Energy Optimization With Dynamic Task Scheduling Mobile Cloud Computing

Yibin Li, Min Chen, *Senior Member, IEEE*, Wenyun Dai, *Student Member, IEEE*, and Meikang Qiu, *Senior Member, IEEE*

***Abstract*—The smartphone is a typical cyberphysical system (CPS). It must be low energy consuming and highly reliable to deal with the simple but frequent interactions with the cloud, which constitutes the cloud-integrated CPS. *Dynamic voltage scaling* (DVS) has emerged as a critical technique to leverage power management by lowering the supply voltage and frequency of processors. In this paper, based on the DVS technique, we propose a novel *Energy-aware Dynamic Task Scheduling* (EDTS) algorithm to minimize the total energy consumption for smartphones, while satisfying stringent time constraints and the probability constraint for applications. Experimental results indicate that the EDTS algorithm can significantly reduce energy consumption for CPS, as compared to the critical path scheduling method and the parallelism-based scheduling algorithm.**

***Index Terms*—Cloud-integrated cyberphysical system (CPS), CPS, dynamic scheduling, dynamic voltage scaling (DVS).**

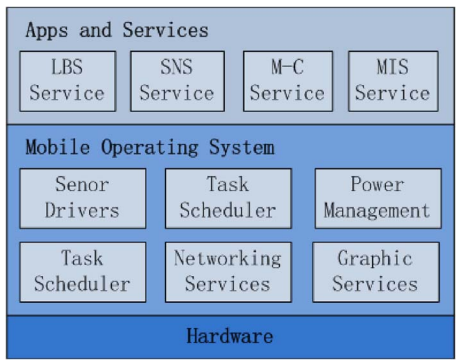


Fig. 1. Typical architecture for mobile device.

I. Introduction

THE smartphone, as a typical *cyberphysical system* (CPS), gains increasing attention nowadays. With the popularity of embedded systems and the unprecedented development of high integrated chips, increasing functions are moving onto smartphones. Meanwhile, smartphones, as the most widely used mobile devices, are facing more severe challenges than others. The ever-increasing demands in rich interactive apps and service, such as location-based service, social networking service, mobile cloud service, and mobile information service, have severely aggravated the energy consumption problem for smartphones. Furthermore, to cater to public taste, manufacturers produce smartphones with increasing large-screen displays. That also increases the burden of batteries.

Meanwhile, with the development of cloud computing, mobile cloud computing (MCC), which is a typical cloud-integrated CPS, becomes a new trend, which is using rich cloud resources to make up the limitations of smartphones. Many researches focus on how to achieve the seamless connection between mobile and cloud [1], but they ignore an important aspect, which is smartphones have limited power not only the computational resource.

To support the connection between mobile and cloud, low-power microprocessors are required to process these simple but frequent tasks in smartphones [2]–[6]. To enhance energy efficiency and process various tasks with different performance requirements, high-end mobile devices are designed as heterogeneous embedded systems, which integrate multiple processors with distinct processing power, such as PowerVR Series7XT GPU family from Imagination. Multiprocessors can offer greater computation per unit of power, leading to longer battery life [7], but it is still critical to investigate tighter energy budget strategies to guarantee functionalities of mobile devices.

Miniature is another vital feature of current CPS, particularly smartphones. Therefore, there is an inherent conflict between the miniaturization and multifunction of these devices and the sustainable usage of their batteries. Most apps on smartphones are not delay tolerant, and their acceleration is often at higher expenses of energy consumption. In order to balance performance and power consumption for these apps, smartphones are usually designed with *dynamic voltage scaling* (DVS) by integrating static CMOS logic into microprocessors [8], [9]. DVS is a powerful technique to reduce energy consumption and is widely employed in various embedded systems [10], [11]. With the aid of this technology, different performance levels for apps can be achieved by adjusting the operating frequency of processors [12], [13].

Currently, the architecture of most mobile devices can be abstracted into three layers, as shown in Fig. 1. The bottom layer is the hardware, which is also known as the physical

Manuscript received February 2, 2015; revised May 1, 2015; accepted May 6, 2015. Date of publication June 25, 2015; date of current version March 10, 2017. The work of Y. Li was supported in part by the International Science and Technology Cooperation Program of China under Grant 2011DFR20090 and Grant 2014DFR70730, and in part by the Public Science and Technology Research Funds Projects of Ocean under Grant 201205036-04. The work of M. Qiu was supported in part by the National Science Foundation under Grant 1457506. M. Chen was supported in part by NSFC under Grant 2011CB302505 and by GuangDong Foundation Team under Grant 201001D0104726115.

Y. Li is with the School of Computer Science and Technology, Shandong University, Jinan 255049, China (e-mail: liyibing@sdu.edu.cn).

M. Chen is with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: minchen@ieee.org; minchen2012@hust.edu.cn).

W. Dai and M. Qiu are with the Department of Computer Science, Pace University, New York, NY 10038 USA.

Digital Object Identifier 10.1109/JSYST.2015.2442994

system. Then, the second layer is the mobile operating system, which has two main functions. The first one is supporting the hardware below, and the second one is providing interface to the upper layer, which is the apps and services layer. The apps and services layer directly offers apps and services to users.

In MCC, as a cloud-integrated CPS, the mobile must have lots of frequent interactions to the cloud, such as requesting resources, sending computational request, and receiving results. These tasks are simple but really power consuming under wireless environment, because they are too frequent. To achieve the perfect performance, we need to solve two problems. The first one is making the energy cost as low as possible, while still maintaining the performance. The second one is making it highly reliable, which represents that the task must be successfully completed at a high probability. As a result, we use a time constraint and a probability constraint to analyze these problems.

We devise an algorithm that utilizes the results from a static scheduling algorithm and attempts to aggressively reduce energy consumption, while satisfying the time and probability constraints. Then, we simulate an Android environment to evaluate our algorithm. The experimental results show that compared to the critical path scheduling method and the parallelism-based scheduling approach, our online scheduling mechanism can reduce total energy consumption by 23.1% and 34.2% on average, respectively, while meeting the given timing constraints.

The major contributions of this paper are threefold as follows:

1) We propose a dynamic scheduling algorithm for dealing with the runtime variations.
2) We use a critical-path-based static scheduling algorithm *Data Flow Graph Critical Path* (DFGCP) to obtain the near-optimal solution, which meets the time and probability constraints.
3) We propose an optimal algorithm *Critical Path Assignment* (CPA) for the critical path with a dynamic programming approach.

II. Related Work

In the past few years, numerous methodologies for low-power CPS design have been proposed at operating system level and architecture level. A scheduler was proposed in [7] to monitor workloads for systems and adaptively schedule real-time tasks, while considering the worst case CPU demands. Through modification of the real-time scheduler and task management services in operating systems, this scheduler can boost system performance and save power consumption for heavy workloads and critical tasks. Targeting multimedia applications, the authors in [14] proposed a soft real-time CPU scheduler for mobile devices to reduce energy consumption. While these studies focus on independent tasks, we consider dependencies and real-time constraints between tasks.

DVS techniques can be used for tackling runtime variation, while considering task dependencies. For example, Gruian [15] applied a stochastic DVS technique on hard real-time systems by taking into account task dependencies. Depending on the probability distribution of the execution conditions for tasks, Lorch and Smith [16] proposed an approach to modify scaling algorithms, while maintaining their performance. However, these methods assume task priorities and estimate CPU requirements offline, which are not suitable for practical study. Consequently, we propose a two-phase scheduling algorithm, i.e., *Energy-aware Dynamic Task Scheduling* (EDTS), which schedules tasks online based on the static scheduling results of an initial scheduling.

