Topology Aware Task Allocation and Scheduling for Real-Time Data Fusion applications in Networked Embedded Sensor Systems

Baokang Zhao[1,2],Meng Wang[1], Zili Shao[1*]
Jiannong Cao[1], Keith C.C. Chan[1]
[1] Department of Computing
Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong, China
{csbzhao,csmewang,cszlshao,
csjcao,cskcchan}@comp.polyu.edu.hk

Jinshu Su[2]
[2] School of Computer Science
National University of Defense Technology
Changsha, Hunan, P.R. of China
sjs@nudt.edu.cn

Abstract

In networked embedded sensor systems, data fusion is a viable solution to significantly reduce energy consumption while achieving real-time guarantee. Emerging data fusion applications demand efficient task allocation and scheduling techniques. However, existing approaches can not be effectively applied concerning both network topology and wireless communications. In this paper, we formally model TATAS*, the Topology-Aware Task Allocation and Scheduling problem for real-time data fusion applications, and show it is NP-complete. We also propose an efficient three-phase heuristic to solve the* TATAS *problem. We implement our technique and conduct experiments based on a simulation environment. Experimental results show that, as compared with traditional approaches, our technique can achieve significant energy saving and effectively meet the real-time requirements as well.*

1 Introduction

Energy-efficiency is the paramount concern for networked embedded sensor systems. In these systems, many emerging applications have real-time requirements, especially for critical applications such as battlefield surveillance, home health care, and so on. Data fusion, that reduces the amount of data volume of sensor nodes, is widely used to improve the energy efficiency while achieving the real-time guarantee for applications in networked embedded sensor systems. Data fusion applications are usually partitioned into small tasks in order to be executed in a distributed manner. To process these tasks, the task allocation and scheduling, that assigns tasks to different sensor nodes and determines their execution order and communications, is the most critical part. Therefore, it becomes an important problem to develop efficient task allocation and scheduling technique to minimize energy consumption for applications with data fusion in networked embedded sensor systems.

Existing approaches focus on task allocation and scheduling for traditional computer systems; however, they cannot be effectively applied to networked embedded sensor systems concerning both network topology and wireless communications. Since topology is one of the most important issues in networked embedded sensor systems, in this paper, we focus on developing a topology-aware task allocation and scheduling scheme for data fusion applications.

For traditional high performance computing and internet based grid computing systems, many topology-aware task allocation and scheduling techniques have been developed. In the communication models of these systems, the processing units are fully connected via wired networks[6, 16], or some special topology such as chains[12], trees[5], 2D-mesh, 3D-Torus[1],etc. However, in networked embedded sensor systems, irregular network topology is highly affected by the wireless channel. Thus, the existing task allocation and scheduling techniques for traditional computer systems cannot be directly applied to the networked embedded sensor systems.

For single-hop wireless sensor networks, the task allocation and scheduling problem has been addressed and investigated in [7, 14, 15, 13, 10]. In [7], Heemin et al. presented a simulated annealing framework for energy-efficient task allocation and migration in sensor networks. The Energy-balanced Task Allocation(EbTA) algorithm is introduced by Yang et al. in [14]. They developed an Integer Linear Programming (ILP) formulation for this problem, and proposed a three-phase heuristic. In [15], the EcoMapS algo-

*The corresponding author

1533-2306/08 $25.00 © 2008 IEEE
DOI 10.1109/RTCSA.2008.30

rithm is proposed for mapping and scheduling tasks jointly in single-hop clustered WSN. In [10], Yuan et al. presented RT-Maps, which can conserve energy with a real-time deadline guarantee. The above techniques concentrate on information processing in one hop range. In realistic networked embedded sensor systems, nodes are usually randomly deployed in a wide area, and they have to work in multi-hop environments. Therefore, in this paper, we focus on developing effective techniques for multi-hop environment.

Recently, research efforts have been put to multi-hop environments. In [9], Yuan et al. proposed a multi-hop in-network processing algorithm. However, the most important issues, including network topology, are not covered in this solution. Without considering the topology, this work cannot be effectively used to solve the task allocation and scheduling problem as the physical locations of different sensor nodes have great impact on the task allocation and scheduling process.

In this paper, we focus on developing the topology-aware energy efficient task allocation and scheduling technique for networked embedded sensor systems. To the best of our knowledge, our work is the first one to deal with the task allocation and scheduling problem in multi-hop sensor systems considering the underlying network topology. Our main contributions are summarized as follows:

- We study and address the Topology-Aware Task Allocation and Scheduling problem (TATAS) for data fusion applications in networked embedded sensor systems, which is vital for reducing the system-level energy consumption and achieving the real-time guarantee. Different from existing approaches, we explore the location of sensor node and the underlying topology issues under multi-hop environments, and propose a system-level model incorporated with the application model, network model, and energy model.
- We propose a three-phase approach to solve the TATAS problem. In this approach, the different phases of task allocation and scheduling algorithm are performed sequentially, and the DVS technique is used to further improve the energy efficiency.
- We have implemented this work in a simulation environment, and compared it with existing approaches. The experimental results show that our technique can significantly reduce energy consumption, meet real-time requirements, and improve the system lifetime.

The rest of this paper is organized as follows. In section 2, we give the motivational example. In section 3, we formally define the TATAS problem. The proposed algorithm is presented and discussed in section 4. The experimental results and analysis are provided in section 5, and the conclusion is given in section 6.

