The Task Assignment Problem for Safety-Critical Networks Considering Communication and Criticality

Franz Wotawa[1,*,†], Julian Proenza[2], Manuel A. Barranco[2] and Alberto Ballesteros[2]

[1]Graz University of Technology (TU Graz), Institute of Software Technology, Inffeldgasse 16b/2, A-8010 Graz, Austria
[2]Universitat de les Illes Balears, Cra. de Valldemossa, km 7.5. 07122 Palma, Illes Balears, Spain

Abstract

The task assignment problem is well-known and of great practical importance. In a previous work, we presented a corresponding answer set programming model and provided an initial experimental evaluation, demonstrating its practical applicability. However, this model falls short in several aspects, making it less than entirely suitable for safety-critical networks. In this paper, we extend the model providing means for representing critical tasks and communication. In particular, we introduce predicates for capturing communication among tasks and their limitations caused by networks, i.e., the available bandwidth. We also provide a preliminary experimental evaluation of the new model, demonstrating its feasibility for minor problem instances.

Keywords

Configuring computing nodes, ASP models for configuration, Experimental analysis

1. Introduction

Knowledge-based configuration, i.e., the composition of elements and parts to fulfill customers' needs, has garnered considerable attention. There has been a lot of research and applications reported in scientific literature, ranging from service configuration [1], governance systems [2], product configuration [3], considering hardware and software in the automotive domain [4], to green configuration in scheduling [5], only to give the most recent examples. Modelling for configuration is often based on logic, see, e.g., [6], and most recently, answer set programming (ASP) for representing models used for configuration and reasoning to obtain valid configuration has gained more attention, e.g., see [7, 3, 8, 9].

In this short paper, we continue work on system configuration, focusing on configuring networks comprising nodes for executing pre-defined tasks. As already mentioned in our previous paper [10], the underlying problem is related to scheduling and shift designs [11] and has also been considered in previous work, e.g., [12]. Furthermore, the underlying problem can be seen as a variant of the well-known knapsack problem [13, 14]. In contrast to our previous paper, we extend the underlying model to bring it closer to the intended application area, which are highly reliable networks for hard real-time systems, where faults occurring during operation need to be mitigated within a pre-defined time see e.g., [15]. Mitigation depends on the type of fault, e.g., a fault in a network node, where tasks need to be reallocated. Such a (re-) configuration needs to be fast, such that no computational real-time requirements are violated. It is worth noting that machine learning has already been suggested [16] to solve re-configuration during operation.

In particular, we introduce concepts for handling communication among nodes and their limitations. We simplify communication by considering only one bus where all nodes are connected to enable allocated tasks to communicate with each other. The formal representation allows us to state which task is communicating with another and also the costs of communication in terms of required bandwidth. In

ConfWS'25: 27th International Workshop on Configuration, Oct 25–26, 2025, Bologna, Italy
*Corresponding author.
†Authors are listed in reverse alphabetical order.
wotawa@tugraz.at (F. Wotawa); julian.proenza@uib.es (J. Proenza); manuel.barranco@uib.es (M. A. Barranco); a.ballesteros@uib.cat (A. Ballesteros)
https://www.tugraz.at/ (F. Wotawa)
0000-0002-0462-2283 (F. Wotawa)

addition, we also capture the challenge of critical tasks. A task is considered critical if its non-execution would cause a safety-relevant effect. Such a task is replicated in a safety-critical network, and voting mechanisms are applied to ensure that it is always executed. Examples of safety-critical tasks include a brake controller that must always enable braking when requested by a driver of a vehicle.

In addition to the extended model, we conduct a first limited experimental evaluation based on several instances of task assignment problems. The evaluation utilizes the answer set programming solver `clingo` [17] and answers the question whether the extended model is appropriate for being used in the context of safety-critical networks, i.e., whether it is fast enough to assign tasks whenever required.

We structure this paper as follows. First, we introduce the underlying configuration problem. Afterward, we discuss the extensions of the model and present an answer set programming solution, followed by the experimental evaluation. Finally, we conclude this short paper.