The energy consumption is the most important constraint when scheduling dependent tasks. Energy-aware static scheduling is usually based on the information of the average case or worst case task execution estimation [17]. At runtime, the real execution time and energy consumption may exhibit high variations [12], due to process variability, physical faults, and voltage/frequency changes. In our model, we expect that each core in the same processor can adjust its voltage and frequency independently.

There are some other researches related on energy-aware scheduling algorithms. For example, Xu *et al.* [18] proposed a job power-aware scheduling mechanism to reduce HPC's electricity bill without degrading the system utilization. However, our research is focused on energy-aware scheduling algorithms on mobile devices. Santinelli *et al.* [19] explored how to efficiently reduce the power consumption of real-time applications with constrained resource. Mei and Li proposed a new algorithm called Energy-Aware Scheduling by Minimizing Duplication [20], which considers the energy consumption and the makespan of applications. In our research, we consider the time constraint and the performance as well, in addition to the energy constraint.

Wang *et al.* [21] presented an approach for variable partitioning and instruction scheduling to maximally exploit the benefits. Their approach was built on a graph model, which strives to capture both performance and power demands. Nevertheless, this approach was based on multiple memory architecture, while current mobile devices are single memory. Some other researchers used genetic algorithm to achieve the optimization of memory [22], [23].

In [24], the authors jointly presented a host of runtime and compilation techniques to conceal the heterogeneity of smartphones from developers. By investigating various features of HTC and Apple, Li *et al.* [25] pointed out that the most significant challenge of reuse in smartphones is the design of software to accommodate heterogeneity of these devices. However, our work focuses on using a dynamic programming task scheduling technique to reduce energy consumption for smartphones with DVS enabled.

There are some other researches related to MCC and cloud-based CPS [26], [27]. Fahim *et al.* [28] analyzed an environment in which computational offloading is adopted among mobile devices. Yang *et al.* propose a framework to provide runtime support for the dynamic computation partitioning and execution of the application. This framework not only allows single-user partitioning but also supports sharing computation instances among multiple users [29]. The main aim of these researches is to build an environment to allow smartphones easily being connected to the cloud. However, they do not consider the process inside smartphones and ignore the importance of the efficiency of smartphones themselves.

In our previous research [30]–[32], we proposed a highly efficient algorithm, which utilizes the results from a static scheduling algorithm and aggressively reduces energy consumption.

In this paper, we improve our algorithm by considering the probability constraint.

III. Basic Models and Algorithms

A. DFG

In general, the tasks in smartphone applications are not standalone. A certain number of tasks will have precedence relationships due to the different functionality of each task and communications between them. We use a *directed acyclic graph* (DAG) to model the precedence constraints of smartphone applications.

A *data flow graph* (DFG) $G = \langle U, ED, T, E, W \rangle$ is a node-weighted DAG, where $U = \langle u_1, \ldots, u_i, \ldots, u_N \rangle$ is a set of task nodes; $ED \subseteq U \times U$ is an edge set that defines the precedence relations among nodes in U. For example, an edge $e(u \to v)$ in the graph indicates that task v cannot be executed until task u completes. T and E are sets of execution time and energy consumption for all nodes in U, respectively. W is a set of communication cost between tasks.

The execution time T of a task can be profiled by *average case execution time* (ACET) or *worst case execution time* (WCET) when the task is executed on a processor core. We assume that the WCET and ACET of a task are always measured at the highest voltage level (i.e., with fastest speed). Our approach uses ACET for the static scheduling. An edge $e \in ED$ is associated with a weight w that represents the worst case communication cost between two dependent tasks, when they are scheduled on two different processors. Generally, the communication cost between two tasks is negligible, when they are executed on the same processor. There is a timing constraint TC for the whole task graph, which defines the time bound to finish executing the entire task graph.

B. Energy Model

The dynamic power consumption ($P_{\rm AC}$) of CMOS circuits integrated in smartphones is calculated by

$$P_{\rm AC} = C_{\rm eff} V_{dd}^2 f \tag{1}$$

where V_{dd} is the supply voltage, f is the operating frequency, and $C_{\rm eff}$ is the effective switching capacitance. DVS reduces dynamic power consumption according to quadratic dependence on voltage.

The frequency f is represented as

$$f = \frac{(V_{dd} - V_{\rm th})^{\alpha}}{k V_{dd}} \tag{2}$$

where $V_{\rm th}$ represents the threshold voltage, and k is a device-dependent constant. α is a technology-dependent constant, which varies between 1 and 2.

C. Algorithms

Here, we devise an algorithm (EDTS) to minimize the total energy consumption, while satisfying the time and probability constraint.

For real-time applications in CPS, we use the following major steps to implement the energy-aware scheduling.

1) First, we partition and map the tasks in a DAG G onto the microprocessors of a CPS platform. Then, an initial schedule of DAG G with the task execution order and communication links is obtained.
2) Second, we identify the *critical path* (CP) by finding the path with the longest execution time. If there are more than one longest path in the graph, we select the one with the largest energy consumption in the DAG G.
3) Third, based on the ACETs for all tasks in the graph, we can obtain a static schedule by our static scheduling algorithm.
4) Finally, within each scanning period, the whole task graph is dynamically scheduled, and the execution order of each task is determined by our dynamic scheduling algorithm.

We consider related architectural constraints, heterogeneity, and resource capacities of CPS platforms; during partitioning and mapping the tasks in a DAG, the available energy of each processor may vary over time for different applications. If the resource availability varies too much, the DAG needs to be repartitioned and remapped onto processors to maintain energy efficiency. We adopt the partitioning scheme, i.e., Voltage Presence Indicator System (VPIS), proposed in [21] to schedule tasks onto microprocessors, with the consideration of various constraints and conditions. Our objective is to balance the load and minimize the total system energy consumption.

1) CPA Optimal Algorithm: We use a dynamic programming method to solve the energy-aware scheduling problem for CPS. Given the timing constraint TC, probability constraint PC, a DAG G, and an assignment A, we give several definitions as follows:

Definition 3.1—Assignment A: An allocation scheme assigns a specific voltage mode to each task in a DAG;

Definition 3.2—G^i: A subgraph i, which starts from the root of the task graph until the node v_i;

Definition 3.3—$E_A(G^i)$, $T_A(G^i)$, and $P_A(G^i)$: The total energy consumption, the total execution time, and the total probability of G^i under the assignment A.

In our algorithm, each step achieves a currently minimum total energy consumption of G^i, while satisfying various timing constraints and probability constraints.

A table $D_{i,j}$ (i represents a node number, and j represents time) will be built, where each entry of this table stores the smallest energy E that has been obtained.

In every step of our algorithm, we will consider at least one task. When two tasks are added together, the total energy consumption is the sum of their energy consumption, i.e., $E'_{i,j} = E^1_{i,j} + E^2_{i,j}$. Meanwhile, the total probability is the average probability of these tasks, $P_A(G^i) = P^1_{i,j} \bigcap P^2_{i,j} = P^1_{i,j} + P^2_{i,j} - P^1_{i,j} \bigcup P^2_{i,j}$. For each entry, we only keep the smallest total energy consumption with reasonable probability and the corresponding voltage level assignment. When there is more than one solution with the same energy consumption, we keep the one with the highest probability. If the probabilities are also the same, all solutions will be kept. When a CP is found, we will use the optimal algorithm, i.e., CPA, to get the

optimal solution for the energy-aware scheduling problem. The algorithm is shown in Algorithm 1.

