Journal of Southeast University (English Edition) Vol. 23, No. 2, pp. 216 – 220 June 2007 ISSN 1003—7985

Dynamic software allocation algorithm for saving power in pervasive computing

Han Songqiao Zhang Shensheng Zhang Yong Cao Jian

(Department of Computer Science and Engineering, Shanghai Jiaotong University, Shanghai 200240, China)

Abstract: A novel dynamic software allocation algorithm suitable for pervasive computing environments is proposed to minimize power consumption of mobile devices. Considering the power cost incurred by the computation, communication and migration of software components, a power consumption model of component assignments between a mobile device and a server is set up. Also, the mobility of components and the mobility relationships between components are taken into account in software allocation. By using network flow theory, the optimization problem of power conservation is transformed into the optimal bipartition problem of a flow network which can be partitioned by the max-flow min-cut algorithm. Simulation results show that the proposed algorithm can save significantly more energy than existing algorithms.

Key words: power aware; software allocation; code mobility; graph theory; pervasive computing

In pervasive computing[1], mobile devices have the potential to become powerful tools to access information and applications from anywhere at any time. However, as mobile devices become more widely used for more advanced applications, their power limitations are becoming more apparent. Although exponential improvements have occurred in hardware components, such improvements have not occurred in battery technology, and we do not anticipate any significant changes in the future[2].

The wireless connectivity to a server provides a great chance to offload computation from a mobile device to a powerful server for saving the energy of the mobile device. In the fields of resource assignments and task scheduling, most researchers focus on the improvement of service performance and workload balance among machines, and little research effort is dedicated to conserving power[3]. Recently, the idea of using remote processing[4] to save power has been explored by some significant simulations and experiments[5-6]. But these approaches are static component assignment approaches which cannot adapt to environmental changes. Besides, remote processing only allows component movement from the client to the server rather than bidirectional component movement which is necessary in pervasive computing. Thus, based on network flow theory, this paper presents an optimal and dynamic software allocation algorithm to conserve mobile device energy.

1 Power Consumption Model

1.1 Power consumption of mobile devices

Suppose a resource-constraint mobile device, called client, and a resource-rich computer, called server, are connected by a wireless network. A component based application runs on the two hosts. From a software perspective, we classify the power consumption of the mobile device into three types of costs: computation cost, communication cost and migration cost.

Computation cost is the cost incurred by the execution of a component or an application. For component c_i, its computation cost can be calculated as

$$\left.\begin{aligned} e_c(c_i) &= \alpha_c(c_i)p_c t_c(c_i) \qquad c_i \in S_c \\ e_c(c_i) &= \alpha_s(c_i)p_i t_s(c_i) \qquad c_i \in S_s \end{aligned}\right\} \tag{1}$$

where S_c and S_s are component sets on the client and the server, respectively; $\alpha_c(c_i)$ and $\alpha_s(c_i)$ are the power consumption coefficients of c_i on the client and the server, respectively; p_c is the mean power consumption rate when the mobile device executes and p_i is the counterpart when the mobile device is idle; $t_c(c_i)$ and $t_s(c_i)$ are the execution time of c_i on the client and the server, respectively. Then the computation cost of the application can be calculated as

$$E_c = \sum_{c_i \in S_c} \alpha_c(c_i)p_c t_c(c_i) + \sum_{c_i \in S_s} \alpha_s(c_i)p_i t_s(c_i) \tag{2}$$

The communication cost is incurred by sending and receiving data from c_i on the client to c_j on the server. It can be calculated as

Received 2006-10-30.

Foundation items: The National Natural Science Foundation of China (No. 60503041), the Science and Technology Commission of Shanghai International Cooperation Project (No. 05SN07114).

