An Effective Dynamic Programming Offloading Algorithm in Mobile Cloud Computing System

Yanchen Liu, Myung J. Lee
Department of Electrical Engineering, City College, the City University of New York
160 Convent Avenue, New York, NY, USA, 10031
Email: {yliu2, lee}@ccny.cuny.edu

Abstract— **Mobile applications are providing increasingly richer functionalities, which generally result in high computational complexity and thus high energy consumption of mobile devices. In this article, to alleviate the computational burden of mobile devices, we present a Dynamic Programming based Offloading Algorithm (DPOA) to quickly find the optimal partitioning between executing subcomponents of a mobile application at the mobile device and the cloud server, taking into account the CPU speed of mobile device, network performance, the characteristics of an application program, and the efficiency of cloud server. DPOA solves the offloading optimization problem with much lower complexity than the Branch & Bound used in [1][2], while significantly reducing the execution time of mobile application proved by the simulations.**

Keywords— Mobile Cloud Computing; application partitioning; offloading algorithm

I. INTRODUCTION

The limited computation capability and the short battery life of mobile devices are the main bottlenecks to the execution of most mobile applications. Mobile Cloud Computing (MCC) is proposed to alleviate the computation load on mobile devices by offloading certain suitable subcomponents of an application to the cloud server to complete the whole application quickly and energy-efficiently, where mobile devices connect to the Internet through wireless network and then communicate with the remote cloud server. The cloud server can provide huge storage, reliable security, as well as high computation power [3], which can help significantly improve the mobile application performance and extend the battery life of mobile devices. And with the development of the advanced wireless communication technology/standard, such as 4G and LTE, which can help offload applications with much less cost to the cloud server, MCC is rapidly expanding its domain of applications.

The problem whether to offload certain components of an application to the cloud depends on the following factors: CPU speed of mobile device, network performance, transmission data size, and the efficiency of the cloud server. With considering these factors, [4] proposes offloading the whole application to the cloud server without partitioning the application. However, not all the components of an application can be offloaded to the cloud end. For example, the methods of implementing mobile I/O devices and user interfaces should be executed at the mobile end for sure.

In order to save the computation time and the energy use of mobile devices, two optimal partitioning solutions to decide which subprograms/methods to be offloaded to the cloud server are presented in CloneCloud [1] and MAUI [2]. Their offloading solutions utilize offline network characteristics without considering the time-varying network condition, and hence cannot guarantee the optimal offloading decision especially while the network changes a lot in a short time. Failure to offload appropriate subprograms to the cloud will inevitably result in inefficiency of the mobile cloud computing system. Therefore, it is very important to have an offloading algorithm that has low time complexity and ideally can achieve the optimal solution as quickly as if it is a real-time offloading decision.

A Linear Programming Solver (LP solver) based on the Branch and Bound (B&B) algorithm is used in schemes of [1], [2] and [5]. B&B is a feasible approach for solving integer linear problems when the number of "branches" is not large; but the number of its optional solutions grows exponentially with the number of subprograms, which means higher time complexity($O(2^n)$). In this paper, we propose Dynamic Programming based Offloading Algorithm (DPOA) that has much lower time complexity, proportion to the square of the number of subprograms($O(n^2)$), and the potential of parallel execution. Therefore, compared to B&B, DPOA can achieve an optimal offloading strategy so quickly, and by shortening the strategy update period, the offloading decision results obtained by DPOA can be considered as running in real-time approximately, which is crucial to the efficiency of real-time applications in the system of MCC.

Besides low time complexity, parallelism is another pursuit for Mobile Cloud Computing. Parallelism is exploited in [6] by parallelizing subprogram/method execution using multiple virtual machines, while [7] utilizes distributed shared memory to offload. Our proposed DPOA focuses on how to quickly find the optimal partition of an application between executing the subcomponents at the mobile device and the cloud server. The output of DPOA can be easily utilized in parallel execution systems.

