Institución Universitaria
SALAZAR Y HERRERA

QUID 2017, pp. 624-640, Special Issue N°1- ISSN: 1692-343X, Medellín-Colombia

PARTITIONING MOBILE APPLICATION BY CLOUD COMPUTING USING A LINEAR PROGRAMMING ALGORITHM IN THE GRAPH

(Recibido el 15-06-2017. Aprobado el 04-09 2017)

Mostafa Ahmadi Meshkani
Department of Computer engineering, kashan Branch, Islamic Azad University, Kashan, Iran[1]
Ahmadi6468@gmail.com[*1]

Mohammad Hadi Yousefi
Department of Mechatronics, kashan Branch, Islamic Azad University, Kashan, Iran[2]
Mhu320@yahoo.com[2]

Mohammad Javad Rashidi
Department of Mechatronics, kashan Branch, Islamic Azad University, Kashan, Iran[3]
mjavadrashidi@yahoo.com[3]

Resumen: En el campo de la computación en nube, la falta de la misma capacidad para los dispositivos, así como la falta de conocimiento claro sobre el número de dispositivos utilizados conducir a no utilizar algoritmos prácticos con facilidad, por lo que el uso de algoritmo óptimo es adecuado cuando cualquier dispositivo no puede ir Más allá de su capacidad máxima. También, en este trabajo, se ha considerado la estabilidad de la partición. De hecho, un subgrafo debe ser seleccionado en términos de conectividad de modo que si un número de enlaces están desconectados, su estabilidad no se perderá y el programa se ejecutará correctamente. En este trabajo, se presentó un método para dividir el gráfico de tareas de una aplicación. Dado que el problema de la partición gráfica a gran escala es de la dureza NP, el algoritmo genético fue propuesto como una estructura selectiva en el método propuesto. En el algoritmo genético, tres criterios, incluyendo el costo, el tiempo de respuesta y la energía se utilizaron como un objetivo combinado. El uso de la programación lineal influye efectivamente en el rendimiento óptimo del algoritmo genético. Los resultados del método propuesto muestran una reducción precisa del consumo de energía y un tiempo de respuesta del 0,5% y del 3%, respectivamente.

Palabras clave: Algoritmo genético, programación lineal, particionamiento de gráficos, cloud computing

Abstract: In the field of cloud computing, lack of same capacity for devices as well as the lack of clear knowledge on the number of used devices lead to not use of practical algorithms easily, so using optimal algorithm is proper when any device fails to go beyond its maximum capacity. Also, in this paper, the stability of partitioning has been considered. In fact, a subgraph must be selected in terms of connectivity so that if a number of links are disconnected, its stability will not be missed and the program will run correctly. In this paper, a method was presented for partitioning the task graph of an application. Since the problem of large-scale graph partitioning is of the NP-hardness, the genetic algorithm was proposed as a selective structure in the proposed method. In the genetic algorithm, three criteria including cost, response time, and energy were used as a combined target. Use of linear programming effectively influences the

optimal performance of the genetic algorithm. The results of the proposed method show accurate reduction in energy consumption and the response time by 0.5% and 3%, respectively.
KEYWORDS: Genetic algorithm, linear programming, graph partitioning, cloud computing

1. INTRODUCTION

Today, the extension of knowledge boundaries is dependent on the development of computational technologies. As a starting point for these technologies, we can refer to computer network emergence in which only a few computers were connected. Subsequently, these small networks were connected to each other and created the Internet that the networks were shared on the Internet by which the concept of a global spreadsheet was created through which information was shared among users. Cloud computing is a new way of providing computing resources and a model for providing service over the Internet. In fact, cloud computing provides the ability to save on IT resources and boost computing power, so that processing power becomes an ever-reaching tool.

Mobile cloud computing is a combination of cloud computing, mobile computing and wireless networks that provide rich computing resources for mobile users, network operators, and cloud computing providers. The main purpose of mobile cloud computing is to run powerful mobile applications on a large number of mobile devices. . In the mobile cloud computing, processing is done in the cloud, data is also stored in the cloud, and mobile serves as a medium for displaying the information. With cloud computing maturing, mobile cloud loading has been become a hopeful way to reduce the runtime and battery life of mobile devices. Its main idea is to boost the run by transferring heavy computing from mobile devices to cloud servers, and then getting results from them through wireless networks. Loading is an effective way to overcome the limited resources and features of mobile phones since it can rid them of massive computing and increase the performance of mobile applications. Loading all of the computing components of an application remotely is not always necessary or effective. In mobile cloud computing, the cell phone should intelligently determine which part of the software should be computed on the mobile or loaded into the cloud. When mobile computing is increasingly interacting with the cloud, a number of methods, for example, Maui [1] and Clone Cloud [2], are provided with the aim of loading some parts of the mobile software in the cloud. In order to achieve a good performance, it must be decided that which parts of the software should be loaded into a remote cloud, and which parts of that should be run locally on mobile devices, so that the total execution cost have been be minimized. The main costs for mobile loading systems are the cost of local computing and remote execution, and communication costs mean additional communication between the mobile device and the cloud remotely. The computing can naturally be described as a graph whose crest represents computational costs, and its edge reflects communication costs [3]. By partitioning the vertices of a graph, we can divide computing among the local mobile processors and remote cloud servers. The traditional algorithm for graph partitioning (for example, [4], [5], [6], and [7]) cannot be used directly in mobile loading systems, because they ignore the weight of each node and only consider the margin weight of the graphs.

Cloud computing is the deepest distributed computing technology that is rapidly penetrating in various aspects of human life, aimed at providing easy access to integrated clusters and a unit of computing resources (storage, processing and storage) on demand, which can be dispersed as conditional elastic with minimal investment, complexity and interaction. [8]. Users use the computing and resources stored in the cloud, and It is possible to access them at any time and place by any device connected to the Internet without the need for additional costs or condition and, for example, the photos, songs and personal documents in the cloud is available without any time or place restrictions [9].

Mobile cloud computing is a technology that directs diverse clouds and network resources toward unlimited capability, storage and access without considering the time and space for many mobile devices by the Internet [10]. Cloud computing brings the user's attention to integral computing by reducing or eliminating confusion, unusual actions, inaccurate and out-of-frame results [11], which provides the users of mobile cloud computing with greater benefits over mobile computing and cloud computing. [12] Researchers use diverse design patterns in mobile cloud computing to design and optimize cloud resources on smartphones as a suggested framework for processing distributed applications on smartphone [13] for example, clone cloud [14].

2. MATERIALS AND METHODS

The challenges for graph partitioning include:

Partitioning software is very important for designing an affordable and efficient adaptive load system. Some critical issues about the partitioning problem are:

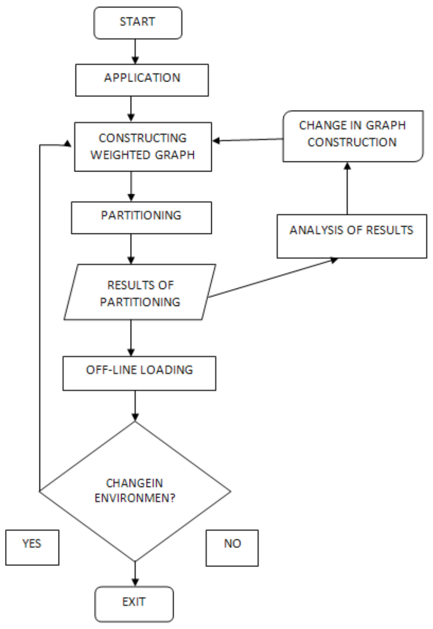
Weighing: When some parts of the software are chosen as transfer mode, we need the weight scale of each piece of used software, including memory, processing time, and bandwidth usage. Communication overhead is introduced by telecommunications between a mobile device and a cloud server.

Real-time compatibility: since the bandwidth of existing networks in wireless environments is different, the partitioning algorithms presented in previous articles assuming a fixed bandwidth for a mobile operating system are inappropriate. Partitioning algorithms must match the network and device changes. Since network conditions can be only measured at runtime, the partitioning algorithm should be an online process and in real time.