2. Problem description

In this section, we define the task-to-node assignment problem, or short *task assignment problem*. We start summarizing the problem and its related constraints from our previous paper [10]. We assume k computing nodes $n_1, \ldots, n_k$ and n tasks $t_1, \ldots, t_n$. For each node n_i, we know the maximum number of tasks $c(n_i)$ it can hold and the available memory $m(n_i)$. For each task t_j, we know its memory consumption $m(t_j)$. In the following, we use this knowledge to formulate several constraints that need to be considered when assigning tasks to nodes. For the constraints, we assume a function $assigned(n_i)$ that returns a set of tasks that is assigned to a node n_i.

1. *Memory limitations:* The required memory by the task shall never exceed the available memory of the node.
$$\forall i \in \{1, \ldots, k\} : \left(\sum_{t_j \in assigned(n_i)} m(t_j) \leq m(n_i) \right) \tag{1}$$
2. *Task limitations:* The number of tasks assigned to a node shall never exceed its capabilities.
$$\forall i \in \{1, \ldots, k\} : (|assigned(n_i)| \leq c(n_i)) \tag{2}$$
3. *Global memory limitations:* The required memory of all tasks shall never exceed the memory provided by all nodes.
$$\sum_{j=1}^{n} m(t_j) \leq \sum_{i=1}^{k} m(n_i) \tag{3}$$
4. *Global task limitations:* The number of available tasks shall never exceed the sum of the number of tasks of all nodes.
$$n \leq \sum_{i=1}^{k} c(n_i) \tag{4}$$

A solution to the *tasks assignment problem* is an assignment of all tasks to nodes such that $\forall j \in \{1, \ldots, k\} : \exists i \in \{1, \ldots, n\} : t_j \in assigned(n_i)$, there is no tasks assigned to two different nodes, i.e., $\forall i, j \in \{1, \ldots, k\}, i \neq j : assigned(n_i) \cap assigned(n_j) = \emptyset$, and all constraints are fulfilled. Such an assignment is a valid one and may not exist due to limitations regarding available memory or the total node capacity. We may also consider optimality criteria such as minimizing the number of nodes where we assign tasks.

In addition to this original `task assignment problem`, we now add further information and constraints to represent the safety-critical network more appropriately. In particular, networks are for communication. Communication channels impose further constraints due to resource limitations. For example, there is only a maximum bandwidth available. If too many tasks are communicating at the same time, the bandwidth might not be sufficient. To simplify communication, we now only consider

Table 1
Predicates used to specify nodes, tasks, and their corresponding knowledge from [10].

Predicate	
`node(`n`)`	specifies that n represents a node
`tcapacity(`n`,`c`)`	maximum number of tasks c that a node n can hold
`mcapacity(`n`,`m`)`	maximum memory m provided by node n
`task(`t`)`	specifies that t is a task
`memory(`t`,`m`)`	memory m required by task t

a bus, where all messages have to pass through. Hence, there is a limitation of the bus in terms of bandwidth BW. We now need to specify the communication needs of each task. We assume that not necessarily each tasks need to communicate with each other. Hence, we introduce a function com that maps a potentially empty set of tasks to a given task, and a function bw that maps a task t and any tasks from $com(t)$ to a necessary bandwidth. However, this bandwidth is only required if the two tasks are not assigned to the same node. If two tasks are in the same node, there is no need to use the bus for communication. Obviously, the required total bandwidth needed for communication shall never exceed the bandwidth of the bus BW.

$$\sum_{j \in \{1,\ldots,n\}} \sum_{\substack{t \in com(t_j) \wedge \\ \nexists i \in \{1,\ldots,k\}: \\ \left(\begin{array}{c} t \in assigned(n_i) \wedge \\ t_j \in assigned(n_i) \end{array} \right)}} bw(t_j, t) \leq BW \tag{5}$$

In addition to communication, we may also want to state that several tasks should never be allocated to the same node. Critical tasks are examples. Such tasks may replicate each other in behavior and are used to add fault tolerance. For simplification purposes, we only introduce a predicate $distinct$ for any two tasks stating that both are not allowed to be assigned to the same node:

$$\begin{aligned} &\forall j \in \{1,\ldots,n\} : \forall j' \in \{1,\ldots,n\}, j \neq j' : distinct(t_j, t_{j'}) \rightarrow \\ &\quad \nexists i \in \{1,\ldots,k\} : \big(t_j \in assigned(n_i) \wedge t_{j'} \in assigned(n_i)\big) \end{aligned} \tag{6}$$

The task assignment problem comprising the additional constraints 5 and 6 is the *extended task assignment problem.*

3. Implementation

After outlining the task assignment problem in the last section, we present a solution using answer set programming where we rely on the syntax of the `clingo` solver [17], which is similar to the Prolog language. For more information regarding answer set programming (ASP), we refer to introductory literature, e.g., [18]. Note also that we do not discuss the ASP model in detail, except for the new addition. The details are described in our previous paper [10]. In Table 1, we summarize the predicates necessary to specify the original task assignment problem.