2 Motivational Example

In this section, we motivate the *TATAS* problem by showing a real life scenario for data fusion. To demonstrate the impact of *topology issues*, we consider a surveillance application with real-time requirements.

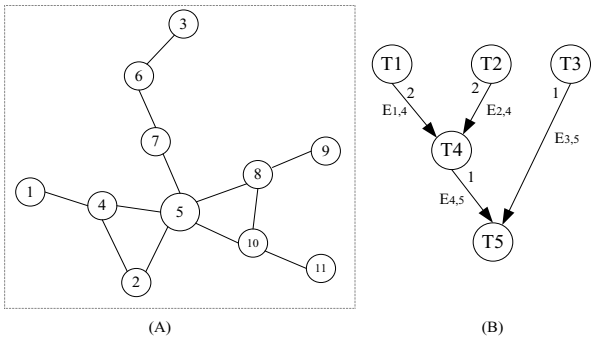

Figure 1. (A) Network Topology. (B)DAG

As shown in Fig. 1(A), a networked embedded sensor system is deployed outdoors. It consists of 11 heterogeneous sensor nodes, and each node is labeled with an integer as indicated in the middle of the node. The edge connecting pairwise nodes indicates that they can communicate directly. This connected graph builds a *network topology*.

A real-time object surveillance application, as represented as a Directed Acyclic Graph(DAG) in Fig. 1(B), is planed to be executed on this system. In this application, the DAG consists of five *computation tasks*. It starts with sensing events (T_1, T_2, T_3), and performs object classification algorithms (T_4, T_5). Since the sensing phenomenon is always be sensed by specific sensors, the staring sensing tasks are initially assigned to those nodes. In this case, tasks T_1, T_2 and T_3 are initially assigned to nodes $1, 2, 3$, respectively.

In this application, the data communication activity between computation task T_i and T_j is denoted as E_{ij}, and it is shown as a edge between T_i and T_j in Fig. 1(B). A communication activity is also regarded as a *communication task*. We denote the number of transmission packets on each communication task as *communication load*, and it is marked as a number in the edge. For instance, the communication load of $E_{1,4}$ is 2.

In this scenario, the *task allocation* problem is to assign computation tasks (T_4, T_5) to sensor nodes, and the *task scheduling* problem is to determine their execution sequence. As mentioned in the previous section, there are many literatures focus on topology free approaches for these two problems. Here, we will demonstrate existing approaches have poor performance without considering network topology.

We first consider the task allocation problem. In existing approaches [15, 9, 14], they attempt task allocation with all sensor nodes in the network. These attempts are inefficient, and they can be improved if considering the underlying network topology. We take task allocation of T_4 as an example. Since task T_1 and T_2 is assigned to node 1 and 2, sensor nodes located far away from 1 and 2, such as 3, 9, 11, can be excluded from the feasible mapping set of Task T_4.

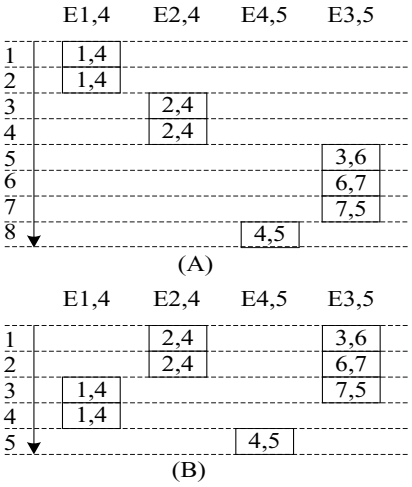

Figure 2. results of comm scheduling

Task allocation phase always follows with task scheduling phase. We will show that the network topology is also very critical in task scheduling. Suppose T_4 and T_5 are assigned to node 4 and 5 in task allocation phase, and we aim at scheduling all the communication tasks. Existing approaches only consider the collision for neighboring communications. For example, since node 5 is in the communication range of node 2, $E_{2,4}$ and $E_{3,5}$ are regarded as a collision, and can not be executed simultaneously. However, in multi-hop environment, they can be scheduled to execute simultaneously. We show their difference in Fig. 2. In this figure, rows represent time slots for packet transmission, rectangles denote time occupation for communication tasks. Fig. 2.(A) shows the scheduling result of traditional scheduling approaches [9]. Since $E_{1,4}$,$E_{2,4}$ and $E_{3,5}$ are regarded as collisions, they are scheduled to be executed in a sequential order. However, when the underlying network topology is considered, we can obtain an optimal scheduling as shown in Fig. 2.(B). In this example, with considering the network topology, the schedule length can be improved with 37.5% through exploiting potential packet level parallelism.

3 Problem Statement

Motivated by the above example, we model *TATAS*, the Topology Aware Task Allocation and Scheduling problem for energy efficient real-time data fusion applications. In this section, we first introduce some system assumptions, and then formally define the related models including application model, network model and energy model. After that, the problem statement of *TATAS* is given.

- System Assumptions

We assume the following system assumptions:

1. Real-time data fusion applications are executed in a networked embedded sensor system, which consists of heterogeneous nodes. These nodes form a logical multi-hop computational environment.
2. All wireless modules are single channel, and conforms the collision free model[13]. TDMA protocols are preferred to avoid link collisions.
3. The network topology information is available.