Partition Efficiency: Partitioning decisions for simple applications (for example, an alarm clock) are not difficult in real time, but some complex applications (for example, speech / face recognition) that contains a lot of methods, require a very efficient algorithm for real-time partitioning.

2.1. Software partitioning architecture

To solve these challenges, the workflow of an environment-adaptive partitioning process is presented. It starts with a program design that can be divided into multiple tasks through static and dynamic analysis technologies. After that, we design a weighted compound graph called WCG for mobile software. Then, the partitioning operation of the graph is performed.



By extracting outputs of this algorithm, we can get initial partitioning results for response time or optimization of energy consumption. During the implementation process, if the mobile environment changes, and these changes reach or exceed a certain threshold, the software graph will be partitioned again according to the new parameters. So, ultimately, elastic partitioning can be understood as a condition-conscious and adaptive environment one. Here, in the context of a mobile environment, it does not only include mobile computing resources inside the device, battery level, CPU, memory, etc., but also includes a mobile external environment, such as network connectivity and Internet speed. After partitioning, it loads automatically the distributed applications requiring remote implementation to a cloud server and does the rest locally on a mobile device according to partitioning results. Therefore, this problem whether the certain parts of an application is loaded into the cloud or not depends on the following factors: the CUP speed of the mobile device, the network bandwidth, the size of the data transfer, and the speed of the cloud server. Given these factors, we create a WCG with regard to computational and communication costs, and we design a new partitioning algorithm, especially for mobile loading systems.

2.2. Software Task Classification

Different applications are created in a mobile device by some processes, each of which consists of several different tasks. Since all software tasks are not suitable for remote implementation, they must be checked and detected:

Non-transferable tasks: Some tasks must be unconditionally run locally on a mobile device, or because the transmission of the data demands extraordinary time and energy or because these tasks must access the local components (camera, GPS, UI, accelerometer or other sensors, etc.) [1]. Tasks that may cause security issues when uploaded in a different location (e.g., e-commerce) should not be loaded. Local processing consumes the battery power of the mobile device, luckily, there is no communication or delay cost.

Loadable tasks: Some components of this software are flexible, which can either be processed locally on a mobile device processor, or remotely in a cloud infrastructure. Many tasks can be included in this category, and the decision for loading depends on whether the communication costs are greater than the difference between local and remote costs. We do not need to make a decision about loading for the loadable components. However, for loadable decisions, since loading all tasks of an application into the cloud remotely in any circumstances is not necessary or effective, it is worth to examine which tasks need to be done locally on the mobile device and which tasks needs to be done remotely on the cloud based on available networks, time loading responses, or energy consumption. The mobile device must adopt a loading decision based on the outcome of a dynamic optimization problem.

2.3. Partitioning models

A weighted graph can be constructed for a variety of applications, but one of the important problems in this regard is the selection of the topology for desired graph. Construction of a weighted graph for software partitioning is of importance. The mobile software can be displayed as a list of partial tasks, and designed in different topologies as shown in Figure 1. Where each node represents a task in the software running either on mobile device or on the cloud space for further execution [15].

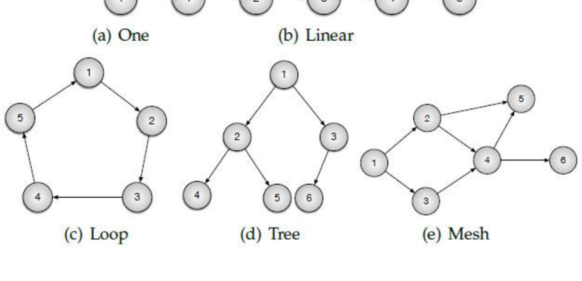


Figure 1. Different topologies for the task flow graph

Examining different types of topologies in Figure 1, we have:

(a) Only an active node: represents a whole application (without partitioning). Such topology is often moderated by the full load plan, which can also be viewed as an example of server software. In this case, the entire program is transmitted to a remote server, including the complete code transfer and program mode. The main drawback of this type of topology is the low flexibility and execution speed of high volume tasks.

(b) Linear Topology: represents an orderly list of tasks as sectional. Each task is executed by receiving output data generated by another task as input data [16].

(c) Loop-based topology: This structure is used for a loop-based application, in which many features are recognized by repeating an executable loop, such as all online social applications. The process flow is characterized by a graph that consists of a given cycle.

(d)Tree topology: This topology displays the tasks in a hierarchical manner and uses the tree for this structure. Primitive node is the module program or main function.

(e) Layer topology: Represents a topology based on a network of tasks.

The partitioning layout should be able to achieve a proper partitioning for loading the computing as local and remote execution. Different partitions can lead to various costs, and the total cost due to loading depends on several factors, including device operating system, networks, clouds, and workload. Therefore, this program may have different optimized partitions for diverse mobile environments and workloads.

2.4. constructing the graph combining weighted tasks

There are two types of cost for loading systems: 1- computing cost that is run on software tasks locally or remotely (including cost of memory, processing time, etc.), and 2- communicational costs for task interactions (in concern with the movement of data and necessary messages). Even this work can dictate a lot of cost on the mobile device and the cloud in terms of running time and energy consumption. Because cloud servers due to having a powerful configuration typically run faster than mobile devices, they can save energy and improve the performance when loading some parts of the computing to a remote server. However, when their vertices are assigned to different parties, the interaction between them leads to additional communicational costs. Therefore, we try to find the optimal assignment of vertices for graph partitioning and loading of computing through transaction of computational costs with communicational costs. Graph naming is widely used to describe the dependency of data in a computing, in which each vertex represents a task, and each of the edges represents the call relationship from the caller to the receiver.

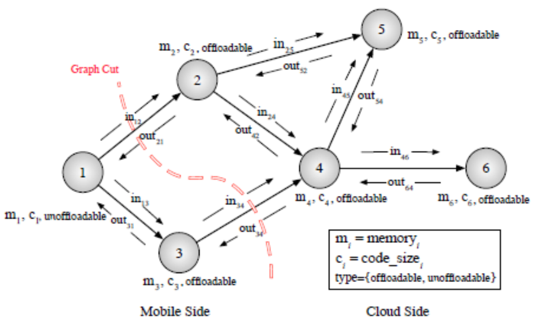


Figure 2. CG graph structure

Figure 2 shows a graph for combination of weighted tasks consisting of six tasks. Computational cost is shown by the vertices, while communication cost is expressed by the edge. The dependence of the tasks of a program and its associated cost has been determined as a distant graph. If the graph G = (V, E) is considered graph in which the vertices V = (v_1, v_2, ..., v_N) represent that the number of software tasks is N and the edge e (v_i, v_j) ∈ E indicate the number of requests and data access between the nodes V_i and V_j, in which the vertices V_i and V_j are neighbors.

The structure used in the proposed method called WCG is shown in Figure 3.

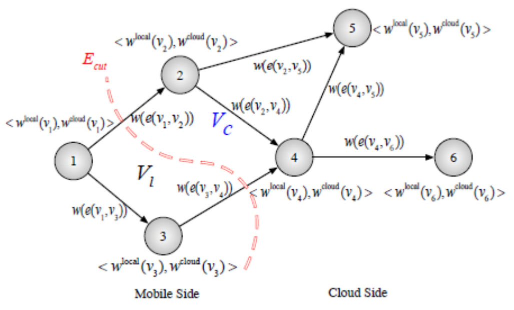

E_{cut}
V_C
V_l
$<w^{local}(v_1), w^{cloud}(v_1)>$
$<w^{local}(v_2), w^{cloud}(v_2)>$
$<w^{local}(v_3), w^{cloud}(v_3)>$
$<w^{local}(v_4), w^{cloud}(v_4)>$
$<w^{local}(v_5), w^{cloud}(v_5)>$
$<w^{local}(v_6), w^{cloud}(v_6)>$
$w(e(v_1,v_2))$
$w(e(v_1,v_3))$
$w(e(v_2,v_5))$
$w(e(v_2,v_4))$
$w(e(v_3,v_4))$
$w(e(v_4,v_5))$
$w(e(v_4,v_6))$
Mobile Side
Cloud Side

Figure 3. WCG Graph structure

Each vertex v ∈ V has weight containing two cost, which is represented by equation (1).

$$W(v) = < W^{Local}(v), W^{Cloud}(v) > \quad (1)$$

Where $W^{Local}(v)$ and $W^{Cloud}(v)$ represents the computational cost of executing task V locally on a mobile device and remotely on the cloud, respectively. The edge set E ⊂ V × V represents the communicational cost among tasks. The weight of an edge w (e (v_i, v_j)) is as follows:

$$w(e(v_i, v_j)) = \frac{in_{ij}}{B_{\text{upload}}} + \frac{out_{ij}}{B_{\text{download}}} \quad (2)$$

Which it includes the input and output communicational cost when the V_i and V_j tasks are executed on different sides, and it depends closely on the bandwidth of the network (upload of the bandwidth B, upload and download of the bandwidth) and transmitted data. A decision is made on a sample loading by a cut in WCG, which divides the vertices into two distinct sets; one represents the tasks performed on the mobile device and the other is loaded onto the remote server. Hence, considering the optimal loading decision is equivalent to the WCG partitioning so that cost, energy, and response time are minimized. Red dot –line in Figures 1 and 2 is a possible partitioning cut showing the division of the computational workload between the mobile device and the cloud.

