Incorporating Uncertainty into in-Cloud Application Deployment Decisions for Availability

Qinghua Lu\textsuperscript{1,2}, Xiwei Xu\textsuperscript{1}, Liming Zhu\textsuperscript{1,2}, Len Bass\textsuperscript{1,2}, Zhanwen Li\textsuperscript{1}, Sherif Sakr\textsuperscript{1,2}, Paul L. Bannerman\textsuperscript{1,2}, Anna Liu\textsuperscript{1}
\textsuperscript{1}Software Systems Research Group, NICTA, Sydney, Australia
\textsuperscript{2}School of Computer Science and Engineering, University of New South Wales, Sydney, Australia
\{firstname.lastname\}@nicta.com.au

Abstract— Cloud consumers have a variety of deployment related techniques, such as auto-scaling policies and recovery strategies, for dealing with the uncertainties in the cloud. Uncertainties can be characterized as stochastic (such as failures, disasters, and workload spikes) and subjective (such as choice among various deployment options). Cloud consumers must consider both stochastic and subjective uncertainties. Analytic support for consumers in selecting appropriate techniques and setting the required parameters in the face of different types of uncertainty is currently limited. In this paper, we propose a set of application availability analysis models that capture subjective uncertainties in addition to stochastic uncertainties. We built and validated the models by using industry best practices on deployment, and actual commercial products for disaster recovery and live migration. Our results show that the models permit more informed and quantitative availability analysis than industry best practices under a wide range of scenarios.

Keywords—availability; uncertainty; stochastic reward nets; cloud computing; deployment architecture

I. INTRODUCTION

Deploying applications in cloud environments will introduce uncertainties for operations that have traditionally been under the direct control of an enterprise. Uncertainties are usually classified into two types: stochastic uncertainty and subjective uncertainty [1]. Stochastic uncertainty arises from the inherent randomness in the behavior of the system. In the cloud environment, day-to-day node and instance failures, rare large-scale disasters (e.g., a power outage or earthquake that could cause availability zones to go down), and workload spikes all have some level of inherent randomness. Subjective uncertainty is due to the lack of knowledge about an appropriate value that is assumed to have a fixed or optimal value for a particular analysis. Cloud consumers have to deal with different types of infrastructure uncertainties by deciding and analyzing their deployment choices (e.g., auto-scaling policies and recovery strategies) and the impact of sporadic activities (e.g., live migration, upgrade, and backup). These decisions and analyses often have high degrees of subjective uncertainties that need to be quantified and isolated from the stochastic uncertainties, especially for high consequence events such as disasters.

Availability is the measurement of a system’s uptime over a sufficiently long duration [2]. Currently, cloud consumers rely heavily on rule-of-thumb practices such as multi-zone deployment and over-provisioning to achieve availability under stochastic uncertainties [3], rather than applying a more informed and quantitative way to analyze application-level availability.

In this paper, we propose an approach to application-level availability analysis from the cloud consumer perspective to assist in deployment and decision-making around sporadic activities. The approach focuses on addressing subjective uncertainties while still taking into consideration stochastic uncertainties. Our contributions are: 1) a list of subjective uncertainties in deployment and sporadic activities decisions caused by corresponding stochastic uncertainties in cloud and; 2) a set of availability analysis models using Stochastic Reward Nets (SRNs) [4] addressing the above uncertainties focusing on subjective uncertainties. This paper provides a more detailed model and validation expanding our early position paper on using SRN for availability [22].

We use Amazon EC2’s features and empirical data for the modeling of auto-scaling and cross-availability-zone recovery. We use the actual features and empirical data from our commercial backup and disaster recovery product Yuruware Bolt\textsuperscript{1} for the modeling of backup and cross-region recovery. We use the features and empirical data from our database live migration research prototype CloudDB AutoAdmin [5] for more realistic modeling of stateful component scaling-out, especially database. Most of the parameters and calibrations are based on the commercial product and prototype.

