2009 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications

A*-Based Task Assignment Algorithm for Context-Aware Mobile Patient Monitoring Systems

Hailiang Mei, Bert-Jan van Beijnum, Pravin Pawar, Ing Widya and Hermie Hermens
Telemedicine Group, Faculty of Electrical Engineering, Mathematics and Computer Science
University of Twente, Enschede, The Netherlands
Email: {meih, beijnum, p.pawar, widya, hermens}@ewi.utwente.nl

***Abstract*—Mobile Patient Monitoring System (MPMS) is positioned to provide high quality healthcare services in the near future. The gap between its application demands and resource supplies, however, still remains and may hinder this process. Dynamic context-aware adaptation mechanisms are required in order to meet the stringent requirements on such mission critical applications. The fundamental model underlying an MPMS includes a set of biosignal data processing tasks distributed across a set of networked devices. In our earlier work, we designed and validated a task distribution framework to support dynamic system reconfiguration of MPMS by means of task redistribution. This paper focuses on its decision-making component that can calculate the optimal task assignment by taking into account the reconfiguration costs. This paper has three major contributions. Firstly, we study a context-aware scenario and derive the design requirements for a task assignment algorithm in MPMS. Secondly, using a graph-based system model, we proposed an A*-based task assignment algorithm that minimizes the system end-to-end delay while guaranteeing required system battery lifetime and availability. We introduce a set of node expansion rules and a pre-processing procedure to calculate the heuristic function ($h(n)$). Thirdly, we evaluate the algorithm performance with experiments and compare this A*-based algorithm with other heuristic approaches, e.g. greedy and bounded A*.**

***Index Terms*—context-aware; task assignment algorithm; mobile patient monitoring system; battery lifetime; availability; end-to-end delay; A* algorithm; dynamic reconfiguration**

I. Introduction

Recently, Mobile Patient Monitoring System (MPMS) is receiving more attention due to its potential to tackle the resource challenges posed by the aging society, to improve the quality of diagnosis and treatment and to reduce the costs of healthcare service delivery ([1], [2]). An MPMS can capture biomedical and context information from a patient while he/she pursues normal daily life activities. The system processes the data and forwards the result to a decision point (e.g. a doctor in a healthcare center, supported by a clinical decision support system). Thus in case the system detects a medical emergency, the decision point can plan the appropriate response, e.g. sending an ambulance to the patient. We refer the underlying computation and communication resource of MPMS as an *m-health platform*. On top of this platform, various telemonitoring applications can operate continuously (24/7). Examples of telemonitoring applications include safety-critical applications such as trauma care, detection of life threatening ventricular arrhythmias, detection of foetal distress and premature labour ([1]) and detection of epileptic seizures ([3]). An epilepsy detection application and an example of m-health platform are depicted in Figure 1.

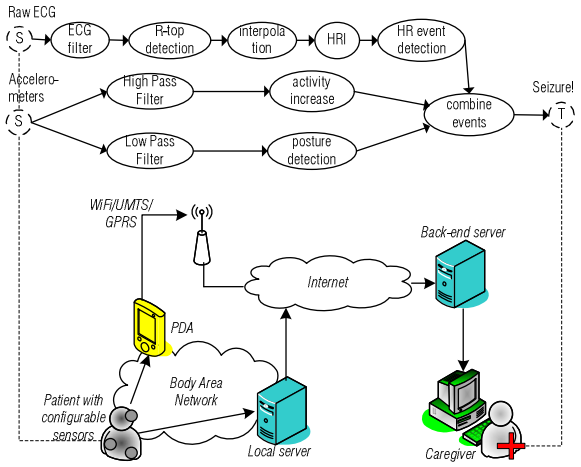

Fig. 1. Biosignal processing tasks in an epileptic seizure detection ([3]) running on top of an m-health platform. Due to the requirements of sensory data collection and final decision-making by a doctor, all the source tasks have to be associated with the sensor devices and the sink task has to be associated with the sink device (end-terminal in this case).

Similar to other applications operating in a mobile environment, an MPMS could be (deeply) affected by context changes and scarcity of m-health platform's resource, e.g. network bandwidth, battery power and computational power of handhelds. For example, a drop in network bandwidth due to patient's mobility can result in transmitted bio-signal loss or excessive delay. When this mismatch between application demand and resource supply exceeds a certain tolerated level, the entire MPMS may fail in responding accurately and timely to an emergency situation ([4]). Thus, the success of an MPMS relies heavily on whether the system can adapt itself and provide adequate and continuous bio-signal processing and transmission services despite context variations.

One possible adaptation approach is to exploit the distributed processing paradigm of MPMS and adjust the assignment of tasks across available devices at run-time. Our simulation results show that this dynamic approach can significantly improve system performance compared to the current static setting. For example the system battery lifetime can be potentially increased by more than 200% compared to

1533-2306/09 $25.00 © 2009 IEEE
DOI 10.1109/RTCSA.2009.34

a static setting ([5]). In our earlier work ([6]), we have proposed a task distribution framework to support dynamic reconfiguration of MPMS by means of task redistribution. This framework consists of a Coordinator and a set of Facilitators (Figure 2). A Facilitator reports the presence/absence of the device and receives control commands on task management from the Coordinator. The Coordinator runs a task assignment algorithm that can identify the best task assignment based on the required telemonitoring application and the current device network's context information, e.g. available devices and their connectivity, device's CPU load, device's remaining battery energy, etc. Once a significant change occurs in the required telemonitoring application or in the device network's context, the Coordinator is triggered to compute, under the new situation, the optimal task assignment together with a few near-optimal assignments as candidates. These candidate assignments are further ranked subject to both their performance enhancements and their reconfiguration cost. If the identified best assignment is different from the current one deployed in the system, the Coordinator constructs a reconfiguration plan and controls the Facilitators to deploy the new task assignment by means of task redistribution.

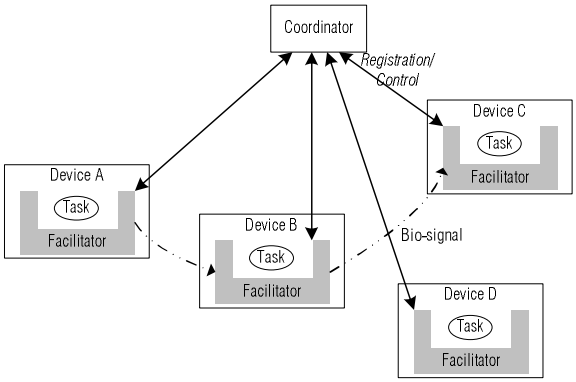

Fig. 2. Task distribution framework

Task assignment algorithms targeting at some special topologies of MPMS were studied earlier ([7], [8]). In this paper we focus on the more general form, task DAG (Directed Acyclic Graph) to resource DAG assignment problem. Normally, the numbers of tasks and devices in MPMS are less than ten. Thus, it is computationally feasible to find the optimal solution by exact algorithms. A* search approach is selected as the base of our algorithm, since it is admissible and generates a smaller search tree than any other admissible search algorithm with the same heuristic ([9]).

