Evaluating High Availability-Aware Deployments Using Stochastic Petri Net Model and Cloud Scoring Selection Tool

Manar Jammal, *Member, IEEE*, Ali Kanso, *Member, IEEE*, Parisa Heidari, *Member, IEEE*, and Aballah Shami, *Senior Member, IEEE*

Abstract—Different challenges are facing the adoption of cloud-based applications, including high availability (HA), energy, and other performance demands. Therefore, an integrated solution that addresses these issues is critical for cloud services. Cloud providers promise the HA of their infrastructure while cloud tenants are encouraged to deploy their applications across multiple availability zones. Moreover, the environmental and cost impacts of running applications in the cloud are integral parts of incorporated responsibility where the cloud providers and tenants intend to reduce. Hence, an analytical stochastic model is needed for the tenants and providers to quantify the expected availability offered by an application deployment. If multiple deployment options can satisfy the HA requirement, the question remains, how can we choose the deployment that satisfies the other providers and tenants requirements? Therefore, this paper proposes a cloud scoring system and integrates it with a Stochastic Petri Net model. While the Petri Net model evaluates the availability of cloud applications deployments, the scoring system selects the optimal HA-aware deployment in terms of energy, operational expenditure, and other norms. We illustrate our approach with a use case that shows how we can use the various deployment options to satisfy both the cloud tenant and provider needs.

Index Terms—Availability, cloud scoring, carbon footprint, OPEX, petri net, stochastic failures, recovery, load balancing

1 Introduction

With the cloud computing era, many business applications are offered as cloud services where they can be accessed anytime and anywhere. Infrastructure-as-a-Service (IaaS) and Platform-as-a-Service (PaaS) are essential forms of cloud services provided for many enterprises. Depending on the cloud user's needs, PaaS and IaaS provide the required web applications and computational resources in the form of virtual machines (VMs). With the widespread of on-demand cloud services, their availability, energy consumption, and other performance issues become paramount aspects for cloud providers and users [1]. Nowadays, cloud users and providers depend on affinity/anti-affinity policies, over-provisioning practices, and multi-zone/region deployments to achieve high availability rather than defining a comprehensive model to analyze the high availability (HA) of cloud applications' deployments. For instance, OpenStack Nova schedulers use anti-affinity/affinity filters and availability zones to deploy applications in geographically distributed data centers (DCs) to maintain HA [2]. Although these notions minimize outage of cloud applications, they are still missing a quantitative model to analyze the applications HA, provide generic guidelines for HA-aware scheduling solutions, and minimize algorithms complexities. It is important to note that the service availability is the percentage of time where this service is available in a given time duration [3].

Although an evaluation model provides generic guidelines to maintain HA of cloud applications, there are still other concerns with respect to the energy, performance, and cost challenges associated with cloud. It is necessary to provide a solution that integrates the HA constraints with the other cloud challenges and provides integrated-aware deployments (e.g., HA and green-aware deployments). This paper proposes an approach that associates a cloud scoring tool with a comprehensive availability analysis model to select the best HA, energy efficient, and/or cost-aware deployments of applications. Fig. 1 summarizes this approach. First, a cloud scheduler generates a set of applications' deployments. Then an availability analysis approach using Stochastic Petri Net model (SPN) is defined. The SPN model evaluates the deployments of different application's components by considering the impact of cloud infrastructure and applications failures, recovery duration, applications redundancy and interdependency relations, load balancing delay, and processing time of the user's request. Once evaluated, these deployments are then inputted to the cloud scoring tool to select the optimal one according to predefined policies, such as lower operational expenditure (OPEX) and/or low carbon footprint. The

- *M. Jammal and A. Shami are with with the Department of Electrical and Computer Engineering, Western University, London, ON N6A 3K7, Canada. E-mail: {mjammal, Abdallah.Shami}@uwo.ca.*
- *A. Kanso is with IBM T.J. Watson Research Center, Yorktown, NY 10598. E-mail: akanso@us.ibm.com.*
- *P. Heidari is with Ericsson Montreal Canada, Saint-Laurent, Québec H4S 0B6, Canada. E-mail: parisa.heidari@ericsson.com.*

Manuscript received 17 Nov. 2016; revised 14 July 2017; accepted 5 Dec. 2017. Date of publication 11 Dec. 2017; date of current version 3 Feb. 2021. (Corresponding author: Manar Jammal.)
Digital Object Identifier no. 10.1109/TSC.2017.2781730

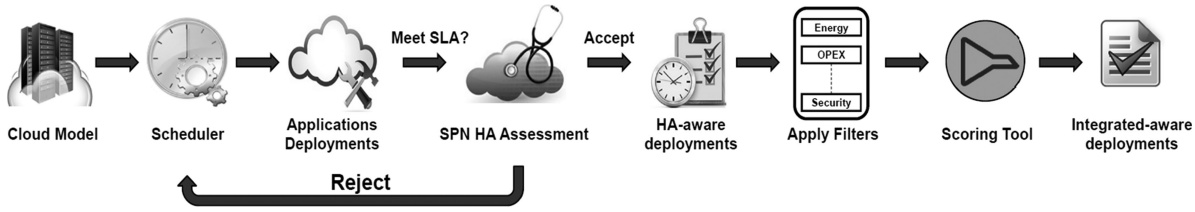

Fig. 1. SPN and scoring selection approach.

scoring system provides a policy-driven ranking system to weight the best HA-aware deployments and select the optimal ones accordingly. It is a generic approach where the evaluation criterion is determined based on the cloud providers preferences, and the selection process is modified accordingly.

The work of this paper is an extension of two other papers [4], [5]. Although [4] and [5] proposed an availability analysis approach of cloud-deployed applications, they discarded SPN practicality and other challenges associated with cloud applications deployments, such as energy and cost efficiency. It is necessary to design a system that integrates HA objectives with other cloud challenges. Therefore, we escalate that work to the following:

- Associate the SPN model with a policy-driven cloud scoring system.
- Capture energy/OPEX as scoring policies to provide HA and green/cost-aware scheduling of cloud applications.
- Integrate the scoring policies with the functionality and availability constraints to select best placements of application components to maximize HA and maintain energy/cost needs.
- Envision user needs and assess DCs capabilities to filter out best HA-aware deployments to minimize Greenhouse Gas (GHG) emissions and OPEX in DCs.
- Evaluate the deployments' results of the SPN model using the scoring tool and comprehensive analysis.
- Provide an extensible scoring system that depends on the generic cloud environment.
- Modify the evaluation criterion based on the capabilities and preferences offered by the cloud providers, such as green and cost criteria to evaluate cloud DCs.
- Generate different patterns and/or guidelines to facilitate the selection between cloud deployment models (public, private, or hybrid) and to improve existing scheduling or DCs models.
- Use the defined guidelines as preliminary analysis to improve algorithms complexities.

The rest of this paper is organized as follows. Section 2 defines the problem background where it presents the need for SPN models and scoring selection system for deployments of cloud applications. Section 3 describes the cloud stack, its failures, the proposed SPN model, and the scoring selection approach. Section 4 describes the evaluation and results of the SPN model and the scoring selection tool. Finally, Section 5 concludes the paper and describes the future work.

2 Background

HA, energy efficiency, and OPEX are gaining a lot of interest in information and communication technology sector and cloud market. With the high energy consumption, DCs are supposed to have performance- and energy-aware configuration measures that can lessen the power use and save OPEX, all aimed at having HA, green, and cost-aware solutions. This section explains the HA-aware scheduling challenges, the need for an appropriate dependability analysis model to handle them, and the necessity to associate the analytical model with a cloud scoring selection tool.

2.1 Stochastic Petri Nets in Cloud

The stochastic nature of service failures and the urgent need for availability solutions require an availability evaluation model that identifies failures and mitigates the associated risks and service outages. It has been shown that analytical models, such as SPNs and Markov chains have been used to analyze the availability of many complicated IT systems [6]. However, the complicated nature of cloud infrastructure configurations and dynamic state changes require a comprehensive and analytical availability-centric model [7]. Petri Nets (PNs) are widely used to model the behavior of different Discrete Event Systems [8]. They are graphically presented as directed graphs with two types of nodes: places and transitions. Deterministic Stochastic Petri Nets (DSPN) are one of PNs extensions for modeling the systems with stochastic and deterministic behaviors [9]. DSPN is presented as a tuple of $(P, T, I, O, H, G, M_0, \tau, W, \Pi)$ where P and T are the non-empty disjoint finite sets of places and transitions, respectively. I and O are the forward and backward incidence functions. H describes the inhibition conditions. G is an enabling function that given a transition and a model state determines whether the transition is enabled. M_0 is the initial marking. The function τ associates timed transitions with a non-negative rational number. The function W associates an immediate transition with a weight (relative firing probability). Finally, Π associates an immediate transition with a priority to determine a precedence among some simultaneously firable immediate transitions.

TimeNET is a powerful PN analysis tool that is maintained regularly, and therefore, it is used to the simulate and analyze the SPN model [10]. TimeNET evaluates DSPN, Automata, and SCPN models. Although DSPN imposes the restriction of only one enabled deterministic transition in each marking and does not support random delays distributions, TimeNET provides transient and stationary analysis of Stochastic Colored Petri Net (SCPN) without any restriction on the number of concurrently enabled transitions. SCPN supports both stochastic and deterministic events, and it is a class of DSPN models where the tokens can have different colors (types) [11]. Also, SCPNs allow random distributions of transitions including "global guards", "zero delays", "time guards", and "complex types of tokens". With this in mind, the paper uses SCPN to model the behavior of an

application running on the cloud with stochastic failures and deterministic recovery events, but it does not make use of the token type feature.

Although the SCPN model captures the cloud characteristics and translates them into elements of an availability model, it overlooks the other challenges associated with the cloud. In the following, we explain the need for a policy-driven scoring system that weights the HA-aware deployments and select the optimal one according to a predefined policy (i.e., green/cost).

