IEEE TRANSACTIONS ON
EMERGING TOPICS
IN COMPUTING

Received 15 July 2014; revised 9 November 2014; accepted 21 November 2014. Date of publication 7 January, 2015;
date of current version 6 March, 2015.

Digital Object Identifier 10.1109/TETC.2014.2387688

Just-in-Time Code Offloading for Wearable Computing

ZIXUE CHENG, (Member, IEEE), PENG LI, (Member, IEEE), JUNBO WANG, (Member, IEEE), AND SONG GUO, (Senior Member, IEEE)

School of Computer Science and Engineering, University of Aizu, Aizuwakamatsu 965-8580, Japan
CORRESPONDING AUTHOR: P. LI (lipengcs@gmail.com)

ABSTRACT Wearable computing becomes an emerging computing paradigm for various recently developed wearable devices, such as Google Glass and the Samsung Galaxy Smartwatch, which have significantly changed our daily life with new functions. To magnify the applications on wearable devices with limited computational capability, storage, and battery capacity, in this paper, we propose a novel three-layer architecture consisting of wearable devices, mobile devices, and a remote cloud for code offloading. In particular, we offload a portion of computation tasks from wearable devices to local mobile devices or remote cloud such that even applications with a heavy computation load can still be upheld on wearable devices. Furthermore, considering the special characteristics and the requirements of wearable devices, we investigate a code offloading strategy with a novel just-in-time objective, i.e., maximizing the number of tasks that should be executed on wearable devices with guaranteed delay requirements. Because of the NP-hardness of this problem as we prove, we propose a fast heuristic algorithm based on the genetic algorithm to solve it. Finally, extensive simulations are conducted to show that our proposed algorithm significantly outperforms the other three offloading strategies.

INDEX TERMS Wearable computing, just-in-time, code offloading, cloud.

I. INTRODUCTION

Along with the popularity of various wearable devices, such as Google glass [1] and Magic Ring [2], wearable computing has attracted more and more attentions since it facilitates a new form of cyber-physical interaction comprising small body-worn devices that are always powered on and accessible [3]–[6]. Various emerging applications, such as healthy monitoring, reality augmentation, and gesture or object recognition, require wearable devices to provide fast processing and communication capability in an energy-efficient manner. On the other hand, hardware equipped on wearable devices is usually with limited size and weight, hardly to provide enough capability and power for complicated applications.

To fill the gap between resource demand and supply on wearable devices, we propose a novel architecture that offloads some codes to nearby mobile devices with stronger processing capability or a remote cloud with unlimited computation resources. Specifically, we consider a three-layer architecture as shown in Fig. 1. Wearable devices with limited computation capability forms the first layer closest to users.

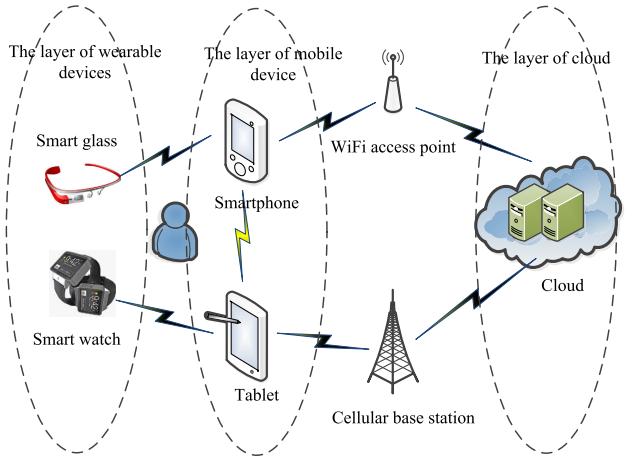


FIGURE 1. Architecture.

Several mobile devices, such as smartphones or tablets, are in the middle layer, which can communicate with wearable devices using short-range communication technologies like ZigBee or Bluetooth. Meanwhile, these mobile devices

can communicate with remote cloud as the third layer via WiFi or LTE networks.

Under this three-layer architecture, we investigate how to efficiently offload codes from wearable devices in the first layer to computation resources in the second and the third layers. In this paper, wearable applications are represented as task graphs, in which methods or functions are denoted by nodes and their relationship by edges. Note that some tasks like sensing or display cannot be offloaded, i.e., they should be executed only on wearable devices. These tasks are referred to as w-tasks in the rest of our paper. For other non-w-tasks, we propose a code offloading algorithm to schedule them on mobile devices or cloud. To guarantee a certain level of user experience, we consider a just-in-time objective for code offloading, i.e., maximizing the number of w-tasks that are executed within a given delay from their direct previous ones. It is motivated by the fact that w-tasks directly interact with users who cannot tolerate long delay between any two adjacent w-tasks.

The main contributions of this paper are summarized as follows.

- We propose a novel three-layer architecture for code offloading from wearable devices to local mobile devices and remote cloud. Different layers have distinct processing capability, and they communicate with each other using different wireless communication technologies.
- We consider an optimization problem for code offloading with a just-in-time objective with respect to user experience. This problem is proved to be NP-hard, and we develop a formulation that deals with the challenges of both task assignment and task scheduling, i.e., to determine where and in which order these tasks should be executed, respectively.
- We develop a fast algorithm based on genetic algorithm to approximate the optimal solution. Instead of directly applying the standard genetic algorithm with high complexity, we propose an enhanced algorithm by creating chromosomes for global scheduling only, and leaving the determination of other variables to a simplified optimization problem.
- Finally, extensive simulations are conducted to evaluate the performance of our proposed algorithm. The results show that our algorithm can quickly converge to performance close to the optimal solution.

The rest of this paper is organized as follows. We review some important related work in Section 2. The system model is presented in Section 3. Section 4 formulates the problem, whose hardness is analyzed in Section 5. Section 6 presents our proposed algorithm. The performance evaluation is given in Section 7. Section 8 finally concludes this paper.

II. RELATED WORK

Code offloading is a critical technique to enable mobile cloud computing [7]–[10] that resource-constrained mobile devices can outsource their computation and storage to the remote cloud. Luo et al. [11] have proposed the idea of using cloud computing to enhance the capabilities of mobile devices. Hyrax [12] has been proposed as a mobile cloud computing platform that allows mobile devices to use cloud computing platforms for data processing. Oberheide et al. [13] have proposed to outsource antivirus services from mobile devices to the cloud.

