International Journal of Grid and Distributed Computing
Vol. 9, No. 6 (2016), pp.83-92
http://dx.doi.org/10.14257/ijgdc.2016.9.6.09

Application Multi-partitioning for Offloading Computation to Multiple Computing Resources around Mobile Terminals

Wenhao Fan[1, 2], Bihua Tang[1, 2], Yuan'an Liu[1, 2]
[1]*School of Electronic Engineering, Beijing University of Posts and Telecommunications, Beijing 100876, China*
[2]*Beijing Key Laboratory of Work Safety Intelligent Monitoring, Beijing University of Posts and Telecommunications, Beijing 100876, China*
whfan@bupt.edu.cn

Abstract

Different with traditional approaches that offload computation to a single remote server, the performance of an application can be further enhanced by simultaneously distributing its computing tasks to multiple computing resources around the mobile terminal. How to effectively multiply partition application components is critical for parallel computation offloading. In this paper, an application multi-partitioning scheme is proposed, which optimally offloads application's components to surrounding resources. A graph mapping model is converted and set up to represent components and resources as undirected graphs, and the A algorithm is employed to efficiently search the optimal mapping from component graph to resource graph, which minimizes computing costs and inter-resource communication costs. Simulation results demonstrate that the performance can be efficiently promoted by our algorithm, which outperforms the traditional approaches to a large degree.*

Keywords*: computation offloading, multi-partitioning, mobile terminals, heterogeneous networks, graph mapping, combinatorial optimization*

1. Introduction

Mobilization is an important feature in the development of information technologies. Mobile terminals nowadays need to host diverse computation-occupying and energy-consuming applications, whereas, as embedded systems, they are not able to provide sufficient performance to cope with various requirements of these applications. Computation offloading, which migrates computing tasks from resource-constrained mobile terminals to resourceful computing resource(s), is a promising approach to alleviate performance degradation caused by resource limitations of terminals, such as computing capabilities, storage and battery capacities, etc.

Continuous efforts have been done on computation offloading in recent years [1][2]. Traditional approaches address the problems of offloading computation tasks to a single remote server [3][4][5] or serval remote servers [6][7][8]. The assumptions in above research neglect the potential facts that multiple computing resources in environments where mobile terminals locate can be exploited, and parallelism can be used to further promote performance of applications by offloading tasks to these resources simultaneously.

In this paper, we consider a scenario where multiple computing resources around a mobile terminal, such as laptops, PCs, tablets, wireless routers, air conditioners, printers, TVs, base stations, etc., are employed for computation offloading in parallel. These resources are connected with the mobile terminal via multiple heterogeneous networks, for example, mobile networks, WiFi, Bluetooth, Zigbee, etc. When employing computation offloading technologies in such scenario, a fundamental problem is how to optimally partition the components of an application installed in a mobile terminal into multiple clusters, and offload them to the terminal's surrounding resources. The components represent the computation tasks of the application from the view of program structures. Here, we propose an application multi-partitioning scheme which models the application's components, and the network consisting of the terminal and resources as undirected graphs, respectively, and searches the optimal mapping from the component graph to resource graph, which minimizes computing costs and inter-resource communication costs via the A* algorithm.

The rest of this paper is organized as follows. Section 2 presents the graph mapping model, where the component graph and resource graph are defined. The application multi-partitioning scheme is described in Section 3, including how to solve the problem of the cost function derived from our model via the A* algorithm. Section 4 shows the simulation results and analysis of our scheme. Our work is concluded in Section 5.

2. Graph Mapping Model

The program of an application can be expressed as an undirected graph [9], called component graph. The vertices of the graph denote the application's components, which describes the application's composition in different granularity, such as classes, objects, modules, interfaces, functions and threads. These vertices are divided into two categories: offloadable vertices and unoffloadable vertices. The former represents the components that can run either locally or externally, whereas, the latter denotes the components that can only run in the terminal, such as the components which operate the terminal's hardware or is in charge of user interactions, etc. The weight of a vertex is the amount of computation generated by the corresponding component. An edge between two vertices represents the communication between the two corresponding components. The weight of an edge is the amount of data required to be transmitted in the inter-component communication if the two components are located differently in computation offloading because data transmission between the two computing resources or the terminal and corresponding resource is needed.