V_L and V_C are sets of vertices in which V_L is a set of local tasks executed locally, and V_C is a collection of tasks for the cloud, i.e. those tasks loaded directly into the cloud. We have $V_L \cap V_C = \emptyset$ and $V_L \cup V_C = V$.

2.5. Partitioning Algorithm for Off-Line proposed loading

In this section, the GALP_OP algorithm is introduced for the WCG weighted graph having an arbitrary topology. The GALP_OP algorithm considers a WCG graph as an input in which each node represents the operation / computation of a program and their interrelationships are considered as the edges. Each node has two costs: the first one is the cost of doing the operation locally (for example, on a mobile phone), and the second one is the cost of doing it elsewhere (for example, in the cloud). The weight of the edge represents the ratio of communication cost to that of loading computing cost. It is assumed that the cost of communication between operations in the same place is negligible. The result includes information about costs and reports that should be done locally and then uploaded.

2.5.1. *The steps of the GALP_PO method*

The GALP_OP algorithm can be divided into two stages as follows.

2.5.2. *Merge*

Definition: If $s, t \in V (s \neq t)$, therefore, s and t can be merged into the following form:

1) Select the s and t nodes.

2) The s and t nodes are replaced by the new node $X_{s,t}$. Therefore, all edges were previously attached to s or t node are now connected to X node, except the edges between s and t nodes.

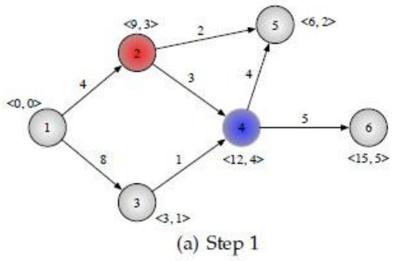
3) The output edges of s and t reaching a common node are integrated together and form an edge. The weight of the X node is equal to the sum of the weights of s and t nodes. Algorithm 1 shows the structure of the merge operation.

Figure 4 shows the merge process of two nodes in a graph.

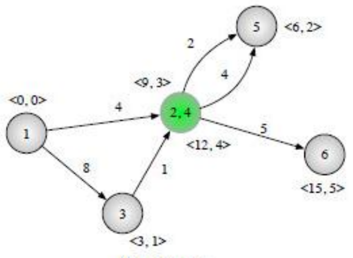
Algorithm 1 The *Merging* function

```
//This function takes $s$ and $t$ as vertices in the given graph and merges them into one
Function: $G' = Merge(G, w, s, t)$

Input:  $G$: the given graph, $G = (V, E)$
        $w$: the weights of edges and vertices
        $s, t$: two vertices in previous graph that are to be merged
Output: $G'$: the new graph after merging two vertices

1:  $x_{s,t} \Leftarrow s \cup t$
2:  for all nodes $v \in V$ do
3:      if $v \neq \{s, t\}$ then
4:          $w(e(x_{s,t}, v)) = w(e(s, v)) + w(e(t, v))$
5:          //adding weights of edges
6:          $[w^{local}(x_{s,t}), w^{cloud}(x_{s,t})] = [w^{local}(s) + w^{local}(t), w^{cloud}(s) + w^{cloud}(t)]$
7:          //adding weights of nodes
8:          $E \Leftarrow E \cup e(x_{s,t}, v)$    //adding edges
9:      end if
10:     $E' \Leftarrow E \backslash \{e(s, v), e(t, v)\}$    //deleting edges
11: end for
12: $V' \Leftarrow V \backslash \{s, t\} \cup x_{s,t}$
13: return $G' = (V', E')$
```

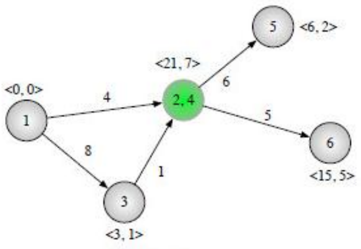
Alghorithm1- The structure of the merge operation



(a) Step 1



(b) Step 2



(c) Step 3

Figure 4. Merge of two nodes of a graph

2.5.3. *Process of Genetic Algorithm*

In this section, the steps of the proposed method are examined step-by-step and based on the genetic algorithm for partitioning the graph. The structure of the suggested method for selecting nodes that can be moved to the cloud space is as follows.

The first step: generating the initial population

The second step: measuring the value of the initial population using the fitness or adaptation function

The third step: Combining two sets having higher value

The fourth step: Random displacement of a node in the structure of partitioned graph

The fifth step: Choosing the best nodes as a new generation

The sixth step: Choosing the best chromosome as the best partitioning suggestion

2.5.3.1 Production of initial population

The table of nodes for WCG weighted graph in each column represents the node of the graph hold as Table 1.

Table 1. Structure view of the indicators' set

Node 1	Node 2	....	Node n

So, the genetic algorithm uses a bit array for the population. It means that each chromosome is composed of a number of zero and one values. The number of chromosome genes depends on the number of graph nodes. For example, if a dataset has 10 nodes, each chromosome will have 10 genes, which is zero or one. If the gene is one, the task associated with this node will be run as local and if it is zero, that is, the node-specific task will be run in the cloud. The selecting of the genes is done randomly. For example, if the nodes 2, 3, 4, 5, 7, and 10 are selected, the generated chromosome will be as shown in Table 2.

Table 2. An example for the structure of the generated chromosome

1	0	0	0	0	1	0	1	1	0

2.5.3.2 Valuation of the population

In each chromosome, there are a number of nodes having zero value and a number of them has the value of one. Between each zero node and one node, there is a cutting in the graph that means the edge is broken between the zero node and the one node and so divided into two parts. So, I_v is considered for being zero or one between each zero node and one node, I_E is considered as a cut in the edge. According to these definitions, three indicators of cost, energy, and response time are important in evaluation of each chromosome.

2.5.3.2.1 Cost

Partitioning of the mobile software to find the optimal partitioning solution will minimize running costs in order to get the best balance between response time, energy and transmission and delay costs. For the cost in the proposed structure, we follow Equation 3.

$$C_{\text{total}} = \sum_{v\in V} I_v \cdot w^{\text{local}}(v) + \sum_{v\in V}(1-I_v)\cdot w^{\text{cloud}}(v) + \sum_{e(v_i,v_j)\in E} I_e \cdot w(e(v_i,v_j))$$

Where, the total cost - the sum of computing costs (local and remote) and the reduced communication cost affects the margins. The nodes of cloud server and the mobile device belong to different partitions. The value of I_V in the above equation is calculated as follows.

$$I_v = \begin{cases} 1, & \text{if } v \in V_l \\ 0, & \text{if } v \in V_c \end{cases} \quad \text{and} \quad I_e = \begin{cases} 1, & \text{if } e \in E_{\text{cut}} \\ 0, & \text{if } e \notin E_{\text{cut}} \end{cases}$$

We are looking to find a good cut in the WCG by which some tasks of the program is running on the mobile side and the rest is running on the cloud. The optimum cut minimizes or maximizes the objective function while it eliminates the resource restriction of the mobile device. The objective function represents the overall purpose of a partition, and this, for example, may minimize the energy consumption and the amount of transferred data and its execution in less than an hour.

