Ann. Telecommun. (2017) 72:371–384
DOI 10.1007/s12243-017-0576-5

The three-dimensional model for dependability integration in cloud computing

Wiem Abderrahim[1] · Zied Choukair[1]

Received: 16 March 2016 / Accepted: 6 April 2017 / Published online: 20 April 2017
© Institut Mines-Télécom and Springer-Verlag France 2017

Abstract Dependability is one of the highly crucial issues in cloud computing environments given the serious impact of failures on user experience. Cloud computing is a complex system based on virtualization and large scalability, which makes it a frequent place for failure. In order to fight against failures in a cloud, we assure dependability differently from the common way where the focus of fault management is on the Infrastructure as a Service and on the cloud provider side only. We propose a model that integrates dependability with respect to three dimensions according to The Open Group Architecture Framework principles. These dimensions involve various cloud actors (consumer, provider, and broker). They take into consideration the interdependency between the cloud service models (Infrastructure as a Service, Platform as a Service, and Software as a Service) and the different architecture levels (contextual, design, logical, procedural, and operations). DMD proves an enhancement of dependability attributes compared to classically designed and executed cloud systems.

Keywords Cloud · Provider · Consumer · Broker · Dependability · Fault management · Infrastructure as a Service · Platform · Software

✉ Wiem Abderrahim
wiem.abderrahim@supcom.tn

Zied Choukair
zied.choukair@supcom.tn

[1] Higher School of Communications of Tunis, Mediatron Laboratory, Tunis, Tunisia

1 Introduction

Cloud service behavior mainly in dependability terms is nowadays a highly requested feature given the serious impacts of failures in these environments on user experience [1]. Failures are frequent in cloud environments [1, 2] and they take place even within well-renowned cloud providers [3]. In order to overcome this problem, tremendous efforts were devoted to implement dependability in cloud computing. They focused on dependability integration on Infrastructure as a Service (IaaS) and implemented its mechanisms on the cloud provider side [4]. However, faults still occur in cloud computing. Consequently, we need to understand the current weaknesses of its implementation. Actually, in cloud computing, failures are not originated from the infrastructure level solely. They may happen at the platform and application levels. Moreover, failures should better be managed by other actors than the cloud provider such as the broker and the consumer because they participate in end-user service delivery.

Therefore, the key solution for failures in cloud computing is to integrate dependability coherently. In this regard, we propose a three-dimensional model that takes into consideration the cloud actors that influence dependability integration and all the cloud service's models. Our proposed three-Dimensional Model for Dependability integration in cloud computing (DMD) defines three layers to design dependable cloud actors' architectures. DMD model also specifies three cornerstone dimensions for dependability integration that structure the interactions between the cloud actors' architectures in the cloud environment. Additionally, DMD model generates architectural artifacts that guide dependability integration and enable its traceability all over the cloud system. DMD model is used during the cloud actors' architecture design and operation. The design of these architectures should be aligned with the models detailed in the layers of DMD. Their communications are managed according to DMD dimensions during operation.

This paper is organized as follows: in Section 2, we explore the main works dealing with dependability implementation in the three cloud service models. In Section 3, we present our proposed model and its design based on principles, layers and dimensions. In Section 4, we execute and evaluate our model in terms of three dependability attributes in comparison to classical cloud systems. Finally, we conclude the paper in Section 5 and cite our perspectives for future work.

2 Related work to dependability integration in cloud computing

Dependability is the ability to deliver a service that can justifiably be trusted [5]. Practically, it is an aggregation of essentially four attributes: availability, reliability, safety, and maintainability. It has its common mechanisms of implementation, which are fault management mechanisms. In this section, we explore related works to dependability integration in cloud computing service models.

2.1 Fault management in IaaS

Fault management in IaaS service model encompasses fault tolerance and fault forecasting.

2.1.1 Fault tolerance in IaaS

Fault tolerance mechanisms in IaaS are classified into two main categories that are error processing and fault treatment [6].

Error processing methods of IaaS are categorized into proactive policy and reactive policy [7]. On the one hand, the proactive policy is applied through Software Rejuvenation and Live Migration techniques. On the other hand, reactive policy is applied through Checkpointing/Restart and Replication techniques [7].

Software rejuvenation treats transient failures and software aging which affects hypervisors and upper cloud applications [6, 8–10]. The most adopted method for software rejuvenation is a transient software termination followed by a cleaning of the internal state of the system [8]. Cleaning the internal state of the system encompasses freeing the resources of the operating system, suppressing the accumulated error conditions, and reinitializing the internal data structures [11]. In the literature, there are measurement-based approaches that monitor the resource usage statistics in order to predict the optimal time to schedule rejuvenation [11].

Rejuvenation of the hypervisor leads to the transient shutdown of the virtual machines that it manages. Therefore, migration of these virtual machines is necessary during rejuvenation. Virtual machines' migration is divided into two types: cold migration and hot migration (also known as live migration). Cold migration powers off the virtual machines before moving them to a new physical machine. However, live migration moves the virtual machines to another physical machine without interrupting the running applications [12].

In live migration, the virtual machine is suspended transiently. Then, its memory content and its local file system are sent to the new physical machine. Finally, the virtual machine is resumed in the new physical machine [13]. Live migration is undertaken when the servers are overloaded [14]. It improves online maintenance and load balancing. In the literature, several live migration techniques are adopted such as pre-copy and post-copy besides some other ones [13, 15]. Some performance metrics of live migration were evaluated. For instance, live migration has an important overhead for applications with strict requirements in availability. Additionally, live migration causes degradation in performance of the processes running in the virtual machines [13].

Checkpointing is the process of saving system states periodically during its healthy execution [7]. In case of failure, the system executes a rollback to the latest checkpoint and restarts from the last stored state. On the one hand, check-pointing mechanisms can be classified into full or incremental mechanisms. Full mechanisms store the whole system running states periodically. In regard to incremental mechanisms, the first checkpoint stores the whole system running states, whereas the following checkpoints store only the modifications. On the other hand, checkpointing mechanisms can be classified into local and global. Referring to local mechanisms, checkpointing data is stored in a local storage platform. However, checkpointing data is stored in a global storage platform in global mechanisms. Local mechanisms treat transient failure whereas local mechanisms treat permanent failure. Checkpointing mechanisms can be combined all together or selectively combined [2].

Replication is the process of replicating data that is frequently accessed. The number of data replicas is set to three or less [2]. Replication adds redundancy to the system [13]. It defines two types of components: active redundant components and cold standby components. In cloud computing, there are primary virtual machines used for the main work and secondary virtual machines used as backup. The secondary virtual machines are used after all the primary virtual machines fail or when their performance decreases [16].

Replication can be semi-active, semi-passive, or passive [17]. The semi-active replication assigns task execution and provides data state to primary as well as secondary virtual machines. However, only the result obtained by the primary virtual machines is sent to the end user. The result obtained by the primary virtual machines is saved to the log by the hypervisor. In case of failure of the primary virtual machines, the secondary virtual machines take over tasks' execution. The semi-passive replication assigns task execution to the primary virtual machines. The secondary virtual machines take only the state of data from the primary virtual machines. In case of failure of the primary virtual machines, the secondary virtual machines are initiated and updated to the last data state. This engenders a reasonable period of downtime. In regard to the passive replication, data state is saved regularly in the secondary virtual machines. In case of failure of the primary virtual machines, the secondary virtual machines are initiated with the stored data state.