Let $G^{(c)} = (\mathbf{V}^{(c)}, \mathbf{E}^{(c)}, \mathbf{X}^{(c)}, \mathbf{Y}^{(c)}, H^{(c)})$ be the component graph of an application with $= m$ vertices and n edges. The vertex set is $\mathbf{V}^{(c)} = \{v_1^{(c)}, \dots, v_m^{(c)}\}$, where $\mathbf{V}_u^{(c)} \subseteq \mathbf{V}^{(c)}$ is the subset containing all unoffloadable components, and $\mathbf{V}_o^{(c)} \subseteq \mathbf{V}^{(c)}$ contains all offloadable ones. The edge set is $\mathbf{E}^{(c)} = \{e_1^{(c)}, \dots, e_m^{(c)}\}$. The weight sets of them are expressed as $\mathbf{X}^{(c)} = \{x_1^{(c)}, \dots, x_m^{(c)}\}$ and $\mathbf{Y}^{(c)} = \{y_1^{(c)}, \dots, y_m^{(c)}\}$, respectively. We define the triangular matrix $H^{(c)}$ with m rows and n columns to further denote $\mathbf{Y}^{(c)}$, where h_{kl} represents the weight of the edge between $v_k^{(c)}$ and $v_l^{(c)}$, and $h_{kl} = 0$ when $k = l$. Hence, the number of non-zero elements in $H^{(c)}$ is equal to n.

The mobile terminal and its surrounding computing resources form a star network where resources are connected to the terminal via heterogeneous networks. The mobile terminal is the center of the star network, which means that the communications between resources need to be transferred by the terminal, here, an assumption that resources belongs to different networks is based, so they cannot communicate with each other directly. The assumption is rational in the mobile terminal-centric ambient intelligence [10]. The star network can be also represented as an undirected graph, called resource graph, where the resources are denoted by vertices, and network connections between the terminal and resources are denoted by edges. The weight of a vertex is the processing speed of the corresponding resource, and the weight of an edge is the data transmission speed of the connection. The weights of vertices or edges are different due to the diversities on computing capabilities of different resources and data transmission capabilities of different networks.

Let $G^{(r)} = (\mathbf{V}^{(r)}, \mathbf{E}^{(r)}, \mathbf{X}^{(r)}, \mathbf{Y}^{(r)})$ be the resource graph of the star network with p vertices and $p-1$ edges. The vertex set is $\mathbf{V}^{(r)} = \{v_1^{(r)}, \dots, v_p^{(r)}\}$, where $v_1^{(r)}$ denotes the mobile terminal fixedly. The edge set is $\mathbf{E}^{(r)} = \{e_2^{(r)}, \dots, e_p^{(r)}\}$. The weight sets of them are defined as $\mathbf{X}^{(r)} = \{x_1^{(r)}, \dots, x_p^{(r)}\}$ and $\mathbf{Y}^{(r)} = \{y_1^{(r)}, \dots, y_p^{(r)}\}$, respectively.

In order to measure the effect of computation offloading, the cost of a resource is introduced, which is a linear combination of the processing time of resources and the transmission time between the terminal and resources. For a certain resource $v_i^{(r)} \neq v_1^{(r)}$, its cost is $C_i = C_i^{(s)} + C_i^{(t)} = \left(\sum_{x_j^{(c)} \in \mathbf{X}_i^{(c)}} x_j^{(c)} / x_i^{(r)} + \sum_{y_j^{(c)} \in \mathbf{Y}_i^{(c)}} y_j^{(c)} / y_i^{(r)}\right)$, where $C_i^{(s)}$ is the processing time of $v_i^{(r)}$, $C_i^{(t)}$ is the transmission time between the terminal and $v_i^{(r)}$, $\mathbf{X}_i^{(c)} \subseteq \mathbf{X}^{(c)}$ is the weight set of the vertices representing the components offloaded to $v_i^{(r)}$, and $\mathbf{Y}_i^{(c)} \subseteq \mathbf{Y}^{(c)}$ is the weight set of the edges representing the connections between the offloaded components and the terminal. $C_i^{(s)}$ can be obtained by summing all weights in $\mathbf{X}_i^{(c)}$ and dividing it by $x_i^{(r)} \subseteq \mathbf{X}^{(r)}$, since the sum of all weights in $\mathbf{X}_i^{(c)}$ is the total amount of computation hosted by $v_i^{(r)}$, and $x_1^{(r)}$ is its processing speed. $C_i^{(t)}$ can be obtained by summing all weights in $\mathbf{Y}_i^{(c)}$ and dividing it by $y_1^{(r)} \subseteq \mathbf{Y}^{(r)}$, since the sum of all weights in $\mathbf{Y}_i^{(c)}$ is the total amount of data required to be transmitted from or to $v_i^{(r)}$, and $y_i^{(r)}$ is the data transmission speed of the network it affiliates with. For the resource $v_1^{(r)}$, only $C_i^{(s)}$ is included in its cost C_1.

