# A Framework for Scheduling Real-Time Systems

Zhuo Cheng*, Haitao Zhang†, Yasuo Tan*, and Yuto Lim*
*School of Information Science, Japan Advanced Institute of Science and Technology, Japan
{chengzhuo, ytan, ylim}@jaist.ac.jp
†School of Information Science and Engineering, Lanzhou University, China
htzhang@lzu.edu.cn

***Abstract*—Real-time system is playing an important role in our society. For such a system, sensitivity to timing is the central feature of system behaviors, which means tasks in the systems are required to be completed before their deadlines. To guarantee this requirement, the design of scheduling is crucial. In this paper, based on *satisfiability modulo theories (SMT)*, we provide a framework to design scheduling for real-time systems. In the framework, the problem of scheduling is treated as a satisfiability problem. The key work is to formalize the satisfiability problem using first-order language. After the formalization, a SMT solver (e.g., Z3, Yices) is employed to solver such a satisfiability problem. An optimal schedule can be generated based on a solution model returned by the SMT solver. To demonstrate the practicality of the framework, we give design guidelines for real-time systems with multiprocessor. Through the demonstration, the framework is found flexible and sufficiently general to apply to different kinds of real-time systems. To the best of our knowledge, it is the first time that systematically introducing SMT to solve a series problems covering a wide range in real-time scheduling domain.**

***Keywords*—real-time scheduling, SMT, multiprocessor, satisfiability problem**

## I. Introduction

Real-time system is playing an important role in our society. For example, chemical and nuclear plant control, space missions, flight control, telecommunications, and multimedia systems are all real-time systems [1]. In such a system, sensitivity to timing is the central feature of system behaviors, which means, tasks in the system are required to be completed before their deadlines. To provide such guarantee, the design of scheduling is crucial.

The research on real-time scheduling has lasted for decades, but still lots of challenges remain [5]. For example, limited task models for multiprocessor systems, limited policies for access to shared resources, ineffective schedulability tests, limited scheduling methods. In this paper, we try to address the challenge *limited scheduling methods*.

For designing scheduling method, many research has contributed to this area [8, 9, 10, 11]. But one important problem is that all the proposed methods are specified on either a specific system architecture (e.g., uniprocessor) or specific scheduling target (e.g., make all task meet deadline). Usually, it is quite difficult, even impossible, to adapt one scheduling method to another application scenario. This becomes a main obstacle for designing scheduling for a new application system and results in a high design cost. In this paper, we try to solve the problem by proposing a framework to design scheduling for real-time systems. The main contributions of this paper are as follows.

*i) We propose a scheduling framework based on satisfiability modulo theories (SMT).* In this framework, the problem of scheduling is treated as a satisfiability problem. The key work is to formalize the satisfiability problem using first-order language. We use a *sat model* to represent the formalized problem. This sat model is a set of first-order logic formulas (within linear arithmetic in the formulas) which express all the scheduling constraints that a desired optimum schedule should satisfy. After the sat model is constructed, a SMT solver (e.g., Z3 [6], Yices [7]) is employed to solve the formalized problem. An optimal schedule can be generated based on a *solution model* returned by the SMT solver. The correctness of this method and the optimality of the generated schedule are straightforward.

*ii) The proposed scheduling framework is flexible.* In the SMT-based scheduling method, we define the scheduling constraints as system constraints and target constraints. It means if we want to design scheduling to achieve other objectives, only the target constraint needs to be modified. Or, if we want to achieve the same scheduling objective for another real-time system with different system architecture, only the system constraints need to be modified.

*iii) We give practical design guidelines for scheduling multiprocessor systems.* These design guidelines are for systems with multiprocessor, and of course, they are also applicable for system with uniprocessor, as scheduling uniprocessor systems is a sub problem of scheduling multiprocessor systems. The model for the multiprocessor system is defined in a very general way, and in the design guidelines, we have considered systems with mixed-criticality (hard and soft) real-time functions, task dependency relation, task migration cost, heterogeneous processors (processors with different processing speed and architectures), heterogeneous network channels (network channels with different data transfer speed and supporting different network protocols). All these efforts make the framework practicable and sufficiently general to apply to different kinds of real-time systems and different scheduling targets, which can benefit system designers to efficiently design scheduling.

The remainder of this paper is organized as follows. In Section II, we present scheduling framework which is based on satisfiability modulo theories. The system model is denoted in Section III. We give the design guidelines for system constraints in Section IV, while Section V gives the design guidelines for target constraints. Related work are summarized in Section VI. Section VII concludes the paper.