In this paper, a Dynamic Programming based Offloading Algorithm (DPOA) in MCC system is presented. The performance simulation shows that the proposed DPOA has much lower time complexity compared to the existing algorithms used in [1][2], and thus can substantially reduce the battery consumption.

The remainder of this paper is organized as follows. The system model is described in Section II. A dynamic programming based offloading algorithm is developed in

978-1-4799-3083-8/14/$31.00 ©2014 IEEE

Section III. Experimental simulation and discussion are presented in Section IV. Finally, concluding remarks and future work are given in Section V.

II. System Model

A mobile application is composed of multiple components/methods. Assume all the methods can be executed at the mobile device, and some of them can be executed at the cloud server. The same method costs differently at the mobile device from at the cloud server in terms of execution time and energy consumption. For two consecutive methods, there will be extra data transition time and energy cost for the mobile device if current method changes its execution position from the one where previous method runs. On the other hand, if both of the consecutive methods executed at the same side, no extra time or cost incurred. The transition time and energy cost for the mobile device varies according to the size of the data to be transmitted and the wireless network condition.

For the problem formulation, $D(i)$ represents the execution cost of method i at the mobile device; $C(i)$ represents the execution cost of the mobile device during the method i running at the cloud, which is not the cost of the cloud server. $CS(i)$ represents the cost of transmitting method i, which is calculated by multiplying the cost per bit with the size of the method. $P(i)$ represents the position where method i executes: {1: in the mobile device; 0: in the cloud server}. The cost mentioned here can be either the related execution time cost or the energy cost for the mobile device.

Therefore, an optimization problem can be formulated to obtain the minimal cost in the system. The objective function is to minimize:

$$\sum_{i=1}^{N}(P(i)\times D(i)+(1-P(i))\times C(i)+\left|P(i)-P(i-1)\right|\times CS(i)) \quad (1)$$

where N is the number of methods of an application.

From function (1), we can learn that the transition cost of method i is counted only when the values of $P(i)$ and $P(i\text{-}1)$ are not equal. Otherwise, if method i and method i-1 take place at the same place, the transition cost is 0 for the system.

There are two constrains for the objective function:

$$\sum_{i=1}^{N} P(i) \le N, P(i)=1 \text{ or } 0 \qquad P(s)=1, s\in UF \quad (2)$$

where UF is the set of the methods' indexes that are unoff-loadable, like the methods implementing mobile I/O devices and user interfaces.

$D(i)$ and $C(i)$ can be measured using a software profiler. ThinkAir[6] utilizes PowerTutor to measure the energy consumption of methods. MAUI[2] uses a power meter attached to the smartphone's battery to build an energy profile. They build a simple linear model by using collected samples of CPU utilization and the corresponding energy consumption, which predicts the energy consumption of a method as a function of the number of CPU cycles it requires to execute.

The transition cost $CS(i)$ can be tracked by a network profiler. Network throughput can be obtained by measuring the time duration when sending a certain number of data as in [1]. It is necessary to measure the network characteristics in a short period of time since the wireless and computing environments keep changing over time.

In this article, $D(i)$, $C(i)$ and $CS(i)$ are assumed to be measured and calculated for each method of the application software. The objective function (1) is solved in [1] [2] by utilizing B&B algorithm, which is an adaptive partition strategy to solve the integer linear programming problem. Although B&B is a feasible approach when the height of a solution tree is small, it usually leads to an exponential time complexity, which is inefficient once the number of methods tends to be large. In this paper, we seek a dynamic programming based algorithm to obtain the optimal partition solution with much lower time complexity.

III. Dynamic Programming based Offloading Algorithm(DPOA)

DPOA provides a stably low time complexity method for determining which methods of the application software should be offloaded to the cloud server in order to save mobile devices' energy and to reduce application execution time. This novel approach utilizes a DP table to find the best solution for the objective of minimizing the whole time/energy cost. Dynamic Programming is a method to solve the optimal problem by breaking it down into simpler sub-problems. Its efficiency has been proved in many areas of wireless communication for solving optimization problem within an acceptable computation time, such as finding the optimal routing strategy [8], scheduling scheme [9], power management[10] [11], etc.

