Resiliency techniques for cloud-based applications

Szilárd Bozóki
Department of Measurement and Information Systems
Budapest University of Technology and Economics
Budapest, Hungary
Email: bozoki@mit.bme.hu

András Pataricza
Department of Measurement and Information Systems
Budapest University of Technology and Economics
Budapest, Hungary
Email: pataric@mit.bme.hu

Abstract— Resilience of runtime environments provided for cloud based systems is a prerequisite of deploying mission critical applications. It requires a tradeoff between the applied redundancy and its dependability increase. Offering cheap, on-demand redundancy is one of the most attractive benefits of cloud computing which makes even high redundancy based solutions affordable. The current paper revisits traditional fault tolerant design patterns form the cloud perspective.

Keywords— Resiliency, redundancy, fault tolerance, cloud, design patterns

I. INTRODUCTION

Migration to the cloud became an appealing alternative for manifold use cases and applications due to the prodigious development of cloud computing. Even mission critical and SLA (*service level agreement*) bound applications started to migrate from proprietary dedicated servers to commercial cloud services with no special platform SLA guarantees because of the huge economic benefits. For instance, telco solution providers, traditionally confined by strict carrier grade requirements, are yet to embrace the benefits of cloud computing through Network Function Virtualization [1].

Cloud computing has unique features that generally affect resiliency. For instance, offering cheap, on-demand redundancy is one of the most attractive benefits of cloud computing. In order to utilize the true potential of cloud computing based runtime environments the age old and proven fault tolerant design patterns need fundamental rethinking.

The logic interface (*cloud stack*) hides the majority of details of the underlying platform. However, assurance of user level SLAs is highly dependent on the underlying platform, thus a measurement and calculation methodology of platform KPIs (*key performance indicator*, such as performance profiles of the virtual network and computing power etc.) is imperative.

These KPIs can guide the architecture design phase of system development from the extra functional requirements point of view, like design for dependability.

The practice of design for fault tolerance searches for a proper tradeoff between the assurance of the designated dependability of the target application and the used redundancy.

Currently there are three known main aspects to revisit:

1. What are the *required level* and *appropriate structure of redundancy* (e.g. a spare pool of VMs) guaranteeing the continuous fulfilment of SLA requirements?
2. What is the proper way to adopt these solutions to different cloud platforms in order to cost efficiently match an application level SLA?
3. What is the proper characterization of the designated user service in order to efficiently fit it to the fault tolerance design pattern?

The current paper tries to answer these questions from cloud user point of view by examining what are the pros and cons of the different resiliency techniques, with focus on downtime and cost.

The rest of the paper is organized as follows. Section 2 introduces the main factors influencing design for resiliency. Section 3 revisits fault tolerant design patterns for architecture design. Section 4 presents the analysis methodology of design patterns in more detail. Section 5 concludes the paper by presenting an outlook on future work.

II. FOUNDATIONS OF CLOUD COMPUTING

A. Cloud computing

NIST defines cloud computing as follows [2]:

"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models."

1) Essential characteristics of cloud computing [2]

- *On-demand self-service* and *rapid elasticity* facilitate active resource management by the user and the applied fault tolerant mechanisms.
- *Broad network access* led to the integration of geographically remote resources into a single virtual platform. However, fault tolerance necessitates to take into account the distributed nature of cloud computing among others by the vulnerability of the interconnection fabric.
- *Resource pooling,* a.k.a. resource sharing, is the fundamental mechanism for an excellent utilization of resources, however, it opens new surfaces for error and intrusion propagation, even between functionally independent applications of different owners. The quality of the provisioned resources varies depending on tenant activity.

(e.g through noisy neighbors) This variance places an extra layer of complexity on resiliency.

- *Measured service* and pricing models of the providers play a vital role in the exact quantification of the cost of resiliency.

2) Service models of cloud computing: [2]

The idea of the *service stack* needs to be summarized, before explaining the service models.

The service stack contains the layers and components which cooperate in a service provisioning application.

Generally, the following main layers of a service stack can be identified from bottom up (Fig. 1): hardware, virtual machine monitor (VMM, aka hypervisor in many cases) usually residing in a host operating system, virtual machine (VM), guest operating system, application container and the application.

The three primary service models of cloud computing, namely *Software as a Service* (SaaS), *Platform as a Service* (PaaS) and *Infrastructure as a Service* (IaaS), are based on the cutting point of the service stack at which provisioning is offered to the user. The subpart below the cutting point is the cloud domain solely under control of the cloud provider, whilst from the interface point on the service belongs to the cloud user. Fig. 1 depicts the relation between the service stack and the different service models, as well.