We use current industry best practices on deployment decisions to validate our models and analyze the impact of uncertainties on availability. The availability results show that they are in line with the expected benefits of these best practices. However, these industry practices are usually coarse-grained options and rule-of-thumb numbers. Our availability models can make more quantitative and informed recommendations and allow what-if analysis under a wide range of scenarios. We also provide insights on some deployment decisions and scenarios.

The predictions of our models are meaningful for situations where the relative availability values are useful for practitioners to compare different deployment options and determine sensitivities of certain decisions. The absolute availability values depend on application-specific measurement and workload assumptions. Although we did measurements on launching different virtual machines (VMs) hosting stateless components in Amazon EC2 and used real data from our commercial backup and disaster recovery product Yuruware Bolt and our live migration prototype CloudDB AutoAdmin [5] to calibrate, we cannot claim the absolute values have high fidelity.

The rest of the paper is organized as follows. Section II provides a motivating example for our work. Section III discusses the connection between stochastic uncertainties

---

\textsuperscript{1} Yuruware Bolt—http://www.yuruware.com/
in cloud and subjective uncertainties in deployment and sporadic activity decisions. Section IV discusses details of the availability models. Section V validates the models using different scenarios. Section VI covers related work and Section VII concludes the paper.

II. MOTIVATING EXAMPLE

As shown in Fig. 1, our motivating example shows the architecture of a typical three-tiered web application deployed in a cloud. The application contains stateless components and stateful components which are deployed in separate VMs. We assume all the components are stateless at the web layer and the application server layer. The stateful components at the database layer provide database services to the upper layers.

The deployment crosses three zones (Zone A.1, Zone A.2, and Zone B.1) and two regions (Region A and Region B). Requests sent to each layer are handled by a load balancer and stateless VMs are deployed into auto-scaling groups. To sustain the availability under rare disaster events, deployment across multiple availability zones has a significant degree of over-provisioning to handle the workload spike after a massive failure [6]. A master-slave database replication is applied, which deploys the master in Zone A.1. A hot standby is hosted in a different zone to support quick failover management. The replicas in the same region as the master could keep a high-level of consistency because of the negligible network latency for the synchronization process. Live migration techniques represent the main tools for achieving scalability, elasticity and effective dynamic provisioning goals for stateful components by enabling the creation of new consistent serving replicas, as necessary, during runtime. Live migration is a resource intensive operation and can come at a price of degraded service performance during migration due to the overhead caused by the extra CPU cycles.

In our motivating example, a cross-region backup and disaster recovery solution is used. The application running in Region A manages a backup in Region B. This is different from the replica setup and the highly expensive hot standby setup. The application manages a periodic backup of the stateful components through VM snapshots. The backup files are stored on Amazon S3, a highly durable and cost-effective data store [7]. To shorten the launching time of VMs for recovery, the backup of the stateless components could also be AMIs (Amazon Machine Images). We do not consider fine-grained component collocation risks inside a single VM in this paper. We have a new deployment strategy [23] considering this risk using optimization.

III. UNCERTAINTIES IN CLOUD

We list the uncertainties in cloud environments based on literature review and the commercial products involved. We first identified a list of stochastic uncertainties in cloud and all the possible types of deployment and sporadic activity decision available to the cloud consumers as the candidate list of subjective uncertainties. We then related the stochastic ones and the subjective ones highlighting the important ones from the consumer perspective.

The cloud infrastructure and operation environment introduce various stochastic uncertainties that may make in-cloud applications unavailable. These stochastic uncertainties include:

- Runtime failures due to different reasons. For example, one availability zone may go down; a physical machine may experience a hardware error; a database might be overloaded; an operating system may crash; an API call may fail [8, 21].
- Workload spikes. Workload can change dramatically which is difficult to predict [9].
- Rare events. These may include disasters which are very difficult to predict [10].
- Availability and performance inference. Availability or performance issues of one application may interfere with other applications sharing the same resources [11].
- Cascading effects of small errors leading to a large-scale system outage [12].