The rest of the paper is organized as follows. In Section II some related work is discussed. Section III studies the requirement of task assignment algorithm based on a dynamic task redistribution scenario in an MPMS. Section IV presents a graph-based system model and formulates the task assignment problem in MPMS. Section V explains the algorithm. Section VI examines the effectiveness and the efficiency of this algorithm in a set of experiments and Section VII concludes our paper.

II. Related Work

Task assignment (also referred as allocation, mapping or partitioning) is a well-known NP-hard problem in the general form. A good survey describing the basic concepts and models of task assignment problem can be found in ([10]). This problem has been studied intensively within various application areas. For example, it is an essential problem in the field of grid computing ([11]), System-on-Chip design ([12]), distributed databases ([13]) and wireless sensor network ([14], [15]). Except for some of our earlier work, we did not find other research work which addresses the task assignment problem in MPMS. Here, we compare our work with other research in somewhat related fields.

In case of smaller problem size, it is computationally feasible to find the optimal task assignment by exact algorithms. Shen and Tsai ([16]) are the first researchers to apply A* algorithm in the task assignment problem in distributed systems. In their paper, a task assignment is defined as a weak homomorphism between a task graph and process graph, i.e. two adjacent tasks are required to be assigned to either the same processor or two adjacent processors. The objective of their algorithm is to minimize the largest total computation and communication costs at a processor. In ([17]), authors proposed an ordering method to create a better task search order that can reduce the number of A* search tree nodes and thus increase the algorithm speed. In ([18]), authors proposed two techniques to further enhance the A* algorithm performance. The first technique is to generate a random task assignment and use the corresponding cost as a pruning criterion to reduce the size of search tree. The second technique is to divide the search tree and speed up the algorithm by parallel processing. While all these existing work on A* algorithm use an abstract communication and computation cost as the performance measure, we are interested in multiple performance measures that are more relevant to the user and network context in MPMS.

Furthermore, the heterogenous m-health platform in MPMS exhibits three special properties that require specific attentions. Firstly, it is modeled as an arbitrary network and has a random topology. Secondly, the network connections are asymmetric thus channel directions have to be modeled. Thirdly, data streams can be relayed by devices. These aspects have been addressed in the past individually, however not together.

- Task assignment problem in arbitrary processor networks was studied in ([15]). A heuristic algorithm was proposed that can assign tasks to the most suitable processor by taking into account the network topology. However, as our experiments show, this kind of greedy algorithms suffer from the risk of finding no suitable task assignment in MPMS.
- The channel directions in processor network were modeled explicitly in ([12], [19]). The objectives of their studied task assignment algorithms are to minimize total

system energy consumption. A Branch-and-Bound algorithm is proposed in ([12]) and a greedy algorithm is proposed in ([19]).
- In ([20], [21]), authors considered the possibility of relaying data streams by devices. However, the performance estimation of a task assignment does not include the extra resource consumption at a relaying device caused by relayed data stream.

Thus, another uniqueness of the task assignment problem in MPMS is that all these three special features are required to be taken into account. In this paper, we use a new graph model, propose specific assignment constraints and node expansion rules to tackle these challenges.

III. Scenario and Requirements

In this section we present a scenario of MPMS referred in the remainder of this paper: a context-aware dynamic reconfigurable epileptic seizure detection application. Based on this scenario, we derive a set of design requirements for the task assignment algorithm in MPMS.

A. Dynamic Reconfigurable Epilepsy Detection Scenario

John is an epileptic patient who had been seizure-free for several years. He wears an MPMS (Figure 1) which monitors his health state and can give him a few seconds' advance warning of an upcoming seizure by real-time processing of ECG and activity data from his body worn sensors. When John is at home, a broadband network is available to transfer his raw ECG and activity data to the remote monitoring center, eg. the back-end server. In this case, all tasks in the detection algorithm are deployed on the back-end server and the doctor can be warned if a seizure is likely to occur. John feels safe because he knows he can get immediate help if he has a seizure.

One afternoon, John is out jogging, following his usual route through the forest. Since there is no broadband network available in the forest, John's biosignals cannot be transmitted due to insufficient network bandwidth. In this case, the task assignment algorithm decides that some processing tasks should be activated at his PDA and his biosignals are processed locally. During his run, the signal processing algorithm detects a possible imminent epileptic seizure. John is immediately warned by his PDA, and stops running. At the same time, an alarm and John's GPS position are sent to the healthcare center via a narrow band connection, e.g. GPRS or GSM. Depending on the circumstances, a medical team or an informal caregiver can be dispatched to the exact location where John is to render emergency assistance.

B. Requirement Analysis

A mismatch between task demands and resource supplies is the initial cause for system adaptation ([22]). Thus the basic requirement of a task assignment algorithm is that resulted task assignments should be mismatch-free. Furthermore, an MPMS may incorporate additional performance requirements due to the mission-critical nature. Some of these are assurance requirements, e.g. "the end-to-end delay should be smaller than 2 seconds"; some are optimization requirements, e.g. "the system battery lifetime should be maximized"; some are the combinations of previous two, e.g. "the system battery lifetime should be maximized while its end-to-end delay should be smaller than 2 seconds". The task assignment algorithm under study should be able to take these user preferences into account and produce satisfactory assignments. In this paper, we consider the following three performance measures that are critical to the success of MPMS.

- "End-to-end delay": defined as the elapsed time between an MPMS receives a unit of patient's biosignal information and sends the processed result of this corresponding data unit to the decision point. This parameter indicates how quickly the biosignal and processed result can be delivered by the system.
- "System battery lifetime": defined as the minimum battery lifetime of all the battery powered devices in the system. An MPMS incorporates a number of battery powered devices, e.g. front sensors and patient's PDA. If the remaining battery energy of a device is lower than a certain level, it cannot support the assigned tasks any more. This parameter indicates the maximum operating time of an MPMS.
- "Availability level": considered here to be steady state availability as defined in ([23]), that is the application mean uptime divided by the sum of the mean uptime and mean downtime. Failures may potentially occur during either data processing or communication.

IV. Task Assignment Problem in MPMS

The aim of the task assignment algorithm under study is to identify several candidate task assignments that can support the required telemonitoring application and provide improved performance. Firstly, we present some graph-based definitions to model MPMS. Secondly, we propose a computational model to estimate the performance of an MPMS given a particular task assignment. Thirdly, we formulate the task assignment problem in MPMS.

A. Definitions

We define a telemonitoring application as a partial order of biosignal streaming tasks. In our model we distinguish two types of streaming tasks: stream processing tasks and stream transmission tasks. Processing tasks typically perform some operation on the biosignal stream such as filtering, transcoding, or higher level m-health application-specific data processing operations. Each processing task consumes one or more data streams and produces one or more data streams. Transmission tasks are the glue between processing tasks and have two functions: firstly they allow us to easily characterize properties of the data stream (for instance the data rate of the stream); and secondly, as we will see later, transmission tasks can be mapped onto a communication path that represents stream relaying by devices. Such a path may be a stream pipe within

a device, or it may be a networked path between different devices.