2.5.3.2.2 Response time

Communication costs depend on the size of data transmission and network bandwidth, while computational costs are affected by the time of computing. If the minimum response time is chosen as an objective function, we can calculate the total amount of spent time in loading as the following way.

$$T_{\text{total}}(\boldsymbol{I}) = \sum_{v\in V} I_v \cdot T_v^l + \sum_{v\in V}(1-I_v)\cdot T_v^c + \sum_{e\in E} I_e \cdot T_e^{tr} \qquad 5$$

Where: $T_v^l = F \times T_v^c$ is the computing time for task V on the mobile device when it runs locally.

F: The acceleration factor is the ratio of the implementation speed in the cloud server to that of in the mobile device. Because computing capacity of cloud infrastructure is stronger than the mobile device, it always has a value greater than one (F> 1).

T_v^c: is the computing time of task V in the off-line cloud server.
The time to load information into the cloud is also obtained by the following equation.

$$T_e^{tr} = \frac{D_e^{tr}}{\text{BW}} \qquad (6)$$

Which D_e^{tr} is the amount of transmitted or received data. BW is a wireless bandwidth to communicate with the cloud.
In this scheme, linear programming should provide a partitioned graph that minimizes the total response time. The response time stored in the partitioning scheme with regard to the layout without partitioning is calculated as follow:

$$T_{\text{save}}(\boldsymbol{I}) = \frac{T_{\text{local}} - T_{\text{total}}(\boldsymbol{I})}{T_{\text{local}}} \cdot 100\% \qquad (7)$$

2.5.3.2.3 Energy consumption

Through Equation 7, the total amount of energy consumed by the mobile device due to loading is calculated.

$$E_{\text{total}}(I) = \sum_{v \in V} I_v \cdot E_v^l + \sum_{v \in V} (1 - I_v) \cdot E_v^i + \sum_{e \in E} I_e \cdot E_e^{tr} \quad (8)$$

In which:

$E_v^l = P_m \times T_i$: determines the energy consumption of work V on a mobile device when it runs locally.

$E_v^i = P_i \times T_v^c$: determines energy consumption of work V on a mobile device when it is loaded in the cloud.

$E_e^{tr} = P_{tr} \times T_v^{tr}$: is the consumed energy used to connect the mobile device and the cloud.

P_i, P_m and P_{tr} are mobile energy in idle / rest, computing / implementation and sending and receiving the information states, respectively. Similarly, if the minimum energy consumption is selected as the objective function, we can calculate the total energy consumed by the mobile device due to loading as follows:

$$E_{\text{save}}(I) = \frac{E_{\text{local}} - E_{\text{total}}(I)}{E_{\text{local}}} \cdot 100\% \quad (9)$$

2.5.3.2.4 Minimum sum of weight of time, cost and energy using linear programming

If the response time, cost and energy consumption are combined, we can consider the objective function of linear programming as the following model for partitioning.

$$Score(I) = w_1 \times C_{Total}(I) + w_r \times R_{Totla}(I) + w_r \times E_{Total}(I) \quad (10)$$

Finally, the objective function is written with the linear programming structure as follows.

Minimize :

$$Score(I) = w_1 \times C(I) + w_r \times R(I) + w_r \times E(I)$$

Subject :

$$C_{\text{total}} = \sum_{v \in V} I_v \cdot w^{\text{local}}(v) + \sum_{v \in V} (1 - I_v) \cdot w^{\text{cloud}}(v) + \sum_{e(v_i, v_j) \in E} I_e \cdot w(e(v_i, v_j)) \quad 11$$

$$T_{\text{total}}(I) = \sum_{v \in V} I_v \cdot T_v^l + \sum_{v \in V} (1 - I_v) \cdot T_v^c + \sum_{e \in E} I_e \cdot T_e^{tr}$$

$$E_{\text{total}}(I) = \sum_{v \in V} I_v \cdot E_v^l + \sum_{v \in V} (1 - I_v) \cdot E_v^i + \sum_{e \in E} I_e \cdot E_e^{tr}$$

2.5.3.3 Cross-Over Operator

In the proposed method, the cross-over operator uses the random two-point method. In this way, the random number t is generated. t is considered as the first point and Length-T is considered as the second point. Then, according to the rule in Fig. 5, the parental genes are combined and they produce new offspring or chromosomes.

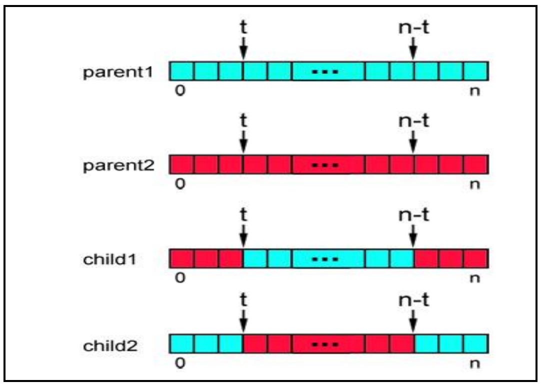


Figure 5. Describing how the cross-over operator functions

Since the chromosome coding in the problem is priority-based and based on random numbers, the one-cutting-point operator cannot be used alone because it is possible the repetition occurs in the generation of newborns in the chromosomes. To avoid this, we use an operator called weighted-written operator, which is actually a modified version of one-cutting-point operator for particular states. In this operator based on the one-cutting-point operator, a point is randomly considered over the two chromosomes chosen as the parent ones, and the chromosomes are split into two parts from that point.

2.5.3.3.4 Mutation Operator

In the actual encoding, the mutation operator leads to random generation of a new value in a particular position on the chromosome. As a result of these random changes in the chromosome population, more regions of the exploration space are investigated and untimely convergence (local abruptness) of the algorithm is prevented. An example of a true mutation operator is the random or uniform mutation. Assuming that C = (C_1, C_2, …, C_n) is a chromosome, and C_i is a gene subjected to mutation, then a new random value in the range of the desired gene will be replaced instead of C_i gene in the new chromosome.

In the proposed method, the mutation operator is performed on A % of the new generation. In this algorithm, the mutation is random, too, so that ‘t” number is randomly generated. In this regard, t is considered as the first point and n-t is considered as the second point and the related gene is replaced. Figure 6 shows how the mutation operator functions.

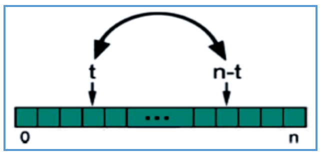


Figure 6. Description for the way of mutation operator’s functioning

After taking mentioned steps, the new generation with higher value (less error rate) is used for repetition and generating the next generation. In this method, the condition for completing the genetic algorithm is that the number of generations should be reach to a given value.

2.5.3.5 Selecting the next generation

Tournament method has been used for selection. In this method, the best chromosomes having higher value are chosen for the next generation.

2.5.3.6 Selecting the best chromosome

The best chromosome is selected as the best structure for partitioning the graph.

3. RESULTS

In this section, the results of implementing the proposed method for partitioning the application graph are examined. Matlab software has been used in order to simulate the proposed method, and we have used the features and tools of this software.

3.1. Simulation parameters

Table 3 shows the characteristics of test environment. The parameters of the Genetic Algorithm are proposed in Table 4.

Table 3. Host properties

Processor type	Main memory	operating system
Intel Core I7 3.2 GHZ	8 GB	Windows 10

Table 4. Parameters of the Genetic Algorithm

Termination condition	mutation coefficient	Selection operator	Number of generations	Initial population
Number of generations	0.1	Tournament	100	100

Table 5 shows the energy consumption of mobile device in three modes of sleep, ttransition and processing.
Table 6 shows the cost of the three modes of running on mobile, ttransition and running on the cloud.

Table 5. Energy consumption

Processing mode	Transition mode	Sleep mode
7 j	3 j	1 j

Table 6. Cost

Run on mobile	Transition mode	Run on the cloud
0.02 $	0.5 $	1 $

3.2. Evaluation Indicators

In each simulation, three factors have been investigated include: response time, cost, and energy consumption.

Response time: The actual response time is the exact time difference between the time of requesting of the job and the delivery time of the conducted work to the user.