To address the above stochastic uncertainties and achieve high availability, cloud consumers employ sophisticated deployment decisions, fine tune different parameters, use different recovery strategies and perform sporadic activities. These largely depend on a person’s limited knowledge of the system, which introduces subjective uncertainties. We list these subjective uncertainties and group them into categories which are reflected in the availability models.

1) Subjective uncertainties related to decisions on auto-scaling/live migration due to stochastic uncertainties in instance/node level failures and workload changes

- Recovery decisions for normal failures (not disasters). Different types of failures have different occurrence rates and different strategies may infer different failure detection time and recovery time which should be reflected in the models as subjective uncertainties. For example, we assume a VM instance running at full capacity and normal capacity has different failure rates and different types of VMs have different launch times.
- Configuration decisions in auto-scaling and live migration policy. For example, a certain degree of over-provisioning on stateless components can be helpful to handle the workload spikes at peak hours or a disaster. If zone 1 goes down, all workload in zone 1 can be distributed to zone 2 if the design adopts significant over-provisioning deployment (30-60%
constructs

SRN constructs

typical

Stochastic Reward Nets (SRNs)

uncertainties in rare disaster events

2) Subjective uncertainties related to decisions on disaster recovery strategies due to stochastic uncertainties in rare disaster events

• Configuration decisions for load balancers. More than one load balancer at each layer may be placed in different availability zones and/or regions to achieve high availability. For example, Amazon elastic load balancer consists of several load balancers with fault-tolerant mechanisms. Their configuration and availability themselves may have a major impact on the application availability.

• Configuration decisions for recovery from region-wide disasters. For example, administrators need to consider whether to have a backup and/or a replica in the other region and their configuration. This introduces a number of uncertainties regarding backup strategies and configuration of replicas and their use during recovery. Regarding the decisions on stateful components recovery, there are tradeoffs between availability and Recovery Point Objective (RPO) and Recovery Time Objective (RTO) [16]. Regarding the configuration of replicas and their numbers, there are tradeoffs between availability and cost.

• Configuration decisions for recovery from availability-zone-wide disasters. For example, Amazon EC2 provides automatic recovery during an availability-zone-wide disaster. We have discussed in early sections about the potential impact of a workload spike generated by such an event and decisions in auto-scaling triggering threshold. There are also other decisions on the number and placement of replicas for stateful components during such disasters.

3) Subjective uncertainties related to sporadic activities

• Configuration decisions for the frequency, resource impact and component freeze-time impact of sporadic activities including live migration, upgrade, and backup. Sporadic activities often need to stop some component of the in-cloud applications for a certain period no matter how short it is. They also consume resources during operation, which may impact the capacity and availability of the components on which they operate. The decisions on the frequency of such sporadic activities and the associated impact mentioned above introduce subjective uncertainties. Furthermore, the frequency of backups and live migrations also impacts cost and Recovery Point Objective (RPO).

IV. AVAILABILITY MODELS

To address the subjective uncertainties discussed in Section III, we build our availability models using Stochastic Reward Nets (SRNs) [4, 13] which are based on Stochastic Petri Nets. Our analysis models reflect the typical three-tiered application deployment architecture discussed in Section II. Here we explain each of the main SRN constructs (refer to Fig. 3-7 for the notation of these constructs). The inputs of the analysis 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.

The large circle in SRN is called a place, which could represent a state of a VM, a zone, a region, or an activity. The filled dots inside the places are tokens which represent VMs, zones, regions, or activities. The unfilled rectangles between places are transitions, which represent the events occurring in the system. Firing a transition could cause the reallocation of the tokens, which results in a new marking. A transition may be associated with a guard function, so that the transition is fired only if the guard function returns true. Different models can interact through guard functions. 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. In our model, the software failure and request arrival intervals follow an exponential distribution—a common assumption in such models [14, 15]. The output measures are expressed in terms of expected values of reward functions, which assign appropriate reward rates to the states of the SRN. The reward rates are assigned based on the output measure of interest, in our case the availability. “1” represents available and “0” represents unavailable. 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.