The framework of DPOA is shown in Figure 1. Device profiler and application analyzer calculate the cost of the method when running on the mobile device according to the CPU speed and the application software complexity. Network profiler calculates the method's transition cost in terms of the network characteristics and the data sent by each method. Constraint analyzer provides the indexes of application methods that are not offloadable. With the above information, DPOA solver calculates and produces the offloading decision for a mobile application before the execution of the application program.

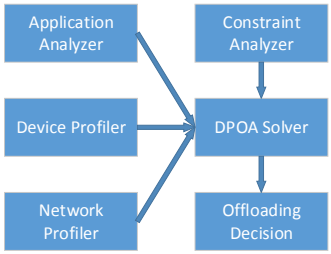

Fig. 1. The framework of Dynamic Programming Offloading Algorithm solver

A. Dynamic Programming Table

As shown in Table 1, a two-dimensional Dynamic Programming (DP) table is built in DPOA system to calculate the shortest path for offloading decision.

$M(i,j)$ saves the actual index of the $(i+j)th$ offloadable method of an application. The actual index could be different from $(i+j)$ since there may be unoffloadable methods within that application. For example, an application contains four methods, and the second method is unoffloadable, thus in the DP table $M(0,1)=M(1,0)=1$, $M(0,2)=M(2,0)=M(1,1)=3$, $M(3,0)=M(0,3)=M(2,1)=M(1,2)=4$. Note that only the offloadable methods are considered in the DP table, hence the sum of i and j is no more than NM, which is the total number of offloadable methods within an application besides the unoffloadable methods.

$T(i, j)$ associated with the cell (i, j) represents the minimal cost of all the offloadable and unoffloadable methods by the $(i+j)th$ offloadable method of an application. Here, the row index i means the number of transitions made between the mobile device and the cloud server, and the column index j means the number of no-transitions. For example, an application contains four methods, and the second method is unoffloadable, then $T(2,1)$ includes three parts, which are the execution cost of the second method at mobile device, and the minimum transition cost and execution cost of the other methods under the condition of two transitions between the mobile device and the cloud server.

TABLE I. DP TABLE FOR OFFLOADING ALGRITHM

$T(0,0)$ $M(0,0)$	$T(0,1)$ $M(0,1)$	$T(0,2)$ $M(0,2)$	….	$T(0,NM-1)$ $M(0,NM-1)$	$T(0,NM)$ $M(0,NM)$
$T(1,0)$ $M(1,0)$	$T(1,1)$ $M(1,1)$	….	$T(1,NM-2)$ $M(1,NM-2)$	$T(1,NM-1)$ $M(1,NM-1)$	
$T(2,0)$ $M(2,0)$	….	$T(i-1,j)$ $M(i-1,j)$	….		
….	$T(i,j-1)$ $M(i,j-1)$	$T(i ,j)$ $M(i ,j)$			
….	….				
$T(NM,0)$ $M(NM,0)$					

1) Calculation of $M(i ,j)$

$M(i ,j)$ is an important identifier for calculation of $T(i,j)$. Each $M(i,j)$ can be calculated as following:

a) $M(0,0)$ is equal to 0.

b) For each of other $M(i,j)$s in the DP table, set it as the index of the next offloadable method after method $M(i,j-1)$ or $M(i-1,j)$ within an application as indicated in equation (3). Either $M(i,j-1)$ or $M(i-1,j)$ can be used for calculation of $M(i,j)$ when both of them exist in the table.

$$M(i,j)=\begin{cases} M(i,j-1)+\#\ of\ unoffloadable \\ \quad methods\ right\ after\ method\ M(i,j-1) & i=0 \\ M(i-1,j)+\#\ of\ unoffloadable \\ \quad methods\ right\ after\ method\ M(i-1,j) & i>0 \end{cases} \tag{3}$$

Filling $M(i,j)$ in this way can avoid counting the unoffloadable methods during the calculation of $T(i ,j)$.