2.2 Why a Cloud Scoring System is Needed?

Nowadays, the size of DCs has increased significantly to satisfy the migration to the cloud and the growth in the usage of internet services [12]. Besides, many telcos are selling their DCs and moving to the cloud, such as Verizon and AT&T [13]. With more DCs being built, more services will be provided to the cloud users, and additional investments and incentives will be brought to the market. This increase in the rate of DCs construction is accompanied by a significant growth in energy consumption that might exceed in some scenarios the thresholds introduced by the power delivery and cooling systems. DCs are also going to face an increase in operational costs due to the high energy consumption [14].

To mitigate the above challenges, one solution could be a migration to the cloud and adoption of virtualization concept. The VMs, containers, and consolidation concepts can eliminate idle servers and reduce OPEX while providing 75 percent increase in server efficiency [14], [15]. With the migration to the cloud, its providers are searching for alternative solutions to reduce the high energy consumption and expenditures. They adopt multiple approaches, such as using renewable energy and building DCs in cooler areas to reduce cooling cost and earn carbon tax credits. Facebook has announced the construction of one of the most sustainable and reliable DC, Lulea [16].

It has been shown that power and cooling solutions in DCs can reduce power bills, capital investments for power plants, and GHG emissions, but one major impediment is raised regarding the reliability and performance. It is necessary to delineate an approach that can compromise between the availability, cost, and green requirements. To ensure redundancy and workload proximity, cloud providers should have multiple geographically distributed DCs, each with a different OPEX. Having a profitable cloud necessitates a scoring mechanism that distributes the workload while satisfying the HA requirements (different availability zones, service level agreement (SLA) level) and minimizing DCs energy consumption and OPEX. Note that the scoring selection tool can use objectives other than green and cost efficiency depending on the predefined options of the cloud providers.

3 Approach

To address the challenges of HA, cost, and green-aware scheduling discussed in the previous section, we need first to elaborate a behavioral model that can capture the stochastic nature of different failures in a system and then associate it with an energy- and cost-aware scoring selection tool.

In the following, we explain the transformation from a cloud system to the corresponding SCPN model. Then we describe the the scoring selection tool and its evaluation mechanism.

3.1 Cloud Stack, Failures, and UML Model

The cloud consists of a set of geographically distributed DCs hosting multiple servers and set of applications with multiple software component types. Each type consists of one or more components that might depend on different sponsor component(s). Each type is associated with a redundancy model that determines the number of active, standby, and/or spare components. Using the appropriate placement solution, the components are hosted on the servers that best fit their requirements using VE (VM or container) mapping.

Different forms of failures can occur in the cloud and can be envisioned as planned and unplanned downtimes. Unplanned downtime is the worse failure causes because it is a result of unexpected failure event, and consequently neither the cloud provider nor the users are notified of it in advance. Unplanned downtime can happen at the cloud infrastructure (i.e., faults in memory chips), application's components (i.e., hypervisor malfunctioning or software bugs), or both (i.e., natural disasters). Each of the previous failure states is associated with a failure rate or mean time to failure (MTTF) and mean time to repair or recover (MTTR). Due to the stochastic nature of the corresponding failure events, it is assumed that they are generated using certain probabilistic distribution functions. However, there is no restriction or specific consent on the distribution type of every failure event. It can follow exponential, Weibull, normal, or any other stochastic model. The exponential failure distribution has been used in many previous failure analysis and availability related work [17], [18], and [19]. Therefore, in this paper, the exponential failure distribution is used to reflect failure rate or MTTF of DCs, servers, and applications/VEs. Such distribution is applied on all the stochastic failure transitions of the proposed SCPN model. As for the repair/recovery timed transitions, there is usually a predictable average time that can be estimated to replace a faulty node [20]. Therefore, deterministic distribution is used to trigger any repair or recovery behavior for the DCs, servers, and VMs/applications. It should be noted that our approach also supports other failure rates, as our model does not depend on a specific probability distribution.

Many modeling approaches are developed to describe the heterogeneity of cloud architectures. General-purpose languages are widely used to describe cloud environment. For instance, Unified Modeling Language (UML) can describe the platform, infrastructure, and software artifacts to reflect the characteristics of different cloud components [21]. It can also reflect the service availability features, but as a semi-formal model, it cannot simulate the behavior of the system or measure the availability of a service while different stochastic failures are happening [21]. Creating the SCPN model manually can be a tedious, time-consuming, and error-prone task. To mitigate this complexity, a UML model is designed to describes the above cloud stack and their availability metrics (MTTF and MTTR). Fig. 2 illustrates our modified UML model that captures such cloud deployment. Each application consists of multiple software

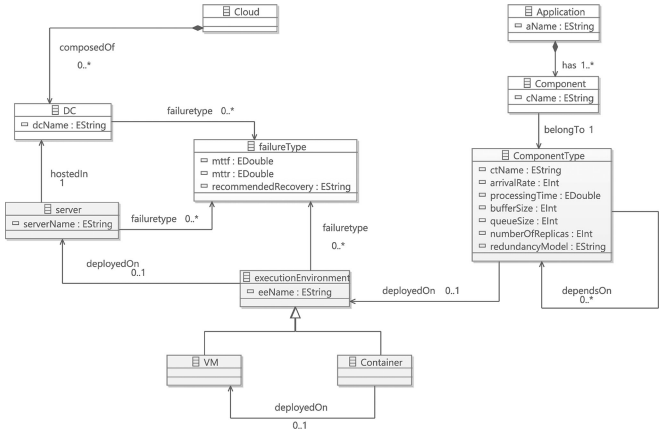

Fig. 2. UML model for a cloud deployment.

components of different types. Each software component has some attributes to capture the incoming workload distribution (*arrivalRate*), the time duration required to process a request (*processingTime*), the number of requests the component can process in parallel (*bufferSize*), the maximum capacity of the requests waiting to be processed (*queueSize*), the number of redundant replicas considered for each component (*numberOfReplicas*), and the redundancy schema of the component (*redundancyModel*) to show which redundancy type a component is capable to accept. Execution environment (VM or container), server, and DC may fail because of different failure types. Each failure type has a failure rate, a recommended recovery action, and recovery duration based on the recommended recovery.

With the transformable property of the UML model, multiple cloud deployments and profiles are generated as reusable templates to identify the mapping between cloud infrastructure and applications. Then these deployments are imported to the SCPN model and scoring system to analyze and select best HA, energy, and cost-aware deployments accordingly.

3.2 SCPN Model Building Blocks

Although many literature studies provide PN models to analyze certain DC or host aspects (i.e., throughput), they model the cloud application as a monolithic one. A monolithic application deployment means that any sudden failure can bring the whole service down. However, cloud providers and users are migrating from monolithic applications toward multi-tiers and microservices architecture. Different studies have shown that overlooking interdependency and redundancy relationships the application level provides undesirable service availability results [18], [19]. Modeling each tier of cloud application's components, their interdependency/redundancy relationships, their virtual environment (VE), and their DC(s)/servers reflects how nowadays software components of a cloud applications are designed to interact. When a sudden failure happens at a certain component, the load balancer redistributes the workload of the faulty component to its redundants. As for its interdependent components, they function normally if they can tolerate its absence; otherwise, the load balancer redistributes their workload as well or executes new scheduling if necessary. Big Data analysis application can be a good example of a three-tier cloud application. At the front end, Filters receive unstructured data and remove redundant/useless data. In the middle, Analysis Engines analyze the data and generate structured data form. At the back end, Databases store the structured data produced by the Analysis Engine. The proposed SCPN model defines a plurality of components of the multi-tier cloud application and a stochastic model including representations of the plurality of components, VEs executing the components, servers executing one or more VEs, and one or more DCs hosting the one or more server. The model generates a dependency graph to reflect the intercommunication between different tiers of the application's components. It identifies a number and order of tiers of the multi-component cloud application. In each tier, it defines a load balancer sub-model, a component sub-model for each of the plurality of redundant application's components, a VE sub-model for the components' VEs, a server sub-model for the components' servers, and a DC sub-model for the components' servers. In this section, we assume that the VE is a VM, and each VM, server, and DC has its own MTTR and MTTF. Then the SCPN model evaluates different deployment possibilities of the multi-component cloud application. This evaluation generates different service availabilities of the multi-component cloud application that is calculate in terms of number of served requests during a given time interval.

Since the SCPN model is evaluating the service availability of a given cloud deployment, each transition is associated with the guards that reflect the transitioning from a healthy to failure state, the failover of the workload between redundant components in same tier, and the workflow of the requests between different application's components' tiers. Although some literature studies provide same conditions for DC, server, and VM states, they overlook the modeling of multi-tier applications' components, fair round robin load balancing, and request workflow and their associated guards, such as failover state that is triggered when a software component or its host(s) fails. In the following, each sub-model and its guard conditions are described to reflect the above states.

1) Data center model: Fig. 3a shows the data center model. A data center has two states: healthy (the place DC_i) and failed (the place DC_i_fail). Failure is modeled using an exponential timed transition (T_i_DCfail) whereas the recovery is a deterministic one (T_i_DCup) [20].

