www.ijiarec.com

ISSN:2348-2079

Volume-5 Issue-2

International Journal of Intellectual Advancements and Research in Engineering Computations

Application offloading based On r-osgi in mobile cloud computing

Mr. S.Sambasivam, Mca. M.Phil. Professor /MCA

T.Manjula Final Mca Nandha Engineering College.

Sammy2173@gmail.com

Manjumayil542@gmail.com

Nandha Engineering College

Erode-52.

Abstract

In this project are advances of cloud computing and wireless communication technologies, Mobile Cloud Computing (MCC) has been focused on as a new paradigm to improve the processing and storage capability of mobile devices mobile application is composed of a number of modules that are eligible for offloading to be executed at a remote server on the Internet and executed at a mobile device. Compared to traditional Internet-centric Cloud service models, the complexity of mobile service management in a dynamic and distributed service environment is increased dramatically. To address this challenge, we propose to establish an R-OSGi-based mobile Cloud service model MCC-R-OSGi that uses R-OSGi Bundles as the basic mobile Cloud service building components We focused on the problem of how to transfer some modules of a mobile application to the server so as to minimize the total execution time of the whole mobile application.. Then, we formulated the offloading problem as a combinatorial optimization problem and proposed two algorithms to solve the offloading problem for a simple-chain application and also for a general application. The proposed algorithms were implemented and examined on the R-OSGi-based framework and the results show that the proposed algorithms are much more efficient than both the cases where no offloading is considered and where all the modules are offloaded to the server.

I. INTRODUCTION

In recent years due to the advances of cloud computing and wireless communication technologies, mobile cloud computing (MCC) has been focused on as a new paradigm to improve the processing and storage capability of mobile devices. In MCC, a mobile device can utilize the rich processing and storage resources of the clouds to strengthen its processing capacity and increase its data accessibility. Furthermore, by moving (offloading) some processing load from a mobile device to the server on the network can also prolong the device battery lifetime.

Authors proposed using a two-level architecture to realize the MCC framework. The first level is the general cloud infrastructure but the second level consists of a number of servers, called *cloudlets* by the authors, that may be much smaller in size and capacity than the cloud servers and are allocated in the *edge networks*, or called the *access networks* in the literature, near to the mobile devices. A cloudlet works as a second-class data center to cache the data needed by the nearby mobile devices and to process the workload of mobile devices. In this paper, we propose offloading some workload of a mobile device to the nearby server over the Internet so as to minimize the total completion time of an application that is executed at the mobile device. In particular, given a condition on which a mobile application is composed of a set In research the authors expressed the relationship between the modules of a mobile application as adirected

chain graph and proposed an offloading approach to determine a series of successive modules on

the chain for offloading. However, if a module has relations with more than one other module, the relationship between modules cannot be shown by a simple chain graph. In this paper, we propose using a directed tree graph to show the the relationship between modules in a mobile application, and then formulate the offloading problem as a combinatorial optimization problem. We propose two offloading algorithms to solve the offloading problem: one is for a simple chain application and the other for a general application. In order to examine the efficiency of the proposed algorithms, we implemented the offloading

algorithms based on the R-OSGi framework and examined the performance of the proposed algorithms.

The contributions of this paper can be summarized as follows. 1) We formulated the problem of how to offload the modules of a mobile application to the server on the network as a combinatorial optimization problem. 2) We proposed an efficient offloading algorithm for a simple-chain application. It is shown that the offloading happens only once and furthermore the offloading end point is the last module on the chain in chain applications. 3) We proposed an optimal offloading algorithm for a general application.

II. RELATED WORK

There are some researches until now related to the computation offloading prob-lems in MCC. The authors in considered each computation task as an independent entity and therefore the offloading decision of each task is performed independently from others. However, a module of a mobile application may actually have relations with other modules; for example, a module may call some other modules with some data or be called by other modules. Therefore, if a module is offloaded to the server, the modules with strong relations with the module may also be better to be offloaded.

