On Decomposing Formal Verification of CTL-based Properties on IaaS Cloud Environment

Chams Eddine Choucha[1,3][a], Mohamed Ramdani[1,3][b], Mohamed Khalgui[1][c] and Laid Kahloul[2][d]
[1] National Institute of Applied Sciences and Technology, University of Carthage, Tunis, Tunisia
[2] Computer Science Department, Biskra University, Biskra, Algeria
[3] University of Tunis El Manar, Tunis, Tunisia

Keywords: Discrete-event System, Reconfiguration, R-TNCES, Computation Tree Logic, CTL, Cloud Computing, Formal Verification.

Abstract: This paper deals with reconfigurable discrete event/control systems (RDECSs) that dynamically change their structures due to external changes in environment or user requirements. RDECSs are complex and critical. The verification of these systems continues to challenge experts in both academia and industry since the generated state spaces are much bigger and the properties to be verified are more complex. Reconfigurable Timed Net Condition/Event Systems (R-TNCESs) are proposed as an extension of the Petri nets formalism for the optimal functional and temporal specification of RDECSs. Real systems model can encompass millions of transitions which, implies huge state spaces and complex properties to be verified. To control the complexity and to reduce the verification time, a new method of CTL properties verification in a cloud-based architecture is proposed. The novelty consists in a new method for state space generation and the decomposition of the complex properties for running an efficient verification. An algorithm is proposed for the incremental state space generation. A case study is exploited to illustrate the impact of using this technique. The current results show the benefits of the paper's contribution.

1 INTRODUCTION

Nowadays, Reconfigurable discrete event control systems are invading international markets in several domains (e.g., medical, aerospace or smart grids) (Hafidi et al., 2019) (Naidji et al., 2019). Any failure of such systems will cause disastrous consequences. Formal verification is, therefore, imperative. RDECSs have flexible configurations that allow them to switch from a configuration to another due to user requirements or to prevent system malfunctions (Hafidi et al., 2018). This verification consists of two major steps: state-space generation and state-space analysis. Mentioned steps applications are usually expensive in terms of computation time and memory occupation (i.e., huge accessibility graph to be generated and complex properties to be verified) (Zhang et al., 2015). The authors in (Ramdani et al., 2018) proposed to classify properties automatically and to introduce a priority order during RDECSs verification to control the high number of properties to be verified. The mentioned method improves verification by reducing the number of properties to be verified by exploiting relationships among properties (equivalence, composition and dominance). However, when the property relationship rate is low which is frequent while verifying complex RDECSs, the said method is equivalent to the classic ones. The authors in (Hafidi et al., 2018) proposed a method for accessibility graph generation with less computing time and less required memory, while preserving the graph semantics. They start by computing the initial TNCES accessibility graph classically, then making updates on it to compute the remaining TNCESs accessibility graphs, while considering similarities between them. Previous methods improve classical ones. However, with large scale systems, their application using a unique machine (i.e., a centralized system) may be expensive in terms of time and calculation (Koubâa et al., 2017). Authors in (Camilli et al., 2014) initiate the cloud-based solution for formal method problems.

[a] https://orcid.org/0000-0003-0194-4890
[b] https://orcid.org/0000-0002-8723-5827
[c] https://orcid.org/0000-0001-6311-3588
[d] https://orcid.org/0000-0002-9739-7715

Choucha, C., Ramdani, M., Khalgui, M. and Kahloul, L.
On Decomposing Formal Verification of CTL-based Properties on IaaS Cloud Environment.
DOI: 10.5220/0009972605440551
In *Proceedings of the 15th International Conference on Software Technologies (ICSOFT 2020)*, pages 544-551
ISBN: 978-989-758-443-5

Authors have proposed a distributed fixed-point algorithm to check CTL properties with basic operators. The said algorithm can analyze DECS efficiently. However, RDECSs complexity forced us to move forward with big data solutions for formal method problems. To cope with RDECSs, Petri nets has been extended and developed by several works (Padberg and Kahloul, 2018). Reconfigurable Timed Net Condition/Event System (R-TNCES) is a novel formalism proposed in (Zhang et al., 2013a), where reconfiguration and time properties with modular specification are provided in the same formalism. This Paper deals with RDECSs modeled by R-TNCES. In our previous works, we propose a method for state space generation which, considers similarities that an R-TNCES can contain, thanks to an ontology-based history. Also, we proposed in (Choucha et al., 2019) to perform CTL properties verification in a parallel way on a cloud-based architecture while considering relationships among properties. The said methods are efficient; However, the first work is only focused on state space generation, and the second one presents limits when properties are complex and properties relationship rate is low. Therefore, we propose in this paper a new work that comes to fill the limits of precedent ones. Hence, we proposed a new method that aims to improve R-TNCES formal verification. RDECSs formal verification may be expensive in terms of computation power and memory occupation, therefore, we resort to a cloud-based solution to increase computation power (resp. memory occupation) thank to the Infrastructure as a service IaaS (reps. Simple Storage Service S3) proposed by Amzon (Hayes, 2008). To control RDECSs formal verification complexity we propose the following contributions:

