Modeling and Evaluation of Service Composition in Commercial Multi-Clouds using Timed Colored Petri Nets

Reza Entezari-Maleki, Sayed Ehsan Etesami, Negar Ghorbani, Arian Akhavan Niaki, Leonel Sousa, *Senior Member, IEEE,* and Ali Movaghar, *Senior Member, IEEE*

***Abstract*—The increasing demand for Web services encourages commercial cloud service providers to publish their own services with various functional and non-functional capabilities in different cloud platforms. The aggregation of atomic services from multiple service repositories is the main idea of the service composition concept in multi-clouds. The cloud Web service composition is a suitable way for satisfying users' complex requests by integrating services from different clouds in order to create a new value-added composite service. The time required to serve a composite service by a multi-cloud environment is an important parameter, which depends on different factors, ranging from the service composition and selection algorithm to the number of atomic services published in the clouds. In this paper, a model based on Timed Colored Petri Nets (TCPNs) is proposed to evaluate the service composition in multi-cloud environments while minimizing the number of clouds involved in serving a composite service request. The proposed TCPN graphically models the process of request submission, composite service analysis, service selection, and service provisioning in a multi-cloud environment. It also assesses both the mean response time of the environment and the probability of dropping composite requests. The verification of the accuracy of the proposed model is done by comparing the results obtained from the TCPN model, in two different scenarios, with the results from the CloudSim framework. These results confirm that our proposed TCPN model can appropriately model the system and evaluate its performance more efficiently than the CloudSim.**

***Index Terms*—Web service, service composition, multi-clouds, modeling, colored Petri net.**

I. Introduction

Cloud computing is an elastic service provisioning model which enables on-demand rapid access to a shared pool of computing resources and large storage facilities [1], [2]. It is revolutionizing the entire IT ecosystem and all aspects of our lives. It brings not only the technical change but also impacts enterprise business applications and business models [3], [4]. Nowadays, cloud computing, as a Web and Internet-based computing model, is increasingly used to provide users with software resources in the form of Web services. This makes cloud computing a prominent platform for providing Web services. A Web service is a modular, self-describing, self-contained, well-defined, and Web-accessible software component which is callable with standard Web technologies [5]–[8]. It can be made available by a service provider and invoked by service requesters over the Internet. Due to the vast interest in Web services, various types of these services have been published by providers and made available to customers. Nowadays, most companies and organizations prefer to keep only the main business in-house and outsource other application services over the Internet [6], [7]. For example, large companies such as Google, YouTube, Amazon, Flickr, Twitter, Facebook, and eBay, have offered Web services to provide users with access to their resources and services. According to the statistics from ProgrammableWeb [9] and Nordic APIs [10], two well-known Web service publication Websites, the number of Web services has grown through the recent years, from about $2,500$ Web services in 2010 to about $10,000$ Web services in 2013, while there were $15,000$ available Web services reported on the Web at the end of 2016, and this number is expected to grow in the near future. The growth among the largest cloud Web service providers remains fast because of the revenues. Amazon Web Service (AWS), ranked first in Gartner's Magic Quadrant for cloud infrastructure services in 2016 [11], followed by Microsoft, IBM, and Google, which contributes about 35% of Amazon's valuation (121.8 $\$Bil$ of 349.4 $\$Bil$). AWS is expected to contribute to about 12% of Amazon's total revenue by 2020, while this amount was 8.6% in 2016 [12].

Although a Web service is beneficial for both users and providers, in many cases, a single Web service cannot satisfy all requirements of a complex request raised by a user. This is the main motivation for the concept of service composition, which focuses on the design of a new value-added coarse-grain composite service that incorporates existing fine-grain atomic services [2], [4], [7], [8], [13]. In this concept, service components from different providers can be integrated into a composite service to serve a complex request, regardless of their locations, platforms, and performance. Determining the atomic services required in order to satisfy a complex request, selecting appropriate services from a service pool, addressing service composition restrictions, considering important Quality of Service (QoS) attributes during service composition and service selection, and handling the rapid changes in services

R. Entezari-Maleki is with the School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran (e-mail: entezari@ipm.ir).

S. E. Etesami is with the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran.

N. Ghorbani is with the School of Information and Computer science, Informatics Department, University of California, Irvine, CA.

A. A. Niaki is with the Computer Science Department, Stony Brook University, Stony Brook, NY.

L. Sousa is with the INESC-ID, Instituto Superior Tecnico, Universidade de Lisboa, Lisbon, Portugal.

A. Movaghar is with the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran.

and requests, are some important challenges and considerations which need to be addressed in the service composition problem to fully satisfy a user [14]–[18]. Most of the methods proposed to solve the service composition problem assume that all Web services found in a composition sequence, originate from the same service repository. However, it is very common for service providers to publish their own Web services at different clouds, which has distinct advantages, such as adaptivity, scalability, and transparency of load scheduling [7], [8]. This leads to an emerging topic in Web service composition which is referred to as service composition in multi-cloud environments [2], [4], [7], [8], [19].

The solutions proposed to solve the problem of Web service composition in multi-clouds should not only satisfy user requested QoS requirements, but it should also minimize the number of clouds involved in serving the composite request of a user. The service composition satisfying QoS requirements is an NP-hard optimization problem [8], [14], [15], [20], [21], and taking the details of multi-cloud environments into account for minimizing the number of clouds involved in serving a composite service, makes it even more difficult to solve. There are several approaches to solve the service composition and service selection problems in cloud systems. The main optimization criteria considered in the traditional service composition problem are some QoS attributes which are important for users (e.g, time, cost, security, reliability, etc.) [13], [18], [19], [22], [23]. Moreover, the service composition problem in multi-clouds has been analyzed via cloud federation. The aim of most of the research presented in this area is to satisfy user expected QoS, while the number of clouds involved in serving the request is minimized [2], [4], [7], [8]. Among the approaches proposed to model the process of service composition in clouds [13], [24], [25], none can formally represent the process of service composition in multi-clouds and simultaneously evaluate its performance considering the specific characteristics of the environment. To fulfill this requirement, a model based on Timed Colored Petri Nets (TCPNs) is presented herein to model and evaluate service composition in multi-clouds. TCPN is an extension of Petri Net formalism introduced by Jensen [26], which is suitable for modeling and analysis of distributed and concurrent systems. Colored Petri Nets (CPNs) provide the feature of defining a data structure for each token type, which helps us to distinguish tokens and conveniently model and analyze large and complex systems. Timed extension of CPNs enable analysis of time relationships (e.g. earlier than, later than, etc.).

The aim of the TCPN model proposed herein is to evaluate the mean response time of a multi-cloud environment for composite service requests, while minimizing the number of involved clouds. To achieve this goal, we model the process of request submission, request analysis, service selection, and service provisioning inside a cloud within the multi-cloud environment. In addition to computing the mean response time, the proposed TCPN represents the workflow of service composition and service selection in a multi-cloud environment. Using our proposed model, we can also compute the probability of dropping requests, which may be rejected by the system due to overloads or the lack of sufficient providers upon request arrival. The proposed model is applied to two sample multi-clouds and the results obtained from the CPN Tools [27], which is used as a computer tool to analyze the proposed TCPN model, are compared against the simulation results obtained by the CloudSim framework [28]. The validation results show that the proposed TCPN model can appropriately assess both mean response time and the dropping probability of composite services in multi-cloud environments.

The remainder of this paper is organized as follows. A literature review in the field of Web service composition and service composition in Geo-distributed clouds is given in Section II. In Section III, the system description and some preliminaries about service composition are provided; moreover, the main assumptions about the reference architecture considered in this paper are also given in Section III. Section IV proposes the TCPN model for the cloud Web service composition with details. Section V presents the numerical results obtained from the CPN Tools and CloudSim framework for two different multi-cloud environments. Finally, Section VI concludes the paper and presents future work.

II. Related Work

Cloud Web services and service composition are both emerging topics in the field of cloud and service computing. Most of the research efforts were devoted to the investigation of the optimal composition and selection of Web services, while some other research focused on the evaluation of performance measures and QoS attributes.

A. Service Composition Considering QoS Attributes

Wu et al. [16] have proposed a service composition method that maximizes the overall QoS while meeting user-specified global QoS constraints to overcome the shortcomings of traditional methods. In [16], the concept of generalized component services was presented for the purpose of expanding the candidate space for service selection in order to achieve a better solution. Deng et al. [29] have studied the problem of service selection and composition in mobile communications while both clients and service providers are moving. To this end, they first proposed a mobile service provisioning architecture, and then presented a service composition approach. The efficiency of the approach proposed in [29] was evaluated through a simulation tool built by the authors. Klai et al. [30] have used Symbolic Observation Graphs (SOG) to check the correction of service composition in clouds regarding event- and state-based LTL. They enriched the verification process of the proposed model by a diagnostic that allows predicting the elasticity of the resource provider service.

Ding et al. [31] have proposed a genetic-based algorithm that optimally solves the service selection problem while considering transactional properties influencing the QoS of the transactional composite Web service. The genetic-based service selection algorithm presented in [31] takes the execution time, price, transactional property, stability, and penalty-factor into consideration to reach a globally optimal service selection. According to [15], the problem of Web service composition satisfying users QoS constraints is a core part of the cloud manufacturing. To address this problem, Chen et al. [15] have proposed a new method based on multi-objective optimization to help users make a flexible decision. Therefore, the Web service composition in cloud manufacturing system was formulated by a multi-objective optimization model, and then an evolutionary algorithm was developed based on this model. Lu et al. [32] have studied knowledge-based service composition and resource planning in a cloud manufacturing system, with the aim of developing an integrated networking environment to quickly allocate resources while considering the policies imposed on requests and resources by users and providers, respectively. The approach proposed in [32] utilizes distributed knowledge for intelligent service composition and adaptive resource planning, which allows accurate mapping between distributed manufacturing resources and dynamic service requests.

B. Service Composition in Multi-clouds

Zou et al. [7] have proposed three different methods to select a cloud combination, not only for finding a feasible service composition sequence but also for involving the minimum numbers of clouds. Experimental results show that the proposed method based on artificial intelligence can appropriately find sub-optimal cloud combinations. According to the methods presented in [7], Kurdi et al. [2] have proposed a combinatorial optimization algorithm for service composition in a multiple cloud domain, which selects the cloud with the maximum number of atomic services for providing services to complete composite service requests with minimum overhead. Yu et al. [8] have presented two algorithms to select service combinations involving the minimum number of clouds from a multi-cloud environment. The first algorithm proposed in [8] is a greedy algorithm and the other is a heuristic algorithm based on ant colony optimization. The results show that the heuristic algorithm based on the ant colony optimization approach can effectively find cloud combinations with the minimum number of clouds.