However, these work simply offloads the whole application to the cloud, which would lead to high communication cost. Thus, a partition scheme has emerged to partially offload applications to cloud for achieving a better performance. CloneCloud [14] seamlessly offloads parts of applications from devices to their clones residing in virtual machines at cloud. Li et al. [15] use the static partitioning method to improve the battery lifetime of mobile devices. Rudenko et al. [16] show the Gaussian application (i.e., to solve a system of linear algebraic equations) can be offloaded into the remote server. Later, several solutions have been proposed to find the optimal decision for partitioning applications before offloading. In [17], the authors present a partition scheme based on the profiling information about computation time and data sharing at the level of procedure calls. The scheme constructs a cost graph, on which a branch-and-bound algorithm [18] is applied with the objective to minimize the total energy consumption of computation and the total data communication cost. The idea of this algorithm is to prune the search space to obtain an approximated solution. In [19], the authors present an approach to decide which components of Java programs should be offloaded. The approach first divides a Java program into methods and uses input parameters to compute the execution costs for these methods. Then, it makes an optimal execution decision by comparing the local execution cost of each method with the remote execution cost estimated based on status of the current wireless channel condition. Wang et al. [20] present a computation offloading scheme on mobile devices and propose a polynomial time algorithm to find an optimal program partition. The proposed scheme partitions a program into the distributed subprograms by producing a program abstraction, where all physical memory references are mapped into the references of abstract memory locations. Yang et al. [21] extend this work by focusing on the system throughput rather than the makespan of the application. Moreover, they propose a genetic algorithm that converges to the global optimal partition running on the cloud side.

Different from existing work, we extend the idea of code offloading to wearable computing by proposing a three-layer architecture. Moreover, we focus on the just-in-time objective with respect to user experience, which has been little studied by existing work.

III. SYSTEM MODEL

A. NETWORK MODEL

We consider a network consisting of a wearable device (WD), several local mobile devices (MD) and a remote cloud (RC),

which can be represented by a graph $G_n(V, E)$, where V denotes a set of nodes including all devices and cloud, and E represents a set of communication links among nodes in V. The processing capability of each node $i \in V$ is denoted by c_i. Typically, we have $c_{WD} \leq c_{MD} \leq c_{RC}$, i.e., the wearable device has the weakest processing capability because of low-end hardware, and the processing speed of mobile devices is faster than wearable device, but slower than cloud.

Each edge $(i, j) \in E$ is associated with a transmission rate r_{ij} depending on the adopted communication technology. Local mobile devices communicate with the wearable device through short-range wireless technologies (e.g., Bluetooth or ZigBee). On the other hand, mobile devices communicate with each other through direct link (e.g., WiFi direct or LTE direct) or wide area network (e.g., 3G networks).

B. APPLICATION MODEL

A wearable application can be represented by a directed acyclic graph $G_a = (N, A)$, where set $N = \{1, 2, ..., n\}$ denotes a number of tasks, and each task $i \in N$ is associated with a weight s_i that represents the number of instructions to be executed. An example of task graph is shown in Fig. 2. The tasks in set N can be divided into two subsets N_W and N_{noW}, which include w-tasks and non-w-tasks, respectively. We have $N_W \cup N_{noW} = N$, and $N_W \cap N_{noW} = \emptyset$. For example, tasks 1, 6, and 8 in Fig. 2 are w-tasks, which may represent user input, picture capture, and result display, respectively, that must be executed on wearable devices. The relationship among tasks is represented by directed links in set A. For a directed link $(i, j) \in A$, we call task i is the predecessor of task j, and task j is the successor of task i. A task can execute only when all its predecessors have finished. In addition, each link $(i, j) \in A$ is associated with a weight e_{ij} that represents the amount of intermediate data from task i to j. We use $P(j)$ to denote the set of predecessors of task j. For example, $P(5) = \{2, 3\}$ in Fig. 2.

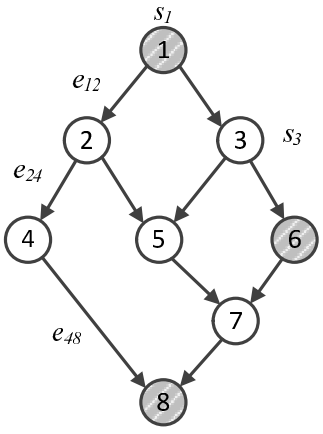


FIGURE 2. An example of task graph.

IV. PROBLEM STATEMENT

Due to size and weight constraints, wearable devices are usually equipped with low-end hardware and powered by batteries with limited capacity. Therefore, they can run only some simple applications with low computation requirement. To support more advanced applications with improved energy efficiency, we propose to offload some codes from wearable devices to local mobile devices and cloud. In other words, instead of executing all tasks in the task graph on the wearable device, we assign some of them to nearby mobile devices with more powerful hardware and more energy supply, or a remote cloud without resource constraints.

In our system model, some tasks, e.g., display or sensing, should be executed on wearable devices. To guarantee user experience, we target on a novel just-in-time objective, i.e., the duration between any two w-tasks should be within a threshold δ. In some cases, this requirement is too strict to generate a feasible solution. For example, there are too many non-w-tasks between two w-tasks, such that the duration between them cannot satisfy the threshold δ under any scheduling. Therefore, we investigate a code offloading problem for wearable devices with a relaxed objective, i.e., seeking a task scheduling to maximize the number of w-tasks that can be executed within δ time after the previous w-task. The problem is formally defined as follows.

Definition 1 [The JCOW (Just-in-time Code Offloading for Wearable Computing) Problem]: given a network consisting of a wearable device, several mobile devices, and a remote cloud, and an application represented by a task graph, we attempt to find a code offloading scheme that maximizes the number of w-tasks, each of which starts within δ time after its previous w-task.

Theorem 1: The JCOW problem is NP-hard.

Proof: It is easy to see that the JCOW problem is in NP class as the objective function associated with a given task scheduling can be evaluated in a polynomial time. The remaining proof is done by reducing the well-known multiprocessor scheduling problem to the JCOW problem. The multiprocessor scheduling problem can be formally described as follows.

INSTANCE: Given a set T of n tasks, and a set P of m processors. Each task $t \in T$ has a length l_t.

QUESTION: Is there a task scheduling such that all tasks can be finished with time δ?

We now describe the reduction from multiprocessor problem to an instance of the JCOW problem. First, we create a wearable device and a remote cloud. For each processor in P, we create a corresponding local device with the same processing capability. We also create a task graph as shown in Fig. 3, which consists of two w-tasks, i.e., i and j, and a set T of non-w-tasks that can be executed in parallel.

