Cluster Computing (2020) 23:2453–2470
https://doi.org/10.1007/s10586-019-03018-9

一种针对qos感知的多云服务组合的混合形式化验证方法

A hybrid formal verification approach for QoS-aware multi-cloud service composition

Alireza Souri[1] · Amir Masoud Rahmani[1] · Nima Jafari Navimipour[2] · Reza Rezaei[3]

Received: 9 April 2019 / Revised: 25 July 2019 / Accepted: 13 November 2019 / Published online: 28 November 2019

Abstract

Today, cloud providers represent their individual services with several functional and non-functional properties in various environments. Discovering and selecting an appropriate atomic service from a pool of activated services are a main challenge in the multi-cloud service composition. Minimizing the number of cloud providers is a critical matter in the service composition problem, which effects on energy consumption, response time and total cost. This paper presents a hybrid formal verification approach to assess the service composition in multi-cloud environments though the decreasing number of cloud providers to gain final service composition with a high level of Quality of Service (QoS). The presented approach provides behavioral modeling to examine the procedure of user' requests, service selection, and composition in a multi-cloud environment. Also, the proposed approach permits analysis of the service composition using a Multi-Labeled Transition Systems (MLTS)-based model checking and Pi-Calculus-based process algebra methods for monitoring the functional specifications and non-functional properties as the QoS standards. In addition, the proposed approach satisfies the functional properties for the multi-cloud service composition. The experimental results proved the feasibility of the proposed approach with performance evaluations and some confirmation setups.

Keywords Service composition · Multi-clouds · Verification · QoS · Specification

1 Introduction

In recent years, cloud computing helps information technology-based organizations to subcontract web service applications [1, 2]. Development growth of this technology has replaced current practices [3]. Cloud environment redirects the distribution of web services, including cloud providers, and smart applications over the internet [4]. There are many reasons to consider cloud computing over general server-based computing, few of the reasons are cost efficiency, high speed, global scale, productivity, performance, and reliability [5, 6].

The multi-cloud environment is considered by a set of simultaneous cloud providers such as private and public clouds to perform a set of appropriate services. The concept of Multi-Cloud Service Composition (MCSC) approach is applied to the process of numerous cloud service aggregations for generating appropriate composited services. In the composition procedure, user requests are received for creating a composite service [7]. After authenticating the user's request, the accepted service is found between the activated services in multi-cloud providers. Then, service description is composed of a transitional language and composer engine executes it on the selected services [8–10]. In the multi-cloud environment, finding the composition of cloud services is a key challenge [11, 12]. Therefore, finding composited service in all of the candidate services that are created with minimal cloud providers is a challengeable issue in the multi-cloud environment.

Some research studies have discussed the evaluation of service composition approach with simulation results

✉ Alireza Souri
a.souri@srbiau.ac.ir

[1] Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran

[2] Department of Computer Engineering, Tabriz Branch, Islamic Azad University, Tabriz, Iran

[3] Department of Computer Engineering, College of Technical and Engineering, West Tehran Branch, Islamic Azad University, Tehran, Iran

[13–15]. Simulation results illustrate that statistical techniques can facilitate performance analysis of non-functional properties such as response time, reliability, availability, and cost. In opposite, formal methods represent that verification techniques can simplify the efficiency of functional properties such as deadlock-free, reachability, fairness, and liveness to evaluate the correctness of system behavior [16]. Therefore, a study directed towards modeling approaches that permit engineers to verify and satisfy the service composition during the early design phase is required [17]. The service composition procedure is performed by finding a particular proof from a set of activated services as assumptions that satisfy related Quality of Service (QoS) factors of each user's request [18].

The main effort of the verification approach in this research is converting semantic interfaces of the cloud service descriptions into verifiable atomic propositions [19]. Satisfying the functional properties with existing atomic propositions of a multi-cloud service composition approach with numerous candidate services has an essential confirmation to support scalable state exploration in verification results [20]. To achieve this effort, a hybrid formal verification approach is presented to support qualified functional properties using model checking approach and proving the correctness of complex service composition behavior in a highly scalable multi-cloud environment with Pi-Calculus-based process algebra. This hybrid approach provides all advantages of two powerful verification methods for a complex multi-cloud service composition approach. Due to complementary characteristics of both model checking and process algebra methods [21], model checking approach has some advantages such as automatic state exploration [22, 23], automated verification, and counterexample generation but cannot provide qualitative analysis for large-scale systems with dynamic behavior [24, 25]. Even though process algebra supports scalability analysis to verify enumerated general mathematical terminologies in large-scale systems.

This research provides complementary nature benefits of both verification approaches to prove the correctness of the integrated multi-cloud service composition methodology. Thus, the main goal of this research is to present a formal study where the behavioral parameters of the system can be easily modified to obtain new results. This paper presents a hybrid formal verification architecture to the cloud service composition in the multi-cloud environment. Also, a Multi-Labeled Transition System for Multi-Cloud Service Composition (MLTS_MCSC) method presents to decrease the number of cloud providers to gain final service composition with a high level of Quality of Service (QoS). The presented approach provides behavioral modeling to examine the procedure of user' requests, service selection, and composition in a multi-cloud environment.

The key contributions of this research are illustrated as follows:

- Proposing the MCSC scenario with supporting QoS of user's requirements.
- Presenting a hybrid formal verification approach to evaluate the correctness and feasibility of the proposed MCSC approach with supporting minimum cloud providers.
- Providing minimum cloud providers selection to compose the examined cloud service using a Multi-Labeled Transition Systems (MLTS)-based model checking.
- Defining critical specification rules using process algebra method to prove the correctness and scalable exploration of the MLTS_MCSC method.
- Demonstrating the use of the NuSMV model checker to evaluate the functional specifications and QoS factors of the MLTS_MCSC method.

The rest of the research study is organized as follows: Sect. 2 illustrates a state of the art related works in this field according to some analytical comparisons. Section 3 illustrates a brief description of multi-cloud computing and service composition approach. Also, the main presentation of considered MCSC architecture in this research is given in Sect. 3. Section 4 shows the formal verification methods for the MCSC approach. Section 5 presents the analytical and experimental results obtained from the NuSMV tool. Finally, Sect. 6 shows the conclusion and future work of this research.

2 Related work

Formal specification and verification of cloud service composition is an emerging matter in the correctness and guarantee of the QoS factors. Most of the research studies have evaluated formal verification on web service composition using model checking and process algebra methods. This section illustrates a brief literature review for comparing existing verification approaches.

Gao et al. [26] presented an approach that utilizes recommendation of formal verification to offer the most appropriate services for abstract workflows. This services combination approach based on cost control, in the first step introduces an inverted index based to improve service search efficiency. Then defines formalize the service composition behavior through model of service and workflows. The disadvantage of this article is that investigated only in the uncertain environment. Also, Li et al. [27] have presented a platform through combination of description model, interaction scenario model and composition process formal model that manufactures cloud service composition based on process calculus cause describes the quality of service (QoS).

The prototype of this platform focused on user and service management, service of cloud manufacturing resource and formal verification.

Bourne et al. [28] proposed a developing type of cloud service to suggest customer's executable and configurable processes of business over the internet using templates of temporal logic. The Business Process as a Service (BPaaS) verified transactional behaviors and requirements of the customers by model checking also BDD analysis guarantees that BPaaS features don't breach the domain service provider constraints. This approach using differences modeling technique such as: BPMN, state charts and feature models. In other work, Souri et al. [29] applied a service composition based on social customer relationship management (CRM) through formal verification methods. They divided behavioral models into the three operational, analytical and collaborative behaviors and then according to the Kripke structure modeling the KP features supported through the three behaviors existing. They translated the models and specification properties into the SMV code.

Ghobaei-Arani et al. [30] proposed a moth-flame optimization (MFO) to solve the Web Service Composition (WSC) problems also it was improving criterion of QoS. In first, the model based approach was presented defines some attributes such as reachability, safety specification, liveness and deadlock. And then they analyzed correctness of their behavior model with NuSMV model checker. Mezni et al. [31] presented an algorithm to solve multi cloud service composition problems through taking analysis advantages of formal concept. Their experimental results showed that the grouping ability of FCA reduce number of providers and clouds and classify it also minimized execution time. This method effects on service composition high quality. Also, Entezari-Maleki et al. [32] proposed a model timed colored Petri nets (TCPNs) based to minimize the number of composite service request clouds. Atomic services provided all the request of composite service. This graphically model was presented represents the request submission process, service selection and composite service analysis furthermore cans evaluate the performance system.

Rai et al. [33] have presented a modeling and verification approach for interaction of web service based on recursive composition graph. They employed it to take the specifications about service interactions and show the interactions between recursive composition specification language (RCSL) and web service. They solved the problem of search for automated composition and recursive composition. Khai et al. [34] have presented a clustering method of web services to improve web service verification and composition based on clustering logic. The results of this approach applied in an on-the-fly sematic for verify service. The advantage of this study is find the exact solution cause don't discard any cluster.

Table 1 shows a comparison analysis for existing research studies on this topic.

3 Multi-cloud service composition

This section illustrates a brief explanation of the service selection and composition approach in the multi-cloud environment. First, a QoS-aware service composition approach is presented to illustrate the non-functional specifications according to user's requirements in service selection and composition. Second, the proposed Multi-Cloud Service Composition (MCSC) approach is presented for mapping the multi-cloud composition scenario into the formal verification approaches.

3.1 QoS-aware service composition

In this paper, four criteria of QoS are considered that include response time, availability, cost and reliability as presented in Table 2. Also in this paper, the SLA is defined with respect to these four criteria [35].

In the QoS-aware service composition, the service composer presents a candidate composition plan for specifying each atomic service functionalities according to user's requirements.

There are four basic patterns to illustrate the service composition in cloud computing according to Fig. 1 [36]: serial (a), parallel (b), combined switch (c), and loop (d). Calculating the QoS of the sequential pattern provides a basis for calculating other patterns.

Table 3 depict aggregation of QoS factors based on four basic composition patterns for the composited service. P_i shows the possibility that executes *i-th* atomic service.