Wang et al. [4] have proposed a composition model taking both QoS of services and cloud network environment into consideration. Moreover, they proposed a genetic-based approach for the Web service composition problem in Geo-distributed clouds with the aim of minimizing the Service Level Agreement (SLA) violations. Li et al. [19] have investigated the service selection problem under the service replica limitation constraint in Geo-distributed clouds. They have proposed a service selection algorithm that estimates the communication latency with the network coordinate system, and then finds the services resulting in low latency under replica limitations.

C. Modeling and Evaluation of Service Composition

Bao et al. [13] used Finite State Machine (FSM) to prescribe the correct invokation order of Web services. Moreover, a tree-pruning-based algorithm was proposed in [13] to create the Web service composition tree and generate feasible execution paths, which helps with the optimal path selection. Although FSMs can be used to model Web service composition, they are unable to analyze the efficiency of the composition and require an additional algorithm to assess the performance of each feasible composition. On the other hand, all FSMs corresponding to a Web service composition should be constructed for all possible compositions before being able to solve the problem. Therefore, FSMs do not seem to be a good choice for analyzing the performance of Web service composition. Liu et al. [24] have modeled and analyzed the dynamic execution of service compositions using Petri nets, while taking into account the effect of reliability on performance. The proposed model mostly uses the probabilities that an atomic service is invoked successfully and tries to estimate the probability of a composite service to be successfully invoked. Since transitions of the Petri net proposed in [24] are simple transitions without any time notion assigned, it cannot be used for quantitative analysis. This kind of Petri nets can be used to construct a graph showing the precedence of actions, which is useful in modeling Web service composition, but cannot be used to evaluate the performance by dynamically analyzing the Petri net. Other timed and stochastic extensions of Petri nets can not only show the order of actions, but also analyze the system by solving the net.

Ma et al. [25] have addressed the problem of service discovery in a cloud system by proposing a formal model for services named Abstract State Services (ASSs), which is based on the Abstract State Machines (ASMs) model. The model helps users to conduct a Web search for usable services, extract service components, and recompose the components. The proposed ASS model contains a finite set of services, and a service composition is a selection of atomic services among all services existing in a database. The formalism applied in [25] is useful for formalizing the notion of sequential and parallel actions, but it cannot be used for the purpose of quantitative analysis. In order to analyze the performance of service composition, we should introduce the time notation into the model which was not considered in [25]. Wang et al. [33] have presented a QoS-aware service selection approach, based on the cloud model, to guarantee reliability and real-time requirements. It was used as the basis of a QoS-aware service selection approach solved by mixed integer programming. The effectiveness of the proposed approach in [33] was evaluated by both real-world and randomly generated Web service QoS datasets.

Abdullah et al. [34] have proposed an agent-based model for Web service composition to construct the Service Dependency Graph (SDG). The model was identified as a subgraph of the SDG which captures the functional profile of the web service together with its direct dependencies to other services. Afterwards, an agent-based algorithm was proposed to compose a Web service upon receiving a request from the user. The number of messages transferred to serve a composite service was introduced and evaluated as a performance measure. Chen et al. [35] have proposed a Pareto set model for the QoS-aware Web service composition problem. Using the proposed model, six categories of QoS attributes, according to their different types of aggregation pattern, were studied. Afterwards, by taking advantage of pruning candidates while considering dominance relationships and constraint validations at candidate level, a distributed partial selection algorithm was proposed. This algorithm can find the optimal compositions by partial selection and composing the potential candidates. Ding et al. [36] have conducted the performance evaluation of transactional composite Web services. In order to address the performance evaluation problem, transactional properties for both single and composite Web services were introduced, and the performance of basic workflow patterns, i.e., sequential, parallel, selectable, and loop, was assessed.

III. System Model

According to the definition provided by IBM, "Web services are a set of emerging standards enabling interoperable integration between heterogeneous IT processes and systems. They can be considered as a new breed of a Web application that is self-contained and self-describing, and it can provide functionality and interoperation ranging from the basic to the most complicated business and scientific processes" [37]. Other definitions and interpretations of a Web service can also be found in the literature, but according to most of the definitions, the main roles involved in Web services are *service provider*, *service registry*, and *service requester*, which correspond to *publishing*, *finding*, and *invoking* Web services, respectively [6]. Although a single Web service can serve some specific and predefined requests of a user, it cannot fully satisfy all complex requests raised by users nowadays. In order to respond to more complex requests, the Web service composition technology has been proposed and widely used by the academy and industry [6], [14], [22], [38]. Web service composition is the process of aggregating multiple services into a single service, which helps users to reach large-granularity and value-added composite services [2], [6], [7], [13], [38]. To show how to reply to a complex request using the combination of Web services instead of using multiple single services separately, we use the classical example of trip planning. Suppose a user wishes to go for a trip and needs to book a flight and train tickets. Furthermore, the user wants to rent a taxi and book a hotel in the destination. In this scenario, all these activities need to be checked by the user individually using separate Web services, but a travel agent service can satisfy the user's request by composing different Web services together. Fig. 1 schematically shows the composition of Web services for this example.

Cloud computing is considered as a prominent platform for providing Web services [4], [8], [14]. By increasing the number of cloud service users worldwide, major cloud service providers have been deploying and operating geographically dispersed data centers to better serve the distributed cloud users [4], [21]. At the same time, the Web services provided on clouds grow rapidly. The complex requests submitted by distributed cloud users may need to be replied by different Web service providers in various clouds. Therefore, the concept of multi-clouds have been introduced and recently have attracted much attention. The concept can be found in the literature as multiple clouds, multi-clouds, Geo-distributed clouds, or cloud federation [2], [4], [7], [8], [13], [19], [20]. The multi-cloud Web service composition is an emerging topic trying to respond to complex requests of cloud users by aggregating

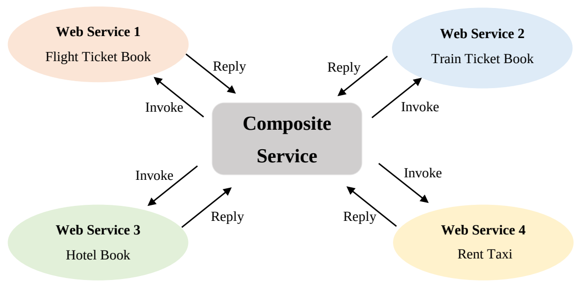

Fig. 1. A simple example of Web service composition

different Web services provided by probably different service providers on various distributed clouds.

A simple architecture of a multi-cloud environment in which different service providers publish their own Web services is shown in Fig. 2. In this figure, there are four different commercial cloud computing platforms, named C_1, C_2, C_3, and C_4, and three different service providers, named *A*, *B*, and *C*. Although there are connections between service providers and clouds in real systems, they are ignored in Fig. 2 to simplify the figure. A service provider may publish different services in a cloud or even publish the same service in different clouds. After publishing a service, it is required that the provider contacts the *service registry* component to inform it about the location of each service. The *service registry* is a repository keeping the required information about the location and capabilities of each service and its relevant provider. It can help the *cloud combiner* component to find the appropriate clouds containing the service requested. The *service requesters*, also named *users*, first provide initial and goal descriptions of the service composition request. Next, the *composition converter* component analyzes the complex request to recognize the required atomic services. Afterwards, it contacts the *cloud combiner* to select appropriate cloud combination from the multi-cloud environment using a predefined combinatorial selection method. Then, the *composition converter* informs the *cloud combiner* of the selected cloud combination, asking to execute the service composition sequence, which can satisfy the goals of the service composition requester.

According to the system model described above, which is compatible with the structures given in [2], [6]–[8], [14], [39], the assumptions considered in the proposed model are as follows.

- The multi-cloud environment is considered as a set of $\mathcal{N}$ clouds, numbered from C_1 to $C_{\mathcal{N}}$, each of which containing a set of service files denoted by $F_i = \{SF_1, SF_2, \cdots, SF_{fi}\}$ for cloud C_i. Each service file includes a set of atomic services as it can be seen in Fig. 2.
- It is possible for a cloud to have different copies of a service among various service files offered by the same or even different service providers. For example, two copies of service S_2 exist in two different service files of cloud C_1, named SF_2 and SF_3, in the multi-cloud shown in Fig. 2. Hence, the set of all services provided by cloud

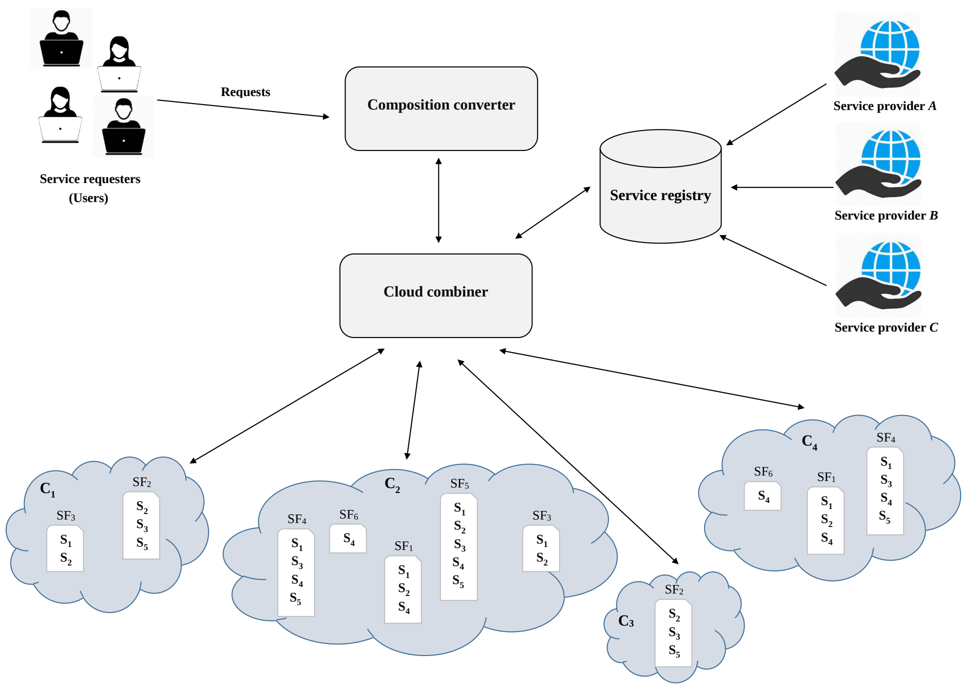

Fig. 2. The architecture of Web service composition in a multi-cloud environment

C_i, without considering the concept of service file, could be represented by $S_{C_i} = \{S_1^i, S_2^i, \cdots, S_{si}^i\}$, where si is the number of different service types provided in C_i.