In the following, we show that the multiprocessor scheduling problem has a solution if and only if the resulting instance of JCOW problem has a scheduling scheme that satisfies the delay requirement. First, we suppose that there exists a feasible scheduling of multiprocessor scheduling problem such that all tasks can be finished before time δ. It is straightforward to verify that the corresponding solution in JCOW problem guarantees that the delay between two

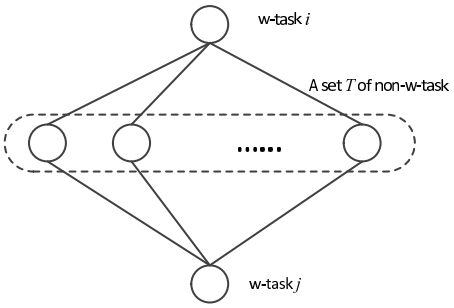


FIGURE 3. An instance of task graph.

w-tasks is less than δ. Then, we suppose that the JCOW problem has a feasible solution such that the delay between w-tasks i and j is less than δ. We schedule the non-w-tasks assigned to each local device to the corresponding processors, which forms a solution of the multiprocessor problem. Based on the above analysis, we conclude that the JCOW problem is NP-complete. Since the decision form of JCOW problem is NP-complete, we can further conclude that the optimization form of the original problem is NP-hard. □

To solve the JCOW problem, we need to deal with the challenges of both task assignment and task scheduling. Task assignment determines on which device each task should be executed. Except w-tasks that should be executed only on the wearable device, other tasks can be offloaded to mobile devices and cloud. Compared with cloud, mobile devices have limited processing capability, but they are closer to the wearable device, leading to small latency for data delivery among tasks. In addition to the tradeoff between processing speed and transmission delay, the existence of multiple mobile devices further complicates task assignment.

Task scheduling determines the execution sequence of tasks assigned to the same device. For example, if tasks 2, 3 and 5 in Fig. 2 are assigned to the same mobile device, we have two possible execution sequences, i.e., $\{2, 3, 5\}$ and $\{3, 2, 5\}$, with different performance. When tasks are executed according to $\{2, 3, 5\}$, task 4 can quickly get its input data after task 2, and run in parallel with task 3 or 5 on the other device. Alternatively, if execution sequence $\{3, 2, 5\}$ is chosen, we can start w-task 6 earlier while delaying the execution of task 4. Therefore, task scheduling on all devices should be jointly considered to achieve the optimal performance.

V. PROBLEM FORMULATION

In this section, we develop an optimization framework for the JCOW problem by jointly considering both task assignment and task scheduling. First, we define a binary variable x_{ik} for task assignment as follows:

$$x_{ik} = \begin{cases} 1, & \text{if task } i \in N \text{ is assigned to node } k \in V; \\ 0, & \text{otherwise.} \end{cases}$$

Since each task can be assigned to one and only one node in the network, we have the following constraint:

$$\sum_{k \in V} x_{ik} = 1, \quad \forall i \in N. \tag{1}$$

For task scheduling, we first define a global scheduling that determines the execution sequence when all tasks are assigned to the same node. When tasks are assigned to multiple devices, the ones in the same device cannot violate the execution sequence defined by the global scheduling. On the other hand, we do not impose any sequence requirement for tasks on different devices. For example, we specify a sequence of $\{1, 2, 3, 4, 5, 6, 7, 8\}$ for tasks in Fig. 2, which will generate three local scheduling $\{1, 6, 8\}$, $\{2, 3, 5\}$, and $\{4, 7\}$ when they are assigned to three devices. Only local scheduling should be obeyed, e.g., task 3 starts after task 2, but task 5 can start before task 4, although it is after task 4 in the global scheduling.

We define a binary variable u_{ij} to specify the global scheduling as follows,

$$u_{ij} = \begin{cases} 1, & \text{if task } j \in N \text{ is scheduled} \\ & \text{immediately after task } i \in N, \\ 0, & \text{otherwise.} \end{cases}$$

If we consider a virtual task n' as both the origin and termination of a circular scheduling, then any task in $N' = \{n'\} \cup N$ should have exactly one successor and one predecessor. These can be described by the constraints:

$$\sum_{j \in N'} u_{ij} = 1, \quad i \in N', \tag{2}$$

$$\sum_{i \in N'} u_{ij} = 1, \quad j \in N'. \tag{3}$$

Now we only need to consider the scheduling of users in N by removing n'. In order to guarantee the resulting scheduling acyclic, we define an integer variable a_i to denote that task i is scheduled in the a_i-th place in the global scheduling. Then, we have the following constraints for a_i:

$$1 \le a_i \le n, \quad \forall i \in N, \tag{4}$$

$$nu_{ij} - n + 1 \le a_j - a_i \le n - 1 - (n-2)u_{ij}, \quad \forall i, j \in N. \tag{5}$$

Note that constraint (5) becomes $a_j - a_i = 1$ if task i is the predecessor of j, i.e., $u_{ij} = 1$, and otherwise $1 - n \le a_j - a_i \le n - 1$ (i.e., $|a_j - a_i| \le n - 1$), which is alway satisfied because of (4).

To simplify the calculation of task execution time later, we define another binary variable y_{ij} for task scheduling as follows:

$$y_{ij} = \begin{cases} 1, & \text{if task } j \in N \text{ is scheduled after } i \in N, \\ 0, & \text{otherwise.} \end{cases}$$

We use an example to explain the differences between u_{ij} and y_{ij}. In the global scheduling $\{1, 2, 3, 4, 5, 6, 7, 8\}$, task 3

is before task 5, so we have $y_{35} = 1$, but $u_{35} = 0$ because task 5 is not a direct successor of task 3. The relationship between y_{ij} and scheduling variables a_j can be represented by:

$$\frac{a_j - a_i}{n} \leq y_{ij} \leq \frac{a_j}{a_i}, \quad \forall i, j \in N. \tag{6}$$

If task i is scheduled before task j, i.e., $a_i < a_j$, above constraint leads to $y_{ij} = 1$ because $0 < \frac{a_j - a_i}{n} < 1$ and $\frac{a_j}{a_i} > 1$. Otherwise, i.e., $a_i > a_j$, we have $y_{ij} = 0$ because $\frac{a_j - a_i}{n} < 0$ and $0 < \frac{a_j}{a_i} < 1$.

Any task $j \in N$ can start to execute when two conditions are satisfied. First, the assigned device should be available, which means no other tasks are currently executing on it. The device available time T_j^a of task j should satisfy the following constraint:

$$T_j^a \geq \sum_{k \in V} x_{jk} x_{ik} y_{ij} t_i^e, \quad \forall i \in P(j), \forall j \in N, \tag{7}$$

where t_i^e is the finish time of task i.

