A Stopwatch Automata-Based Approach to Schedulability Analysis of Real-Time Systems with Support for Fault Tolerance Techniques

Alevtina Glonina[1][0000-0001-8716-4128] and Vasily Balashov[2][0000-0001-5211-805X]

Dept. of Computational Mathematics and Cybernetics, Lomonosov Moscow State University, 119991, Leninskie Gory, MSU, 1, Bldg. 52, Room 764, Moscow, Russia
[1] alevtina@lvk.cs.msu.su
[2] hbd@cs.msu.su

Abstract. During design of a fault tolerant real-time computer system (RTCS), it is necessary to guarantee that real-time constraints on system operation are met (i.e. all jobs are executed within deadlines) despite the increase of workload due to use of fault tolerance techniques (FTT). Checking these constraints can be reduced to schedulability analysis of workload which is modified with respect to the used set of FTT. In this paper, we propose an approach to such analysis, based on simulation of RTCS operation in order to produce a time diagram. Simulation model is automatically constructed from the general RTCS operation model (corresponding to a class of systems) and the RTCS configuration description. A generalization of stopwatch automata networks is chosen as a formal base for RTCS modeling, allowing to prove correctness properties for the models. The approach is implemented as an open-source tool system. Results of experimental evaluation are presented, demonstrating applicability of the approach to checking real-time constraints for solutions of the reliability allocation problem, extending the applicability of an existing evolutionary algorithm for this problem to a larger class of systems.

Keywords: real-time systems, schedulability, simulation, stopwatch automata.

1 Introduction

Modern complex technical systems (e.g. airplanes, power plants) are controlled by real-time computer systems (RTCS). A state-of-the-art RTCS is comprised by a set of standard computational modules connected by a data transfer network. The RTCS workload is a set of periodic tasks: for every period, an instance of a task (referred to as *job*) must be executed. RTCS must meet *real-time constraints*: all jobs must be executed within deadline intervals defined by tasks' periods. Data exchange between jobs of different tasks is performed by message passing. Data dependencies may exist between tasks with same period; in such case, the receiver job must wait for message arrival from the sender job. Every task is bound to a specific module. Execution of tasks bound to a module is controlled by the scheduler according to a scheduling scheme, typically a dynamic one. In this paper, by *RTCS configuration* we mean the set of computational resources (set of modules, number and types of CPU cores on the modules), set and characteristics of the workload (including task periods and worst-case execution times, durations of message transfers through the network), binding of tasks to modules.

An important property of an RTCS is dependability, meant as probability of failure-free operation during a specified time interval [1]. RTCS dependability can be improved by use of fault tolerance techniques (FTT). These techniques allow prevention of RTCS failure in case of fault in some of RTCS components, by means of software and/or hardware redundancy: adding spare computational modules, using several independently developed versions of application tasks (AT), accompanied by service tasks (ST) such as voters and acceptance tests.

Use of hardware redundancy increases the necessary amount of RTCS computational resources, and development of several AT versions increases the work effort for RTCS software development; this leads to increase of RTCS cost, and such increase must be controlled. Moreover, during RTCS development it is necessary to check that the real-time constraints are met, with consideration of the set of FTT used in the system. Thus, the reliability allocation problem (RAP) arises, stated as follows. *Given:* RTCS workload (set and characteristics of the "basic" AT versions, as well as of "alternate" AT versions for use with FTT; set and characteristics of messages to be transferred between AT), constraints on computational resources (number and types of available modules), costs of resources; available types of FTT, costs of FTT use; constraint on the total system cost; *Find:* set of FTT which maximizes *dependability* of the RTCS while meeting *real-time constraints* and *constraint on the total system cost*. Besides the choice of a specific FTT (or no FTT) for every AT, the RAP solution must include binding of all application and service tasks, including those added due to FTT use, to the RTCS modules. The RAP statement may vary, e.g. to minimize the cost while providing minimum required dependability, but the general idea remains the same.