In researches the relationship between the modules of a mobile application is taken into account in offloading decisions. However, the network congestion is ignored in order to make the problem tractable. In the modules of a mobile application are represented as a simple-chain graph or are assumed to be executed only in a parallel or a sequential pattern. The drawback of their approaches is that a general application cannot be represented correctly by a simple chain or by only a parallel or a sequential pattern, since it is common that a module has relations with more than one other module. In the execution flow of a module-based application is expressed as a directed graph and the objective is to minimize the execution time of mobile applications. It is assumed that a mobile application may have multiple execution flows, and therefore the offloading problem becomes NP-hard. A heuristic offloading algorithm was proposed to solve the offloading problem. In this paper, on the other hand, we represent the relationship among the modules of a mobile application using a directed graph and the offloading decisions are made based on how much benefit the offloading can obtain.

It is shown in researches that by allocating some cloudlets in the edge networks near to the mobile devices is an efficient way to improve the user accessibility to the cloud data and furthermore to reduce the execution time of mobile applications and power consumption of the mobile devices by offloading some computation intensive load of the mobile devices to the cloudlets. Since a cloudlet is located near to the mobile devices, the connection between them can be established by using a wireless technology such as Wi-Fi or a cellular communication network. In order to construct the environment for a mobile application at a cloudlet, a mobile user can instantiate customized service software using virtual machine (VM) technology on its nearby cloudlet, and uses that VM via a wireless channel.

The OSGi specification is a set of standards re-leased by the R-OSGi Alliance for modular component services on the Java VM and an application developed based on OSGi composes of a number of modular units, called *bundles*, decoupled through service interface. The OSGi allows one to be able to dynamically manipulate

Bundles. The R-OSGi is an extension of the OSGi framework and enables an application to be transparently distributed and executed

at different machines. In this paper, we installed the R-OSGi framework on Windows and Android OS environments using the Apache Felix and constructed the remote bundle execution platform.

III. SYSTEM MODEL

The offloading system model proposed in this paper is shown as in Figure 1 where a mobile application is composed of multiple modules. We assume that each application execution has only one thread and each module may call more than one other modules but the called modules should be executed sequentially. A module that is eligible for offloading can be executed locally or remotely at a server residing in the edge network. A mobile device, also called a *mobile terminal*, usually communicates with the server via a wireless communication channel, e.g., WiFi or cellular phone channel. If a module is eligible for offloading and if the sum of the module execution time at the server and the module transmission delay from the mobile terminal to the server is shorter than the local execution time, the module will be offloaded to the server for remote processing. Figure 1 shows an example of a module offloading where module *b* of a mobile application is offloaded to the server.

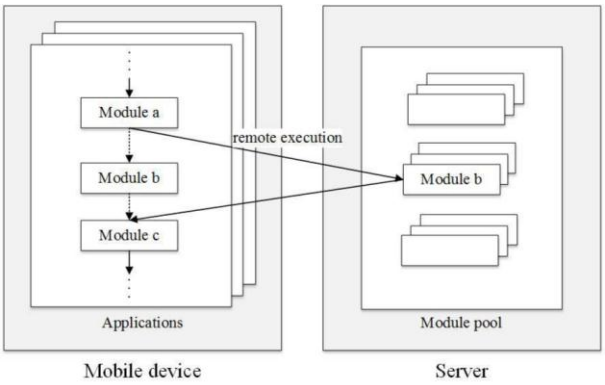


Fig. 1. Module offloading in MCC.

A. Offloading system architecture

In the OSGi framework which follows the Java modularity standard, a module is called a *bundle* and is realized by a JAR package which contains multiple class files and resource files. A lot of functions for managing the bundles such as dynamic bundle addition and execution are provided in the OSGi framework. Using these functions, one can easily develop mobile applications each of which con-sists a set of bundles. The R-OSGi is an extended mechanism to execute a module remotely at another device. In this paper, the module offloading mechanism is imple-mented by using R-OSGi and most modules of a mobile application are developed as the independent components that can be executed at the server over the network or even at another mobile device. Furthermore, the mobile applications considered here are RESTful in the sense that there is no state dependence between a caller and a callee bundles. A callee receives the input data only from its caller and on the other hand a caller receives the return data only from its callee.