Replication can be deployed according to three forms [17]. First, we find "multiple machines within a cluster" replication form where an application is replicated in two hosts within the same cluster. This deployment form has better latency and bandwidth but suffers from failure dependency. Second, there is also the "multiple clusters within a data center" replication form where an application is replicated in two hosts located in different clusters within the same data center. This deployment form still has a good latency and bandwidth and presents a better failure dependency. Finally, the "multiple data centers" replication form replicates an application in two hosts located in different clusters within different data centers. Authors in [2] and [7] mention that reactive policy is the most used given the simplicity of its implementation. However, proactive policy is more efficient for systems with high availability requirement [7, 9] advocates implementing recovery mechanisms that are aware of the granularity of failure in order to perfectly contain it besides reducing execution time and cost for recovery. Moreover, it highlights the importance of the dynamic nature of cloud computing as a challenge for fault tolerance implementation in these environments.

2.1.2 Fault forecasting in IaaS

Traditionally, fault forecasting takes place by evaluating the system tendency for failure. There are two approaches used for fault forecasting: signature-based approaches and anomaly-based approaches. On the one hand, signature-based approaches detect the faulty behavior of the monitored system and reveal them during the system execution. On the other hand, anomaly-based approaches detect the correct state of the monitored system and deduce the abnormal behavior during the system execution [18].

There are several works that have been interested in failure forecasting in IaaS. Guan et al. in [19] used the Bayesian method as a statistical learning approach for failure forecasting. Capelastegui et al. proposed in [20] an Online Failure Prediction architecture that operates on private IaaS environments. The proposed architecture works over two levels: virtual level and physical level. The virtual level is composed of virtual machines and the physical level is composed of physical machines. It is integrated within a monitoring system. It comprises the following elements: data collectors, prediction servers, and monitoring servers working hand in hand with the managers defined for infrastructure and applications.

2.2 Fault management in platform as a service

For Platform as a Service (PaaS), the existing works have been proposed regarding failover and resilience of applications integrated on these platforms.

For instance, in [21], Sharma et al. proposed Reloc: a session state management architecture for the applications that are deployed in PaaS platforms. Reloc performs failover when the application handling the session goes down transparently to the end user. It also scales according to the required workload. Reloc assures these properties thanks to propagating the session states through a message queuing backbone.

Liang and Lee in [22] fought against failures originated by the dynamicity of application constraints on resources and compatibility. Thus, they proposed a time-stamped approach based on graph structure in order to design cloud-based application in PaaS platforms.

Kozmirchuk et al. [23] presented an architecture of a service broker implemented in PostgreSQL. This service broker provides backup and data recovery to the database management systems (DBMS) of the PaaS Cloud Foundry. The offered backup storage can be local (for temporal storage) or external (long-term storage).

Celesti et al. [24] proposed a PaaS platform whose design is based on two overlay network layers: the Signaling Overlay Network (SON) and the Computing Overlay Network (CON). The computing overlay network contains a single component, which is the computing component. The signaling overlay network comprises five components that control the computing component. The fault-tolerance component, which is a signaling component, recovers the workflow in case 2 of failure event by asking for a new node from the infrastructure for task execution.

In [25], authors proposed a multi-cloud PaaS for building distributed applications. High availability is achieved through balancing the workload and forwarding it to the healthy nodes besides offering redundancy in all cloud levels.

Addo et al. [26] proposed an architecture to improve high availability of PaaS based on the infrastructure of multiple cloud providers. This architecture implements an automatic failover solution that treats Disaster Recovery and High Availability scenarios. The adopted solution replicates data between the cloud providers. In case of failure within a current cloud provider, the failover is initiated towards another cloud provider. A failback solution is also enabled in the proposed architecture in order to resume delivery from the first cloud provider when it is consistent again.

2.3 Fault management in software as a service (SaaS)

In literature, dependability in Software as a Service (SaaS) was treated according to its relationship with key characteristics of SaaS such as scalability and multi-tenancy.

For instance, authors in [27] treated fault tolerance and recovery as a factor altering scalability. A node failure implies a system capability to scale down and a node recovery needs a system capability to scale up.

Su et al. in [28] focused on availability analysis in a multi-tenant architecture (MTA) of SaaS. They started with modeling a MTA of SaaS using the Markov model. Availability analysis shows that a single-tenant SaaS is similar to a multi-tenant SaaS in normal delivery cases. However, after failure happens, availability is dramatically affected by multi-tenancy.

WeiTek et al. in [29] state that the majority of current SaaS solutions have their own mechanisms of redundancy and recovery. SaaS mechanisms for recovery and redundancy are more specific to SaaS nature concerning multi-tenancy.

Authors in [30] proposed an algorithm to checkpoint the meta-data of SaaS's tenants separately. The whole system state is saved through multiple operations of checkpointing, tenant by tenant. In the same way, rollback of the system happens gradually, tenant by tenant to enable the correction of the faulty tenants and set priorities between tenants. Authors evaluated this algorithm in [31] and compared its performance in terms of delay with classic checkpointing techniques. They demonstrated that the proposed algorithm outperforms the classic checkpointing techniques analytically and empirically.

Goel et al. [32] proposed an approach to detect silent failure in SaaS. Silent failures don't have any trace of evidence in the failure logs or in the console. In order to detect them, the authors combine violation checking with finite state machine representing the running states of the application. This approach was tested for nine months and detected 36 silent failures.

Stavrinides and Karatza [33] mentioned the application-directed checkpointing. These applications are responsible for storing their own state progress periodically. Authors use various scheduling algorithms to schedule the processes of the application-directed checkpointing. The goal is to provide resiliency for transient failures.

3 The three-dimensional model for dependability integration in cloud computing design

We have endorsed DMD design with TOGAF [34] framework's principles and paradigms. In this way, DMD model designs the cloud architecture for dependability while taking into consideration all the interactions and interdependencies of the architecture levels. Using TOGAF enables building a coherent, efficient and effective cloud system and services.

Fig. 1 DMD model design

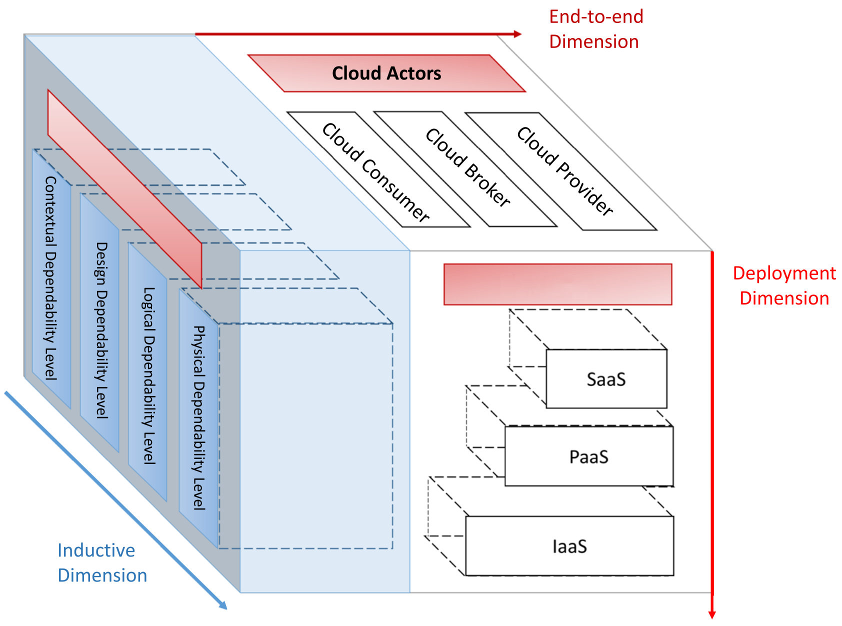
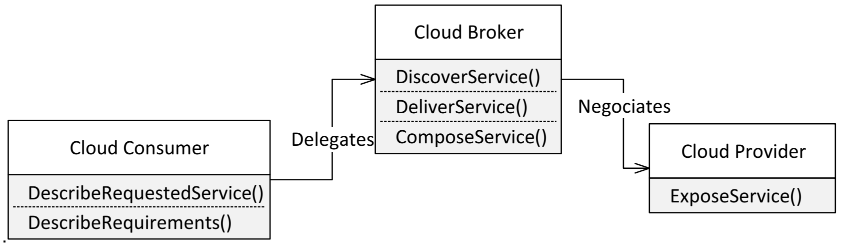


Fig. 2 The cloud actors model



DMD, depicted in Fig. 1, consists of three layers (the cloud actor layer, the dependability management layer, and the cloud service layer) and three dependability integration dimensions (the deployment dimension, the end-to-end dimension, and the inductive dimension). In this section, we detail DMD layers and their cooperation after defining the principles that they should respect.