Quite a lot of papers are dedicated to RAP research (see overviews in [2, 3]), but only few [4–7] propose methods for RAP solving that check real-time constraints on RTCS operation. Of these papers, only in [4] use of different FTT types is supported *and* data dependencies between tasks are taken in account. Checking of real-time constraints is performed in [4] with several simplifying assumptions – in particular, it is assumed that every module executes a single AT (or a group of tasks corresponding to the same "FTT-enhanced" AT); all ATs are supposed to have the same period.

These assumptions significantly limit the class of RTCS to which the evolutionary algorithm for RAP solving from [4] is applicable. In this paper, we propose an approach to checking real-time constraints for RTCS with FTT, which removes these assumptions and extends the class of RTCS to which the algorithm from [4] is applicable. The proposed method is based on [8] and performs schedulability analysis using simulation models with verified correctness.

2 Fault Tolerance Techniques

A number of FTT are used to provide RTCS dependability [9, 10], for instance N-version programming (NVP), N self-checking programming (NSCP), Recovery block (RB). Each of these FTT, applied to a specific AT, assumes execution of several independently developed versions of this AT (software redundancy, to tolerate software faults), as well as service tasks; tolerance to hardware faults is provided by use of additional computational modules (hardware redundancy). FTT of a specific TYPE, making a specific AT tolerant to X hardware faults and Y software faults, is denoted as TYPE/X/Y.

The RAP solving algorithm from [4] supports FTT NVP/0/1, NVP/1/1, and RB/1/1. In the present paper, we consider checking real-time constraints for these FTT, as well as for general FTT of NVP, RB and NSCP types, in which the original AT is replaced by a group of several AT versions and service tasks.

Use of NVP-type FTT for a specific AT involves N versions of this AT ($N \geq 3$, N is odd) and two ST: input data receiver and voter; the voter is combined with output data sender. Receiver ST receives all input messages addressed to the AT and dispatches them to versions of the AT. Every version of the AT is executed and sends the result to the voter, which selects the result produced by most of AT versions and sends this result to the output, as the original AT should do. In NVP/0/1, three AT versions are used, running on a single module. NVP/1/1 differs from NVP/0/1 by running AT versions on separate modules, i.e. three modules are used. Data dependency graph for AT versions and ST in case of NVP/0/1 or NVP/1/1 is shown in Fig. 1.

Use of NSCP-type FTT for a specific AT involves N versions of this AT ($N \geq 2$) and N+2 ST: input data receiver, N acceptance tests (possibly identical), and output data sender. Receiver ST receives all input messages addressed to the AT and dispatches them to versions of the AT. Every version of the AT is executed and sends the result to the corresponding acceptance test, which checks the result for correctness, and in case of success sends this result with pass mark to the sender (otherwise, a fail mark is sent); the sender takes the first received successful result and sends it to the output. With NSCP-type FTT, AT versions usually run concurrently on separate modules, or on one multi-core module. In NSCP/1/1, two modules and four AT versions are used, two versions for every module. Data dependency graph for AT versions and ST in case of NSCP/1/1 is shown in Fig. 1.

Use of RB-type FTT for a specific AT involves N versions of this AT ($N \geq 2$) and N+2 ST, with roles similar to those in NSCP. Main difference of RB from NSCP is that AT versions are executed sequentially (as a chain) and conditionally, so that the next AT version is started only in case the result of the previous version failed the acceptance test. There can be several such chains, each running on a separate module. In RB/1/1, two modules and two AT versions are used, both modules running the same AT chain. Data dependency graph for AT versions and ST in case of RB/1/1 is shown in Fig. 1. It should be noted that real-time constraints must be checked in the worst case, in which all tasks in all chains are executed.