B. Application execution model

The relationship between the modules of an application are represented as a directed tree graph $G(N, E)$ as shown in Figure 2 where N denotes the set of modules that constitutes the application and are eligible for offloading, and $E = \{e_{ij} \mid i, j \in N\}$ denotes the set of edges connecting modules. An edge e_{ij} represents the calling relationship of modules i and j, i.e., from caller i to callee j. In this paper, for the sake of simplicity we consider only one module that is not eligible for offloading. In reality, however, there may be more than one module that is not eligible for offloading. We can simply consider different tree graphs each of which is rooted at a module that is not eligible for offloading like node 0 in Figure 2. For an arbitrary module i, the execution times for processing it locally at the arriving mobile device and remotely at the server are denoted by L_i and R_I, respectively, and we generally have $L_i \geq R_i$. We ignore the queueing delay at a mobile device since generally only one heavy application is executed at a mobile device at a time. Furthermore, since a server is located at the edge network near to the mobile devices, e.g., attached with a Wi-Fi access point, it commonly covers a limited number of mobile devices and therefore in this paper we assume that there is no waiting delay at each server.

We assume that the input data of a module is only from its previous module, and that the output data of the module which is sent back to module 0 is much smaller than any input data and can be neglected. The input data of module i is denoted by D_i and the network speed at current time t is denoted by B_t. Therefore, the data transmission time T_i can be calculated by $T_i = D_i/B_t$. We assume that a module can call more than one other module. On the other hand, a module can be called by only one other module and if a module is called by more than one module the module is cloned and is represented as another independent module on the tree graph.

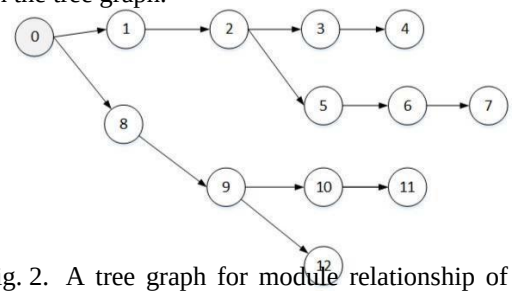


Fig. 2. A tree graph for module relationship of a mobile application.

The execution time of a mobile application depends on the processing power of the mobile device that starts the application, the processing power of the server, and also the data transmission time from the mobile device to the server. Furthermore, the transmission time of a module depends on the size of the module code which includes the data passing to the following module, and heavily on the network congestion. In order to estimate the network speed between a mobile device and the server on the network, we choose to send a small data packet periodically from a mobile device to the server. We assume that there is no data transmission delay between two modules if both the modules are processed at the same location, no matter at the mobile device or at the server.

IV. TASK OFFLOADING ALGORITHMS

In this section, we formulate a combinatorial optimization problem for module offloading and propose two offloading algorithms, one for a chain application and the other for a general application.

A. Problem formulation

We use a decision variable $x_i (i \in N)$ to indicate whether to offload module i to the server or not at time t, and if module i is determined to offload to the server we set x_i to be 1 and otherwise to be 0. We use $X = \{x_i \mid i \in N\}$ to denote the offloading strategy for all the modules of a mobile application. We define an objective function for the offloading problem, similar to the optimization problem in as follows where a gain indicating the benefit for a offloading strategy X is to be maximized.

$$\max G = \qquad x_i(L_i - R_i) - \qquad (1 - x_i)x_j T_j$$

$$\sum_{i \in N} \sum_{e_{ij} \in E} -x_i(1 - x_j)T_j, \quad (1)$$

subject to

$$T_i > 0, \ \forall e_{ij} \in E, \quad (2)$$

$$x_0 = 0, \quad (3)$$

$$x_i \in \{0, 1\}, \ \forall i \in N. \quad (4)$$

From the objective function (1), we see that once a module is offloaded to the server all the following modules intend to be offloaded and processed at the server. We can have th

e following theorem.