- Application Model

A real-time data fusion application is modeled as a DAG $TG = (V_T, E_T, V_{ET}, vw, ew, TC, \mu)$. V_T denotes the computational tasks, and vw represents tasks' computational overhead. E_T consists of communication tasks, and ew represents their overhead. In data fusion applications, entry tasks are always executed on specified entry nodes, and these tasks are denoted as V_{ET}. TC is the timing constraint for the application, and μ represents the data fusion ratio. The data fusion ratio is introduced to the denote the average fusion factor, which represents the magnitude of accumulative sum of outgoing packets relative to those of incoming.

- Network Model

The network topology is modeled as a connected graph $NG = (V_G, E_G, cc, dw)$. V_G is the set of sensor nodes, E_G is the set of communication edges. cc is the maximum computation load of each node, and dw denotes the communication distance between two neighborhood nodes.

- Energy Model

We adopt the same energy consumption model as [11].

$$P_{cpu} = \alpha C_L * V^2 * f + I_{leak} * f \quad (1)$$

$$P_{TX}(d) = E_{elec} + \varepsilon_{amp} d^{\partial} \quad (2)$$

$$P_{RX} = E_{elec} \quad (3)$$

In the CPU power model, α, C_L and I_{leak} are processor dependent parameters, while V and f denote the working

voltage and frequency, respectively. The transmitting and receiving power of the wireless module are shown in equation 2 and 3. E_{elec} and ε_{amp} are electronic parameters, d is the transmitting distance, and $2 \leq \partial \leq 4$.

- Task Allocation and scheduling

The goal of task allocation is to assign tasks to sensor nodes. Let m represents the *mapping function* of a allocation, that is, task T_i is assigned to node $m(T_i)$. After the allocation, the communication edge is mapped to the routing path between nodes. Suppose $E^{(m)}_{comp}(T_i)$ denotes the energy consumption of task T_i, $E^{(m)}_{comm}(e_{ij})$ denotes the energy consumption of communication edge e_{ij}, and $L(m)$ is the finish time of application. The task scheduling problem is to determine executing sequences of these computation and communication tasks.

- Problem Definition

Given DAG and network topology, the objective of *TATAS* is :
Minimize:

$$E^{(m)}_{total} = \sum_{t_i \in V_T} E^{(m)}_{comp}(t_i) + \sum_{e_{i,j} \in E_T} E^{(m)}_{comm}(e_{i,j}) \quad (4)$$

Subject to:

$$L(m) \leq TC \quad (5)$$

We have proved the NP-completeness of *TATAS* through a polynomial reduction from subgraph isomorphism problem [8].

4 The Proposed Scheme

Since the *TATAS problem* is NP-complete, we propose an efficient three-phase heuristic to solve it. In this section, we first give an overview of this scheme, and then focus on the topology aware task allocation and scheduling algorithms.

4.1 System Overview

As shown in Fig. 3, our proposed scheme consists of three phases: *initialization phase*, *task allocation and scheduling phase* and *DVS phase*.

In the *initialization phase*, we first sort the task set in a topological order. This topological ordering of the task set ensures that, the precedence constraints between tasks maintains when we perform task allocation. The voltage levels for all computation tasks are set to the highest one, and other global information is initialized in this phase.

In the *task allocation and scheduling phase*, task allocation and scheduling process runs iteratively to obtain a

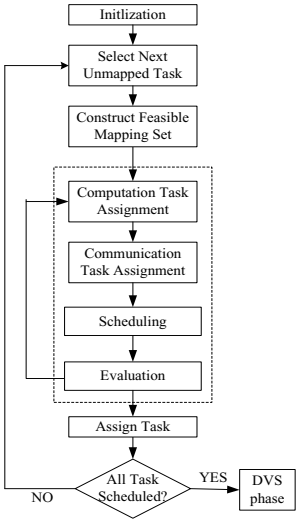

Figure 3. Overview of the proposed solution

feasible mapping for all tasks. Since all computation tasks have been ordered in a topological order, the algorithm executes by performing partial mapping of tasks in this order. When the next unmapped computation task is selected, the algorithm builds its candidate node set considering the network topology, tries to assign current unmapped task to one unused node in the candidate set, and maps related communication tasks to routing paths. Thereafter, the communication scheduling algorithm executes to maximize the parallel packet delivery. When all candidate mappings are obtained, we evaluate these mappings, and choose the best one as the final result. This process continues until all task allocation and scheduling are done.

After the task allocation and scheduling phase, *DVS* techniques are applied to further reduce energy consumption of computation tasks. Since DVS techniques have been widely studied in existing literatures, in our current implementation, we adopt a *DVS* technique similar to [14]. Due to space limitation, we leave details in [8].

In general, this scheme provides a extendable open platform. Existing topology-unrelated techniques, such as *DVS* and *DMS* [13], can be easily integrated into this platform. In the following subsections, we concentrate on topology-aware issues, including task allocation algorithm and communication scheduling mechanism.

4.2 Task allocation algorithm

Given an unmapped task, task allocation algorithm determines its node mapping. Recall that in existing approaches, the candidate set consists of all sensor nodes in the network. For each node in this candidate set, evaluating algorithms

should be executed for evaluating the mappings. This process is very costly, and which makes existing approaches very time-consuming. Thus, it is critical to reduce the candidate node set. We solve this problem through exploiting topology information.

