Scheduling Overload for Real-Time Systems using SMT Solver

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*—In a real-time system, tasks are required to be completed before their deadlines. Due to heavy workload, the system may be in overload condition under which some tasks may miss their deadlines. To alleviate the degrees of system performance degradation cased by the missed deadline tasks, the design of scheduling is crucial. Many design objectives can be considered. In this paper, we 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 solver such a satisfiability 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. The time efficiency of the proposed method is demonstrated through various simulations. To the best of our knowledge, it is the first time introducing SMT to solve overload problem in real-time scheduling domain.**

***Keywords*—real-time scheduling, SMT, overload, 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. 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 complete before their deadlines (i.e., 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 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 cased 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 help solve it [2]. 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 [3, 4] can be considered. For example, (*i*) maximizing effective processor utilization, (*ii*) maximizing obtained values of completed tasks, (*iii*) maximizing total number of tasks that meet deadlines. The first two objectives are frequently adopted and studied in literature (e.g., [5, 6, 7]). Compared with them, the last one is rarely studied, and it 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 [2]. Research on this objective is still at preliminary stage. This motivates our work. In this paper, we focus on designing scheduling for overloaded real-time systems with uniprocessor. Our objective is to maximize the total number of tasks that meet their deadlines. 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 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 [13], Yices [14]) 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. 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 with some well-known algorithms. To the best of our knowledge, it is the first time introducing SMT to solve overload problem in real-time scheduling domain.

ii) 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 (e.g., maximizing effective processor utilization), 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 (e.g., multiprocessor), only the system constraints need to be modified. This means the proposed method is flexible and sufficiently general.

The remainder of this paper is organized as follows. In

TABLE I. SYMBOLS AND DEFINITIONS

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

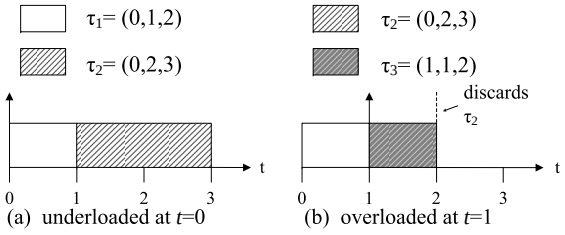

Fig. 1. Example for *underloaded* & *overloaded*

section II, we present the system model and give the research background. The details of the SMT-based scheduling are described in section III. Simulation and performance evaluation are shown in section IV. Section V summarizes the related works. Conclusions are given in section VI.

II. SYSTEM MODEL, DEFINITION & BACKGROUND

A. System Model

We adopt the general *firm-deadline* model proposed in [7]. 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 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[1]. $f_{i,j}$ denotes the j-th fragment of τ_i. For convenient, we use $f_{i,e}$ to denote the last fragment of task τ_i. 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_j in τ_i, and for $\forall f_j \in \tau_i, \sum c_{i,j} = c_i$. 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 $rc_i > d_i - t$. For convenience, symbols used throughout the paper are summarized in Table I.

Applied to this task model, we require all the parameters of the tasks are known a prior. This requirement makes the task 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 dealing with overload problem which are specified on period task model.

B. Definition

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

When there exists a schedule 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 can make all the tasks meet their deadlines, the system is* ***overloaded****, and the task set is* ***infeasible*** *(ref. [1]).*

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 includes 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 [10] 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, at $t = 1$. 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.

C. Background

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_4 and τ_2 sequentially.

[1]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.

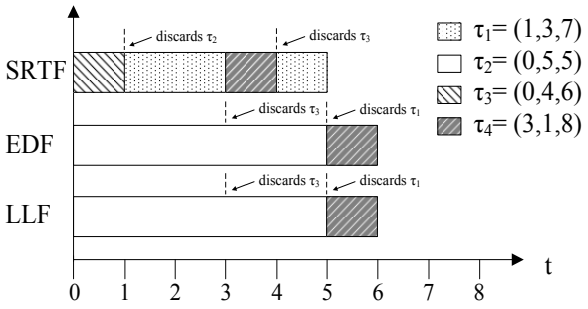

Fig. 2. Performance of scheduling algorithms

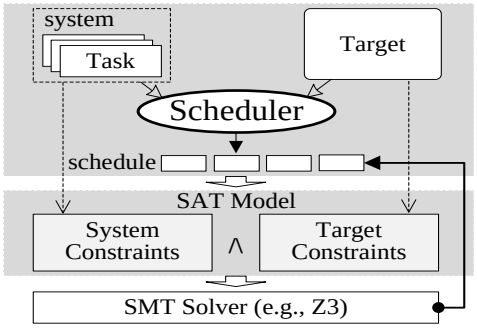

Fig. 3. Overview of the SMT-based scheduling method

(*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.

