Analysis of Real-Time Execution Models for Container-Based Control Applications

Moritz Walker(✉), Timur Tasci, Armin Lechler, and Alexander Verl

Institute for Control Engineering of Machine Tools and Manufacturing Units, University of Stuttgart, 70174 Stuttgart, Germany
moritz.walker@isw.uni-stuttgart.de

Abstract. Software-defined Manufacturing (SDM) aims to enhance the flexibility of production systems. Classical automation systems are not a suitable technological basis for SDM. While their hierarchical, rigid structures are increasingly being dissolved. Container-based virtualization, and modular software architectures, gain traction in automation systems. However, today's PLCs are not a perfect fit for virtualization, as the control program still is a monolithic piece of software. We analyze cyclic and event-based real-time scheduling models for modular PLCs. Furthermore, techniques for reconfiguration at runtime are developed based on the selected execution models.

Keywords: Containers · Real-Time · Virtualization · Automation Systems · Software-defined Manufacturing

1 Introduction

Today's manufacturing systems only support manual reconfiguration at the application level. However, the control software, e.g., the NC kernel, is fixed and bound to hardware. Thus, adapting core functionality is impossible or requires high manual effort. For this reason, the fixed programming of physical machines via PLC or NC code must be replaced by an adaptable software layer to enable Software-defined Manufacturing [1]. Software development for Programmable Logic Controllers (PLCs) is typically done monolithically. As monolithic control applications age, they become increasingly difficult to maintain. Components, e.g., function blocks, have low reusability and scalability [2]. Modular software architectures, such as Service-oriented Architectures (SOAs), address these drawbacks. Monolithic architectures are modularized into services, forming cohesive applications by loosely coupled interaction. Software containers are increasingly used to deploy modular architectures. We extend previous work [3,4] towards a modular control platform, which implements a Microservices architecture. Specifically, we extend the container-based control system by event-based and cyclic execution, i.e. orchestration, models that meet the requirements of modular software architectures and the real-time requirements of control systems.

N. Kiefl et al. (Eds.): SCAP 2022, ARENA2036, pp. 14–24, 2023.
https://doi.org/10.1007/978-3-031-27933-1_2

2 Related Work

Cucinotta et al. [5] present an SOA in which real-time communication bypasses the Simple Object Access Protocol (SOAP) stack and directly uses UDP/IP. Service invocations are scheduled as sporadic tasks using Earliest Deadline First (EDF). Dai et al. [6] present an industrial SOA based on IEC 61499, within which function blocks offer functionalities as services. The communication takes place via SOAP and TCP/IP. Tsai et al. [7] present RTSOA, an SOA extended by soft real-time capabilities. A concept for a container-based, real-time automation platform is defined in [8]. The scheduling algorithm is Fixed Priority Preemptive Scheduling (FPS). While the other publications do not provide temporal synchronization of tasks across a compute node's boundaries, Telschig's container-based architecture [9] does so by introducing globally valid time slots for message exchange and task execution. Furthermore, not all architectures support a guaranteed execution order of tasks and the deployment or update of components at run-time. EDF, commonly used on single-core and multicore systems, is also suitable for scheduling applications whose tasks have dependencies that can be modeled as DAGs. A DAG is transformed into deadlines and activation times of the subtasks for this purpose [10,11]. The methods known for single-processor and multiprocessor systems are extended by Rivas et al. [12] for distributed systems. Saifullah et al. [10] present a scheduling method for parallel real-time execution of multiple periodic DAGs on a multicore processor. An algorithm decomposes one or more DAGs into sequential tasks by assigning activation offsets and deadlines to individual tasks. Global EDF is used as the scheduling algorithm. Jiang et al. [11] present a similar decomposition algorithm for DAG tasks. Peng et al. [13] present methods for the FPS and EDF of DAGs, where no decomposition is necessary.