This paper is submitted as a Regular Research Paper. Please contact Zhuo Cheng for any inquiry.

TABLE I. SYMBOLS AND DEFINITIONS

| Symbol | Definition |
|---|---|
| $t$ | system time instant |
| $\delta$ | network precision |
| $\mathcal{F}$ | set of functions of a real-time system |
| $\mathcal{FH} \subseteq \mathcal{F}$ | set of functions with hard deadlines |
| $\mathcal{FS} \subseteq \mathcal{F}$ | set of functions with soft deadlines |
| $F_i \in \mathcal{F}$ | function of a real-time system, $i$ is the index of the function |
| $r_i$ | triggered time instant of function $F_i$ |
| $d_i$ | deadline of function $F_i$ |
| $v_i$ | obtained value by completing function $F_i$ before deadline |
| $\mathcal{T}$ | set of all the tasks in the real-time systems |
| $T_i \subseteq \mathcal{T}$ | set of tasks corresponding to function $F_i$ |
| $\tau_j \in \mathcal{T}$ | task, $j$ is index of the task |
| $c_j$ | computation cost of $\tau_j$ |
| $m_j$ | migration cost of $\tau_j$ from a processor to another one |
| $\tau s_i$ | start task of task poset $(T_i, \prec)$ |
| $\tau e_i$ | end task of task poset $(T_i, \prec)$ |
| $\mathcal{P}$ | set of processors |
| $p_a \in \mathcal{P}$ | processor, where $a$ is the index of the processor |
| $ps_a$ | speed of processor $p_a$ |
| $TS_a \subseteq \mathcal{T}$ | task set that can be completed by processor $p_a$ |
| $TS_{a\to b} \subseteq \mathcal{T}$ | task set that can migrate on network channel $n_{a\to b}$ |
| $\mathcal{N} \subseteq \mathcal{P} \times \mathcal{P}$ | set of network channels |
| $n_{a\to b} \in \mathcal{N}$ | network channel from processor $p_a$ to $p_b$ |
| $ns_{a\to b}$ | speed of $n_{a\to b}$ |

## II. THE SMT-BASED SCHEDULING FRAMEWORK

### A. Satisfiability Modulo Theories (SMT)

Satisfiability modulo theories checks the satisfiability of logic formulas in first-order formulation with regard to certain background theories like linear integer arithmetic or bit-vectors [2]. A first-order logic formula uses variables as well as quantifiers, functional and predicate symbols, and logic operators [3]. A formula $F$ is *satisfiable*, if there is an interpretation that makes $F$ `true`. For example, formula $\exists a, b \in \mathbb{R}, (b > a + 1.0) \wedge (b < a + 1.1)$, where $\mathbb{R}$ is real number set, is satisfiable, as there is an interpretation, $a \mapsto -1.05, b \mapsto 0$, that makes $F$ `true`. On the contrast, a formula $F$ is *unsatisfiable*, if there does not exist an interpretation that makes $F$ `true`. For example, if we define $\exists a, b \in \mathbb{Z}$, where $\mathbb{Z}$ is integer set, the formula $(b > a+1.0) \wedge (b < a+1.1)$ will be unsatisfiable.

For a satisfiability problem that has been formalized by first-order logic formulas, a SMT solver (e.g., Z3, Yices) can be employed to solver such a problem. If all the logic formulas are satisfiable, SMT solver returns the results `sat` and a *solution model* which contains an interpretation for all the variables defined in the formulas that makes the formulas `true`. For the case $\exists a, b \in \mathbb{R}$, the model is: $a \mapsto -1.05, b \mapsto 0$. If there is an unsatisfiable logic formula, SMT solver returns the results `unsat` with an empty model, for the case $\exists a, b \in \mathbb{Z}$.

### B. The Scheduling Framework

The framework of the SMT-based scheduling is illustrated in Fig. 1. In a real-time system, a schedule (execution order of tasks) is generated by a scheduler. The problem of scheduling can be treated as a satisfiability problem.

In order to use SMT to solve this satisfiability problem, the key work is to formalize the problem using first-order language. We use a *sat model* to represent the formalized problem. This sat model is the set of first-order logic formulas (within linear arithmetic in the formulas) which expresses all the constraints that the desired schedule should satisfy. There are two kinds of constraints: *system constraints* and *target constraints*. System constraints are based on the specific system. For example, if two tasks run on a processor, a schedule should make sure that the execution of these two tasks cannot have overlap in time domain. Target constraint is based on the scheduling target. For example, under normal workload condition, the desired schedule should make all the functions meet their deadlines (completed before deadlines).