3.1 DMD model principles

DMD model exposes a set of design principles that are used to design a coherently dependable cloud system architecture. The DMD model defines the three following principles:

- *Principle 1:* Our model has to consider internal and external cloud actors.
- *Principle 2:* Our model must consider failure in all cloud service models i.e. IaaS, PaaS and SaaS.
- *Principle 3:* In our model, dependability in cloud computing must be subject to traceability.

3.2 DMD model layers

From DMD abstract principles, we extract three operational layers that endorse dependability integration in the cloud. These layers distinguish between the actors affecting dependability integration in the cloud, the needed policy for dependability management and the cloud service model to be considered for dependability integration. It is possible also that cloud actors deal with cloud services without involving dependability management. DMD model defines the three following layers:

3.2.1 The cloud actors layer

In the cloud actors model, depicted in Fig. 2, we have designed the main cloud actors necessary to accurate cloud service discovery and delivery. Namely, these actors are the cloud consumer, the cloud provider and the cloud broker [35].

DMD focuses on the cloud broker who is an intermediary part between the cloud consumer and the cloud provider. He is responsible for discovering and delivering trusty services to the cloud consumer.

3.2.2 The dependability management layer

The main role of the dependability management layer is to define the dependability policy to be applied. Depending on the service requirement in dependability and on the impact

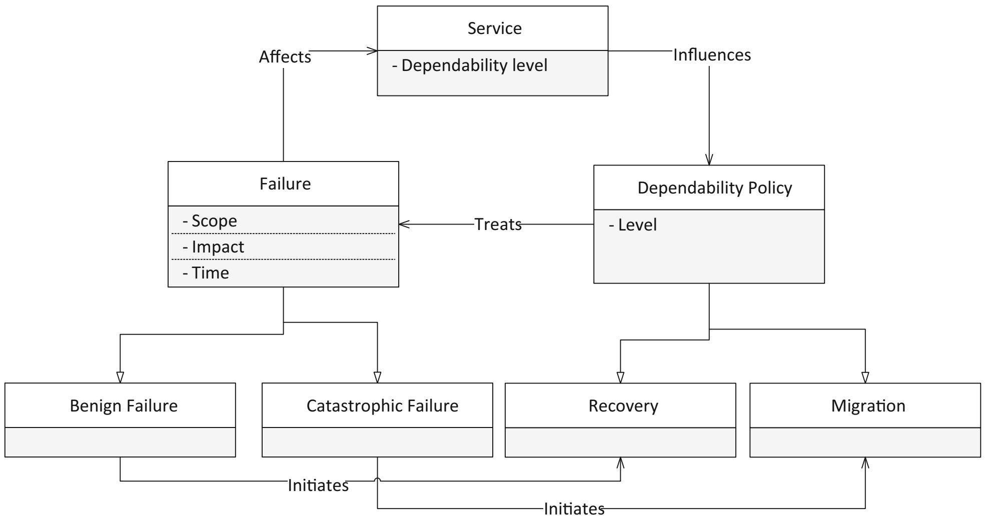


Fig. 3 The dependability management model

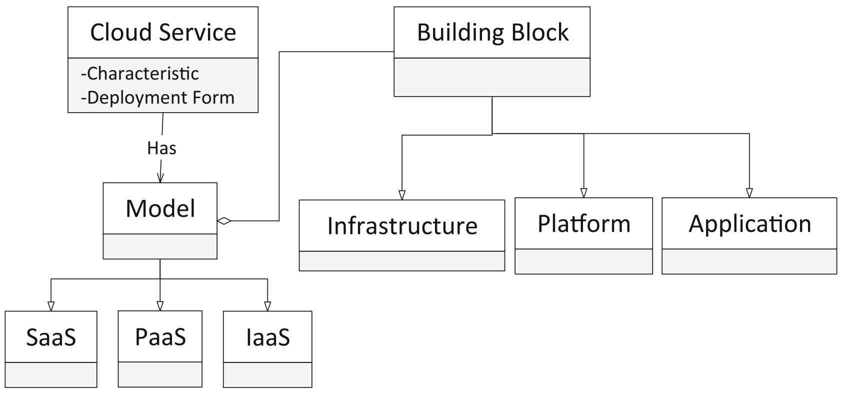


Fig. 4 The cloud services model

of the failure, a specific dependability policy is adopted as illustrated in Fig. 3. The dependability management distinguishes between catastrophic failure and benign failure. Depending on the service requirement, a recovery policy is triggered in case of benign failures and migration towards a new cloud provider is initiated in case of catastrophic failure.

3.2.3 The cloud services layer

Three service models have been defined for cloud computing [35]. They are namely IaaS, PaaS, and SaaS. These models were thoroughly discussed in chapter one (c.f. chapter 1, section 1.2.2).

Each cloud service model has its own characteristics and components. Among these components, there are always fundamental building blocks as depicted in Fig. 4. Indeed, IaaS is made up of the processing building block, the storage building block, and the network building block. PaaS consists of Application Development and Integration building block, Application Administration and Management building block, and Infrastructure Provisioning building block. PaaS can be deployed over its own infrastructure or over an IaaS. SaaS comprises the logic building block, the presentation building block, and the data building block. SaaS applications can be developed and deployed using PaaS. SaaS can also use the IaaS infrastructure for data storage.

3.3 DMD model dimensions

In order to structure the communications within a single DMD layer and the interoperation between the three DMD layers, we define the three following dimensions for dependability integration in DMD model:

3.3.1 Deployment dimension

The deployment dimension (cf. Figs. 1 and 5) encompasses dependability integration in the cloud service layer all over the cloud service models. This is due to the interdependency that exists between the cloud service models.

It is necessary that dependability be integrated in every service model separately and be handled between the cloud service models. The interdependencies between the cloud models are illustrated in Table 1:

In this regard, dependability in SaaS is linked to dependability in PaaS and IaaS. SaaS should implement a dependable data connection with IaaS and a dependable logic connection with PaaS. Similarly, dependability in PaaS is linked to dependability in IaaS through resource provisioning from IaaS infrastructure. PaaS has to implement a dependable resource-provisioning module.

3.3.2 End-to-end dimension

The end-to-end dimension (cf. Figs. 1 and 6) embraces dependability integration in the cloud actor layer. This dimension takes into consideration the most involved actors in dependability implementation and modification.

