PAPER

SMT-Based Scheduling for Overloaded Real-Time Systems*,**

Zhuo CHENG[†a)], **Haitao ZHANG**[††], ***Nonmembers***, **Yasuo TAN**[†], ***and*** **Yuto LIM**[†], ***Members***

SUMMARY In a real-time system, tasks are required to be completed before their deadlines. Under normal workload conditions, a scheduler with a proper scheduling policy can make all the tasks meet their deadlines. However, in practical environment, system workload may vary widely. Once system workload becomes too heavy, so that there does not exist a feasible schedule can make all the tasks meet their deadlines, we say the system is *overloaded* under which some tasks will miss their deadlines. To alleviate the degrees of system performance degradation caused by the missed deadline tasks, the design of scheduling is crucial. Many design objectives can be considered. In this paper, we first focus on maximizing the total number of tasks that can be completed before their deadlines. A scheduling method based on *satisfiability modulo theories (SMT)* is proposed. In the method, 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 solve this satisfiability problem. An optimal schedule can be generated based on the solution model returned by the SMT solver. The correctness of this method and the optimality of the generated schedule can be verified in a straightforward manner. The time efficiency of the proposed method is demonstrated through various simulations. Moreover, in the proposed SMT-based scheduling method, we define the scheduling constraints as system constraints and target constraints. This means if we want to design scheduling to achieve other objectives, only the target constraints need to be modified. To demonstrate this advantage, we adapt the SMT-based scheduling method to other design objectives: maximizing effective processor utilization and maximizing obtained values of completed tasks. Only very little changes are needed in the adaption procedure, which means the proposed SMT-based scheduling method is flexible and sufficiently general.

key words: *real-time scheduling, SMT, overload, satisfiability problem*

1. Introduction

Real-time systems play important roles in our society. For example, chemical and nuclear plant control, space missions, flight control, telecommunications, and multimedia systems are all real-time systems [2]. 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. The execution order of the tasks (i.e., schedule) is set by a scheduler. Under normal workload conditions, a scheduler with a proper scheduling policy can make all the tasks completed before their deadlines (i.e., meet their deadlines). However, in practical environment, system workload may vary widely because of dynamic changes in the work environment. Once system workload becomes too heavy so that there does not exist a feasible schedule can make all the tasks meet their deadlines, we say the system is *overloaded*.

When overload problem happens, it is important to minimize the degrees of system performance degradation caused by the missed deadline tasks. A system that panics and suffers a drastic fall in performance when a problem happens, is likely to contribute to this problem, rather than helping to solve it [3]. To achieve this target, the design of scheduling is crucial, as different scheduling policies will lead to different degrees of performance degradation. Many objectives for the design of scheduling policies described in [4], [5] can be considered. For example, (*i*) maximizing total number of tasks that meet deadlines, (*ii*) maximizing effective processor utilization, (*iii*) maximizing obtained values of completed tasks. Which objectives are appropriate in a given situation depends, of course, upon the application. We first focus on maximizing total number of tasks that meet their deadlines. This objective is reasonable upon the application that when a missed deadline task corresponds to a disgruntled customer, and the aim is to keep as many customers satisfied as possible [3]. Then, we adapt the SMT-based scheduling method to the design objectives: maximizing effective processor utilization and maximizing obtained values of completed tasks. The main contributions of this paper are:

i) We propose a scheduling method based on satisfiability modulo theories (SMT). In this method, the problem of scheduling overload is treated as a satisfiability problem. The key work is to formalize the problem using first-order language. After the formalization, a *sat model* is constructed to represent the satisfiability 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 [14], Yices [15]) 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

Manuscript received September 3, 2016.
Manuscript revised December 9, 2016.
Manuscript publicized January 23, 2017.

[†]The authors are with School of Information Science, Japan Advanced Institute of Science and Technology, Nomi-shi, 923–1292 Japan.
[††]The author is with School of Information Science and Engineering, Lanzhou University, China.

*The preliminary version is in [1].

**This work is partially supported by the National Science Foundation of China (Grants No. 61602224) and the Fundamental Research Funds for the Central Universities (Grants No. lzujbky-2016-142 and No. lzujbky-2016-k07).

a) E-mail: chengzhuo@jaist.ac.jp (Corresponding author)

DOI: 10.1587/transinf.2016EDP7374

of this method and the optimality of the generated schedule can be verified in a straightforward manner. We also conduct various simulations to evaluate the time efficiency of the proposed method. The simulation results demonstrate that the SMT-based scheduling method is more time efficient compared to existing scheduling algorithms. To the best of our knowledge, this is the first work to use SMT to solve the overload problem in the real-time scheduling domain.

ii) The proposed SMT-based scheduling method is flexible and sufficiently general. In the SMT-based scheduling method, we define the scheduling constraints as system constraints and target constraints. This means if we want to design scheduling to achieve other objectives (e.g., maximizing effective processor utilization), only the target constraints need to be modified while system constraints can be totally reused. Alternatively, if we want to achieve the same scheduling objective for another real-time system with different system architecture (e.g., distributed real-time systems), only the system constraints need to be modified. We believe, due to these advantages, the proposed SMT-based scheduling method can help developers easily and efficiently design scheduling for real-time systems with low design cost.

Extension to the preliminary version [1]. Work in [1] only applies the proposed SMT-based scheduling method to one scheduling target, maximizing total number of tasks that meet their deadlines, and does not give the design guidelines for other scheduling targets. It means, work in [1] does not clearly address the advantage of the proposed scheduling method, that is, when we apply the method to other scheduling targets, only the target constraints need a little modification while system constraints can be totally reused (More details can be found in Sect. 5). Moreover, in this paper, the results of the scheduling overheads measured by the number of task preemption, discussions on considering other critical resources, and comparison of the proposed SMT-based scheduling method and static priority based scheduling algorithms are also given, which are not shown in [1].

Organization of this paper. In Sect. 2, we present the system model and give the research background. The details of the SMT-based scheduling are described in Sect. 3. Simulation and performance evaluation are shown in Sect. 4. In Sect. 5, we adapt the SMT-based scheduling to different scheduling targets. Discussions on considering other critical resources, and comparing the SMT-based scheduling and static priority based scheduling methods are conducted in Sect. 6. Section 7 summarizes the related works. Conclusions and future work are given in Sect. 8.

2. System Model, Definition and Background

2.1 System Model

We adopt the general *firm-deadline* model proposed in [7] for system with uniprocessor. The "firm-deadline" means only tasks completed before their deadlines are considered valuable, and any task missed its deadline is worthless to system. A real-time system comprises a set of real-time tasks waiting to execute. These tasks request processor to execute when they arrive in the system. Each task τ_i is a 3-tuple $\tau_i = (r_i, c_i, d_i)$, where i is the index of a task, r_i is the request time instant, c_i is the required execution time, and d_i is the deadline. A reasonable task should meet that $r_i + c_i \leq d_i$. Symbol rc_i represents the remaining execution time of task τ_i. Initially, it equals to c_i. After τ_i has been executed for δ ($\delta \leq c_i$) time slots, $rc_i = c_i - \delta$. If $rc_i = 0$, it means τ_i has been completed. Symbol $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ denotes the set of tasks comprised in the system, where n is the number of tasks.