2) Server model: Fig. 3b presents the server model. The server also has two states: healthy (S_i) and failed (S_i_fail). The server can fail, and the failure is an exponential transition (T_i_sfail). It can also fail immediately due to the failure of its hosting data center ($T_i\_sDCfail$). We represent the data center hosting S_i with $S_{(i)DC}$. In the following, we use the place name in the formulas to show the number of the tokens available in that place. The immediate transition $T_i\_sDCfail$ is guarded with

$$G_{T_i\_sDCfail} = (S_{(i)DC} == 0). \tag{1}$$

The recovery occurs according to a deterministic transition (T_i_sUP). A server cannot be recovered unless its host data center is healthy. Thus, T_i_sUP is guarded with

$$G_{T_i\_sUP} = (S_{(i)DC} == 1). \tag{2}$$

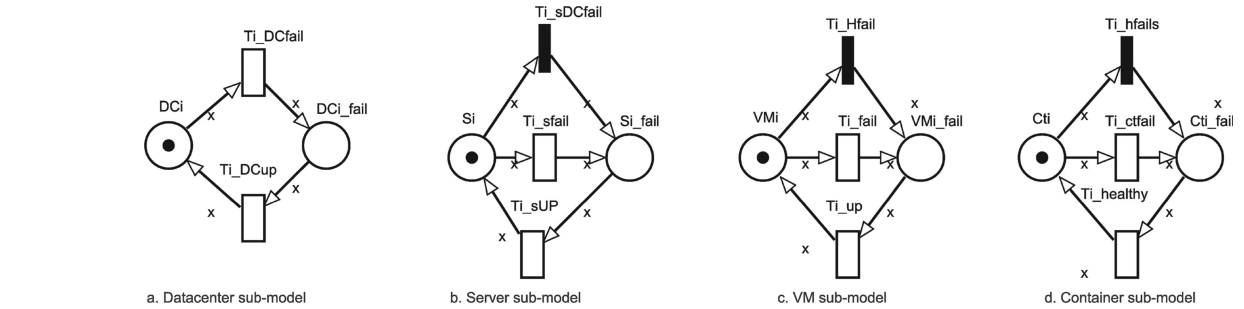

Fig. 3. Data center, server, VM, and container sub-models.

3) VM model: A VM (Fig. 3c) can fail through an exponential transition (T_i*_fail*) or can fail immediately due to the failure of its hosting server or data center (T_i*_Hfail*). We refer to the server and DC hosting the VM with $VM_{(i)Server}$ and $VM_{(i)DC}$, respectively. T_i*_Hfail* is guarded with:

$$G_{T_i\_fail} = \big(VM_{(i)DC} == 0 \vee VM_{(i)Server} == 0\big). \tag{3}$$

The recovery happens after a deterministic delay (T_i*_up*). In this case, also a VM cannot be recovered unless its hosting data center and server are healthy. Thus, T_i*_up* is guarded with

$$G_{T_i\_up} = \big(VM_{(i)DC} == 1 \wedge VM_{(i)Server} == 1\big). \tag{4}$$

4) Container model: A VE (Fig. 3d) can also be a container hosted directly on a server deployed on a DC or hosted on a VM deployed on a server. The container can fail through an exponential transition (T_i*_ctfail*) or can fail immediately due to the failure of its host or DC (T_i*_hfails*). We refer to the host and DC of the container with $Ct_{(i)H}$ and $Ct_{(i)DC}$, respectively where $Ct_{(i)H}$ can be $Ct_{(i)VM}$ or $Ct_{(i)Ser}$. If the container is hosted on a VM then T_i*_hfails* is guarded with: $G_{T_i\_ctfail} = (Ct_{(i)DC} == 0$

$$\vee Ct_{(i)Ser} == 0 \vee Ct_{(i)VM} == 0). \tag{5}$$

If the container's host is a server then T_i*_hfails* guard is

$$G_{T_i\_ctfail} = \big(Ct_{(i)DC} == 0 \vee Ct_{(i)Ser}\big). \tag{6}$$

The recovery happens after a deterministic delay (T_i*_healthy*). Note that in this case, a container cannot be recovered unless its underlying infrastructure are all healthy. Its T_i*_healthy* is guarded with

$$G_{T_i\_healthy} = (Ct_{(i)DC} == 1 \wedge Ct_{(i)Ser} == 1) \tag{7}$$

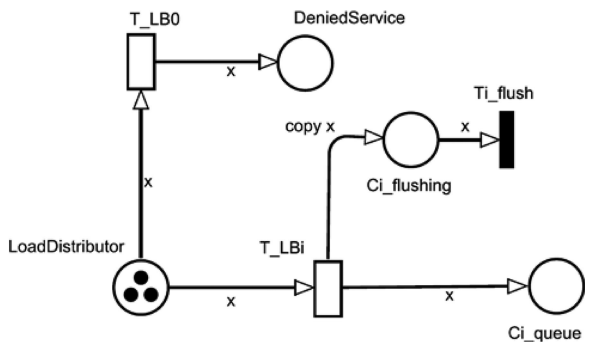

Fig. 4. Load balancer model.

OR

$$G_{T_i\_healthy} = (Ct_{(i)DC} == 1 \wedge Ct_{(i)Ser} == 1 \wedge Ct_{(i)VM} == 1). \tag{8}$$

5) Load balancer model: Load balancing distributes traffic among multiple compute instances. It is an effective way to maintain the availability of a given cloud system. It provides fault tolerance policy in a given application deployment [22]. Upon failure of some instances, load balancer seamlessly replace them while maintaining the normal operation of other nodes/instances.

Fig. 4 illustrates the load distributor and round robin load balancer sub-model. The place *LoadDistributor* has a fixed number of tokens, and the load balancer transitions (T_LB_i and T_LB_0) distribute the workload among the active replicas of the same component. Each component has a queue place (C_i*_queue*) to represent the number of requests it can queue for processing and a flushing place (C_i*_flushing*) for the load balancing mechanism to ensure a round robin distribution. The transitions T_LB_i and T_i*_flush* are guarded such that they model a round robin policy. When a component C_i receives a token in its queue, its flushing place is marked, and the component will not receive another token until its flushing place is unmarked. Let the round robin order be $C_1, C_2, C_3, \ldots C_M$ where M is the number of replicas $(numberOfReplicas)$, and then the same order repeats. The transition T_LB_1 is the first one that becomes enabled, and its clock starts elapsing. Once it is fired, one token is produced in C_1*_queue*, and one token is produced in C_1*_flushing*. As long as C_1*_flushing* is marked, C_1 cannot receive another token. On the other hand, T_1*_flush* cannot be fired until all other components have received their share. As soon as C_1 receives a token, the transition T_LB_2 becomes enabled, and its clock starts elapsing. Then, T_LB_2 fires, and C_2*_queue* and C_2*_flushing* receive a token. The same way other components receive their share until C_M receives a token. At this time, T_1*_flush* is enabled, and C_1*_flushing* is unmarked. Subsequently, T_2*_flush*, T_3*_flush*, $\ldots T_M$*_flush* also fire. According to the nature of workload arrival of the system, T_LB_i can have different distributions. Table 1 lists the timed transitions of the load balancer sub-model.

Note that if a component is not available due to a full queue or a failure in the underlying stack, it should give its turn to the next available component. For M being the number of replicas, L being the maximum capacity of a component queue $(queueSize)$, $VM_{(i)Server}$ and $VM_{(i)DC}$ being the

TABLE 1
Time Function Transitions of Load Balancer and Component Models

Transition Name	Type	Time Function
T_LB_0	Deterministic	DET(comp.arrivalRate)
T_LB_i	Deterministic	DET(comp.arrivalRate)
$T_i\_processed$	Deterministic	DET(comp.processTime)

host server and DC of VM_i, we define $VSD_{H(i)}$ and $VSD_{F(i)}$ as follows:

$$VSD_{H(i)} = [VM_i == 1 \wedge VM_{(i)Server} == 1 \wedge VM_{(i)DC} == 1] \tag{9}$$

$$VSD_{F(i)} = [VM_i == 0 \vee VM_{(i)Server} == 0 \vee VM_{(i)DC} == 0] \tag{10}$$

T_LB_i is guarded with $G_{T\_LB_i}$:

$$\begin{aligned}&\forall_{i\in 1:M} G_{T\_LB_i} = \\ &(C_i\_flushing == 0 \wedge VSD_{H(i)} \wedge C_i\_queue < L) \\ &\bigwedge_{k=1:i-1} (C_k\_flushing == 1 \vee VSD_{F(k)}) \\ &\bigwedge_{j=i+1:M} (C_j\_flushing == 0 \vee VSD_{F(j)})\end{aligned} \tag{11}$$

And T_i_flush is guarded with G_{Ti_flush}

$$\begin{aligned}\forall_{i\in 1:M} G_{T_i\_flush} = &\bigwedge_{j=1:i-1} (C_j\_flushing == 0 \vee VSD_{F(j)}) \\ &\bigwedge_{k=i+1:M} (C_k\_flushing == 1 \vee VSD_{F(k)}).\end{aligned} \tag{12}$$

If all the components fail or their queues are full, the requests are dropped and sent to the place *DeniedService*. Transition T_LB_0 is guarded with

$$G_{T\_LB_0} = \bigwedge_{i=1:M} ((VM_i == 0) \vee (VM_{(i)Server} == 0) \vee (VM_{(i)DC} == 0) \vee (C_i\_queue \geq L)). \tag{13}$$

An alternative solution to model the load distribution is the loop back arcs from T_LB_i and T_LB_0 to the place *LoadDistributor* to continuously re-enable the load balancer transitions and regenerate the workload infinitely. Note that with this alternative approach, we can over-flood the model with tokens if their request arrival rate is faster than the the processing rate of the requests (tokens). To avoid this issue, we fix the number of tokens in the place *LoadDistributor* and do not consider the feedback input arcs. The transitions and their guards remain the same to model the round robin policy. We include both techniques in the paper so that the reader can select the one that best fits their simulation needs.

