Applying Computational Intelligence for Enhancing the Dependability of Multi-Cloud Systems Using Docker Swarm

Nitin Naik
Defence School of Communications and Information Systems
Ministry of Defence, United Kingdom
Email: nitin.naik100@mod.uk

***Abstract*—Multi-cloud systems have been gaining popularity due to the several benefits of the multi-cloud infrastructure such as lower level of vendor lock-in and minimize the risk of widespread data loss or downtime. Thus, the multi-cloud infrastructure enhances the dependability of the cloud-based system. However, it also poses many challenges such as non-standard and inherent complexity due to different technologies, interfaces, and services. Consequently, it is a challenging task to design multi-cloud dependable systems. Virtualization is the key technology employed in the development of cloud-based systems. Docker has recently introduced its container-based virtualization technology for the development of software systems. It has newly launched a distributed system development tool called Swarm, which allows the development of a cluster of multiple Swarm nodes on multiple clouds. Docker Swarm has also incorporated several dependability attributes to support the development of a multi-cloud dependable system. However, making Swarm cluster always available requires minimum three active manager nodes which can safeguard one failure. This essential condition for the dependability is one of the main limitations because if two manager nodes fail suddenly due to the failure of their hosts, then Swarm cluster cannot be made available for routine operations. Therefore, this paper proposes an intuitive approach based on Computational Intelligence (CI) for enhancing its dependability. The proposed CI-based approach predicts the possible failure of the host of a manager node by observing its abnormal behaviour. Thus, this indication can automatically trigger the process of creating a new manager node or promoting an existing node as a manager for enhancing the dependability of Docker Swarm.**

Keywords*—*Computational Intelligence, CI, Fuzzy Inference, Dependability, Docker Swarm, Multi-Cloud, Container, Virtual Machine, VM, Availability

I. Introduction

Cloud computing is one of the biggest boons for small businesses and ordinary users to cope with their growing or fluctuating demands for the IT infrastructure by driving down their cost. Multi-cloud infrastructure is another facet of the cloud computing, which extends its benefits and resolves some of the early issues [1], [2]. One of the greatest benefits of the multi-cloud infrastructure is the enhancement of dependability of the cloud-based system. Dependability is a collective term which includes several attributes such as availability, reliability, maintainability, safety, integrity and confidentiality. Depending on the system, different emphasis may be given to the different attributes of the dependability [3]. The dependability of a cloud-based system is mostly dependent on the technology employed in the design [4]. Virtualization is the key technology employed in the development of cloud-based systems. However, the requirement of significant resources and issues of interoperability and deployment across multiple clouds are major concerns for system developers [5], [6].

Docker container-based virtualization has recently emerged as an alternate lightweight technology for the development of cloud-based systems and gaining popularity in the cloud industry [7], [8]. Docker offers the ability to package applications and their dependencies into lightweight containers that move easily between different distros, start up quickly and are isolated from each other [8], [9]. It aims to address the challenges of resource, speed and performance of virtualization in the system development process [1], [10], [11]. In addition to all these facilities, its greatest advantage is that it provides developer's workflow [12]. Thus, Docker Containers-as-a-Service (CaaS) platform empowers developers and sysadmins to build, ship and run distributed applications anywhere [13]. Docker has newly launched a distributed system development tool called Swarm, which allows the development of a cluster of multiple Swarm nodes on multiple clouds [14]. Docker Swarm has also incorporated several dependability attributes to support the development of a multi-cloud dependable system. However, Docker Swarm-based dependable system development is a new approach for the cloud industry, and making Swarm cluster always available requires minimum three active manager nodes which can safeguard one failure. This essential condition for the dependability is one of the main limitations because if two manager nodes fail suddenly due to the failure of their hosts, then Swarm cluster cannot be made available for routine operations.

All the Swarm nodes (managers and workers) are created on lightweight Virtual Machines (VMs). VMs may compete for resources, or their behaviour may change drastically due to an unexpected failure of the host machine. Therefore, the manager's availability issue can be resolved externally by observing the behaviour of lightweight VMs used by all Swarm manager nodes. This can be done by designing a CI-based intelligent system. Therefore, firstly, this paper presents the simulation of a multi-cloud system using Docker Swarm for evaluating its dependability feature. This simulation and evaluation of the dependable system are based on Docker Swarm, VirtualBox and Mac OS X. However, the same dependable system can be easily created on any of the Docker supported cloud by just selecting the appropriate

driver name such as Amazon Web Services, Microsoft Azure, Digital Ocean, Google Compute Engine, Exoscale, Generic, OpenStack, Rackspace, IBM Softlayer, VMware vCloud Air [15]. This multi-cloud dependable system can be created on the above clouds, but it must require a valid subscription account on those clouds. Finally, this paper proposes an intuitive approach based on Computational Intelligence (CI) for enhancing its dependability. The proposed CI-based approach predicts the possible failure of the host of a manager node by observing its abnormal behaviour. Thus, this indication can automatically trigger the process of creating a new manager node or promoting an existing node as a manager for enhancing the dependability of Docker Swarm.

