Introducing Differentiated Availability in Cloud Computing SLAs

Astrid Undheim
Telenor ASA
Corporate Development
Trondheim, Norway
astrid.undheim@telenor.com

Ameen Chilwan and Poul Heegaard
Department of Telematics
Norwegian University of Science and Technology (NTNU)
Trondheim, Norway
chilwan@stud.ntnu.no, poul.heegaard@item.ntnu.no

Abstract—Cloud computing is the new trend in service delivery, and promises cost savings and agility for the customers. However, some challenges still remain to be solved before widespread use can be seen. This is especially relevant for enterprises, which currently lack the necessary assurance for moving their critical data and applications to the cloud. The cloud SLAs are simply not good enough.

This paper focus on the availability attribute of a cloud SLA and argues that differentiated availability is needed for different customers requiring different services levels. Different techniques in use today for increasing the availability for stateless and stateful services in a virtualized system. The resulting availability is quantified using a two-level model, also taking into account the physical deployment of a service.

I. INTRODUCTION

Cloud computing presents a new computing paradigm that has attracted a lot of attention lately. It enables on-demand access to a shared pool of highly scalable computing resources that can be rapidly provisioned and released. This is achieved by offering computing resources and services from large data centers, where the physical resources (servers, network, storage) are virtualized and offered as services over a network. The resources are managed autonomously resulting is cheaper and easier service deployment, both in the private cloud deployment model and using a public cloud provider.

A large part of cloud applications have been targeted to consumers with low willingness to pay, and low expectations to the service QoS (dependability, performance and security). Recently, more and more enterprises are also investigating how to leverage on the cloud computing advantages such as the pay per use model and rapid elasticity [?]. However, major challenges has to be faced in order for enterprises to trust cloud providers with their core IT services. These challenges are mainly related to QoS, in our view covering aspects such as dependability, performance and security, and a comprehensive Service Level Agreement (SLA) is needed to cover all aspect of QoS. This is in contrast to the insufficient SLAs offered today. In this paper, we focus on dependability, and more specifically the availability attribute. Availability is defined in [?] as "the readiness for correct service", i.e., the probability of providing service according to the requirements.

In order to avoid costly down times contributing to service unavailability, fault avoidance and fault tolerance are used in the design of dependable systems. Traditionally, fault tolerance have been implemented in hardware resulting in expensive systems, or using cluster software, often very specific for each application [?]. In cloud computing, the approach to fault-tolerance has mainly been to use cheap, off-the shelf hardware, allowing failures and then tolerating these in software. The reason for this is partly the large size of cloud data centers, which means that hardware will fail constantly. Adding additional hardware resources to account for failures and let the failover be handled by software is then a more cost-effective approach than using special-built hardware. An other reason for using off-the shelf hardware is that with virtualization every virtual instances can be migrated to arbitrary physical machines, sharing redundant capacity between a large number of VMs. The standby resources needed are thus much less than for a standard system. However, also with virtualization, fault tolerance means adding additional resources in the system, which adds cost, fault-tolerance should therefore be targeted to specific needs.

Different types of applications with different requirements as well as different users with different requirements will co-exist in the cloud. Differentiating the fault-tolerance techniques and deployments for different applications hence give better resource utilization, being cost effective for the provider while still delivering service according to user expectations. This is particular important when deploying stateless and stateful applications, where the latter require synchronized/updated replicas for tolerating failures, while stateless applications that tolerate short downtimes can be implemented using non-updated replicas. In addition, the physical deployment location of replicas affect the fault-tolerance, where some failures may affect a small or larger part of a cloud data center, or even several data centers at different locations.

This paper focuses on availability differentiation for cloud services, where the availability of an application running in a Virtual Machine (VM) is considered. The goal and the objective is to motivate for more comprehensive SLAs, where different availability levels can be chosen by the user for a given cost, and where this cost is directly reflected in the deployment costs for the cloud provider.

The rest of this paper is organized as follows. In Section ??, the SLAs offered by commercial cloud providers today
and what are the missing pieces are described, together with some new proposals for autonomic SLA enforcement in virtual infrastructures. In Section ??, principles for achieving fault-tolerance are described, as well as its application in cloud computing. A high level model of a cloud service deployment is described in Section ??, and the different types of failures are described. In Section ??, we introduce the different approaches to fault-tolerance for cloud applications using Markov models. Numerical results are presented in Section ??M. Finally, we conclude the paper in Section ??, together with some thoughts on future work.