Average energy consumption: Average energy consumption is the total amount of energy consumed over to the number of tasks.

Cost: The cost is sum of cost for each task plus the cost of data transfering.

3.3. The first test sample graph

In the evaluation, the tested graph has 15 tasks. The graph storage structure is as follows (figure 7). In this structure, a 3D matrix is used for the graph. Figure 7 shows the graph used in the first experiment.

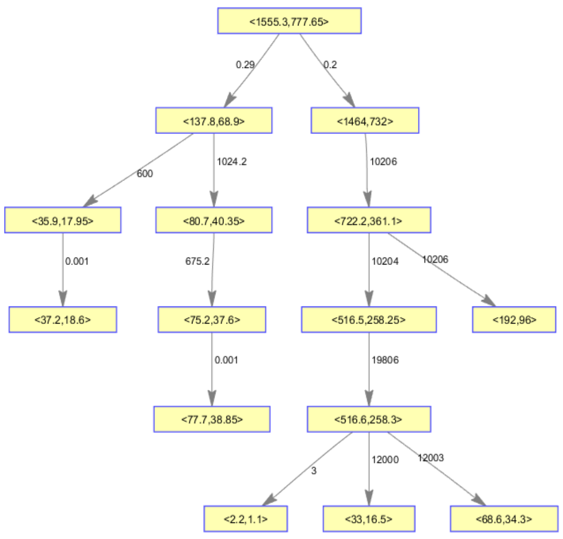


Figure 7. The first test graph

The first, the second and the third dimensions are the input nodes, the destination nodes and the weight of the edge between this two nodes, respectively. The structure of this matrix is as follows.

[1 1 2 2 3 4 5 6 6 8 9 12 12 12]

[2 3 4 5 6 7 8 9 10 11 12 13 14 15]

[0,29 0,2 600 1024,2 10206 0,001 675,2 10204 10206 0,001 19806 3 12000 12003]

The second matrix is also used to hold the weight entered in each node as the runtime of the task in the cloud.

[1555,30 137,80 1464,00 35,90 80,70 722,20 37,20 75,20 516,50 192,00 77,70 516,60 2,20 33,0 68,60]

3.4. First experiment

In this experiment, the bandwidth is 3 Mbps and the amount of increase in the speed of processing in the cloud were considered 2 to 10 times more than local processing. The graph of Figure 7 is used as a test graph. At first, the function of the genetic algorithm is checked. For this reason, the values of the fitness function are checked over 100 generations. The more the value of the fitness function approaches to zero, the more favorable performance of the genetic algorithm is achieved. In this simulation, the mean value of the fitness function is 0.00154 and the best or lowest value is 0.00130. In Fig. 8, the black points represent the best value of the fitness function and the blue points show the mean value of the whole generation.

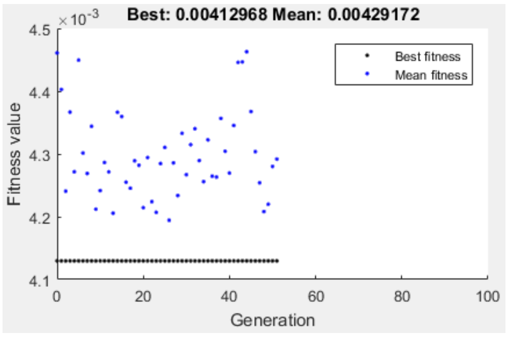


Figure 8. The values of the fitness function in the graph partitioning

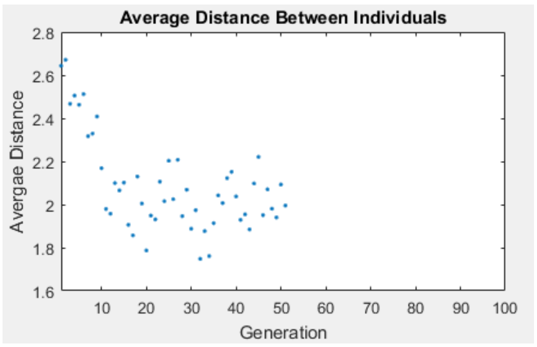


Figure 9. The sum of distance between each generation of the population in the partitioning of the graph

Figure 9 shows the total distance between each generation of the population. The more this distance be less, the more convergence of responses will be better, and if the distance between generations is zero, the algorithm ends. In Figure 10, the best, worst, and average scores for each generation are displayed. The more each member of the generations has a higher value, the more effect will have in next generation since the Tournament selector operator has been used. Figure 11 represents the selected nodes to run on the cloud. Figure 12 and Figure 13 show the final graph resulted from the partitioning by proposed method and reference method, respectively.

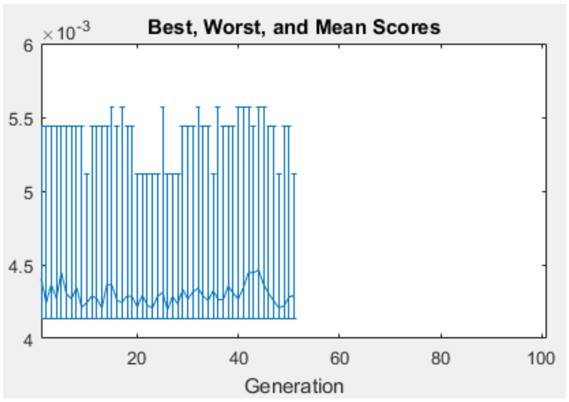


Figure 10. The best, the worst, and the average scores achieved for each generation

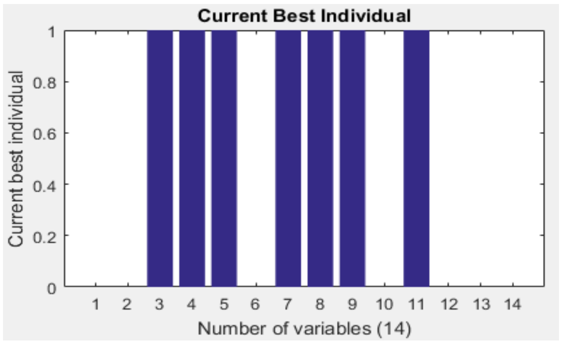


Figure 11. Selected nodes to run on the cloud

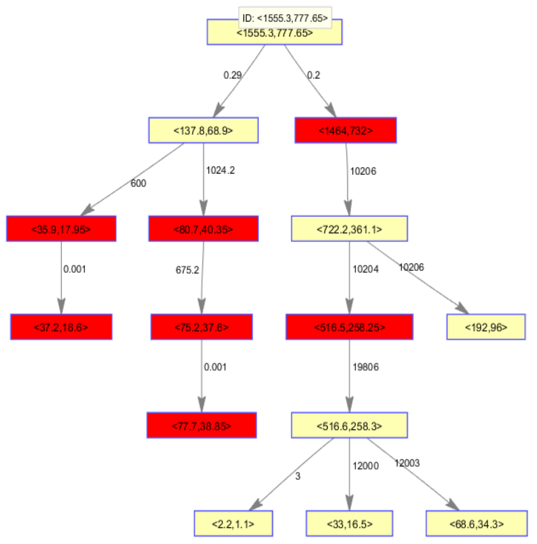


Figure 12. The final graph resulted from the partitioning by proposed method

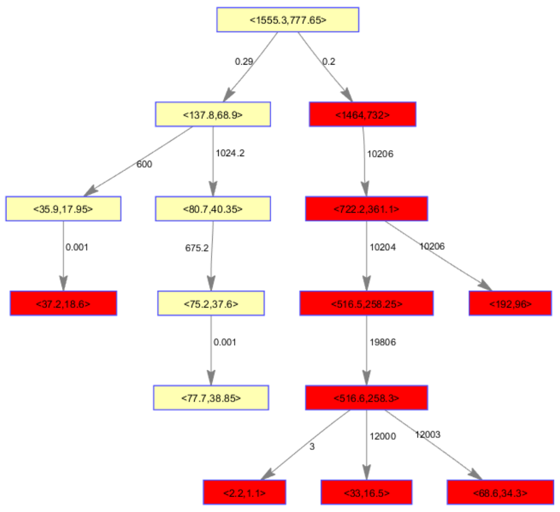


Figure 13. The final graph resulted from the partitioning by the reference method

In Fig. 14, the yellow nodes are performed locally, and the red nodes are performed as offload and in the cloud. Figure 14 shows the response time in three partitioning states including the proposed method, completely local method and completely in the cloud method.

