CLOUD STANDBY SYSTEM AND QUALITY MODEL

Alexander Lenk, Frank Pallas
FZI Forschungszentrum Informatik
Friedrichstr. 60, 10117 Berlin, Germany
{lenk, pallas}@fzi.de

Abstract

Contingency plans for disaster preparedness and concepts for resuming regular operation as quickly as possible have been an integral part of running a company for decades. Today, large portions of revenue generation are taking place over the Internet and it has to be ensured that the respective resources and processes are secured against disasters, too. Cloud-Standby-Systems are a way for replicating an IT infrastructure to the Cloud. In this work, the Cloud Standby approach and a Markov-based model is presented that can be used to analyze and configure Cloud Standby systems on a long term basis. It is shown that by using a Cloud-Standby-System the availability can be increased, how configuration parameters like the replication interval can be optimized, and that the model can be used for supporting the decision whether the infrastructure should be replicated or not.

Keywords: Cloud-Standby, Warm-Standby, BCM, Cloud Computing, IaaS, Disaster Recovery

1. INTRODUCTION

The effort of companies to protect their production facilities, distribution channels or critical business processes against possible risks is not a new phenomenon. Instead, contingency plans for disaster preparedness and concepts for resuming regular operation as quickly as possible have been an integral part of running a company since the times of the industrial revolution. In this context, disasters are fire, earthquakes, terrorist attacks, power outages, theft, illness, or similar circumstances. The respective measures that must be taken in order to being prepared for such disasters and for keeping up critical business processes in the event of an emergency are commonly referred to as *Business Continuity Management (BCM)* (Hiles, 2010) in economics. The effectiveness of BCM can be controlled via the key figures *Recovery Time Objective (RTO)* and *Recovery Point Objective (RPO)* (Hiles, 2010). RTO refers to the allowed time for which the business process may be interrupted and the RPO relates to the accepted amount of produced units or data that may be lost by an outage.

Today, with the Internet being production site as well as distribution channel, BCM faces different challenges. One of the most important tasks in IT-related emergency management is the redundant replication of critical systems. Depending on the system class, different mechanisms are used to secure a system against prolonged outages. In this regard the RTO specifies the maximum allowed time within which the IT system must be up again and the RPO is the accepted period of data updates that may be lost, i.e. generally the time between two backups (Wood et al., 2010).

This work presents an approach for a Cloud Standby system and the modeling of costs and availability based on Markov chains. Cloud Standby uses a *meta model* approach to describe distributed systems (Tanenbaum & Van Steen, 2002) in a machine readable language that can be used to deploy the system on several cloud providers. For modeling the quality attributes the basic idea is to carry out a random walk (Gilks, Richardson, & Spiegelhalter, 1996) on the system's state graph according to defined transition probabilities. The costs and the availability can then be calculated by means of the Markov chain and the probability distribution for staying in each state. The presented model is illustrated by means of a simple example and it is shown that the model can be used to calculate optimal configuration options, like the replication interval, of the Cloud-Standby-Systems.

The remainder of this paper is structured as follows: First the related work and a description of the Cloud-Standby-System is presented. Then the quality model itself is developed and it is shown how it can be used to make deliberate configuration decisions on the basis of a simple example. Finally, the conclusion sums up the paper and gives an outlook to future work in this field[1].

2. RELATED WORK

In this work we present a) the general approach of Cloud Standby, a warm standby for the Cloud and b) based on the states of the proposed Cloud Standby System a quality model for predicting the long term availability and costs with certain parameters.

2.1 Cloud Standby.

Wood et al. (2010) describe a generic warm standby system in order to evaluate whether Cloud Computing is beneficial for this use case. This paper does however concentrate on the economic part and describes no real warm standby system.

Klems, Tai, Shwartz, and Grabarnik (2010) describe a running system that allows to use the Cloud as a warm standby environment, using BPNM processes to orchestrate an IaaS provider's infrastructure services. In contrast to the

[1] This article is an extended version of the paper "Modeling Quality Attributes of Cloud-Standby-Systems" (Lenk & Pallas, 2013)

model presented herein, their approach focuses on single machines and does not allow to secure whole distributed systems.

PipeCloud (Wood, Lagar-Cavilla, Ramakrishnan, Shenoy, & Van der Merwe, 2011) targets private Clouds where the user has access to the hypervisor and uses this access for capturing disk writes and replicating them to the Cloud. Remus and SecondSite (Cully et al., 2008; Rajagopalan, Cully, O'Connor, & Warfield, 2012) basically follow a similar approach: direct the access to the hypervisor is used to dynamically replicate the whole virtual machine to another location. Due to the requirement of hypervisor access, these approaches can, however, not be applied in a public Cloud scenario.

2.2 Quality Model.

The calculation of quality metrics addressed in this paper can generally be subdivided into the two fields of cost and availability calculation. Regarding these calculations, related work already exists in the field of virtualized infrastructures, Cloud Computing, and warm standby systems.

The approach of Alhazmi and Malaiya (2012) describes a way of evaluating disaster recovery plans by calculating the costs. The approach is of generic nature and is not focusing on the field of Cloud Computing with its own specific pricing models.

Wood et al. (2010) describe a way of replicating data from one virtual machine to a replica machine. The respective cost calculation is limited to this specific approach and cannot be adapted to Cloud-Standby-Systems like the ones considered herein.

Dantas, Matos, Araujo, and Maciel (2012) present a Markov-based approach to model the availability of a warm-standby Eucalyptus cluster. Even if the approach is related to the work presented herein with regards to the used mathematical model and also shows that Markov chains can be used to model availabilities in Cloud Computing, it is not used to model the costs and the calculation of the availability is restricted to a single Eucalyptus installation with different clusters and does not consider settings with several cloud providers.

Klems et al. (2010) present an approach for calculating the downtime of a Cloud-Standby-System. This approach evaluates the system in general but is a rather simplistic short term approach, comparing a Cloud-Standby-System with a manual replication approach.

2.3 IaaS Deployment Meta Model.

Over the years several IaaS deployment standards were proposed in the industry and science. Some of them are related to the deployment model presented in this paper.

Amazon Web Services (AWS Inc., 2013) is an IaaS provider that added additional services over the time. Today Amazon Web Services cannot just be used to start virtual machines but to deploy whole scalable application stacks. These "Cloud Formations" allow constraint definition and sophisticated monitoring. As one the industry leader in IaaS Cloud computing Amazon has a rich toolset and API. However Amazon does not support provider independence since as an industry leader they are more interested in creating lock-in effects than reducing them.