- Since a specific service can be provided in different service files of a cloud, there is a capacity for each service in a cloud to represent the maximum instances of the service provided in the cloud. We name this capacity as q_j^i for service S_j provided in cloud C_i. For example, in Fig. 2, we have $q_2^1 = 2$, which shows two instances of service S_2 are provided in cloud C_1. It is worth mentioning that $\sum_{j=1}^{si} q_j^i$ is the number of all services provided in C_i, and consequently, $\mathcal{S} = \sum_{i=1}^{N} \sum_{j=1}^{si} q_j^i$ denotes the number of all services provided in the multi-cloud environment.
- There are $\mathcal{M}$ users in the system, each user submits a composite request to the *composition converter*. The composite request of a user (e.g., *user i*) is replied by $\mathcal{K}$ different atomic services provided by $\mathcal{C}$ different clouds, where $\mathcal{C} \leq \mathcal{K}$.
- For the Web service composition problem considered in this paper, the solution satisfying the composite request of a user is a sequence of tuples, $< S_i, C_\alpha >$, $< S_j, C_\beta >$, $\cdots$, $< S_\mathcal{K}, C_\mathcal{C} >$, where S_i, S_j, $\cdots$, $S_\mathcal{K}$ are atomic services, and C_α, C_β, $\cdots$, $C_\mathcal{C}$ are their corresponding clouds selected to serve the user.
- The only way to submit a composite request to the environment is delivering it to the *composition converter* component. This component is responsible for receiving composite requests from users, decomposing and analyzing these requests, specifying the service composition sequences, and contacting other relevant components of the environment to serve the composite requests.

IV. The Proposed TCPN Model

This section presents the TCPN proposed to model Web service composition in multi-cloud environments. To evaluate the proposed model, we use the well-known CPN Tools [27], which enables modeling, verification, and analysis of CPN models. As an industrial-strength computer tool, the CPN Tools helps the modeler to graphically construct the models, and to analyze them in steady-state. However, because of the inherent constraints of the tool, the modeler needs to change the original model to be able to make use of the CPN Tools. Due to the tool constraints and in order to facilitate the understanding of the model, we first introduce the model in its abstract form in Section IV-A, and then change some parts of it and propose the low-level model, in Section IV-B, that can be modeled and analyzed with the CPN Tools.

A. Abstract View of the Proposed TCPN Model

The high-level TCPN model proposed for the cloud Web service composition is shown in Fig. 3. As it can be seen in this figure, each component of the system, corresponding to the components shown in Fig. 2, is surrounded by a dashed line box. The left most component is the *service requester*, which models the arrival process of requests to the environment. The place P_{Start} represents the potential requests from the clients. There are $\mathcal{M}$ tokens in this place at the beginning, which shows the number of potential users who can submit composite requests. The existence of a token in place P_{Start} enables timed transition T_{Arrive}. After enabling, it takes some time for transition T_{Arrive} to fire. The firing time of this transition, which shows the inter-arrival time of composite requests, is considered to follow an exponential distribution with the rate λ. The assumption of considering exponential distribution

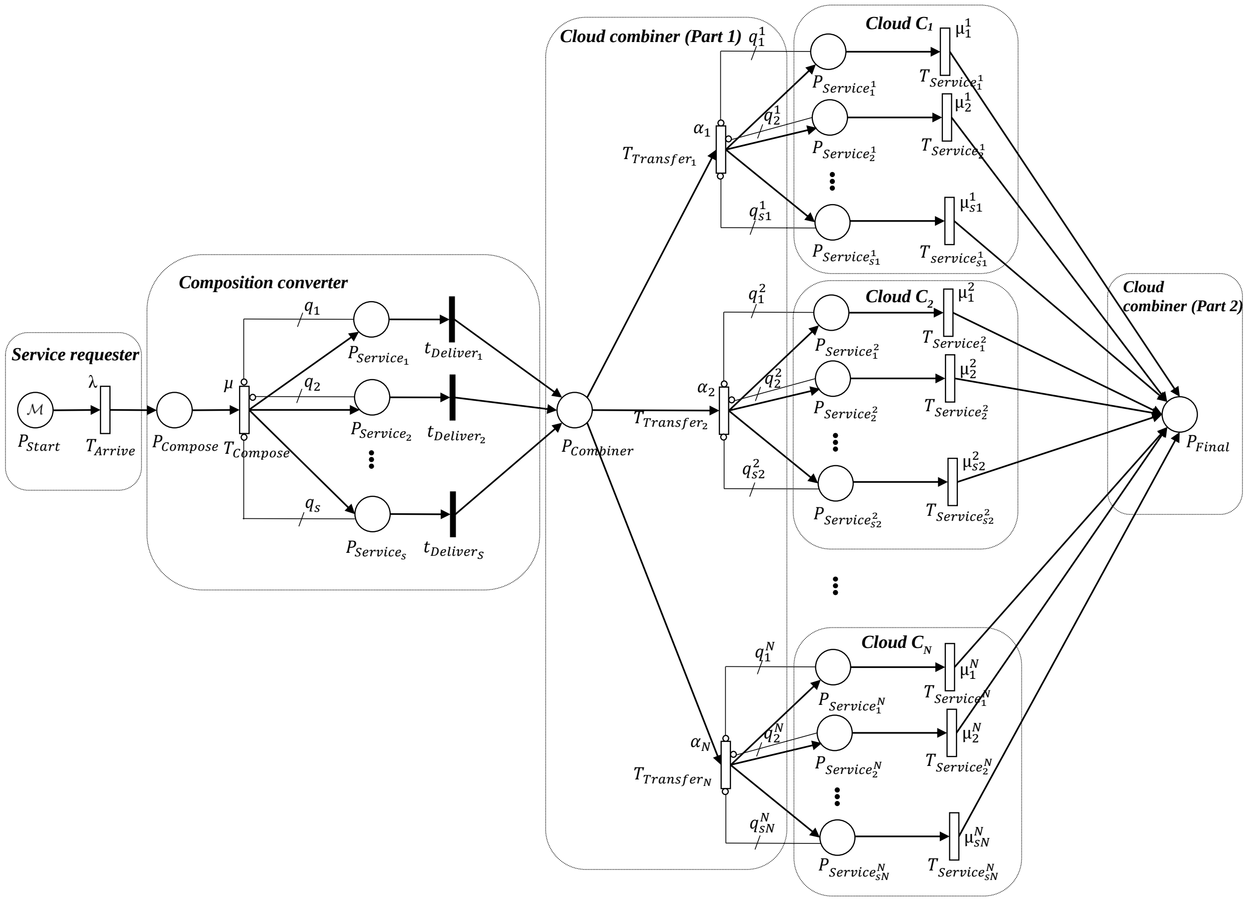

Fig. 3. The high-level TCPN model of cloud Web service composition

for the inter-arrival time between two consecutive requests is an acceptable assumption, adopted in many research work in this area [40]–[45]. In addition to the inter-arrival times of requests, the service time of requests and data transmission time are considered to follow exponential distributions, which is also compatible with the previously presented analytical models in the related art [40]–[44].

Once timed transition T_{Arrive} fires, a token is removed from place P_{Start} and deposited into place $P_{Compose}$, which means a request is submitted to the *composition converter*. As mentioned in Section III, the component *composition converter* is responsible for receiving and analyzing users' composite requests in order to specify the number/types of atomic services of each composite service. This component's task is modeled by timed transition $T_{Compose}$. The firing time of this transition follows an exponential distribution function with the mean $1/\mu$, which represents the mean time of serving a composite service request which is defined as the time to analyze the composite request and recognize the atomic services required for satisfying the composite request. Suppose that a composite request requires $\mathcal{K}$ different services named $S_1, S_2, \cdots, S_{\mathcal{K}}$ to be served. To model this, transition $T_{Compose}$ removes a token from place $P_{Compose}$ and puts $\mathcal{K}$ tokens in places $P_{Service_1}$ to $P_{Service_{\mathcal{K}}}$, a single token per place. As shown in Fig. 3, there is an inhibitor arc from each place $P_{Service_j}$, $1 \le j \le \mathcal{S}$, to transition $T_{Compose}$. These arcs correspond to the maximum instances of each service type provided by the environment. For example, the number q_1, specified as the multiplicity of the inhibitor arc between place $P_{Service_1}$ and transition $T_{Compose}$, represents that there are in total q_1 instances of service S_1 in the multi-cloud environment. More precisely, $q_1 = \sum_{i=1}^{\mathcal{N}} q_1^i$, where $\mathcal{N}$ is the number of all clouds in the environment. If there are more than q_1 requests for service S_1 in a time instant, q_1 requests are dispatched among the clouds and the remaining are dropped from the system. It is worth mentioning that dropping only one atomic service of a composite service causes the whole composite service to be dropped. Therefore, the *dropping probability* of composite services can be computed by counting the number of tokens dropped at this point.

The existence of a token in place $P_{Service_j}$ shows that the service S_j should be served by one of the clouds offering this service. By adding a token to each place $P_{Service_j}$, its related immediate transition, named $t_{Deliver_j}$, is enabled and can fire. The firing time of an immediate transition is *zero*, so these types of transitions are used to model immediate actions. Once transition $t_{Deliver_j}$ fires, a token is removed from place $P_{Service_j}$, and deposited into place $P_{Combiner}$. This process shows that the *composition converter* component has finished its task, and delivered the specification of the required services to the *cloud combiner* component. Therefore, there will be $\mathcal{K}$ tokens with different colors (services) inside place $P_{Combiner}$, which should be dispatched among the clouds providing the services. The *cloud combiner* has enough information about the specification of the services provided in each cloud. Hence, it can figure out the proper cloud to select for serving the requested service.

Without loss of generality, we assume that the clouds C_1, C_2, $\cdots$, $C_{\mathcal{N}}$ shown in Fig. 3 are sorted descendingly according to the number of service types they provide, i.e. the number of the service types provided in cloud C_i is greater than or equal to that of cloud C_{i+1}. Considering this assumption, if both clouds C_i and C_{i+1} offer the service S_j, we assign higher priority to C_i. This simple assumption helps us to straightforwardly model the system and find the set of required services with the minimum number of clouds involved. According to the description given above, assume there is a request for service S_j in place $P_{Combiner}$, and this service is provided by both C_i and C_{i+1}. In this case, both timed transitions $T_{Transfer_i}$ and $T_{Transfer_{i+1}}$ are enabled. If the numbers of waiting requests for service S_j in C_i and C_{i+1} do not exceed the maximum number of service instances provided by these clouds, called q_j^i and q_j^{i+1} respectively, cloud C_i is selected to host the new request based on the priority mechanism mentioned above. However, if the capacity of the waiting queue of service S_j in C_i is saturated, cloud C_{i+1} is selected to provide this service to the user. The number of all instances of service S_j provided by cloud C_i is modeled by an inhibitor arc with multiplicity q_j^i. These numbers correspond to the numbers used as arc multiplicities in part *composition converter* in which $q_j = \sum_{i=1}^{\mathcal{N}} q_j^i$ for all j, $1 \leq j \leq \mathcal{S}$.