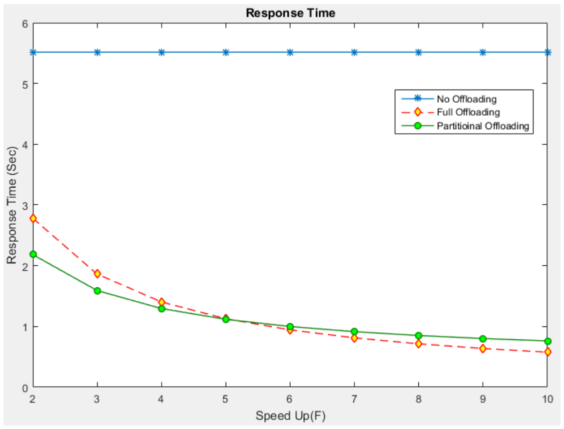


Figure 14. Response time by changing the Speed Up parameter

In Figure 14, because of the proper function of the genetic algorithm, the partitioning response time is lower for choosing the node or tasks for running in the cloud. Specifically, the time for executing tasks as locally does not change by making change in the cloud speed, so the local response time is constant. As the cloud speed increases, the response time by the cloud is reduced. But the challenge that should be addressed is that whether the amount of consumed energy is reduced too or not.

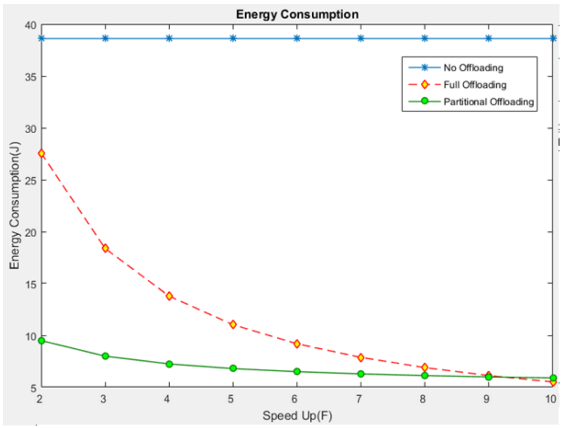


Figure 15. Consumption of energy by changing the Speed Up parameter

Figure 15 shows the amount of energy consumption in three partitioned modes including the proposed method, completely local method and completely in the cloud method. Due to selecting suitable nodes for run in the cloud environment and the holding the appropriate nodes in the local system by the genetic algorithm selector, the energy consumption is lower. Specifically, executing the tasks locally does not change with the change in cloud speed, so the amount of local consumed energy is constant.

3.5. Second experiment (comparing the results of the graph)

In this experiment, the bandwidth is 3 Mbps and the amount of increase in the speed of processing in the cloud were considered 2 to 10 times more than local processing. The graph of Figure 7 is used as the test graph. The response time of the proposed method and reference paper was compared in partitioning mode. Figure 16 shows the response time in the proposed method the reference paper.

In Figure 16, due to the proper function of the genetic algorithm for choosing the node or tasks for run in the cloud, the partitioning response time is lower than the reference method.

Figure 17 shows the energy consumption in the proposed method and reference paper. It is worth to note that the reference method emphasizes on energy and cost. According to Figure 17, it is clear that this two methods do not differ significantly in terms of energy consumption, but because of the proper function of the genetic algorithm, the choice of nodes or suitable tasks for run in the cloud, consumed energy by proposed method is less than the reference method.

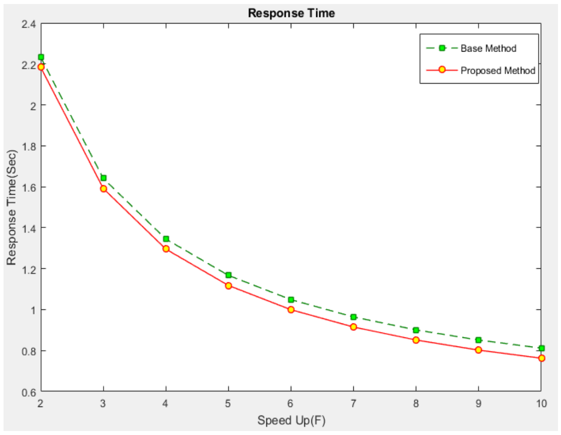


Figure 16. Comparison of response time with change in Speed Up parameter in proposed and reference paper

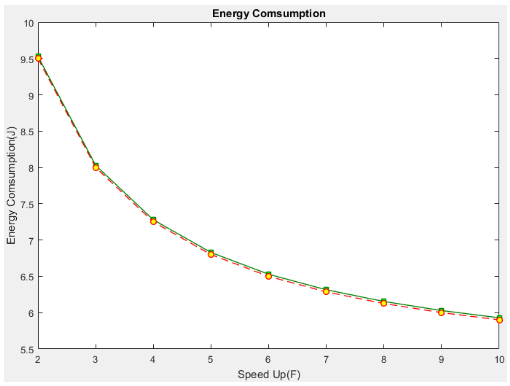


Figure 17. Comparison of consumed energy by changing the Speed Up parameter in the proposed method and reference paper

3.6. The second test sample graph

The test graph in the next evaluations has 29 tasks. Figure 18 shows the graph used in the first experiment.

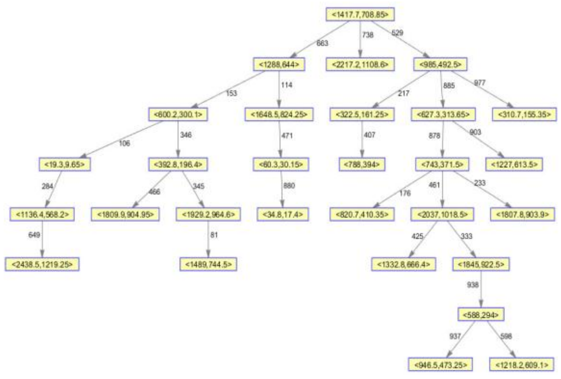


Figure 18. Second test graph

3.7. Third experiment

In this experiment, the bandwidth is 3 Mbps and the amount of increase in the speed of processing in the cloud were considered 2 to 10 times more than local processing. The graph of

Figure 18 is used as a test graph. At first, the function of the genetic algorithm is checked.

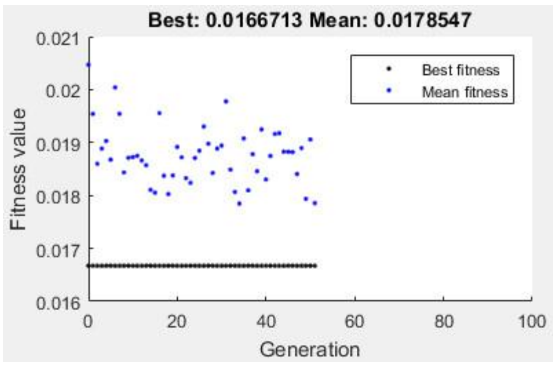


Figure 19. The values of the fitness function in the graph partitioning

For this reason, the values of the fitness function are checked over 100 generations. The more the value of the fitness function approaches to zero, the more favorable performance of the genetic algorithm is achieved. In this simulation, the mean value of the fitness function is 0.00154 and the best or lowest value is 0.00130. In Fig. 19, the black points represent the best value of the fitness function and the blue points show the mean value of the whole generation.