II. SLAs in Cloud Computing

Cloud computing gives the customers less control of the service delivery, and they need to take precautions in order not to suffer low performance, long downtimes or loss of critical data. Service Level Agreements (SLAs) have therefore become an important part of the cloud service delivery model. An SLA is a binding agreement between the service provider and the service customer, used to specify the level of service to be delivered as well as how measuring, reporting and violation handling should be done. Today, most of the major cloud service providers include QoS guarantees in their SLA proposals, specified in Service Level Specification (SLS), as seen in Figure ??M. The focus in most cases is on dependability, measured as service availability usually covering a time period of a month or a whole year. Credits are issued if the SLA is broken, e.g. the Amazon EC2 SLA include an “Annual uptime percentage” of 99.99% and issues 10% service credits 1.

Fig. 1. The Structure of an SLA

Recently, we have seen many examples where cloud services have been unavailable for the customer, but where the unavailability has no been covered by the SLA 2. One reason is that the cloud SLAs are not specific enough in defining KPIs for availability, e.g., excluding performance guarantees. From the customers point of view this is a major drawback. Performance (e.g. response time) above a certain threshold will be perceived by the customer as service unavailability and should be credited accordingly. This issue is covered in [?], where the throughput of a load-balanced application is studied under the events of failures. It is clear that the availability parameter alone is not enough to ensure a satisfactory service delivery.

In addition to the traditional QoS issues, security has been stated as one of the largest threats to widespread use of cloud computing [?]. Security breaches may lead to service downtime and thus contribute to the unavailability of a service. An approach for adding security in cloud computing SLAs is described in [?], where focus is on general security mechanisms as well as cloud specific security mechanisms.

The on-demand characteristic of cloud computing is one aspect that complicates the QoS provisioning and SLA management. The cloud infrastructure needs to adjust to changing user demands, resource conditions and environmental issues. Hence, the cloud management system needs to autonomically allocate resources to match the SLAs and also detect possible violations and take appropriate action in order to avoid paying credits. Several challenges for autonomic SLA management still remain. First, resources need to be allocated according to a given SLA. Next, measurements and monitoring are needed to detect possible violations and react accordingly, e.g., by allocating more resources. For availability violations this may require adding more standby resources to handle a given number of failures, and for performance violations this may require moving a VM to an other physical machine if the current machine is overloaded and results in high response times. All these actions require a mapping between low-level resource metrics and high-level SLA parameters. One proposal on how to do this mapping is given in [?], where the amount of allocated resources are adjusted on the fly to avoid an SLA violation. An other proposal for dynamic resource allocation using a feedback control system is proposed in [?]. Here, the allocation of physical resources (e.g. physical CPU, memory and input/output) to VMs is adjusted based on measured performance.

With deployment of widely different services in the cloud, there is clearly a need for cloud providers to offer differentiated SLAs, both with respect to dependability, performance and security. Examples are core business functions such as production systems and billing versus applications targeted to consumers such as email and document handling. Also, different user groups may have different requirements. One example is Gmail, where the SLA for email services for consumers and business users are differentiated, offering the business users an availability of 99.9% at a fixed price, while consumers have a free offering without any SLA 3. Differentiated QoS and availability in particular should be formalize in SLAs, quantifying the different levels, and offering credits if this is not achieved. The rest of this paper aim at quantifying the availability provided by some cloud fault-tolerance techniques, and how differentiation can then be achieved.

III. Fault Tolerance in Cloud Computing

In the design of dependable systems, a combination of fault avoidance (called fault prevention in [?]) and fault tolerance is

1http://aws.amazon.com/ec2-sla/
used to increase availability. Fault avoidance aims at avoiding faults being introduced, through use of better components (i.e. SSD instead of HDD), debugging of software or protecting the system against environmental faults. Fault tolerance is often used in addition to fault avoidance, allowing a fault leading to error but preventing errors leading to service failure. Fault tolerance thus use redundancy in order to remove or compensate for errors. This section gives a short overview of general fault tolerance techniques used in design of dependable communication systems today, and then looks at how fault tolerance is achieved in virtualized environments.

A. Fault Tolerance Principles

Cloud infrastructure is built using off-the shelf hardware, and standby redundancy is the preferred fault tolerance technique. In this case, there are two or more replicas of the system (VMs in our case). Only the active replica will produce result to be presented to the receiver, while the standby replicas are ready to take over should the active replica fail. Standby redundancy can be further classified as hot or cold standby. Hot standbys are powered standbys, capable of taking over service execution with no downtime (as long as the state is updated). Cold standbys are non-powered and need some time to be started in case of failure in the active replica, but also has a lower failure rate than a hot standby.