Since all tasks have been ordered in the topological order, it is deterministic that all precedent tasks of the next pending unmapped task t have been allocated. We denote the precedent node set of task t as $PS(t)$. In *TATAS*, both energy consumption and packet delivery latency are relevant to $PS(t)$. Therefore, we want to reduce the size of candidate node set through quantifying the impact of the precedence node set. Inspired by the mechanical models in physics, we propose a novel *potential field model*.

In the *potential field model*, the impact of a precedence node $S_i \in P(t)$ on target node t is estimated as:

$$PNI(S_i, t) = D(S_i) * d(S_i, t) \tag{6}$$

Where $D(S_i)$ is the data volume to be transferred from S_i to t, and $d(S_i, t)$ denotes the distance of the path from S_i to t. We introduce *Universal Attractive Force*(UAF), a fitness function for constructing the candidate set. The UAF on node t is:

$$UAF(t) = \sum_{S_i \in PS(t)} PNI(S_i, T) \tag{7}$$

In our algorithm, all sensor nodes are evaluated with UAF. And after that, a random variable in [0.3-0.6] is generated to determine the proportion of nodes to be included in the candidate set. This algorithm greatly reduce the size of candidate set, and in our experiments, it achieves more than 50% improvement in executing time comparing with traditional approaches.

4.3 Communication Scheduling

During the *task allocation and scheduling phase*, for data fusion applications, packets scheduling is required to transmit packets from some designated sensor nodes to one target node, and this many-to-one communication pattern belongs to convergecast. The convergecast problem has attracted a lot of research efforts recently [3, 4]. However, all these literatures aim at collecting sensed identical amount of data from all nodes in the network, while in TATAS, we should transmit heterogeneous amount of data on partial sensor nodes to one target node, it is more complicated. We denote this particular problem as the *HPCS problem*. To our best, this is the first attempt in generating scheduling for this problem.

Since the HPCS problems highly depend on different network topologies, we propose several algorithms with them. We first start with a simple linear topology, and then consider multi-line case and tree topologies. In these algorithms, a scheduling is generated by exploiting both *intra-line parallelism* and *inter-line parallelism*.

4.3.1 An efficient algorithm for Simple Linear Path

We first consider a simple case when the network topology is a *simple linear path*. This happens when all paths from source nodes $PS(t)$ to the target node t form a linear topology, and the target node t is on one end of the path.

We start from studying the interference model for wireless communication. In single channel wireless environment, each node has its interference range, data transmissions within this range may conflict with others. As shown in Fig. 4(A), when V_3 transmits to V_4, V_2 hears this transmission, and it will cause the transmission from V_1 to V_2 fails. Fortunately, we can still schedule multiple transmissions simultaneously. Data transmissions which are two hops away, such as (V_1,V_2) and (V_4,V_5), can transmit in the same time. We exploit this kind of parallelism as *intra-line parallelism*. This kind of parallelism, also called *in-line pipelining*, is available when the hop count between two concurrent data transmissions is bigger than 3.

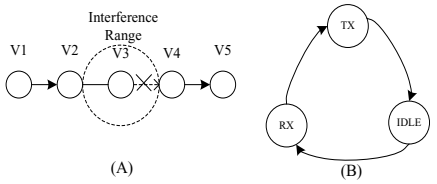

Figure 4. Interf model and State Trans Map

In order to exploit in-line parallelism, we propose an algorithm named *HPCS_Line*. Similar with [3], we assume that each sensor node has an exclusive state in each time slot. The potential states include *TX, RX and IDLE*. Node is able to transmit data in *TX* state, receive data in *RX* state, and do nothing in *IDLE*. In each time slot, sensor nodes in the line transit their state to another in the next time slot as shown in Fig. 4(B).

In the beginning, each node is assigned to an initial state based on its hop count from the target node. Assume the hop count of node i is $h(i)$, the initial status of node i is :

$$status(i) = \begin{cases} RX, & h(i) \mod 3 = 0 \\ TX, & h(i) \mod 3 = 1 \\ IDLE, & h(i) \mod 3 = 2 \end{cases} \tag{8}$$

To illustrate the running of our algorithm, an example is demonstrated in Fig. 5. There are five sensor nodes in a linear network, and the status of each node is signed on top. The node marked with "t" in the right side is the destination

node. In each time slot, the remain packet of each node is depicted in the center of itself, and the arrow between nearby nodes indicates the packet transmission in current time slot.

```
   Input: A line network L with N hops, where V[N] is the
          source node with maximum hop counts, and V[0] is
          the target node.
   Output: a scheduling for line convergecast.
1  init system variables;
2  if line length of L ≥ 3 then
3      assign V[N] with state TX;
4      for I from N-1 to 1 do
5          assign V[I] with prev_state(V[I + 1]);
6      end
7  end
8  while not all data collected do
9      /* epilog phase, no potential parallelism */
10     if current line length of L < 4 then
11         transmit packets from the node with biggest hop
           count to its neighbor node, until no data left in this
           node;
12     end
13     if CurrentTimeSlots ≤ lineLen − 2 then
14         /* in prolog phase, try to TX greedy */
15         foreach node V[K] do
16             if N have data and no collision then
17                 transmit data of N;
18             end
19         end
20     else
21         /* enter the kernel phase of pipelining*/
22         foreach node V[K] in L do
23             if V[K]'state is TX and have data then
24                 send data forward;
25             end
26         end
27     end
28     foreach node V[K] in L do
29         transit the state of V[K] to next_state;
30     end
31     update the lineLen value of line L;
32     TotalTimeSlots++;
33 end
```

