ACE: Availability-Aware CloudSim Extension

Manar Jammal, Hassan Hawilo, Ali Kanso, and Abdallah Shami, *Senior Member, IEEE*

***Abstract*—In the interconnected globe where service delivery is the success measure, cloud high availability (HA) is an indispensable area for enterprises. An HA-aware cloud system provides different approaches to handle the outages. This includes geo-redundancy, failover schemes, and HA-aware placement solutions. However, using real-cloud platforms to model HA-aware approaches is hindered by the configuration settings. To this end, simulation tools, such as CloudSim, can be used to evaluate HA solutions and a cloud resiliency against failures. CloudSim allows implementing of scheduling policies, but it does not support HA properties. This paper provides availability-aware CloudSim extension (ACE). ACE extends CloudSim with a graphical and textual modeling to ensure simplicity and reusability of cloud scenarios. ACE has added HA-aware modeling (HA metrics and failure/redundancy/interdependency models) and HA-aware scheduling (HA-aware placements, failover, repair, and load balancing policies) into CloudSim. With ACE, the creation of cloud scenarios is facilitated, and multiple HA-aware deployment solutions can be evaluated under different stochastic and deterministic events. ACE can assess the impact of different redundancy/failure models, and other performance policies to extract HA-aware lessons. In this paper, ACE is assessed on a cloud application to evaluate different redundancy/failure models and provide availability analysis of the HA-aware placement solution.**

***Index Terms*—High availability, failure injection, software components, virtual machines, repair, load balancing, failover, redundancy, simulation, CloudSim, computational path.**

I. Introduction

Although cloud computing is not new, it is considered a game-changing concept in the information and communications technology fields. Studies show that cloud services have evolved to everything or anything as a Service (XaaS), which will be responsible for the growth in the market of the cloud services [1]. The XaaS includes software as a service (SaaS), infrastructure as a service (IaaS), and platform as a service (PaaS) [2]. Therefore, different challenges should be addressed to ensure the cloud adoption in many enterprises. These issues range from compliance concerns, security, interoperability, and other services management issues [3]. However, availability is one of the key factors to ensure an optimal cloud performance and satisfy quality of service (QoS) and quality of experience (QoE). Although cloud data are safely stored in well-managed cloud platforms, outages can happen even on these platforms. For example, GitLab has faced data loss due to accident deletion on Feb. 1, 2017, which has caused the permanent loss of "six hours' worth" of data [4]. Similarly, Dropbox, Microsoft Azure, Google, and Amazon Web Services have suffered cloud outages in the last few years [5]–[8]. Therefore, availability is a main concern to be addressed in large distributed systems and applications, mainly cloud platforms [9]–[12]. It is necessary to note that availability is the measure of the percentage of time a system is available for normal usage in a given time interval [13].

In order to ensure highly available cloud services, it is necessary to design a cloud model and simulation that can emulate real cloud outages, execute repairing policies, and recover failures accordingly. Using real cloud settings (i.e., Amazon Elastic Compute Cloud (EC2)) to model applications and evaluate their behavior under certain performance policies is restricted by cloud platform configurations and infrastructure. As an alternative, modeling and simulations can be used to emulate the cloud. These approaches can build new algorithms and policies, test them before the actual deployment in a real cloud, and enhance the performance of large-scale distributed systems. This can save the tenants significant time and effort and some degree of reassurance about the level of high availability (HA). On the other hand, cloud provider can benefit from the simulations to evaluate new features/extensions to their cloud and check if their HA guarantees are realistic.

Due to their scalability and efficiency characteristics, discrete event simulators (DES) can be used in the modeling and evaluation of the distributed systems [14]. CloudSim is a simulation framework used for the scheduling and resource allocation algorithms on cloud infrastructure. However, it is not designed to model HA constructs and therefore, overlooks the availability and failures of the cloud applications. In this paper, we propose Availability-aware CloudSim Extension (ACE) that models HA-aware policies and metrics in the cloud. The contributions of ACE can be summarized as follows:

- Allow the injection of failures and failure-dependency between cloud applications.
- Implement load balancing technique.
- Differentiate between the cloud as a provider consisting of data centers (DCs) and servers, and as a user where applications components are modeled to form functional chains and protection groups.
- Provide recovery and repair solutions.
- Schedule cloud applications with HA objectives and evaluate fault-tolerant cloud scheduling approaches.

Manuscript received November 22, 2017; revised July 13, 2018 and October 7, 2018; accepted October 30, 2018. Date of publication November 6, 2018; date of current version December 10, 2018. This work is partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC-STPGP 447230) and Ericsson Research. The associate editor coordinating the review of this paper and approving it for publication was B. Stiller. *(Corresponding author: Manar Jammal.)*

M. Jammal, H. Hawilo, and A. Shami are with the Electrical and Computer Engineering Department, Western University, London, ON N6A 5B9, Canada (e-mail: mjammal@uwo.ca; hhawilo@uwo.ca; abdallah.shami@uwo.ca).

A. Kanso is with IBM Cloud, IBM T. J. Watson Research Center, Yorktown Heights, NY 10598 USA (e-mail: akanso@us.ibm.com).

Digital Object Identifier 10.1109/TNSM.2018.2879665

- Support reactive, proactive, and adaptive fault-tolerant approaches. Any failure type can be injected into the cloud infrastructure and applications as long as it is associated with its mean time to failure (MTTF) and mean time to repair (MTTR).
- Provide generic and repeatable input templates for cloud simulators, GITS, using JavaScript Object Notation (JSON) data format.

The rest of this paper is structured as follows. In Section II, the related work is presented for cloud simulators. Section III presents the problem background and motivation including different fault types, measures, cloud scheduling, and the complexity of cloud models. In Section IV, ACE design and implementation are described. Section V defines the evaluation results of ACE. Finally, Section VI presents the conclusion and future work.

II. Related Work

This section provides a literature study on some of the existing cloud simulators. Wickremasinghe *et al.* [15] propose CloudAnalyst as an extension to CloudSim. CloudAnalyst extends CloudSim with a module for visualizing the simulation results as a Portable Document Format (PDF) file. Although CloudAnalyst focuses on modeling simulations rather than development, it supports neither HA-aware metrics nor a generic input template with HA features. Kliazovich *et al.* [16] propose GreenCloud as an energy-aware simulator for cloud DCs. GreenCloud models the energy consumption of cloud infrastructure (DCs, servers, and network links) and packet-level communication configurations. Gupta *et al.* [17] propose Green Data Center Simulator (GDCSim) as another energy-aware simulator to model DC behavior and resource management in terms of power objectives. Although energy and HA are main concerns in the cloud, GreenCloud and GDCSim exclude any HA modeling in the cloud. Zhou *et al.* [18] extend CloudSim with FTCloudSim to include reliability mechanisms. It evaluates the system performance under faulty events and generates the necessary details to determine the pros and cons of the approach under evaluation. Although FTCloudSim supports some reliability features, it discards redundancy between applications components as well as the dependency relations. It does not support the automated generation of requests within the functional chain of a certain application. Also, the recovery policies do not ensure a failover to a redundant component and do not trigger the repair policy of the faulty component.

Tighe *et al.* [19] propose Data Center Simulator (DCSim) to evaluate different DC management and scheduling algorithms. Although DCSim models multi-tier applications and supports the dependency and replication simulations between virtual machines (VMs), it discards other HA features (failure injection, repair, recovery, and load balancing). Ostermann *et al.* [20] propose GroudSim as Grid and Cloud simulator based on discrete events. According to different distribution functions, GroudSim can simulate the execution of jobs on computing resources and calculate the associated cost and workload. Unlike other literature studies, we distinguish ourselves with a unique well-defined availability-aware extension of CloudSim simulator. Fig. 1 shows a comparative analysis between ACE and some of the existing cloud simulators in terms of HA-aware modeling and scheduling. These simulators do not support HA-aware scheduling as they do not include availability modeling features (in terms of HA metrics, redundancy models, and load balancing). The extension does not only capture an HA-aware input template for the simulator, but it supports different HA metrics and features. This includes failure injection module, applications components recovery and repair, HA-aware allocation mechanism, automated request generation to maintain application functional chains, and load balancing. Besides, ACE captures different redundancy models and multiple distributions functions for the failure/repair rates.

III. Background and Motivation

Realizing an HA-aware cloud system entails an intricate planning. However, to design a new and innovative HA-aware cloud solution, a modeling and simulation environment is needed to model several cloud properties, such as availability, security, and energy. A simulation environment can be applied to evaluate multiple scenarios under different performance and HA constraints.

CloudSim is an extensible cloud-based simulator built in the CLOUDS Laboratory at the University of Melbourne, Australia. It models and simulates cloud systems and different scheduling and allocation policies [21]–[23]. CloudSim is an open-source simulator and is built on the top of a discrete event simulator, SimJava [15], [24]. However, CloudSim does not support availability-aware properties, constraints, and/or allocation policy. Also, it does not support a "ready-to-use" setting to generate cloud scenarios, but it needs a Java-based code to create any cloud set-up using its entities. Therefore, this paper aims at extending CloudSim with HA features and generic input template for creating cloud scenarios while ensuring repeatability, portability, and simplicity. This section addresses the fault types and measures of the cloud as well as the structure of the cloud model.

