electronics

MDPI

Article

A New Hybrid Fault Tolerance Approach for Internet of Things

Mehdi Nazari Cheraghlou [1], Ahmad Khadem-Zadeh [2] and Majid Haghparast [3,*]

[1] Department of Computer Engineering, South Tehran Branch, Islamic Azad University, Tehran, Iran; St_m_NazariCheraghlou@azad.ac.ir
[2] Iran Telecommunication Research Center (ITRC), Tehran, Iran; zadeh@itrc.ac.ir
[3] Department of Computer Engineering, Yadegar-e-Imam Khomeini (RAH) Shahre Rey Branch, Islamic Azad University, Tehran, Iran
* Correspondence: haghparast@srbiau.ac.ir

Received: 30 March 2019; Accepted: 5 May 2019; Published: 9 May 2019

Abstract: In the Distributed Management Task Force, DMTF, the management software in the Internet of things (IoT) should have five abilities including Fault Tolerance, Configuration, Accounting, Performance, and Security. Given the importance of IoT management and Fault Tolerance Capacity, this paper has introduced a new architecture of Fault Tolerance. The proposed hybrid architecture has used all of the reactive and proactive policies simultaneously in its structure. Another objective of the current paper was to develop a measurement indicator to measure the fault tolerance capacity in different architectures. The CloudSim simulator has been used to evaluate and compare the proposed architecture. In addition to CloudSim, another simulator was implemented that was based on the Pegasus-Workflow Management System (WMS) in order to validate the architecture that is proposed in this article. Finally, fuzzy inference systems were designed in a third step to model and evaluate the fault tolerance in various architectures. Based on the results, the positive effect of using various combined Reactive and Proactive policies in increasing the fault tolerance in the proposed architecture has been prominently evident and confirmed.

Keywords: Internet of things; cloud computing; fault tolerance; fault detection; fault recovery

1. Introduction

The evolution and transformation of the Internet have transitioned from the Internet of content to the Internet of services and the Internet of people, and today there is the Internet of things. The Internet of things, hereinafter referred to as IoT, consists of a series of smart sensors, which, directly and without human intervention, work together in new kinds of applications. It is obvious that the management and traditional architecture of the Internet must be changed. For example, the address space for support must be changed from IPv4 to IPv6. The first and most important thing for upgrading the management of the IoT is a requirement for a layered and flexible architecture. Many models have been proposed for the IoT architecture. A basic model is a three-layered architecture. This architecture has been formed from sensors, network, and process layers (See e.g., [1]).

The explosive growth of smart objects and their dependencies on wireless communication technologies make the Internet more vulnerable to faults. These faults may be established due to different reasons, including cyber-attacks, which are referred to in [2]. These faults may provide daunting challenges to experts, as it becomes more important to manage the participating components in IoT. The Distributed Management Task Force, briefly referred to as DMTF, has announced that the cloud management software should have the ability of FCAPS. The first letter of the word that is character F is an acronym for fault tolerance. In other words, the first feature of the management

software should entail fault tolerance. Other management capabilities can be considered if there is a fault tolerance feature. However, in the absence of fault tolerance, other features are not important and they accompany no management ability. Fault tolerance refers to providing an uninterrupted service system, even in the presence of an existing fault in the system. It is quite clear that, if we want to review and implement the management in the IoT layered architecture, the primary focus should be on its first feature, i.e., fault tolerance.

Different methods have been introduced to increase the fault tolerance in the IoT. For example, in [3], the virtual form of cluster head nodes are utilized when considering that CH nodes have the role of forwarding collected data in the IoT application. Therefore, the tolerance of the CH nodes failure will increase in the network. Furthermore, in [4], the virtualization technique is used in wireless sensor networks due to the growth of the IoT service. When a fault is developed in wireless sensor network communications, it has significant impact on many virtual networks running IoT services. The framework that is proposed in [4] provides the optimization of the fault tolerance in virtualized wireless sensor networks, with a focus on heterogeneous networks for service-based IoT applications. Regardless of other methods, the main techniques for increasing fault tolerance can be categorized in three broad categories. The techniques for increasing fault tolerance have been divided into three main groups. The first group includes redundancy techniques. These techniques can be implemented as hardware redundancy, software redundancy, or time redundancy. The second category includes load balancing techniques. These techniques can also be implemented as hardware, software, or in the network. Finally, the third group of techniques to increase the fault tolerance (FT) capability is related to the use and benefit of different policies. These policies are dependent on the environment in which they are implemented. For example, two types of policies are used in cloud computing environments: the proactive policy and the reactive policy. The fault will not be allowed in proactive policies and design should be done in such a way that the fault is forecasted before creating the fault, which is possible to be prevented. Accordingly, these types of policies cover two phases, including fault forecasting and fault prevention. This procedure is also followed in reactive policies in which the fault tolerance operation happens after the occurrence of the fault in the form of fault detection, fault isolation, fault diagnosis, and fault recovery. Of course, it should be noted that the use of each of these methods in increasing the FT capability imposes overhead costs to the system, but the highest performance for the system is achieved when the reactive and proactive policies are used and simultaneously implemented.

The aim of the present study was evaluating the management of fault tolerance in the IoT communication platform, i.e., its second layer architecture. In this regard, the analysis that was carried out in [5] was used. Subsequently, a new architecture was proposed, whereby the maximum coverage of various phases enjoyed the FT capability. The simultaneous benefit from the reactive and proactive policies achieved the highest possible performance in the output.

The Internet has created extensive variations in industrial scopes and business models. Industrial internet is the result of combining internet and big data and artificial intelligence and economy in the world. A developed digital channel is responsible for information delivery based on the latest technologies regarding the industrial Internet, so that intelligent decisions can be done in the real time format to enhance the efficiency. To this end, there is a need for a platform in the real world to realize the rapid integrity and reply to the market as fast as possible. The rapid integrity can be realized and it can reply to the market as long as it uses resources given the fact that the proposed architecture in this article was mainly applicable in real time systems. Hence, it can have a wide range of applications in industrial internet platform and IIoT architecture.

The contributions of this paper are as follows:

- We offer a hybrid modern architecture that simultaneously uses proactive and reactive policies.
- Our proposed architecture uses all three types of reactive policies at the same time.
- Both proactive policies are implemented at the same time in our proposed architecture structure.
- Maximum use of different fault detection/recovery methods is also considered in our proposed architecture.

- In addition to simulations that were carried out, the design and implementation of scientific workflows in the past architectures and our proposed architecture are one of the most important achievements and innovations of this study.

The remainder of the article has been divided into eight sections. The related previous architectures are presented in Section 2 of the article. In Section 3, the proposed architecture is introduced and described. Subsequently, in Section 4, the CloudSim simulator is used to simulate the proposed architecture and it has compared its performance with previous architectures; in Section 5, scientific workflows are introduced and the proposed architecture is simulated. In Section 6, the evaluation systems in fuzzy logic are designed and implemented, and the analysis of the results is discussed; and ultimately, in Section 7, optimal decision-making is discussed. In Section 8, the conclusions and related ideas for future work are expressed.

2. Related Works

As expressed in [5], the fault-tolerant architectures of cloud computing are divided into two general categories according to the policies that will benefit them. The first group is formed by proactive architectures and the second group includes reactive architectures. As expressed in [6–8], the preemptive migration and self-healing policies are used in proactive architectures. The Check Point/Restart, Replication, and Job Migration policies have also been used in reactive architectures. The Map-Reduce and FT- cloud architectures are examples of proactive architectures. The Map-Reduce that was introduced in [9–13] has used both proactive policies in its structure. The FT- cloud described in [14] has only used the self-healing policy.