Theorem 1: *The offloading happens only once for a chain application and the optimal offload end point is the last module on the chain.*

Proof: Suppose that a chain application offloads k ($k > 1$) times to the server. For each offloading i ($1 \le i \le k$), suppose the starting and the end modules are s_i and e_i, respectively, then we have $s_i \le e_i$ and $e_i < s_{i+1}$. From function (1), the gain G obtained by offloading the modules k times can be written as follows.

$$G = \sum_{i=1}^{k} \sum_{j=s_i}^{e_i} (L_j - R_j) - (T_{si} + T_{ei+1}). \quad (5)$$

Since $L_i \ge R_i$ ($i \in N$), we have

$$G \le \sum_{j=s_1}^{e_k} (L_j - R_j) - \sum_{i=1}^{k} (T_{si} + T_{ei+1}) \quad (6)$$

$$\le \sum_{j=s_1}^{e_k} (L_j - R_j) - (T_{s1} + T_{ek+1}) \quad (7)$$

$$\le \sum_{j=s_1}^{|N|} (L_j - R_j) - T_{s1}. \quad (8)$$

The right hand of the inequality (7) is nothing else when the offloading starts at s_1 and ends at e_k; i.e., the offloading happens only once. Furthermore, the inequality (8) shows that the largest gain can be obtained if the end module for offloading is the last module, since the output data from the last module back to node 0 is much less than the input data to any module and the data transmission delay between two modules can be neglected if the two modules are processed at the same location. Therefore, we can conclude that the offloading happens only once for a chain application and the optimal end module for offloading is the last one.

We propose the following offloading algorithm according to Theorem 1. The algorithm searches the best starting module for offloading from the last module towards node 0. For each module i the gain obtained starting from it to the last module is calculated by $G_i = \sum_{j=i}^{|N|} (L_j - R_j) - T_i$. The best offloading point will be the module that yields the maximum gain. When the maximum value of G_i is negative, no module will be offloaded. It is easy to see that the complexity of the algorithm is bound by $O(|N|)$.

Algorithm 1 Offloading algorithm for a chain application.

1: Set maximum gain $G_{max} = G_{|N|}$ and offloading point $p_{max} = |N|$
2: for module i from $|N| - 1$ to 1 do
3: if $G_{max} < G_i$ then
4: $G_{max} = G_i$ and $p_{max} = i$
5: end if
6: end for
7: Obtain the maximum gain G_{max} and the offloading point is module p_{max}

C. Offloading algorithm for a general application

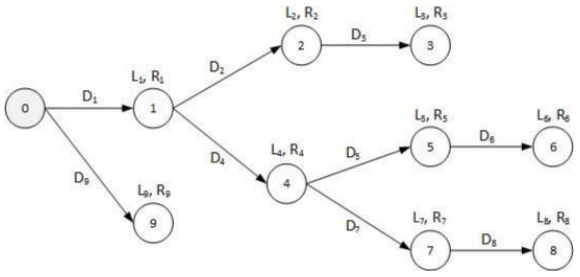


Fig. 4. A general application.

Here, we only consider the case with one module that is not eligible for offloading and represent the modules by a directed tree topology as shown in Figure 4. The root of the tree denoted by node 0 represents the module that is not eligible for offloading. For a general mobile application, we have the following theorem.

In this paper, we propose an offloading algorithm that starts to determine the best offloading point from each leaf module towards to the root of the tree graph until a joint point, e.g., module 1 or 4 shown in Figure 4. Then, the algorithm determines whether to offload the module at the joint point and forwards the decisions to the upper module. The offloading decision of a joint point module can be made only if all the offloading decisions of its children have been

made, e.g., the offloading decision of module 4 can be made only if the offloading decisions of modules nodes 5, 6, 7, and 8 have all been done. The algorithm terminates when node 0 is reached and all the module offloading decisions have been finished.