6) Component model: Fig. 5 illustrates the model of a component including partially the load balancer delivering the workload to the component. Each component has a queue (C_i_queue) to model the maximum capacity of the requests waiting to be processed and also a buffer to model the maximum number of requests a component can process in parallel ($C_i\_processing$), such as multi-threaded components. The requests stored in the queue can enter the buffer only if the component, its corresponding server, and VM are healthy, and the number of tokens already in the buffer is below the maximum. When a component fails, all the requests in its buffer are lost and transferred to the place $Lost\_in\_phase_i$ where 'i' is the tier number. The transition $T_i\_Lost\_in\_Processing$ is guarded with

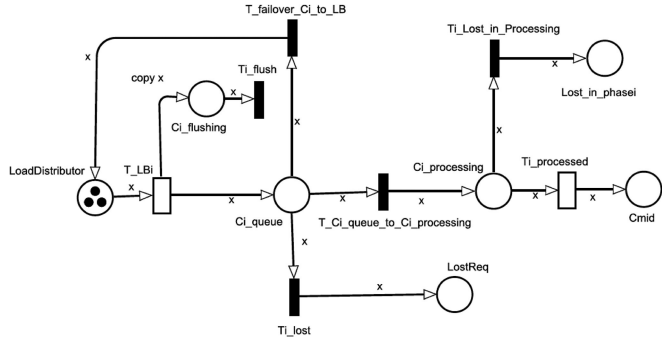

Fig. 5. Component model.

$$\begin{aligned}G_{T_i\_Lost\_in\_Processing} = &((VM_i == 0) \\ &\vee (VM_{(i)Server} == 0) \vee (VM_{(i)DC} == 0))\end{aligned} \tag{14}$$

In addition, in each tier, if all the replicas fail at the same time, all the tokens stored in the component queue are transferred to the place *LostReq*. The transition T_i_Lost is guarded with

$$\begin{aligned}G_{T_i\_Lost} = &\bigwedge_{i=1:M} ((VM_i == 0) \\ &\vee (VM_{(i)Server} == 0) \vee (VM_{(i)DC} == 0)).\end{aligned} \tag{15}$$

When a component fails, the requests already stored in its queue are transferred again to the load distributor to be failed over to the other healthy components. This behavior simulates a multi-active stateful redundancy where each component is equally backed up by the other components. The transition $T\_failover\_C_i\_to\_LB$ is guarded with:

$$\begin{aligned}&\forall_{i\in 1:M} G_{T\_failover\_C_i\_to\_LB} \\ &= (VM_{(i)} == 0 \vee VM_{(i)Server} == 0 \vee VM_{(i)DC} == 0) \\ &\wedge \bigvee_{\substack{j=1:M \\ j\neq i}} (VM_{(j)} == 1 \wedge VM_{(j)Server} == 1 \wedge VM_{(j)DC} == 1).\end{aligned} \tag{16}$$

The tokens successfully processed are stored in the place *Cmid*. Note that in a multi-tier system, the tokens successfully processed in one tier are carried to the next tier where they are load balanced among the replicas of the next tier. The tokens successfully processed in all the tiers are stored in a final place. The availability of the system is only determined by those tokens that reach this final place. Table 1 presents the list of timed transitions and their information. These building blocks are combined to form the complete SCPN model. Fig. 6 illustrates a snapshot of the SCPN model of a 3-tier application running in a cloud environment with 3 active replicas in each tier. The depicted model is using only VM as VE, but it can be easily modified to include containers. In the latter case, the above container sub-model and its complementary guards

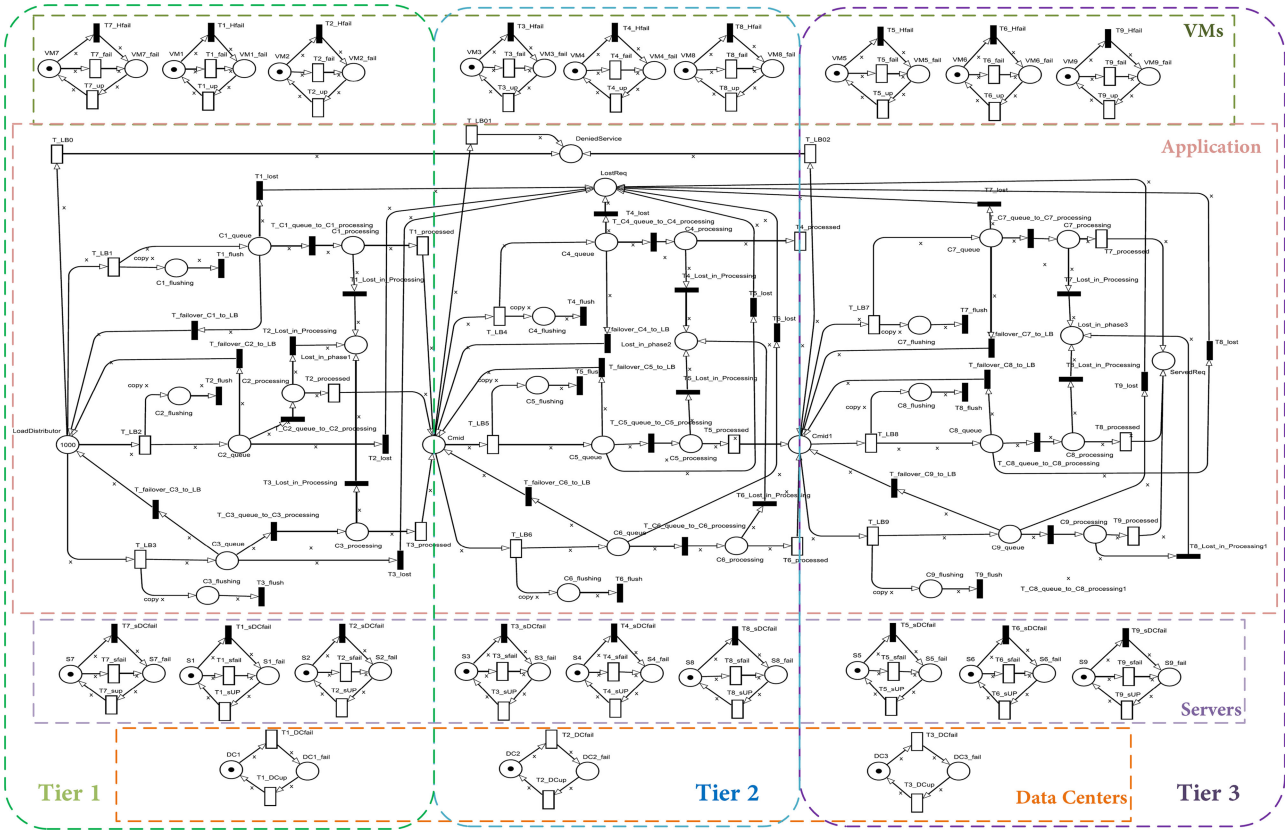

Fig. 6. SCPN model of a three-tier Amazon web application running in a cloud environment.

can be added to the SCPN model to perform availability analysis and quantification.

3.3 Transforamtion Algorithm of the UML to the SCPN Model

In TimeNET, the PN classes are built from an Extensible Markup Language (XML) schema. Taking this into consideration, the transformation approach performs a one to one mapping to generate a solvable SCPN model. Fig. 7 summarizes this approach. It starts with defining an an instance of the UML model (an object model) that represents a certain cloud deployment scenario. It then parses and wraps this instance into the XML data format supported in TimeNET that builds the application's components dependency graph to identify the number of tiers and their orders. Once the XML schema is generated, it is imported to the TimeNET SCPN analysis tool. Fig. 8 shows the XML schema for creating the places, transitions (timed/immediate), and the arcs and measures/expressions that connect map each place to its corresponding transitions. In the XML schema, the transformation algorithm creates the places and transitions that are common in all SCPN models, such as the *LoadDistributor*, *LostReq*, and *DeniedService* places. Then the algorithm iterates over each tier creating the load balancer, all the component replicas, their VMs, and their corresponding servers. For instance, if the model includes five VMs, the VM building block is replicated five times. However, the transition and guards of each building blocks may be different. Then, in the final stage, the DCs are created, the transitions are annotated with the proper rates, and the guards are annotated with the corresponding conditions. It is the annotation phase that glues the model together reflecting the actual deployment and the failure cascading effects. The overall transformation algorithm is described in Fig. 9. Then the approach analyzes this SCPN model using TimeNET to quantify the expected availability of the application.

3.4 Cloud Scoring Approach

Multiple HA-aware deployments might be eligible for the application components with certain MTTF and MTTR

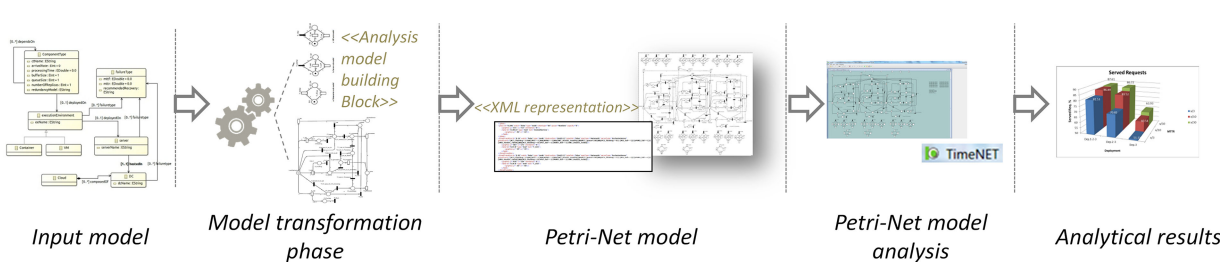

Fig. 7. Overall approach.