Algorithm 1: The HPCS_Line algorithm

As shown in Algorithm.1, our proposed algorithm HPCS_Line initializes the system variables and the status of each node in L in steps 1-7. Thereafter, the algorithm runs in three phases: the *prolog* phase, the *kernel* phase and the *epilog* phase. In the prolog phase (step 14-19), since the intermediate nodes may have data to transfer, we try to let them transmitting data according to their position in the line. For instance, in time slot 0 to time slot 2, when node 4 transmits data, node 1 can transmit data without collision, so it transmits data forward. In the kernel phase (step 21-26), the pipelining has been established, so only the nodes with data and in TX state can transmit. This kind of in-line pipelining is shown in time slot 3. Where the line length is smaller than 4, no parallelism can be exploited, the algorithm will enter the epilog phase (step 9-12), just transmitting packets as step 11 from time slot 4 to time slot 7. During all phases above, the status of all the node in the line and the line length information are updated from step 28 to 32. Since this is the maximum available parallelism in the line, we obtain an optimal solution.

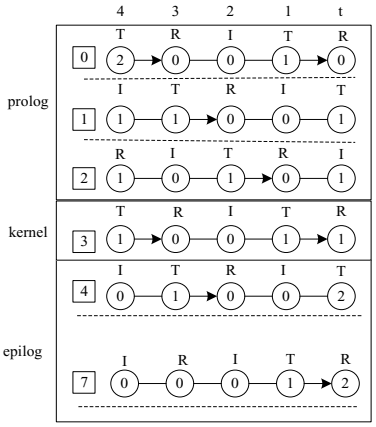

Figure 5. An Intra-Line scheduling example

4.3.2 An efficient algorithm for Multi-Line Case

We now consider a more complicated case when paths from source nodes to the target node are disjointed with each other. We call this case as a *Multi-Line Case*. In this case, packet transmissions among different paths are independent, *Inter-Line Parallelism* is achievable among these paths, and it brings potential packet delivery inside each line to be in parallel when the communications among multiple lines are independent with each other.

For multi-line case, an algorithm named *HPCS_MLINE* is proposed to exploit both intra and inter-line parallelism, and shown in algorithm 2. An example is given in Fig. 6. *HPCS_MLINE* starts with an initial phase from step (1-7). In these steps, step (2) is the most critical to perform *Inter-Line Parallelism*. This parallelism relies on the arbitration of the target node. Since multiple paths may transmit data packets to the target node, we should define priorities on these paths. Clearly, the priority of one path highly depends on its length (determining possible intra-line parallelism) and the total duration of all its packets. Formally, assume the hop count of path P is $h(P)$, each node u on the path P has packets $d(u)$, and the priority of path P is $p(P)$, we have $p(P_i) > p(P_j)$ when $h(P_i) > 3$ and $h(P_j) < 4$, or $\sum_{k=1}^{h(P_i)} d(k) * k \geq \sum_{k=1}^{h(P_j)} d(k) * k$.

Input: A multi-line network $N = (L, t)$ with M lines, where L[I] denotes the Ith line, t is the target node. Each node V[I][J] in L[I] has data and state info.
Output: a scheduling for multi-line convergecast.

```
1  init variables including TotalTimeSlots and lineLen;
2  Sort the lines using line compare rules;
3  foreach line L[I] in the network do
4      if current line length of L ≥ 4 then
5          assign V[I][J] with state TX as in single line;
6      end
7  end
8  while not all data collected do
9      /* target node act as an arbiter to select an line */
       foreach line L[I] in the pre-determined order do
10         if first node V[L][0] of line L has data then
11             if (length of L[I] > 3) && (status of V[L][0]
               == TX ) then
12                 selectLine = current line;
13             else
14                 if the length of L[I] ≤ 3 then
15                     selectLine = current line;
16                 end
17             end
18         end
19         if any line is selected then
20             break;
21         end
22     end
23     /* intra-line parallelism */
24     foreach line L[I] in the network do
25         if current line is selected && L[I] ≤ 3 then
26             continue;
27         end
28         call HPCS-Line;
29         Update all node status and the line length info;
30     end
31     TotalTimeSlots++;
32 end
```

Algorithm 2: The HPCS_MLine algorithm

From step 8 to 32, the algorithm starts scheduling. In each time slot, the algorithm will run in two-phases: arbitrating phase and inter-line parallelism phase. In the arbitrating phase, the target node T selects one path in the predetermined order. Once a line is selected, it transmits one packet to the target node T. For example, in time slot 1, L1 have data to transfer, and its first node V[1][0] is in the status of TX, thus it is granted to transmit data. In the inter-line parallelism phase, each line schedules its packet transmission as in HPCS-Line. There is one exception in step 25. If hop count of the selected line is smaller than 4, it will be idle for no intra-line parallelism.