3 Analysis of Scheduling Methods

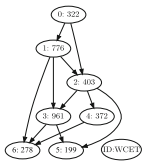
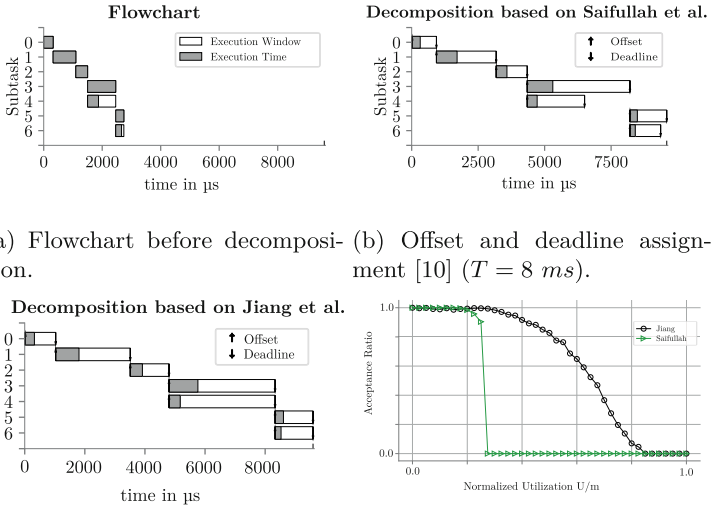


Fig. 1. An exemplary DAG task consists of subtasks with worst-case execution times (WCETs). Every subtask is deployed as a container.

Cyclic Scheduling of Dependent Tasks: Some use cases, such as sensor fusion, benefit from executing tasks in a specified order because this can reduce the end-to-end latency. Therefore, we compare two [10,11] principally applicable methods for cyclic scheduling of task sets modeled as DAGs. Both methods rely



(a) Flowchart before decomposition.

(b) Offset and deadline assignment [10] ($T = 8\ ms$).

(c) Offset and deadline assignment [11] ($T = 10\ ms$).

(d) Number of schedulable task sets on a m-core processor.

Fig. 2. Comparison of Jiang's [11] and Saifullah's [10] decomposition algorithms.

on the decomposition of DAG tasks, i.e., the assignment of offsets and deadlines to subtasks. Furthermore, both methods use EDF at runtime. Figure 1 depicts an exemplary task modeled as a DAG. In the following, a DAG is called a task and its components are called subtasks. Arrows symbolize precedence constraints. The methods considered in this work are the algorithms of Saifullah et al. [10] and Jiang et al. [11]. Figure 2 shows the decomposition of the DAG in Fig. 1 under Saifullah's (Fig. 2b) and Jiang's (Fig. 2c) methods. Vertical markers at the end of the abscissa symbolize specified deadlines. Both methods are compared as follows. Random task sets with varying parameters are generated, and the numbers of schedulable task sets are compared. The task set generation follows the following scheme based on [10,14]:

1. Determination of the parameters: The connection probability $p \in (0, 1)$ specifies the probability of precedence constraints. n is the number of subtasks per task, U_{set} is the desired processor utilization of the entire task set. $\beta < 1$ is a factor used to influence individual task utilization.
2. The generation of n random task is done using the $G(n, p)$ method according to Ernős-Rényi.
3. The WCET $C_{i,j}$ of a subtask j in task i is randomly chosen from the interval $[100\ \mu s, 1000\ \mu s]$. The length of the critical path L_i, i.e., the sum over the WCETs of subtasks on the longest path in the DAG, and the workload W_i, i.e., the sum over the WCETs of all subtasks in the DAG, are calculated.

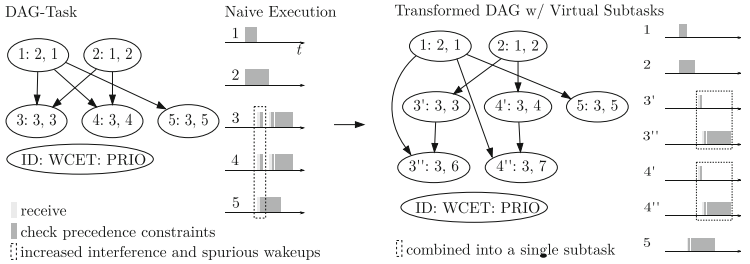


Fig. 3. DAG-transformation to minimize intra-task-interference under G-FPS.

4. The period of a task T_i is chosen randomly from the interval $[L_i, W_i/\beta)$. Thus, the processor utilization U_i lies in the interval $[\beta, W_i/L_i]$. The deadline is implicitly set to be identical to the period.
5. Step 2 to 4 is repeated until $\sum_{i=1}^{n} U_i > U_{set}$ holds, and the period of the last task is chosen so that exactly the desired total utilization is reached.

1000 random task sets per utilization were generated using the parameters $p \in [0.01; 0.2]$, $n \in [20; 100]$, $\beta = 0.1$, and $C_{i,j} \in [100\ \mu s; 1000\ \mu s]$. Figure 2d shows the acceptance rates for the randomly generated task sets with different utilizations. Since Jiang's decomposition strategy can decompose significantly more task sets in a schedulable manner, it is applied if the application requires cyclic scheduling considering the execution order.

