Studying timed aspects for cloud configuration management tools: validation and recommendations for safe execution

Evgenii Vinarskii, Natalia Kushik, Djamal Zeghlache

Studying timed aspects for cloud configuration management tools: validation and recommendations for safe execution

Evgenii Vinarskii
SAMOVAR, Télécom SudParis,
Institut Polytechnique de Paris
Palaiseau, France
vinarskii.evgenii@telecom-sudparis.eu

Natalia Kushik
SAMOVAR, Télécom SudParis
Institut Polytechnique de Paris
Palaiseau, France
natalia.kushik@telecom-sudparis.eu

Djamal Zeghlache
SAMOVAR, Télécom SudParis
Institut Polytechnique de Paris
Palaiseau, France
djamal.zeghlache@telecom-sudparis.eu

***Abstract*—Tools for automated deployment and management are often used in the cloud computing platforms. Since this technology is rapidly evolving, it is important to guarantee the reliability and the efficiency of these tools. The paper proposes a framework *CMT-VSE* for validation of configuration management tools and for providing tenant requests with timed recommendations for their safe execution. Such recommendations can be utilized to execute the chains of commands in an asynchronous mode. Experimental evaluation with Ansible-2.16 and SaltStack-3006 showcases the efficiency of *CMT-VSE*.**

***Index Terms*—Cloud computing, validation, timed aspects, tenant request recommendations, safe execution**

I. Introduction & Background

Configuration management tools (CMTs) such as Ansible [1] and SaltStack [2] play a crucial role in executing chains of processes (workflows) on remote instances, e.g., virtual machines. Such processes can be categorized as *configuration processes* that modify VMs (e.g., open a port) and *action processes* that are executed without modifying configurations of VMs (e.g., send a message). These chains need to be well-written and well-synchronized for their safe execution on remote virtual machines. Let $C = \{c_1, \ldots, c_m\}$ denote a set of *positive* configuration processes that make configuration changes and $\overline{C} = \{\overline{c_1}, ..., \overline{c_m}\}$ be a set of *negative* ones that revert them; $C^+ = \{c_1^+, \ldots, c_m^+\}$ and $\overline{C^+} = \{\overline{c_1^+}, ..., \overline{c_m^+}\}$ denote the notifications of the termination of the corresponding configuration processes. Let $A = \{a_1, ..., a_n\}$ be a set of action processes, $A^+ = \{a_1^+, ..., a_n^+\}$ ($A^- = \{a_1^-, ..., a_n^-\}$) denote the notifications of correct (incorrect) executions of processes of A. Since the result of the execution depends on the current VM configuration, for every $a \in A$ there is a *positive*[1] *pre-post condition* $\varphi := C' \Rightarrow a$ where $C' \subseteq C$; φ means that a can be terminated correctly (a^+) if and only if i) the related VMs are configured in a way that a can be executed (C'), and ii) a appears in the corresponding tenant request. For example, $\varphi_{send} := (open, allow) \Rightarrow send$ means that a message will be delivered to $vm2$ if and only if i) $vm2$ is configured in a way that the port is open, the traffic from $vm1$ is allowed, and ii) $vm1$ sends the message to $vm2$.

The work was supported by CCMC2 BPI project No DOS0185474/00.

[1]We assume that positive pre-post conditions describe the system behavior that should be respected.

