Availability Analysis for Deployment of In-Cloud Applications

Xiwei Xu, Qinghua Lu, Liming Zhu, Jim(Zhanwen) Li, Sherif Sakr, Hiroshi Wada, Ingo Weber
NICTA, Sydney, Australia
School of Computer Science and Engineering, University of New South Wales, Sydney, Australia
firstname.lastname@nicta.com

ABSTRACT

Deploying critical applications in the cloud introduces uncertainties for availability that have traditionally been under the direct control of the application owner. The cloud infrastructure impact to availability is due to dynamic resource sharing as well as limited visibility/control of the underlying infrastructure and its quality of service. It is important to assess the availability of the critical application considering the weak availability guarantees provided by the cloud infrastructures under a broad range of scenarios, including rare scenarios like infrastructure failures and disasters. In this paper, we propose a deployment architecture-driven availability analysis model that considers uncertain rare events explicitly and bridges the gap of weak infrastructure availability and critical application availability. The models require initial calibration and validation, which is achieved by using data from commercial products and industry best practices. We use the proposed models to reevaluate the industry best practice under rare infrastructure events.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification – Formal methods.

General Terms

Management, Measurement, Reliability.

Keywords

Deployment architecture; Cloud computing; Availability

1. INTRODUCTION

Deploying critical applications in the cloud introduces additional uncertainties [1] to the strong availability guarantee for a number of reasons. First, shared resources and multi-tenancy require both cloud providers and consumers to have new ways of understanding and isolating uncertain behavior of co-located entities [2]. Second, the granularity and assumptions of the availability levels guaranteed by cloud infrastructure providers vary widely and are often weak. For example, their use of aggregated metrics on larger units (e.g. whole data center rather than individual instances) makes assessing application availability difficult [3]. Third, to deal with uncertainty, one must explicitly consider rare but high consequence events such as disasters, especially for critical applications. Large-scale cloud failures do happen and at the moment cloud users heavily rely on rule-of-thumb practices of over-provisioning to achieve availability [4] rather than more informed analysis and guarantees. In cloud world, over-provisioning also directly translates to extra monetary costs.

In this work, we focus on the analysis of deployment architecture and infrastructure failures rather than internal software architecture and component failures because how to deploy existing critical applications in relatively unreliable cloud is the major concern. In general, in-cloud deployment architectures can contain both stateless components (e.g. web servers) and stateful components (e.g. databases) deployed on Virtual Machines (VM). The stateless components can scale-out/in relatively quickly, to react to workload changes or infrastructure failures. The best practice for stateful components is to have hot standbys in other zones of the same region, where failover is relatively quick. The stateful components can scale out, but less quickly, by adding consistent replicas. However, major disasters affecting whole regions may require stateful components to be recovered from backups, which may take longer depending on the technology involved. Recovery-Point-Objective (RPO), data consistency Recovery-Time-Objective (RTO) and transaction consistency RTO [5] are the main factors that constrain the selection of backup and recovery strategies during a disaster.

In this paper, we propose an approach for application availability analysis using deployment architecture from the perspective of cloud consumers. We build availability analysis models as Stochastic Reward Nets (SRNs) [6] for a typical in-cloud application deployment architecture. SRN-based models are particularly suitable for analyzing deployment architectures of concurrent application at a high-level. The usage of SRN for modelling and analysing availability in different contexts has been investigated in the last few decades. We apply SRN in the context of deploying applications in cloud. We overcome a number of unique challenges dealing with real world deployment and analysis.

First, we use the actual SLA guarantees and available options from major cloud infrastructure providers as inputs to our availability models. The models bridge the gap between weak infrastructure availability and critical application availability. Thus, users can evaluate their deployment decisions, scaling policies and recovery plans in a realistic context.

ISARCS'13, June 17–21, 2013, Vancouver, BC, Canada.

Second, the failover and recovery scenarios and their modeling are based on actual features offered by commercial products. For example, if an availability zone fails, Amazon EC2's auto-scaling mechanism is used to balance the workload over the available instances to minimize the effect of the failing instances. A cross-region recovery scenario is based on a commercial disaster recovery product – Yuruware Bolt[1].

