Resource-Aware Scheduling of Distributed Ontological Reasoning Tasks in Wireless Sensor Networks

Tim De Pauw*†, Stijn Verstichel*, Bruno Volckaert*, Filip De Turck* and Veerle Ongenae†
*Department of Information Technology (INTEC)
Ghent University – IBBT, Gaston Crommenlaan 8 bus 201, 9050 Ghent, Belgium
†Faculty of Applied Engineering Sciences (INWE)
Ghent University College, Schoonmeersstraat 52, 9000 Ghent, Belgium
Email: tim.depauw@intec.ugent.be

***Abstract*—As the number of wireless sensor network applications continues to grow, the need for specialized task scheduling mechanisms, aware of the sensor devices' capabilities and real-time resource availability, is becoming more and more apparent. In this paper, we therefore propose a generic model for task scheduling in heterogeneous networks, which we subsequently use to schedule distributed reasoning tasks, originating from a real-world WSN monitoring and management application. By means of simulation, we evaluate several developed scheduling heuristics and compare the results to an optimal solution of the same WSN task scheduling problem, obtained using ILP. Experiments show that our heuristics produce acceptable task schedules while maintaining a low resource footprint.**

I. Introduction

Wireless sensor networks or *WSNs*—i.e., networks containing wireless nodes instrumented with sensor equipment—have steadily become a premier research topic. By aggregating data from these sensors, one may provide applications with augmented context information, such as time-stamped temperature readings, user locations, etc. Typical scenarios for WSN deployment include home automation, medical monitoring, and battlefield surveillance. [1], [2] The IBCN research group's testbed, for example, is comprised of 400 wireless mesh network nodes and 200 sensor nodes, measuring temperature, light and humidity—ideal for researching every aspect of WSNs, from hardware to application. [3]

The machines in a WSN can range from heavily resource-constrained sensor nodes to powerful back-end processing machines. Due to this heterogeneity, planning the execution of a distributed application in a WSN often turns out to be a far more facetious problem than in a more traditional network environment: sensor nodes may run different operating systems than traditional machines, offer fewer computing resources, be equipped with a limited supply of energy, etc. [4]
The allocation of resources to tasks (i.e., *matching*) and the ordering of these tasks (i.e., *scheduling*) is a problem generally known to be $\mathcal{NP}$-complete. [5] This is no different in a sensor network, introducing a clear need for intelligent near-optimal matching and scheduling techniques.

An important aspect of WSNs is power management. Sensor nodes are generally battery-operated, so it is imperative that their autonomy be maximized. As wireless communication is a relatively costly operation, it can often be beneficial to carry out as much processing as possible on the node itself, however limited its resources may be. In doing so, one can avoid the expensive radio transmissions required for the delegation of tasks to other nodes in the network. [6], [7]
This paradigm has been applied to a monitoring and management application for wireless sensor networks, based on *distributed reasoning* technology [8]. Reasoning decouples business logic from the application, allowing for a formal specification independent of program code. Through the use of *ontologies* [9] in this process, real-world concepts can be modeled in an expressive fashion.
By subsequently distributing ontological reasoning mechanisms across a set of interconnected machines, the computations as well as the information behind them can be managed in a decentralized manner. In the case of the WSN monitoring and management application, this allows for highly flexible sensor data processing and fault detection.

Of course, the monitoring and management application is just one example of a distributed WSN application. In this paper, we therefore propose a generic model for the off-line scheduling of tasks; while the model was designed with WSNs in mind, it can also be applied to other heterogeneous network environments. The characteristics of the monitoring and management application as well as the network are subsequently expressed in terms of the newly introduced model. In addition, we apply a set of solvers to the problem model. Firstly, an *integer linear program (ILP)* [10] is used to obtain optimal schedules. Secondly, a set of *heuristics* is introduced, in order to produce suboptimal schedules within reasonable time. We attempt to satisfy two distinct objectives. The first is to minimize the execution time of the full workload; the second is to use a minimal number of processing nodes to carry out the workload. For the purpose of validating the heuristic solvers, simulations were carried out, based on real-world scenarios; the results of these simulations are also discussed.

The remainder of this paper is structured as follows. In Sect. II, we provide an overview of related work. Sect. III focuses on distributed reasoning, elaborating on its components and their interaction. Next, in Sect. IV, we outline our assumptions, which lead to the formal problem specification we present in Sect. V. Sect. VI then describes the solution techniques we used to tackle the scheduling problem—an ILP formulation and a set of heuristics. In Sect. VII, we outline our approach toward simulating distributed reasoning scenarios from the monitoring and management application. The results of these simulations are then discussed in Sect. VIII. We end this paper with our conclusions and ideas for future work.