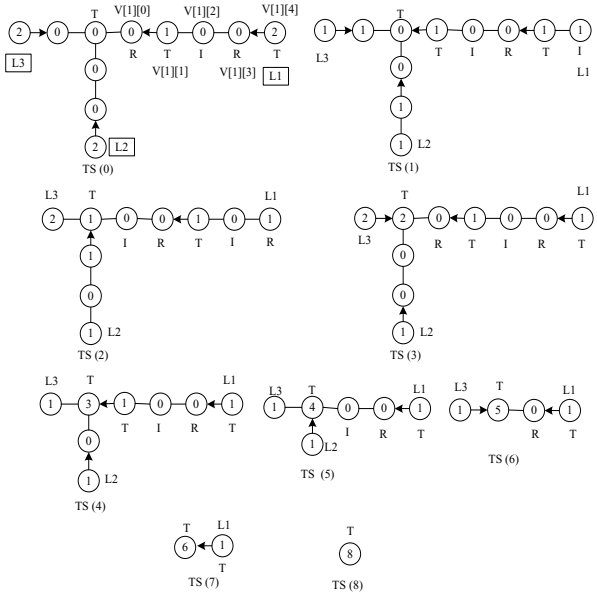

Figure 6. the Inter-Line schedule example

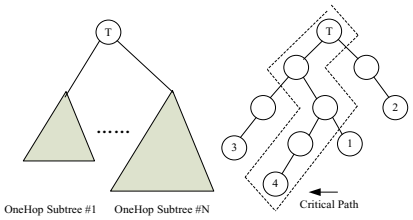

Figure 7. the subtree and critical path

4.3.3 An effective algorithm for Tree

We now consider a more general case when some paths overlap with others, the network topology becomes a tree, and *HPCS_TREE*, an algorithm shown in Fig. 3, is proposed based on *HPCS_MLINE*.

Its basic idea is exploiting both the *inter-line parallelism* and *intra-line parallelism*. As indicated in Fig.7, to fully utilize algorithms in *HPCS_MLINE*, the algorithm first build the target node T's one hop subtree, then *inter-line parallelism* can be achieved among subtrees. In step (1-5), it divides all the paths from the source node set to target node t into several groups according to the one-hop subtree. In steps (7-12), the algorithm will try to find a critical path in each subtree, and adopt the comparison rules in previous multi-line case directly. The critical path acts as a base line, and the nodes' status are initialized. From steps (14-22), in each one hop subtree, the packets in the critical path are forwarded according to the single line forwarding rule, and

other paths in the same subtree can also transmit data using the HPCS_line up to the cross point between current path and the critical path if no collsion. The algorithm runs until all packets has been collected to the target node.

```
   Input: A network G = (T, S, t), where T is a tree, S is the
          precedent node set, which includes all non-leaf nodes
          of T, and t is the target node(root of T)
   Output: a scheduling for tree convergecast.
 1 /* first find all the one-hop-sub tree of T */
 2 Construct the direct sub node set S, which includes all sub
   nodes of t.;
 3 foreach node s in S do
 4 |   classify all the paths from s to t according to its direct
   |   subnode of t;
 5 end
 6 /* find the critical path for each subtree */
 7 foreach node s in S do
 8 |   find the critical path for subtree ST(s);
 9 |   if this critical path is with hop count > 3 then
10 |   |   init its status;
11 |   end
12 end
13 /* do critical path first scheduling */
14 while not all data collected do
15 |   foreach sub-tree ST(s) do
16 |   |   schedule the critical path with HPCS_line;
17 |   |   foreach other path inside ST(s) do
18 |   |   |   if no collision with critical path then
19 |   |   |   |   transmit it greedy using HPCS_MLINE;
20 |   |   |   end
21 |   |   end
22 |   end
23 end
```

Algorithm 3: The HPCS_TREE algorithm

5 Experimental Results and Analysis

In order to evaluate the performance of our proposed approach, we build a simulation environment, and conduct extensive simulations in this simulation environment. In this section, we first introduce our simulation environment, and show our experiments results and perform detailed analysis.

5.1 The Simulation Environment and Simulation Parameters

We build a simulation environment as shown in Fig. 8. The major modules of the simulation environment includes: DAG customizer, Network Topology generator, and TATAS module. The DAG customizer is based on the *TGFF*[2] DAG tool. We modified the *TGFF* tool to generate DAG for data fusion applications. In our experiments, we set the number of entry tasks to be 8, the maximum in degree and out degree to be 3 and 5, respectively. The computation workload and communication throughput are randomly chosen within the range of (100Kcps, 600Kcps) and (500bits, 1000bits), and the battery capacity of a sensor node is set to 1000Amh. Network Topology Generator(*NTG*) is used to generate random network topologies. In NTG, the network is assumed to be deployed in a 1km*1km area, and the sink node is placed in the center of the area. The parameters of wireless module of sensor nodes are configured with bandwidth setting to 250 kbps, and the communication range is 100 meters. The energy consumption adopts the parameters of μAMPS [11].

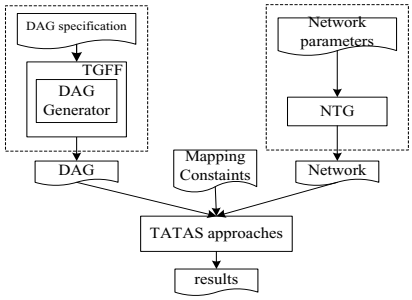

Figure 8. the simulation environment

5.2 Results and Analysis

We compare our algorithm with DCA[11]. DCA is a typical approach for traditional task allocation and scheduling. It executes entry tasks on corresponding sensor nodes, and finishes other tasks on the sink node. We extend DCA with multi-hop support by constructing routing paths from entry sensor nodes to the sink node. In the following part, We denote *TATAS-3H* as our proposed 3-phase heuristic, and *DCA-MH* as the Multi-Hop extension version of DCA.