The FTWS, LLFT, FTM-2, FTM, MPI, Gossip, BFT-Cloud, Haproxy PLR, AFTRC, Vega Warden, MagiCube, and Candy, as presented in [15–28], are among the reactive architectures. Each of these architectures simultaneously uses one or two policies. Of course, the architectures of the PLR, AFTRC, FTM, as presented in [15–17], have used all three reactive policies. The difference between the FTM with two architectures is that AFTRC and PLR are devoted to real-time systems, whereas FTM is not devoted to such systems. The difference between PLR with AFTRC is that the PLR has a lower Wall Clock Time than the AFTRC. An explanation of all architectures mentioned in [5] has been described in detail. As expressed in [5,16], the AFTRC architecture has benefited from five modules: Recovery Cache (RC), Decision Maker (DM), Reliability Assessor (RA), Time Checker (TC), and Acceptance Test (AT) in its internal structure. The AT module is responsible for checking the output value of the virtual machine. TC performs time checking, i.e., the time that is required for generating the output of the virtual machine. The RA module that evaluates the reliability of the output, in fact, identifies the percentage of the output credit that is based on two previous parameters, i.e., the AT and TC. DM module is responsible for determining and selecting the final output and, finally, the RC module is the storage of the checkpoints in the case of operation replication. Figure 1 shows the structure of this architecture.

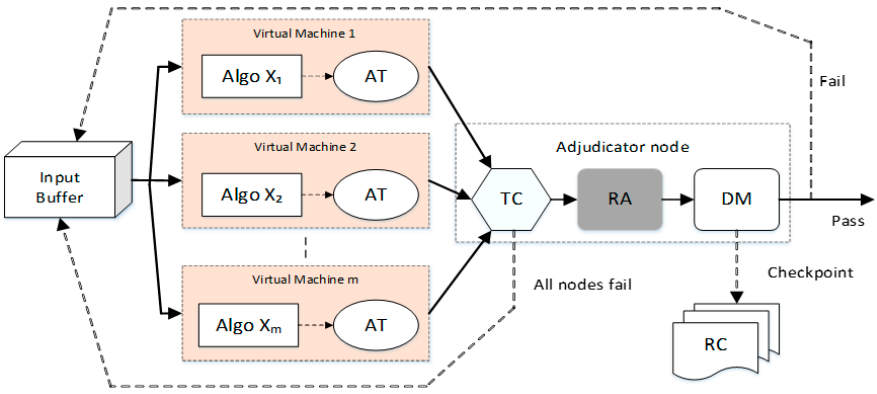

Figure 1. AFTRC architectural structure presented in [16].

The two main phases of the FT capability are fault detection and fault recovery. Fault detection and fault recovery can be conducted in different ways. The Gossip architecture that is presented in [20], the FTM presented in [15], and the PLR presented in [17] have used the self-detection method. This is the weakest method of fault detection, because a node itself should detect the fault. Since the PLR architecture has only used the method in the fault detection phase, it has a very weak capability of fault detection, and it is a great disadvantage of this architecture. Nevertheless, the architectures of the FTM and Gossip have also used the methods of other detection and group detection, in addition to this method.

The majority of the architectures that are proposed in the FT field of cloud computing have implemented the other detection method in their own fault detection phase. It should be noted that only the fault group detection method has been in architectures, such as the LLFT and BFT-Cloud, as presented in [19,22]. The dominant method in fault recovery is also the system recovery method whereby the recovery is performed at the overall level of the system. Again, it can be seen in this phase that the LLFT and Vega-Worden architectures that are presented in [22,26] perform the fault recovery at the node level, which is a weaker method than the previous method. The weakest recovery method is the fault mask mode in which it sought to exploit the techniques for removing and covering the fault effect. The FTM and AFTRC architectures that are introduced in [15,18] have used this technique with other methods.

It is extremely important that all of the studied architectures so far are reactive or proactive. In addition, the architecture described in [27] is a hybrid one, which is termed as VFT. VFT architecture utilizes Self-healing, Preemptive Migration, and Replication policies. In [28], it is stated that the architecture introduced in [27], simultaneously used both proactive and reactive policies. Figure 2 shows the structure of this architecture.

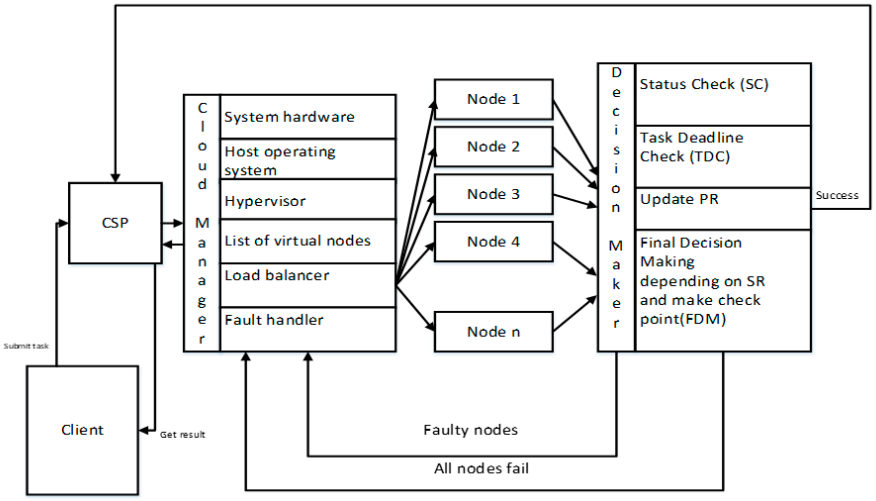

Figure 2. VFT architectural structure introduced in [27], which is the only hybrid architecture.

A computational algorithm, called the success rate for each node, has been individually applied in VFT architecture. The algorithm decides based on two factors. The first factor is PR, which represents the performance rate. The second factor is the max success rate, which represents the maximum level that is considered for the success rate. If the SR of a node is less than or equal to zero, in this case, the load balancer does not assign tasks to the virtual machine in the next cycles.

It would be possible to decide the failure of a node according to the parameters of SC and TDC. Parameter SC stands for Status Checker and parameter TDC stands for Task Deadline Checking. If all of the nodes also fail, the FDM, Which stands for Final Decision-Making, sends feedback to the fault handler in order to make it aware of this issue. The fault handler detects and recovers the fault, based on the techniques that are defined and implemented for fault detection and fault recovery. It is worth noting that the architecture in the fault detection phase acts based on the other detection method

according to the scenario of the VFT architecture. The fault recovery method in the VFT architecture has also been implemented at the system level.

The approaches that are presented in [29–33] can be mentioned as some instances of architectural models of fault tolerance, which have been presented according to other architectural models of the IoT. Fault tolerance implemented on the internet of military objects has been investigated in [29]. In the presented model, which is shortly called IOMT, is an approach called MM and presented by Malek and Maeny, which has been used in order to facilitate fault detection. This method is the majority of the duplicate entries that have been sent to a dual-processor. In this architecture, the fault mask technique has also been used on the fault recovery phase. In addition, fault tolerance on routing IoT is the proposed method in [30]. Layer fault management, which as a plan for end-to-end transfer, has been introduced in [31]. In [32], another method has been presented using the concept of virtual services. In the present article, a genetic algorithm, which is known as NSG-ii, has been used. Fault tolerance that is based on the architecture of the middle base ware has been shown in [33]. In this method, redundancy on the services' level has been implemented. To conclude, the shortcomings of the existing solutions are as follows:

- The highest efficiency of the fault tolerance architecture is obtained through hybrid architectures. Among all of the investigated architectures, VFT architecture is the hybrid one, so the lack of hybrid architecture is extensively felt in this scope.
- VFT architecture enjoys proactive policies in the integrative way. However, it has only used the replication method among reactive policies and it has not utilized all of the methods in an integrative way. Thus, it cannot have the maximum reliability.
- Fault detection method in architecture has been done in another detection way, which is an average method among the detecting methods.
- The fault recovery method of the VFT architecture has been the only previous hybrid architecture in the system level, in which the refinement of faults has not happened in VMs' level. Moreover, the fault mask method has not enjoyed this method in fault recovery phase of architecture.

3. Proposed Approach

The architecture that is proposed in this paper has been called PRP-FRC. This architecture is considered as a hybrid architecture in terms of implementing the reactive and proactive policies. It has been sought in the PRP-FRC that proactive policies, including preemptive migration and self-healing and reactive policies, including Checkpoint/Restart and Job Migration and Replication can be simultaneously implemented. Figure 3 shows the proposed PRP-FRC architectural structure.