3.2 Multi-cloud selection scenario

To achieve an optimal cloud service composition with minimal sub-set of clouds and a high level of QoS, a set of multiple clouds are proposed that hold cloud providers with their existing services. To minimize energy consumption and communication time between cloud providers and user's requests, a multi-cloud selection scenario is provided to find the reduced combination of cloud services that potentially provides the service composition according to minimal sub-set of clouds and high level of QoS factors. Table 4 presents an example to illustrate the MSCS approach. In this example, 32 services with different functional and QoS factors named $S_1 \ldots S_{32}$ have existed with 7 providers named $P_1 \ldots P_7$, which are hosted in 3 clouds named $C1$, $C2$, and $C3$. A cloud service provider P can distribute various atomic services $S_i \ldots S_j$ in a cloud such as C_1 or even numerous clouds such as C_i and C_k. For example, according to Table 4, service S_3 is distributed

Table 1 Comparison of the proposed work with other methods for verification of service composition

Research	Formal model	Correctness property	Specification logic	Tool/Method	Development phase	Case study
[26]	Model checking	Yes	CTL	PRISM	Implementation	Web service
[27]	Process algebra	No	CTL	MyEclipse	Implementation	Cloud service
[28]	Model checking	No	CTL	NuSMV	implementation	Cloud service
[29]	Model checking	Yes	CTL	NuSMV/Kripke structure	Implementation	Cloud service
[30]	Model checking	Yes	CTL,LTL	NuSMV	Implementation	Web service
[31]	Process algebra	Yes	Operational Flow Language	Formal concept analysis	Implementation	Multi-cloud service
[32]	Theorem proving	No	–	CloudSim/colored petri net	Design	Multi-cloud service
[33]	Process algebra	Yes	CTL,LTL	NuSMV/Kripke structure	Implementation	Web service
[34]	Process algebra	No	Clustering logic	Labeled transition system	Design	Web service

Table 2 Examples of QoS metrics for web services [9]

QoS metrics	Description	Unit
Response time	Executing time between the moment a requisition and the moment when the result is achieved	ms
Cost	The money that the applicant should pay to the service provider for using the service	$
Availability	The probability of the availability of a service	%
Reliability	The reliability of the service	%

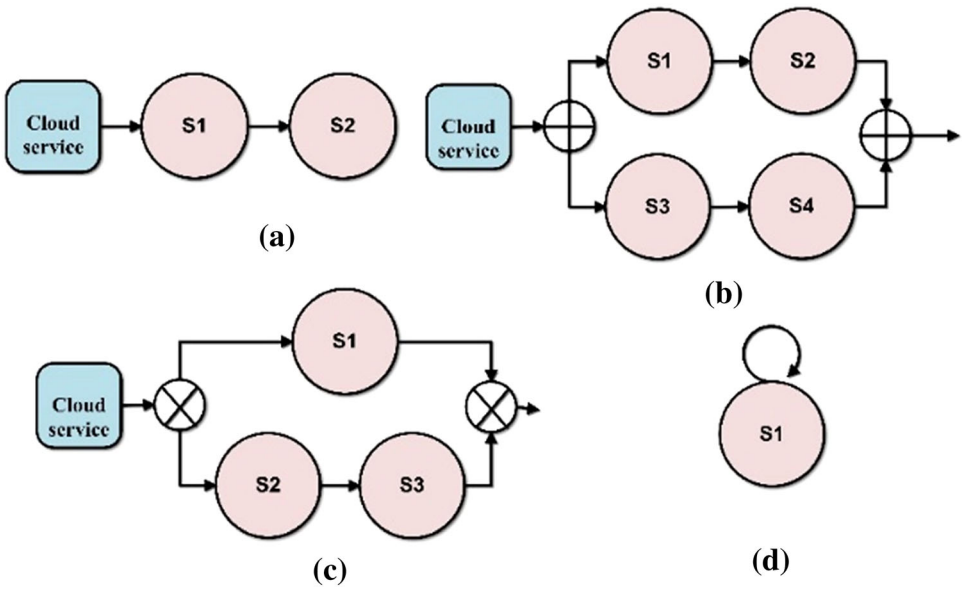

Fig. 1 The service composition patterns: **a** serial, **b** parallel, **c** combined switch, **d** loop

into three different cloud $C1$, $C2$, and $C3$ with cloud provider P_2.

Suppose a user request is confirmed in form of $RS = (S_1, S_5, S_8)$, all of the candidates composited services are suggested in forms of *Cloud i* (*Provider j*) according to the requested QoS factors as follows in Table 5.

After suggesting the candidate services, all three cloud indexes are compared with together. This comparison is a cloud provider reduction method. If we find the same cloud index in candidate clouds, then the cloud indexes are merged together as follows in Table 6.

Also, all three provider indexes are compared together. If we find the same provider index in candidate providers, then the provider indexes are merged together as follows in Table 7.

After specifying all candidate composited services with cloud provider reduction method, the communication time between appropriate cloud providers is calculated for each candidate composited service. We proposed Tables 8 and 9 that illustrate the communication time between cloud connections and provider connections consequently.

Table 3 Aggregation of QoS factors based on four basic composition patterns [36]

QoS criteria	Abbreviation	Sequential	Parallel	Probabilistic	Circular
Response time	$RT(S_i)$	$\sum_{i=1}^{n} RT(S_i)$	$Min(RT(S_i))$	$\sum_{i=1}^{n} P_i * RT(S_i)$	$k * RT(S)$
Cost	$C(S_i)$	$\sum_{i=1}^{n} C(S_i)$	$Min(C(S_i))$	$\sum_{i=1}^{n} P_i * C(S_i)$	$k * C(S)$
Availability	$A(S_i)$	$\prod_{i=1}^{n} A(S_i)$	$Max(A(S_i))$	$\prod_{i=1}^{n} P_i * A(S_i)$	$A(S)^k$
Reliability	$R(S_i)$	$\prod_{i=1}^{n} R(S_i)$	$Max(R(S_i))$	$\prod_{i=1}^{n} P_i * R(S_i)$	$R(S)^k$

According to the communication time of existing clouds and providers, Total Communication Time (TCT) for each candidate service is evaluated as follows (Eq. 1) that C_{ij}*(-time)* and P_{ij}*(time)* show communication time between clouds C_i and C_j and providers P_i and P_j respectively:

$$TCT = \sum_{i,j=1}^{n} Cij(time) + \sum_{i,j=1}^{n} Pij(time) \quad (1)$$

After evaluating the *TCT* of each candidate composited service, the minimum value of the *TCT* is considered as final service composition as follows: $\text{Min} \forall TCT_i$ where $1 < i < n$ that n is the number of candidate services.

We examine the total communication time *TCT* for each candidate service between each cloud and provider as follows where a ⇔ b denotes the *TCT* value for each communication between a and b according to Tables 8 and 9:

- TCT1 = {C1 (P1), C2 (P5), C3 (P7)} = ((C1 ⇔ C2) + (C2 ⇔ C3) + (C1 ⇔ C3)) + ((P1 ⇔ P5) + (P5 ⇔ P7) + (P1 ⇔ P7)) = (200 + 250 + 180) + (25 + 30 + 16) = **701**

- TCT2 = {C1 (P1), C3 (P5, P7)} = (C1 ⇔ C3) + ((P1 ⇔ P5) + (P5 ⇔ P7) + (P1 ⇔ P7)) = (180) + (25 + 30 + 16) = **251**

- TCT3 = {C1 (P1), C3 (P8)} = (C1 ⇔ C3) + (P1 ⇔ P8) = (180) + (22) = **202**

- TCT4 = {C1 (P1), C2 (P6), C3 (P7)} = ((C1 ⇔ C2) + (C2 ⇔ C3) + (C1 ⇔ C3)) + ((P1 ⇔ P6) + (P6 ⇔ P7) + (P1 ⇔ P7)) = (200 + 250 + 180) + (27 + 30 + 30) = **717**

- TCT5 = {C1 (P1), C2 (P6), C3 (P8)} = ((C1 ⇔ C2) + (C2 ⇔ C3) + (C1 ⇔ C3)) + ((P1 ⇔ P6) + (P6 ⇔ P8) + (P1 ⇔ P8)) = (200 + 250 + 180) + (27 + 22 + 25) = **704**

- TCT6 = {C1 (P1), C3 (P5, P8)} = (C1 ⇔ C3) + ((P1 ⇔ P5) + (P5 ⇔ P8) + (P1 ⇔ P8)) = (180) + (25 + 22 + 20) = **247**

By comparing total communication time for each candidate composited service, we see that final cloud composited service is Candidate 3 with interconnecting clouds *C1* and *C3*, by providers *P1* and *P8* with minimum total communication time 202 ms.

According to the MCSC example, some preliminaries are defined as follows:

Definition 1 (*Multi-Cloud Services*). A MCS is a set of Cloud providers *MCS = {CS1, CS2… CSN}*, where *CSi* $(1 \leq i \leq N)$ is a cloud that gets a set *SP* of service providers, *SP = {Pi1, Pi2… PiG}*, where *Pij* $(1 \leq j \leq G)$ is the *j-th* service provider in cloud *CSi*. A service provider has belonged to more than one atomic service. A service provider suggests a set of services *S = {Sj1, Sj2… SjL}*, where *Sjk* $(1 \leq k \leq L)$ is the *k-th* service suggested by service provider *Pj* [31].

Definition 2 (*User requests*). For showing a multi-cloud composition candidate, user requests are denoted by requested services $RS = \{S_1, S_2, S_3, \ldots, S_n\}$ that is responded using a selected services with $W = \{CS_1(CP_1), CS_2(-CP_2), \ldots, CS_i(CP_j)\}$, where $CS_1, CS_2, \ldots, CS_k$ are the cloud services involved in the service composition method, and $CP_1, CP_2, \ldots, CP_k$ represent the minimum Cloud providers.

According to the above definitions, the MCSC approach is specified as the minimum set of applied cloud providers by evaluating total communication time and selected QoS factors. Next section illustrates the formal verification of the proposed MCSC approach to cover proof analysis of the functional specifications in the selection and composition procedure.