A. Outages and Fault-Tolerant Measures

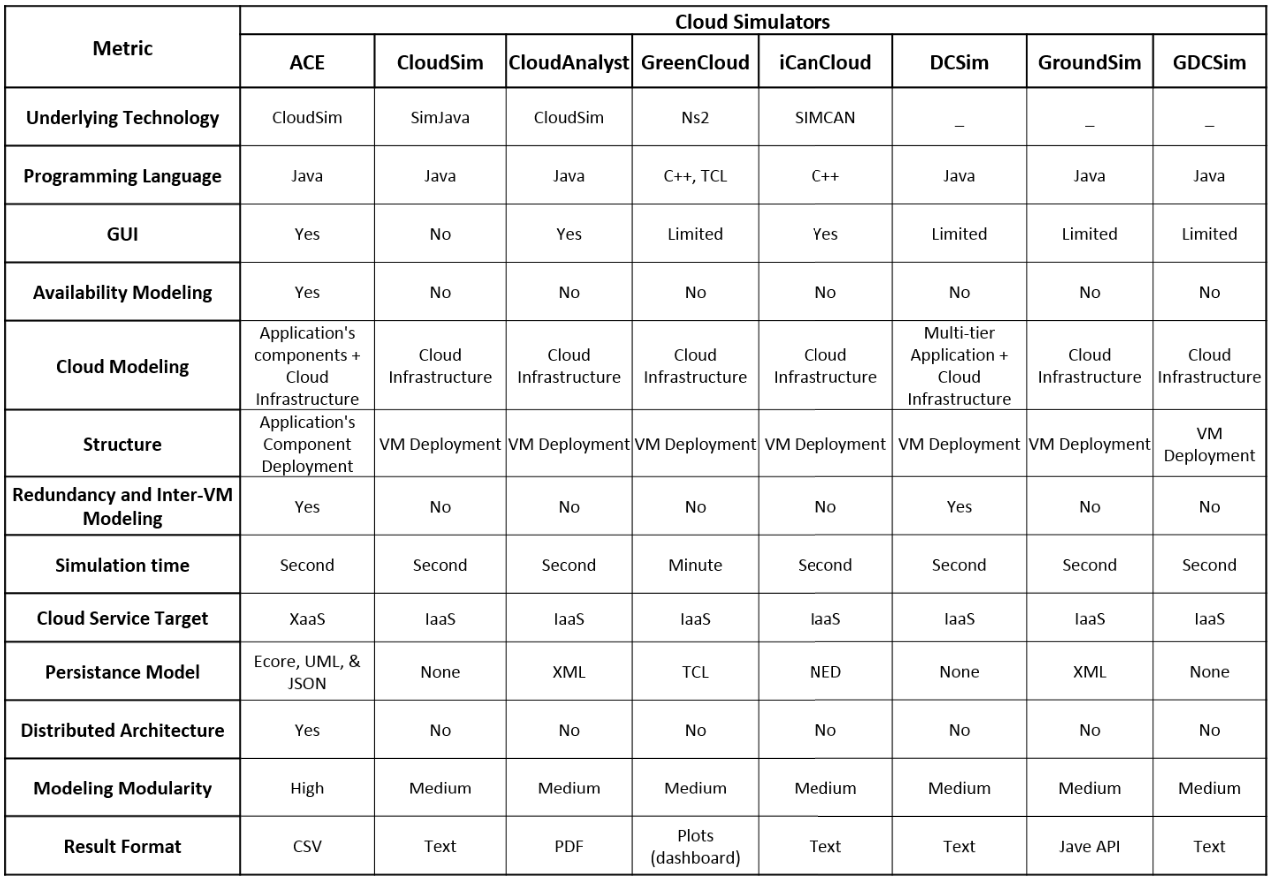
In a cloud system, faults are realized as resources failures whether the resource is application or infrastructure [25]–[28]. The two common types of failures behaviors in the cloud are:

Fail-stop/Crash failures: The entity of a system changes to a failure state that is detected by other entities [29], [30].

Byzantine failures: Upon a failure, the component shows malicious and random behavior, which sometimes collides with other components and causes the system to perform in an arbitrary mode [25], [29], [31].

With this in mind, fault-tolerance or availability of a system is expressed in terms of MTTF and MTTR where MTTF determines the time in which the system functions normally before failure, and MTTR is the time needed to resume the functionality of a failed system [32]. The availability A is calculated as follows:

$$A = \frac{MTTF}{MTTF + MTTR}. \quad (1)$$



Metric	Cloud Simulators							
	ACE	CloudSim	CloudAnalyst	GreenCloud	iCanCloud	DCSim	GroundSim	GDCSim
Underlying Technology	CloudSim	SimJava	CloudSim	Ns2	SIMCAN	_	_	_
Programming Language	Java	Java	Java	C++, TCL	C++	Java	Java	Java
GUI	Yes	No	Yes	Limited	Yes	Limited	Limited	Limited
Availability Modeling	Yes	No	No	No	No	No	No	No
Cloud Modeling	Application's components + Cloud Infrastructure	Cloud Infrastructure	Cloud Infrastructure	Cloud Infrastructure	Cloud Infrastructure	Multi-tier Application + Cloud Infrastructure	Cloud Infrastructure	Cloud Infrastructure
Structure	Application's Component Deployment	VM Deployment	VM Deployment	VM Deployment	VM Deployment	VM Deployment	VM Deployment	VM Deployment
Redundancy and Inter-VM Modeling	Yes	No	No	No	No	Yes	No	No
Simulation time	Second	Second	Second	Minute	Second	Second	Second	Second
Cloud Service Target	XaaS	IaaS	IaaS	IaaS	IaaS	IaaS	IaaS	IaaS
Persistance Model	Ecore, UML, & JSON	None	XML	TCL	NED	None	XML	None
Distributed Architecture	Yes	No	No	No	No	No	No	No
Modeling Modularity	High	Medium	Medium	Medium	Medium	Medium	Medium	Medium
Result Format	CSV	Text	PDF	Plots (dashboard)	Text	Text	Jave API	Text

Fig. 1. Comparative analysis of cloud simulators in terms of HA-aware modeling and scheduling.

B. Scheduling in the Cloud

To ensure the fully-exploitation of cloud capabilities, it is necessary to design an HA-aware solution while maintaining an efficient utilization of computational resources [33], [34]. Each cloud DC hosts thousands of servers with hundreds of VMs. While VMs process multiple tasks, the cloud receives new batches of users' requests. In order to have a seamless processing, these requests should be hosted by the VM/server that can satisfy computational needs while maximizing their availability. Therefore, task scheduling and assignment are paramount approaches to prevent any service level agreement (SLA) violation in terms of HA and performance of the cloud [35]. With scheduling, different cloud metrics and objectives can be evaluated in terms of each other (HA-energy-security or HA-performance-fairness) to generate a trade-off that satisfies the desirable SLA and QoS. In order to perform scheduling in a cloud environment, different phases should be executed:

- *Determination phase:* Define type of "to-be-processed" requests/task, such as rigid tasks (predefined resources by users), evolving tasks (changeable resources through simulation), and moldable tasks (constrained resources by the scheduler) [36].
- *Discover phase:* Resource/HA/Energy-based pooling and filtering of available infrastructure.
- *Decision phase:* Choose target host (DC, server, and VM).
- *Process phase:* Submit the request/task to the host to be processed.

C. Cloud Model

Similar to Service Oriented Architecture (SOA), different roles can be defined in any cloud environment [37]. Fig. 2 shows the cloud model. These roles can be distributed as follows:

Cloud provider offers PaaS and IaaS to the users. It consists of multiple DCs hosting thousands of servers. Each infrastructure component is characterized by its resources and HA metrics.

Cloud broker is an intermediate negotiator between the cloud service provider and consumer.

Cloud aggregators combine different cloud providers to offer a larger and hybrid infrastructure to cloud customers.

Cloud users consist of multiple applications components that use the cloud capabilities to execute certain computations or to process requests. A 3-tier Web application is an example of cloud applications [38]. At the front-end, a Hypertext Transfer Protocol Secure (HTTPS) server processes requests and forwards them to an App server. At the back-end, a database (DB) server stores the users' data and sponsors the App server that generates the required information. The dependency interaction between these component types constitutes the functional/computational path that should be followed by a request to be successfully executed.

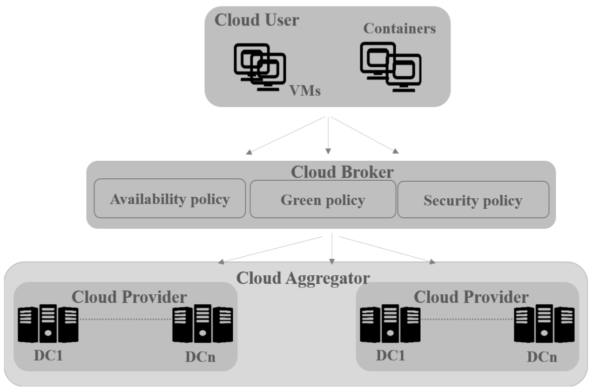


Fig. 2. Different roles in the cloud model.

To this end, this paper provides an abstract and generic simulation approach where different cloud nodes (DC, server, application components, VMs), load balancer, and HA features are well-defined and modeled.

IV. ACE Design

This section describes the design of ACE to simulate and evaluate cloud systems having an erroneous nature. ACE contributions are summarized as follows:

- Define an architecture for HA-aware cloud (generic template for cloud model that captures HA features).
- Provide automated generation of requests while discovering the functional chains (computational path) and protection group (redundancy group) for cloud applications.
- Integrate HA-aware cloud allocation algorithm that places cloud applications while maximizing their HA and satisfying other SLA performance requirements.
- Design a load balancing algorithm at each tier of a cloud application.
- Provide a failure injection module and recovery/repair mechanisms to ensure self-healing upon failures of DCs, servers, and VMs (representing cloud applications).
- Implement a modular and reusable HA-aware extension for CloudSim.
- Evaluate availability of different HA-aware deployments of cloud applications.

The source codes of ACE are released in the Atlassian Bitbucket repository (https://bitbucket.org/manarjammal/ace-availability-aware-cloudsim-extension.git).

A. ACE Modules

Input template module: CloudSim is extended with a user-friendly method, GITS (generic input template for cloud simulators), for generating a scenario, without exposing the cloud user to the details of development and coding in the simulator. Manual creation of scenarios in CloudSim can be a tedious and erroneous job. Therefore, GITS aims at providing an imperative way to generate cloud scenarios that ensure configurations reusability, applications portability, and automated orchestration between different cloud providers while minimizing error, cost, and time-to-value.

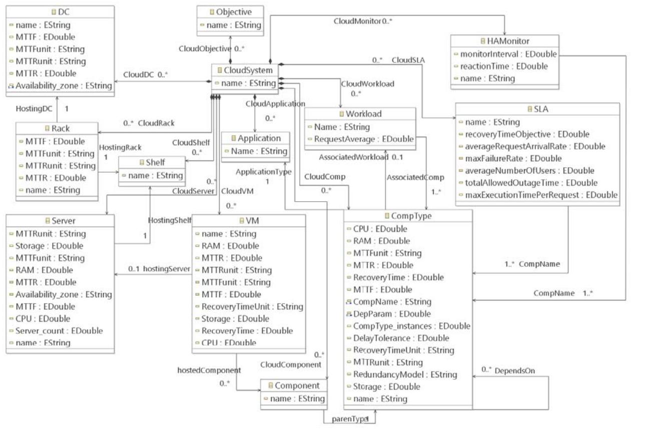


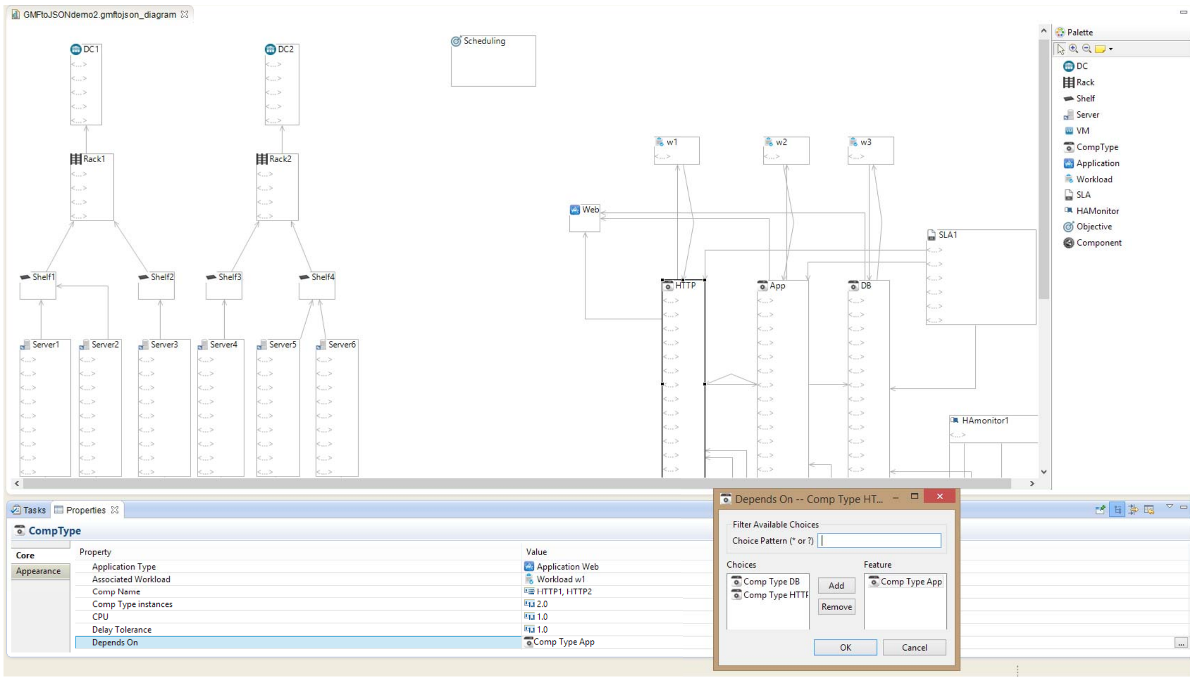
Fig. 3. ACE model (Using Eclipse Ecore representation).