In the proposed architecture of PRP-FRC, tasks are initially divided between physical nodes, which are the hosts, by the load balancer. Subsequently, they are divided between virtual nodes by the manager available in each node with the help of the mapping table. Output accuracy and checking the reliability of the output for a virtual machine is achieved by the AT module. If the output validity of a virtual node is not confirmed by the AT, then the task will be assigned to another virtual machine by the manager via feedback that is available from the AT to the manager. The TC module makes decisions on the time validity of a virtual machine's generated output. In fact, the task of the TC is to check that the virtual machine has produced an output in the time that is taken to respond or has spent a more defined period to generate the time departure.

The importance of the mentioned module is very critical, because the proposed architecture of PRP-FRC, such as the AFRC architecture, is intended for real-time applications. The RA module decides on the reliability rate of a node that is based on the output values of the two previous parameters (AT and TC). In other words, if the percentage of node reliability is less than the limit that is defined in the RA modules, then a new task will not be assigned to the virtual node in the next cycles. In the process of verifying the time validity of the virtual nodes, if none of the nodes have the desired output, the TC issues an "All Nodes Failed" message. The request for job migration is activated in this case, and the guest virtual node on another host is selected by the MPI architecture and the job migration is done.

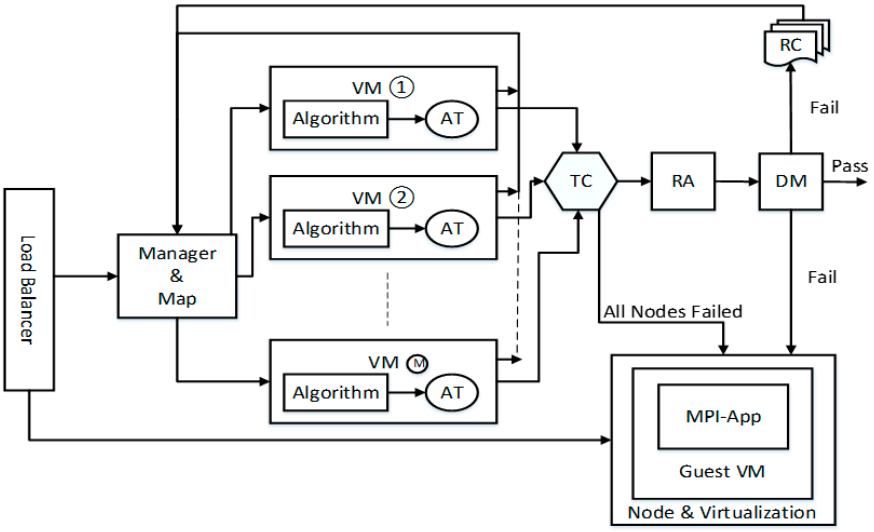

Figure 3. Our proposed Architecture proposed in this paper (PRP-FRC) hybrid architectural structure.

In the case where all of the nodes are not failed and only some of them have gained the reliability that is necessary for the production of the Trust Output, and then two modes will be implemented based on the decision of the DM module, which stands for Decision Making. The first case is that the task's restart operation is performed from the last checkpoint that is based on the storage of checkpoints in the architectural structure. The second case is that the job migration to a virtual machine happens on another host, regardless of the checkpoint. The choice of a method in this step depends on the policies that are defined in the DM module. Algorithm 1 shows the algorithm of the proposed architecture of the PRP-FRC.

Algorithm 1: The algorithm of the PRP-FRC

```
Begin
SRL = System-Reliability-Level
RC = Repository-of-Checkpoints
AT = Acceptance-Test (its check the result-validity)
TC = Time-Check (its check cloudlet-execution-time)
T = Max Acceptable Time
k = Number of Cloudlet Replica
#Step 1
Cloudlet-List = Create_Job (with Random-details)
#Step 2
Load-Balancer (input: Cloudlet-List):
chosen-host = Choose execution-environment with Host-Allocation-Policy
# it can be a local-host
# or chosen from an external Host & communicate with MPI-Application
for each Cloudlet in Cloudlet-List:
{
    Send Cloudlet to Manager of chosen-host
    #Step 3
    Manager (input: cloudlet):
    Allocate K new 'VM'
    Create K 'Replica' of each 'Cloudlet'
    Add Replica to Replica-List
    for each Replica in Replica-List:
    {   Send Replica to VM[i]
```

```
25:          i = i + 1
26:          #Step 4
27:          result = Run (Replica in VM & create CheckPoint in every t second)
28:          if (result = 'OK'):
29:          {
30:             AT=True
31:             if (execute_Time < T):
32:             {
33:                TC=True
34:             }
35:             else:
36:             {
37:                TC=False
38:             }
39:          }
40:          else:
41:          {
42:             AT=False
43:             TC=False
44:          }
45:          if (AT=True & TC=True):
46:          {
47:             {VM(i)-Status = Success
48:          }
49:          else:
50:          {
51:             VM(i)-Status = Fail
52:          }
53:       }
54:       #Step 5
55:       if (All-VM-Status is 'Fail'):
56:       {
57:          Cloudlet Migrate to other Host with 'MPI-Application'
58:       }
59:       RA = Calculate Reliability of VM's that run this Cloudlet
60:       DM = Find VM with Best Reliability and save Best-RA
61:       #Step 6
62:       if (Best-RA < SRL):
63:       {
64:          #do Preemptive-Migration:
65:          Kill VM with Minimum Reliability
66:          Allocate New VM
67:          Add Cloudlet to67 Cloudlet-List again
68:          Restart Cloudlet68 on other VM - from last CheckPoint stored in RC
69:          #or
70:          #do Job-Migration
71:          Send Cloudlet to MPI-Application
72:          Start Cloudlet from 'Begin'
73:       }
74:       else:
75:       {
76:          Final-Result = Result of VM with Best-RA
77:       }
78:    }
79:    End
```

As a result, the following issues can be mentioned if we want to state the proposed strength to cover the weaknesses of the previous architecture.

- The proposed architecture is a hybrid architecture, so it is expected to have the highest efficiency.
- The proposed architecture enjoys all reactive and proactive policies in an integrated way, so it leads the system to achieve the highest level of fault tolerance.
- The proposed architecture has simultaneously utilized two methods of self-detection and other detection in the detection phase of the faults.
- The proposed architecture has used every refinery fault, which is unlike the previous architectures and hybrid VFT ones. It also masks the faults and refines the faults that are in Vms levels. Finally, it prepares the refinement of the faults in the system level.

4. Simulation of the Proposed Architecture in CloudSim

CloudSim is used to simulate and compare the proposed architecture with previous architectures. This simulator can generate various reports on energy consumption, cost, and execution time of each Cloudlet. The ease of implementation, simulation of different types of network topologies and architectures, the defining of multiple DataCenter, ease of implementation of different policies for allocating Vm and Host, and support for Space Shared and Time-Shared scheduling are the benefits of this simulator.

The implemented scenario in this article has been designed in the cloudSim simulator in a two-host DataCenter with a number of different processors. Each host has four VMs with various specifications. The ability of VMs is differently defined to deal with different versions of each Cloudlet, whose different behaviors and functions have been examined. Tables 1 and 2 show the configuration specifications of DataCenter and Vm, respectively.

Table 1. Datacenter Configuration.

Date Center Configuration		
-	**Host#0**	**Host#1**
PE's Number	5	2
RAM	4096	4096
Storage (MB)	1000000	1000000
BandWidth	10000	10000
MIPS	3000	3000

Table 2. Virtual Machine Configuration.

Virtual Machine Configuration				
-	**VM#0**	**VM#1**	**VM#2**	**VM#3**
Image Size (MB)	10000	10000	10000	10000
RAM	512	512	512	512
MIPS	1600	700	2900	1200
BandWidth	1000	1000	1000	1000
PE's Number	1	1	1	1
VMCostPerSec($)	3	1	4	2

Moreover, a number of Cloudlet have been designed and implemented with a different processing length. The purpose of this type of design was to have a different length of Cloudlets to create different modes of faults on the system.

After designing DataCenter, Host, Vm, and Cloudlet, and before simulation, it has become necessary to extend the CloudSim based on the proposed architecture. According to which, the manner of distribution of computational resources, such as Ram and especially PEs among requests, should be determined. This scheduling can be in the form of Space Shared, Time-Shared, or Customized policy definition. In all simulations, we have used all resources in Space Shared manner.