To find solutions for this problem, we further introduced a predicate `select` that takes a task `T` as the first parameter and a node `N` as the second. The ASP solver selects tasks for nodes such that no constraint is violated. To be self-contained, we summarize the `clingo` source code comprising additional predicates for handling memory and task requirements:

```
% Generate a selection of a node for each task
{ select(T,N) : node(N) } = 1 :- task(T).
```

```
% Constraints
% No 1: Do not exceed the max. number of tasks
noTasksAssigned(M,N) :-
   M = #count { T : select(T,N)}, node(N).
:- noTasksAssigned(M,N), tcapacity(N,C), M>C.

% No 2: Do not exceed the max. memory of a node
memRequired(M,N) :-
   M = #sum { NM,T :select(T,N), memory(T,NM)},
   node(N).
:- memRequired(M,N), mcapacity(N,C), M > C.

% Global constraints
totalCapacity(C) :- C=#sum {T,N :tcapacity(N,T)}.
totalNrTasks(C) :- C=#count {T :task(T) }.
totalMemReq(C) :- C=#sum {M,T :memory(T,M)}.
totalMem(C) :- C=#sum {N,CN :mcapacity(CN,N)}.
:- totalCapacity(C), totalNrTasks(TC), C < TC.
:- totalMemReq(Ctask), totalMem(Cnode),
   Ctask > Cnode.
```

It is worth noting that this implementation corrects two shortcomings of the original one, which may have led to some results that should not have been consistent solutions to the task assignment problem. In the following, we now discuss the extension to this model to allow computing solutions for the *extended task assignment problem.*

Let us handle the communication between nodes first. For stating communication needs between tasks, we introduce a predicate `comReq\3` that states communication needs between two tasks and the required bandwidth. Hence, this predicate more or less captures the functions com and bw. For example, `comReq(t1,t2,10)` states that there is a message transfer from task `t1` to `t2` requiring a bandwidth of 10. What we need to formalize is the communication need and a constraint stating bandwidth violation. For the former, we introduce a predicate `comNeed\3` that summarizes communication needs between two tasks depending on their assignment to nodes. If they are at the same node, there is no communication bandwidth required. Otherwise, it is stated as defined in `comReq`. The following rules capture this behavior:

```
comNeed(T1,T2,0) :- select(T1,N), select(T2,N).
comNeed(T1,T2,B) :-
   select(T1,N1), select(T2,N2),
   N1!=N2, comReq(T1,T2,B).
```

Based on `comNeed`, we can now specify the sum of the communication bandwidth required, which we define as follows, utilizing the predicate `comRequired` for the whole bus:

```
comRequired(M) :-
   M=#sum {B,T1,T2 :comNeed(T1,T2,B),
                    task(T1), task(T2)}.
```

Finally, we state the communication constraint that the communication required is not allowed to exceed the total bandwidth provided by the bus (which is 50 in this particular case):

```
comChannel(50).
:- comChannel(B), comRequired(M), M>B.
```

