A Formal Model for the Availability Analysis of Cloud Deployed Multi-Tiered Applications

Manar Jammal
ECE Department
Western University
London ON, Canada
mjammal@uwo.ca

Ali Kanso
Ericsson Research
Ericsson
Montreal, Canada
ali.kanso@ericsson.com

Parisa Heidari
Ericsson Research
Ericsson
Montreal, Canada
parisa.heidari@ericsson.com

Abdallah Shami
ECE Department
Western University
London ON, Canada
ashami2@uwo.ca

***Abstract*—High availability is a critical requirement for cloud deployed services. Cloud providers offer different availability zones with geo-redundancy to protect their infrastructure and consequently their tenants against failures and natural disasters. Nevertheless, different zones may have different reliability levels depending on the hardware equipment, the geo-location, the energy source powering the facility, etc. Hence, the ability to assess the expected availability of a given deployment is extremely important for both the cloud tenants and providers that are bound by a service level agreement. Due to the stochastic nature of failures, a formal stochastic model is needed to quantify the expected availability offered by an application deployment. This paper presents a Stochastic Petri Net model to evaluate the availability of cloud services and their deployment in geographically distributed data centers. The proposed Stochastic Petri Net model captures the characteristics of the cloud provider and user. It translates them into elements of an availability model that can be solved to calculate the expected availability and subsequently be used to guide the cloud scheduling solution.**

***Index Terms*—High availability, cloud applications, software components, virtual machines, stochastic failures, stochastic Petri Net, recovery, load balancing.**

I. Introduction

With the cloud computing era, many business applications are offered as cloud services where they can be accessed anytime and anywhere [1] [2]. Depending on the needs of cloud users, Infrastructure-as-a-Service (IaaS) and Platform-as-a-Service (PaaS) provide the required web applications and computational resources in the form of virtual machines (VMs). With the widespread of on-demand cloud services/VMs, their availability becomes a paramount aspect for cloud providers and users. It is important to note that availability is the percentage of time where these services are available in a given duration. Cloud services encounter different types of hardware and software failures and consequently become unavailable [3]. As for the cloud users, they cannot prevent or mitigate the service downtime unless they have their proprietary high availability (HA) solutions, such as the Netflix HA approach [4]. Therefore, the stochastic nature of service failures and the urgent need for availability solutions require an availability evaluation model that identifies failures, their underlying causes, and mitigates associated risks and service outages. It has been shown that analytical models, such as Stochastic Petri Nets (SPNs) and Markov chains have been used to analyze the reliability/availability of many complicated information technology (IT) systems [5] [6]. However, the complicated nature of cloud infrastructure configurations and dynamic state changes require a comprehensive and analytical availability-centric model.

A few literature studies address scheduling of cloud services and their availability analysis using different extensions of Petri Net models. In [7], the authors propose an availability analysis approach for cloud computing systems using Stochastic Reward Net (SRN) and Markov chain models. They analyze the impact of changing the number of physical machines, their mean time to failure (MTTF), and mean time to repair (MTTR) on the service availability. In [8], the authors propose statistical models to predict availability of a distributed system and to find host subsets with related statistical characteristics and availability models. While [9] and [10] propose queuing and Stochastic Petri Net service availability models through software rejuvenation and failure prevention, [11] describes the impact of adding servers on service availability using Colored Stochastic Petri Net model. When it comes to applications scheduled in a cloud model, their availability is not only affected by the existing infrastructure or the cloud user side, but it depends on of both, in addition to the effect of failover solutions, requests processing time, and interaction between different components and their hosts.

In this paper, our availability analysis approach is based on a Stochastic Petri Net model to evaluate the availability of cloud services and their deployment in geographically distributed data centers (DCs). The main contributions of this approach are:

- Ability to capture the stochastic nature of failures according to different probability distribution functions.
- Ability to capture the cloud elements (DCs, servers, and VMs) and the correlation aspect of their failures.
- Ability to capture the functional workflow between the components of multi-tiered applications (queuing and request forwarding) as well as the high availability mechanisms they employ (load balancing and redundancy schemes).
- Ability to assess and quantify the expected availability of application components according to their different

978-1-5090-3684-4/16 $31.00 © 2016 IEEE
DOI 10.1109/IC2EW.2016.21

deployments in the cloud (inter-DC vs. intra-DC deployments).