The end-to-end dimension delineates each actor role for dependability integration and tracks its responsibility and

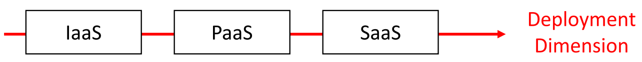


Fig. 5 The deployment dimension

Table 1 Dependency between cloud service models

Interdependency between	Connection
SaaS and PaaS	Logic tier
SaaS and IaaS	Data tier
PaaS and IaaS	Infrastructure provisioning

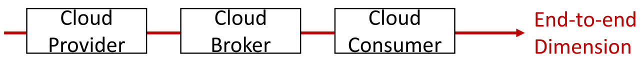


Fig. 6 The end-to-end dimension

accountability in case of failure occurrence as illustrated in Table 2.

3.3.3 Inductive dimension

The inductive dimension (cf. Figs. 1 and 7) undertakes dependability integration coherently in the three layers of DMD. It constitutes the linking dimension between DMD layers. It generates an inductive dependability architecture made up of four levels.

These levels are built based on the development phases of the TOGAF Architecture Development Method (ADM) [34]. They are concretized with architectural artifacts generation to describe the obtained architecture. In the following subsections, we develop the artifacts of each level in the inductive dependability architecture of DMD.

- The dependability contextual level and its artifacts

The dependability contextual level is a description of the context on which the dependability cloud system is developed. This level is built during the “Preliminary phase” and the “Architecture Vision” phase of ADM method.

In the preliminary phase, the needed dependability artifacts are defined based on discussions with the key actors of the cloud system. In the architecture vision phase, dependability related cloud actors and their requirements for a dependable architecture approval are identified. The DMD model ensures to provide final artifacts and deliverables that address the dependability concerns of these actors appropriately.

The main generated artifacts in this level are described in Table 3:

Table 2 Responsibility for dependability integration in cloud computing per cloud actor

Cloud actor	Dependability integration responsibility
The cloud provider	– Implements dependability mechanisms in his service model
The cloud consumer	– Specifies dependability requirements – Monitors his consumption from the cloud service model – Negotiates dependability attributes with the cloud provider
The cloud broker	– Establish dependability as a term in the SLA – Guarantees dependability delivery to the cloud consumer

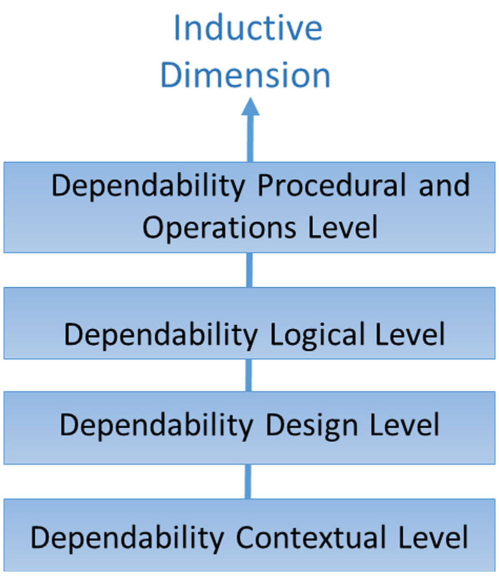

Inductive Dimension
Dependability Procedural and Operations Level
Dependability Logical Level
Dependability Design Level
Dependability Contextual Level

Fig. 7 The inductive dimension

- The dependability design level and its artifacts

The dependability design level is an analysis of the cloud system vulnerabilities by the cloud actors. Responsibilities for dependability integration are defined for the participating cloud actors. This level is built during the Business Architecture phase of ADM method. The main generated artifacts in this level are described in Table 4:

- The dependability logical level and its artifacts

The dependability logical level consists of logical dependability services made up of dependability mechanisms such as fault tolerance, fault forecasting and fault removal. These services are offered by the cloud provider and delivered by

Table 3 The generated artifacts in the contextual level

Generated artifact	Role
The dependability principles catalog	– Captures the architecture principles that describe a dependable solution or architecture – These principles are used to evaluate and agree an outcome for architecture – DMD principles have been already defined
The dependability drivers catalog	– It comprises the drivers that require dependability integration in the cloud system – These drivers are specified basically by the cloud consumer
The dependability actors catalog	– Consists of the dependability related cloud actors whose approval of a final dependable architecture is compulsory – In DMD model, these actors are the cloud provider and the cloud consumer

Table 4 The generated artifacts in the design level

Generated artifact	Role
The Dependability Domain Model (DDM)	– Distinguishes between assets of different dependability levels – It classifies the assets with the same dependability requirements into the same group – Defines the actors who influence dependability integration in the cloud
The Actor/ Dependability Role Matrix	– Delineates their responsibilities in dependability integration in the cloud – In DMD, this matrix has been defined in the end-to-end dimension

the cloud broker. They depend on the logical policy defined by the dependability management layer. This level is developed during the Information Systems Architecture phase of ADM method. The main generated artifact in this level is described in Table 5:

- The dependability procedural and operations level and its artifacts

The dependability procedural and operations level incorporates the dependability procedures (e.g., checkpoint and live migration) that implement dependability services specified in the dependability logical level. These procedures depend on the logical policy defined by the dependability management layer. This level is built during the Technology Architecture phase of ADM method. The main generated artifact in this level is described in Table 6:

4 DMD experimentation and results

In this section, we execute our DMD and evaluate the introduced improvement of three dependability attributes based on on Reliability Block Diagram Method (RBD) [5] and Semi-Markov Process (SMP) [36]. RBD is used to assess reliability given the decomposition of cloud environments into parallel and series elements. SMP is used to assess

Table 5 The generated artifacts in the logical level

Generated artifact	Role
The dependability services catalog	– It presents the offered dependability services composed with cloud service – It is mapped to the principles and drivers determined in upper dependability levels to provide traceability and justification

Table 6 The generated artifacts in the dependability procedural and operations level

Generated artifact	Role
The dependability Procedures catalog	– Includes the dependability procedures that are applied on the cloud service models i.e. SaaS, PaaS and IaaS in order to fulfill logical dependability services

maintainability and safety because it is able to deal with the dynamicity of the cloud.

4.1 DMD execution

DMD execution generates the inductive dependability architecture. It starts with building the dependability contextual level until reaching the dependability procedural and operations level. At each level, there are generated artifacts as outputs. They are used as inputs by the lower levels. Figure 8 depicts the main steps and exchanges between DMD layers and inductive dependability levels.

First, the communication (1) starts between the cloud actors and the dependability contextual level. This leads to the generation of the required artifacts in the dependability contextual level (2). Then, these artifacts are entered as inputs to the dependability design level in (3). Via the interaction with the cloud actors (4), the dependability design level generates the dependability domain model. This artifact is used by the dependability logical level in (6) to translate the contextual policy for dependability into a logical policy. The dependability logical level can construct, then, the dependability services catalog in (7). Afterwards (8), the dependability services catalog is sent to the dependability procedural and operations level. This latter corresponds the logical dependability services with the physical procedures (9) in order to build the dependability procedures catalog (10). Finally, the dependability procedures catalog is applied to the three cloud service models SaaS, PaaS, and IaaS, respectively, in (10), (10'), and (10"). If SaaS is the service required by the cloud consumer (10), supplementary requests are triggered on PaaS (11) and IaaS (12). If PaaS is the service required by the cloud consumer (10') though, a supplementary request is triggered on IaaS (11').

The exchanged messages between DMD layers and dependability levels are detailed in Fig. 9. The messages are exchanged within a single dependability level and between the adjacent levels.