Thus, the cost of the application can be formulated as

$$C = \max_{1 \le i \le p} C_i \tag{1}$$

where it can be found that C is actually the total time consumed by the terminal and its surrounding computing resources to complete the computation of the whole application in parallel.

The application multi-partitioning problem can be converted into the graph mapping problem. A mapping from the component graph to the resource graph is defined as $Z: \mathbf{V}^{(c)} \rightarrow \mathbf{V}^{(r)}$, which satisfies that: 1) All vertices in $\mathbf{V}_u^{(c)}$ are mapped to $v_1^{(r)}$ fixedly since they are unoffloadable components that can only be processed in the terminal; 2) A vertex in $\mathbf{V}_o^{(c)}$ can be only mapped to a unique vertex in $\mathbf{V}^{(r)}$ because the duplication of components are not allowed in the system. A matrix D with p rows and m columns is defined to describe the result of a mapping, where the value of an element at the ith row and jth column is

$$d_{ij} = \begin{cases} 1, & if\ v_j^{(c)}\ is\ mapped\ to\ v_i^{(r)} \\ 0, & otherwise \end{cases} \tag{2}$$

By Formula (2), the cost of a certain resource $v_i^{(r)}$ can be rewritten as

$$C_i = \begin{cases} C_i^{(s)} + C_i^{(t)}, & i \neq 1 \\ C_i^{(s)}, & i = 1 \end{cases} \tag{3}$$

$$= \begin{cases} \frac{\sum_{j=1}^{m}\left(x_j^{(c)} d_{ij}\right)}{x_i^{(r)}} + \frac{\sum_{k=1}^{m}\sum_{l=1}^{m}(h_{kl}|d_{ik} - d_{il}|)}{y_i^{(r)}}, & i \neq 1 \\ \frac{\sum_{j=1}^{m}\left(x_j^{(c)} d_{ij}\right)}{x_i^{(r)}}, & i = 1 \end{cases}$$

Note that, when $d_{ik} = d_{il} = 0$, it means the two components are not mapped to $v_i^{(r)}$, so the transmission time between them is irrelevant with $v_i^{(r)}$ when $d_{ik} = d_{il} = 1$, it means both of the two components are mapped to $v_i^{(r)}$, whereas, the transmission time the transmission time is omitted since communication time among components inside a resource or the terminal is very short; when $d_{ik} = 1$, $d_{il} = 0$ or $d_{ik} = 0$, $d_{il} = 1$, it means one of the two components is mapped to $v_i^{(r)}$, hence, the transmission time between them is considered.

3. Application Multi-partitioning Scheme

3.1. Optimization Problem

The aim of our application multi-partitioning scheme is to find an optimal mapping which minimize the application's cost, thus we have the optimization problem, which is shown below:

$$\min_D C = \min_D\left(\max_{1 \le i \le p} C_i\right) \tag{4}$$

Formula (4) belongs to the combinatorial optimization problem, and is NP-hard. In fact, it is very complicated to solve the problem directly. Enormous expenditures need be paid to find out the optimal solution $D^{(*)}$ because a large number of feasible solutions need to be traversed and compared. Therefore, algorithms are required to cope with the problem with high efficiency.

3.2. A* Algorithm

The A* search algorithm, which is well-known in artificial intelligence and widely adopted in pathfinding of graph, is employed to solve the above problem. A* uses a best-first search and finds a least-cost path from a given initial node to one goal node (out of one or more possible goals). As A* traverses the graph, it follows a path of the lowest expected total cost or distance, keeping a sorted priority queue of alternate path segments along the way.