To allow task preemption, for all tasks in $\mathcal{T}$, task τ_i is defined as consisting of a series of indivisible fragment (atomic operation), denoted by $\tau_i : (f_1, f_2, \ldots, f_m)$, where $m = |\tau_i|$ is the number of fragments in task τ_i[†]. $f_{i,j}$ denotes the j-th fragment of τ_i, and the last fragment of task τ_i is denoted by $f_{i,e}$. We use $s_{i,j}$ to represent the start execution time of $f_{i,j}$. Symbol $c_{i,j}$ denotes the required execution time of $f_{i,j}$, and $\forall f_j \in \tau_i, \sum c_{i,j} = c_i$. For $1 \leq i < m$, f_{i+1} can start to run only when f_i has been completed. A successfully completed task τ_i means $f_{i,e}$ has been allocated $c_{i,e}$ time slots in time interval $[r_i, d_i)$. A task τ_i should be discarded at system time t, if it features $rc_i > d_i - t$, as such task cannot be successfully completed. For convenience, symbols used throughout the paper are summarized in Table 1.

Table 1 Symbols and Definitions

Symbol	Definition
t	system time instant
$\mathcal{T}$	set of real-time tasks
τ_i	real-time task, $\tau_i \in \mathcal{T}$, i is index of the task
r_i	request time instant of τ_i
c_i	required execution time of τ_i
rc_i	remaining execution time of τ_i
d_i	deadline of τ_i
$f_{i,j}$	j-th indivisible fragment of τ_i
$f_{i,e}$	last indivisible fragment of τ_i
$s_{i,j}$	start execution time of $f_{i,j}$
$c_{i,j}$	required execution time of $f_{i,j}$

Applied to this task model, we require all the parameters of the tasks are known a priori. This requirement makes the task model become a generalization of the widely studied *periodic* task model, in which all the tasks in the system are released periodically. This means our method applies more widely than other methods dealing with overload problem which are specified on periodic task model.

2.2 Definition

In a real-time system, scheduler can use different schedules to schedule task set $\mathcal{T}$.

[†]The proposed SMT-based scheduling can also deal with condition that task preemption is prohibited, by just constraining every tasks consisting only one indivisible fragment.

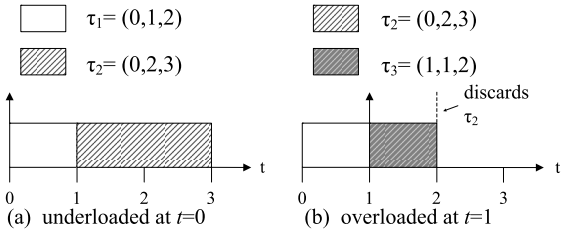

Fig. 1 Example for *underloaded* & *overloaded*

When there exists a schedule that can make all tasks meet their deadlines, the system is ***underloaded****, and the task set is* ***feasible****. On the contrast, when there does not exist a schedule that can make all the tasks meet their deadlines, the system is* ***overloaded****, and the task set is* ***infeasible*** *(Ref. [2]).*

An example in Fig. 1 is used to elaborate this definition. As shown in Fig. 1 (a), at $t = 0$, $\mathcal{T}' = \{\tau_1, \tau_2\}$, where $\mathcal{T}'$ is the set of tasks that have arrived in the system (not including tasks that have been successfully completed or missed deadlines). Using earliest deadline first (EDF) algorithm to schedule $\mathcal{T}'$ can make all tasks meet their deadlines, where EDF first schedules the task with the earliest deadline. Thus, the system is underloaded, and the task set $\mathcal{T}'$ is feasible. EDF algorithm proposed in literature [11] has been proven as an *optimal* scheduling algorithm on uniprocessor. That is, if using EDF to schedule a task set cannot make all tasks meet their deadlines, no other algorithms can. Thus, EDF scheduling algorithm can be used to tell if a task set is feasible.

After system passed a time unit, as shown in Fig. 1 (b), τ_1 has been successfully completed, and a new task τ_3 arrives in the system. At that time, $\mathcal{T}' = \{\tau_2, \tau_3\}$. Using EDF to schedule $\mathcal{T}'$ can only make τ_3 meet its deadline. Task τ_2 should be discarded at $t = 2$, as $rc_2 > d_2 - t$, where $d_2 = 3, rc_2 = 2$. Thus, the system is overloaded, and the task set $\mathcal{T}'$ is infeasible.

2.3 Background

2.3.1 Three Representative Scheduling Algorithms

There are many scheduling algorithms used in various real-time systems. In this subsection, we study three widely used scheduling algorithms: shortest remaining time first (SRTF), EDF, and least laxity first (LLF). Through an example described in Fig. 2, we can study their performance when system is overloaded. In the example, the lengths of all the indivisible fragments in all the tasks are set to one.

Scheduling results: (*i*): SRTF first schedules the task with the shortest remaining execution time. The scheduling sequence is $(\tau_3, \tau_1, \tau_1, \tau_4, \tau_1)$. By this sequence, τ_4 and τ_1 can be completed sequentially. (*ii*): EDF first schedules the task with the earliest deadline. The result of scheduling sequence is $(\tau_2, \tau_2, \tau_2, \tau_2, \tau_2, \tau_4)$. It can complete τ_2 and τ_4 sequentially. (*iii*): LLF first schedules the task with the least laxity. For τ_i, the laxity l_i is computed as $l_i = d_i - rc_i - t$. It can complete tasks τ_2 and τ_4 sequentially with the same scheduling sequence generated by EDF.

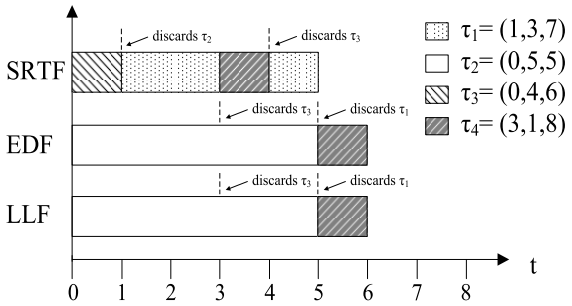

Fig. 2 Performance of scheduling algorithms

All of the three scheduling algorithms achieve two as the number of task completion. We wonder if it is the maximum value. For this simple example with only four tasks, we can enumerate all the schedule to find the maximum number of task completion. An optimal schedule is $(\tau_3, \tau_3, \tau_3, \tau_3, \tau_1, \tau_1, \tau_1, \tau_4)$ which can complete three tasks τ_3, τ_1, and τ_4 sequentially.

Based on the analysis above, through this example, we can see that, for overloaded real-time system, a new scheduling method is needed. This motives our work. A SMT-based scheduling method will be proposed in Sect. 3.

2.3.2 Satisfiability Modulo Theories (SMT)

Satisfiability modulo theories check the satisfiability of logic formulas in first-order formulation with regard to certain background theories like linear integer arithmetic or bit-vectors [12]. A first-order logic formula uses variables as well as quantifiers, functional and predicate symbols, and logical operators [13]. 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 solve such a problem. If all the logic formulas are satisfiable, the SMT solver will return the result `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 result `unsat` with an empty model, for the case $\exists a, b \in \mathbb{Z}$.