1. Incremental state space generation to facilitate the access to different parts of the accessibility graph; Certain properties do not require the entire exploration of the accessibility graph in order to be validated or not, therefore, a partial exploration of the accessibility graph is sufficient. Indeed, we introduce the modularity concept to the state-space generation step, which allows us to access different parts of the accessibility graph (modules). This contribution allows us to proceed to a targeted verification.
2. Decomposition of CTL properties to control complexity during the state-space analysis. Due to the RDECSs complexity, properties to be verified in order to ensure the correctness of the system behavior are more complex. Thereby, increasing the complexity of the analysis step. In order to fix the mentioned issue, we check the possibility of decomposition of the complex properties into several simple or less complex properties that can be verified in less computation time using fewer resources.
3. Development of a distributed cloud-based architecture to perform parallel computations during formal verification and to store large scale data. The huge generated state spaces and the high number of properties to be verified forced us to opt for a big data solution to control the complexity of RDECSs formal verification. Computation tasks are ensured by the master and the workers via virtual machines allocated thanks to the EC2 product proposed by amazon. Data storage is ensured by S3.

The main objective of this paper is to propose a new formal verification method that improves the classical ones by controlling complexity. As a running example, a formal case study is provided to demonstrate the relevance of our contributions. The obtained results are compared with different works. The comparison shows that the verification is improved in terms of execution time (i.e., less complexity to perform RDECSs formal verification).

The remainder of the paper is organized as follows. Section 2 presents some required concepts. The distributed formal verification is presented in Section 3. Section 4 presents the evaluation of the proposed method. Finally, Section 5 concludes this paper and gives an overview about our future work.

2 BACKGROUND

In this section, we present basic concepts which are required to follow the rest of the paper.

2.1 Reconfigurable Timed Net Condition/Event System

R-TNCES is a modeling formalism used to specify and verify reconfigurable discrete event control systems (RDECSs). R-TNCES is based on Petri nets and control components CCs. A control component (CC) is defined as a software unit. Control components are applied as a formal model of the controller of a physical process and are modeled by TNCES as shown in Figure 1. Each CC resumes the physical process in three actions: Activation, working and termination. An R-TNCES RTN is defined in (Zhang et al., 2013b) as a couple $RTN = (B,R)$, where R is the control module and B is the behavior module. B

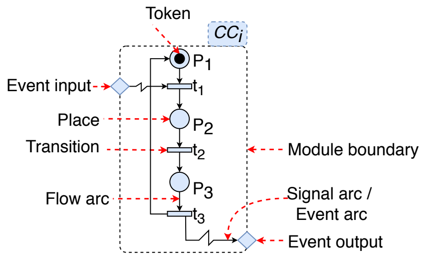


Figure 1: Graphical model of a generic control component modeled by TNCES.

is a union of multi TNCES-based CC modules, represented by

$$B = (P;T;F;W;CN;EN;DC;V;Z_0)$$

where,

a) P (resp, T) is a superset of places (resp, transitions),

b) F is a superset of flow arcs,

c) W: $(P \times T) \cup (T \times P) \rightarrow \{0,1\}$ maps a weight to a flow arc, $W(x,y) > 0$ if $(x,y) \in F$, and $W(x,y) = 0$ otherwise, where $x,y \in P \cup T$,

d) CN (resp, EN) is a superset of condition signals (resp, event signals),

e) DC is a superset of time constraints on input arcs of transitions,

f) $V: T \rightarrow \wedge, \vee$ maps an event-processing mode for every transition;

g) $Z_0 = (M_0, D_0)$, where M_0 is the initial marking, and D_0 is the initial clock position.

R is a set of reconfiguration functions $R = \{r_1, ..., r_n\}$. r is structured as follow: $r = (Cond, s, x)$ such that: 1. $Cond \rightarrow$ {true, false} is the pre-condition of r, which means specific external instructions, gusty component failures, or the arrival of certain states. 2. $s: TN(^*r) \rightarrow TN(r^*)$ such that $TN(^*r)$(resp. $TN(r^*)$) be the original (resp. target) TNCES before (resp. after) r application is the structure modification instruction. 3. $x: last_{state}(TN(^*r)) \rightarrow initial_{state}(r^*)$ is the state processing function, where $last_{state}(TN(^*r))$ (resp. $initial_{state}(TN(r^*))$) is the last (resp. the initial) state of $TN(^*r)$ (resp. $TN(r^*)$).