Algorithm 2 Offloading algorithm for a general application.

1: Mark all modules as *undetermined*
2: Run Algorithm 1 to determine maximum gain G_k of the branch k from each leaf module to its nearest joint point and mark modules on the branch as *determined*
3: while not reach module 0 or there is any undetermined module do
4: /* suppose there is more than one branch from joint point i and all the best offloading points of the branches are set in P_i */
5: if $G_i > \sum_{k \in P_i} G_k$ then
6: Mark module i as "to be offloaded" and *determined*
7: Send gain G_i and $P_i = \{i\}$ to parent module
8: else
9: $G_i =$
10: Mark i as *determined*, send gain G_i and best offloading point set P_i to parent module
11: end if
12: Run Algorithm 1 to decide maximum gain of the branch from i' parent to next nearest joint point and mark modules on the branch as *determined*
13: end while
14: Obtain maximum gain and offloading point set for each branch of node 0

OFFLOADING SYSTEMIMPLEMENTATION

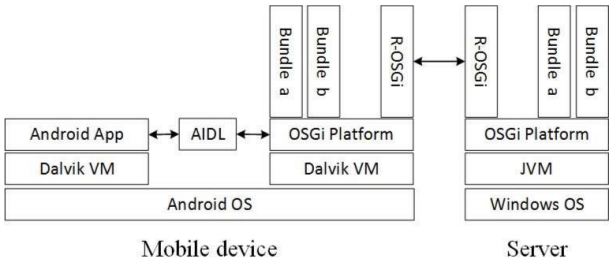


Fig. 5. System architecture.

We implemented our proposed offloading system on the R-OSGi framework that is extended from a modular decoupled components and pluggable dynamic service models, OSGi, and developed for remote component execution. There are many kinds of OSGi implementations and we chose to use an open source implementation package called Apache Felix. The system architecture of our proposed system is shown in Figure 5. In the experiments, we employed one mobile device and one server that are connected by a 4Mpbs IEEE 802.11g based wireless LAN access point. In order to examine the effect of network congestion on the offloading decisions, we established another steady data transmission flow via the access point by tuning the flow quantity. The mobile device being used in our implementation experiments is a Google Nexus 7 tablet device with 2GB RAM and Anroid OS 5.0.2. Furthermore, the server being used is a laptop computer with the Intel Core i5-5200 2.2GHz CPU, 8GB memory, and Windows 10 and is connected to the access point by a wired line.

A. Module remote execution

In our offloading system, any module that is eligible for offloading can be processed locally at a mobile device or the server located in the edge network. If the execution code of the module does not exist at the server, it will be firstly transferred to the server and then executed at the server. The code transferred to the server will be kept at the server for possible future execution. Therefore, only when the execution code of a module does not exist in the server the execution time of the module is little longer due to the code transmission delay. On the other hand, if the code exists in the server, it can be executed right away.

B. Data transmission delay estimation

The servers located in the edge networks may mostly communicate with mobile devices via the wireless LAN or the cellular communication network. In this paper, we employed one laptop computer as the server and one Google Nexus tablet terminal as the mobile device. The mobile device is connected to the server by a 54Mbps Wireless LAN access point. As described in previous section, the offloading decision depends heavily on the data transmission delay, which is determined by the network congestion, especially in a wireless communication environment. In this paper, in order to estimate the communication delay between the mobile device and the server on the network we chose to transmit a fixed size data packet of 1MB to the server periodically (once every 60s) and measured the sample round-trip time (SRTT) for each transmission. Each time when the data packet is sent to the server the mobile device times how long it takes for it to be acknowledged and obtained a round-trip time sample, say, T . In order to avoid the influence by the network fluctuation, we calculate the average RTT using an exponentially weighted moving average (EWMA) approach as follows.