The service model has a huge impact on the responsibilities and the available possibilities cloud users have regarding fault tolerance, as he has no influence on the services below the cutting point, if it is a control domain of a separate provider.

3) Deployment Models of cloud computing: [2]

The deployment models define the nature of the cloud provider. Self-operating a cloud (*Private cloud*) or part of it (*Hybrid cloud*) gives insight and control into the cloud providers domain, thus enables fault tolerant activities in a larger part of the service stack. On the other hand, cloud environments operated by an organization (*Public cloud*) or a group of organizations (*Community clouds*) lack of such extended observability and controllability into the underlying layers.

B. Software-Defined Network [3]

In essence, *Software-Defined Network* (SDN) relies on two core ideas: separation of the control and data planes of the data transmission, and abstracting network control into application programming interfaces. These two concepts allow higher level programming of networks and give greater flexibility over traditional networking solutions.

SDN greatly enhances the agility of network management. It fundamentally complements the designated cloud computing characteristics of on-demand self-service and rapid elasticity, providing even greater freedom in resource configuration.

C. Scope of the paper

The IaaS service model deployed on a public cloud is the primary scope of this paper. This means that a cloud provider is solely responsible for maintaining the physical infrastructure hidden from the users, who submit requests regarding the resources.

Network virtualization [4] allows cloud users to run their own *network operating systems*, offering full control over the virtual networks they define and use. Simultaneously, cloud providers can hide the peculiarities of their physical infrastructure by mapping it to virtual networks.

III. Fault Tolerant Design Patterns

Software design patterns extract the reusable abstract design information of proven solutions to recurring common problems within a context. They also highlight the reasons why the chosen solution is appropriate.

Patterns for fault tolerant software target improved *resiliency*. They offer general purpose solutions applicable at different levels of the architecture to ensure undisrupted processing. [5]

A. Physical cloud layout

Cloud resources are usually hosted in multiple locations named *regions* providing geographic diversity. [6] *Availability zones* are physically isolated locations within regions interconnected through low-latency links. Inter-region communication is done via Internet, implying higher cost, lower security and quality.

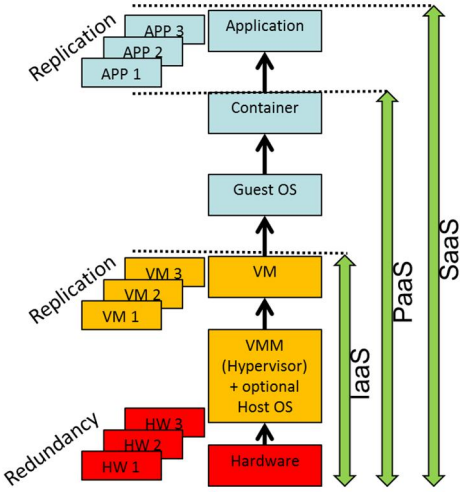

Fig. 1. Service stack, cloud service models and alternatives for the redundancy architectural pattern

The *redundancy architectural pattern* relies on the use of multiple separate copies of a resource in order to ensure fault tolerance. It can be applied at several layers as depicted in Fig. 1. For instance, hardware redundancy mostly resides in the domain of the cloud provider. In contrast to this, the replication of VMs relying on the same design pattern is within the domain of both the cloud user and cloud provider.

B. Phases of fault tolerance

Fault tolerance has four phases: *error detection*, *error recovery*, *error mitigation* and *fault treatment*.

- Error detection aims at finding errors within a system state.
- Error recovery aims at returning an erroneous system to an error free state by applying correcting actions that eliminate the error.
- Error mitigation aims at adapting a system to maintain operation in the presence of errors.

- Fault treatment aims at fixing the system by removing the faults, the causes of errors.

Patterns for fault tolerant software are grouped into five categories. Four groups are aligned along the four phases of fault tolerance, while the fifth group is based on architectural design.

C. IaaS level fault tolerance patterns

A few examples are given in this short section to showcase the high level applicability of fault tolerant patterns.

VMs are among the primary units of resources in IaaS, therefore they represent a natural candidate for the role of unit of fault tolerance. (*Units of Mitigation architectural pattern*)

Operating multiple VMs concurrently (*Redundancy architectural pattern*) and switching between them (*Failover error recovery pattern*) is also an obvious idea. Moreover, an example of the multiple applicability of the redundancy architectural pattern was already depicted in Fig. 1.

Storing information independent from the VMs is a regular cloud service, thus the *Remote Storage error recovery pattern* is also a natural candidate.

IV. Redundancy and Failover Patterns

Fault tolerance aims at the maintenance of the continuous operation of a service despite of faults. Failover assures fault masking by substituting a faulty resource with a fault free backup one. This substitution should be seamless and invisible to the services utilizing the corresponding resource.