Chieu et al (2010) introduce in their work the concept of a "Composite Appliance"'. A composite appliance is a collection of virtual machine images that have to be started together in order to run a multi-tier application. The proposed description language lacks support for runtime features like the current configuration of the virtual machines. The architecture proposed in this work requires the language to be implemented in the vendor's Cloud computing platform and thereby does not allow using different cloud providers.

Konstantinou et al. (2009) describe in their work a model-driven deployment description approach. This approach is based on different parties that participate in the deployment process. A deployment is first modeled as a "Virtual Solution Model" and then by another party translated to a vendor-specific "Virtual Deployment Model". Maximilien et al. (2009) introduce a model that allows deploying middleware systems on a Cloud infrastructure. In the aspects of deployment both the meta model proposed by Konstantinou et al. and Maximilien et al. are related to our meta model. The approach of Maximilien et al. however is focused on the deployment of middleware technologies we are aiming on a more holistic approach that has the final distributed system in focus and not just middleare technologies. The approach of Konstantinou et al. does not allow having different Cloud vendors in a single deployment. However some of both the ideas of Konstantinou et al. and Maximilien et al. have influenced our meta model.

Mietzner et al. (2009) orchestrate services by using workflows and web services using e.g. WS-BPEL. They concentrate on high level composition of services, rather than proposing an actual language for deployments. The focus on WS-BPEL gives the approach a great flexibility but also adds an overhead. The work we present in this paper is influenced by the work done and the general ideas developed by Mietzner et al. However while Mietzner et al. is focusing on the BPEL deployment processes, our work is focusing on the distributed application itself and enriches it with meta data that can also be used for deployment and thereby supporting higher level processes like the ones described by Mietzner et al.

3. CLOUD STANDBY SYSTEM

The recovery of IT systems can be achieved through different replication mechanisms (Henderson, 2008; Schmidt, 2006). "Hot standby" is on the one side of the spectrum: A second data center with identical infrastructure is actively operated on another site with relevant data being continuously and consistently mirrored in almost real-time from the first to the second data center. The operating costs of such a

hot standby system, however, amount to the operating costs of the secondary system plus the cost of the mirroring. On the other side of the spectrum is the "cold standby", the low-cost backup, e.g. on tape, without retaining a second site with backup infrastructure resources. A tape backup is not possible during productive operation and is usually done at times of low load like at night or during weekends. In this case, a RPO of days or weeks is common. Due to the fact that the IT infrastructure has to be newly procured in case of a disaster, an RTO of several days to months is possible. Between these two extremes lies the concept of "warm standby". Although a backup infrastructure is kept at another location in this case, it is not fully active and must be initially put into operation in case of a disaster. A warm standby system usually has a RPO and RTO between minutes and hours.

A common option for reducing the operating costs of only sporadically used IT infrastructure, such as in the case of the "warm standby" (Henderson, 2008; Schmidt, 2006), is Cloud Computing. Cloud Computing provides the user with a simple, direct access to a pool of configurable, elastic computing resources (e.g. networks, servers, storage, applications, and other services, with a pay-per-use pricing model) (Lenk, Klems, Nimis, Tai, & Sandholm, 2009; Mell & Grance, 2011). More specifically, this means that resources can be quickly (de-)provisioned by the user with minimal provider interaction and are also billed on the basis of actual consumption. This pricing model makes Cloud Computing a well-suited platform for hosting a replication site offering high availability at a reasonable price. Such a warm standby system with infrastructure resources (virtual machines, images, etc.) being located and updated in the Cloud is herein referred to as a "Cloud-Standby-System". The relevance and potential of this cloud-based option for hosting standby systems gets even more obvious in the light of the current situation in the market. Only fifty percent of small and medium enterprises currently practice BCM with regard to their IT-services while downtime costs sum up to $12,500-23,000 per day for them (Symantec, 2011).

The calculation of quality properties, such as the costs or the availability of a standby system, and the comparison with a "no-standby system" without replication is an important basis for decision-making in terms of both the introduction and the configuration of Cloud-Standby-Systems. However, due to the structure and nature of standby systems, this calculation is not trivial, as in each replication state different kinds of costs (replication costs, breakdown costs, etc.) with different cost structures incur. Furthermore, determining the quality of the system is difficult due to the long periods of time and the low probability of disasters (e.g. only one total outage every 10 years). A purely experimental determination by observing a reference system over decades is therefore not feasible. Instead, a method for simulating and calculating the long-term quality characteristics of different configurations is needed.

In this work we introduce Cloud-Standby. It is a novel Cloud based warm standby approach where a formally described primary distributed system (PS) running in Cloud 1 (C1) is periodically synced as a replica system (RS) to Cloud 2 (C2). The states of this Cloud-Standby-System are depicted in Fig. 1.

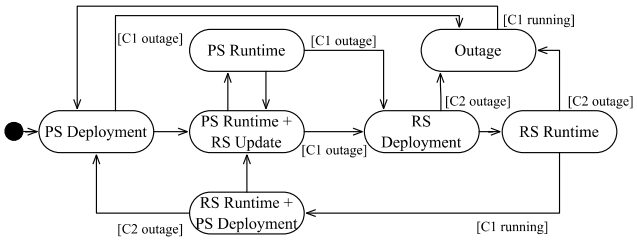

Fig. 1. State chart of a Cloud-Standby-System

3.1 States.

PS Deployment. The PS is deployed on C1 at first and goes into runtime after t_{depl}. The time the deployment takes depends highly on its structure. For each use case the deployment time can be determined by means of experimentation.

PS Runtime. During PS runtime, the RS is turned off and generates no costs. The PS data are, however backed up using standard backup tools. This ensures the RPO can be met when a disaster occurs.

PS Runtime + RS Update. Periodically (after $t_{updateInt}$) the RS is started and changes are updated on the RS. This ensures that the deployment time of the RS is reduced when an actual disaster occurs. The time the update process lasts is defined as t_{update}

RS Deployment. When C1 fails the RS is started and takes over the service. The time for the deployment is $t_{replDepl}$. The deployment time varies with the amount of data that needs to be installed or stored during the deployment process. This means that $t_{replDepl}$ decreases with decreasing $t_{updateInt}$. The correlation between $t_{replDepl}$ and $t_{updateInt}$ can be determined through experiments or monitoring over time.

RS Runtime. In case of an outage on C1, the RS takes over and only if during this time an outage also takes place on C2 the whole system is unavailable.

RS Runtime + PS Deployment. As soon as C1 is up again, the PS can be redeployed and then takes over the service.

Outage. If the systems on both Clouds are not available the outage of the service could not be avoided. Now the whole system must be recovered from the scratch by hand.