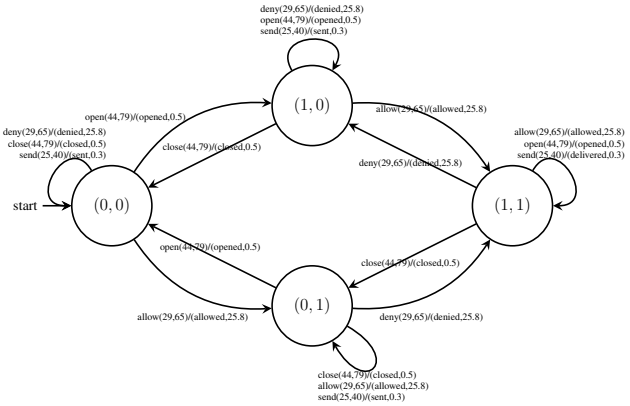

Fig. 1: TFSM $\mathcal{A}$

CMTs implement processes sequentially on remote instances to configure an *instance configuration* (installed software packages, firewall settings, ...) preserving pre-post conditions. However, some processes can be time-consuming, therefore for optimization purposes, CMTs allow to run processes asynchronously. This feature introduces the risk of the violation of pre-post conditions caused by concurrency issues among the processes. Ansible and SaltStack do not provide explicit guidelines for setting timed parameters to prevent process conflicts. The latter motivates to provide a tenant request (a trace of configuration and action processes) with recommendations on how to correctly set timed parameters to avoid potential conflicts.

CMT is a real-time sequential reactive system whose behavior can be described by a Timed Finite State Machine (TFSM) [3]. The TFSM behavior depends on the current state, the timestamps of inputs, and the processing time for each input. A transition $s \xrightarrow{i,(u,v)/(o,d)} s'$ takes place if and only if input i is applied within timed guard (interval) (u, v). Output

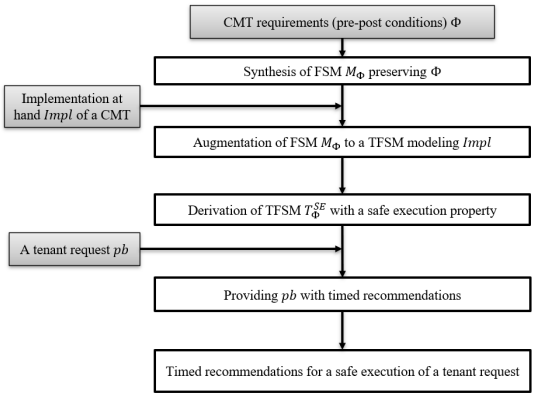

Fig. 2: *CMT-VSE* overview

delay d indicates the specified time needed to produce output o after input i is applied. In this case, the TFSM contains implicit concurrent procedures that can execute the next input even if the output for the previous one is still pending[2].

Consider TFSM $\mathcal{A}$ (see [3] for details) shown in Fig. 1 that describes the behavior of Ansible-2.16 (see [4]), $pb = open, allow, send$ and its timed extensions $pb_t^1 = (open, 44.1), (allow, 73.2), (send, 98.3)$ and $pb_t^2 = (open, 44.1), (allow, 73.2), (send, 98.8)$ of pb. The output reaction of $\mathcal{A}$ for pb_t^1 is $(opened, 44.6), (delivered, 98.6), (allowed, 99.0)$ where $delivered$ is produced before $allowed$, i.e., φ_{send} is violated. Otherwise, the output reaction of $\mathcal{A}$ for pb_t^2 is $(opened, 44.6), (allowed, 99.0), (delivered, 99.1)$, i.e., φ_{send} is preserved. Thus, the TFSM allows to provide a timed extension for a trace of processes to guarantee that the pre-post conditions are preserved. In this paper, we introduce *CMT-VSE* – a framework based on TFSM theory for validation of CMTs and for providing tenant requests with timed recommendations for their safe execution.

II. *CMT-VSE* FRAMEWORK

CMT-VSE takes three inputs: i) CMT requirements Φ given as pre-post conditions that describe the expected behavior of a CMT, ii) an implementation at hand $Impl$ of a CMT, and iii) a tenant request pb; as an output, *CMT-VSE* returns recommended timeouts pb_t for its safe execution (see Fig. 2). In this section, we just briefly discuss the main modules of *CMT-VSE*, their detailed description can be found in [4].