2.2 Computation Tree Logic CTL

Computational tree logic CTL is a temporal logic for branching-time based on propositional logic used by (Baier and Katoen, 2008) for model checking. CTL can describe the context and branching of the system state, it models system evaluation as a tree-like structure where each state can evolve in several ways (i.e., specify behavior systems from an assigned state in which the formula is evaluated by taking paths). CTL has a two-stage syntax where formulae in CTL are classified into state and path formulae. The former is formed according to the following grammar:

$$\Phi ::= true | AP | \Phi_1 \wedge \Phi_2 | \Phi_1 \vee \Phi_2 | \neg\Phi | E\varphi | A\varphi$$

While path formulae which express temporal properties of paths are formed according to the following grammar:

$$\varphi ::= X\Phi | F\Phi | G\Phi | \Phi_1 U \Phi_2$$

where Φ, Φ_1 and Φ_2 are state formulae. AP is the set of atomic propositions. The CTL syntax include several operators for describing temporal properties of systems: A (for all paths), E (there is a path), X(at the next state), F (in future), G (always) and U (until).

Definition 1. Equivalence of CTL Formulae: CTL formulae σ_1 and σ_2 (over AP) are called equivalent, denoted $\sigma_1 \equiv \sigma_2$ whenever they are semantically identical. Therefore, $\sigma_1 \equiv \sigma_2$ if$Sat(\sigma_1) = Sat(\sigma_2)$ for all transition systems TS over AP such that Sat(σ)=$\{s \in S | s \models \sigma\}$. Table 1 presents an important set of equivalences rules (expansion and distributive laws).

Table 1: Some equivalence rules for CTL.

expansion laws:
$EG\phi \equiv \phi \wedge EXEG\phi$
$AF\phi \equiv \phi \vee AXAF\phi$
$EF\phi \equiv \phi \vee EXEF\phi$
$A[\phi U \psi] \equiv \psi \vee (\phi \wedge AXA[\phi U \psi])$
$E[\phi U \psi] \equiv \psi \vee (\phi \wedge EXE[\phi U \psi])$
distributive laws:
$AG(\sigma_1 \wedge \sigma_2) \equiv AG\sigma_1 \wedge AG\sigma_2$
$EF(\sigma_1 \vee \sigma_2) \equiv EF\sigma_1 \vee EF\sigma_2$

2.3 Infrastructure as a Service IaaS

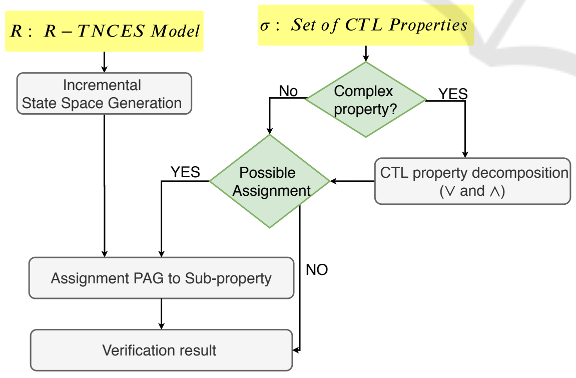
Cloud computing is an increasingly popular paradigm for ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources. In practice, cloud service providers tend to offer services that can be grouped into three categories as follows: (i) software as a service, (ii) platform as a service, and (iii) infrastructure as a service. IaaS is defined by (Hayes, 2008) as web service that provides provision processing, storage, networks, administrative services needed to store applications and a platform for running applications. It is designed to make web-scale cloud computing easier for developers. Amazon Web Services Elastic Compute Cloud (EC2) and Secure Storage Service (S3) are examples of IaaS offerings (Hayes, 2008).

3 DISTRIBUTED CLOUD BASED FORMAL VERIFICATION

We present in this section the proposed distributed cloud-based formal verification of R-TNCESs.

3.1 Motivation

R-TNCES is an expressive formalism, which allows considering different aspects of RDECS (time, probability, reconfigurability and concurrency) (Housseyni et al., 2017). The correctness of RDECSs modeled by R-TNCES can be ensured by formal verification. However, such a formalism makes the verification process complex, due to the combinatorial growth of the state space according to the model size, and due to the high number and complexity of the properties that the designer wants to verify. Thus, we aim to make model checking more efficient by reducing the time validation of properties to be verified. Therefore, we propose a new method for R-TNCES verification, which facilitates both generation and analysis of state space. To ensure our objective, we implement different tasks as follows: a) Incremental state space generation, which is to construct the state space by part, b) the complex properties are decomposed to simple or less complex ones, then c) if possible, we assign a partial graph to the property to be verified. d) Finally, we proceed to properties verification. Figure 2 presents scheduling of the presented tasks.