3.2 IaaS Deployment Meta Model.

For the automatic deployment of the PS and RS a machine readable deployment description is needed which

allows the deployment of distributed systems on different IaaS providers. In this section we present a modeling approach for formalizing distributed systems on top of federated IaaS Clouds. In this context a distributed system is a software stack capable of providing a distributed application and all its artifacts (Object Management Group – OMG, 2011). Software artifacts are applications like a self-contained webserver binary or application packages like jar-files. Even a whole operation system including kernel, binaries, and other files, packaged in an image file is considered as an artifact. In the UML-notation the Component acts as a "DeploymentTarget" for the "DeployedArtifacts" (Object Management Group – OMG, 2011).

Deployment languages in federated Clouds have to meet several requirements (Lenk, Danschel, Klems, Bermbach, & Kurze, 2011). A Cloud service, by definition, needs to be elastic, meaning to scale up and down with changing demand within a short time (Mell & Grance, 2011). Since it should also be possible to deploy a service not only on a single Cloud provider, the model must also support Cloud federation (Kurze et al., 2011; Lenk et al., 2011).

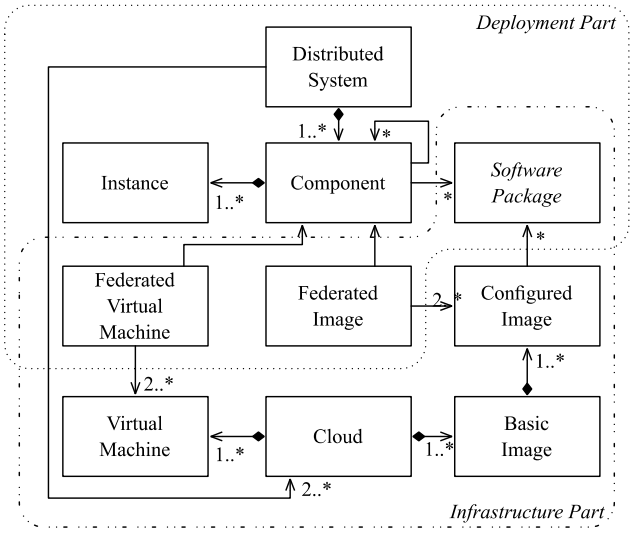

Fig. 2. Distributed system deployment meta model

In Fig. 2 we depict a meta model for distributed systems in federated Clouds. A distributed system consists of several elastic components that have dependencies between each other ("component a" requires "component b"), representing the tiers in the distributed system. These components can be application servers, databases, key-value stores, etc. All components are running on a single cloud provider at a time but can - in the case of a disaster - be deployed on another provider in the federated Cloud. Each of these components has several instances running a configured operation system with a predefined software stack. This software stack is either stored in a basic image held by Cloud provider or is applied during the deployment process via installation of software packages on a single basic image or all the images of a component.

This modeling approach of having the federated image, federated virtual machine, and installation tasks assigned to the component ensures that all instances of a component have the same configuration and are thereby horizontally scalable. We assume that the functionality of the load balancing is done via an external service having access to the model, or the load balancer being one component, required by the component it controls.

Infrastructure part. This part is modeled by a professional (the IaaS Admin) knowing the different Cloud providers, images, and software stacks. The task of the IaaS Admin is to select feasible virtual machines and basic images from the available infrastructure and software packages that can be installed on top of the basic images for the preparation of configured images. Configured images are the functional design time representations of instances. Differently configured images for different Cloud providers which are functionally equivalent will be grouped to federated images. This ensures that the federated image has the same functionality even when instantiated on different Clouds.

In order to instantiate an instance the image needs to be deployed on a runtime, or virtual machine (VM). The virtual machine represents the non-functional attributes of the instance. The selected VM determines the properties like price, performance etc. for the instance. By defining intervals for these properties and by grouping the VMs along these dimensions in the intervals, the IaaS Admin guarantees that a federated virtual machine has the same or at least similar non-functional qualities on each provider.

Furthermore, the IaaS Admin groups for each Cloud exactly one VM and one configured image in the corresponding federated element. Thus, during the deployment phase the federated image and federated VM can be deployed on any Cloud provider represented in the infrastructure model.

One challenge in in this approach is to determine the virtual machines and basic images of different Cloud providers that are grouped to federated virtual machines and federated images. Furthermore, to determine the software that should be installed on a certain image is a highly manual task that should be automated in the future. Concepts like feature modeling for selection of Cloud services (Wittern, Kuhlenkamp, & Menzel, 2012) could be used to automate this process.

Deployment part. In the deployment part the System Administrator defines the structure of the final distributed system on the basis of the infrastructure model. Since it is not clear how the different components are orchestrated for the final service, this task (currently) has to be carried out manually by the system administrator. He defines the components, the dependencies between them, and the software packages installed on the components. By grouping instances in components the model enables elasticity.[2]

[2] It is, however, not subject of this work to describe mechanisms that are used to add/remove resources based on monitoring data or that deal

Deployment parallelization. The relation "component requires component" allows to calculate the dependency graph G that is essential for the deployment process of the PS and the RS. It depends on the structure of the deployment modeled with the description language how long the actual deployment takes. If there are many single software packages to be installed and the deployment graph allows no parallelization this process takes longer than in cases with completely configured virtual machines that all can be deployed in parallel.

Without this graph it is not clear in which order the components have to be deployed. When having the dependency graph calculated with a topological sort the sequential deployment order of the components can be calculated. This sequential deployment order is not optimal when it comes to deployment time. In the deployment graph there are often parallel paths where deployments can be started at the same time without violating the constraints introduced by the "requires" relation.

```
calcActivityTimes := proc(G :: Graph)
 local topSort, startTime, duration, neighbors, max, n, v;

 # compute a linear ordering consistent with a given partial ordering
 topSort := TopologicalSort(Edges(G));
 for v in Reverse(topSort) do
   #check output relations
   if Departures(G, v) = [ ] then
       startTime := 0;
   else
       neighbors := Departures(G, v);
       for n in neighbors do
           max := GetVertexAttribute(G, n, "EFD");
           if max > startTime then
              startTime := max;
           fi;
       od;
   fi;
   SetVertexAttribute(G, v, "ESD" = startTime);
   duration := DurationSum(G, v);
   SetVertexAttribute(G, v, "EFD" = startTime + duration);
od;
end proc:
```

Fig. 3. Cloud Standby Deployment Algorithm

In computer science there are several algorithms for the parallel execution of jobs or tasks. All these algorithms, however, have in common that they assume that the resources are limited. In Cloud computing the assumption is that there are unlimited resources available. Therefor we use in this work a modified version of the MPM net planning algorithm (Thulasiraman & Swamy, 2011) from the discipline of operations research which is listed in Fig 3.