Third, we explicitly distinguish the time interval of transitions between states of VMs in addition to typical failure/workload-caused triggering rates of the transitions. The transition time intervals are critical for realistically reflecting the characteristics of application and replication technologies regarding auto-scaling, regeneration or stateful component recovery.

Our availability models capture the three-tier in-cloud applications deployment architecture where infrastructure failure is a major concern. Our analytic models consider an overall application as available only if a configurable number of data/transaction-consistent stateful and stateless VMs are operative. Analysis of different scenarios can be achieved through manipulation of parameters in the models.

We use the proposed models to reevaluate industry best practices under rare events and various recovery scenarios so their effectiveness on critical applications can be quantitatively understood. The availability models can also predict and compare application-level availability of different deployment options under different scenarios. The predictions of our models are meaningful for scenarios where the relative availability values are useful to practitioners. Absolute availability values depend on application-specific measurement and workload assumptions. We did measurements on launching different VMs hosting stateless components in Amazon EC2 and used typical RTO data from Yuruware Bolt for improving our models.

The rest of this paper is organized as follows. Section 2 covers related work. Section 3 introduces the overview of our availability analysis approach. Section 4 gives a deployment of a typical in-cloud application. Section 5 discusses details of the availability models. Section 6 presents our initial results, and Section 7 concludes the paper, outlining future directions.

2. RELATED WORK

There has been a lot of work focusing on infrastructure availability analysis in terms of placing VMs onto physical machines assuming certain failures in the physical machines [7]. Dynamic application placement [8] also helps cloud and infrastructure providers to optimize application deployment in terms of reducing allocated resources while maintaining high availability guarantees for critical applications. Our approach uses similar types of analytical models, such as SRN, for modeling availability but works at the application level with the actual deployment options (e.g. auto-scaling and disaster recovery) available to consumers.

At the application deployment level, approaches like [9] were proposed to optimize reliability, latency and energy when application components are deployed onto physical machines. However, the deployment platform involves physical machines where one has full control/visibility rather than infrastructures with specific scaling facilities and failures ranging from individual nodes to entire region. Furthermore, the reliability model in [9] seems simplistic, since it only considers communication frequency and network reliability.

Other efforts focus on how to convert SysML-based application architectures into availability analysis models [10] which is costly and requires expertise in complicated analytical models. Our analysis models are more generalized and applicable to typical in-cloud deployments involving stateful and stateless components. Different applications and deployment decisions usually only require different parameters settings rather than complete re-modeling.

3. AVAILABILITY ANALYSIS OVERVIEW

The core of our analysis approach comprises four analytic sub-models, namely the architectural model, maintenance model, backup model, and the recovery model. Our analytic models are based on SRN [6]. In this paper, we only show the architecture model and the recovery model due to space limitations. Live backups/migrations (especially on stateful components) and maintenance (such as rolling upgrades) can also have a major impact on performance and availability due to the necessity of temporarily stopping the application even under an extreme short period of time. They are out of the scope of the initial models, and subject to ongoing work.

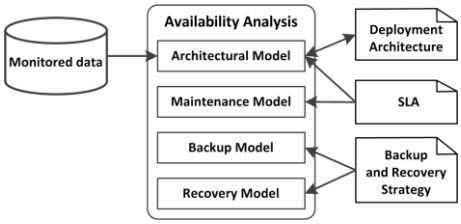

Figure 1. Overview of our availability analysis

As shown in Fig.1, deployment decisions and patterns (best practice) such as scaling policies (CPU threshold) and multi-zone configurations are usually reflected in the availability models as parameters. We consider that the overall application is available only if a certain number of stateful/stateless VMs are available under certain workload. The SLAs provided by the infrastructure provide a direct input to the architecture analysis model. Application-specific monitoring data (live or historic), such as actual VM startup time and replication techniques, is required by the architecture model as well. The recovery model reflects the impact of different recovery strategy such as auto-regeneration of stateless components, or (slower) recovery of stateful components from backup images.

Overall, our approach follows the suggested ways of dealing with uncertainty in software engineering [11]. Particularly, rare uncertain events such as region/zone failures and disaster recovery that are important to critical applications are explicitly represented in our models, so that they can well represent the fault-tolerance design challenges. Rather than relying on statistical average-based application availability models [9] which provide insufficient guarantees for critical applications, we use SRN-based analysis to combine an exponential distribution assumption of some events with application-specific data.