GITS models the cloud provider, cloud user, and virtualization mapping between them through VM/containers. It captures different HA attributes associated with each entity of the cloud including HA statistical measures (MTTF, MTTR, and recovery time), redundancy model, failure types, and recovery mechanisms. GITS models the cloud as a cloud provider consisting of multiple DCs hosting multiple racks and servers and a cloud application consisting of multiple components of different types. Each type is associated with a failure type, redundancy model, SLA requirements, workload characteristics, and redundancy model. Different redundancy and dependency relations between component types are captured as well. Components can be modeled in an active-active redundancy model, active-standby (cold and hot) model, and active-spare model. Fig. 3 shows Ecore diagram for GITS cloud model.

To maintain modularity and easy-to-use features, GITS consists of a multi-layer input model. At the frontend layer, an Eclipse graphic modeling framework (GMF) project is built to provide a user-friendly approach. An Extensible Markup Language (XML) file is generated from the GMF, which will be inputted to the mid-layer and parsed into a JSON template. The latter is used because it is a human readable and reusable approach, which is mapped to a Unified Modeling Language (UML) class diagram at the backend layer. GITS is generic in a sense that it can be easily modified to fit any cloud simulator. Fig. 4 and Fig. 5 show the GMF and JSON template of GITS.

It is necessary to note that the output of any simulation is saved as an excel sheet where requests information is included.

Computational path and request generation module: Each application consists of multiple components. Each component belongs to a certain type that can depend on and/or sponsor other types. For example, a Web application consists of 3 types: HTTPS-based component type, App-based type, and DB-based type where HTTPS depends on the App that is sponsored by DB. This interdependency communication between applications components forms the computational path or functional chain. In other words, it is the route followed by a user request to be successfully executed. Note that a request refers to a CloudSim cloudlet. The CloudSim cloudlet is refereed by cloudlet in the rest of the paper. CloudSim is extended to include this components chaining. Each computational path



CompType

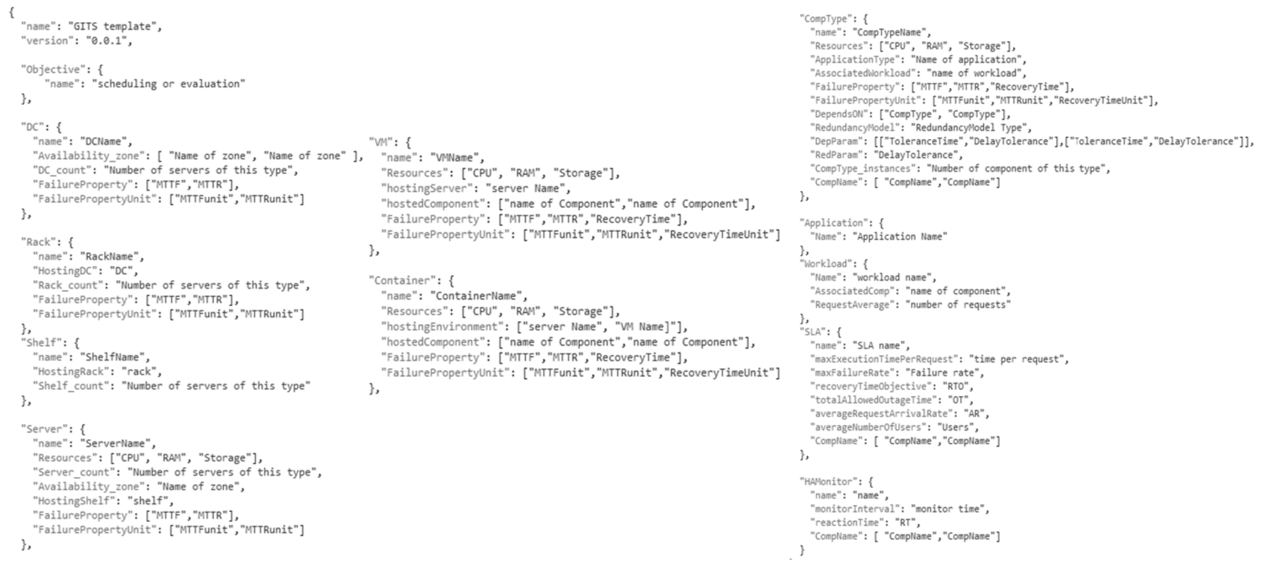
Property	Value
Application Type	Application Web
Associated Workload	Workload w1
Comp Name	HTTP1, HTTP2
Comp Type instances	2.0
CPU	1.0
Delay Tolerance	1.0
Depends On	Comp Type App

Filter Available Choices

Choice Pattern (* or ?)

Choices	Feature
Comp Type DB	Comp Type App
Comp Type HTTP	

Fig. 4. ACE graphical editor.


```
{
  "name": "GITS template",
  "version": "0.0.1",

  "Objective": {
      "name": "scheduling or evaluation"
  },

  "DC": {
    "name": "DCName",
    "Availability_zone": [ "Name of zone", "Name of zone" ],
    "DC_count": "Number of servers of this type",
    "FailureProperty": ["MTTF","MTTR"],
    "FailurePropertyUnit": ["MTTFunit","MTTRunit"]
  },

  "Rack": {
    "name": "RackName",
    "HostingDC": "DC",
    "Rack_count": "Number of servers of this type",
    "FailureProperty": ["MTTF","MTTR"],
    "FailurePropertyUnit": ["MTTFunit","MTTRunit"]
  },
  "Shelf": {
    "name": "ShelfName",
    "HostingRack": "rack",
    "Shelf_count": "Number of servers of this type"
  },

  "Server": {
    "name": "ServerName",
    "Resources": ["CPU", "RAM", "Storage"],
    "Server_count": "Number of servers of this type",
    "Availability_zone": "Name of zone",
    "HostingShelf": "shelf",
    "FailureProperty": ["MTTF","MTTR"],
    "FailurePropertyUnit": ["MTTFunit","MTTRunit"]
  },

  "VM": {
    "name": "VMName",
    "Resources": ["CPU", "RAM", "Storage"],
    "hostingServer": "server Name",
    "hostedComponent": ["name of Component","name of Component"],
    "FailureProperty": ["MTTF","MTTR","RecoveryTime"],
    "FailurePropertyUnit": ["MTTFunit","MTTRunit","RecoveryTimeUnit"]
  },

  "Container": {
    "name": "ContainerName",
    "Resources": ["CPU", "RAM", "Storage"],
    "hostingEnvironment": ["server Name", "VM Name]"],
    "hostedComponent": ["name of Component","name of Component"],
    "FailureProperty": ["MTTF","MTTR","RecoveryTime"],
    "FailurePropertyUnit": ["MTTFunit","MTTRunit","RecoveryTimeUnit"]
  },

  "CompType": {
    "name": "CompTypeName",
    "Resources": ["CPU", "RAM", "Storage"],
    "ApplicationType": "Name of application",
    "AssociatedWorkload": "name of workload",
    "FailureProperty": ["MTTF","MTTR","RecoveryTime"],
    "FailurePropertyUnit": ["MTTFunit","MTTRunit","RecoveryTimeUnit"],
    "DependsON": ["CompType", "CompType"],
    "RedundancyModel": "RedundancyModel Type",
    "DepParam": [["ToleranceTime","DelayTolerance"],["ToleranceTime","DelayTolerance"]],
    "RedParam": "DelayTolerance",
    "CompType_instances": "Number of component of this type",
    "CompName": [ "CompName","CompName"]
  },

  "Application": {
    "Name": "Application Name"
  },
  "Workload": {
    "Name": "workload name",
    "AssociatedComp": "name of component",
    "RequestAverage": "number of requests"
  },
  "SLA": {
    "name": "SLA name",
    "maxExecutionTimePerRequest": "time per request",
    "maxFailureRate": "Failure rate",
    "recoveryTimeObjective": "RTO",
    "totalAllowedOutageTime": "OT",
    "averageRequestArrivalRate": "AR",
    "averageNumberOfUsers": "Users",
    "CompName": [ "CompName","CompName"]
  },

  "HAMonitor": {
    "name": "name",
    "monitorInterval": "monitor time",
    "reactionTime": "RT",
    "CompName": [ "CompName","CompName"]
  }
```

Fig. 5. ACE JSON template.

consists of the different levels (three levels in case of the Web application). The first level represents the components types that do not have any dependents (HTTPS type in case of the Web application). The requests arrive at the load balancer to be forwarded to the first level/tier of the chain. The first tier represents the primary component and its redundant ones. It is necessary to note that this redundancy relation forms a protection group (primary and redundant components). The requests are distributed on the active components of the first level. Once a request is processed, a sub-request is generated and forwarded again to the load balancer to be distributed on the active components of the second tier. The same process goes on until the request reaches the last tier. Fig. 6 shows the Web application with computational path and protection group. A request is successfully processed if all the subrequests created at all the tiers of the path are successfully executed. CloudSim is also extended to include automatic generation of requests. Upon completion, a new request is automatically generated and distributed by the load balancer to the different tiers of the chain. It is necessary to note that user can either define a number of requests at the beginning of simulation or trigger the automated generation of requests while defining the simulation time.

HA-aware placement module: CloudSim provides space and time-based allocation policies, but it overlooks HA objective and constraints. ACE provides an HA-aware allocation policy

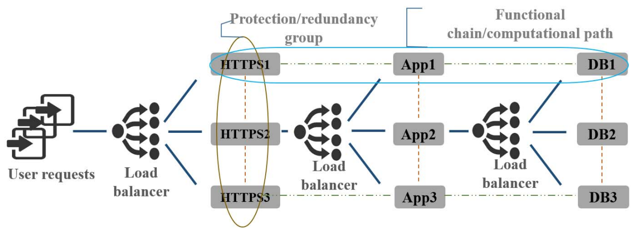


Fig. 6. Example of three-tier Web application.

for applications components. A simulator user can use either the default policy or the proposed HA-aware approach. The HA-aware approach is divided into sub-algorithms. Prior to the applications' components placement, a criticality analysis is performed to differentiate between applications components priorities [39]. For example, if a 3-tier Web application has only one DB active component, the failure of the DB instance would have a high impact on the application's requests.

