Published in proceedings of the 1st International Workshop on Real-Time compliant Multi-Agent Systems co-located with the Federated Artificial Intelligence Meeting, Stockholm, Sweden, 15 July 2018, which should be cited to refer to this work.

A Task-sets Generator for Supporting the Analysis of Multi-Agent Systems under General Purpose and Real-Time Conditions

Davide Calvaresi[1,2], Giuseppe Albanese[2], Fabien Dubosson[2], Mauro Marinoni[1], and Michael Schumacher[2]

[1] Scuola Superiore Sant'Anna,Pisa, Italy
[2] University of Applied Sciences Western Switzerland, Sierre, Switzerland
m.marinoni@sssup.it, {name.surname}@hevs.ch

Abstract. The adoption of Multi-Agent Systems (MAS) is permeating Internet of Things (IoT) and Cyber-Physical Systems (CPS). Timing reliability of MAS is a daring challenge. The study of local task execution and negotiation of workloads are catalyzing considerable interest. By adopting techniques typical of Real-Times Systems (RTS), MAS's ability to comply with strict timing constraints has been proven. However, a complete formalization is still missing, and some of the existing mathematical models introduce considerable pessimism in the performance analysis. Therefore, the need for tools supporting the study of the behavior of agent-based systems is rising. Particularly, the capability of systematic assessment and comparison of their performance.
This paper presents a system to generate task-sets and operating scenarios, to support the study of timing reliability, behavior, and performance of MAS. The parameters required for such a generation are characterized by randomly extracted values (e.g., the number of agents, single agent utilization factors, and single task utilization factor). For each parameter, it is possible to select a given statistical distribution to be applied to user-defined ranges. In particular, logic, constraints, and dependencies characterizing the generation algorithm are detailed and framed in a functional work-flow. Moreover, such a system integrates a MAS simulator powered by both general-purpose and real-time algorithms, named MAXIM-GPRT. Hence, the presented tool is also able to show the logs of the tested scenarios equipped with graphs to enable the performance analysis.

Keywords: Task-Set Generator, Timing-Reliability, Deadline Miss Ratio, Scheduling Simulation, Multi-Agent Simulator, Performance Analysis

1 Introduction

The maximal expression of distributed systems pervading humankind's daily living is represented by the Cyber-Physical Systems (CPS) [9]. The distributed entities composing a CPS interact with each other and with their surrounding, collecting data, negotiating, and providing services. An increasing range of distributed applications has been developed according to the agent-based paradigm. For example, Multi-Agent Systems (MAS) can be mentioned in domains such as e-health [5, 6], telerehabilitation [8], energy [15], and manufacturing [14].

However, their employment in safety-critical scenarios has been recently questioned [9]. This is due to the fact that a system is considered able to guarantee its correct execution if it can deliberate the right output at or within a given time, even in the worst-case scenario [3], a feature not yet provided by current MAS.

This class of systems is characterized by a multitude of elements with factors operating simultaneously. To study their primitive mechanisms, the compliance with strict timing constraints, the load balancing, and the overall performance, they have to be individually and collectively studied.

Investigating the timing-reliability, Calvaresi et al. studied the behavior and role of the local-scheduler [7], and the assumptions, requirements, strengths, and limitations of current negotiation protocols [4] in MAS.

Currently, the most used agent-based framework and systems take the negotiation, allocation, and execution of tasks/services for granted [7]. Therefore, Calvaresi et al. developed an agent-based simulator, named MAXIM-GPRT, to analyze qualitatively and quantitatively the behavior of a set of local schedulers, negotiation protocols, and the overall dynamics of the studied agency [1]. Still unique in its purpose, such a simulator accepts a broad range of parameters in inputs, enabling the outcomes listed in Table 1.

Table 1: Possible simulator outcomes

Id	**Indicator**	**Description**
I1	Deadline Miss Ratio (DMR)	Number of deadlines missed by a task in a given simulated time.
I2	Lateness (LT)	Extra time required by a task missing its deadline to complete.
I3	Response Time (RTM)	Amount of time required to complete a given task.
I4	Performance Analysis	Assessment of the obtained results.

However, by investigating the state of the art, the need for a standardized and semi-automatic manner to generate task-sets and MAS scenarios is still clearly unmet. Therefore, in the context of MAS, our aim is to support studies of performances analysis, risk and failure detection [3], in both General-Purpose (GP) and Real-Time (RT) scenarios.