Assuming that cloud C_i is selected to provide service S_j, timed transition $T_{Transfer_i}$ fires and puts a token in place $P_{Service_j^i}$. The time assigned to transition $T_{Transfer_i}$, which follows an exponential distribution with rate α_i, represents the time required to redirect the service request to the selected cloud, and transfer the required data. This process is done for all $\mathcal{K}$ tokens in $P_{Combiner}$, until all tokens of a composite request are sent to appropriate clouds. The existence of a token in place $P_{Service_j^i}$ enables timed transition $T_{Service_j^i}$ that models providing service S_j in cloud C_i. The time assigned to transition $T_{Service_j^i}$ follows an exponential distribution with mean $1/\mu_j^i$. After finishing service S_j by cloud C_i, a token is removed from place $P_{Service_j^i}$ and put into place P_{Final}. Logically, this place is located inside the component *cloud combiner*, so the *cloud combiner* is specified as two parts in the model. When all $\mathcal{K}$ tokens of a composite service request reach the place P_{Final}, the multi-cloud environment finishes serving the composite request, and the user can exit the system. In this case, after measuring the time taken by each of the $\mathcal{K}$ atomic services belonging to a composite service, the maximum time is reported as the *response time* of the multi-cloud environment for the composite service. This represents the total time a composite service request spends in the system (sojourn time). Serving all $\mathcal{M}$ users and computing the response time of the multi-cloud environment for $\mathcal{M}$ composite requests, we can compute the *mean response time* of the environment for a composite service.

The abstract TCPN model presented in Fig. 3 cannot be evaluated in this form because the data structures (colors) should be defined, and each request should be assigned with an ID so that we can track it and compute its corresponding response time. Moreover, some details should be added to the model for gathering data from the tokens circulating in the model and analyzing it. The capacity of each cloud for providing a specified service, and the mechanism which occurs for each condition should be also defined clearly. Therefore, we need to refine the model one step further to reach an analyzable version.

B. The Concrete TCPN Model

The model presented in this section is based on the high-level model proposed in Fig. 3, with some important details appended to it in order to satisfy the tool constraints and to analyze the model. At the beginning, we need to define data structures for the colors. In describing the model shown in Fig. 3, we only mentioned tokens without referring to their colors or data types. However, one of the main strengths of the model proposed in this paper is to use different token types, which helps us distinguish various composite service requests and atomic services. Using TCPN formalism and CPN Tools, we can define different token types and change the types or their values when a token transits through a timed/immediate transition. In addition to the token types, each token in TCPN has a time attribute. When the global time of CPN tools is equal or greater than the token time attribute, the corresponding transition is enabled and can fire.

In the first step, two parts of the model labeled *service requester* and *composition converter* in Fig. 3 are modified as depicted in Fig. 4. At the beginning, place P_{Start} contains a token with token type (color) named *REQSET* including three numbers: an integer to keep the ID of each composite request (id); a real number to store the arrival time of the request (t); and another real number to save the total time a request spends in the system (T). The initial values of these parameters are $id = 1$, $t = 0.0$, $T = 0.0$, which show a composite service request with ID number 1 is submitted to the multi-cloud environment at time 0.0, and the sojourn time of the request is 0.0 at the beginning. The timed transition T_{Arrive} models the inter-arrival times of composite requests and is responsible for loading parameter T in token type *REQSET*. Upon firing this transition, a token of the type *REQSET* named *request* is removed from place P_{Start} and a token with the same type is deposited into place $P_{Compose}$ by applying function *addTime*. This function loads parameter T of the token *request* with a real number generated by an exponential distribution function with rate λ. As it can be seen in Fig. 4, there exists only one token in place P_{Start} at each time instant. However, we need to analyze the model when the number of potential users is $\mathcal{M}$, as shown in Fig. 3. To model this, transition T_{Arrive} is connected to place P_{Start} with an output arc which puts a new token of the type *REQSET* in this place when it fires. The value of the parameter named id of tokens, which are deposited into place

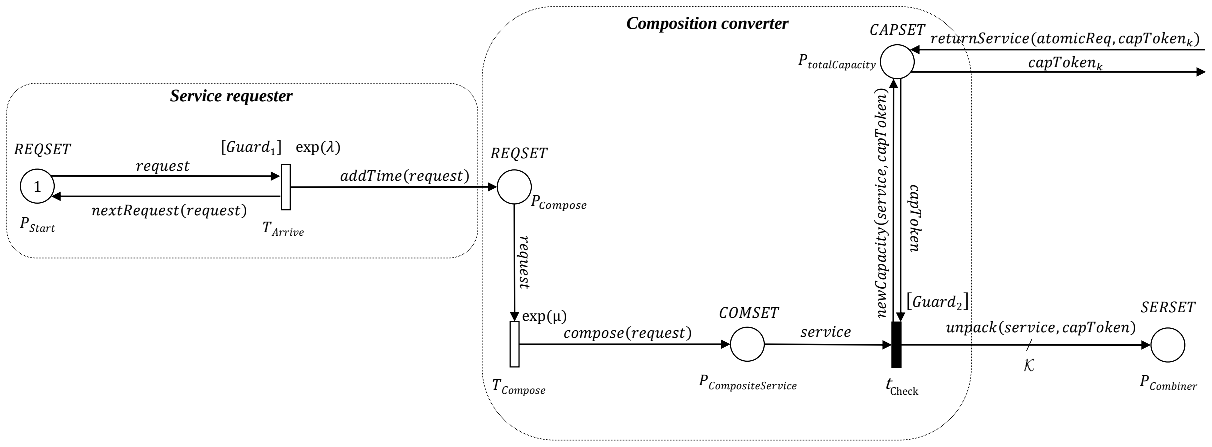

Fig. 4. The concrete TCPN model: the *service requester* and the *composition converter*

P_{Start} by transition T_{Arrive}, is increased by one. We define a guard function for transition T_{Arrive}, named $Guard_1$, to control the number of firings of the transition to accept only $\mathcal{M}$ composite service requests. The guard function $Guard_1$ is represented in Table I.

The existence of a token in place $P_{Compose}$ enables timed transition $T_{Compose}$. Upon firing this transition, a token of the type *REQSET* named *request* is removed from $P_{Compose}$ and a new token of the type *COMSET* is deposited into place $P_{CompositeService}$ by applying function *compose*. The token type *COMSET* extends the type *REQSET* by introducing a new variable named $Serv[]$ in order to save the atomic services of a composite service request. The new token in place $P_{CompositeService}$ contains the atomic services of a composite service in the form of $Serv[]$, which are randomly determined from the set of service types provided by the multi-cloud environment. Moreover, the time parameter T in token type *REQSET* is updated by adding a real number obtained from an exponential function with rate μ when a token transits through $T_{Compose}$. Since the concept of the inhibitor arc is considered as an extension to the basic definition of Petri nets, some tools (e.g., CPN Tools) do not provide support for it. To model this concept in CPN Tools, we add a new place named $P_{totalCapacity}$ to the original model. The data structure of the tokens inside place $P_{totalCapacity}$, named *CAPSET*, represents the capacity of the multi-cloud environment in providing each service. This token type contains two integer numbers: the first integer, called j, shows the ID of each service type provided in the multi-cloud environment $1 \leq j \leq \mathcal{S}$, and the second integer number shows the capacity of the environment in providing service j, which is the maximum instances of service j provided in the environment (q_j in Fig. 3).

The immediate transition t_{Check} removes a token of the type *COMSET* from place $P_{CompositeService}$, called *service*, together with another token of the type *CAPSET* from place $P_{totalCapacity}$, called *capToken*, and puts $\mathcal{K}$ tokens of the type *SERSET* into place $P_{Combiner}$ by applying the *unpack* function. Each token in $P_{Combiner}$ represents an atomic service which should be provided in order to respond the composite service. A guard function, named $Guard_2$ in Fig. 4, is assigned to transition t_{Check} to check the availability of an atomic service when requested. The pseudocode of this guard function is shown in Table I. As mentioned earlier, if a composite service request needs an atomic service that is not available at that time instant (the capacity parameter of that atomic service in type *CAPSET* is 0), the composite service is dropped. We compute the dropping probability of composite requests by counting the number of composite requests dropped by transition t_{Check} and diving it by the total number of composite requests submitted to the *composition converter*. The token type *SERSET* inherits parameters id, t and T from the type *COMSET*, and considers two new parameters named *Num* and *Flag*. The *Num* parameter is an integer number showing the number of atomic services required to serve the current composite service (the size of vector $Serv[]$ in type *COMSET*), and *Flag* is the ID of each atomic service. Since it is assumed that a composite service request contains $\mathcal{K}$ atomic services, we assign $Num = \mathcal{K}$. Upon firing transition t_{Check}, tokens in place $P_{totalCapacity}$ are updated by decreasing the capacity parameter of each service sent to place $P_{Combiner}$, by using the function *newCapacity*. It is worth mentioning that the capacity of each atomic service, which is decreased upon firing t_{Check}, is increased when the service is provided, as will be discussed later.