3.2.1. Converting to the Pathfinding Problem: The optimization problem can be converted to the pathfinding problem handled by the A* algorithm. The search space can be denoted by a search graph, which is a layered graph composed of multiple nodes, their connections, a starting node and a goal node. These nodes are arranged into m levels by the vertices in $G^{(c)}$, where the node z_{ij} at the ith location ($1 \le i \le p$) of level j ($1 \le i \le m$) represents the mapping from $v_j^{(c)} \in \mathbf{V}^{(c)}$. Each node at an upper level is connected with all nodes at its supper level. Thus, a solution of the optimization is defined as a path in the search tree of the problem from the top to one of the leaves, which is expressed a sequence S of the nodes on the path. Figure 1 shows the search graph, where the red lines from the left to the right form a solution path, and can be formulated as $S =< z_{p1}, z_{12}, \dots, z_{2m} >$.

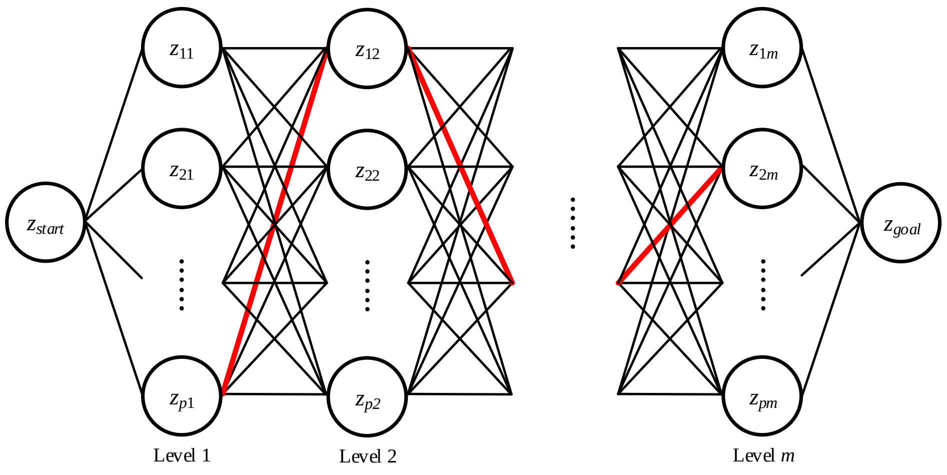

Figure 1. The Search Graph and a Solution Path

The process of a pathfinding is a move from a node in level 1 to a node in level m. The solution path is accumulated as the movement goes on, and the corresponding feasible solution is obtained when the movement ends at the last level. A partial solution path after the movement going to level b is expressed as S_b, thus, finally, the solution path $S = S_m$.

3.2.1. Designing the Path-cost Function: For S_{b-1} $(1 \le b-1 \le m)$, a node at level b is selected, then the movement goes to level b and chooses a node at next level, so forth until the selection at level m completes. In the A* algorithm, the node selection at level b is based on both of the past path-cost and the future path-cost. The former is the known cost from the starting node to the current node, while the latter is an admissible "heuristic estimate" of the distance from current node to z_{goal}. The path-cost of them for a selection z_{ab} are denoted by $g(a,b)$ and $h(a,b)$, respectively. Thus, the path-cost function for z_{ab} can be given by

$$f(a,b) = g(a,b) + h(a,b) \tag{5}$$

$g(a,b)$ can be obtained by Formula (1) and (3), namely,

$$g(a,b) = \max_{1<i<b} C_i = \max\left(C_1, \max_{2<i<b}\left(C_i^{(s)} + C_i^{(t)}\right)\right) \tag{6}$$

$$= \max\left(\frac{\sum_{j=1}^{m}\left(x_j^{(c)} d_{ij}\right)}{x_i^{(r)}}, \max_{2<i<b}\left(\frac{\sum_{j=1}^{m}\left(x_j^{(c)} d_{ij}\right)}{x_i^{(r)}} + \frac{\sum_{k=1}^{m}\sum_{l=1}^{m}\left(h_{ij}|d_{ik} - d_{il}|\right)}{y_i^{(r)}}\right)\right)$$

$h(a,b)$ represents the estimated path-cost from z_{ab} to z_{goal}. We define the minimum estimated path-cost from z_{ab} to z_{goal} as $h^{(*)}(a,b)$, thus, the optimal solution path can be found by the A* algorithm if $h(a,b) \le h^{(*)}(a,b)$ always holds for each z_{ab} in the search graph. Here, $h(a,b)$ can be obtained by Algorithm 1.