Algorithm 1 The *CPA* Algorithm for Critical Path.

Input: M different voltage levels, a critical path, a timing constraint TC, and a probability constraint PC.
Output: An optimal assignment for the critical path.
Build a local table $B_{i,j}$ for each node on the critical path;
1: Let $D_{1,j} = B_{1,j}$
2: Start from $u_1 \rightarrow u_{|U|}$, compute D step by step;
3: **for** each time j of node $u_i, i > 1$ **do**
4: **for** each time k in $B_{i,k}$, $1 \leq k \leq j$ **do**
5: **if** $D_{i-1,j-k}! = NULL$ **then**
6: $D_{i,j} = D_{i-1,j-k} + B_{i,k} + \alpha w_i$;
7: **if** $P_{i,j} < PC$ **then**
8: **return**;
9: **else**
10: $D_{i,j} = \max(D_{i,j}, \text{Probability})$ // keep the maximum one if there are multiple results with same energy cost;
11: $D_{i,j} = \min(D_{i,j}, \text{Energy})$; // keep the minimum one if there are multiple results with same probability;
12: **end if**
13: **else**
14: Continue;
15: **end if**
16: **end for**
17: **end for**
18: **return** $D_{|U|,j}$;

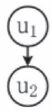
In the CPA algorithm, we first build a local table $B_{i,j}$ for each node. The table $B_{i,j}$ stores the energy consumption of a node under different voltage levels. In the next step of the algorithm, when $i = 1$, there is only one node. We set the initial value, and let $D_{1,j} = B_{1,j}$ (line 1). Then, we build the table $D_{i,j}$ (lines 3–10), by using the dynamic programming method. For each node u_i for each j, we vary time k ($1 \leq k \leq j$) in table $B_{i,j}$ (line 4). We add energy consumption together in the two tables $B_{i,k}$ and $D_{i-1,j-k}$ (line 6). Then, we will check whether the probability meets the requirement PC. If no, we keep the CPA method until some scheme meets the requirement P. If yes, we keep the results with the minimum energy cost. Meanwhile, we keep the results with the maximum probability as well. We also consider the communication cost w_i, when tasks i and $i+1$ are scheduled on different microprocessors. If the two tasks are implemented on the same core, α is 0; otherwise, α is 1. Finally, we keep the smallest total energy and the corresponding voltage selection. The energy in $D_{i,j}$ is the minimum total energy with reasonable probability for graph G^i under the timing constraint j and the probability constraint PC.

For example, for the DFG shown in Fig. 2(b), the initial parameters are shown in Fig. 2(a). We compute the corresponding B table of node u_1 and u_2 as follows:

1) From node u_1, we can get the (T (time): E (energy): P (probability)) pairs, as follows: (1: 20: 0.98), (2: 20: 0.98), (3: 10: 0.94), and (4: 10: 0.94).
2) We sort them by the time constraint from small to big, as shown in Fig. 2(c).

Node	V_1			V_2		
	T_1	E_1	P_1	T_2	E_2	P_2
u_1	1	20	.98	3	10	.94
u_2	2	28	.96	3	22	.94

(a)



(b)

B Table

Node	j =1		j = 2		j = 3		j = 4	
	T=1	P	T=2	P	T=3	P	T=4	P
u1	20	.98	20	.98	10	.94	10	.94
u2	--	--	28	.96	22	.94	22	.94

(c)

D Table

Node	j =1		j = 2		j = 3		j = 4			
	T=1	P	T=2	P	T=3	P	T=4	P	T=4	P
u1	20	.98	20	.98	10	.94	10	.94	10	.94
u2	--	--	--	--	48	.94	48	.94	42	.92

(d)

Fig. 2. (a) Initial parameters. (b) DFG. (c) Corresponding B table. (d) Part of the corresponding D table.

3) We use the same way to calculate node u_2 and get the (T (time): E (energy): P (probability)) pairs, as follows: (2: 28: 0.96), (3: 22: 0.94), and (4: 22: 0.94).
4) Meanwhile, we calculate the path with lowest energy cost to u_1 and u_2. Fig. 2(d) shows the corresponding $D_{i,j}$ table.
5) The first row of D table, which is the paths from u_1 to u_1. Hence, the energy costs are the same with the first row of B table.
6) We calculate the paths with lowest energy cost under different time constraints to u_2, which is the second row in the D table. There is no way from u_1 to u_2 that only consumes one or two times; hence, the first two items are empty.
7) Entry $D_{2,3}$ represents the path $u_1 \rightarrow u_2$ with the time constraint $TC = j = 3$. The only path that satisfies the time constraint is $D_{2,3} = D_{1,1} + B_{2,2}$, and the energy consumption of this path is $20 + 28 = 48$. The probability of this path is $98\% + 96\% - 100\% = 94\%$.
8) Entry $D_{2,4}$ represents the path $u_1 \rightarrow u_2$ with the time constraint $TC = j = 4$. Using the CPA algorithm, two cases can satisfy this time constraint.
   Case 1: $4 = 2 + 2$, with $D_{2,4} = D_{1,2} + B_{2,2}$. Energy consumption for this assignment is: $20 + 28 = 48$. The probability of this path is $98\% + 96\% - 100\% = 94\%$.
   Case 2: $4 = 1 + 3$, with $D_{2,4} = D_{1,1} + B_{2,3}$. Energy consumption under this assignment mode is: $20 + 22 = 42$. The probability of this path is $98\% + 94\% - 100\% = 92\%$. Energy consumption of Case 2 is less than that of Case 1, while the probability of Case 1 is bigger than that of Case 2.
9) Similar to above steps, calculating all the path with the lowest energy cost, and fill the D table.

Time Complexity: It takes $O(M)$ to compute one value of $D_{i,j}$, where M is the maximum number of voltage levels. Thus, the complexity of the CPA algorithm is $O(|N| * TC * M)$, where $|N|$ is the number of nodes and TC is the given timing constraint. Usually, the execution time of each node is bounded by a constant. Hence, TC equals $O(|N|^c)$ (c is a constant). In this case, CPA is a polynomial algorithm.

2) DFGCP Static Scheduling Algorithm: Here, we propose a highly efficient algorithm, i.e., DFGCP, to solve the static scheduling problem. The algorithm is shown in Algorithm 2.

Algorithm 2 The *DFGCP* Algorithm

Input: M different voltage levels, a DFG $G = \langle U, ED, T, E, W \rangle$, a timing constraint TC, and a probability constraint PC.
Output: An assignment for the DFG.
1: $CP = findCriticalPath(G)$; // Find a critical path CP of DFG G;
2: **while** Time(CP) $> TC$ or Probability(CP) $< PC$ **do**
3: Flag $\leftarrow$ No;
4: $\mathrm{Opt} = CPA(M, CP, TC)$; //Use our *CPA* to find the minimal total energy consumptions and corresponding voltage assignments;
5: **if** Time(Opt) $\leq TC$ && Probability(Opt) $\geq PC$ **then**
6: Flag $\leftarrow$ Yes;
7: **end if**
8: **end while**
9: **if** Flag $==$ Yes **then**
10: Output the assignment of G;
11: **else**
12: Output "No Solution"; exit;
13: **end if**
14: For the nodes on the non-critical path (non-CP), we will use CPA algorithm to find the minimal energy consumptions and keep the corresponding voltage levels.
15: Add together the energy of CP and non-CP, we get the minimal total energy consumptions.