A. Auto-scaling/live migration model for instance/node-level failures and workload spikes

The model shown in Fig. 2 is proposed to deal with normal node/instance-level failures and handle the increasing workload using the preconfigured auto-scaling (for stateless components) and live migration (for stateful database components) strategy. The broken line in the middle separates two zones. The token in Fig. 2 represents a VM. When a stateless VM is serving workload, it is in Pza1running or Pza2running. When a stateful VM is serving workload for both read and write operations, it is in Pza1master. The stateful VM in Pza1replica or Pza2replica is serving workload for read only operations. If any failures happen inside a VM, the corresponding tokens are moved from these places to the corresponding place Pza1stopped or Pza2stopped for stateless VMs or to Pza1dstopped or Pza2dstopped for stateful VMs. If a stateless VM in Zone A.1 reaches a predefined CPU threshold, the incoming workload is handled by adding more stateless VMs through triggering the transition Tza1scaleself or Tza1scaleother which represent the auto-scaling mechanism with load balancers. Whether to trigger Tza1scaleself or Tza1scaleother 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, the guard function Tza1scaleself is satisfied which fires Tza1scaleself and creates a new VM in Zone A.1. Otherwise, a new VM is created in Zone A.2. Zone A.2 is modeled in the same way as Zone A.1.

On the other hand, if a stateful VM hosting replica in Zone A.1 reaches a predefined CPU threshold: more replicas are added to handle the increasing workload by live migration mechanism, which is represented by the
transition \( Tza1 \text{migrate} \text{self} \) or \( Tza1 \text{migrate} \text{other} \). Similarly, the triggering of \( Tza1 \text{migrate} \text{self} \) or \( Tza1 \text{migrate} \text{other} \) follows a load balancing strategy.

\[ \text{Figure 2. Auto-scaling/live migration model for instance/node-level failures and workload spikes.} \]

\[ \text{B. Zone/Region Recovery} \]

For business continuity, an organization needs to prepare for rare but high consequence events, for example, a zone or a region goes down. Fig. 3 shows the zone recovery model. The models representing the lifecycle of Zone A.1 and Zone A.2 are in the lower part of Fig. 3 (below the solid line). The token in the lifecycle model represents an availability zone. The token in the models located in the upper part represents VMs. Initially, the number of running stateless VMs in each zone is \( M \) and the number of stopped stateless VMs in each zone is \( N \).

\[ \text{Figure 3. Zone recovery model.} \]

When Zone A.1 is down, the lifecycle models relocate the token in \( Pza1 \text{up} \) to \( Pza1 \text{down} \). When Zone A.1 is down, two transitions for stateless VMs are triggered. One transition moves all the tokens in \( Pza1 \text{running} \) to \( Pza1 \text{stopped} \) and the other one moves VMnum tokens from \( Pza2 \text{stop} \) to \( Pza2 \text{running} \). The value of VMnum is equaled to or smaller than \( M \) depending on the degree of over-provisioning provided by the VMs running in Zone A.2.

For stateful VMs, when Zone A.1 is down, the token in \( Pza2 \text{hot standby} \) is relocated in \( Pza2 \text{master} \), which represents that the hot standby in Zone A.2 becomes the master database when Zone A.1 is down.

Fig. 4 shows our disaster region recovery model, which involves Region A, and Region B. The models within the bold box represent three fundamental strategies for disaster recovery: backup, replicas, and hot standbys. In this model, we do not consider the situation when both Region A and Region B are down at the same time. Thus, there is only one separate model in the lower part to represent the lifecycle of Region A. When Region A is down, the token in \( Przaup \) transits to \( Przadown \).

\[ \text{Figure 4. Region recovery model.} \]

For stateless VMs, the recovery is based on backup. When Region A is down, the model consumes the token in \( Pza1 \text{running} \) and \( Pza2 \text{stopped} \), and generates a token in \( Pza1 \text{stopped} \) and \( Pza2 \text{running} \), which represent the stateless components are recovered.