2) Calculation of $T(i ,j)$

Right after the calculation of each $M(i,j)$, each $T(i ,j)$ is calculated and filled in the DP table in sequence from left to right, or top to bottom of the table. Whether to calculate $T(i,j)$ based on $T(i,j-1)$ or $T(j,i-1)$ depends on its position in the DP table and which candidate can make it a smaller value. $T(i,j)$ can be obtained as following:

$$T(i,j)=\begin{cases} 0, & i=0\ \&\ j=0 \\ T(i,j-1)+\sum_{k=M(i,j-1)}^{M(i,j)} D(k), & i=0\ \&\ j>0 \\ T(i-1,j)+Sum\_D \\ +CS\left(M(i-1,j)+1\right)+D\left(M(i,j)\right), & i=even\ \&\ j=0 \\ T(i-1,j)+Sum\_D \\ +CS(M(i,j))+C\left(M(i,j)\right), & i=odd\ \&\ j=0 \\ \min(T(i,j-1)+\Delta x, T(i-1,j)+\Delta y) & i>0, \&\ j>0 \end{cases} \tag{4}$$

where

$$Sum\_D=\begin{cases} 0 & if\ (M(i,j)=M(i,j-1)+1) \\ \sum_{k=M(i,j-1)+1}^{M(i,j)-1} D(k) & otherwise \end{cases} \tag{5}$$

$$\Delta x=\begin{cases} D(M(i,j)) & i\ is\ even\ \& \\ & M(i,j)=M(i,j-1)+1 \\ C(M(i,j)) & i\ is\ odd\ \& \\ & M(i,j)=M(i,j-1)+1 \\ \sum_{k=M(i,j-1)+1}^{M(i,j)-1} D(k)+D(M(i,j)) & i\ is\ even\ \& \\ & M(i,j)\neq M(i,j-1)+1 \\ \sum_{k=M(i,j-1)+1}^{M(i,j)-1} D(k)+C(M(i,j)) & i\ is\ odd\ \& \\ +CS(M(i,j-1)+1)+CS(M(i,j)) & M(i,j)\neq M(i,j-1)+1 \end{cases} \tag{6}$$

$$\Delta y=\begin{cases} D(M(i,j))+CS(M(i,j)) & i\ is\ even \\ & M(i,j)=M(i-1,j)+1 \\ C(M(i,j))+CS(M(i,j)) & i\ is\ odd \\ & M(i,j)=M(i-1,j)+1 \\ \sum_{k=M(i,j-1)+1}^{M(i,j)-1} D(k)+D(M(i,j)) & i\ is\ even \\ +CS(M(i-1,j)+1) & M(i,j)\neq M(i-1,j)+1 \\ \sum_{k=M(i,j-1)+1}^{M(i,j)-1} D(k)+C(M(i,j)) & i\ is\ odd \\ +CS(M(i,j)) & M(i,j)\neq M(i-1,j)+1 \end{cases} \tag{7}$$

a) When both i and j equel 0: $T(i,j)$ equals 0 since there is no *0th* method defined in DP table.

b) When i equels 0 and j is larger than 0: this means there is no transition up to the $(i+j)th$ method, $T(i,j)$ equals the value of $T(i,j-1)$ plus the sum of the execution costs of method from $M(i,j-1)+1$ to $M(i,j)$ on the mobile device.