Biographies: Han Songqiao (1975—), male, graduate; Zhang Shensheng (corresponding author), male, doctor, professor, sszhang@sjtu.edu.cn.

$$\left.\begin{aligned} e_s(c_i, c_j) &= \frac{s(c_i, c_j)p_s}{\lambda b_s} \\ e_r(c_i, c_j) &= \frac{s(c_j, c_i)p_r}{\lambda b_r} \end{aligned}\right\} \tag{3}$$

where $s(c_i, c_j)$ is the size of data transferred from c_i to c_j; p_s and p_r are the mean power consumption rates when the client sends and receives data, respectively; λ is a network coefficient; b_s and b_r are available sending and receiving network bandwidths, respectively. Then the total communication cost between the two hosts can be calculated as

$$E_{sr} = \sum_{c_i \in S_c, c_j \in S_s} \left[\frac{s(c_i, c_j)p_s}{\lambda b_s} + \frac{s(c_j, c_i)p_r}{\lambda b_r} \right] \tag{4}$$

The component migration cost is induced by moving c_i from the client to the server, and vice versa. It can be calculated as

$$\left.\begin{aligned} e_{c\to s}(c_i) &= \frac{s(c_i)p_s}{\lambda b_s} \\ e_{s\to c}(c_i) &= \frac{s(c_i)p_r}{\lambda b_r} \end{aligned}\right\} \tag{5}$$

where $s(c_i)$ is the size of c_i. The total migration cost of the application is expressed as

$$E_{cm} = \sum_{c_i \in S_{c\to s}} \frac{s(c_i)p_s}{\lambda b_s} + \sum_{c_i \in S_{s\to c}} \frac{s(c_i)p_r}{\lambda b_r} \tag{6}$$

where $S_{c\to s}$ and $S_{s\to c}$ are the sets of components moving from the client to the server and vice versa, respectively.

Thus, when the application runs N cycles, the total energy consumption is calculated as

$$E = NE_c + NE_{sr} + E_{cm} \tag{7}$$

Note that there are also other estimation methods of energy consumption[7]. But the proposed algorithm is independent of estimation methods.

1.2 Problem formulation

Software architecture of an application can be represented by an undirected graph where vertex v_i represents component c_i and edge e_{ij} represents the interaction between components c_i and c_j. To denote the three types of costs, we add the weights to the vertices and the edges in the graph, called the cost graph. The weight of vertex v_i is a triple set $(C_m(c_i), C_c(c_i), C_s(c_i))$, where $C_m(c_i)$ denotes the migration cost of c_i, $C_c(c_i)$ and $C_s(c_i)$ are its computation costs on the client and the server, respectively. The weight $C_c(c_i, c_j)$ of edge e_{ij} is equal to the communication cost between c_i and c_j.

Thus, the problem of the optimal software allocation on the client and server for minimizing power consumption can be transformed into the optimal bipartition problem of the cost graph, subject to the following minimal power consumption formula:

$$\min \Big(\sum_{c_i \in S_c} C_c(c_i) + \sum_{c_j \in S_s} C_s(c_j) + \sum_{e(c_i, c_j) \in S_e} C_c(c_i, c_j) + \sum_{c_k \in S_m} C_m(c_k) \Big) \tag{8}$$

where S_e is the set of all edges connecting components on different hosts, and S_m is the set of migrating components.

2 Software Allocation Algorithm

2.1 Characteristics of software architecture

As two major elements in software architecture, components and connectors have some attributes that can influence software allocation between machines.

According to component mobility[8], components can be classified into two categories: fixed components and mobile components. The former are located on a machine at runtime while the latter can move between machines during execution. If component c_i is fixed, $C_m(c_i) = \infty$. If c_i cannot run on the client or the server, $C_c(c_i) = \infty$ or $C_s(c_i) = \infty$.

Given components c_i and c_j connected by a connector, the movement of one component may affect the location of another component. From this perspective, connectors or edges have three major types: link, pull and stamp.