$R:\ R-TNCES\ Model$
$\sigma:\ Set\ of\ CTL\ Properties$
Incremental State Space Generation
Complex property?
No
YES
Possible Assignment
YES
CTL property decomposition (∨ and ∧)
NO
Assignment PAG to Sub-property
Verification result
Figure 2: Global idea for the formal verification according to the distributed cloud based verification.

3.2 Formalization

In this section, we present formal verification steps according to the distributed cloud-based formal verification of R-TNCESs.

3.2.1 Incremental State Space Generation

Incremental state space generation consists of generating accessibility graphs by part, while preserving models semantics. Let $RTN(R,B)$ be an R-TNCES model, this task consists in two steps:

(i) Basic accessibility graph generation BAG, which consists of generating accessibility graphs for each $CC_i \in TNCES_j$ where, $i \in 0 \ldots NumberCC(TNCES_j)$ and $j \in 0 \ldots NumberTN(B)$. This step is implemented in algorithm 1. It takes an R-TNCESs as input and proceed to BAGs generation through several function including $Generate\_State\_Space(CC)$ which, take a CC modeled by TNCES and return its accessibility graph using SESA tool (Patil et al., 2012).

(ii) Partial accessibility graphs (PAGs) composition: This step is implemented in Algorithm 2. It consists of composing pair of graphs computed during the first step (BAGs) and throughout iterations of the second step (PAGs), mainly by using the function $Compose(AG,AG)$ that takes two graphs and composes them and returns a new composed graph.

Algorithm 1: Basic accessibility generation.

```
Input: $RTN$: R-TNCES; $TN_0$: TNCES;
Output: $S\_BAG$: Set of elementary accessibility graphs;
for $int\ i = 0\ to\ |\sum TN|$ do
    for $each\ CC \in TN$ do
        if ( !Tagged (CC)) then
            $Insert(S\_BAG, Generate\_State\_Space(CC))$;
            tag(CC);
        end
    end
end
return $S\_BAG$
```

Algorithm 2: Accessibility graph construction.

```
Input: $S\_BAG$: Set of Elementary accessibility graphs ; $\sum CChain$: Set of $Cchains$;
Output: $S\_AG$: Set Accessibility graphs;
for $int\ i = 0\ to\ |\sum CChain|$ do
    $AG \leftarrow BAG_{CC_i^0}$;
    for $int\ j = 0\ to\ |\sum CC_i|$ do
        $AG \leftarrow Compose(AG, BAG_{CC_i^j})$;
    end
    $Insert(S\_AG, AG)$
end
return $S\_AG$
```

3.2.2 Complex CTL Properties Decomposition

We assume that properties that contain the operators ($\wedge$) or ($\vee$) are complex. Two kinds of complex properties are distinguished as follows:

- Decomposable: The operators ($\wedge$) or ($\vee$) are not linked to factors (State operators or path quantifiers). This kind of properties are directly splited into a set of sub-properties (e.g., $\Phi = P_1 \wedge P_2$ gives $\sigma_1 = P_1$ and $\sigma_2 = P_2$).
- Non-decomposable: The operators ($\wedge$) or ($\vee$) are linked to factors. For this kind of properties, we firstly applied expansion or distribution laws and then re-check if they are decomposable or not.

Figure 3 shows the majors steps of complex CTL properties decomposition task.

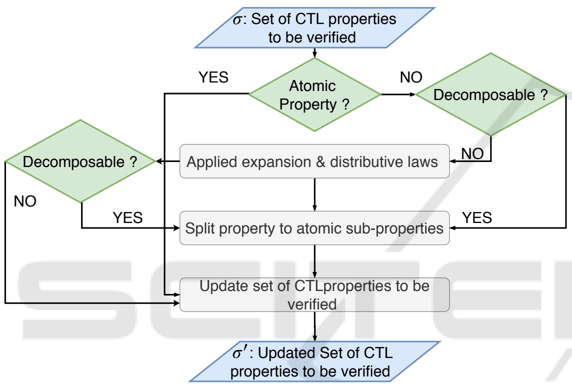

σ: Set of CTL properties to be verified
YES
Atomic Property ?
NO
Decomposable ?
Decomposable ?
Applied expansion & distributive laws
NO
NO
YES
Split property to atomic sub-properties
YES
Update set of CTLproperties to be verified
σ': Updated Set of CTL properties to be verified

Figure 3: Complex CTL properties decomposition.