The concrete model of the *cloud combiner* together with one of the clouds is shown in Fig. 5. To simplify the description of the model, we only represent the interaction between the component *cloud combiner* and cloud C_i, ignoring other clouds in the multi-cloud environment because the connections between other clouds and the *cloud combiner* are the same as those shown in Fig. 5. The decision about sending a service request to a cloud is made by the *cloud combiner*, according to the mechanism mentioned in Section IV-A in which the clouds are checked based on their indexes; the lower index shows the higher priority. If a cloud with higher priority is offering the requested atomic service and it has enough capacity to host the service request, the request is assigned to it. Regarding this mechanism, suppose that the *cloud combiner* decides to

TABLE I
GUARD FUNCTIONS OF THE CONCRETE TCPN MODEL

Guard	Relevant transition	Function
$[Guard_1]$	T_{Arrive}	`guardFunc Guard1(request:REQSET){`
		if `request.id <= M`
		`return true;`
		else
		`return false;}`
$[Guard_2]$	t_{Check}	`guardFunc Guard2(service:COMSET, capToken:CAPSET){`
		`x = true;`
		for `i=1:K`
		for `each (id, num) in capToken`
		if `(id = service.Serv[i]) && (num > 0)` **then**
		`x=true;`
		else
		`x=false;`
		end for
		end for
		`return x;}`
$[Guard_3]$	$T_{Transfer_i}$	`guardFunc Guard3(atomicReq:SERSET, capToken_i:CAPSET){`
		for `each (id, num) in capToken_i`
		if `(id = atomicReq.Flag) && (num > 0)` **then**
		`return true;`
		else
		`return false;}`
$[Guard_4]$	$T_{Transfer_i}$	`guardFunc Guard4(atomicReq:SERSET, capToken_j:CAPSET){`
		for `each (id, num) in capToken_j   // 1 ≤ j ≤ i-1`
		if `(id = atomicReq.Flag) && (num > 0)` **then**
		`return false;`
		else
		`return true;}`

assign service S_k to cloud C_i. In this case, timed transition $T_{Transfer_i}$ removes a token of the type *SERSET* from place $P_{Combiner}$, named $atomicReq$ in which the $Flag$ parameter of $atomicReq$ is equal to k, and searches for a relevant token of the type *CAPSET* in place $P_{Capacity_i}$ named $capToken_i$. The place $P_{Capacity_i}$ keeps track of the available services provided in cloud C_i. Its functionality in this part of the model is similar to the functionality of place $P_{totalCapacity}$ in the sub-model presented in Fig. 4. The token type in place $P_{Capacity_i}$ is the same as the one used for $P_{totalCapacity}$. In order to model the aforementioned mechanism, timed transition $T_{Transfer_i}$ is equipped with two guard functions, named $Guard_3$ and $Guard_4$, as shown in Table I. The guard function $Guard_3$ checks the capacity of the current cloud, C_i, for providing the requested atomic service, S_k, while $Guard_4$ checks the clouds with higher priorities, C_1 to C_{i-1}, to make sure that none of them have the capacity to provide service S_k. If both guard functions are evaluated to true, transition $T_{Transfer_i}$ fires and moves a token of the type *SERSET*, named *atomicReq*, from place $P_{Combiner}$ to place $P_{Service_k^i}$. When the token transits thorough $T_{Transfer_i}$, the parameter T of the token is updated by adding a random number obtained from an exponential distribution function with rate α_i. It should be noted that the target place $P_{Service_k^i}$ is selected by the functions *sendService* which are associated with the arcs connecting transition $T_{Transfer_i}$ to places $P_{Service_j^i}$ $\forall j, 1 \leq j \leq si$, according to the *Flag* parameter of token *atomicReq*. Therefore, we only load the place $P_{Service_k^i}$ with a token upon firing $T_{Transfer_i}$ in Fig. 5, since it is assumed that service S_k is sent to cloud C_i.

The existence of a token in place $P_{Service_k^i}$ enables timed transition $T_{Service_k^i}$ modeling the process of providing service S_k by a service provider in cloud C_i. As can be seen in Fig. 5, the firing time of this transition follows an exponential distribution with rate μ_k^i. This transition moves a token of the type *SERSET* from place $P_{Service_k^i}$ and puts a token with the same type into place P_{Final}, while updating the parameter T of the token named *atomicReq*. It is worthwhile to note that all transitions $T_{Service_j^i}, 1 \leq j \leq si$ act as transition $T_{Service_k^i}$, moving a *SERSET* token from their own input places, and depositing another *SERSET* token into their output place, P_{Final}. However, the focus in Fig. 5 is on providing service S_k by cloud C_i, so we do not explain other services provided by this cloud. In addition to moving a token from $P_{Service_k^i}$ to P_{Final}, transition $T_{Service_k^i}$ is responsible for returning the service S_k back to both places $P_{totalCapacity}$ and $P_{Capacity_i}$, which is done by the function *returnService* that increases the capacity parameter of tokens $capToken_k$ by one upon firing $T_{Service_k^i}$.

The place P_{Final} acts as a repository, which collects all tokens representing atomic services of a composite service provided by various clouds. Since we assign a unique ID to a composite service request at the beginning and keep the numbers of all atomic services of the composite request inside token type *SERSET*, named id and Num respectively, we can compute the time a composite service is completely served. To model this, we use a timed transition named T_{Final} with an exponential firing distribution. The firing rate of this transition

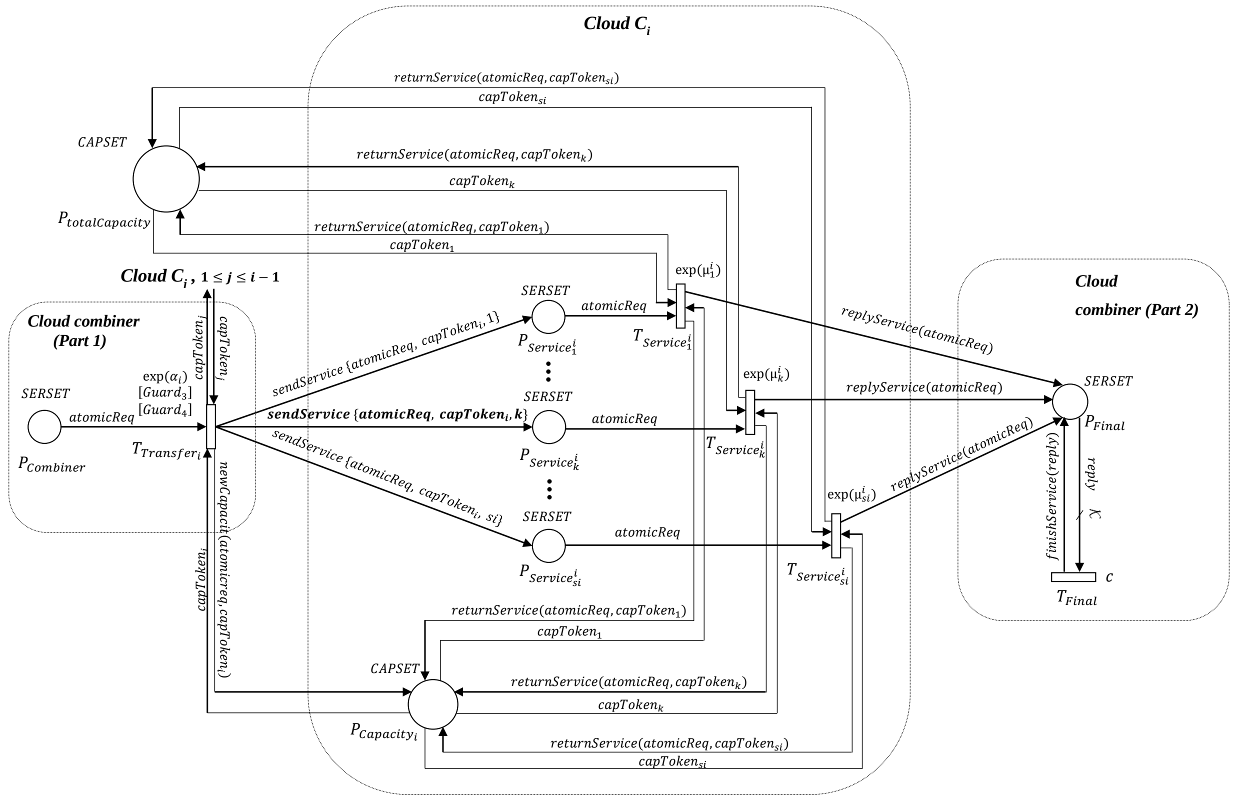

Fig. 5. The concrete TCPN model: the *cloud combiner* and *cloud* C_i

is a constant number named c, which can be set to any number, since it is not added to the parameters of token type *SERSET*. This is a dummy transition for the purpose of emptying place P_{Final}. It only counts the number of tokens with the same id queued in P_{Final} and, if the number reaches $\mathcal{K}$, it fires and removes all $\mathcal{K}$ tokens with the same id from P_{Final}. Once transition T_{Final} fires, the maximum value saved in parameter T among all $\mathcal{K}$ tokens with the same id is reported as the time a composite service request spent in the multi-cloud environment to receive all its requested services. By obtaining these maximum times for all $\mathcal{M}$ composite service requests, we can compute the mean response time of the multi-cloud environment for composite service requests.

V. Numerical Results

In this section, numerical results obtained by the proposed TCPN model are reported. As mentioned earlier, the aim of the proposed TCPN is to model the process of service composition and selection in multi-cloud environments and evaluate both the mean response time and the dropping probability of composite service requests. In order to study different combinations and service varieties in the clouds, we consider two scenarios in this section. In the first scenario, given in Section V-A, a small multi-cloud environment with *three* clouds and a limited number of services is studied, while in the second scenario, given in Section V-B, a larger system with higher number of clouds and services is considered. In order to validate the results obtained from the TCPN models, we simulate the sample systems considered in both scenarios with the CloudSim framework [28]. CloudSim is a Java based development platform which supports the simulation of large-scale cloud computing environments. It provides users with the ability to extend the existing functions and introduce new ones to satisfy the requirements of the system under study. We extend the CloudSim framework by defining appropriate functions to model the multi-cloud environment and reducing the minimum period between events to increase the precision of the results. The results gained from the CloudSim toolkit are presented and compared with the results obtained using the CPN Tools for each scenario.

A. First Scenario

In this scenario, a multi-cloud environment with *three* clouds and *five* different service types is considered ($\mathcal{N} = 3$ and $\mathcal{S} = 5$). The number of service instances provided in each cloud q_j^i, and their related rates μ_j^i ($1 \le i \le 3$, $1 \le j \le 5$) are shown in Table II. Using the information provided in Table II, we can compute the number of all instances of a given service existing in the environment ($q_j = \sum_{i=1}^{3} q_j^i$, $1 \le j \le 5$).