c) When j equals 0 and i is larger than 0: this means there must be a transition either from the mobile device to the

cloud, or from the cloud to the mobile device between method $M(i\text{-}1,j)$ and method $M(i,j)$ in DPOA system. Therefore, the value of $T(i,j)$ is the sum of $T(i\text{-}1,j)$, Sum_D (the execution costs on the device of the possibly existing unoffloadable method between method $M(i\text{-}1,j)$ and $M(i,j)$-1), transition cost $CS(M(i\text{-}1,j)+1)$, and the execution cost of method $M(i,j)$ at the cloud/mobile device. If i is even, it indicates there has been even number times transitions up to $T(i,j)$. The transition happens between $M(i\text{-}1,j)$ and $M(i\text{-}1,j)+1$ and $M(i,j)$ would execute at the mobile device. Thus, the last two iterms of the expression should be $CS(M(i\text{-}1,j)+1)$ and $D(M(i,j))$. Otherwise, the transition happens between $M(i,j)$-1 and $M(i,j)$, and $M(i,j)$ happens at the cloud, then the last two items of the expression should be $CS(M(i,j))$ and $C(M(i,j))$.

d) When both i and j are larger than 0: For $T(i\ ,j)$ in the rest place of DP table, its value equals either its left variable $T(i, j\text{-}1)$ plus Δx that is the "distance" between two adjacent cells on the table row direction, or its upper variable $T(i\text{-}1, j)$ plus Δy that is the "distance" between two adjacent cells on the table column direction. Which path to choose depends on which one makes $T(i\ ,j)$ smaller. Δx is calculated as equation (6): If $M(i,j)$ is one larger than $M(i,j\text{-}1)$, representing there is no other method belonging to UF set between these two methods, Δx is chosen as $D(M(\text{i}\ ,j))$ or $C(M(\text{i}\ ,j))$ depending on whether i is even(representing even times transition by far) or not. If $M(i\ ,j)$ is not equal to $M(i\ ,j\text{-}1)+1$, which means that there are some unoffloadable methods exist between these two methods, Δx is the sum of the costs of these methods on the mobile cloud, the execution cost of $M(i,j)$ and possible transition costs. When i is even, the execution cost is $D(M(i,j))$ and there is no transition costs counted. Otherwise, the execution cost is $C(M(i,j))$, and two corresponding transition costs are added. Similarly, Δy is calculated in the following way: If $M(i,j)$ is equal to $M(i\text{-}1,j)+1$, Δy is the sum of $M(i\ ,j)$ execution cost on the mobile device or the cloud server, and its transition cost. Otherwise, the additional unoffloadable methods' execution cost should also be counted.

B. Selection of Offloading Decision

Based on the rules for the calculation of $M(i,j)$ and $T(i,j)$ in the previous section, Dynamic Programming Offloading Algorithm can be described as the pseudo code in Table II.

At each row of the DP table, we calculate each $M(i, j)$ and $T(i, j)$ according to formula (3) and (4). In addition, as described in line 17 and 21 of Table II, at the end of each row of the DP table, we replace *minCost* by the current $T(i,j)$ if $T(i,j)$ is less than *minCost*. At the meantime, record related i and j as *final_i* and *final_j*. When the whole table is completed, *minCost* is the minimal cost of executing an application that this mobile computing system can achieve. According to *final_i*, *final_j* and the associated $T(final\_i, final\_j)$, the optimal path in the DP table can be tracked using Backtracking approach [12]. Following the optimal path, an optimal decision about whether to offload each method of an application can be achieved.

TABLE II. DYNAMIC PROGRAMMING OFFLOADING ALGORITHM

```
1:  Initialize M(0,0) to 0.
2:  Initialize T(0,0) to 0.
3:  Initialize minCost to  sum_{i=1}^{N} D(i) .
4.: for (j=1 to NM):
5:      set M(0, j);
6.     set T(0,j) according to equation (4);
7:  end for
8:  for (i=1 to NM):
9:      set M(i,0);
10.    set T(i,0) according to equation (4);
11: end for
12: for (i =1 to NM):
13:        for (j = 1 to NM-i):
14:            set M(i,j);
15:            set T(i,j) according to equation (4);
16:        end for
17:        if T(i ,j) is less than minCost, then:
18:                set minCost to T(i ,j);
19:                set final_i =  i;
20:                set final_j = j;
21:        end if
22: end for
```

IV. EVALUATION

In this section, we evaluate the performance of the proposed Dynamic Programming based Offloading Algorithm using a simulator written in MATLAB. In the simulation, the parameters, such as the number of methods, are selected as an example for performance of illustration. The improved performance of execution time and energy cost saving is evaluated by comparing with B&B algorithm, which is used in CloneCloud [1] and MAUI [2]. After investigating the simulation results, a further discussion about the requirements of being an efficient offloading algorithm in MCC is followed.