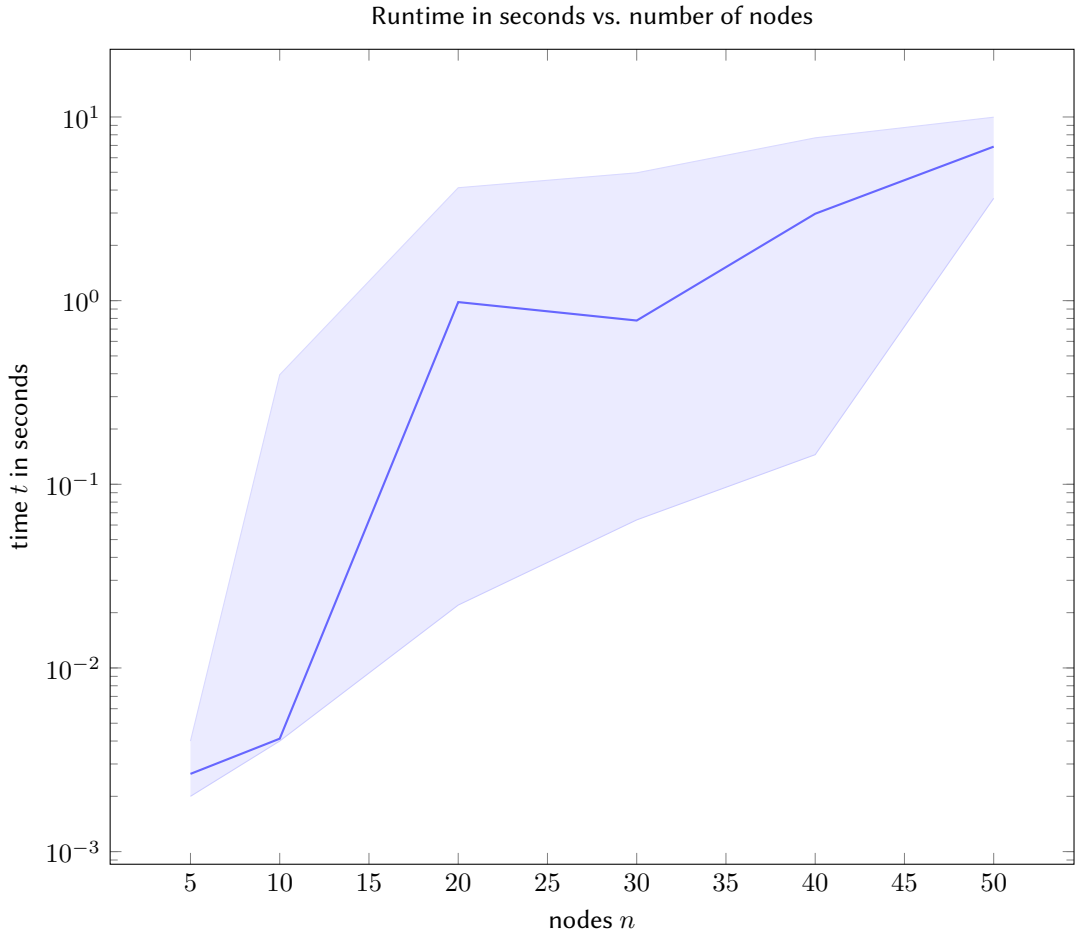

Figure 1: Solving runtime of satisfiable and unsatisfiable instances

For handling information about critical nodes, we introduce the `distinct\2` predicate to state that two tasks should never be assigned to the same node, e.g., `distinct(t1,t2)` states that tasks `t1` and `t2` has to be assigned at different node. Stating this constraint is straightforward:

```
:- select(T1,N), select(T2,N), distinct(T1,T2).
```

It is worth noting that this model is still a simplified representation of the task assignment for safety-critical networks. But it covers certain important aspects, which have not been considered before.

4. Experimental evaluation

Similar to the experimental evaluation in our previous paper [10], we want to investigate the runtime behavior of the ASP solver `clingo` when using systems comprising a different number of tasks and nodes. In particular, it is interesting to know how many nodes can be handled within a fixed time span of, e.g., 0.01 or 0.1 seconds. In addition, we are interested in the effects of the additional constraints on the runtime.

Experimental setup: We used a Java program for generating model instances automatically, where we ranged the number of nodes from 5, 10, 20 to 100 and the number of tasks randomly between the number of nodes and its double. The capacity of each node was randomly set from 1 to 10. The memory provided by each node was randomly chosen from 20, 40, 60,..., 200. The memory required by every task was randomly set to 10, 20, or 30. Moreover, we randomly selected whether a task communicates

Nodes n	SAT + UNSAT	SAT
5	0.003	0.003
10	0.041	0.041
20	0.982	0.752
30	0.779	0.361
40	2.973	2.298

Table 2
Average runtime in seconds of satisfiable and unsatisfiable instances in comparison with satisfiable instances only.

with another and also whether two tasks are distinct. We obtained two different test sets, each of size 110, considering two different probability settings. We conducted the experiments using an Apple MacBook Pro, with an Apple M1 CPU comprising 8 cores and 16 GB of main memory, running under macOS Sequoia Version 15.5. For computing solutions, we relied on `clingo` version 5.7.1 and applied the standard setup. Note that this setup (with the exception of the underlying operating system) is the same as used in our previous paper [10] to allow for a comparison.