A telemonitoring application consisting of distributed tasks can be defined as a tuple of (P, T, A_t, L_P, L_T), where P is a set of stream processing tasks $\{p_1, p_2, ...\}$, T is a set of transmission tasks $\{t_1, t_2, ...\}$, A_t is a set of precedence relations between tasks, such that $A_t \subseteq P \times T \cup T \times P$, L_P is a set of labels over each processing task, L_T is a set of labels over each transmission task. L_P and L_T indicate the resource demand of processing tasks and transmission tasks respectively, a detailed overview is given in Table I. The structure $\{P, T, A_t\}$ is a DAG termed as *task DAG*. An example of a task DAG is presented in Figure 3.

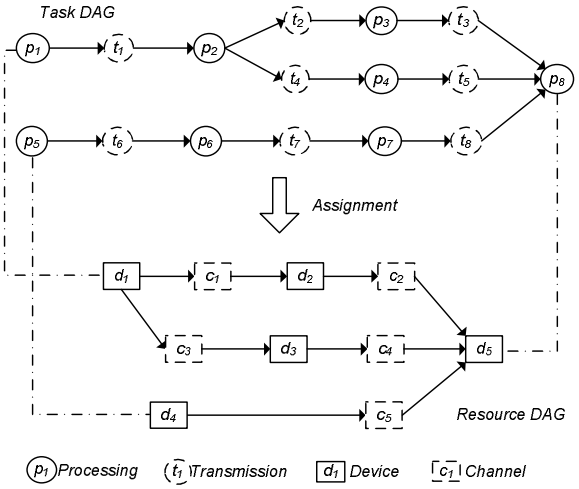

Fig. 3. Model of task assignment in an m-health system. Due to the fixed associations, "p_1" has to be assigned to "d_1", "p_5" has to be assigned to "d_5" and "p_8" has to be assigned to "d_3".

Similarly, an m-health platform is defined as a tuple of (D, C, A_r, L_D, L_C), where D is a set of device resources $\{d_1, d_2, ...\}$, C is a set of (communication) channel resources $\{c_1, c_2, ...\}$, A_r is a set of precedence relations between resources, such that $A_r \subseteq D \times C \cup C \times D$, L_D is a set of labels over each device resource, L_C is a set of labels over each channel resource. L_D and L_C model the resource supply of devices and channels respectively, further details are given in Table I. The structure $\{D, C, A_r\}$ is a DAG termed as *resource DAG*, an example of such a graph is shown in Figure 3.

We assume that device resources in an MPMS can relay biosignal data streams, therefore a transmission task may be assigned to a directed communication path. A communication path is a directed path in the resource DAG starting at d_i and ending at d_j. In general, there exist multiple paths connecting d_i to d_j. When $i = j$, it is a special communication path within device d_i and can be denoted the same as the device. We define CP as the set of all communication paths in a resource DAG.

Based on the two graph models, a task assignment (Φ) is a mapping function of tasks onto resources such that each processing task is mapped to one device and each transmission task is mapped to a communication path ($\Phi : P \bigcup T \rightarrow D \bigcup CP$).

Notation	Belongs to	Meaning
$n^{\mathrm{P}}_{p_i}$	L_P	the number of operations per time unit at processing task p_i
rr_{p_i}	L_P	the required resource of processing task p_i, e.g. minimum required CPU, minimum required memory, etc
$de^{\mathrm{P}}_{p_i,d_j}$	L_P	the processing delay of processing task p_i to process one data frame at device d_j
av_{p_i,d_j}	L_P	the availability of running processing task p_i at device d_j
$n^{\mathrm{T}}_{t_i}$	L_T	the number of transmitted data units per time unit of transmission task t_i
$de^{\mathrm{T}}_{t_i,d_j}$	L_T	the transmission delay of transmission task t_i to transfer one data frame at device d_j
$de^{\mathrm{T}}_{t_i,c_j}$	L_T	the transmission delay of transmission task t_i to transfer one data frame at channel c_j
av_{t_i,c_j}	L_T	the availability of performing transmission task t_i at channel c_j
av_{t_i,d_j}	L_T	the availability of performing transmission task t_i at device d_j
$e^{\mathrm{TO}}_{d_i}$	L_D	the total available battery energy at device d_i[1]
$e^{\mathrm{HK}}_{d_i}$	L_D	the energy consumption rate of the "housekeeping" activities at device d_i, e.g. CPU, display, powering network interface cards, etc
$e^{\mathrm{op}}_{d_i}$	L_D	the energy consumption of one operation at device d_i
rs_{d_i}	L_D	the available resource supply at device d_i , e.g. CPU type, available memory, etc
bw_{c_i}	L_C	the available bandwidth at channel c_i
lo_{c_i}	L_C	the current load information (influenced by other users in the same channel) at channel c_i
$e^{\mathrm{S}}_{c_i}$	L_C	the energy consumption of sending one data unit through channel c_i
$e^{\mathrm{R}}_{c_i}$	L_C	the energy consumption of receiving one data unit through channel c_i

TABLE I
NOTATIONS FOR LABELING

B. Performance Estimation for A Given Task Assignment

In this section, we present the computational model to estimate system performance, i.e. end-to-end delay, system battery lifetime and availability, given a particular assignment.

1) End-to-End Delay: In a task DAG, we define task path as a directed path connecting two tasks. The entire set of task paths is denoted as TP. Every task path, tp_i, is an ordered sequence of processing tasks and transmission tasks, and the m^{th} task can be denoted as $tp_i(m)$. Upon different task assignment Φ, tp_i exhibits a different end-to-end delay, i.e. the summation of processing delay and transmission delay along the path:

$$\Omega^{\mathrm{de}}(tp_i) = \sum_{tp_i(m) \in P} de^{\mathrm{P}}_{tp_i(m),\Phi(tp_i(m))} + \sum_{tp_i(m) \in T} de^{\mathrm{T}}_{tp_i(m),\Phi(tp_i(m))} \quad (1)$$

where $de^{\mathrm{P}}_{p_i,d_j}$ is defined as the processing delay for processing task p_i to process one frame of biosignal data at device

Notation	Meaning
P	a set of processing tasks
T	a set of transmission tasks
D	a set of device resources
C	a set of (communication) channel resources
TP	a set of task paths $\{tp_i\}$ in the task DAG
CP	a set of communication paths $\{cp_i\}$ in the resource DAG
A_t	a set of precedence relations between tasks
A_r	a set of precedence relations between resources
Ω^{de}	the measure of end-to-end delay
Ω^{av}	the measure of availability
Ω^{li}	the measure of battery lifetime
$min\Omega^{\mathrm{av}}$	the minimal required system availability
$min\Omega^{\mathrm{li}}$	the minimal required system battery lifetime
nrc	the number of required candidate task assignments
$de^{\mathrm{T}}_{t_i,cp_j}$	the transmission delay of transmission task t_i to transfer one data frame through communication path cp_j
$e^{\mathrm{P}}_{d_i}$	the energy consumption rate of data processing at device d_i
$e^{\mathrm{S}}_{d_i}$	the energy consumption rate of sending data stream at device d_i
$e^{\mathrm{R}}_{d_i}$	the energy consumption rate of receiving data stream at device d_i

TABLE II
LIST OF SYMBOLS (EXCLUDE LABELING SYMBOLS)

d_j, $de^{\mathrm{T}}_{t_i,cp_j}$ is defined as the transmission delay of transmission task t_i to transfer one frame of biosignal data over a communication path cp_j. $de^{\mathrm{T}}_{t_i,cp_j}$ can be computed based on the transmission delay occurred at device d_j, $de^{\mathrm{T}}_{t_i,d_j}$, and at channel c_j as $de^{\mathrm{T}}_{t_i,c_j}$. $de^{\mathrm{P}}_{p_i,d_j}$, $de^{\mathrm{T}}_{t_i,d_j}$ and $de^{\mathrm{T}}_{t_i,c_j}$ can be estimated from the profiling information, e.g. $n^{\mathrm{P}}_{p_i}$, $n^{\mathrm{T}}_{t_i}$, bw_{c_i} and lo_{c_i} based on the knowledge from real measurements, e.g. in [24].