$$RTT = \alpha RTT + (1 - \alpha) SRTT,$$

where RTT is the average RTT and α is a smoothing factor with a constant between 0 and 1 to control how quickly the RTT adapts to changes. Similarly to the TCP specification in we used $\alpha = 7/8$. Therefore, we can Figure 7 shows the results for executing a computation intensive application. Here, we varied the total execution times of the chain application, denoted by Cases 1, 2, 3, and 4, and the input data to each module was only 4 bytes and the data transmission delay can be ignored. Furthermore, the network speed was around 4Mbps. We see that using our

proposed algorithm all the modules are offloaded to the server since the network speed is fast enough. We see that when the computation time of the application becomes large, the gain obtained from offloading becomes significantly large. the cases where there is no offloading and all the modules are offloaded. We also see that the network speed has a key effect on the offloading decision in our proposed algorithm. Most of the modules are offloaded application by 20 executions. We see from the figure that our proposed algorithm performs better than both.

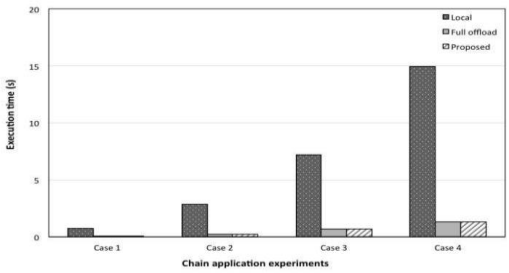
We examined a chain application that has 5 modules eligible for offloading as shown in Figure 6. All the modules have the same execution times. For each parameter setting in the experiments, the execution times of the 5 modules were fixed and and the average

$$\sum_{k \in P_i} c_k$$

completion time of the application calculated from 20 independent executions is shown in the figures.

We also examined the chain application for the case where the input data and the data transmission delay cannot be neglected; i.e., the application is both computation and data intensive. Figure 8 shows the application completion time for various network speed where the input data to each module was set to 5MB. We see from Figure 8, the network speed has a significant effect on the application completion time and on the offloading decision in our proposed algorithm. In our proposed algorithm, if the network speed is fast enough, e.g., 4MB/s, most of modules are offloaded to the server.

In the figures, the total completion time when all the modules are processed locally is denoted by ”Local” and on the other hand, the total completion time when all the modules are processed at the server is denoted by ”Full offload”. The completion time using our proposed algorithm is denoted by ”Proposed”. We also examined the general application when both the input data to each module and the data transmission delay cannot be ignored; i.e., the application is computation and data intensive. The input data to modules 1 through 9 are 18, 5, 3, 4, 2, 1, 2, 2, and 2 MB, respectively. The results shown in Figure 10 are obtained as the average completion times of the from 20 executions as shown in Figure 9. We see from this figure that, similar to the results for the chain application, when the completion time of the application becomes larger, the gain for offloading becomes larger.



B. General applications

We examined a general computation intensive application that has 9 modules eligible for offloading as shown in Figure 4. Similar to the chain application in previous subsection, we first examined the application using 4 parameter settings each with a different execution time denoted by Cases 1 no or less modules to Offloaded.

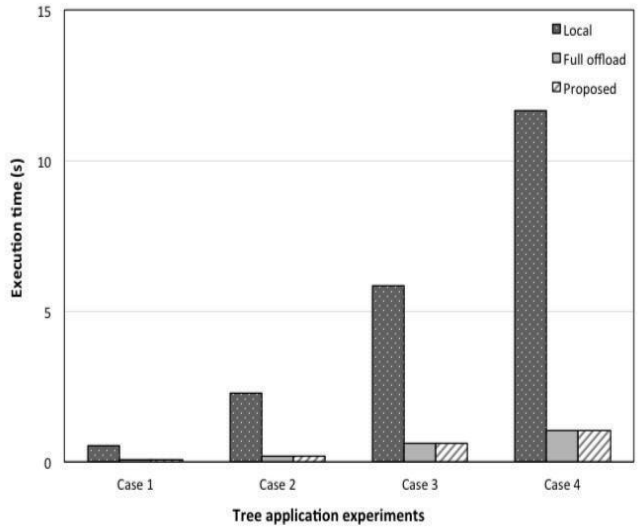


Fig. 9. Computation intensive tree application.