II. Related Work

An extensive amount of research has already been conducted in the field of task scheduling for distributed systems. Given the $\mathcal{NP}$-complete nature of the matching and scheduling problem, one generally strives to obtain a near-optimal solution. Dhodhi et al. cite examples of "optimal selection theory-based approaches, graph-based approaches, genetic algorithm-based techniques and other heuristics." [11]

Specifically for heterogeneous systems, Braun et al. compared eleven heuristics and found that genetic algorithms consistently produced the best results for the scenarios considered. [12] The possibility of multitasking is however not considered and, most importantly, requirements are modeled by stating whether or not a task can be executed on a given node, whereas our approach is not limited to a binary option.

Regarding the generation of network topologies, popular tools such as *BRITE* [13] were not designed with sensor networks in mind. *GenSeN* [14] and *topo_gen* [15], on the other hand, target this type of configuration specifically. However, as these tools focus on characteristics not part of our model, we opted for the implementation of our own topology generator.

The integration of semantic technologies with sensor networks is an active area of research. The Semantic Sensor Web, for instance, enhances sensor networks with spatial, temporal and thematic semantic metadata, resulting in an ontological model unburdened by interoperability issues. [16] Hu et al. have used ontologies to enable adaptive sensor information provisioning. [17] WSN middlewares enabling semantic formalisms have also emerged. The platform introduced by da Rocha et al., for instance, applies rule-based reasoning and fuzzy logic to ontologies. [18]

III. Distributed Reasoning

The distributed application whose tasks we will be scheduling allows for monitoring and management of a wireless sensor network: by collecting and analyzing data from the sensors, the application detects various issues in the network. An important aspect of the application is that there is no need to reengineer source code upon alterations to the network, as it is based on *ontological reasoning* in a distributed manner. In this section, we provide a brief introduction to the concept.

Recent years have seen an increase in the research on intelligent services. Adding intelligence to services creates the added value that the functionality and business logic exposed to a client is adapted and enhanced according to the context and environment in which client and service collaborate. Using ontologies as a modeling language—more specifically, the *Web Ontology Language* [9]—creates the possibility to formally reason over model and data. This aspect is supported by the foundation of ontologies in first-order description logic. However, as this process can become rather resource-intensive in large data-oriented systems, there is a clear need to study these mechanisms in a distributed environment, allowing for the development of an enabling service platform for distributed ontology-based reasoning.

Fig. 1 gives an overview of the workflow with its different phases and components in the platform.
First of all, the submitted query is analyzed and partitioned according to the meta-information available. [19] The reasoning tasks are specified through *SPARQL* [20] queries. Logically defined concepts in these queries trigger the reasoning process on the individual nodes.
Once the partitioning is completed, the subqueries are scheduled on the nodes in the network. These nodes contain both the model and the data, so that, given a specific query, the correct reasoning tasks are initiated. Different technologies are used to implement such nodes. These can either be natively ontology-based, making use of existing ontology-based libraries such as *Jena* [21], or they can also be based on conversion libraries like *D2R* [22] or *RDF123* [23]. These are combined with description logic reasoners, such as *Pellet* [24] or *Racer* [25]. After the completion of the reasoning tasks, the nodes return their individual results to the back-end platform. Finally, the back-end platform merges the information received and provides the eventual result to the originally requesting client.

For a more elaborate overview of the workings of this distributed reasoning platform, we refer to [8].

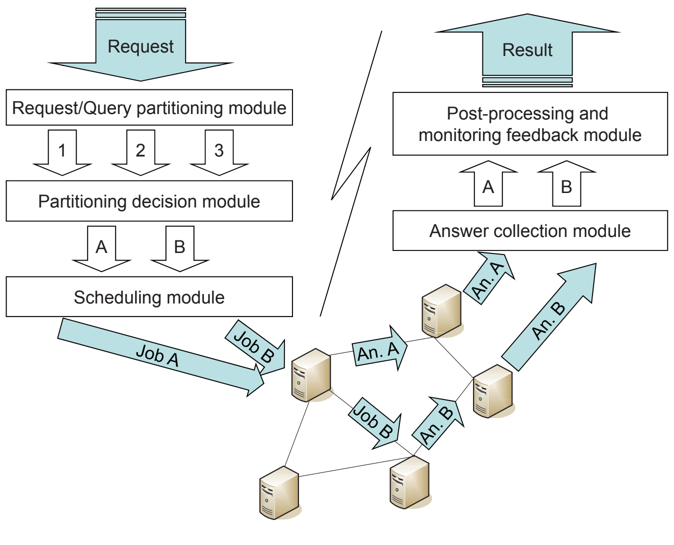

Request
Request/Query partitioning module
1
2
3
Partitioning decision module
A
B
Scheduling module
Job A
Job B
Job B
An. A
An. B
An. A
An. B
Answer collection module
A
B
Post-processing and monitoring feedback module
Result

Fig. 1: Distributed Reasoning Workflow

IV. Assumptions

A. Topology Model

A *topology* consists of *nodes*—i.e., machines capable of executing tasks in parallel. To model the nodes' capabilities, each of them receives a set of *properties*. Rather than using predefined properties, we require that they conform to either one of these *property types*:

- A *binary property* is a property which can be either true or false. An example would be a property indicating the presence of a temperature sensor.
- A *discrete property* is a property describing the total supply of a depleting resource, expressed as a positive integer. Such a property could, for instance, represent the amount of available memory.

