Bandwidth Aware Application Partitioning for Computation Offloading on Mobile Devices

Feifei Wu, Jianwei Niu, and Yuhang Gao

School of Computer Science and Engineering, Beihang University,
Xueyuan Road. 37, 100191 Beijng, China
{wufeifei,niujianwei,gaoyuhang}@buaa.edu.cn

Abstract. Computation offloading is a promising method for reducing power consumption of mobile devices by offloading computation to remote servers. For computation offloading, application partitioning is a key component. However, making a good application partitioning is challenging, as it needs to carefully consider the tradeoffs between the communication cost and computational benifits. Most of previous work makes application partitioning by using a static bandwidth to measure the communication cost and thus cannot adapt to scenarios with dynamic bandwidth. To address this problem, in this paper, we propose a Bandwidth Aware Application Partitioning Scheme (BAAP). BAAP models the bandwidth as a random variable and formulate the application partition as a 0-1 Integer Programming with Probability (IPP) problem. Then BAAP adopts Branch and Bound algorithm to solve the problem. Experimental results show that BAAP can greatly reduce energy consumption while satisfying the cost and time constraints with guaranteed confidence probabilities regardless of different network bandwidth.

Keywords: Computation offloading, Graph Partitioning, Energy Saving, Mobile Devices, Confidence Probability.

1 Introduction

With the fast development of mobile technologies, mobile devices, such as smart phones, have become the primary computing platform for many users, which can provide a range of services and applications. However, the limited battery life is still a big obstacle for the further growth of mobile devices. Various studies have identified longer battery lifetime as the most desired feature of mobile devices, therefore, prolonging the battery life of mobile devices has become one of the top challenges.

Much work has been done to address this problem by offloading computation from smart phones to remote resource rich servers [1][2][3][4] to reduce energy consumption, which is called computation offloading or code offloading. For example, Diaconescu [4] proposed a compiler and runtime infrastructure for automatically partitioning and offloading java applications, and Roelof Kemp proposed Cockoo[3], an offloading system for android applications.

Joel J.P.C. Rodrigues et al.: (Eds.): GreeNets 2011, LNICST 51, pp. 63–72, 2012.

The key component of computation offloading is application partitioning, which partitions the application into one local execution part and one or more remote execution parts. However, making a good application partitioning is challenging, as it needs to carefully consider the tradeoffs between the communication cost and computational cost: running a part of application locally will cause much energy consumption as the CPU needs to conduct complex computation while running the part remotely will led to extra communication cost (energy, money and delay) for transmitting the application state, code, and so on.

A traditional solution to application partitioning is to model it as a graph partitioning problem. The application is represented as a Weighted Object Relation Graph (WORG), whose nodes represent the application components and edges represent the interaction between components. The weights of nodes and edges represent the computational cost and communication cost between components respectively. Then an optimization algorithm can be applied to solve the graph partitioning problem to minimize the cost.

Apparently, an accurate measurement of communication cost is critical to partitioning decision, and the communication cost is largely decided by the network bandwidth. Some previous work [4][5][6] uses a static bandwidth to compute the communication cost when making application partitioning and thus cannot be applicable to those scenarios in which bandwidth is changing dynamically. In this paper, we propose a Bandwidth Aware Application Partitioning (BAAP) algorithm, which can adapt to dynamic bandwidth. We model the bandwidth as a random variable and construct a probabilistic WORG of the application with the weights of edges as a function of the random variable. Then, we formulate the application partitioning problem as a 0-1 Integer Programming with Probability (IPP) problem. Then BAAP adopts Branch and Bound to solve the problem to optimally determines which part of the application should be run locally and which part should be run remotely. Experimental results show that our algorithm can greatly reduce energy consumption while satisfying the cost and time constraints with guaranteed confidence probabilities regardless of different network conditions.

The rest of this paper is organized as follows: Section 2 provides a detailed description of our Bandwidth Aware Application Partitioning scheme. Experiment and analysis are presented in section 3. Section 4 provides some concluding remarks.