The end-to-end delay of a given assignment Φ, $\Omega^{\mathrm{de}}(\Phi)$, is then defined as the maximum of all task paths' end-to-end delays:

$$\Omega^{\mathrm{de}}(\Phi) = \max(\{\Omega^{\mathrm{de}}(tp_i)|tp_i \in TP\}) \tag{2}$$

2) Battery Lifetime: Based on the power consumption model of a mobile device [25], we estimate the battery life time for a specific device d_i, $\Omega^{\mathrm{li}}(d_i)$, as:

$$\Omega^{\mathrm{li}}(d_i) = \frac{e^{\mathrm{TO}}_{d_i}}{e^{\mathrm{HK}}_{d_i} + e^{\mathrm{P}}_{d_i} + e^{\mathrm{R}}_{d_i} + e^{\mathrm{S}}_{d_i}} \tag{3}$$

Where $e^{\mathrm{TO}}_{d_i}$ is the total available battery energy at device d_i, $e^{\mathrm{HK}}_{d_i}$ is energy consumption rate of device's "housekeeping" activities, e.g. powering CPU, display and network interface cards; $e^{\mathrm{P}}_{d_i}$ is the energy consumption rate by local data processing; $e^{\mathrm{R}}_{d_i}$ is the energy consumption rate for receiving data stream; $e^{\mathrm{S}}_{d_i}$ is the energy consumption rate for sending data stream. Given a particular task assignment Φ, the latter three parameters can be calculated as:

$$e^{\mathrm{P}}_{d_i} = e^{\mathrm{op}}_{d_i} \sum_{\{p_j|\Phi(p_j)=d_i\}} n^{\mathrm{P}}_{p_j} \tag{4}$$

$$e^{\mathrm{S}}_{d_i} = \sum_{(d_i,c_j)\in A_r} (e^{\mathrm{S}}_{c_j} \sum_{c_j\in\Phi(t_k)} n^{\mathrm{T}}_{t_k}) \tag{5}$$

$$e^{\mathrm{R}}_{d_i} = \sum_{(c_j,d_i)\in A_r} (e^{\mathrm{R}}_{c_j} \sum_{c_j\in\Phi(t_k)} n^{\mathrm{T}}_{t_k}) \tag{6}$$

Once the battery lifetime of all devices are estimated, the minimum of all device's battery lifetime determines the overall system lifetime for a given assignment:

$$\Omega^{\mathrm{li}}(\Phi) = \min(\{\Omega^{\mathrm{li}}(d_i)|d_i \in D\}) \tag{7}$$

3) Availability: Since we assume only AND semantics in task DAG, the availability of the application depends on all the included tasks: The application performs successfully only when all tasks perform successfully. Therefore, the application availability level for a given assignment Φ, $\Omega^{\mathrm{av}}(\Phi)$, can be computed as:

$$\Omega^{\mathrm{av}}(\Phi) = \prod_{\mathrm{p_i}\in\mathrm{P}} \mathrm{av}_{\mathrm{p_i},\Phi(\mathrm{p_i})} \prod_{\mathrm{t_i}\in\mathrm{T},\alpha\in\Phi(\mathrm{t_i})} \mathrm{av}_{\mathrm{t_i},\alpha} \tag{8}$$

C. Problem Formulation

In many MPMS, it is always desirable to minimize the system end-to-end delay while guarantees a certain system battery lifetime and system availability. Thus, we treat the end-to-end delay as the objective function for our task assignment algorithm and the other two measures as assignment constraints. We formulate the task assignment problem in MPMS as follows:

- **Given**: (1) a telemonitoring application (P, T, A_t, L_P, L_T); (2) an m-health platform (D, C, A_r, L_D, L_C); (3) Three performance evaluation function Ω^{de}, Ω^{li} and Ω^{av}.
- **Goal**: To find a number of candidate task assignments $\{\Phi^{\mathrm{can}}_m|m = 1, 2, nrc\}$ among all possible task assignments such that the value of $\Omega^{\mathrm{de}}(\Phi^{\mathrm{can}}_m)$ are minimized.
- **Subject to**: four assignment constraints namely *type constraint*, *local constraint*, *assurance constraint* and *reachability constraint*.

The *type constraint* specifies that each processing task must be mapped to one and only one device resource and each transmission task must be mapped to one and only one communication path, hence:

$$\forall p_i \in P : \Phi^{\mathrm{can}}_m(p_i) \in D, \forall t_i \in T : \Phi^{\mathrm{can}}_m(t_i) \in CP \tag{9}$$

The *local constraint* indicates that the resulted assignments should be mismatch-free as motivated earlier. It comprises two parts:

- For every processing task p_i in P, its host device must be able to provide the task's required resources. That is, assuming we have a Boolean function $satisfy(rr, rs)$ to evaluate whether a required resource, rr, can be satisfied by a resource supply, rs, we must guarantee:

$$\forall p_i \in P : satisfy(rr_{p_i}, rs_{\Phi^{\mathrm{can}}_m(p_i)}) = true \tag{10}$$

- For each channel resource, the total assigned transmission tasks must not exceed its offered bandwidth:

$$\forall c_i \in C : \sum_{c_i\in\Phi^{\mathrm{can}}_m(t_i)} n^{\mathrm{T}}_{t_i} < bw_{c_i} \tag{11}$$

For an MPMS, we denote the minimum required system lifetime as $min\Omega^{\text{li}}$ and the minimum required system availability as $min\Omega^{\text{av}}$. The *assurance constraint* for selecting candidate task assignments is defined as:

$$\Omega^{\text{av}}(\Phi_m^{\text{can}}) > min\Omega^{\text{av}}, \Omega^{\text{li}}(\Phi_m^{\text{can}}) > min\Omega^{\text{li}} \tag{12}$$

The *reachability constraint* specifies that for each processing task p_i assigned to a device resource $\Phi_m^{\text{can}}(p_i)$, its predecessor transmission task must be assigned to a communication path ending at $\Phi_m^{\text{can}}(p_i)$, and its successor transmission task must be assigned to a communication path starting at $\Phi_m^{\text{can}}(p_i)$.