B. Task Set Model

The *tasks* which are to be carried out all have a *duration*, expressed as an integer amount of time slots. Similarly, each task specifies a *deadline*; this is the index of the time slot before which the task must be completed.

Each task imposes *requirements* on the executing node, which calls for corresponding *requirement types*:

- A *binary requirement* determines whether or not a certain binary property must be true on the executing node. Tasks requiring the same binary property may be simultaneously active on the same node.
- A *discrete requirement* states how much of the supply of a certain discrete property the task is expected to use during its execution.

Each task must await the completion of all its *predecessors*. Tasks may be represented as the vertices of an unweighted *directed acyclic graph* or *DAG* [26], in which each arc represents a dependency between predecessor and task.

V. Problem Formulation

To reiterate, the problem tackled here is that of scheduling a set of interdependent tasks T on a network topology N, respecting time constraints as well as resource requirements. As mentioned, times are expressed in terms of discrete slots.

We envisage two possible *objectives* for our scheduler. The first is to minimize the *workload execution time c*. The second objective is to minimize the number of nodes used by the task set. We expect there to be a trade-off between these two metrics: the more nodes are used, the more concurrency, reducing the workload execution time.

For each task $t \in \mathrm{T}$, its *duration d_t* is the number of time slots required to complete the task. Thus, if a task t's *start time* is given by s_t, the workload execution time c is readily obtained using the expression

$$c = \max_{t \in \mathrm{T}} (s_t + d_t) \tag{1}$$

It is required that each task t be completely carried out before time slot e_t, defined as its *deadline*:

$$s_t + d_t \leq e_t \quad \forall\, t \in \mathrm{T} \tag{2}$$

However, task t cannot be initiated until all of the tasks $p \in \mathrm{P}_t$, its *predecessor set*, have been carried out:

$$s_p + d_p \leq s_t \quad \forall\, t \in \mathrm{T}, p \in \mathrm{P}_t \tag{3}$$

Whether or not a task can be matched to a node and scheduled to be executed on it is determined by comparing t's requirements to n's properties.
For each of a task t's *binary requirements q_t*, the corresponding *binary property q_n* must also be present on the executing node. In other words, if we express true and false values by one and zero, respectively, the following is a prerequisite for the matching of t and n:

$$q_t \leq q_n \quad \forall\, t \in \mathrm{T}, n \in \mathrm{N} \tag{4}$$

The value of each *discrete requirement r_t* is an integer amount. The corresponding *discrete property r_n*'s value must be at least that of r_t. If we define $r_{n,\upsilon}$ to represent the remaining supply of r_n at slot υ, t can only be matched to n and scheduled to start at slot τ if

$$r_t \leq r_{n,\upsilon} \quad \forall\, t \in \mathrm{T}, t \in \mathrm{N}, \upsilon = \tau, \ldots, \tau + d_t - 1 \tag{5}$$

If task t is indeed scheduled to start at slot τ on node n, its start time s_t is assigned the value τ and the associated supplies $r_{n,\upsilon}$ are decremented by r_t.

If all of the tasks could be scheduled without violating property constraints and/or deadlines, the schedule is said to be *complete*; otherwise, it is *incomplete*.

VI. Solution Techniques

A. Integer Linear Program

The first of our solvers is based on integer linear programming. It uses the *simplex* and *branch and bound* algorithms [10] to produce optimal schedules. These can then be used as reference solutions, to assess our heuristics. The ILP solver was implemented in Java 6, using *ILOG CPLEX 11* [27]. In this section, the underlying ILP formulation is provided.

1) Parameters: In Table I, the ILP formulation's parameters are shown. These are in fact identical to the ones introduced in the problem formulation from Sect. V.

2) Variables: Discrete and binary requirements are to be expressed as ILP constraints. In order to do so, the decision variables of the ILP need to take binary values. As the task start times s_t are not binary, we define decision variables $x_{t,n,\tau} \in \{0,1\}$ to express whether task t is matched to node n and scheduled to start at time slot τ. For convenience, we also define variables $m_{t,n} \in \{0,1\}$, each of which indicates whether task t is matched to node n.