3.2.3 Properties Assignment to PAGs

We assign to each property one or several state spaces computed during incremental state space generation. The assignment is done based on two criteria: (i) Path quantifier and state operators, and (ii) places concerned by the property such that we assign the smallest state space that contains the concerned places.

3.3 Distributed Architecture for Formal Verification

In this subsection, we present the proposed distributed cloud-based architectures shown in Figure 4. The idea that motivates the development of this architecture is to increase computation power and storage availability. It is composed of computational and storage resources. To develop the architecture shown in Figure 4, we use IaaS to allocate the following resources:

- Computation resources: which represent the master that coordinates the executed tasks, and the workers that execute the presented tasks above.

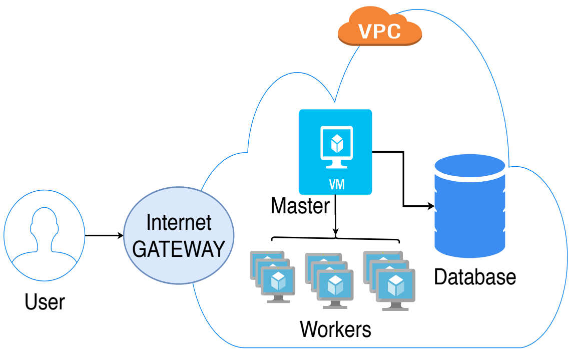


Figure 4: Distributed cloud-based architecture.

- Storage resources: represents the allocated cloud database that stores accessibility graphs computed during verification.

4 EXPERIMENTATION

In this section, to validate and demonstrate the gain of our proposed contributions, we use a formal case study.

4.1 Case Study

To demonstrate the performance and the gain of the proposed contribution, we use an R-TNCES to model a sequential system S_{01}, to be denoted by $RTN_{S_{01}}(B_{S_{01}}, R_{S_{01}})$. S_{01} is composed of 11 physical processes modeled by 11 CCs. The behavior module of the system ($B_{S_{01}}$) is modeled graphically as shown in Figure 5. This model covers three configurations (C_1,C_2,C_3). It is assumed that every configuration has one control chain ($CChain_i$) as follows.

- $CChain_1 : CC_1, CC_2, CC_3, CC_9$.
- $CChain_2 : CC_1, CC_2, CC_3, CC_4, CC_5, CC_{11}, CC_7, CC_8$.
- $CChain_3 : CC_1, CC_2, CC_3, CC_{10}, CC_5, CC_6, CC_7, CC_8$.

This behavior module can be reconfigured automatically and timely between the three configurations ($C_i, i = 1,..,3$), according to the environment changes or to the user requirements. $RTN_{S_{01}}$ can apply six different reconfiguration scenarios according to the control module $R_{S_{01}}$, which are described as follows: $R_{S_{01}} = (C_1, C_2); (C_1, C_3); (C_2, C_1); (C_2, C_3); (C_3, C_1); (C_3, C_2)$.

4.2 Application

In this section, we present the application of the formal verification of $RTN_{S_{01}}$ according to the cloud-based formal verification.

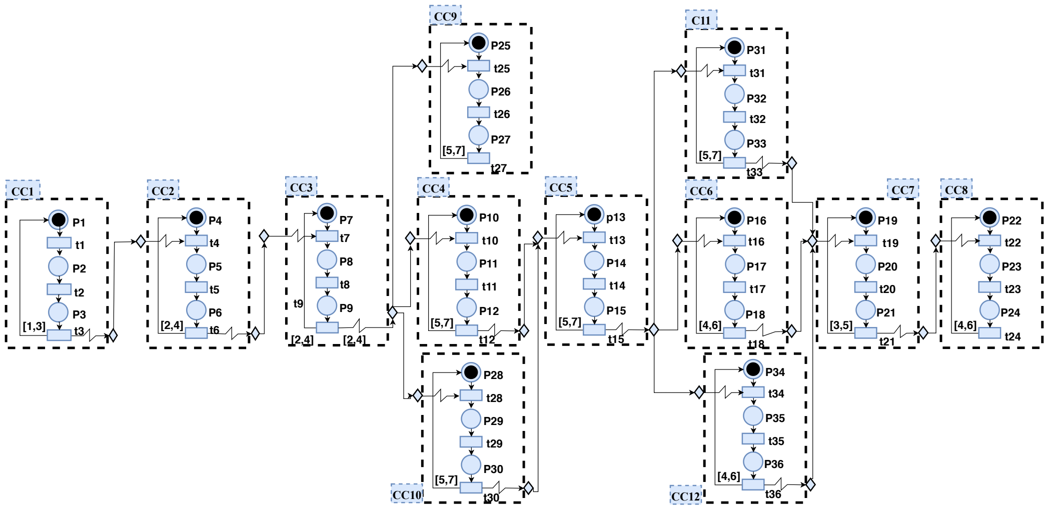


Figure 5: Behavior model with three configurations process.