[1] Yuruware Bolt— http://www.yuruware.com/

4. IN-CLOUD DEPLOYMENT

Fig. 2 illustrates deployment architecture of a typical in-cloud application. The in-cloud application contains stateless components and stateful components that are deployed in separate VMs. The VMs are then placed into different availability zones, assisted by a load balancer. Availability zones are distinct geographic locations that are designed to isolate failures occurred in other availability zones. Regions, containing one or more availability zones, are geographically distributed in different countries or separate geographic areas. The deployment of our example application crosses two availability zones, namely, Zone A.1 and Zone A.2.

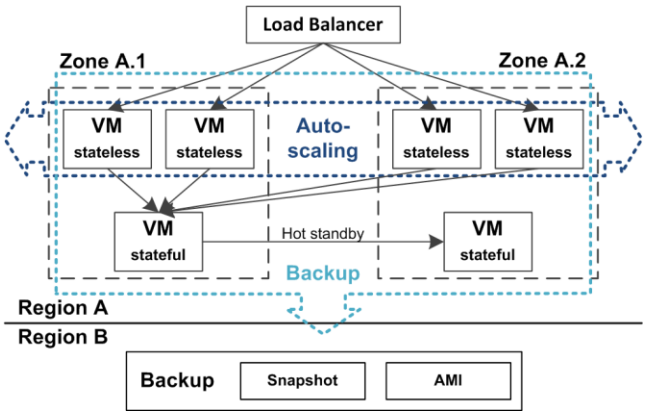

Figure 2. Typical in-cloud application deployment

Stateless VMs are deployed into auto-scaling groups, where the number of running VMs can be automatically adjusted according to pre-defined policies such as resource unitization. The auto-scaling mechanism ensures that the capability of the deployed application increases and decreases seamlessly depending on varying workload. The auto-scaling mechanism could also minimize the impact of the instance failures.

Stateful components (such as databases) can have hot standbys inside a region, which supports quick failover management in case of a failure, and establish new replicas for serving increasing workload.

Disaster recovery is a process that aims to achieve the continuation of critical applications after a large-scale technical problem, natural disaster or human-induced disaster [5]. A typical approach for achieving this goal is to duplicate the entire deployment infrastructure to ensure spare capacity in the event of a disaster [12]. As shown in Fig. 2, the application running in Region A manages a backup in Region B so that if an entire region goes down, disaster recovery could be initiated in the other region from the backup. The backup files could be stored on Amazon S3, a highly durable and cost-effective data store [13]. To shorten the launching time of VMs for recovery, the backup of the stateless components could also be AMIs (Amazon Machine Images) rather than through VM snapshots, recreated from the backups in Region B where the degree of data and transaction consistency and recovery time depends on the chosen backup strategy.

5. AVAILABILITY ANALYSIS MODELS

We build our availability models using Stochastic Reward Nets (SRNs), which provide a graphical notation for the formal description of dynamic behaviour of systems. SRN is a variant of Stochastic Petri Net (SPN) – a high-level Petri-net in which random firing delays are associated with transitions whose firing is an atomic operation [14]. SRN extends SPN to allow specifying reward structures [6]. Below are the main SRN constructs, please refer to Fig. 3 for the corresponding notations. Intuitively, we model one availability zone in one SRN, which is represented in the rectangle with dashed line.The two smaller SRNs (right hand side) represent the recovery scenario when one zone goes down.

- *Place:* The large circles in SRN are called places, which represent the different conditions that hold in system.
- *Token:* Tokens are the filled dots inside places. The corresponding condition of a place holds if one or more tokens are inside the place.
- *Transition:* The unfilled rectangles between places are called transitions, which represent the events occurring in the system. Firing a transition could cause the reallocation of the tokens.
- *Guard function:* A transition may be associated with a guard function. The transition is enabled if (i) all input places have enough tokens (depending on a so-called arc function, which may require 0 tokens), and (ii) the guard function returns *true* (or 1). The default guard is the constant function *true*.
- *Transition rate:* The events associated with the transitions take a period of time to happen after the transition is enabled. The rate of the time delay is called transition rate, which could represent the frequency of the transition or represent the delay before the transition fires.
- *Reward function:* The output measures are expressed in terms of expected values of reward rate functions, which assign appropriate reward rates to the states of the SRN. It allows the computation of various quantitative measures. The reward rates are assigned based on the output measure of interest.