A. Synthesis of an FSM preserving CMT requirements

Let $\varphi_a := C_a \Rightarrow a$ be a pre-post condition, $I = C \cup \overline{C} \cup A$ and $O = C^+ \cup \overline{C^+} \cup A^+ \cup A^-$ be a set of processes and notifications of their terminations (see Section I). We say that an FSM $\mathcal{A}$ *preserves* φ_a *for an input/output trace* β/γ over I^*/O^* if for every appearance of input/output pair a/a^+ in β/γ the following holds: i) $c \prec_\beta a$ for every configuration process $c \in C_a$, and ii) if there exists $\overline{c} \in \overline{C_a}$ such that $\overline{c} \prec_\beta a$, then $\overline{c} \prec_\beta c \prec_\beta a$. Consequently, $\mathcal{A}$ *preserves* Φ if $\mathcal{A}$ preserves every φ of Φ for every input/output trace.

The synthesis module of *CMT-VSE* takes pre-post conditions Φ and synthesizes the transition graph of FSM $\mathcal{M}_\Phi = (S, I, O, h_S, s_0)$ *preserving* Φ. In order to do this, *CMT-VSE* utilizes a breadth-first search algorithm. States represent possible configurations of VMs, namely, a state is a Boolean vector $[is\_c_1, \ldots, is\_c_m]$ that indicates which configuration processes have been executed and have not been reverted. The initial state is $[is\_c_1 \leftarrow false, \ldots, is\_c_m \leftarrow false]$. *CMT-VSE* starts at the initial state, and at every step takes an unvisited state and builds transitions from that state in the following way: i) the next state for every positive configuration process $c \in C$ is $is\_c \leftarrow true$ and the transition is labeled by pair c/c^+, ii) for every negative configuration process $\overline{c} \in \overline{C}$, it is $is\_c \leftarrow false$ and the transition is labeled by pair $\overline{c}/\overline{c^+}$, iii) for every action process $a \in A$ with pre-post condition $\varphi_a := C_a \Rightarrow a$ if at the current state φ_a is preserved, the transition is labeled by pair a/a^+, otherwise the transition is labeled by pair a/a^-. The following theorem holds.

Theorem II.1. *Let C and A be sets of configuration and action processes and Φ be a set of pre-post conditions over C and A, CMT-VSE derives FSM $\mathcal{M}_\Phi = (S, I, O, h_S, s_0)$ preserving Φ, where $|S| \leq 2^{|C|}$.*

B. FSM augmentation with timed parameters

To augment FSM $\mathcal{M}_\Phi$, *CMT-VSE* works with the direct simulation of an implementation $Impl$ at hand. Given FSM $\mathcal{M}_\Phi$ preserving pre-post conditions Φ, *CMT-VSE* extends $\mathcal{M}_\Phi$ to TFSM $\mathcal{T}$ using the following algorithm. At the first step, timed parameters (timed guards and delays) for every transition are estimated. Namely, let $tran = (s, i, o, s')$ be a transition of $\mathcal{A}$, and a trace $\alpha' = i_1, \ldots, i_k$ bring $\mathcal{A}$ to state s, pb_α defines a tenant request for a CMT of interest for trace $\alpha = \alpha', i$. *CMT-VSE* runs pb_α $N > 0$ times to compute timed intervals (u, v) and (p, q), where (u, v) denotes the fluctuation for the timestamp of process i, while (p, q) denotes the fluctuation for the response time for command i. We assign (u, v) as a timed guard of $tran$, since our purpose is to satisfy pre-post conditions; further, we assign a delay for transition $tran$ in the following way. If i is a configuration process, then the output delay for $tran$ is equal to the right boundary of the interval, i.e., $d \leftarrow q$, otherwise $d \leftarrow p$.

TFSM $\mathcal{T}$ does not necessarily preserve pre-post conditions Φ (see Section II-C). At the second step, we derive TFSM $\mathcal{T}_\Phi^{SE}$ by modifying timed guards (u, v) for transitions of $\mathcal{T}$ to guarantee a *safe execution* property. The *safe execution* means that if for input trace pb FSM $\mathcal{M}_\Phi$ preserves pre-post conditions Φ, then there exists a timed extension pb_t of pb such that $\mathcal{T}_\Phi^{SE}$ preserves Φ for pb_t. In order to transform TFSM $\mathcal{T}$ to $\mathcal{T}_\Phi^{SE}$, *CMT-VSE* extends the right boundaries of every transition labeled by a positive configuration or an action process (see details in [4]). Thus, TFSM $\mathcal{T}_\Phi^{SE}$ which models the implementation of a CMT at hand and guarantees a safe execution property is derived.