2 Bandwidth Aware Application Partitioning

2.1 Overview

Figure 1 shows the schematic workflow of our Bandwidth Aware Application Partitioning (BAAP) Algorithm. Currently, BAAP is target to Java Applications, but can be extended to any other Object-Oriented Applications (e.g., C++, C#). Taking a Java Application as input, BAAP first constructs the WORG of the application by static Call Graph Analysis and dynamic Profiling. Then, the constructed WOGR is passed to the Graph Partitioning module for partitioning. Based on the collected Bandwidth information and Application Specific Constraints(time constraint, cost constraint and energy constraint), the Graph Partitioning module works out the optimal assignment of each WORG node to run locally or remotely that fulfill the

constrains. Once the graph is partitioned, a distributed version of application will be generated to automatically offload the code to remote server according to the partitioning results. The details of WORG constructing and Graph Partitioning will be discussed in section 2.2 and 2.3, respectively.

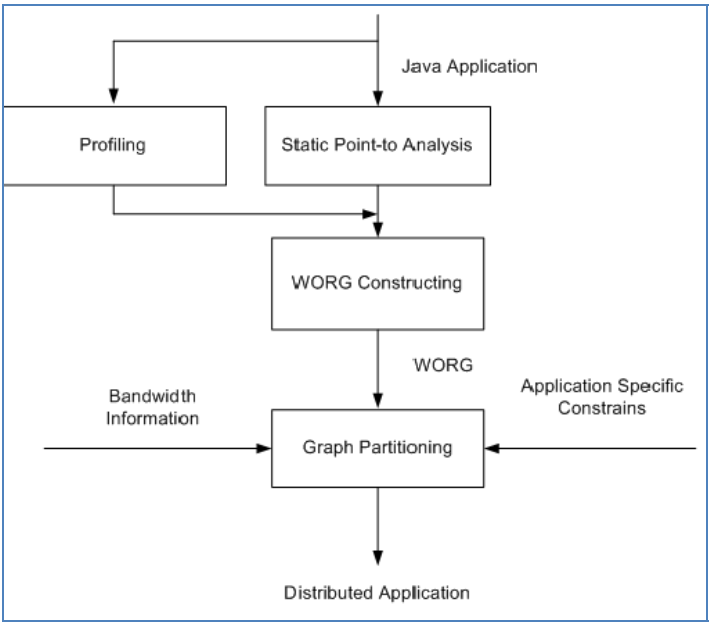

Fig. 1. Schematic workflow of BAAP

2.2 WORG Construction

BAAP models the application as a Weighted Object Relation Graph(WORG), with each node represents a run time object of the application and each edge represents the interaction between objects (we define three types of interaction: C<Creation>, I<Invocation>, D<DataAccess>). Moreover, each node is associated with a weight <CPU> to indicate the CPU execution time for each object and each edge is associated with a weight <DataCount> to indicate the total data amount that needed to be transmitted between two nodes.

In order to get an accurate WORG of the application, we first construct an initial Object Relation Graph (ORG) of the application by static point to analysis [8] and then we use offline profiling to assign weights to the ORG. Our WORG constructing algorithm is based on the method proposed in [7]. For convenience, we will use a simple example to work through our WORG constructing algorithm.

An Example. Figure 2 shows an example of java application. The example contains two main classes: an Account class describes a bank account and a Bank class describes a bank that processes those bank accounts. In addition, the Bank class

creates a Vector object to save the accounts. The main function of the Bank class create one bank object and two account objects, performing operations on them through method calls. Our target is to construct a graph to represent these objects and the interaction between them.

```
class Account{
      int id;
      String name;
      int money;
      public Account(int id,String name,int money){
          this.id = id;
          this.name = name;
          this.money = money;
      }
      public void setMoney(int money){
          this.money = money
      }
      ......................
   }
public class Bank{
   Vector accounts = new Vector();
   String name;
   public Bank(String name){
           accounts = new Vector();
           this.name = name;
   }
   public void addNewAccount(Account e){
           accounts.add(e);
   }
   public Account getAccount(int accountID){
           ..............
   }
   public void saveMoney(int accountID, int money){
           this.getAccount(accountID).setMoney(
           this.getAccount(accountID).checkMoney()+money);
   }
   public static void main(String args[]){
                Bank bank = new Bank("ICBC");
                Account A1 = new Account(1,"jane");
                Account A2 = new Account(2,"anne");
                bank.addNewAccount(e1);
                bank.addNewAccount(e2);
                bank.savaMoney(2, 20000);
                ..........................................
        }
}
```

Fig. 2. An Example of java application

Static Point-to Analysis. We use the Soot analysis framework [8] to perform Point-to Analysis of the application. Soot is a Java optimization framework which provides tools and APIs for analyzing and transforming java byte code. By using Soot's build-in Pointer Analysis Research Kit (Spark) and call graph analysis, we can construct an initial ORG of the example application, which is showed in Figure 3.