Table 2: Incremental state space generation.

R-TNCES Model	Control Chains	PAGs
$RTN_{S_{01}}$	$CChain_1$	$(CC_1,CC_2);(CC_1,CC_2,CC_3);(CC_1,CC_2,CC_3,CC_9)$.
	$CChain_2$	(CC_1,CC_2,CC_3,CC_4), $(CC_1,CC_2,CC_3,CC_4,CC_5)$, $(CC_1,CC_2,CC_3,CC_4,CC_5,CC_{11})$, $(CC_1,CC_2,CC_3,CC_4,CC_5,CC_{11},CC_7)$, $(CC_1,CC_2,CC_3,CC_4,CC_5,CC_{11},CC_7,CC_8)$.
	$CChain_3$	(CC_1,CC_2,CC_3,CC_{10}), $(CC_1,CC_2,CC_3,CC_{10},CC_5)$, $(CC_1,CC_2,CC_3,CC_{10},CC_5,CC_6)$, $(CC_1,CC_2,CC_3,CC_4,CC_5,CC_6,CC_7)$, $(CC_1,CC_2,CC_3,CC_4,CC_5,CC_6,CC_7,CC_8)$.

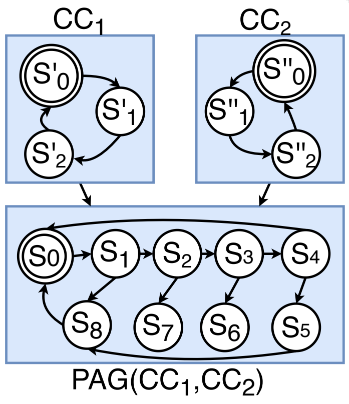


Figure 6: Example of composition of two accessibility graphs.

4.2.1 Incremental State Space Generation

In order to generates $RTN_{S_{01}}$ accessibility graph $AG_{RTN_{S_{01}}}$, we apply algorithms 1 and 2. First, we generates accessibility graphs for each physical process, which are denoted by (BAG_i, $i =_{1,..,11}$). Then, we proceed to successive pair graphs compositions until we constitute $AG_{RTN_{S_{01}}}$. Table 2 shows PAGs computed during the first step of the system verification. note that each computed PAG is stored in the cloud database. Moreover, Figure 6 shows an example of a pair graphs composition.

Table 3: Set of CTL properties to be verified.

σ: Set of CTL Properties
$P_1 : EF(p_3)$, $P_2 : AF(p_9)$,
$P_3 : AF(p_{15})$, $P_4 : AF(p_{21})$,
$P_5 : AF(p_{24})$, $P_6 : AF(p_{17})$,
$P_7 : AF(p_{32})$, $P_8 : AF(p_{35})$.
$P_9 : EF(p_{12} \wedge EG(p_{24}))$,
$P_{11} : EG(p_{12} \wedge EGp_{35}))$,
$P_{12} : EG(p_{12} \wedge EG(p_{33}))$,
$P_{13} : \neg EF(p_{27} \wedge EG(p_{24}))$,
$P_{14} : EF(p_{12}) \wedge EF(p_{18})$,
$P_{15} : AF(p_{12}) \wedge EG(p_{33}) \wedge EG(p_{21}) \wedge AF(p_{24})$,
$P_{16} : AF(p_{12}) \wedge EG(p_{33}) \wedge EG(p_{21}) \wedge AF(p_{24})$.

4.2.2 Decomposition and CTL Properties Assignment to PAGs

In order to validate the basic behavior of the system and to guarantee that system model satisfies the good requirements, we must ensure the CTL functional properties. In particular, to ensure: a) The safety, the system allows only one process to be executed at any time, i.e., no activation of two CCs from two different configurations at the same time, b) the liveness, whenever any process wants to change the configuration, it will eventually be allowed to do so, and c) the non-blocking, any active CC is eventually ended.

Table 3 presents the above mentioned properties

Table 4: CTL properties decomposition and assignment.