3. SMT-Based Scheduling

3.1 Overview of the SMT-Based Scheduling

The overview of the SMT-based scheduling is illustrated in

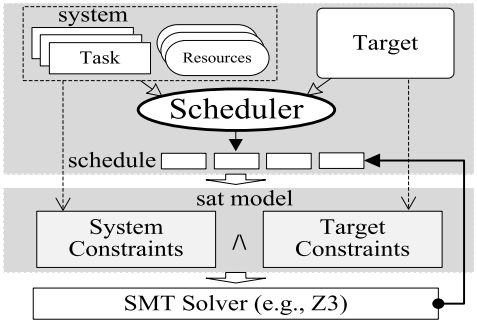

Fig. 3 Overview of the SMT-based scheduling method

Fig. 3. In a real-time system, a schedule (execution order of tasks) is generated by a scheduler. When overload problem happens, under the specific system and desired scheduling target, the scheduling problem 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 a set of first-order logic formulas (within linear arithmetic in the formulas) which express all the constraints that a desired optimum schedule should satisfy. There are two kinds of constraints: *system constraints* and *target constraints*. System constraints are based on the specific system. For example, for uniprocessor, a schedule should make sure that the execution of two tasks cannot have overlap in time domain. Target constraints are based on the scheduling target. Specific to our target, a desired optimum schedule should achieve the maximum number of task completion.

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, a desired optimum schedule can be generated.

3.2 Scheduling Constraints

This subsection describes all the constraints expressed in the sat model.

System Constraints

3.2.1 Constraint on Start Execution Time of Tasks

As a task can only start to execute after it requests to run, the start time of the first fragment of a task should be larger than the request time instant r_i.

$$\forall \tau_i \in \mathcal{T}$$
$$s_{i,1} \geq r_i$$

3.2.2 Constraint on Start Execution Time of Different Fragments

The series of fragments consisted in a task should be executed sequentially. Therefore, a fragment of a task can start to run only when its previous fragments of the task have been completed.

$$\forall \tau_i \in \mathcal{T}, \forall f_a, f_b \in \tau_i$$
$$b > a \implies s_{i,b} \geq s_{i,a} + c_{i,a}$$

Symbol $\implies$ denotes implication logical operator.

3.2.3 Constraint on Processor

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

$$\forall \tau_i, \tau_j \in \mathcal{T}, i \neq j, \forall f_a \in \tau_i, \forall f_b \in \tau_j$$
$$(s_{i,a} \geq s_{j,b} + c_{j,b}) \vee (s_{j,b} \geq s_{i,a} + c_{i,a})$$

3.2.4 Constraint on Task Dependency

In practical system, tasks usually have dependency relation with each other. For example, task τ_j may require the computed result of τ_i, thus, τ_j can start to run only after τ_i has been completed. We denote such dependency relation as $\tau_i \prec \tau_j$.

$$\forall \tau_i, \tau_j \in \mathcal{T}$$
$$\tau_i \prec \tau_j \implies (s_{j,1} \geq s_{i,e} + c_{i,e}) \wedge$$
$$(s_{i,e} + c_{i,e} > d_i \implies s_{j,1} = +\infty)$$

This formula expresses that any two tasks that have dependency relation $\tau_i \prec \tau_j$, the first fragment of task τ_j can start to run only when the last fragment of task τ_i has been completed. As the series of fragments consisted in a task are executed sequentially, this formula can make sure that task τ_j starts to run only after τ_i has been completed. Moreover, if task τ_i has not been successfully completed, task τ_j cannot start to run.

Target Constraints

3.2.5 Constraint on Scheduling Target

A successfully completed task τ_i should be completed before its deadline. As all the fragments consisted in a task run sequentially, this constraint can be interpreted as: the last fragment of a successfully completed task should be completed before its deadline. Let n be the number of successfully completed tasks, and its initial value is set to be 0.

Algorithm 1 Schedule Synthesis

Input: task set $\mathcal{T}$
Output: schedule $\mathcal{S}$
1: $\mathcal{A}$:= `Assert`($\mathcal{T}, |\mathcal{T}|$)
2: $\mathcal{M}$:= `CallSMTSolver`($\mathcal{A}$)
3: **if** $\mathcal{M} \neq \emptyset$ **then**
4: **return** $\mathcal{S}$ based on model $\mathcal{M}$
5: **end if**
6: $start := 0, end := |\mathcal{T}|$
7: **while** `true` **do**
8: $mid := start + \lfloor (end - start)/2 \rfloor$
9: $\mathcal{A}$:= `Assert`($\mathcal{T}, mid$)
10: $\mathcal{M}$:= `CallSMTSolver`($\mathcal{A}$)
11: **if** $\mathcal{M} = \emptyset$ **then**
12: $end := mid - 1$
13: **else**
14: $\mathcal{A}'$:= `Assert`($\mathcal{T}, mid + 1$)
15: $\mathcal{M}'$:= `CallSMTSolver`($\mathcal{A}'$)
16: **if** $\mathcal{M}' \neq \emptyset$ **then**
17: $start := mid + 1$
18: **else**
19: **return** $\mathcal{S}$ based on model $\mathcal{M}$
20: **end if**
21: **end if**
22: **end while**

$$\forall \tau_i \in \mathcal{T}$$
$$\texttt{if}\ (s_{i,e} + c_{i,e} \leq d_i)$$
$$n := n + 1$$
$$\texttt{end}$$

Let symbol sn denote the maximum number of tasks in $\mathcal{T}$ that can be successfully completed, and obviously, $sn \in [0 \ \ |\mathcal{T}|]$. The constraints on scheduling target can be expressed as:

$$n \geq sn$$

Note that, here we set $n \geq sn$ rather than $n = sn$ is based on our experience. With $n \geq sn$, following the schedule synthesis described in the next subsection, the time efficiency of the SMT-based scheduling method is better compared to with $n = sn$.

3.3 Schedule Synthesis

After all the constraints are defined, now we can employ a SMT solver to generate a desired schedule. The process of schedule synthesis is summarized in Alg. 1. Function `Assert`($\mathcal{T}, |\mathcal{T}|$) (line 1) interprets the constraints defined in Sect. 3.2 as *assertions* (boolean formulas that can be inputted into a SMT solver) with $|\mathcal{T}|$ as the maximum number of successfully completed tasks (i.e., set $sn := |\mathcal{T}|$ in *constraint on scheduling target*). The variables of these boolean formulas are the start time $s_{i,j}$ for $\forall \tau_i \in \mathcal{T}, \forall f_j \in \tau_i$. Function `CallSMTSolver`($\mathcal{A}$) (line 2) calls a SMT solver to find a solution model $\mathcal{M}$ for $\mathcal{A}$. If such a model does exist, it will be returned by the function, otherwise, an empty model will be returned.

We first set $sn := |\mathcal{T}|$ in constraint on scheduling target, that is to expect all the tasks in $\mathcal{T}$ can be successfully completed. If this expectation can be satisfied, which means overload problem does not happen, solution model $\mathcal{M}$ will be returned. As $\mathcal{M}$ contains all the values of $s_{i,j}$, for $\forall \tau_i \in \mathcal{T}, \forall f_j \in \tau_i$, we can extract the start execution time of all the fragments of all the tasks, which means the schedule $\mathcal{S}$ for task set $\mathcal{T}$ can be generated (line 1–5).

