM MANUKUMAR (mshanthithangam@gmail.com)
College of Engineering Guindy, Anna University, Chennai https://orcid.org/0000-0001-5026-4889
Vijayalakshmi Muthuswamy
College of Engineering Guindy, Anna University, Chennai
Bushra H
College of Engineering Guindy, Anna University, Chennai

Research Article

Keywords: Agent based, Mobile cloud, Offloading, Computational device

Posted Date: April 26th, 2021

DOI: https://doi.org/10.21203/rs.3.rs-452089/v1

Abstract

The usages of mobile devices are drastically increasing every day with high end support to the users. The high end configurations mobile devices such as smart phones, laptops, tablets, etc., computations are complex in these devices. Computation intensive and data intensive are plays a vital role in the mobile devices. The main challenges in the mobile devices are handling the mobile applications in the devices with high computation and high storage. The above mentioned challenges can be overcome by using mobile cloud computing. The limitations while handling the mobile cloud computing is offloading decision making, which part of computation should offload and which should execute in the mobile side. The proposed work provides the solution to the limitations and challenges mentioned earlier by providing agent based offloading decision maker for mobile cloud. The decision maker should decide which computation part is executed in the mobile side and the cloud side. The evaluation shows the mobile applications having high complexity get benefited over other high applications. The proposed system achieves the better response time, low latency, cost-effective and minimizes the energy consumed by data-intensive and computational-intensive mobile applications.

1. Introduction

Cloud computing is one of the most important information technologies used by most of the industries and devices. The cloud plays vital role in resource management, computing power and storage services in low cost, on-demand by using virtualization. Fog computing are the emerging technology as it uses the edge devices to minimize the cost and energy consumed by communication between the cloud and the mobile devices, IoT devices and any other devices for computing services. Mobile device users are increased drastically and high computation mobile applications are also increased in the same manner. Energy consumption of mobile devices is increased and facing battery, power consumption in the mobile devices by using the high computation applications and huge data storage and access. To overcome the above limitations, mobile cloud computing is introduced, which is the combination of mobile computing and cloud computing. Mobile cloud computing deals with main technologies are (i) cloud resources such as Microsoft Azure, AWS etc., (ii) Mobile devices and its application such as laptop, smart phones, tablets and applications such as AR/VR app, video streaming apps, etc., with the help of internet, mobile device applications are connected to the high end resource provider, the cloud. The two tier mobile cloud provides the way to use the cloud resource by combining the cloud resource and mobile devices (Niroshine Fernando 2013). With the support of high end resource provider mobile applications are performed well. Offloading is technique in which the computation of mobile application is executed in the cloud side. Computation intensive offloading and data intensive offloading are important part in the offloading. Deciding which part of the application should offload and which part should execute in the local is the difficulties facing while offloading. The main challenge faced by the mobile side is energy consumption when offloading from the mobile side to the cloud. Another main problem arises while offloading, which part of the computation should execute in the mobile side and in the cloud side. To solve these problems the proposed work provides the best solution based on the offloading decision

making by introducing agent based system. Thangam et al. (2016) the focus is about the resource provisioning algorithms, its requirements in Mobile Cloud Computing.

The main contributions of this paper are:

1. Handling the mobile cloud computing is offloading decision making, which part of computation should offload and which should execute in the mobile side
2. Data-intensive and computational-intensive mobile applications offload and selection of cloud.
3. The better response time, low latency, cost-effective and minimizes the energy consumed

The rest of the paper is organized as, section 2 related works consists of how the previous researchers face the challenge, issues and how the researcher provides the result in their aspect. Section 3 system architecture which provides how this proposed system works. Section 4 describes about the implementation and experimental works. Section 5 provides the result and performance evaluation and section 6 describes about the conclusion and future work.

2. Related Works

Zhang et al. (2016) discusses about how the decisions are made for computational offloading, and they compare the computational offloading framework and data intensive offloading with the big data. Satyanarayanan et al. (2009) suggested cloudlet based on virtual machine (VM) support with mobility, scalability, and elasticity. Cloudlets are assigned as VM and offloads the entire application based the VM support application partitioning while offloading from mobile to the cloud. Zhang et al. (2016) developed an algorithm for offloading from the mobile to the cloudlet by considering the load in the local side and cost availability of the cloudlets. Based on mobility, cloudlet cost and obtain minimized offloading cost with Markov Decision process. Manukumar et al. (2020) proposed A Novel Resource Management Framework for Fog Computing by Using Machine Learning Algorithm to allocate the resource in the effective manner. Fernando et al. (2013) presents a survey about Mobile cloud computing, offloading and its issues. Offloading were introduced the part of the application is offloaded to the cloud side, VM migration are introduced based. Cuervo et al. (2010) suggested a new offloading framework MAUI which main aim is to save energy in the smart phone. The decision is based on the profiler introduced in the MAUI framework. Accelerating task completion in mobile offloading by restart mechanism, it allows several offloading retries before a job is restarted locally and completed in local side if several offloading attempts fail (Qiushi Wang, Katinka Wolter 2018). Fabio Palumbo (2021) revived about the cloud-network performance, the making of deployment decisions and cloud to user latency.