The failure of each application's component can cause an outage *(O)* or a service degradation (*D*). With N_{fail} being the number of failures, the component's criticality is the product of its N_{fail} and its unavailability [40]. Front-end *(FE)* components can cause an outage as defined in (2). If a dependent *(DeC)* can tolerate the outage *(OT)*, of its sponsor component *(SC)*, the latter's failure causes a degradation as defined in (3). Finally, the sponsor's failure causes a degradation and an outage as defined in (4) [39].

$$criticality_{FE} = \left(N_{fail} \times MTTR\right)_o \tag{2}$$

$$criticality_d = \left(N_{fail} \times MTTR\right)_d \tag{3}$$

$$criticality_{do} = \sum_{DeC} (Degradation + Outage)$$
$$where \begin{cases} Degradation = \left(\left(N_{fail}\right)_{SC} \times OT_{DeC}\right)_d \\ Outage = \left(\left(N_{fail}\right)_{SC} \times \left(MTTR_{SC} - OT_{DeC}\right)\right)_o \end{cases} \tag{4}$$

Once criticality is defined, the applications components are then inputted to the placement algorithm. The latter finds a pool of servers that satisfy the performance demands of the applications components (computation resources and latency). The servers pool is imported to the availability algorithm to find the best server while maximizing the HA of the applications components. To that end, the availability sub-algorithm is executed to select a server from the pool with the highest availability measure (highest MTTF and lowest MTTR). However, the chosen server should satisfy the delay, affinity, and anti-affinity constraints. As for the affinity constraints, the availability algorithm restricts the placement of a component and its redundant ones on the same server (geo-redundancy policy). It also places the dependent components on their sponsor server if they cannot tolerate the sponsor failure. Otherwise, the algorithm provides different locations for the sponsor component and its dependents. Fig. 7 shows the flowchart of the placement algorithm.

The HA-aware allocation algorithm generates the mapping between applications components and their hosts (servers and DCs). It is necessary to note that the VMs represent the applications' components. Prior to the simulation, the algorithm is

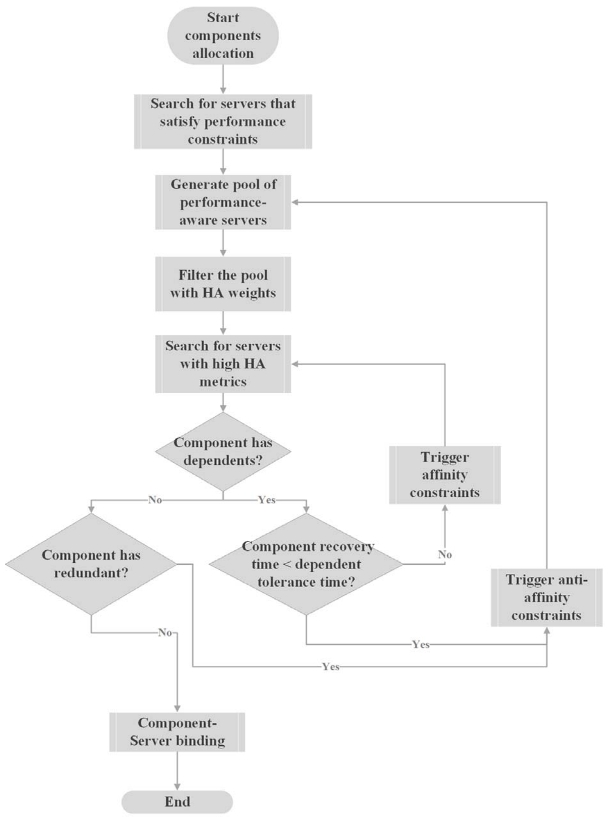

Start components allocation
Search for servers that satisfy performance constraints
Generate pool of performance-aware servers
Filter the pool with HA weights
Search for servers with high HA metrics
Component has dependents?
No
Yes
Trigger affinity constraints
Component has redundant?
Component recovery time < dependent tolerance time?
No
Trigger anti-affinity constraints
No
Yes
Yes
Component-Server binding
End

Fig. 7. Flowchart of the HA-aware placement algorithm in ACE.

executed, and the VM-host (component-host) is defined. To that end, the CloudSim broker is extended to include the VM-host binding at the beginning of the simulation. This extended broker class binds the VM to the required server in order to ensure that we can access the VM list of any server and the server of any VM, especially upon failure.

Load balancing module: A load balancing algorithm is added to CloudSim. At each tier of the computational path, a load balancer is responsible for the distribution of the requests between available VMs. A fair load balancing algorithm is implemented to ensure a fair workload distribution among different entities. This algorithm is implemented using weighted round robin technique [41]. First, the load balancer searches for active components (VMs) to process a request. Then it adds weights to the VMs having the least waiting queue size (least number of requests in its queue) to handle the workload. The load balancer does not only distribute the requests on the relevant VMs, but it is also responsible for the redistribution of requests upon failure of their corresponding VMs and/or hosts (servers and/or DCs). Fig. 6 shows the load balancing model of ACE for a 3-tier Web application.

Failure injection module: Each cloud entity is associated with availability measures. With MTTF and MTTR, the availability ratio is calculated using (1). The failure time is then determined by multiplying this ratio with the simulation duration. Once determined, a failure is injected into the simulation. The faulty entity is considered "destroyed", and its corresponding requests are redistributed to the redundants.

Recovery and repair modules: Once the failure is injected, the extended broker detects and isolates it to protect the rest

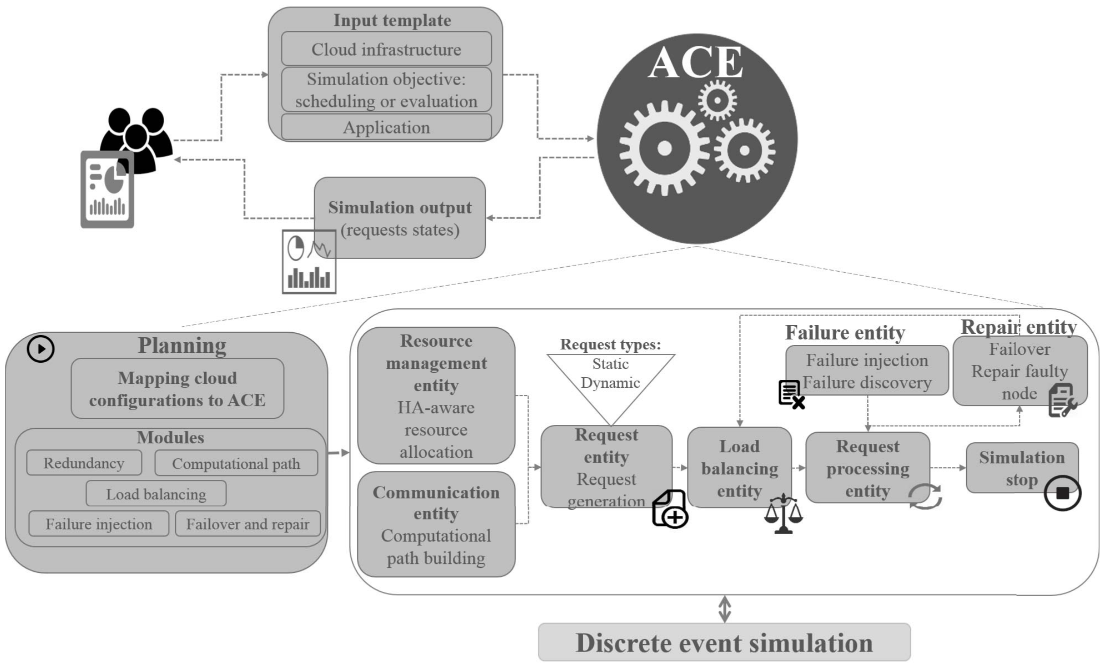


Fig. 8. ACE architecture and different modules.

of the cloud system. Simultaneously, it triggers the recovery and repair policies. If the failure happens at the VM, the latter's "recovery time" determines when to trigger the recovery policy, and the MTTR determines when the repair policy is launched. The broker triggers "DestroyVM" method to generate a failure and acknowledges the DC. To that end, the "CloudSim class" in the core engine is extended to include dynamic future queue where its size can be updated anytime due to any unplanned event (failure, recovery, and repair) during the simulation. Since we can access the host identifier (ID) from a VM, the broker iterates over the "VM HashMap" of the host of faulty VM and changes the latter status to inactive, which is simultaneously updated in the VM list of the load balancer. The broker then determines the cloudlet queue of the faulty VM, which is extracted from the scheduling policy. The cloudlet in the execution queue is considered "failed". As for the cloudlets (requests) in the waiting queue, they are released and associated with a "failed" flag. Concurrently, the broker calls the load balancer to determine the available active redundant VM with the least cloudlet queue size.

It is necessary to note that the broker is extended to act as the brain of the simulation and to ensure modularity and reusability of the code. For instance, any load balancer policy can replace the proposed one without affecting the simulation. Once the broker gets the apt redundant VM, it generates new cloudlets holding the same IDs as the old ones and triggers their failover to the corresponding VM. After VM repair time, the faulty VM is active again and ready to process new requests. If the failure happens at the level of the host, the broker iterates over its "VM HashMap" and repeats the previous VMs recovery and repair policies. After MTTR of the faulty server, its status changes to normal, and it is ready to host new VMs. Similarly, if the failure happens at the DC level, its servers fail automatically, and the same applies to their hosted VMs. The VMs and its cloudlets are recovered/repaired as discussed above. The faulty DC and its servers are considered healthy again after their MTTR. It is necessary to note we do not consider a hardware recovery policy (hardware redundancy), redundancy is only assigned to the applications' components.

B. ACE Building Blocks

This section explains the different classes used to extend CloudSim with ACE. Fig. 8 shows ACE architecture and its main classes. The user starts with generating the cloud scenarios using the *input template module*. Once the data is defined, the JSON template is added to the source file of ACE. When the user runs the simulation, ACE access the JSON template and populates its cloud model accordingly. Once the cloud model is populated, ACE executes the *HA-aware placement module* to deploy the application's components on the servers that satisfy the performance and HA-aware constraints. Once this module generates the placements, it maps the servers to their corresponding component using VMs. ACE executes then the *computational path and request generation module* to build the functional and protection groups within each type. It also executes the routine responsible for generating the requests (tasks) for the application's components. At this stage, the *load balancing module* is executed to distribute the requests between the redundant instances of each component type. Once the failure time is determined using the *failure injection module*, the latter injects failure and destroys the faulty node. Simultaneously, the *recovery and repair module* is executed to isolate the failure and trigger the recovery and repair policies. The *load balancing module* is re-executed to failover the workload to the redundant instance(s) of the faulty entity. Once the repair policy is executed and the faulty component is active again, the *load balancing module* is re-executed to distribute the request accordingly.