4 Formal verification of the MCSC approach

This section illustrates a hybrid formal verification approach to prove the correctness of the proposed MCSC approach. This formal approach uses the Multi-Labeled Transition System (MLTS) method as the model checking

Table 4 Existing multi-cloud attributes and providers with service distribution

Clouds	C1			C2				C3			
Providers	P1	P2	P3	P2	P4	P5	P6	P2	P5	P7	P8
Services	S1	S3	S2	S3	S2	S3	S5	S3	S3	S2	S3
	S2	S6	S3	S6	S3	S5	S6	S6	S5	S6	S4
	S3		S4		S6	S6	S7		S6	S7	S5
	S4					S7			S7	S8	S8

approach through NuSMV tool and the Pi-Calculus method as the process algebra approach [37]. Figure 2 presents the hybrid formal verification approach. According to Fig. 2, a multi-cloud composition scenario is proposed as a Business Process Model (BPM) workflow, which behavioral modeling is applied to this model [38]. For satisfying scalability of the proposed MCSC approach, the Pi-Calculus method is applied for proving the soundness of the minimum subset cloud services in the composited scenario. So, the formal specification of the proposed MCSC approach is applied using Pi-calculus expressions. To verify the MCSC approach, the proposed workflow is translated into the MLTS method to the behavioral modeling of the MCSC approach. After translating the MLTS model, this model can be converted to SMV codes. Consequentially, the specification rules of the behavioral model as the functional specifications are defined in forms of Computation Tree Logic (CTL) formulas [31]. Finally, the converted SMV codes and temporal logic formulas are inputted to the NuSMV model checker.

Figure 3 illustrates a service composition workflow according to the presented multi-cloud example in Table 4. A user requests a set of priority-based services in forms of *UR = {S1, S5, S6, S8, S2, S7, S3, S4, S5, S7}* to the composition navigator, which there are some composition results in the various clouds that may fulfill the proposed request. For behavioral modeling this scenario, the next subsection presents formal specification and verification of the MCSC approach.

4.1 A formal specification using the Pi-Calculus method for service composition

The proposed formal specification provides a sound foundation for automated verification of the proposed MCSC approach. First, a conceptual model of the multi-cloud composition scenario is proposed. Then, the syntaxes and semantics of Pi-Calculus are summarized. Afterward, based on this model, a formal specification approach is defined for the MCSC approach.

There are four semantic abbreviations for the Pi-Calculus method that represent the service composition patterns in forms of sequential, parallel, conjunctive, and disjunctive operators consequently as follows $\&$, $|$, $\oplus$, and $\otimes$. Also, data transactions and existing channels are illustrated with lowercase letters such as a, b, and c. The process conditions are represented by uppercase types such as U, D, and S. The intuitive meanings of concepts and prefixes are listed in Table 10 [39–41].

Definition 3 The main channel $\ell = \{a, b, c, d, e, f\}$ is a set of channel's names that input and output messages with the relations $a\ (msg).P$ and $\overline{a}\ <msg>\ .P$ consequently with process P in a Pi-Calculus relation formula.

There are 5 functional processes for navigating MCSC approach that is described as follows, where each functional process can have a set of sub-processes $Process = (p_1, p_2, p_3, \ldots, p_n)$ and messages $M_{process} = (msg_1, msg_2, msg_3, \ldots, msg_n)$ that interconnect with channel link $c \in \ell$:

- User request:: $U = (start, assignment, set)$, $M_U = (initialize, process, send)$ by channel link a.
- Service discovery:: D= *(discover component, workflow synthesis, semantic analysis)*, M_D= *(check, compare, calculate)* by channel link b.
- Service selection:: S = *(concept matchmaking, SLA filtering, proper cloud, proper provider, reduce communication, final set, notify inappropriately)*, M_S= *(check, unsuitable, suitable, forward, send-back, process)* by channel link c.
- Composition generation:: G = *(graph generation, workflow analysis, workflow execution, candidate service)*, M_G= *(build, process, investigate, perform)* by channel link d.
- Optimal composition:: O = *(optimal search, graph optimization, Max QoS, Min communication, optimal composition, end)*, M_O = *(process, choose, perform, satisfy, finish)* by channel link e.

For each functional process, a Pi-Calculus expression is demonstrated as follows according to the channel links, processes and existing messages [24]:

1. User request: A user request process is created from a set of constructed processes as an input in the MCSC approach.
   U = $\overline{a}$ <*initialize*> .*Start* & *a*(*initialize*).*assignment*, $\overline{a}$ <*process*> .*assignment* & *a*(*process*).*set*, $\overline{a}$ <*send*> .*set* & *a*(*send*).discover_component.
2. Service discovery: This process shows a set of preliminary processes for discovering existing cloud

Table 5 Suggested candidate cloud services in the first step

Requested Service	S1	S5	S8
Candidate 1	C1 (P1)	C2(P5)	C3 (P7)
Candidate 2	C1 (P1)	C3 (P5)	C3 (P7)
Candidate 3	C1 (P1)	C3 (P8)	C3 (P8)
Candidate 4	C1 (P1)	C2 (P6)	C3 (P7)
Candidate 5	C1 (P1)	C2 (P6)	C3 (P8)
Candidate 6	C1 (P1)	C3 (P5)	C3 (P8)

Table 6 Suggested candidate cloud services after cloud index reduction

Requested Service	S1	S5	S8
Candidate 1	C1 (P1)	C2 (P5)	C3 (P7)
Candidate 2	C1 (P1)	C3 (P5, P7)	
Candidate 3	C1 (P1)	C3 (P8, P8)	
Candidate 4	C1 (P1)	C2 (P6)	C3 (P7)
Candidate 5	C1 (P1)	C2 (P6)	C3 (P8)
Candidate 6	C1 (P1)	C3 (P5, P8)	

Table 7 Suggested candidate cloud services after provider index reduction

Requested Services	S1	S5	S8
Candidate 1	C1 (P1)	C2 (P5)	C3 (P7)
Candidate 2	C1 (P1)	C3 (P5, P7)	
Candidate 3	C1 (P1)	C3 (P8)	
Candidate 4	C1 (P1)	C2 (P6)	C3 (P7)
Candidate 5	C1 (P1)	C2 (P6)	C3 (P8)
Candidate 6	C1 (P1)	C3 (P5, P8)	

Table 8 Communication time between existing clouds

	C1	C2	C3
C1	0	200	180
C2	200	0	250
C3	180	250	0

Table 9 Communication time between existing cloud providers

	P1	P2	P3	P4	P5	P6	P7	P8
P1	0	20	15	30	25	27	30	22
P2	20	0	18	15	26	12	24	23
P3	15	18	0	28	34	25	28	25
P4	30	15	28	0	17	15	25	27
P5	25	26	34	17	0	18	16	20
P6	27	12	25	15	18	0	30	25
P7	30	24	28	25	16	30	0	15
P8	22	23	25	27	20	25	15	0

services according to the QoS levels of the user's request. First, the same QoS components are discovered between all of the cloud providers. Then, a workflow synthesis process is started in order to compare each QoS factor that is activated in each cloud service. In addition, a semantic analysis is applied to recognize service suitability for use. Finally, semantic analysis results are used to find applicable cloud services as well as multiple cloud service composition with high QoS levels using matchmaking concepts [42].

D = $\overline{b}$<*check*>.discover_component & *b*(*check*). Workflow_synthesis, $\overline{b}$<*compare*>.Workflow_synthesis & *b*(*compare*).semantic_analysis, $\overline{b}$ <*calculate*>.semantic_analysis & *b*(*calculate*).concept_matchmaking.

3. Service selection: In this stage, semantic matching of QoS parameters is performed that filters user's restrictions according to the SLA conditions. By selecting each discovered service from the appropriate cloud, the communication time is computed for selecting the proper provider that supports existing service. The optimal cloud and provider selection are done according to the minimum communication time factor.

   S = $\overline{c}$<*check*>.concept_matchmaking & *c*(*check*). SLA_filtering, $\overline{c}$<*suitable*>.SLA_filtering & *c*(*suitable*). proper_cloud, $\overline{c}$<*unsuitable*>.SLA_filtering &*c*(*unsuitable*).notify_inappropriate, $\overline{c}$<*forward*>. proper_cloud &*c*(*forward*). proper_provider, $\overline{c}$<*forward*>. proper_provider & *c*(*forward*).reduce_communication, $\overline{c}$<*process*>.reduce_communication & *c*(*process*).final_set, $\overline{c}$<*sendback*>.notify_inappropriate & *c*(*sendback*).concept_matchmaking, $\overline{c}$<*forward*>.final_set & *c*(*forward*).graph_generation.
4. Composition generation: After selecting a set of cloud services, the composition graph is generated that navigates workflow execution process to create all of the possible candidate cloud services.

   G = $\overline{d}$<*build*>.graph_generation & *d*(*build*).workflow_analysis, $\overline{d}$<*process*>.workflow_analysis & *d*(*process*). workflow_execution, $\overline{d}$<*investigate*>.workflow_execution & *d*(*investigate*).candidate_service, $\overline{d}$<*perform*>.candidate_service & *d*(*perform*). optimal_search.
5. Optimal composition: The optimal composite process explains selecting the near optimal composite service after comparing maximum QoS levels and minimum communication time of each candidate composed

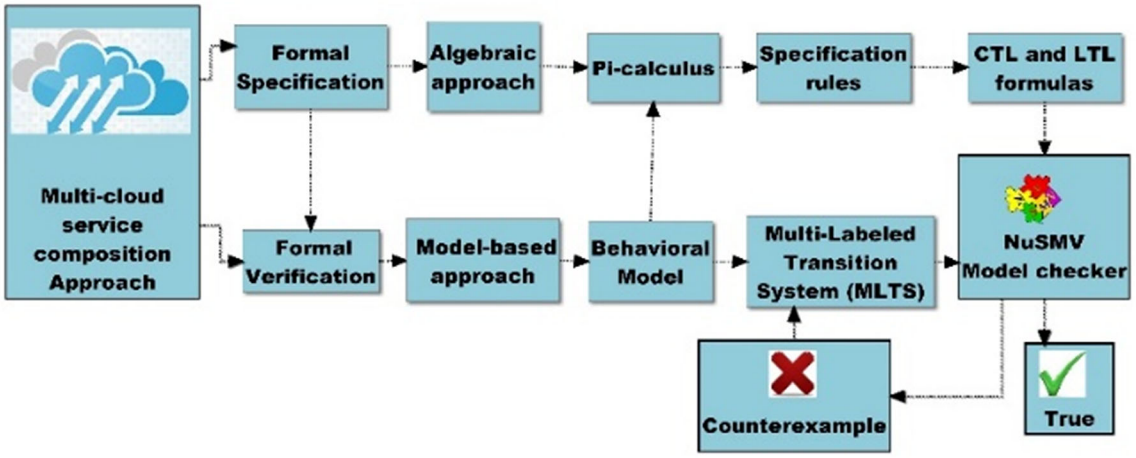

Fig. 2 The hybrid formal verification method for the MCSC approach

service. Finally, the optimized service composition workflow is allocated to the user.