When overload problem happens, tasks in $\mathcal{T}$ cannot all be successfully completed. This condition is indicated by an empty model returned by function `CallSMTSolver`($\mathcal{A}$), which means *constraint on scheduling target* cannot be satisfied. We need to decrease the setting value of sn. To achieve the maximum number of task completion means to find the maximum value of sn with which there exists a solution model. We use *binary search* to find the maximum value of sn (line 6–22). With the maximum value of sn, a solution model can be returned by function `CallSMTSolver`($\mathcal{A}$). Meanwhile, with $sn := sn + 1$, `CallSMTSolver`($\mathcal{A}$) will return an empty model. This is the criterion to judge if the value of sn is the maximum value. When we get the solution model $\mathcal{M}$ with the maximum value sn, based on $\mathcal{M}$, the schedule $\mathcal{S}$ can be generated (line 19).

Through the procedure of the schedule synthesis, we can make sure that the maximum value of sn can be found. Meanwhile, as all the constraints of a desired optimum schedule have been satisfied, which means $\mathcal{S}$ can achieve the maximum number of task completion. This has demonstrated the optimality of the schedule generated by the proposed SMT-based scheduling method.

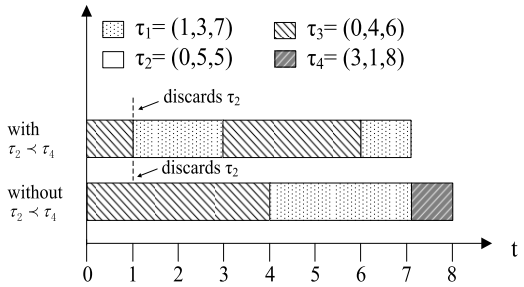

Fig. 4 Results by using the SMT-based scheduling method for the example shown in Fig. 2

3.4 Scheduling Results

Recall the example shown in Fig. 2. In this example, $\mathcal{T} = \{\tau_1, \tau_2, \tau_3, \tau_4\}$. Based on the schedule synthesis shown in Alg. 1, we can get the solution model $\mathcal{M}$ which defines the values of $s_{i,j}$ for $\forall \tau_i \in \mathcal{T}, \forall f_j \in \tau_i$. The model is as follows: $s_{1,1} = 4, s_{1,2} = 5, s_{1,3} = 6, s_{2,1} = 8, s_{2,2} = 9, s_{2,3} = 10, s_{2,4} = 11, s_{2,5} = 12, s_{3,1} = 0, s_{3,2} = 1, s_{3,3} = 2, s_{3,4} = 3, s_{4,1} = 7$. Based on this model, as shown in Fig. 4 (without $\tau_1 \prec \tau_4$), we can get the scheduling sequence $\mathcal{S} = (\tau_3, \tau_3, \tau_3, \tau_3, \tau_1, \tau_1, \tau_1, \tau_4)$ (as τ_2 cannot be successfully completed, it should not be included in $\mathcal{S}$). This scheduling sequence can complete three tasks τ_3, τ_1, and τ_4 consequently, which is the maximum number of task completion for $\mathcal{T}$.

If we add a task dependency relation $\tau_2 \prec \tau_4$, we can get the model: $s_{1,1} = 1, s_{1,2} = 2, s_{1,3} = 6, s_{2,1} = 7, s_{2,2} =$

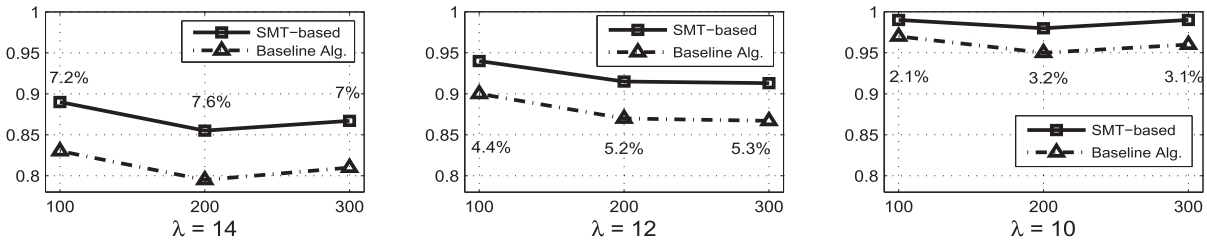

Fig. 5 Success ratio of the SMT-based scheduling and the baseline algorithms (The x-axis is the total number of input tasks, and the y-axis is the success ratio)

$8, s_{2,3} = 9, s_{2,4} = 10, s_{2,5} = 11, s_{3,1} = 0, s_{3,2} = 3, s_{3,3} = 4, s_{3,4} = 5, s_{4,1} = 12$. Based on this model, as shown in Fig. 4 (with $\tau_2 \prec \tau_4$), we can get the scheduling sequence $\mathcal{S} = (\tau_3, \tau_1, \tau_1, \tau_3, \tau_3, \tau_3, \tau_1)$. This scheduling sequence can complete two tasks τ_3 and τ_1 consequently, which is also the maximum number of task completion for $\mathcal{T}$ with the dependency relation $\tau_2 \prec \tau_4$.

4. Simulation and Evaluation

In this section, we present the results of simulations which are conducted to study the performance of the SMT-based scheduling method. We have implemented a prototype tool for the proposed SMT-based scheduling based on the system model, constraints formulation, and schedule synthesis described above. The underlying SMT solver employed by the tool is Z3. Note that, after the scheduling problem is formalized by first-order logic formulas, we can employ any reliable SMT solver to solve the scheduling problem. This means our method is not limited to a specific SMT solver. Here we choose Z3 rather than other SMT solvers (e.g., Yices, MathSAT [22], Zap [23]) is mainly because of our expertise and also because Z3 has been proved as an efficient reliable SMT solver [14].

4.1 Simulation Settings

The metrics used to evaluate the scheduling performance are: *i) success ratio*, which denotes the ratio of the input tasks that have been completed before their deadlines; and *ii) scheduling time*. For the baseline scheduling algorithms (SRTF, EDF, and LLF), as they are dynamic scheduling algorithms, which schedule tasks when system is running, the *scheduling time* represents the time that scheduling algorithms spent in scheduling all the input tasks at system runtime. For the proposed SMT-based scheduling method, as it is a static scheduling method, which generates scheduling table at design phase before system runtime, the *scheduling time* represents the time that SMT solver (Z3) spent in generating the scheduling table.

The input tasks are generated according to uniform distribution with arriving rate λ which represents the number of tasks that arrive in the system per 100 time units. For a fixed time interval, a system with a larger value of λ means more tasks arrive in the system during the time interval, which result in a heavier workload condition compared to a smaller value of λ. As the workload can be changed by λ, the attributes of tasks in our simulations are given a simple setting. For each task τ_i, c_i varies in [1 13]. The number of the indivisible fragments of a task varies in [1 3] The assignment of d_i is according to the equation: $d_i = r_i + sf_i * c_i$, where sf_i is the slack factor that indicates the tightness of task deadline. For each task τ_i, sf_i varies in [1 4]. Task dependency relation among tasks is randomly assigned.

In a well-defined system, usually the length of system overload time is not long, and the degree of system overload is not serious. If a system is under overload condition for a long time or the degree of the system overload is very serious, it means the capacity of the system is not enough to handle its work. Based on this observation, we set the values of λ as 14, 12, and 10 to represent different degrees of system overload conditions. The input total numbers of tasks are set as 100, 200, and 300 to represent different lengths of system overload time[†]. All the simulations are run on a 64bit 4-core 2.5 GHz Intel Xeon E3 PC with 32GB memory.