In the DFGCP algorithm, we first get a CP of the DFG G. To meet the timing constraint and the probability constraint, we need to judge whether this CP meets these requirements. If the total execution time of the CP is larger than the timing constraint TC or the probability of the CP is less than the probability constraint PC, we will use the CPA algorithm to find the optimal assignment for a new CP with the minimal total energy consumption, the maximum probability, and the corresponding voltage selections. In each step, we will consider the voltage level transfer overheads when using DVS. For each node, if it is not on the same processor with its parent nodes, the communication cost with its parent nodes w_i will be considered. Finally, if we find a solution for CP within TC, the algorithm continues using CPA to find the optimal solution for non-CP paths. At this time, we fix the assignments of the overlapping nodes of CP and non-CP paths.

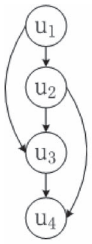
Time Complexity: DFGCP is a polynomial-time algorithm. The complexity of the CPA algorithm is $O(|N| * TC * M)$, where $|N|$ is the number of nodes and TC is the given timing constraint. M is the maximum number of voltage levels. We use CPA to compute every path once. The total number of paths is bounded by $O(|N|^2)$. Hence, CPA is a polynomial-time algorithm. For a sparse graph, the number of paths is very small, assuming a constant c; then, the complexity is approximately linear, and the amount of computation time is very small.

3) EDTS Dynamic Scheduling Algorithm: The static scheduling algorithm DFGCP gives a solution by assuming all tasks run at ACETs. However, in real-life scenarios, we do not know in advance the actual execution time of a task for CPS applications. The information of these tasks will change greatly in runtime; thus, even an optimal static schedule can become invalid in the dynamic case. Here, we present an aggressive dynamic programming based on a scheduling algorithm called EDTS, as shown in Algorithm 3. The EDTS algorithm uses the results from the DFGCP static scheduling algorithm, which obtains a near-optimal schedule based on the knowledge of ACET of each task.

Algorithm 3 The EDTS Algorithm

Input: M different voltage levels, a DFG $G = \langle U, ED, T, E, W \rangle$, and a timing constraint TC.
Output: A dynamic scheduling for the DFG.
1: Get the initial scheduling by DFGCP algorithm;
2: Topologically sort the nodes, getting node sequence $u_i \in U$;
3: **for** each node $u_i, 1 \leq i \leq |U|$, not visited, get the one with the earliest start time **do**
4: **if** required execution time is substantially different from ACET **then**
5: Mark it as visited;
6: Run DFGCP algorithm for the remaining nodes and find the new static schedule with minimal energy consumption while satisfying the new timing constraint ($TC = TC - \sum_{k=1}^{i} T_k$, where $\sum_{k=1}^{i} T_k$ is the time used);
7: **else**
8: Continue;
9: **end if**
10: Finish node u_i, and update system energy overhead and the information (such as the starting time) of nodes that are dependent on u_i;
11: **if** current static schedule is not followed **then**
12: Run DFGCP algorithm for the remaining nodes and find the new static schedule with minimal energy consumption while satisfying the new timing constraint ($TC = TC - \sum_{k=1}^{i} T_k$, where $\sum_{k=1}^{i} T_k$ is the time used);
13: **else**
14: Continue to the next node;
15: **end if**
16: **end for**

The actual execution time of a task may be greater or less than its ACET; we first obtain a static schedule with DFGCP by assuming every task takes its ACET. However, if every task aggressively runs at this statically computed average case speed during runtime, some of them may miss their deadlines. Our EDTS algorithm uses the path information to track any changes of tasks in CPS applications. When a task node is finished, EDTS checks whether the schedule is followed. If not, then the remaining task graph will be recomputed with the DFGCP static scheduling algorithm. In addition, in the course of the implementation of each node, whenever the variation of execution time exceeds the prespecified threshold value, DFGCP will be used to recompute. For example, we set difference ratios to be $\pm 5\%$ between the real execution time and its ACET used previously in DFGCP. The new computation will only implement the remaining subgraph with the updated ACET values.



(a)

Node	V_1		V_2	
	T_1	E_1	T_2	E_2
u_1	1	20	3	10
u_2	2	28	3	22
u_3	2	30	4	8
u_4	3	32	4	11

(b)

Fig. 3. Motivational example. (a) Simple DAG. (b) Execution time and energy consumption for each task at different voltage modes.

E[j]	j											
	1	2	3	4	5	6	7	8	9	10	11	12
$E_1[j]$	20_1	20_1	10_2	10_2	10_2	10_2	10_2	10_2	10_2	10_2	10_2	10_2
$E_2[j]$	--	--	48_1	42_2	38_1	32_1	32_1	32_1	32_1	32_1	32_1	32_1
$E_3[j]$	--	--	--	--	78_1	72_1	56_2	50_2	46_2	40_2	40_2	40_2
$E_4[j]$	--	--	--	--	--	--	--	110_1	89_2	83_2	67_2	61_2

Fig. 4. Procedures to derive the minimal energy consumption for the DAG. $E_i[j]_k$ represents the optimal energy consumption when assigning voltage level k to task i with the time constrain j.

Time Complexity: Our dynamic scheduling algorithm, i.e., EDTS, progressively improves performance based on the schedule obtained by the static scheduling, i.e., DFGCP. The EDTS algorithm is shown in Algorithm 3. For a sparse graph, the complexity of this algorithm is $O(|N|(|N| * TC * M))$, where $|N|$ is the number of nodes, TC is the given timing constraint, and M is the maximum number of voltage levels. Hence, EDTS is a polynomial-time algorithm. For general task graphs, since DFGCP is a polynomial-time algorithm and EDTS calls $O(|N|)$ times of DFGCP, EDTS is also polynomial.

IV. Working Example

Fig. 3(a) shows a simple application on a CPS with two different voltage levels, namely, V_1 and V_2. This application includes four tasks, and the execution time and energy consumption of each voltage mode are shown in Fig. 3(b). Our objective is to schedule all tasks in the graph with the minimum energy consumption, while satisfying a given time constraint.

Based on Fig. 3(a), the CP of the task graph is $u_1 \rightarrow u_2 \rightarrow u_3 \rightarrow u_4$. Assuming that the *timing constraint* (TC) of the CPS application is 12 time units, Fig. 4 illustrates the procedure to achieve the minimal energy consumption by our proposed static scheduling algorithm. The voltage level assignment for each task is recorded in a 2-D matrix $E_i[j]_k$ (i represents a task, j represents a time period, and k represents the voltage mode assignment to task i).

In Fig. 4, it is shown that the minimum energy consumption, i.e., 61, is achieved, assigning the voltage mode $V_1 \rightarrow u_1, V_2 \rightarrow u_2$, $V_2 \rightarrow u_3$, and $V_2 \rightarrow u_4$, respectively. First, we calculate $E_4[8]$, because it is impossible to finish the task u_4 within seven time units.

Then, we can obtain the assignment as follows:

1) Starting from the minimum energy consumption at $E_4[12]$, we know V_2 is assigned to u_4 and its execution time $t_4(2)$ is 4 (for $t_i(k)$, i represents node number, and k represents a voltage level), as shown in Fig. 3(b).