CloudletExtension: This class extends the Cloudlet class in CloudSim to reflect availability metrics (ID, the status of completion, and dependents/sponsors).

CloudletScheduler classes: These classes include CloudletSpaceSchedulerSpaceSharedExtension and CloudletTimeSchedulerSpaceSharedExtension, which are extended to release the resources of the faulty VM and ensure that it has its full resources when it becomes healthy.

Comp2CloudletAdapter: This class maps the components of the ACE input template (JSON template) to the extended cloudlet class (HA measures (MTTF, MTTR, tolerance time) and dependents/sponsors).

Comp2VMCloudSimAdapter: This class represents each application component with a VM in the simulation.

CP4ComponentsVMs: This class is used to determine the dependent(s) and/or sponsor(s) of each component type and generate the computational path or the functional chain of the corresponding applications.

CPStructure: This class determines the structure of the computational path where the application component type of the first tier of the path, application components of the path, and a number of active components in each tier are defined.

CreateCloudlet4VM: This class generates the cloudlets of each VM (a component of a specific type) where each cloudlet has the characteristics of its corresponding VM.

DCBrokerExtension: This class is considered the brain of ACE simulation. It is extended to include failure injection of VM/server/DC, dynamic generation of cloudlets/VMs and computational path, dynamic destruction of VM/server/DC upon failure, recovery, and repair policies. The broker is also extended to support static requests, dynamic requests, and fluctuated workload generation.

DCExtension: This class captures HA measures (MTTF and MTTR) of the DC and includes the acknowledgment for a VM failure and the binding between the VM and host according to the proposed HA-aware algorithm.

FT classes: These classes include DatacenterFailureTime, HostFailureTime, and VMFailureTime. They determine the data structure of the failure time of the DC, server, and VM.

DC2CloudSimDCAdapter: This class maps the DC of the ACE JSON template to the extended DC class in CloudSim.

HAUtilities: This class is used to calculate the time to inject the failure based on the simulation duration.

HostExtension: This class is extended to include HA measures of the server and its map of the hosted VMs.

IntroduceVMFailureAndRecovery: This class tracks the simulation time to inject failures and trigger recovery. This class considers failure priority in a sense that if DC, server, and VM fail at the same time, it triggers DC failure then host failure followed by VM failure. Also, if the MTTF is given same as MTTR of a VM, this class can handle this error. It will initially trigger a VM failure followed by a repair. This feature can be used to redistribute requests of a certain VM to its redundants upon its overload.

LB: This load balancing class distributes the requests to the active VMs at each tier of the computational path. It also redistributes the requests to the redundant VMs upon DC/server/VM failure.

PopulatingFromHAAllocator: This class generates the placements of the applications components on the best servers while maximizing the components HA.

RedundancyModelTags: This class defines the tags for redundancy types (active, standby, or spare) of each VM.

RequestStructure: This class determines the structure of the request where it defines the request unique ID, status, final request state, and the sub-cloudlets.

VMAllocationPolicyExtension: This class releases the resources of the faulty VM from its host.

VMExtension: This class is extended to capture HA measures of a VM, its component type, broker, host ID, and cloudlet.

CloudSim: It is one of the core engine classes of CloudSim. This class adds a dynamic update of the future queue. For instance, when the failure of an entity is injected during the simulation, a failure event is generated. This event should be added to the future queue of the DES, and consequently, the queue size should be updated accordingly.

V. ACE Evaluation

This section provides an evaluation of ACE to show the impact of availability metrics on the cloud performance. ACE is assessed on a three-tier Web application. Amazon Web application is an example [42]. The MTTF, MTTR, and recovery time the measures of the HA of deployed components (VMs), inject failures, and recover faulty nodes. It is necessary to note that the downtime of an application component C is calculated in terms of outage hours per year, and its availability A_C is calculated as follows [43]:

$$A_C = \left(\frac{8760 - downtime_C}{8760}\right) \times 100 \qquad (5)$$

In this section, the availability of each deployed VM (components) is measured in terms of outage hours per year, and the availability of a cloud scenario where its VMs are already deployed is measured in terms of a number of successfully processed requests. ACE can be used to test and evaluate different cloud-based objectives:

- Evaluate multiple availability and performance-aware allocation techniques.
- Assess the resiliency of cloud model under study in terms of different failures and recovery policies.
- Provide availability analysis of any cloud placement solution. The analysis does not only detect failures, their effects, and recovery/repair schemes, but it calculates the availability of a cloud model under various stochastic and deterministic events (failure, recovery, and overload).
- Assess the capability of each application component to process user requests under different configurations.
- Evaluate the impact of redundancy models and failures on the number of served requests, their response, and waiting time.
- Extract different HA-aware lessons to improve the cloud resiliency to failure in the future (anticipated elasticity).
- Model and evaluate different requests' distribution where ACE can model fixed number requests, workload

TABLE I
DIFFERENT HA METRICS DISTRIBUTION

HA measures	Distribution	Distribution metrics (hours)
MTTF	Exponential	μ=2500
Tolerance time	Exponential	μ=10
MTTR	Truncated Normal	μ=[0.05-3]; σ=[0.016-1]

fluctuation (different workload's distribution to model real-case scenarios, such as peak/normal periods), and automated generation of requests while defining their arrival rate *AR*.

ACE is implemented in Eclipse on a Linux VM with 26GB of RAM and 6 vCPUs running Ubuntu12.04. For all scenarios, simulations are run multiple times to define a confidence level of 95% based on the t-Table [44].

A. Modeling a Scenario in ACE

This section describes how to model a three-tier Web application on a network of 2 DCs, 2 racks, and 6 servers using GITS. GITS is implemented in Eclipse on a Linux VM running Ubuntu12.04. In order to create a cloud scenario for ACE, the user can either run the GMF project as a Java application or use the JSON template.

Each GITS template has an objective; "scheduling" of cloud applications or "evaluation" of a certain scheduling solution. We assume "scheduling" objective for this scenario. This section also assumes that each DC consists of a rack, 2 shelves, and 3 servers where each is associated with its own computational resources and availability attributes. Each DC is associated by an availability zone; i.e., "Z1". This zone means all the DC's servers are located in it. Since we have one DC with these characteristics, the *DC_count* is set to one. As for the rack, the user should define its resources, HA attributes, and the parent DC. Similarly, the shelves and servers are defined in the GMF or JSON template.

The VMs represent the mapping between the cloud infrastructure and the cloud applications. Since the scenario's objective is "scheduling", the VM's hosting server and the hosted component are set to "Null" as they are determined automatically after the execution of the application's placement algorithm.

As for the redundancy model, GITS support different models, such as active-active, active-standby, or active-spare. In this case, an active-active redundancy model is used. It is necessary to note that upon failure of a component, its workload's failover time is determined by the redundancy model. For example, if the redundancy model is active/standby, the failover time is the summation of the fetch state delay, parsing state delay, recuperation duration, execution time, and the termination duration. As for the dependency relation, it is defined by a delay tolerance (acceptable delay between these types) and a tolerance time (time that a dependent can tolerate upon the outage of its sponsor).

Since a 3-tier Web application is being modeled, the HTTPS depends on the App server and thus it has the *DependsON* property set as "App" and the *DepParam* is set according to the corresponding delay tolerance and tolerance time. Similarly, the App type is populated. As for the DB type, it does not have sponsors and thus its *DependsON* and *DepParam* are set to "Null". As for the redundancy model, it is defined using the *RedundancyModel*, *CompType_instances*, *names*, and *RedParam*. Each component has an SLA to be defined. The SLA is associated with the average request arrival rate, allowed outage time, and other HA and scheduling metrics.

TABLE II
COMPUTING METRICS

Scenario metrics	Server	VM
CPU (cores)	16-32 cores	1-2 core(s)
RAM	25-35 GB	256-1024 MB

B. ACE Configuration

Once the input is defined, the user should determine if the requests are fixed or dynamically generated during the simulation. For this purpose, the user can define the number of requests (Arrival Rate (*AR*)) arriving simultaneously at the active nodes. The user should also define the Simulation Duration (*SD*) to run certain scenario. The results are evaluated on a network of 3 DCs, 6 racks, and 70 servers. The MTTF of the cloud nodes is generated using an exponential distribution, and MTTR time is generated using truncated normal distribution [45], [46]. Different HA metrics are available online [47]–[49]. Table I and Table II show the different configuration metrics of the cloud scenario [50], [51].

To capture the intercommunication relations between applications' components (VMs), ACE is evaluated on a real-time 3-tier Web application. The redundancy model of this application is active/active where the number of active components is changed during the simulation to capture their impact.

It is necessary to note that given the same configuration of the cloud infrastructure and applications, the confidence level of the evaluation results of ACE exceeds 95%. In other words, ACE shows the same performance and generates consistent results using the same cloud settings (scenarios and attributes).

C. Results

In this section, ACE is evaluated to measure the following.

1) Redundancy Impact on Request States: The number of applications' components per type is changed to measure the impact of redundancy model on the number of served requests, their response, and waiting times. Different redundancy models are defined where 2-RED means that an application's component type has 2 active redundant components (i.e., an HTTP component type with 2-RED means it has 2 active redundant components HTTP1 and HTTP2), 3-RED means that an application's component type has 3 active components (i.e., an DB component type with 3-RED means it has 3 active redundant components DB1, DB2, and DB3), and 4-RED means that an application's component type has 4 active redundant components (i.e., an DB component type with 4-RED

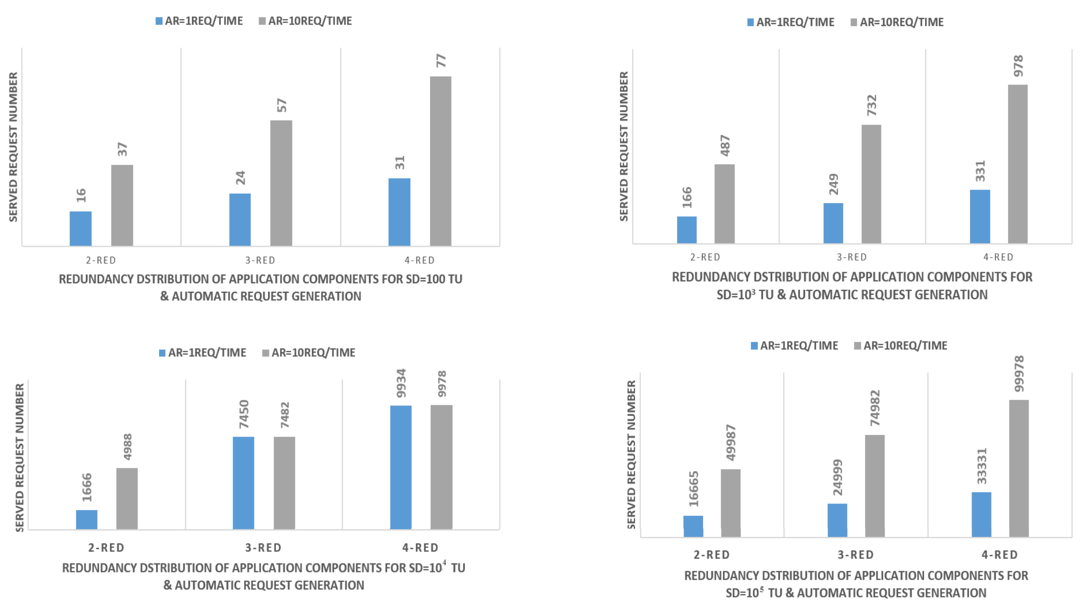


Fig. 9. Impact of redundancy models on the number of the served requests for automatic request generation.