- Link type means c_i and c_j can be located on different hosts. When c_i moves from one host to another, c_j will not move away or move along with c_i. The weight of the link type of edge is equal to $C_c(c_i, c_j)$.
- Pull type means c_i and c_j must be located on the same host. When c_i moves from one host to another, c_j must move along to the location of c_i. The weight of the pull type of edge is infinite.
- Stamp type means c_i and a type instance of c_j maybe located on the same host. When c_i moves from one host to another which has a type instance c_j' of c_j, c_i cuts off the interaction with c_j and resumes setting up an interaction with c_j' on the new host. The weight of the stamp type of edge is zero.

Suppose that an application is comprised of four components, called a, b, c and d. Its software architecture with connector types is shown in Fig. 1. Fig. 2 shows the cost graph of the application. Initially, d stays on the client while others stay on the server.

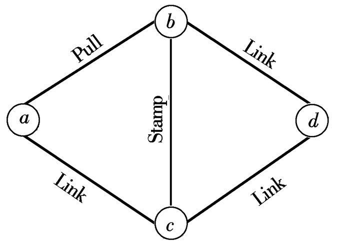

Fig. 1 Software architecture with connector types

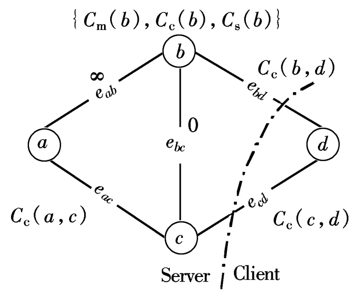

Fig. 2 Cost graph with initial component allocation

2.2 Software allocation algorithm

The software allocation algorithm includes two parts. First, the cost graph is transformed into a flow network. Then the flow network is partitioned by a max-flow min-cut algorithm.

1) Cost graph transformation

In graph theory, a flow network is a directed graph with a source vertex S and a sink vertex T. Each edge has a capacity which is equal to the max allowed flow on the edge. Based on network flow theory, the cost graph $G = (V, E)$ can be transformed into an equivalent flow network $G' = (V', E')$ according to the following steps:

Step 1 Each node $v \in V$ in graph G is transformed into a node $v' \in V'$ in network G'.

Step 2 For each edge e_{ij} between nodes v_i and v_j in G, two edges e'_{ij} and e'_{ji} between nodes v_i' and v_j' are added to G', and their capacities are equal to the weight of e_{ij}.

Step 3 Add a source node S and a sink node T to G'. The nodes S and T represent the client and server machine, respectively.

Step 4 For each node other than S and T, add an edge from S to that node and an edge from that node to T. The capacities of the two edges may be different due to different locations of the nodes. Let cap(e) be the capacity of edge e, let e_{si} be the edge from S to c_i, and let e_{it} be the edge from c_i to T. The capacities of two edges can be calculated as

$$\left.\begin{aligned}\text{cap}(e_{si}) &= C_s(c_i) + C_m(c_i)\\ \text{cap}(e_{it}) &= C_c(c_i)\end{aligned}\right\}\quad c_i \in S_c \qquad (9)$$

$$\left.\begin{aligned}\text{cap}(e_{si}) &= C_s(c_i)\\ \text{cap}(e_{it}) &= C_c(c_i) + C_m(c_i)\end{aligned}\right\}\quad c_i \in S_s \qquad (10)$$

For example, Fig. 3 shows the flow network that is transformed from the cost graph in Fig. 2.