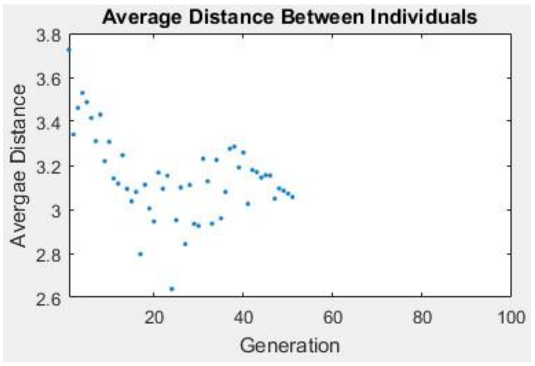


Figure 20. The total distance between generations of the population in the graph partitioning

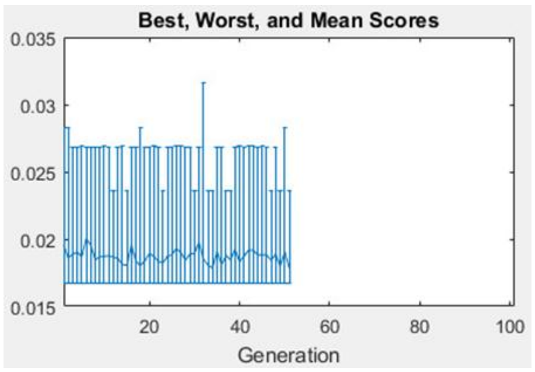


Figure 21. The Best, Worst, and Average scores achieved for each generation

Figure 20 shows the total distance between each generation of the population. The more this distance be less, the more convergence of responses will be better, and if the distance between generations is zero, the algorithm ends. In Figure 21, the best, worst, and average scores for each generation are displayed. The more each member of the generations has a higher value, the more effect will have in next generation since the Tournament selector operator has been used. Figure 22 and Figure 23 show the final graph resulted from the partitioning by proposed method and reference method, respectively.

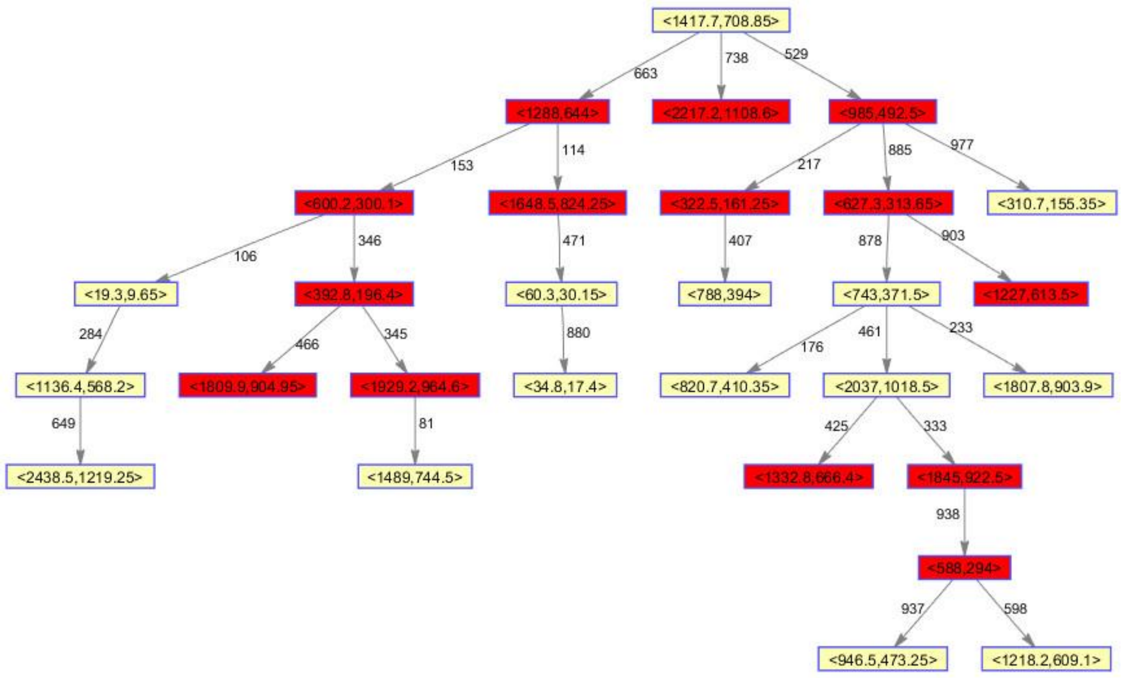


Figure 22. The final graph resulted from partitioning by the proposed method

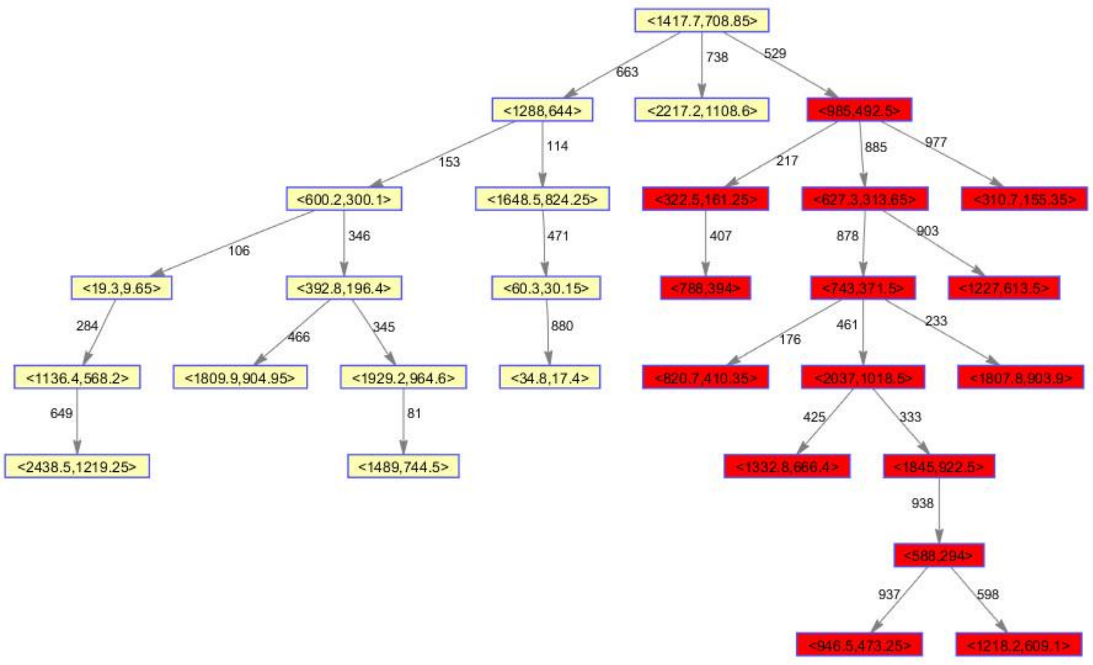


Figure 23. The final graph resulted from the partitioning of the reference method

In Fig. 23, yellow nodes are performed locally, and red nodes are performed in an offload and in the cloud.

Figure 24 shows the response time in three partitioning states including the proposed method, completely local method and completely in the cloud method.

In Figure 24, due to the proper function of the genetic algorithm, the partitioning response time for choosing the node or tasks for execution in the cloud is lower. Specifically, the time for executing tasks locally does not change with the change in cloud speed, which is why the local response time is constant. As the cloud speed increases, the response time by the cloud is reduced. But the challenge that should be addressed is that whether the amount of consumed energy is reduced too or not.

Figure 25 shows the energy consumption in the three partitioned modes by the proposed method, completely local method and completely in the cloud method.

In Figure 24, due to the selecting suitable nodes for run in the cloud environment and the holding the appropriate nodes in the local system by the genetic algorithm selector, the energy consumption is lower. Specifically, executing tasks locally does not change with the change in cloud speed, which is why the amount of local energy consumption is constant.