It can be observed that the computation of $h(a,b)$ is actually to compute a lower bound of $h^{(*)}(a,b)$. The bound can be obtained by the optimal mapping of the current unmapped components that communicate with the components in the resource with current maximum cost. In Algorithm 1, search of the resource with current maximum cost is conducted by the 1st step; the 2nd step first get all components in the resource, then find the current unmapped components that communicate with them; in the 3rd step, the minimum cost for each selected component are computed by comparing the costs induced by offloading to the current resource and other resource; finally, $h(a,b)$ can be obtained by sum up all of those costs in the 4th step.

Algorithm 1

```
1. Find the $v_i^{(r)}$ of a corresponding computing resources chosen by $S_b$, which satisfies that $C_i = \max_{1\le j\le b} C_j$;
2. Get the set $\boldsymbol{\alpha}_i$ containing all components that are mapped to $v_i^{(r)}$. For each component $v_k^{(c)} \in \boldsymbol{\alpha}_i$, find the components communicating with it from all components that are not involved by $S_b$, and add the result to set $\boldsymbol{\beta}_i$;
3. For each component $v_l^{(c)} \in \boldsymbol{\beta}_i$, compute the minimum cost $\theta_l$ induced by offloading it to other resources besides $v_i^{(r)}$, and the cost $\pi_l$ induced by offloading it to $v_i^{(r)}$. Thus, the cost of $v_l^{(c)}$ is defined as $\gamma_l = \min(\theta_l, \pi_l)$;
4. $h(a,b) = \sum_{v_l^{(c)} \in \boldsymbol{\beta}_i} \gamma_l$.
```

3.2.2. Solving the Optimization Problem: Through the A* algorithm, the optimal path $S^{(*)}$ which has the minimum path-cost $f(a,b)$ can be found, consequently, the optimization problem described in Formula (4) can be solved by converting $S^{(*)}$ to $D^{(*)}$. The process of our algorithm is described in Algorithm 2.

Algorithm 1

```
Input: $G^{(c)}$, $G^{(r)}$
Output: $D^{(*)}$
Initial: an empty priority queue OPEN, an empty priority queue CLOSED.
1.  Create the search graph from $G^{(c)}$ and $G^{(r)}$, put $z_{start}$ on OPEN;
2.  while the first node $z_{ij}$ in OPEN is not $z_{goal}$, do
3.    remove $z_{ij}$ from OPEN, then put it on CLOSED;
4.    if $v_{j+1}^{(c)} \in \mathbf{V}_u^{(c)}$, then      // $v_{j+1}^{(c)}$ is unoffloadable
5.      compute $f(1, j+1)$;
6.      if $z_{1(j+1)}$ is in OPEN and $f(1, j+1) <$ it stored path-cost $\tilde{f}(1, j+1)$, then
7.        let $\tilde{f}(1, j+1) = f(1, j+1)$, set the parent of $z_{1(j+1)}$ to $z_{1j}$;
8.      else if $z_{1(j+1)}$ is not in OPEN and $z_{1(j+1)}$ is not in CLOSED, then
9.        put $z_{1(j+1)}$ on OPEN, let $\tilde{f}(1, j+1) = f(1, j+1)$, set the parent of $z_{1(j+1)}$ to $z_{ij}$;
10.     end if
11.   else   // offloadable components
12.     for $k = 1{:}p$
13.       compute $f(k, j+1)$;
14.       if $z_{k(j+1)}$ is in OPEN and $f(k, j+1) < \tilde{f}(k, j+1)$, then
15.         let $\tilde{f}(k, j+1) = f(k, j+1)$, set the parent of $z_{k(j+1)}$ to $z_{ij}$;
```

```
16.         else if $z_{k(j+1)}$ is not in OPEN and $z_{k(j+1)}$ is not in CLOSED, then
17.             put $z_{k(j+1)}$ on OPEN, let $\tilde{f}(k,j+1) = f(k,j+1)$, set the parent of $z_{k(j+1)}$ to $z_{ij}$;
18.         end if
19.     end for
20.   end if
21.   reorder the nodes in OPEN in ascending order of path-cost;
22. end while
23. reconstruct the reverse path from goal to start by following parent pointers and
    exclude $z_{start}$ and $z_{goal}$, then the optimal solution path $S^{(*)}$ is formed;
24. for each $z_{ij} \in S^{(*)}$
25.   set $d_{ij} = 1$, and set other elements in column $j$ of $D^{(*)}$ to 0;
26. end for
```