A particular solution for the RAP problem described in Introduction corresponds to an RTCS configuration, which includes a set of modules with redundancy required by the set of FTT, a set of tasks with respect to the set of FTT (including service tasks and versions of application tasks, with necessary data dependencies); binding of tasks to modules is also specified. As real-time constraints are checked for the worst case, in which all versions of AT and all ST are executed (this is important for RB-type FTT), all information on FTT significant for checking these constraints is present in the RTCS configuration in form of computational resources and workload description. Thus, checking real-time constraints for a RAP solution amounts to checking these constraints for the corresponding RTCS configuration in which the used FTT are taken in account but are not explicitly present. The method for real-time constraints checking must perform schedulability analysis (as several tasks, possibly with different periods, can be bound to the same module), as well as account for data dependencies between tasks (both initially present between different AT, and emerging with use of FTT).

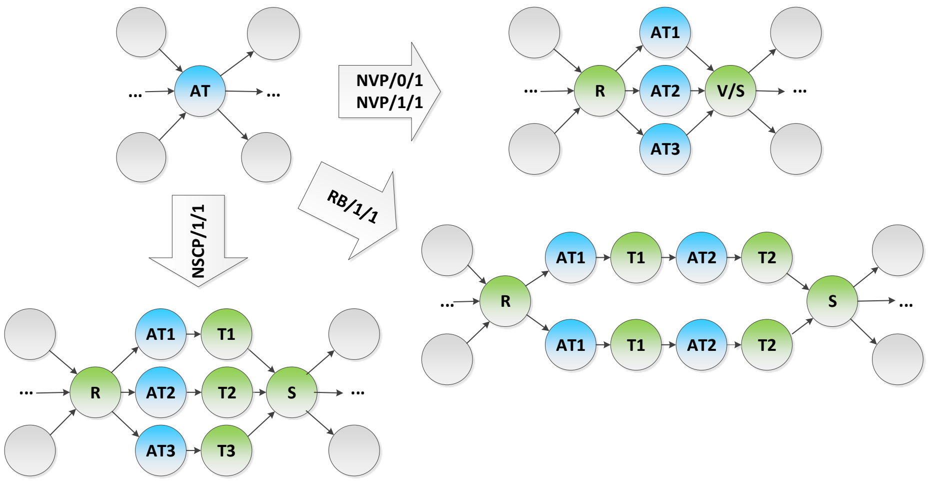

Fig. 1. Replacing an application task with a group of its versions and service tasks with introduction of FTT (AT: application task, V: voter, R: receiver, S: sender, T: acceptance test)

3 Related Work on Schedulability Analysis

In Section 2 we concluded that to check real-time constraints for an RTCS with FTT (e.g. obtained as a RAP solution), it is necessary to perform schedulability analysis for an RTCS configuration, the set of modules and the workload of which are specified with respect to the set of used FTT. During this analysis, both the scheduling scheme and data dependencies between tasks must be taken in account.

A group of analytical methods for schedulability analysis is known, based on classical approaches [11, 12]. These methods have a number of modern modifications [13–15]. A common disadvantage of these methods is significant over-estimation of tasks' worst case response times in case of task sets with data dependencies. This leads to false negative assessment of schedulability.

Another approach to schedulability analysis is construction and formal verification (e.g. by model checking) of RTCS model [16, 17]. This approach produces exact worst case response times of tasks, but severely limits the scale of analyzed systems (no more than several dozens of tasks), as its complexity is exponential by the number of tasks.

A more flexible and scalable approach to schedulability analysis for RTCS with a given configuration is construction of a time diagram for RTCS operation with job execution times equal to worst-case execution times (WCET) of tasks [18, 19]. Time diagram is constructed by running a simulation model of RTCS and contains intervals of job execution on CPU cores. With time diagram available, it is possible to check real-time constraints by direct comparison of job finish times to their deadlines.

RTCS simulation model must account for all aspects of RTCS operation significant for checking real-time constraints (e.g. task scheduling schemes, task preemption, data dependencies). Correctness of the model must be verified, as it is impossible during RTCS design to experimentally compare behavior of the model to behavior of actual RTCS. Since modern RTCS use different task scheduling schemes, integration of user-developed models of schedulers must be supported, in case correctness of these models was verified. Analysis of freely available RTCS simulation tools such as Cheddar, HSSim, MAST, DYANA, MASIW led the authors to the conclusion that none of them completely meets these requirements. Thus, it is reasonable to use the new method for checking real-time constraints for RTCS configurations, developed by the authors [8] along with supporting tools. According to this method, an RTCS simulation model is constructed from verified models of RTCS components (including scheduler models) and used to produce the time diagram of RTCS operation. In this paper, we present an adaptation of this method for use with algorithms for solving RAP.