Event-Based Scheduling of Dependent Tasks. For event-based scheduling of DAG tasks, global EDF (G-EDF) [15] and global FPS (G-FPS) [14] are suitable algorithms. Using global FPS, the subtasks within a task are assigned priorities according to their topological order. Different tasks (DAGs) are assigned priorities according to Deadline Monotonic Scheduling (DMS) and thus, in the case of implicit deadlines, according to Rate Monotonic Scheduling (RMS). Due to the implementation-specific details of the SCHED DEADLINE scheduler, schedulability cannot be tested according to Melani et al. [18]. The deadline of a thread is relative to its activation time on Linux. For the schedulability test, according to Melani et al., and similar tests, its deadline must be relative to the activation time of the source subtask. Thus, G-FPS is used for the event-based scheduling of container-based DAG tasks. Pathan's test [14] is applied to check the schedulability. As discussed in [3,4], the container-based control system uses the socket API for inter-service communication. The `select`, `poll`, or `epoll` syscalls can be used to wake up tasks, when a message is delivered. The ideal implementation of the execution model would require a syscall that blocks until a task has received messages from all of its predecessors. Such syscall is not available on Linux, which leads to unnecessary context switches and increased intra-task interference between the subtasks of a DAG, as illustrated on the left in Fig. 3. Our approach to minimizing the intra-task interference is described in the following and shown in Fig. 3. Subtasks that have more than one predecessor are decomposed into sequential virtual subtasks. The number of virtual

subtasks corresponds to the number of predecessors. For a subtask $v_{i,j}$ of a DAG task Π_i with predecessors $pred_{i,j} = \{pred_{i,j,1}, pred_{i,j,2}, ...\}$, the decomposition into sequential virtual subtasks is done in four steps:

1. First, the response time analysis (RTA) is performed using the original task set and the test of Pathan et al. [14]. Here, the worst-case response time $R_{i,j}$ is calculated for each subtask $v_{i,j}$.
2. The subtask $v_{i,j}$ is divided according to the cardinality of the predecessors into $n_{i,j}^{virt} = |pred_{i,j}|$ virtual sequential subtasks $v_{i,j}^k$. The first $k = 1, ..., n_{i,j}^{virt} - 1$ subtasks are assigned the WCET $C_{i,j}^k = \tau_{recv} + \tau_{proc}$, where τ_{proc} corresponds with the message processing time and τ_{recv} corresponds with the delay, needed to fetch the message from the messaging system. The last virtual subtask is assigned the WCET $C_{i,j}^{|pred_{i,j}|} = \tau_{recv} + \tau_{proc} + \tau_{task} + n_{i,j}^{sub}\tau_{send}$. τ_{send} denotes the time needed to send a message.
3. The predecessors $pred_{i,j}$ of the subtask are sorted in descending order according to their response times $R_{i,j}$. The virtual subtasks are ordered in ascending order for $k = 2, ..., n_{i,j}^{virt}$ and the predecessors $pred_{i,j,k}$ and $v_{i,j}^{k-1}$ are assigned to the subtask. The first virtual subtask receives only $pred_{i,j,1}$ as predecessor.
4. Finally, we evaluate schedulability by applying Pathan's RTA [14] to the transformed DAG tasks.

If a DAG extends across the boundaries of a system, schedulability under global FPS is evaluated by the RTA of Peng et al. [13]. This method considers that subtasks not executed on the same processor cores cannot interfere with each other.

4 Execution Models for the Container-Based PLC

4.1 Cyclic Execution Model

Scheduling: To support independent and DAG applications simultaneously and guarantee schedulability for high processor loads, the cyclic execution model uses the decomposition method of Jiang et al. [11] and global EDF. This execution model is suitable for implementing cyclic DAG tasks with implicit or constrained deadlines ($D_i \leq T_i$). The interaction between the communication system and application subtasks follows the synchronous interaction pattern. Thus, the execution time of a subtask results in

$$\tau_{total} = n_{pub}\tau_{recv} + \tau_{task} + n_{sub}\tau_{send} + \tau_{aug}. \tag{1}$$

The task's execution time is augmented with the latency τ_{aug}, which is the time needed until all messages are transmitted to the successors. n_{pub} is the number of predecessors and n_{sub} is the number of successors. In the schedulability test, only the actual execution time of the subtasks is considered.

