Scate: A Scalable Time and Energy Aware Actor Task Allocation Algorithm in Wireless Sensor and Actor Networks

Mohsen Sharifi and Morteza Okhovvat

In many applications of wireless sensor actor networks (WSANs) that often run in harsh environments, the reduction of completion times of tasks is highly desired. We present a new time-aware, energy-aware, and starvation-free algorithm called Scate for assigning tasks to actors while satisfying the scalability and distribution requirements of WSANs with semi-automated architecture. The proposed algorithm allows concurrent executions of any mix of small and large tasks and yet prevents probable starvation of tasks. To achieve this, it estimates the completion times of tasks on each available actor and then takes the remaining energies and the current workloads of these actors into account during task assignment to actors. The results of our experiments with a prototyped implementation of Scate show longer network lifetime, shorter makespan of resulting schedules, and more balanced loads on actors compared to when one of the three well-known task-scheduling algorithms, namely, the max-min, min-min, and opportunistic load balancing algorithms, is used.

Keywords: Wireless sensor actor networks, task allocation algorithm, energy-awareness, scalability, max-min, min-min.

Manuscript received June 13, 2011; revised Dec. 5, 2011; accepted Dec. 27, 2011.

Mohsen Sharifi (phone: +98 21 7391 3307, msharifi@iust.ac.ir) and Morteza Okhovvat (okhovvat@comp.iust.ac.ir) are with the School of Computer Engineering, Distributed Systems Laboratory, Iran University of Science and Technology, Tehran, Iran.

http://dx.doi.org/10.4218/etrij.12.0111.0366

I. Introduction

Advances in the technologies of micro-electro-mechanical systems have been instrumental in the evolution of wireless sensor actor networks (WSANs) [1], [2] that consist of a set of densely deployed sensor nodes alongside a set of sparsely deployed actor nodes connected via wireless links. Sensor nodes collect environmental information and actors make appropriate actions on the environment based on sensory information they receive from sensors.

There are basically three architectures depending on the strategies adopted by actors to send commands [2]-[6]: semi-automated, automated, and cooperative.

In automated architecture, the network operates in a fully distributed way with the actors that autonomously undertake the appropriate actions upon receiving sensory information. In semi-automated architecture, sensors collect and transmit environmental information to a singleton network sink, and the sink determines the proper actions that actors should execute in response and allocates these actions (tasks) to appropriate actors. In cooperative architecture [4], sensors transmit sensing data to actuators in a single hop or multiple hops. Actuators analyze data and may consult the sink(s) before taking any action. That is, actuators may use their peer-to-peer network to make decisions and take action, possibly informing the sink about the action taken, or could inform the sink and wait for further instructions from the sink.

In this paper, we only consider WSANs with semi-automated architecture wherein data from sensors is routed to a sink node that determines actions to be performed by actors.

WSANs are well suited to quickly respond to environmental events. Since these networks are usually used in critical applications, actors must respond before specified deadlines; otherwise, delays may lead to disaster. However, due to existing constraints such as energy limitations and dynamic and fault-prone attributes of environments, the meeting of deadlines is very challenging. To make efficient use of capabilities of WSANs, the employment of efficient task-scheduling algorithms is inevitable. Hence, many task-scheduling algorithms for distributed systems have been presented so far with the aim of minimizing the total completion time (CT) of tasks [7]-[10]. These algorithms allocate tasks to most suitable actors to minimize the overall CT. However, the reduction of the overall CT of actors alone does not necessarily lead to the reduction of execution times of tasks.

Three famous examples of such algorithms are the opportunistic load balancing algorithm (OLB), the min-min algorithm, and the max-min algorithm [10]. Load balancing is the main goal of OLB, achieved by keeping all actors as busy as possible [10]. OLB schedules the tasks in arbitrary order without considering the execution times or the CTs of tasks [7]. This simple approach can result in schedules with long makespans [10], making OLB inappropriate for WSANs that are critically constrained by time. In contrast, the min-min and max-min algorithms consider the execution times of tasks when assigning tasks to actors.

The min-min algorithm considers the approximate execution times and CTs of all tasks on each actor and only then assigns the task with the shortest CT to an actor with minimal execution time [10], [11]. However, since it always gives priority to smaller tasks and allocates small tasks to faster actors, if the number of small tasks is bigger than the number of large tasks, then larger time-consuming tasks determine the makespan. To overcome this problem, the max-min algorithm gives higher priority to larger tasks and assigns large tasks to actors with minimum execution time. It seems that the max-min algorithm outperforms the min-min algorithm, especially when the number of tasks with longer task CTs is more than the number with shorter ones, but it may result in longer system response time in some cases. Both the min-min and max-min algorithms may lead to task starvation because they give absolute priorities to small and large tasks, respectively.

In this paper, we propose a starvation-free task-allocation algorithm for WSANs, nicknamed *Scate*, in which actors execute concurrently a mix of large and small tasks belonging to one or more applications. Scate achieves a longer network lifetime, a shorter makespan of resulting schedules, and more balanced loads on actors compared with when the max-min algorithm, the min-min algorithm, or OLB is used. Scate achieves its superiority by considering both the CTs of tasks and the remaining energies of actors in its selection of actors.

The rest of paper is organized as follows. Section II presents notable related works. Section III presents the problem statement. Section IV describes our assumptions. Section V presents our proposed algorithm, Scate. Section VI presents the experiment results, and section VII concludes the paper.

II. Related Work

Due to limitations of WSANs such as their energy constraints, dynamicity, and fault-proneness, general scheduling algorithms are often not applicable to WSANs. There have thus been many efforts to propose optimal scheduling algorithms particularly for wireless sensor networks (WSNs) with the purpose of reducing response time and energy consumption. However, there has been little research done on optimal task scheduling in WSANs.

Tian and others [12] presented a multi-hop task-scheduling approach for multi-hop clustered WSNs using the min-min algorithm for task assignment. The approach considers both communication and computation requirements but neither enforces any order on tasks nor guarantees to meet execution deadlines of applications.

Yu and Prasanna [13] proposed an energy-balanced task-allocation algorithm for WSNs. Their algorithm calculates the voltage settings of tasks, assigns communication actions to channels, and schedules communication and calculation activities. They believe that the deployment of their algorithm, depending on the scale of the problem, can lead to a 3.5- to 5-fold network lifetime improvement. However, they do not consider application energy consumption requirements and cannot guarantee to satisfy energy consumption constraints.

Okhovvat and others [14] presented a two-phase task-allocation technique based on queuing theory for allocating tasks to actors in WSANs considering time and energy. Firstly, capabilities of actors in performing tasks are evaluated. Secondly, tasks are allocated to actors according to their capabilities to reduce the total CT of tasks. Their approach improves the makespan of resulting task schedules by 45% compared to OLB, providing a suitable tradeoff between a balanced load on actors and the CTs of all tasks. However, they ignored the limitation in the real world on the size of the queue associated with each actor.