Failover can be realized in several ways: resource reallocation, task migration or the re-arrangement of the system topology.

For instance, an external entity could act as a *proxy* [7] (load-balancer [8]), initiating a failover by task reallocation upon detecting an error. However, fast network reconfiguration facilitated by SDN is a new candidate for fault tolerance. In this case the network can execute the failover by rearranging the network routing topology. A similar idea, named *network topology transparency*, could be found within a draft of Network Function Virtualization. [9]

The tradeoff between the different redundancy and failover solutions should be resolved using a cost analysis and the loss function associated with downtime (service outage).

A. Availability considerations

Downtime (service outage) is the duration when the service is unavailable. Fig 3 depicts the downtime to be the duration of repair, the time between the crash of the primary virtual machine and the activation of the secondary virtual machine.

The execution time of resource reconfiguration becomes extremely important in systems based on dynamic resource and redundancy allocation. Although on-demand self-service and rapid elasticity are essential features of cloud computing, the level of agility, the speed of reconfiguration (scaling) is far from deterministic. For instance, typical virtual machine startup may vary between 100 and 800 seconds [10]. Moreover, a more detailed study [11] aimed at modeling the overhead of launching virtual machines concluded that the total overhead has an even larger variation.

The reconfiguration time of SDN nodes also have huge variation. The installation of a new *forwarding rule* generally takes between 2 and 12 milliseconds, but the overall duration can reach the order of seconds. This can occur due to the latency of *packet in* and *packet out* messages in case of a saturated control plane. [12]

The cumulative effect of these variation factors may lead in worst case to critical degradations in service due to the delayed failover actions despite of the availability of a sufficient redundancy.

1) Stateless failover

Stateless failover does not need transmission of any state information from the failed to the backup node, thus the main contributors to downtime are network reconfiguration and activation of the backup node. The use of hot (active) backups eliminates the latter one.

2) Stateful failover

Stateful failover necessitates the transmission of state information from the primary to the backup node. This state transfer significantly affects downtime. Standard virtualization has a significantly varying impact on network, thus on state transfer performance [13]. State information can be transferred directly between VMs or indirectly by means of an intermediate shared resource.

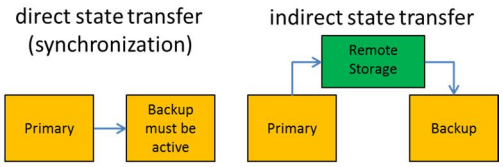

Fig. 2. State transfer alternatives

Direct state transfer (synchronization) requires active backup VMs, thus operates with a continuous overhead of running redundant nodes. (Fig. 2) This solution eliminates state transfer during a failover.

Indirect state transfer requires a mediating entity, usually a remote storage (*Remote storage error recovery pattern*, Fig. 2). This solution does not require an active backup, thus avoids the ongoing expense of the active backup nodes, but the duration of state transfer is significant. Naturally, any combination of the methods is feasible.

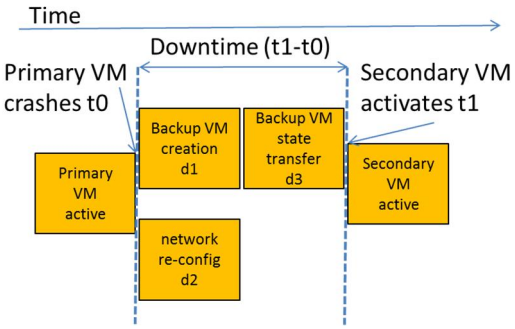

Fig. 3. Downtime and reconfiguration durations

d1: backup creation duration
d2: network reconfiguration duration
d3: state transfer duration

Fig. 3 presents the relation between *downtime* and the different *reconfiguration actions*. Backup VM creation and state transfer run sequentially, whilst network re-configuration runs in parallel with them. With an active backup d1 becomes 0, without state transfer d3 becomes 0 too. (Table I) The total recovery time is heavily application dependent and the best mechanisms are subject of further research.

TABLE I. DOWNTIME EVAULATION OF REDUNDANCY MODES

Redundancy / Mode	Downtime
Stateless / none	Max (d1,d2)
Stateless / 1 active	d2
Stateful (synchronizing) / 1 active	d2
Stateful (synchronizing via remote storage) / 1 active	d2
Stateful (remote storage) / none	Max (d1+d3,d2)

3) State to be transferred

State can be transferred many ways with different limitations: application data, the application, and the virtual machine just to name a few. For example, *offline migration* of VMs is possible among different architectures, while *live migration* of VMs is not, as transferring of memory pages requires binary compatibility.