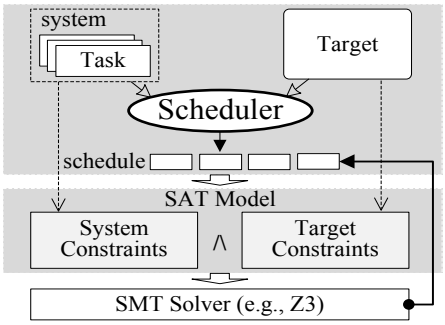


Fig. 1. The framework for scheduling real-time system based on SMT

After the sat model is constructed, it can be inputted into a SMT solver (e.g., Z3). A *solution model* will be returned by the SMT solver. This solution model gives an interpretation for all the variables defined in the sat model, and under the interpretation, all the logic formulas in the sat model are evaluated as `true`. It means the satisfiability problem represented by the sat model is solved, and based on this interpretation, the desired schedule can be generated.

## III. SYSTEM MODEL

### A. Function Set

Function set define the functions that can be achieved by a real-time system. Let $\mathcal{F} = \{F_1, F_2, \ldots, F_n\}$ denote the function set. Each function $F_i \in \mathcal{F}$ is achieved by a corresponding series tasks, represent as *poset* $(T_i, \prec)$, $T_i \neq \emptyset$ denotes the set of the corresponding task, and $\prec$ denotes the dependency relation of tasks in $T_i$ (the detail of the poset will be explained in the next subsection). For real-time systems, when functions are triggered at system time instant $r$, they are required to be completed before a specific time, which is called *deadline*, represented by $d_i$. Moreover, different functions have different degrees of importance to the system, we use $v_i$ to denote the values obtained by the system through completing functions $F_i$ before its deadline $d_i$. Based on above explanation, we define the function $F_i = ((T_i, \prec), r_i, d_i, v_i)$.

Note that, unlike many research on real-time scheduling that set deadlines to tasks, we set deadline to the function level rather than task level. This setting can better reflect the reality that the deadline requirement is for the functions of real-time systems, while a function is achieved by a series tasks cooperated together.

This function definition denotes the functions which have hard real-time properties. That is, for such a function, if it

misses its deadline, system will not obtain any value through completing it. Usually, in a complex real-time system, not all functions are hard real-time functions. Some functions are soft real-time functions. For such a function, if it misses deadline, it will still be useful for the system, but the value obtained by completing such function will be less than completing it before its deadline. To denote the function with soft real-time properties, we define such functions as: $F_i = ((T_i, \prec), r_i, d_i, v_i * f(cp_i - d_i))$, where $cp_i$ is the time when the system completes the function, and the coefficient function $f(cp_i - d_i)$ characters how the value $v_i$ will decrease when the function misses its deadline. The reasonable value of the coefficient function is in interval $[0\ 1]$. For convenience, we use $\mathcal{FH} \subseteq \mathcal{F}$ to denote the set of functions with hard real-time properties, and use $\mathcal{FS} \subseteq \mathcal{F}$ to denote the set of functions with soft real-time properties.

### B. Task Poset

A multiprocessor real-time system comprises a set of tasks, denoted by $\mathcal{T}$. For each function, it is achieve by a series tasks cooperated together. Poset $(T_i, \prec)$ is used to denote such a series tasks, where $T_i \subseteq \mathcal{T}$ is the task set corresponding to $F_i$, and $T_i = \{\tau_1, \tau_2, \ldots, \tau_m\}$, where $\tau_j \in T_i$ is a task, and $m$ is the number of tasks. We use $\tau_{i,j}$ to indicate task $\tau_j \in T_i$. We assume that, `if` $|\mathcal{F}| > 1$, `then` $\forall T_i, T_j \subset \mathcal{T}, i \neq j \implies T_i \cap T_j = \emptyset$. That is, no tasks are shared by different functions [1]. The symbol $\prec$ indicates the dependency relation between two tasks. That is, $\tau_k, \tau_j \in T_i, \tau_k \prec \tau_j$ indicate that task $\tau_j$ can start to run only after task $\tau_k$ has been completed. The dependency relation is transitive. That is, $\tau_k \prec \tau_j, \tau_j \prec \tau_l \implies \tau_k \prec \tau_l$.