2) Flow network partitioning

In a flow network G', a feasible flow is a flow originating from the source node and ending at the sink node such that: ① At each intermediate node, the sum of the flows into the node is equal to the sum of the flows out the node; and ② The flow in any branch does not exceed the capacity of that branch. The value

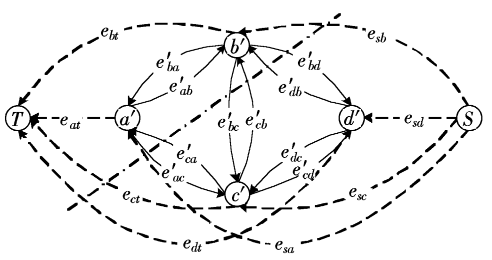

Fig. 3 Flow network transformed from cost graph

of a flow is the sum of the flows out of the source node. Then a maximum flow is a feasible flow whose value is the maximum among all feasible flows. The maximum flow is related to a cut set of the network. A partition of the network creates two disjoint node sets V_s and V_t and $V' = V_s \cup V_t$, $S \in V_s$, $T \in V_t$. A cut edge is an edge (u, v) where $u \in V_s$ and $v \in V_t$. The cut set is the set of all cut edges in G'. In Fig. 3, the cut set denoted by the dot-dashed line is $\{e'_{db}, e'_{cb}, e'_{ca}, e_{sb}, e_{sa}, e_{ct}, e_{dt}\}$. The weight of a cut set is equal to the sum of the capacities of the edges in the cut set.

We associate a component assignment with each cut set such that if the cut set partitions a node into the subset V_s, then the corresponding component is assigned to the client. Similarly, if a node is partitioned into V_t, then the corresponding component is assigned to the server. Thus component assignments and cut sets of the flow network are in one-to-one correspondence. Then we can obtain theorem 1.

Theorem 1 The weight of a cut set in the flow network is equal to the cost of the corresponding software allocation.

Proof A component assignment incurs computation cost, communication cost and migration cost. The cut set corresponding to a component assignment has two types of edges. The first type of edges, such as e'_{db}, e'_{cb} and e'_{ca} in Fig. 3, represent the communication cost between components. The second type of edges are edges incident to S or T, such as e_{sb}, e_{sa}, e_{ct} and e_{dt}. If a node, such as c' in Fig. 3, on the server is assigned to the client, it induces a computation cost on the client and a migration cost. In network G', the edge from the node to T must belong to the cut set and it carries a capacity that is just equal to the sum of the computation cost on the client and the migration cost (see Eq. (10)). Moreover, no other edges that connect the node and S (or T) are included within the cut set. In the case that a node is on the client, the proof procedure is similar. Thus the weight of a cut set accounts for the power consumption of the component assignment. This proves the theorem.

Then the following corollary ensures the soundness of the proposed algorithm.

Corollary 1 The minimum cost of the component assignment is equal to the weight of a min-cut set or the value of max-flow in the flow network G'.

Proof From theorem 1, we can see that the minimum cost of the component assignment should be equal to the weight of a min-cut set in G'. The max-flow min-cut theorem means that the value of max-flow in a flow network is equal to the weight of a min-cut set of the network. Thus the minimum cost of the component assignment is also equal to the value of the max-flow in G'. This proves the corollary.

Thus, we use a max-flow min-cut algorithm, such as the preflow-push algorithm, to partition G', thereby obtaining the optimal component allocation and the minimal energy consumption. This algorithm has $O(N^3)$ time complexity. For example, if the dot-dashed line in Fig. 3 represents a min-cut, the optimal software allocation results are that a and b are assigned to the server, c and d are assigned to the client, and the minimal energy cost is equal to $\text{cap}(e'_{db}) + \text{cap}(e'_{cb}) + \text{cap}(e'_{ca}) + \text{cap}(e_{sb}) + \text{cap}(e_{sa}) + \text{cap}(e_{ct}) + \text{cap}(e_{dt})$.

3 Simulation Results

We have implemented the proposed algorithm and applied the results to partition a game prototype. According to the software architecture of the game, we construct its cost graph of communication amount, as shown in Fig. 4. The edge weight labeled on the edge indicates the communication amount between components within an execution cycle, i. e. $N=1$.