The search graph are generated by $G^{(c)}$ and $G^{(r)}$. Two priority queues are defined and called OPEN and CLOSED, respectively. The former includes the nodes which are candidates for examining; the latter contains the nodes that have already been examined. Initially, the start node is put on OPEN, whereas, CLOSED is empty. Each node in the search graph keeps a pointer to its parent node to guarantee the linked list of the solution path can be found. The main loop pulls out and examines the first node on OPEN until the node is z_{goal}. In the loop, the nodes at next level are chosen, if the vertices corresponding to the level are unoffloadable, or offloadable, each of the nodes at this level are iteratively examined, if the node is on OPEN and the path-cost of current path with the node is lower than the stored patch-cost, then we update its stored path-cost and set the parent of the node. Finally, when z_{goal} is reached. The optimal solution path $S^{(*)}$ can be obtained by reversing the path from z_{goal} to z_{start}. Therefore, the optimal solution can be transformed from $S^{(*)}$ by setting the elements in each column of $D^{(*)}$.

4. Simulation Results and Analysis

The performance of our application multi-partitioning scheme is evaluated by simulations from different aspects. We first measure the effects on performance promotion by our scheme and the traditional approach. Then we test the complexity for computing the optimal solution via the A* algorithm used in our scheme and the Dijkstra algorithm. Finally, an example is given to show the computation of mapping between the component graph and the resource graph through our scheme. Table 1 shows the parameters used in simulations.

Table 1. The Parameters used in Simulations

Name	Meaning	Value
m	the number of vertices in $G^{(c)}$	$[5,25]$
n	the number of edges in $G^{(c)}$	$[5,25]$
$\left\|\mathbf{V}_u^{(c)}\right\|$	the number of unoffloadable components	$10\%m$
p	the number of vertices in $G^{(r)}$	$[3,11]$
$x_i^{(c)}$	the values of weights in $\mathbf{X}^{(c)}$	$[0.1,1.1]\mathrm{MI}$
$y_j^{(c)}$	the values of weights in $\mathbf{Y}^{(c)}$	$[0.05,0.5]\mathrm{MB}$
$x_k^{(r)}$	the values of weights in $\mathbf{X}^{(r)}$	$[1,5]\mathrm{MIPS}$
$y_l^{(r)}$	the values of weights in $\mathbf{Y}^{(r)}$	$[1,20]\mathrm{MB/S}$

The application's costs via our scheme and the approach which offloads components to the best resource in all surrounding resources are compared with that of the approach which offloads no components. Simulations are carried out with different parameters of $G^{(c)}$ and $G^{(r)}$ shown in Table 1. The parameters are randomly generated unless stated clearly. As illustrated in Figure 2, where the parameters of $G^{(c)}$ are variable and those of $G^{(r)}$are fixed ($p = 7$). The cost via our scheme is the lowest among the 3 approaches. The cost by offloading no components is the highest since all components are processed only in the terminal. The cost by offloading components to the best resource is worse than that by our scheme, because only a single resource is used for computation offloading in the former, whereas, all surrounding resources participate in the latter, so the performance is further promoted by fully utilizing parallelism. Figure 3 shows the costs via 3 approaches where the parameters of $G^{(r)}$ are variable and those of $G^{(c)}$ is fixed ($m = 15, n = 15$). The cost via our scheme is still the lowest, and that via the approach offloading no components is the highest. As p grows, the scale of resource graph expands, the leading advantage of our scheme is more significant since parallelism is promoted with the increase of the number of computing resources, so there are more chances that components can be offloaded to multiple resources.