We use the forward calculation of the MPM algorithm to get the Early Start Dates (ESD) and the Early Finish Dates (EFD). The ESD give us an idea when approximately the component will be deployed but it is no guarantee. With the EST however the deployment scheduling can be calculated: components with the same start time can be deployed in parallel. The EFD give an approximation on how long the whole process will take with:

$$t_{Depl} = \max(EFD)$$

4. Cloud Standby Quality Model

Using the Cloud Standby approach on the one hand leads to additional costs but on the other hand increases availability. In order to provide decision support regarding the question whether the introduction of such a Cloud Standby System is useful or not, the states are transferred into a mathematical model. In this chapter we build such a quality model using a graph and Markov chain, based on the UML chart in Fig.1.

In order to facilitate the calculation of quality properties at all, some variables must be defined and parameterized for calculation. Some of the parameters are defined in the use case, or of experimental origin, others are taken from external sources and some can only be estimated. Together with results from previous experiments, average start times can then be calculated. Table 1 represents the time variables to be parameterized as well as the underlying source for its parameterization.

Table 1. Parameters

Type	Variable	Unit
Duration of the initial deployment	t_{depl}	min.
Backup interval	$t_{updateInt}$	min.
RS update time	t_{update}	min.
Duration of the replica deployment	$t_{replDepl}$	min.
Transition from emergency to normal state	t_{error}	min.
Primary Cloud provider costs	$cost_1$	€/h/server
Secondary Cloud provider costs	$cost_2$	€/h/server
Unavailability costs	$cost_e$	€/h
Primary Cloud availability	$avail_1$	years
Secondary Cloud availability	$avail_2$	years

To calculate the total costs, the costs for the run-time of each server must be known. These data can be found in the offers of the Cloud providers. For some evaluations, the costs / loss of profit faced by the company in the case of system unavailability must also be known or at least estimated. All types of costs included in the following analysis are summarized in Table 1. The availability of the Cloud provider is an important basis for the calculation of the overall availability of the system and thus also of the costs.

with the problems of migrating of the data and so on. By using this design we just make sure that distributed systems described with this language are able to be elastic.

Many Cloud providers declare such availability levels in their SLA. However, this availability is less interesting in the context of this calculation because this work focuses on global, long-term outages caused by disasters that cannot be handled by traditional backup techniques. The availability described in the third part of Table 1 indicates the average time period in which exactly one such global outage of the respective Cloud provider is likely to be expected.

Even if elasticity (Mell & Grance, 2011) is a key concept of Cloud Computing and although the prices for Cloud resources constantly changed during the past years, we use static values for the average amount of servers and for the costs over the years. These dynamic aspects could nonetheless easily be added in future work by not having constant prices and servers but functions representing these values. For a first step towards modeling the costs of Cloud-Standby-Systems, however, the use of static values appears acceptable.

4.1 Units.

The states for the state graph that should represent the basis for further calculations can be directly derived from the different states of the UML state chart (Fig. 1). In that regard, S_i corresponds to the description of the state i from the state space I. To calculate the quality properties of the system, stopping times must be assigned to each of the states (see Table 2). It is assumed that the step length of the Markov chain is one minute and the stopping time is $d_i \; \forall \; i \in I$ in a state S_i.

Table 2. Designation of the states from the process steps

Process Step	Model State
PS Deployment	S_1
PS Runtime	S_2
PS Runtime + RS Update	S_3
RS Deployment	S_4
RS Runtime	S_5
RS Runtime + PS Deployment	S_6
Outage	S_7

$$\vec{d} := \begin{bmatrix} t_{depl} \\ t_{updateInt} \\ t_{update} \\ t_{replDepl}(t_{updateInt}) \\ t_{error} - t_{replDepl}(t_{updateInt}) - t_{restore} \\ t_{restore} \\ t_{error} \end{bmatrix}$$

As shown in the definition of the stopping times $\vec{d}$, all times except those of d_2, d_4 and d_5 can be determined from the previously set parameters (Table 1). The update interval $t_{updateInt}$ is part of the configuration and has a major influence on the costs and the availability of the system. The time it takes to start the replica deployment (d_4) strongly depends on when the server has last been updated. Consequently, the start time of the replica is increased by a long update interval. Hence, an increase of the backup interval results in a reduction of the deployment time and accordingly the function $t_{replDepl}(t_{updateInt})$ is increasing monotonically. For d_5 it is assumed that the time t_{error} is constant, regardless of the use of a standby system. The run-time of the standby system is therefore made up of the outage time less replication deployment time (d_4) and the time for the return to the production system (d_6).

4.2 Markov chain and transition graph.

The quality properties of the standby system can be calculated by modeling the states as a Markov chain and a long-term distribution of the stopping time probabilities in the states S. Due to the lack of memory of the Markov chain (Markov property) it is not possible to directly model the stopping times. The stopping times must be transferred into recurrence probabilities. These must be designed so that, on average, in d_i of the cases the state is maintained and in one case the state is left. It follows that the total number of possible cases is $d_i + 1$. Thus, the recurrence probabilities have to be calculated with $\lambda_i \; \forall \; i \in I$:

$$\lambda_i = \frac{d_i}{d_i + 1} \forall \; i \in I$$

In addition to the recurrence probabilities, the probabilities of an outage are required. These are calculated analogously to the recurrence probabilities. On the average, normalized to the iteration step of the Markov chain of one minute, one outage in the period of $avail_i, i \in \{1, 2\}$ should incur:

$$\varepsilon_i = \frac{1}{avail_i * 365 * 24 * 60}, i \in \{1,2\}$$

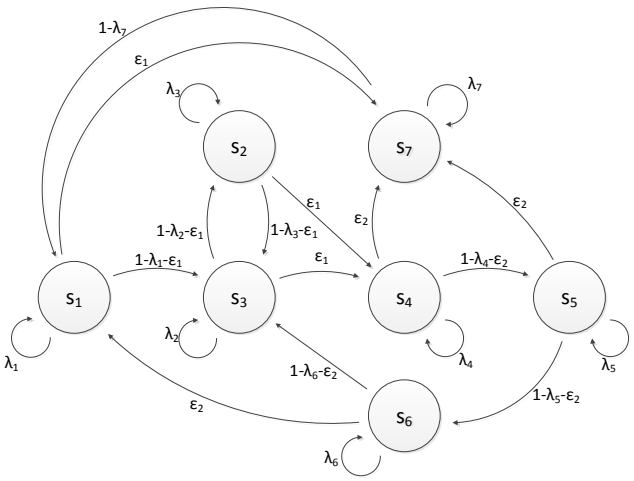

Fig. 4. States of the standby system as a Markov chain (MC_1)

Standby System. Considering these probabilities, the Markov chain MC_1 for the standby system can now be established as shown in Fig 4.