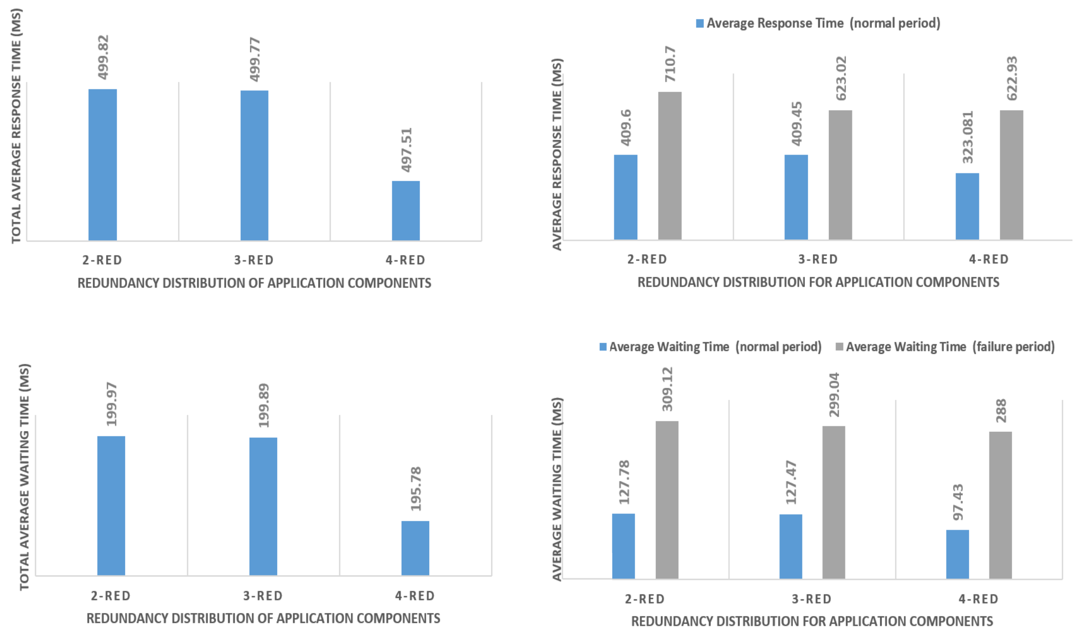


Fig. 10. Impact of the redundancy models on the request's response and waiting times.

means it has 4 active redundant components DB1, DB2, DB3, and DB4). The request states are evaluated under different *SD* and *AR* where $SD = \text{x } TU$ (x is simulation time measured in Time Unit (TU)) and $AR = \text{X}$ req/time (X request arrives at each active node). Fig. 9 shows the impact of the redundancy model on the number of served requests for different *SD* and *AR*. It is noticeable that the number of served requests increases as the number of components increases. For example, the system can serve 37 requests for 2-RED while 4-RED allows the serving of 77 requests under same $SD = 100\ TU$ and $AR = 10$ req/time. Changing the *SD* allows serving more requests, which increase from 77 to 99,978 for 4-RED.

To measure the impact of the redundancy model on the request response and waiting times upon failure, we define $AR = 10$req/time for $SD = 10^2\,TU$ while requests are dynamically generated as long as a healthy active component(s) are available. In Fig. 10, the total average response and waiting times of the requests are measured. Although a failure is injected to the system, the response and waiting times do not exceed the allowed response and waiting times (500 and 200 milliseconds (ms) respectively) [52]. It is noticeable that the response and waiting times decrease with the increase in the number of components per type. In this case and upon a failure, the requests failover, and the load balancer distributes the requests of a faulty node to its redundant. When the number of components per type increases, a wider request redistribution space is available, and consequently, its response and waiting times decrease. Fig. 10 shows these measures. The

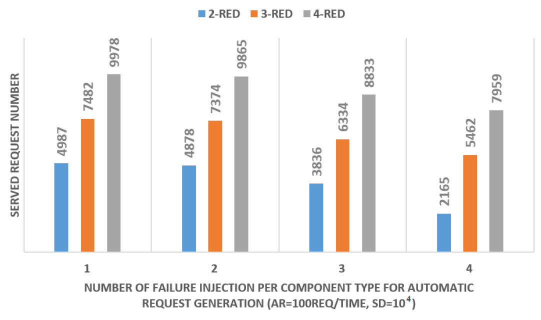


Fig. 11. Impact of the number of failure injections on the request number for different redundancy models.

average response and waiting times for the requests of the faulty node(s) represent those times during the outage states. Although these measures decrease as more components are added to a type, the requests' response and waiting times during the outage period might violate the acceptable times (response of 500 ms and waiting of 200 (ms)).

2) Failure Injection Impact on Request States: The number of failure injection per component type is changed to measure the impact of failures on the number of served requests, their response, and waiting times. The requests are evaluated under $SD = 10^4 TU$ and $AR = 100$req/time. Fig. 11 shows the impact of failure injection on the number of served requests. As more failures are injected per type, the number of served requests drops from 9,978 to 7,959 for 4-RED. Although a recovery solution is executed upon failure, the number of requests decreases in case of a faulty system because the requests' response and waiting times increase.

To measure the impact of the failure injection on the request response and waiting times, we define $AR = 10$req/time for $SD = 10^2 TU$ for 2-RED while requests are dynamically generated as long as a healthy active component(s) are available. In Fig. 12, the total average response and waiting times of the requests are measured. It is noticeable that the response and waiting times increase with the increase in the failure injections per type. For example, the response and waiting times for one failure injection/type are 499.77 and 199.64 ms respectively, which increase to 499.82 and 199.97 ms respectively for 3 failures/type. The numbers do not violate the allowed ones (response of 500 ms and waiting of 200 (ms)) [52]. In this case and upon failures, the load balancer redistributes the requests. When multiple failures are injected, the number of waiting requests for each node increases and consequently, causes an increase in the response and waiting times. Scaling up the system can be a solution as shown above. Fig. 12 shows these measures. For the 2-RED scenario under study, the average response and waiting times of the requests of faulty nodes increase from 703.66 and 309.12 to 725.49 and 362.23 ms respectively and thus violate the SLA.

3) Availability of Deployed Components: The HA-aware placement algorithm is executed to place the components on the servers while maximizing their availability, which is measured using (1). Fig. 13 shows the availability of the components of the three-tier Web application where each tier consists of 3 components/type. The availability of different components ranges between three to four nines of availability. The proposed algorithm prioritizes the component types to ensure that mission-critical applications are given the priority to be allocated first [39]. This allocation results in the change of the availability nines where high-priority components are placed on the servers that guarantee the highest HA.

4) ACE Scalability: In order to evaluate the ability of ACE to model a real-cloud scenario, we use the Google public dataset on workload traces [53]. These traces are taken from a Borg cell over a period of 7 hours. The data has a set of tasks where each is characterized by its consumed memory and CPU cores. Each task is associated with its parent; in this case, we assume the parent is the component type of an application. A parent (or component type) can have multiple tasks (requests). It is necessary to note that the dataset has been anonymized where the tasks have a numeric IDs instead of names and the CPU/memory requirements are determined using a linear transformation. The data consists of the following:

- Time (integer) - time (seconds) since the start of data collection
- parentID (integer) - unique ID of the job
- TaskID (integer) - unique ID of the task
- Type (0, 1, 2, 3) - class of job
- Normalized Task Cores (float) - normalized value of the average number of cores
- Normalized Task Memory (float) - normalized value of the average memory

Since a 3-tier Web application is used in our case, we assume that there are three job types (HTTPS, DB, and App).

Generally, Amazon's DCs host 50,000 to 80,000 servers [54] and Google's DC hosts ~70,000 servers [55]. However, due to the time complexity and limited computational resources, the Google workload traces are tested on 2-RED model and a network of 3 DCs, 6 racks, and 70 servers.

Different simulations are performed to measure the number of requests ACE can process during different *SDs* given this scenario. These simulations are executed on 2-RED model. Fig. 14 shows the scalability of ACE. For 2-RED case, ACE can process 42 requests for $SD = 100TU$ to reach ~four million requests for $SD = 10^7 TU$. With these experiments, it is noticeable that ACE can model and simulate multiple cloud scenarios where thousands of requests are processed.

Although ACE can simulate a high number of requests, the time for simulating such cases on a limited resources environment can be a hiccup. The simulation time increases with the increase in the SD and the number of served requests. For $SD = [100 - 10^7 TU]$, the 2-RED time reaches ~7,348,255 ms.

D. Discussion on ACE Fault-Tolerant Approaches

This section describes how ACE can be used to support different fault-tolerant approaches.

Reactive fault-tolerance approach: ACE supports different redundancy models as a reactive fault-tolerant policy. Using ACE, the user can define any type of redundancy model (active/active, active/standby,...) to reduce the failures' impact

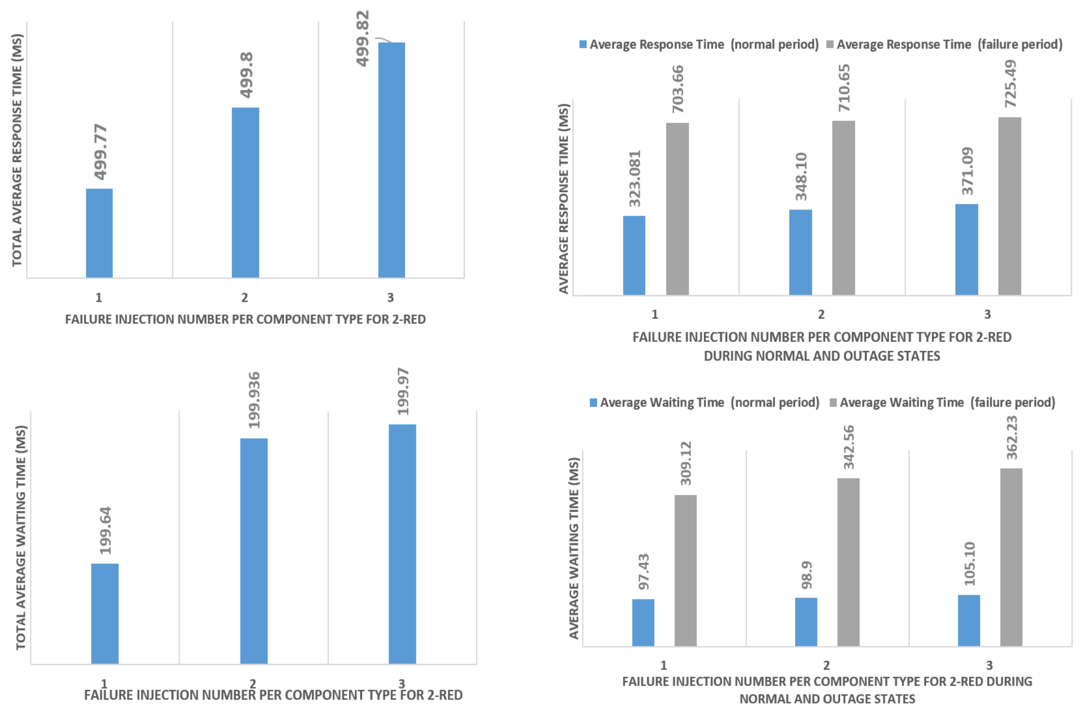


Fig. 12. Impact of the number of failure injections on the request's response and waiting times.