Lanitha et al. (2020) proposed a method for selecting the virtual machines in cloud computing with optimization algorithm to improve the performance of the cloud security. Multi-Component Application Placement Problem (MCAPP) in Mobile Edge Computing, the users' application is split into several components and it can be able to run in the cloud side or in the edge side. Mixed Integer Non-Linear Program (MINLP) is used to formulate the problem and this work objective is to minimize the total cost to

run the applications (Tayebeh Bahreini, Daniel Grosu 2020). Yun Li et al. (2019) formulated Queueing model using Lyapunov optimization to provide the better system offloading for the mobile users and to analyze the queue and system offloading. S. Deng et al. (2020) optimize the cost of application development and provide a solution for the application portioning and allocation by placing the whole application in the single server and not considering but assigning to edge servers. Shreshth Tuli et al. (2020) proposed an Asynchronous-Advantage-Actor-Critic (A3C) based real-time scheduler for stochastic Edge-Cloud environments allowing decentralized learning, concurrently across multiple agents. The Residual Recurrent Neural Network (R2N2) architecture is used to capture a large number of host and task parameters together with temporal patterns to provide efficient scheduling decisions. Manukumar et al (2019) proposed a novel framework for offloading the mobile application task in an effective way using multi-objective decision making in mobile edge computing. Yun Li et al. (2018) Resource reservation and allocation (RRA) with uncertain demands of mobile users is formulated as a robust optimization model. Mukherjee et al. (2019) focused on power and latency aware cloudlet selection for multi-cloudlet environment with the introduction of proxy server. To offload a code of an application for a given optimum cloudlet selection method is proposed and it reduces the power consumption and system response time.

3. System Architecture

Figure 1 shows the overall architecture of computation offloading framework which comprises of mobile application as client and cloud resources as a server providing service to the client. The framework consists of a profiler, decision engine, and offloading manager on the client-side. The server consists of a Mobility-RPC server running as a standalone server. The profilers profile the mobile device and provide information regarding the device and network. The decision engine takes up the profile information as input and determines if offloading should be done. The offloading manager is a mobile agent that gets input from the decision engine and offloads the code partition to the cloud for execution. The agent executes the code in the cloud and integrates the result back with the client.

The detailed design of the profiler module includes a device profiler and network profiler. The device profiler monitors the device status and the network profiler monitors the network status. The decision engine gets the profiler result as input to determine the device resource limitation. The device profiler monitors the hardware profiles such as CPU usage, battery level, memory usage which represent the operating conditions of the mobile device being monitored. The battery status information contains the battery level and whether the device is charging or not. If it is charging, then the charging type is identified. The device profiler is fed into the decision engine for offloading decisions. The network monitor determines whether the device is connected to the network, connection type, and also determines the strength of the signal. It also maintains the log about the network details of connected devices at runtime to identify any modification. The program profiler tracks the execution of a program on the method level. The program is analyzed on the basis of lines of code without comments(nloc), Cyclomatic Complexity Number(CCN), and token count of functions and the type of mobile cloud infrastructure resource for the execution (e.g., local, cloud).

The Application partitioner partitions the application into separate tasks while preserving the semantics of the original application. Static partitioning is done where off loadable tasks are preset in the application program as offload table. Method based partitioning is performed here. Each method in the application is considered as a separate partition. The result from the program profiler is used to determine the complex methods of the program. These complex methods are set as offloadable. The offloadable methods are partitioned at runtime for execution in the cloud using a mobile agent. The decision engine determines the resource limitation of the device. It gets various inputs such as battery level, battery charging type if connected to the network, and connection type from the profilers and determines whether the code should be offloaded for execution in the cloud or the code should be executed locally. The offloading manager is a mobile agent it automatically changes from one network to another depends on the need and availability for the completion of the task, it gets input from the decision engine. If the decision is to offload then the solver of the mobile agent determines code which is set as offloadable and the communication manager determines the IP address of the server to which the code is to be offloaded. The object of the corresponding method class is passed as input to the agent which executes the corresponding code in the cloud server and brings back the result to the client. Application partitioning is used for partitioning the application task and executed without modifying or changing the original of the application. Computation offloading uses partitioning of the mobile application to separate the operational logic of a mobile application into distinct partitions, which are capable of operating independently in a distributed environment. Partitioning of elastic application is a pre-phase of computational offloading in MCC. Both application partitioning and computation offloading are parts of the execution framework mechanism.