The rate of the exponential functions assigned to transitions $T_{Transfer_i}$, $1 \le i \le 3$, which model the times required to redirect a service request to the clouds and transfer the required data between the cloud combiner and clouds, are set to 20, 30, and 25 req/sec for clouds C_1, C_2, and C_3, respectively ($\alpha_1 = 20$, $\alpha_2 = 30$, and $\alpha_3 = 25$). Moreover, the mean time for serving a composite request to analyze it and recognize

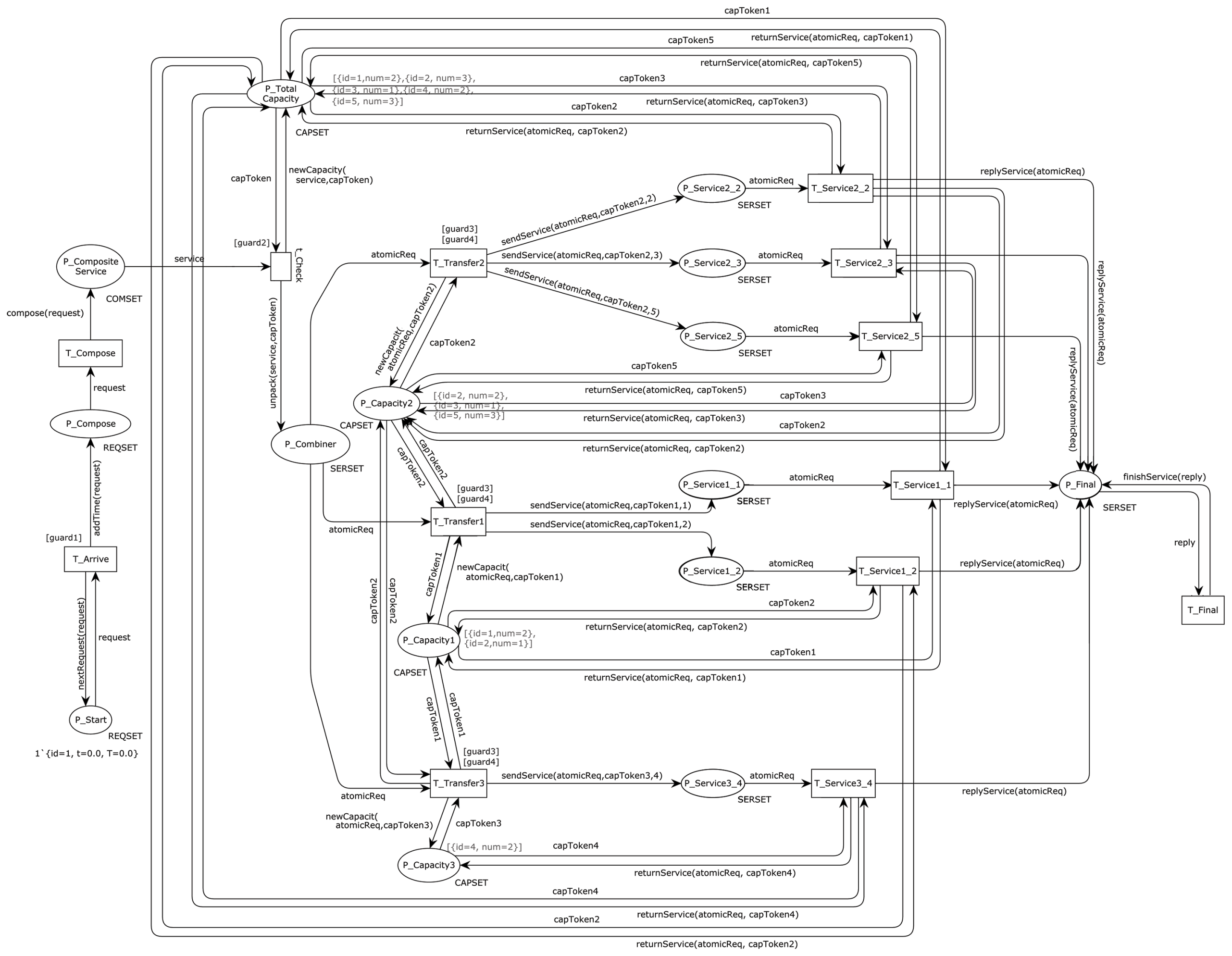
P_Total Capacity
[{id=1,num=2},{id=2, num=3}, {id=3, num=1},{id=4, num=2}, {id=5, num=3}]
CAPSET
capToken
newCapacity(service,capToken)
[guard2]
t_Check
P_Composite Service
service
COMSET
compose(request)
T_Compose
request
P_Compose
REQSET
addTime(request)
[guard1]
T_Arrive
nextRequest(request)
request
P_Start
REQSET
1`{id=1, t=0.0, T=0.0}
unpack(service,capToken)
P_Combiner
SERSET
atomicReq
T_Transfer2
[guard3] [guard4]
sendService(atomicReq,capToken2,2)
sendService(atomicReq,capToken2,3)
sendService(atomicReq,capToken2,5)
P_Service2_2
P_Service2_3
P_Service2_5
T_Service2_2
T_Service2_3
T_Service2_5
P_Capacity2
[{id=2, num=2}, {id=3, num=1}, {id=5, num=3}]
newCapacit(atomicReq,capToken2)
capToken2
capToken5
capToken3
capToken1
capToken4
returnService(atomicReq, capToken1)
returnService(atomicReq, capToken5)
returnService(atomicReq, capToken3)
returnService(atomicReq, capToken2)
returnService(atomicReq, capToken4)
T_Transfer1
sendService(atomicReq,capToken1,1)
sendService(atomicReq,capToken1,2)
P_Service1_1
P_Service1_2
T_Service1_1
T_Service1_2
newCapacit(atomicReq,capToken1)
P_Capacity1
[{id=1,num=2}, {id=2,num=1}]
T_Transfer3
sendService(atomicReq,capToken3,4)
P_Service3_4
T_Service3_4
newCapacit(atomicReq,capToken3)
P_Capacity3
[{id=4, num=2}]
replyService(atomicReq)
P_Final
finishService(reply)
reply
T_Final

Fig. 6. The graphical representation of the proposed TCPN model for the first scenario obtained by the CPN Tools

TABLE II
THE SERVICE TYPES AND THE NUMBER OF INSTANCES PROVIDED IN EACH CLOUD WITH THEIR RELATED RATES IN THE FIRST SCENARIO

Clouds	**Services**	**Number (q_j^i)**	**Rate (μ_j^i) (req/sec)**
C_1	S_1	2	20
	S_2	1	15
C_2	S_2	2	10
	S_3	1	30
	S_5	3	25
C_3	S_4	2	35

its atomic services is set to $1/50\ sec$ ($\mu = 50\ req/sec$). The arrival rate of composite requests and the number of potential users are set to $10\ req/sec$ and $100,000\ users$, respectively ($\lambda = 10$ and $\mathcal{M} = 100,000$). All these numbers are selected randomly and can be replaced by any other values.

The above-mentioned scenario is modeled by the CPN Tools and simulated by the CloudSim framework, and the mean response time of the multi-cloud environment for composite service requests and the dropping probability of them are evaluated. Fig. 6 shows the proposed TCPN model established with the CPN Tools. As shown in this figure, the clouds are sorted according to the number of service types provided in each of them. Cloud C_2 is the one with highest priority and C_3 the lowest one. As noted before, the total number of composite service requests which should be served by the environment is considered to be $\mathcal{M} = 100,000$ requests. To make sure that the transient state of the system and the initial conditions do not affect the final results and to reach more dependable results, we take the first $5,000$ requests away and compute the steady-state mean response time and dropping probability for the remaining $95,000$ requests. Fig. 7 and Fig. 8 show the mean response time and dropping probability of composite service requests for the first $5,000$ requests, respectively. These figures show that the results converge to the steady-state values after serving a few number of requests in both the TCPN model and CloudSim.

Table III shows the mean response time of the considered multi-cloud for composite service requests and the dropping probability of composite requests obtained both using the CPN Tools and CloudSim in the steady-state. As it can be seen in this table, the relative errors between the values obtained with the proposed model and simulation are 1.85% and 0.62% for

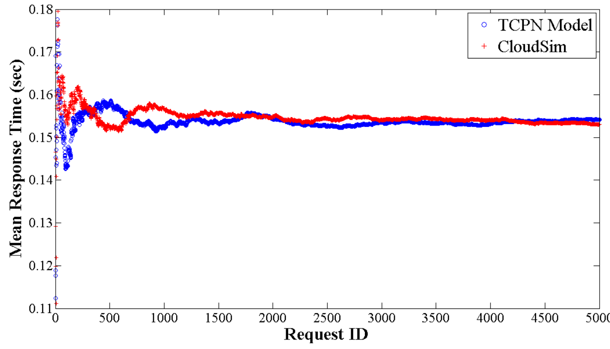

Fig. 7. The convergence of the mean response time to the steady-state value in the first scenario

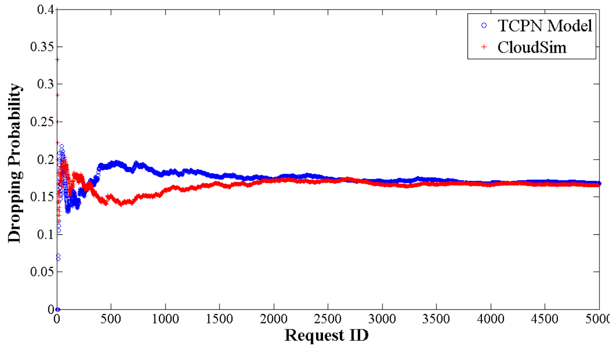

Fig. 8. The convergence of the dropping probability to the steady-state value in the first scenario

the mean response time and dropping probability, respectively. Hence, we can conclude that the proposed TCPN model allows to appropriately model and evaluate the service composition in the small multi-cloud environment considered in the first scenario.

B. *Second Scenario*

In the scenario with a larger multi-cloud environment, 10 clouds and 15 different service types are considered ($\mathcal{N} = 10$ and $\mathcal{S} = 15$). Table IV shows the number of instances of each service type provided in each cloud and its related rate, which are randomly chosen in the ranges $[1, 10]$ and $[1, 100]$, respectively. Each cell in Table IV contains a tuple (q_j^i, μ_j^i), where q_j^i is the number of instances of service S_j provided in cloud C_i and μ_j^i is its related rate, where $1 \leq i \leq 10$ and $1 \leq j \leq 15$. The notation *NA*, in row i column j of Table IV, means that cloud C_i does not provide service S_j. The dimension of all rates and times are req/sec and sec, respectively. The rate of the exponential functions assigned to transitions $T_{Transfer_i}$, $1 \leq i \leq 10$, are randomly selected between 200 and 500 resulting in $\alpha_1 = 237$, $\alpha_2 = 305$, $\alpha_3 = 226$, $\alpha_4 = 487$, $\alpha_5 = 398$, $\alpha_6 = 473$, $\alpha_7 = 350$, $\alpha_8 = 360$, $\alpha_9 = 489$, and $\alpha_{10} = 497$. Moreover, the mean time of splitting a composite service into atomic services is considered to be $1/1000$ sec ($\mu = 1000$ req/sec). The total number of composite service requests is set to $100,000$ ($\mathcal{M} = 100,000$). Similar to the first scenario, we eliminate the first $5,000$ requests from the

TABLE III
THE STEADY-STATE MEAN RESPONSE TIME AND DROPPING PROBABILITY OF COMPOSITE SERVICE REQUESTS RESULTED FROM THE TCPN MODEL AND CLOUDSIM IN THE FIRST SCENARIO

Measures	**CPN Tools**	**CloudSim**	**Error**
Mean Response Time (sec)	0.1538	0.1510	1.85%
Dropping Probability	0.1618	0.1608	0.62%

results and compute the mean response time and the dropping probability of requests for the remaining $95,000$ requests to reach the steady-state.