A. Simulation and Results

DPOA is programmed using MATLAB as the algorithm described in Section III, while B&B solver is simulated based on the description in [13]. Except B&B, we also compare DPOA with the approaches of Local Execution (all the methods of an application are executed at the mobile device) and Offloading All (all the offloadable methods are offloaded to the cloud server to run). The computation time of the four algorithms and their corresponding application execution time and energy cost are shown and analyzed below.

Firstly, we simulate and compare DPOA, Local Execution and B&B on face recognition application. We utilize the execution costs on both mobile device and the cloud, and the transition cost of each method in the face recognition application studied in [2] as the inputs of our DPOA calculation. Based on the

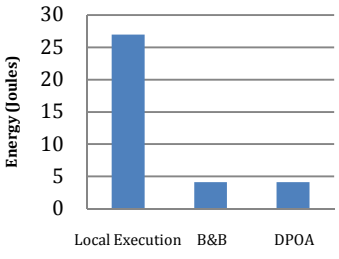

Fig. 2. Comparison of consumed energy by using local execution, B&B and DPOA on face recognation application

execution cost, transition cost and the offloading decision the three approaches made, the energy costs of an application on the mobile phone are calculated according to function (1). The results are shown in Fig. 2, from where we find that the offloading decisions calculated by DPOA and B&B can help smart phone save around 85% energy cost compared with Local Execution approach.

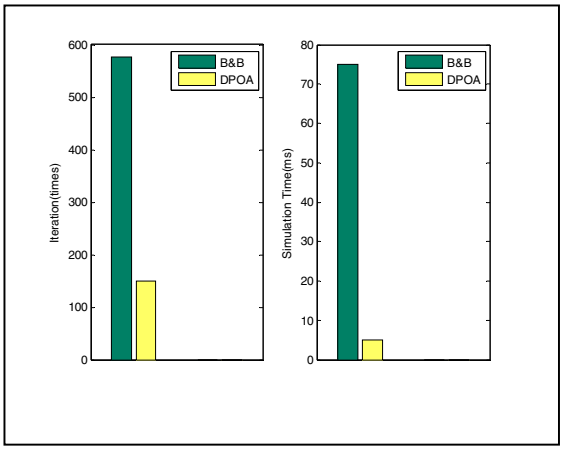

Fig. 3. Comparisons of algorithm simulation iterations and simulation time on face recognition application

During the calculation of offloading decision, the number of visiting of each DP table cell in DPOA and the number of visiting of each branch of B&B approach are counted as their own iteration number. As shown in Fig.3, the iteration number and the actual simulation running time of DPOA are respectively reduced by 75% and 95% compared to the simulation results of B&B approach. Therefore, for the example of face recognition application, DPOA cannot only provide the optimal solution as B&B but also is a much faster algorithm than B&B.

Secondly, we increase the number of methods of the running application to see the computation speed of DPOA, Local Execution, and Offloading All. The local execution time, the cloud execution time, and the transition time of each method is randomly chosen during (0.5ms, 1.5ms), (4.0ms, 6.5ms), and (0.5ms, 2.5ms), respectively. For fairness, the same execution time and transition time is used during the simulation of the three approaches. Fig. 4 presents the comparison of operating time of the same application using different offloading strategy decided by the three approaches. From the results, we can tell that Offloading All performs better than Local Execution, because the execution cost of running methods in the cloud server is significantly lower than at the mobile devices under certain adequate wireless network condition; But DPOA performs best, because it will avoid offloading methods in the case of the large transition cost between consecutive methods compared to Offloading All approach, and offload the more appropriate methods to the cloud server.