Figure 2 shows the overall flow representing the various interactions between the modules. The application executes on mobile and the application profiling mechanism evaluates computing resources utilization, availability of resources, and computing requirements of the mobile application. In a critical situation when there are insufficient resources on the mobile device, the mobile agent is activated to separate the computational intensive components of the application at run time. The agent determines the available server node and the intensive partition of the application is outsourced to the cloud server node for remote processing. Upon successful execution of the remote components of the application, the result is returned to the main application running on the mobile device.

4. Implementation And Experimental Setup

The Device and Network profiling are implemented using the algorithm shown in the algorithm 1.

Algorithm 1 Device and NetworkProfiler

```
1: Get battery status batteryLevel,  isCharging
2: BatteryManager.getBattery();
3: if BatteryManager.BatteryStatusCharging BatteryManager.BatteryStatusFull==true
then
4:      isCharging=true
5: else
6:      isCharging=false
7: end if
8: Get cpuinfo noOfCpu
9: Get memoryinfo totalmemory, freememory, usedmemory
10: Get network status networktype, isConnected
11: WiFiManager.getConnectionInfo();
12: if isConnected==true then
13: networkType=TelephonyManager.getNetworkType();
14: else return NotConnected
15: end if
16: Get networkspeed wifispeed
```

The battery status includes current battery level and if the device is charging or not. This is determined using the BatteryManager class. The CPU info gives detail about the number of cores by reading the Android system files located in the root directory. Memory info contains information about total bytes of memory, free memory, and used memory. The connection status gives information on whether the device is connected or not connected to the network. If the device is connected to the network then network type is found. If the network type is WiFi then link speed is determined using WiFiManager class. The output from the profiler is given to the decision engine as input. Program Profiler gives the complexity of the program is determined using Lizard tool. Lizard is an extensible Cyclomatic Complexity Analyzer for programming languages. It does static code analysis. It determines the complexity of the methods in the program. The program is analyzed on the basis of lines of code without comments (nloc), Cyclomatic Complexity Number (CCN), and token count of functions. The complex methods are determined and set as offloadable based on the static program analysis. These offloadable codes are given to the offload manager as input. The decision engine takes the battery level, charging state, network connection state and determines if the code should be offloaded for execution in a remote cloud or executed locally in a

mobile device. If the decision is not to offload then code is executed locally. If the decision is to offload then the offload manager gets the input from the decision engine and offloads the offloadable codes set by using static program profiling to the cloud server for remote execution. Mobility-RPC is used as an offload manager. Mobility-RPC is a Java library to bring seamless code mobility to any application. for this purpose.

Code mobility is the ability to transfer objects and the supporting bytecode, dynamically and statefully from one machine or application to another at runtime. The library allows the client application to transfer a regular Java object from the local JVM into the application in the remote JVM. Once in the remote JVM, the object is free to call any methods in the remote application, and if it wishes, return objects from the remote JVM back to the client application. The code mobility aspect of the library comes into play because it does not require the classes which implement the object to exist in advance in the remote application. The Mobility-RPC library automatically loads classes from the client application into the remote application and caches them there for the duration of the session. In the Mobility-RPC library, serialization, bytecode transfer, and caching are handled transparently, using a communication protocol designed and optimized specifically.

Algorithm 2 Decision Engine

- **Input:** Device Profiler, Network Profiler
- **Output:** Offloading decision

```
1: if batterylevel < threshold or isCharging == false
2: if isInternetConnected==true then
3:     OffloadingDecision=true
4: else
5:     OffloadingDecision=false
6: endif
7: endif
```

The approach to serialization is based on Kryo, and as such is much better than Javas built-in serialization - for example, there is no need to implement java.io.Serializable, and therefore most standard Java objects are inherently compatible with the library without modification. Offload Manager gets input from the decision engine. The program profiler does static analysis of the code complexity and determines the complex program code which should be set to be offloaded. This offlodable code object is given as input to the execute method of Quickstart class in Mobility-RPC library. The code is executed in a cloud server and the result is integrated back with the program.

5. Performance Evaluation And Results

The device profile information such as battery percentage level and charging status whether it is charging or not. The CPU information gives number of CPU. The memory status gives available memory, total memory at runtime, the memory used at runtime and amount of free memory at runtime in bytes. It also gives network profile information such as network type, network connection status. If network type is WiFi the current link speed is also displayed in the result. The device profile and network profile information helps the decision engine to determine the available resources of mobile device and take optimal offloading decision. The methods in NQueens solver application. The number of lines of code without comment (NLOC), The Cyclomatic Complexity Number (CCN) and token count of the method is listed. Based on the result of program profiling the method totalNQueens is set as offloadable.

The NQueen solver application is taken as test application. The application calculates the total number of solution available to place N Queens on an NxN chessboard so as none of them is able to capture another using the chess standard moves. The result shows total number of solution along with total time taken to find the solution. The output of local execution of NQueens solver application. The application after offloading to cloud and the execution of the program in cloud using Mobility-RPC as mobile agent. The totalNQueens method instance of the NQueens solver program is sent to the cloud based on the decision engine result using Mobility-RPC.