Definition (start task). *A start task of $(T_i, \prec)$ is such a task $\tau_i \in T_i$ that starts earliest of all the tasks in $T_i$, that is, $\forall \tau_j \in T_i, i \neq j \implies \tau_i \prec \tau_j$.*

Definition (end task). *A end task of $(T_i, \prec)$ is such a task $\tau_i \in T_i$ that starts latest of all the tasks in $T_i$, that is, $\forall \tau_j \in T_i, i \neq j \implies \tau_j \prec \tau_i$.*

Without losing generality, we assume that there is one start task and one end task of $(T_i, \prec)$, and use $\tau s_i$ and $\tau e_i$ to indicate the start task and end task of task poset $(T_i, \prec)$, respectively [2]. Each task has two parameters, $\tau_j = (c_j, m_j)$, where $j$ is the index of the task. $c_j$ is the required computation cost, which means the number of time slots (ticks of processor) needed by a unit speed processor to complete task $\tau_j$; and $m_j$ is the required migration cost for task $\tau_j$ migrating from a processor to another one. We use the parameter $m_j$ combined with parameters of network (the details will be explained later) to calculated the overheads of migrating tasks.

### C. Processor Set

In multiprocessor real-time systems, different processors are used to execute tasks. We use $\mathcal{P} = \{p_1, p_2, \ldots, p_l\}$ to denote the set of processors, where $l$ is the number of processors. Each processor $p_a$ is a 2-tuple, $p_a = (ps_a, TS_a)$, where $a$ is the index of the processor. $ps_a$ is the speed of the processor. When task $\tau_i$ running on processor $p_a$, the number of time slots needed for processor $p_a$ to complete task $\tau_i$, represented by *task completion* $tc_a^i$:

$$tc_a^i = \frac{c_i}{ps_a} \tag{1}$$

$TS_a$ is the task set that can be completed by processor $p_a$. This parameter is for *heterogeneous* systems, as in such systems, processors have different architectures, some tasks can only be executed on some specific processors. If $TS_a = \emptyset$, it means processor $p_a$ cannot be used to execute any task in the system.

Processors have independent local clocks, they are synchronized with each other in the time domain through synchronization protocol. The maximum difference between the local clocks of any two processors in the networked systems is called *network precision* (also called synchronization jitter) which is a global constant. We denote the network precision with $\delta$.

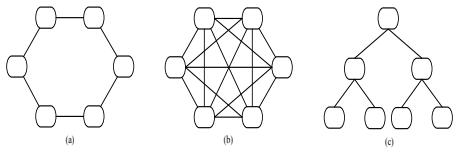


Fig. 2. Different types of network topologies: $a$. ring, $b$. mesh, $c$. tree

### D. Network Channel Set

In multiprocessor real-time systems, processors are connected through network channels. We use $\mathcal{N} \subseteq \mathcal{P} \times \mathcal{P}$ to denote the set of network channel. $n_{a \to n} \in \mathcal{N}$ denotes the network channel from processor $p_a$ to $p_b$, where $p_a, p_b \in \mathcal{P}, a \neq b$. Since we consider bi-directional network channel, we have $\forall n_{a \to b} \in \mathcal{N} \implies n_{b \to a} \in \mathcal{N}$. We use $ns_{a \to b}$ to represent the speed of $n_{a \to b}$.

Note that, define the network channel set as $\mathcal{N} \subseteq \mathcal{P} \times \mathcal{P}$ makes the system model become very general which includes any types of network topologies. For example, as shown in Fig. 2, the network channel set for mesh topology ($b$ in the Fig. 2) equals to $\mathcal{P} \times \mathcal{P}$, while the ring and tree topologies is the subset of $\mathcal{P} \times \mathcal{P}$. Moreover, this definition is also suitable for processor with multi-cores. For example, for a processor $A$ with four cores, in this definition, can be represented as four processors connect with network channels in mesh topology, and the speed of networks is set based on the data transfer speed inside the processor $A$.

When the data of the computed results of task $\tau_i$ migrates from processor $p_a$ to $p_b$[3], the time slots spent on network channel, represented by $tm_{a \to b}^i$, can be calculated as:

$$tm_{a \to b}^i = \frac{m_i}{ns_{a \to b}} \tag{2}$$

[1]Note that, this assumption is for concise expression. In real systems, if task $\tau_k \in \mathcal{T}$ is used by function $F_i$ and $F_j$, we can use two tasks $\tau_{ik}$ and $\tau_{jk}$, to represent $\tau_k$ used in function $F_i$ and $F_j$, respectively.

[2]To express a function with many starts (end) tasks, we can set a virtual task, with empty operation, start before all the starts tasks (start after all the end task) to be the start (end) task.

[3]For conciseness, we say "task $\tau_i$ migrates from processor $p_a$ to $p_b$" to mean "the data of the computed results of task $\tau_i$ migrates from processor $p_a$ to $p_b$" in the reset of the paper.