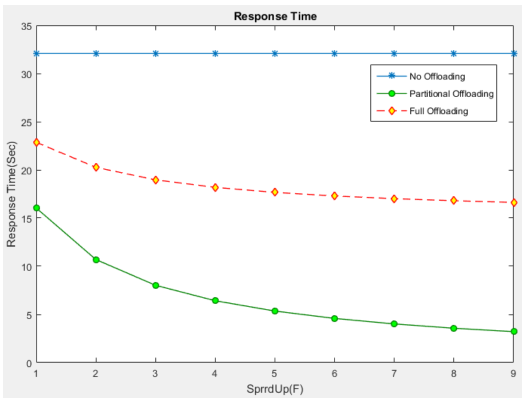


Figure 24. Response time by changing the Speed Up parameter

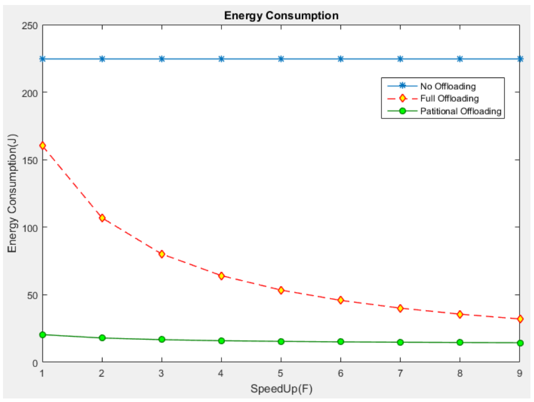


Figure 25. Energy consumption by changing the Speed Up parameter

3.8. Fourth Test (Comparing the results of Graph 18 with the reference paper)

In this experiment, the bandwidth is 3 Mbps and the amount of increase in the speed of processing in the cloud were considered 2 to 10 times more than local processing. The graph of Figure 18 is used as the test graph. The response time of the proposed method and reference paper was compared in partitioning mode. Figure 26 shows the response time in the proposed method the reference paper.

In Figure 26, due to the proper function of the genetic algorithm for choosing the node or tasks for run in the cloud, the partitioning response time is lower than the reference method.

Figure 27 shows the energy consumption in the proposed method and reference paper. It is worth to note that the reference method emphasizes on energy and cost. According to Figure 27, it is clear that this two methods do not differ significantly in terms of energy consumption, but because of the proper function of the genetic algorithm for the choice of nodes or suitable tasks for run in the cloud, consumed energy by proposed method is less than the reference method.

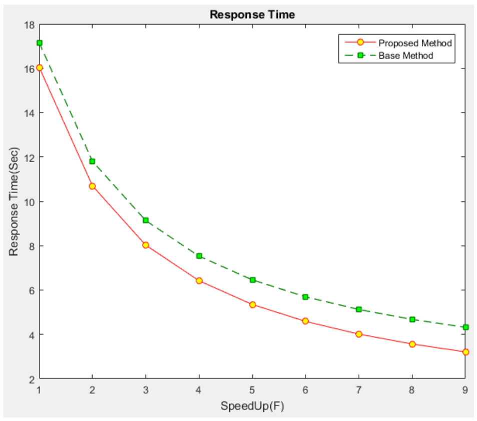


Figure 26. Comparison of response time with change in Speed Up parameter in proposed and reference methods

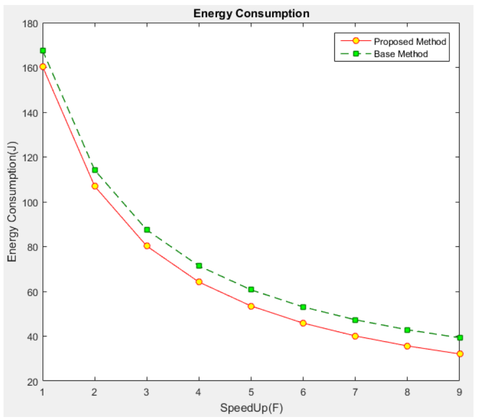


Figure 27. Comparison of energy consumption by changing the Speed Up parameter in the proposed method and reference methods

In this section, the results of the proposed method were evaluated. Two experiments were conducted with changing the speed of the cloud respect to the local system. Comparing the proposed method with the reference represented that due to the use of the genetic algorithm and the effect of the valuation function based on three factors-cost, the response time and energy- the performance of the proposed method is better than the reference method. Specifically, the proposed method reduces the response time and consumed energy.

4. CONCLUSION

Mobile cloud computing is a combination of cloud computing, mobile computing and wireless networks that provide rich computing resources for mobile users, network operators, and cloud computing providers. The main purpose of mobile cloud computing is to run powerful mobile applications on a large number of mobile devices.

In this paper, a method was presented for partitioning the task graph of an application. The purpose of this work was to optimum portioning of tasks for local implementation and implementing on the cloud platform. Since the problem of large-scale graph partitioning is of the NP-hardness, the genetic algorithm was proposed as a selective structure in the proposed method. In the genetic algorithm, three criteria including cost, response time, and energy were used as a combined target. Use of linear programming effectively influences the optimal performance of the genetic algorithm. The results of the proposed method show accurate reduction in energy consumption and the response time by 0.5% and 3%, respectively.

5. SUGGESTIONS

Use of other meta-heuristic algorithms such as particle swarm and etc.

The effect of other parameters for quality service (QOS) such as reliability and availability in the fitness function.

6. REFERENCES

1. Elham Shamsi Nejad and Asadollah Shah Ebrahimi and Alireza Hedayati, Resource Allocation in Cloud Computing through Valuation of Services, the National Conference on Commercialization in Sari, 2013.
2. Farzaneh Motazaleh Hagh and Faramarz Safi, Proposing an Efficient Algorithm for Resource Allocation Based on Service Level Agreement in the Cloud Computing System, the 5th National Conference on Electrical and Electronic Engineering, Iran, 2013.
3. M Seyyedi, P. Mohammad Poor, and. Morgan, The Principles Governing Cloud Computing and Service- Based Architecture and Their Relationship at Architecture Level, the First National Workshop on Cloud Computing, Amirkabir University of Technology, Tehran, 2012.
4. Ebrahimi, Farzaneh; Ahmad Farahi and Akbar Farhudinejad, 2012; Review of resource allocation and its importance in cloud computing environment, the First National Conference on Information Technology and Computer Networks, Payame Noor University, Tabas, 2012.
5. Elahe Dost Sadiq and Reza Asemi, A Review of Timing Tasks in Cloud Computing, the First National Conference on Modern Approaches in Computer Engineering and Data Recovery, Rudsar, 2013.
6. Ebrahim Behroziannezhad and Rezvan Ali Pourbeshvari, A Review of Resource Allocation Methods in Cloud Computing with the Aim of Reducing Time, National Conference on Computer Engineering and Information Technology, 2013.
7. Faramarz Safi and Hamidreza Sadrarhami, A Review of Several Scheduling Algorithms in Cloud Computing, National Conference on Computer Engineering and Information Technology Management, Tehran, Tolo Farzin Science and Technology Co, 2014.
8. Mell P, Grance T. The NIST Definition of Cloud Computing (Draft) U.S. Department of Commerce,National Institute of Standards and Technology; 2011.
9. Abolfazli S, Sanaei Z, Ahmed E, Gani A, Buyya R. Cloud-based augmentation for mobile devices: Motivation, taxonomies, and open challenges. IEEE Communications Surveys & Tutorials. 2013;
10. Sanaei Z, Abolfazli S, Gani A, Buyya RK. Heterogeneity in Mobile Cloud Computing: Taxonomy and Open Challenges. IEEE Communications Surveys & Tutorials. 2013; InPress
11. Sanaei Z, Abolfazli S, Gani A, Khokhar RH. Tripod of Requirements in Horizontal Heterogeneous Mobile Cloud Computing. In: Proc. CISCO; 2012.
12. Alizadeh M, Hassan WH. Challenges and opportunities of Mobile Cloud Computing. In: 9th Interna-tional Conference in Wireless Communications and Mobile Computing Conference (IWCMC); 2013. p.660–666
13. Shiraz M, Gani A, Khokhar RH, Buyya R. A Review on Distributed Application Processing Frameworks in Smart Mobile Devices for Mobile Cloud Computing. IEEE Communications Surveys & Tutorials. 2012November;15(3):1294 – 1313
14. Chun BG, Ihm S, Maniatis P, Naik M, Patti A. CloneCloud: Elastic execution between

mobile device and cloud. In: Proc.EuroSys. ACM; 2011. p. 301–314.

15. J. Shneidman, C. Ng, D. C. Parkes, A. AuYoung, A. C. Snoeren, A. Vahdat, and B. N. Chun, "Why Markets Could (But Don't Currently) Solve Resource Allocation Problems in Systems," in HotOS, 2005.
16. D. C. Marinescu, "Chapter 6 – Cloud Resource Management and Scheduling," Cloud Computing Theory and Practice, pp. 163-203, 2013.
17.