In the Reliability calculation algorithms, the calculations are done in such a way that the effect of a fail or a Success causes changes with a slower slope in the reliability of each step. Additionally, at the beginning of a Vm, the fail has a different effect with the fails occurred after several requests. Obviously, the reliability in each stage of the simulation becomes equal to the average reliability of all of the VMs at that stage. The simulation results in Tables 3–5 and Figure 4 is shown. These results have demonstrated the positive effect of using all policies as hybrids in the proposed architecture.

Table 3. Results of Simulating AFTRC in CloudSim.

Event	Host	VM#0			VM#1			VM#2			VM#3		
	-	AT	TC	RA	AT	TC	RA	AT	TC	RA	AT	TC	RA
-	0	1	1	1	1	1	1	1	1	1	1	1	1
-	0	1	1	1	1	0	0.52	1	1	1	1	1	1
VM#3 Failed	0	1	1	1	1	1	0.61	1	1	1	0	-	0.53
-	0	1	1	1	1	0	0.34	1	1	1	1	1	0.63
-	0	1	0	0.56	1	0	0.21	1	1	1	1	0	0.37
All-Failed	0	1	0	0.35	1	0	0.15	1	0	0.58	1	0	0.24
Jon Mig. To Host#1	1	1	1	0.67	1	1	0.57	1	1	0.79	1	1	0.62
VM#3 CheckPointing & Failed	0	1	1	0.74	1	1	0.66	1	1	0.83	0	-	0.37
VM#3 Restart on VM#1	0	-	-	0.74	1	1	0.83	-	-	0.83	-	-	0.62
-	0	1	1	0.79	1	1	0.86	1	1	0.86	1	1	0.69
VM#2 Failed	0	1	1	0.83	1	1	0.89	0	-	0.58	1	1	0.75
VM#0 & VM#2 Failed	0	0	-	0.56	1	1	0.91	0	-	0.43	1	1	0.80
-	0	1	1	0.65	1	1	0.93	1	1	0.54	1	1	0.84
VM#2 CheckPointing & Failed	0	1	1	0.72	1	0	0.60	0	-	0.43	1	1	0.87
Restat on VM#3	-	-	-	0.72	-	-	0.60	-	-	0.54	1	1	0.93
VM#2 CheckPointing & Failed	0	1	1	0.77	1	0	0.45	0	-	0.44	1	1	0.95
Restat on VM#1	-	-	-	0.77	1	1	0.72	-	-	0.54	-	-	0.95
-	0	1	1	0.82	1	0	0.52	1	1	0.63	1	1	0.96
VM#2 Failed	0	1	1	0.85	1	1	0.61	0	-	0.51	1	1	0.96
-	0	1	1	0.88	1	1	0.80	1	1	0.61	1	1	0.97
-	0	1	1	0.90	1	1	0.84	1	1	0.69	1	1	0.97
VM#0 & VM#2 Failed	0	0	-	0.74	1	1	0.87	0	-	0.57	1	1	0.98
-	0	1	1	0.79	1	1	0.90	1	1	0.66	1	1	0.98
VM#0 & VM#2 Failed	0	0	-	0.70	1	1	0.92	0	-	0.57	1	1	0.98

Table 4. Results of Simulating VFT in CloudSim.

Event	Cloudlet		Host#1								Host#2							
			VM#1		VM#2		VM#3		VM#4		VM#1		VM#2		VM#3		VM#4	
	Number	Length	Successful	RA	Successful	RA	Successful	RA	Successful	RA	Successful	RA	Successful	RA	Successful	RA	Successful	RA
VM#1 Failed	2	250	0	0.5	1	1	1	1	1	1	-	-	-	-	-	-	-	-
Pree. Mig. To VM#2	-	-	-	0.83	1	1		1	-	1	-	-	-	-	-	-	-	-
VM#2 Failed	3	500	1	0.88	0	0.66	1	1	1	1	-	-	-	-	-	-	-	-
Pree. Mig. To VM#1	-	-	1	0.92	-	0.88	-	1	-	1	-	-	-	-	-	-	-	-
VM#3 Failed	4	750	1	0.95	1	0.92	0	0.75	1	1	-	-	-	-	-	-	-	-
Pree. Mig. To VM#2	-	-	1	0.96	-	0.92	-	0.91	-	1	-	-	-	-	-	-	-	-
VM#4 Failed	5	1000	1	0.97	1	0.95	1	0.94	0	0.8	-	-	-	-	-	-	-	-
Pree. Mig. To VM#1	-	-	1	0.98	-	0.95	-	0.94	-	0.93	-	-	-	-	-	-	-	-
Fails > 1 => Migration	6	7500	0	0.66	0	0.66	0	0.66	0	0.66	-	-	-	-	-	-	-	-
Pree. Mig.	-	-	0	0.57	0	0.57	0	0.57	0	0.57	-	-	-	-	-	-	-	-
Fails > 1 => Migration	7	7800	0	0.5	0	0.5	0	0.5	0	0.5	-	-	-	-	-	-	-	-
Pree. Mig.	-	-	0	0.44	0	0.44	0	0.44	0	0.44	-	-	-	-	-	-	-	-
Fails > 1 => Migration	8	3700	0	0.4	0	0.4	1	0.5	0	0.5	-	-	-	-	-	-	-	-
Pree. Mig.	-	-	-	0.4	0	0.36	1	0.54	-	0.5	-	-	-	-	-	-	-	-
Fails > 1 => Migration	9	3800	0	0.36	0	0.33	1	0.58	1	0.54	-	-	-	-	-	-	-	-
Pree. Mig.	-	-	0	0.33	-	0.33	-	0.58	1	0.58	-	-	-	-	-	-	-	-
Fails > 1 => Migration	10	3600	0	0.30	0	0.30	1	0.61	1	0.61	-	-	-	-	-	-	-	-
Pree. Mig.	-	-	0	0.28	0	0.28	-	0.61	-	0.61	-	-	-	-	-	-	-	-
Fails > 1 => Migration	11	3650	0	0.26	0	0.26	1	0.64	1	0.64	-	-	-	-	-	-	-	-
Pree. Mig.	-	-	0	0.25	0	0.25	-	0.64	-	0.64	-	-	-	-	-	-	-	-
VM#1 Failed	12	250	0	0.23	1	0.29	1	0.66	1	0.66	-	-	-	-	-	-	-	-
Pree. Mig. To VM#2	-	-	-	0.23	1	0.33	1	0.68	1	0.68	-	-	-	-	-	-	-	-
VM#2 Failed	13	500	1	0.27	0	0.31	1	0.70	1	0.70	-	-	-	-	-	-	-	-
Pree. Mig. To VM#1	-	-	1	0.31	0	0.3	1	0.72	-	0.70	-	-	-	-	-	-	-	-
VM#3 Failed	14	750	1	0.35	1	0.33	0	0.68	1	0.72	-	-	-	-	-	-	-	-
Pree. Mig. To VM#2	-	-	1	0.38	-	0.33	-	0.68	-	0.72	-	-	-	-	-	-	-	-
VM#4 Failed	15	1000	1	0.40	1	0.36	1	0.7	0	0.68	-	-	-	-	-	-	-	-
Pree. Mig. To VM#1	-	-	1	0.43	-	0.36	-	0.7	-	0.68	-	-	-	-	-	-	-	-
VM#1 Failed	16	250	0	0.41	1	0.39	1	0.71	1	0.7	-	-	-	-	-	-	-	-
Pree. Mig. To VM#2	-	-	-	0.41	1	0.41	-	0.71	-	0.7	-	-	-	-	-	-	-	-
VM#1 Failed	17	250	0	0.4	1	0.44	1	0.72	1	0.71	-	-	-	-	-	-	-	-
Pree. Mig. To VM#2	-	-	-	0.4	1	0.46	-	0.72	-	0.71	-	-	-	-	-	-	-	-
VM#1 Failed	18	250	0	0.38	1	0.48	1	0.73	1	0.72	-	-	-	-	-	-	-	-
Pree. Mig. To VM#2	-	-	-	0.38	1	0.5	-	0.73	-	0.72	-	-	-	-	-	-	-	-
VM#1 Failed	19	250	0	0.37	1	0.51	1	0.75	1	0.73	-	-	-	-	-	-	-	-
Pree. Mig. To VM#2	-	-	-	0.37	1	0.53	-	0.75	-	0.73	-	-	-	-	-	-	-	-
VM#1 Failed	20	250	0	0.35	1	0.54	1	0.76	1	0.75	-	-	-	-	-	-	-	-
Pree. Mig. To VM#2	-	-	-	0.35	1	0.56		0.76		0.75	-	-	-	-	-	-	-	-

Table 5. Results of Simulating PrP-FRC in CloudSim.