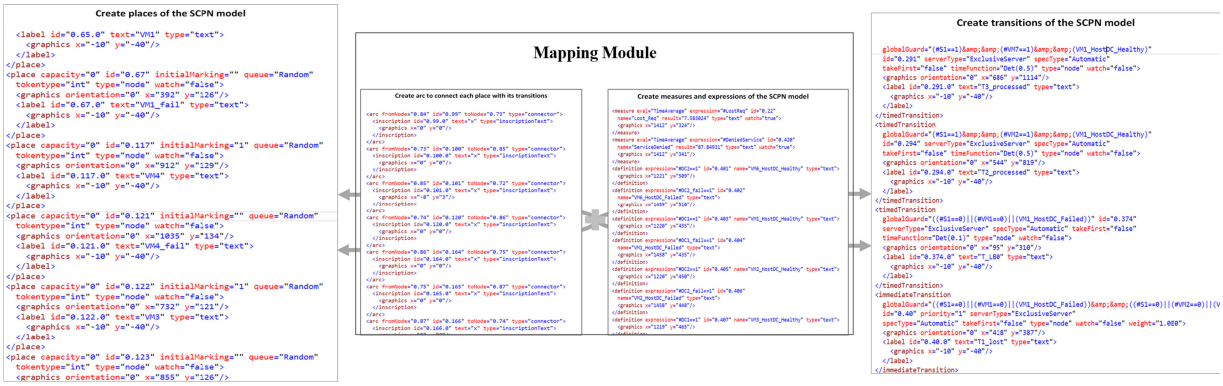

Fig. 8. XML schema to create the SCPN model.

values of examined DCs. For instance, if the cloud user is looking for HA-baseline greater than 90 percent, SCPN evaluation can end up with more than one satisfactory solutions. Therefore, a scoring selection tool is needed to add weights to the selected deployments and select optimal ones among them. The scoring selection tool is extensible and can address different preferences of cloud providers. It has an evaluation criterion with multiple options to allow scoring the deployments. In order to determine a pragmatic evaluation methodology, some afore steps are considered:

1) User Requirements Envisioning: The scoring approach envisions the user requirements and usage patterns to generate certain groupings of the application components. For instance, if the deployment of a 3-tier web application is evaluated using the scoring tool, the envisioning process should consider the interdependencies between components and examine tolerance time of the dependent ones to generate the possible groupings. If the Hypertext Transfer Protocol (HTTP) of a 3-tier web application cannot tolerate the absence of the business logic application (App) component, both components should share same host. In this case, the envisioning process eliminated the maximum distribution deployment option for both HTTP and App. In this work, we focus on green and cost objectives as the evaluation criterion to select optimal placement of the applications components.

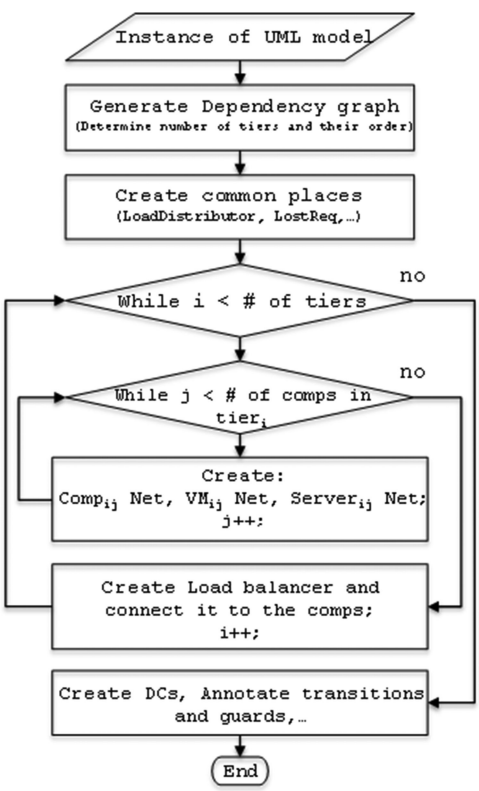

Fig. 9. Transformation algorithm.

2) Cloud Infrastructure Assessment: It is necessary to measure the DCs capabilities in terms of OPEX, carbon footprint, governmental regulations, usage patterns, etc [23]. With these measures, DC workloads can be evaluated, and consequently, the overload factor can be calculated for each DC. Overload represents the increased load that a DC can handle upon a sudden failure, slashdot effect, or any other growth in workload. Therefore, each DC is associated with its overload factor to help select best DC upon load distribution or redirection process. In order to determine the overload factor, it is necessary to select a baseline DC. The baseline DC, DC_b, is the DC that has the highest GHG emissions and OPEX. Therefore, we have assumed that DC_b does not improve OPEX, GHG emissions, or other metrics preference compared to other DCs with higher metrics. Once the baseline is determined, it is assigned an overload factor OL_b of 1. Then the overload factors of remaining DCs, DC_{r_i}, are calculated accordingly. For example, if DC_{r_1} has low carbon footprint, it is assigned up to $x\%$ overload. Subsequently, its overload factor is calculated as follows:

$$OL_{r_i} = OL_b + \frac{x}{100}. \tag{17}$$

Generally, the $x\%$ overload is determined by the cloud provider during the DC planning strategy. This overload percentage is affected by DC size, CPU, network, storage, memory, and power modeling in the corresponding DC [24].

This paper uses carbon footprint and OPEX as DCs assessment metrics. The assessment phase is not only bounded to green and cost metrics, it can be extended to other objectives based on the capabilities and choices of the cloud providers.

3) Evaluation Criteria Extraction: The envisioning process is integrated with the assessment phase, and the suitable criterion is generated accordingly. For instance, if the cloud user requires HA-aware deployments for interdependent application components while taking into consideration energy

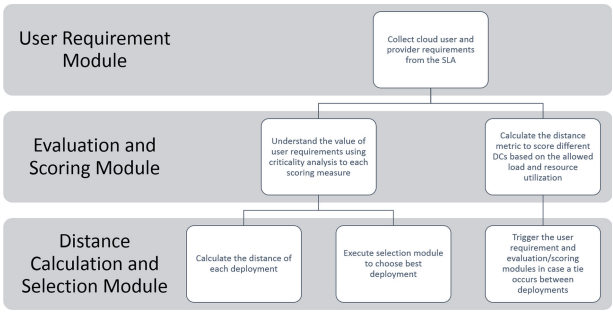

Fig. 10. Modules of scoring selection tool.

efficiency, the evaluation criterion will have low, medium, and high carbon footprint options. Then the overload factors of the DCs are evaluated. Also, an evaluation criterion can be a combination of multiple features/preferences.

3.5 Scoring Selection System of Cloud Deployments

The proposed scoring tool consists of evaluation criteria with multiple options and a scoring methodology. Fig. 10 shows the different modules of the scoring selection tool.

1) Evaluation Criteria: The tool consists of user requirement and assessment modules to determine the measures that add the scores to the deployment solutions. Multiple measures can be used as evaluation criteria.

In order to inject cost and green objectives into the proposed approach, the evaluation criterion assesses the cloud infrastructure in terms of OPEX and carbon footprint. During the assessment process, each DC is examined, and its overload factor is calculated subsequently. For a given OPEX or carbon footprint baseline, or a combination of both, the examined DC operates at a higher load factor, the overload factor, compared to default/baseline DC. This increase in the load factor gives preference for one DC over the others.

2) Scoring Methodology: Once the evaluation criterion is determined, the scoring methodology selects the optimal one. The scoring selection algorithm is depicted in Fig. 11. Each DC is characterized by a distance metric that represents its available capacity before reaching the allowed load. Also, each deployment is characterized by a distance attribute that refers to its corresponding DCs' distances. For the initial deployment, a default preference is defined as the baseline. For subsequent deployments, the algorithm evaluates each eligible deployment distance and selects the one offering the largest distance. In other words, the scoring tool selects the deployment that has one or more DC(s) with the highest capacity to process new workload.

Let *NumDC* be the total number of available DCs and CL_i the current load of corresponding DC_i, then the relative average utilization (*RU*) of DC_i is calculated as follows:

$$\underset{i\in 1:NumDC}{\forall} DC_i.RU_i = \frac{(\sum_{j=1:NumDC|j\neq i} DC_j.CL_j)}{(NumDC-1)}. \quad (18)$$

Let *OL* be the overload factor of each DC, the maximum allowed workload (*AL*) is calculated as follows:

$$\underset{i\in 1:NumDC}{\forall} DC_i.AL_i = DC_i.RU_i \times DC_i.OL_i. \quad (19)$$

Algorithm 1 Scoring Selection Algorithm

INPUT: $DC = (DC_1, DC_2, ..., DC_p)$
$Deployment = (D_1, D_2, ..., D_n)$
$DcDepMap = (D_1, D_2, ..., D_n)(DC_1, DC_2, ..., DC_p)$
$metric = (M_1, M_2, ..., M_t)$
$currentLoad = (CL_1, CL_2, ..., CL_p)$
OUTPUT: $Distance = (Dist_1, Dist_2, ..., Dist_n)$
$maxDistance$
$selectedDeploymentID$

```
begin:
for dc_i ∈ DC do
    metric_total^{dc_i} = Σ_t metric_t^{dc_i}
end for
for dc_i ∈ DC do
    for dc_j ∈ DC do
        if metric_total^{dc_i} < metric_total^{dc_j} then
            findBaseline = metric_total^{dc_i}
            findDcID = i
        end if
    end for
end for
for dc_i ∈ DC do
    if findDcID == i then
        overload_{dc_i} = 1
    else if findDcID ≠ i then
        overload_{dc_i} = calculateOverloadFactor()
    end if
    relativeAverageUtilization_{dc_i} = calculateRU(CL_{dc_i})
    allowedWorkload_{dc_i} = calculateAL(relativeAverageUtilization_{dc_i}, overload_{dc_i})
    dist_{dc_i} = calculatedistance(allowedWorkload_{dc_i}, CL_{dc_i})
end for
for dep_j ∈ Deployment do
    depDistanceSum_{dep_j} = Σ_{dc_i} (dist_{dc_i} × DcDepMap_{dep_j dc_i})
    Distance_{deployment_j} = calculateDeploymentDistance(depDistanceSum_{deployment_j})
end for
for deployment_j ∈ Deployment do
    for deployment_h ∈ Deployment do
        if Distance_{deployment_j} < Distance_{deployment_h} then
            findMaxDist = Distance_{deployment_j}
            findDeploymentID = j
        end if
    end for
end for
selectedDeploymentID = findDeploymentID
maxDistance_{deployment_{selectedDeploymentID}} = findMaxDist
end
```

Fig. 11. Scoring selection algorithm.

Then the distance (*dist*) for each DC is calculated as follows:

$$\underset{i\in 1:NumDC}{\forall} DC_i.dist_i = DC_i.AL_i - DC_i.CL_i. \quad (20)$$

Suppose *Dep* is the set of DCs used in a deployment, and *DepN* is the number of elements in the set *Dep*. Then for every eligible deployment, its distance (*Deployment.dist*) is calculated as follows:

$$Deployment.dist = \frac{(\sum_{\forall i\in Dep} DC_i.dist_i)}{DepN}. \quad (21)$$

Then the eligible deployment that corresponds to the maximum deployment distance is chosen as the optimal solution. The maximum distance measure captures the imbalance between the examined DCs and the preferences of cloud providers (low OPEX/carbon footprint).

