Deadline Verification for Web Services Using Timed Automata

Yamen El Touati
Faculty of Computing and IT
Northern Border University
Kingdom of Saudi Arabia
yamen.touati@nbu.edu.sa

Abstract—Many computation tasks are made today on remote cloud platforms using web services. Beyond the advantages provided by such services, many new challenges arise. One of the challenging problems is ensuring that web services respect critical deadlines. This is a critical issue, especially for real-time systems that use remote web services. This paper aims to propose a framework for deadline verification using Timed Automata (TA).

Keywords—web services; deadlines; timed automata; verification

I. INTRODUCTION
In the last few years, most companies migrated to remote cloud-based services. In these systems, most of the communication is ensured via web services that offer quasi-unlimited resources and computational power. However, companies face new challenges concerning security [1-4] and timing constraints [9-11, 19]. A web service is a software system designed to support interoperable machine-to-machine interaction over a network [6]. They provide flexible and efficient solutions to the sharing of information between people and businesses [6, 20]. By nature, a web service can face many random interferences that can cause delays in the required tasks [12-15, 18]. In many cases, it is not easy to check whether the specification deadlines are respected. In general, this is because some web services are composed of many correlated tasks, where delays and timing are propagated according to a dynamic schedule. In this particular context, timed automata [6-10] are suitable for modeling the evolution of a system focusing on time constraints. These are presented as an extension of finite-state automata with clocks into location. Transitions can be executed when a linear constraint on clocks, called guard, is verified. This framework presents an interesting graphical interface combined with powerful computational power. Moreover, it is more suitable to solve verification problems using a timed automata framework than performing algebraic calculus, which may be difficult to resolve if the size of the system increases. This paper proposes a new method based on timed automata reachability checking in order to resolve the problem of deadline verification.

II. BACKGROUND OF TIMED AUTOMATA AND TIME TRANSITION SYSTEMS

Corresponding author: Yamen El Touati

\( \mathbb{R}_+ \) denotes nonnegative reals, \( X=\{x_1, x_2, \ldots, x_n\} \) denotes the finite set of real-valued variables, and the circumflex (\(^\wedge\)) is used to denote an element of the set of operators \( \{\leq, \geq, \neq\} \). A simple inequality over \( X \) is an inequality of the form \( x-c \), where \( x \in X \) and \( c \in \mathbb{R}_+ \). A rectangular predicate over \( X \) is a conjunction of linear inequalities over \( X \). \( C(X) \) denotes the set of linear predicates on \( X \).

A. Timed Automata
A timed automaton [1,3] is a tuple \( A=(L, l_0, X, \Sigma, E, \text{inv}, F) \) where:

- \( L \) is a finite set of locations (nodes)
- \( l_0 \) is the initial state
- \( X \) is a finite set of non-negative real valued clocks
- \( \Sigma \) is a finite set of actions or events (labels)
- \( E \subseteq L \times C(X) \times \Sigma \times 2^{\mathbb{R}_+} \times L \) is a set of transitions. \( e=(l, \delta, a, R, l') \) is defined, where \( l \) is the start location, \( \delta \) is the guard, \( a \) is the action, \( R \) is the set of clocks to reset, and \( l' \) is the end location
- \( \text{inv} \in C(X) \) is a function that associates an invariant to each location
- \( F \subseteq L \) is the set of final states

B. Semantic of Timed Automata
The semantic of timed automata \( A=(L, l_0, X, \Sigma, E, \text{inv}) \) is given by a Time Transition System [2] \( SA=(Q, q_0, \rightarrow) \), where:

- \( Q=L \times \mathbb{R}^X_+ \)
- \( q_0=(l_0, 0) \) is the initial state
- \( \rightarrow \subseteq Q \times \Sigma \cup \mathbb{R}_+) \times Q \) is defined by:
  - \( (l,v) \rightarrow (l',v') \) (discrete transition) if \( \exists (l, \delta, a, R, \rho, l') \in E \) such that \( \delta(v)=true \), and \( v'=v[R \leftarrow 0][\rho] \) and \( \text{inv}(l')(v')=true \)
  - \( (l,v) \rightarrow (l',v') \) (continuous transition) if \( l=li \) and \( v=v+i \) and \( \text{inv}(l)(v)=true \)
It is noted that \((l,v)\rightarrow^*(l',v')\) in the case of a transition sequence of leading from \((l,v)\) to \((l',v')\).

C. Reachability in Timed automata

A run of a timed automaton \(A\) is a path in \(\Sigma\) starting from \(l_0\). \([\{A\}]\) denotes the set of all runs in the timed automaton \(A\). It is noted that \((l,v)\rightarrow^* (l',v')\rightarrow^* (l'',v'')\) is equivalent to \((l,v)\rightarrow^* (l''',v''')\). A state \((l,v)\) is considered as reachable, if:

\[
\exists (l_0, v_0) \rightarrow t^0_{a_0} (l_1, v_1) \rightarrow t^{i_1}_{a_1} (l_2, v_2) \rightarrow t^{i_2}_{a_2} \ldots \rightarrow t^{i_t}_{a_t} (l_t, v_t)
\]

where \((l_0, v_0) = q_0\). A run:

\[
(l_0, v_0) \rightarrow t^0_{a_0} (l_1, v_1) \rightarrow t^{i_1}_{a_1} (l_2, v_2) \rightarrow t^{i_2}_{a_2} \ldots \rightarrow t^{i_t}_{a_t} (l_t, v_t)
\]

starting from \(q_0 = (l_0, v_0)\) is a timed trace, denoted as:

\[
w = (a_0 = e_0, \delta_0) \rightarrow (a_2, \delta_2) \rightarrow (a_3, \delta_3) \rightarrow \ldots
\]

where \(w\) is a sequence of pairs \((a_i, \delta_i), a_i \in E\) is a transition, and \(\delta_{i+1} \in \mathbb{R}_+\) is the delay between the two successive events \(a_i\) and \(a_{i+1}\) where; \(\delta_0 = 0\) and \(\forall i \geq 1:\
\delta_i = t_i - t_{i-1}.