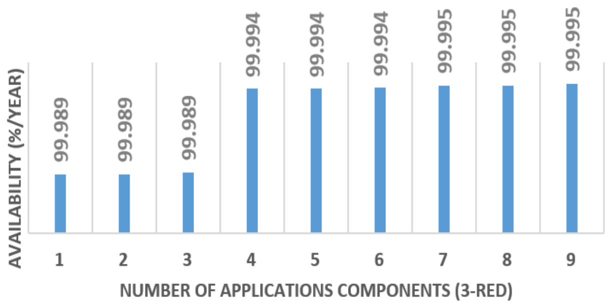


Fig. 13. Availability of each deployed component.

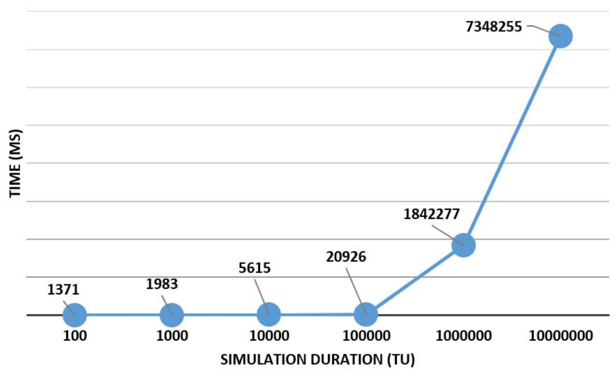


Fig. 15. ACE time complexity for 2-RED model.

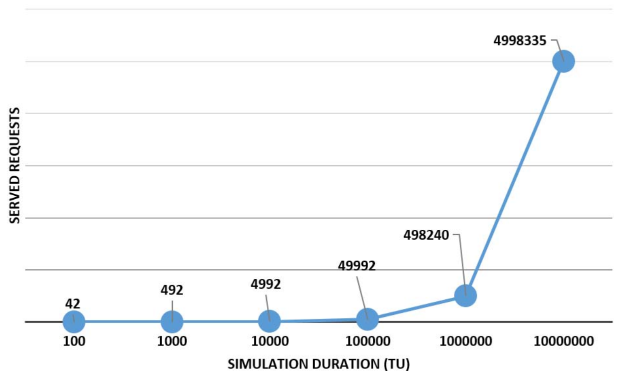


Fig. 14. ACE scalability: Number of request processed using ACE for 2-RED model.

on the execution of the application instances. The reactive fault-tolerant approach can be extended to include elasticity approach where the redundant instances can be scaled up or down based on the examined performance-aware objective.

Proactive fault-tolerant approach: ACE implements a load balancing approach as a proactive fault-tolerant model. The workload is distributed among the redundant instances of certain component type to minimize the faults' recovery and failures. It is necessary to note that a migration approach can be easily implemented in ACE as another proactive fault-tolerant approach.

Adaptive fault-tolerant approach: At this stage, ACE implements an HA-aware placement solution as an adaptive fault-tolerant approach. The placement solution selects the best deployments of the application's components depending on their current state and other functional (computational resources, delay..) and non-functional (availability; i.e., redundancy, interdependency...) constraints. The redundancy model type is one of the input data of ACE; therefore, this approach can be easily extended to an algorithm that runs in parallel with the simulation, monitor the application's components' states and (de)allocate redundant instances according to the criticality and other performance requirements.

VI. Conclusion

Providing a resilient cloud is imperative to underpin enterprises availability and performance requirements. It is of great importance to design an approach that does not only provide HA-aware placements of applications but also assess the cloud elasticity and provide the necessary HA-based lessons to improve the services' availability. Simulation tools are one of the best ways to model the cloud and simulate it in terms of multiple QoS objectives. With this in mind, we extended CloudSim simulator with ACE to include HA properties in a sense that failures can be injected and recovered from. To that end, we proposed a JSON template, GITS, to generate cloud scenarios while keeping the development complexities behind the scene. GITS does not only model the cloud, but it captures different HA properties. ACE implements these properties in CloudSim. Once the simulation starts, ACE generates HA-aware placements of different cloud applications and builds the application functional chain to capture dependency and redundancy. It also injects failures, provides failover using redundancy models, and repairs the faulty nodes. Also, it provides a load balancing algorithm to distribute the dynamic and static requests. In the future work, ACE will be extended to include elasticity features where scaling up and down will be implemented to overcome failures and meet performance requirements.

References

[1] TechRepublic. (Jan. 31, 2017). *The State of IaaS: Growing as Cloud Adoption Continues*. Accessed: Feb. 15, 2017. [Online]. Available: http://www.techrepublic.com/article/the-state-of-iaas-growing-as-cloud-adoption-continues/?ftag=TRE9ae7a1a&bhid=25335435715452818046449410599561

[2] Chargify. (Feb. 7, 2017). *The Future is XaaS: What You Need to Know About Everything-as-a-Service*. Accessed: Feb. 19, 2017. [Online]. Available: https://www.chargify.com/blog/xaas-everything-as-a-service/

[3] L. Wang, R. Ranjan, J. Chen, and B. Benatallah, *Cloud Computing, Methodology, Systems, and Applications*. Boca Raton, FL, USA: CRC Press, Oct. 2011. [Online]. Available: http://www.infosys.tuwien.ac.at/Staff/sd/papers/Buchbeitrag%20V.%20Emeakaroha.pdf

[4] WHIR Hosting Cloud. *GitLab's Not Alone: AWS, Google, and Other Clouds Can Lose Data, Too*. Accessed: Feb. 15, 2017. [Online]. Available: http://www.thewhir.com/web-hosting-news/gitlabs-not-alone-aws-google-and-other-clouds-can-lose-data-too?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed%3A+thewhir+%28theWHIR.com+-+Daily+Web+Hosting+News%2C+Features%2C+Blogs+and+more%29

[5] ZDNet. *Dropbox Sync Glitch Results in Lost Data for Some Subscribers*. Accessed: Oct. 13, 2017. [Online]. Available: http://www.zdnet.com/article/dropbox-sync-glitch-results-in-lost-data-for-some-subscribers/

[6] InformationWeek. (May 14, 2014.) *Social Science Site Using Azure Loses Data*. Accessed: Dec. 9, 2016. [Online]. Available: http://www.informationweek.com/cloud/cloud-storage/social-science-site-using-azure-loses-data/d/d-id/1252716

[7] Computer World. (Aug. 20, 2015). *OOPS: Google 'Loses' Your Cloud Data (Sky Falling; Film at 11)*. Accessed: Dec. 9, 2016. [Online]. Available: http://www.computerworld.com/article/2973600/cloud-computing/google-cloud-loses-data-belgium-itbwcw.html

[8] BusinessInsider. (Apr. 28, 2011). *Amazon's Cloud Crash Disaster Permanently Destroyed Many Customers' Data*. Accessed: Jan. 11, 2017. [Online]. Available: http://www.businessinsider.com/amazon-lost-data-2011-4

[9] H. Hawilo, A. Kanso, and A. Shami, "Towards an elasticity framework for legacy highly available applications in the cloud," in *Proc. IEEE World Congr. Services (SERVICES)*, Jul. 2015, pp. 253–260.

[10] S. Ayoubi, Y. Zhang, and C. Assi, "A reliable embedding framework for elastic virtualized services in the cloud," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 3, pp. 489–503, Sep. 2016.

[11] H. Hawilo, A. Shami, M. Mirahmadi, and R. Asal, "NFV: State of the art, challenges, and implementation in next generation mobile networks (vEPC)," *IEEE Netw.*, vol. 28, no. 6, pp. 18–26, Dec. 2014.

[12] S. Ayoubi, Y. Chen, and C. Assi, "Towards promoting backup-sharing in survivable virtual network design," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 3218–3231, Oct. 2016.

[13] M. Toeroe and F. Tam, *Service Availability: Principles and Practice*. Hoboken, NJ, USA: Wiley, May 2012. [Online]. Available: http://ca.wiley.com/WileyCDA/WileyTitle/productCd-1119954088.html

[14] A. Boteanu and C. Dobre, "A simulation model for fault tolerance evaluation," *Sci. Bull.*, vol. 73, no. 1, pp. 13–26, 2011.

[15] B. Wickremasinghe, R. N. Calheiros, and R. Buyya, "CloudAnalyst: A CloudSim-based visual modeller for analysing cloud computing environments and applications," in *Proc. 24th IEEE Int. Conf. Adv. Inf. Netw. Appl.*, 2010, pp. 446–452.

[16] D. Kliazovich, P. Bouvry, Y. Audzevich, and S. U. Khan, "GreenCloud: A packet-level simulator of energy-aware cloud computing data centers," in *Proc. IEEE Glob. Telecommun. Conf. (GLOBECOM)*, 2010, pp. 1–5.

[17] S. K. S. Gupta *et al.*, "GDCSim: A tool for analyzing Green Data Center design and resource management techniques," in *Proc. Int. Green Comput. Conf. Workshops*, 2011, pp. 1–8.

[18] A. Zhou, S. Wang, Q. Sun, H. Zou, and F. Yang, "FTCloudSim: A simulation tool for cloud service reliability enhancement mechanisms," in *Proc. Middleware Posters Demos Track*, Dec. 2013, p. 2.

[19] M. Tighe, G. Keller, M. Bauer, and H. Lutfiyya, "DCSim: A data centre simulation tool for evaluating dynamic virtualized resource management," in *Proc. 8th Int. Conf. Netw. Service Manag. (CNSM) Workshop Syst. Virtual. Manag.*, 2012, pp. 385–392.

[20] S. Ostermann, K. Plankensteiner, R. Prodan, and T. Fahringer, "GroudSim: An event-based simulation framework for computational grids and clouds," in *Proc. Euro Par Parallel Process. Workshops*, Aug. 2010, pp. 305–313.

[21] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw. Pract. Exp.*, vol. 41, no. 1, pp. 23–50, 2011.