Event	Host	VM#0			VM#1			VM#2			VM#3			RA
	-	AT	TC	RA	AT	TC	RA	AT	TC	RA	AT	TC	RA	-
-	0	1	1	1	1	1	1	1	1	1	1	1	1	1
-	0	1	1	1	1	0	0.52	1	1	1	1	1	1	0.88
VM#3 Failed	0	1	1	1	1	1	0.61	1	1	1	0	-	0.53	-
Pree. Mig. To VM#1	0	-	-	1	1	1	0.80	-	-	1	-	-	1	0.95
-	0	1	1	1	1	0	0.45	1	1	1	1	1	1	0.86
TC-Fail > 2	0	1	0	0.58	1	0	0.27	1	1	1	1	0	0.56	-
Pree. Mig.	0	-	-	-	-	-	-	1	1	1	-	-	-	0.68
All-Failed	0	1	0	0.37	1	0	0.19	1	0	0.60	1	0	0.35	0.38
Jon Mig. To Host#1	1	1	1	0.79	1	1	0.73	1	1	0.86	1	1	0.78	0.79
VM#3 CheckPointing & Failed	0	1	1	0.83	1	1	0.78	1	1	0.89	0	-	0.46	-
VM#3 Restart on VM#1	0	-	-	0.83	1	1	0.89	-	-	0.89	-	-	0.78	0.85
-	0	1	1	0.86	1	1	0.91	1	1	0.91	1	1	0.82	0.88
VM#2 Failed	0	1	1	0.89	1	1	0.93	0	-	0.62	1	1	0.86	-
VM#2 Pree. Mig. To VM#1	0	-	-	0.89	1	1	0.96	-	-	0.91	-	-	0.86	0.90
VM#0 & VM#2 Failed	0	0	-	0.62	1	1	0.97	0	-	0.63	1	1	0.88	-
Pree. Mig. To VM#1 & VM#3	0	-	-	0.89	1	1	0.98	-	-	0.91	1	1	0.94	0.93
-	0	1	1	0.91	1	1	0.98	1	1	0.93	1	1	0.95	0.94
VM#2 Failed	0	1	1	0.93	1	0	0.67	0	-	0.66	1	1	0.96	-
VM#2 Pree. Mig. To VM#3	0	-	-	0.93	-	-	0.67	-	-	0.93	1	1	0.98	0.88
VM#2 Failed	0	1	1	0.94	1	0	0.52	0	-	0.67	1	1	0.98	-
VM#2 Pree. Mig. To VM#1	0	-	-	0.94	1	1	0.76	-	-	0.93	-	-	0.98	0.90
-	0	1	1	0.95	1	0	0.57	1	1	0.94	1	1	0.98	0.86
VM#2 Failed	0	1	1	0.96	1	1	0.65	0	-	0.70	1	1	0.99	-
VM#2 Pree. Mig. To VM#1	0	-	-	0.96	1	1	0.82	-	-	0.94	-	-	0.99	0.93
-	0	1	1	0.97	1	1	0.86	1	1	0.95	1	1	0.99	0.94
-	0	1	1	0.97	1	1	0.89	1	1	0.96	1	1	0.99	0.95
VM#0 & VM#2 Failed	0	0	-	0.84	1	1	0.91	0	-	0.76	1	1	0.99	-
Pree. Mig. To VM#1 & VM#3	0	-	-	0.97	1	1	0.95	-	-	0.96	1	1	0.99	0.97
-	0	1	1	0.98	1	1	0.96	1	1	0.97	1	1	0.99	0.97
VM#0 & VM#2 Failed	0	0	-	0.87	1	1	0.97	0	-	0.79	1	1	0.99	-
Pree. Mig. To VM#1 & VM#3	0	-	-	0.98	1	1	0.98	-	-	0.97	1	1	0.99	0.98

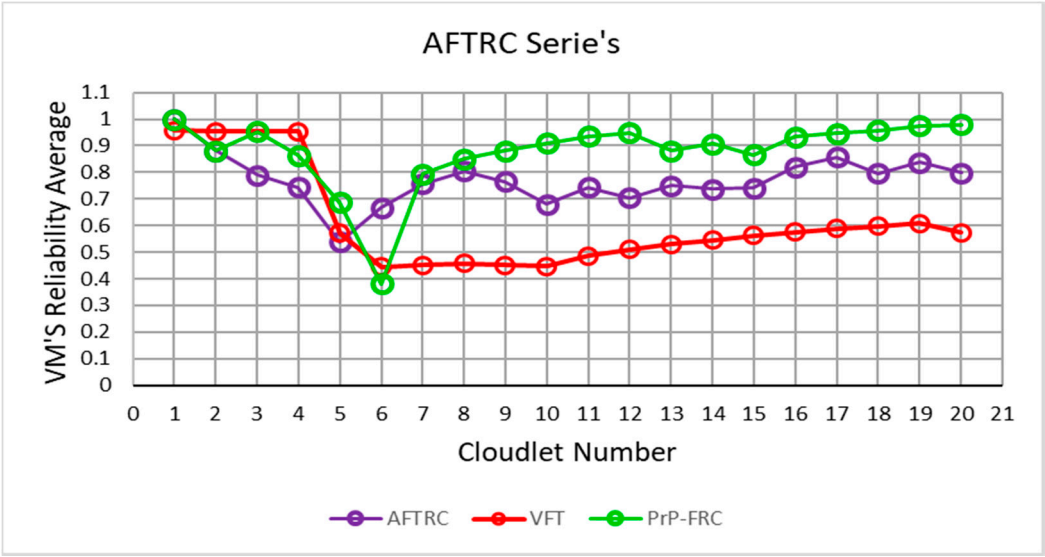

Figure 4. Comparison of Diagrams of Reliability of Architectures.

As it can be understood from the data center configuration and Vms of Tables 1 and 2, the Cloudlets had a data size with a constant value of 300. However, the lengths of jobs were considered to be different, so that, when it faced different faults, various behaviors could be appeared. Regarding the time variation value, the acceptable time for the implementation was considered to be between 1 to 7 s. Thus, in the case that one Vm presented an output in less than 1 s, it would be unaccepted, and if this time was more than 7 s, the output would be unacceptable, as well. Therefore, this time range should be considered in the first phase of analyzing virtual machines' outputs. Regarding the second phase, if the output was produced in the acceptable time range, the validation of the output's amount would be considered to see whether the amount of the produced output of Vm is acceptable or not.

The reliability of a Host Vm equals the average of all Vms' reliability of one Host. It is well evident that, when all Vms have acceptable outputs, the amount of total reliability would be increased. Moreover, when all of the Vms fail, it is quite natural that their total reliability would be decreased. These two, are respectively the best and worst possible modes, which include the Max and Min of the reliability. In the case that Vm has some acceptable outputs and lacks the expected output, the reliability would be high or low. On the other hand, as time passes, the results become more important. Our criteria is the current time and status of each Vm. It is possible that a Vm may have had an acceptable output before, but it lacks proper output at the present, or vice versa. Therefore, the number of successful outputs or failed ones, as well as the success-effect coefficient versus the fail-effect coefficient is considered to be the effective parameters when doing calculations.

For example, we assume that the Success-Effect coefficient, which is indicated as SE, equals to 0.1 and the Fail-Effect coefficient, indicated as FE, equals to 0.03, the importance factor, indicated as IF, equals 0.6, whose highest value would be 1. The number of success and failure would be considered, respectively, as CSC and CFC, in which CSC is the abbreviation for Continues Success Counter and CFC is the abbreviation for Continues Fail Counter, whose first values are considered to be zero. As a result, the amount of every Vm's reliability equals to the amount of previous round's reliability, plus the multiplication of the previous round's reliability when multiplied in CSC and the success-effect or fail-effect coefficient, which is multiplied to the importance factor.