The output of the partitioned method is integrated back with the other methods of the application running in local mobile device and the result of the application is displayed on the device output screen. In this section, the performance of the mobile agent based offloading framework is analyzed with respect to execution time of NQueens solver application. The NQueens puzzle is the problem of placing N chess queens on an N x N chessboard such that no two queens can attack each other. Solutions for the problem exist for all natural number N except for 2 and 3. The NQueens solver application finds all possible solution and displays the total number of solution. The application is run on the local device and offloaded to cloud separately, for N = 4 to N = 16. The mobile device cannot handle more than 16 queens due to memory limitations.

The proposed framework is compared with different offloading frameworks based on time taken for executing the NQueen solver application in cloud. The mobile agent based offloading framework executes the task in much lesser time when compared with other offloading frameworks. Figure 4 shows comparison of offloaded execution time taken by different offloading frameworks and the proposed mobile agent based offloading framework.

6. Conclusion And Future Work

Thus in the proposed system, a dynamic computation offloading framework for MCC based on autonomous application modules offloaded to the cloud using mobile agents was developed. The proposed framework takes advantage of the high performance and reliability characteristics of mobile agents to execute computationally intensive tasks of mobile application, with an easy to adopt interface

and minimal cloud infrastructure requirements. The profiler monitors the device state and network state. The decision engine determines resource limitation on the mobile devices based on the output from the device and network profiler and takes the decision whether to execute the task locally or remotely using a mobile agent. Experiments with NQueens solver application show that the proposed framework achieves significantly higher performance than on-device execution of computationally intensive tasks. The execution time is reduced by up to 40% in the NQueens solver application by offloading the computationally intensive tasks using a mobile agent. This work can be further enhanced by making the decision engine more optimum by considering which part of the code should be offloaded based on cost estimation for local and remote execution and partition the code automatically. The Cloud server selection can be done based on the speed of execution in the Cloud and capacity of the Cloud server. The Cloud server can be made more energy efficient by scheduling the offloaded tasks from clients to avoid Cloudlet task overloading.

Declarations

Acknowledgements

This research work is supported by Visvesvaraya PhD Scheme for Electronics and Information Technology, New Delhi. One of the authors Miss Shanthi Thangam Manukumar is thankful to the Government of India for providing financial support to carry out this research work.

Compliance with ethical standards

Ethical approval

This article does not contain any studies with the participants of humans or animals.

Funding details

This work is supported by Visvesvaraya PhD Scheme for Electronics & IT for the first authorAwardee Number is VISPHD-MEITY-2559.

Conflict of interest

The authors declare that they have no conflict of interest.

Informed consent

Informed consent was obtained from all individual participants included in the study.

Author's contribution

The authors contributed to each part of this paper equally.

References

Cuervo E, Balasubramanian A, Cho DK, Wolman A, Saroiu S, Chandra R, Bahl P (2010) MAUI: Making Smartphones Last Longer with Code Offload. Proceedings of the 8th International Conference on MobileSystems, Applications, and Services, ACM MobiSys 49–62

Chun BG, Ihm S, Maniatis P, Naik M, and Patti (2011) Clonecloud: ElasticExecution between Mobile Device and Cloud. Proceedings of the Sixth Conference on Computer Systems, ACM, EuroSys 301–314

Manukumar ST, Muthuswamy V (2019) A Novel Multi-Objective Efficient Offloading Decision Framework in Cloud Computing for Mobile Computing Applications. Wireless Pers Commun 107:1625–1642

Kosta S, Aucinas A, Hui P, Mortier R, Zhang X (2012) ThinkAir: Dynamic Resource Allocation and Parallel Execution in the Cloud for Mobile Code Offloading, In INFOCOM, 2012 Proceedings IEEE, 945–953

Kovachev D, Yu T, Klamma R (2012) Adaptive Computation Offloading from Mobile Devices into the Cloud. Proceedings of IEEE 10th International Symposium on Parallel and Distributed Process with Applications (ISPA) 784–791

Bowen Zhou, Dastjerdi V, Calheiros RN, Srirama SN, Buvya R (2017) mCloud:A Context-Aware Offloading Framework for Heteroge- neous Mobile Cloud. IEEE Trans Serv Comput 10:797–810

Hao Qian and Daniel Andresen (2014) Jade: An Efficient Energy-Aware Computation Offloading System with Heterogeneous Network Inter- face Bonding for Ad-Hoc Networked Mobile Devices. Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 15th IEEE/ACIS International Conference, 1–8

Niroshine Fernando S, Loke W, Rahayu W (2013) Mobile Cloud Computing: A Survey. Future Generation Computer Systems 29:84–106

Keke Gai M, Qiu H, Zhao L, Tao, Zong Z (2016) Dynamic energy-aware cloudlet-based mobile cloud computing model for green computing. Journal of Network Computer Applications 59:46–54