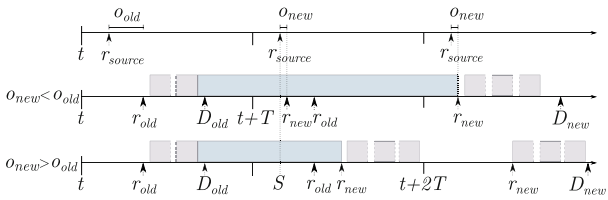


Fig. 4. Concept for reconfiguration by reassigning deadlines and offsets.

Runtime Reconfiguration: The reconfiguration, that is, adding a subtask v_{i,n_i+1} to a DAG task Π_i with subtasks $\{v_{i,1}, ..., v_{i,n_i}\}$ and period T_i, is done in three steps. v_{source} is any source subtask of Π_i, e.g. $v_{i,1}$. The update strategy is exemplified for one subtask $v_{i,j}$, but is applied simultaneously to all subtasks of the DAG. To simplify the notation, $T = T_i$ is the period of DAG task Π_i and $r^k_{source} = r^k_{i,1}$ is the request time of the k-th instance of the source subtask. The relative offset $o = o_{i,j}$ of the subtask $v_{i,j}$ refers to r^k_{source}, which is increased by T for each invocation of the DAG task: $r^k_{source} = r^{k-1}_{source} + T$. The request time r^k of the k-th instance of subtask $v_{i,j}$ is $r^k = r^k_{source} + o$. The deadline D of subtask $v_{i,j}$ is relative to its request time. For r^k_{source} and r^k, the notations r_{source} and r are used. First, new deadlines and offsets are assigned to subtasks by the decomposition procedure. The old and new offsets and deadlines of subtask $v_{i,j}$, are denoted o_{old} and o_{new}, and D_{old} and D_{new}. The next step is to notify the subtasks about their new deadlines and offsets. A message is sent to each subtask, containing the new deadlines and offsets and a global synchronization point. The global synchronization point is derived from the request time r^k_{source} of the source subtask: $S = r^k_{source} + xT$. $x \in \mathbb{N}$ can be freely chosen. While, e.g., Xenomai natively supports the allocation of offsets between threads, on Linux, this can only be done using a timed sleep. If the request time of a subtask is shifted to the left, that is ($o_{new} < o_{old}$), the corresponding subtask would have to be executed a second time within one period. Since the deadline of the subtask may already be exceeded at this point, and the Constant Bandwidth Server (CBS) may have no remaining bandwidth, the subtask might be throttled. For this reason, the reallocation follows the strategy shown in Fig. 4. If $o_{new} < o_{old}$, the subtasks are paused until $S + T + o_{new}$, and the new deadline is assigned. If $o_{new} > o_{old}$, the subtasks are paused until $S + o_{new}$ and D_{new} is assigned. The third step includes the deployment of the new subtask. A consistent state transfer is necessary if a subtask is updated, i.e., replaced. First, the schedulability test is used to check whether the new subtask can be executed parallel, i.e., with identical offsets and deadlines, to the subtask to be replaced. If this is the case, the subtask starts. Otherwise, the deployment follows the procedure described above. Next, the necessary communication channels are initialized. The new subtask does not process incoming messages and leaves them in the message queue. A time-stamped image of the state of the internal variables of the component to be replaced is successively transmitted to the new subtask. Once the state is completely transmitted, the actual state is reconstructed using the messages

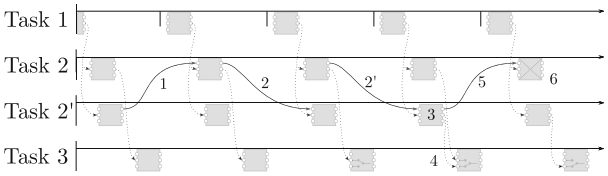


Fig. 5. Example of an update to replace a stateful subtask.