The rest of this paper is organized as follows. Section II defines the problem background where it presents the challenges of scheduling application components and the need for Stochastic Petri Net models. Section III describes the cloud model and the proposed Stochastic Petri Net model. The evaluation and results of this model are presented in Section IV. Finally, Section V concludes the paper.

II. Background

Many high availability solutions have been proposed to mitigate the software or hardware failures of a virtualized system [12], [13], and [14]. However, these approaches do not associate their HA-aware solutions with an availability assessment model. Different types of failures affect the cloud infrastructure and applications. Besides, some challenges are raised when choosing best deployment of cloud applications while satisfying the HA and functionality requirements. Therefore, it is necessary to understand the HA-aware scheduling challenges and the need for a SPN model to handle them.

A. Challenges in modeling multi-tiered applications:

When it comes to HA-aware scheduling of applications in a cloud environment, the challenge lies in selecting the best deployment model while analyzing the impact of the adopted HA mechanism, different failure types, functionality constraints, the redundancy, and interdependency models between different components. For instance, suppose a typical multi-tiered application that consists of three-tiers with a front-end HTTP servers, a business logic application (App) on the middle tier, and a database (DB) storing the system state at the back-end. The HTTP servers depend on the App, which, in turn, is sponsored by the DB. Each component type consists of a primary component and multiple active/standby replicas. Each type is associated with certain failure types. When it comes to deploying such application in a single cloud with geographically distributed DCs, multiple options are to be considered on whether inter- or intra-DC deployment should be selected. It is not always the case that maximum inter-DC distribution is preferable because this decision depends on the above factors.

B. Modeling systems with Stochastic Petri Nets:

Petri Nets are widely used to model the behavior of different Discrete Event Systems (DES) [15]. They are graphically presented as directed graphs with two types of nodes: places and transitions. Deterministic Stochastic Petri Nets (DSPN) are one of the Petri Nets extensions for modeling the systems with stochastic and deterministic behavior [16]. Three transition types are defined in DSPN: immediate transitions that fire without any delay under a condition, timed transitions that fire after a deterministic delay, and stochastic transitions that fire after an exponentially distributed delay.

DSPN is formally 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,

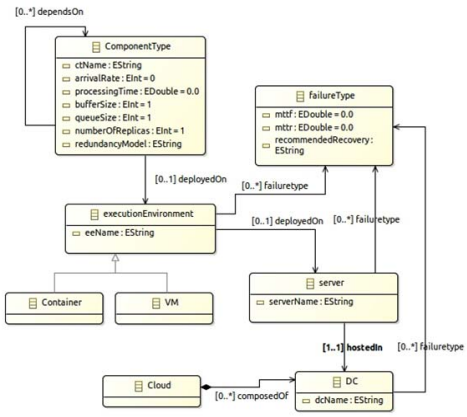

Fig. 1: Simplified UML model for a cloud deployment.

respectively. I and O are the forward and backward incidence functions such that $I, O: (P \times T) \cup (T \times P) \longrightarrow \mathbb{N}$ where $\mathbb{N}$ is the set of non-negative integers. 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 ($\tau : T \longrightarrow \mathbb{Q}^+$, where $\mathbb{Q}^+$ stands for the set of non-negative rational numbers). 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. To model the behavior of an application running on cloud with stochastic failures, we have used Stochastic Colored Petri Nets (SCPN), which is a class of DSPN models where the tokens can have different colors (types) [17]. SCPN can model both deterministic (recoveries) and stochastic (failures) cloud behaviors. It is simulated and analyzed using TimeNet [18]. In the following section, we explain the SCPN model proposed for a multi-tier application deployed in cloud.

III. Approach

A typical cloud deployment is composed of different software components running on an execution environment (e.g VM or container). The VM is hosted on a server, and the server in turn is hosted on a DC. Fig. 1 illustrates our simplified Unified Modeling Language (UML) model that captures such cloud deployment. 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*). Execution environment (VM), 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

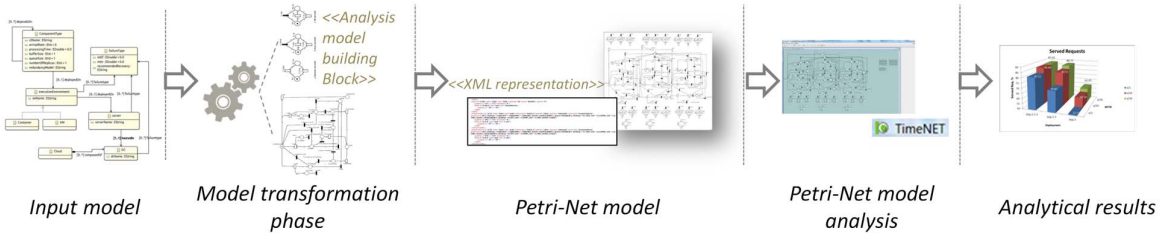

Fig. 2: Overall approach.

on the recommended recovery.