Second, task i should be ready, i.e., it has received data from all its predecessors in the task graph. The task ready time T_j^r can be calculated by:

$$T_j^r = \max_{i \in P(j)} \left\{ t_i^e + \sum_{(k,l) \in E} \frac{e_{ij} x_{ik} x_{jl}}{r_{kl}} \right\}, \quad \forall j \in N. \tag{8}$$

If task j and one of its predecessor $i \in P(j)$ are assigned to different devices k and l, respectively, we need to consider the communication delay $\frac{e_{ij}}{r_{kl}}$. Otherwise, data delivery between them can be implemented via shared memory without communication delay. In this case, the second term in the right hand of constraint (8) will be zero.

The execution start time t_j^s of task j is determined by:

$$t_j^s = \max\{T_j^a, T_j^r\}, \quad \forall j \in N. \tag{9}$$

The relationship between t_j^s and t_j^e can be expressed as:

$$t_j^e = t_j^s + \sum_{k \in V} \frac{x_{jk} s_j}{c_k}, \quad \forall j \in N. \tag{10}$$

Finally, we define a binary variable z_{ij} to describe whether w-task j starts within δ time from its previous w-task i, i.e.,

$$z_{ij} = \begin{cases} 1, & \text{if w-task } j \text{ starts within } \delta \text{ time from} \\ & \text{the previous w-task } j, \\ 0, & \text{otherwise.} \end{cases}$$

We have the following constraints for z_{ij}:

$$\frac{y_{ij}[\delta - (t_j^s - t_i^s)]}{T} \leq z_{ij} \leq \frac{\delta y_{ij}}{t_j^s - t_i^s}, \quad \forall i, j \in N_W, \tag{11}$$

where T is a large constant. By defining a binary variable w_j to represent whether w-task j starts within δ time from any previous w-task, the JCOW problem can be formulated as a mixed-integer nonlinear programming (MINLP) problem as follows.

$$\begin{aligned} \textbf{JCOW}: \ & \max \sum_{j \in N_W} w_j \\ & w_j \leq \sum_{i \in N_W} z_{ij}, \quad \forall j \in N_W \\ & \text{s.t.} \quad (1)\text{–}(11). \end{aligned} \tag{12}$$

Note that the MINLP problem is in general NP-hard, and no mathematical solvers are available because of nonlinear constraints. Thus, we are motivated to design a fast heuristic algorithm to approximate the optimal solution in next section.

Algorithm 1 The Genetic Algorithm Framework

Input:
A task graph G_a, a network G_n, a threshold δ;
Output:
The scheduling of tasks on devices;
1: generate a set of feasible solutions as an initial population.
2: **while** number of generations is not exhausted **do**
3: **for** each population **do**
4: randomly select two chromosomes and apply crossover operation
5: randomly select one chromosomes and apply mutation operation
6: **end for**
7: evaluate all chromosomes in the population and perform selection
8: **end while**

VI. ALGORITHM

A. BASIC IDEA

In this section, we propose a fast algorithm based on genetic algorithm [22] to solve the JCOW problem. The basic idea is to start with a population consisting of a set of feasible solutions that are represented by chromosomes. Chromosomes in one population are randomly selected to produce a new population by crossover and mutation operations. The chromosomes in the new population, which are also referred to as offspring, are selected for survival according to their fitness that is evaluated using our objective function. This heuristic selection mimics the process of natural selection, i.e., the more suitable chromosomes are, the more chances they have to reproduce. This process is repeated until some condition, for example, number of populations or improvement of the best solution, is satisfied. The pseudo codes of our proposed algorithm are shown in Algorithm 1.

To apply the genetic algorithm, we need to first define chromosomes that represent feasible solutions of our problem. A straightforward method is to define a variable as a gene, such that a chromosome contains all variables to form a feasible solution of the problem. However, by including all

kinds of variables, such as the ones for task assignment (x_{ik}) and scheduling (u_{ij}, a_j, and y_{ij}) in a chromosome, it would be difficult to guarantee its feasibility after crossover and mutation operations.

Instead of including all variables in a chromosome, we propose to create chromosomes for global scheduling only, and leave the determination of other variables to a simplified optimization framework. We still use the task graph example in Fig. 2 to illustrate our chromosome construction. As shown in Fig. 4(a), we create two chromosomes {1, 2, 3, 4, 5, 6, 7, 8} and {1, 3, 2, 6, 5, 7, 4, 8}, which represent two possible global scheduling sequences. In the following, we give the detailed design of crossover and mutation operations on our defined chromosomes.

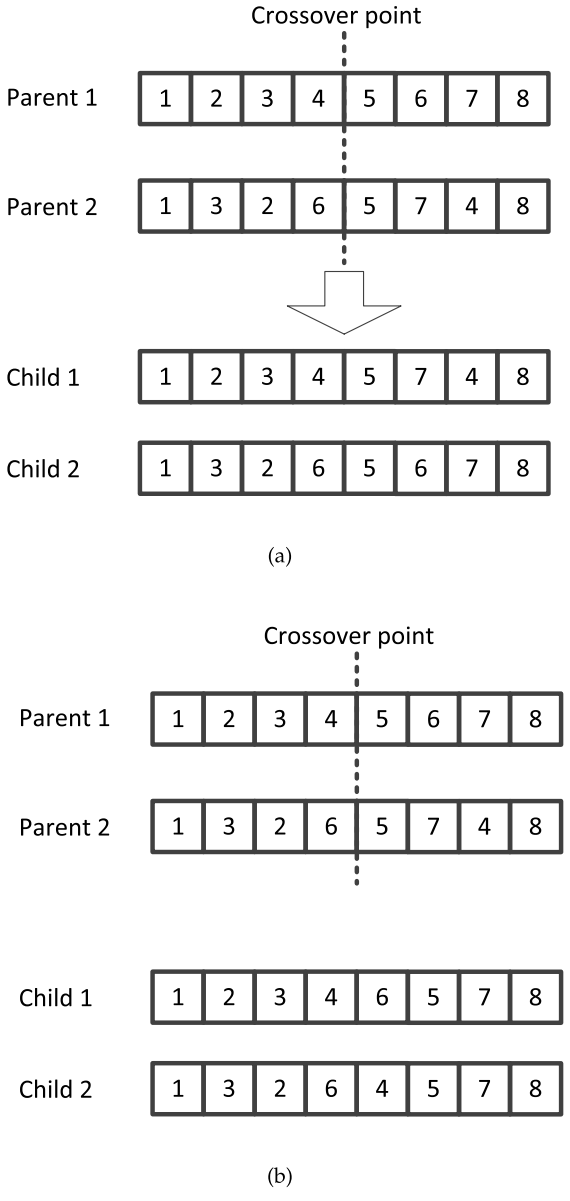


FIGURE 4. An example of crossover operation. (a) Standard crossover. (b) Order crossover.