The OGR has five nodes and each node is annotated with S_ or D_ prefix to indicate static or dynamic objects. The entry point of the ORG is S_Bank which contains the main function. The main function first creates a Bank Object and two Account Objects which are represented by D_Bank,D_Account_1, D_Account_2 respectively. Meanwhile three edges are added from S_Bank to D_Bank, D_Account_1, D_Account_2 to indicate the creation and invocation interaction between them. In addition, the main function invokes D_Bank's "addNewAccount" method to add D_Account_1 and D_Account_2, thus two edges are added from

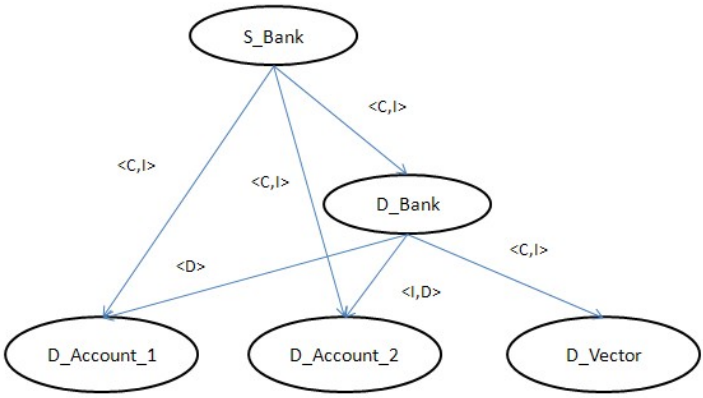

Fig. 3. Initial ORG of the Example

D_Bank to D_Account_1 and D_Accout_2 for the DataAccess interaction. Then, the main function calls the D_Bank's "saveMoney" function to set D_Account_2's "money" attribute which will in turn invocate D_Account_2's "setMoney" method, thus the edge between D_Bank and D_Account_2 is labeled with I (Invocation).

Offline Profiling. After constructing the initial ORG, we perform Offline Profiling to assign weights to the nodes and edges of the ORG to form the Weighted ORG (WORG). We assign each node with a <CPU> weight to indicate the execution time of the corresponding object, and each edge with a <DataCount> weight to indicate the total data amount that needs to be transmitted between two nodes. In order to estimate the <CPU> and <DataCount> metrics, we combine Soot's flow analysis framework with the byte code rewriting to add instrument code to collect the <CPU> and <DataCount> metrics of each node and edge.

2.3 Graph Partitioning

The task of Graph Partitioning module is to work out the optimal assignment (locally execution or remote execution) for each node of the constructed WORG. We formulate the Graph Partitioning problem as a 0-1 Integer Programming with Probability (IPP) problem and adopt Branch and Bound algorithm to solve the IPP problem. Section 2.3.1 and 2.3.2 present the details of IPP and Branch and Bound, respectively.

Problem Formulation

Input. The IPP problem takes the following inputs:

1) G = <V, E>: The WORG of the application, with each node Vi associated with weight <cpui> and each edge with weight <datacounti> .
2) F(b): The Probability Distribution Function of the bandwidth b;
3) $<E_C, T_C, C_C, P_C>$: The application specific constrains, which indicates the total energy consumption, execution time and communication cost should not exceed EC, TC and CC respectively with guaranteed probability PC.

4) B: Current bandwidth.
5) SL: The set of the nodes that needs to be run locally (e.g., the objects that process user interaction).

Target. With a given input, we can compute the total energy consumption, execution time and cost of the application by the following equations:

$$Energy(G) = \sum x_i * E(i) + \sum |x_i - x_j| * E(e_{ij}) . \quad (1)$$

$$Time(G) = \sum_{1 \le i \le n} x_i * t_{nli} + (1 - x_i) * t_{nsi} + \sum_{1 \le i,j \le n} | x_i - x_j | * t_{ij} . \quad (2)$$

$$Cost(G) = \sum_{1 \le i,j \le n} | x_i - x_j | * c_{ij} . \quad (3)$$

x_i indicates the assignment of each node: $x_i = 1$ means local execution of the node while $x_i = 0$ means remote execution.