The remainder of this paper is organised as follows: Section II explains the theoretical background of Docker Containers, Docker Swarm, fuzzy reasoning and **R** software; Section III illustrates the architecture of a multi-cloud system using Docker Swarm; Section IV presents an experimental simulation of a multi-cloud system using Docker Swarm; Section V presents an experimental evaluation of dependability (in particular availability) of a multi-cloud system using Docker Swarm; Section VI proposes a CI-based solution for enhancing the dependability of a multi-cloud system using Docker Swarm; Section VII concludes the paper and suggests some future areas of extension.

II. Theoretical Background

A. Docker Containers

Containers provide an isolated environment akin to virtualization but without virtual hardware emulation [1]. The concept of a container is quite old in computing; however, they never employed at large-scale. Containers run in user space on top of an operating system's kernel; therefore, container virtualization is known as OS-level virtualization [16]. Container technology allows multiple isolated user space instances to be run on a single host [16]. Containers can also be classified into two categories: *system containers* and *application containers*. A system container is similar to a full OS and runs all process such as *init, inetd, sshd, syslogd,* and *cron*. Whereas, an application container only runs an application. Both types of container are useful in different circumstances [17]. There are so many popular container technologies available such as OpenVZ, LXC, Solaris Zones, systemd-nspawn, lmctfy and Warden [16], [17]. Docker technology is an example of container virtualization.

Docker is an open-source engine that automates the deployment of applications into containers [16]. Docker is developed by Docker, Inc., which was previously known as dotCloud, Inc. that was one of the pioneering company in Platform-as-a-service market. Docker provides an isolated container (see Fig. 1) based on a major Linux kernel feature known as *cgroups* and *namespace*. Consequently, each container can have strong isolation, own network stack, storage stack, file system and resource management capabilities to allow friendly co-existence of multiple containers on a single host [16]. A container does not have its own operating system as it shares the same kernel; however, it contains all binaries and libraries to run an application inside it as shown in Fig. 1.

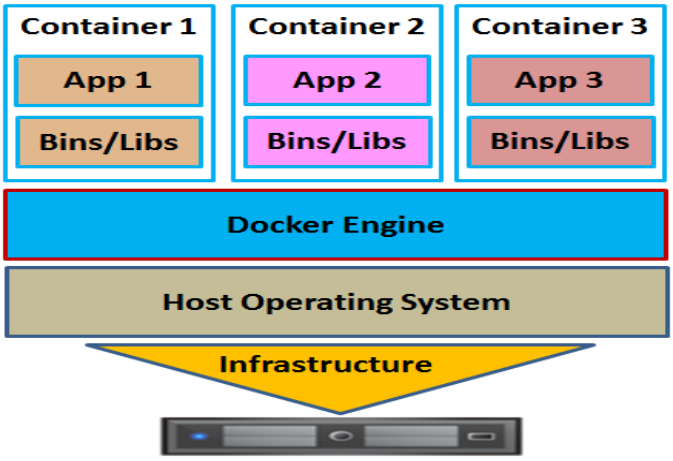

Fig. 1. Docker Containers on Linux Host

B. Docker Swarm

Distributed applications need distributed system with predictable dependability. Docker Swarm is a clustering tool, which offers functionalities to turn a group of Docker Nodes into a single dependable system [18]. It builds a cooperative group of systems that can provide redundancy if one or more nodes fail for high availability and reliability. Swarm also provides workload balancing and maintainability for containers and their services. It assigns containers to underlying nodes and optimizes resources by the automatically scheduling of container workloads to run on the most appropriate host with adequate resources while maintaining necessary performance levels [19]. Thus, the Docker Swarm-based dependable system is a *fault tolerant, highly available and reliable system.*

C. Docker Containers on Non-Linux Host

Docker is originally designed for Linux-based machines, where the host Linux OS also plays a part of Docker Host. The implementation of Docker containers on a non-Linux host is different from the Linux-based implementation. In non-Linux OS-based systems, Docker needs an additional component called *Docker Host* as shown in Fig. 2. Docker Host is a lightweight VM, which requires very few resources and operational overheads. The Docker Engine runs inside this Linux VM called "default" as shown in Fig. 3. The great success of Docker is this small VM, which runs completely in RAM and loads only in few seconds. This is largely due to its minuscule size (i.e., 35 MB in this implementation).

D. Fuzzy Reasoning

Fuzzy reasoning is the process of deriving logical conclusions from an existing fuzzy rule base [20]. It mimics the ability of the human mind to summarize data and focus on decision-relevant information [21]. Fuzzy reasoning is more effective and useful for those systems where a system cannot be defined in precise mathematical terms or models due to uncertainties, unpredicted dynamics and other unknown phenomena [22]. In many computing applications when data is incomplete and imprecise in nature; fuzzy reasoning is comparatively more suitable than other types of reasoning approaches [23], [24], [25], [26]. Fuzzy reasoning is based on a