B. DETAILED DESIGN

1) CROSSOVER OPERATION

To conduct crossover operations in the standard genetic algorithm, we randomly select a point as the crossover point, and exchange the portions beyond the crosspoint to generate two new chromosomes. Unfortunately, this operation would lead to infeasible scheduling that violates the precedence constraints imposed by our task graph. As an example shown in Fig. 4(a), standard crossover operation generates two children that are both infeasible because task 4 appears twice and task 6 even does not show up in the generated chromosome {1, 2, 3, 4, 5, 7, 4, 8}.

As standard crossover operations may violate the precedence constraints, we adopt the order crossover operation [22], [23] that always generates valid scheduling lists from two valid parent chromosomes. Specifically, given any two parent chromosomes, we first randomly choose a crossover point and pass the left segment from the first parent to the child. Then, we construct the right fragment of the child by taking the remaining parts of the first parent, but in the order of the other parent. For example, we set the crossover point in the middle of two chromosomes shown in Fig. 4(b). Then, we create a child chromosome with {1, 2, 3, 4} as its first 4 elements, and other tasks are scheduled according to their order in parent 2, i.e., {6, 5, 7, 8}. The other child can be constructed in a similar way.

2) MUTATION OPERATION

We conduct mutation operation by swapping two randomly selected tasks in the global scheduling list. Note that such a mutation operation may generate invalid scheduling. For example, if we swap tasks 3 and 5 of the chromosome in Fig. 5, the resulting scheduling {1, 2, 5, 4, 3, 6, 7, 8} is invalid because task 5 cannot start before task 3 according to the task graph. To guarantee the feasibility, we check every chromosome after mutation and abandon invalid ones.

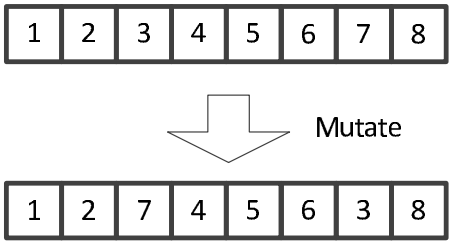


FIGURE 5. An example of mutation operation.

3) FITNESS EVALUATION

The fitness of each chromosome is evaluated by the number of w-tasks that satisfy the just-in-time requirement. Since each chromosome determines a task scheduling sequence, we only need to deal with task assignment now, which can be obtained by solving a simplified optimization framework. Given a chromosome, variables related with task scheduling,

i.e., u_{ij}, a_j, and y_{ij}, can be fixed, which will significantly reduce the complexity of solving the MINLP problem formulated in last section. We use $\hat{u}_{ij}$, $\hat{a}_j$, and $\hat{y}_{ij}$ to denote the fixed values of u_{ij}, a_j, and y_{ij}, respectively, and the task assignment problem can be formulated as:

$$\max \sum_{j \in N_W} w_j,$$

$$\text{subject to: } T_j^a \geq \sum_{k \in V} x_{jk} x_{ik} \hat{y}_{ij} t_i^e, \quad \forall i \in P(j), \forall j \in N, \quad (13)$$

$$\frac{\hat{y}_{ij}[\delta - (t_j^s - t_i^s)]}{T} \leq z_{ij} \leq \frac{\delta \hat{y}_{ij}}{t_j^s - t_i^s}, \quad \forall i, j \in N_W,$$

$$(1), (8), (9), (10), \text{ and } (12). \quad (14)$$

Although many variables and constraints are eliminated, above formulation is still difficult to solve because of non-linear constraints (8) and (13). To linearise these constraints, we define a new binary variable v_{ij}^{kl} as:

$$v_{ij}^{kl} = x_{ik} x_{jl}, \quad \forall i, j \in N, \forall k, l \in V, \quad (15)$$

such that the constraint (8) can be written in a linear form as:

$$T_j^r \geq t_i^e + \sum_{(k,l) \in E} \frac{e_{ij} v_{ij}^{kl}}{r_{kl}}, \quad \forall i \in P(j), \forall j \in N. \quad (16)$$

Constraint (15) can be equivalently replaced by the following linear constraints:

$$0 \leq v_{ij}^{kl} \leq x_{ik}, \quad \forall i, j \in N, \forall k, l \in V, \quad (17)$$

$$x_{ik} + x_{jl} - 1 \leq v_{ij}^{kl} \leq x_{jl}, \quad \forall i, j \in N, \forall k, l \in V. \quad (18)$$

To linearize constraint (13), we define a binary variable ϕ_{ij}^k as:

$$\phi_{ij}^k = v_{ij}^{kk} t_i^e, \quad \forall i \in P(j), \forall j \in N, \forall k \in V, \quad (19)$$

which can be equivalently replaced by:

$$0 \leq w_{ij}^k \leq t_i^e, \quad \forall i \in P(j), \forall j \in N, \forall k \in V, \quad (20)$$

$$t_i^e - T(1 - v_{ij}^{kk}) \leq w_{ij}^k \leq T v_{ij}^{kk}, \quad \forall i \in P(j), \forall j \in N, \forall k \in V. \quad (21)$$

In a similar way, the constraint (14) can be linearized by introducing a new variable $\psi_{ij} = z_{ij}(t_j^s - t_i^s)$, such that task assignment problem can be formulated as follows.

$$\max \sum_{j \in N_W} w_j,$$

$$\text{subject to: } T_j^a \geq \sum_{k \in V} w_{ij}^k \hat{y}_{ij}, \quad \forall i \in P(j), \forall j \in N, \quad (22)$$

$$\hat{y}_{ij}(\delta - t_j^s + t_i^s) \leq z_{ij} T, \quad \forall i, j \in N_W, \quad (23)$$

$$\psi_{ij} \leq \delta \hat{y}_{ij} \quad \forall i, j \in N_W, \quad (24)$$

$$0 \leq \psi_{ij} \leq t_j^s - t_i^s, \quad \forall i, j \in N_W, \quad (25)$$

$$t_j^s - t_i^s - T(1 - z_{ij}) \leq \psi_{ij} \leq T z_{ij}, \quad \forall i, j \in N_W, \quad (26)$$

$$(1), (9), (10), (16)-(18), (20), \text{ and } (21).$$

Although above formulation is a mixed-integer linear programming (MILP) that is generally NP-hard, it can be quickly solved by advanced algorithms like branch-and-bound and mathematical tools like CPLEX. Since we focus on problem formulation and genetic-based algorithm design in this paper, the discussion of solving the MILP problem is omitted due to space limit.