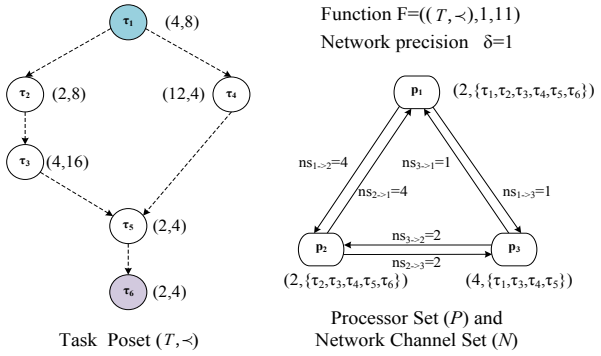


Fig. 3. An example for scheduling multiprocessor real-time systems

Based on $tm^i_{a\to b}$, we can get the time instant that processor $p_b$ receives the data of task $\tau_i$ migrating from processor $p_a$ through network channel $n_{a\to b}$, represented by $r^i_{a\to b}$, as

$$r^i_{a\to b} = s^i_{a\to b} + tm^i_{a\to b} + \delta \tag{3}$$

where, $s^i_{a\to b}$ is the start time of $\tau_i$ migrating through network channel $n_{a\to b}$, and $\delta$ is the network precision.

For a distributed real-time system, different processors are connected through network channels which are built by routers. As different routers support different network protocols, some tasks may not be migrated through some network channels. To capture this characteristics, similar as the heterogeneous processors, we also can define the heterogeneous network channels. We use $TS_{a\to b}$ to denote that task set that can be transferred through network channel $n_{a\to b}$.

### E. Assumptions

Applied to this system model, we require that all the parameters of the functions and tasks are known a prior. This requirement makes the model become a generalization of the widely studied *period task model*, in which all the tasks in the system are released periodically. This means our method applies more broadly than other methods which are specified on period task model. To guarantee a certain level of determinacy, in this paper, task preemption is not allowed.

To illustrate the defined system model, an example of scheduling for multiprocessor real-time systems is shown in Fig. 3. In this example, there are three processors $p_1, p_2, p_3$ in the system. These processors are connected with each other through six network channels $n_{1\to2}, n_{2\to1}, n_{1\to3}, n_{3\to1}, n_{2\to3}, n_{3\to2}$, and these network channels support all the migration of all the tasks in $T$. The network precision $\delta$ is 1. In the system, a hard real-time function $F = ((T, \prec), 1, 11)$ is waiting to be executed on the processors. The task poset of the function is $(T, \prec)$ which consists of six tasks. Task dependency relations are described in a directed acyclic graph. An edge starting from task $\tau_i$ to task $\tau_j$ represented by a dotted line denotes a dependency relation $\tau_i \prec \tau_j$.

## IV. System Constraints

This subsection describes all the system constraints expressed in the sat model for the defined multiprocessor systems.

### A. Constraint on start execution time of functions

Task set $T_i$ corresponding to function $F_i$ can start to run only after the function is triggered. That is, the start execution time of the start task of the poset $(T_i, \prec)$ should be larger than the triggered time of function $F_i$.

$$\forall F_i \in \mathcal{F}, \forall p_a \in \mathcal{P}$$
$$s^{\tau s_i}_a \geq r_i$$

where symbol $s^{\tau s_i}_a$ denotes the start execution time of task $\tau s_i$ on processor $p_a$.

### B. Constraint on start time of task migration

If a task $\tau_i$ migrates from processor $p_a$ to processor $p_b$ through network channel $n_{a\to b}$, it means *i):* task $\tau_i$ has been completed by processor $p_a$; or *ii):* $\tau_i$ has migrated to processor $p_a$ from another processor. For the first case, task $\tau_i$ can start to migrate after it has been completed, and for the second case, task $\tau_i$ can start to migrate after it has already migrated to processor $p_a$.

$$\forall \tau_i \in \mathcal{T}, \forall n_{a\to b} \in \mathcal{N}, \exists n_{c\to a} \in \mathcal{N}$$
$$(s^i_{a\to b} \geq s^i_a + tc^i_a) \vee (s^i_{a\to b} \geq r^i_{c\to a})$$

where symbol $s^i_{a\to b}$ denotes the start time of task $\tau_i$ migrating through network channel $n_{a\to b}$, $s^i_a$ denotes the start execution time of task $\tau_i$ on processor $p_a$.