Shivle and others [15] presented new task assignment and scheduling heuristics for mobile ad hoc networks containing an individual communication channel for each node. Each node can simultaneously transmit and receive data. The very assumption of an individual channel for each node as well as the capability of simultaneous data transmission and reception by each node makes this approach inappropriate for WSANs because sensor nodes in WSANs often lack these capabilities.

Awadallah and Darwish [16] proposed a QoS-constraint task-scheduling algorithm for multi-hop clustered WSNs. They used a modified linear task-mapping algorithm augmented by a task-migration algorithm to reduce the costs of inter-task communication. Although their task-scheduling algorithm guarantees that real-time deadlines are met, it is not suitable for large-scale applications. However, since sensor nodes are stationary and do not affect their environment, this approach is not applicable to WSANs where actors are usually mobile. In fact, the nature of the tasks in WSANs is different from the actions that sensor nodes execute in WSNs, and this is one of the reasons that scheduling approaches presented for WSNs (for example, those presented in [16]) are not applicable to WSANs.

Given this background on task allocation and scheduling in sensor networks, we propose Scate, which is a new scalable, time-aware, and energy-aware actor task-allocation algorithm for WSANs with the objective of prolonging the lifetime of a network, shortening the makespan of resulting task schedules, and balancing the loads on all actors. Our algorithm is significantly different from aforementioned algorithms in that it considers scalability, load balancing, and time and energy constraints as cumulative effective parameters in the scheduling of tasks to proper actors.

III. Problem Statement

The task assignment problem in WSANs is often considered the allocation of n tasks T_i (i=1, 2,…, n) to m actors A_j (j=1, 2,…, m), wherein the schedule of each task amounts to the allocation of one or more time slots to one or more actors [17]. As in many other heterogeneous distributed systems, the response time of WSANs is measured in terms of makespan [10], [11], which denotes the overall CT of all tasks in the network. To achieve a minimum makespan, an optimized mapping of tasks to actors is required that is an NP-complete problem to solve [8], [10], [17], [18].

In WSANs, the scheduling problem amounts to the mapping of a set of tasks to a set of actors to minimize the CT of a specific task or the makespan of resulting task schedules. Schedulers can consider other parameters such as load balancing, system throughput, service reliability, service cost, and system utilization as well. Furthermore, schedulers can work either in the instant mode, wherein each task arriving at the scheduler is allocated by the scheduler to an actor, or in the bunch mode, wherein tasks arriving at the scheduler are first put into a set by the scheduler and then scheduled collectively.

There are also some atomic tasks that are not decomposable. These independent tasks, known as meta-tasks [19], do not communicate with each other. Some schedulers use approximate execution time of meta-tasks to assign them to actors that can perform the tasks fastest. Such algorithms are known as minimum execution time algorithms [8], [10], [11].

There are other scheduling algorithms that are categorized as minimum CT (MCT) algorithms. They assign tasks to actors that are expected to complete those tasks the fastest [8], [10], [11]. Although each task is assigned to an actor that is expected to complete the task the fastest, the actor may not execute the task in minimum time since MCT depends on the availability of the actor and the current workload of the actor.

As mentioned before, OLB is a well-known scheduling algorithm with the aim of balancing the loads on actors. It assigns tasks to actors in an arbitrary order without considering the CTs or the execution times of tasks. It tries to create a load-balanced network by keeping all actors as busy as possible [7]-[10]. Its ignorance of expected task execution time in its assignment policy may lead to long makespans of resulting schedules though [10]. Depending on the type of implementation, the complexity of OLB is variable. For example, the complexity of the implementation of OLB reported by Maheswaran and others [11] that examines m number of actors to find an assignment is $O(m)$.

In contrast to OLB, tasks are assigned to actors based on execution times of tasks in both the max-min and min-min algorithms. The min-min algorithm estimates the execution and CTs of all unscheduled tasks (U) on each actor first. It then repeatedly selects a task with the shortest CT and assigns it to an actor with the least execution time until all tasks in U are assigned [11], [12].

In the pseudo code of the min-min algorithm in Fig. 1, a_j represents the expected time that actor A_j can perform a task after finishing the execution of all its previously assigned tasks. To determine C_{ij}, that is the CT of task T_i on actor A_j, the expected execution time E_{ij} of T_i on A_j is added to the availability time a_j of A_j. The entire C_{ij} is placed in the C matrix that is used to find an appropriate actor for each task. The i-th row of this matrix is scanned for each T_i, an appropriate actor with the fastest expected CT (ECT) is chosen, and vector a and matrix C are accordingly reorganized. This same process is repeated for all other unassigned tasks.

Although the max-min and min-min algorithms have similar structures, they differ in their selections of actors to assign them tasks. In the max-min algorithm, once an actor that can provide the earliest CT for a task is determined, the task T_k with the maximum earliest CT is chosen and mapped to the corresponding actor [11]. Therefore, at line 6 of Fig. 1, “minimum” should be replaced by “maximum.” It seems that the max-min algorithm yields a quicker response time than the min-min algorithm, especially in cases where the number of

```
1.  For all tasks $T_i$ in meta-task set $M_v$
2.      For all actors $A_j$
3.          $C_{ij} = E_{ij} + a_j$
4.  Do until all tasks in $M_v$ are assigned
5.  For each task in $M_v$ find the earliest CT and the actor that
    can perform the task
6.      Find the task $T_k$ with the minimum earliest CT
7.      Assign the task $T_k$ to actor $A_l$ that gives the earliest CT
8.      Delete the task $T_k$ from $M_v$
9.      Update $a_l$
10.     Update $C_{il}$ for all $i$
11. End Do
```

Fig. 1. Pseudo code for min-min algorithm.

small tasks is more than the number of large tasks. In other words, the max-min algorithm has a better performance than the min-min algorithm when the number of smaller tasks is higher than the number of larger tasks.

Both the max-min and min-min algorithms have the same time complexity equal to $O(mn^2)$, where n denotes the number of unscheduled tasks and m denotes the total number of available actors [18].

IV. Assumptions

We assume a semi-automated architecture WSAN containing a singleton sink node. Sensors are only responsible for gathering information from the environment and transmitting it to the sink. The sink node determines the proper tasks to be executed by actors and then dispatches tasks to proper actors that are selected based on an allocation algorithm.