5. Validation Platform

5.1. Simulation Environment

Many scientific calculations use workflow technologies to manage complexity [6]. Workflow technology is used to the schedule calculating tasks on distributed resource to manage task interdependence. The aim of this scheduling is to optimize the mapping between tasks and appropriate resources. One of the important factors that have a great influence on choosing a scheduling strategy is the dependence and independence among the tasks. Some tasks have to be done in succession; these kinds of tasks are called dependent tasks. In contrast, there is another category of tasks that are simultaneously run or in a special order; these kinds of tasks are called independent tasks. Scheduling dependent tasks is known as workflow scheduling.

A simulated environment has been implemented based on the Pegasus-WMS workflow management system to validate the architecture proposed in this article (Pegasus-WMS). In Pegasus, the workflows are described as Direct A cycle Graphs (DAGs). In DAG, each node represents one of the tasks. The edges of a DAG also represent the interdependence between those tasks. Montage and CyberShake are the most famous scientific workflows. Montage is applied for the processing and transmitting images that have been used in NASA. Figure 5a shows the Montage Workflow. Figure 5b also shows CyberShake Workflow. Cybershake Workflow has been used to process the waves at the California Seismological Center.

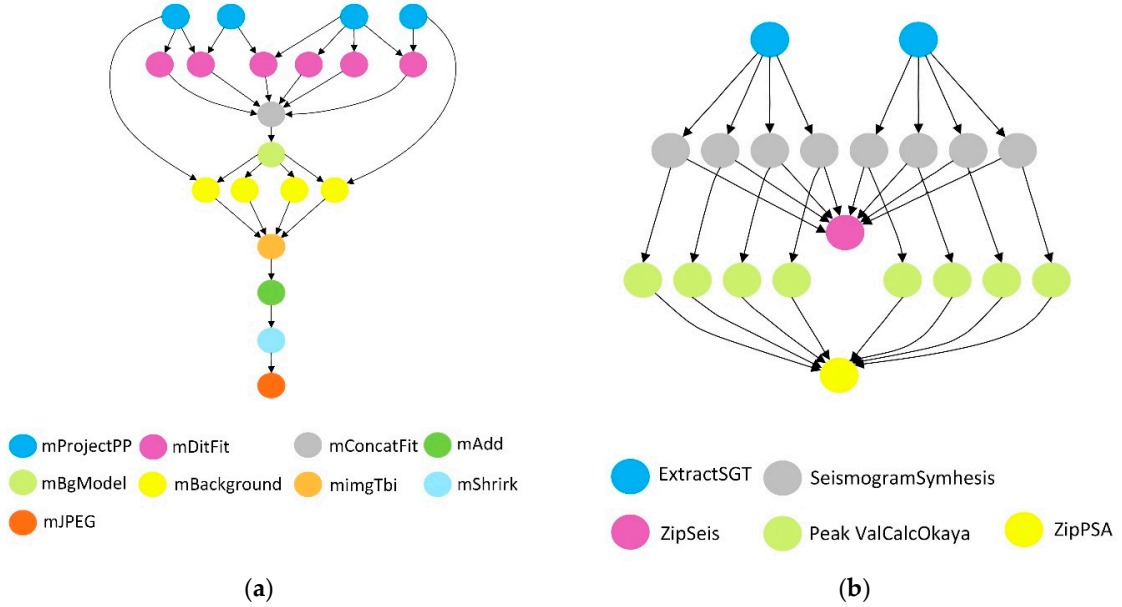

Figure 5. (**a**) Montage workflow; (**b**) Cybershake workflow.

The Pegasus-WMS approach acts as a bridge between the field of science and the field of action expressing the connection between them. In addition to executing a described abstract workflow, the Pegasus Workflow Management System has the ability to translate the tasks into the jobs. Subsequently, it also attends the running of those jobs. This workflow management system is capable of simultaneously executing data management and running the jobs. Additionally, it has the ability to monitor job execution and tracking. Eventually, Pegasus can handle them in the case of failure. The mentioned actions are performed by the five Pegasus subsystems. Figure 6 shows the architecture of the Pegasus workflow management system.

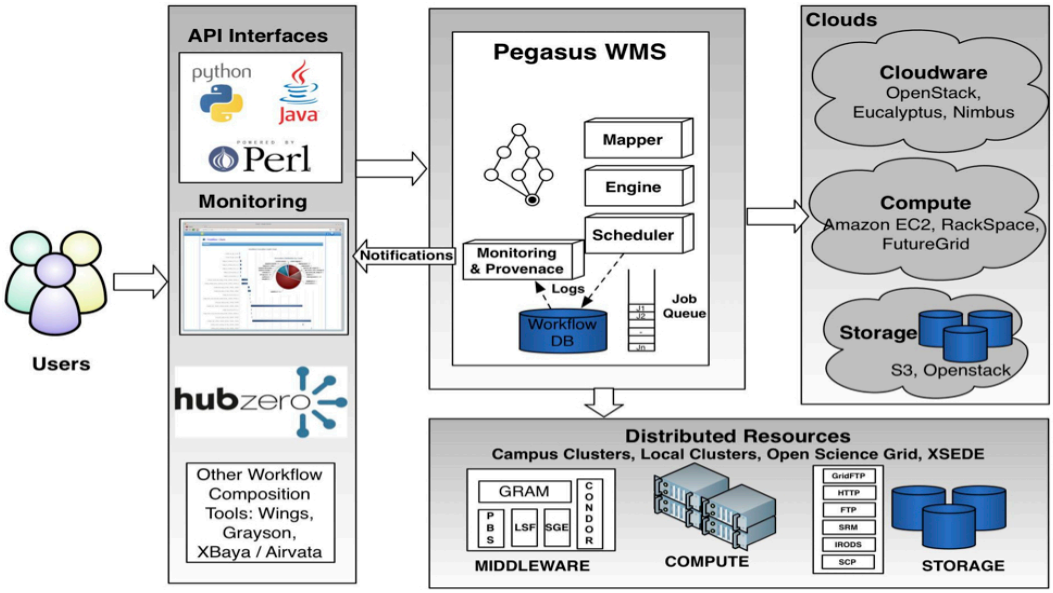

Figure 6. Pegasus architecture [34].

The first major Pegasus subsystem is Mapper. Mapper is the producer of an executive workflow that is based on an abstract workflow that was provided by the user. The second sub-system is the local execution engine, which is responsible for submitting the jobs that are defined by the workflow. Submitting the jobs is done based on the workflow. Subsequently, jobs' states are tracked and the readiness timing of running those jobs is determined. The next sub-system is job scheduler, which is responsible for the management of unique job scheduling and monitoring their implementation on local or remote resources. The remote execution engine's sub-system manages the execution of one or more tasks based on the possible or probable structures of a sub-workflow on one or more remote computing nodes. Finally, the subsystem of the monitoring component is responsible for monitoring at run time, which monitors the execution of a workflow. The analysis of jobs in a workflow, and populating them in a workflow database, is the responsibility of this subsystem.

Basic structures or main components of a scientific workflow include process, pipeline, data distribution, data aggregation, and, finally, data redistribution. The VFT and AFTRC architectures are depicted in Figure 7a,b, respectively.