V. Task Assignment Algorithm

A. Algorithm Overview

The A* algorithm is a best-first search algorithm that finds the optimal solution with the lowest cost by constructing and traversing a search tree. In this search tree, the root node represents a null solution, an intermediate node represents a partial solution and a leaf (goal) node represents a complete solution to the problem. Each node, n, has an associated cost function $f(n)$ which is a sum of two functions $g(n)$ and $h(n)$: $g(n)$ is the actual cost of the partial solution represented by n and $h(n)$ is a lower-bound estimation of the additional cost from n to the leaf node. The A* algorithm expands a node by generating all of its child nodes and calculating the value of f for each of them. All newly generated child nodes are sorted by the value of f and stored in a sorted list $OPEN$. The A* algorithm thus finds the optimal solution by always removing the node in the $OPEN$ list with a minimal $f(n)$ and expanding the search tree from this node.

In the search tree of our proposed A*-based algorithm, every intermediate node represents a partial-assignment and every leaf node represents a full-assignment. When an intermediate node (or root node) is expanded to its child nodes, a new task is selected according to a pre-defined task order (c.f. Section V-B). Then based on a set of expansion rules (c.f. Section V-C), this task is assigned to a set of proper resources, e.g. devices or communication paths, to generate new partial-assignments. Based on the proposed computational models, $g(n)$ and $h(n)$ can be calculated for a particular partial-assignment n (c.f. Section V-D). The pseudo code of this algorithm is illustrated in Algorithm 1.

Algorithm 1 Pseudo code of A*-based task assignment algorithm

```
input the number of required candidate assignments as nrc
input the minimal required availability min Ω^av and the minimal required system battery lifetime min Ω^li
determine a task search order: order
initialize a sorted list OPEN to contain the visited partial-assignments
initialize a sorted list candidates to contain the found candidate full-assignments
create a partial-assignment root, f(root) ⇐ 0, root.toBeAssignedTask ⇐ order.first
add root into OPEN
while true do
  if OPEN is empty then
    return candidates
  end if
  remove a partial-assignment n with lowest f(n) from OPEN
  if n is a full-assignment then
    add n into candidates
    if candidates.size() = nrc then
      return candidates
    end if
  end if
  index ⇐ order.getIndex(n.toBeAssignedTask)
  expand from n to a set of child partial-assignments {cn_i}
  for all partial-assignments in {cn_i} do
    if Ω^li(cn_i)≥min Ω^li ∧ Ω^av(cn_i)≥min Ω^av then
      f(cn_i) ⇐ g(cn_i) + h(cn_i)
      cn_i.toBeAssignedTask ⇐ order.get(index+1)
      add cn_i into OPEN
    end if
  end for
end while
```

B. Task Search Order

The efficiency of A*-based algorithm closely relates to the size of generated search tree. If we can order tasks such that, at shallow tree levels, less number of nodes are expanded and the cost difference between the expanded nodes are larger, then a smaller search tree could be generated by the A* algorithm. Earlier work ([17], [18]) has shown that assigning a task with a larger bearing on $f(n)$ first can result a smaller search tree. In our problem model, it is the reachability constraint that plays a more dominant role. Thus, we propose to order the tasks following the reachability of task DAG. Our ordering strategy is very similar to a topological sort of DAG. The only difference is that if a transmission task (α) is visited, then its direct successor (processing) task (β) should be visited next immediately. This is even the case when not all direct predecessor (transmission) tasks of β are visited. For example, a task order generated following this "semi-topological" approach is illustrated in Figure 4.

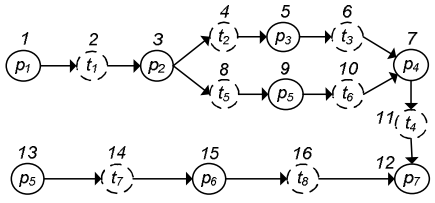

Fig. 4. A "semi-topological" task order for an example task DAG. Different from a topological sort, p_4 comes before t_6 and p_7 comes before t_8

C. Expansion Rules

Due to the reachability constraint, a number of nodes in the search tree will represent invalid assignments. In order

to perform an effective and yet efficient search for optimal assignment, it is ideal to only generate the branches that lead to at least one valid full assignment. Thus, we define the following rules for the algorithm to assign the to-be-assigned task i to avoid as many un-necessary branches as possible. To better illustrate these rules, a search tree (partial view) for the problem setting in Figure 3 is presented in Figure 5.

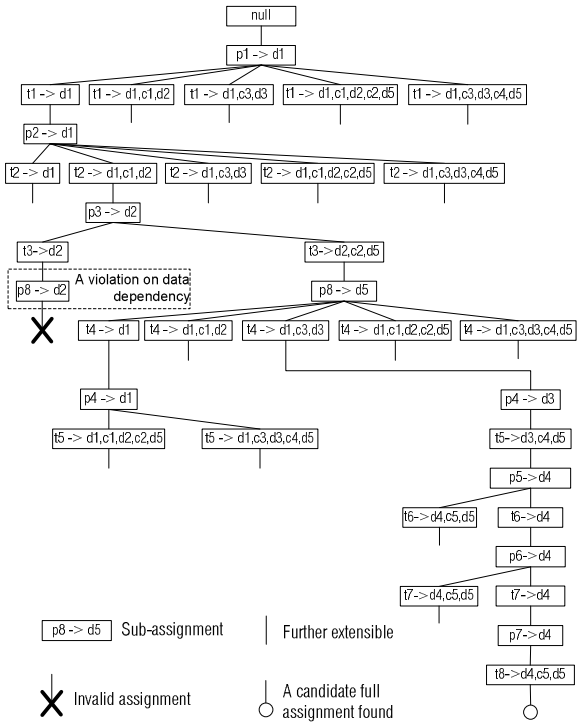

Fig. 5. Search tree for the problem setting shown in Figure 3 with a depth-first task visit order. For clarity, each node is only labeled with the assignment of latest visited task.

(1) When i is a processing task, there are two possibilities:

- If i is a source processing task, then it is assigned to all possible devices subject to its local constraint, e.g. "$p_1 \rightarrow d_1$".
- If i is NOT a source processing task, then one of the direct predecessor (transmission) tasks of i must be assigned already to a communication path, denoted as cp_x. This is because of the "semi-topological" task ordering approach. In this case, the search tree should be expanded by assigning i to the ending device of cp_x, e.g. "$p_3 \rightarrow d_2$".

(2) When i is a transmission task, we also distinguish between two cases. The direct predecessor (processing) task of i must have been assigned to a device already due to the "semi-topological" task search order and we term this device as d_α.

- When i's direct successor (processing) task is not assigned, there will be some freedom to assign i: we need to find all directed communication paths starting with d_α and assign i to each of these paths, e.g. "$t_3 \rightarrow d_2$" and "$t_3 \rightarrow d_2, c_2, d_5$".
- If i's direct successor (processing) task is already assigned to a device vertex (d_β), i has to be assigned to a directed communication path connecting device d_α with device d_β, e.g. "$t_8 \rightarrow d_4, c_5, d_5$". If it is not possible to find such a path, then the current sub-assignment is not valid and this branch of the search tree has to be pruned.