For stateful VMs, there are three optional ways to recover from disaster: backup, replica, and hot standby. Each of these three strategies need some time to achieve data consistency and transaction consistency [16]. The backup strategy regularly takes backups of the application. The backup files are stored in a different region to the region where the application resides. When a disaster occurs, the application can be restored from the backup stored at the other region, which takes some time. For recovery from backup, the model first moves the token from \( Pzb1 \text{stopped} \) to \( Pzb1 \text{inconsistent} \), which represents the process of re-launching stateful components. Then the model moves the token from \( Pzb1 \text{inconsistent} \) to \( Pzb1 \text{consistent} \), which represents the process of checking state consistency. If replica-based recovery or hot standby-based recovery is selected as the recovery strategy, it only takes some time to check state consistency. The time for
hot standby-based recovery is very short and can be negligible. In our model, for these two types of recovery, the token transits from Pz1replica or Pz1hotstandby to Pz1master with different transition rates.

C. Sporadic Activities

Fig. 5 shows the sporadic activities model. The token located in the lower part of the model represents a sporadic activity while the token in the upper part represents VMs. Here, we only take Zone A.1 as an example.

![Figure 5. Sporadic activities model.](image)

Within a short moment of the sporadic activity period, the application freezes and is not able to serve any requests. Besides, there is a resource impact on the nodes being operated as the reduced processing capacities of these nodes. When an activity starts, the model moves the token from Pz1running and Pz1master to Pz1stopped and Pz1master respectively. When the token is in Pz1idle, and Pz1idle, at the same time, the model consumes the tokens in Pz1stopped and Pz1master, and generates a token in Pz1running and Pz1master respectively.

Nowadays system upgrades are a common occurrence since continuous deployment [17] has become a goal of companies providing web-based applications. One strategy for upgrading is a “rolling upgrade”, which replaces the old version of an application with a new version one component at a time. When a specific component instance is being upgraded, other instances of the same component are still available to run and process requests. The overall availability is not impacted greatly but the new problem of a mixed version race condition may be introduced. Fig. 6 shows the rolling upgrade model.

![Figure 6. Rolling upgrade model.](image)

In Fig.6, there are N VMs in Pz1running. When the model moves the token from Pz1idle to Pz1started, gstart returns “1”, which means the rolling upgrade starts. Tokens then transit from Pz1running to Pz1stopped one by one. When the model consumes all tokens in Pz1running, gfinish returns “1”, meaning that the rolling upgrade is completed. The model then moves the token from Pz1started to Pz1idle, which enables grun. Thus, all tokens are moved from Pz1stopped to Pz1running.

V. IMPACT OF UNCERTAINTIES ON AVAILABILITY: MODEL VALIDATION AND ANALYSIS

A. Impact of auto-scaling with over-provisioning

One type of industry best practice [3] recommends live replication across multiple availability zones and a certain degree of over-provisioning to handle the load spike after an availability-zone-wide failure. Amazon EC2 only provides availability agreements on availability zones. We use the Amazon EC2 SLA commitment of 99.95% availability in our experiments [18]. In particular, we assume an availability zone to be down once per year. The corresponding zone fail rate is 0.00011, and the mean time to repair is 4.38 hours per year.

We now assume one availability zone is entirely down. We measured the availability of our application under different over-provisioning policies using the availability model shown in Fig. 3. Our model assumes the healthy availability zone can scale out VMs quickly while accepting the shifting workload from the failed availability zone after disaster occurs. Launching new VMs takes longer time than turning under-utilized VMs towards full capability. We assume the launch time of the first VM is 2 minutes and launching one additional VM takes an extra 6 seconds. For example, the corresponding scaling time for scaling out three VMs is 2.2 minutes. We also assume that the application has 10 VMs in each zone when one of the availability zones down. The higher the over-provisioning, the fewer new VMs need to be launched during workload spikes. For example, if the CPU threshold decreases from 75% to 70%, the number of scale out VMs decreases from 5 to 4. Table I shows the values of relevant parameters and the resultant availability.