The goals of our experiments are (1) to compare the performance of our proposed *TATAS-3H* approach against *DCA-MH* approach; and (2) to evaluate the impact of several critical system parameters, including the timing constraint, the number of tasks, the network scale, and the fusion ratio. Our evaluation metrics mainly concentrate on the energy consumption, schedule length and the network lifetime. The network lifetime is defined as the time after the first node runs out its battery. Several experiments are conducted on our simulation environment. All simulation results presented here correspond to the average of 1000 times of random DAG, network topology combinations. For clarify, we group the results into two categories: the impact of the number of tasks and the impact of network scale.

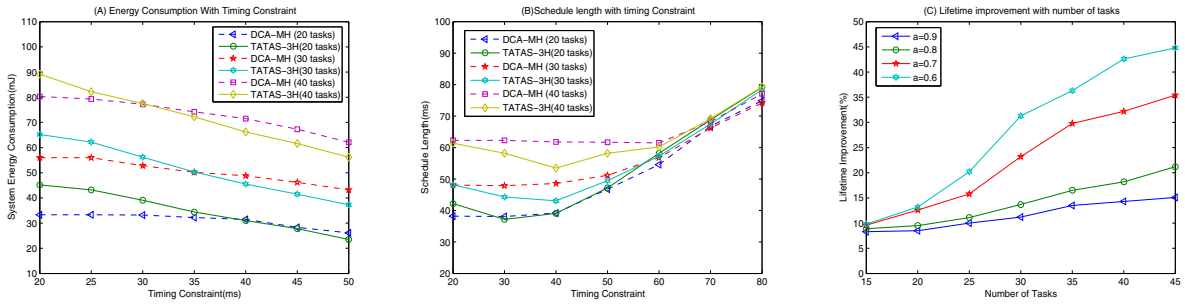

Figure 9. impact of the number of tasks

5.2.1 Impact of the number of tasks

We first study energy consumption with the impact of number of tasks. Experiments are conducted with 60 nodes, 0.9 fusion ratio, 20 to 40 tasks with the increment step of 10, and the timing constraint varying from 20 to 50 with the increment step of 5. From the results shown in Fig. 9(A), the system energy consumption of both approaches increase with the number of tasks. When the timing constraint is small, our *TATAS-3H* has higher energy consumption than *DCA-MH*. However, with the increase of timing constraint, the energy consumption of our *TATAS-3H* decrease rapidly than that of *DCA-MH*. The reason is that our communication scheduling algorithms can greatly reduce the time for packet delivery, and more opportunities can thus be exploited by DVS techniques.

The schedule length with different number of tasks is shown in Fig. 9(B). In these experiments, the range of timing constraints has been changed to vary from 20ms to 80ms in increments of 10. From the results, we observe that the schedule length of both approaches increase with the number of tasks. Both approaches starts to produce valid schedules with the increment of timing constraint due to the efficiency of DVS techniques. When the timing constraint is larger than 30ms, we see our *TATAS-3H* is better than *DCA-MH*. This efficiency come from our proposed communication algorithms. We also noticed that the schedule length of *TATAS-3H* decreases when timing constraint is small. This is because our proposed scheduling algorithms seeks to reduce the schedule length for fulfilling realtime requirements.

To compare the network lifetime improvements of *TATAS-3H* against *DCA-MH*, we also conduct several experiments with 60 nodes, 15 to 45 tasks with the increment step of 5, the timing constraint TC = 35ms, and fusion ratios varying from 0.6 to 0.9 with the increment step of 0.1. As shown in Fig. 9(C), *TATAS-3H* achieved significant lifetime improvements comparing with *DCA-MH*. The reason is that our proposed *DCA-MH* algorithm allocates tasks among a lot of computing sensors, yet *DCA-MH* process all high level tasks in the cluster head node. The lifetime improvements of *TATAS-3H* increase dramatically with the increments of number of tasks, it indicates the efficiency of our task allocation and scheduling algorithms. Furthermore, the system lifetime is highly improved with the decrease of data fusion ratio. This result shows that the improvements of data fusion can greatly enhance the system lifetime.

5.2.2 Impact of the network scale

We first evaluate the system energy consumption with different network scale. We conduct experiments with 20 tasks, 0.6 fusion ratio, 20 to 40 nodes varying with the increment step of 10, and the timing constraint varying from 20 to 50 with the increment step of 5. As shown in the Fig. 10 (A), both approaches can reduce system energy consumption with the increment of timing constraint. When timing constraint is small, comparing with *TATAS-3H*, the energy consumption of *DCA-MH* grow slowly with the increment of number of nodes. The reason is that the average hop count increase with number of nodes. Since *DCA-MH* transfer raw data directly, its energy consumption is insensitive with hop count, while *TATAS-3H* allocates the tasks to multiple nodes. However, when timing constraint is bigger than 40ms, our *TATAS-3H* is able to preserve more energy with the increment of number of nodes by exploiting more parallelism through communication scheduling.

Results of schedule length with different network scale is shown in Fig. 10(B). We use the same parameters as the above experiments. It can be observed that *DCA-MH* is not sensitive with network scale, and *TATAS-3H* is better than *DCA-MH*. When timing constraint is small, the schedule length of *TATAS-3H* increase with the increment of number of nodes. However, when the timing constraint changes from 20ms to 30ms, it is better for large network scale. The reason is that the communication scheduling algorithm can take the advantage of large number of nodes to reduce the schedule length.