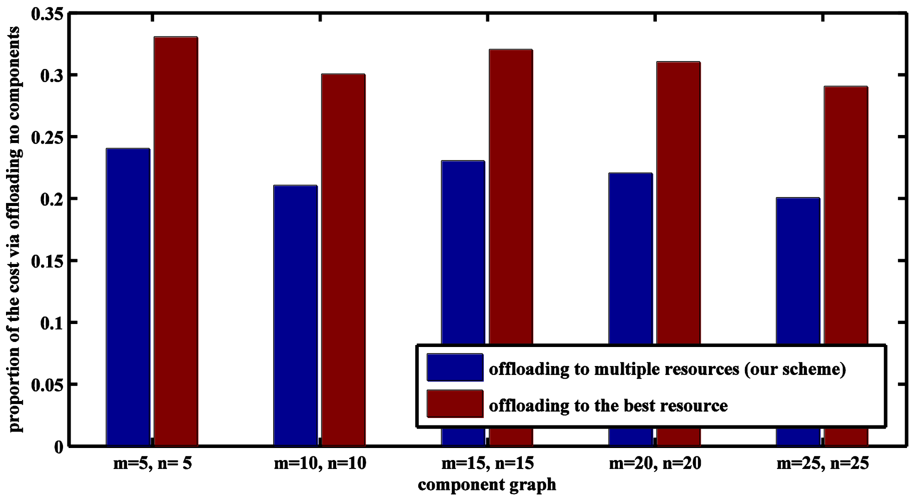

Figure 2. Comparison between the Costs via 3 Approaches with Variable $G^{(c)}$

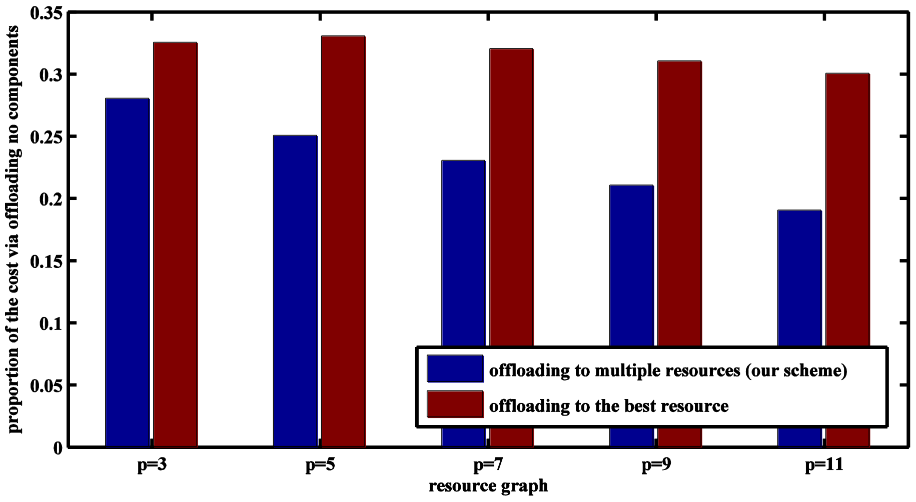

Figure 3. Comparison between the Costs via 3 Approaches with Variable $G^{(r)}$

In above simulations, the performance of the A* algorithm is also compared with that of the Dijkstra algorithm. The latter is actually a simple version of the former by setting $h(n) = 0$ for all nodes in the search graph. Although the optimal solution can be found by both of the two algorithm, the search counts of the A* algorithm is 36% of those of the Dijkstra algorithm on average, which proves the efficiency of our scheme.

An example is illustrated in order to analyze the process of our scheme in detail. The component graph and resource graph of the example is shown in Figure 4, where the weights of the vertices and edges are also configured. Note that $v_4^{(c)}$ in the component graph is unoffloadable. The optimal path of the example is $S^{(*)} = \{z_{41}, z_{42}, z_{23}, z_{14}, z_{55}, z_{46}\}$, based on which, the optimal solution is