We apply SRN in context of in-cloud deployment architecture. Our models capture the essential characteristics of critical application concerns and fault-tolerance techniques such as using availability zones, auto-scaling mechanisms, and disaster recovery strategies. We use SPNP [15] to edit our model and calculate the availability.

5.1 Availability Zones

Inside each zone, we model states of a VM as places. A VM is represented by a token. The traditional ways of modelling availability only consider up and down states of components [10]. In our analytic models, we explicitly distinguish two states representing a VM up, namely *Running* and *Full*, and two states representing a VM down (not working), namely, *Stopped* and *Failed*. The distinction is important to reflect the unique features of in-cloud application deployment. The applications are allowed to be deployed onto maximum *N* VMs, as indicated by *Stopped* state. A VM is in *Running* state when it is serving workload.

Switching between states of a VM is modeled by transition. The corresponding transition rate can represent node failures rates

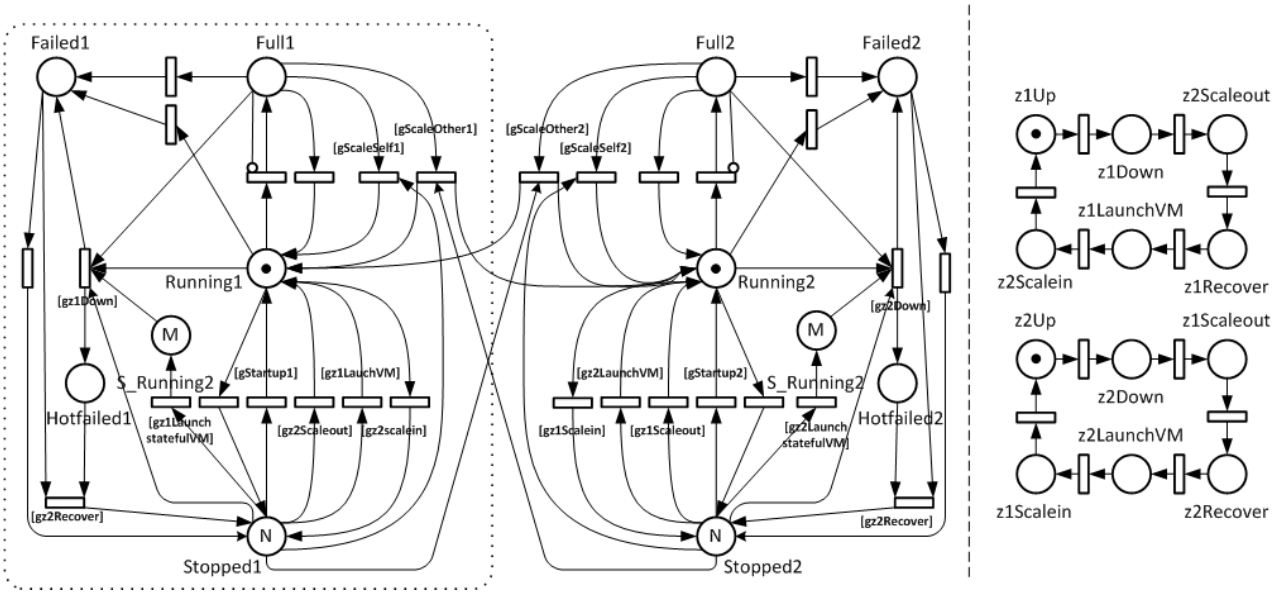

Figure 3. SRN models for deployment of cloud-based application

(frequency) or the time taken to launch a VM image (delay before the transition fires). The software failure and request arrival intervals (different from the intensity/pattern of the workload itself) follow exponential distribution – a common assumption in such models [7, 10].

Conditions of switching states are modelled by the guard functions associated with transitions. In the model shown in Fig. 3, corresponding guard functions of transition are annotated in square brackets. Some example guard functions are shown in Table 1. The place name with a number sign (#) in front represents the number of tokens in the place.