### C. Constraint on task dependency

For processor $p_a$, if $\tau_i \prec \tau_j$, task $\tau_j$ can start to run only after $\tau_i$ has been completed. Similar to the constraints on start time of task migration, there are two cases. *i):* task $\tau_i$ has been completed by processor $p_a$; *ii):* task $\tau_i$ has migrated to processor $p_a$ from another processor. For the first case, $\tau_j$ can start to run after $\tau_i$ has been completed, and for the second case, $\tau_j$ can start to run after $\tau_i$ has already migrated to processor $p_a$.

$$\forall \tau_i, \tau_j \in \mathcal{T}, \forall p_a \in \mathcal{P}, \exists n_{b\to a} \in \mathcal{N}$$
$$\tau_i \prec \tau_j \implies (s^j_a \geq s^i_a + tc^i_a) \vee (s^j_a \geq r^i_{b\to a})$$

### D. Constraint on execution of processors

A processor can execute only one task at a time. This is interpreted as: there is no overlap of the execution time of any two tasks.

$$\forall \tau_i, \tau_j \in \mathcal{T}, i \neq j, \forall p_a \in \mathcal{P}$$
$$(s^i_a \geq s^j_a + tc^j_a) \vee (s^j_a \geq s^i_a + tc^i_a)$$

### E. Constraint on network channels

A network channel can transfer data of only one task at a time. That is, there is no overlap of the migration time of any two tasks on a network channel.

$$\forall \tau_i, \tau_j \in \mathcal{T}, i \neq j, \forall n_{a\to b} \in \mathcal{N}$$
$$(s^i_{a\to b} \geq s^j_{a\to b} + tm^j_{a\to b}) \vee (s^j_{a\to b} \geq s^i_{a\to b} + tm^i_{a\to b})$$

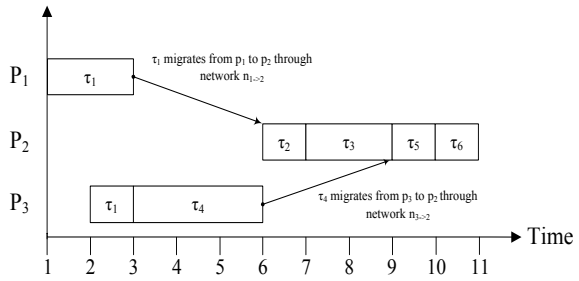


Fig. 4. The scheduling result for example shown in Fig. 3 by using the proposed SMT-based scheduling

### F. Constraint on heterogeneous processors

In heterogeneous systems, processors have different architectures, some tasks can only be executed on some specific processors. For tasks that cannot be executed on some processors, the start execution time of the tasks in such processors are set to $+\infty$, which means the tasks will never start to run on these specific processors.

$$\forall p_a \in \mathcal{P}, \forall \tau_i \in \mathcal{T} - TS_a$$
$$s_a^i = +\infty$$

### G. Constraint on heterogeneous network channels

For a distributed real-time system, different processors are connected through network channels which are built by routers. As different routers support different network protocols, some tasks may not be migrated through some network channels. Similar as the constraint on heterogeneous processors, for tasks that cannot migrate on some network channels, the start migration time of the tasks in such network channels are set to $+\infty$, which means the tasks will never start to migrate on these specific network channels.

$$\forall n_{a\to b} \in \mathcal{N}, \forall \tau_i \in \mathcal{T} - TS_{a\to b}$$
$$s_{a\to b}^i = +\infty$$

## V. Target Constraints

There are many targets can be considered when we design scheduling for real-time systems. Which objectives are appropriate in a given situation depends, of course, upon the application. In this section, we give design guidelines for different scheduling targets.

### A. Make all the functions meet their deadlines

Under normal workload conditions, the desired schedule should make sure that every triggered function can be completed before its deadline.

$$\forall F_i \in \mathcal{F}, \exists p_a \in \mathcal{P}$$
$$s_a^{\tau e_i} + tc_a^{\tau e_i} \le d_i$$

where symbol $s_a^{\tau e_i}$ is the start execution time of task $\tau e_i$ on processor $p_a$, and $tc_a^{\tau e_i}$ is the number of time slots needed for processor $p_a$ to complete task $\tau e_i$.