TABLE I
Execution Time, Energy Consumption, and Probability for Each Task at Different Voltage Modes

Node	V_1			V_2		
	T_1	E_1	P_1	T_2	E_2	P_2
u_1	1	20	98%	3	10	94%
u_2	2	28	96%	3	22	94%
u_3	2	30	96%	4	8	92%
u_4	3	32	94%	4	11	92%

2) Calculating the suboptimal combination of task modes before adding task u_4, we can get the index for $E_3[j]$ by subtracting $t_4(2)$ from TC: $TC - t_4(2) = 12 - 4 = 8$.
3) Then, we arrive at the location $E_3[8]$, which means that the optimal energy consumption to execute all the tasks from the root to u_3 is 50. By checking the mode assignment, we can see that V_2 is allocated to u_3. Therefore, the execution time of task u_3 is $t_3(2) = 4$.
4) In a similar way, we can determine that V_2 is assigned to u_2 and its execution time is $t_2(2) = 3$. Furthermore, V_1 is assigned to u_1, and its execution time is $t_1(1) = 1$. Thus, the total execution time from u_2 to u_4 is $1 + 3 + 4 + 4 = 12$ (which is not greater than 12), and the total energy consumed is $20 + 22 + 8 + 11 = 61$.

Then, we add the probability of whether some node can complete the task in time. We use p_i to represent the probability of node i, if it can complete the task within the stipulated time. p_i is not a constant, and it may change greatly with different nodes.

We change the motivational example, as shown in Table I, as follows:

1) u_1 has 98% probability of completing its task in $T_1(1)$ using energy E_1 under V_1 voltage. u_2 has 94% probability of completing its task in $T_2(1)$ using energy E_2 under V_2 voltage.
2) u_2 has 96% probability of completing its task in $T_1(2)$ using energy E_1 under V_1 voltage. u_2 has 94% probability of completing its task in $T_2(2)$ using energy E_2 under V_2 voltage.
3) u_3 has 96% probability of completing its task in $T_1(3)$ using energy E_1 under V_1 voltage. u_3 has 92% probability of completing its task in $T_2(3)$ using energy E_2 under V_2 voltage.
4) u_4 has 94% probability of completing its task in $T_1(4)$ using energy E_1 under V_1 voltage. u_4 has 92% probability of completing its task in $T_2(4)$ using energy E_2 under V_2 voltage.

In this situation, when calculating the energy cost of every path, we take the probability into consideration. When choosing which path is the optimal among some optional choices, we not only compare their energy cost but also compare their probability.

For example, we set the time constraint to 10. We calculate every path from node 1 to node 4 under different voltages, and the result is shown in Fig. 5.

1) For $E_1[j]$ and $P_1[j]$, if the time constraint is under 3, there is only one path, which is V_1, and the energy cost of using is 20, while the probability is 98%. If the time constraint is or higher than 3, there are two candidate paths. Using V_2, the energy cost is 10, while the probability is 94%.

E[j] /P[j]	j									
	1	2	3	4	5	6	7	8	9	10
$E_1[j]$ $P_1[j]$	20_1 .98	20_1 .98	10_2 .94	10_2 .94	10_2 .94	10_2 .94	10_2 .94	10_2 .94	10_2 .94	10_2 .94
$E_2[j]$ $P_2[j]$	--	--	48_1 .94	42_2 .92	38_1 .90	32_1 .88	32_1 .88	32_1 .88	32_1 .88	32_1 .88
$E_3[j]$ $P_3[j]$	--	--	--	--	78_1 .80	72_1 .78	56_2 .76	68_1 .76 50_2 .74	62_1 .74 46_2 .72	40_2 .70
$E_4[j]$ $P_4[j]$	--	--	--	--	--	--	--	-- 110_1 .84	89_2 .82 95_1 .80	83_2 .80 100_1 .80 88_1 .82

Fig. 5. Procedures to derive the minimal energy consumption for the DAG. $E_i[j]_k$ represents that assigning voltage level k to task i is optimal, when the time constraint is j. $P_i[j]_k$ represents the probability of node i.

2) For $E_2[j]$ and $P_2[j]$, if the time constraint is under 3, there are no solutions. If the time constraint is 3, there is only one solution, whose path is "u_1 using $V_1 \rightarrow u_2$ using V_1." The energy cost is $(20 + 28) = 48$, while the probability is $98\% + 96\% - 100\% = 94\%$. If the time constraint is more than 6, the solution with lowest energy is "u_1 using $V_2 \rightarrow u_2$ using V_2." The energy cost is $(10 + 22) = 32$, while the probability is 94%.
3) For $E_3[j]$ and $P_3[j]$, if the constraint is 7, there are two solutions with different energy costs and probabilities. The first path is "u_1 using $V_1 \rightarrow u_2$ using $V_1 \rightarrow u_3$ using V_2." The second path is "u_1 using $V_2 \rightarrow u_2$ using $V_1 \rightarrow u_3$ using V_1." The energy cost and the probability of the first path are $(20 + 28 + 8) = 56$ and $98\% + 96\% + 92\% - 200\% = 86\%$, respectively. The energy cost and the probability of the second path are $(10 + 28 + 30) = 58$ and $94\% + 96\% + 96\% - 200\% = 86\%$, respectively. The first solution has lower energy but lower probability. To meet any kinds of requirements, we keep these two solutions both in the candidate solutions.
4) For $E_4[j]$ and $P_4[j]$, if the time constraint is 10, there are three kinds of path. The first path is "u_1 using $V_1 \rightarrow u_2$ using $V_2 \rightarrow u_3$ using $V_1 \rightarrow u_4$ using V_2." The second path is "u_1 using $V_2 \rightarrow u_2$ using $V_1 \rightarrow u_3$ using $V_1 \rightarrow u_4$ using V_1." The third path is "u_1 using $V_1 \rightarrow u_2$ using $V_1 \rightarrow u_3$ using $V_2 \rightarrow u_4$ using V_1." The probability of the first path is the same as that of the third path, which is $98\% + 96\% + 92\% + 94\% - 300\% = 78\%$. Because the energy cost of the first path is lower than that of the third one, we only put the first and the second path into the candidate solutions.

If we set the probability as more than 80% probability of successfully completing the task, we can obtain the assignment, as shown in Fig. 5, from u_4 to u_1, under the time constraint, which is 10, and the probability constraint, which is 82%, and the energy cost is 88.

V. Experiments

Here, we use an Android emulator to simulate an Android system, and conduct experiments with the EDTS algorithm on a set of benchmarks, including *wave digital filter* (WDF), *infinite impulse filter*, *2-D filter* (2D), *Floyd–Steinberg algorithm* (Floyd), and *all-pole filter*. The number of tasks for these benchmarks has been augmented with the unfolding technique (unfolding rate is 5).

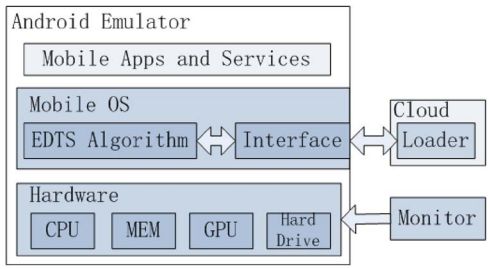


Fig. 6. Setup of our experiment. The EDTS algorithm is used in the mobile OS to schedule the tasks produced by the interface. The interface interacts with local mobile apps and services and remote cloud tasks.

A. Setup

The proposed runtime system has been implemented, and a simulation framework to evaluate its effectiveness has been built, as shown in Fig. 6. We use the Android Software Development Kit (SDK) to create a mobile device emulator, and simulate an Android Nexus 7 with 1.7-GHz CPU, 2-GB RAM memory, and 7.0-in screen. We set the Android emulator as follows: the SDK is Android 4.0 with application programming interface (API) level 14, the memory size is 2 GB, the virtual machine heap is 64, and the internal storage is 64 GB. Then, we use a PC with i7-4810MQ 2.80-GHz CPU and 16-GB memory as a cloud.