TABLE I: ILP Parameters

d_t =	the duration of task t
e_t =	the deadline of task t
P_t =	the predecessor set of task t
r_t =	the amount of discrete property r that is required for task t
r_n =	the amount of discrete property r that is available at node n
q_t = 1	if the binary property q is required for task t
= 0	otherwise
q_n = 1	if the binary property q is available at node n
= 0	otherwise

3) Objective Function: As discussed in Sect. V, our goal is to minimize either the workload execution time or the number of nodes used by the schedule. The latter is expressed through an additional set of variables $u_n \in \{0,1\}$, each indicating whether node n is matched to any of the tasks. In order to allow for a trade-off between both objectives, we add weight factors α and β, resulting in the objective function

$$\min\ \alpha c + \beta \sum_{n \in \mathrm{T}} u_n \tag{6}$$

4) Constraints: The program's first set of constraints defines the value of the auxiliary variables $m_{t,n}$, which take the value 1 if task t is matched to node n. They are derived from the decision variables $x_{t,n,\tau}$:

$$m_{t,n} = \sum_{\tau \in \mathbb{N}} x_{t,n,\tau} \quad \forall\ t \in \mathrm{T}, n \in \mathrm{N}$$

Next, they are used to obtain the values of the variables u_n. As max is not a linear function, the expression is converted to a set of inequalities, yielding

$$\begin{aligned} u_n &= \max_{t \in \mathrm{T}}(m_{t,n}) \quad \forall\ n \in \mathrm{N} \\ \Rightarrow \quad u_n &\geq m_{t,n} \quad \forall\ n \in \mathrm{N}, t \in \mathrm{T} \end{aligned}$$

Again using the variables $m_{t,n}$, we ensure that each of the tasks be matched to *exactly* one node:

$$\sum_{n \in \mathrm{N}} m_{t,n} = 1 \quad \forall\ t \in \mathrm{T}$$

As task *start times* are not used as decision variables in this formulation, their values are derived as:

$$s_t = \sum_{n \in \mathrm{N}} \sum_{\tau \in \mathbb{N}} \tau x_{t,n,\tau} \quad \forall\ t \in \mathrm{T}$$

The other constraints of the ILP were introduced in Sect. V. The first three are given by (1), (2) and (3). As we again encounter the nonlinear max function in (1), this expression requires reformulation as well:

$$c = \max_{t \in \mathrm{T}}(s_t + d_t) \quad \Rightarrow \quad c \geq s_t + d_t \quad \forall\ t \in \mathrm{T}$$

For *binary* properties, we defined (4). To extrapolate this to the ILP formulation, we again use $m_{t,n}$:

$$m_{t,n} q_t \leq q_n \quad \forall\ t \in \mathrm{T}, n \in \mathrm{N}$$

Discrete properties are slightly more complex. In order to formulate (5) as a set of ILP constraints, we now see to it that the sum of each resource consumed at a given node during a given time slot, does not exceed the supply of that resource. Which and how many resources are consumed during each time slot depends on the tasks which are scheduled to be active during that slot. Hence, we need to take into account each task t that started up to $d_t - 1$ slots ago; we again use the variable υ to iterate over these consecutive slots.

$$\sum_{t \in \mathrm{T}} r_t \sum_{\upsilon = \max(0, \tau - d_t + 1)}^{\tau} x_{t,n,\upsilon} \leq r_n \quad \forall\ n \in \mathrm{N}, \tau \in \mathbb{N}$$

B. Heuristics

A set of heuristics was also developed to tackle the same problem: scheduling a set of tasks on a topology, respecting deadlines and requirements. Like the ILP solver, the heuristic solvers were implemented in Java 6.

The heuristics described here are all based on *bin packing* problems [28]. While four separate heuristics are introduced, they all share a common workflow; a pseudocode representation of this workflow is given in Fig. 2.

First, *topological sorting* [26] is applied to the DAG representing the task set. This process results in a list of tasks in which each task's predecessors come before the task itself.
For each task, bounds for the start time are determined. The lower bound is deduced from the end times of the task's predecessors. Due to the topological order, these have all been scheduled at this point; their end times are therefore known. The upper bound, on the other hand, is easily deduced from the task's deadline and duration.
Then, an attempt is made to schedule the task at each subsequent time slot. For every slot, the task is matched to every node in the topology, until a combination is encountered which does not violate any requirements. The order in which nodes are selected depends on the heuristic used; this is expressed through the abstract function *next_node*(). As soon as a suitable slot and node are encountered, the task is scheduled and the algorithm proceeds to the next task.

The described version of the workflow focuses on minimizing the workload execution time. Therefore, tasks will always be scheduled as early as possible. However, by swapping the *for* and *while* loops in Fig. 2, priority is shifted to a minimization of the number of nodes used.

Now, let us consider the heuristics individually.

1) First Fit: In the *First Fit* heuristic, the nodes are selected in the order in which they are defined in the topology. For each task and for each time slot, the topology is examined node by node, until one is encountered which can execute the task. Thus, this heuristic imposes more load on the first nodes.