4 The Approach to Checking Real-Time Constraints

4.1 Problem statement

Let us introduce informally the problem of checking real-time constraints for an RTCS configuration; formal statement of this problem is given in [8]. Checking these constraints for a RAP solution, i.e. for an RTCS with a specific set of FTT, is reduced to this problem.

Given the RTCS configuration: set of computational modules, workload (set of tasks, set of messages), binding of tasks to modules; for every module – number and types of CPU cores, as well as the scheduling scheme (e.g. fixed priority preemptive); for every task – its period, WCETs for different CPU core types, priority unique within a module; for every message – sender task, receiver task, maximum durations of transfer through module's memory and through the network.

Construct the time diagram of RTCS operation and *check* the time diagram for meeting the real-time constraints (these constraints are formulated below in terms of time diagram).

While checking the real-time constraints, RTCS operation is analyzed on the scheduling interval, duration of which equals to the least common multiple of all tasks' periods. For every task T_i with period p_i, on the scheduling interval of duration L a job set $W_i = \{w_{ij}\}_{j=1}^{L/p_i}$ is defined.

Event in the RTCS is a tuple $\langle EType, Src, t\rangle$, where $EType$ is the event type (EX – starting or resuming job execution; PR – job preemption; FIN – finishing job execution); $Src \in \bigcup_i W_i$ – event source job; $t \in \overline{1, L}$ – event time. The time is considered discrete, time unit can be chosen equal to RTCS CPU clock cycle.

Time diagram of RTCS operation is a set of events during the (real or simulated) RTCS operation. In practice during RTCS design it is assumed that execution time of every job is fixed and equals to WCET of corresponding task, message transfer times are fixed and equal to maximum possible ones, and all schedulers operate deterministically. Under these assumptions, a time diagram unambiguously corresponds to an RTCS configuration. RTCS simulation model is used for construction of this unambiguously defined time diagram.

Execution intervals for a job w_{ij} are intervals between the events $\langle w_{ij}, EX, t_1\rangle$ and $\langle w_{ij}, PR, t_2\rangle$, or between the events $\langle w_{ij}, EX, t_1\rangle$ and $\langle w_{ij}, FIN, t_2\rangle$, in the time diagram. Such intervals must not contain other events of types EX, PR, FIN with source job w_{ij}.

Real-time constraints are formulated in terms of time diagram as follows: for every job, total duration of its execution intervals equals duration of this job (i.e. WCET of corresponding task). If this condition is not met for some job, it indicates that execution of this job was terminated before completion, solely due to reaching the job's deadline. In other words, the job execution was not completed before its deadline. To check this condition for a specific RTCS configuration, it is necessary to construct time diagram of RTCS operation with this configuration. Since every task is bound to some module, it is correct to formulate the real-time constraints as such condition only if all CPU cores in a module have the same type, i.e. for every core of the same module a task has the same WCET. Real-time computer systems with modular structure, which are known to the authors, meet this assumption.

4.2 General Networks of Stopwatch Automata

Networks of stopwatch automata [20] were chosen as the basic formalism for modeling RTCS operation, aimed at time diagram construction via simulation. A stopwatch automaton (farther referred to as automaton) is a finite state machine with integer variables and special timer variables. A network of such automata is a set of co-operating automata, interacting via shared variables and synchronization through channels. *Parameterized automaton* is an automaton, expressions in which (e.g. for variable assignment, for transition conditions) include integer parameters, besides variables and numeric constants. By assigning values to all parameters, an *instance* of a parametrized automaton is created, which is a "normal" automaton.