First, the development of the dependability contextual level begins with the interaction between the actors of the cloud actor layer. The cloud provider starts the instantiation of its service model (0.1) then, the cloud consumer initiates the process by asking for a cloud service from the cloud

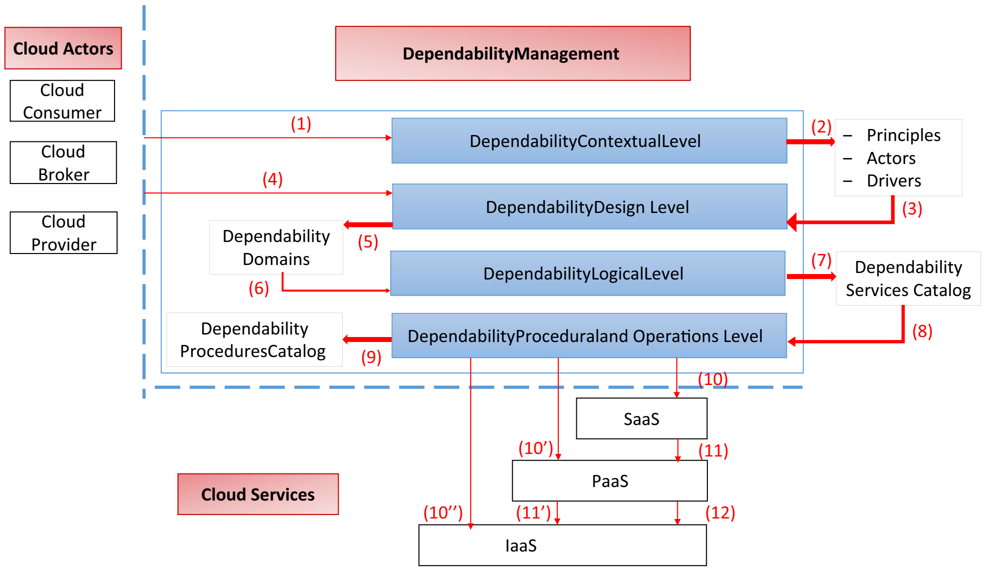
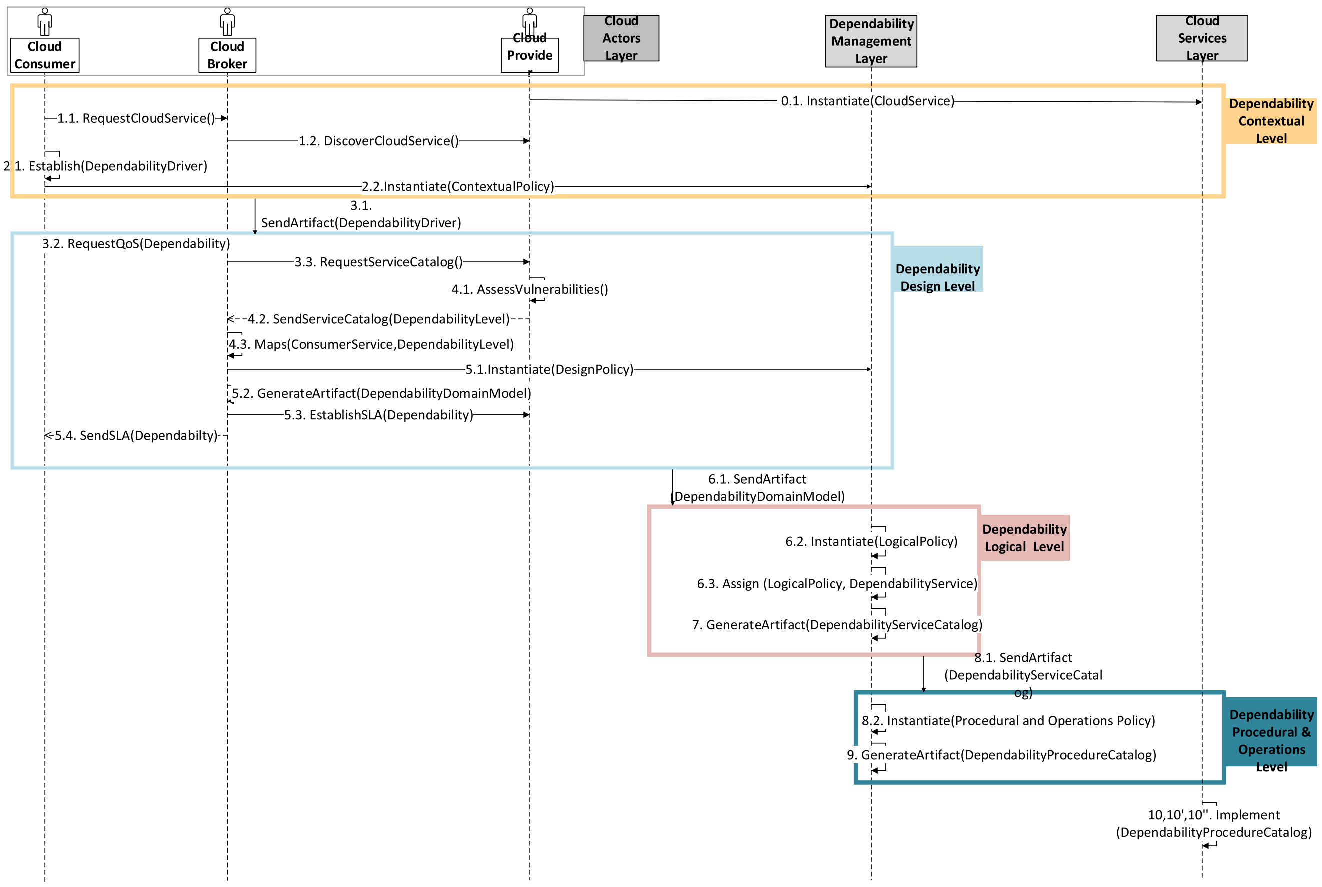


Fig. 8 DMD model execution exchanges

broker (1.1). The cloud broker discovers the required services from different cloud providers (1.2). In the meantime, the cloud consumer establishes his drivers for dependability integration (2.1). Consequently, the contextual policy is set based on these drivers. They are sent as an artifact to the dependability design level (3.1).



Cloud Consumer
Cloud Broker
Cloud Provide
Cloud Actors Layer
Dependability Management Layer
Cloud Services Layer
0.1. Instantiate(CloudService)
Dependability Contextual Level
1.1. RequestCloudService()
1.2. DiscoverCloudService()
2.1. Establish(DependabilityDriver)
2.2.Instantiate(ContextualPolicy)
3.1. SendArtifact(DependabilityDriver)
3.2. RequestQoS(Dependability)
3.3. RequestServiceCatalog()
4.1. AssessVulnerabilities()
4.2. SendServiceCatalog(DependabilityLevel)
4.3. Maps(ConsumerService,DependabilityLevel)
5.1.Instantiate(DesignPolicy)
5.2. GenerateArtifact(DependabilityDomainModel)
5.3. EstablishSLA(Dependability)
5.4. SendSLA(Dependabilty)
Dependability Design Level
6.1. SendArtifact (DependabilityDomainModel)
6.2. Instantiate(LogicalPolicy)
6.3. Assign (LogicalPolicy, DependabilityService)
7. GenerateArtifact(DependabilityServiceCatalog)
Dependability Logical Level
8.1. SendArtifact (DependabilityServiceCatalog)
8.2. Instantiate(Procedural and Operations Policy)
9. GenerateArtifact(DependabilityProcedureCatalog)
Dependability Procedural & Operations Level
10,10',10''. Implement (DependabilityProcedureCatalog)

Fig. 9 Collaboration diagram of DMD model mechanisms

Second, the development of the dependability design level begins. The cloud consumer, based on his dependability drivers, requires dependability in cloud service among other Quality of Service parameters (3.2). The cloud broker asks for the service catalogs afforded by the cloud providers to compare the different offers (3.3). In order to prepare service catalog, the cloud provider assesses accurately the vulnerabilities of his/her system (4.1). Afterwards, he sends the service catalog to the cloud broker with the dependability level of his service (4.2). Then, the cloud broker maps the required level of dependability by the consumer service to the offered level by the cloud provider service (4.3). Next, he initiates the establishment of the design policy by the dependability management layer (5.1). It generates the dependability domain model artifact (5.2) and establishes the SLA with the cloud provider (5.3). SLAs are sent to the cloud consumer (5.4) and the dependability domain model is sent to the dependability logical level (6.1).