2) Next Fit: The *Next Fit* heuristic remembers the node where a task was last scheduled successfully. Thus, upon each attempt to schedule the next task, the last node that was used is first considered before examining the rest of the topology. Next Fit thereby aims to reuse nodes as much as possible.

3) Best Fit: The *Best Fit* heuristic keeps track of how much load is imposed on each node and uses this information to order the nodes. Nodes whose resources have already been partly consumed are the first candidate locations for the current task. Hence, the heuristic orders nodes by their accumulated load, allowing it to always reuse the most active nodes.

4) Fair Fit: The *Fair Fit* heuristic uniformly selects the next candidate node. It bears a resemblance to *First Fit*, with the exception that the list of nodes is shuffled each time the algorithm proceeds to the next task to schedule. In doing so, it is expected that each node be treated equally, hence distributing loads evenly.

```
$task\_list \leftarrow topological\_sort(task\_set)$
**for all** $task \in task\_list$ **do**
    $min\_start \leftarrow 0$
    **for all** $pred \in task.predecessors$ **do**
        $pred\_end \leftarrow get\_start(pred) + pred.duration$
        **if** $pred\_end > min\_start$ **then**
            $min\_start \leftarrow pred\_end$
    $max\_start \leftarrow task.deadline - task.duration$
    **for** $time \leftarrow min\_start$ **to** $max\_start$ **do**
        **while** $node \leftarrow next\_node()$ **do**
            **if** $feasible(task, node, time)$ **then**
                $allocate(task, node, time)$
                **next** $task$
```

Fig. 2: Heuristic Scheduler Pseudocode

VII. Problem Generator

In order to apply the distributed reasoning workflow to the generic problem formulation, the associated software components were benchmarked in a test setup. In this setup, five ALIX-based nodes [29] were tasked with reasoning on temperature, light and humidity readings, for the purpose of detecting faulty data originating from the sensors.

Using the resulting benchmarks, a problem generator was developed, which simulates more elaborate versions of the scenario just outlined. The generated problems all take place in a fictitious office environment of six by four rooms, arranged in a grid structure; an example is shown in Fig. 3. Each of these rooms is equipped with an ALIX node, which collects readings for the room and is able to reason on them. Each node is installed in a randomly selected corner, placing it close enough to any adjacent rooms to also obtain the readings from those. Most readings are therefore present at multiple nodes. A binary property is used to model this.

As in any distributed reasoning scenario, the goal is to transmit the results of the individual reasoning processes from the nodes to a back-end server. We assume the presence of a single back-end server, distinguished from the ALIX nodes by an additional binary property.
Contrary to the test setup, the ALIX nodes do not transmit their results directly to the back-end server. Instead, clusters of two by two nodes are created, as displayed in alternating colors in Fig. 3. Within each cluster, reasoning results are provided to one of the nodes, which then forwards the aggregated data to the back-end server. Each aggregation task requires a binary property, which is shared by the clustered tasks. The rationale behind the aggregation is that, because a transmission to the back-end server can consume quite a bit of power, it is better to sacrifice only one in four nodes' resources. Evidently, this concept can be extended in various ways.

To summarize, there are four types of tasks. The back-end *preprocessor* analyzes and splits the incoming query. For each set of sensor readings associated with a room, a subquery is created. Each of these is processed by a *reasoning* task on one of the ALIX nodes aware of the appropriate readings. Next,

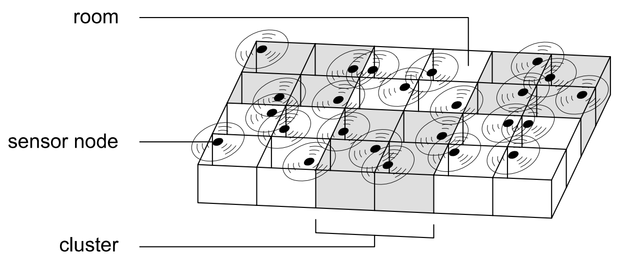


Fig. 3: Office Environment with Sensors

in each ALIX cluster, one node *aggregates* the results of the cluster, and finally returns them to the back-end *postprocessor*.

In each generated problem, three such cycles are to be scheduled for simultaneous execution. However, the distributed reasoning tasks are not the only ones to be scheduled. As the IBCN testbed, for one, is shared between researchers, multitasking is commonplace. Thus, in addition to the distributed reasoning workflow, it is assumed that several *miscellaneous* tasks are also competing for resources. Half of these receive an additional binary requirement, expressing that they depend on sensor readings from a single, uniformly selected room; the other half may be scheduled on any node.

Each of the tasks requires a certain amount of memory, which we model using a discrete property. The parameter values used are listed in Table II. When a range of (discrete) values is listed, a value is in fact uniformly selected from that range. Durations are expressed in units of five seconds. The corresponding topology parameters are shown in Table III.