In order to study the impact of inter-arrival time of composite service requests on the final results, we vary the arrival rates from 100 to 500 req/sec with incremental step 50, and study the behavior of output measures. Fig. 9 and Fig. 10 show the mean response time and the dropping probability of composite service requests, respectively. As shown in these figures, the proposed TCPN model can appropriately evaluate the system and reach the results obtained with the CloudSim. Moreover, it can be concluded from Fig. 10 that the blocking probability of composite requests increases with the increase of the arrival rate of requests. This is a reasonable conclusion since the capacity of the multi-cloud environment in providing service is fixed. Hence, more requests will be dropped when they enter the environment more frequently.

Although the results obtained from the proposed TCPN model and the simulation framework are very close, as shown in Fig. 9 and Fig. 10, the time required to reach the steady-state results are very different in CPN Tools and CloudSim framework, emphasizing the power of the proposed modeling approach in analyzing complex systems. For example, in the above-mentioned setting if we set $\lambda = 500$ and $\mathcal{M} = 100,000$, the time required to serve all requests in the CPN Tools and CloudSim will be around 2 min and 97 min, respectively. Furthermore, the proposed TCPN model is scalable with respect to the number of total composite requests served in the environment, which allows us to set the population of the system to a large number in all experiments, but the limiting factor is the simulation with CloudSim which takes much more time to get results. For example, if $\lambda = 500$ and $\mathcal{M} = 1,000,000$, it takes about 10 min for the TCPN model to serve all requests, while CloudSim needs more than 12 $hours$ to finish. It is worth mentioning that all experiments are conducted on a Linux Server with Intel® Xeon® CPU E5-2690 v2 @ 3.00 GHz and 256 GB of RAM running 64-bit Ubuntu 14.04.1.

VI. Conclusion and Future Work

Multi-cloud environments provide users with a variety of services published by different service providers with different Quality of Services (QoS). In these environments, different services from Geo-distributed clouds come together to respond to composite service requests raised by users. A composite service request is first analyzed by the cloud broker, and then redirected to the clouds to be served. A composite service request is served when all its atomic services are provided by

TABLE IV
THE NUMBER OF SERVICES AND THEIR RELATED RATES IN THE SCENARIO WITH A LARGER MULTI-CLOUD ENVIRONMENT

Clouds	**Services**														
	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}
c_1	(6, 40)	*NA*	(10, 87)	*NA*	*NA*	*NA*	*NA*	*NA*	(10, 80)	*NA*	*NA*	*NA*	(9, 95)	*NA*	*NA*
c_2	(10, 39)	*NA*	*NA*	*NA*	(1, 55)	*NA*	*NA*	(10, 12)	*NA*	*NA*	(2, 67)	*NA*	(9, 18)	*NA*	*NA*
c_3	*NA*	(1, 18)	(3, 96)	*NA*	*NA*	(10, 25)	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	(6, 53)	*NA*	(8, 85)
c_4	*NA*	(6, 14)	*NA*	*NA*	*NA*	*NA*	*NA*	(4, 10)	(6, 59)	(3, 31)	*NA*	(2, 35)	*NA*	*NA*	*NA*
c_5	(5, 82)	*NA*	*NA*	*NA*	(8, 95)	*NA*	(9, 66)	*NA*	(1, 88)	(4, 85)	*NA*	*NA*	*NA*	(10, 99)	(6, 4)
c_6	*NA*	(7, 95)	(3, 91)	(10, 96)	*NA*	*NA*	(4, 56)	*NA*	(6, 62)	(3, 20)	(1, 27)	*NA*	(8, 19)	(8, 4)	*NA*
c_7	*NA*	*NA*	(1, 33)	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	(5, 64)	*NA*	(5, 47)	(4, 11)	(6, 1)
c_8	*NA*	(4, 92)	(3, 22)	(8, 69)	(1, 55)	(4, 66)	(2, 13)	*NA*	*NA*	(9, 33)	(3, 1)	*NA*	(3, 79)	(6, 56)	(7, 30)
c_9	(2, 80)	*NA*	(7, 52)	(7, 83)	*NA*	*NA*	*NA*	(4 57)	*NA*	(9, 98)	*NA*	(3, 43)	*NA*	*NA*	(5, 31)
c_{10}	*NA*	(3, 90)	*NA*	*NA*	*NA*	(2, 19)	*NA*	*NA*	(4, 31)	*NA*	*NA*	*NA*	(2, 79)	(9, 63)	(6, 53)

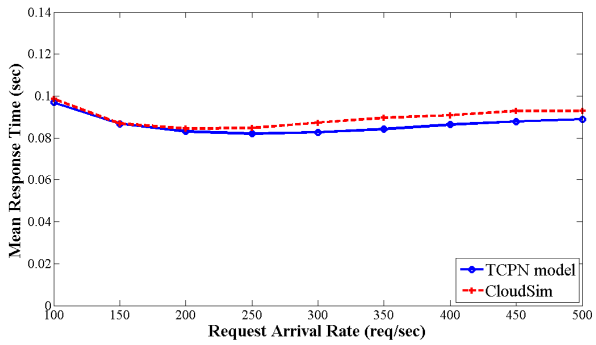

Fig. 9. The mean response time of the multi-cloud environment considered in the second scenario for composite service requests

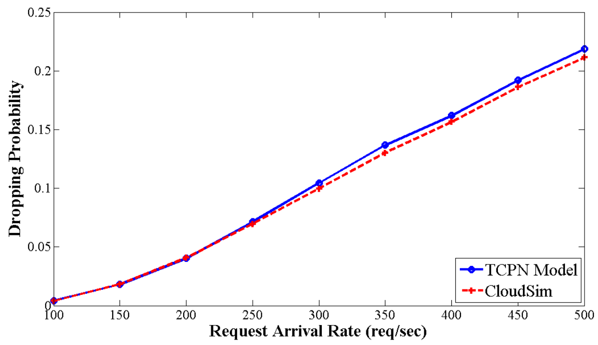

Fig. 10. The dropping probability of composite service requests in the second scenario

the multi-cloud environment. In order to formally model the procedure of request submission to a multi-cloud and service provisioning in each cloud, we proposed a Timed Colored Petri Net (TCPN) model in this paper. We simulated the system under study using the CloudSim framework and compared the results obtained from the proposed TCPN model with the results achieved from the CloudSim. We concluded that our proposed model can appropriately assess the performance in a very short time compared to CloudSim.

The previously presented approaches in this area mostly focused on finding the semi-optimal composition of services in order to achieve better QoS while ignoring any formal modeling or analysis of the system [2], [4], [7], [8], [15], [16], [19], [29], [31], [32], [35]. Since the nature of the Web service composition and selection problem is NP-complete, most of the approaches used heuristics to solve the above-mentioned problem [4], [15], [16], [29], [31], [32]. On the other hand, some of the previous methods focused on modeling the problem of the Web service composition without taking into account any performance measures [13], [24], [25]. These models, which have basically used different extensions of state machines and Petri nets, can represent the workflow of the system and composition of atomic services to serve a complex service, but they fail in evaluating the performance of real world systems. However, the model proposed in this paper not only can graphically represent the structure and workflow of serving a composite request, but it can also analyze the system and evaluate the performance. The proposed model, which considers the hierarchical structure of multi-clouds, is able to accurately estimate two important measures of these kinds of systems: the mean response time of the environment for composite service requests and the dropping probability of requests.

By applying the colored extensions of Petri nets, we can distinguish different requests and atomic services which is crucial in modeling Web service composition in multi-clouds. This is critical because each cloud in a multi-cloud environment can provide a various number of a specific type atomic service and each composite request can demand different atomic services. TCPNs provide us with the capability of modeling such a complex system whereas other extensions of Petri nets could not handle it easily [24]. In addition to modeling the system and computing the performance measures, our proposed model reduces the number of clouds involved in serving a composite service by applying a simple mechanism for sorting the clouds according to the number of services provided by each of them. Since the proposed model is designed to be applied to multi-clouds, reducing the number of clouds serving a single composite service is of utmost importance [2], [4], [8], [19].

There is a number of research issues remaining open for future work. One interesting extension would be to use Markov Decision Process (MDP) in the body of the proposed TCPN model. Herein, we sort the clouds according to the number of services provided in each cloud, and then select a cloud with the maximum numbers of services to host an atomic service if it provides the requested service and has enough capacity. Although this approach can reduce the number of clouds involved in serving a composite service request, it reduces neither the mean response time of the environment nor the dropping probability of requests. To simultaneously reduce the number of clouds involved in serving a composite service request and improve the performance (e.g., mean response time, dropping probability, etc.), one can use Markov Decision Petri Nets (MDPNs) and Markov Decision Well-formed Nets (MDWNs) which combine MDP with PNs. Using MDPNs and MDWNs in modeling such a system and taking more details of a real system into consideration in a mathematical model may lead to simultaneously minimizing the number of clouds involved in providing a service and the mean response time of requests (or the dropping probability).

Another interesting extension to the proposed TCPN model is modifying the model and adding more details to the tokens to consider the deadline factor for each composite service request. In this case, the cost function and the penalty which should be imposed upon missing a deadline could be taken into account. Considering QoS attributes in token types, taking bandwidth variations across the interconnected clouds into account, considering performance effects due to the contention on shared resources, and studying different administration policies applied to each cloud, we can reach a more comprehensive service composition model. Moreover, assigning priority to composite service requests from different classes of users, and assigning the faster and more reliable services to the users with higher priorities can make the model to be more practical and applicable.

REFERENCES

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009.

[2] H. Kurdi, A. Al-Anazi, C. Campbell, and A. A. Faries, "A combinatorial optimization algorithm for multiple cloud service composition," *Computers & Electrical Engineering*, vol. 42, pp. 107–113, 2015.

[3] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[4] D. Wang, Y. Yang, and Z. Mi, "A genetic-based approach to web service composition in geo-distributed cloud environment," *Computers & Electrical Engineering*, vol. 43, pp. 129–141, 2015.

[5] A. Tsalgatidou and T. Pilioura, "An overview of standards and related technology in web services," *Distributed and Parallel Databases*, vol. 12, no. 2, pp. 135–162, 2002.

[6] Q. Sheng, X. Qiao, A. Vasilakos, C. Szabo, S. Bourne, and X. Xu, "Web services composition: A decades overview," *Information Sciences*, vol. 280, pp. 218–238, 2014.

[7] G. Zou, Y. Chen, Y. Xiang, R. Huang, and Y. Xu, "AI planning and combinatorial optimization for web service composition in cloud computing," in *International Conference on Cloud Computing and Virtualization*, Singapore, 17-18 May 2010, pp. 1–8.

[8] Q. Yu, L. Chen, and B. Li, "Ant colony optimization applied to web service compositions in cloud computing," *Computers & Electrical Engineering*, vol. 41, pp. 18–27, 2015.