E(i) is the energy consumption of node *i* when i is running locally, which can be computed through the following equation:

$$E(i) = < cpu_i > * P_{cpu} . \quad (4)$$

$<cpu_i>$ is the <*CPU*> weight of node i and P_{cpu} is the power of CPU.

$E(e_{ij})$ is the energy consumption for data transmission between node i, j when they are not running together. $E(e_{ij})$ can be computed by equation (5).

$$E(ij) = < datacount_{ij} > / b * P_{wi-fi} . \quad (5)$$

$<datacount_{ij}>$is the <DataCount> weight of the edge that connect node i and j.

t_{nli} and t_{nsi} is the execution time of node *i* when running locally and remotely, which can be computed by equation (6) and (7) , respectively:

$$t_{nli} = < cpu_i > . \quad (6)$$

$$t_{nsi} = t_{nli} / k . \quad (7)$$

k indicates that the server is k times faster than local devices.

t_{ij} is the transmission time for the communication data between node *i* and *j* :

$$t(ij) = < datacount_{ij} > / b . \quad (8)$$

c_{ij} is the money cost for transmitting data between node *i* and *j* :

$$c(ij) = < datacount_{ij} > * c . \quad (9)$$

c is the money taken for transmitting 1bit data.

The task of IPP problem is to assign each node of the WORG with the optimal x_i, to make the execution time, energy consumption and cost fulfill the given constrains with given probability confidence:

$$P\{Energy(G) < E\} > P_c . \quad (10)$$

$$P\{Time(G)\} < T\} > P_c \cdot \tag{11}$$

$$P\{Cost(G)\} < C\} > P_c \cdot \tag{12}$$

Branch and Bound. We perform Branch and Bound algorithm to solve the IPP problem. First, in order to reduce computational complexity, we simplify the constraints defined in IPP (denoted as constraint A) problem as follows (denoted as constraint B):

$$Energy(G)_b < E \cdot \tag{13}$$

$$Time(G)_b < T \cdot \tag{14}$$

$$Cost(G)_b < C \cdot \tag{15}$$

b is the critical bandwidth that meets $P\{B>=b\}>P_c$ and $Energy(G)_b$, $Time(G)_b$, $Cost(G)_b$ represents the energy consumption, execution time and cost of a particular partitioning scheme respectively when bandwidth is b.

Constraint B is an approximate to constraint A: if a partitioning scheme can fulfill constraint B, it will fulfill constraint A with a high probability.

Then we conduct Branch and Bound on the simplified IPP. We first transform the WORG to a DAG, perform topologic sort on the DAG, and then branch form the first node in topologic sort. We use depth-first search to traverse the search tree, and for every encountered nodes, compute the $Energy(G)_b$, $Time(G)_b$, $Cost(G)_b$, compare it with E, T, C and the current minimal energy $MinEnergy$. If $Energy(G)_b$, $Time(G)_b$, $Cost(G)_b$ don't fulfill the constraints or $E > MinEnergy$, the sub tree of the node will be cut and the search back traverses to the parent node. After finishing searching, we will get the optimal partitioning that fulfills the given constrains.

3 Evaluation

This section presents the experimental results of our BAAP algorithm. We first show the drawbacks of those partitioning algorithms that using static bandwidth, then we present the performance of our BAAP algorithm and compare it with the static bandwidth based application partitioning algorithm.

3.1 Drawbacks of Static Bandwidth Based Partitioning

To show the drawbacks of static bandwidth based partitioning algorithms that using a static bandwidth to compute the communication cost, we perform the following experiment:

1) With a given WOGR and bandwidth B(100kb/s in our experiment), work out the optimal partition scheme (denoted as A) that minimize the energy consumption of the WORG through Branch and Bound algorithm.
2) Change the bandwidth (from 10kb/s to 100kb/s), work out the corresponding optimal partition scheme (denoted as B), and compare the energy consumption of partition scheme A and that of scheme B at different bandwidth.

We perform the experiment on three random generated graphs that simulating the WORG of actual applications. The properties of the graphs are listed in Table 1 and the experimental results are showed in Figure 4.