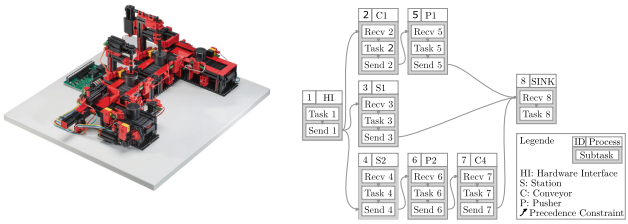
in the message queue. The new subtask starts executing its program logic and transmitting the output messages to its successors. The successors always apply the messages of the new subtask during the update process, as far as these are available. Finally, the original subtask terminates. Figure 5 illustrates the principle flow of a stateful reconfiguration. The DAG task consists of the source subtask task 1, task 2, and task 3, which are executed in this order. The stateful task 2 is replaced by task 2' at runtime. The reassignment of offsets and deadlines has already been done, and the required communication links have been established. (1) Task 2' requests the transfer of task 2's internal state and starts receiving messages from task 1. (2) The state's transmission begins, which extends over two cycles. (3) State reconstruction is conducted based on the messages in the message queue and the transmitted state. In this example, only one cycle is needed for this. Task 2' begins with the execution of the program. (4) Task 3 receives messages from tasks 2 and 2'. Messages from task 2 are dropped, and messages from task 2' are applied. Furthermore, task 2 is notified that the deployment of task 2' has been completed (5). (6) Task 2 terminates.

4.2 Event-Based Execution Model

Based on the results of the previous sections, the event-based execution model of the container-based control system is presented. The assignment of priorities and the execution at runtime is performed according to the procedures explained in Sect. 3. The developed model ensures that only one event-based activation is necessary for each subtask since all other precedence messages have already been delivered and can be processed without interrupting the subtask. To check schedulability, the RTA of Pathan et al. [14] and to check schedulability on a distributed system, the RTA by Peng et al. [13] are used. In event-based systems, it is unpredictable when an event may occur. For this reason, reconfiguration is done either offline or without consistent state transmission.

5 Validation of a Sample Use-Case

The real-time performance of the execution models was evaluated based on a sample application. A production line model is controlled by a single DAG task with eight services and seven deployment units (see Fig. 6). The WCETs of the PLC subtasks are shown in Table 1.



(a) Model production line[1] (b) Corresponding DAG task

Fig. 6. Production line model with two processing stations.

Table 1. WCETs of the subtasks of the exemplary application for validation

Task	T1	T2	T3	T4	T5	T6	T7	T8
WCET [µs]	476	676	698	777	794	697	683	725

Validation of the Event-Based Execution Model: To validate the event-based execution model, two instances of the DAG task Π_1 and Π_2 were executed at a rate of $T_1 = 5$ ms and $T_2 = 10$ ms. The temporal behavior was recorded over a time span of six hours on a Raspberry Pi 3 equipped with PREEMPT_RT-Patch and Linux version 5.2.21. Figure 7 depicts the measured execution of Π_1 on the test system compared to the previously calculated response times of the subtasks. Only tasks 1, 2 and 3 exceeded the calculated response times. However, this was caused by WCET overruns and the jitter of the test system. If the scheduling method is to be used for safety-critical applications, the WCET must therefore be estimated sufficiently pessimistically. The second DAG task Π_2 also did not exceed the calculated response time.

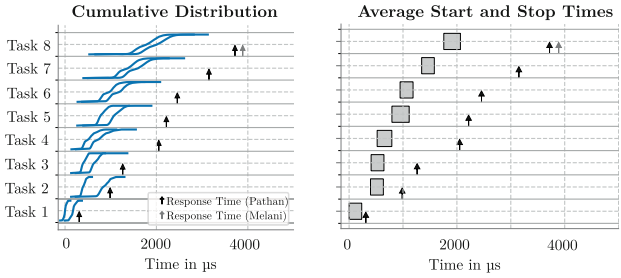


Fig. 7. Comparison between the calculated response times of DAG Π_1 with $T_1 = 5$ ms and the actual system behavior.

Validation of the Cyclic Execution Model: The cyclic execution model was also validated by running the application consisting of Π_1 and Π_2 ($T_1 = T_2 = 8$ ms) with implicit deadlines for six hours on the test system. Figure 8 shows the resulting artificial deadlines and measured behavior of Π_1 and Π_2. The precedence constraints were not violated at any time. The cyclic and event-based dependent execution models' evaluation shows that both can be employed under real-time requirements.

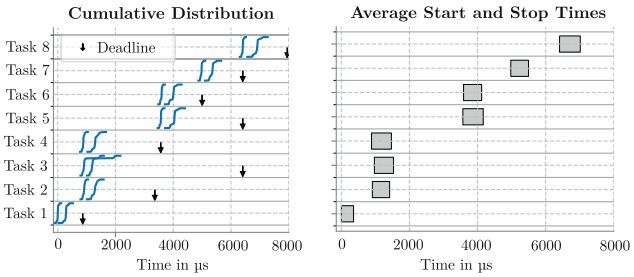


Fig. 8. Comparison of the calculated response times of DAG Π_1 with $T_1 = 8$ ms and the actual system behavior.