The proposed scoring tool is an automated extensible module that can be easily modified to include another evaluation module.

4 Case Study

The system under study is a three-tier web application, such as Amazon Web application deployed using AWS Elastic Beanstalk [25]. In each tier, the software component is running on a VM that is hosted on a server. The server, in turn, is hosted on a DC. Each tier is replicated three times using an active/active redundancy model. In each tier, an elastic

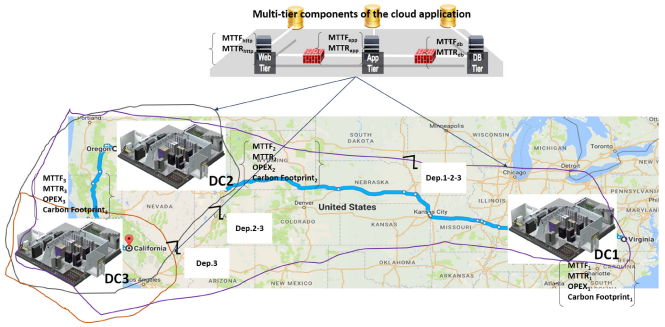

Fig. 12. Case study of multi-tier cloud web application distributed among three DC deployment distributions.

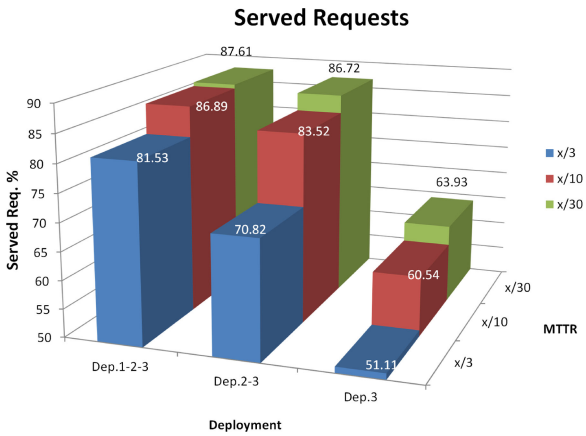

Fig. 13. Service availability of different deployments and different MTTRs. DCs have similar MTTF.

load balancer distributes the workload among the replicas based on a round robin policy.

To investigate different application inter or intra DC deployments, we have considered three deployments cases: the first deployment maximizes the distribution among the DCs, such that in each tier at least one of the replicas is on DC_1, one is on DC_2, and one is on DC_3 (named Dep.1-2-3). In our case, we have assumed that DC_1, DC_2, and DC_3 are located in Virginia, Oregon, and California respectively [26]. In the second deployment, we put one replica of each tier on DC_2 and two other replicas of each tier on DC_3 (called Dep. 2-3). In the third deployment, all the replicas are hosted by the most reliable DC, which is DC_3 (Dep.3 afterward). Fig. 12 shows the case study to be evaluated.

4.1 SCPN Evaluation and Results

The failure of hosting DC has a cascaded impact where its servers, corresponding VMs, and applications' components fail as well. Also, each DC has different OPEX, energy, and other capabilities. Therefore, in this case study, we are particularly interested to compare inter- and intra-DC deployments.

Analyzing the service availability can be done either by (1) quantifying the percentage of time a given service is in a healthy state, or (2) by analyzing the percentage of served requests in comparison to the total number of received requests. We used the latter technique and fixed the number of tokens in the initial *LoadDistributor* place. In each tier, the served requests are stored in a place, which serves as the load distributor of the next tier (e.g., $Cmid$ and $Cmid_1$ places in Fig. 6). The tokens successfully processed in all the tiers are stored in the place *ServedReq* in the 3^{rd} tier. If all the components fail, or their queues are full, the requests are dropped and sent to the place *DeniedService*. When a component fails, the requests already stored in its queue are resent to the load distributor to be failed over to the other healthy components. $Lost\_in\_phase_1$, $Lost\_in\_phase_2$, and $Lost\_in\_phase_3$ collect in each phase the lost requests from the components buffers. If all the replicas of a tier fail at the same time, all the tokens waiting in the components queues are transferred to the place *LostReq*.

The VMs and servers can fail due to DC failure through immediate transitions $T_i\_sDCfail$ and T_i_Hfail. The VMs and servers MTTF (used in T_i_fail and T_i_sfail) are fixed in these experiments. We consider that DCs can have similar or different MTTF. As a baseline, they all have the same MTTF (x, x, x). Then we modify MTTF of the DCs (x, 2x, 3x) assuming that DC_1 fails more frequently, DC_3 is always the most reliable one, and DC_2 has a MTTF between the two others. Then, we consider different MTTR for each variation of the MTTF. However, recovery time is always the same among the DCs. We aim to evaluate which of the above three deployments would maximize the availability of the application. If DC_3 is the most reliable one, is it better to choose the third deployment and put all of the replicas on the most reliable DC or is it better to maximize the distribution among the DCs? The model presented in Fig. 6 is analyzed with transient simulation of TimeNET4.2 running on a Linux VM with 225GB of RAM and 20 vCPUs running Ubuntu12.04. The results are the outcome of multiple repetitions of the simulation. First, we consider the case where all of the DCs have the same MTTF (x, x, x), and we vary the MTTR among DCs. 'x' ranges from 30 to 90 weeks, the MTTR is measured as a ratio of 'x' where MTTR values are x/3, x/10, and x/30 hours, and the request processing time ranges between 0.1 to 1 second. The MTTF and MTTR are instantiated to maintain their within the allowed downtime for cloud providers [25], [27]. Fig. 13 shows the results for the above three deployments. When the DCs have the same MTTF, we should go for a maximum distribution as it reduces the probability of the service outage due to multi-DC failures.

In the second step, we change the MTTF of DC_1, DC_2, and DC_3 to x, 2x, and 3x, respectively and change the MTTR to x/3, x/10, and x/30. The results are presented in Fig. 14. Based on these results, when the reliability of DCs differs, we can opt for the most reliable ones instead of maximum distribution. A single DC deployment is not the optimal choice.

In the last experiment, a comparative analysis is performed between the SCPN results of the deployments of multi-tier application's components and redundancy-agnostic deployment of a monolithic application. Fig. 15 shows the results of the comparative analysis. In case of multi-tier cloud application, we assume DCs have same MTTF and same MTTR. The MTTF value changes from 30 t_u to 90 t_u and MTTR is the MTTF/3 where t_u is TimeNET time unit. In case of the redundancy-agnostic deployment, the whole application is placed in the same DC due to the monolithic architecture. Since the redundancy-agnostic

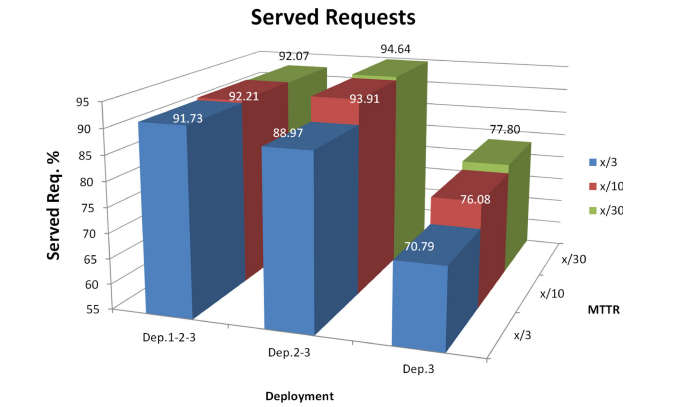

Fig. 14. Service availability of different deployments and different MTTRs. DCs have different MTTF (x, 2x, 3x).

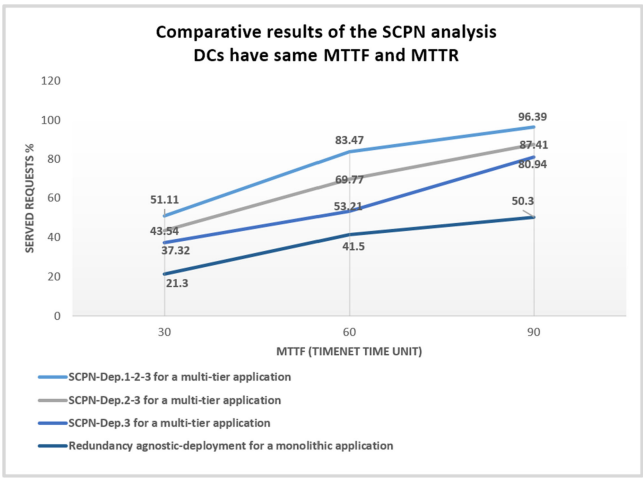

Fig. 15. Comparative results of the SCPN model for multi-tier and monolithic application.

TABLE 2
DC Evaluation Metrics of the First Case

DC	OPEX option (%)	Carbon footprint option (%)	OL (%)	OL factor	CL (%)
DC_1	medium	none	20	1.2	42
DC_2	none	low	10	1.1	41
DC_3	none	none	0	1.0	40

deployment does not support any redundancy model, a failure can then brings the whole application down. Therefore, it shows the lower number of served request as shown in Fig. 15. As for multi-tier cloud application, the maximum distribution shows the highest number of served requests because the DCs have same MTTF. In this comparison, the analysis of the SCPN model can improve the redundancy-agnostic deployment by extract the following guidelines:

- Reschedule the application and deploy it in the DC with the highest MTTF, or
- Migrate from a monolithic architecture to a multi-tier one and opt to the maximum DCs distribution, or
- Scale up the application to include redundant one(s).

The proposed SCPN approach is a framework providing HA-aware placement guidelines where these inferred clues can be applied to different scheduling scenarios. Note that solving a model may take some hours due to the complicated stochastic analysis.