Table 1. Graph Properties

Parameter	Setting		Description
Graph size	Graph1	<10,43>	The number of node and edge
	Graph2	<15,103>	
	Graph3	<30,416>	
Node weight	1-720(s)		Execution time of a node
Edge weight	1-2000(kb)		Interaction data count
Cpu power	480(mw)		Cpu's power
Wi-Fi power	880(mw)		The power of wi-fi interface
cost	0.01(RMB)		Money cost for transmitting 1 bit
k	5		Speed up of server

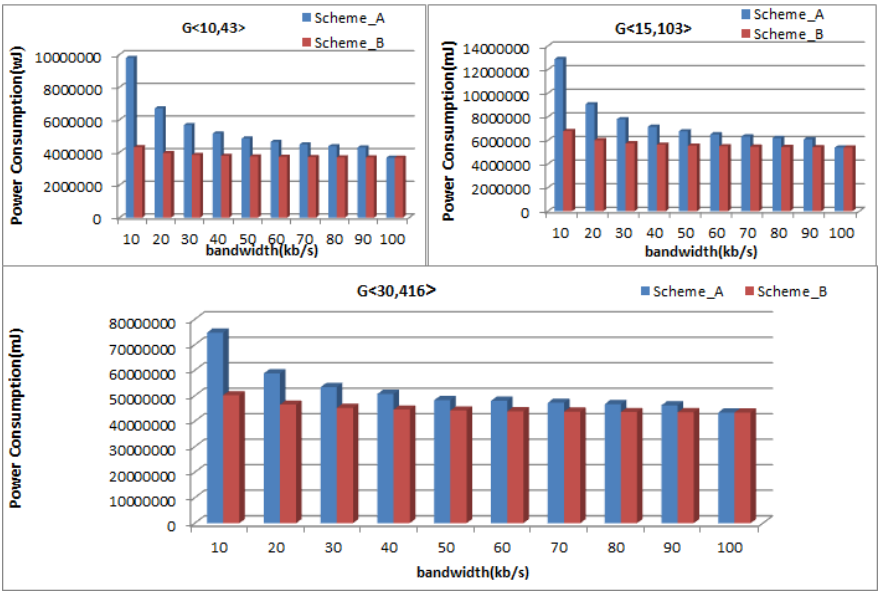

Fig. 4. Comparison of energy consumption between scheme A and B with different bandwidth

From Figure 4 we can see that with the bandwidth decreases, the energy consumption of scheme A increases and the energy consumption of scheme A is much larger than that of scheme B, especially when the bandwidth is low. The experiment results reflect that the static bandwidth based partitioning scheme cannot adapt to dynamic bandwidth: the optimal partitioning at a bandwidth may consume much more energy at another bandwidth. Therefore a bandwidth aware partitioning scheme is needed.

3.2 Performance of BAAP

To evaluate the performance of our BAAP algorithm, we work out the partitioning scheme (denoted as BAAP) through our BAAP algorithm of the three Graph that we used in 3.1 and compare the energy consumption of BAAP with that of scheme A and scheme B in 3.1 under different bandwidth(from 10kb/s to 100kb/s).

Table 2. Experiment setting

Parameter	Setting	Description
B	100	Current bandwidth
b	10	Critical bandwidth with 90% confidence
E	1.2 *E(b)	E(b) is the minimal energy with bandwidth=b
C	1.2*C(b)	C(b) is the minimal cost with bandwidth=b
T	1.2*T(b)	T(b) is the Minimal time with bandwidth=b
F(B)	B/100	B obeys the uniform distribution between 0-100