4.2 Evaluation

The simulation results are shown in Fig. 5 and Fig. 6. The values shown in the figures are the average value of running simulation 100 times. As the performance of the three baseline algorithms (SRTF, EDF, and LLF) are almost the same (the maximum differences among these three algorithms are within 1% in terms of both success ratio and scheduling time), for conciseness, we use one line *Baseline Alg.* to denote the performance of all the baseline algorithms in Fig. 5 and Fig. 6. The percentage numbers shown in the figure are the percentages of the performance improvement by using the SMT-based scheduling method compared to the baseline algorithms.

For success ratio, through the analysis in Sect. 3, the SMT-based scheduling can achieve the optimum result. As shown in Fig. 5, the values of success ratio for the SMT-based scheduling method are larger than the baseline algorithms under all the combinations of λ and total number of input tasks. In addition, from the Fig. 5, we can see that system with different values of λ have different workload conditions. When $\lambda = 10$, success ratio of the SMT-based

[†]For a specific value of λ, the overload time for system with 300 total number of input tasks is three times than system with 100 total number of input tasks.

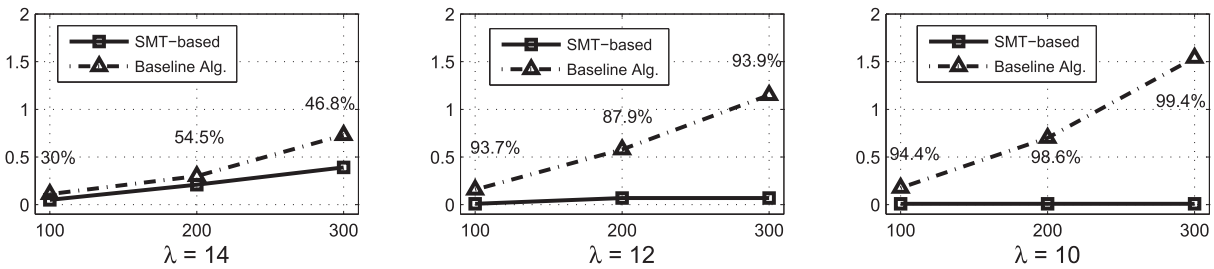

Fig. 6 Scheduling time (in second) of the SMT-based scheduling and the baseline algorithms (The x-axis is the total number of input tasks, and the y-axis is the scheduling time)

Table 2 Memory (megabytes) required by the SMT solver Z3 (The first row is total number of input tasks)

	100	200	300
$\lambda = 14$	2.9	3.1	3.3
$\lambda = 12$	2.9	3.1	3.3
$\lambda = 10$	2.9	3.1	3.2

scheduling is just little less than 1, which means system is under light overload condition as most of the tasks can meet their deadlines. While when $\lambda = 14$, success ratio of the SMT-based scheduling is less than 0.9, which means system is under more serious overload condition compared to the condition when $\lambda = 10$. These results are consistent with the explanation described in Sect. 4.1 and denote the validity of λ in changing the degrees of system overload conditions.

For scheduling time, the performance of the SMT-based scheduling method is also the best among all the methods. From Fig. 6, it can be seen that the values of scheduling time for the SMT-based scheduling are much smaller than the values for the baseline algorithms, and the improvements are quite obvious. This demonstrates the time efficiency of the SMT-based scheduling method. For completeness, we also give the amount of memory that is required by the SMT solver Z3 to generate the scheduling table. The results are shown in Table 2. From the table, we can see that the required memories for the SMT solver Z3 to generate the scheduling table is around 3.0 megabytes which is quite a small amount considering most of the current configurations of the systems. Moreover, from the table, we also can see that parameter λ almost does not affect the required memory, while total number of input tasks has a little affection on the required memory. This is mainly because the proposed SMT-based scheduling method generates scheduling table at design phase rather at system runtime.

To evaluate scheduling overheads of different scheduling methods, Fig. 7 shows the number of task preemption in each task. From the figure, we can see that the performance of the proposed SMT-based scheduling is almost the same as the baseline algorithms EDF and SRTF, while LLF performs a little better than other scheduling methods. This means the performance improvement (make more tasks completed before their deadlines) of the SMT-based scheduling method does not incur additional overheads.

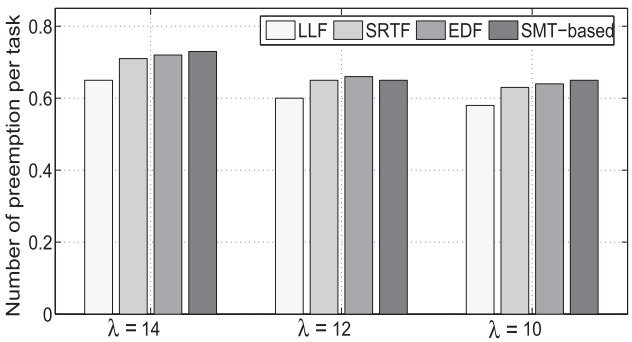

Fig. 7 Number of preemption per task

5. Adaption for Other Scheduling Targets

As mentioned in Sect. 3, in the SMT-based scheduling method, there are two kinds of scheduling constraints: system constraints and target constraints. This kind of design makes the proposed SMT-based scheduling method flexible and sufficiently general for other design objectives by just modifying the target constraints. In this section, we give two examples on modifying the target constraints to apply the SMT-based scheduling method to other design objectives.

5.1 Maximizing Effective Processor Utilization

Effective Processor Utilization (EPU) measures the fraction of time that the processor spends on executing tasks which are successfully completed before their deadlines. EPU may be a reasonable measure in situations where customers pay at a uniform rate for the use of the processor, but are billed only if tasks given by the customers are successfully completed [5].

5.1.1 Constraint on Scheduling Target

A successfully completed task τ_i should be completed before its deadline. As mentioned in Sect. 3.2.5, since all the fragments consisted in a task run sequentially, this constraint can be interpreted as: the last fragment of a successfully completed task should be completed before its deadline. Let symbol e be effective processor time (the time that the processor spends on executing tasks which are successfully

Algorithm 2 Schedule Synthesis for Maximizing Effective Processor Utilization

Input: task set $\mathcal{T}$
Output: schedule $\mathcal{S}$
1: $\mathcal{A} :=$ `Assert`$(\mathcal{T}, 1)$
2: $\mathcal{M} :=$ `CallSMTSolver`$(\mathcal{A})$
3: **if** $\mathcal{M} \neq \emptyset$ **then**
4: **return** $\mathcal{S}$ based on model $\mathcal{M}$
5: **end if**
6: $start := 0, end := 1$
7: **while** `true` **do**
8: $mid := start + (end - start)/2$
9: $\mathcal{A} :=$ `Assert`$(\mathcal{T}, mid)$
10: $\mathcal{M} :=$ `CallSMTSolver`$(\mathcal{A})$
11: **if** $\mathcal{M} = \emptyset$ **then**
12: $end := mid - step$
13: **else**
14: $\mathcal{A}' :=$ `Assert`$(\mathcal{T}, mid + step)$
15: $\mathcal{M}' :=$ `CallSMTSolver`$(\mathcal{A}')$
16: **if** $\mathcal{M}' \neq \emptyset$ **then**
17: $start := mid + step$
18: **else**
19: **return** $\mathcal{S}$ based on model $\mathcal{M}$
20: **end if**
21: **end if**
22: **end while**

completed before their deadlines), and its initial value is set to be 0.

$$\forall \tau_i \in \mathcal{T}$$
$$\texttt{if } (s_{i,e} + c_{i,e} \leq d_i)$$
$$e := e + c_i$$
$$\texttt{end}$$