Satyanarayanan M, Bahl P, C´aceres R, Davies N (2009) The Case for VM-Based Cloudlets in Mobile Computing. IEEE Pervasive Comput 8:14–23

Zhang Z, Li S (2016) A Survey of Computational Offloading in Mobile Cloud Computing. 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud), Oxford, 81–82

Zhang Y, Niyato D, Wang P (2015) Offloading in Mobile Cloudlet Systems with Intermittent Connectivity. IEEE Trans Mob Comput 14:2516–2529

Manukumar ST, Vijayalakshmi M (2020) A Novel Resource Management Framework for Fog Computing by Using Machine Learning Algorithm. In Goundar S, Bhushan SB, Rayani PK (eds), Architecture and Security Issues in Fog Computing Applications 42–52. IGI Global

Choy S, Wong B, Simon G, Catherine Rosenberg (2014) A hybrid edge-cloud architecture for reducing on-demand gaming. latency Multimedia Systems 20:503–519

Qiushi Wang K, Wolter (2018) Accelerating task completion in mobile offloading systems through adaptive restartSoftware. System Model 17:397–413

Lanitha B, Karthik S (2020) Performance improvement of cloud security with parallel anarchies society optimization algorithm for virtual machine selection in cloud computing. Soft Comput 24:15081–15092

Bahreini T, Grosu D (2020) Efficient Algorithms for Multi-Component Application Placement in Mobile Edge Computing in IEEE Transactions on Cloud Computing

Shreshth Tuli S, Ilager K, Ramamohanarao, Buyya R (2020) Dynamic Scheduling for Stochastic Edge-Cloud Computing Environments using A3C learning and Residual Recurrent Neural Networks in IEEE Transactions on Mobile Computing,

Fabio Palumbo G, Aceto A, Botta D, Ciuonzo V, Persico A, Pescapé (2021) Characterization and analysis of cloud-to-user latency: The case of Azure and AWS. Comput Netw 184:107693

Yun Li J, Liu B, Cao, Wang C (2018) Joint Optimization of Radio and Virtual Machine Resources with Uncertain User Demands in Mobile Cloud Computing. IEEE Trans Multimedia 20:2427–2438

Yun Li; Shichao Xia; mengyan Zheng; Bin Cao; Qilie, Liu (2019) Lyapunov Optimization Based Trade-Off Policy for Mobile Cloud Offloading in Heterogeneous Wireless Networks. IEEE Transactions on Cloud Computing

Deng S, Xiang Z, Taheri J, Mohammad KA, Yin J, Zomaya A, Dustdar S,(2020) Optimal application deployment in resource constrained distributed edges. IEEE Trans. on Mobile Computing

Thangam SM, Vijayalakshmi M, Precillah, Manju M, Resource Provisioning Algorithmsin Mobile Cloud Computing - Survey (January 8, 2016). Australian Journal of Basic and Applied Sciences, 10(2): 156–161

Dinh HT, Lee C, Niyato D, Wang P (2013) A survey of mobile cloud computing: Architecture, applications, and approaches. Wireless Commun Mobile Comput 13(18):1587–1611

Mukherjee D De and D. G. Roy, "A Power and Latency Aware Cloudlet Selection Strategy for Multi-Cloudlet Environment" in IEEE Transactions on Cloud Computing, vol. 7, no. 1, pp. 141–154, 1 Jan.-March 2019