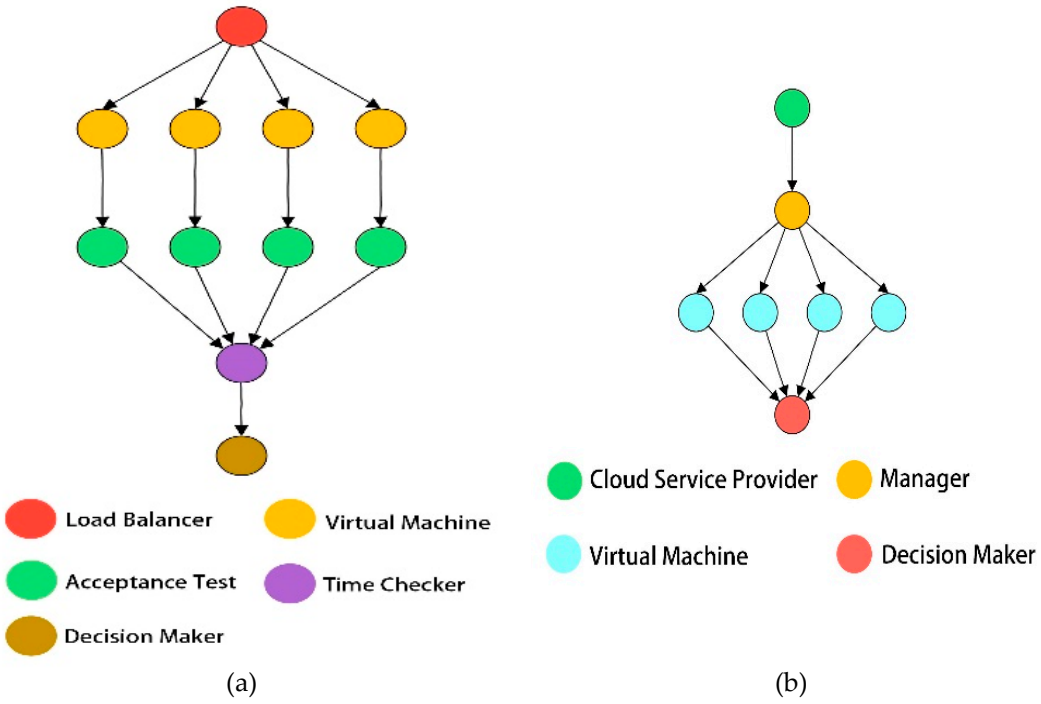

Figure 7. (**a**) Our proposed AFTRC workflow; (**b**) Our proposed VFT workflow.

The PrP-FRC workflow proposed architecture has been presented in Figure 8. Obviously, the basic structures of the process, data distribution, data aggregation, and pipeline have been exploited, in the implementation of the workflows of VFT and AFTRC architectures and the proposed PrP-FRC architecture.

The evaluations of the proposed architecture when compared to the VFT and AFTRC architectures have been conducted in terms of the run time of the relevant workflows. The experiments have been performed using the above-described simulation environment. In the following subsections, the results of the carried-out simulations have been described in detail.

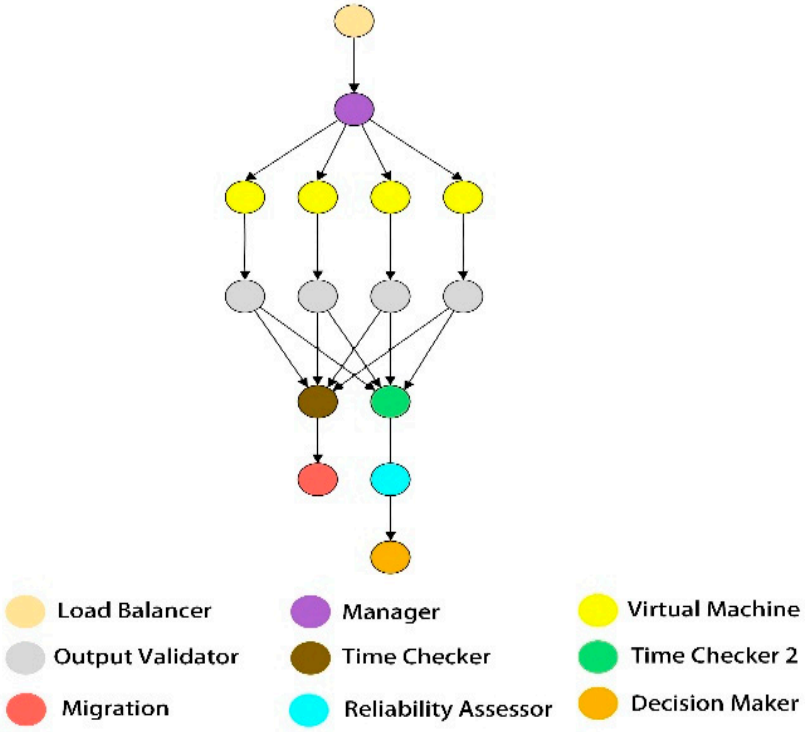

Figure 8. Our proposed PrP-FRC workflow.

5.2. Simulation Results

Tables 6–8 presented the results of the simulations of AFTRC and VFT architectures and the proposed PrP-FRC architecture in the Pegasus-WMS simulator.

The proposed PrP-FRC architecture has been evaluated with VFT and AFTRC architectures in terms of two criteria. The average execution time and reliability are considered as two comparative criteria.

Figure 9 shows the average execution time for each architectural workflow. It is clear that the PrP-FRC architecture had the highest average of execution time, since it has implemented all Proactive and Reactive policies. The AFTRC architecture also had the lowest average of execution time, which was not a hybrid architecture, rather it was considered as Reactive architecture.

On the other hand, as shown in Figure 10, the highest level of fault tolerance was provided by the proposed PrP-FRC architecture, and the VFT hybrid architecture was in the second place. The reliability of this architecture was less than PrP-FRC and more than the reliability of AFTRC architecture. Additionally, in Figure 11, the number of failed and succeed tasks or jobs has been illustrated in one of the proposed simulation rounds as an example.

Table 6. Results of simulating AFTRC in Pegasus-WMS.

Event	AFTRC										
		VM #1		VM #2		VM #3		VM #4			
	Job-Number	AT &TC	Reliability	AT & TC	Reliability	AT & TC	Reliability	AT & TC	Reliability	Fail-Count	AVG.
-	Start	1	0.500	1	0.500	1	0.500	1	0.500	0	0.50
-	1	0	0.49	1	0.53	1	0.53	0	0.49	2	0.51
-	2	0	0.47	1	0.57	1	0.57	1	0.53	1	0.53
-	3	0	0.44	0	0.55	1	0.62	0	0.51	3	0.53
Job #5 Migrating to other Host	4	0	0.49	0	0.59	0	0.66	0	0.56	4	0.57
Job #6 Migrating to other Host	5	0	0.51	0	0.61	0	0.68	0	0.58	4	0.59
-	6	1	0.56	1	0.68	0	0.61	1	0.65	1	0.62
-	7	1	0.63	0	0.66	0	0.53	1	0.73	2	0.63
-	8	1	0.71	1	0.75	1	0.59	0	0.7	1	0.68
-	9	1	0.81	1	0.86	0	0.56	1	0.8	1	0.75
-	10	1	0.93	0	0.81	0	0.52	0	0.77	3	0.75

Table 7. Results of simulating VFT in Pegasus-WMS.

Event	VFT										
		VM #1		VM #2		VM #3		VM #4			
	Job-Number	SC & TDC	SR	SC &TDC	SR	SC & TDC	SR	SC& TDC	SR	Fail-Count	AVG.
-	1-default	1	0.50	1	0.50	1	0.5	1	0.50	0	0.50
Replica #1 & #2 failed	1	0	0.49	1	0.53	1	0.53	0	0.49	2	-
Increase Failed-VM Reliability	-	half-recovery	0.49	-	-	-	-	half-recovery	0.49	-	-
Replica #1 preemptive migrate to VM #4	-	-	-	-	-	-	-	1	0.52	-	-
Replica #4 preemptive migrate to VM #1	-	1	0.52	-	-	-	-	-	-	-	-
Final Reliability	-	-	0.52	-	0.53	-	0.53	-	0.52	-	0.52
above events done for each failed job's						-					
-	2	0	0.54	1	0.57	1	0.57	1	0.56	1	0.56
-	3	0	0.55	0	0.58	1	0.62	0	0.57	3	0.58
-	4	0	0.55	0	0.59	0	0.63	0	0.58	4	0.58
-	5	0	0.54	0	0.59	0	0.64	0	0.58	4	0.58
-	6	1	0.59	1	0.66	0	0.63	1	0.65	1	0.63
-	7	1	0.66	0	0.68	0	0.60	1	0.73	2	0.66
-	8	1	0.75	1	0.77	1	0.68	0	0.74	1	0.73
-	9	1	0.86	1	0.88	0	0.69	1	0.84	1	0.81
-	10	1	0.99	0	0.88	0	0.69	0	0.84	3	0.85

Table 8. Results of simulating PrP-FRC in Pegasus-WMS.