<table>
<thead>
<tr>
<th>CPU Threshold</th>
<th>Scale out VM number</th>
<th>Scale out time (min)</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>50%</td>
<td>0</td>
<td>0</td>
<td>0.9999997500</td>
</tr>
<tr>
<td>55%</td>
<td>1</td>
<td>2.0</td>
<td>0.99999974761</td>
</tr>
<tr>
<td>60%</td>
<td>2</td>
<td>2.1</td>
<td>0.99999974751</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>85%</td>
<td>7</td>
<td>2.6</td>
<td>0.99999974707</td>
</tr>
<tr>
<td>90%</td>
<td>8</td>
<td>2.7</td>
<td>0.99999974699</td>
</tr>
<tr>
<td>95%</td>
<td>9</td>
<td>2.8</td>
<td>0.99999974692</td>
</tr>
</tbody>
</table>

Since the application is deployed on two zones, the probability that at least one zone is up is 1-(1-0.9995)*(1-0.9995) = 0.99999975. The availability value shown in the last column shows that auto-scaling with over-provisioning can help approach maximum. This marginal change is mainly due to relatively short time duration of scaling in comparison with an assumed timeframe of 1 year.

B. Impact of disaster recovery strategy

There are commercial products that support disaster recovery strategies. For backup, we use our disaster recovery product Yuruware Bolt for our usage scenarios and model calibration. Yuruware Bolt adopts a low cost VM-snapshot-based backup and recovery. For replica-based recovery, we assume it manages a cross-region replica of the application where the replica could accept requests when the primary region is down due to disasters. Both backup and replica strategies have the problem of achieving consistency due to backup frequencies and
replication delays across regions. Hot standby strategy can solve the problem through managing a consistent “copy” of the application using dedicated fiber optics communications over full replications across regions. However, hot standby is much more expensive than the first two strategies.

Recovery-Point-Objective (RPO), data consistency Recovery-Time-Objective (RTO) and transaction consistency RTO [16] are the main factors that constrain the selection of backup and recovery strategies during a disaster. We conducted sensitivity analysis of the parameters on data consistency RTO regarding three different strategies using typical industry data and empirical data from Yuruware Bolt. We assume the region failure rate is equal to zone failure rate. Due to space limitation, we cannot show the results. The results are in line with the expectation that a longer RTO produces a lower availability.

Here we extract from the sensitivity analysis data to show data-consistency RTO and estimated cost level of three disaster recovery strategies for a particular availability level - 99.98%, as shown in TABLE II.

<table>
<thead>
<tr>
<th>Yuruware Bolt</th>
<th>Cost</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replica-based recovery</td>
<td>Low</td>
<td>1 h (45 min; 15 min)</td>
</tr>
<tr>
<td>Hot standby-based</td>
<td>Medium</td>
<td>1 h</td>
</tr>
<tr>
<td></td>
<td>High</td>
<td>5 sec</td>
</tr>
</tbody>
</table>

To achieve above 99.98% availability over a year under the same assumption of outage frequency as above, Yuruware Bolt implies a 45mins for achieving data consistency and 15mins for re-launching the application from backup, assuming a 4-hour backup frequency with 1-hour snapshot activity impacting the resource usage. The failover of the application by a hot standby located in another region is very quick (5s) for full data consistency and easily achieves the above 99.98% availability guarantees. The replica strategy requires 1 hour to achieve data consistency for guaranteeing above-99.98% availability. TABLE II gives the cost estimation of the three strategies and their derived data-consistent RTO (for backup-based recovery and replica-based recovery) through sensitivity analysis and estimated data-consistent RTO (for hot standby-based recovery).

Our availability model is validated in a weak form as it aligns with the rough availability expectation of the different backup/recovery strategies regarding RTOs. Our models further show quantitatively that there is a trade-off among RPO, data-consistent RTO and cost regarding availability. Generally, when backup frequency increases, both data consistency RTO and RPO reduce and it has positive impact on availability even assuming resource impact of the backup activities. A low-cost backup strategy such as using Yuruware Bolt can achieve a reasonable level of availability.