Figures

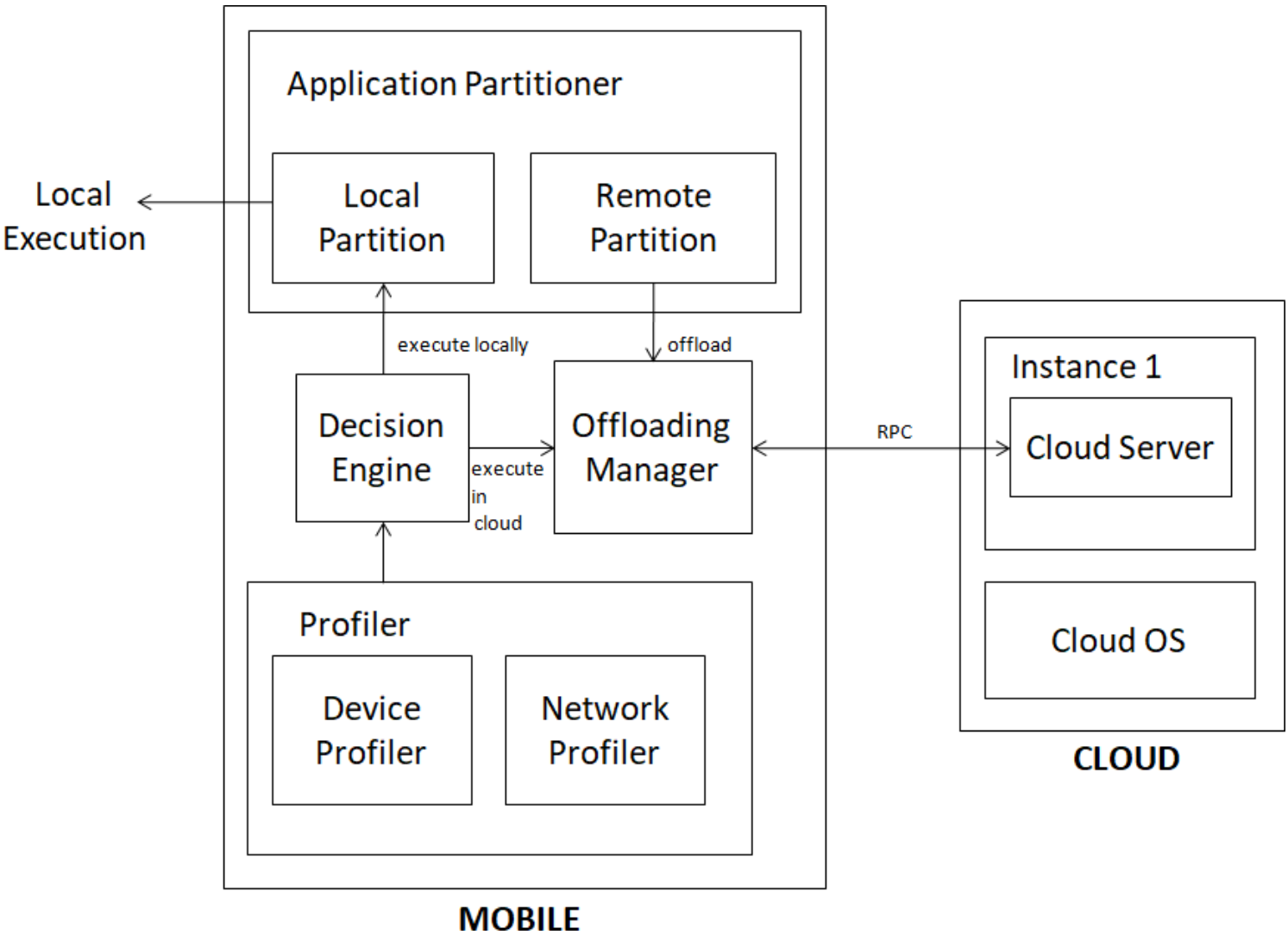

Figure 1

Overall Architecture of Computation Offloading Mobile and Cloud side

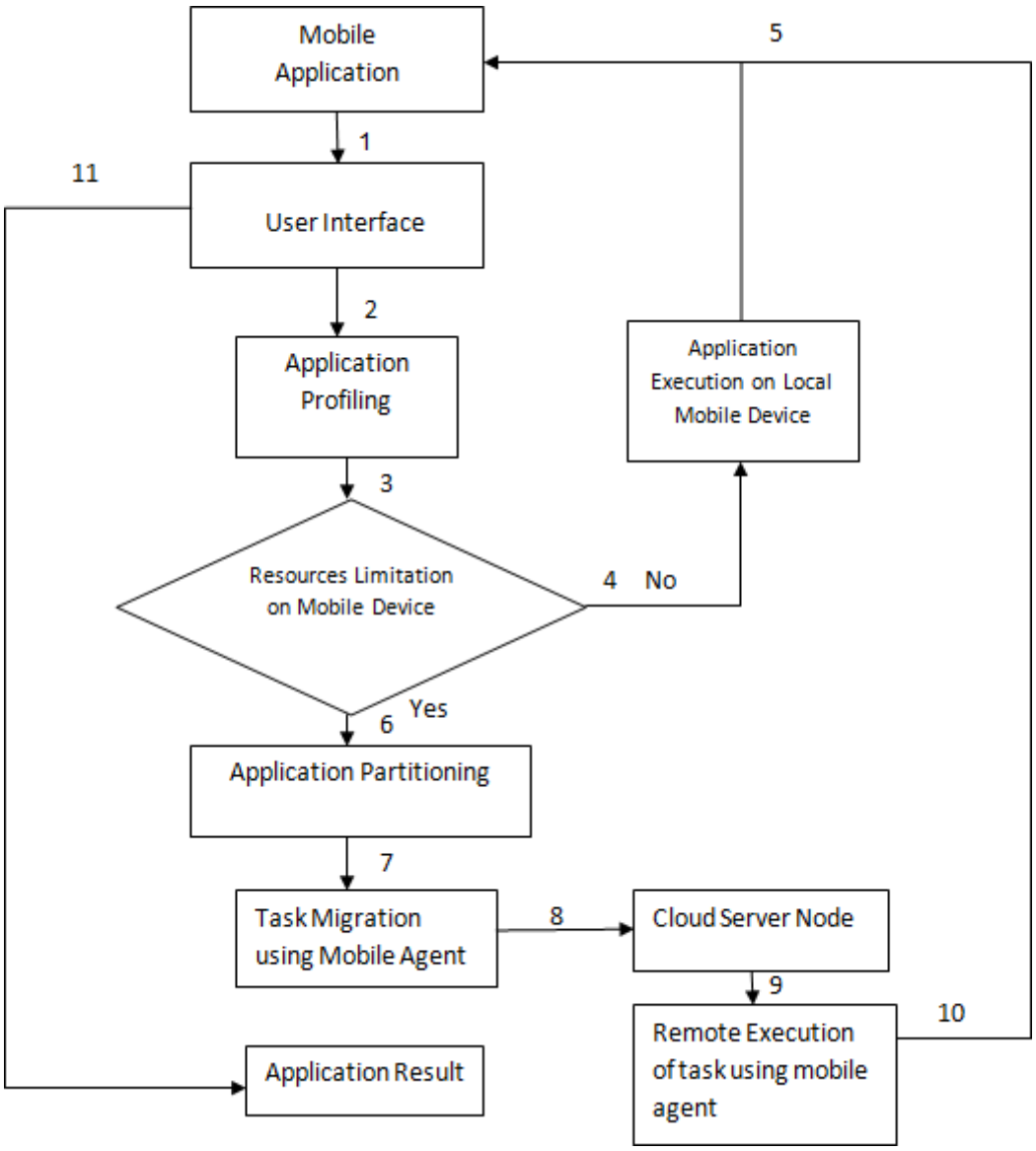

Figure 2

Overall Architecture of Computation Offloading.