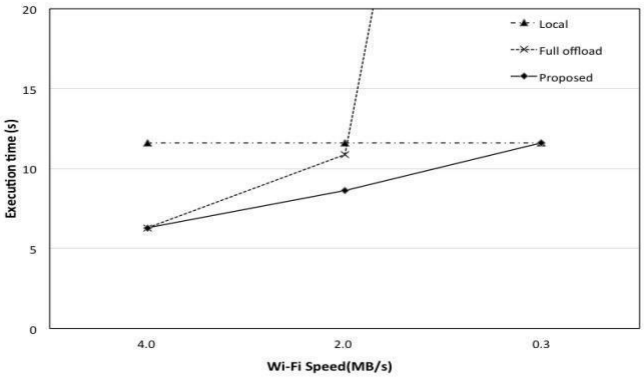


Fig. 10. Data intensive tree application.

EXISTING SYSTEM

In the existing system, proposed an offloading algorithm for a simple-chain application. It is shown that the offloading happens only once and furthermore the offloading end point is the last module on the chain in chain applications. The proposed algorithm in the existing system is an optimal offloading algorithm for a general application. It is shown that the computational complexity of this algorithm is proportional to the number of modules in the application.

Furthermore, the existing works fail to either meet the user's quality of service (QoS) requirements or to incorporate some basic principles of cloud computing such as the elasticity and heterogeneity of the computing resources. The existing system develops a static cost-minimization, deadline-constrained heuristic for scheduling a workflow application in a cloud environment. The approach considers fundamental features of IaaS providers such as the dynamic provisioning and heterogeneity of unlimited computing resources.

DRAWBACK OF THE EXISTING SYSTEM

- Adaptable only in situations where same initial set of resource availability.
- Suitable for single cloud service provider environment only.
- Data transfer cost is not considered between different cloud data centers.

PROPOSED SYSTEM

To achieve both resource provisioning and scheduling are merged and modeled as an optimization problem. PSO is then used to solve such problem and produce a schedule defining not only the task to resource mapping, but also the number of nodes to be assigned. In the thesis the process referred in the single cloud provider which is used to compute the consumption time and execution cost for running the process in the environment. The scheduling process is done in the basis of set of resources, number of task which are defined to that resource in the environment. Here to computing the result of total consumption cost and total execution time using PSO logic.

The project presented the algorithm named SEMO (Superior Element Multitude Optimization) which is compare the total execution time and total execution cost between one processes to another process. In addition, it extends the resource model to consider the data transfer cost between data in cloud environment so that nodes can be deployed on different regions.

Also, it assigns different options for the selection of the initial resource pool. For example, for the given task, the different set of initial resource requirements is assigned. In addition, data transfer cost between data environment are also calculated so as to minimize the cost of execution in multi-cloud service provider environment.

ADVANTAGES OF THE PROPOSED SYSTEM

- Adaptable in situations where multiple initial set of resource availability.
- Suitable for multiple cloud service provider environments.
- Data transfer cost is reduced between different cloud area.

EXECUTION TIME MATRIX GENERATION

This module generates the execution time matrix in which number of resources is taken as columns and tasks are taken as rows and the time the tasks taken to complete in those resources are stored as values.

TRANSFER TIME MATRIX GENERATION

This module generates the transfer time matrix in which number of taken are taken as columns and rows (square matrix is prepared) and the time a task transfers the data to other task is stored as values. So the diagonal elements are always zero since same task has no data transfer operation.

SCHEDULE GENERATION

Initially, the set of resources to lease R and the set of task to resource mappings M are empty and the Total Execution Cost TEC and time TET are set to zero. After this, the algorithm estimates the execution time of each workflow task on every resource r_i $R_{initial}$. This is expressed as a matrix in which the rows represent the tasks, the columns represent the resources and the entry $ExeTime_{i,j}$ represent the time it takes to run task ti on resource r_j. This time is calculated using Fig a.