More precisely, we assume a WSAN with m actors A_j (j=1,…, m) that should perform n tasks T_j (i=1,…, n). Tasks are independent, non-preemptive, and not decomposable. Actors can search the whole network without any restriction on routing hops. Actors are idle at first. The total time taken by the actor A_j with no load at the time of assignment to execute a task T_i is called the execution time of task T_i on actor A_j, and it is denoted by E_{ij}. The time taken by actor A_j to finish the execution of task T_i is called the CT, and it is denoted by C_{ij}. The CT of a task T_i on an actor A_j is greater or at best equal to its execution time on the same actor, that is, $C_{ij} \geq E_{ij}$. The expected time that actor A_j can perform a task after finishing the execution of all its previously assigned tasks is called the availability time of actor A_j, and it is denoted by a_j. In fact, C_{ij} denotes the sum of the availability time (a_j) of actor A_j and the execution time (E_{ij}) of task T_i on A_j, that is, $C_{ij} = (E_{ij+}\ a_j)$.

It is important to note that our choice of semi-automated architecture for WSANs does not restrict the applicability of Scate to real large-scale WSANs. As stated by Liu and others [20], some large-scale sensor networks may be single hop in terms of wireless communication needed for reporting. A sink, or several sinks, can be mobile and move around the network. This allows them to get close to sensors so that report collecting can be done in a single hop. In other examples, embedded sensors can move toward a fixed sink. For example, sensors can be embedded into sea mammals to trace their locations over time. When a sea mammal approaches a fixed base station, reports can be downloaded.

Another example that has used semi-architected WSANs in a real large-scale application is the real-time climate control and monitoring of greenhouses [21]. Given the above evidence, we also show through experiment the applicability of our proposed algorithm to large-scale as well as to small-scale networks in section VI. Given that the scope of our proposition has been limited to semi-architecture WSANs, we have only compared our proposed algorithm with related algorithms that have considered this architecture, to ensure a fair comparison.

V. Proposed Scate Task Allocation Algorithm

According to the assignment policy of the min-min algorithm that is based on allocating small tasks to fast actors, suppose T_1 is the first task that is assigned to actor A_j. It is expected that A_j finishes T_1 in the least possible time compared to other actors. Other remaining tasks are also mapped to A_j if the total execution time of tasks mapped to A_j is less than their execution time on other actors. Because the min-min algorithm assigns tasks to the fastest actors, it can shorten the makespan of resulting task schedules in WSANs only if the difference between the execution time of tasks is short. Otherwise, large tasks may be mapped to slower actors significantly increasing the makespan.

To resolve the above weakness and get a shorter makespan in cases where the number of large tasks is less than the number of small tasks, we propose to use the max-min algorithm instead. The bigger the difference is between the number of large tasks and the number of small tasks, the bigger the difference is in the makespan of resulting schedules under these two algorithms. In such cases, the expected makespan can be approximated by the execution time of the largest task.

Because the min-min algorithm assigns small tasks prior to assigning large ones, it leads to a longer makespan compared to the result from using the max-min algorithm. In contrast, assigning the largest task to the fastest actor, as with the max-min algorithm, provides a better chance for simultaneous execution of small tasks on other actors. Furthermore, the max-min algorithm leads to a better balance of loads on actors [10], which is critically desired in WSANs to prevent network partitioning.

Generally, load balancing is a desirable property in

```
1.  For all tasks $T_i$ in the meta-task $M_v$
2.    For all actors
3.      $(C_{ij}, e_{ij\text{-remain}})=[(E_{ij}+a_j), e_{ij\text{-remain}}=e_{j\text{-now}}-(C_{ij}\times\partial_j)]$
4.      If $(Z>1)$ Then $Z=\text{abs}(Z)$; $W=1$; // $Z$ as defined in (4)
5.      Else $W=\text{abs}(1/Z)$; $Z=1$
6.  Do until all tasks in $M_v$ are mapped
7.    For $Z$ down to 1
8.      For each task in $M_v$ find the earliest CT an actor can perform
        the task
9.      Find a task $T_h$ with maximum earliest CT
10.     If no actor $A_l$ with maximum energy can be found
        to complete the execution of the task $T_h$ earliest
11.     Then Assign $T_h$ to the fittest actor $A_l$ with highest value of
12.           $\beta\times(e_{hl\text{-remain}})$ / $(CT_h)$
13.     Else Assign $T_h$ to actor $A_l$ that yields the earliest CT and
        has maximum energy
14.     End if
15.     Delete task $T_h$ from $M_v$; Update $a_l$
16.     Update $(C_{ij}, e_{ij\text{-remain}})$ for all $i$
17.   For $W$ down to 1
18.     For each task in $M_v$ find the earliest CT an actor can
        perform the task
19.     Find a task $T_k$ with minimum earliest CT
20.     If no actor $A_l$ with maximum energy can be found to
        complete the execution of the task $T_k$ earliest
21.     Then Assign $T_k$ to the fittest actor $A_l$ with highest value of
22.           $\beta\times(e_{kl\text{-remain}})$ /$(CT_k)$
23.     Else Assign $T_k$ to actor $A_l$ that yields the earliest CT and
.       and has maximum energy
24.     End if
25.     Delete task $T_k$ from $M_v$; Update $a_l$
26.     Update $(C_{ij}, e_{ij\text{-remain}})$ for all $i$
27. End Do
```

Fig. 2. Pseudo code for Scate.

distributed systems, especially in small-scale distributed systems requiring a small makespan. However, load balancing can lead to a large makespan in large-scale distributed systems. This is not true in our proposed approach because the scheduling algorithms in different situations are properly chosen in line with the restrictions of WSANs and because there is a focus on minimizing the total CTs of tasks.

Figure 2 represents the proposed algorithm. Scate builds a matrix *c*' of ordered pairs whose first element C_{ij} denotes the expected CT of task T_i by actor A_j and the second element $e_{ij\text{-remain}}$ denotes the expected remaining energy of actor A_j after performing the task T_i. To start with, each actor has its maximum energy and its workload is zero. Hence, in the first round, the largest task in the set of tasks requiring the longest execution time is allocated to an actor that can complete the task earlier than others. In the next rounds, the next largest and smallest tasks in the set of unallocated tasks (UT_i) are assigned to the proper actors that can complete the tasks in the minimum amount of time and also save maximum energy compared to other actors.

Although the minimization of the total CT of tasks is the main goal of Scate, the mapping of tasks to the fastest (highest throughput) actors without considering the remaining energies of actors can be fatal because actors may run out of energy, resulting in network partitioning [22]. This problem may not be easily resolved since WSANs are mostly deployed in harsh environments making recharging of sensors and actors very difficult and sometimes impossible. It is necessary to consider remaining energies of actors when scheduling tasks.