Contributions:

This paper presents a generator of task-sets and scenarios that we have recently developed. Although it has been conceived to cope with MAS, it can be employed to study any type of system requiring the generation of task-sets according to a given set of parameters and bounds. The task-sets are composed of tasks, which are randomly generated and subject to given statistical distributions applied to user-defined bounds. The scenarios are characterized by a set of parameters representing the operating conditions and the selected algorithms (see Table 2).

The paper is organized as follows: Section 2 presents and elaborates the state of the art, Section 3 describes the task-set and scenario generator, Section 4 details the steps and a case study used as an example. Section 5 discusses about the design approach and assesses the accuracy of the generator. Finally, Section 6 concludes the paper.

[3] with particular emphasis for safety-critical systems

2 State of the art

On one hand, by investigating the state of the art in the domain of multi-agent systems, the notion of *scheduling* applies mainly to mechanisms of task/resources allocation *among* the agents rather than concerning the arrangement and execution of the local tasks/behaviors *within* a given agent [7]. Therefore, no similar tool can be mentioned. On the other hand, a few solutions arose in the domain of real-time systems. De Bock et al. [11] presented a tool allowing the generation of synthetic task-sets. Its main purpose is the evaluation of the performance obtained from the tested scheduling algorithms and to determine the Worst Case of the computation times (C) of the tested tasks.

Such a tool is composed of three main modules operating sequentially:

(i) *the generator* produces a synthetic task-set composed of periodic tasks with implicit deadlines (i.e. Deadline (D) = Period (T)). Such tasks are generated according to the following parameters:
- range of task-set utilization;
- step value between two utilization;
- number of task-sets per utilization;
- number of tasks per task-set;
- lower and upper bound of the period;
- level of granularity of the period;
- seed value for pseudo-random generators.

Firstly, the generator assigns a utilization U^{τ_i} to each task according to the constraint that $\sum_{i=0}^{n} U^{\tau_i} = U_t$, where U_t is the total utilization of the task-set.
Then, the period of each task is generated within the given bounds, and the computation time is calculated according to Equation 2.

(ii) Secondly, *the Timing Analysis on Code-Level Benchmark suite* (TACLeBench) evaluates the Worst Case Execution Time (WCET) for each generated task.

(iii) Finally, the task-set is implemented as a collection of C++ functions and a makefile. Compiling that source code produces a set of executables which can be run on a target hardware.

Emerson et al. [12] outlined the Randfixedsum algorithm to study the underlying mathematical problem of the efficient generation of uniformly distributed random points whose elements sum-up to a predefined value (i.e., 1, indicating a feasible task set). In the uni-processor case, the UUniFast algorithm created by Bini and Buttazzo [2] has proven to be able to generate an unbiased distribution of task-sets. The extension for the multi-processor case is given by UUniFast-Discard, suggested by Davis and Burns [10]. This algorithm allows the application of the classic UUniFast for values of $U > 1$. It randomly generates n task utilization, picking each one from a predefined range $[0, U_t]$, where U_t can be greater than 1. If any of the generated tasks has utilization greater than 1, such a solution is discarded. The main issue of this algorithm is its efficiency (i.e., it becomes increasingly inefficient as the value of U approaches $n/2$, with n indicating the number of the cores). Once the task-set is generated, it is distributed among the cores. Stafford et al. [16] compare the UUniFast and the UUniFast-Discard with the Randfixedsum. This last resulted in being more efficient to generate feasible task-sets. Finally, on top of the task-sets generation, the Multi-Core Real-Time

Systems simulator (MCRTsim) [17] provides a workload generator. Each workload is characterized by a base speed, a set of shared resources, and a set of tasks. Each task is characterized by unique id, type, arrival time, period, relative deadline, computation time, and a set of critical sections. Aperiodic tasks are only characterized by an arrival time and a computation amount. These features are defined in an XML file that is given as an input to the above mentioned simulator.

However, the requirements of agent-based systems, particularly the ones of the simulator MAXIM-GPRT, go beyond what is currently available. Therefore, the next section presents the tool that we have developed to cope with the still unmet requirements.

3 The generator tool

To support a comprehensive and meaningful analysis of the indicators provided by MAXIM-GPRT (see Table 1), there is the need for coordinating many parameters and inter-dependencies. This section presents elements, constraints, and logic standing behind the presented tool.