The transition matrix P_1 can be read directly from the Markov chain in Fig. 4:

$$\begin{pmatrix} \lambda_1 & 0 & 1-\lambda_1-\varepsilon_1 & 0 & 0 & \varepsilon_1 & 0 \\ 0 & \lambda_2 & 1-\lambda_2-\varepsilon_1 & \varepsilon_1 & 0 & 0 & 0 \\ 0 & 1-\lambda_3-\varepsilon_1 & \lambda_3 & \varepsilon_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \lambda_4 & 1-\lambda_4-\varepsilon_2 & 0 & \varepsilon_2 \\ 0 & 0 & 0 & 0 & \lambda_5 & 1-\lambda_5-\varepsilon_2 & \varepsilon_2 \\ \varepsilon_2 & 0 & 1-\lambda_6-\varepsilon_2 & 0 & 0 & \lambda_6 & 0 \\ 1-\lambda_7 & 0 & 0 & 0 & 0 & 0 & \lambda_7 \end{pmatrix}$$

No-Standby System. As the properties of the standby system should in the end be compared to the original system, now the Markov chain MC_2 and the transition matrix P_2 must be created as a reference for the system without replication. The two chains only differ in the fact that no update is performed, which means $t_{updateInt} \to \infty$, the stopping time in the states $S_3 - S_6$ are equal to zero and no second provider exists, the probability of outage ε_2 is therefore 1. In case these parameters are applied to MC_1, the states S_5 and S_6 are no longer obtainable. With a probability of 1 the state of S_4 merges directly with S_7 and can thus be combined with S_7.

Due to the fact that the update interval is infinite, the recurrence probability of S_2 is one[3]. This also results in a negative transition probability from S_2 to S_3. However, as the recurrence probability of S_3 is zero, this negative transition probability can be resolved by combining the vertices S_2 and S_3 to S_2. Eventually, this results in a new recurrence probability for S_2 of $1-\varepsilon_1$.

The new Markov chain is therefore MC_2 (as shown in Fig. 5).

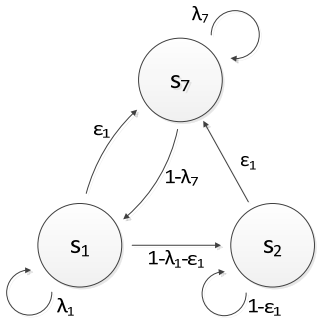

Fig. 5. States of the no-standby system as a Markov chain (MC_2)

The transition matrix P_2 was created similarly to P_1 as a $\mathbb{R}^{7x7}$ matrix, so that the same algorithms are applicable on both matrices. The transitions to and from the states $S_3 - S_6$ have a probability of zero:

$$\begin{pmatrix} \lambda_1 & 1-\lambda_1-\varepsilon_1 & 0 & 0 & 0 & 0 & \varepsilon_1 \\ 0 & 1-\varepsilon_1 & 0 & 0 & 0 & 0 & \varepsilon_1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1-\lambda_7 & 0 & 0 & 0 & 0 & 0 & \lambda_7 \end{pmatrix}$$

4.3 Long-term distribution.

The stationary distribution of a Markov chain MC can be calculated in order to reach a long-term distribution of the system. This distribution $\pi_i, i \in I$ states the probability of the system to be in the state $S_i, i \in I$ at any given time $n \in \mathbb{N}$. With the help of the probability distribution, long-term quality properties such as the cost of γ and the overall availability of α can easily be calculated. The algorithm for determining the stationary distribution is represented in shortened form as follows[4]. In this case E_r is the unit matrix and $\overrightarrow{b_r}$ is the unit vector with the rank r.

$$Q = P - \mathrm{E}_7$$

$$Q' = \begin{pmatrix} q_{1,1} & \cdots & q_{1,7} & 1 \\ \vdots & \ddots & \vdots & \vdots \\ q_{7,1} & \cdots & q_{7,7} & 1 \end{pmatrix}^T$$

The result of the equation system

$$Q' * \vec{\pi} = \overrightarrow{b_7}$$

is the stationary distribution $\vec{\pi}$. This distribution is a vector of which point $\pi_i \in I$ indicates the probability to be in the state S_i at a given step n.

5. Quality Metrics and Decision Support

After defining the stationary distribution $\pi_i, i \in I$, the quality properties of costs and availability can be determined.

5.1 Cost.

The costs c_1 for provider 1 result from the sum of the costs in the states S_1, S_2, S_3, S_6. The costs c_2 incur for provider 2 during the update, in the emergency mode in S_3, S_4, S_5 and the recurrence via state S_6. The costs c_ε for the non-availability of the system incur in the states S_1, S_4, S_7.

[3] $\lim_{t_{updateInt} \to \infty} \lambda(t_{updateInt}) = \lim_{t_{updateInt} \to \infty} \frac{t_{updateInt}}{t_{updateInt}+1} = 1$

[4] A detailed description of the calculation of the stationary distribution is given in [6].

$$\gamma(\vec{\pi}, \vec{c}, n_{server}) := c_1 n_{server} \sum_{i}^{i\in\{1,2,3,6\}} \pi_i + c_2 n_{server} \sum_{i}^{i\in\{3,4,5,6\}} \pi_i + c_\varepsilon \sum_{i}^{i\in\{1,4,7\}} \pi_i$$

5.2 Availability.

The availability results from the sum of the probabilities of the states in which the system is available (S_2, S_3, S_4, S_6) or from the recurrence probability for the states in which the system is unavailable (S_1, S_4, S_7):

$$\alpha(\vec{\pi}) := \sum_{i}^{i\in\{2,3,5,6\}} \pi_i = 1 - \sum_{i}^{i\in\{1,4,7\}} \pi_i$$

5.3 Decision support based on the quality metrics

In case of a decision having to be made whether the standby system should be used in a particular configuration or not, it is useful to compare the quality properties of the different options. Especially during the introduction phase such a direct comparison between the no-standby and the standby system makes sense.

In many cases companies cannot accurately predict certain parameters such as the cost of an outage ($cost_e$) and can only make estimations in a specific interval. Therefore, it is appropriate to make quality properties not only dependent on the update interval, but also on other parameters.