Third, the development of the dependability logical level begins. The dependability management layer sets the logical policy (6.1). Subsequently, the policy is matched to the corresponding logical dependability services (e.g. fault tolerance, fault forecasting, etc.) (6.3). This leads to the generation of the dependability services catalog (7) which is sent to the dependability procedural and operations level (8.1).

Finally, the development of the dependability procedural and operations level begins. The dependability services catalog is used to establish the procedural and operations policy (8.2). The dependability management layer maps each logical dependability service with the required dependability procedure. This leads to the generation of the dependability procedure catalog (9). This catalog is sent to the cloud service layer in order to be applied on the cloud service models SaaS (10), PaaS (10'), and IaaS (10").

Figure 10 illustrates the obtained inductive architecture built according to DMD principles for dependability integration in cloud computing.

4.2 Results and analysis

In order to evaluate DMD, we are considering three dependability attributes: reliability, maintainability, and safety. We have only focused on the dependability contextual level where the cloud consumer has diversity of cloud providers thanks to the intermediation of the cloud broker.

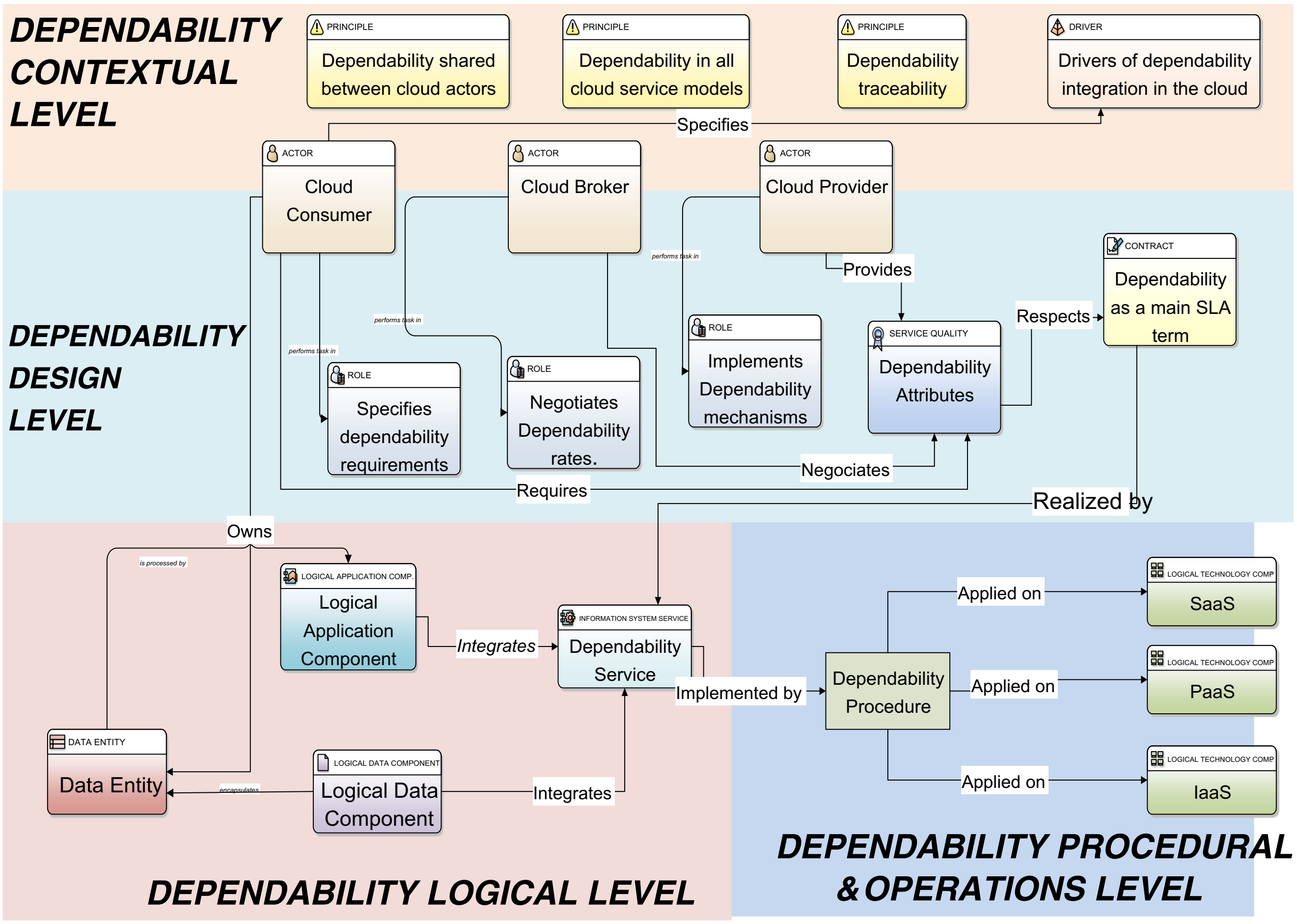
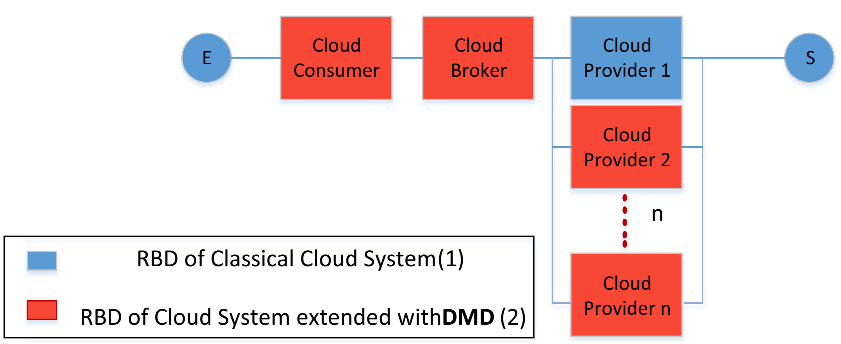


Fig. 10 The developed inductive dependability architecture

Fig. 11 The RBD of the considered cloud systems



4.2.1 Reliability

Reliability is reached through eluding tolerable or non-catastrophic failure. Reliability is enhanced when the Mean Time between Failures is reduced. It is reached thanks to the provider diversity that the cloud broker has.

We adopt a qualitative method for dependability evaluation, which is the Reliability Block Diagram Method (RBD) [5]. We consider two cases for cloud service delivery. The first one is not based on DMD's cloud actor model. The cloud consumer asks a unique cloud provider for the service delivery. The second case, contrarily, uses DMD cloud actor model. The cloud broker is intermediated in the delivery. We assume that failure events are independent for cloud actors. We get the RBD illustrated in Fig. 11. Based on the above RBD, we compute the reliabilities of both cloud systems.