The improvement in network lifetime is obtained with configurations of 0.6 to 0.9 data fusion ratio, 20 tasks, 20

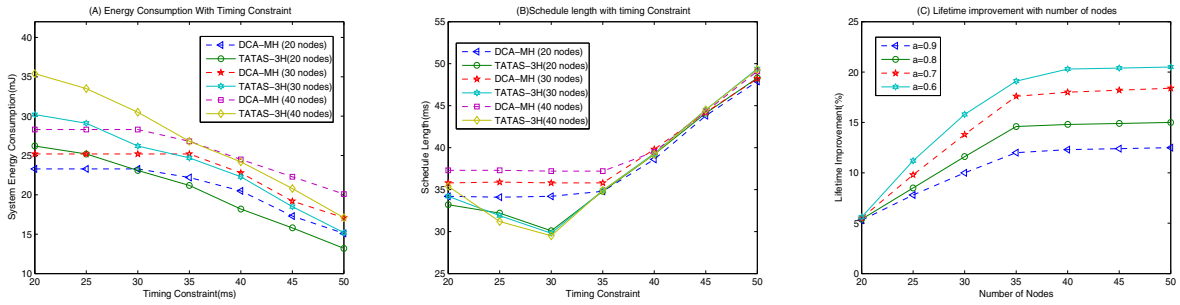

Figure 10. impact of the network scale

to 50 nodes with the increment step of 5, and the timing constraint varying from 20 to 50 with the increment step of 5. As shown in Fig. 10 (C), when number of nodes is small, network lifetime is enhanced with the increments of number of nodes. The reason is that *TATAS-3H* can allocate tasks with more nodes, which improve the network lifetime by better load blance. However, we can observe that this increment slows down after the number of nodes exceeds 40. The reason is that the allocation of tasks have reached the threshold, and the network lifetime cannot be improved with the increasing network scale.

6 Conclusions

In this paper, we formulated TATAS, the topology-aware task allocation and scheduling problem for energy efficient data fusion applications in wireless sensor networks. We also proposed a three-phase heuristic to solve the TATAS problem. Comparing with previous work, our algorithm can utilize the network topology information effectively. We implemented and simulated our proposed algorithms, and compared the results with existing approaches. Experimental results show that our approach can achieve significant energy efficiency and improve the system lifetime.

Acknowledgment

The work described in this paper is partially supported by the grants from the Research Grants Council of the Hong Kong Special Administrative Region, China (CERG 526007(PolyU B-Q06B), PolyU A-PA5X), the National Research Foundation for the Doctoral Program of Higher Education of China (No.20049998027), and the National Science Foundation of China (No.90604006).

References

[1] T. Agarwal, A. Sharma, A. Laxmikant, and L. Kale. Topology-aware task mapping for reducing communication contention on large parallel machines. In *Proc. IPDPS '06*.

[2] R. P. Dick, D. L. Rhodes, and W. Wolf. Tgff: task graphs for free. In *Proc. CODES/CASHE '98*.

[3] S. Gandham, Y. Zhang, and Q. Huang. Distributed minimal time convergecast scheduling in wireless sensor networks. In *Proc. ICDCS '06*.

[4] W.-Z. S. F. Y. R. LaHusen. Time-optimum packet scheduling for many-to-one routing in wireless sensor networks. In *Proc. MASS '06*.

[5] C.-H. Lee and K. G. Shin. Optimal task assignment in homogeneous networks. *IEEE Trans. Parallel Distrib. Syst.*, 1997.

[6] G. Malewicz, A. L. Rosenberg, and M. Yurkewych. On scheduling complex dags for internet-based computing. In *Proc. IPDPS '05*.

[7] H. Park and M. B. Srivastava. Energy-efficient task assignment framework for wireless sensor networks. Technical report, 2003.

[8] B. Z. M. W. Z. Shao. Topology-aware energy efficient task allocation and scheduling for in-network processing in distributed sensor systems. Technical report, 2008.

[9] Y. Tian. Cross-layer collaborative in-network processing in multihop wireless sensor networks. *IEEE Trans on Mobi Comp*, 2007.

[10] Y. Tian, J. Boangoat, E. Ekici, and F. Ozguner. Real-time task mapping and scheduling for collaborative in-network processing in dvs-enabled wireless sensor networks. In *Proc. IPDPS '06*.

[11] A. Wang and A. Chandrakasan. Energy-efficient dsps for wireless sensor networks. *IEEE Signal Processing Magazine*, 43(5):68–78, 2002.

[12] C.-C. Yeh. Power-aware allocation of chain-like real-time tasks on dvs processors. *IEICE - Trans. Inf. Syst.*, 2006.

[13] Y. Yu. *Information Processing and Routing in Wireless Sensor Networks*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2007.

[14] Y. Yu and V. K. Prasanna. Energy-balanced task allocation for collaborative processing in wireless sensor networks. *Mob. Netw. Appl.*, 10(1-2):115–131, 2005.

[15] F. Yuan Tian; Ekici, E.; Ozguner. Energy-constrained task mapping and scheduling in wireless sensor networks. In *Proc. MASS '05*.

[16] M. Yurkewych. Toward a theory for scheduling dags in internet-based computing. *IEEE Trans. Comput.*, 55(6):757–768, 2006.