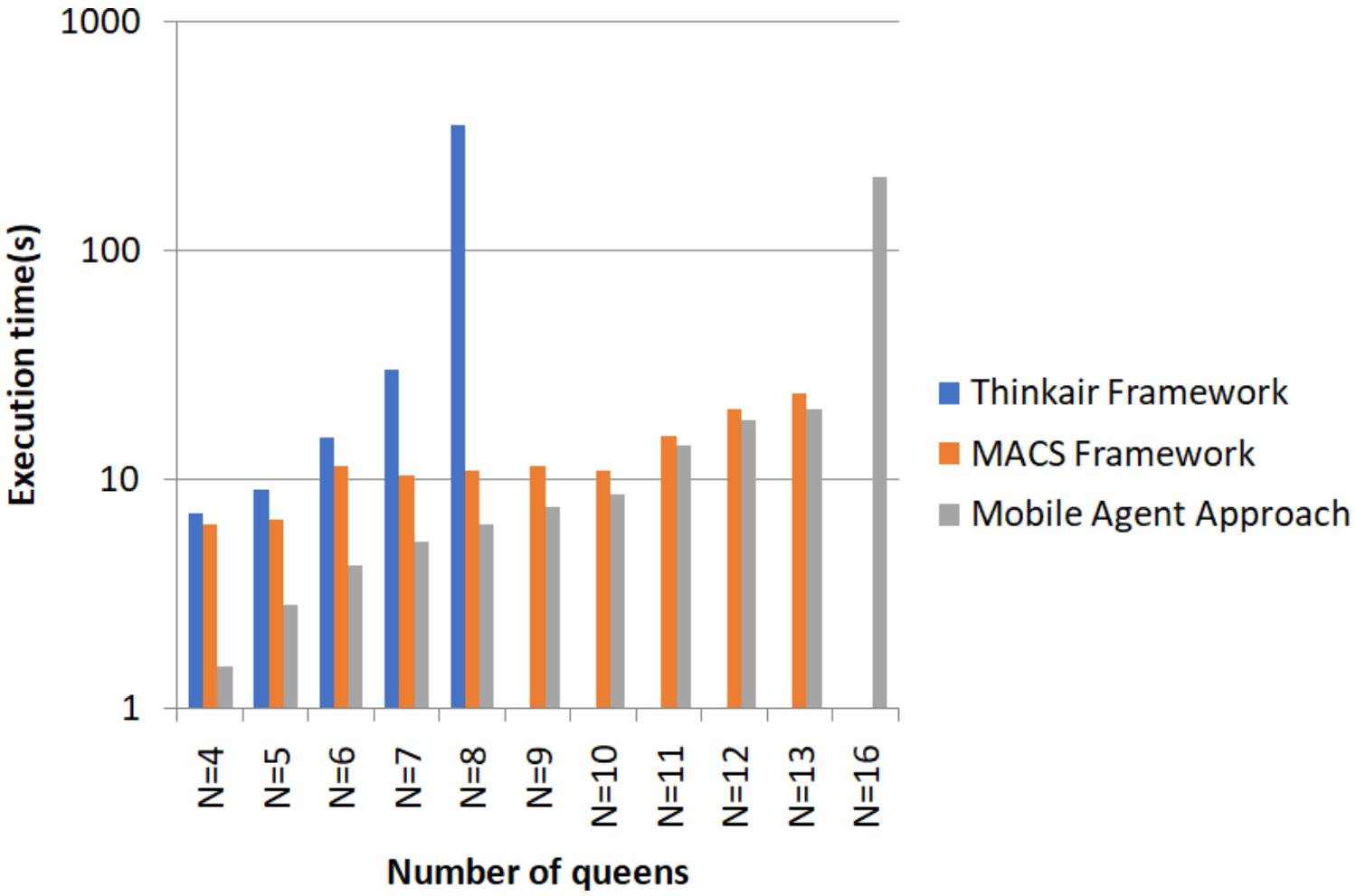

Figure 3

Comparison of Offloading with Mobile Agent Based Offloading Framework