Ratio of outage costs to replication interval. To perform a comparison of the total costs in relation to the outage costs and update interval, the total costs with variable outage costs ($cost_e$) and update interval ($t_{updateInt}$) have to be calculated first. We represent these total costs as:

$$\gamma_i(t_{updateInt}, cost_e) := \gamma(\vec{\pi}_i(t_{updateInt}), \vec{c}(cost_e), n_{server}), \quad i \in \{1,2\}$$

By using these variable cost calculation functions the area in which the two systems have the same cost can be determined. This is achieved by sectioning the function:

$$cost_e(t_{updateInt}) := \gamma_1(t_{updateInt}, cost_e) \cap \gamma_2(t_{updateInt}, cost_e)$$

The function will facilitate the consideration of the limit of value. In this case limits for the update interval are the value of continuous updates and an update interval tending to infinity. Due to the cost structure of the Cloud provider (billing period of an hour) the continuous replication is to be equated with a replication interval of 1 hour or 60 minutes:

$$cost_e^{min} := \lim_{t_{updateInt}\to\infty} cost_e\,(t_{updateInt})$$

$$cost_e^{max} := \lim_{t_{updateInt}\to 60} cost_e\,(t_{updateInt})$$

Outside of the interval $[\mathrm{cost}_e^{\min}, \mathrm{cost}_e^{\max}]$ a Cloud-standby replication such as described in this work doesn't make sense. Should the costs $\mathrm{cost}_e^{\min}$ decrease, the no-standby system is always cheaper and should the update interval be 60 minutes, two systems are operated in parallel. In this case, there would be a direct transition to a hot standby approach because it guarantees an even higher availability.

Ratio of availability to the replication interval. To establish a ratio between availability and replication interval, the availability α is represented as a function that is dependent on $t_{updateInt}$:

$$\alpha_i(t_{updateInt}) := \alpha(\vec{\pi}_i), i \in \{1,2\}$$

This ratio allows a determination of the interval in which the standby system can ensure availability:

$$\alpha_1^{min} := \lim_{t_{updateInt}\to\infty} \alpha_1(t_{updateInt})$$

$$\alpha_1^{max} := \lim_{t_{updateInt}\to 60} \alpha_1(t_{updateInt})$$

The system without replication availability is independent of $t_{updateInt}$:

$$\overline{\alpha}_2 = \alpha_2(t_{updateInt})$$

As the availability function $\alpha_1(t_{updateInt})$ is convex, $\alpha_1^{min} < \alpha_1^{max}$ always applies. Furthermore, it also applies:

$$\overline{\alpha}_2 \leq \alpha_1^{min}$$

This connection which is surprising at first glance can be explained by the fact that in case of an error in the no-standby system it will be directly changed to the state S_7, while in case of a replication the outage time can be bridged by using Cloud provider 2. Only in case of $t_{error} = 0$ applies:

$$\overline{\alpha}_2 = \alpha_1^{min}$$

i.e. for $t_{error} > 0 \vee t_{updateInt} > 60$ applies:

$$\overline{\alpha}_2 < \alpha_1(t_{updateInt})$$

Thus, it can be assumed that from an availability point of view, the outage time $t_{error} > 0$ should definitely be used on a standby system, even if a very large update interval is chosen.

Determining the cost neutral update interval. In order to decide on the length of the replication interval it makes sense to perform a comparison of the systems on a cost basis. It is assumed that outage costs $cost_e$ can be quantified. In order to perform a cost comparison, the two total cost functions are set up:

$$\gamma_{i,cost_e}(t_{updateInt}) = \gamma_i(t_{updateInt}, cost_e), i \in \{1,2\}$$

The maximum and minimum costs for the standby system can easily be determined by considering the limit values:

$$\gamma^{min}_{1,cost_e} := \lim_{t_{updateInt}\to\infty} \gamma_{1,400}(t_{updateInt})$$

$$\gamma^{max}_{1,cost_e} := \lim_{t_{updateInt}\to 60} \gamma_{1,400}(t_{updateInt})$$

The cost neutral update interval can be determined by the intersection of the two cost functions:

$$\overline{t}_{updateInt} := \gamma_{1,cost_e}(t_{updateInt}) \cap \gamma_{2,cost_e}(t_{updateInt})$$

6. USE CASE

In section 3 we motivated that a quality model is needed to evaluate if a Cloud-Standby-System is useful in a given use case. In this chapter we evaluate the model by applying it to a given use case. We demonstrate how the quality model can be applied to a server deployment of 10 servers and given or experimentally determined metrics (see Table 3). It is further illustrated how the administrator of the application can be supported in his decision whether to use Cloud Standby or not.

Table 3. Input Parameter (see Table 1-3) Assumptions

Variable	Value
t_{depl}	60 min.
t_{backup}	30 min.
t_{error}	1440 min.
n_{server}	10
$cost_1$	0,68€/h/server[5]
$cost_2$	0,68€/h/server
$avail_1$	10 years
$avail_2$	10 years

[5] „Extra Large“ Amazon EC2 instance in the availability zone EU-West or performance wise comparable instance on another vendor [5]

For the calculation of the quality properties, it is necessary to determine the time for the deployment of the replica ($t_{replDepl}$). As the time depends on the update frequency, it must be adopted via a function. We assume that 50% of the deployment process is fixed and 50% may be affected by the update interval. The strictly monotonically increasing function should have its lowest point at an update interval of 60 and approach the limit of the time for the initial deployment t_{depl} at infinity (see Fig. 6):

$$t_{replDepl}(t_{updateInt}) = t_{depl}\left(1 - 0{,}5\,\frac{60}{t_{updateInt}}\right), t_{updateInt} \in [60, \infty]$$

This function will in our future work be determined by interpolation of data points from real experiments.

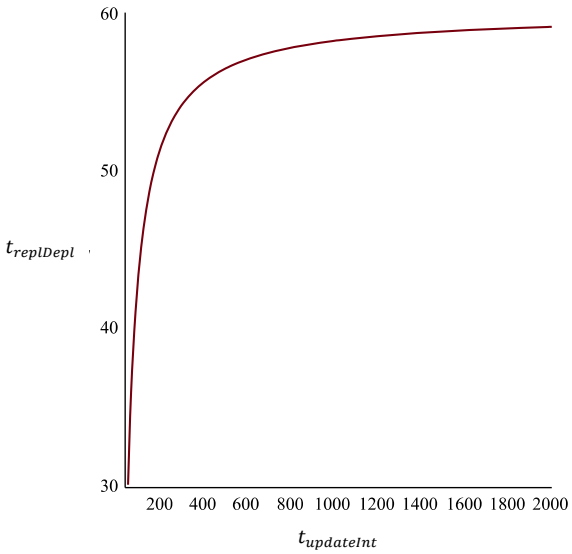

Fig. 6. RS deployment time

6.1 Ratio of outage costs to the replication interval.

With the help of the stationary distributions π_i (see Section 4.3) and the costs in Table 5 the cost functions γ_i can now be defined depending on $t_{\text{updateInt}}$ and cost_e using formula $\gamma_i(t_{updateInt}, cost_e)$ with $i = 1$ (Cloud Standby System) and $i = 2$ (No-Standby-System).