To allocate tasks to actors, Scate uses (1) to estimate the CT of each UT_i on every actor. It also uses (2) to estimate the approximate remaining energy of each actor:

$$C_{ij}=(E_{ij}+a_j), \tag{1}$$

$$e_{ij\text{-remain}}=e_{j\text{-now}}-(C_{ij}\times\partial_j). \tag{2}$$

In (1), a_j indicates the expected time for actor A_j to perform a new task after finishing the execution of all its previously assigned tasks and E_{ij} denotes the expected execution time of T_i on A_j. To evaluate the remaining energy of each actor, which is the next effective parameter in the allocation of tasks, the definition of (2) includes C_{ij} as a derivative of (1). In (2), ∂_j represents the average energy consumption of actor j per second and $e_{j\text{-now}}$ represents the remaining energy of actor A_j at allocation time.

Scate tries to allocate a new task to an actor that is expected to have more energy after performing that task than other actors and at the same time can perform the new task earlier than other actors. However, there may be some cases where no actor is found to fulfill these requirements. In such cases, an actor with maximum energy or an actor that can complete the task earliest may be chosen arbitrarily to perform the new task. However, to avoid blind arbitration, Scate takes into account the fact that different applications may have different precedence on energy saving and shortening of the total CT of tasks. Some applications may have higher priority for energy saving, some applications may have higher priority for shortening the CTs of tasks, and some applications may be neutral, considering energy saving and CT to be of equal importance. We denote this precedence by β. Taking this application-dependent precedence parameter into account, Scate selects an actor A_j out of all actors in the network to execute a new task T_i if this actor has the highest fitness value defined by

$$\textit{Fitness Metric}=\beta\times(e_{ij\text{-remain}})/(CT_i), \tag{3}$$

wherein $e_{ij\text{-remain}}$ represents the expected remaining energy of actor A_j after performing the task T_i and CT_i represents the expected CT of the task T_i on actor A_j.

So in cases where an actor with a high amount of remaining energy and the ability to complete a new task earlier than other actors cannot be found, Scate considers the reciprocal of the

CT over the remaining energy of each candidate actor and weights this ratio by the precedence of the application that is given and the constant during the run of the application, then selects the actor with the highest fitness value. For example, if an application considers the amount of an actor's remaining energy to be twice as relevant as task CT, then β is equal to 2 and any new task is assigned to the actor whose remaining energy is highest after dividing its CT of the previous task by its expected CT of the new task and multiplying by 2.

It may seem that when there is a single run of a given application in a WSAN, a constant value of β for this application in (3) is redundant and that the ratio of remaining energy of an actor over the CT of a task can on its own determine the fitness degree of the actor to run the task. However, this is true only if we consider that there is always a single run of an application in the network. In case there are many concurrent runs of the same application or different applications in the network, the precedence parameter β plays its real role in deciding on the best actor to execute a task by weighting the above ratio for different runs. It should be pointed out that a single application may also be given different precedence for different concurrent or single runs based on pertaining changes, for example, in its mission or context. In fact, the very strength of Scate in comparison with other related algorithms lies in weighting the remaining energy over the CT ratio with β allowing changes in precedence of applications running concurrently in the network. It is this very feature that makes Scate adaptable to different applications and to different runs of the same application. This is why we claim Scate is scalable. We illustrate this claim in section VI.

Returning to the pseudo code of Scate in Fig. 2, in each iteration, Scate allocates the largest task and then the smallest task to actors by making suitable tradeoffs between the CTs of tasks and the remaining energies of actors. The total value of C_{ij} and $e_{ij\text{-remain}}$ is placed in the matrix c', which is used for finding an appropriate actor for each task.

To find an appropriate actor for each task, the i-th row of matrix c' is scanned for each T_i, and an appropriate actor with the shortest CT and maximum remaining energy is selected. If the difference between the number of large tasks and the number of small tasks is large, the use of the min-min algorithm or the max-min algorithm alone can lead to a long makespan as well as starvation of large or small tasks.

We divide tasks into large tasks and small tasks based on the ECT of tasks. Hence, we define parameter Z as the ratio of the number of large tasks (NT_{large}) over the number of small tasks (NT_{small}) as

$$Z = (\text{number of large tasks})/(\text{ number of small tasks}). \qquad (4)$$

Specifically, a task that has a greater ECT than the average of ECTs is placed in the group of large tasks and other tasks are placed in the group of small tasks. Finally, upon every allocation, the task with the maximum ECT and then the task with the minimum ECT are assigned to corresponding actors that have the highest remaining amount of energy and can perform within the least amount of time, and this process continues for the remaining tasks.

Table 1. CTs of tasks on two actors.

CT_i / Actor	CT_1	CT_2	CT_3	CT_4	CT_5
A_1	2	3	1	4	7
A_2	17	19	20	12	18

Table 2. Required energy for execution of tasks on two actors.

RE_i / Actor	RE_1	RE_2	RE_3	RE_4	RE_5
A_1	2	3	3	4	5
A_2	5	6	7	8	9

As a simple example, let us consider there are two heterogeneous actors in a WSAN with the same communication bandwidth on their links but with different throughputs and speeds of movement. Let us also consider that there are five tasks, T_1, T_2, T_3, T_4, and T_5, to be allocated to actors by the sink for execution and that the CT of each task (CT_i) on each actor is given as in Table 1 and the required energy for the execution of each task on each actor is given as in Table 2. As mentioned before, CTs are determined by the sink based on E_{ij} and the distance between the actors and the locations where tasks should be performed. Also, the expected required energies for executions of tasks on A_1 and A_2 are evaluated by the sink as given in Table 2. However, since the main goal of this example is to show the operations of the three mentioned algorithms compared to those of Scate, and to keep the example simple, we have focused on the allocation results of scheduling algorithms.

We assume that actors are idle at first. Since the main goal of the min-min and max-min algorithms is to minimize the makespan of resulting task schedules, and to describe the flow of these two algorithms compared with the flow of Scate, we assume that the same energy level is required by each actor A_1 and A_2 to perform each task. Hence, to have a fair comparison, we consider β to be equal to one.