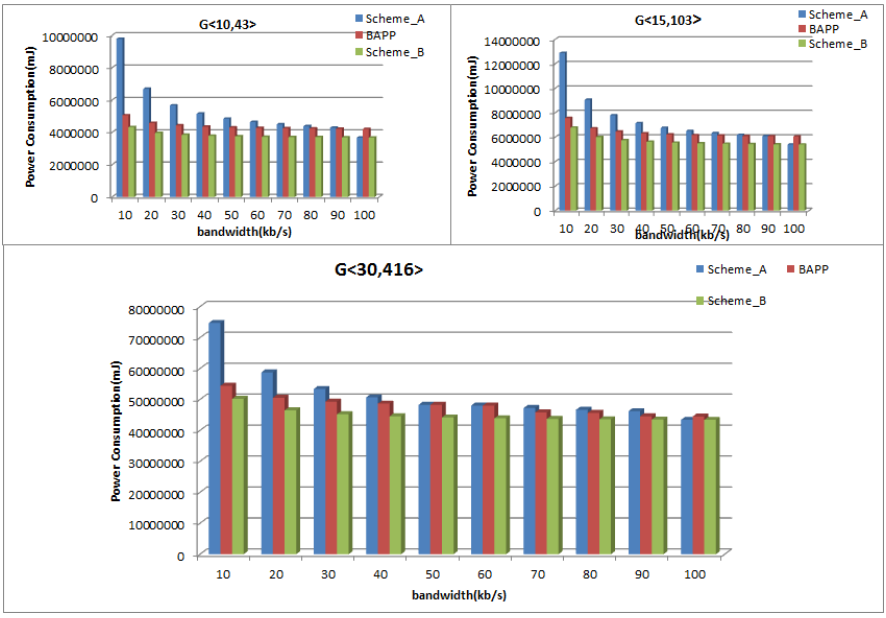

Fig. 5. Comparison of energy consumption among scheme A scheme B and BAAP with different bandwidth

Table 2 lists the constraints and parameters we set in this experiment for BAAP and Figure 5 shows the experimental results.

From figure 5, we can see that BAPP consumes less energy than Scheme_A, and thus outperforms those static bandwidth based partitioning schemes. By setting constrains to the possible partitioning schemes, BAPP can exclude those partitioning schemes that may consume much energy at low bandwidth, though they may be the optimal scheme at current bandwidth.

4 Conclusion

This paper proposed a Bandwidth Aware Application Partitioning Scheme (BAAP) for computation offloading to save energy of mobile devices. BAAP models the bandwidth as a random variable and formulates the application partition as a 0-1 Integer Programming with Probability (IPP) problem. Then, BAAP adopts Branch and Bound algorithm to solve the problem. Experimental results show our BAAP algorithm outperforms static bandwidth based partitioning schemes and can greatly reduce energy consumption while satisfying the cost and time constraints with guaranteed confidence probabilities regardless of different network bandwidth.

Acknowledgments. This work was supported by the Research Fund of the State Key Laboratory of Software Development Environment under Grant No. BUAA SKLSDE-2010ZX-13, the National Natural Science Foundation of China under Grant No. 60873241, the Fund of Aeronautics Science granted No. 20091951020, the Program for New Century Excellent Talents in University under Grant No. 291184.

References

1. Yang, K., Ou, S.: On Effective Offloading Services for Resource-Constrained Mobile Devices Running Heavier Mobile Internet Applications. IEEE Communications Magazine 46(1), 56–63 (2008)
2. Cuervo, E., Balasubramanian, A., Cho, D.: MAUI: Making Smart Phone Last Longer with Code Offloading. In: Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services, San Francisco (2010)
3. Kemp, R., Palmer, N., Kielmann, T., Bal, H.: Cuckoo: a Computation Offloading Framework for Smart phones. In: Proceedings of the 2nd International ICST Conference on Mobile Computing, Application and Services, Santa Clara (2010)
4. Diaconescu, R.E., Wang, L., Mouri, Z., Chu, M.: A Compiler and Runtime Infrastructure for Automatic Program Distribution. In: 19th IEEE International Parallel and Distributed Processing Symposium, Denver (2005)
5. Diaconescu, R.E., Wang, L., Franz, M.: Automatic distribution of java byte-code based on dependence analysis. Technical Report, School of Information and Computer Science, University of California (2003)
6. Li, Z., Wang, C., Xu, R.: Computation Offloading to Save Energy on Handheld Devices: A Partition Scheme. In: Proceeding of the 4th ACM International Conference on Compilers, Architecture and Synthesis for Embedded Systems, Atlanta (2001)
7. Wang, L., Franz, M.: Automatic Partitioning of Object-Oriented Programs with Multiple Distribution Objectives. Technical Report, Donald Bren School of Information and Computer Science, University of California, Irvine (2007)
8. Soot, `http://www.sable.mcgill.ca/soot/`