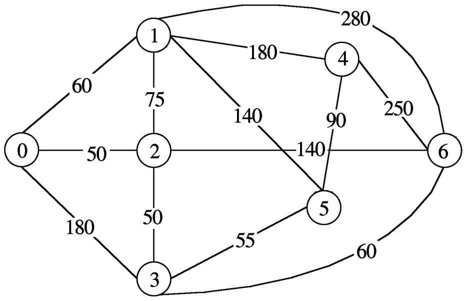

Fig. 4 Cost graph of communication amount

For component c_i, its size $s(c_i)$, execution time $t_c(c_i)$ and $t_s(c_i)$ can be obtained by the profiling approach and are listed in Tab. 1. Suppose that the application is initially located on a mobile device and runs for N cycles. Let network bandwidth $b = 1\ 000$ kbit/s, the network coefficient $\lambda = 0.7$, power consumption coefficients $\alpha_c(c_i) = \alpha_s(c_i) = 1$, power consumption rates $p_i = 1.65$ W, $p_c = 2.4$ W, $p_t = p_s = p_r = 2.2$ W which are borrowed from Ref. [9]. Then the cost graph is transformed into a flow network by using Eqs. (1), (3) and (5). The flow network of energy is shown in Fig. 5, where $\alpha = p_c, \beta = p_i, \gamma = p_t/(\lambda b)$.

Tab. 1 The values of execution parameters of components

Parameters	Nodes						
	C_0	C_1	C_2	C_3	C_4	C_5	C_6
$s(c_i)$/kbit	680	200	360	320	100	150	560
$t_c(c_i)$/s	1	0.10	0.19	0.36	0.08	0.104	0.46
$t_s(c_i)$/s	0.24	0.024	0.04	0.09	0.02	0.026	∞

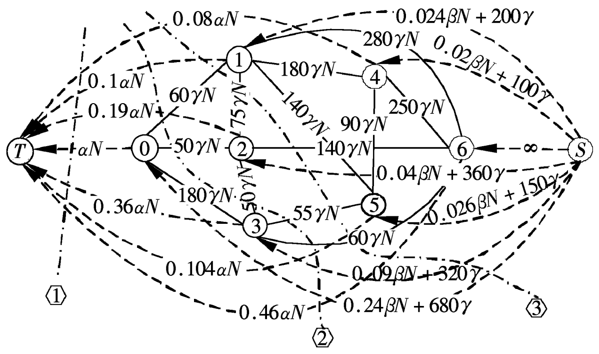

Fig. 5 Flow network of energy

When the system runs for N cycles, the average energy consumption per execution cycle $P = E/N$, where E is the total energy consumption. To validate our algorithm, three software allocation algorithms, including monolithic application on the client (MONO), remote processing (RP) and the proposed algorithm, are compared in terms of power cost. Fig. 6 shows the simulation results where N increases in steps of 1. MONO consumes constant energy with increasing N. RP makes most of components move from the client to the server, resulting in a great increase of migration cost. When N is small, the movement of numerous components makes RP consume more energy than MONO. But with the increase of N, the computation cost is much more than the migration cost. Since the computation cost of RP is in general less than that of MONO, the former consumes less energy than the latter when the system runs for a long time.

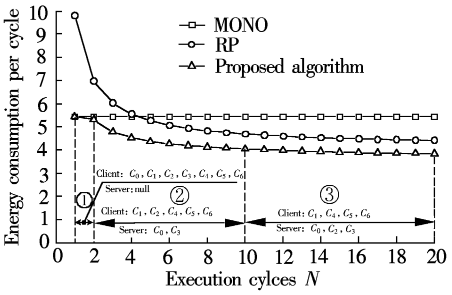

Fig. 6 Energy consumption comparison of different algorithms

The proposed algorithm can dynamically find the optimal software allocations when execution time (or N) changes. When $N=1$, the whole system runs on the client. Our algorithm saves about 45% energy compared to RP. When $2 \leqslant N < 10$, our algorithm moves C_0

and C_3 to the server, which saves more energy than MONO by about 2% to 25% and RP by about 14% to 24%. When $10 \leqslant N \leqslant 20$, our algorithm moves C_0, C_2 and C_3 to the server, which saves more energy than MONO by about 26% to 32% and RP by about 13% to 14%. Moreover, network bandwidth also influences software allocation results. Given $N = 20$, when $b = 100$ kbit/s, our algorithm saves about 39% more energy than RP, and when b = 2 000 kbit/s, our algorithm saves about 43% more energy than MONO. The simulation results show that the proposed algorithm always consumes the least energy.