[2]The number L of these procedures, if finite, is limited by the ratio of the maximal output delay of the TFSM to the minimal left boundary [3].

C. Timed recommendations for a tenant request

Let β be an input trace of $\mathcal{A}$ and β_t be a timed extension of β such that β_t/γ_t is a timed input/output trace of $\mathcal{T}$. The order of outputs in γ_t can be not the same as the order of the corresponding inputs in β, see for example the output reaction for pb_t^1 in Section I. Therefore, even if an FSM preserves pre-post conditions for an untimed input trace, TFSM augmented with timed parameters can violate them for a timed input trace. The main purpose of this module is to provide a tenant request with the timed recommendations for its safe execution w.r.t. TFSM $\mathcal{T}_\Phi^{SE}$. We propose a procedure $Providing\_timed\_recommendation$ that takes an untimed trace $pb = i_1, \ldots, i_k$ preserving Φ and returns the timed extension $pb_t = (i_1, t_1), \ldots, (i_k, t_k)$ preserving Φ w.r.t. the timed parameters of $\mathcal{T}_\Phi^{SE}$. To assign timestamps, the procedure works according to the following algorithm. Let L be the number of procedures of $\mathcal{T}_\Phi^{SE}$, ε is chosen as $\frac{1}{2L}$. Since $\mathcal{T}_\Phi^{SE}$ extends $\mathcal{M}_\Phi$, $\mathcal{T}_\Phi^{SE}$ has the sequence of transitions $s_0 \xrightarrow{i_1,(u_1,v_1)/(o_1,d_1)} s_1 \ldots s_{k-1} \xrightarrow{i_k,(u_k,v_k)/(o_k,d_k)} s_k$. The procedure assigns $t_1 \leftarrow u_1 + \varepsilon$ and t_j, $j \in \{2, \ldots, k\}$, is chosen in the following way.

1) If $i_j \in C$ and there exists $\ell \in \{1, \ldots, j-1\}$ such that $i_\ell = \overline{i_j}$ and $t_\ell + d_\ell > t_{j-1} + u_j + d_j$, then $t_j \leftarrow t_\ell + d_\ell - d_j + \varepsilon$; otherwise $t_j \leftarrow t_{j-1} + u_j + \varepsilon$[3].
2) If $i_j \in A$ and there exists $\ell \in \{1, \ldots, j-1\}$ such that $i_\ell \in C$ and $t_\ell + d_\ell > t_{j-1} + u_j + d_j$, then $t_j \leftarrow max(t_1 + d_1, \ldots, t_{j-1} + d_{j-1}) - d_j + \varepsilon$; otherwise $t_j \leftarrow t_{j-1} + u_j + \varepsilon$[4].

Theorem II.2. *Given* $pb = i_1, \ldots, i_k$, $\mathcal{M}_\Phi$ *preserves* Φ *for timed input trace* $pb_t = Providing\_timed\_recommendation(pb)$.

Corollary 1. *Given* $pb = i_1, \ldots, i_k$, *the complexity of* $Providing\_timed\_recommendation$ *is* $O(k^2)$.

As an example we consider $pb = open, allow, send$ and TFSM shown in Fig. 1, correspondingly. As mentioned in Section I, $pb_t^1 = (open, 44.1), (allow, 73.2), (send, 98.3)$ does not preserve $\varphi_{send} := (open, allow) \Rightarrow send$. However, if we choose $\varepsilon = 0.1$, since $73.2 + 25.8 < 99.1$, we conclude that $pb_t^2 = Providing\_timed\_recommendation(pb) = (open, 44.1), (allow, 73.2), (send, 98.8)$ preserves φ_{send}.