D. $g(n)$ and $h(n)$

In our A* based algorithm, $g(n)$ gives the exact end-to-end delay for a partial-assignment. Thus, we have $g(n) = \Omega^{\mathrm{de}}(n)$. $h(n)$ should indicate the lower-bound estimation of the additional delay caused by all un-assigned tasks. We assume for a partial-assignment n, $g(n)$ is determined by a task path α, i.e. $\Omega^{\mathrm{de}}(n) = \Omega^{\mathrm{de}}(\alpha)$, and the last task in α as y. Now, we can define $h(n)$ as the minimal aggregated delay contributed by y's successor tasks. This minimal aggregated delay is denoted as $minde_y^{\mathrm{task}}$ and calculated as:

$$minde_y^{\mathrm{task}} = \max\{\sum_{m=1}^{|tp_j^y|}(\min_{tp_j^y(m)\in P}(de^{\mathrm{P}}_{tp_j^y(m),*}) + \min_{tp_j^y(m)\in T}(de^{\mathrm{T}}_{tp_j^y(m),*})) \mid tp_j^y \in TP^y, j = 1, 2, ...|TP^y|\} \quad (13)$$

where TP^y is the set of task paths $\{tp_j^y\}$ in the task DAG which connect a direct successor task of y with a sink (processing) task, "*" denotes any device or channel resource.

We propose a labeling procedure to calculate $minde_y^{\mathrm{task}}$ for each task y in a task DAG. This is a pre-processing procedure that is executed before the task assignment algorithm. The labeling starts from sink tasks and propagates reversely along the orientation in the task DAG. All the sink tasks are labeled with "0".

- For a processing task y, we calculate its $minde_y^{\mathrm{task}}$ as follows:

$$minde_y^{\mathrm{task}} = \max_{(y,x)\in A_t}\{minde_x^{\mathrm{task}} + \min\{de^{\mathrm{T}}_{x,*}\}\} \quad (14)$$

  where x denotes any direct successor transmission task of y, "*" denotes any device or channel resource.
- For a transmission task y, we calculate its $minde_y^{\mathrm{task}}$ as follows:

$$minde_y^{\mathrm{task}} = minde_x^{\mathrm{task}} + \min\{de^{\mathrm{P}}_{x,*}\} \quad (15)$$

  where x denotes the direct successor processing task of y, "*" denotes any device resource.

This labeling procedure visits every task exactly once. For each task, we examine its processing or transmission delay at most once per device or channel resource. Thus, the time complexity of this labeling procedure is $(|P|+|T|)(|D|+|C|)$.

VI. Experiments

To evaluate the effectiveness and efficiency of our proposed task assignment algorithm, we implemented a test environment in Java. It contains two modules. The first module is a task/resource DAGs generator. It reads in the sizes of

task/resource DAGs and the range of profiling information and generates a random problem setting, i.e. the topologies and labeling parameters of task/resource DAGs. We set a fixed association (capturing the local constraints) for every source (sink) processing task to a randomly selected source (sink) device. At this moment, the number of source vertexes in the DAGs can be adjusted and the number of sink vertexes is a constant one. Thus, each problem setting can be represented as $(|P|, |T|, |D|, |C|, |SP|, |SD|)$, where six parameters indicate the number of processing tasks, transmission tasks, devices, channels, source processing tasks and source devices respectively. The second module implements the proposed A*-based algorithm and reports a number of candidate task assignments with their estimated performance, i.e. end-to-end delay, battery lifetime and availability.

In this section, we reported a set of simulation results from this test environment. All tests are performed on a PC running Windows XP with Intel Core(TM)2 CPU 2.4GHz and 2G RAM. All simulation results presented here are averaged over 50 runs with a user-defined problem setting ($(|P|, |T|, |D|, |C|, |SP|, |SD|)$) and randomly generated topology and profiling information. All the measured algorithm completion time are in the time unit of milliseconds.

A. Effectiveness on Minimizing End-to-End Delay

We first conducted an effectiveness test to understand better what is the added-value of enforcing an optimal task assignment in MPMS. Six different problem settings are included and the simulation results are presented in Table III. In each setting's test, we ran the A*-based algorithm to compute the optimal task assignment with a minimal delay ($\Omega^{\mathrm{de}}_{\mathrm{opt}}$). As a comparison, we also computed the average end-to-end delay ($\Omega^{\mathrm{de}}_{\mathrm{ave}}$) of all possible task assignments. The improvement on end-to-end delay by enforcing an optimal task assignment is defined as $\frac{\Omega^{\mathrm{de}}_{\mathrm{ave}} - \Omega^{\mathrm{de}}_{\mathrm{opt}}}{\Omega^{\mathrm{de}}_{\mathrm{ave}}}$. The results show that the average improvement we can expect for an optimal task assignment is between 25% and 30%.

Setting	Delay (optimal)	std (optimal)	delay (average)	std (average)	Improvement
(7,9,6,8,3,2)	336.86	66.45	484.32	51.98	30%
(7,9,6,8,4,3)	302.88	49.08	379.99	47.14	20%
(8,10,6,8,3,2)	368.74	53.48	503.44	57.55	26%
(8,10,6,8,4,3)	313.02	46.03	399.05	51.57	21%
(9,11,6,8,3,2)	398.52	60.45	531.16	62.40	25%
(9,11,6,8,4,3)	367.12	58.37	449.56	51.55	18%

TABLE III
THE DIFFERENCE ON END-TO-END DELAY BETWEEN OPTIMAL ASSIGNMENT AND RANDOM ASSIGNMENTS

B. Impact of Number of Candidate Assignments (nrc)

We tested the algorithm performance on computing candidate task assignments ($\{\Phi^{\mathrm{can}}_m | m = 1, 2, nrc\}$) in different required number (nrc). The testing value of nrc ranges from 1 to 5. In each nrc test, we measured the number of assignment nodes visited by the A*-based algorithm and the algorithm completion time. The simulation results for two problem settings are reported in Table IV and Table V. We made the following two observations:

- The number of visited assignment nodes (search tree size) and the completion time are spread over a large range. The reason is that some generated task and resource DAGs only have a limited number of possible assignments while some of them have many more possibilities. Besides the number of vertexes and edges, the topology of those DAGs also has a large impact on the number of possible task assignments (which is closely related to the search tree size).
- Between searching for one best assignment (the optimal one) and searching for five best assignments, the difference regarding tree size and completion time is not so significant. Thus, this is a very promising algorithm for computing a set of candidate task assignments, as was our intention in the design of this algorithm.

nrc	♯Nodes (mean)	♯Nodes (std)	Time (mean)	Time (std)
1	3756.95	1607.35	129.65	73.23
2	3779.60	1579.38	120.40	55.56
3	3966.85	1515.96	126.45	53.06
4	3968.65	1515.90	120.25	46.53
5	4074.20	1571.89	120.60	52.70

TABLE IV
EXPERIMENTS ON DIFFERENT nrc ON SETTING OF (9,11,6,8,3,2)

nrc	♯Nodes (mean)	♯Nodes (std)	Time (mean)	Time (std)
1	8793.80	5736.82	421.05	474.66
2	8975.05	5812.00	400.80	457.99
3	8994.80	5815.38	404.65	451.59
4	9052.60	5833.04	403.25	440.15
5	9334.35	6026.83	431.90	451.05

TABLE V
EXPERIMENTS ON DIFFERENT nrc ON SETTING OF (10,12,6,8,3,2)