UML as a semi-formal model cannot simulate the behavior of the system or measure the availability of a service while different stochastic failures are happening. On the other hand, Stochastic Petri Nets are behavioral models that have proven to be suitable to model and simulate the behavior of such systems with stochastic behavior. However, building the SCPN model manually can be a fallible job; in order to alleviate this intricacy, we have proposed a one to one mapping approach. The latter is based on mapping an instance of the UML model describing a given deployment of the application in the cloud to the corresponding SCPNs model. The transformation approach builds a components dependency graph to identify the number of tiers and their orders. Next, it creates the places and transitions that are common in all the SCPN models and iterates over each tier creating the load balancer, all the component replicas, their VMs, and their corresponding servers. Then the DCs are created, the transitions are annotated with the proper rates, and the guards are annotated with the proper conditions. Fig. 2 summarizes this approach.

In the following, we explain a one to one mapping to achieve the transformation from the UML model of a cloud system to the corresponding SCPN model.

A. SCPN model building blocks

This section explains the SCPN model used to evaluate various HA-aware deployments of application components in a cloud environment. We define five different SCPN building blocks that we use in our model transformation phase. Each VM, server, and DC has its own failure rate (MTTF) and recovery time (MTTR). As in [1], [2], [19], and [20], the exponential failure distribution is used in this paper to reflect failure rates or MTTF of DCs, servers, and applications/VMs. As for the repair/recovery timed transitions, a deterministic distribution is applied on them to trigger any repair or recovery behavior. It should be noted that our approach also supports other failure rates as our model does not depend on a specific probability distribution. Figures provided in this paper follow the representation of TimeNet. In TimeNet, the immediate transitions are shown as black bars while deterministic and exponential timed transitions are shown as thick white-filled bars.

1) Data center model: Fig. 3a shows the DC model. A DC 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).

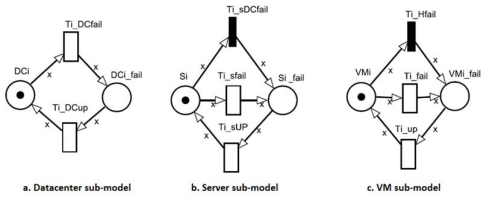

Fig. 3: Data center, server, and VM sub-models.

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 failure is modeled using 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:

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

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:

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

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 DC (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:

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

The recovery happens after a deterministic delay (T_i_up). Note that 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:

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

4) Load Balancer model: 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 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 ($C_i\_flushing$) to ensure a round robin distribution. 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 $C_1, C_2, C_3, ...C_M$ where M is the number of replicas, 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. 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, $T_2\_flush, T_3\_flush, ...T_M\_flush$ also fire. According to the nature of workload arrival of the system, T_LB_i can have different distributions (e.g. deterministic, exponential, ...).

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

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

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

$\forall_{i \in 1:M} G_{T\_LB_i} =$

$(C_i\_flushing == 0 \wedge VSD_{H(i)} \wedge C_i\_queue < L)$

$\bigwedge_{k=1:i-1} (C_k\_flushing == 1 \vee VSD_{F(k)})$

$\bigwedge_{j=i+1:M} (C_j\_flushing == 0 \vee VSD_{F(j)})$

And T_i_flush is guarded with G_{Ti_flush}:

$\forall_{i \in 1:M} G_{T_i\_flush} = \bigwedge_{j=1:i-1} (C_j\_flushing == 0 \vee VSD_{F(j)})$

$\bigwedge_{k=i+1:M} (C_k\_flushing == 1 \vee VSD_{F(k)})$

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:

$G_{T\_LB_0} =$

$\bigwedge_{i=1:M} (VM_i == 0 \vee VM_{(i)Server} == 0 \vee VM_{(i)DC} == 0 \vee C_i\_queue \geq L)$