$$D^{(*)} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{7}$$

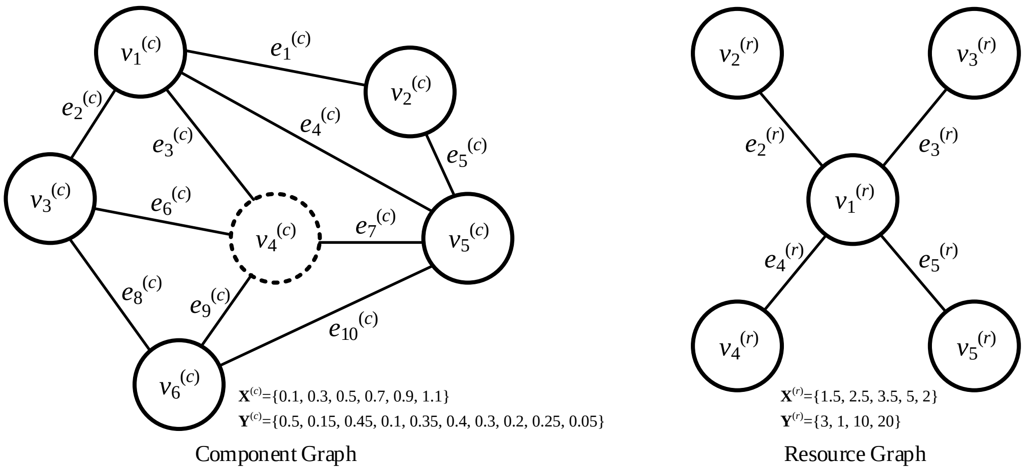

Figure 4. The Component Graph, Resource Graph, and Weights of the Example

5. Conclusion

The performance of an application can be further promoted by simultaneously offloading the application's components to multiple computing resources around the mobile terminal. In this paper, we propose an application multi-partitioning scheme, which minimize the computing and inter-resource costs. Our scheme converts the multi-partitioning problem into graph mapping problem, and transforms the optimization problem into pathfinding problem, then employs the A* algorithm to solve the problem efficiently. Simulations demonstrate the efficiency of our scheme, and the promotion of application's performance.

Acknowledgements

This work is supported in part by National Natural Science Foundation of China (Grant No. 61502050, 61170275), YangFan Innovative & Entrepreneurial Research Team Project of Guangdong Province, Civil Aerospace Science and Technology Project and Fundamental Research Funds for the Central Universities.

References

[1] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, “A survey of computation offloading for mobile systems,” Mobile Networks and Applications, vol. 18, no. 1, **(2013)** pp. 129–140.
[2] X. Ma, Y. Zhao, L. Zhang, H. Wang, and L. Peng, “When mobile terminals meet the cloud: computation offloading as the bridge.” IEEE Network, vol. 27, no. 5, **(2013)**, pp. 28–33.
[3] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, “Maui: making smartphones last longer with code offload,” in Proceedings of the 8th international conference on Mobile systems, applications, and services. ACM, **(2010)**, pp. 49–62.
[4] H. Wu, Q. Wang, and K. Wolter, “Tradeoff between performance improvement and energy saving in mobile cloud offloading systems,” in Communications Workshops (ICC), 2013 IEEE International Conference on. IEEE, **(2013)**, pp. 728–732.
[5] J. Oueis, E. C. Strinati, and S. Barbarossa, “Multi-parameter decision algorithm for mobile computation offloading,” in Wireless Communications and Networking Conference (WCNC), 2014 IEEE. IEEE, **(2014)**, pp. 3005–3010.
[6] S. Ou, K. Yang, and A. Liotta, “An adaptive multi-constraint partitioning algorithm for offloading in pervasive systems,” in Pervasive Computing and Communications, 2006. PerCom 2006. Fourth Annual IEEE International Conference on. IEEE, **(2006)**, pp. 116–125.
[7] K. Sinha and M. Kulkarni, “Techniques for fine-grained, multi-site computation offloading,” in Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. IEEE Computer Society, **(2011)**, pp. 184–194.
[8] C. Wang, Y. Li, and D. Jin, “Mobility-assisted opportunistic computation offloading,” Communications Letters, IEEE, vol. 18, no. 10, Oct **(2014)**, pp. 1779–1782.
[9] L. Wang and M. Franz, “Automatic partitioning of object-oriented programs for resource-constrained mobile devices with multiple distribution objectives,” in Parallel and Distributed Systems, 2008. ICPADS’08. 14th IEEE International Conference on. IEEE, **(2008)**, pp. 369–376.
[10] F. Sadri, “Ambient intelligence: A survey,” ACM Computing Surveys (CSUR), vol. 43, no. 4, p. 36, 2011.

Authors

Wenhao Fan received the B.E. and Ph.D. degree from Beijing University of Posts and Telecommunications (BUPT), Beijing, China, in 2008 and 2013, respectively. He is currently an assistant professor at the School of Electronic Engineering in BUPT. His main research topics include mobile computing, parallel computing and transmission, information security for mobile terminals and software engineering for mobile internet.