C. Impact of sporadic activities

Sporadic activities often need to stop some components of the in-cloud applications for a certain period. They also consume resources during operation, which may impact the capacity and availability of the components on which they operate. As live migration and other sporadic activities differ, we discuss them separately.

Analysis of availability interference

We use live migration as an example to illustrate analysis of availability interference caused by sporadic activities occupying extra CPU usage. We use the model in Fig. 3 to analyze how different strategies of live migration impact the overall availability, and quantify such impact. The overall availability during one year can be calculated by the following equation:

$$A = A_{LM} \times \frac{LMT_{normal}}{365 \times 24} + A_{normal} \times \left(1 - \frac{LMT_{normal}}{365 \times 24}\right)$$

where $A_{LM}$ represents the system availability during live migration while $A_{normal}$ represents the system availability without live migration. The overall availability of an entire year is also affected by the frequency of live migration ($LMT$) and the migration time ($LMT_{normal}$). Assume the database server fails once every 6 months; the value of $A_{normal}$ is 99.746%.

When live migration is being processed, it may affect performance and availability of the database server being migrated. We assume 1) the CPU usage of live migration is from 1% to 20%; 2) the failure rate of database increases to 0.0028 if CPU usage of live migration is 20%, which implies the mean time between failures is half month; and; 3) the mean time between failures is inversely proportional to the usage of CPU. The rates in our analysis are shown in TABLE III.

<table>
<thead>
<tr>
<th>CPU usage (pct.)</th>
<th>Failure rate (1/h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.00%</td>
<td>0.00039</td>
</tr>
<tr>
<td>6.00%</td>
<td>0.00042</td>
</tr>
<tr>
<td>7.00%</td>
<td>0.00044</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>18.00%</td>
<td>0.00154</td>
</tr>
<tr>
<td>19.00%</td>
<td>0.00198</td>
</tr>
<tr>
<td>20.00%</td>
<td>0.00277</td>
</tr>
</tbody>
</table>

We examined the master and replica databases separately by using the same set of values. The result is shown in Fig. 7, which presents the relationship between $A_{LM}$ and the CPU usage of live migration. The red line represents the situation when the master database is migrated. The blue line represents the replica. It shows that when the master is being migrated, the overall availability decreases from 99.65% to 97.05% as the CPU usage increases from 5% to 20%. However, when the replica is being migrated, the overall availability decreases slightly as the CPU usage increases. Most web applications use master-slave database replication. Under this situation, the live migration of a specific replica does not impact the overall availability significantly since all the other replicas still can process requests when a certain replica is being migrated. Unlike replica, the master is the bottleneck of the system and the overall availability is more sensitive to the change of CPU occupation during live migration of the master.

Given information of the type of database server being migrated, the percentage of CPU occupied by the live migration, the frequency of live migration and the migration time, the overall availability could be estimated. Due to space limitations, we do not show the value of overall availability changes. There is a tradeoff between cost and data consistency when selecting the database.
server as the source to create a new replica through live migration. Migrating from a master could essentially guarantee the data consistency with high cost in terms of the performance and availability interference during live migration. Migrating from a replica cost less than migrating from a master as discussed above. Replicas across zones or regions are not as consistent as replicas at the same zone with the master.

Comparing to the application using non-rolling upgrade strategy tends to have higher availability, the overall availability. Furthermore, the mean time between upgrade has positive impact on upgrade and upgrade freeze time separately. Fig. 8 shows the upgrade interval is set to be 12 hours, and 1 minute for upgrade freeze time.