Figure 3 includes four Gantt charts that show the total CTs of five tasks tabulated in Tables 1 and 2 when the min-min algorithm, the max-min algorithm, OLB, and Scate are deployed. As Fig. 3 shows, OLB yields the maximum

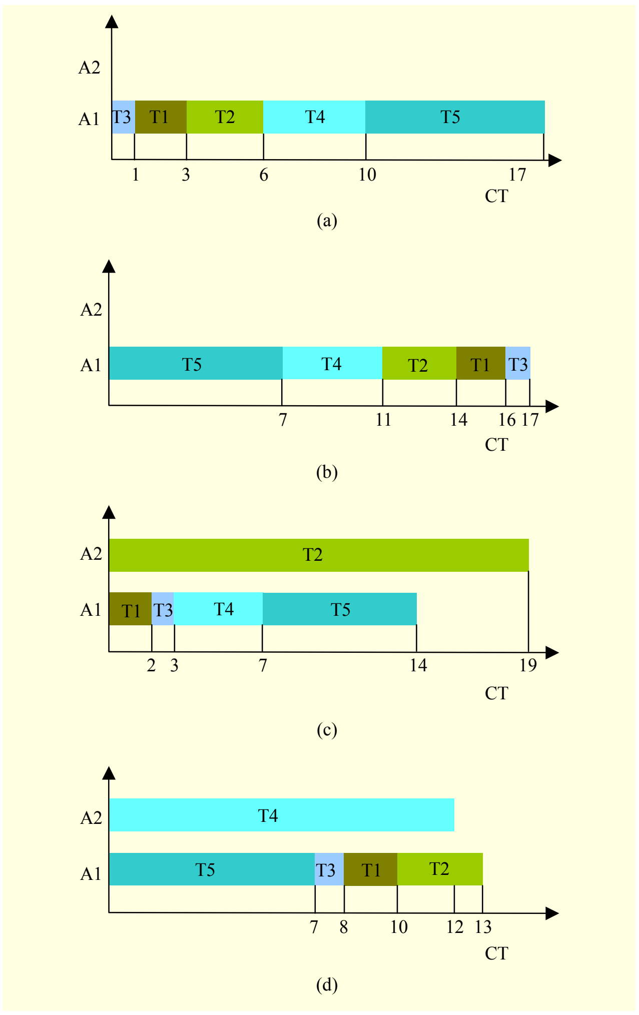

Fig. 3. CTs of five tasks by two actors with different schedules: results of (a) min-min algorithm, (b) max-min algorithm, (c) OLB, and (d) Scate.

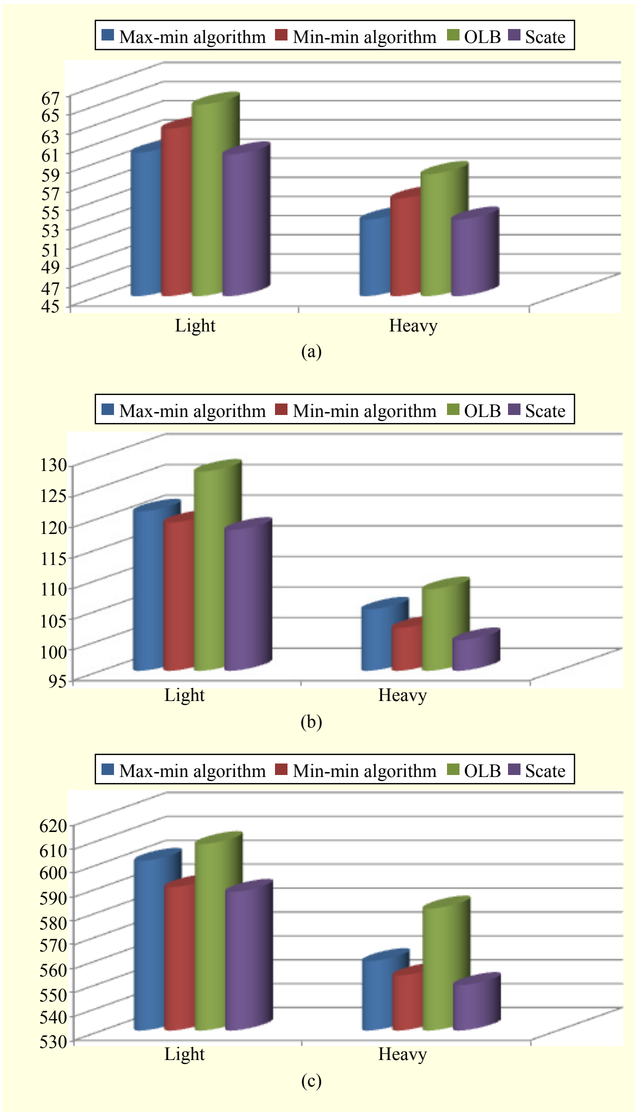

Fig. 4. Experiment results in small scale: (a) $Z<1$, (b) $Z=1$, and (c) $Z>1$. Vertical axis shows total CTs of tasks and horizontal axis shows workloads.

makespan (19 s) and Scate yields the minimum makespan (13 s). The max-min and min-min algorithms yield the same makespan (17 s) but with a different order of scheduled tasks. In this example, although OLB gives the worst makespan, it provides better balance of load than the max-min and min-min algorithms, resulting in a longer actor lifetime. From this perspective, Scate performs best in yielding both the smallest makespan and the best overall balance of loads on actors.

Scate performs best in the assumed simple example, but it may not perform best in other scenarios. Therefore, we need to present a more general and thorough evaluation, which is the subject of the next section.

VI. Experiment Results

To show the efficiency of our approach, in a typical scenario, Scate is compared with OLB, the min-min algorithm, and the max-min algorithm in terms of total CT of tasks and total remaining energies of actors. In addition, to study the effect of scale on the efficiency of Scate, we perform simulations in both large and small scales in two different settings. In the small scale (Setting I), we assume a two-dimensional square field, 10 m×10 m, containing 100 sensor nodes with a 1 m transmission range and 10 actor nodes. We assume that the tasks to be executed by actors are independent and that actors can browse the whole network with no restrictions on routing hops. The primary energy of each actor is assumed to be the same as others and equal to 50 J. In the large scale (Setting II), we assume a two-dimensional square field, 100 m×100 m, containing 1,000 sensors with a 10 m transmission range and 10 actor nodes. We assume that the tasks to be executed by actors are independent and that actors can search the whole network with no constraints on routing hops. The primary energy of each actor is assumed equal to 50 J.

In our simulations, we assume that each actor runs only a single task at any time and that tasks are selected from three different classes with slow, medium, and fast service rates. We also assume that tasks are independent and are submitted to actors by the sink. To evaluate the effect of the workload on the network, light and heavy loads are considered, wherein workload is defined as the number of tasks to be executed by the actors. To achieve a better evaluation, we consider all possible values of parameter Z in our simulations and assume that the priority of time is the same as the priority of energy such that β=1. Figures 4 and 5 depict the CTs of tasks under four scheduling algorithms in both Setting I and Setting II, respectively.

As Fig. 4(a) shows, in the small scale setting, where the number of small tasks is bigger than the number of large tasks, the max-min algorithm results in a smaller makespan. However, as shown in Fig. 4(b), when the number of small tasks and large tasks are nearly the same, the min-min algorithm leads to a smaller makespan compared to OLB and the min-min algorithm, while Scate results in the smallest makespan.