When the deployment of an application involves two availability zones in a single region, two SRNs are used to represent the two availability zones (left hand side). The two SRNs interact through the transitions connecting the two availability zones, which represent the auto-scaling mechanism.

Table 1. Guard functions and reward function

Function	Definition
gz1Down	if (#z1Down==1) 1 else 0
gz2Recover	if(#z2Recover==1) 1 else 0
gScaleSelf1	if(#Running1<=#Running2 && #Stopped1>0) 1 else 0
gScaleOther1	if(#Running2<#Running1 && #Stopped2>0) 1 else 0
gStartup1	if((#Running1==0) 1 else 0
gz2Scaleout	if(((#z2Scaleout==1) 1 else 0
gz2Scalein	if(((#z2Scalein==1) 1 else 0
gz1LaunchVM	if(((#z1LaunchVM==1) 1 else 0
gz1Launch statefulVM	
Reward Function	if((#Running1>0 &&#S_Running1>0) ‖ (#Running2>0 &&#S_Running2>0)) 1 else 0

5.2 Stateless VMs with Auto-scaling

Full state is introduced to facilitate the modelling of auto-scaling mechanism. If a running VM reaches a predefined CPU threshold, it is then temporarily transits into *Full* state. This may trigger a transition for adding one more VM, according to a preconfigured auto-scaling strategy to handle the increasing workload. For the deployment with two availability zones, the auto-scaling mechanism could launch new VMs inside a single zone or across zones (within one region) according to the associated policies.

The auto-scaling policies with load balancing are modeled by four guard functions, namely, *gScaleSelf1*, *gScaleOther1*, *gScaleSelf2* and *gScaleOther2*, as shown in Table 1. Definitions of the latter two functions are omitted due to length limitation. Whether to trigger *gScaleSelf1* or *gScaleOther1* is dependent on the load balancing strategy. If the number of running VMs in Zone A.1 is not greater than the number of running VMs in Zone A.2, and there are spare VMs in the *Stopped1* place, the guard function *gScaleSelf1* is satisfied, which fires the corresponding transition and moves one VM from *Stopped1* to *Running1*. Otherwise, a VM is moved from *Stopped2* to *Running*1. *gScaleSelf2* and *gScaleOther2* are defined in a similar way.

5.3 Stateful VM

Scaling and failover of stateful VMs are more complicated than stateless VMs. In this paper, we assume the number of stateful VMs is defined at design time, and fixed at runtime without scaling (as indicated by M). In our future work, mechanism of scaling stateful VM should be considered in our model, like live migration, which could be used for achieving scalability, elasticity and effective dynamic provisioning goals through enabling the creation of new consistent serving replicas as necessary during runtime.

For stateful components, it takes longer time to transit from *Stopped* to *S_Running* to achieve consistency. Thus, transition rates of the transition between *Stopped* and *Running* and the

transition between *Stopped* and *S_Running* are different. A *Failed* VM hosting stateful components will trigger its hot standbys for failover purposes.

5.4 Disaster Recovery

As mentioned earlier, the two smaller SRNs represent the recovery scenario. Due to space limitation, we cannot show the SRN model for the "region down" scenario. The assumption is that once all zones in one region go down, a relatively slower recovery process, e.g., using a commercial product, is triggered in another region where other availability zones can be used. The smaller recovery SRN model and the corresponding bigger VM SRN model also interact through the guard functions associated with transitions. The VMs hosting the application will become *Failed* once the availability zone supporting them is down. Thus, the function *gz1Down* checks the number of tokens in the place *z1Down*. When *z1Down* has one token, the corresponding transition is enabled and consumes all tokens at places *Full1, Running1 and S_Running1* representing full stateless VM and running stateless and stateful VMs, and generates number of tokens in place *Failed1* and *HotFailed1*.Similarly, once the place *z1Up* has one token, the corresponding transition is enabled and consumes all tokens at places *Failed1* and *HotFailed1* representing failed stateless and stateful VMs, and generates the same number of tokens in place *Stopped1*, which means that Zone A.1 is recovered and VMs are ready to be launched.