### Table IV. Rate in Analysis of Upgrade

<table>
<thead>
<tr>
<th>Mean time between upgrade (h)</th>
<th>Upgrade rate (1/h)</th>
<th>Upgrade freeze time (min)</th>
<th>Upgrade freeze rate (1/h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>0.0833</td>
<td>1</td>
<td>60.0000</td>
</tr>
<tr>
<td>24</td>
<td>0.0417</td>
<td>2</td>
<td>30.0000</td>
</tr>
<tr>
<td>36</td>
<td>0.0278</td>
<td>3</td>
<td>20.0000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>216</td>
<td>0.0046</td>
<td>18</td>
<td>3.3333</td>
</tr>
<tr>
<td>228</td>
<td>0.0044</td>
<td>19</td>
<td>3.1579</td>
</tr>
<tr>
<td>240</td>
<td>0.0042</td>
<td>20</td>
<td>3.0000</td>
</tr>
</tbody>
</table>

Fig. 8 and Fig. 9 give the result of meantime between upgrade and upgrade freeze time separately. Fig. 8 shows that the mean time between upgrade has positive impact on the overall availability. Further, the application using rolling upgrade strategy tends to have higher availability, comparing to the application using non-rolling. Fig. 9 suggests that the overall availability is more sensitive to changes for upgrade freeze time of non-rolling strategy. In the freeze time of 1 to 20 minutes, the overall availability decreases from 99.79% to 95.99% in the case of rolling upgrade strategy. The corresponding figure for non-rolling is 99.26% and 87.20%.

### Impact of freeze time by sporadic activities

All of the sporadic activities we consider need to stop the system or specific components for a certain time period. The freeze time period of live migration and backup is very short, which is normally several seconds, in order to guarantee the data consistency. Through the examination, it shows that the overall availability could be increased by either decreasing the frequency of backup and live migration or decreasing the freeze time. We do not show these obvious results due to limited space. However, there is a tradeoff between availability and data consistency when selecting the frequency of backup and live migration.

Normally, the upgrade also causes a freeze time period of several minutes to hours at the component level, which is much longer than live migration and backup. Thus, we only show the result regarding upgrade. We compare two kinds of upgrade: rolling and non-rolling. We examined the impact of mean time between upgrade and upgrade freeze time on the overall availability for both cases. TABLE V gives the value of parameters we used. The interval is set to be 12 hours for mean time between upgrade, and 1 minute for upgrade freeze time.

![Figure 7. Sensitivity analysis of CPU usage by live migration.](image)

![Figure 8. Sensitivity analysis of mean time between upgrade on availability.](image)

![Figure 9. Sensitivity analysis of upgrade freeze time on availability.](image)

The sensitivity analysis implies that the impact of sporadic activities on the overall availability could be significantly reduced by introducing a rolling strategy even though the upgrade frequency is very high. In addition, the overall availability could be further improved through scheduling the sporadic activities at the same time to overlap the freeze time caused by different sporadic activities as much as possible.

### VI. Related Work

There has been much work focusing on infrastructure availability analysis in terms of placing VMs onto physical machines, assuming certain failures in the physical machines [14]. Our approach uses similar types of analytical models for modeling availability but works at the application level with the actual deployment options available to consumers and the subjective uncertainties involved.

At the application deployment level, approaches like [20] 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 auto-scaling facilities and failures ranging from individual nodes to an entire region. Furthermore, the reliability model in [20] seems simplistic, since it only considers communication frequency and network reliability.
Other work focuses on how to convert individual SysML-based application architectures into availability analysis models [15] which requires expertise in complicated analytical models and could be costly. Our analysis models consider uncertainties in cloud and are more generalized and applicable to typical in-cloud deployments involving stateful and stateless components. Different applications and deployment decisions usually only require different parameter settings rather than complete re-modeling. We also consider a wider range of scenarios (viz., sporadic activities, region disasters and different recovery strategies) using more realistic options and mechanisms available to the consumers.

VII. CONCLUSION

We presented an approach to application-level availability analysis from the cloud consumer perspective focusing on subjective uncertainties while still taking into consideration stochastic uncertainties. We listed subjective uncertainties in deployment and sporadic activities decisions caused by the corresponding stochastic uncertainties in cloud. To address the uncertainties, we proposed a set of availability analysis models using Stochastic Reward Nets. We validated our models by re-evaluating industry best practices and comparing if our models predict the same results. On the other hand, our models gave more quantitative and informed comparisons of different decisions under a wider range of what-if scenarios. We also provided insights into some deployment decisions.

VIII. ACKNOWLEDGEMENT

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.

REFERENCES