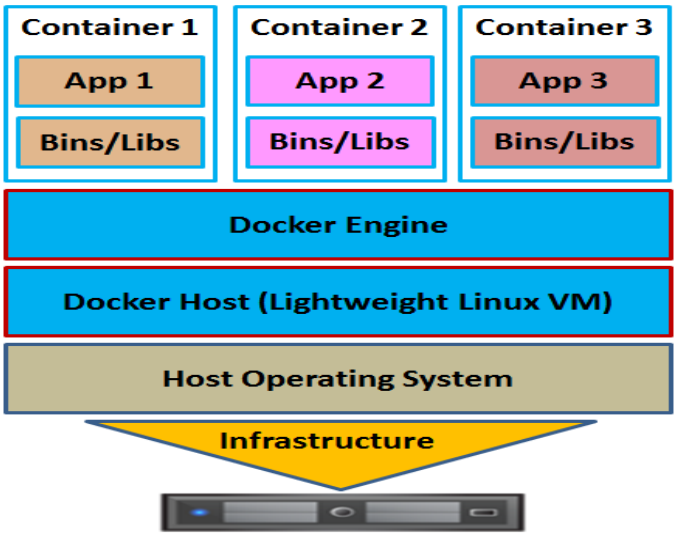

Fig. 2. Docker Containers on Non-Linux Host

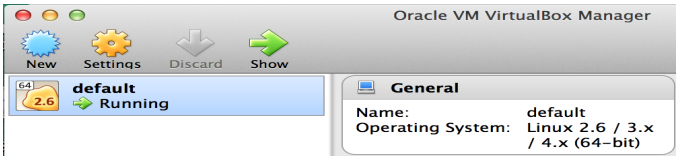

Fig. 3. Docker Engine running in "default" Virtual Machine

fuzzy rule base, and it can be derived by subject matter experts or extracted from data through a rule induction process. If the fuzzy rule base is a dense rule base then, any rule inference method such as Mamdani inference [27] or Takagi-Sugeno inference [28] can be used.

E. R Software

R is an open-source statistical computation and data visualisation software. It is a creation of the huge team of developers, researchers, statisticians and data scientists from around the world. **R** is available for all the main operating systems such as UNIX, Windows and MacOS platforms. **R** comprises data handling facilities, a superior mechanism for matrix computations, a plethora of data analysis and graphical packages, and a simple programming language [29]. The most powerful feature of **R** is subsumption i.e. its support to external packages. Currently, **R** has incorporated around 5000 packages through the CRAN family of Internet sites [30]. **R** also hosts several CI packages related to artificial neural networks, evolutionary algorithms, fuzzy systems and hybrid intelligent systems for designing intelligent systems. It is also used in many other sectors such as finance, retail, manufacturing, science, and academic research, which is making it a popular tool among statisticians and researchers [29].

III. Architecture of a Multi-Cloud System Using Docker Swarm

Fig. 4 shows the architecture of a multi-cloud (only Docker supported clouds) system using Docker Swarm. This Docker Swarm cluster has 3 manager and 2 worker nodes. In Docker Swarm, the manager is responsible for the entire cluster and

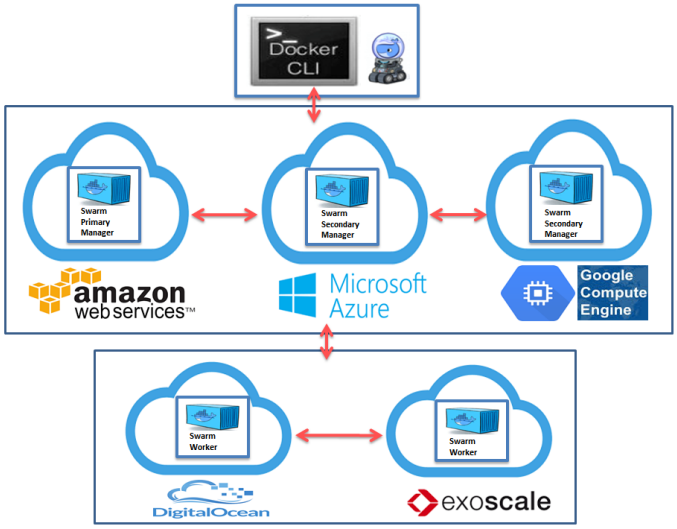

Fig. 4. Architecture of a Multi-Cloud System using Docker Swarm

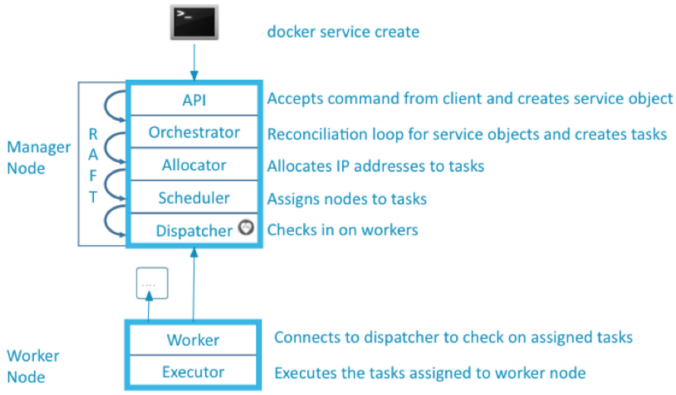

Fig. 5. Docker Swarm node breakdown and workflow [32]