Figure 4(c) depicts the results in the case of Z being bigger than 1, wherein Scate has the smallest makespan in both heavy and light workloads and the min-min algorithm results in a smaller makespan than OLB and the max-min algorithm. However, experiment results in Setting I shown in Fig. 4, wherein the required time to pass the distance between a selected actor A_j and the location of the task T_i allocated to actor A_j is not much compared with E_{ij}, demonstrate that, in small scales, OLB results in the worst makespan while Scate results in the best makespan.

Figure 5 shows the makespan of the four algorithms in Setting II. As Fig. 5(a) shows, when the number of small tasks is larger than the number of large tasks, Scate results in the shortest makespan compared with other approaches in both heavy and light loads; thereafter, the min-min algorithm results in a shorter makespan than OLB and the max-min algorithm. When the number of large tasks is increased (Z=1), the min-min algorithm and Scate result in nearly the same makespan, which is shorter than the makespan resulting from other algorithms in both light and heavy workloads (Fig. 5(b)). Figure 5(c) shows the case where Z is bigger than 1 and the workload is light. Although the makespan resulting from the min-min algorithm is much better than the makespan resulting from OLB and the max-min algorithm, it is a little bigger than the makespan resulting from Scate. The results are similar in both light and heavy workloads, but the difference between the

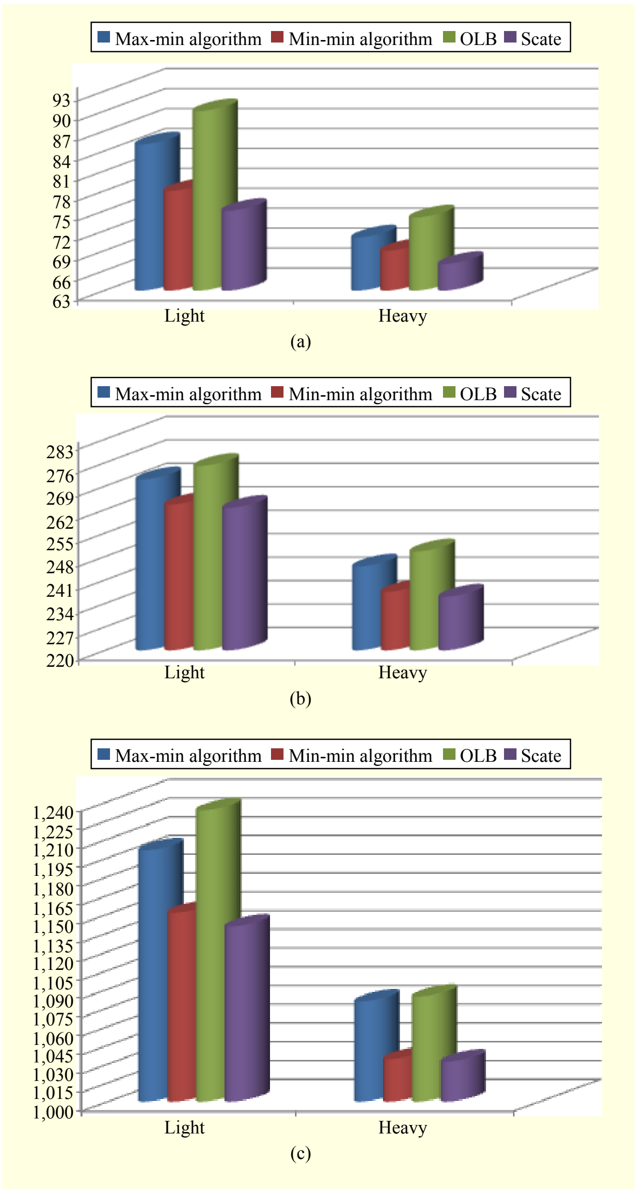

Fig. 5. Experiment results in large scale: (a) Z<1, (b) Z=1, and (c) Z>1. Vertical axis shows total CTs of tasks and horizontal axis shows workloads.

makespan of the min-min algorithm and Scate is a little bit less.

For the sake of clarity, the total CT of tasks in each of the four algorithms in small and large scales are averaged and shown in Figs. 6 and 7, respectively. As these figures show, Scate is more efficient in terms of makespan than OLB, the max-min algorithm, and the min-min algorithm since the total CT of tasks by Scate is less than other algorithms in the two chosen settings. To evaluate Scate in terms of total remaining energies of actors, we compare it with the other three mentioned algorithms (Fig. 8). The averages of total remaining energies of actors in both small and large scales are higher in Scate than in other algorithms demonstrating the preeminence

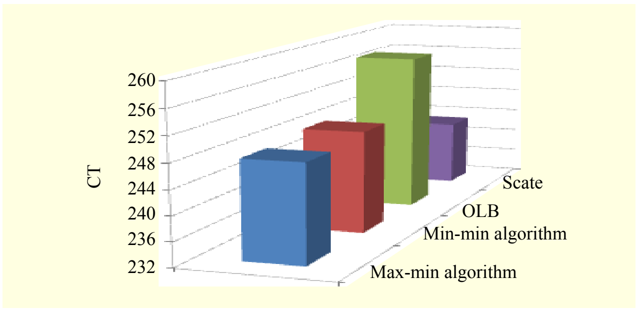

Fig. 6. Average CTs in small scale case.

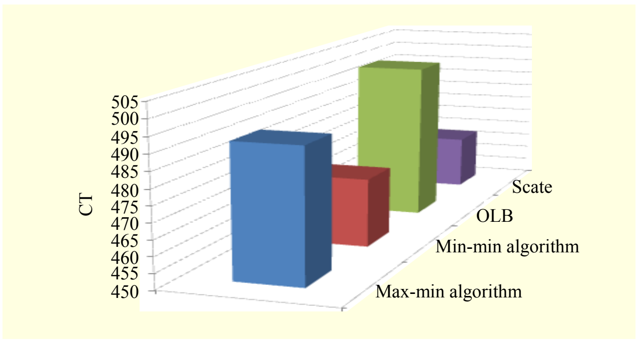

Fig. 7. Average CTs in large scale case.