3.1 Task-set and Setup Generation

The parameters characterizing the generated task-sets and setups are listed in Table 2.

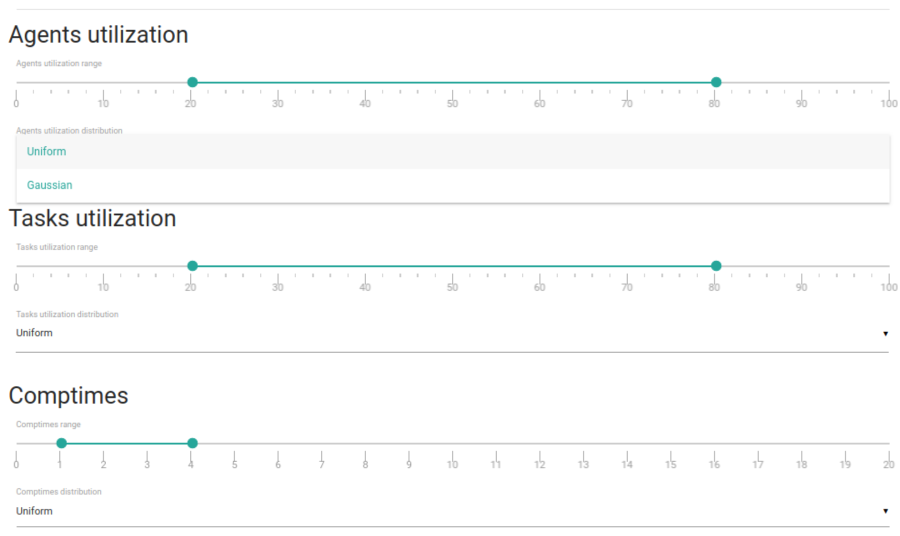
Table 2: Configurable parameters

Id	Parameter	Description
P1	Number of agents	number of agents participating in the simulation.
P2	Agent knowledge	set of tasks an agent is able to execute.
P3	Agent task-set	set of running tasks.
P4	Agent Services	set of tasks an agent might execute on demand.
P5	Agent Needs	set of tasks an agent needs, but it is unable to execute.
P6	Tasks models	typology of running tasks.
P7	Agent utilization	load of the agent's CPU (see Equation 1).
P8	Tasks utilization	load of a single task (see Equation 2).
P9	Tasks Computation time	computation time of a single task (see Equation 2).
P10	Negotiation prot.	mechanisms used to negotiate task execution.
P11	Local scheduler	algorithm scheduling the agent tasks/behaviors.
P12	Heuristics	policies used by agents to select possible contractors and to award them.

The features P4 and P5 are expressed in the form of a percentage. Such a value indicates the number of services and needs requiring generation with respect to the number of tasks composing the task-set. Moreover, it is possible to define a range indicating the time of release of the needs[4]. Similarly, the generation of the parameters P7, P8, and P9

[4] It triggers the needs release during the simulation, abstracting the "will" of the agent .

depends on the ranges chosen by the user and related statistical distributions (e.g., *uniform* and *Gaussian*). Figure 1 shows that P7 and P8 will be generated in the $[0.2 - 0.8]$ range, and that P9 will be generated in the $[2 - 4]$ range, according to the selected distributions. Hence, it is possible to select the statistical distribution singularly.



Agents utilization

Agents utilization range

Agents utilization distribution

Uniform

Gaussian

Tasks utilization

Tasks utilization range

Tasks utilization distribution

Uniform

Comptimes

Comptimes range

Comptimes distribution

Uniform

Fig. 1: Selectors for the ranges of P7, P8, and P9 (web interface).

The number of agent composing the simulated community (P1) indicates the number of task-sets to be generated. The user can define it assigning an integer value $(x \geq 1)$ to such a variable;

A task is characterized by: id, executor, demander, computation time [5], residual computation time, arrival time[5], relative deadline, period[5], number of executions, first activation time, last activation time, public flag, and server id. The task-set is generated in CSV and XML (see listing 1) formats.

The set of tasks that an agent is capable of executing represents its knowledge (P2). Elements of such tasks can be labeled as *public*, which are *services* (P4) eligible to be demanded for by other agents[6]. The *needs* (P5) are tasks that an agent have to execute at a certain point in time. Such tasks might be part of its knowledge (P2) and/or marked *public* by other agents. For each agent, the running tasks (among those contained in their P1) compose the task-set (P3).