σ: Set of CTL Properties	Decomposition	Assignment
$P_1 : EF(p_3)$, $P_2 : AF(p_9)$, $P_3 : AF(p_{15})$, $P_4 : AF(p_{21})$, $P_5 : AF(p_{24})$, $P_6 : AF(p_{17})$, $P_7 : AF(p_{32})$, $P_8 : AF(p_{35})$.	Non-decomposable	$P_1 : EAG_1$, $P_2 : CC_1, CC_2, CC_3$, $P_3 : CC_1, .., CC_5$, $P_4 : CC_1, .., CC_7$, $P_5 : CC_1, .., CC_8$, $P_6 : CC_, .., CC_6$, $P_7 : CC_, .., CC_{11}$.
$P_9 : EF(p_{12} \wedge EG(p_{24}))$, $P_{12} : EG(p_{12} \wedge EG(p_{33}))$, $P_{13} : \neg EF(p_{27} \wedge EG(p_{24}))$,	Non-decomposable	$P_9 : CC_1, .., CC_8$, $P_{12} : CC_1, .., CC_{11}$, $P_{13} : CC_1, .., CC_8$.
$P_{14} : EF(p_{12}) \wedge EF(p_{18})$.	$P'_{14} : EF(p_{12})$. $P''_{14} : EF(p_{18})$.	$P'_{14} : CC_1, .., CC_4$ $P''_{14} : CC_1, .., CC_6$.
$P_{15} : AF((p_{12}) \wedge EG((p_{33})) \wedge EG(p_{21})) \wedge AF(p_{24})$.	$P'_{15} : AF(p_{12})$. $P''_{15} : EG((p_{33}) \wedge EG(p_{21}))$. $P'''_{15} : AF(p_{24}$.	$P'_{15} : CC_1, .., CC_4$, $P''_{15} : CC_1, .., CC_7$, $P'''_{15} : CC_1, .., CC_8$.
$P_{16} : AF(p_{12}) \wedge EG(p_{33}) \wedge EG(p_{21}) \wedge AF(p_{24})$.	$P'_{16} : AF(P_{12})$. $P''_{16} : \neg AF(p_{30})$.	$P'_{16} : CC_1, .., CC_4$, $P''_{16} : CC_1, .., CC_{10}$.

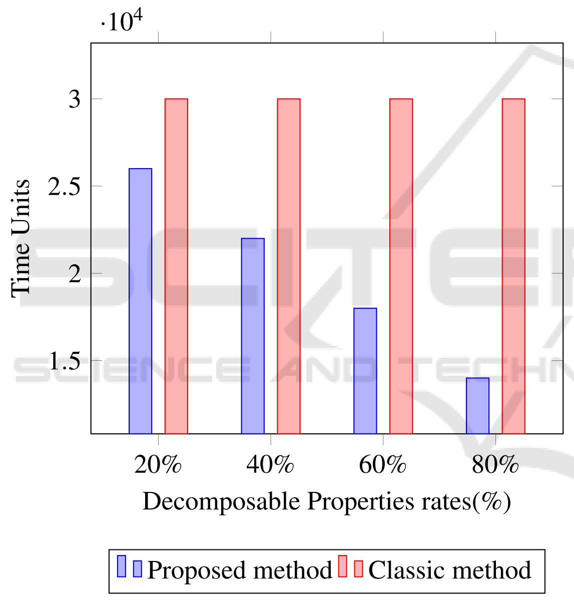


Figure 7: Classic method VS Proposed method.

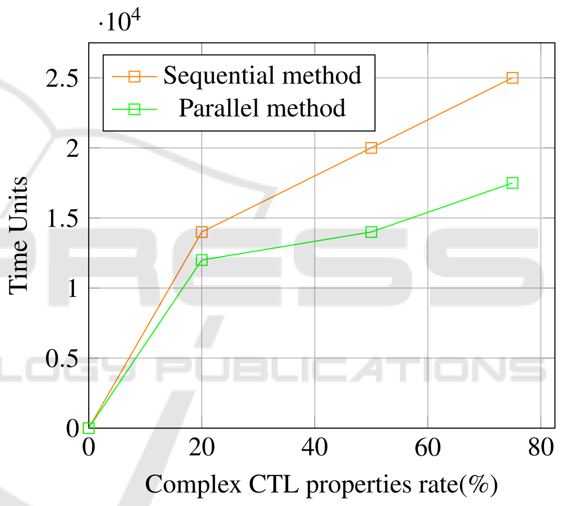


Figure 8: Improved performance of Parallel verification.

specified by CTL. We apply the possible decomposition to the CTL properties in σ, then we assign each property to be verified to the correspondent accessibility graph (BAG or PAG). The results are shown in Table 4.