III. Evaluation & Conclusion

We have experimentally evaluated the performance of *CMT-VSE* with Ansible-2.16 and SaltStack-3006 (run for OpenStack 2023.2 [5]). The goal of our experimental evaluation is to estimate a *CMT-VSE* performance by comparing the average execution time for *naive* and *CMT-VSE*-based strategies. The former applies a next command of a tenant request only if all previous commands of the request have been already finished. The latter relies on the timeouts recommended by *CMT-VSE* and thus, the next command is executed at a given timestamp suggested by the framework. Let $\varphi := (vm2\_open, vm1\_allow\_sub2) \Rightarrow vm1\_send\_vm2$ be a pre-post condition, the derived FSM preserving φ can be found in [4]. We estimated the average execution time for *naive* and *CMT-VSE*-based strategies for input traces with length $10, 20, \ldots, 500$. The execution of the *CMT-VSE*-based strategy is on average 1.24 and 1.86 times faster for Ansible-2.16 and SaltStack-3006 than the *naive* one (see Fig. 3) [4].

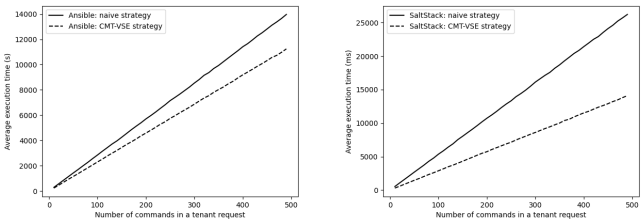

Fig. 3: Performance evaluation of *CMT-VSE*

One of the ways to analyze the obtained experimental results is to compare *CMT-VSE* with scheduling approaches [6], [7] for assigning proper timeouts to tenant requests. Due to potential network channel instability, there can be significant fluctuations for the delivery response time, for each task. Moreover, the scheduling problem is NP-hard [8]. In TFSM-based approach, to handle the complexity, *CMT-VSE* uses a preprocessing to estimate timed parameters, and further takes a quadratic time w.r.t. the length of the tenant request to provide recommendations. The efficiency is showcased by the experiments, even if performed for rather simple pre-post conditions when building the CMT specifications. We recall that *CMT-VSE* relies on simulating an implementation of a CMT at hand to derive timed parameters for transitions, another way to extract such parameters is based on the theoretical analysis of the devices involved; we plan to develop such an approach in the future.

[3]Therefore, $t_j + d_j \geq t_\ell + d_\ell - d_j + d_j + \varepsilon > t_\ell + d_\ell$.

[4]Therefore, $t_j + d_j \geq max(t_1 + d_1, \ldots, t_{j-1} + d_{j-1}) - d_j + d_j + \varepsilon > max(t_1 + d_1, \ldots, t_{j-1} + d_{j-1})$.

References

[1] Ansible-2.16. https://github.com/ansible/ansible, Accessed 2024.
[2] Saltstack-3006. https://github.com/saltstack/salt, Accessed 2024.
[3] Evgenii M. Vinarskii, Natalia Kushik, Nina Yevtushenko, Jorge López, and Djamal Zeghlache. Timed transition tour for race detection in distributed systems. In *Proceedings of the 18th, ENASE 2023*, pages 613–620, 2023.
[4] E. Vinarskii. Cmt-vse. https://github.com/vinevg1996/CMT-VSE, Accessed 2024.
[5] Openstack-2023.2. https://github.com/openstack, Accessed 2024.
[6] Said Nabi, Muhammad Aleem, Masroor Ahmed, Muhammad Arshad Islam, and Muhammad Azhar Iqbal. RADL: a resource and deadline-aware dynamic load-balancer for cloud tasks. *J. Supercomput.*, 78:14231–14265, 2022.
[7] Chandrashekar C., Krishnadoss P., K. Poornachary, B. Ananthakrishnan, and K. Rangasamy. Hwacoa scheduler: Hybrid weighted ant colony optimization algorithm for task scheduling in cloud computing. *Appl. Sci*, 33, 2023.
[8] Jelke J. van Hoorn, Agustín Nogueira, Ignacio Ojea, and Joaquim A. S. Gromicho. An corrigendum on the paper: Solving the job-shop scheduling problem optimally by dynamic programming. *Comput. Oper. Res.*, 78:381, 2017.