Figure 2 proposes a graphical representation of P2, P3, P4, and P5. In particular, such sets are generated as follows: *(i)* generation of P3, *(ii)* possibly marking a given percentage of the tasks in P3 *public* and generating new services, and *(iii)* When the

[5] Values computed according to a uniform or Gaussian probability distribution.
[6] The execution of such tasks is subject to negotiation mechanisms.

P4 of all the agents have been generated, a percentage of needs with the related release time [7] is associated to each agent.

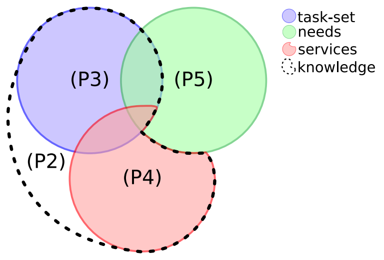


Fig. 2: Graphical representation of P2, P3, P4, and P5.

The task (P6) models that can be generated are: periodic, periodic in an interval, and aperiodic [3]. Recalling that the *processor utilization factor* U (P7) is the sum of the fractions of processor-time spent to execute a task-set composed of n tasks [3], it is calculated according to Equation (1).

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \tag{1}$$

The user can define a *lower bound* (u_l^a) and an *upper bound* (u_h^a), such that P7 can assume a real value $(0 < u_l^a \leq x \leq u_h^a \leq 1)$. Thus, all the generated task-sets are schedulable under the *Earliest Deadline First* (EDF) conditions (*Processor utilization Factor* $U \leq 1$)[8].

Moreover, the user can define two ranges in which the following features of the generated tasks are extracted:

- a *lower bound* (u_l^τ) and an *upper bound* (u_h^τ), such that the single task-utilization U_i is a real number $x \mid (0 < u_l^\tau \leq x \leq u_h^\tau \leq 1)$ (P8),
- a *lower bound* (c_l^τ) and an *upper bound* (c_h^τ), such that the *computational time* C_i is an integer number $x \mid (0 < c_l^\tau \leq x \leq c_h^\tau)$ (P9).

Thus, according to the generated U_i and C_i, the *period of the task* T_i is computed as shown in Equation (2). Finally, the generation of the tasks composing a given task-set stops when the sum of their U_i matches (P7).

$$T_i = \frac{C_i}{U_i} \tag{2}$$

Concerning the negotiation protocols (P10), the demonstrated tool proposes the selection of a generic implementation of the Contract Net Protocol and Reservation-Based Negotiation Protocol (RBN) [9]. Concerning the local scheduler (P11), the First Come

[7] values subject to given ranges and distributions.

[8] EDF has been chosen since, according to Horn [13], it produces optimal schedules and its *lower upper bound* U_{lub} is equal to 1.

First Served (FCFS), Round Robin (RR), EDF, and Constant Bandwidth Server (CBS) selection is proposed [3]

Heuristics can be selected for key activities such as *(i)* selecting which agents require a given service, *(ii)* how to elaborate a proposal for a given request, and *(iii)* how to select the agent to *award* for the execution of the negotiated service. Those heuristics are:

- (H1) select first agent in the list,
- (H2) select a random agent,
- (H3) select a random subset of agents,
- (H4) select the best offer according to a cost function.

P1 and P10 are the only parameters valid for the entire community. The remaining parameters can be set differently for each agent.

```
<taskParameters>
  <id>1</id>
  <agentExecuter>0</agentExecuter>
  <agentDemander>0</agentDemander>
  <computationTime unit="s">3</computationTime>
  <residualComputationTime unit="s">3</residualComputationTime>
  <arrivalTime unit="s">0</arrivalTime>
  <relativeDeadline unit="s">7</relativeDeadline>
  <period unit="s">7</period>
  <n_exec>-1</n_exec>
  <firstActivationTime>-1</firstActivationTime>
  <lastActivationTime unit="s">-1</lastActivationTime>
  <isPublic>true</isPublic>
  <server>0</server>
</taskParameters>
```

Listing 1: Generated task in XML format.

Finally, Figure 3 shows the distribution and characterization of presented parameters. It is worth to notice that the elements of both the *agents* and *tasks* have been included in the generator tool.

3.2 Functionalities

The operations and functionalities required to generate task-sets and scenarios according to the input set by the user are reported in the form of pseudocode by the Listing 2.