5) 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) and 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:

$G_{T_i\_Lost\_in\_Processing} =$

$VM_{(i)} == 0 \vee VM_{(i)Server} == 0 \vee VM_{(i)DC} == 0$

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:

$G_{T_i\_Lost} =$

$\bigwedge_{i=1:M} (VM_{(i)} == 0 \vee VM_{(i)Server} == 0 \vee VM_{(i)DC} == 0)$

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 redundant components. The transition $T\_failover\_C_i\_to\_LB$ is guarded with:

$\forall_{i \in 1:M} G_{T\_failover\_C_i\_to\_LB} =$

$(VM_{(i)} == 0 \vee VM_{(i)Server} == 0 \vee VM_{(i)DC} == 0) \wedge$

$\bigvee_{j=1:M | j \neq i} (VM_{(j)} == 1 \wedge VM_{(j)Server} == 1 \wedge VM_{(j)DC} == 1)$

The tokens are successfully processed through deterministic transition ($T_i\_processed$). The latter tokens are then 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.

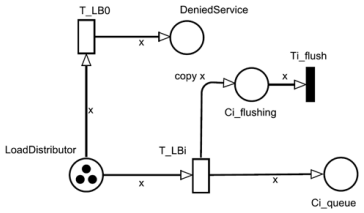

Fig. 4: Load balancer model.

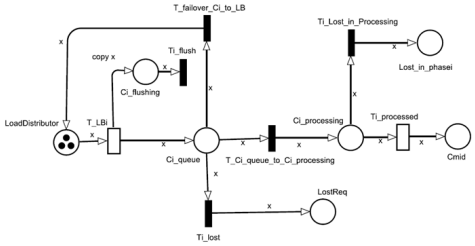

Fig. 5: Component model.

IV. Case Study

This section provides an example of a cloud deployment modeled by SCPN, and then the model is used to evaluate different deployments from HA perspective. The system under study is a three-tier application, mainly Big Data analysis application. At the front end, the *Filters* receive unstructured

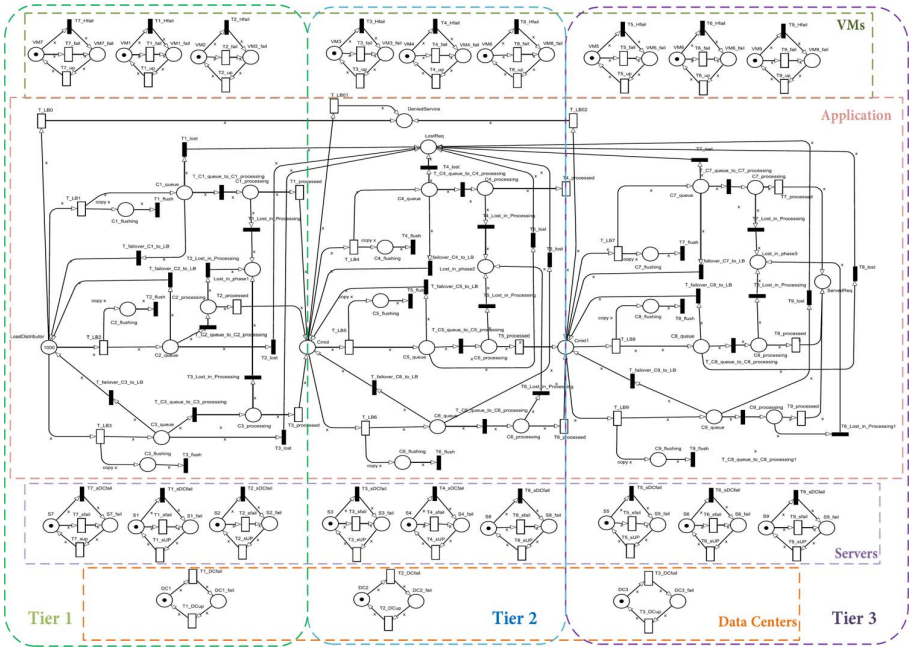

Fig. 6: SCPN model of a three-tier application running in a cloud environment.