We modify the Android mobile operating system (OS) by adding an interface module to deal with the interactions with the cloud. Then, the interface module send requests to the scheduling module, in which we implement our EDTS algorithm. The EDTS algorithm schedules all the tasks to the hardware in an Android device and sends back the results to the interface module. In the interface module, we use Android Secure Socket Layer protocol and the Handler to implement a *messaging application programming interface* (MAPI), which simulates the interaction of clouds and mobile devices.

The execution time (ACET and WCET) and energy consumption are based on the profiling. The execution time of each node follows a Gaussian distribution. We conducted experiments using three different methods:

1) Method 1: Dynamic version *parallelism-based* (PS) algorithm [33];
2) Method 2: *Critical path dynamical scheduling* (CPDS) [34];
3) Method 3: Our EDTS algorithm.

Method 1 uses a greedy technique to further reclaim the slack generated during runtime. Initially, all tasks are assigned with a statically computed processing speed. All the available slacks from a task due to its earlier completion are given to the next expected task running on the same processor. The speed for the next expected task will be adjusted based on its ready time.

The experiments are conducted based on the power model of 70-nm processor [35]. Then, energy consumption per cycle can be calculated by using [35, eq. (9)]. The power is derived from the formula $E_{\text{cyc}} = P/f$. In experiments, we use M different voltage types with a descending processing speed in $V_1, V_2, \ldots, V_M$. The time and energy overheads during a voltage transition among the aforementioned voltage levels are calculated based on [36, eqs. (15) and (20)]. We compare our results with those from Methods 1 and 2.

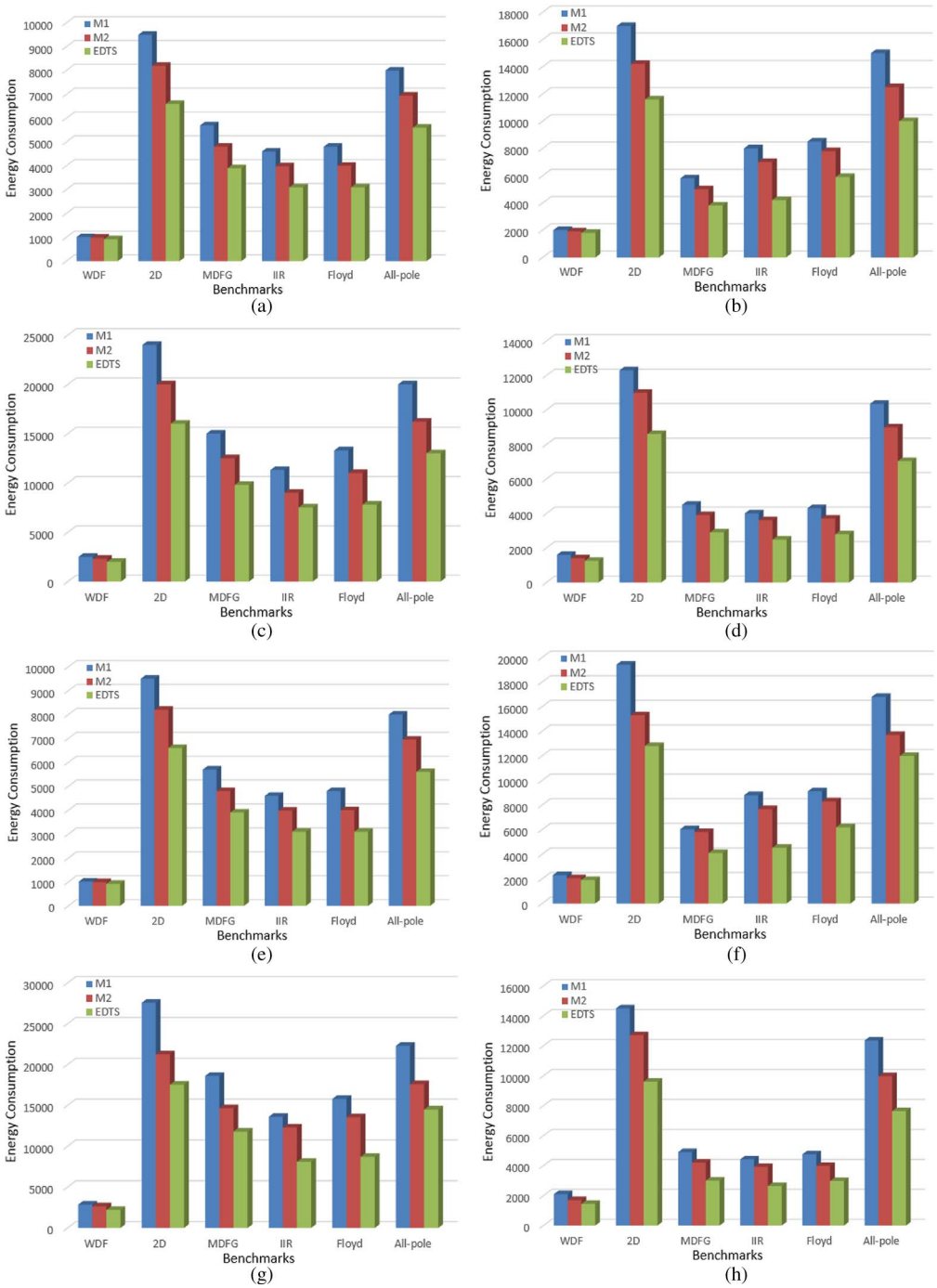


Fig. 7. Comparison of total energy consumption for Method 1, Method 2, and EDTS, when (a) $TC = 2000$ ns and $M = 3$; (b) $TC = 3000$ ns and $M = 4$; (c) $TC = 4000$ ns and $M = 5$; (d) $TC = 2000$ ns and $M = 5$; (e) $TC = 2000$ ns, $PC = 80\%$, and $M = 3$; (f) $TC = 3000$ ns, $PC = 80\%$, and $M = 4$; (g) $TC = 4000$ ns, $PC = 80\%$, and $M = 5$; (h) $TC = 2000$ ns, $PC = 80\%$, and $M = 5$, across a set of benchmarks.

B. Results

The experimental results are shown in Fig. 7(a)–(h). The value of "TC" is 2000, 3000, 4000, and 2000, while the amount of voltages is 3, 4, 5, and 5, respectively. In these figures, M1, M2, and EDTS represent the Method 1, Method 2, and our proposed dynamic scheduling algorithm.

Fig. 7(a)–(d) shows experiments without probability constraints. In Fig. 7(a), when the time constraint is 2000 ns and the amount of candidate voltage is 3, our algorithm can save average 30.1% energy cost than Method 1. Meanwhile, the energy cost of our algorithm is 20.2% less than the Method 2. Then, we set the time constraint as 3000 ns, and the amount of candidate voltage as 4. As shown in Fig. 7(b), our algorithm saves over 32.67% energy cost than Method 1 and 21.53% than the Method 2. In Fig. 7(c), when the time constraint is 4000 ns and the amount of candidate voltage is 5, our algorithm can save average 33.06% energy cost than Method 1. Meanwhile, the energy cost of our algorithm is 20.67% less than the Method 2. Then, we set the time constraint as 2000 ns and the amount of candidate voltage as 5. As shown in Fig. 7(d), our algorithm saves over 29.4% energy cost than Method 1 and 19.1% than the Method 2.