Once the initial ranges for all the parameters have been set, the generation can be triggered. The tool starts iterating on the agents number to generate the related task-sets. For each of them, it generates the agent utilization (P7), and it iterates generating tasks to reach it. By defining U^a_{temp} equal to P7 minus the sum of the generated P8 for that task-set, the stop conditions are:

(i) if $(U^\tau_l \leq U^a_{temp} < U^\tau_h)$, a task is generated with u^τ equal to the remaining utilization to match P7;

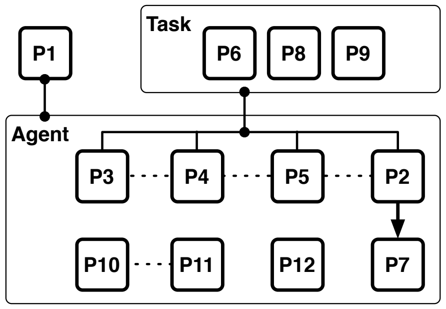


Fig. 3: Graphical representation of a Scenario.

(ii) if $(U^a_{temp} < U^\tau_l)$, the task generation for that task-set is stopped (no task respecting the preset range and distribution can be generated).

```
/* Given that: na = number of agents, Ua = agent utiliaztion,
Ut = task utiliaztion, Ct = task computation time,
ut_min = min task utilization, ut_max = max task utilization,
ct_min = min task computation, ct_max = max task computation,
Tt = task period
*/

CreateScenario(na)
    n=0, k=0
    while ( n < na)
        Ua = 0
        P7 = GenerateUa()
        while ((P7 - Ua) <= ut_max) :
            GenerateTask()
            Ua = Ua + P8
        if (( P7 - Ua) >= ut_min) :
            GenerateTask()
            Ua = Ua + P8
        GenerateServices()
        n++
    while ( k < na)
        GenerateNeeds()
        n++

GenerateTask()
    P8 = generateUt(ut_min,ut_max)
    P9 = generateCt(ct_min,ct_max)
    Tt = P9 / P8
```

Listing 2: Operation of the presented tool.

According to Buttazzo [3], C and T have to be an integer. To compute T (already subject to relation expressed in Equation 2) we impose constraints to U^τ and C^τ. Thus, the values generated initially require some adjustments. For C, i only the integer is taken, whilst for T, the ceiling of the targeted value is taken. The next section proposes a case study to better understand the generation of the task-sets.

4 Examples and Results

Table 3 proposes the setup characterizing the case study used to show the main functionality of the presented tool. Concerning the generation of the parameters within the ranges $(U_l^a - U_h^a)$, $(U_l^\tau - U_h^\tau)$, and $(C_l^\tau - C_l^\tau)$, the current case study employs a uniform distribution. This case study focuses on the generation of task-sets. Therefore, the generation of needs and services has been neglected[9]. Table 4 presents the task-

Table 3: Setup of the case study

N^a	U_l^a	U_h^a	U_l^τ	U_h^τ	Needs Interval	C_l^τ	C_h^τ	Needs %	Needs Interval
3	0.6	0.9	0.1	0.4	0	1	10	0	0

sets generated according to Table 3, detailing the values of the main variables at each step. The objective is to show the behavior of the algorithm shown in Listing 2, thus facilitating its understanding.

Table 4: Steps and values of the presented case study.

step	id^a	id$^\tau$	t. U^τ	g. U^τ	t. C	g. C	t. T	g. T	t. U^a	g. U^a	$U^a - \sum U^\tau$	s.c.
1	0								0.626			
2		0	0.161	0.160	9.899	9	55.772	56			0,466	
3		1	0.384	0.375	3.542	3	7.808	8			0.091	
4										0.535		$0.091 < U_l^\tau$
5	1								0.786			
6		0	0.393	0.391	9.318	9	22.894	23			0.393	$U_l^\tau < 0.393 < U_h^\tau$
7		1	0.393	0.333	1.58	1	2.542	3		0.724	0.060	$0.06 < U_l^\tau$
8	2								0.800			
9		0	0.1	0.100	8.19	8	79.9	80			0.700	
10		1	0.123	0.120	3.627	3	24.376	25			0.580	
11		2	0.2	0.200	1.855	1	4.917	5			0.380	$U_l^\tau < 0.38 < U_h^\tau$
12		3	0.38	0.330	1.141	1	2.626	3		0.753	0.050	$0.050 < U_l^\tau$

Legend

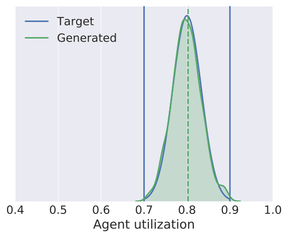
id^a = agent id; id$^\tau$ = task id; t. = targeted; g. = generated; s.c. = stop criteria.