Representing the two functions in a graph (Fig. 7) reveals combinations where γ_1 has lower function values (total costs) and others where γ_2 is lower. The intersection of the functions establishes a curve on which both systems have the same level of costs. This function is represented in Fig. **8**. Besides the combinations leading to the same costs (grey line), the combinations in which the standby system is monetarily inferior to the normal system (grey area) as well as those in which the standby system is cheaper (white area) can be identified.

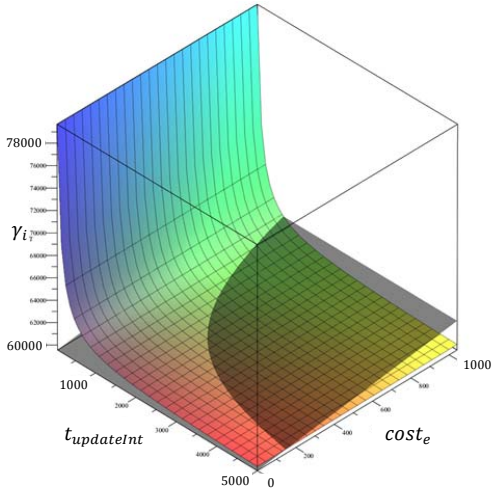

Fig. 7. Comparison of the total costs γ_1 (colored area) and γ_2(grey area) at variable $t_{updateInt}$ and $cost_e$

The limits of the function $cost_e(t_{updateInt})$ result in the interval in which a Cloud-standby approach on the basis of total costs makes sense:

$$cost_e^{min} = \lim_{t\to\infty} cost_e\,(t) = 6.79€/h$$

$$cost_e^{max} = \lim_{t\to 60} cost_e\,(t) = 8198.79€/h$$

In the case of the costs for the outage being lower than the assumed values for server costs, costs for outage times, etc. at more than 8198.79 € per hour, a standby system should be deployed in any case. However, such high costs suggest the approach of a hot standby as two systems can be operated in parallel without any further costs. Given the above-mentioned assumptions, the use of a standby system does not make sense when the outage costs are less than 6.79 € per hour. In this case no matter how large the replication interval is selected, the use of a simple, unsecured system makes more sense from a cost perspective (but not in terms of availability).

6.2 Ratio of availability to the replication interval.

Applying the values from Table 5, the availability functions of α_1 and α_2 can be calculated depending on $t_{updateInt}$.

The overall availability of the system increases noticeably by introducing the standby system. The limit of the function α_1 and the value of $\overline{\alpha_2}$ are:

$$\alpha_1^{min} = \lim_{t_{updateInt}\to\infty} \alpha_1(t_{updateInt}) = 0.9999883$$

$$\alpha_1^{max} = \lim_{t_{updateInt}\to 60} \alpha_1(t_{updateInt}) = 0.9999940$$

$$\overline{\alpha_2} = \alpha_2(t_{updateInt}) = 0.999988201$$

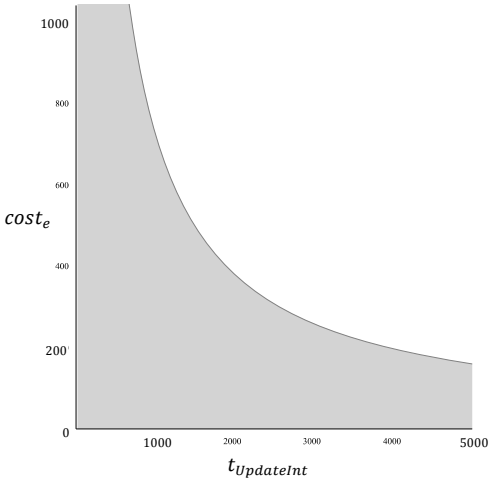

Fig. 8. $t_{updateInt}$ and $cost_e$ combinations in which the standby system is more expensive (grey area), costs the same (grey line) and is cheaper (white area).

Since an outage time of $t_{error} > 0$ was assumed, thus it always makes sense in terms of availability to use the standby system as already presumed.

6.3 Determining the cost neutral update interval.

Now the cost neutral update interval has to be defined, i.e. the time $\overline{t}_{updateInt}$ in which the no-standby system and the standby system produce the same costs. Therefore, it is exemplarily assumed that the outage costs are determined: $cost_e = 400€/\text{h}$. With the help of these outage costs, the new cost functions can be set up now:

$$\gamma_{i,400}(t_{updateInt}) = \gamma_i(t_{updateInt}, 400), i \in \{1,2\}$$

Consideration of the limit value easily depicts the minimal and maximal costs:

$$\gamma_{1,400}^{min} = \lim_{t_{updateInt}\to\infty} \gamma_{1,400}(t_{updateInt}) = 59650.34\,€\,/\,\text{year}$$

$$\gamma_{1,400}^{max} = \lim_{t_{updateInt}\to 60} \gamma_{1,400}(t_{updateInt}) = 99772.07€\,/\,\text{year}$$

The costs for the use of the system without replication can be calculated with the function $\gamma_{2,400}(t_{updateInt})$. These costs are independent of t and thus constant. It is evident that the costs of $\gamma_{1,400}$ are reduced with an increasing update interval and at some point cut with $\gamma_{2,400}$. By calculating the equation

$$\gamma_{1,400}(t_{updateInt}) = \gamma_{2,400}(t_{updateInt})$$

to $t_{updateInt}$, the update interval that can be selected without additional monetary expenses can be determined: $\bar{t}_{updateInt} = 1923.03\ min$.

Considering the outage costs, the system assumed in the example can be made more available without higher costs at an update interval of 1923 minutes, which is a bit less than a daily update (every 1.33 days). The following changes in the availability arise from this: $\alpha_1(1923) - \alpha_2(1923) = 0.000274$. This means that the system in the given use case is within 10 years 1440 minutes or one day more available and consequently the availability class will rise from 3 to 4 with the same costs[6].