Experimental results: After generating the problem instances, we ran `clingo` to compute one solution, i.e., we ran `clingo` using the prompt `clingo -time-limit=10 -outf=2` where we set a time limit of 10 seconds and obtained all results in JSON format. What we observed is that the underlying new constraints impact the runtime. For the first test set, we exceeded the time limit 76 times. From the remaining instances, 29 were satisfiable and 5 were unsatisfiable. For the second test, the number of instances where we could not establish a solution drops to 67. The number of satisfiable instances increases to 43, and no unsatisfiable instance was obtained.

Figure 1 depicts the minimum, maximum, and average runtime for all satisfiable and unsatisfiable runs for each category where data was available. In comparison with the results from our previous paper [10], we see a big difference. Only smaller instances comprising less than 10 nodes can now be configured in less than 0.1 seconds, which was 20 in our other publication. Hence, the additional constraints have a substantial impact, which is also visible by the high number of instances that cannot be analyzed within the 10-second boundary.

We further compared the average runtime of all satisfiable and unsatisfiable instances with the one obained considering only satisfiable instances. Table 2 summarizes the results where we only consider nodes where enough instances for a comparison remain. We see that when considering unsatisfiable instances the runtime increases on average, which is in line with results obtained in our previous paper.

Threats to validity: The presented results are from an initial experimental evaluation. The experimental setup is limited, not considering the entire range of potential parameters. Due to the time boundary set, there are many instances where satisfiability or unsatisfiability cannot be assigned. The setup also does not allow for answering several interesting questions, like the responsibility of certain constraints for the increased runtime.

5. Conclusions

In this paper, we extend an existing model for the task assignment problem, considering constraints for communication and also for tasks that should not run on the same computing node. We further present the results of an initial experimental evaluation, which show that there is an impact on the overall runtime, potentially limiting its practical use to minor instances. However, the current evaluation is limited, and further experimental evaluations and a more in-depth analysis are necessary. In future work, we aim to address the questions regarding the influence of specific constraints on the overall runtime and develop a more sophisticated test set that considers a wider variety of parameters.

Acknowledgments

The work was supported by the Austrian Science Fund (FWF) Cluster of Excellence Bilateral AI under contract number 10.55776/COE12.

Declaration on Generative AI

During the preparation of this work, the authors used Grammarly in order to: Grammar and spelling check, Paraphrase, and Reword. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] E. M. Strøm, T. M. Münsberg, L. Hvam, Identifying potential applications of service configuration systems in a logistics company, in: Proc. of the 25th Intern. Workshop on Configuration (ConfWS 2023), Málaga, Spain, September 6-7, 2023, volume 3509 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023, pp. 60–66. URL: https://ceur-ws.org/Vol-3509/paper9.pdf.

[2] S. Muñoz-Hermoso, D. Benavides, F. J. D. Mayo, Multi-level configuration in smart governance systems, in: Proc. of the 25th Intern. Workshop on Configuration (ConfWS 2023), Málaga, Spain, September 6-7, 2023, volume 3509 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023, pp. 67–74. URL: https://ceur-ws.org/Vol-3509/paper10.pdf.

[3] R. Comploi-Taupe, G. Friedrich, T. Niestroj, Dynamic aggregates in expressive ASP heuristics for configuration problems, in: Proc. of the 25th Intern. Workshop on Configuration (ConfWS 2023), Málaga, Spain, September 6-7, 2023, volume 3509 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023, pp. 75–84. URL: https://ceur-ws.org/Vol-3509/paper11.pdf.

[4] F. Jost, C. Sinz, Challenges in automotive hardware-software co-configuration, in: Proc. of the 26th Intern. Workshop on Configuration (ConfWS 2024), Girona, Spain, September 2-3, 2024, volume 3812 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2024, pp. 17–20. URL: https://ceur-ws.org/Vol-3812/paper2.pdf.

[5] C. M. Moya, C. Pérez, M. A. Salido, Developing an algorithm selector for green configuration in scheduling problems, in: Proc. of the 26th Intern. Workshop on Configuration (ConfWS 2024), Girona, Spain, September 2-3, 2024, volume 3812 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2024, pp. 41–49. URL: https://ceur-ws.org/Vol-3812/paper6.pdf.