Different levels of synchronization are possible for the hot standbys, and the backup resources can be dedicated or shared, for both the hot and cold standbys. This gives the overall classification as shown in Figure ??.

![Fig. 2. Standby Redundancy Classification](image-url)

The choice between hot or cold standbys will decide the service restoration time, but will also depend on the applications need of an updated state space, as described in the next section.

B. Fault Tolerance in Cloud Computing

Cloud computing uses virtualization of computing resources made available as VMs, virtual storage and virtual networks. We concentrate here on computation services and the use of VMs. In this case, full image backup is made easy with virtualization, since the virtual image contains everything that is needed to run the application and can be transparently migrated between physical machines. One of the downsides of virtualization though is that one single hardware fault in a physical server can affect several VMs and hence many applications. Replicas of the same VMs need to therefore always be deployed on different physical machines. The standby resources must also be dimensioned to handle the high number of failed VMs in case of a physical server failure. In addition, mechanism for prioritized restart of failed applications must be in place, allowing for differentiation of important applications in case of failures.

Virtualization facilitates live migration of VMs, where a running VM instance can be transferred between physical machines. Live migration has been implemented both for the Xen hypervisor [?] and for VMware with its VMotion [?] and ensures zero downtime in case of planned migrations due to resource optimization or planned maintenance. In case of hardware failures in the physical host running the VM, live migration is not possible. The configuration file for the VM should then be available on possible new host machines in order to restart the application, and heartbeats are used to detect failure in either active or standby replica. In addition, the configuration file should be stored at a centralized location should both replicas fail.

1) Hot Standby: For stateful applications, state must be stored on the standby virtual machine in order to allow failover. In traditional fault-tolerance terminology, this requires the use of updated hot standbys. Non-updated hot standbys are also required for applications which require a short restoration time. Different levels of updating/synchronization between the active and standby replica is possible, either the input is evaluated at each replica, or the state information is transferred at specified checkpoints. The former method will then consume more compute resources than the latter, and is denoted dedicated in our classification (Figure ??). The latter method allows many replicas to share backup resources and is hence denoted shared. Hot standbys can be used for both stateless and stateful applications, but since all replicas consumes resources it is most often used for stateful applications.

Examples of hot standby techniques are VMware’s Fault Tolerance [?] and Remus for the Xen hypervisor [?] as seen in Figure ??, VMware Fault Tolerance is designed for mission-critical workloads, using a technique called virtual Lockstep, and ensure no data or state loss, including all active network connections etc. Both active and standby replicas execute all instructions, but the output from the standby replica is suppressed by the hypervisor. The hypervisor thus hides the complexity from both the application and the underlying hardware. This scheme is classified as updated and dedicated since the standby replica is fully synchronized and consumes resources equal to the active replica.

In Remus, fault tolerance is achieved by transmitting state information to the standby VM at frequent checkpoints, and buffering intermediate inputs. The standby can hence be up and running with a complete state space in case of failures, with only a short downtime needed for catchup of the input buffer. The standby is not executing any inputs, which means that less resources are consumed compared to VMware Fault
Tolerance, and a short downtime and loss of ongoing transaction is experienced in case of failure of the active replica. This scheme is classified as updated and shared since the standby only consume a small amount of resources compared to the active.

The not updated hot standby is also a possibility, which will ensure a lower down time compared to cold standbys for stateless applications, however we don’t have any concrete implementation examples for this option. [Sjekk dette mer grundig]

2) Cold Standby: The cold standby solution requires less resources and should in general be used for stateless applications that allows short downtimes. The same is true in cloud computing. But in addition, functionality is added in a virtualized environment that is valuable for fault tolerance. Virtualization facilitate the running of different VMs on top of the same hardware and the standby resources can be shared by different VMs, reducing the total resource needs. Dedicated standby resources are still possible for cold standbys, and should be used for stateless applications with high availability requirements. In practice, the dedicated solution can be implemented by prioritizing the restart of a standby VM in case of a failure. The low priority VMs may then experience a longer down time, and possible migration to a different part of the cloud.