VII. PERFORMANCE EVALUATION

In this section, we conduct extensive simulations to evaluate the performance of our proposed algorithm. The simulation settings will be first introduced, followed by simulation results that demonstrate the advantages of our proposed algorithm.

A. SIMULATION SETTINGS

We first describe a default simulation setting with a number of parameters, and then study the performance by changing one parameter while fixing others. We randomly generate task graphs [21] with 15 tasks, whose node and link weights are Gaussian distributed with mean 100 and variance 10. Among these tasks, 40% of them are randomly selected as w-tasks. We create random networks, each containing of a wearable device, 3 mobile devices and the cloud. The link rate relationship can be described as $r_{MM-MM} = \gamma r_{MM-WD} = \gamma^2 r_{MM-RC}$, and the default value of γ is 50. For comparison, we also consider other three schemes as follows.

Offloading nothing (OLN): all tasks are executed at the wearable device.

Offloading all to cloud (OLAC): we offload all tasks except w-tasks to the cloud.

Simple greedy offloading (SGO): we start from the first task in the task graph, and greedily assign one by one in the following to the network node that results in the earliest finish time.

Our proposed algorithm is denoted by OLGA in the following. All simulation results are averaged over 30 random instances.

B. SIMULATION RESULTS

We first investigate the influence of number of tasks, and show the percent of w-tasks that satisfy the just-in-time requirement in Fig. 6. As the number of tasks grows, the performance of all algorithms decreases. For example, the percent of just-in-time tasks under OLGA is 94.3% when the total number of tasks is 5. As the number of tasks increases to 25, the corresponding percent is 79.5%, leading to 18% degradation. Meanwhile, the performance of OLN, OLAC, and SGO is always lower than OLGA, and their performance degradation is more obvious. For example, OLN has about 28% performance degradation as the number of tasks increases from 5 to 25. OLN shows poor performance because the processing capability of wearable device is very limited, and imposing all tasks to it will seriously delay the execution of w-tasks. Although OLAC can improve the performance of OLN, it is much lower than OLGA because

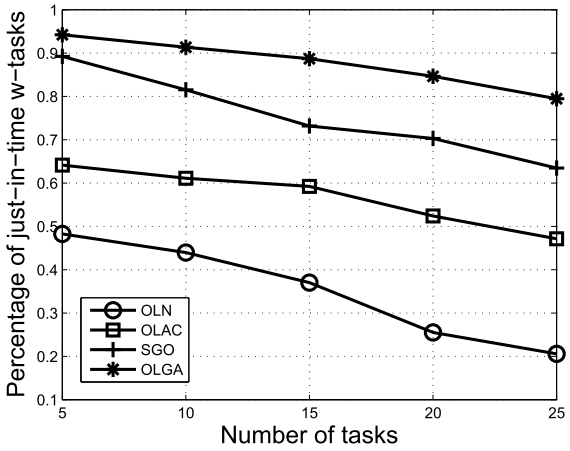


FIGURE 6. Percentage of just-in-time w-task versus different number of tasks.

when each w-task finishes, it delivers results to the cloud that will later return data for next w-tasks. The frequent communication between wearable device and cloud still incurs significant delay.

We then study the effect of w-tasks portion by fixing the number of total tasks to 15. As shown in Fig. 7, the performance of all algorithm increases as number of w-tasks grows. For example, when 10% w-tasks exist in the task graph, the percentage of just-in-time w-tasks are 18.6%, 49.8%, 71.2% and 81.4% under OLN, OLAC, SGO and OLGA, respectively. As the portion of w-tasks grows to 50%, their performance increases to 39.6%, 60%, 75.1% and 92.4%, respectively. The reason is that when more w-tasks exist, there are less other tasks between any two w-tasks, and the just-in-time requirement can be easily satisfied. Also, OLGA always outperforms the other three algorithms because too many tasks are assigned to the wearable device with low processing speed in OLN, and frequent message exchange happens between wearable device and cloud under OLAC.

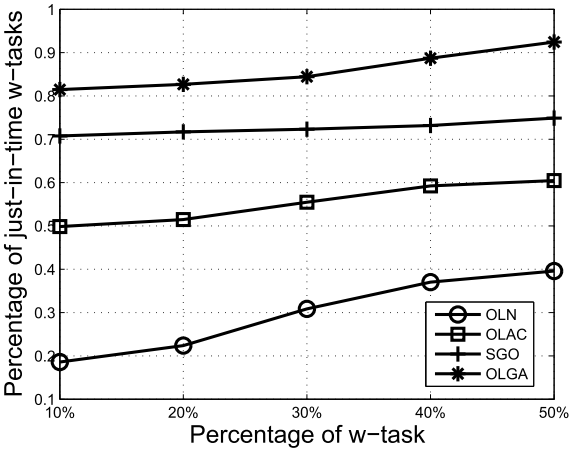


FIGURE 7. Percentage of just-in-time w-task versus different number of w-tasks.

The influence of mobile devices is investigated by changing its number from 1 to 5. As shown in Fig. 8, the performance of our proposed algorithm increases as the number of mobile devices grows. For example, when there is only one mobile device, the percentage of just-in-time w-tasks is 25.5%. The performance increases to 87.5% as the number of mobile device grows to 5. Moreover, we observe that performance improvement becomes less as more mobile devices join the network. For example, two devices brings about 20% performance improvement compared with the case with only one device. However, the performance gap decreases to 6% when the number of mobile devices increases from 4 to 5. There are two reasons for this phenomenon. First, the computation capability of mobile devices has been fully exploited by our algorithm as more devices are added into the network. Second, the overhead of data exchange among mobile devices will overwhelm the benefits of code offloading when more mobile devices are involved.

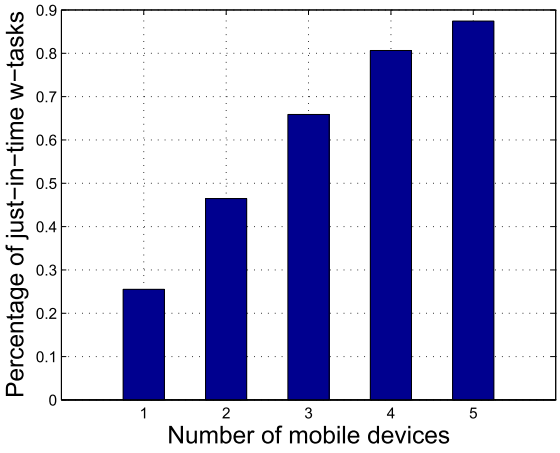


FIGURE 8. Percentage of just-in-time w-task versus different number of mobile devices.