Regarding deadlines, by default, it is assumed that none of the tasks have to wait until adequate resources become available. This implies that a task can be initiated as soon as all of its predecessors have been processed. Of course, this is not very realistic, as tasks will often compete for resources. Therefore, for each task type, we introduce a *grace period* g_t: the number of time slots by which the task's completion may be delayed. Unused grace periods propagate to tasks' successors. Thus, if t_2 follows t_1, the deadline of t_2 becomes $e_{t_2} = e_{t_1} + d_{t_2} + g_{t_2}$, where $e_{t_1} = d_{t_1} + g_{t_1}$.
The ALIX reasoning tasks all receive such a grace period of 60 slots. In the event that resources become scarce, they may therefore wait for (at least) two of their siblings to complete on the same ALIX node, without violating any deadlines.

TABLE II: Simulated Task Types

Task Type	Node	Count	Duration	Memory
Preprocessing	Server	3×1	1	100–200 MB
Reasoning	ALIX	3×24	1–30	30–90 MB
Aggregation	ALIX	3×1	1	100–200 MB
Postprocessing	Server	3×1	1	100–200 MB
Miscellaneous	ALIX	20	1–20	0–100 MB

TABLE III: Simulated Node Types

Node Type	Real-life Equivalent	Memory
Server	Generic back-end machine	1,000 MB
ALIX	ALIX-based processing node	100 *or* 200 MB

VIII. Evaluation

The solution techniques introduced in Sect. VI were assessed by applying them to a set of 100 problems created by the problem generator from Sect. VII. All simulations were carried out on a dual quad-core Intel Xeon L5420 machine with 16 GB of RAM, running Scientific Linux.

A phenomenon common to ILP solvers is that their execution times are difficult to predict. Therefore, in order to avoid spending too many computational resources on a single simulation, problems which could not be solved within four hours were abandoned.

A. Minimal Workload Execution Time

The ILP objective function (6) was first tailored toward a minimization of workload execution times: α was set to 1 and β to 0. In doing so, the *ILP solver* yielded optimal schedules for 37 problems, while 40 others could not be solved within four hours. Interestingly, applying the *heuristics* to these 37 problems, every schedule obtained was also complete.
Applied to all 100 problems, the heuristics still produced largely complete schedules. On average, just 1.10% of the tasks could not be scheduled; the worst-case loss was 5.17%.

Of course, our main intent was to compare workload execution times. Again for the 37 completed problems, the ILP solver required an average of 42.9 slots, while the heuristics introduced a delay of about 36%. Exact results are shown in Fig. 4a. Also displayed in this chart is the number of nodes used, which was consistently close to the maximum of 25.
Best Fit produced the lowest workload execution times, with First Fit performing only slightly worse. Interestingly, Fair Fit was a close third, implying that uniform selection of nodes yields competitive results. Next Fit's attempts to reuse nodes were most likely thwarted by the numerous constraints.

It is important to note that the delay introduced by the heuristics is mitigated by the time required to produce schedules. The heuristic solvers' execution times averaged at 291 milliseconds, compared to 79 minutes for the ILP solver. Note that this discrepancy would be even more apparent if we were to lift the four-hour time limit for the latter.
Memory usage revealed similar patterns. The heuristic solvers barely used 50 MB, while the ILP solver averaged at 1.95 GB and peaked at 9.31 GB. However, the latter is indispensable in obtaining optimal schedules.

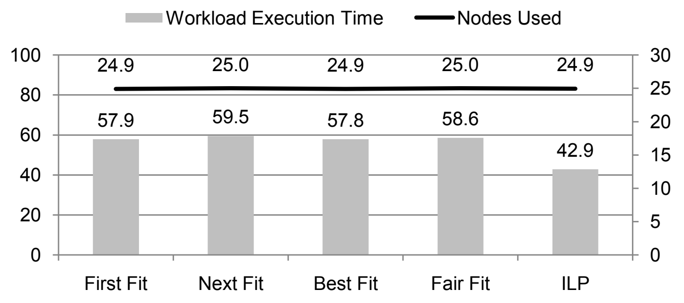
B. Minimal Number of Nodes Used

Here, the parameters of the ILP objective function were chosen to minimize the number of nodes used: α now becomes 0 and β becomes 1. In this case, the *ILP solver* was less affected by the four-hour time limit: 47 problems resulted in optimal schedules. The *heuristics*, however, suffered: complete schedules were returned by all four in a mere 20 cases.
Consequently, looking at all 100 problems, unscheduled task rates were slightly elevated, with averages between 1.34% (Fair Fit) and 1.65% (Best Fit). In the worst case, First Fit reached losses of up to 8.62%.