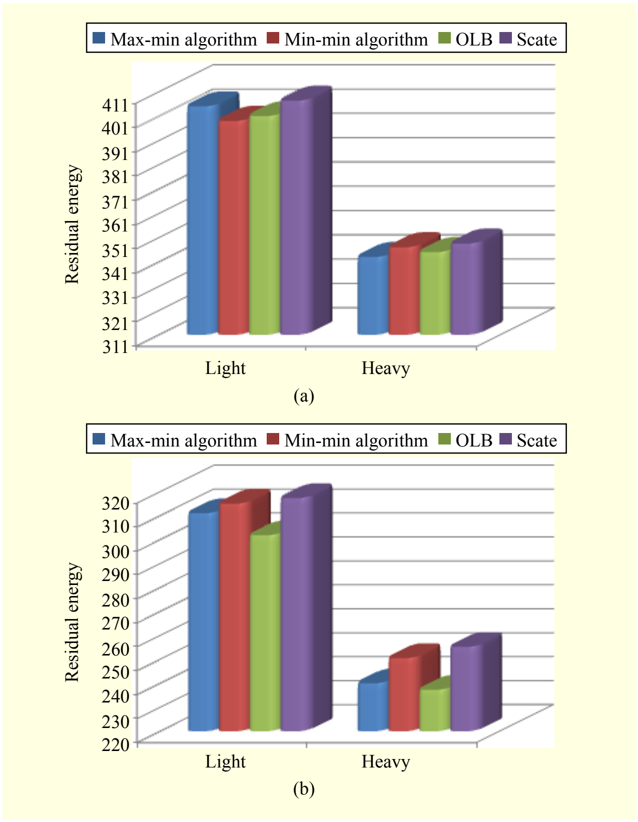

Fig. 8. Residual energies of actors (a) in small scale case and (b) in large scale case. Vertical axis shows total residual energies of actors and horizontal axis shows workloads on actors.

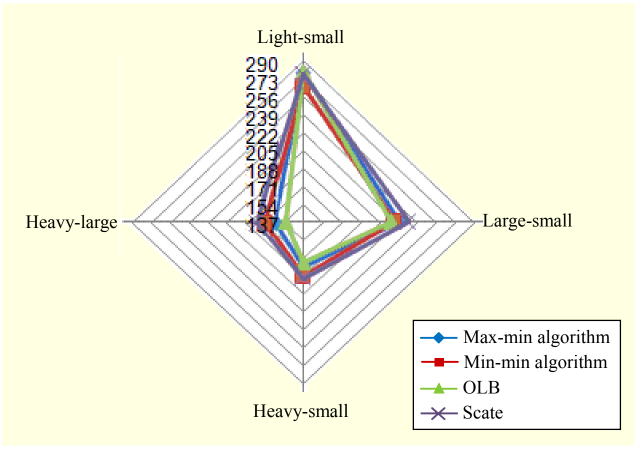

Fig. 9. Average remaining energies of actors.

of Scate in preserving energy and increasing the lifetime of actors.

In the small scale case, when the workload is light, the max-min algorithm and Scate consume the same level of energy but quite less than the other two algorithms. Under heavy workloads, the max-min algorithm has the worst energy consumption while the min-min algorithm and Scate have lower or nearly the same consumption. In the large scale case, Scate results in the maximum residual energy of actors, with the min-min algorithm being the next in row. OLB and the max-min algorithm have the worst consumption rate and the least energy preservation.

The total remaining energies of actors when using different algorithms are rescaled for the sake of clarity. As shown in Fig. 9, the maximum residual energy resulting from Scate is 280, and the residual energies of the other three algorithms are rescaled with respect to this amount.

As Fig. 9 shows, the four different cases studied are considered vertices of a regular foursquare. The vertices of this figure are related to light-small, heavy-small, light-large, and heavy-large, in which light and heavy represent light and heavy workloads, respectively, and small and large represent small and large scales of network, respectively.

The deployment of OLB in small scale networks and the use of the min-min algorithm for large scale networks are not appropriate for applications with high priority for energy (Figs. 6 and 7). Scate is a better choice because of its flexibility in allowing the adjustment of the β parameter to save the energy of the actors.

As mentioned before, some applications may have a higher priority for energy saving while others may have a higher priority for shortening the CTs of tasks, but neither OLB nor the max-min and min-min algorithms consider any priority between energy saving and shortening the CTs of tasks. Hence, to have a fair comparison between these algorithms and Scate,

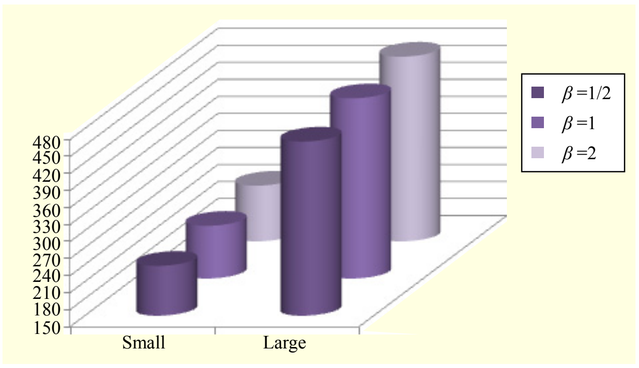

Fig. 10. Average CTs running Scate with different values of β. Vertical axis shows averages of CTs and horizontal axis shows scale.

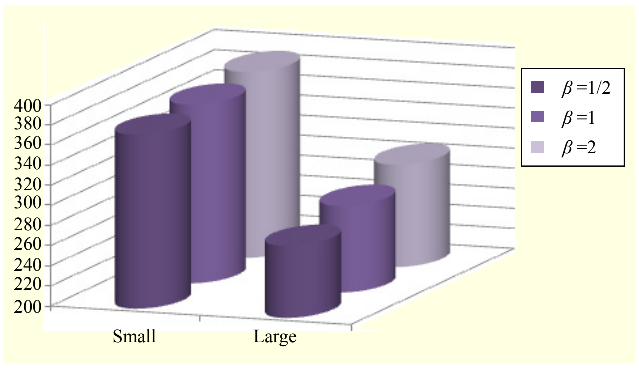

Fig. 11. Residual energies of actors running Scate with different values of β. Vertical axis shows average total residual energies of actors and horizontal axis shows scale.

we assume equal priorities for time and energy, such that β=1. It is obvious that Scate operates better than these three mentioned algorithms in applications that have different priorities regarding makespan and energy saving. Nevertheless, to show the effects of time and energy priorities on the performance of Scate through experiment, we repeat the above simulations with the same conditions but with different values of β. To do this, we firstly assume that the priority of energy saving is twice that of shortening the CTs of tasks (β=2), and then we repeat the simulations with β=1/2 for the cases where the priority of energy saving is half of the priority of shortening the CTs of tasks. Figures 10 and 11 show the results of running Scate with three values of β. Figures 10 and 11 show that when the priority of energy saving is higher than the priority of shortening the CTs of tasks, Scate rightly saves more energy compared to the other two cases, resulting in improved network lifetime. Similarly, Scate yields a shorter makespan in cases where the priority of time is higher than energy. It is thus shown that Scate can be adapted to different applications that require the same or different priorities between shortening the CTs of tasks and energy savings of actors. However, as the results of simulations in both large and small scales with different conditions show, Scate is a more favorable choice than OLB, the min-min algorithm, and the max-min algorithm because of its flexibility in allowing the adjustment of the β parameter to save the energy of actors and to minimize the network makespan.