Based on this scheduling target, recall the example shown in Fig. 3, we can get the solution model $\mathcal{M}$ which defines the values of the start time of task execution on processor, $s_a^j$, and the start time of task migration through network, $s_{b\to c}^j$, for $\forall f_i \in \mathcal{F}, \forall \tau_j \in T_i, \forall p_a \in \mathcal{P}, \forall n_{b\to c} \in \mathcal{N}$. Based on the model $\mathcal{M}$, we can get the scheduling results as shown in Fig. 4. This scheduling sequence can make the function $F$ in Fig. 3 meet its deadline. Some characteristics of this scheduling sequence should be noticed:

- Task $\tau_1$ has been executed on processor $p_1$ from system time $t = 1$ to $t = 3$, and it has also been executed on processor $p_3$ from system time $t = 2$ to $t = 3$. This means, the SMT-based scheduling framework can handle the parallel execution of tasks, and can make a task repeatedly run on different processors when such repeated execution is necessary.
- Task $\tau_2$ runs on processor $p_2$ from $t = 6$ to $t = 7$. Although task $\tau_2$ needs the computed results from completing task $\tau_1$, such computed results can not only be obtained by completing task $\tau_1$ on processor $p_2$ itself, but also can be obtained by transferring the computed results from other processor that has completed task $\tau_1$. Specified to this example, at system time $t = 6$, processor $p_2$ gets the computed results of task $\tau_1$ from processor $p_1$.

### B. Maximize obtained values of completed functions

Under normal workload conditions, there exist a schedule can make all the triggered functions meet their deadlines. However, in practical environment, system workload may vary widely because of dynamic changes of work environment. Once system workload becomes too heavy so that there does not exist a feasible schedule can make all the functions meet their deadlines, we say the system is *overloaded*. When system is overload, one reasonable scheduling target is to maximize the obtained values of the completed functions.

Let symbol $v$ be the obtained values of the completed functions, and its initial value is set to be 0. For functions with hard deadlines, system can obtain their values only when such functions have been completed before their deadlines.

$$\forall F_i \in \mathcal{FH}$$
$$\texttt{if } \exists p_a \in \mathcal{P}, s_a^{\tau e_i} + tc_a^{\tau e_i} \le d_i$$
$$v := v + v_i$$
$$\texttt{end}$$

For completing functions $F_i$ with soft deadlines, the value that the system can obtain is according to the coefficient functions $f(cp_i - d_i)$, where $cp_i$ is the time when the system completes the function. As more earlier completing the function, more values the system can obtain, the completing time should choose the earliest time that completing the function $F_i$ among all the processors. Based on this analysis, we can get the formula

$$\forall F_i \in \mathcal{FS}, \exists p_a \in \mathcal{P}$$
$$\texttt{if } s_a^{\tau e_i} = \texttt{min}(s_{\mathcal{P}}^{\tau e_i})$$
$$v := v + v_i * f(s_a^{\tau e_i} + tc_a^{\tau e_i} - d_i)$$
$$\texttt{end}$$

where, function $\texttt{min}(s_{\mathcal{P}}^{\tau e_i})$ returns the minimum value $s^{\tau e_i}$ among all the processors in set $\mathcal{P}$. Let symbol $sv$ denote the maximum obtained values of the completed functions, and

obviously, $sv$ is no less than 0 and no larger than $\sum v_i$ for $\forall F_i \in \mathcal{F}$. The constraints on the scheduling target can be expressed as:

$$v = sv$$

### C. Make hard deadline functions meet deadlines while maximizing obtained values of the completed soft deadline functions

Since hard deadline functions usually play important roles in a real-time system, when system is under overload condition, a reasonable scheduling target is to first make sure that all the hard deadline functions meet their deadlines, meanwhile, maximizing obtained values of the completed soft deadline functions. To make hard deadline functions meet deadlines, we can get

$$\forall F_i \in \mathcal{FH}, \exists p_a \in \mathcal{P}$$
$$s_a^{\tau e_i} + tc_a^{\tau e_i} \leq d_i$$

To maximize the obtained value of the completed soft deadline functions, the formula is similar as it for the previous scheduling target. Let symbol $v$ be the obtained values of the completed functions, and its initial value is set to be 0.

$$\forall F_i \in \mathcal{FS}, \exists p_a \in \mathcal{P}$$
$$\texttt{if } s_a^{\tau e_i} = \texttt{min}(s_{\mathcal{P}}^{\tau e_i})$$
$$\quad v := v + v_i * f(s_a^{\tau e_i} + tc_a^{\tau e_i} - d_i)$$
$$\texttt{end}$$