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 is proposed in section III.

2) 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 [11]. A first-order logic formula uses variables as well as quantifiers, functional and predicate symbols, and logic operators [12]. 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 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 $\exists a, b \in \mathbb{Z}$.

III. SMT-BASED SCHEDULING

A. Overview of the SMT-based Scheduling

The overview of the SMT-based scheduling is illustrated in 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 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 this paper, 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.

B. Scheduling Constraints

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

System Constraints

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.

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

2) Constraint on start time of different fragments: The series of fragments consisted in a task should be execute sequentially. Therefore, a fragment of a task can start to run only when its previous fragments of the task have been completed.

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

Algorithm 1 Schedule Synthesis

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

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 fragments of any different tasks.

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

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

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.

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

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

$$n = sn$$

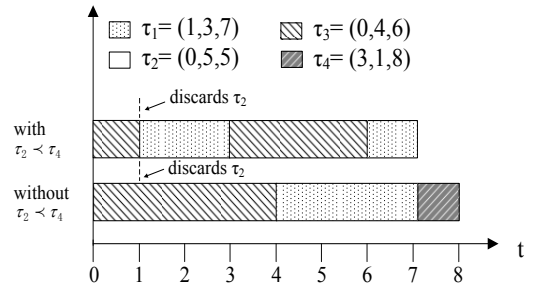

Fig. 4. Results by using the SMT-based scheduling method for the example shown in Fig. 2

C. 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 $\texttt{Assert}(\mathcal{T}, |\mathcal{T}|)$ (line 1) interprets the constraints defined in section III-B 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 $\texttt{CallSMTSolver}(\mathcal{A})$ (line 2) calls a SMT solver to find a solution model 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, 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 $\mathcal{S}$ 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 $\texttt{CallSMTSolver}(\mathcal{A})$, 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 $\texttt{CallSMTSolver}(\mathcal{A})$. Meanwhile, with $sn := sn+1$, $\texttt{CallSMTSolver}(\mathcal{A})$ 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 is 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

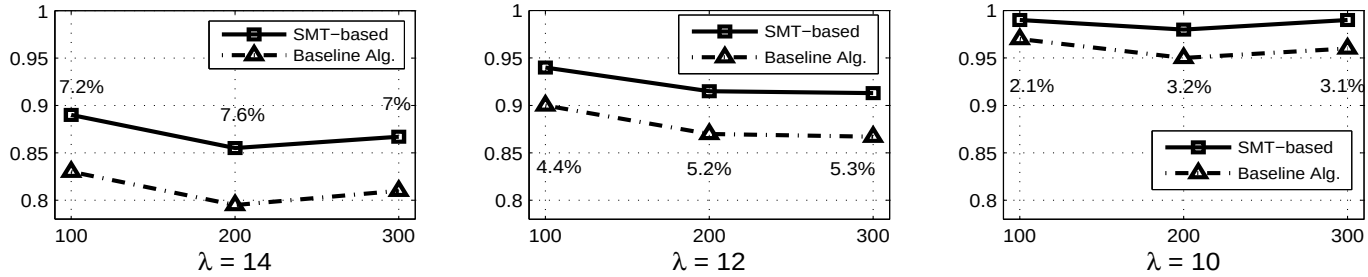

Fig. 5. Success ratio of the SMT-based scheduling and the baseline algorithms (The x-axis is the total number of input tasks, and the y-axis is the success ratio.)