7. CONCLUSION

In this work we presented a novel approach for warm standby in the Cloud. Our Cloud Standby approach replicates the modeled primary system periodically to another Cloud provider. The quality attributes of this new Cloud Standby System are formalized by a novel Markov chain based approach. In this paper it was presented that this formal model can be used to calculate the availability and long-term costs of a Cloud Standby System. It was also shown that a Cloud Standby System has an advantage over a no-standby system in matters of availability even if the replication is not even performed once. It was also shown how the model can be used to configure a Cloud Standby System. Since it was proven that a Cloud Standby System provides a higher availability by design, future work is to develop a reference architecture for this kind of systems. Future work might concentrate on the support of the different roles when formalizing the distributed system. It can also concentrate on the introduction of more dynamic parameters regarding provider costs, outage costs, etc. into the model presented herein.

8. REFERENCES

Alhazmi, O. H., & Malaiya, Y. K. (2012). Assessing Disaster Recovery Alternatives: On-site, Colocation or Cloud. In Software Reliability Engineering Workshops (ISSREW), 2012 IEEE 23rd International Symposium on (pp. 19–20). IEEE.

AWS Inc. (2013). Amazon Web Services, Cloud Computing: Compute, Storage, Database. Retrieved May 30, 2013, from https://aws.amazon.com/

Cully, B., Lefebvre, G., Meyer, D., Feeley, M., Hutchinson, N., & Warfield, A. (2008). Remus: High Availability Via Asynchronous Virtual Machine Replication. In Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation (pp. 161–174).

Dantas, J., Matos, R., Araujo, J., & Maciel, P. (2012). An availability model for eucalyptus platform: An analysis of warm-standy replication mechanism. In Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on (pp. 1664–1669). Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6377976

Gilks, W. R., Richardson, S., & Spiegelhalter, D. J. (1996). Markov chain Monte Carlo in practice (Vol. 2). CRC press.

Henderson, C. (2008). Building scalable web sites. O'reilly.

Hiles, A. (2010). The definitive handbook of business continuity management. Wiley.

Klems, M., Tai, S., Shwartz, L., & Grabarnik, G. (2010). Automating the delivery of IT Service Continuity Management through cloud service orchestration. In Network Operations and Management Symposium (NOMS), 2010 IEEE (pp. 65–72).

Konstantinou, A. V., Eilam, T., Kalantar, M., Totok, A. A., Arnold, W., & Snible, E. (2009). An architecture for virtual solution composition and deployment in infrastructure clouds. In Proceedings of the 3rd international workshop on Virtualization technologies in distributed computing (pp. 9–18).

Kurze, T., Klems, M., Bermbach, D., Lenk, A., Tai, S., & Kunze, M. (2011). Cloud federation (pp. 32–38). Presented at the CLOUD COMPUTING 2011, The Second International Conference on Cloud Computing, GRIDs, and Virtualization.

Lenk, A., Klems, M., Nimis, J., Tai, S., & Sandholm, T. (2009). What's inside the Cloud? An architectural map of the Cloud landscape. In Software Engineering Challenges of Cloud Computing, 2009. CLOUD'09. ICSE Workshop on (pp. 23–31).

Lenk, Alexander, & Pallas, F. (2013). Modeling Quality Attributes of Cloud-Standby-Systems. In Service-Oriented and Cloud Computing (pp. 49–63). Springer. Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-40651-5_5

Maximilien, E. M., Ranabahu, A., Engehausen, R., & Anderson, L. C. (2009). Toward cloud-agnostic middlewares. In Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications - OOPSLA '09 (p. 619). Orlando, Florida, USA. doi:10.1145/1639950.1639957

Mell, P., & Grance, T. (2011). The NIST definition of cloud computing. NIST special publication, 800, 145.

Mietzner, R., Unger, T., & Leymann, F. (2009). Cafe: A generic configurable customizable composite cloud application framework. On the Move to Meaningful Internet Systems: OTM 2009, 357–364.

Object Management Group, Inc. (OMG). (2011). Unified Modeling Language (UML), Superstructure Specification Version 2.4.1. Retrieved from http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF

Rajagopalan, S., Cully, B., O'Connor, R., & Warfield, A. (2012). SecondSite: disaster tolerance as a service. In Proceedings of the 8th ACM SIGPLAN/SIGOPS conference on Virtual Execution Environments (pp. 97–108). Retrieved from http://dl.acm.org/citation.cfm?id=2151039

Schmidt, K. (2006). High availability and disaster recovery. Springer.

Symantec. (2011). 2011 SMB Disaster Preparedness Survey - Global Results. Retrieved from http://www.symantec.com/content/en/us/about/media/pdfs/symc_2011_SMB_DP_Survey_Report_Global.pdf?om_ext_cid=biz_socmed_twitter_facebook_marketwire_linkedin_2011Jan_worldwide_dpsurvey

Tanenbaum, A. S., & Van Steen, M. (2002). Distributed systems (Vol. 2). Prentice Hall.

Thulasiraman, K., & Swamy, M. N. (2011). Graphs: theory and algorithms. Wiley-Interscience.

Trieu Chieu, Karve, A., Mohindra, A., & Segal, A. (2010). Simplifying solution deployment on a Cloud through composite appliances. In Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on (pp. 1–5). doi:10.1109/IPDPSW.2010.5470721

[6] The introduction of the Cloud-Standby-System may, however, introduce other costs that are not included herein but are subject to future work.

Wittern, E., Kuhlenkamp, J., & Menzel, M. (2012). Cloud service selection based on variability modeling. In Service-Oriented Computing (pp. 127–141). Springer.

Wood, T., Lagar-Cavilla, H. A., Ramakrishnan, K., Shenoy, P., & Van der Merwe, J. (2011). PipeCloud: using causality to overcome speed-of-light delays in cloud-based disaster recovery. In Proceedings of the 2nd ACM Symposium on Cloud Computing (p. 17).

Wood, Timothy, Cecchet, E., Ramakrishnan, K. K., Shenoy, P., Van der Merwe, J., & Venkataramani, A. (2010). Disaster recovery as a cloud service: Economic benefits & deployment challenges. In 2nd USENIX Workshop on Hot Topics in Cloud Computing. Retrieved from http://www.usenix.org/event/hotcloud10/tech/full_papers/Wood.pdf

Authors

Alexander Lenk is department manager at the FZI Research Center for Information Technology (Berlin office) and researcher at the Karlsruhe Institute of Technology. He has been focusing on Cloud Computing since 2008 with his main research interests in disaster recovery, deployment description, and distributed systems.

Frank Pallas is postdoc researcher at the FZI Research Center for Information Technology (Berlin office) as well as at the Karlsruhe Institute of Technology. His main research areas include the manageability and governability of complex systems like Cloud Computing and Smart Grids. Furthermore, he holds a professorship for data protection and information economics at the TU Berlin.