manages the resources of multiple Docker hosts at scale [31]. Managers are responsible for orchestrating the cluster, serving the Service API, scheduling tasks (containers) and addressing containers that have failed health checks [32]. A primary manager (leader) is the main point of contact within the Docker Swarm cluster. In Docker Swarm, there could be one primary manager (leader) and multiple secondary managers (reachable managers) in case the primary manager fails [31]. Primary manager works as a leader of the system and all the secondary managers contact with it regarding services and information. It is also possible to talk to secondary managers (replica instances) that will act as backups. However, all requests issued on a secondary manager are automatically proxied to the primary manager. If the primary manager fails, a secondary manager takes away the lead. Therefore, it facilitates a highly available and reliable cluster [31]. Worker nodes serve only simpler functions such as executing the tasks to spawn containers and routing data traffic intended for specific containers [32]. The complete breakdown and workflow of Docker Swarm node are shown in Fig. 5.

```
Nitins-MacBook-Pro:~ nitinnaik$ docker-machine create --driver virtualbox domain-1
Nitins-MacBook-Pro:~ nitinnaik$ docker-machine create --driver virtualbox domain-2
Nitins-MacBook-Pro:~ nitinnaik$ docker-machine create --driver virtualbox domain-3
Nitins-MacBook-Pro:~ nitinnaik$ docker-machine create --driver virtualbox domain-4
Nitins-MacBook-Pro:~ nitinnaik$ docker-machine create --driver virtualbox domain-5
```

Fig. 6. Creating Docker lightweight Virtual Machines (VMs)

```
Nitins-MacBook-Pro:~ nitinnaik$ docker-machine ls
NAME       ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER
RS
default    -        virtualbox   Running   tcp://192.168.99.100:2376           v1.12.0-rc5
domain-1   -        virtualbox   Running   tcp://192.168.99.101:2376           v1.12.0-rc5
domain-2   -        virtualbox   Running   tcp://192.168.99.102:2376           v1.12.0-rc5
domain-3   -        virtualbox   Running   tcp://192.168.99.103:2376           v1.12.0-rc5
domain-4   -        virtualbox   Running   tcp://192.168.99.104:2376           v1.12.0-rc5
domain-5   -        virtualbox   Running   tcp://192.168.99.105:2376           v1.12.0-rc5
```

Fig. 7. Cluster of five running domains (lightweight VMs) with different private IP addresses and standard Docker Port 2376

IV. Experimental Simulation of a Multi-Cloud System Using Docker Swarm

This experimental simulation of the multi-cloud system is based on Docker Swarm, VirtualBox and Mac OS X. Here, this system is implemented as a cluster of five Swarm Nodes/VMs (3 managers and 2 workers) in VirtualBox on the same host computer (Mac OS X) as shown in Fig. 6. However, all these lightweight VMs and, subsequently, Swarm nodes can be created on different clouds (shown in Fig. 4) by just changing the driver name from *–driver virtualbox* to *–driver amazonec2/azure/google/digitalocean/exoscale* in Fig 6. The only requirement before doing this is to have a valid subscription account on the desired cloud. Fig. 7 shows a cluster of five running domains (lightweight VMs) with different private IP addresses and standard Docker Port 2376 using the most recent version *v1.12.0–rc5* of Docker. In Docker Swarm cluster, all the commands should be run on the manager's node.

Later, 3 manager nodes are created on domain-1, domain-2 and domain-3 and 2 worker nodes are created on domain-4 and domain-5, which are shown in Fig. 8. The domain-1 is the primary manager (leader) as it is created first but this may be changed in due course. When the node is assigned the responsibility of a manager, it joins a *RAFT Consensus group* to share information and perform leadership election. The leader is the primary manager that maintains the state, which includes lists of nodes, services and tasks across the swarm in addition to making scheduling decisions [32]. This state is circulated across the each manager node through a built-in RAFT store. Consequently, managers have no dependency on an external key-value store such as *etcd or Consul*. Non-leader managers function as hot spares and forward API requests to the current elected leader [32].

The experimental simulation and evaluation are focused on Docker Swarm only; therefore, the study is limited to the evaluation of the dependability of Docker Swarm and it does not cover the cloud and its characteristics. However, the additional advantage of using multiple cloud infrastructures is the enhanced dependability, which makes this Swarm-based system a highly dependable system by protecting the system failure due to the failure of an individual cloud infrastructure.

```
docker@domain-1:~$ docker node ls
ID                           HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS
0idmd3oydx4j5iaualcil3046 *  domain-1  Ready   Active        Leader
48oi2mtgpld7kkdzcsgtwyrz3    domain-4  Ready   Active
9wu00aewtyqu4ceclgxejd8ua    domain-5  Ready   Active
bj03eu0qjhwyd2e0xjztmudwa    domain-3  Ready   Active        Reachable
e7s1qfoiamug2gh0scxvpah8m    domain-2  Ready   Active        Reachable
```

Fig. 8. Docker Swarm cluster with 3 managers (with domain-1 as a leader) and 2 workers

```
Nitins-MacBook-Pro:~ nitinnaik$ docker-machine stop domain-1
Stopping "domain-1"...
Machine "domain-1" was stopped.
Nitins-MacBook-Pro:~ nitinnaik$ docker-machine ls
NAME       ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER
RS
default    -        virtualbox   Running   tcp://192.168.99.100:2376           v1.12.0-rc5
domain-1   -        virtualbox   Stopped                                       Unknown
domain-2   -        virtualbox   Running   tcp://192.168.99.102:2376           v1.12.0-rc5
domain-3   -        virtualbox   Running   tcp://192.168.99.103:2376           v1.12.0-rc5
domain-4   -        virtualbox   Running   tcp://192.168.99.104:2376           v1.12.0-rc5
domain-5   -        virtualbox   Running   tcp://192.168.99.105:2376           v1.12.0-rc5
```