Let symbol $ss_{i,e}$ denote the maximum value of $s_{i,e}$ for tasks in $\mathcal{T}$ that have been successfully completed. The effective processor utilization, epu, can be calculated as:

$$epu = \frac{e}{ss_{i,e} + c_{i,e}}$$

Let symbol $sepu$ denote the maximum value of effective processor utilization, and obviously, $sepu \in [0 \;\; 1]$. The constraints on scheduling target can be expressed as:

$$epu \geq sepu$$

5.1.2 Schedule Synthesis

The process of schedule synthesis is summarized in Alg. 2. This process of schedule synthesis is quite similar as the process described in Alg. 1. Function `Assert`$(\mathcal{T}, 1)$ (line 1) interprets the system constraints defined in Sect. 3.2 and target constraints descried in Sect. 5.1.1 as assertions with $sepu := 1$ in *constraint on scheduling target*. If this expectation can be satisfied, which means overload problem does not happen, model $\mathcal{M}$ will be returned by function `CallSMTSolver`$(\mathcal{A})$ (line 2). Based on $\mathcal{M}$, the schedule $\mathcal{S}$ can be generated (line 4).

When overload problem happens, $sepu := 1$ cannot be satisfied. We need to decrease the setting value of $sepu$. We use binary search to find the maximum value of $sepu$ (line 6–22). With the maximum value of $sepu$, a solution model can be returned by function `CallSMTSolver`$(\mathcal{A})$. Meanwhile, with $sepu := sepu + step$, `CallSMTSolver`$(\mathcal{A})$ will return an empty model. This is the criterion to judge if the value of $sepu$ is the maximum value. Note that, the variable $step$ is predefined and can be used to control the search space of SMT solver. Increasing $step$ makes the algorithm faster but also reduce the solution space.

Through the procedure of adapting the SMT-based scheduling method to the scheduling target *maximizing effective processor utilization*, we can see that only little modification on *i) target constrain* and *ii) schedule synthesis* needs to be made, while the system constraints defined in Sect. 3.2 can be total reused. This demonstrates the flexibility of our proposed SMT-based scheduling method. In the next subsection, we further show this advantage by adapting the proposed method to another scheduling target *maximizing obtained values of completed tasks*.

5.2 Maximizing Obtained Values of Completed Tasks

When tasks are equally important, the values obtained by the system through completing different tasks are the same. Thus, in this case, the design objective maximizing obtained values of completed tasks is the same as the design objective maximizing total number of tasks that meet deadlines. However, when tasks are with different degrees of importance, these two objectives become different.

For system within tasks that are with different degrees of importance, each task τ_i is a 4-tuple $\tau_i = (r_i, c_i, d_i, v_i)$, where v_i is the value of task τ_i that can be obtained by the system when τ_i is successfully completed.

5.2.1 Constraint on Scheduling Target

Let symbol v be the obtained values of the completed tasks, and its initial value is set to be 0.

$$\forall \tau_i \in \mathcal{T}$$
$$\texttt{if } (s_{i,e} + c_{i,e} \leq d_i)$$
$$v := v + v_i$$
$$\texttt{end}$$

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

$$v \geq sv$$

5.2.2 Schedule Synthesis

Quite similar as Alg. 1 and Alg. 2, we can get the scheduling synthesis for design objective maximizing obtained values of completed tasks. For conciseness, we omit it.

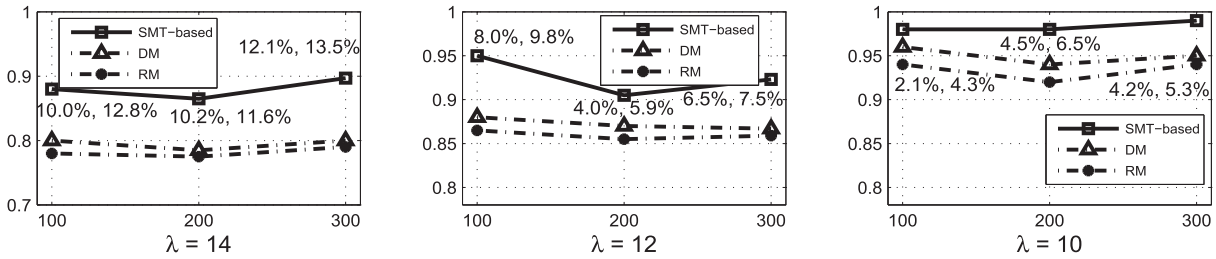

Fig. 8 Success ratio of the SMT-based scheduling and the static priority based scheduling algorithms: DM and RM (The x-axis is the total number of input jobs, and the y-axis is the success ratio)

6. Discussion

6.1 Extension for Other Critical Resources

When designing scheduling for real-time systems, processor is the mainly considered computation resources in this paper. Actually, when other critical resources (e.g., printer) are considered, the proposed SMT-based scheduling method can be easily applied. To achieve this, we only need to add a scheduling constraint when constructing the SAT model. The constraint can be expressed as:

$$\forall \tau_i, \tau_j \in \mathcal{T}, i \neq j, \forall f_a \in \tau_i, \forall f_b \in \tau_j, \forall r_\alpha \in \mathcal{R}$$
$$(s^\alpha_{i,a} \geq s^\alpha_{j,b} + tc^\alpha_{j,b}) \vee (s^\alpha_{j,b} \geq s^\alpha_{i,a} + tc^\alpha_{i,a})$$

Symbol $s^\alpha_{i,a}$ represents the time instant at which fragment $f_{i,a}$ requests the resources $r_\alpha \in \mathcal{R}$, where $\mathcal{R}$ is the set of critical resources. Symbol $tc^\alpha_{i,a}$ means the time slots that fragment $f_{i,a}$ needs to occupy the resources r_α. By adding this scheduling constraint, we can make sure that no multiple tasks access to a critical resource at the same time.

From this example, we can see that, for a newly encountered scheduling problem (e.g., considering other critical resources), through modifying or adding scheduling constraints, developers can easily solve the problem by applying the proposed SMT-based scheduling method.

6.2 Compared to Static Priority Based Scheduling

Currently, static priority based scheduling algorithms (e.g., RM (rate monotonic), DM (deadline monotonic)) are widely adopted in many real-time systems. Such kind of scheduling algorithms is applied to the periodic or sporadic task model [10]. In both models, tasks give rise to a potentially infinite sequence of executions (usually called *jobs*). In the periodic task model, the jobs of a task arrive strictly periodically, separated by a fixed time interval. In the sporadic task model, each job of a task may arrive at any time once a minimum inter-arrival time has elapsed since the arrival of the previous job of the same task. Under static priority based scheduling algorithms, different jobs of a task are assigned as the same priority.

In the task model studied in this paper, a task is executed only once. The proposed SMT-based scheduling method only requires the parameters of the tasks in the system are known a priori. This requirement makes the task model applied to the SMT-based scheduling method can be treated as a generalization of the periodic task model. As in periodic task model, all the parameters of the tasks are known a priori and the jobs of a task arrive periodically, while our proposed SMT-based scheduling method does not require jobs of a task arrive periodically. Because of this, static priority based scheduling algorithms cannot be applied to the task model adopted in the previous simulation study. Meanwhile, when they are applied to the periodic task model, the proposed SMT-based scheduling method can also be applied. However, when they are applied to the sporadic task model, the SMT-based scheduling method cannot be applied, as in sporadic task model, the arriving time of a job cannot be known a priori.