[9] setting to 0 the related parameters neither needs or services have been generated.

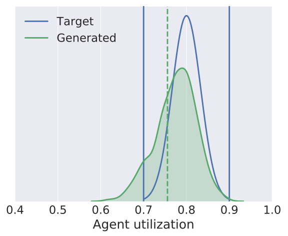
5 Discussion

To realize this generator tool, we followed a top-down approach. Hence, as shown in Listing 2, the sequence to break down the problem starts by *(i)* generating the parameters per agent, then *(ii)* generating the parameters per tasks-set (for every given agent), and finally *(iii)* generating parameters for any given task.

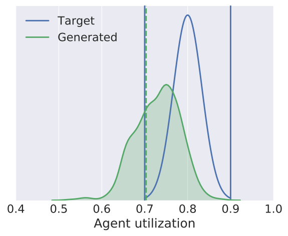
Concerning the accuracy of the generated task-sets, Figure 4 shows how close generated task-sets meet the requirements expressed in input. In particular, Figure 4 plots the agents utilization values obtained by generating a task-set with 1000 agents on three different ranges of U^τ $(0.01 - 0.05; 0.1 - 0.2; 0.2 - 0.3)$. The targeted U^a is defined in the range $U^a_l = 0.7$ and $U^a_h = 0.9$. Both U^a and U^τ employ a Gaussian distribution.



(a) $0.01 \leq U^\tau \leq 0.05$



(b) $0.1 \leq U^\tau \leq 0.2$



(c) $0.2 \leq U^\tau \leq 0.3$

Fig. 4: Graphical representation of U^a target and U^a generated [classic method].

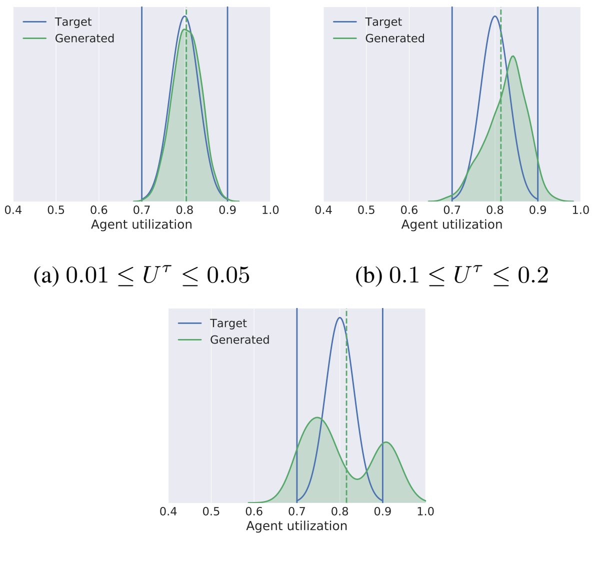
Thus, it is possible to notice that the accuracy is given by U^τ_l. Its impact, in the form of a shift between the centers of the two bells (targeted and generated), becomes more relevant when U^τ_l rises.

The principal cause generating this behavior is that *the last task is generated only if the remaining U^a is greater than U^τ_l* (see line 16 in Listing 2). This results in an inevitable shift downwards of the distribution.

To reduce this phenomena, we allowed the task utilization range to expand. Mean and distribution are initially kept centered and then shifted. This is possible by overriding the final criteria to generate the last task employed in the algorithm presented above. The precision (pr) is now defined as $pr = U^\tau_l/2$. Consecutively, the new U^a

target becomes $U^a + pr$. Such a tweak, helps to recenter the distribution, setting pr as the new precision. Nevertheless, borderline cases (e.g., U_h^τ = 1) still need attention. A possible solution would be to recompute those task-sets that exceeded such a bound.

Figure 5 is obtained by repeating the tests performed to generate Figure 4 and applying the updated stop condition.



(a) $0.01 \leq U^\tau \leq 0.05$ (b) $0.1 \leq U^\tau \leq 0.2$

(c) $0.2 \leq U^\tau \leq 0.3$

Fig. 5: Graphical representation of U^a target and U^a generated [update method].