TABLE 3
DC Distances of the First Case

DC	RU(%)	AL(%)	dist(%)
DC_1	40.5	48.6	6.6
DC_2	41	45.1	4.1
DC_3	41.5	41.5	1.5

TABLE 4
Deployment Distances of the First Case

Dep	Deployment Distances
Dep.1-2-3.dist	4.06
Dep.2-3.dist	2.8

TABLE 5
Optimal Deployments of First Case

Dep	HA-baseline $\geq 80\%$
Eligible *Dep(s)*	Dep.1-2-3 & Dep.2-3
Optimal *Dep*	Dep.1-2-3

4.2 Scoring Selection System Evaluation and Results

To select the optimal deployment, the scoring selection algorithm is applied to the above SCPN evaluation results. Since we focus in this paper on the DC failures impact on HA, the evaluation criterion is applied to DCs. Two cases are presented to evaluate the selected deployments against different policies. In the first case, the criterion is OPEX and carbon footprint while in the second case only carbon footprint is considered.

The scoring selection algorithm is applied to the above SCPN evaluation cases: (same MTTF, different MTTR) and (different MTTF and MTTR) using Dep.1-2-3, Dep.2-3, and Dep.3 deployments. We aim to select the best deployment if multiple eligible ones are chosen by the SCPN model.

1) First scoring case: In this case, each DC is examined in terms of OPEX and carbon footprint, and its corresponding overload factor is generated. Table 2 shows an example of metrics that characterize each DC, such as current load (CL), overload factor (OL), OPEX, and carbon footprint improvement options. The option can be either high, medium, low, or none where "high" represents high improvement in OPEX or carbon footprint reduction, and "none" reflects the opposite state.

Table 3 shows the calculated (RU), (AL), and $(dist)$ for each DC using (18)–(20). Using values of Table 3 and (21), the deployment distances are calculated for each of evaluated placements as shown in Table 4.

The scoring selection algorithm is applied to the three cases introduced in Section 4.1. The results are shown in Table 5. In the first case (same DCs MTTF, different DCs MTTR), Dep.1-2-3 and Dep.2-3 are the eligible solutions for MTTF of (x) and MTTR of (x/10 and x/30) if the desired HA-baseline is greater than 80 percent. In the second case, Dep.1-2-3 and Dep.2-3 are the eligible solutions for MTTF of

TABLE 6
DC Carbon Metrics in 2013 Used in Second Case

DC	Carbon Emission (kg/million Btu)	Carbon footprint option (%)	OL (%)	OL factor	CL (%)
DC_1	52.5	none	0	1.0	55
DC_2	35.6	medium	39	1.39	10
DC_3	51.4	low	2	1.02	25

(x, 2x, and 3x) and MTTR of (x/3, x/10, and x/30) if the desired HA baseline is greater than 80 percent. Once the eligible solutions are selected, the scoring algorithm calculates the (RU), (AL), and, $(dist)$ for each DC. With these parameters, the $Deployment.dist$ is calculated, and consequently, Dep.1-2-3 is the optimal deployment since it has maximum distance compared to the others.

If the desired HA baseline is greater than 87 percent, first case generates one eligible solution, Dep.1-2-3 for MTTF of (x) and MTTR of (x/3). With the same HA-baseline applied to the second case, Dep.1-2-3 and Dep.2-3 are the best placements for MTTF of (x, 2x, 3x) and MTTR of (x/10 and x/30). Therefore, the scoring algorithm is only applied to the second case where DCs have different MTTF of (x, 2x, 3x).

2) Second scoring case: In this case, each DC is examined in terms of carbon emission based on the U.S. energy report [28]. Table 6 shows the carbon emissions of industrial sectors in California, Oregon, and Virginia where the above three DCs are located [28]. Since Virginia has highest carbon emissions, its DC, DC_1, is considered the baseline one, and consequently its (OL) is one. The deployments evaluation is based only on the carbon emission factor. Similarly, the option can be either high, medium, low, or none.

Tables 7 and 8 show the calculated (RU), (AL), $(dist)$, and deployment distances for each DC and evaluated placements using (18)-(21).

The scoring selection algorithm is applied to the three cases introduced in Section 4.1. The results are shown in Table 9. In the first case (same DCs MTTF, different DCs MTTR), Dep.1-2-3 and Dep.2-3 are the eligible solutions for MTTF of (x) and MTTR of (x/10 and x/30) if the desired HA-baseline is greater than 80 percent. In the second case, Dep.1-2-3 and Dep.2-3 are the eligible solutions for MTTF of (x, 2x, and 3x) and MTTR of (x/3, x/10, and x/30) if the desired HA baseline is greater than 80 percent. The scoring algorithm calculates the (RU), (AL), and, $(dist)$ for each DC of the eligible deployments. Then, the $Deployment.dist$ is calculated, and consequently, Dep.2-3 is the optimal deployment since it has maximum distance compared to the others.

Note that a change in the DC workload, its OPEX, or carbon footprint option affects the (RU), (AL), and, $(dist)$ calculation. Consequently, different deployment might win the scoring test since $Deployment.dist$ of the eligible solutions will be modified.

TABLE 7
DC Distances of the Second Case

DC	RU(%)	AL(%)	dist(%)
DC_1	17.5	17.5	−37.5
DC_2	40	55.6	45.6
DC_3	32.5	33.15	−8.15

TABLE 8
Deployment Distances of the Second Case

Dep	Deployment Distances
Dep.1-2-3.dist	−0.016
Dep.2-3.dist	18.725

4.3 Approach Discussion

The understandability and practicality of both the SCPN model and the cloud scoring tool are discussed below.

4.3.1 SCPN Discussion

The Petri net model is used to perform the following:

i) Provide guidelines to improve HA-schedulers. It provides a preliminary analysis that allows eliminating some of the deployment options when executing the algorithm. Consequently, the complexity of an HA-aware scheduling algorithm is reduced. For instance, let us assume we have a scheduling algorithm with 3 DCs of different reliability values, and each DC hosts 100 servers. Since the DCs have different reliability, the proposed SCPN model indicates that the maximum DCs distribution option can be eliminated from the scheduling search. If we assume that the scheduling algorithm has O(n^2) complexity, the latter is reduced to O($(n - n_e)^2$) where n is number of servers and n_e is number of eliminated servers. In this case, the best case scenario is O($((n - ((n_{dc} - 1) * n_e^{dc}))^2$) and the worst case scenario is O($(n - n_e^{dc}))^2$) where n_e^{dc} is number of eliminated servers in one DC. In both cases, the assessment and the guidelines extracted from the SCPN model can enhance the scheduling complexity.

ii) Evaluate existing deployments of cloud applications in terms of HA objective. Once assessing the deployments, it can be determined if they meet the SLA.

In this paper, the focus of the SCPN model is to extract different directives to improve the availability of cloud applications' deployments and reduce the complexity of the scheduling algorithms. In other words, if a deployment setting has large number of DCs, servers, and VMs, there is no need to evaluate the model with this setting. It would be enough to sample a number of VMs and their distribution in DCs and extract the DCs that can be eliminated in the scheduling policy. State explosion is one of the challenges of a state space models, such as PN. In this case, the number of states increases exponentially. The state explosion occurs when the PN analysis tool cannot process the PN model due to the lack of memory and/or absence of abstraction and model

TABLE 9
Optimal Deployments of the Second Case

Dep	HA-baseline $\geq$ 80%
Eligible *Dep(s)*	Dep.1-2-3 & Dep.2-3
Optimal *Dep*	Dep.2-3

construction techniques. However, the existing VMs have high computational resources, which increase the efficiency of the PN analysis and verification tools [29]. With this in mind, this paper uses a powerful machine with 225 GB of RAM and 20 vCPUs, and the proposed model is divided into sub-analyzable models where each tier is analyzed before aggregating the whole system. In this case, the state spaced is reduced where the DC impact on the application's components is evaluated while preserving the VM and server states. In this transparent construction-time reduction mechanism, evaluation questions can be answered with the same tool as with the whole state space model.

4.3.2 Scoring Tool Discussion

HA is one of major issues in the cloud; a failed application's components can hinder the functionality of the whole application and can have huge impact on the customer relation managements. With this in mind, HA is the primary objective in this work. Once a set of application's deployments is defined and assessed, the scoring tool is then used to select the best while considering green, cost, and other performance aspects. It is necessary to note that the scoring tool is not a scheduling algorithm, but it is a selection mechanism that opts for the best deployment (among set of many) while satisfying certain performance requirement(s). However, if the cloud providers and users consider for instance, energy/cost reduction as their primary objective when deploying an application component, the scoring tool can be used before the SCPN model to select best set of deployments from a green perspective. Then the SCPN model can assess this deployment and decides whether it satisfies the SLA or not. In this case, the above SCPN results change because the evaluated deployments are HA-agnostic ones.

The scoring tool does not only determine which cloud deployment is the best fit for a given application, but it can also decide whether a cloud deployment solution is suitable for a business application. In this case, cloud and no-cloud deployments can be inputted to the scoring tool. Then each DC of the assessed deployments can be associated with its cloud business factors (OPEX versus Capital Expenditure, time to market, or return investment) and architecture aspects (energy, security, latency, or data restriction/locality). As shown above, each deployment option is then associated with a score (overload and distance weights) to choose between cloud or no-cloud deployment for a certain business applications. Similarly, the scoring tool can be used to choose among public, private, or hybrid cloud solution. In this case, deployment of each cloud model is associated with the desired business and architecture aspects. Once scores are applied to each deployment, the tool selects the deployment with the largest score. Then it answers whether a public cloud model is an applicable option or the applications should be deployed in private or hybrid cloud model. It is necessary to note that the scoring tool can be associated with any cloud scheduling solution for applications. It is not limited to the use with the SCPN model.