4 Conclusion

A dynamic software allocation algorithm is proposed to save energy on mobile devices. The algorithm is able to reassign dynamically the appropriate components between machines to minimize energy consumption as the environment changes. Besides the computation cost and the communication cost, the migration cost is also considered in the power consumption model, which is necessary in dynamic software allocation but neglected by the previous static approaches. After analyzing the characteristics of the software architecture, we use network flow theory to reduce an optimization problem of software allocation to an optimal bipartition problem of a cost graph. It is worth emphasizing that the proposed algorithm is a general algorithm that can also allocate an application for conserving network bandwidth and improving response time. As future work, the proposed algorithm would be extended to dynamically allocate an application on multiple machines or devices, such as sensor networks.

References

[1] Weiser M. Some computer science issues in ubiquitous computing [J]. *Communications of the ACM*, 1993, **36**(7): 75 - 84.

[2] Want R, Farkas K I, Narayanaswami C. Energy harvesting and conservation [J]. *IEEE Pervasive Computing*, 2005, **4**(1): 14 - 17.

[3] Ahmed J, Chakrabarti C. A dynamic task scheduling algorithm for battery powered DVS systems [C]//*Proc of IEEE International Symposium on Circuits and Systems*. Vancouver, BC, Canada, 2004: 813 - 816.

[4] Balan R, Flinn J, Satyanarayanan M, et al. The case for cyber foraging [C]//*Proc of the* 10*th Workshop on ACM SIGOPS European Workshop*. Saint-Emilion, France, 2002: 87 - 92.

[5] Rudenko A, Reiher P, Popek G J, et al. The remote processing framework for portable computer power saving [C]//*Proc of the ACM Symposium on Applied Computing*. San Antonio, TX, USA, 1999: 365 - 372.

[6] Li Z, Wang C, Xu R. Computation offloading to save energy on handheld devices: a partition scheme [C]//*Proc of International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*. Atlanta, Georgia, USA, 2001: 238 - 246.

[7] Flinn J, Satyanarayanan M. Managing battery lifetime with energy-aware adaptation [J]. *ACM Transactions on Computer Systems*, 2004, **22**(2): 137 - 179.

[8] Fuggetta A, Picco G P, Vigna G. Understanding code mobility [J]. *IEEE Transactions on Software Engineering*, 1998, **24**(5): 342 - 361.

[9] Li Z, Wang C, Xu R. Task Allocation for distributed multimedia processing on wirelessly networked handheld devices [C]//*Proc of the* 16*th International Symposium on Parallel and Distributed Processing*. Fort Lauderdale, Florida, USA, 2002: 312 - 317.

普适计算中节约电量的动态软件部署算法

韩松乔　张申生　张　勇　曹　健

（上海交通大学计算机科学与工程系，上海 200240）

摘要：为了节约移动设备的电量消耗，提出了一种适合于普适计算环境的动态软件部署算法. 综合考虑了软件组件的计算、通信和移动所消耗的费用，建立了一个在移动设备和服务器间组件部署的电量消耗模型. 在软件部署中同时也考虑了组件的移动性和组件间的移动关系. 利用网络流理论，将节约电量的最优化问题转化为一个流网络的最优分割问题，而后者可采用最大流最小切割算法实现最优切分. 实验结果表明提出的算法比现有算法能够节约更多的电量.

关键词：电量感知；软件部署；代码移动；图论；普适计算

中图分类号：TP311