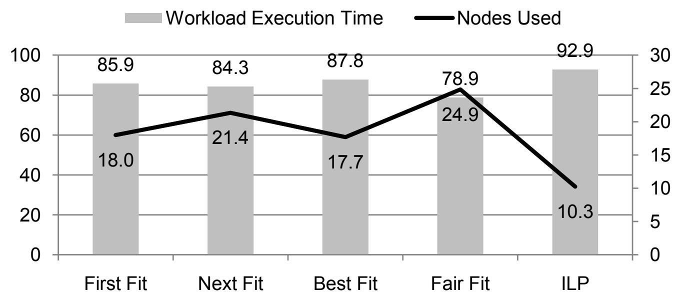
Fair Fit's apparent advantage is easily explained: as nodes are selected on a random basis, the heuristic is actually the worst alternative in terms of minimizing the number of nodes used. Considering the 20 sets of complete schedules, Fair Fit required an average of no less than 24.9 out of 25 nodes, compared to just 17.7 for Best Fit and even 10.3 in the optimal case, using the ILP solver. These results are shown in Fig. 4b. Consequently, on average, even our best heuristic used 73% more nodes than optimally required—a significant rise compared to the first case.

Also displayed in Fig. 4b are the average workload execution times. Clearly, there is a trade-off between our two objectives, time and resources. While the ILP solver uses far fewer nodes, its workload execution times are now the highest. This is not surprising: as α is 0, c is not minimized at all. Additional tuning of both parameters would improve the results. However, preliminary experiments suggested that this would increase most solution times well beyond four hours.
Taking both objectives into account, Best Fit and First Fit are close contenders; depending on one's priorities, both may be beneficial. Like Fair Fit, Next Fit may produce low execution times, but its load distribution is unsatisfactory.

As in the previous case, the heuristic solvers completed virtually instantly; their execution times averaged at 449 milliseconds. Similarly, their memory usages were nearly identical to those of their counterparts minimizing workload execution times. The ILP solver, however, was slightly less greedy, taking up an average of 1.74 GB of memory and peaking at 5.40 GB. Its execution took 47 minutes, on average; again, we must remark that this value is affected by the fact that a maximum solution time of four hours was enforced.



(a) Minimal Workload Execution Time



(b) Minimal Number of Nodes Used

Fig. 4: Solver Metrics (averages)

IX. Conclusions and Future Work

In this paper, we discussed the scheduling of distributed reasoning tasks in wireless sensor networks. First, we described a generic model for task scheduling in heterogeneous networks. An ILP formulation was subsequently devised, allowing us to obtain optimal schedules. A set of bin-packing heuristics was also introduced and assessed. For this purpose, we developed a realistic problem generator, using benchmarks of our WSN monitoring and management application based on distributed reasoning. Through the simulation of a large amount of such testing scenarios, we showed that the heuristics perform adequately, while maintaining a very low resource footprint.

The model and algorithms devised in this paper can already be applied to various scheduling problems in heterogeneous networks. In future research, we aim to extend them with characteristics such as network link awareness, and the specification of task durations on a per-node basis.
Issues specific to WSNs will also be modeled. One example is the periodic availability of resources, like a sensor node's wireless link; this helps the node conserve its power.
Finally, alternative objective functions will be considered. One might, for instance, prefer to minimize energy consumption.

While this paper only covers off-line scheduling, it is our intent to also investigate the on-line adaptation of schedules. Thus, at a later stage, the introduced model and algorithms will be repurposed to this end. This will allow us to cope with varying circumstances, such as a change in demand for specific components, or hardware failure.

Acknowledgment

Tim De Pauw would like to thank the University College Ghent Research Fund for financial support through his Ph.D. grant. Stijn Verstichel would like to thank the IWT (Institute for the Promotion of Innovation through Science and Technology in Flanders) for financial support through his Ph.D. grant. Our gratitude is also due to the IBBT project DEUS [30] for supporting this research. Computational resources and services used in this work were provided by Ghent University.

References

[1] K. Romer and F. Mattern, "The design space of wireless sensor networks," *IEEE Trans. Wireless Commun.*, vol. 11, no. 6, pp. 54–61, Dec. 2004.

[2] D. Culler, D. Estrin, and M. Srivastava, "Overview of sensor networks," *IEEE Trans. Comput.*, vol. 37, no. 8, pp. 41–49, 2004.

[3] L. Tytgat, B. Jooris, P. De Mil, B. Latré, I. Moerman, and P. Demeester, "Demo abstract: WiLab, a real-life wireless sensor testbed with environment emulation," in *European conference on Wireless Sensor Networks, EWSN adjunct poster proceedings*, Cork, Ireland, Feb. 2009.

[4] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *IEEE Commun. Mag.*, vol. 40, no. 8, pp. 102–114, Aug. 2002.

[5] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, Jan. 1979.