O = $\overline{e}$<*process*>.optimal_search & *e* (*process*).-graph_optimization, $\overline{e}$<*choose*>.graph_optimization & *e*(*choose*).Max_QoS, $\overline{e}$<*perform*>.Max_QoS & *e*(*perform*).Min_communication, $\overline{e}$<*satisfy*>.Min_-communication & *e*(*satisfy*).optimal_composition, $\overline{e}$<*finish*>.optimal_composition & *e*(*finish*).end.

Definition 4 To verify the Pi-Calculus expressions, some specification patterns are presented in forms of $\frac{\varphi}{\psi}$, where φ denotes a condition rule then ψ is satisfied according to the operational semantics of φ.

- Specification pattern (1):

$$\mathrm{U} \xrightarrow{\mathrm{msg}} \mathrm{D}$$

This relation denotes that the process *U* performs the action *msg* then, process *D* becomes.

- Specification pattern (2):

$$\frac{\mathrm{U} \xrightarrow{\mathrm{msg}} \mathrm{D}}{\mathrm{S\&U} \xrightarrow{\mathrm{msg}} \mathrm{D}}$$

This relation denotes that if the process *U* performs *msg* and becomes *D*; then, the process *S* and *U* can sequentially perform *msg* and become *D*.

- Specification pattern (3):

$$\frac{\mathrm{U} \xrightarrow{\mathrm{msg}} \mathrm{D}}{\mathrm{S|U} \xrightarrow{\mathrm{msg}} \mathrm{D}}$$

This relation denotes that if process *U* executes the action *msg* and becomes process *D*; then, the process *S* and process *U* can be perfumed parallel transportation by the action *msg* to achieve the process *D*.

- Specification pattern (4):

$$\frac{\mathrm{U} \xrightarrow{\mathrm{msg}} \mathrm{D}}{\mathrm{S} + \mathrm{U} \xrightarrow{\mathrm{msg}} \mathrm{D}}$$

This relation denotes that if process *U* executes the

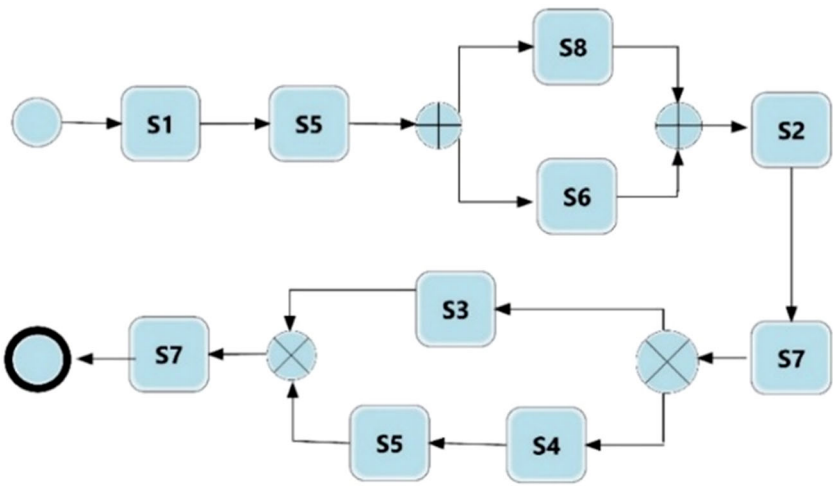

Fig. 3 A composition workflow in a multi-cloud environment

Table 10 The used terminologies in the Pi-Calculus method

Terminology	Description
Uppercase U, D, S, G, C, O	The functional processes
Lowercase *a, b, c, d*	The channels in the system
:: U	Define a new process
0	The null process in the system
U&D	A sequence composition
U ‖ D	A parallel composition
U ⊕ D	A conjunctive composition
U ⊗ D	A disjunctive composition
msg_1, msg_2, msg_3	The existing messages as actions
a(msg).P	A message input
$\bar{a}$<msg>.P	A message output
∧, ∨	And, Or for messages

action *msg* and becomes process *D*; then, the process *S* and process *U* can be perfumed conjunction by the action *msg* to achieve the process *D*.

- Specification pattern (5):

$$\frac{U \xrightarrow{msg} D}{S \times U \xrightarrow{msg} D}$$

This relation denotes that if process *U* executes the action *msg* and becomes process *D*; then, the process *S* and process *U* can be perfumed a disjunction by the action *msg* to achieve the process *D*.

- Specification pattern (6):

$$\frac{U \xrightarrow{msg1} D \,\&\, D \xrightarrow{msg2} S}{U \xrightarrow{msg1\ \&\ msg2} S}$$

If process *U* executes the action *msg1* and becomes process *D* then process *D* performs the action *msg2* for achieving the process *S*, then process *U* can be perfumed by the actions *msg1* and *msg2* sequentially to achieve the process *S*.

After describing some specification patterns, the MSCSC approach can be modeled as a proposed formal specification Eq. (2):

$$\begin{aligned} OptimalMCSC = {} & (U(MU \looparrowright a) + \cup(MD \looparrowright b) \\ & + \cap S(MS \looparrowright c) + \prod G(MG \looparrowright d) \\ & + Max\left(\sum(Min(O(MO \looparrowright e)\right) \end{aligned} \tag{2}$$

According to Eq. (2), the user's request is sent with functional process *U* and set of messages *MU* by an interconnecting channel *a*. So, service discovery stage is performed with process *D* and set of messages *MD* by interconnecting channel *b* in a union of enabled cloud services. Then, the service selection stage applies the intersection of appropriated services to generating QoS-based workflow with process *S* and set of messages *MS* by interconnecting channel *c*. Afterward, composition generation stage produces candidate composed services according to the product of the SLA and beneficial cloud providers with process *G* and set of messages *MG* by interconnecting channel *d*. Finally, optimal composition stage chooses optimal service composition set according to the summation of the minimum communication time value of each candidate composited service and maximum QoS level with process *O* and set of messages *MO* by interconnecting channel *e*.

After formalizing the proposed MCSC approach, some specification rules are modeled according to the Pi-Calculus expressions.

(1) $\mathcal{R}1$ = ($\bar{a}$ <*process*> *assignment* & *a*(*process*).*set*) ⊕ ($\bar{a}$ <*send*>.*set* & *a*(*send*).discover component) → ($\bar{b}$ <*compare*>.workflow synthesis) ‖ ($\bar{b}$ <*calculate*>.semantic analysis) ⊕ (*b*(*compare* ∧ *calculate*).concept matchmaking).

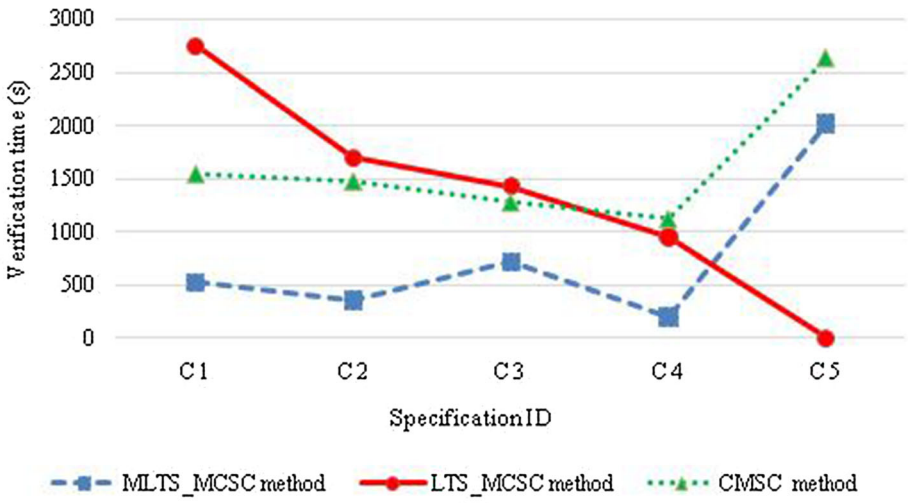

Fig. 4 The comparison of the verification time for each specification rule in scenario 1

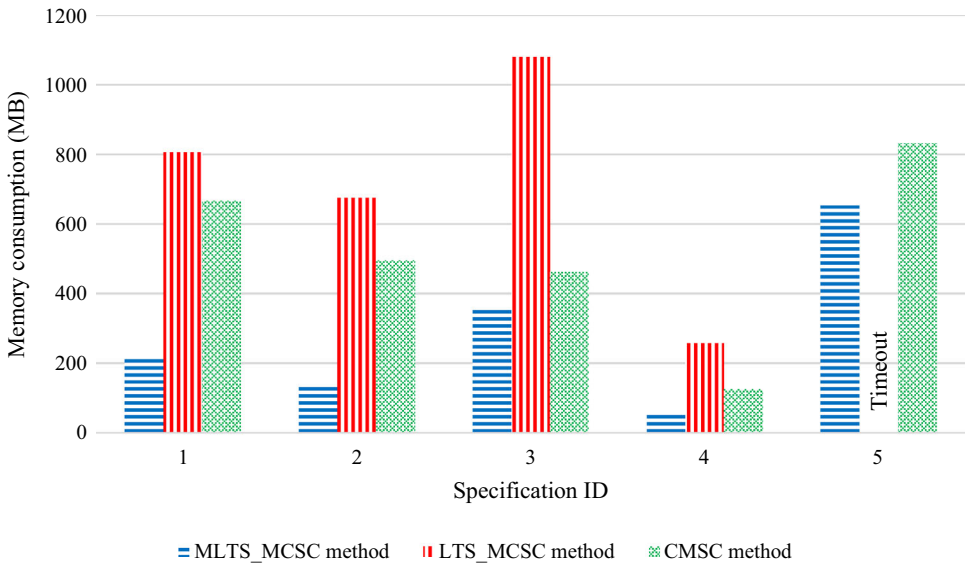

Fig. 5 The comparison of the memory consumption for each specification rule in scenario 1

(2) $\mathcal{R}2$ = ($\overline{c}$ <*suitable* ∨ *unsuitable*>.SLA filtering) ⊗ ((*c*(*suitable*). proper cloud) ‖ (*c*(*unsuitable*).notify inappropriate)) → ($\overline{c}$ <*forward*>. proper cloud & *c*(*forward*). proper provider) ‖ ($\overline{c}$ <*send back*>.notify inappropriate & *c*(*send back*).concept matchmaking).