To construct a general (i.e. abstract) model of RTCS operation, with support for modeling different (in particular, user-developed) schedulers, and to prove the correctness of this model, we propose a new abstraction level of automata networks: *general networks of stopwatch automata*. Main concepts of this abstraction level are introduced below.

Set of variables and synchronization actions, by which the automaton interacts with other automata in a network, is called *interface* of the automaton. *Base automata type* is described only by parameters and interface; no locations or transitions are specified for it. A parametrized automaton *implements* a base automata type if the interface and the set of parameters of the parametrized automaton match those of the base automata type. A set of base automata types is called *general automata network*, set of parametrized automata – *parametrized automata network*, network of instances of parametrized automata – *instance of automata network*.

These concepts can be explained by the example of scheduler modeling. A scheduler model, abstract from the specific scheduling scheme used in the module, is a base automata type. A model of a fixed priority preemptive scheduler is a parametrized automaton. A model of a scheduler operating in a specific module of an RTCS with a given configuration (including the set of tasks to schedule) is an instance of automaton, i.e. a "normal" automaton without parameters.

4.3 General Model of Real-Time System Operation

In [8] a general model for operation of a time-partitioned RTCS with modular structure was proposed, in form of a general automata network. In the present paper the RAP problem is considered for RTCS without time partitioning, thus an RTCS is modeled using following base automata types from [8]: base type `T`, modeling a task (with respect to its interaction with a scheduler); base type `TS`, modeling a scheduler operating in an RTCS module; base type `L`, modeling a virtual link (as means for message transfer with limited duration).

General model of RTCS operation specifies the interface for every base automata type comprising it. For instance, automata of base type `L` send signals through $\texttt{send}_{\texttt{ij}}$ and $\texttt{receive}_{\texttt{ij}}$ channels, where channel (*i,j*) corresponds to the *j*-th task of the *i*-th module. Structure of the general RTCS operation model is shown in Fig. 2; arrows represent directed channels, shared variables are not shown for brevity.

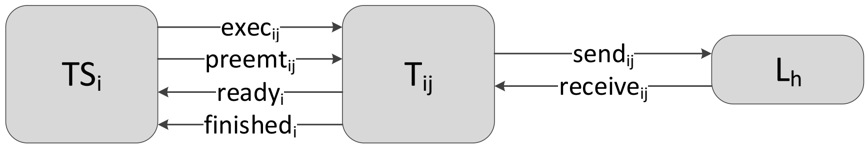

Fig. 2. Base automata types comprising the general model of RTCS operation

Only uniprocessor dynamic schedulers are modeled in [8], thus at present our capabilities for modeling RTCS with FTT are limited to systems in which every module contains only one CPU core (e.g. the central computer system of MC-21 airplane), or systems with multicore modules in which every task is bound to a specific CPU core.

4.4 Constructing a Simulation Model for Schedulability Analysis

Parameterized automata network implementing the general automata network described in Section 4.3, is a parameterized model of RTCS operation. Such model proposed in this paper contains following parameterized automata developed by the authors: task model, virtual link model, and models of three schedulers (fixed priority preemptive, fixed priority non-preemptive, preemptive EDF). Parameter values specify the set of computational resources, set and characteristics of the workload, binding of workload to computational resources.

According to values of parameters, defined by RTCS configuration, an instance of parametrized automata network (i.e. an instance of RTCS model) is constructed: channels and variables for automata interaction are created, for every RTCS component a parametrized automaton of corresponding type is created, and values are assigned to its parameters (e.g. period is set for a task model). An instance of RTCS model unambiguously corresponds to an RTCS configuration by construction. The result of running the RTCS model instance is the time diagram of RTCS operation. Running of the model, i.e. simulation of RTCS operation, is supported by stopwatch automata runtime library developed by the authors.

In [21] a set of correctness requirements was formulated for models of scheduler, task, virtual link, and RTCS as a whole. Using formal properties of general network of stopwatch automata, the authors proved that the developed parametrized models and the RTCS model constructed from them meet these requirements. E.g. for a fixed-priority preemptive scheduler model we proved that it always selects for execution a ready job of the task with highest priority.