With the increase of time constraint and amount of voltages, our approach can save more energy cost. More amount of voltages indicates that there are more candidate strategies with different complete time, probability, and energy cost; thus, our approach can offer more time–probability–energy pairs to meet various kinds of requirements, and select the lowest energy consuming strategy via the optimal algorithm CPA. However, comparing the results in Fig. 7(a) and (c), we can find that, under the same time constraint, the reduction in energy consumption is less prominent than increasing time constraint.

Fig. 7(e)–(h) shows the experimental results with considering the probability constraint. We set the probability constraint "PC" as 80%. From these results, we can see that the solutions with considering the probability constraint spend more energy than the ones without considering the probability constraint. However, if we add the consideration of the probability to Methods 1 and 2, our approach still shows 33.8% and 27.4% reduction in energy consumption, respectively. Hence, our EDTS algorithm can significantly improve the performance of CPS.

VI. Conclusion

To meet the energy cost and reliability constraints of cloud-based CPS, we have studied how to minimize the total energy consumption on smartphones. We have presented the EDTS algorithm, which utilizes the results from a static scheduling algorithm and aggressively reduces energy consumption. Experimental results across a suite of benchmarks on Android system have shown that our algorithm can achieve significantly higher energy efficiency for CPS. In the future, we will test our approach on more kinds of mobile devices, in addition to Android devices. Our additional research will focus on combining more MCC techniques with other algorithms to improve our algorithm.

References

[1] S. Abolfazli, Z. Sanaei, E. Ahmed, A. Gani, and R. Buyya, "Cloud-based augmentation for mobile devices: Motivation, taxonomies, and open challenges," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 1, pp. 337–368, 1st Quart. 2014.

[2] J. Sorber, N. Banerjee, M. D. Corner, and S. Rollins, "Turducken: Hierarchical power management for mobile devices," in *Proc. ACM MobiSys*, Jun. 2005, pp. 261–274.

[3] B. Priyantha, D. Lymberopoulos, and J. Liu, "Enabling energy efficient continuous sensing on mobile phones with LittleRock," in *Proc. ACM/IEEE IPSN*, Apr. 2010, pp. 420–421.

[4] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proc. 38th ISCA*, 2011, pp. 365–376.

[5] R. Teodorescu and J. Torrellas, "Variation-aware application scheduling and power management for chip multiprocessors," in *Proc. 35th ISCA*, 2008, pp. 363–374.

[6] E. Ahmed, A. Gani, M. K. Khan, R. Buyya, and S. U. Khan, "Seamless application execution in mobile cloud computing: Motivation, taxonomy, and open challenges," *J. Netw. Comput. Appl.*, vol. 52, pp. 154–172, Jun. 2015.

[7] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *Proc. ACM SOSP*, Oct. 2001, pp. 89–102.

[8] T. D. Burd, T. A. Pering, A. J. Stratakos, and R. W. Brodersen, "A dynamic voltage scaled microprocessor system," *IEEE J. Solid-State Circuits*, vol. 35, no. 11, pp. 1571–1580, Nov. 2000.

[9] P. T. Bezerra *et al.*, "Dynamic frequency scaling on android platforms for energy consumption reduction," in *Proc. 8th ACM MSWiM*, 2013, pp. 189–196.

[10] Z. Wang, Z. Gu, M. Yao, and Z. Shao, "Endurance-aware allocation of data variables on NVM-based scratchpad memory in real-time embedded systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, Apr. 2015, DOI: 10.1109/TCAD.2015.2422846.

[11] Z. Wang, Z. Gu, and Z. Shao, "WCET-aware energy-efficient data allocation on scratchpad memory for real-time embedded systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, Dec. 2014, DOI: 10.1109/TVLSI.2014.2379635.

[12] M. Qiu and E. H.-M. Sha, "Cost minimization while satisfying hard/soft timing constraints for heterogeneous embedded systems," *Trans. Des. Autom. Electron. Syst.*, vol. 14, no. 2, p. 25, Mar. 2009.

[13] Z. Wang, Z. Gu, and Z. Shao, "Optimizated allocation of data variables to PCM/DRAM-based hybrid main memory for real-time embedded systems," *Embedded Syst. Lett.*, vol. 6, no. 3, pp. 61–64, Sep. 2014.

[14] W. Yuan and K. Nahrstedt, "Energy-efficient soft real-time CPU scheduling for mobile multimedia systems," in *Proc. ACM SOSP*, Oct. 2003, pp. 149–163.

[15] F. Gruian, "Hard real-time scheduling for low-energy using stochastic data and DVS processors," in *Proc. ISLPED*, Aug. 2001, pp. 46–51.

[16] J. R. Lorch and A. J. Smith, "Improving dynamic voltage scaling algorithms with PACE," in *Proc. ACM SIGMETRICS*, Jun. 2001, pp. 50–61.

[17] J. Luo and N. K. Jha, "Static and dynamic variable voltage scheduling algorithms for real-time heterogeneous distributed embedded systems," in *Proc. ASP-DAC*, Jan. 2002, pp. 719–726.

[18] X. Yang *et al.*, "Integrating dynamic pricing of electricity into energy aware scheduling for HPC systems," in *Proc. SC*, 2013, pp. 60:1–60:11.

[19] L. Santinelli *et al.*, "Energy-aware packet and task co-scheduling for embedded systems," in *Proc. 10th ACM EMSOFT*, 2010, pp. 279–288.

[20] J. Mei and K. Li, "Energy-aware scheduling algorithm with duplication on heterogeneous computing systems," in *Proc. ACM/IEEE 13th GRID*, 2012, pp. 122–129.

[21] Z. Wang and X. S. Hu, "Energy-aware variable partitioning and instruction scheduling for multibank memory architectures," *ACM Trans. Des. Autom. Electron. Syst.*, no. 2, pp. 369–388, Apr. 2005.

[22] M. Qiu, M. Zhong, J. Li, K. Gai, and Z. Zong, "Phase-change memory optimization for green cloud with genetic algorithm," *IEEE Trans. Comput.*, Mar. 2015, DOI: 10.1109/TC.2015.2409857.

[23] M. Qiu *et al.*, "Data allocation for hybrid memory with genetic algorithm," *IEEE Trans. Emerging Topics Comput.*, Feb. 2015, DOI: 10.1109/TETC.2015.2398824.

[24] F. X. Lin, Z. Wang, R. LiKamWa, and L. Zhong, "Transparent Programming of Heterogeneous Smartphones for Sensing," *arXiv preprint arXiv:1103.2348*, 2011.

[25] X. Li *et al.*, "Smartphone evolution and reuse: Establishing a more sustainable model," in *Proc. ICPPW*, Sep. 2010, pp. 476–484.

[26] J. Wan, D. Zhang, S. Zhao, L. T. Yang, and J. Lloret, "Context-aware vehicular cyber-physical systems with cloud support: Architecture, challenges, and solutions," *IEEE Commun. Mag.*, vol. 52, no. 8, pp. 106–113, Aug. 2014.

[27] J. Wan *et al.*, "VCMIA: A novel architecture for integrating vehicular cyber-physical systems and mobile cloud computing," *Mobile Netw. Appl.*, vol. 19, no. 2, pp. 153–160, Apr. 2014.

[28] A. Fahim, A. Mtibaa, and K. A. Harras, "Making the case for computational offloading in mobile device clouds," in *Proc. 19th ACM MobiCom*, Miami, FL, USA, 2013, pp. 203–205.

[29] L. Yang *et al.*, "A framework for partitioning and execution of data stream applications in mobile cloud computing," *SIGMETRICS Perform. Eval. Rev.*, vol. 40, no. 4, pp. 23–32, Apr. 2013.