[6] M. Ghazvini, M. Vahabi, M. Rasid, R. Abdullah, and W. Musa, "Low energy consumption MAC protocol for wireless sensor networks," in *Proc. 2nd International Conference on Sensor Technologies and Applications*, 2008, pp. 49–54.

[7] J. Haapola, Z. Shelby, C. Pomalaza-Ráez, and P. Mähönen, "Multihop medium access control for WSNs: an energy analysis model," *EURASIP J. Wirel. Commun. Netw.*, vol. 2005, no. 4, pp. 523–540, 2005.

[8] S. Verstichel, F. Ongenae, B. Volckaert, F. De Turck, B. Dhoedt, T. Dhaene, and P. Demeester, "An autonomous service platform to support distributed ontology-based context-aware agents," *Expert Systems: The Journal of Knowledge Engineering on Engineering Semantic Agent Systems*, Accepted for publication.

[9] D. L. McGuinness and F. van Harmelen. (2004, Feb.) OWL Web Ontology Language Overview. [Online]. Available: http://www.w3.org/TR/owl-features/

[10] R. J. Vanderbei, *Linear Programming: Foundations and Extensions*, 3rd ed. Springer, Nov. 2008.

[11] M. Dhodhi, I. Ahmad, A. Yatama, and I. Ahmad, "An integrated technique for task matching and scheduling onto distributed heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, vol. 62, no. 9, pp. 1338–1361, Sep. 2002.

[12] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810–837, Jun. 2001.

[13] A. Medina, I. Matta, and J. Byers, "BRITE: A flexible generator of internet topologies," Boston University, Boston, MA, USA, Tech. Rep., 2000.

[14] T. Camilo, J. Silva, A. Rodrigues, and F. Boavida, "GENSEN: A topology generator for real wireless sensor networks deployment," *Lecture Notes in Computer Science*, vol. 4761, pp. 436–445, 2007.

[15] I-LENSE. Topology generator (topo_gen). [Online]. Available: http://www.isi.edu/ilense/software/topo_gen/topo_gen.html

[16] A. Sheth, C. Henson, and S. Sahoo, "Semantic Sensor Web," *IEEE Internet Computing*.

[17] Y. Hu, Z. Wu, and M. Guo, "Ontology driven adaptive data processing in wireless sensor networks," in *Proc. 2nd International Conference on Scalable Information Systems*, 2007.

[18] A. da Rocha, F. Delicato, J. de Souza, D. Gomes, and L. Pirmez, "A semantic middleware for autonomic wireless sensor networks," in *Proc. 2009 Workshop on Middleware for Ubiquitous and Pervasive Systems*. ACM Press, 2009, pp. 19–25.

[19] S. Verstichel, M. Strobbe, P. Simoens, F. De Turck, B. Dhoedt, and P. Demeester, "Distributed reasoning for context-aware services through design of an OWL meta-model," in *Proc. 4th International Conference on Autonomic and Autonomous Systems*. IEEE Computer Society, 2008, pp. 70–75.

[20] E. Prud'hommeaux and A. Seaborne. (2008, Jan.) SPARQL query language for RDF. [Online]. Available: http://www.w3.org/TR/rdf-sparql-query/

[21] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson, "Jena: Implementing the Semantic Web recommendations," in *Proc. 13th International World Wide Web Conference*. ACM Press, 2004, pp. 74–83.

[22] C. Bizer and R. Cyganiak, "D2R Server: Publishing relational databases on the Semantic Web," *5th International Semantic Web Conference*, 2006.

[23] L. Han, T. Finin, C. Parr, J. Sachs, and A. Joshi, "RDF123: A mechanism to transform spreadsheets to RDF," in *Proc. 21st National Conference on Artificial Intelligence*. Menlo Park, CA, USA: AAAI Press, 2006.

[24] E. Sirin, B. Parsia, B. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical OWL-DL reasoner," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 5, no. 2, pp. 51–53, Jun. 2007.

[25] V. Haarslev and R. Möller, "Racer: An OWL reasoning agent for the Semantic Web," in *Proc. International Workshop on Applications, Products and Services of Web-based Support Systems*, vol. 13, Halifax, Canada, Oct. 2003, pp. 91–95.

[26] J. Gross and J. Yellen, *Graph Theory and its Applications*. Boca Raton, FL, USA: CRC Press, 1999.

[27] ILOG, Inc. ILOG CPLEX. [Online]. Available: http://www.ilog.com/products/cplex/

[28] C. Kenyon, "Best-fit bin-packing with random order," in *Proc. 7th Annual ACM-SIAM Symposium on Discrete Algorithms*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, Oct. 1995, pp. 359–364.

[29] PC Engines GmbH. ALIX system boards. [Online]. Available: http://pcengines.ch/alix.htm

[30] IBBT v.z.w. DEUS: Deployment and Easy Use of wireless Services. [Online]. Available: http://www.ibbt.be/en/project/deus