Let symbol $sv$ denote the maximum obtained values of the completed functions, and obviously, $sv$ is no less than 0 and no larger than $\sum v_i$ for $\forall F_i \in \mathcal{FS}$. The constraints on scheduling target can be expressed as:

$$v = sv$$

## VI. Related Work

The research on real-time scheduling has lasted for decades, many research have been conducted on this area. For research on designing scheduling for multiprocessor systems, a comprehensive survey can be found in [5]. In [8], the *Proportionate Fair* (Pfair) algorithm was introduced. Pfair is a schedule generation algorithm which is applicable to periodic tasksets with implicit deadlines. It is based on the idea of fluid scheduling, where each task makes progress proportionate to its utilization. Pfair scheduling divides the timeline into equal length quanta or slots. Authors in [8] showed that the Pfair algorithm is optimal for periodic tasksets with implicit deadlines. In [9], authors extended the PFair approach to sporadic tasksets, showing that the EPDF (earliest pseudodeadline first) algorithm, a variant of Pfair, is optimal for sporadic tasksets with implicit deadlines executing on two processors, but is not optimal for more than two processors.

Some approaches focus on studying task and messages schedule co-synthesis in switched time-triggered networks. In [10], authors studied time-triggered distributed systems where periodic application tasks are mapped onto different end stations (processing units) communicating over a switched Ethernet network. They try to solve the scheduling problem using a MIP multi-objective optimization formulation. In [11], authors studied the system consisting of communicating event- and time-triggered tasks running on distributed nodes. These tasks are scheduled in conjunction with the associated bus messages by using dynamic and static scheduling methods, respectively.

Hitherto, most of the presented methods are either limited to specific task model (e.g., [8, 10] limited to periodic tasksets) or simple system architecture (e.g., [9] limited to two processors, [11] simple bus network topologies). Compared with these works, our proposed framework is flexible and sufficiently general to apply to various kinds of real-time systems and various scheduling targets, which makes that our framework applies much more widely.

## VII. Conclusion

In this paper, based on satisfiability modulo theories (SMT), we provide a framework to design scheduling for real-time systems. In the framework, the problem of scheduling is treated as a satisfiability problem. After using first-order language to formalize the satisfiability problem, a SMT solver is employed to solver such a problem. An optimal schedule can be generated based on a solution model returned by the SMT solver. To demonstrate the practicality of the framework, we give design guidelines for real-time systems with multiprocessor. Through the demonstration, the framework is found flexible and sufficiently general to apply to different kinds of real-time systems. By giving the practical design guidelines, we believe that our framework can benefit system designers to efficiently design scheduling.

For the future work, in order to study the performance of the SMT-based scheduling framework in a real application, we would like to implement the proposed framework in a real multiprocessor real-time system.

## References


[1] F. Zhang and A. Burns, "Schedulability analysis for real-time systems with EDF scheduling," *IEEE Trans. Comput.*, vol. 58, no. 9, pp. 1250–1258, Apr. 2009.

[2] C. Barrett, R. Sebastiani, R. Seshia, and C. Tinelli, "Satisfiability modulo theories," Handbook of Satisfiability, vol. 185. IOS Press, 2009.

[3] L.d. Moura N. Bjrner, "Satisfiability Modulo Theories: An Appetizer," *Formal Methods: Foundations and Applications*, vol. 5902, pp. 23–26, 2009.

[4] S.S. Craciunas and R.S. Oliver, "SMT-based Task- and Network-level Static Schedule Generation for Time-Triggered Networked Systems," *Proc. 22th Int. Conf. on Real-Time Networks and Systems*, NY, USA, pp. 45–54, October, 2014.

[5] R.I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comput. Surv.*, vol. 43, no. 5, pp. 35:1–35:44, Oct. 2011.

[6] L. Moura and N. Bjrner, "Z3: an efficient SMT solver," *Proc. 14th Int. Conf. on Tools and Algorithms for the Construction and Anal. of Syst.*, Budapest, Hungary, LNCS 4963, pp. 337–340, Springer-Verlag, 2008.

[7] B. Dutertre, "Yices 2.2," *Proc. 26th Int. Conf. on Comput. Aided Verification*, Vienna, Austria, LNCS 8559, pp. 737–744, Springer International Publishing, 2014.

[8] S.K. Baruah, N. Cohen, G. Plaxton, and D. Varvel, "A notion of fairness in resource allocation," *Algorithmica*, vol. 15, no. 6, pp. 600–625, 1996.

[9] J. Anderson and A. Srinivasan, "Early-release fair scheduling," *Proc. of the Euromicro Conference on Real-Time Systems*, 2000.

[10] L. Zhang, D. Goswami, R. Schneider, and S. Chakraborty, "Task- and network-level schedule co-synthesis of Ethernet-based time-triggered systems," *Proc. of ASP-DAC*, 2014.

[11] T. Pop, P. Eles, and Z. Peng, "Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems," *Proc. of CODES*, 2002.