(3) $\mathcal{R}3$ = ($\overline{c}$ <*forward*>.proper cloud &*c*(*forward*).proper provider) ⊕ ($\overline{c}$ <*forward*>. proper provider & *c*(*forward*).reduce communication) → ($\overline{c}$ <*forward*>.final set & *c*(*forward*).graph generation) ‖ ($\overline{d}$ <*process*>.-workflow analysis & *d*(*process*).workflow execution) ⊕ ($\overline{d}$ <*investigate*>.workflow execution & *d*(*investigate*).-candidate service)

(4) $\mathcal{R}4$ = ($\overline{d}$ <*perform*>.candidate service) → $\overline{e}$ <*choose* ∧ *perform*>.graph optimization→(*e* (*choose*).Max QoS) ⊕ (*e* (*perform*).Min communication)

(5) $\mathcal{R}5$ = ($\overline{a}$ <*process*>.*assignment*) ⊕ ($\overline{b}$ <*check*>.discover component) ⊕ ($\overline{c}$ <*check*>.concept matchmaking) ⊕ ($\overline{d}$ <*investigate*>.workflow execution) ⊕ ($\overline{d}$ <*perform*>.candidate service) ⊕ ($\overline{e}$ <*perform*>.Max QoS) ⊕ ($\overline{e}$ <*satisfy*>.Min communication) ⊕ ($\overline{e}$ <*finish*>.optimal composition).

In the next subsection, the proposed MCSC approach is translated to the LTS model for verifying and proving above specification rules.

4.2 MLTS-based model checking approach

After describing the formal specification of the MCSC approach, mapping the MCSC approach into MLTS method is presented. An MLTS is an action-based model that presents the communications between the states and the events of the behavioral model [43]. First, we define the LTS method as an initial model.

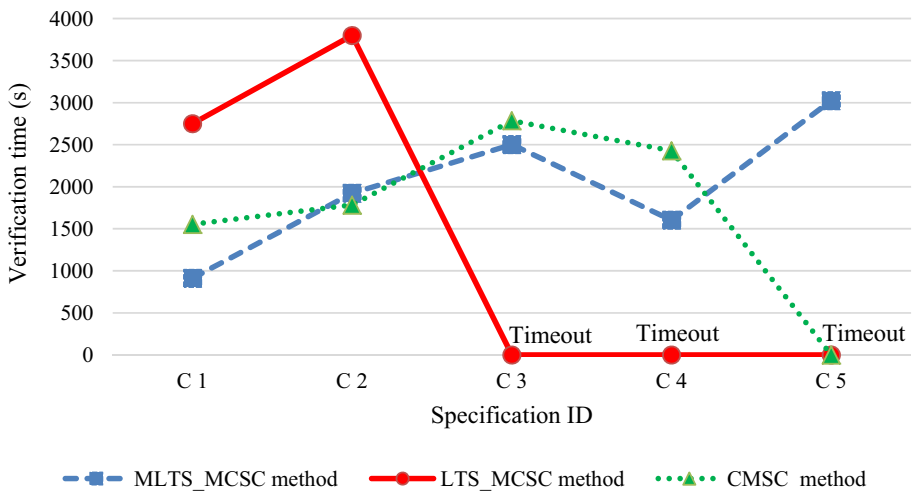

Fig. 6 The comparison of the verification time for each specification rule in scenario 2

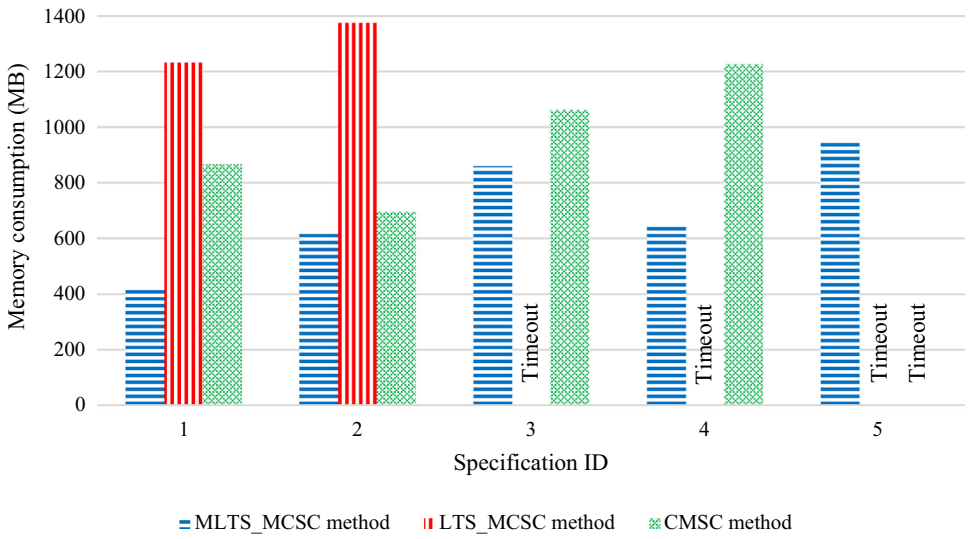

Fig. 7 The comparison of the memory consumption for each specification rule in scenario 2

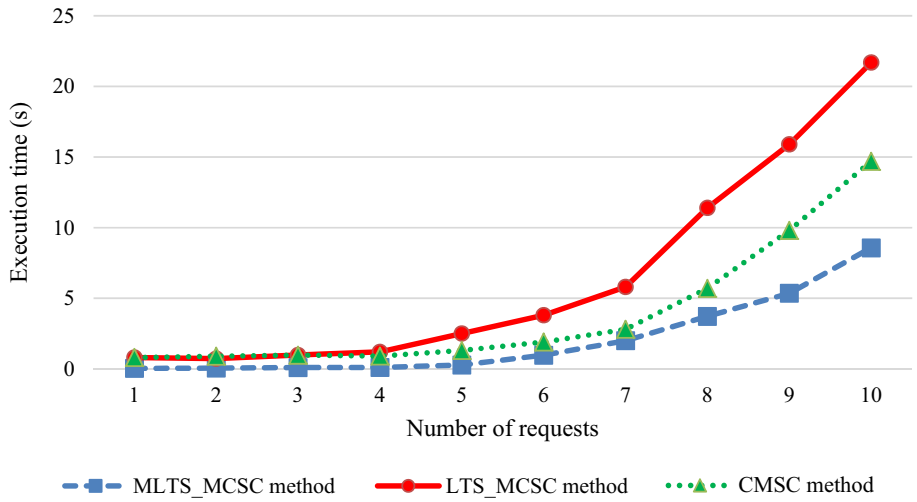

Fig. 8 The comparison of response time for multi-cloud service composition in scenario 1

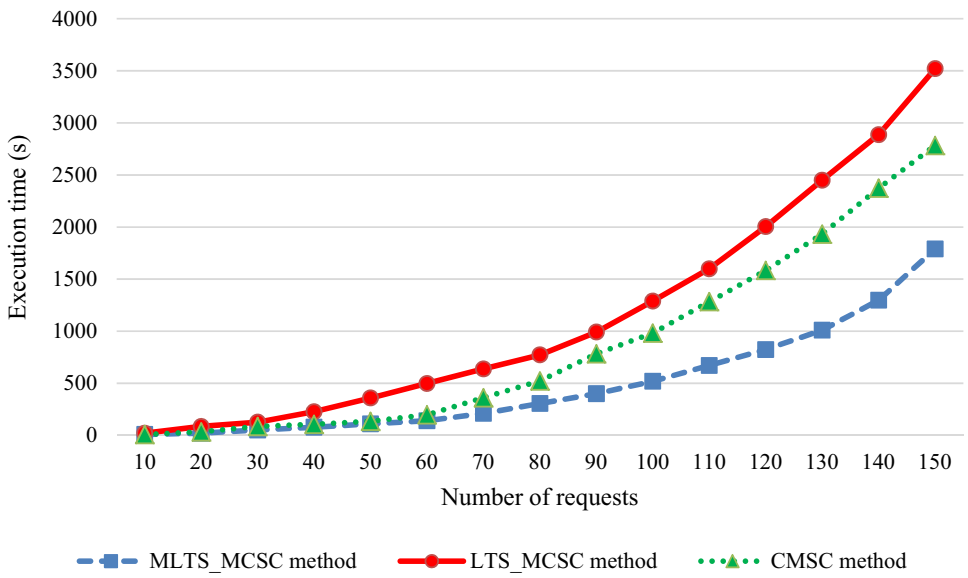

Fig. 9 The comparison of execution time for multi-cloud service composition in scenario 2