In order to give a performance comparison of the SMT-based scheduling method and the static priority based scheduling algorithms (RM, DM), in this subsection, we conduct simulations by using randomly-generated periodic task set. The scheduling target, maximizing total number of tasks that meet deadlines, is chosen as an example. In the periodic task set, each task τ_i gives rise to an execution (called *job*[†]) every p_i time units, where p_i is the period of task τ_i and varies in [30 200]. The settings of task parameters, c_i, sf_i and d_i, are the same as the setting used in Sect. 4. Similarly, the input total numbers of jobs are set as 100, 200, and 300 to represent different lengths of system overload time, and the values of λ are set as 14, 12, and 10 to represent different degrees of system overload conditions.

The simulation results are shown in Fig. 8, Fig. 9, and Fig. 10. For success ratio, as shown in Fig. 8, the SMT-based scheduling method performs better than DM and RM under all the combinations of λ and total number of input tasks. The percentage numbers shown in the figure are the percentages of the performance improvement by using the SMT-based scheduling method compared to DM and RM, respectively. This result is consist with our previous analysis that the SMT-based scheduling can achieve the optimum result in terms of success ratio.

For scheduling time, the performance of the SMT-based scheduling method is also the best among all the

[†]*job* can be analogous to *task* used in the previous simulation study described in Sect. 4.

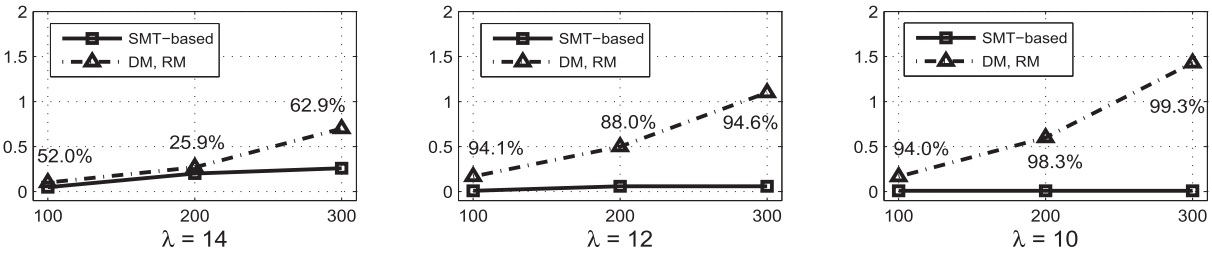

Fig. 9 Scheduling time (in second) of the SMT-based scheduling and the static priority based scheduling algorithms: DM and RM (The x-axis is the total number of input jobs, and the y-axis is the scheduling time)

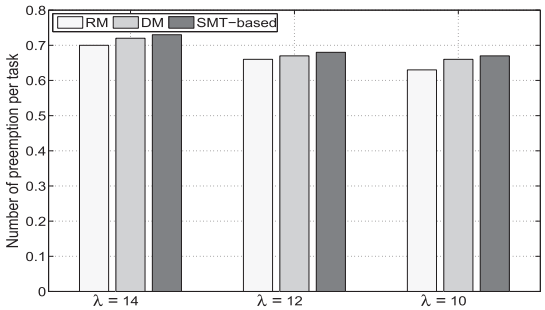

Fig. 10 Number of preemption per job

scheduling methods. From Fig. 9, it can be seen that the improvements are quite obvious. Note that, as the results for DM and RM are almost the same, for conciseness, we use one line *DM, RM* to denote the performance of scheduling algorithms DM and RM in Fig. 9. This result is also consist with the results that obtained from our previous simulation study described in Sect. 4.

Figure 10 shows the number of preemption per job. We can see that the performance of the proposed SMT-based scheduling is almost the same as DM and RM, which means the performance improvement by using the SMT-based scheduling method does not incur much additional overheads. For the amount of memory that is required by the SMT solver Z3 to generate the scheduling table, the simulation results are almost the same as the results shown in Table 2. For conciseness, we omit it.

7. Related Work

Research on scheduling has lasted for decades. Numerous researches have been conducted. There are mainly two categories of works which are mostly related to our research.

7.1 Scheduling for Overloaded Real-Time Systems

In the literature on real-time systems, several scheduling algorithms have been proposed to deal with the overload problem. A scheduling algorithm called DMB (dynamic misses based) was proposed in [8]. It is capable of dynamically changing the importance of tasks for adjusting their timing faults rate (ratio of tasks that missed deadlines). The main goal of DMB is to allow the prediction of timing faults during system overload. In [9], the problem of selecting tasks for rejection in an overloaded system is considered. Random criticality values are assigned to tasks. The goal is to schedule all the critical tasks and make sure that the weight of rejected non-critical tasks is minimized. These methods cannot apply to systems in which tasks are equally important. Compared to these works, our method can apply to both kinds of systems in which tasks are equally important or with different degrees of importance.

Some approaches focus on providing less stringent guarantees for temporal constraints. The elastic task model (ETM) proposed in [6] aims at increasing task periods to handle overload in adaptive real-time control systems. In ETM, periodic tasks are able to change their execution rate to provide different qualities of service. Authors in [7] introduced skippable tasks which are allowed to miss deadlines occasionally. Each task is assigned to a skip parameter which represents the tolerance of this task to miss deadline. A scheduling algorithm was proposed to adjust the system workload such that tasks adhere to their timing and skip constraints. Compared to these works, the parameters of tasks in our system are set a priori, and the system workload is decided by outside environment. Thus, the methods of adjusting system workload or changing parameters of tasks are not suitable.

In [3], authors studied some special cases of overloaded systems. They impose certain constraints on the values of task attributes. For example, under a special case *equal to request times*, all tasks have the same request time. Compared to this work, our proposed SMT-based scheduling only requires the request times of all the tasks are known in advanced rather than having the same value, which means our method is much more practical than the methods studied in [3].

All of these works are specified on their design objectives and cannot be applied to other design targets. Compared to them, our proposed SMT-based scheduling method is flexible and sufficiently general to adapt to different scheduling targets.

7.2 Scheduling Based on SAT and SMT

Authors in [16] presented a SAT-based approach to address the problem of periodic task and message allocation for dis-

tributed real-time systems. Through a special SAT solver extension enhanced with real-time scheduling theory, the proposed approach is guaranteed to find an optimal allocation for realistic task systems. In [19], authors studied the problem of assigning speeds to resources serving distributed applications with delay, buffer and energy constraints. They proposed a approach by coupling a SAT solver with the background theory of Real-Time Calculus (RTC).

Authors in [17] proposed a SAT-based technique to optimize throughput of homogeneous synchronous dataflow graphs on multiprocessor platforms, in which maximum cycle ratio analysis is integrated with SAT solver. With similar ideas of the integrated approach used in [17], in [18], authors also targeted the problem of makespan optimization of task graph scheduling on multiprocessors. An integrated optimization framework was constructed with relevant backjumping and continued search operations that accelerates the design space exploration.

In [20], [21], authors applied SMT solvers to address the pipelined scheduling and mapping problem for synchronous dataflow (SDF) graphs on shared memory multicores with instantaneous inter-core communication. They considered both throughput and latency constraints simultaneously, but limited the cases of acyclic SDF graphs (i.e., the applications with no feedback loops).