5 Conclusion

It is not enough to provide HA-aware cloud solution that can mitigate failures and maintain certain availability baseline, but it is necessary to assess such solution and its resiliency to any failure modes. Additionally, it is essential to integrate such assessment with green and cost requirements to uphold the quality of service with lower carbon footprints and OPEX. With these objectives, this paper proposed a SCPN model that evaluates the inter and intra-DC deployments of cloud services. The SCPN model inputted the HA-aware deployments into a scoring selection tool. Using the latter algorithm, HA-aware placements are filtered in terms of energy and cost metrics to select the optimal deployment. The scoring selection tool is extensible to different criteria and is not limited to the aforementioned measures. In future work, the proposed scoring tool will be extended to include a visualization module and a machine learning algorithm to generate patterns about user requirements assessment.

Acknowledgments

This work is partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC-STPGP 447230) and Ericsson Research. We would like to thank Prof. Armin Zimmermann for his insights.

References

[1] H. Hawilo, A. Kanso, and A. Shami, "Towards an elasticity framework for legacy highly available applications in the cloud," in *Proc. IEEE World Congr. Serv.*, Jul. 2015, pp. 253–260.

[2] OpenStack, "Filter scheduler," 2010. [Online]. Available: http://docs.openstack.org/developer/nova/filter_scheduler.html, Accessed on: 17-Jun.-2016

[3] TechTarget, "Reliability, availability and serviceability (RAS)," 2017. [Online]. Available: http://whatis.techtarget.com/definition/Reliability-Availability-and-Serviceability-RAS, Accessed on: Jun. 2017

[4] M. Jammal, A. Kanso, P. Heidari, and A. Shami, "Availability analysis of cloud deployed applications," in *IEEE Int. Conf. Cloud Eng.*, Apr. 2016, pp. 234–235.

[5] M. Jammal, A. Kanso, P. Heidari, and A. Shami, "A formal model for the availability analysis of cloud deployed multi-tiered applications," in *Proc. 3rd IEEE Int. Symp. Softw. Defined Syst.*, Apr. 2016, pp. 82–87.

[6] K. S. Trivedi, D. Kim, and R. Ghosh, "System availability assessment using stochastic models," *Appl. Stochastic Models Bus. Ind.*, vol. 29, no. 2, pp. 94–109, 2013.

[7] R. Ghosh, D. Kim, and K. S. Trivedi, "System resiliency quantification using non-state-space and state-space analytic models," *Rel. Eng. Syst. Safety*, vol. 116, pp. 109–125, 2013.

[8] C. Petri, *Kommunication Mit Automaten*. Bonn, Germany: University of Bonn, 1962.

[9] G. Ciardo and C. Lindemann, "Analysis of deterministic and stochastic Petri Nets," in *Proc. 5th Int. Workshop Petri Nets Perform. Models*, 1993, pp. 160–169.

[10] A. Zimmermann, "Modeling and evaluation of stochastic Petri Nets with TimeNET 4.1," in *Proc. 6th Int. Conf. Perform. Eval. Methodologies Tools*, 2012, pp. 54–63.

[11] A. Zimmermann and M. Knoke, "A software tool for the performability evaluation with stochastic and colored Petri Nets," Mar. 2007. [Online]. Available: http://www2.tu-ilmenau.de/sse_file/timenet/ManualHTML4/UserManual.html, Accessed on: Jun. 2017

[12] Data Center Knowledge, "Undertaking the challenge to reduce the data center carbon footprint," Dec. 2014. [Online]. Available: http://www.datacenterknowledge.com/archives/2014/12/17/undertaking-challenge-reduce-data-center-carbon-footprint/, Accessed on: Nov. 2015

[13] Data Center Dynamics, "Verizon to auction its data centers report," Jan. 2016. [Online]. Available: http://www.datacenterdynamics.com/design-strategy/verizon-to-auction-its-data-centers-report/95445.article, Accessed on: Jan. 2016

[14] Ingram Micro Advisor, "How data center design impacts efficiency and profitability," Jul. 2015. [Online]. Available: http://www.ingrammicroadvisor.com/data-center/how-data-center-design-impacts-efficiency-and-profitability, Accessed on: Jan. 2016
[15] H. Hawilo, A. Shami, M. Mirahmadi, and R. Asal, "NFV: State of the art, challenges, and implementation in next generation mobile networks (vEPC)," *IEEE Netw.*, vol. 28, no. 6, pp. 18–26, Dec. 2014.
[16] EuroNews, "Facebook boasts green data centre in Lule, Sweden," Oct. 2015. [Online]. Available: https://goo.gl/SaHSD3, Accessed on: 25-Oct-2015
[17] J. Xu, X. Li, Y. Zhong, and H. Zhang, "Availability modeling and analysis of a single-server virtualized system with rejuvenation," *J. Softw.*, vol. 9, no. 1, pp. 129–139, Jan. 2014.
[18] M. Jammal, A. Kanso, and A. Shami, "High availability-aware optimization digest for applications deployment in cloud," in *Proc. IEEE Int. Conf. Commun.*, Jun. 2015, pp. 6822–6828. [Online]. Available: http://vixra.org/pdf/1410.0193v1.pdf
[19] M. Jammal, A. Kanso, and A. Shami, "CHASE: Component high-availability scheduler in cloud computing environment," in *IEEE Int. Conf. Cloud Comput.*, 2015, pp. 477–484.
[20] J. O. Grady, *System Requirements Analysis*. Amsterdam, The Netherlands: Elsevier, Dec. 2013.
[21] S. Bernardi, J. Merseguer, and D. Petriu, "An UML profile for dependability analysis and modeling of software systems," *Tech. Rep.*, May 2008. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.205.4357&rep=rep1&type=pdf
[22] Amazon Web Services, "Web application hosting," 2016. [Online]. Available: https://media.amazonwebservices.com/architecturecenter/AWS_ac_ra_web_01.pdf, Accessed on: May 2016
[23] Oracle, "Oracle's approach to cloud," 2012. [Online]. Available: https://goo.gl/RTtg7c, Accessed on: Dec.-2015
[24] S. Shen, V. Beek, and A. Iosup, "Statistical characterization of business-critical workloads hosted in cloud datacenters," in *5th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput.*, 2015. pp. 465–474.
[25] A. Adegoke and E. Osimosu, "Service availability in cloud computing-threats and best practices," *Bachelor Thesis*, Jun. 2013. [Online]. Available: http://www.diva-portal.se/smash/get/diva2:646329/FULLTEXT01.pdf
[26] Amazon Web Services, "AWS global infrastructure," 2016. [Online]. Available: https://aws.amazon.com/about-aws/global-infrastructure/, Accessed on: May-2016
[27] CloudHarmony, "Service status," 2017. [Online]. Available: https://cloudharmony.com/status, Accessed on: 13-Jun.-2017
[28] U.S. Energy Information Administration, "Energy-related carbon dioxide emissions at the state level, 2000–2013," Oct. 2015. [Online]. Available: http://www.eia.gov/environment/emissions/state/analysis/pdf/stateanalysis.pdf, Accessed on: Apr. 2016
[29] M. Camilli, "Coping with the state explosion problem in formal methods: Advanced abstraction techniques and big data approaches," *Doctor Philosophy Thesis*, Feb. 2015. [Online]. Available: https://air.unimi.it/retrieve/handle/2434/264140/367004/phd_unimi_R09619.pdf

Manar Jammal is a Research Associate at Western University. She received her B.E. in Electrical and Computer Engineering in 2011 from the Lebanese University, Beirut Lebanon. In 2012, she received her M.E.Sc. in Electrical and Electronics Engineering from the Ecole Doctorale des Sciences et de Technologie, Beirut Lebanon and from the University of Technology of Compiegne, France. In 2017, she received her Ph.D. degree in high availability of cloud applications from Western University, London Canada. Her research interests include cloud computing, virtualization, high availability, cloud simulators, machine learning, software defined network, and virtual machine migrations. She is the chair of IEEE Women In Engineering (WIE) group, London, ON and vice-chair of IEEE Canada WIE group.

Ali Kanso is a senior Cloud Software Engineer at IBM T.J. Watson research center working on the IBM next generation container Cloud. He is also an adjunct research professor at Western University. Dr. Kanso earned his masters and Ph.D. degrees in Electrical and Computer Engineering from Concordia University, Montreal Canada in 2008 and 2012 respectively. He holds to his credit over 50 publications including 12 patents granted and several pending. He previously held the position of a senior researcher at Ericsson research Cloud technologies. Dr. Kanso has over a decade of industrial research experience where his research interests include distributed systems and lightweight virtualization in cloud computing environments. He is a member of the IEEE.

Parisa Heidari is currently working at Ericsson, Montreal, Canada. She received her Ph.D. on controller synthesis of real time systems modeled by Time Petri Nets and her M.Sc. on tracing virtual systems both from Ecole Polytechnique de Montreal, Canada in 2012 and 2007, respectively. She worked as a research associate on high availability middleware for cloud systems at Concordia University from 2013 to 2015. In 2015, she joined Ericsson first as postdoctoral research fellow and later as software designer. Her research interests include cloud storage, container technologies, new generation of cloud, resource dimensioning, and different aspects of QoS assurance in cloud systems. She is a member of the IEEE.

Abdallah Shami received the B.E. degree in electrical and computer engineering from the Lebanese University, Beirut, Lebanon, and the Ph.D. degree in electrical engineering from the Graduate School and University Center, City University of New York, in 1997 and September 2002. In September 2002, he joined the Department of Electrical Engineering, Lakehead University, Thunder Bay, ON, Canada as an assistant professor. Since July 2004, he has been with Western University, Canada where he is currently a professor in the Department of Electrical and Computer Engineering. His current research interests include the area of network optimization, cloud computing, and wireless networks. He is an editor of the *IEEE Communications Tutorials and Survey* and has served on the editorial board of the *IEEE Communications Letters (2008–2013)*. He is an IEEE Distinguished Lecturer and senior member of the IEEE and was the elected chair of the IEEE London Section and chair of IEEE Communications Society Technical Committee on Communications Software.