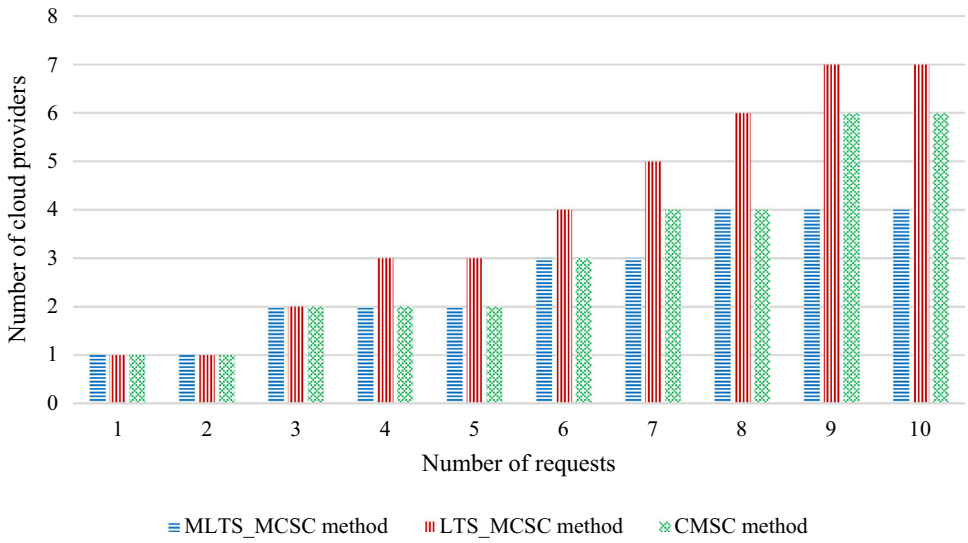

Fig. 10 A number of cloud providers in the multi-cloud service composition in scenario 1

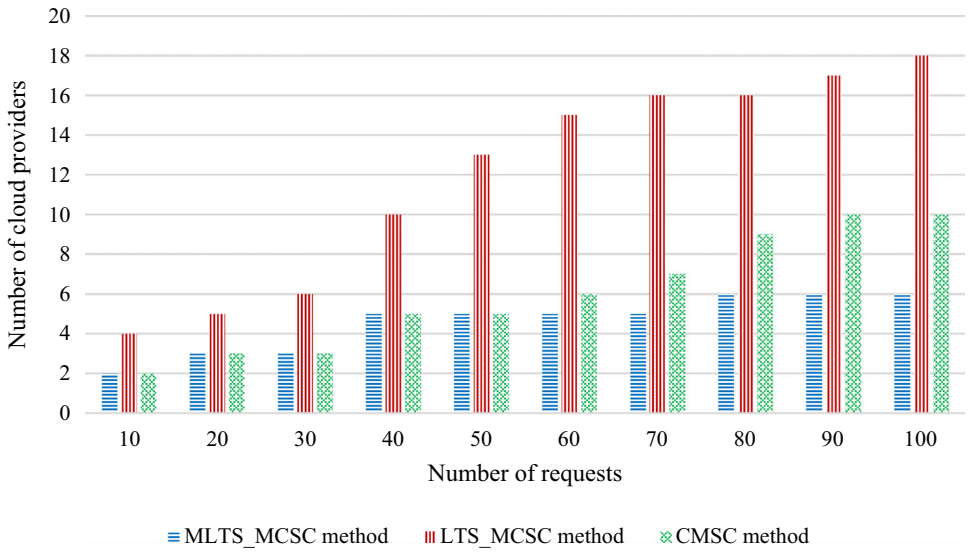

Fig. 11 A number of cloud providers in the multi-cloud service composition in scenario 2

Table 11 The CTL specification rules

CTL rule	Description
C1	*AG(assignment & process → set & send) → AX (discover component & send → EF (workflow synthesis & compare & semantic analysis & calculate)) → EX (concept matchmaking & calculate & compute)*
C2	*EG((SLA filtering & suitable → proper cloud) | (SLA filtering & unsuitable → notify inappropriate)) → EX (proper cloud & forward → proper provider)*
C3	*AG(proper provider & forward → reduce communication & final set) → AX (graph generation & forward) → EF(workflow execution & process → candidate service & investigate)*
C4	*EG(candidate service & perform → graph optimization) → EX (Max QoS & Min communication → optimal composition & choose)*
C5	*AG(EF(assignment & process → discover component & check → concept matchmaking → workflow execution & investigate → candidate service & perform → Max QoS & choose → Min communication & satisfy → optimal composition & finish))*

Table 12 State reachability comparison in verification results

CTL rule	MLTS_MCSC method			LTS_MCSC method		
	Total states	Reachable states	Reachability (%)	Total States	Reachable States	Reachability (%)
C1	566,224	537,913	95	3,223,762	2,321,108	72%
C2	328,407	318,554	97	1,970,447	1,517,244	77%
C3	728,691	670,395	92	4,372,148	3,016,782	69%
C4	122,326	121,102	99	733,958	601,845	82%
C5	1,636,534	1,456,515	89	Timeout	Timeout	Timeout

Table 13 A side by side comparison of verification analysis for the first scenario

CTL rule	MLTS_MCSC method			LTS_MCSC method			MLTS_MCSC method		LTS_MCSC method	
	Result	#State	#Transition	Result	#State	#Transition	Verification time (S)	Memory (MB)	Verification time (S)	Memory (MB)
C1	True	566,224	1,26,241	True	3,223,762	6,101,475	524	220	2750	806
C2	True	328,407	720,779	True	1,970,447	4,530,221	358	135	1700	675
C3	True	728,691	1,902,645	True	4,372,148	7,665,041	721	359	1430	1080
C4	True	122,326	330,263	True	733,958	1,530,221	200	53	953	257
C5	True	1,636,534	2,963,338	Timeout	Timeout	Timeout	2024	653	Timeout	Timeout

Definition 5 A Labeled Transition *LT* is a 4-tuple $LT = (S, s, E, T)$ where [13]:

- S is a set of states.
- s is the primary state: $s \in S$.
- E is a set of exsting events.
- T is an overall transition relation: $T \subseteq S \times E \times S$. In other words, the relation $s_1 \xrightarrow{e} s_2 (s_1, s_2 \in S \, and \, e \in E)$ is applied for stating that $(s_1, e, s_2) \in T$.

Definition 6 A Multi-Labeled Transition System *MLTS* is a 7-tuple $MLTS = (Q, q, A, E, M, T)$ where:

- Q is a set of states.
- q is the set of initial state: $q \in Q$.
- A is a set of attributes.
- E is a set of events.
- M is a set of multi-action labels that can have a multi-event and multi-attribute schema with some events E and some attributes A as follow:
  $(e_1\langle att_1.value\rangle, e_2\langle att_2.value\rangle, e_3\langle att_3.value\rangle, \ldots e_n\langle att_n.value\rangle)$ where $e_1, e_2, e_3, \ldots, e_n \in E$, $att_1, att_2, att_3, \ldots, att_n \in A$ and $value \in \mathbb{N}$ ($\mathbb{N}$ is set of the natural numbers).
- T is a final transition relation: $T \subseteq q \times M \times q$. This means the relation $q_1 \xrightarrow{a} q_2$ ($q_1, q_2 \in q$ *and* $a = e\langle att.value \in M$) is used for stating that $(q_1, a, q_2) \in T$.

Definition 7 A labeled transition path *LTP* is a determinate sequence of the states and events starting from state q_1 and finishing at state q_2 (q_1 *and* $q_2 \in q$) denoted as [44]:

$$LTP = q_1 \xrightarrow{a1} q_2 \xrightarrow{a2} q_3 \ldots q_{n-1} \xrightarrow{an} q_n \text{ such that } \forall (k, v) : (q_k, a_v, q_{k+1}) \in T.$$

Definition 8 A cloud service provider *cs* is a 6-tuple $cs = (P, ws, Q)$ where:

- P is a set of service providers where $P = (p_1, p_2, p_3, \ldots, p_n)$.
- ws is an atomic service type for service provider p where $p_1 = (ws_{11}, ws_{12}, ws_{13}, \ldots, ws_{1n})$.
- Q is a set of QoS attributes for each web service $ws_{ij} \in p_i$ that $QoS = (Cv, tv, Av, Rv)$:
  - Cv is a set of cost values in QoS attribute for each $ws_{ij} \in p_i$.
  - tv is a set of response time values in QoS attribute for each $ws_{ij} \in p_i$.
  - Av is a set of availability values in QoS attribute for each $ws_{ij} \in p_i$.
  - Rv is a set of reliability values in QoS attribute for each $ws_{ij} \in p_i$.

Definition 9 (*Cloud Services to MLTS*) Let $CS = (S_1,\ldots,S_n)$ be a set of cloud services, the *MLTS* model for cloud services is a guarded $MLTS_{CS}$ where:

- $q = (s_1, s_2, s_3,\ldots, s_n)$ is a set of service states.
- q is the set of initial service state.
- $A = \left(\sum_i^n CV\right) \oplus \left(\sum_i^n TV\right) \oplus \left(\prod_i^n AV\right) \oplus \left(\prod_i^n RV\right)$ is a 2-arrays aggregation of attributes as QoS factors that consists of a binary valued functions *(attribute, value)* between four QoS attributes in terms of cost, response time, availability and reliability for $ws_{ij} \in P_i$.
- E is a set of events.
- $M = \{(e_1, \langle CV_1.value\rangle, \langle TV_1.value\rangle, \langle AV_1.value\rangle, \langle RV_1.value\rangle), \ldots, (e_n, \langle CV_n.value\rangle, \langle TV_n.value\rangle, \langle AV_n.value\rangle, \langle RV_n.value\rangle)\}$.
- T is a total transition relation where $S_i \times (e_1, \langle cv_1.value\rangle, \langle tv_1.value\rangle, \langle av_1.value\rangle, \langle rv_1.value\rangle) \times S_j$.

Since each QoS value is evaluated in various bounds, a boundary normalization based on unity feature is applied to balance different QoS values in the instance *(0, 1)* in Eq. (3) [45, 46]. The *hb* is *1* as the high bound and the *lb* is *0* as low bound. The a_{max} and a_{min} are the maximum and minimum values of each QoS attribute. The a' is the normalized value.

$$a' = lb + \frac{(a - a_{min})(hb - lb)}{a_{max} - a_{min}} \tag{3}$$

For example, s_1 and s_2 are two services in provider p_1 that holds their QoS factors as follows, where cv_1 *is* cost, tv_1 *is* response time, av_1 is availability and rv_1 is reliability of service s_1 that are normalized in [0,1]. All of the QoS attributes are activated using event *calculate*. The value of each factor is determined with (factor.value), for examle, cost with value 0.2 is denoted by $<cv_1.0.2>$.