5 Case Study

The proposed approach to checking real-time constraints for configurations of RTCS with FTT was implemented as an open-source tool system [22]. An experimental evaluation was performed to assess applicability of the approach to RTCS of realistic scale. We analyzed dependency of simulation time (RTCS model construction and execution time) on RTCS configuration scale and share of application tasks with FTT. The experiments were performed on AMD Ryzen 5 2600 3.4 GHz CPU, using one core.

In the first group or experiments, for an RTCS configuration with realistic scale (10 modules, around 150 ATs, more than 160 messages [18]), the share of AT with FTT was gradually increased (from 0% to 100% with 10% step); following FTT were used in equal numbers: NVP/0/1, NVP/1/1, RB/1/1. In the second group of experiments, for an RTCS configuration with 50% of AT using FTT, such quantitative characteristics were consistently increased as number of AT, messages and modules. Results of experiments are shown in Fig. 3.

RTCS considered in [18] is an avionics system with architecture described in patent RU2014115662A. According to this architecture, modules are connected by a switched network with support for virtual links, providing predictable message transfer times. The architecture is scalable, as it supports adding new computational modules grouped in crates by up to five; this allows using FTT that require hardware redundancy. Every crate has its own network switch module, which connects computational modules of the crate to each other and to other crates.

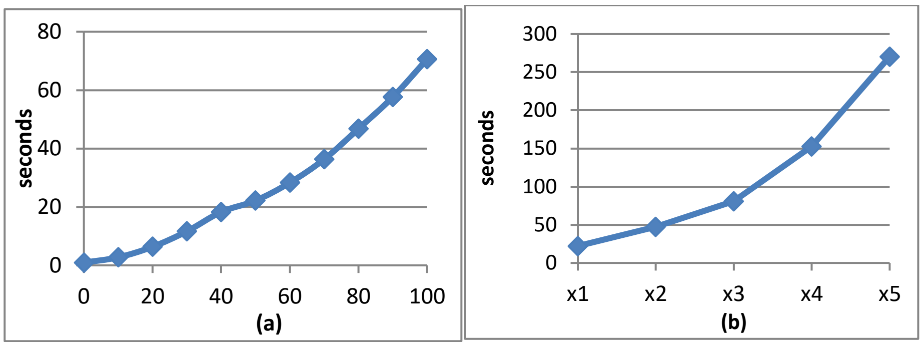

Fig. 3. Dependency of simulation time on share of AT with FTT (a), on configuration scale (b)

The experiments demonstrated quadratic dependency of simulation time on variation of selected characteristics of RTCS configuration. We can conclude that the developed tool system can be used with algorithms for RAP solving that enumerate hundreds of solutions (or thousands, if solutions are analyzed concurrently on a multicore CPU), including the evolutionary algorithm from [4].

6 Conclusion

In this paper, we proposed an approach to checking real-time constraints for configurations of RTCS with fault tolerance techniques, e.g. obtained as solutions of the reliability allocation problem. The approach is based on automatic construction of RTCS simulation model, and its running to obtain a time diagram of RTCS operation, taking in account the scheduling scheme(s) used in the RTCS modules. Choice of generalized stopwatch automata networks allowed formal proof of correctness for the RTCS models. The proposed approach significantly extends applicability of RAP solution algorithm from [4] and can be used with other RAP solution algorithms.

Future research is aimed at development of multiprocessor dynamic scheduler models and proof of their correctness, in order to extend the supported class of RTCS. Support for time-partitioned fault tolerant RTCS is another possible goal.

Acknowledgements. This work is partially supported by the Russian Foundation for Basic Research under grant №19-07-00614.

References

1. Laprie, J. C., Coste, A.: Dependability: A unifying concept for reliable computing. Proceedings of the 12th Fault Tolerant Computing Symposium, 18–21 (1982).
2. Kuo, W., Wan, R.: Recent advances in optimal reliability allocation. IEEE Trans. on Systems, Man, and Cybernetics - Part A: Systems and Humans 2(37), 143–156 (2007).
3. Coit, D. W., Zio, E.: The evolution of system reliability optimization. Reliability Engineering & System Safety 192 (2019).