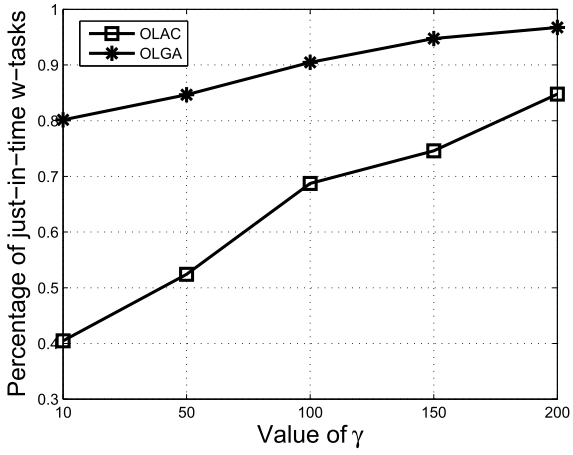


FIGURE 9. Percentage of just-in-time w-task versus different value of γ.

We study the influence of γ by changing its value from 10 to 200. Since OLN is not affected by γ, we only show the performance of OLAC and OLGA in Fig. 9. As the

value of γ grows, the percentage of just-in-time w-tasks increases under both algorithms. For example, when $\gamma = 10$, there are 40% and 80% w-tasks that satisfy the just-in-time requirement under OLAC and OLGA, respectively. As γ grows to 200, the corresponding percentage increases to 84.8% and 96.7%, respectively. We also observe that the performance gap between OLAC and OLGA decreases from 40% to 14% as γ grows from 10 to 200. That is because the communication overhead becomes less under larger value of γ.

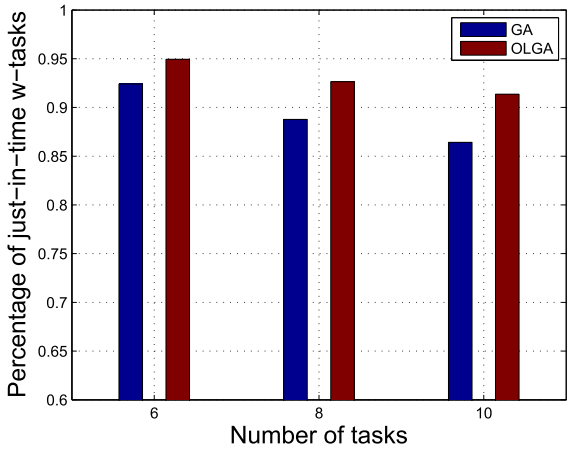


FIGURE 10. Percentage of just-in-time w-task versus different number of tasks.

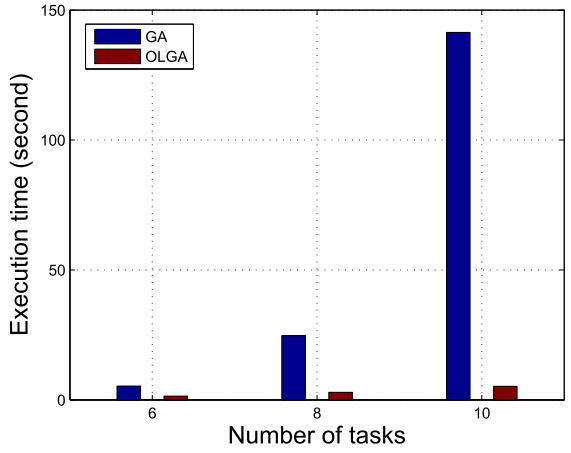


FIGURE 11. Execution time versus different number of tasks.

Finally, we compare our proposed algorithm with traditional genetic algorithm (GA) that uses all binary variables as genes. We apply the crossover operation by randomly selecting a crossover point for two chromosomes and exchanging their portions after the point. The mutation operation can be conducted by randomly mutating a binary variable. If the generated chromosomes represent infeasible solutions, we abandon them and repeat above crossover and mutation operations until we obtain feasible chromosomes. As shown in Fig. 10, our proposed algorithm always outperforms traditional GA. On the other hand, the execution time of GA is significantly higher than OLGA because GA spends a large portion of time to generate feasible chromosomes. As shown in Fig. 11, when there are 10 tasks, GA needs more then 2 minutes to guarantee 86% just-in-time w-tasks, while OLGA achieves the percentage of 91.3% within 5 seconds.

VIII. CONCLUSION

In this paper, we investigate just-in-time code offloading for wearable computing. Instead of offloading all codes directly to the remote cloud, we employ mobile devices nearby to form a local mobile cloud with low communication delay with the wearable device. In such a three-layer architecture, we study the problem of task assignment and scheduling for a given task graph with the just-in-time objective, i.e., the time interval between any two w-tasks that should be executed on wearable device cannot be greater than a threshold. This problem is proved to be NP-hard, and an efficient code offloading algorithm based on genetic algorithm is proposed. Extensive simulation results show that our proposal outperforms other three offloading strategies significantly.

ACKNOWLEDGMENT

The authors would like to thank members in Computer Network Lab of the University of Aizu, especially, Xin Fan and Zhitao Deng,and Nariyoshi Chida for discussions and simulations on this paper.

REFERENCES

[1] *GoogleGlass*. [Online]. Available: https://www.google.com/glass/start/

[2] L. Jing, Y. Zhou, Z. Cheng, and T. Huang, "Magic ring: A finger-worn device for multiple appliances control using static finger gestures," *Sensors,* vol. 12, no. 5, pp. 5775–5790, 2012.

[3] G. Ngai, S. C. Chan, J. C. Y. Cheung, and W. W. Y. Lau, "Deploying a wearable computing platform for computing education," *IEEE Trans. Learn. Technol.*, vol. 3, no. 1, pp. 45–55, Jan./Mar. 2010.

[4] M.-H. Cho and C.-H. Lee, "A low-power real-time operating system for ARC (actual remote control) wearable device," *IEEE Trans. Consum. Electron.*, vol. 56, no. 3, pp. 1602–1609, Aug. 2010.

[5] C. Setz, B. Arnrich, J. Schumm, R. La Marca, G. Troster, and U. Ehlert, "Discriminating stress from cognitive load using a wearable EDA device," *IEEE Trans. Inf. Technol. Biomed.*, vol. 14, no. 2, pp. 410–417, Mar. 2010.

[6] A. Gruebler and K. Suzuki, "Design of a wearable device for reading positive expressions from facial EMG signals," *IEEE Trans. Affective Comput.*, vol. 5, no. 3, pp. 227–237, Jul./Sep. 2014.