$s_1.calculate<cv_1.0.2>,<tv_1.0.3>,<av_1.0.9>,<rv_1.0.8>$& $s_2.calculate<cv_2.0.1>,<tv_2.0.4>,<av_2.0.8>,<rv_2.0.8>$.

To design the CTL specification rules of the proposed MCSC approach, the CTL grammar is shown as follow [19]

$$\Phi ::= \text{True} \mid p \mid \neg\Phi \mid \Phi\wedge\Phi' \mid \Phi\vee\Phi' \mid AX\Phi \mid EX\Phi \mid AF\Phi \mid EF\Phi \mid EG\Phi \mid A\text{G}\Phi$$

- True shows a correct proposition.
- The *AP* illustrates set of atomic propositions, $p \in AP$.
- The Φ is ranged over CTL formulas.
- *E*ventually (E) and *Always* (A) are the quantifiers on the paths.
- *G*lobally (G), *neXt* (X) and *Future* (F) show temproal benchmarks.

The Pi-calculus-based specification rules of the MCSC approach ($\mathcal{R}1$, $\mathcal{R}2$, $\mathcal{R}3$, $\mathcal{R}4$, $\mathcal{R}5$) are translated to the CTL formulas which can be evaluated for the MLTS model. These specification rules are converted into functional properties in terms of CTL formulas according to Table 11. We can let $\rightarrow$ as the logical implication. Each specification rule can be satisfied with state space of model and violated using a counterexample in the NuSMV model checker.

5 Experimental results

In this section, some experimental results are presented using a model checking approach in different scenarios. The first experiment depicts the verification analysis for functional properties of proposed MCSC approach with MLTS method called *MLTS_MCSC* in compare to the labeled transition system (*LTS_MCSC*) method that presented in [29] and the Commercial Multi-clouds Service Composition (*CMSC*) method that presented in [47]. The second experiment illustrates experimental analysis to evaluate the non-functional properties of the *MLTS_MCSC* and other methods. The experimental results are implemented on a system using an Intel Core i5 (2.60 GHz), 8 GB RAM, Windows 10 (64 bit) and the NuSMV model checker. According to real-business scenarios, a cloud service cannot be represented in the great number of cloud providers by notice to management complexity and high communication costs. To do this, two scenarios for evaluating the MLTS_MCSC method are proposed. In the first scenario, we have considered 3 clouds, 8 providers that present 35 services. The second scenario shows a scalable environment with 6 clouds, 20 providers that present 100 services.

5.1 Verification analysis

The goal of the first experiment is to analyze the correctness of the specific rules that should be satisfied in the MCSC approach. To achieve the best performance by proving the specification rules, we have considered the proposed workflow approach according to Fig. 3 that is mapped on the behavioral model of the MLTS_MCSC method. Table 12 demonstrates the number of reachable states in the proposed MLTS_MCSC method and LTS_MCSC method for each specification rule. Also, the percentage of the state reachability in the MLTS_MCSC method is higher than the LTS_MCSC method for each examined specification rule.

Afterward, Table 13 illustrates the verification results of the existing specification rules with the number of examined states and transitions. According to Table 13, we observed that the C5 specification rule was timed out in the LTS_MCSC method. Also, the number of states and transitions for the MLTS_MCSC method are lower than the number of states and transitions for the LTS_MCSC method. In addition, memory consumption of the MLTS_MCSC method is lower than the other approach according to satisfaction of existing specification rules. Finally, the verification time and memory consumption of running the C5 rule are out of model checking rate.

Figure 4 shows verification time for satisfying each specification rule in the MLTS_MCSC, LTS_MCSC and CMSC methods. We observed that the verification time of the MLTS_MCSC method is lower than the other methods according to the satisfaction of existing specification rules. Also, the verification time of the C5 specification rule was timed out in the LTS_MCSC method. Also, Fig. 5 presents the memory consumption of the verification results for the CTL rules in the first scenario according to a number of user's requests for composited services. Based on the multi-labeled method, the proposed MLTS_MCSC method has minimum memory consumption than other methods. The memory consumption value of the C5 specification rule was timed out in the LTS_MCSC method.

Figure 6 illustrates verification time for satisfying each specification rule in the second scenario with 6 clouds, 20 providers that present 100 services. We observed that the verification time of the MLTS_MCSC method is lower than the other methods according to the satisfaction of existing specification rules. Also, the verification time of the C3, C4, and C5 specification rules were timed out in the LTS_MCSC method and verification of the C5 specification rule was timed out in the CMSC method.

Moreover, Fig. 7 shows the memory consumption of the verification results for the existing specification rules in the second scenario the MLTS_MCSC, LTS_MCSC, and CMSC methods. The experimental results showed that the proposed MLTS_MCSC method has minimum memory consumption than other methods. The memory consumption value of the C3, C4 and C5 specification rules were timed out in the LTS_MCSC method. Also, the memory consumption value of the C5 specification rule was timed out in the LTS_MCSC method.

5.2 Experimental analysis

Figure 8 shows the execution time for the first scenario according to the number of user's requests for composited services in proposed MLTS_MCSC, LTS_MCSC and CMSC methods. We observed that the execution time growths when the number of requests in the LTS_MCSC and CMSC methods increases exponentially. The mean execution time of composited requests in the MLTS_MCSC is lower than the other methods. When the number of requests is increased, this is a realistic comparison since the growth level of the MLTS_MCSC is lighter than the other methods.

Figure 9 shows the execution time of the MCSC approach for the second scenario according to the number of user's requests for composited services. We observed that the MLTS_MCSC method has minimum growth of execution time than other methods when the service composition problem is applied up to 150 requests.

In addition, Figs. 10 and 11 depict the number of examined cloud providers in the service composition of the scenarios 1 and 2 respectively for the MLTS_MCSC, LTS_MCSC and CMSC methods. We have observed that by applying the MLTS_MCSC method, the minimum number of cloud providers are selected and composed for user's requests. This reduction leads to decrease communication time, cost and energy consumption between cloud data centers. According to Fig. 7, when the number of requests is specified between 1 and 3, the number of selected final cloud providers are equal for two multi-cloud approaches.

When the number of requests is increased, the capability of the scalable proposed MLTS_MCSC method can be observed to select minimum cloud providers from the set of candidate composited services. For example, when the number of requests is increased between 80 and 100 requests (Fig. 11), the number of selected cloud providers are fixed to compose existing cloud services with 6 cloud providers in the MLTS_MCSC method. In opposite, the number of examined cloud providers are 16, 17 and 18 respectively for the existing number of requests in the LTS_MCSC method.

5.3 Discussion

Using a hybrid formal verification approach can support the complementary advantages of the model checking and the process algebra in the multi-cloud service composition. On the other hand, by reducing the number of cloud providers for interconnecting users and cloud services in the multi-cloud environment, some beneficial aspects of the SOA are affected to the efficiency of the cloud service composition as follows [48–50]:

- Performance of cloud resources is an important matter for the multi-cloud environment. Selecting the minimum number of cloud providers in the composition procedure can affect the performance of cloud resources.
- Bandwidth usage is the main challenge for decreasing consumed cost and time in composited cloud services with the minimum number of cloud providers.
- Interoperability is one of the main factors to interchange information and resources between smart devices and cloud services in the multi-cloud service composition. The multi-cloud interoperability offers a large-scale architecture to manage the interconnection of the cloud providers with each other. By reducing cloud providers in the selection and composition of appropriate services, interoperability is managed and data migration can be applied effectively with high scalable coverage.

6 Conclusion and future work

This paper presented a hybrid formal verification approach to analyze cloud service composition problem in the multi-cloud environment. Also, a Multi-Cloud Service Composition (MCSC) approach was presented to decrease the number of cloud providers to gain final service composition with a high level of Quality of Service (QoS). The presented approach provided behavioral modeling to examine the procedure of user' requests, service selection, and composition in a multi-cloud environment. Also, the presented approach permitted the analysis of the service composition using a Multi-Labeled Transition Systems (MLTS)-based model checking and Pi-Calculus-based process algebra methods for monitoring some functional specifications and non-functional specifications as the QoS standards. In addition, the proposed verification approach satisfied the functional specifications. The experimental results supported the feasibility of the proposed approach with performance evaluations and some confirmation set-ups. We observed that the verification time of the MLTS_MCSC method was lower than the other methods according to the satisfaction of existing specification rules. Also, the proposed MLTS_MCSC method has a minimum memory consumption than other methods. To evaluation of execution time factor, we observed that the execution time of the MLTS_MCSC method is lower than the LTS_MCSC and CMSC methods. In addition, the MLTS_MCSC method selects minimum cloud providers to compose the examined cloud service in multiple clouds that can decrease the communication time and energy consumption. In the future work, we will try to use meta-heuristic algorithms in the model checking approach for avoiding the state space explosion problem in a highly scalable multi-cloud environment with a huge number of requested services. Also, a general computational intelligence design framework can be utilized to produce the smart design process of the specification rules informs of CTL and LTL formulas.