C. Bounded A*

Although A* based algorithm is very effective on finding the optimal task assignment, it still suffers the worst-case exponential complexity. In order to provide a reasonable solution in a bounded running time, it has been suggested to limit the size of the OPEN list [26]: when the number of expanded nodes exceeds this allowed maximum number, some still expandable nodes with larger costs will be forgotten by the algorithm, i.e. removed from the OPEN list. This bounded A* algorithm is more "greedy" because it only traverses into a number of branches with lower costs. Four different size limits for OPEN list, i.e. $|P| * |D|$, $0.8 * |P| * |D|$, $0.5 * |P| * |D|$ and $0.2 * |P| * |D|$, are tested against running an original A*-based algorithm (unbounded). For each different size limit, we examined so-called quality of assignment (QoA) that is

defined as:

$$QoA = \frac{mean\{\Omega^{\text{de}}(\Phi_m^{\text{can}} \textit{ found by bounded A*})\}}{mean\{\Omega^{\text{de}}(\Phi_m^{\text{can}} \textit{ found by original A*})\}} \quad (16)$$

Thus, a value of QoA closer to "1" indicates that the candidate task assignment's end-to-end delay is closer to the optimal one. We experimented this bounded A* algorithm in three problem settings. The results averaged over 50 runs are reported in Table VI, Table VII and Table VIII. It is observed that with a reasonable tight bound, e.g. $0.8*|P|*|D|$, the candidate assignments found by bounded A* are still satisfactory since they are on average worse than the optimal assignment only within a 5% range. However, one additional drawback of the bounded A* algorithm is that it does not always produce a sufficient number of candidate assignments. The reason lies in its "greedy" nature, since some branches with valid full assignments are removed from the (bounded) OPEN list.

Max size of $OPEN$	**♯Nodes (mean)**	**♯Nodes (std)**	**Time (mean)**	**Time (std)**	**♯Found**	**QoA**
unbounded	2440.85	1196.85	77.55	44.20	5.00	1
$\|P\|*\|D\|$	1102.70	309.08	24.15	17.98	5.00	1.04
$0.8 * \|P\|*\|D\|$	966.20	279.18	21.05	7.50	5.00	1.04
$0.5 * \|P\|*\|D\|$	564.90	175.61	14.10	4.84	5.00	1.07
$0.2 * \|P\|*\|D\|$	270.45	52.13	3.15	6.47	4.95	1.17

TABLE VI
EXPERIMENTS ON BOUNDED A* ON SETTING OF (8,10,6,8,3,2)

Max size of $OPEN$	**♯Nodes (mean)**	**♯Nodes (std)**	**Time (mean)**	**Time (std)**	**♯Found**	**QoA**
unbounded	2296.55	1671.26	88.30	77.62	5.00	1.00
$\|P\|*\|D\|$	1076.70	570.93	25.85	16.25	5.00	1.05
$0.8 * \|P\|*\|D\|$	934.70	470.40	19.30	8.57	5.00	1.05
$0.5 * \|P\|*\|D\|$	611.80	242.55	15.00	10.74	5.00	1.05
$0.2 * \|P\|*\|D\|$	284.20	111.48	7.75	7.96	4.95	1.09

TABLE VII
EXPERIMENTS ON BOUNDED A* ON SETTING OF (8,10,6,8,4,3)

D. Advantage over Greedy Algorithms

Since the task assignment problem in the general form is a NP-hard problem, different heuristic algorithms have been proposed ([15], [19]). These algorithms compute one sub-optimal task assignment by greedily assigning a task to a device with a minimal assignment cost. We implemented an algorithm called "Pure Greedy" based on this heuristic approach and tested its performance against our A*-based algorithm. We conducted the experiments of finding one (sub-)optimal task assignment in two problem settings as shown in Table IX and Table X. We examined two measures. The first is "success ratio" that is the possibility of finding one (sub-)optimal assignment successfully. The second is "quality of the best assignment" (QoB) that is defined as:

Max size of $OPEN$	**♯Nodes (mean)**	**♯Nodes (std)**	**Time (mean)**	**Time (std)**	**♯Found**	**QoA**
unbounded	3577.80	3709.58	157.80	236.93	5.00	1.00
$\|P\|*\|D\|$	1675.15	906.77	31.95	19.24	5.00	1.02
$0.8 * \|P\|*\|D\|$	1527.60	802.96	28.25	10.90	5.00	1.03
$0.5 * \|P\|*\|D\|$	975.05	519.21	18.65	11.96	5.00	1.04
$0.2 * \|P\|*\|D\|$	395.85	162.44	9.45	7.92	5.00	1.10

TABLE VIII
EXPERIMENTS ON BOUNDED A* ON SETTING OF (9,11,6,8,4,3)

$$QoB = \frac{\Omega^{\text{de}}(\textit{(sub-)optimal assignment found})}{\Omega^{\text{de}}(\textit{optimal assignment})} \quad (17)$$

From the experiment results, we see that the "Pure Greedy" algorithm performs worse in both settings. In about 40 out of 50 runs, it can not find a possible task assignment. The reason is that this heuristic algorithm is very shortsighted and does not work well in our formulated task assignment problem in MPMS. It often runs into an invalid assignment that either violates a local constraint, e.g. exceeds the available bandwidth on a particular channel, or violates a reachability constraint, e.g. two connected tasks are not assigned to two connected resources. This experiment result indicates that in order to find a set of candidate task assignments for MPMS, a search tree has to be designed carefully with special attention on task search order and node expansion rules.

Max size of $OPEN$	**♯Nodes (mean)**	**♯Nodes (std)**	**Time (mean)**	**Time (std)**	**Success ratio**	**QoB**
unbounded	7192.15	13816.39	559.45	1614.08	100%	1
$\|P\|*\|D\|$	1122.25	837.13	23.35	23.60	100%	1.02
$0.8 * \|P\|*\|D\|$	966.65	652.91	20.25	11.39	100%	1.02
$0.5 * \|P\|*\|D\|$	712.45	433.29	14.75	9.48	100%	1.02
$0.2 * \|P\|*\|D\|$	381.65	192.07	4.70	7.37	100%	1.05
Pure Greedy	9.60	4.81	3.95	7.02	16%	1.14

TABLE IX
COMPARISON THE BOUNDED A* WITH PURE GREEDY ALGORITHM ON SETTING OF (10,12,6,8,4,3)

VII. CONCLUSION

Similar to other applications operating in a mobile environment, an MPMS could be (deeply) affected by context changes and scarcity of m-health platform's resource, e.g. network bandwidth, battery power and computational power of handhelds. Dynamic context-aware adaptation mechanisms are required in order to meet the stringent requirements on

Max size of $OPEN$	♯Nodes (mean)	♯Nodes (std)	Time (mean)	Time (std)	Success ratio	QoB
unbounded	3151.60	3347.59	120.30	168.25	100%	1
$\|P\|*\|D\|$	1127.90	801.76	25.05	17.90	100%	1.02
$0.8 * \|P\|*\|D\|$	912.05	540.47	17.85	11.55	100%	1.02
$0.5 * \|P\|*\|D\|$	602.10	370.99	12.60	9.62	100%	1.03
$0.2 * \|P\|*\|D\|$	324.60	196.67	8.65	8.03	100%	1.07
Pure Greedy	10.65	4.03	3.80	8.42	22%	1.17

TABLE X
COMPARISON THE BOUNDED A* WITH PURE GREEDY ALGORITHM ON SETTING OF (9,11,6,8,4,3)

such mission critical applications. The core of the adaptation mechanism is a decision-making component that can calculate/select the optimal task assignment to be enforced by taking into account the reconfiguration costs. This paper first studied the performance requirements of MPMS using the example of a novel epilepsy detection scenario. We identified three key performance measures that are critical to the success of system, i.e. end-to-end delay, system battery lifetime and availability level. Secondly, using a graph-based system model, we proposed an A*-based task assignment algorithm that minimizes the system end-to-end delay while guaranteeing required performance on the other two aspects. In particular, we propose a set of node expansion rules and a pre-processing procedure to calculate the heuristic function ($h(n)$). Thirdly, we evaluate the algorithm performance with experiments and provide recommendations for further improvements, e.g. the use of bounded A* algorithm.