[9] "Programmableweb," https://www.programmableweb.com/, accessed: April 2017.

[10] "Nordic apis," http://nordicapis.com/, accessed: April 2017.

[11] "Magic quadrant for cloud infrastructure as a service, worldwide," https://www.gartner.com/, accessed: April 2017.

[12] "Business and financial news," https://www.forbes.com, accessed: April 2017.

[13] H. Bao and W. Dou, "A QoS-aware service selection method for cloud service composition," in *The IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, Shanghai, China, 21-25 May 2012, pp. 2254–2261.

[14] A. Jula, E. Sundararajan, and Z. Othman, "Cloud computing service composition: A systematic literature review," *Expert Systems with Applications*, vol. 41, no. 8, pp. 3809–3824, 2014.

[15] F. Chen, R. Dou, M. Li, and H. Wu, "A flexible QoS-aware web service composition method by multi-objective optimization in cloud manufacturing," *Computers & Industrial Engineering*, vol. 99, no. C, pp. 423–431, 2016.

[16] Q. Wu, F. Ishikawa, Q. Zhu, and D.-H. Shin, "QoS-aware multigranularity service composition: Modeling and optimization," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 46, no. 11, pp. 1565–1577, 2016.

[17] F. Wagner, F. Ishikawa, and S. Honiden, "QoS-aware automatic service composition by applying functional clustering," in *The IEEE International Conference on Web Services*, Washington, DC, 4-9 July 2011, pp. 89–96.

[18] S.-Y. Hwang, E.-P. Lim, C.-H. Lee, and C.-H. Chen, "Dynamic web service selection for reliable web service composition," *IEEE Transactions on Services Computing*, vol. 1, no. 2, pp. 104–116, 2008.

[19] X. Li, J. Wu, and S. Lu, "QoS-aware service selection in geographically distributed clouds," in *The 22nd International Conference on Computer Communications and Networks*, Nassau, Bahamas, 30 July-2 August 2013, pp. 1–5.

[20] J. Zhu, Z. Zheng, Y. Zhou, and M. R. Lyu, "Scaling service-oriented applications into Geo-distributed clouds," in *The 7th IEEE International Symposium on Service Oriented System Engineering*, San Francisco, CA, 25-28 March 2013, pp. 335–340.

[21] N. J. and A. Vakili, "Comprehensive and systematic review of the service composition mechanisms in the cloud environments," *Journal of Network and Computer Applications*, vol. 81, pp. 24–36, 2017.

[22] Z. Xiang, S. Deng, and H. Gao, "Service selection using service clusters," in *The IEEE International Conference on Services Computing*, New York, NY, 27 June-2 July 2015, pp. 769–772.

[23] S. Wang, A. Zhou, F. Yang, and R. N. Chang, "Towards network-aware service composition in the cloud," *IEEE Transactions on Cloud Computing*, vol. Published online, 2016.

[24] S. Liu, X. Liu, H. Zhao, and W. Fu, "Composite service execution Petri net and service composition optimization," in *The IEEE International Conference on Service Operations and Logistics and Informatics*, Suzhou, China, 8-10 July 2012, pp. 273–278.

[25] H. Ma, K.-D. Schewe, and Q. Wang, "An abstract model for service provision, search and composition," in *The IEEE Asia-Pacific Services Computing Conference*, Singapore, 7-11 December 2009, pp. 95–102.

[26] K. Jensen, *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*. EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1994, vol. 1, 2, 3.

[27] K. Jensen, L. M. Kristensen, and L. Wells, "Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems," *International Journal on Software Tools for Technology Transfer*, vol. 9, no. 3, pp. 213–254, 2007.

[28] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[29] S. Deng, L. Huang, J. Taheri, J. Yin, M. Zhou, and A. Y. Zomaya, "Mobility-aware service composition in mobile communities," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, no. 3, pp. 555–568, 2017.

[30] K. Klai and H. Ochi, "A formal approach for service composition in a cloud resources sharing context," in *The 16th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, Cartagena, Colombia, 16-19 May 2016, pp. 458–461.

[31] Z. Ding, J. Liu, Y. Sun, C. Jiang, and M. Zhou, "A transaction and QoS-aware service selection approach based on genetic algorithm," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 45, no. 7, pp. 1035–1046, 2015.

[32] Y. Lu and X. Xu, "A semantic web-based framework for service composition in a cloud manufacturing environment," *Journal of Manufacturing Systems*, vol. 42, pp. 69–81, 2017.

[33] S. Wang, Z. Zheng, Q. Sun, H. Zou, and F. Yang, "Cloud model for service selection," in *The 30th IEEE International Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, Shanghai, China, 10-15 April 2011, pp. 666–671.

[34] A. Abdullah and X. Li, "Agent-based model to web service composition," in *The IEEE International Conference on Services Computing*, San Francisco, CA, 27 June-2 July 2016, pp. 523–530.

[35] Y. Chen, J. Huang, C. Lin, and J. Hu, "A partial selection methodology for efficient QoS-aware service composition," *IEEE Transactions on Services Computing*, vol. 8, no. 3, pp. 384–397, 2015.

[36] Z. Ding, Y. Sun, C. Jiang, M. Zhou, J. Liu, and W. Song, "Performance evaluation of transactional composite web services," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 46, no. 8, pp. 1061–1074, 2016.

[37] "Ibm: Standards and web services," https://www.ibm.com/developerworks/webservices/standards/, accessed: April 2017.

[38] G. Selvakumar and B. Kaviya, "A survey on RESTful web services composition," in *The International Conference on Computer Communication and Informatics*, Coimbatore, India, 7-9 January 2016, pp. 1–4.

[39] N. Xi, C. Sun, J. Ma, and Y. Shen, "Secure service composition with information flow control in service clouds," *Future Generation Computer Systems*, vol. 49, pp. 142–148, 2015.

[40] M. A. Azgomi and R. Entezari-Maleki, "Task scheduling modelling and reliability evaluation of grid services using coloured petri nets," *Future Generation Computer Systems*, vol. 26, no. 8, pp. 1141–1150, 2010.

[41] R. Ghosh, F. Longo, V. K. Naik, and K. S. Trivedi, "Modeling and performance analysis of large scale IaaS clouds," *Future Generation Computer Systems*, vol. 29, no. 5, pp. 1216–1234, 2013.

[42] R. Entezari-Maleki, K. S. Trivedi, and A. Movaghar, "Performability evaluation of grid environments using stochastic reward nets," *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 2, pp. 204–216, 2015.

[43] D. Bruneo, A. Lhoas, F. Longo, and A. Puliafito, "Modeling and evaluation of energy policies in green clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 11, pp. 3052–3065, 2015.

[44] R. Entezari-Maleki, L. Sousa, and A. Movaghar, "Performance and power modeling and evaluation of virtualized servers in iaas clouds," *Information Sciences*, vol. 394-395, pp. 106–122, 2017.

[45] Y. Xia, M. Zhou, X. Luo, S. Pang, and Q. Zhu, "A stochastic approach to analysis of energy-aware DVS-enabled cloud datacenters," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 45, no. 1, pp. 73–83, 2015.

Reza Entezari-Maleki is a Post-Doctoral Researcher in the School of Computer Science at Institute for Research in Fundamental Sciences (IPM) in Tehran, Iran. He received his Ph.D. in Computer Engineering (Software) from the Sharif University of Technology, Tehran, Iran in 2014, and M.S. and B.S. degrees in Computer Engineering (Software) from the Iran University of Science and Technology, Tehran, Iran in 2009 and 2007, respectively. He visited the Seoul National University in South Korea, twice in 2012 and 2017, Duke University in NC, USA, in 2013, and Instituto Superior Tecnico, Universidade de Lisboa in Portugal in 2015. His main research interests are performance/dependability modeling and evaluation, distributed computing systems, cloud computing, and task scheduling algorithms.

Sayed Ehsan Etesami received his B.S. degree in computer engineering (Information Technology) from the Department of Computer Engineering, Isfahan University of Technolgy, Iran, in 2015. He is currently, an M.S. student in computer engineering (Information Technology) at the Department of Computer Engineering, Sharif University of Technology, Iran. His general research interests are in the field of performance/dependability modeling and evaluation, virtualization and cloud computing

Negar Ghorbani received her B.S. degree in computer engineering (Software) from Sharif University of Technology, Iran in 2016. She is currently a Ph.D. student in software engineering at the University of California, Irvine. Her general research interests are in the field of software engineering, and specifically software architecture, software security, software analysis, and performance evaluation.

Arian Akhavan Niaki received his B.S in Computer Engineering (Software) from Sharif University of Technology, Tehran, Iran in 2016. He is currently a Ph.D. student in the Computer Science Department of Stony Brook University in New York, USA. His current research interests are in the area of computer networks, network measurement and, performance evaluation.

Leonel Sousa (M'01 – SM'03) is the Chair of the Department of Electrical and Computer Engineering of Instituto Superior Tecnico (IST), where he is working from 1992 and is Full Professor since 2010. In 2016 he was a Visiting Professor at Tsukuba University, with a JSPS Invitation Fellowship for Research in Japan, and at Carnegie Mellon University, USA. From 2009 to 2013 he was President of INESC-ID, a R&D Institute affiliated with IST, and Vice-Chair of the Scientific Council of IST, from 2013 to 2016. His research interests include computer architectures, high performance computing and multimedia systems. He has contributed with more than 250 papers for international journals and conferences and to the organization of several international conferences, being currently Associate Editor of the IEEE Transactions on Multimedia, IEEE Transactions on Circuits and Systems for Video Technology and IEEE Access, and the Editor-in-Chief of the Eurasip JES. He has contributed to several international research projects, currently the H2020 FutureTPM, and has been PI of four research projects funded by the Portuguese Foundation for Science and Technology. He is member of the HiPEAC Network and leader of the Working Group 1 of EU COST Action NESUS. He is a Distinguished Scientist of the ACM.

Ali Movaghar is a Professor in the Department of Computer Engineering at Sharif University of Technology in Tehran, Iran and has been on the Sharif faculty since 1993. He received his B.S. degree in Electrical Engineering from the University of Tehran in 1977, and M.S. and Ph.D. degrees in Computer, Information, and Control Engineering from the University of Michigan, Ann Arbor, in 1979 and 1985, respectively. He visited the Institut National de Recherche en Informatique et en Automatique in Paris, France and the Department of Electrical Engineering and Computer Science at the University of California, Irvine in 1984 and 2011, respectively, worked at AT&T Information Systems in Naperville, IL in 1985-1986, and taught at the University of Michigan, Ann Arbor in 1987-1989. His research interests include performance/dependability modeling and formal verification of wireless networks and distributed real-time systems. He is a senior member of the IEEE and the ACM.