[22] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. De Rose, and R. Buyya, "CloudSim: A novel framework for modeling and simulation of cloud computing infrastructure and services," GRIDS Lab., Univ. Melbourne, Melbourne, VIC, Australia, Rep. GRIDS-TR-2009-1, 2009.

[23] R. Buyya, R. Ranjan, and R. N. Calheiros, "Modeling and simulation of scalable cloud computing environments and the CloudSim toolkit: Challenges and opportunities," in *Proc. Int. Conf. High Perform. Comput. Simulat.*, 2009, pp. 1–11.

[24] W. Zhao, Y. Peng, F. Xie, and Z. Dai, "Modeling and simulation of cloud computing: A review," in *Proc. IEEE Asia–Pac. Cloud Comput. Congr.*, 2012, pp. 20–24.

[25] M. Jammal, A. Kanso, P. Heidari, and A. Shami, "Availability analysis of cloud deployed applications," in *Proc. IEEE Int. Conf. Cloud Eng. (IC2E)*, Apr. 2016, pp. 234–235.

[26] M. Jammal, A. Kanso, P. Heidari, and A. Shami, "A formal model for the availability analysis of cloud deployed multi-tiered applications," in *Proc. 3rd IEEE Int. Symp. Softw. Defined Syst.*, Apr. 2016, pp. 82–87.

[27] M. Jammal, H. Hawilo, A. Kanso, and A. Shami, "Mitigating the risk of cloud services downtime using live migration and high availability-aware placement," in *Proc. IEEE Int. Conf. Cloud Comput. Technol. Sci. (CloudCom)*, Dec. 2016, pp. 578–583.

[28] E. Bauer and R. Adams, *Reliability and Availability of Cloud Computing*. Piscataway Township, NJ, USA: Wiley, 2012. [Online]. Available: https://onlinelibrary.wiley.com/doi/book/10.1002/9781118393994

[29] K. Bilal *et al.*, "Fault tolerance in the cloud," in *Encyclopedia of Cloud Computing*. Chichester, U.K.: Wiley, May 2016. [Online]. Available: http://sameekhan.org/pub/B_K_2015_BC_MB.pdf

[30] S. Loveland, E. M. Dow, F. LeFevre, D. Beyer, and P. F. Chan, "Leveraging virtualization to optimize high-availability system configurations," *IBM Syst. J.*, vol. 47, no. 4, pp. 591–604, 2008.

[31] A. Agbaria and R. Friedman, *Overcoming Byzantine Failures Using Checkpointing*, Univ. Illinois at Urbana–Champaign, Champaign, IL, USA, 2003. [Online]. Available: https://www.perform.illinois.edu/Papers/USAN_papers/03AGB02.pdf

[32] I. Koren and C. M. Krishna, *Fault Tolerant Systems*, Elsevier, Amsterdam, The Netherlands, 2007. [Online]. Available: ftp://doc.nit.ac.ir/cee/y.baleghi/Fault-Tolerant/Books/fault-tolerant-systems.pdf

[33] H. A. Alameddine, S. Sebbah, and C. Assi, "On the interplay between network function mapping and scheduling in VNF based networks: A column generation approach," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 4, pp. 860–874, Dec. 2017.

[34] M. Khabbaz, K. Shaban, and C. Assi, "Delay-aware flow scheduling in low latency enterprise datacenter networks: Modeling and performance analysis," *IEEE Trans. Commun.*, vol. 65, no. 5, pp. 2078–2090, May 2017.

[35] S. Ayoubi, S. Sebbah, and C. Assi, "A Cut-and-solve based approach for the VNF assignment problem," *IEEE Trans. Cloud Comput.*, to be published.

[36] J. Weinberg. (2005). *Job Scheduling on Parallel Systems*. [Online]. Available: http://cseweb.ucsd.edu/∼j1weinberg/papers/weinberg06researchExam.pdf

[37] K. Jeffery and B. Neidecker-Lutz, "The future of cloud computing opportunities for European cloud computing beyond 2010," Eur. Commission Inf. Soc. Media, Rep., 2012. [Online]. Available: http://cordis.europa.eu/fp7/ict/ssai/docs/executivesummary-forweb_en.pdf

[38] A. E. Elsanhouri, M. A. Ahmed, and A. H. Abdullah, "Cloud applications versus Web applications: A differential study," in *Proc. 1st Int. Conf. Commun. Comput. Netw. Technol.*, 2012, pp. 31–36.

[39] M. Jammal, A. Kanso, and A. Shami, "CHASE: Component high availability scheduler in cloud computing environment," in *Proc. IEEE Int. Conf. Cloud Comput. (CLOUD)*, 2015, pp. 477–484.

[40] Reliability HotWire. (2017). *Basic Concepts of FMEA and FMECA*. Accessed: May 1, 2017. [Online]. Available: http://www.weibull.com/hotwire/issue46/relbasics46.htm

[41] KEMP Application Delivery. (2018). *Load Balancing Algorithms*. Accessed: May 22, 2018. [Online]. Available: https://kemptechnologies.com/load-balancer/load-balancing-algorithms-techniques/

[42] Amazon Web Services. (Sep. 2010). *AWS Template Format*. Accessed: Mar. 2016. [Online]. Available: https://s3-us-west-2.amazonaws.com/cloudformation-templates-us-west-2/AutoScalingMultiAZWithNotifications.template

[43] M. Jammal, A. Kanso, and A. Shami, "High availability-aware optimization digest for applications deployment in cloud," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2015, pp. 6822–6828.

[44] T table. (2007). *t-Table*. Accessed: Dec. 2016. [Online]. Available: http://www.sjsu.edu/faculty/gerstman/StatPrimer/t-table.pdf

[45] Reliability HotWire. (Sep. 2007). *Availability and the Different Ways to Calculate It*. Accessed: Sep. 20, 2016. [Online]. Available: http://www.weibull.com/hotwire/issue79/relbasics79.htm

[46] EventHelix. (2014). *System Reliability and Availability*. Accessed: Sep. 20, 2016. [Online]. Available: http://www.eventhelix.com/RealtimeMantra/FaultHandling/system_reliability_availability.htm#.WKz9jVUrKUk

[47] C. Cerin *et al.* (Jun. 2013). *Downtime Statistics of Current Cloud Solutions*. [Online]. Available: http://iwgcr.org/wp-content/uploads/2013/06/IWGCR-Paris.Ranking-003.2-en.pdf

[48] The Availability Digest. (Jan. 2015). *Comparing Clouds With CloudHarmony*. Accessed: Sep. 20, 2016. [Online]. Available: http://www.availabilitydigest.com/public_articles/1001/cloud_comparisons.pdf

[49] CloudHarmony. (2017). *CloudSquare Service Status*. Accessed: Feb. 13, 2017. [Online]. Available: https://cloudharmony.com/status-1year

[50] P. Garraghan, P. Townend, and J. Xu, "An empirical failure-analysis of a large-scale cloud computing environment," in *Proc. IEEE 15th Int. Symp. High Assurance Syst. Eng.*, 2014, pp. 113–120.

[51] C. Jaiswal. *DBHAaaS—Database High Availability as a Service for Cloud Computing*, Univ. Missouri, Columbia, MO, USA, 2016. [Online]. Available: https://mospace.umsystem.edu/xmlui/bitstream/handle/10355/50815/Dissertation_2016_Jaiswal.pdf?sequence=1&isAllowed=y

[52] InfoQ. (Nov. 2015). *Real-time Data Processing in AWS Cloud*. Accessed: Jan. 20, 2017. [Online]. Available: https://www.infoq.com/articles/real-time-data-processing-in-aws-cloud

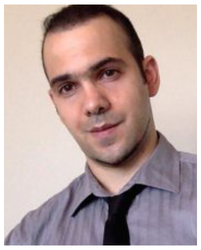
[53] Cluster-Data. (Aug. 2015). *Google Cluster Data, Trace Version 1*. Accessed: Jun. 12, 2018. [Online]. Available: https://github.com/google/cluster-data/blob/master/TraceVersion1.md

[54] R. Miller. (Sep. 2015). *Inside Amazon's Cloud Computing Infrastructure*. Accessed: May 24, 2018. [Online]. Available: https://datacenterfrontier.com/inside-amazon-cloud-computing-infrastructure/

[55] DataCenter Knowledge. (Aug. 2011). *Report: Google Uses About 900,000 Servers*. Accessed: Jul. 11, 2018. [Online]. Available: http://www.datacenterknowledge.com/archives/2011/08/01/report-google-uses-about-900000-servers

Manar Jammal received the B.Sc. degree in electrical and computer engineering from Lebanese University, Beirut, Lebanon, in 2011, the M.E.Sc. degree in electrical and electronics engineering from the Ecole Doctorale des Sciences et de Technologie, Beirut, and the University of Technology of Compiègne, France, in 2012, and the Ph.D. degree in High Availability of Cloud Applications from Western University, London, ON, Canada, in 2017, where she is a Post-Doctoral Associate. Her research interests include cloud computing, virtualization, high availability, cloud simulators, machine learning, software defined network, and virtual machine migrations. She is the Chair of IEEE Canada Women in Engineering and the Past Chair of IEEE Women in Engineering, London, ON, Canada.

Hassan Hawilo received the B.E. degree in communication and electronics engineering from Beirut Arab University, Lebanon, in 2012 and the M.E.Sc. degree in computer and software engineering from Western University, Canada, in 2015, where he is currently pursuing the Ph.D. degree in computer networks and cloud computing virtualization technologies. His research interests include cloud computing, virtualization technologies, software defined network, network function virtualization, distributed systems, and highly available software. He is the Vice Chair of IEEE London Computer Society, London, ON, Canada.



Ali Kanso received the master's and Ph.D. degrees in electrical and computer engineering from Concordia University, Montreal, Canada, in 2008 and 2012, respectively. He is a Senior Cloud Software Engineer with IBM T. J. Watson Research Center working on the IBM next generation container Cloud. He is also an Adjunct Research Professor with Western University. He holds to his credit over 50 publications including 12 patents granted and several pending. He was a Senior Researcher with Ericsson research Cloud technologies. He has over a decade of industrial research experience where his research interests are focused on distributed systems and lightweight virtualization in cloud computing environments.

Abdallah Shami (SM'09) received the B.E. degree in electrical and computer engineering from Lebanese University, Beirut, Lebanon, in 1997 and the Ph.D. degree in electrical engineering from the Graduate School and University Center, City University of New York, New York, in 2002. In 2002, he joined the Department of Electrical Engineering, Lakehead University, Thunder Bay, ON, Canada, as an Assistant Professor. Since 2004, he has been with Western University, Canada, where he is currently a Professor with the Department of Electrical and Computer Engineering. His current research interests are in the area of network optimization, cloud computing, and wireless networks. He is an Editor of IEEE COMMUNICATIONS SURVEYS AND TUTORIALS and has served on the editorial board of IEEE COMMUNICATIONS LETTERS from 2008 to 2013. He has chaired key symposia for IEEE GLOBECOM, IEEE ICC, IEEE ICNC, and ICCIT. He is an IEEE Distinguished Lecturer and is the Elected Chair of the IEEE London Section and the Chair of IEEE Communications Society.