Fig. 9. Failover procedure of a manager (leader) on the domain-1 when the domain-1 is failed

```
docker@domain-3:~$ docker node ls
ID                           HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS
0idmd3oydx4j5iaualcil3046    domain-1  Down    Active        Unreachable
48oi2mtgpld7kkdzcsgtwyrz3    domain-4  Ready   Active
9wu00aewtyqu4ceclgxejd8ua    domain-5  Ready   Active
bj03eu0qjhwyd2e0xjztmudwa *  domain-3  Ready   Active        Leader
e7s1qfoiamug2gh0scxvpah8m    domain-2  Ready   Active        Reachable
```

Fig. 10. Successful failover procedure of a manager (leader) on the domain-1 when it is automatically taken over by the domain-3 as a new manager (leader)

V. Experimental Evaluation of the Dependability of a Multi-Cloud System Using Docker Swarm

The dependability of a Docker Swarm-based system is one of the most important features which guarantees the availability, reliability and maintainability of the system. Amongst all, availability is the most important feature because if the Docker Swarm-based system is available, then it has built-in mechanisms to offer reliability and maintainability. Consequently, its availability is the most important characteristic for making it dependable. Therefore, this experimental evaluation will be focused on the availability feature only. Availability feature allows Docker Swarm to gracefully handle the failover of a primary manager (leader) instance. For evaluating the availability feature of Docker Swarm, the primary manager (leader) on the domain-1 is abruptly stopped (see Fig. 9) because this primary manager (leader) is responsible for the successful running of Swarm cluster.

However, this failure of the primary manager (leader) on the domain-1 could not stop the normal working of Docker Swarm and could not affect its availability because Swarm has immediately promoted the domain-3 as a primary manager (leader) as shown in Fig. 10. This demonstrates the first success of Swarm cluster and its availability of a primary manager (leader) to support the reliability and maintainability of desired operations.

This Docker-based dependable system has been designed with three managers and for further rigorous testing of high availability feature. Therefore, it can be tested for another failure of the new primary manager (leader) on the domain-3, and now the domain-3 is also stopped abruptly as shown in Fig. 11.

At this stage, two manager nodes domain-1 and domain-3 are stopped, however, the manager on domain-2 is still alive. Unfortunately, this time, failure of the primary manager (leader) on domain-3 could not facilitate the availability of

```
Nitins-MacBook-Pro:~ nitinnaik$ docker-machine ls
NAME       ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER
RS
default    -        virtualbox   Running   tcp://192.168.99.100:2376           v1.12.0-rc5
domain-1   -        virtualbox   Stopped                                       Unknown
domain-2   -        virtualbox   Running   tcp://192.168.99.102:2376           v1.12.0-rc5
domain-3   -        virtualbox   Stopped                                       Unknown
domain-4   -        virtualbox   Running   tcp://192.168.99.104:2376           v1.12.0-rc5
domain-5   -        virtualbox   Running   tcp://192.168.99.105:2376           v1.12.0-rc5
```

Fig. 11. Failover procedure of a manager (leader) on the domain-3 when the domain-3 is failed

```
docker@domain-2:~$ docker node ls
Error response from daemon: rpc error: code = 2 desc = raft: no elected cluster leader
docker@domain-2:~$
```

Fig. 12. Unsuccessful failover procedure of a manager (leader) on the domain-3 by the domain-2 manager when the domain-3 is failed

Swarm cluster for normal working as shown in Fig. 12. This failure has left Swarm cluster without a primary manager (leader) as shown in Fig. 12, because it could not promote its third manager on Domain-2 as a new primary manager (leader). Therefore, it is not available in this simulated situation.

This failure test is further extended for an in-depth evaluation of the nature of availability of Swarm cluster. It is now obvious that one alive manager can not be promoted as a primary manager (leader) to run Swarm cluster successfully. Thus, the manager on the domain-1 is started again (see Fig. 13) to check the availability of this Docker-based dependable system. After making the domain-1 and its manager up and running, this Swarm-based dependable system is now available again and has promoted a new primary manager (leader) on the domain-2 as shown in Fig. 14.

This illustrates that Swarm can only offer high availability with at least three managers and can only protect one failure. This is because of the employment of *RAFT Consensus* algorithm for the management and controlling of managers. The latest version of Docker Swarm 1.12 handles controller node failures by adding additional (3-7) replica manager nodes to the cluster as per given in Table I [33]. Thus, Swarm cluster can protect maximum three failures when it must have seven managers.