All the works mentioned above try to address some scheduling problems based on SAT or SMT. However, all the methods proposed in these works cannot be applied to tackle the overload problem studied in this paper, as their studied problems are mainly for systems under normal workload conditions. Even though, it should be noticed that the methods collected and demonstrated in these works can be potentially referred when we extend the proposed SMT-based scheduling method to more complicated systems, such as distributed real-time systems.

8. Conclusion

In this paper, to solve the overload problem of real-time systems, a SMT-based scheduling method is proposed. In the proposed method, 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. The correctness of this method and the optimality of its generated schedule can be verified in a straightforward manner. The results of various simulations, demonstrated that the proposed SMT-based scheduling method is more time efficient compared to existing scheduling algorithms.

In the SMT-based scheduling method, we define the scheduling constraints as system constraints and target constraints. This means if we want to design scheduling to achieve other objectives, only the target constraints need to be modified. To demonstrate this advantage, we apply the SMT-based scheduling method to three different design objectives. Very little modification is needed when adapting the SMT-based scheduling method to different scheduling targets, which means the proposed SMT-based scheduling method is flexible and sufficiently general.

For the future work, a promising direction is to adapt the SMT-based scheduling method to more complicated systems, such as distributed real-time systems.

References

[1] Z. Cheng, H. Zhang, Y. Tan, and Y. Lim, "Scheduling overload for real-time systems using SMT solver," Proc. 17th IEEE/ACIS Int. Conf. on Software Eng., Artificial Intell., Networking and Parallel/Distributed Computing, Shanghai, China, pp.189–194, May 30 – June 1, 2016.
[2] F. Zhang and A. Burns, "Schedulability analysis for real-time systems with EDF scheduling," IEEE Trans. Comput., vol.58, no.9, pp.1250–1258, April 2009.
[3] S.K. Baruah, J. Haritsa, and N. Sharma, "On-line scheduling to maximize task completions," Proc. 15th IEEE Real-Time Syst. Symp., San Juan, Puerto Rico, pp.228–236, Dec. 1994.
[4] A. Burns, "Scheduling hard real-time systems: a review," Software Eng. J., vol.6, no.3, pp.116–128, May 1991.
[5] S.K. Baruah and J.R. Haritsa, "Scheduling for overload in real-time systems," IEEE Trans. Comput., vol.46, no.9, pp.1034–1039, Sept. 1997.
[6] G.C. Buttazzo, G. Lipari, and L. Abeni, "Elastic task model for adaptive rate control," Proc. 19th IEEE Real-Time Syst. Symp., Madrid, Spain, pp.286–295, Dec. 1998.
[7] A. Marchand and M. Chetto, "Dynamic scheduling of periodic skippable tasks in an overloaded real-time system," Proc. 6th IEEE/ACS Int. Conf. on Comput. Syst. and Applicat., Doha, Qatar, pp.456–464, April 2008.
[8] C. Tres, L.B. Becker, and E. Nett, "Real-time tasks scheduling with value control to predict timing faults during overload," Proc. 10th IEEE Int. Symp. on Object and Component-Oriented Real-Time Distributed Computing, Santorini Island, Greece, pp.354–358, May 2007.
[9] S.-I. Hwang, C.-M. Chen, and A.K. Agrawala, "Scheduling an overloaded real-time system," Proc. 15th IEEE Int. Phoenix Conf. on Comput. and Commun., Arizona, USA, pp.22–28, March 1996.
[10] 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.
[11] C.L. Liu and J.W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," J. ACM, vol.20, no.1, pp.40–61, Jan. 1973.
[12] C. Barrett, R. Sebastiani, R. Seshia, and C. Tinelli, "Satisfiability modulo theories," in Handbook of Satisfiability, vol.185, IOS Press, 2009.
[13] L.D. Moura and N. Bjørner, "Satisfiability Modulo Theories: An Appetizer," Formal Methods: Foundations and Applications, vol.5902, pp.23–26, 2009.
[14] L. Moura and N. Bjørner, "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.
[15] B. Dutertre, "Yices 2.2," Proc. 26th Int. Conf. on Comput. Aided Verification, Vienna, Austria, LNCS 8559, pp.737–744, Springer International Publishing, 2014.
[16] A. Metzner and C. Herde, "RTSAT - An optimal and efficient approach to the task allocation problem in distributed architectures," Proc. RTSS, pp.147–158, 2006.
[17] W. Liu, M. Yuan, X. He, Z. Gu, and X. Liu, "Efficient SAT-based mapping and scheduling of homogeneous synchronous data flow graphs for throughput optimization," Proc. RTSS, pp.492–504, 2008.

[18] W. Liu, Z. Gu, J. Xu, X. Wu, and Y. Ye, "Satisfiability modulo graph theory for task mapping and scheduling on multiprocessor systems," IEEE Trans. Parallel Distribution Systems, vol.22, no.8, pp.1382–1389, 2011.
[19] P. Kumar, D.B. Chokshi, and L. Thiele, "A satisfiability approach to speed assignment for distributed real-time systems," Proc. DATE, pp.749–754, 2013.
[20] P. Tendulkar, P. Poplavko, and O. Maler, "Strictly Periodic Scheduling of Acyclic Synchronous Dataflow Graphs using SMT Solvers," Verimag Research Report, pp.1–19, 2014.
[21] P. Tendulkar, Mapping and Scheduling on Multi-core Processors using SMT Solvers, Ph.D. Thesis, 2014.
[22] R. Bruttomesso, A. Cimatti, and et al., "The MathSAT 4 SMT Solver," Proc. CAV, pp.299–303, 2008.
[23] T. Ball, S.K. Lahiri, and M. Musuvathi, "Zap: Automated Theorem Proving for Software Analysis," Proc. LPAR, pp.2–22, 2005.

Zhuo Cheng holds B.E. and M.E. degree from Tianjin University, China, and received M.S. degree from Japan Advanced Institute of Science and Technology (JAIST) in 2013. He is currently a Ph.D. candidate in School of Information Science, JAIST. His research interests include real-time scheduling, and satisfiability modulo theories.

Haitao Zhang is presently a lecture in Lanzhou University, China. He received his Ph.D. degree from Japan Advanced Institute of Science and Technology (JAIST) in 2015. He received his M.S. degree from JAIST & Tianjin University (China) in 2012. His current research interests include model checking, software testing.

Yasuo Tan received his Ph.D. from Tokyo Institute of Technology in 1993. He joined JAIST as an assistant professor of the School of Information Science in 1993. He has been a professor since 1997. He is interested in Ubiquitous Computing Systems especially Home Networking Systems. He is a leader of Residential ICT SWG of New Generation Network Forum, a chairman of Green Grid Platform at Home Alliance, an advisory fellow of ECHONET Consortium, and a member of IEEE, ACM, IPSJ, IEICE, IEEJ, JSSST, and JNNS.

Yuto Lim received the B.Eng. and M. Inf. Tech. degrees from Universiti Malaysia Sarawak, Malaysia in 1998 and 2000, respectively. He received the Ph.D. degree in communications and computer engineering from Kyoto University in 2005. He was a visiting researcher at Fudan University, China. During 2005-2009, he was an expert researcher at NICT, Japan. Since 2009, he has been working at JAIST as an associate professor. He is a member of IEEE, IEICE, and IPSJ. His research interests include wireless sensor networks, home networks, wireless mesh networks, network coding, and CPS.