Fig (a) Matrix representation of execution time

Fig (b) Matrix representation of transfer time

The next step is the calculation of the data transfer time matrix. Such matrix is represented as a weighted adjacency matrix of the workflow DAG (Directed acyclic graph) where the entry $TransferTime_{i,j}$ contains the time it takes to transfer the output data of task t_i to task t_j. This value is taken from database and is zero whenever ij or there is no directed edge connecting t_i and t_j. An example of these matrices is shown in Figure a) and b) below.

VII. CONCLUSION

In this paper, we express a general mobile application using a directed tree graph that consists of multiple independent modules, and formulated a combinatorial optimiza-tion problem of how to offload the modules of a mobile application to the server over the network. We proposed two efficient offloading algorithms to obtain the solutions for the offloading problem. Then, we implemented the proposed of-floading algorithms on the R-OSGi framework and examined the performance of the proposed algorithms. The experiment results show that, for a computation extensive application with small amount of input data, most of modules are of-floaded to the server. On the other hand, for a data intensive application the data transmission delay plays a key role in module offloading decisions. If the data transmission delay is negligibly small, most modules intend to be processed at the server. However, if the data transmission delay is large enough, most modules choose to be processed locally at the mobile devices.

ACKNOWLEDGMENT

This research is partially supported by Collaboration Research Grant from National Institute of Informatics, Japan.

REFERENCES

[1] X. Fan, J. Cao, and H. Mao: A Survey of Mobile Cloud Computing, *ZTE Communications*, Vol. 9, No. 1, pp. 4–8 (2010).

[2] K. Kumar and Y. Lu: Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?, *Computer*, Vol.43, No.4, pp. 51-56 (2010).

[3] H. Dinh, C. Lee, D. Niyato, and P. Wang: A Survey of Mobile Cloud Computing: Architecture, Applications, and Approaches, *Wireless Communications and Mobile Computing*, Vol. 13, No. 18, pp. 1587– 1611 (2013).

[4] M. Satyanarayanan, P. Bahl, and R. Caceres, and N. Davies: The Case for VM-based Cloudlets in Mobile Computing, *IEEE Pervasive Computing*, Vol. 8, No. 4, pp. 14–23 (2009).

[5] M. Satyanarayanan, R. Schuster, M. Ebling, G. Fettweis, H. Flinck, and K. Joshi: An Open Ecosystem for Mobile-Cloud Convergence, *IEEE Commun. Mag.*, Vol. 53, No. 3, pp. 63–70 (2015).

[6] Y. Zhang, H. Liu, L. Jiao, and X. Fu: To offload or not to offload: An efficient code partition algorithm for mobile clouding computing, *Proc. Int. Conf. Cloud Netwotking (CLOUDNET 2012)*, pp. 80–86, Paris, France (Nov. 2012).

[7] J.S. Rellermeyer, G. Alonso, and T. Roscoe: R-OSGi: Distributed Applications Through Software Modularization, *Middleware 2007, LNCS 4834*, Eds. R. Gerqueira and R.H. Campbell, pp. 1–20, Springer (2007)

[8] C.C. Lin, H.H. Chin, and D.J. Deng: Dynamic Multi-Service Load Balancing in Cloud-based Multimedia System, *IEEE Syst. J.* , Vol. 8, No. 1, pp. 225–234 (2013).

[9] L. Yang, J. Cao, Y. Yuan, T. Li, A. Han, and A. Chan: A Framework for Partitioning and Execution of Data Stream Applications in Mo-bile Cloud Computing, *ACM SIGMETRICS Performance Evaluation Review*, Vol. 40, No. 4, pp. 23–32 (2013).

[10] C. Shi, K. Habak, P. Pandurangan, M. Ammar, M. Naik, and E. Zegura: COSMOS: Computation Offloading as a Service for Mobile Devices.