Alongside the assurance of availability of a primary manager (leader) for the management and controlling of a cluster,

```
Nitins-MacBook-Pro:~ nitinnaik$ docker-machine ls
NAME       ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER
RS
default    -        virtualbox   Running   tcp://192.168.99.100:2376           v1.12.0-rc5
domain-1   -        virtualbox   Running   tcp://192.168.99.101:2376           v1.12.0-rc5
domain-2   -        virtualbox   Running   tcp://192.168.99.102:2376           v1.12.0-rc5
domain-3   -        virtualbox   Stopped                                       Unknown
domain-4   -        virtualbox   Running   tcp://192.168.99.104:2376           v1.12.0-rc5
domain-5   -        virtualbox   Running   tcp://192.168.99.105:2376           v1.12.0-rc5
```

Fig. 13. Failover procedure of a manager (leader) on the domain-3 when the domain-1 is again up and running

```
docker@domain-2:~$ docker node ls
ID                           HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS
0idmd3oydx4j5iaualcil3046    domain-1  Ready   Active        Reachable
48oi2mtgpld7kkdzcsgtwyrz3    domain-4  Ready   Active
9wu00aewtyqu4ceclgxejd8ua    domain-5  Ready   Active
bj03eu0qjhwyd2e0xjztmudwa    domain-3  Down    Active        Unreachable
e7s1qfoiamug2gh0scxvpah8m *  domain-2  Ready   Active        Leader
```

Fig. 14. Successful failover procedure of a manager (leader) on the domain-3 by the domain-2 manager when the domain-3 is failed but the domain-1 is again up and running

TABLE I. FAILURE TOLERATION CAPABILITY OF DOCKER SWARM-BASED DEPENDABLE SYSTEMS

Controller and Replicas of Manager/Leader	Number of Failures Tolerated
1	0
3	1
5	2
7	3

the availability of a worker node is equally important. The failure of a worker can be easily and frequently checked for the progress and successful completion of the assigned task on that node. For this, workers send managers the status of their assigned tasks in their assignment set and a heartbeat; therefore, the managers can confirm that the worker is still alive. Heartbeats of workers can be checked by using the flag *–dispatcher-heartbeat duration*. This flags sets the frequency with which worker nodes are told to use as a period to report their health. The default dispatcher heartbeat period is 5 seconds. In summary, workers failure can be easily managed by the manager.

VI. PROPOSED CI-BASED SOLUTION FOR ENHANCING THE DEPENDABILITY OF A MULTI-CLOUD SYSTEM USING DOCKER SWARM

As previously mentioned that Docker Swarm uses *RAFT Consensus* algorithm for the management and controlling of managers. Based on the *RAFT Consensus* strategy, decisions can only be made if the majority of managers reach consensus. In the Swarm cluster of 3 manger nodes, *RAFT* majority is 2/3; therefore, if two managers are down, a majority can never be reached, and a primary manager (leader) cannot be selected. This is one of the design issues of Docker Swarm and its employed algorithm. In the previous section, all the Swarm nodes are created on lightweight VMs and VMs compete for resources or their behaviour may change drastically due to an unexpected failure of the host machine. Therefore, the manager's availability issue can also be resolved externally by observing the behaviour of lightweight VMs used by all Swarm manager nodes. This can be done by designing a CI-based intelligent system. If the VM of a manager node starts behaving abnormally, then, this would be an early indication of the failure of that manager node. In this situation, a new manager node can be created, or an existing worker node can be promoted as a manager. It should be noted that this issue arises when the Swarm cluster is designed with 3 manager nodes. If it has 5 or 7 manger nodes, then this would not be an issue and a new primary manager (leader) can be selected from the group of existing secondary managers. However, the operational overheads for running a cluster of 5 or 7 managers is much higher than the cluster of 3 managers.

There are several CI techniques available to design an intelligent system to observe the behaviour of VMs of manager nodes. This paper is proposing a CI-based approach to resolving this problem, where a fuzzy inference system is developed to observe the behaviour of VMs of manager nodes

Process Name	% CPU	CPU Time	Threads	Idle Wake Ups	PID	User
VBoxHeadless	4.7	1:05.61	25	341	668	nitinnaik
VBoxHeadless	10.6	53.86	24	387	771	nitinnaik
VBoxHeadless	4.2	50.00	25	354	818	nitinnaik
VBoxHeadless	3.2	30.94	25	288	867	nitinnaik
VBoxHeadless	1.6	29.34	24	286	906	nitinnaik

Fig. 15. CUP usage of Headless (lightweight) VMs used for Docker Swarm nodes

Process Name	Memory	Compressed M...	Threads	Ports	PID	User
VBoxHeadless	561.7 MB	0 bytes	24	85	906	nitinnaik
VBoxHeadless	561.6 MB	0 bytes	25	86	867	nitinnaik
VBoxHeadless	562.1 MB	0 bytes	25	86	818	nitinnaik
VBoxHeadless	561.9 MB	0 bytes	24	85	771	nitinnaik
VBoxHeadless	561.8 MB	0 bytes	25	86	668	nitinnaik