4. Volkanov, D. Yu. et al: Simulation modeling based method for choosing an effective set of fault tolerance mechanisms for real-time avionics systems. Progress in Flight Dynamics, GNC, and Avionics 6, 487–500 (2013).
5. Jha, P. C. et al: Optimal component selection of COTS based software system under consensus recovery block scheme incorporating execution time. International Journal of Reliability, Quality and Safety Engineering 3(17), 209–222 (2010).
6. Attiya, G., Hamam, Y.: Reliability oriented task allocation in heterogeneous distributed computing systems. In: 9th International Symposium on Computers And Communications Proceedings, pp.68–73. IEEE, Alexandria, Egypt (2004).
7. Kang, Q., He, H., Wei. J.: An effective iterated greedy algorithm for reliability-oriented task allocation in distributed computing systems. Journal of Parallel and Distributed Computing 8(73), 1106–1115 (2013).
8. Glonina, A. B., Bahmurov, A. G.: Stopwatch automata-based model for efficient schedulability analysis of modular computer systems. In: Malyshkin, V. (ed) PACT 2017, LNCS, vol. 10421, pp. 289–300. Springer, Cham (2017).
9. Laprie J.C. et al: Definition and analysis of hardware and software-fault-tolerant architectures. IEEE Computer 23(7), 39–51 (1990).
10. Wattanapongsakorn, N., Levitan, S.: Reliability optimization models for fault-tolerant distributed systems. In: International Symposium on Product Quality and Integrity Proceedings, pp. 193–199. IEEE, Philadelphia, USA (2001).
11. Liu, C. L., Layland, J. W.: Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM 20(1), 46–61 (1973).
12. Audsley, N. et al: Applying new scheduling theory to static priority preemptive scheduling. Software Engineering Journal 8(5), 284–292 (1993).
13. Kim, J. et al.: A novel analytical method for worst case response time estimation of distributed embedded systems. In: 50th Annual Design Automation Conference Proceedings, pp. 1–10. ACM, New York, NY, USA (2017).
14. Palencia, J. C. et al: Response-time analysis in hierarchically-scheduled time-partitioned distributed systems. IEEE Trans. on Parallel and Distributed Systems 28(7), 2017–2030 (2017).
15. Amurrio, A.: Response-time analysis of multipath flows in hierarchically-scheduled time-partitioned distributed real-time systems. IEEE Access 8, 196700–196711 (2020).
16. Han, P. et al: Schedulability analysis of distributed multicore avionics systems with UPPAAL. Journal of Aerospace Information Systems 16(11), 473–499 (2019).
17. André, É. et al: Parametric schedulability analysis of a launcher flight control system under reactivity constraints. In: 19th International Conference on Application of Concurrency to System Design Proceedings, pp. 13–22. IEEE, Aachen, Germany (2010).
18. Balashov, V. V., Balakhanov, V. A., Kostenko, V. A.: Scheduling of computational tasks in switched network-based IMA systems. In: OPTI'2014 International Conference Proceedings, pp. 1001–1014. NTUA, Athens, Greece (2014).
19. Cheramy, M. et al: Simulation of real-time scheduling with various execution time models. In: 9th International Symposium on Industrial Embedded Systems, pp.1–4. IEEE, Pise, Italy (2014).
20. Cassez, F., Larsen, K.: The impressive power of stopwatches. In: Palamidessi, C. (ed) CONCUR 2000, LNCS, vol. 1877, pp. 138–152. Springer, Berlin (2000).
21. Glonina, A. B., Balashov, V. V.: On the correctness of real-time modular computer systems modeling with stopwatch automata networks. Automatic Control and Computer Sciences 7(52), 817–827 (2018).
22. https://github.com/AlevtinaGlonina/MCSSim, last accessed 2021/02/27.