VMware High Availability (HA) is one example of the use of cold standbys and supports both dedicated and shared resource usage, i.e., by allowing for different priority levels when restarting failed VMs. The availability of applications using VMware HA is studied in [?].

With this simple classification, we end up with four different service levels as seen in Figure ??.

![Different classes of service with respect to the availability requirements and the stateful service requirement](image)

Next, we investigate the difference in the resulting availability from implementing different standby configurations in a virtualized system. These principles can lay the foundation for offering differentiated availability levels in cloud SLAs.

The dependability differentiation problem is studied in two dimensions, first looking at the physical deployment of the standby resources and next using different standby redundancy schemes. The goal is to quantify the difference in availability along these two dimensions.

IV. HIGH LEVEL MODEL

We model the cloud infrastructure in different layers. First, each cloud provider will typically have two or more data centers, physically dislocated. Each data center is then organized into rack of servers. We chose here to see a collection of servers (one or more racks) as a cluster. What defines a cluster is then a collection of physical resources, sharing some infrastructure elements such as power distribution and network switches.

VMware denote a cluster a collection of resources presenting a pool of resources and where failover can be made transparently. Organizing resources into cluster is thus common for fault-tolerance purposes. However, a cluster does not need to be defined in a specific physical location. Resources of one cluster may even belong to different data centers. It is worth noting that VMs operating in one cluster will be vulnerable to single points of failures from the cluster management software.

In order to deploy fault-tolerant applications in the cloud, we look at different deployment options and the resulting dependability when replicas are located in the same cluster, in different clusters of a data center or even in different data center. The latter two deployment options provide tolerance also towards power and network failures.

![High Level Cloud Environment](image)

A. Failure Classification

From the high level model, we focus on four different types of failures, namely failures in the power distribution/cooling, network failures, management software failures and server failures. These are described next.

1) Power Failures: An overview of power distribution in data centers can be found in [?]. The general power supply to the data center is from the utility power supply (UPS), adding generators and battery for backup. In case of failure in the UPS, the battery will take over while the generator is starting up. We can not assume perfect failover and a Markov model
is needed to model the complexity of the power supply. Since the power supply is not the main focus in this paper, we use availability numbers from [insert reference here].

Within the data center, clusters will be connected to a (duplicated) Power Distribution Unit (PDU) which is connected to the central power supply over a power bus. A failure in the distribution system is assumed to only affect one cluster. These failures are independent from failures in the power supply and the two parts can be modeled in a series structure as seen in Figure ??.

![Environmental Model](image)

**Fig. 5. Environmental Model**

2) **Network Failures**: The cloud services are accessed over the Internet, and the high level can be seen in Figure ??.

In addition, we model the data center internal network in two levels [?]. First there is one (duplicated) level 1 switch connecting all servers in one cluster. Next, there is one (duplicated) level 2 switch connecting all level 1 switches from all clusters. These are again connected to the WAN gateways of which there is also two since we assume the cloud provider to be multi-homed to two independent ISPs. The resulting structure model, including the core Internet and the user access network is then seen in Figure ??.

![Network Model](image)

**Fig. 6. Network Model**

3) **Software Failures**: Cloud computing requires extensive management systems, and these are exposed for failures. Depending on what level of management software these failures affect, a cluster (VM Management), the whole data center (Virtual Infrastructure Management) or the whole cloud (Cloud Management) can be affected. The resulting model is shown in Figure ??.

![Management Software Model](image)

**Fig. 7. Management software model**

4) **Server Failures**: The server failures include both hardware and software failures that take down a service running on the physical server. This part is modeled in detail dependent on the fault tolerance techniques used, and are discussed in details in Section ??.

**B. Location of Replicas**

For the high level model described in the previous section we identified four main failure types, i.e. management software, power failures, network failures and server failures (including hardware and software). These failures are independent which means that reliability block diagrams can be used on the high level, and where individual block (here the power block and the server block) is detailed using Markov models.

1) **Same Cluster**: The easiest deployment is to place replicas in the same cluster. This means low network latency in upgrading replicas etc., but it also means that both power, management software and network failures may lead to unavailability of both replicas. The resulting model is shown in Figure ??.

![Deployment in the same cluster](image)

**Fig. 8. Deployment in the same cluster**

2) **Same Data Center**: Next, replicas are placed in two different clusters, but in the same data center. The block diagram will then incorporate the cluster part of the management software, power and network blocks in the cluster block, together with the server block. The resulting model is shown in Figure ??.