Fig. 16. Memory usage of Headless (lightweight) VMs of Docker Swarm nodes

and predict the possible failure. This is a preliminary model, therefore, only two important parameters CPU usage and Memory usage are chosen as important criteria to check the health of all VMs. Docker implementation based on VirtualBox runs Docker Host as a Headless VM (i.e., lightweight VM) as shown in Figs. 15 and 16. Fig. 15 shows the CPU usage of all running Headless (lightweight) VMs of Docker Swarm nodes at a particular moment. Similarly, Fig. 16 shows the Memory usage of all running Headless (lightweight) VMs of Docker Swarm nodes at a particular moment. These two parameters and their ranges are carefully chosen based on the prolonged observation and analysis of all Docker VMs in VirtualBox. Later, they are used to derive two fuzzy input variables called *cpuu* (CPU Usage) and *memu* (MEMory Usage). The range of *cpuu* was determined based on the minimum value 0 and maximum value 100; and, the range of *memu* was determined based on the minimum value 0 and maximum value 2048. It should be noted that the CPU usage and memory requirement for Headless (lightweight) VMs of Docker Swarm nodes are machine specific and may vary depending on the hardware configuration of the host. However, this approach can be easily adapted for several other parameters and their system specific values.

The complete fuzzy inference system is designed using software *R*. The main reason for choosing *R* is that it is supported by Docker, and its image can be run in containers. Based on these two fuzzy input variables, the fuzzy output variable called *vmavail* (VM AVAILability) is determined which inform the availability level of VMs of manager nodes in percentage (0-100). All the input and output fuzzy variables are divided into three fuzzy range *under, low* and *normal* as shown in Fig. 17. Afterwards, a sample fuzzy rule base (see Fig. 18) is designed for the fuzzy inference system (see Fig. 19) to inform only the unavailability of the VM of a Swarm manager. This intelligent system can observe the health of all VMs of manager nodes and inform their availability level. Consequently, the failure of a manager due to the failure of its VM can be managed if the VM starts behaving abnormally. Thus, this indication can automatically trigger the command or process to create a new manager node or promote an existing worker node as a manager to maintain the minimum number of required manager nodes. The command or process should be similar that is explained in the Sections IV and V.

```
> variables <-
    set(
        cpuu =
            fuzzy_variable(under =
                               fuzzy_triangular(corners = c(0, 2.5, 5)),
                           low =
                               fuzzy_triangular(corners = c(4, 7, 10)),
                           normal =
                               fuzzy_triangular(corners = c(8, 54, 100))),
        memu =
            fuzzy_variable(under =
                               fuzzy_triangular(corners = c(0, 281, 562)),
                           low =
                               fuzzy_triangular(corners = c(562, 768, 1024)),
                           normal =
                               fuzzy_triangular(corners = c(768, 1024, 2048))),
        vmavail =
            fuzzy_variable(under =
                               fuzzy_triangular(corners = c(0, 20, 40)),
                           low =
                               fuzzy_triangular(corners = c(30, 50, 70),
                           normal =
                               fuzzy_triangular(corners = c(60, 80, 100)))
    )
```

Fig. 17. Defining linguistic fuzzy variables in R

```
> rules <-
    set(
        fuzzy_rule(cpuu %is% under && memu %is% under,
                   vmavail %is% under),
        fuzzy_rule(cpuu %is% under && memu %is% low,
                   vmavail %is% under),
        fuzzy_rule(cpuu %is% under && memu %is% normal,
                   vmavail %is% under),
        fuzzy_rule(cpuu %is% low && memu %is% under,
                   vmavail %is% under),
        fuzzy_rule(cpuu %is% normal && memu %is% under,
                   vmavail %is% under)
    )
```

Fig. 18. Designing fuzzy rule base in R

VII. Conclusion

This paper presented the simulation of a multi-cloud system using Docker Swarm for evaluating its dependability feature. Subsequently, it proposed an intuitive approach based on Computational Intelligence (CI) for enhancing its dependability. The proposed CI-based approach predicts the possible failure of the host of a manager node by observing its abnormal behaviour. Thus, this indication can automatically trigger the process of creating a new manager node or promoting an existing worker node as a manager. This simulation and evaluation of the dependable system are based on Docker Swarm, VirtualBox, and Mac OS X. However, the same multi-cloud dependable system can be easily created on any of the Docker supported cloud. Docker Swarm-based dependable system development is a new approach for the cloud industry and supported by only a few selected cloud service providers. Also, the proposed CI-based approach is a promising to assess and control the external failure and maintain the availability of Docker Swarm cluster. However, it is a preliminary model and needs further expansion and testing to determine the effective-

```
> system <- fuzzy_system(variables, rules)
> print(system)
```

Fig. 19. Resultant fuzzy inference system in R

ness of this approach. In the future, it may be worthwhile to develop the real multi-cloud Docker Swarm system and apply this CI-based approach on that system.

REFERENCES

[1] N. Naik, "Building a virtual system of systems using Docker Swarm in multiple clouds," in *IEEE International Symposium on Systems Engineering (ISSE)*. IEEE, 2016, pp. 191–193.

[2] ——, "Connecting Google cloud system with organizational systems for effortless data analysis by anyone, anytime, anywhere," in *IEEE International Symposium on Systems Engineering (ISSE)*. IEEE, 2016, pp. 202–207.

[3] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE transactions on dependable and secure computing*, vol. 1, no. 1, pp. 11–33, 2004.