[30] M. Qiu, Z. Chen, L. T. Yang, X. Qin, and B. Wang, "Towards power-efficient smartphones by energy-aware dynamic task scheduling," in *Proc. IEEE 9th HPCC-ICESS*, 2012, pp. 1466–1472.

[31] J. Niu, C. Liu, Y. Gao, and M. Qiu, "Energy efficient task assignment with guaranteed probability satisfying timing constraints for embedded systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 8, pp. 2043–2052, Aug. 2014.

[32] M. Qiu, Z. Chen, and M. Liu, "Low-power low-latency data allocation for hybrid scratch-pad memory," *IEEE Embedded Syst. Lett.*, vol. 6, no. 4, pp. 69–72, Dec. 2014.

[33] Y. Liu, B. Veeravalli, and S. Viswanathan, "Critical-path based low-energy scheduling algorithms for body area network systems," in *Proc. IEEE RTCSA*, Aug. 2007, pp. 301–308.

[34] R. Mishra, N. Rastogi, D. Zhu, D. Mosse, and R. Melhem, "Energy aware scheduling for distributed real-time systems," in *Proc. IPDPS*, Apr. 2003, pp. 1–9.

[35] S. Martin, K. Flautner, T. Mudge, and D. Blaauw, "Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads," in *Proc. IEEE/ACM ICCAD*, 2002, pp. 721–725.

[36] J. Park, D. Shin, N. Chang, and M. Padram, "Accurate modeling and calculation of delay and energy overheads of dynamic voltage scaling in modern high-performance microprocessors," in *Proc. ACM ISLPED*, Aug. 2010, pp. 419–424.

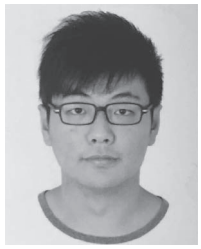
Yibin Li received the Ph.D. degree from the School of Electronic, Electrical and Systems Engineering, Loughborough University, Loughborough, U.K., in 2009.

During his Ph.D. wrap-up stage, he was with The Institute of Electronics, Communications and Information Technology (ECIT), Queen's University of Belfast, Belfast, U.K. In 2010, he joined the Embedded Group in the Department of Computer Science and Engineering, Shandong University, Jinan, China, as an Associate Professor. He has been engaged in several projects, including embedded design for Lidar system, the reconfigurable hardware implementation of Advanced Encryption Standard, etc.

Min Chen (M'08–SM'09) received the Ph.D degree in Electrical Engineering from South China University of Technology in 2004.

He is a Professor with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China. From September 2009 to February 2012, he was an Assistant Professor with the School of Computer Science and Engineering, Seoul National University (SNU), Seoul, Korea. From 2008 to 2009, he was the R&D Director with Confederal Networks Inc. He was a Postdoctoral Fellow with the Department of Electrical and Computer Engineering, University of British Columbia (UBC), Vancouver, BC, Canada, for three years. Before joining UBC, he was a Postdoctoral Fellow with SNU, for one and a half years. He has more than 180 paper publications, including 85 Science Citation Index papers. His research focuses on Internet of Things, big data, machine-to-machine communications, body area networks, E-healthcare, mobile cloud computing, ad hoc cloudlets, cloud-assisted mobile computing, ubiquitous network and services, and multimedia transmission over wireless network, etc.

Prof. Chen was a Technical Program Committee Member of the IEEE International Conference on Computer Communications (INFOCOM) 2013 and INFOCOM 2014. He was a recipient of the Best Paper Award from the IEEE International Conference on Communications (ICC 2012) and the Best Paper Runner-Up Award from the International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness (Qshine 2008). He serves as an Editor or Associate Editor for *Information Sciences*, *Wireless Communications and Mobile Computing*, *IET Communications*, *IET Networks*, Wiley International Journal of *Security and Communication Networks*, *Journal of Internet Technology*, *KSII Transactions on Internet and Information Systems*, and *International Journal of Sensor Networks*. He serves as a Managing Editor for the *International Journal of Autonomous and Adaptive Communications Systems* (IJAACS) and the *International Journal of Arts and Technology* (IJART). He serves as a Guest Editor for the IEEE NETWORK, the IEEE WIRELESS COMMUNICATIONS MAGAZINE, etc. He is the Chair of the IEEE Computer Society Special Technical Communities on Big Data. He was a Cochair of the IEEE ICC 2012 Communications Theory Symposium and of the IEEE ICC 2013 Wireless Networks Symposium. He was the General Cochair of the IEEE International Conference on Computer and Information Technology (CIT) 2012 and Mobimedia 2015. He is the General Vice Chair of Tridentcom 2014. He was also a Keynote Speaker for CyberC 2012 and Mobiquitous 2012.



Wenyun Dai (S'13) received the B.S. degree from Xiamen University, Xiamen, China, and the M.S. degree Shanghai Jiao Tong University, Shanghai, China. He is currently working toward the Ph.D. degree with the Seidenberg School of Computer Science and Information Systems, Pace University, New York, NY, USA.

Meikang Qiu (SM'07) received the B.E. and M.E. degrees from Shanghai Jiao Tong University, Shanghai, China. He received the M.S. and Ph.D. degrees in computer science from The University of Texas at Dallas, Richardson, TX, USA, in 2003 and 2007, respectively.

He is currently an Associate Professor of computer engineering with Pace University, New York, NY, USA. He was with the Chinese Helicopter R&D Institute, IBM, etc. His research interests include cyber security, embedded systems, cloud computing, smart grid, microprocessor, data analytics, etc. A lot of novel results have been produced, and most of them have already been reported to the research community through high-quality journals [such as IEEE TRANSACTIONS ON COMPUTER, *ACM Transactions on Design Automation*, IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, and *Journal of Parallel and Distributed Computing (JPDC)*] and conference papers [such as the ACM/IEEE Design, Automation and Test in Europe (DATE); the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS); and the Design Automation Conference]. He has published four books, more than 200 peer-reviewed journal and conference papers (including more than 100 journal articles and more than 100 conference papers), and he is the holder of three patents.

Prof. Qiu is an ACM Senior Member. He was a recipient of the ACM Transactions on Design Automation of Electrical Systems (TODAES) 2011 Best Paper Award. His paper about cloud computing has been published in *JPDC* (Elsevier) and ranked #1 in the 2012 Most Downloaded Paper of *JPDC*. He has also been a recipient of four Conference Best Paper Awards [IEEE/ACM International Conference on Embedded Software Systems (ICESS'12), IEEE International Conference on Green Computing and Communications (GreenCom'10), IEEE International Conference on Embedded and Ubiquitous Computing (EUC'10), IEEE International Conference on Computational Science and Engineering (CSE'09)] in the recent four years He was also a recipient of the Navy Summer Faculty Award in 2012 and the Air Force Summer Faculty Award in 2009. His research is supported by the National Science Foundation and industrial companies such as Nokia, TCL, and Cavium. He is currently an Associate Editor of the IEEE TRANSACTIONS ON COMPUTERS and the IEEE TRANSACTIONS ON CLOUD COMPUTING. He is the General Chair of the IEEE International Conference on High Performance and Communications, International Conference on Embedded Software and Systems, and International Symposium on Cyberspace Safety and Security (HPCC/ICESS/CSS), the General Chair of the IEEE International Conference on Cyber Security and Cloud Computing (CSCloud'15) and the International Conference on Network and System Security (NSS'15), and the Steering Committee Chair of the IEEE International Symposium on Big Data Security and Cloud (BigDataSecurity 2015).