B. Pricing model

The efficiency of redundancy mechanisms increases with the independence of the primary and backup resources. For instance, a backup deployed in another region is insensitive to faults affecting the region of the primary one. However, state storage and transfer between primary and backup nodes may differ essentially depending on the resource allocation from the cost point of view.

An initial investigation of the basic features of the IaaS pricing model is presented in this section, because a low *total cost of ownership* (TCO) of redundancy is uttermost desirable.

The *pay per use* pricing model is generally used, but with different pricing policies by the different cloud providers. The pricing model of Amazon EC2 [14], an industry leader, is selected as a reference.

Basically, Amazon EC2 has four categories of virtual machine setups: general purpose, compute optimized, memory optimized, storage optimized. The costs of the resources scale linearly within each category. However, the relative costs of resources between categories differ.

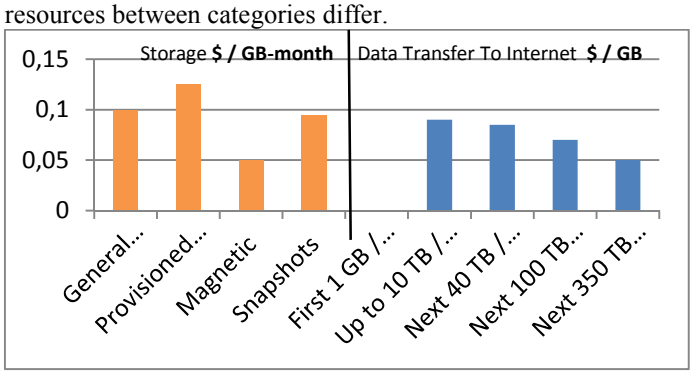

Fig. 4. Storage and data transfer cost comparison

Data transfer within the same availability zone is either free or low cost; inter availability zone data transfer has moderate cost, while external data transfer to the Internet, including another cloud region, is more expensive.

Remote storage pricing is based on the total amount of stored data per month. However, provisioned storage costs extra for the guaranteed input/output operations per second available to the user. Remote storage is tied to an availability zone and inter zone migration is charged as inter zone network data transfer. Fig. 4 depicts the price of a gigabyte of data stored for a month and the same amount transferred out to the internet. The extra price for provisioned storage is omitted. It is visible, that storing a gigabyte of data for one month costs similar to transferring it out to the internet.

C. Cost analysis

TABLE II. COST FACTORS (0=NOT APPLICABLE, 1=SHOULD BE APPLIED)

Redundancy / Mode	VM Cost	State Storage 0,05-0,15 [$/GB-month]	State Transfer 0-0,2 [$/GB]
Stateless / none	0	0	0
Stateless / 1 active	1	0	0
Stateful (synchronizing) / 1 active	1	0	1
Stateful (remote storage) / 1 active	1	1	1
Stateful (remote storage) / none	0	1	1

The different redundancy modes have different ongoing costs. Having an active VM costs hourly. Storing and transferring state across availability zones and regions also costs money.

V. CONCLUDING REMARKS AND FUTURE WORK

The current paper introduced the problem, highlighted cloud computing features relevant from resiliency point of view, and evaluated a selected set of patterns respective to downtime and cost. As future work, a deeper analysis of the selected patterns is planned along with the investigation of other patterns.

REFERENCES

[1] Network Functions Virtualisation – Introductory White Paper

[2] Peter Mell, Timothy Grance, The NIST Definition of Cloud Computing

[3] Software-Defined Networking: The New Norm for Networks

[4] OpenVirteX (2015.01.29): http://ovx.onlab.us/

[5] Robert Hanmer "Patterns for Fault Tolerant Software"

[6] Amazon AWS documentation (2015.01.29): http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html

[7] HA Proxy (2015.01.29): http://www.haproxy.org/

[8] ELB (2015.01.29): http://aws.amazon.com/elasticloadbalancing/

[9] gs_NFV-REL001v010101p - Resiliency Requirements (2015.01.29): http://docbox.etsi.org/ISG/NFV/Open/Published/

[10] Ming Mao, Marty Humphrey, "A Performance Study on the VM Startup Time in the Cloud"

[11] Hao Wu, Shangping Ren, Gabriele Garzoglio, Steven Timm, Gerard Bernabeu,Seo-Young Noh, "Modeling the Virtual Machine Launching Overhead under Fermicloud"

[12] Brent Stephens, Alan Cox, Wes Felter, Colin Dixon, John Carter "PAST: Scalable Ethernet for Data Centers"

[13] Guohui Wang, T. S. Eugene Ng "The Impact of Virtualization on Network Performance of Amazon EC2 Data Center"

[14] EC2 pricing model (2015.01.29): http://aws.amazon.com/ec2/pricing/