In the first case, the classic cloud system is a series system. The computed reliability $R_1$for the RBD is given by the following equation:

$$R_1 = R_{CloudProvider} \tag{1}$$

In the second case, the cloud system with DMD is a parallel system. The computed reliability R_2 for the RBD is given by the following equation:

$$R_2 = R_{CloudConsumer} \times R_{CloudBroker} \times \sum_{ContractedProviders} R_{CloudProvider} \tag{2}$$

Assuming that:

$$R_{CloudConsumer} = R_{CloudBroker} = 1 \tag{3}$$

because they are dependable.

We get:

$$R_2 = \sum_{i=1}^{n} R_{CloudProvider_i} \tag{4}$$

Therefore,

$$R_2 > R_1 \tag{5}$$

4.2.2 Maintainability

Maintainability is reached through decreasing the necessary time for failure recovery. In this regard, DMD shortens unnecessary recovery time in case of catastrophic failure and triggers migration as a dependability policy.

In order to evaluate maintainability improvement thanks to DMD, we adopt the analysis method proposed in [36]. We model SMP of a classical cloud system and SMP of a cloud system designed according to DMD in Fig. 12. Each model consists of four states: ready (RE), undergoing a safe failure (S), undergoing a non-recoverable failure (NR), and undergoing catastrophic failure (C). The transition probabilities between these different states in both SMPs is denoted by Pi/$0 < i < 8$.

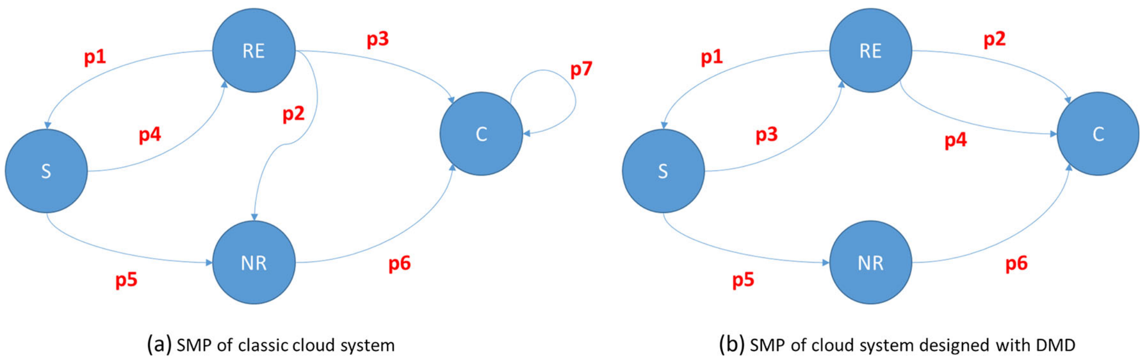

RE
S
C
NR
p1
p2
p3
p4
p5
p6
p7
(a) SMP of classic cloud system
RE
S
C
NR
p1
p2
p3
p4
p5
p6
(b) SMP of cloud system designed with DMD

Fig. 12 SMPs of a classical cloud system and a cloud system with DMD

We notice that the main difference between the two SMPs is the state C, which is absorbing for the classic cloud system (c.f. Fig. 12a). This is due to the inability of classic cloud system to recover catastrophic failures. In DMD, C state is transient and offers the possibility to go back to RE state (c.f. Fig. 12b).

Given the above models and parameters, the transition probability matrices are given by P_1 and P_2:

Transition probability matrix of a classical cloud system (1)	Transition probability matrix of a cloud system with DMD (2)
$P_1 = \begin{pmatrix} 0 & p1 & p2 & p3 \\ p4 & 0 & p5 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$	$P_2 = \begin{pmatrix} 0 & p1 & 0 & p2 \\ p3 & 0 & p5 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix}$

According to [36], maintainability is evaluated through the Mean Time To Maintainability Failure (MTTMF). For maintainability failure, we consider a set of transient states and a set of absorbing states. We also take into consideration the sojourn time in transient states RE and S denoted, respectively, by h_{RE} and h_S. MTTMF is given by the following equations [36]:

MTTMF of a classical cloud system (1)	MTTMF of a cloud system with DMD (2)
$MTTMF_1 = \frac{h_{RE}+h_S}{1-p1} = \frac{h_{RE}+h_S}{p2+p3}$	$MTTMF_2 = \frac{h_{RE}+h_S}{1-p1} = \frac{h_{RE}+h_S}{p2}$

Since p2 and p3 are probabilities $p2 \geq 0$ and $p3 \geq 0$, we have: $p2 + p3 > p2$.

Hence

$$\frac{1}{p2+p3} < \frac{1}{p2}$$

Given that h_{RE} and h_S are sojourn times, $h_{RE} > 0$ and $h_S > 0$, we get: $h_{RE} + h_S > 0$.

Consequently

$$\frac{h_{RE}+h_S}{p2+p3} < \frac{h_{RE}+h_S}{p2}$$

We conclude that

$$MTTMF_1 < MTTMF_2 \tag{6}$$

4.2.3 Safety

Safety is achieved through avoiding catastrophic failure through migration. According to the related work, catastrophic failures are not avoided through migration of cloud services.

In order to evaluate safety improvement thanks to DMD, we adopt the same SMP of the classical cloud system and of the cloud system with DMD. According to [36], safety is evaluated through the Mean Time To Safety Failure (MTTSF). For safety failure, we consider a set of transient states $X_t = \{RE, S, NR\}$ and a set of absorbing states $X_a = \{C\}$. We take into consideration also the sojourn time in transient states RE, S, and NR denoted, respectively, by h_{RE}, h_{NR}, and h_S. $MTTSF$ is given by the following equations [36]:

MTTSF of a classical cloud system (1)	MTTSF of a cloud system with DMD (2)
$MTTSF_1 = \frac{h_{RE}+h_S+h_{NR}}{1-p1} = \frac{h_{RE}+h_S+h_{NR}}{p2+p3}$	$MTTSF_2 = \frac{h_{RE}+h_S+h_{NR}}{1-p1} = \frac{h_{RE}+h_S+h_{NR}}{p2}$

Since p2 and p3 are probabilities $p2 \geq 0$ and $p3 \geq 0$, we have: $p2 + p3 > p2$.

Hence,

$$\frac{1}{p2+p3} < \frac{1}{p2}$$

Given that h_{RE}, h_S and h_{NR} are sojourn times, $h_{RE} > 0$, $h_S > 0$ and $h_{NR} > 0$ we get: $h_{RE} + h_S + h_{NR} > 0$.

Consequently,

$$\frac{h_{RE}+h_S+h_{NR}}{p2+p3} < \frac{h_{RE}+h_S+h_{NR}}{p2}$$

We conclude that

$$MTTSF_1 < MTTSF_2 \tag{7}$$

5 Conclusion and future work

In this paper, we have proposed a three-dimensional model for integrating dependability in cloud services. Our proposed three-Dimensional Model for Dependability integration in cloud computing (DMD) is based on The Open Group Architecture Framework (TOGAF) principles and paradigms. It is comprised of three layers, which are namely the cloud actor layer, the dependability management layer and the cloud service layer. All over these layers, we have defined three dimensions for dependability integration, which are the end-to-end dimension, the deployment dimension, and the inductive dimension. We have focused on the development of the inductive dependability architecture since it assures the inter-operability between the three layers and fulfills the three dimensions integration. Finally, we have evaluated DMD through reliability-building diagrams and semi-Markov process formalism. DMD proved an enhancement of dependability attributes compared to classically designed and operated cloud systems. In future work, we are going to implement DMD through a cloud broker architecture. This architecture implements the four level of the inductive dimension. It adopts the two policies defined in dependability management layer to treat failures. It considers also dependability's integration in the three cloud service models.