4.3 Evaluation

Let assume we have to verify a system model with 2500 TNCESs. In order to ensure the well-behave of the system we have to verify at least 4 properties for each TNCES. Thus, we need to verify 10000 CTL properties. We assume that the properties to be verified are complex and the rate of decomposable one can be: (i) *Low* in $0,20\%$, (ii) *Medium* in $20,60\%$, or (iii) *High* when more than 60%. Figure 7 presents the result of properties verification using the classic method presented in (Hafidi et al., 2018) and the proposed method. The results show that the gain increases proportionally to decomposable properties rate. Thus, the gain is clearly shown when rate is *High*. Moreover, in Figure 8 we can observe an important gain when performing parallel verification thanks to the proposed architecture. The proposed architecture allows us to reduce considerably times execution by (i) Avoiding redundant calculation, (ii) Avoiding wait time execution, (iii) Performing several properties verification at the same time.

5 CONCLUSION

This work deals with the formal verification of RDECSs modeled by R-TNCES using CTL specifications. In this paper, we present a big data solution for the formal verification problem. A distributed cloud-based architecture is developed with two hierarchical levels (Master and worker) where, data storage is ensured by Amazon Simple Storage S3 (Murty, 2008)). It allows us to increase computational power, data availability, and to perform parallel execution. We introduce the modularity concept to the generated state spaces which allow us to execute the generation step in a parallel way via several workers (virtual machines). We detect the complex CTL properties and decompose them into several simple or less complex properties, then proceed to their verification via workers using the SESA tool (Patil et al., 2012). Incremental state space generation and the decomposition of CTL properties allow us to run a targeted verification, which is less complex and more efficient in terms of execution time. This work opens several perspectives; first, we plan to apply our approach in verification of real-case with a complex properties to check the functional and the temporal specifications. Then, automatize the detection of complex properties by using the IA thanks to ontologies.

REFERENCES

Baier, C. and Katoen, J.-P. (2008). *Principles of model checking*. MIT press.

Camilli, M., Bellettini, C., Capra, L., and Monga, M. (2014). Ctl model checking in the cloud using mapreduce. In *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2014 16th International Symposium on*, pages 333–340. IEEE.

Choucha, C. E., Ougouti, N. S., Khalgui, M., and Kahloul., L. (2019). R-tnces verification: Distributed state space analysis performed in a cloud-based architecture. In *Proceedings of the the 33rd Annual European Simulation and Modelling Conference*, pages 96–101. ETI, EUROSIS.

Hafidi, Y., Kahloul, L., Khalgui, M., Li, Z., Alnowibet, K., and Qu, T. (2018). On methodology for the verification of reconfigurable timed net condition/event systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, (99):1–15.

Hafidi, Y., Kahloul, L., Khalgui, M., and Ramdani, M. (2019). On improved verification of reconfigurable real-time systems. In *Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering*, pages 394–401. SCITEPRESS-Science and Technology Publications, Lda.

Hayes, B. (2008). Cloud computing. *Communications of the ACM*, 51(7):9–11.

Housseyni, W., Mosbahi, O., Khalgui, M., Li, Z., Yin, L., and Chetto, M. (2017). Multiagent architecture for distributed adaptive scheduling of reconfigurable real-time tasks with energy harvesting constraints. *IEEE Access*, 6:2068–2084.

Koubâa, A., Qureshi, B., Sriti, M.-F., Javed, Y., and Tovar, E. (2017). A service-oriented cloud-based management system for the internet-of-drones. In *2017 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 329–335. IEEE.

Naidji, I., Ben Smida, M., Khalgui, M., Bachir, A., Li, Z., and Wu, N. (2019). Efficient allocation strategy of energy storage systems in power grids considering contingencies. *IEEE Access*, 7:186378–186392.

Padberg, J. and Kahloul, L. (2018). Overview of reconfigurable petri nets. In *Graph Transformation, Specifications, and Nets*, pages 201–222. Springer.

Patil, S., Vyatkin, V., and Sorouri, M. (2012). Formal verification of intelligent mechatronic systems with decentralized control logic. In *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012)*, pages 1–7. IEEE.

Ramdani, M., Kahloul, L., and Khalgui, M. (2018). Automatic properties classification approach for guiding the verification of complex reconfigurable systems. In *ICSOFT*, pages 625–632.

Zhang, Jiafen, and et al. (2013a). ”r-tnces: A novel formalism for reconfigurable discrete event control systems. *Systems, Man, and Cybernetics: Systems, IEEE Transactions on 43.4*, pages 757–772.

Zhang, J., Khalgui, M., Boussahel, W. M., Frey, G., Hon, C., Wu, N., and Li, Z. (2015). Modeling and verification of reconfigurable and energy-efficient manufacturing systems. *Discrete Dynamics in Nature and Society*, 2015.

Zhang, J., Khalgui, M., Li, Z., Mosbahi, O., and Al-Ahmari, A. M. (2013b). R-tnces: a novel formalism for reconfigurable discrete event control systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 43(4):757–772.