The recovery of stateless VMs is based on backup. For stateful VMs, there are three optional ways to recover from disaster: backup, replica, and hot standby. All of these three strategies need some time to achieve data consistency and transaction consistency respectively [5]. The backup strategy regularly takes backups of the application, which are stored in a different region other than the region that the application resides in. When a disaster occurs, the application can be restored from the backup stored at the other region, which takes some time. In the future, we will enable the model to express the recovery process under different recovery strategies.

5.5 Availability Reward Function

Availability is the measurement of a system's uptime over a period of time [16]. We consider that the overall application is available only if at least one stateless VM and one stateful VM hosting a master database are available with consistent state. In our model, the reward function is assigned based on availability. As shown in the last line of Table 1, our reward function uses 1 to represent available, and uses 0 to represent unavailable.

6. INITIAL RESULTS

We evaluated our models by assessing the effectiveness of recommended deployment options and industry best practices for different types of scenarios. The inputs of the analytic model come from the SLAs provided by the infrastructure, application-specific monitoring data (live or historic), such as actual VM startup time and replication techniques.

Some cloud providers, such as Rackspace, provide an instance-level performance agreement, which can be used directly as assumed failure rates on nodes. Amazon EC2 only provides availability agreements on availability zones. We use the Amazon EC2 SLA commitment of 99.95% availability in our experiments.

6.1 Availability of Multi-zone Deployment

One type of recommendation from Amazon EC2 is to deploy applications in different availability zones. We use our model to measure the availability of a cloud-based application deployed on a single-zone and on multi-zone deployments. Table 2 gives values of parameters, key transition rates, and definition of the reward function for the model described in the previous section. In particular, we assume an availability zone goes down once per year according to Amazon EC2's availability guarantee. The zone fail rate is 0.00011 [10], and the mean time to repair (MTTR) is 4.38 hours per year. In this experiment, we also assume the two zones do not go down at the same time.

Table 2. Parameters

Parameter	Value
N	4
M	2
Zone failure rate	0.00011 (1/h)
Zone recovery rate	0.2283 (1/h)

Table 3 shows the availability of 1-zone deployment and 2-zone deployment under different scaling-out and over provisioning policies (see next section). For all thresholds, the availability of two zones is higher than the availability of one zone, which reflects conventional wisdom. However, the predicted differences between the two are useful for a quick what-if analysis when optimizing availability.

Table 3. 1-zone vs. 2-zone deployment under different scaling-out policy

CPU Threshold	1-zone	2-zone
30%	99.05%	99.45%
40%	99.03%	99.29%
50%	98.97%	99.24%
60%	98.85%	99.21%
70%	98.71%	99.17%
80%	98.54%	99.16%
90%	98.38%	99.14%

6.2 Auto-recovery when One Availability Zone Down

Amazon EC2 suffered from a number of outages in the past [4]. To sustain the required availability under such events, industry best practice [4] recommends live replication across multiple availability zones and a certain degree of over-provisioning to handle the load spike after a massive failure. Netflix summarized their strategy as "*Deploy in multiple AZ with no extra instances – target autoscale 30-60% until you have 50% headroom for load spikes. Lose an AZ leads to 90% utilization*". However, it is difficult to know whether these rule-of-thumb suggestions can satisfy the requirements of critical applications. We evaluate these options quantitatively. Using the same parameters as above, we now assume one availability zone is entirely down. We compare the over-provisioning (30-60% CPU threshold) deployment with typical overload configuration (70-90% CPU threshold). In Table 3, the results show that the overall availability depends on the degree of the over-provisioning. The more headroom each VM has, the higher overall availability is.

This validates our model, which further provides some quantitative and informed understanding, allowing what-if analyses.

6.3 Disaster Recovery when One Region Down

For critical applications, an organization needs to prepare for rare but high consequence events. Our third scenario assumes an entire region is down (once a year) and no load-balancing-based recovery is available. There are commercial products that support such disaster recovery situations. Yuruware Bolt is one such product which allows a low cost snapshot-based backup and recovery.