TABLE I: Different MTTF, MTTR, and processing time

MTTF(DC_1; DC_2; DC_3)	x*;x;x	x;1.5x;2x	x;2x;3x
MTTR	x/3	x/10	x/30

*: x is the failure rate of DC_1.

data and remove redundant/useless ones. In the middle, the *Analysis Engines* analyze the data and generate structured data form. At the back end, the *Databases* store the structured data produced by the Analysis Engine. In this case study, we are particularly interested in comparing inter and intra-DC scheduling, and we change alternatively the DC hosting the servers and VMs. Fig. 6 illustrates a snapshot of the SCPN model of this system. 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; therefore we have fixed the number of tokens in the initial *LoadDistributor* place. The percentage of the requests that are successfully processed through the three tiers (*ServedReq*) indicates the service availability of the cloud application.

A. Evaluation and results

To investigate different DC scheduling, we have considered multiple scenarios and conducted some experiments with the SCPN model. Our main focus in this paper is the effect of DC failures on HA. The failure rates of VMs and servers (used in T_i_fail and T_i_sfail) are fixed throughout these experiments. We consider that DCs can have similar or different failure rates. As a base line, they all have the same MTTF (x; x; x). Then we modify the failure rate of the DCs assuming that DC_1 fails more frequently, DC_3 is always the most reliable

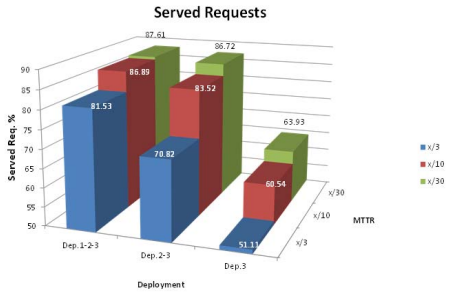

Fig. 7: Service availability of different deployments and different MTTRs. DCs have similar MTTF.

one, and DC_2 has a failure rate between the two others. Then we consider different MTTR for each variation of the MTTF. Table I shows different parameters altered in our experiments. We have considered three deployments: 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 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 afterwards). We aim to evaluate which of the three deployments would maximize the availability of the application. The model presented in Fig. 6 is analyzed with transient simulation of TimeNet4.2 running on a VM in a private cloud with 225GB of RAM and 20 vCPUs running Ubuntu12.04. The results presented in this paper 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 as

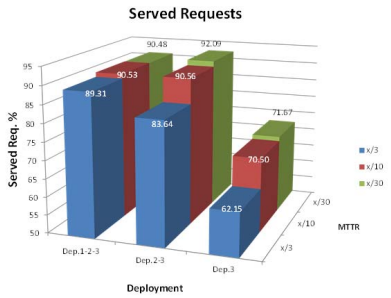

Fig. 8: Service availability of different deployments and different MTTRs. DCs have different MTTF (x; 1.5x; 2x).

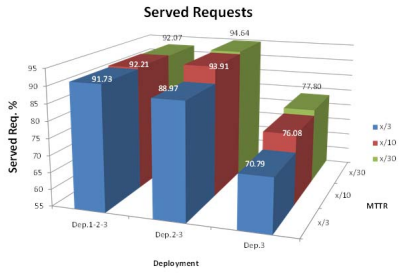

Fig. 9: Service availability of different deployments and different MTTRs. DCs have different MTTF (x; 2x; 3x).

presented in Table I. Fig. 7 depicts the corresponding results for the three deployments mentioned above. When the DCs have the same failure rates, 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 failure rates of DC_1, DC_2, and DC_3 to x, 1.5x, and 2x, respectively and change the recovery time as listed in Table I. Fig. 8 presents the results. Finally, we consider the case where DCs have different MTTF of x, 2x, and 3x, respectively. Again, we vary the MTTR according to Table I. The results are presented in Fig. 9. Based on the results of Fig. 8 and Fig. 9, 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.

The approach presented in this paper allows verifying which scheduling options among different placement possibilities can meet the required level of availability.

V. Conclusion

Cloud services experience various stochastic failures and consequently become unavailable. With the always on and always available trend, inoperative services halt the business continuity. It is not enough to provide a HA-aware 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. The paper proposed a SCPN model that evaluates the availability of cloud services and their deployments in inter- or intra-DCs. This model considers different failure types, functionality constraints, redundancy models, and interdependencies between different application components. Consequently, different decisions had been extracted from this model that aid in designing the best HA solution of an existing cloud model.

Acknowledgment

This work is supported by the Natural Sciences and Engineering Research Council of Canada and Ericsson Research. We would like to thank Prof. Armin Zimmermann for his technical insights on TimeNet.