References

1. Chowdhury A, Tripathi P (2014) Enhancing cloud computing reliability using efficient scheduling by providing reliability as a service. In: Proceedings of the international conference on parallel, distributed and gridcomputing (PDGC)
2. Sun D, Chang G, Miao C, Wang X (2013) Analyzing, modeling and evaluating dynamic adaptive fault tolerance strategies in cloud computing environments. J Supercomput 66:193–228
3. Kiran M, Awan IU, Mohammed B, Maiyama KM (2016) Optimising fault tolerance in real-time cloud computing IaaS environment. In: Proceedings of the international conference on future internet of things and cloud
4. Neto CL, de Carvalho Filho PB, Duarte A (2013) A systematic mapping study on fault management in cloud computing. In: Proceedings of the conference on parallel and distributed computing, applications and technologies (PDCAT)
5. Avizzienis A, Laprie J-C, Randell B, Landwehr C (2004) Basic concepts and taxonomy of dependable and secure computing. IEEE Trans Dependable Secur Comput 1. Janvier
6. Khomh F (2014) On improving the dependability of cloud applications with fault-tolerance. In: Proceeding of the working IEEE/IFIP conference on software architecture (WICSA)
7. Ganesh A, Sandhya M, Shankar S (2014) A study on fault tolerance methods in cloud computing. In: Proceeding of the international advance computing conference (IACC)
8. Liu J, Zhou J, Buyya R (2015) Software rejuvenation based fault tolerance scheme for cloud applications. In: Proceeding of international conference on cloud computing (CLOUD), pp 1115–1118
9. Kounev S, Reinecke P, Joshi K, Bradley J, Brosig F, Babka V, Gilmore S, Stefanek A (2012) Providing dependability and resilience in the cloud: challenges and opportunities. Resilience Assessment and Evaluation of Computing Systems, pp 65–81
10. Liu J, Zhao J (2016) Providing proactive fault tolerance as a service for cloud applications. In: Proceeding of world congress on of services computing (SERVICES), pp 126–127
11. Sudhakar C, Shah I, Ramesh T (2014) Software rejuvenation in cloud systems using neural networks. In: Proceeding of international conference on parallel, distributed and grid computing (PDGC), pp 230–233
12. Melo M, Maciel P, Araujo J, Matos R, Araujo C (2013) Availability study on cloud computing environments: live migration as a rejuvenation mechanism. In: Proceedings of the international conference on dependable systems and networks, pp 1–6
13. Kapil ES, Pilli D, Joshi RC (2013) Live virtual machine migration techniques: survey and research challenges. In: Proceedings of the international advance computing conference (IACC)
14. Jayasree P, Sharma M (2014) Polvm: parallel optimized live vm migration. In: Proceedings of the international conference on computational intelligence and computing research, (ICCIC), pp 1–8
15. Patel M, Chaudhary S (2014) Survey on a combined approach using prediction and compression to improve pre-copy for efficient live memory migration on xen. In: Proceedings of the international conference on parallel, distributed and grid computing (PDGC), pp 445–450
16. He P, Zhao X, Tan C, Zheng Z, Yuan Y (2016) Evaluation and optimization of the mixed redundancy strategy in cloud-based systems. China Commun 13(9):237–248
17. Jhawar R, Piuri V (2012) Fault tolerance management in iaas clouds. In: Proceedings of AESS European conference on satellite telecommunications (ESTEL)
18. Xin R (2016) Self-healing cloud applications. In: Proceedings of the international conference on software testing, verification and validation (ICST), pp 389–390
19. Guan Q, Zhang Z, Fu S (2011) Ensemble of bayesian predictors for autonomic failure management in cloud computing. In: Proceedings of the international conference on computer communications and networks (ICCCN)
20. Capelastegui P, Navas A, Huertas F, Garcia-Carmona R, Duenas JC (2013) An online failure prediction system for private iaas platforms. In: Proceedings of the 2nd international workshop on dependability issues in cloud computing, pp 4:1–4:3
21. Sharma VS, Sengupta S, Reloc AKM (2012) A resilient loosely coupled application architecture for state management in the cloud. In: Proceedings of the international conference on cloud computing (CLOUD), pp 906–913
22. Liang Q, Lee B (2011) Delivering high resilience in designing Platform-as-a-Service clouds. In: Proceedings of the international conference on cloud computing (CLOUD), pp 676–683
23. Kozmirchuk A, Kokorev A, Nesterov V (2016) Postgresql service with backup and recovery for cloud foundry. In: Proceedings of the international FRUCT conference on intelligence, social media and web (ISMW FRUCT), pp 23–28
24. Celesti A, Peditto N, Verboso F, Villari M (2013) Draco PaaS: a distributed resilient adaptable cloud oriented platform. In: Proceedings of the international symposium on parallel and distributed processing workshops and PhD forum (IPDPSW), pp 1490–1497
25. Paraiso F, Merle P, Seinturier L (2014) Socloud: a service-oriented omponent-based PaaS for managing portability, provisioning, elasticity and high availability across multiple clouds. J Comput, Special Issue on Cloud Computing
26. Addo ID, Ahamed SI, Chu WC (2014) A reference architecture for high-availability automatic failover between PaaS cloud providers. In: Proceedings of the international conference on trustworthy systems and their applications (TSA), pp 4–21 and 9–10
27. Tsai WT, Huang Y, Bai X, Gao J (2012) Scalable architecture for saas. In: Proceedings of the international symposium on object component service-oriented real-time distributed computing, (ISORC)
28. Su W, Lin C, Meng K (2014) Modeling and analysis of availability for SaaS multi-tenant architecture. In: Proceedings of the international symposium on service oriented system engineering (SOSE), pp 365–369
29. Tsai W, Bai X, Huang Y (2014) Software-as-a-service (saas): perspectives and challenges. Sci China Inf Sci 57(5):1–15
30. Yousef B, Zhu H, Younas M (2015) Tenant level checkpointing of meta-data for multi-tenancy SaaS. In: Proceedings of the international symposium on service oriented system engineering (SOSE), pp 148–153
31. Zhu H, Yousef B, Younas M (2015) Evaluation of a tenant level checkpointing technique for SaaS applications. In: Proceedings of the international conference on cloud computing (CLOUD), pp 949–989
32. Goel G, Roy A, Ganesan R (2013) Identifying silent failures of saas services using finite state machine based invariant analysis. In: Proceedings of the international symposium on software reliability engineering workshops (ISSREW), pp 290–295

33. Stavrinides GL, Karatza HD (2016) Scheduling real-time parallel applications in SaaS clouds in the presence of transient software failures. In: Proceedings of the international symposium on performance evaluation of computer and telecommunication systems (SPECTS)
34. The open group architecture framework togaf. http://pubs.opengroup.org/architecture/togaf9-doc/arch/index.html, 2011. Version 9.1, [Online; accessed 27-02-2017].
35. Liu F, Tong J, Mao J, Bohn R, Messina J, Badger L, Leaf D (2011) Nist special publication 500-292: Nist cloud computing reference architecture. http://ws680.nist.gov/publication/get_pdf.cfm?pub_id=909505. [Online; accessed 27-02-2017]
36. Huang J, Lin C, Kong X, Zhu Y (2011) Modeling and analysis of dependability attributes of service computing systems. In: Proceedings of the international conference on services computing (SCC), pp 184–191