6 Conclusion and Future Work

In this work, we developed concepts for event-based and cyclic execution models. Both execution models support applications with independent tasks and applications, where the tasks' execution must occur in an explicitly specified order. This order is modeled as DAG. The execution models allow the development, deployment, and dynamic reconfiguration of distributed control applications. As part of future work, we plan to extend the used socket-based messaging system to support Time Sensitive Networking.

Acknowledgment. This work was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - Project-ID 420528256 and by the German Federal Ministry of Education and Research (BMBF) within the research campus ARENA2036 (Active Research Environment for the Next generation of Automobiles) (funding number 02P18Q620).

References

1. Lechler, A., Verl, A.: Software defined manufacturing extends cloud-based control. In: Proceedings of the ASME 12th International Manufacturing Science and Engineering Conference - 2017, New York, NY. The American Society of Mechanical Engineers (2017)
2. Mazzara, M., Meyer, B.: Present and Ulterior Software Engineering. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67425-4
3. Tasci, T., Melcher, J., Verl, A.: A container-based architecture for real-time control applications. In: Conference Proceedings ICE/IEEE ITMC, Piscataway, NJ, pp. 1–9. IEEE (2018)
4. Tasci, T., Fischer, M., Lechler, A., Verl, A.: Predictable and real-time message-based communication in the context of control technology. In: Weißgraeber, P., Heieck, F., Ackermann, C. (eds.) Advances in Automotive Production Technology – Theory and Application. A, pp. 264–271. Springer, Heidelberg (2021). https://doi.org/10.1007/978-3-662-62962-8_31
5. Cucinotta, T., Mancina, A.: A real-time service-oriented architecture for industrial automation. IEEE Trans. Ind. Inform. **5**(3), 267–277 (2009)
6. Dai, W., Vyatkin, V., Christensen, J.H., Dubinin, V.N.: Bridging service-oriented architecture and IEC 61499 for flexibility and interoperability. IEEE Trans. Ind. Inform. **11**(3), 771–781 (2015)
7. Tsai, W.T., Lee, Y.-H., Cao, Z., Chen, Y., Xiao, B.: RTSOA: real-time service-oriented architecture. In: Second IEEE International Workshop on Service-Oriented System Engineering 2006, Los Alamitos, California, pp. 49–56. IEEE Computer Society (2006)
8. RT-CASEs: Container-Based Virtualization for Temporally Separated Mixed-Criticality Task Sets. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik GmbH, Wadern/Saarbruecken, Germany (2019)
9. Telschig, K., Schonberger, A., Knapp, A.: A real-time container architecture for dependable distributed embedded applications, pp. 1367–1374. IEEE (2018)
10. Saifullah, A., Ferry, D., Li, J., Agrawal, K., Chenyang, L., Gill, C.: Parallel real-time scheduling of DAGs. IEEE Trans. Parallel Distrib. Syst. **25**(12), 3242–3252 (2014)
11. Jiang, X., Long, X., Guan, N., Wan, H.: On the decomposition-based global EDF scheduling of parallel real-time tasks. In: RTSS 2016, Piscataway, NJ, pp. 237–246. IEEE (2016)
12. Rivas, J.M., Javier Gutierrez, J., Carlos Palencia, J., Harbour, M.G.: Deadline assignment in EDF schedulers for real-time distributed systems. IEEE Trans. Parallel Distrib. Syst. **26**(10), 2671–2684 (2015)
13. Peng, X., Han, M., Deng, Q.: Response time analysis of typed DAG tasks for G-FP scheduling. In: Guan, N., Katoen, J.-P., Sun, J. (eds.) SETTA 2019. LNCS, vol. 11951, pp. 56–71. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-35540-1_4
14. Pathan, R., Voudouris, P., Stenstrom, P.: Scheduling parallel real-time recurrent tasks on multicore platforms. IEEE Trans. Parallel Distrib. Syst. **29**(4), 915–928 (2018)
15. Qamhieh, M., Fauberteau, F., George, L., Midonnet, S.: Global EDF scheduling of directed acyclic graphs on multiprocessor systems. In: Proceedings of the 21st International Conference on Real-Time Networks and Systems, RTNS 2013, pp. 287–296. Association for Computing Machinery, New York (2013)