Table 4. Availability with disaster recovery mechanisms

RTO	RPO	Availability	Cost
2 hours	10 minutes	99.67%	Low
5 minutes (Yuruware Bolt)	3 hours	99.68%	
5s	5 s	99.88%	High

Assuming the same deployment architecture and parameters as the previous scenarios, Table 4 shows an evaluation of three RTO/RPO pairs with *data-consistent* RTO (used in transition times of launching a *Stopped* VM to data-consistent *Running* state), based on Yuruware Bolt data and other alternatives. There is usually a trade-off between RPO and RTO under low cost. A high-cost solution, such as hot standbys across regions using fiber optics communication, is presented in the last row. For rare events, like *region down*, we enable in-cloud application providers to assess the availability they want for their critical applications under different assumptions.

7. CONCLUSION AND FUTURE WORK

We presented deployment architecture-driven availability analysis models for in-cloud applications. Our models capture the essential characteristics of in-cloud critical application concerns and realistically available reaction options. We explicitly consider high consequence events and fault-tolerance designs for those events. We calibrated the models using Amazon EC2 guarantees and data from a commercial disaster recovery product, Yuruware Bolt. We reevaluated existing industry best practices under high consequence events as the initial evaluation.

One of our ongoing works is generalizing the current model of availability zones into interacting sub-models, each of which represents one availability zone separately. The interacting SRN sub-models approach is more scalable compared to single model approach.

For future work, we will systematically measure the scalability of our models and collect more empirical data to validate the model, while building separate backup and maintenance models to cover even more real-world scenarios. We also consider the degree of state consistency of stateful VMs, and different backup and replication strategies.

8. ACKNOWLEDGMENTS

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

9. REFERENCES

[1] F. Faniyi, *et al.*, "Evaluating Secruity Properties of Architectures in Unpredictable Environments: A Case for Cloud," 9th Working IEEE/IFIP Conference on Software Architecture (WICSA2011), Boulder, Colorado, US, 127-136, 2011.

[2] S. Chaudhuri, "What next?: a half-dozen data management research goals for big data and the cloud," 31st symposium on Principles of Database Systems Scottsdale (PODS2012), AZ, USA, 1-4, 2012.

[3] S. A. Baset, "Cloud SLAs: Present and Future," *Operating Systems Review*, vol. 46, pp. 57-66, 2012.

[4] Tohorsten, "Amazon EC2 outage: summary and lessons learned," in *RightScale Blog*,2011..

[5] M. Thordal, *et al.*, *Using the SVC for Business Continuity*, 2007.

[6] J. K. Muppala, *et al.*, "Stochastic reward nets for reliability prediction," *SAE International Journal of Communications in Reliability, Maintainability and Serviceability*, vol. 1, pp. 9-20, 1994.

[7] F. Longo, *et al.*, "A scalable availability model for Infrastructure-as-a-Service cloud," 41st International Conference on Dependable System & Networks (DSN2011), Hong Kong, China, 335-346, 2011.

[8] C. Tang, *et al.*, "A Scalable Application Placement Controller for Enterprise Data Centers," 16th international conference on World Wide Web (WWW2007), Banff, Alberta, Canada, 331-340, 2007.

[9] S. Malek, *et al.*, "An Extensible Framework for Improving a Distributed Software System's Deployment Architecture " *IEEE Transactions on Software Engineering*, vol. 38, pp. 73-100, 2012.

[10] F. Machida, *et al.*, "Candy: Component-based Availability Modeling Framework for Cloud Service Management Using SysML," 30th IEEE International Symposium on Reliable Distributed Systems (SRDS2011), Madrid, Spain, 209-218, 2011.

[11] D. Garlan, "Software Engineering in an Uncertain World," FSE/SDP workshop on Future of software engineering research, New Mexico, USA, 125-128, 2012.

[12] AWS. *AWS Reference Architectures*, 2012, Available: http://aws.amazon.com/architecture/

[13] AWS..*Amazon Simple Storage Service (Amazon S3)*, 2012, Available: http://aws.amazon.com/s3/

[14] M. A. Marsan, "Stochastic Petri Nets: an Elementary Introduction," in *Advances in Petri nets*, Springer-Verlag, 1989, pp. 1-29.

[15] G. Ciardo, *et al.*, "SPNP: Stochastic Petri Net Package," 3rd International Workshop on Petri Nets and Performance Models, Los Alamitos, CA, 1989.

[16] L. Bass, *et al.*, *Software architecture in practice*: Addison-Wesley Professional, 2003.