References

[1] M. Jammal, A. Kanso, and A. Shami, "High Availability-Aware Optimization Digest for Applications Deployment in Cloud," *IEEE International Conference on Communications (ICC),* pp.6822-6828, June 2015.

[2] M. Jammal, A. Kanso, and A. Shami, "CHASE: Component High-Availability Scheduler in Cloud Computing Environment," *IEEE International Conference on Cloud Computing (CLOUD),* pp. 477-484, 2015.

[3] P. Yong and H. Ning,"Research on dependability of cloud computing systems," *International Conference on Reliability, Maintainability, and Safety (ICRMS),* pp. 435-439, August 2014.

[4] NETFLIX, "AWS Re:Invent - High Availability Architecture at Netflix," http://www.slideshare.net/adrianco/high-availability-architecture-at-netflix, December 2012. [August 20, 2015]

[5] D. S. Kim, F. Machida, and K. S. Trivedi, "Availability modeling and analysis of a virtualized system," *15th IEEE Pacific Rim International Symposium on Dependable Computing,* pp. 365-371, November 2009.

[6] W. E. Smith, K. S. Trivedi, L. A. Tomek, and J. Ackaret, "Availability analysis of blade server systems," *IBM Systems Journal,* vol. 47, no. 4, pp. 621-640, 2008.

[7] F. Longo, R. Ghosh, V. Naik, and K. Trivedi, "A scalable availability model for infrastructure-as-a-service cloud," *IEEE/IFIP International Conference on Dependable Systems & Networks,* pp. 335-346, June 2011.

[8] B. Javadi, D. Kondo, J. Vincent, and D. Anderson, "Discovering statistical models of availability in large distributed systems: An empirical study of seti@home," *IEEE Transactions on Parallel and Distributed Systems,* vol. 22, no. 11, pp. 1896-1903, November 2011.

[9] F. Salfner and K. Wolter, "Analysis of service availability for time-triggered rejuvenation policies," *Journal of Systems and Software,* vol. 83, no. 9, pp. 1579-1590, May 2010.

[10] F. Salfner and K. Wolter, "Service Availability of Systems with Failure Prevention," *IEEE Asia-Pacific Services Computing Conference,* pp. 1219-1224, December 2008.

[11] F. Salfner and K. Wolter, "A Petri Net model for Service Availability in Redundant Computing Systems," *Winter Simulation Conference (WSC),* pp. 819-826, December 2009.

[12] S. Loveland, E. M. Dow, F. LeFevre, D. Beyer, and P. F. Chan, "Leveraging virtualization to optimize high-availability system configurations," *IBM Systems Journal,* vol. 47, no. 4, pp. 591-604, 2008.

[13] B. Cully, G. Lefebvre, et al., "Remus: high availability via asynchronous virtual machine replication," *5th USENIX Symposium on Networked Systems Design and Implementation,* pp. 161-174, 2008.

[14] E. M. Farr, R. E. Harper, L. F. Spainhower, and J. Xenidis, "A case for High Availability in a virtualized environment (HAVEN)," *Third International Conference on Availability, Reliability and Security,* pp. 675-682, March 2008.

[15] C. Petri, "Kommunication mit Automaten," University of Bonn, 1962.

[16] G. Ciardo and C. Lindemann, "Analysis of deterministic and stochastic Petri nets," *5th International Workshop on Petri Nets and Performance Models,* pp. 160-169, 1993.

[17] N. Gharbia, C. Dutheilletb, and M. Ioualalen, "Colored Stochastic Petri Nets for modelling and analysis of multiclass retrial systems," *Math. Comput. Model.,* vol. 49, pp. 1436-1448, 2009.

[18] A. Zimmermann, "Modeling and Evaluation of Stochastic Petri Nets with TimeNET 4.1," *6th International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS),* pp. 54-63, 2012.

[19] F. Machida, D. S. Kim, and K. S. Trivedi, "Modeling and analysis of software rejuvenation in a server virtualized system," *IEEE Second International Workshop on Software Aging and Rejuvenation,* pp. 1-6, November 2010

[20] J. Xu, X. Li, Y. Zhong, and H. Zhang, "Availability modeling and analysis of a single-server virtualized system with rejuvenation," *Journal of Software,* vol. 9, no. 1, pp. 129-139, January 2014.