Although minimally observable, the utilization of the generated task-sets for $0.1 \leq U_l^\tau < 0.2$ (Figure 5b) starts to shift towards a bi-modal distribution. In the case of $0.2 \leq U_l^\tau < 0.3$ (Figure 5c), such a utilization assumes a pronounced bi-modal Gaussian distribution.

Such behavior is due to the employement of a Gaussian distribution which, with the above mentioned bound, generates in average $U^\tau \sim 0.25$. By summing up three tasks we get $U^a \sim 0.75$. Therefore, recalling that the stop condition is $U^a - \sum U^\tau < U_l^\tau$, if $U^a > 0.8$ for $U_l^\tau = 0.2$, the task generation stops. Thus, explaining the distribution-gap.

Observing the inputs listed in Table 2 and related constraints highlights how the presented tool copes with an over-determined system. Let us consider, for example, the constraints (statistical distributions) applied to U^a and U^τ. Being related to each other as shown in Equation 1, results in the behavior shown in Figure 5c as a direct consequence.

Over-determined systems have been studied for decades. A possible solution simplifying the approach can be used for *constrained optimization techniques* (e.g., by employing linear programming solvers). Adopting a bottom-up approach results in the sequence *(i)* generating the parameters following their distributions, *(ii)* aggregating such parameters to compose tasks, and *(iii)* distributing tasks among the agents. Such a new approach might optimize the overall system while respecting all the preset constraints. This new approach is under evaluation to be included in the future version of the presented tool.

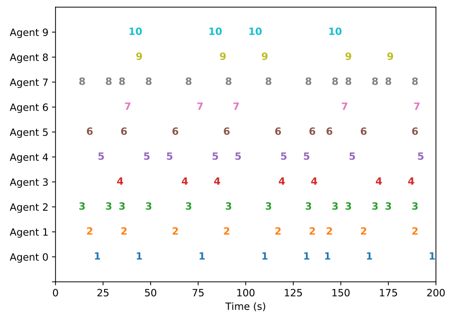
5.1 Additional features

With respect to the traditional command-line interface, the presented tool also provides a web interface. Such a version is still an ongoing work. However, at the current status, it provides a few additional features such as the enabling of the visualization of graphs supporting the analysis of the results obtained by the generated task-sets and scenarios simulated in MAXIM-GPRT [1].

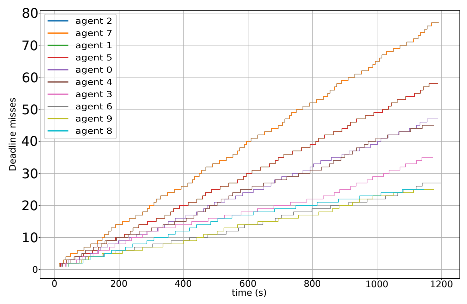
Concerning the indicators provided by MAXIM-GPRT (see Table 1), it is possible to visualize:

(i) which tasks missed their deadlines within an agent in a given time window (see Figure 6a)

(i) the trend of the deadline-missed of all the tested agents in a given time window (see Figure 6b).



(a) Deadline misses for 10 agents, time window [0-200s].



(b) Deadline misses for 10 agents time window [0-1200s].

Fig. 6: Results of the deadline miss analysis.

6 Conclusions

Fostering the study of timing-reliability and load balancing in MAS, this work presented a tool to generate task-sets and scenarios for testing purposes. The current implementation provides the output in the format supported by the MAXIM-GPRT, but can be easily extended to handle different output formats.

Considering that the performances of general-purpose scheduling algorithms are strongly dependent from the features of the scheduled task-set, it is impossible to provide any off/on-line timing guarantee. Such a study faces more challenges if it refers to communities of agents scheduling and negotiating tasks executions in a broad range of scenarios. Therefore, the employment of a task-set and scenario generator is strategical to many *unpredictable* scenarios with the support of a simulator.

In light of the data produced by the presented generator, it has proven to be a strategical tool fulfilling the need for a generator of task-sets and scenarios of Multi-Agent Systems.

As future work, we aim at: *(i)* improving the functionalities related to the performance analysis, *(ii)* enabling possible benchmarks by indexing and saving the generated scenarios, and *(iii)* providing customizable scenarios directly from the industrial domain.