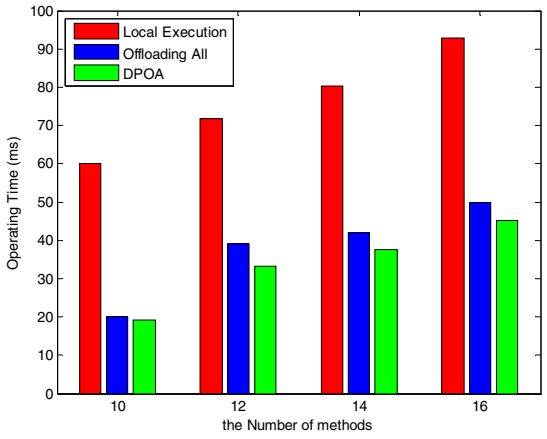

Fig. 4. Comparison of execution time among LE, OA, and DPOA

The iteration numbers and simulation time of B&B and DPOA are obtained through simulation against the increase of the number of application methods. Fig. 5 presents the iteration number during the simulation, while Fig. 6 presents the simulation time in a PC computer. It is indicated that as the number of methods increases, the iteration number of B&B rises shapely, which can be explained by its $O(2^n)$ computation complexity. Although B&B usually stops searching before exhaustion, which saves computation time, it is still substantially outpaced by DPOA, whose computation complexity is $O(n^2)$. The simulation time in Fig.6 also shows a remarkably superior performance of DPOA over B&B. B&B takes the computation time from 9ms to 80ms when number of methods increases. In contrast, DPOA completes the calculation within 1ms even when the number of methods reaches 20.

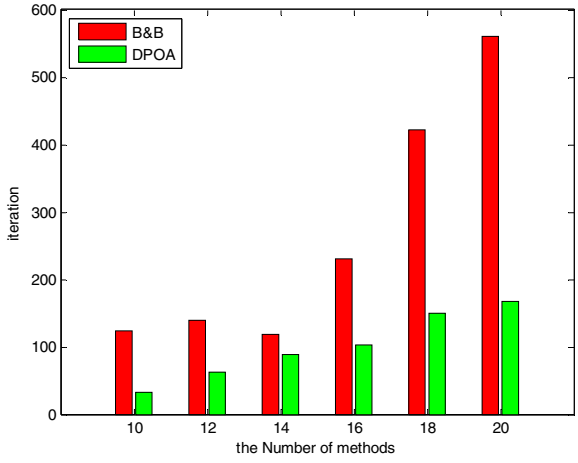

Fig. 5. Improvement of iteration number of DPOA comparing with B&B Offloading algorithm

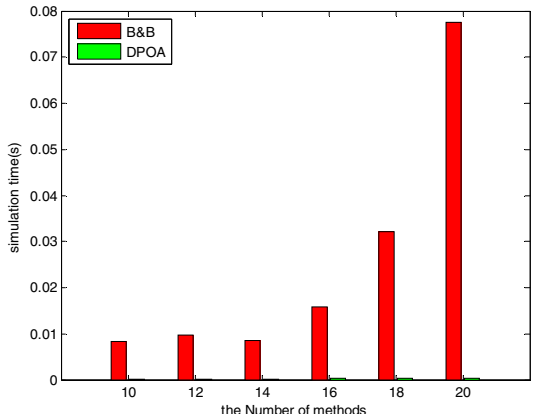

Fig. 6. Improvement of simulation time of DPOA comparing with B&B Offloading algorithm

Based on the results of the simulation, we can tell that: (1) a non-optimal offloading algorithm costs mobile device more resource no matter whether to run the application locally or to transmit it to the remote for calculation; (2) DPOA performs much better than B&B when the application is larger with more methods in terms of running time. It means DPOA can complete the offloading calculation at the mobile device with even less time and energy.