References

1. Maamar, Z., et al.: Towards a seamless coordination of cloud and fog: illustration through the internet-of-things. In Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, pp. 2008–2015. ACM, Limassol, Cyprus (2019)
2. Shojafar, M., et al.: Recent advances in cloud data centers toward fog data centers. Concurr. Comput. **31**(8), e5164 (2019)
3. Buyya, R., Broberg, J., Goscinski, A.M.: *Cloud Computing: Principles and Paradigms*, vol. 87. Wiley, Hoboken (2010)
4. Tajiki, M.M., et al.: CECT: computationally efficient congestion-avoidance and traffic engineering in software-defined cloud data centers. Clust. Comput. **21**(4), 1881–1897 (2018)
5. Aceto, G., et al.: Cloud monitoring: a survey. Comput. Netw. **57**(9), 2093–2115 (2013)
6. Stergiou, C., et al.: Secure integration of IoT and cloud computing. Future Gener. Comput. Syst. **78**, 964–975 (2018)
7. Ghobaei-Arani, M., Souri, A.: LP-WSC: a linear programming approach for web service composition in geographically distributed cloud environments. J. Supercomput. **75**(5), 2603–2628 (2019)
8. Simon, B., Goldschmidt, B., Kondorosi, K.: A metamodel for the web services standards. J. Grid Comput. **11**(4), 735–752 (2013)
9. Piprani, B., Sheppard, D., Barbir, A.: Comparative analysis of SOA and cloud computing architectures using fact based modeling. In: Proceedings of the OTM Confederated International Conferences on the Move to Meaningful Internet Systems. Springer (2013)
10. Portchelvi, V., Venkatesan, V.P., Shanmugasundaram, G.: Achieving web services composition–a survey. Softw. Eng. **2**(5), 195–202 (2012)
11. Brahmi, Z., Faten, M.: Service composition in a multi-cloud environment based on cooperative agents
12. Barkat, A., Okba, K., Bourekkache, S.: Service composition in the multi cloud environment. Int. J. Web Inf. Syst. **13**(4), 471–484 (2017)
13. Keshanchi, B., Souri, A., Navimipour, N.J.: An improved genetic algorithm for task scheduling in the cloud environments using the

priority queues: formal verification, simulation, and statistical testing. J. Syst. Softw. **124**, 1–21 (2017)
14. Naseri, A., Navimipour, N.J.: A new agent-based method for QoS-aware cloud service composition using particle swarm optimization algorithm. J. Ambient Intell. Hum. Comput. **10**, 1851–1864 (2018)
15. Ghobaei-Arani, M., et al.: CSA-WSC: cuckoo search algorithm for web service composition in cloud environments. Soft Comput. **22**, 8353–8378 (2017)
16. Imran, M., et al.: Formal verification and validation of a movement control actor relocation algorithm for safety–critical applications. Wireless Netw. **22**(1), 247–265 (2016)
17. Dumez, C., et al.: Model-driven approach supporting formal verification for web service composition protocols. J. Netw. Comput. Appl. **36**(4), 1102–1115 (2013)
18. Diekmann, C., et al.: Verified iptables firewall analysis and verification. J. Autom. Reason. **61**(1), 191–242 (2018)
19. Ghobaei-Arani, M., et al.: A moth-flame optimization algorithm for web service composition in cloud computing: simulation and verification. Software **48**(10), 1865–1892 (2018)
20. Souri, A., Rahmani, A.M., Jafari Navimipour, N.: Formal verification approaches in the web service composition: a comprehensive analysis of the current challenges for future research. Int. J. Commun. Syst. **31**(17), 3808 (2018)
21. Souri, A., Navimipour, N.J., Rahmani, A.M.: Formal verification approaches and standards in the cloud computing: a comprehensive and systematic review. Comput. Stand. Interfaces **58**, 1–22 (2018)
22. Amato, F., Moscato, F.: Model transformations of MapReduce Design Patterns for automatic development and verification. J. Parallel Distrib. Comput. **110**, 52–59 (2017)
23. Souria, A., Sharifloua, M.A., Norouzia, M.: Analyzing SMV & UPPAAL model checkers in real-time systems. Comput. Sci. **1**, 631–639 (2012)
24. Frenkel, H., Grumberg, O., Sheinvald, S.: An automata-theoretic approach to model-checking systems and specifications over infinite data domains. J. Autom. eason. **63**, 1077–1101 (2018)
25. Yu, B., et al.: Verifying temporal properties of programs: a parallel approach. J. Parallel Distrib. Comput. **118**, 89–99 (2018)
26. Gao, H., et al.: Research on cost-driven services composition in an uncertain environment. J. Internet Technol.y **20**(3), 755–769 (2019)
27. Li, Y., Yao, X.: Cloud manufacturing service composition and formal verification based on extended process calculus. Adv. Mech. Eng. (2018). https://doi.org/10.1177/1687814018781287
28. Bourne, S., Szabo, C., Sheng, Q.Z.: Transactional behavior verification in business process as a service configuration. IEEE Trans. Serv. Comput. **12**(2), 290–303 (2019)
29. Souri, A., et al.: Formal modeling and verification of a service composition approach in the social customer relationship management system. Inf. Technol. People (2019). https://doi.org/10.1108/ITP-02-2018-0109
30. Ghobaei-Arani, M., et al.: A moth-flame optimization algorithm for web service composition in cloud computing: simulation and verification. Software **48**, 1865–1892 (2018)
31. Mezni, H., Sellami, M.: Multi-cloud service composition using formal concept analysis. J. Syst. Softw. **134**, 138–152 (2017)
32. Entezari-Maleki, R., et al.: Modeling and evaluation of service composition in commercial multiclouds using timed colored petri nets. In: IEEE Transactions on Systems, Man, and Cybernetics: Systems. pp. 1–15 (2017)
33. Rai, G.N., et al.: Web service interaction modeling and verification using recursive composition algebra. IEEE Transactions on Services Computing. pp. 1–1 (2018)
34. Khai, H.T., Thang, B.H., Tho, Q.T.: One size does not fit all: logic-based clustering for on-the-fly web service composition and verification. Int. J. Web Grid Serv. **14**(3), 237–272 (2018)
35. Saeed, S., et al.: A location-sensitive and network-aware broker for recommending Web services. Computing **101**(5), 455–475 (2019)
36. Wang, H., et al.: Integrating modified cuckoo algorithm and creditability evaluation for QoS-aware service composition. Knowl. Based Syst. **140**, 64–81 (2018)
37. Souri, A., et al.: A symbolic model checking approach in formal verification of distributed systems. Human Centric Comput. Inf. Sci. **9**(1), 4 (2019)
38. Gyftopoulos, S., Efraimidis, P.S., Katsaros, P.: Formal analysis of DeGroot Influence Problems using probabilistic model checking. Simul. Model. Pract. Theory **89**, 144–159 (2018)
39. Arapinis, M., et al.: Statverif: verification of stateful processes. Journal of Computer Security **22**(5), 743–821 (2014)
40. Dardha, O., Gay, S.J.: A new linear logic for deadlock-free session-typed processes. In: International Conference on Foundations of Software Science and Computation Structures. Springer (2018)
41. Ryan, M.D., Smyth, B.: Applied pi calculus. J ACM **65**, 1 (2011)
42. Rodriguez-Mier, P., et al.: An integrated semantic web service discovery and composition framework. IEEE Trans. Serv. Comput. **9**(4), 537–550 (2016)
43. Souri, A., Jafari Navimipour, N.: Behavioral modeling and formal verification of a resource discovery approach in Grid computing. Expert Syst. Appl. **41**(8), 3831–3849 (2014)
44. Souri, A., et al.: A model checking approach for user relationship management in the social network. Kybernetes **48**(3), 407–423 (2019)
45. Zhao, X., et al.: An improved discrete immune optimization algorithm based on PSO for QoS-driven web service composition. Appl. Soft Comput. **12**(8), 2208–2216 (2012)
46. Mardukhi, F., et al.: QoS decomposition for service composition using genetic algorithm. Appl. Soft Comput. **13**(7), 3409–3421 (2013)
47. Entezari-Maleki, R., et al.: Modeling and Evaluation of Service Composition in Commercial Multiclouds Using Timed Colored Petri Nets. In: IEEE Transactions on Systems, Man, and Cybernetics: Systems (2017)
48. Arunkumar, G., Venkataraman, N.: A novel approach to address interoperability concern in cloud computing. Proc. Comput. Sci. **50**, 554–559 (2015)
49. Rezaei, R., et al.: A semantic interoperability framework for software as a service systems in cloud computing environments. Expert Syst. Appl. **41**(13), 5751–5770 (2014)
50. Fatma, L., Haithem, M.: Multicloud service composition: a survey of current approaches and issues. J. Softw. **30**(10), e1947 (2018)

Alireza Souri received his B.S. degree in Software Engineering from University College of Nabi Akram, Iran, and his M.Sc. and Ph.D. degrees in Software Engineering from Science and Research Branch, Islamic Azad University, Iran. He is a researcher and lecturer at Islamic Azad University. Up to now, he has authored/co-authored 30 academic articles. He served on the program committees and the technical reviewer of several ISI-index journals and international conferences. He currently is an Associate Editor member of Human-Centric Computing and Information Sciences (Springer), Cluster Computing (Springer) and IET Communications (IEEE) journals. His research interests include Formal Specification & Verification, Model checking, Grid & Cloud computing, IoT and Social networks. Now, He is a member of The Society of Digital Information and Wireless Communications.

Amir Masoud Rahmani received his B.S. in Computer Engineering from Amir Kabir University, Tehran, in 1996, the M.S in Computer Engineering from Sharif University of Technology, Tehran, in 1998 and the Ph.D. degree in Computer Engineering from IAU University, Tehran, in 2005. Currently, he is a Professor in the Department of Computer Engineering at the IAU University. He is the author/co-author of more than 190 publications in technical journals and conferences. His research interests are in the Areas of Distributed Systems, Ad hoc and Wireless Networks and Evolutionary Computing.

Nima Jafari Navimipour received his B.S. in computer engineering, software engineering, from Tabriz Branch, Islamic Azad University, Tabriz, Iran, in 2007; the M.S. in computer engineering, computer architecture, from Tabriz Branch, Islamic Azad University, Tabriz, Iran, in 2009; the Ph.D. in computer engineering, computer architecture, from Science and Research Branch, Islamic Azad University, Tehran, Iran in 2014. He is an assistance professor in the Department of Computer Engineering at Tabriz Branch, Islamic Azad University (the world's third largest university), Tabriz, Iran. He has published more than 60 papers in various journals and conference proceedings. His research interests include Cloud Computing, Social Networks, Fault-Tolerance Software, Knowledge Management, Evolutionary Computing, and Formal Verification.

Reza Rezaei received his B.S. in computer engineering, software engineering, from Central Tehran Branch, Islamic Azad University, Tehran, Iran, in 2001; the M.S. in computer engineering, software engineering, from Science and Research Branch, Islamic Azad University, Tehran, Iran, in 2006; the Ph.D. in computer engineering, software engineering, from University of Malaya, Kuala Lumpur, Malaysia in 2014. He is assistance professor in the Department of Computer Engineering at Saveh Branch, Islamic Azad University, Saveh, Iran. He has published more than 50 papers in various journals and conference proceedings. His research interests include Software Engineering, Software Architecture, Enterprise Architecture, and Ultra Large scale Systems.