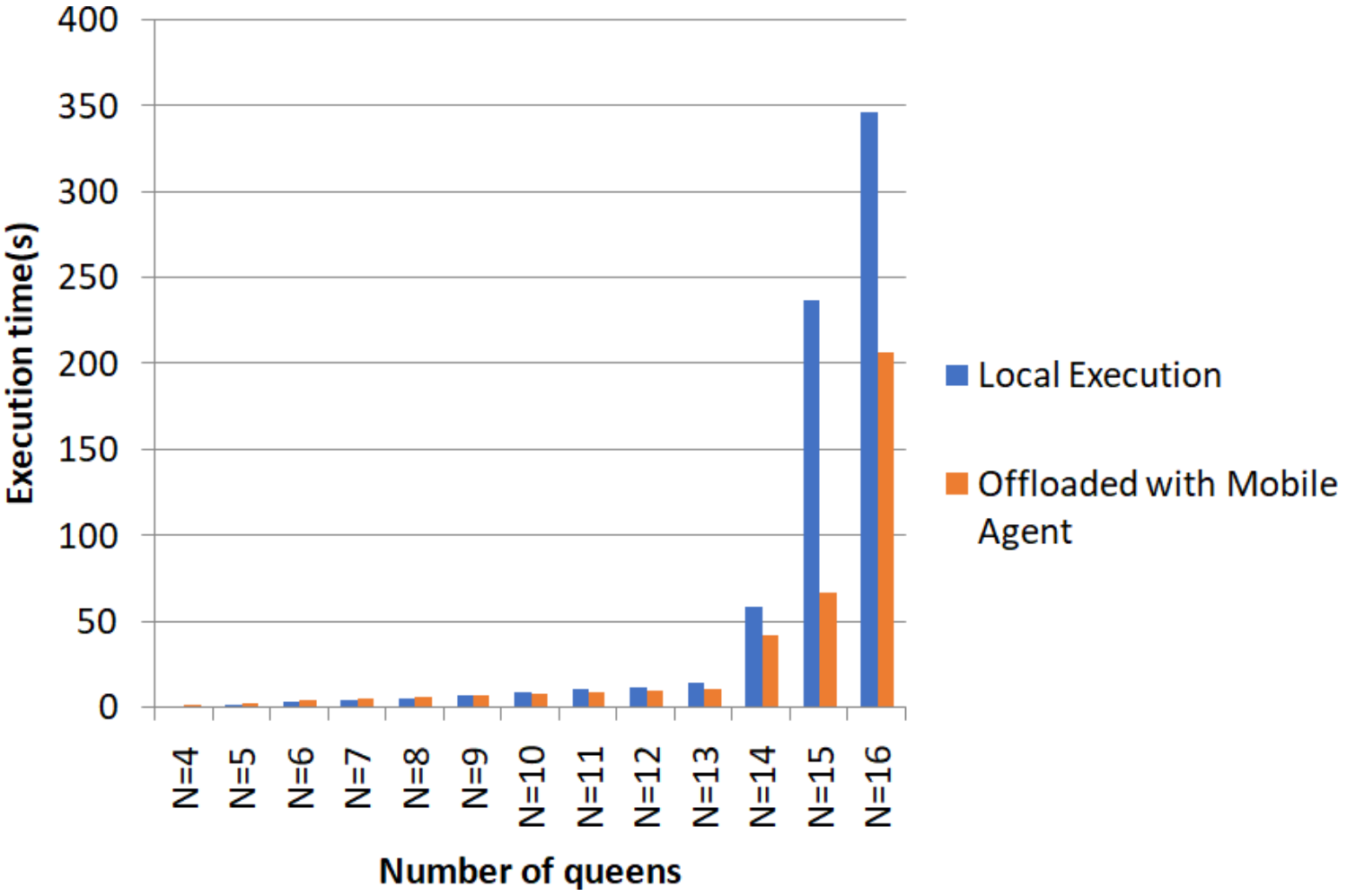

Figure 4

Performance Analysis for Execution Time in NQueens Solver Application