[6] A. Felfernig, G. Friedrich, D. Jannach, M. Stumptner, Consistency based diagnosis of configuration knowledge-bases, in: Proceedings of the Tenth International Workshop on Principles of Diagnosis, Loch Awe, 1999.

[7] S. Mishra, Product configuration in answer set programming, Electronic Proceedings in Theoretical Computer Science 345 (2021) 296–304. URL: http://dx.doi.org/10.4204/EPTCS.345.46. doi:10.4204/eptcs.345.46.

[8] N. Rühling, T. Schaub, T. Stolzmann, Towards a formalization of configuration problems for asp-based reasoning: Preliminary report, in: Proc. of the 25th Intern. Workshop on Configuration (ConfWS 2023), Málaga, Spain, September 6-7, 2023, volume 3509 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023, pp. 85–94. URL: https://ceur-ws.org/Vol-3509/paper12.pdf.

[9] R. Comploi-Taupe, A. A. Falkner, S. Hahn, T. Schaub, G. Schenner, Interactive configuration with ASP multi-shot solving, in: Proc. of the 25th Intern. Workshop on Configuration (ConfWS 2023), Málaga, Spain, September 6-7, 2023, volume 3509 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023, pp. 95–103. URL: https://ceur-ws.org/Vol-3509/paper13.pdf.

[10] F. Wotawa, Using answer set programming for assigning tasks to computing nodes, in: Proc. of the 26th Intern. Workshop on Configuration (ConfWS 2024), Girona, Spain, September 2-3, 2024, volume 3812 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2024, pp. 64–67. URL: https://ceur-ws.org/Vol-3812/paper9.pdf.

[11] M. Abseher, M. Gebser, N. Musliu, T. Schaub, S. Woltran, Shift design with answer set programming, in: Logic Programming and Nonmonotonic Reasoning - 13th International Conference, LPNMR 2015, Lexington, KY, USA, September 27-30, 2015. Proceedings, volume 9345 of *Lecture Notes in Computer Science*, Springer, 2015, pp. 32–39. URL: https://doi.org/10.1007/978-3-319-23264-5_4. doi:10.1007/978-3-319-23264-5_4.

[12] M. Nica, B. Peischl, F. Wotawa, A constraint model for automated deployment of automotive control software, in: Proceedings of the Twentieth International Conference on Software Engineering & Knowledge Engineering (SEKE'2008), San Francisco, CA, USA, July 1-3, 2008, Knowledge Systems Institute Graduate School, 2008, pp. 899–904.

[13] V. Cacchiani, M. Iori, A. Locatelli, S. Martello, Knapsack problems – an overview of recent advances. part i: Single knapsack problems, Computers & Operations Research 143 (2022) 105692. URL: https://www.sciencedirect.com/science/article/pii/S0305054821003877. doi:https://doi.org/10.1016/j.cor.2021.105692.

[14] V. Cacchiani, M. Iori, A. Locatelli, S. Martello, Knapsack problems – an overview of recent advances. part ii: Multiple, multidimensional, and quadratic knapsack problems, Computers & Operations Research 143 (2022) 105693. URL: https://www.sciencedirect.com/science/article/pii/S0305054821003889. doi:https://doi.org/10.1016/j.cor.2021.105693.

[15] A. Ballesteros, M. Barranco, J. Proenza, L. Almeida, F. Pozo, P. Palmer-Rodríguez, An infrastructure for enabling dynamic fault tolerance in highly-reliable adaptive distributed embedded systems based on switched ethernet, Sensors 22 (2022) 7099. URL: https://doi.org/10.3390/s22187099. doi:10.3390/S22187099.

[16] R. Rotaeche, A. Ballesteros, J. Proenza, Speeding task allocation search for reconfigurations in adaptive distributed embedded systems using deep reinforcement learning, Sensors 23 (2023) 548. URL: https://doi.org/10.3390/s23010548. doi:10.3390/S23010548.

[17] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Multi-shot asp solving with clingo, Theory and Practice of Logic Programming 19 (2019) 27–82. doi:10.1017/S1471068418000054.

[18] T. Eiter, G. Ianni, T. Krennwallner, Answer set programming: A primer, in: Reasoning Web. Semantic Technologies for Information Systems: 5th International Summer School 2009, Brixen-Bressanone, Italy, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 40–110. URL: https://doi.org/10.1007/978-3-642-03754-2_2. doi:10.1007/978-3-642-03754-2_2.