![Deployment in the same data center](image)

**Fig. 9. Deployment in the same data center**

3) **Same Cloud Provider**: The final option is to deploy replicas in two different data centers. The DC block will then include the whole power block, as well as the cluster and data center part of the network and management software blocks. The resulting model is shown in Figure ??.

**V. SERVER MODELS**

For the server models, we look at the physical failures. We chose to study the schemes that are currently deployed.
in commercial products, i.e., the VMware FT, Remus, and VMware HA with two different priority levels. These models are described in details in Section ?? - ??.

A. Hot Standby, Dedicated, Updated

The hot standby option with dedicated, updated standbys provides the highest availability and the most updated state, corresponding to VMware FT scheme [?]. We model two identical VMs running on two different physical servers, controlled by some common management software for making migration decisions. The replicas receive the same input and performs the same operations, but only the active VM delivers services. In case of a failure in the active replica, the hypervisor will immediately detect the failure and switch to the standby replica which is ready to perform service without any delay or loss of data. The management software will then deploy a new standby VM. With the failure of the standby VM, the management software will likewise deploy a new standby VM. This means that in a dependability context, which of the VMs fail does not affect the downtime, and can be excluded from the model. This setup will always tolerate one failure, however, it may happen that the resources are exhausted when trying to deploy a new standby VM, in which case the service will fail with the next failure. The resulting model is shown in Figure ???. Here, λ is the hardware failure rate of the server, μ is the restart rate of a new VM, and c is the coverage factor, i.e., the probability that a restart is successful. We assume here that the resources are dimensioned so that there will always be enough resources for restarting a hot standby, since these should host the highest priority applications.

B. Hot Standby, Shared, Updated

The hot standby option with shared updated standbys is different from the dedicated option in that the state information is transmitted at regular intervals instead of running the replicas in a synchronized fashion. This scheme corresponds to the Remus scheme for Xen [?], and as seen above this scheme experience a short downtime and loss of data in case of failures. This also means that it matters which replica that fails, since failure of the active replica will cause a short downtime. The model will therefore be different than for the dedicated standby.

Since the standbys share the backup resources, there will be a non-zero probability that the standby will not have enough resources to start in case of failure of the primary. We assume here that the overall load on the cluster is low such that there will always be enough resources.

The resulting model is shown in Figure ???, where the parameters are the same as for the hot, dedicated model. However, one additional parameter is introduced, α which is the time needed to switch to the standby replica in case of failure in the active replica. It is then clear that when this time is short enough, this model will be equal to the previous model.

C. Cold Standby

For the cold standby setup, no state information is retained in the standby and the standby is simply restarted in case of a failure, corresponding to VMware HA [?] hence heartbeats are used to detect failures in the active replica and restart the VM in a new physical location. This detection usually takes some seconds or up to minutes, in which time the service is not available. Also, the backup resources may be shared between different VMs, and are usually dimensioned to allow for a specific number of physical server failures. Hence, it may happen that some VMs can not be restarted in case of a failure in the active replica.

Here we look at two different classes, both with shared backup resources, but where the high priority class has preemptive priority over the low priority class. Given that the resources are properly dimensioned, the high priority class will experience having dedicated standby resources.

The resulting models are then shown in Figure ???. The additional parameter δ is then the preemption rate from a higher priority application, and we introduce the parameter p which is the probability that there are enough resources for restarting. This is different from the previous models since we can no longer assume that the resources are dimensioned to handle these restarts for the lowest priority applications.

VI. RESULTS

Parameters, insert table with the most important. One table for the failure/repair times for the Markov models and one
table with the availability parameters for the reliability blocks. Discuss the value of the parameters.

What types of results are needed.
• The availability depending on the load in the system.
  Compare for the different fault-tolerance scenarios and with the same deployment option.
  – Specifically for the cold standby, when is the high priority cold standby solution superior?
• How does the choice on the “best” location get affected by the failure rates of network, power, software etc.
• More

Include a discussion of the results.

VII. CONCLUSIONS AND FUTURE WORK

SLAs have received a lot of attention with the cloud computing, and especially availability is covered by public cloud SLAs. However, there are some important improvements to be made. First, the SLAs must become more detailed with respect to actual KPIs used to define availability. Next, in order to deploy also important company services in clouds, different levels of availability should be offered, depending on the actual user requirements. The final point is then that the SLAs should be available on demand, and be adjusted according to needs.

Future work:
• Tiered application
• Software model
• Performance

REFERENCES