References

1. Albanese, G., Calvaresi, D., Sernani, P., Dubosson, F., Dragoni, A.F., Schumacher, M.: Maxim-gprt: A simulator of local schedulers, negotiations, and communication for multi-agent systems in general-purpose and real-time scenarios. In: Proceedings of 16th International Conference on Practical Applications of Agents and Multi-Agent Systems (Jul 2018), https://doi.org/10.1007/978-3-319-94580-4_23
2. Bini, E., Buttazzo, G.C.: Measuring the performance of schedulability tests. Real-Time Systems 30(1-2), 129–154 (2005)
3. Buttazzo, G.: Hard real-time computing systems: predictable scheduling algorithms and applications, vol. 24. Springer Science & Business Media (2011)
4. Calvaresi, D., Appoggetti, K., Lustrissimini, L., Marinoni, M., Sernani, P., Dragoni, A.F., Schumacher, M.: Multi-agent systems negotiation protocols for cyber-physical systems: Results from a systematic literature review. In: Proceedings of 10th International conference on agents and artificial intelligence (2018)
5. Calvaresi, D., Cesarini, D., Sernani, P., Marinoni, M., Dragoni, A., Sturm, A.: Exploring the ambient assisted living domain: a systematic review. Journal of Ambient Intelligence and Humanized Computing pp. 1–19 (2016)
6. Calvaresi, D., Claudi, A., Dragoni, A., Yu, E., Accattoli, D., Sernani, P.: A goal-oriented requirements engineering approach for the ambient assisted living domain. In: Proceedings of the 7th International Conference on PErvasive Technologies Related to Assistive Environments. pp. 20:1–20:4. PETRA ’14 (2014), http://doi.acm.org/10.1145/2674396.2674416
7. Calvaresi, D., Marinoni, M., Lustrissimini, L., Appoggetti, K., Sernani, P., Dragoni, A.F., Schumacher, M., Buttazzo, G.: Local scheduling in multi-agent systems: getting ready for safety-critical scenarios. In: Proceedings of 15th European Conference on Multi-Agent Systems. Springer (Dec 2017)
8. Calvaresi, D., Schumacher, M., Marinoni, M., Hilfiker, R., Dragoni, A., Buttazzo, G.: Agent-based systems for telerehabilitation: strengths, limitations and future challenges. In: proceedings of X Workshop on Agents Applied in Health Care (2017)
9. Calvaresi, D., Marinoni, M., Sturm, A., Schumacher, M., Buttazzo, G.: The challenge of real-time multi-agent systems for enabling iot and cps. in proceedings of IEEE/WIC/ACM International Conference on Web Intelligence (WI’17) (Aug 2017)
10. Davis, R.I., Burns, A.: Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. In: Real-Time Systems Symposium, 2009, RTSS 2009. 30th IEEE. pp. 398–409. IEEE (2009)

11. De Bock, Y., Altmeyer, S., Huybrechts, T., Broeckhove, J., Hellinckx, P.: Task-set generator for schedulability analysis using the taclebench benchmark suite. SIGBED Rev. 15(1), 22–28 (Mar 2018), http://doi.acm.org/10.1145/3199610.3199613
12. Emberson, P., Stafford, R., Davis, R.I.: Techniques for the synthesis of multiprocessor tasksets. In: proceedings 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2010). pp. 6–11 (2010)
13. Horn, W.: Some simple scheduling algorithms. Naval Research Logistics (NRL) 21(1), 177–185 (1974)
14. Hsieh, F.: Modeling and control of holonic manufacturing systems based on extended contract net protocol. In: American Control Conference, 2002. Proceedings of the 2002. vol. 6, pp. 5037–5042. IEEE (2002)
15. McArthur, S.D.J., Davidson, E.M., Catterson, V.M., Dimeas, A.L., Hatziargyriou, N.D., Ponci, F., Funabashi, T.: Multi-agent systems for power engineering applications 2014;part i: Concepts, approaches, and technical challenges. IEEE Transactions on Power Systems 22(4), 1743–1752 (Nov 2007)
16. Stafford, R.: Random vectors with fixed sum. See http://www. mathworks. com/matlabcentral/fileexchange/9700 (2006)
17. Wu, J., Huang, Y.C.: Mcrtsim: A simulation tool for multi-core real-time systems. In: 2017 International Conference on Applied System Innovation (ICASI). pp. 461–464 (May 2017)