B. Discussion

A good offloading algorithm plays an indispensable role in the MCC system, and is expected to meet the following requirements:

1) Fast speed of calculation

Since some of important factors, like available cloud servers, mobile users' position, and wireless network keep changing during the execution of applications, a fast offloading algorithm is necessary to guarantee the efficiency of the MCC system. $O(n^2)$ computation complexity of DPOA supports a much faster speed for calculating the optimal offloading decision, while exponential computation complexity limits the realistic use of B&B..

2) Dynamic update of offloading decision:

Once the wireless environment or the being executed methods change, the related offloading decision should be recalculated. The capacity of dynamic update for offloading decision is supported by DPOA, for which the only thing needs to be adjusted before the processing of program is the related costs of methods in DP table. Then, a new offloading decision can be recalculated and updated quickly.

V. Conclusion and Future Work

In this paper, we present a novel DPOA for the optimal application offloading algorithm in Mobile Cloud Computing system. The high efficiency of our algorithm allows the mobile device to calculate the optimal offloading decision at the local end with much lower time complexity and energy consumption. The superior performance of DPOA over B&B and other algorithms is verified by extensive simulations. One of our future work is to upgrade DPOA to an online algorithm helping the mobile cloud computing system to calculate the optimal offloading decision during the application execution, to adapt to the dynamic network characteristics.

References

[1] Byung-Gon Chun, Sunghwan Ihm, CloneCloud: Elastic Execution between Mobile Device and Cloud, EuroSys' 11, April 2011.

[2] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, MAUI: Making Smartphones Last Longer with Code Offload, MobiSys'10, June, 2010.

[3] Karthik Kumar, J. Liu, and Y. Lu, B. Bhargava. A Survey of Computation Offloading for Mobile Systems. Mobile Networks and Applications, April 2012.

[4] Karthik Kumar and Yung-Hsiang Lu, Cloud Computing for Mobile Users: Can Offloading Computation Save Energy? IEEE Computer Society, April 2010.

[5] Ryan Newton, Sivan Toledo, Lewis Girod, Hari Balakrishnan, and Samuel Madden, Wishbone: Profile-based Partitioning for Sensornet Applications, NSDI '09, April, 2009.

[6] Sokol Kosta et al. ThinkAir: Dynamic resource allocation and parallel execution in cloud for mobile code offloading, INFOCOM, 2012 Proceedings IEEE, March 2012.

[7] Mark S. Gordon, D. Anoushe Jamshidi, Scott Mahlke, Z. Morley Mao, and Xu Chen, "COMET: Code Offload by Migrating Execution Transparently," In Proc. of USENIX OSDI, 2012.

[8] Terrence Mak, Peter Y. K. Cheung, Kai-Pui Lam, and Wayne Luk, Adaptive Routing in Network-on-Chips Using a Dynamic-Programming Network, Industrial Electronics, IEEE Transections on, VOL. 58, NO. 8, August 2011.

[9] Aditya Dua, Carri W. Chan, Nicholas Bambos, and John Apostolopoulos, Channel, Deadline, and Distortion (CD2) AwareScheduling for Video Streams over Wireless, Wireless Communications, IEEE Transections on, VOL. 9, NO. 3, March 2010.

[10] Ali Sharifkhani, and Norman C. Beaulieu, Dynamic Power Allocation over Block-Fading Channels with Delay Constraint, Globecom 2009, November 2009.

[11] Afshin Fallahi and Ekram Hossain, a Dynamic Programming Approach for QoS-Aware Power Management in Wireless Video Sensor Networks, Vehicular Technology, IEEE Transections on, VOL. 58, NO. 2, February 2009.

[12] C. Tripp and R. Shachter, Backtraking for More Efficient Large Scale Dynamic Programming, Machine learning and Application(ICMLA), 2012 11th International Conference, December 2012.

[13] Ninad Thakoor and Jean Gao, Branch-and-Bound for Model Selection and Its Computational Complexity, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 23, NO. 5, May 2011