Our future work consists of the following two aspects: (1) to incorporate reconfiguration cost in the search of optimal task assignment; (2) to integrate and test the proposed task algorithm in real-life applications in MPMS.

ACKNOWLEDGMENT

This work is part of the Freeband AWARENESS Project. Freeband is sponsored by the Dutch government under contract BSIK 03025. (http://awareness.freeband.nl).

REFERENCES

[1] V. M. Jones, A. V. Halteren, I. Widya, N. Dokovsky, G. Koprinkov, R. Bults, D. Konstantas, and R. Herzog. Mobihealth: Mobile health services based on body area networks. In Robert S. H. Istepanian, S. Laxminarayan, and C.S. Pattichis, editors, *M-health Emerging Mobile Health Systems*, pages 219–236. Springer, 2006.

[2] A. D. Jurik and A. C. Weaver. Remote medical monitoring. *IEEE Computer*, 41(4):96–99, 2008. 0018-9162.

[3] T. Tonis, H. J. Hermens, and M. Vollenbroek-Hutten. Context aware algorithm for discriminating stress and physical activity versus epilepsy. Technical report, AWARENESS deliverables (D4.18), 2006.

[4] V. M. Jones, F. Incardona, C. Tristram, S. Virtuoso, and A. Lymberis. Future challenges and recommendations. In Robert S. H. Istepanian, S. Laxminarayan, and C.S. Pattichis, editors, *M-health Emerging Mobile Health Systems*, pages 267–270. Springer, 2006.

[5] H. Mei, B. J. van Beijnum, I. Widya, V. Jones, and H. Hermens. Performance optimization in mobile healthcare systems - a task assignment approach. *submitted to IEEE Journal on Selected Areas in Communications*, 2009.

[6] H. Mei, B. J. van Beijnum, P. Pawar, I. Widya, and H. Hermens. Context-aware dynamic reconfiguration of mobile patient monitoring systems. In *4th International Symposium on Wireless Pervasive Computing (ISWPC 2009)*, 2009.

[7] P. Pawar, H. Mei, I. Widya, B. J. van Beijnum, and A. van Halteren. Context-aware task assignment in ubiquitous computing environment - a genetic algorithm based approach. In *IEEE Congress on Evolutionary Computation (CEC 2007)*, pages 2695–2702, Sept. 2007.

[8] H. Mei, P. Pawar, and I. Widya. Optimal assignment of a tree-structured context reasoning procedure onto a host-satellites system. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS 2007)*, pages 1–9, March 2007.

[9] R. Dechter and J. Pearl. Generalized best-first search strategies and the optimality of A*. *Journal of the ACM (JACM)*, 32(3):505–536, 1985.

[10] M. G. Norman and P. Thanisch. Models of machines and computation for mapping in multicomputers. *ACM Computing Surveys*, 25(3):263–302, 1993. 56 citations.

[11] K. Cooper, A. Dasgupta, K. Kennedy, C. Koelbel, A. Mandal, G. Marin, M. Mazina, J. Mellor-Crummey, F. Berman, H. Casanova, A. Chien, H. Dail, X. Liu, A. Olugbile, O. Sievert, H. Xia, L. Johnsson, B. Liu, M. Patel, D. Reed, W. Deng, C. Mendes, Z. Shi, A. YarKhan, and J. Dongarra. New grid scheduling and rescheduling methods in the GrADS project. In *18th International Parallel and Distributed Processing Symposium (IPDPS'04)*, page 199, 2004.

[12] J. Hu and R. Marculescu. Energy- and performance-aware mapping for regular NoC architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(4):551–562, 2005. 0278-0070.

[13] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer. Network-aware operator placement for stream-processing systems. In *22nd International Conference on Data Engineering (ICDE'06)*, 2006.

[14] Y. Gu, Y. Tian, and E. Ekici. Real-time multimedia processing in video sensor networks. *Signal Processing: Image Communication*, 22(3):237–251, 2007.

[15] B. Zhao, M. Wang, Z. Shao, J. Cao, K. Chan, and J. Su. Topology aware task allocation and scheduling for real-time data fusion applications in networked embedded sensor systems. In *14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA '08)*, pages 293–302, Aug. 2008.

[16] C. C. Shen and W. H. Tsai. A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion. *IEEE Transactions on Computers*, C-34(3):197–203, 1985.

[17] S. Ramakrishnan, I. H. Cho, and L. A. Dunning. A close look at task assignment in distributed systems. In *Tenth Annual Joint Conference of the IEEE Computer and Communications Societies. (INFOCOM '91)*, pages 806–812 vol.2, 1991.

[18] M. Kafil and I. Ahmad. Optimal task assignment in heterogeneous distributed computing systems. *IEEE Concurrency [see also IEEE Transactions on Parallel and Distributed Technology]*, 6(3):42, 1998. 1092-3063.

[19] W. Alsalih, S. Akl, and H. Hassanein. Energy-aware task scheduling: Towards enabling mobile computing over MANETs. *19th International Parallel and Distributed Processing Symposium (IPDPS'05)*, 2005.

[20] L. Franken. *Quality of Service Management: a Model-Based Approach*. PhD thesis, University of Twente, The Netherlands, 1996.

[21] C. H. Lee and K. G. Shin. Optimal task assignment in homogeneous networks. *IEEE Transactions on Parallel and Distributed Systems*, 8(2):119–129, 1997. 1045-9219.

[22] M. Satyanarayanan. The many faces of adaptation. *IEEE Pervasive Computing*, 2004.

[23] J. K. Muppala, R. M. Fricks, and K. S. Trivedi. Techniques for dependability evaluation. In W. Grasman, editor, *Computational Probability*, pages 445–480. Kluwer Academic Publishers, 2000.

[24] Y. Xiao, J. Rosdahl, S. L. D. Center, M. Linear, and U. T. Draper. Throughput and delay limits of ieee 802.11. *IEEE Communications Letters*, 6(8):355–357, 2002.

[25] A. Rahmati and L. Zhong. Context-for-wireless: Context-sensitive energy-efficient wireless data transfer. In *5th international conference on Mobile systems, applications and services (Mobisys'07)*, pages 165–178, 2007.

[26] S. Russell. Efficient memory-bounded search methods. In *10th European conference on Artificial intelligence (ECAI '92)*, pages 1–5, New York, NY, USA, 1992. John Wiley & Sons, Inc.