[7] A. R. Khan, M. Othman, S. A. Madani, and S. U. Khan, "A survey of mobile cloud computing application models," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 1, pp. 393–413, Feb. 2014.

[8] 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, Feb. 2014.

[9] R. Kaewpuang, D. Niyato, P. Wang, and E. Hossain, "A framework for cooperative resource management in mobile cloud computing," *IEEE J. Sel. Areas Commun.*, vol. 31, no. 12, pp. 2685–2700, Dec. 2013.

[10] M. R. Rahimi, N. Venkatasubramanian, and A. V. Vasilakos, "MuSIC: Mobility-aware optimal service allocation in mobile cloud computing," in *Proc. IEEE 6th Int. Conf. Cloud Comput.*, Jun. 2013, pp. 75–82.

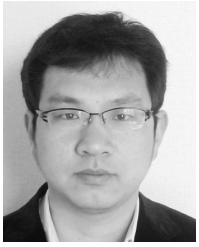
[11] X. Luo, "From augmented reality to augmented computing: A look at cloud-mobile convergence," in *Proc. Int. Symp. Ubiquitous Virtual Reality*, Jul. 2009, pp. 29–32.

[12] E. E. Marinelli, "Hyrax: Cloud computing on mobile devices using MapReduce," Carnegie Mellon Univ.: Pittsburgh, PA, USA, Tech. Rep. CMU-CS-09-164, Sep. 2009.

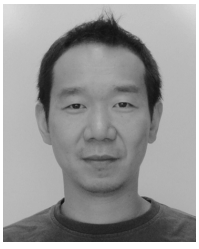
[13] J. Oberheide, K. Veeraraghavan, E. Cooke, J. Flinn, and F. Jahanian, "Virtualized in-cloud security services for mobile devices," in *Proc. 1st Workshop Virtualization Mobile Comput.*, 2008, pp. 31–35.
[14] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "CloneCloud: Elastic execution between mobile device and cloud," in *Proc. ACM 6th Int. Conf. Comput. Syst.*, 2011, pp. 301–314.
[15] Z. Li, C. Wang, and R. Xu, "Task allocation for distributed multimedia processing on wirelessly networked handheld devices," in *Proc. IEEE-IEE Veh. Navigat. Inf. Syst. Conf.*, Oct. 1993, pp. 15–19.
[16] A. Rudenko, P. Reiher, G. J. Popek, and G. H. Kuenning, "Saving portable computer battery power through remote process execution," *ACM SIGMOBILE Mobile Comput. Commun. Rev.*, vol. 2, no. 1, pp. 19–26, Jan. 1998.
[17] Z. Li, C. Wang, and R. Xu, "Computation offloading to save energy on handheld devices: A partition scheme," in *Proc. Int. Conf. Compil., Archit., Synth. Embedded Syst.*, 2001, pp. 238–246.
[18] W. Jigang and S. Thambipillai, "A branch-and-bound algorithm for hardware/software partitioning," in *Proc. 4th IEEE Int. Symp. Signal Process. Inf. Technol.*, Dec. 2004, pp. 526–529.
[19] G. Chen, B.-T. Kang, M. Kandemir, N. Vijaykrishnan, M. J. Irwin, and R. Chandramouli, "Studying energy trade offs in offloading computation/compilation in Java-enabled mobile devices," *IEEE Trans. Parallel Distrib. Syst.*, vol. 15, no. 9, pp. 795–809, Sep. 2004.
[20] C. Wang and Z. Li, "A computation offloading scheme on handheld devices," *J. Parallel Distrib. Comput.*, vol. 64, no. 6, pp. 740–746, Jun. 2004.
[21] L. Yang, J. Cao, S. Tang, T. Li, and A. T. S. Chan, "A framework for partitioning and execution of data stream applications in mobile cloud computing," in *Proc. IEEE 5th Int. Conf. Cloud Comput.*, Jun. 2012, pp. 794–802.
[22] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, 1st ed. Boston, MA, USA: Addison-Wesley, 1989.
[23] K. Shahookar and P. Mazumder, "A genetic approach to standard cell placement using meta-genetic parameter optimization," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 9, no. 5, pp. 500–511, May 1990.

ZIXUE CHENG (M'95) received the master's and Dr. Degrees in engineering from the Tohoku University, Sendai, Japan, in 1990 and 1993, respectively. He joined the University of Aizu, Aizuwakamatsu, Japan, in 1993, as an Assistant Professor, became an Associate Professor in 1999, and has been a Full Professor since 2002. His interests are design and implementation of protocols, distributed algorithms, distance education, ubiquitous computing, ubiquitous learning, embedded systems, functional safety, and Internet of Things. He served as the director of University Business Innovation Center from 2006 to 2010, the Head of the Division of Computer Engineering from 2010 to 2014, and has been the vice president of the University of Aizu, since 2014. He is a member of the Association for Computing Machinery, Institute of Electronics, Information and Communication Engineers, and Information Processing Society of Japan.

PENG LI (M'11) received the B.S. degree from the Huazhong University of Science and Technology, Wuhan, China, in 2007, and the M.S. and Ph.D. degrees from the University of Aizu, Aizuwakamatsu, Japan, in 2009 and 2012, respectively, where he is currently an Associate Professor. His research interests include networking modeling, cross-layer optimization, network coding, cooperative communications, cloud computing, smart grid, performance evaluation of wireless and mobile networks for reliable, energy-efficient, and cost-effective communications.



JUNBO WANG received the B.E. degree in electrical engineering and automation and M.E. degree in electric circuits and systems from YanShan University, Qinhuangdao, China, in 2004 and 2007, and the Ph.D. degree in computer science with the University of Aizu, Aizuwakamatsu, Japan, in 2011, where he is currently an Associate Professor. His current research interests include Internet of Things, ubiquitous computing, context/situation awareness, and wireless sensor networks.



SONG GUO (M'02–SM'11) received the Ph.D. degree in computer science from the University of Ottawa, Ottawa, ON, Canada, in 2005. He is currently a Full Professor at the School of Computer Science and Engineering, University of Aizu, Aizuwakamatsu, Japan. His research interests are mainly in the areas of protocol design and performance analysis for wireless networks and distributed systems. He has authored over 250 papers in refereed journals and conferences in these areas and received three IEEE/ACM best paper awards. Dr. Guo currently serves as an Associate Editor of IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, an Associate Editor of the IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING with duties on emerging paradigms in computational communication systems, and on the editorial boards of many others. He has also been in Organizing and Technical Committees of numerous international conferences. He is a Senior Member of the ACM.