[4] S. Kounev, P. Reinecke, F. Brosig, J. T. Bradley, K. Joshi, V. Babka, A. Stefanek, and S. Gilmore, "Providing dependability and resilience in the cloud: Challenges and opportunities," in *Resilience Assessment and Evaluation of Computing Systems*. Springer, 2012, pp. 65–81.

[5] N. Naik and P. Jenkins, "A secure mobile cloud identity: Criteria for effective identity and access management standards," in *2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*. IEEE, 2016, pp. 89–90.

[6] ——, "An analysis of open standard identity protocols in cloud computing security paradigm," in *14th IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC 2016)*. IEEE, 2016.

[7] N. Naik, "Migrating from virtualization to dockerization in the cloud: Simulation and evaluation of distributed systems," in *IEEE 10th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Environments, MESOCA 2016*. IEEE, 2016, pp. 1–8.

[8] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.

[9] G. M. Tihfon, J. Kim, and K. J. Kim, "A new virtualized environment for application deployment based on docker and aws," in *Information Science and Applications (ICISA)*. Springer, 2016, pp. 1339–1349.

[10] C. Anderson, "Docker [Software Engineering]," *IEEE Software*, no. 3, pp. 102–c3, 2015.

[11] P. Marinescu, P. Hosek, and C. Cadar, "Covrig: A framework for the analysis of code, test, and coverage evolution in real software," in *Proceedings of the 2014 International Symposium on Software Testing and Analysis*. ACM, 2014, pp. 93–104.

[12] W. Gerlach, W. Tang, K. Keegan, T. Harrison, A. Wilke, J. Bischof, M. D'Souza, S. Devoid, D. Murphy-Olson, N. Desai *et al.*, "Skyport: container-based execution environment management for multi-cloud scientific workflows," in *Proceedings of the 5th International Workshop on Data-Intensive Computing in the Clouds*, 2014, pp. 25–32.

[13] Docker.com. (2016) Docker use cases. [Online]. Available: https://www.docker.com/use-cases

[14] N. Ferry, A. Rossini, F. Chauvel, B. Morin, and A. Solberg, "Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems," in *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*. IEEE, 2013, pp. 887–894.

[15] Docker.com. (2016) Supported drivers. [Online]. Available: https://docs.docker.com/machine/drivers/

[16] J. Turnbull, *The Docker Book: Containerization is the new Virtualization*. James Turnbull, 2014.

[17] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and linux containers," in *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On*. IEEE, 2015, pp. 171–172.

[18] Docker.com. (2016) Docker swarm. [Online]. Available: https://www.docker.com/products/docker-swarm

[19] M. Rouse. (2016) Docker swarm. [Online]. Available: http://searchitoperations.techtarget.com/definition/Docker-Swarm

[20] L. A. Zadeh, "Fuzzy sets," *Information and control*, vol. 8, no. 3, pp. 338–353, 1965.

[21] N. Naik, R. Diao, C. Quek, and Q. Shen, "Towards dynamic fuzzy rule interpolation," in *IEEE International Conference on Fuzzy Systems*, 2013, pp. 1–7.

[22] N. Naik, R. Diao, and Q. Shen, "Genetic algorithm-aided dynamic fuzzy rule interpolation," in *IEEE International Conference on Fuzzy Systems*, 2014, pp. 2198–2205.

[23] N. Naik, "Fuzzy inference based intrusion detection system: FI-Snort," in *IEEE International Conference on Dependable, Autonomic and Secure Computing*, 2015, pp. 2062–2067.

[24] N. Naik and P. Jenkins, "Fuzzy reasoning based windows firewall for preventing denial of service attack," in *IEEE International Conference on Fuzzy Systems*, 2016.

[25] N. Naik, R. Diao, and Q. Shen, "Application of dynamic fuzzy rule interpolation for intrusion detection: D-FRI-Snort," in *IEEE International Conference on Fuzzy Systems*, 2016.

[26] N. Naik and P. Jenkins, "Enhancing windows firewall security using fuzzy reasoning," in *IEEE International Conference on Dependable, Autonomic and Secure Computing*, 2016.

[27] E. H. Mamdani and S. Assilina, "An experiment in linguistic synthesis with a fuzzy logic controller," *International Journal of Man-Machine Studies*, vol. 7, no. 1, pp. 1–13, 1975.

[28] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its applications to modeling and control," *Systems, Man and Cybernetics, IEEE Transactions on*, no. 1, pp. 116–132, 1985.

[29] B. Oancea and R. M. Dragoescu, "Integrating R and hadoop for Big Data Analysis," *arXiv preprint arXiv:1407.4908*, 2014.

[30] Cran.r-project.org. (2014, April 9) ff: memory-efficient storage of large data on disk and fast access functions. [Online]. Available: https://cran.r-project.org/web/packages/ff/index.html

[31] Docker.com. (2016) High availability in docker swarm. [Online]. Available: https://docs.docker.com/swarm/multi-manager-setup/

[32] ——. (2016, July 28) Docker built-in orchestration ready for production: Docker 1.12 goes ga. [Online]. Available: https://blog.docker.com/2016/07/docker-built-in-orchestration-ready-for-production-docker-1-12-goes-ga/

[33] ——. (2016) Set up high availability. [Online]. Available: https://docs.docker.com/ucp/high-availability/set-up-high-availability/