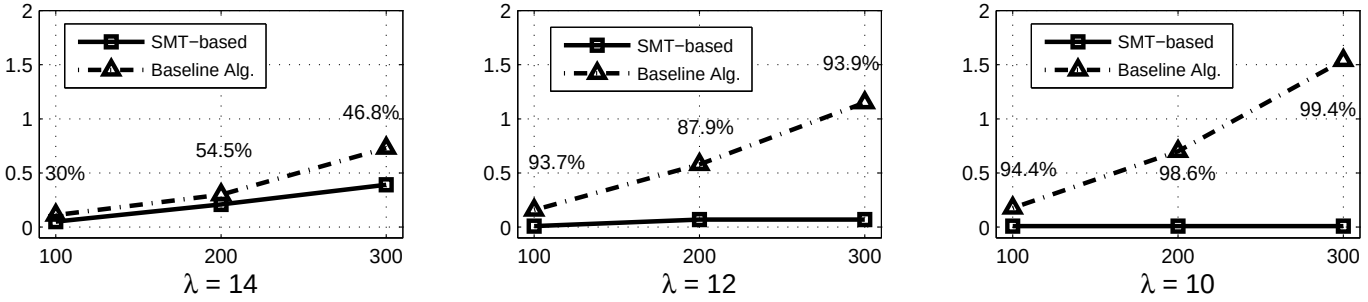

Fig. 6. Runtime (in second) of the SMT-based scheduling and the baseline algorithms (The x-axis is the total number of input tasks, and the y-axis is the simulation runtime.)

the schedule generated by the proposed SMT-based scheduling method.

D. 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: $s_{1,1} = 1, s_{1,2} = 2, s_{1,3} = 6, s_{2,1} = 7, s_{2,2} = 8, s_{2,3} = 9, s_{2,4} = 10, s_{2,5} = 11, s_{3,1} = 0, s_{3,2} = 3, s_{3,3} = 4, s_{3,4} = 5, s_{4,1} = 12$. 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 $\tau_2 \prec \tau_4$.

IV. Simulation & 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, which is a state-of-the art SMT solver. Three well known algorithms: SRTF, EDF, and LLF are adopted as the baseline algorithms.

A. Simulation Settings

The metrics used to evaluate the scheduling performance are: *i) success ratio*, which denotes the ratio of input tasks that have been completed before their deadlines; and *ii) simulation runtime*, which denotes the time of the corresponding scheduling methods scheduling all the input tasks. 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. 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 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 $[1\ 4]$.

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 number 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.

B. 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 are almost the same (their differences are within 1%) in terms of both the success ratio and the simulation runtime, we use the line *Baseline Alg.* to denote the baseline algorithms in Fig. 5 and Fig. 6. The percentage numbers shown in the figures are the percentage of the performance improvement by using the SMT-based scheduling method compared with the baseline algorithms.

For successful ratio, through the analysis in section III, the SMT-based scheduling can achieve the optimum result. As shown in Fig. 5, the values of success ratio for the SMT-based scheduling are larger than the baseline algorithms under all the combinations of λ and total number of input tasks. For simulation runtime, 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 value of simulation runtime for the SMT-based scheduling is much smaller than it for the baseline algorithms, and the improvement is quite obvious. This has demonstrated the time efficiency of the SMT-based scheduling method.

V. Related Work

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 of the critical tasks and make sure that the weight of rejected non-critical tasks is minimized. Compared to these works, we study the systems in which tasks are equally important. Therefore, the methods of scheduling tasks based on their importance cannot be applied.

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 priory, and the system workload is decided by outside environment. Thus, the methods of adjusting system workload or changing tasks' parameters are not suitable.

In [2], 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 tasks are known in advanced rather have the same value, which means our method is much more practical than the methods studied in [2].

VI. Conclusion

In this paper, to solve the overload problem of real-time systems, a SMT-based scheduling method is proposed. In the 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 are straightforward. Through various simulations, the simulation results demonstrate that the SMT-based scheduling method is more time efficient compared with the well-know baseline algorithms.

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] 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.

[3] A. Burns, "Scheduling hard real-time systems: a review," *Software Eng. J.*, vol. 6, no. 3, pp. 116–128, May 1991.

[4] 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.

[5] P. Mejia-Alvarez, R. Melhem, D. Mosse, and H. Aydin "An incremental server for scheduling overloaded real-time systems," *IEEE Trans. Comput.*, vol. 52, no. 10, pp. 1347–1361, Oct. 2003.

[6] G. 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, 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, Apr. 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. 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, Mar. 1996.

[10] 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.

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

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

[13] 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.

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