III. Deadline Verification in Web Services

A web service can be modeled by a set of tasks with a timed automaton, where each location corresponds to a specific configuration in the web service’s lifecycle. Transitions between locations correspond to a configuration change. Thus, locations can be considered as elementary subtasks to fulfill the entire web service. Some of these tasks may take a longer than expected time to be fulfilled. Since the sequence of tasks and their exact duration is not easy to tackle in these systems, it would be interesting to investigate a way to check whether any critical deadline was broken.

A. Checking Deadline in Timed Automata

Consider the example shown in Figure 1 and suppose that this timed automaton represents the behavior of a web service composed of 5 subtasks (those represented by locations L1 to L5). It can be assumed that the deadline of this web service is 16 time units (t.u). Checking whether this deadline is respected is equivalent to checking if L3 is reached always before 16 t.u. In this case, L3 is always reached before 8 t.u, which means that the previous deadline is respected by the web service. To resolve this problem, a universal clock \(x_0\) (a clock that is never reset) can be added and check if the constraint \(x_0 \leq 15\) is always verified.

B. Generalization

This problem can be generalized for a timed automaton \(A=(L, l_0, X, \Sigma, E, inv)\) as:

\[
\forall l \in L, \forall v \in \mathbb{R}_+, \text{Deadline} \in \mathbb{R}_+, \text{Deadline} \text{ is respected if:}
\]

\[
\forall l \in L, \forall v \in \mathbb{R}_+, \text{such that} (l_0, 0) \rightarrow^* (l, v), x(0) \leq \text{Deadline} \quad (1)
\]

C. Deadline Verification Steps

To ensure that property (1) is respected, this problem is transformed into a reachability verification problem as follows.

- A new derivative automaton is constructed \(A_d=(L, l_0, X, \Sigma, E, inv)\), such that:

\[
E_d=\{(l, x_0)\rightarrow^{\text{Deadline}}, \{d\}, \{l_0\}\}
\]

- The reachability of \(l_d\) should be checked. If \(l_d\) is reachable, then the deadline is not guaranteed to be respected by the web service.

The \(l_d\) location has the behavior of a typical dead state. Transitions are added from each nonfinal location to \(l_d\) with the guard \(x_0 > \text{Deadline}\), without any reset.

![Fig. 1. Example of a timed automaton.](image1)

![Fig. 2. Updated Timed Automaton with dead state and universal clock – \(l_d\) is not reachable.](image2)
to \( L_d \) will have the form \( t_0 > 6 \), as shown in Figure 3. In this case, the location \( L_d \) is reachable from any non-final location. In general, if the location \( L_d \) is accessible from even a single location, then the deadline constraint becomes unsatisfied.

For any web service modeled by a timed automaton, the problem of checking deadlines is decidable. By construction, checking the deadline respect on automaton \( A \) is equivalent to checking the reachability of \( L_d \) in automaton \( A^2 \). If a transition is executed in \( E_d \), then the universal clock \( x_0 \) exceeds the defined deadline, since the guard \( x_0 > \text{Deadline} \) becomes true. Furthermore, the reachability problem is known to be decidable for general timed automata [1], ensuring that the verification of the deadline is decidable. The proposed method to check a deadline is structural. It is sufficient to add a new \( L_d \) location, a new universal clock, and as many transitions as the cardinality of the locations set.

The proposed method was applied to the examples in Figures 2 and 3 using the Phaver tool [17]. Phaver allows the computation of reachable space of timed automata. The automata and the computation of reachable space were defined, conforming to the Phaver syntax, as shown in Figure 4. The result reachable space is shown in Figure 5.

### D. How to Respect Deadline Constraints

Some web services may not meet some deadline requirements. When the number of clocks/locations increases, it is difficult to determine what are the possible updates on the behavior of the system to converge to the requirements. It is interesting to find a solution that describes the required values of the clocks with respect to the deadline constraints by following the following steps:

1. **Mark all locations where \( L_d \) is reachable.** This set is denoted as \( L_d \).
2. **\( \forall l \in L_d \) update \( \text{inv}(l) \) by adding the constraint \( t_0 \leq \text{Deadline} \).**
3. **Foreach \( l \in L_d \), perform a backward analysis considering the reverse automata and the new location environment [16].**
4. **Merge all the reachable space.**

The result of these steps is a new automaton where the \( L_d \) location is never reachable. The implementability of these new constraints on the web service can be checked by the developers or designers based on the new constraints obtained on locations and guards. Moreover, it could be a starting point to enhance and upgrade the web service considering the new constraints.
IV. CONCLUSION

This paper focused on the problem of deadline verification for web services that could be modeled by timed automata. This framework offers the possibility to model the internal behavior of a web service with explicit consideration of timing constraints. The contribution of this paper is to propose a way to transform this problem into a reachability verification problem of a derived automaton. This new automaton is obtained by a structural and linear transformation. Future work consists of the generalization of this problem for a sequence of subtasks of a web service with multi-deadline constraints. Moreover, it is possible to tackle the deadline verification in the context of web service composition [4, 8]. In this case, it is necessary to deal with the synchronous composition of timed automata as modeling frameworks.

REFERENCES


www.etasr.com