VII. Conclusion and Future Works

We proposed a scalable, time-aware, and energy-aware algorithm for allocating tasks to actors in WSANs. Reducing the total CT of tasks and increasing the residual energies of actors simultaneously was the dual objective of the proposed algorithm called Scate. Results of our experiments in both large and small scale networks under different conditions showed that Scate yields the shortest makespan and the highest total remaining energy of actors in almost all different conditions compared to three well-known traditional scheduling algorithms, namely, the max-min algorithm, the min-min algorithm, and OLB. The results also showed longer network lifetime and more balanced workload on each actor when using Scate compared to when these three algorithms were used.

Consideration of task deadlines in support of real-time applications, fault-tolerance of actors and communications links, and other QoS parameters are the open issues of Scate.

References

[1] F. Xia et al., "Wireless Sensor Actuator Network Design for Mobile Control Applications," *Sensors*, vol. 7, 2007, pp. 2157-2173.

[2] I.F. Akyildiz and I.H. Kasimoglu, "Wireless Sensor and Actor Networks: Research Challenges," *Ad-Hoc Netw.*, vol. 2, 2004, pp. 351-367.

[3] A. Nayak and I. Stojmenovic, *Wireless Sensor and Actuator Networks: Algorithms and Protocols for Scalable Coordination and Data Communication*, Hoboken, New Jersey: Wiley Press, 2010.

[4] I. Stojmenovic, *Energy Conservation in Sensor and Sensor-Actuator Networks, Wireless Ad-Hoc Networking: Personal-Area, Local-Area, and Sensory-Area Networks*, S.-L. Wu and Y.-C. Tseng, Eds., Auerbach Publications, 2007, Ch. 4, pp. 107-133.

[5] X. Cao et al., "Building Environment Control with Wireless Sensor and Actuator Networks: Centralized vs. Distributed," *IEEE Trans. Ind. Electron.*, vol. 57, no. 11, 2010, pp. 3596-3605.

[6] J. Chen et al., "Distributed Collaborative Control for Industrial Automation with Wireless Sensor and Actuator Networks, *IEEE Trans. Ind. Electron.*, vol. 57, no. 12, 2010, pp. 4219-4230.

[7] R. Armstrong, D. Hensgen, and T. Kidd, "The Relative Performance of Various Mapping Algorithms is Independent of Sizable Variances in Run-Time Predictions," *Proc. IEEE Int.*

Workshop Heterogeneous Comput., 1998, pp. 79-87.

[8] R.F. Freund et al., “Scheduling Actors in Multi-User, Heterogeneous, Computing Environments with SmartNet,” *Proc. IEEE Int. Workshop Heterogeneous Comput.*, 1998, pp. 184-199.

[9] R.F. Freund and H.J. Siegel, “Heterogeneous Processing,” *IEEE Comput.*, vol. 26, 1993, pp. 13-17.

[10] T.D. Braun et al., “A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems,” *Parallel Distrib. Comput.*, vol. 61, 2001, pp. 810-837.

[11] M. Maheswaran et al., “Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems,” *Parallel Distrib. Comput.*, vol. 59, 1999, pp. 107-121.

[12] Y. Tian, E. Ekici, and F. Ozguner, “Energy-Constrained Task Mapping and Scheduling in Wireless Sensor Networks,” *Proc. IEEE Int. Conf. Mobile ad-hoc Sensor Syst.*, 2005, pp. 8-16.

[13] Y. Yu and V.K. Prasanna, “Energy-Balanced Task Allocation for Collaborative Processing in Wireless Sensor Networks,” *Mobile Netw. Appl.*, vol. 10, 2005, pp. 115-131.

[14] M. Okhovvat, M. Sharifi, and H. Momeni, “Task Allocation to Actors in Wireless Sensor Actor Networks: An Energy and Time Aware Technique,” *Procedia Computer Science*, vol. 3, 2011, pp. 484-490.

[15] S. Shivle et al., “Static Mapping of Subtasks in a Heterogeneous Ad-Hoc Grid Environment,” S*ymp. Parallel Distrib. Process.*, 2004.

[16] M.H.A. Awadalla and R.R. Darwish, “Quality of Service Constrained Task Mapping and Scheduling Algorithm for Wireless Sensor Networks,” *Computer Eng. Research*, vol. 2, 2011, pp. 8-18.

[17] P. Brucker, *Scheduling Algorithms*, 5th ed., Springer Press, 2007.

[18] O.H. Ibarra and C.E. Kim, “Heuristic Algorithms for Scheduling Independent Tasks on Non-Identical Processors,” *J. ACM*, vol. 24, 1977, pp. 280-289.

[19] K.S. Golconda and F.O. Zguner, “A Comparison of Static QoS-based Scheduling Heuristics for a Meta-Task with Multiple QoS Dimensions in Heterogeneous Computing,” *Symp. Parallel Distrib. Process.*, 2004.

[20] H. Liu, Y.W. Leung, and X. Chu, Eds., *Ad-hoc and Sensor Wireless Networks: Architectures, Algorithms and Protocols*, Bentham Science Publishers, 2009, Ch. 1.

[21] A. Cenedese, L. Schenato, and S. Vitturi, “Wireless Sensor/Actor Networks for Real-Time Climate Control and Monitoring of Greenhouses,” *ING-INF/04 Automatica*, 2008. Available at paduaresearch.cab.unipd.it/1045/01

[22] B.H. Calhoun et al., “Design Considerations for Ultra-Low Energy Wireless Microsensor Nodes,” *IEEE Trans. Comput.*, vol. 54, no. 6, 2005, pp. 727-740.

Mohsen Sharifi is an associate professor of software engineering in the School of Computer Engineering of Iran University of Science and Technology. He directs a distributed systems research group and laboratory. He is mainly interested in the engineering of distributed systems, solutions, and applications, particularly for use in various fields of science. The development of a true distributed operating system is on top of his wish list. He received his BSc, MSc, and PhD in computer science from Victoria University, Manchester, UK, in 1982, 1986, and 1990, respectively. His website is http://webpages.iust.ac.ir/msharifi/.

Morteza Okhovvat received his BSc and MSc in computer engineering from Mazandaran University in 2008 and Iran University of Science and Technology in 2011, respectively. He is a member of the Iranian Elite Foundation. His research interests include distributed systems, wireless sensor actor networks, task scheduling algorithms, modeling, and performance analysis in the context of wireless sensor actor networks.