Event	PRP-FRC										
		VM #1		VM #2		VM #3		VM #4			
	Job-Number	SC&TDC	SR	SC&TDC	SR	SC&TDC	SR	SC&TDC	SR	Fail-Count	AVG.
-	1-default	1	0.50	1	0.50	1	0.5	1	0.50	0	0.50
Replica #1 & #2 failed	1	0	0.49	1	0.53	1	0.53	0	0.49	2	-
Increase Failed-VM Reliability	-	half-recovery	0.49	-	-	-	-	half-recovery	0.49	-	-
Replica #1 preemptive migrate to VM #4	-	-	-	-	-	-	-	1	0.52	-	-
Replica #4 preemptive migrate to VM #1	-	1	0.52	-	-	-	-	-	-	-	-
Final Reliability	-	-	0.52	-	0.53		0.53	-	0.52	-	0.52
Repeat first 5 formula	2	0	0.54	1	0.57	1	0.57	1	0.56	1	0.56
Repeat first 5 formula	3	0	0.55	0	0.58	1	0.62	0	0.57	3	0.58
Job migration - Reliability increasing	4	0	0.57	0	0.62	0	0.66	0	0.61	4	0.61
Job migration - Reliability increasing	5	0	0.56	0	0.63	0	0.68	0	0.63	4	0.62
Repeat first 5 formula	6	1	0.62	1	0.69	0	0.67	1	0.69	1	0.66
Repeat first 5 formula	7	1	0.69	0	0.71	0	0.64	1	0.77	2	0.70
Repeat first 5 formula	8	1	0.78	1	0.80	1	0.7	0	0.78	1	0.77
Repeat first 5 formula	9	1	0.89	1	0.91	0	0.73	1	0.89	1	0.85
Repeat first 5 formula	10	1	1	0	0.91	0	0.7	0	0.89	3	0.88

Figure 9. Average of execution time of the architectures.

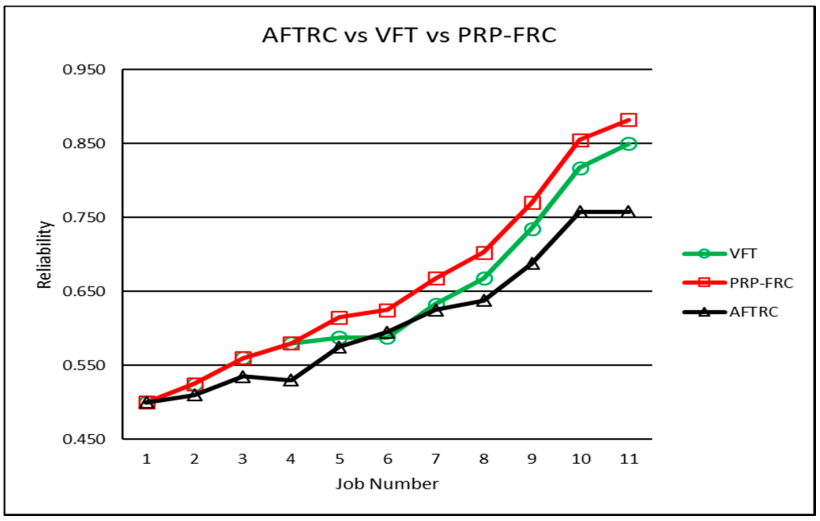

Figure 10. Reliability Rate for each architecture.

Workflow Wall Time	2 mins 46 secs
Workflow Cumulative Job Wall Time	31 secs
Cumulative Job Walltime as seen from Submit Side	1 min 7 secs
Workflow Cumulative Badput Time	0 secs
Cumulative Job Badput Walltime as seen from Submit Side	0 secs
Workflow Retries	0

Workflow Statistics

This Workflow

Type	Succeeded	Failed	Incomplete	Total	Retries	Total + Retries
Tasks	43	0	0	43	0	43
Jobs	51	0	0	51	0	51
Sub Workflows	0	0	0	0	0	0

Entire Workflow

Type	Succeeded	Failed	Incomplete	Total	Retries	Total + Retries
Tasks	43	0	0	43	0	43
Jobs	51	0	0	51	0	51
Sub Workflows	0	0	0	0	0	0

Figure 11. The number of jobs and tasks and their status.

6. Modelling and Fuzzy Analysis of Architectures

Our real world is the world of uncertainties and ambiguities. Given that fault tolerance is a qualitative parameter and it is associated with uncertainty, the fuzzy logic is used in modelling and analysing this important feature. Different methods have been presented in different papers for reliability analysis. One of these methods, as referred in [35], is the modelling of the fault tolerance based on fuzzy logic. The support of fuzzy systems from the rapid pattern generation and incremental optimization increases the importance of the results. Furthermore, the evaluating frameworks having the tolerant characteristic of the faults have been introduced in the various architectures of [36,37]. The evaluating frameworks were fuzzy-base, which analysed and measured the intended capability while considering various parameters, like those methods that were used in detection phases and fault refinements of various designed fuzzy Inference systems. A fuzzy evaluation has been formed of four main parts, including the Fuzzier, Defuzzier, and Fuzzy Inference System, which are briefly referred to as FIS and eventually the Fuzzy Data Base Rules.

The role of the fuzzier of this system is to convert the input terms to a linguistic term set. This is conducted to be the membership function. The fuzzy inference engine uses the database of fuzzy rules in order to obtain the fuzzy output. It is clear that the fuzzy rules have been stored on a particular database and the fuzzy inference engine exploits them. Additionally, the defuzzier converts the fuzzy output of the fuzzy inference engine to a crisp value. An assessment of the architectures has been carried out on four separate fuzzy engines. These engines are termed FIS (1), FIS (2), FIS (3), and FIS (4), respectively. Figure 12 shows the inputs and outputs of the engines.

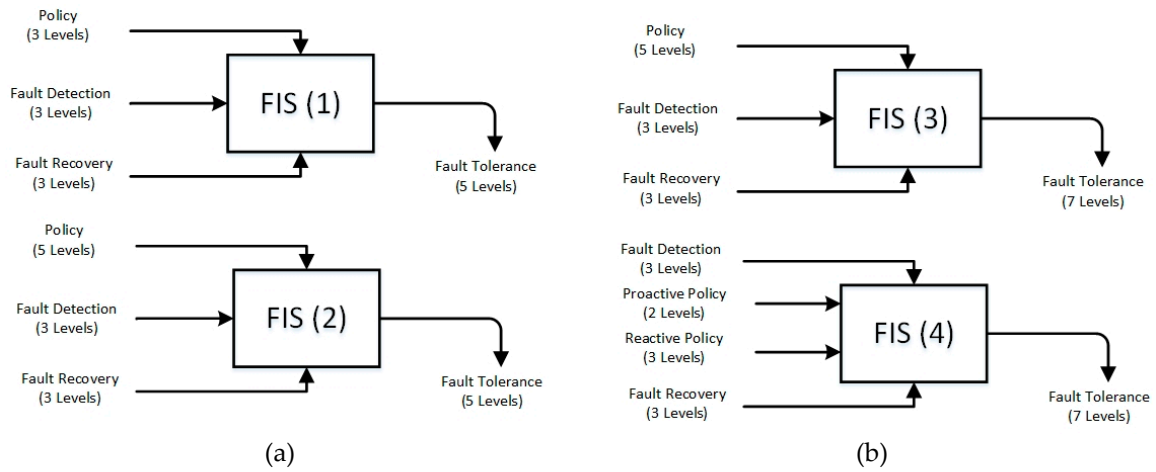

Figure 12. (**a**) Inputs of the fuzzy engines; (**b**) outputs of the fuzzy engines.

All of the designed engines have an output. The engines of FIS (1), FIS (2), and FIS (3) have three inputs. In addition, the number of database rules and engine membership functions have similarities and differences. The trapezoid membership functions have been used for designing the FIS (1) engine. Each of the triple inputs of these engines has been considered as a three-level input, including the low, medium, and high levels. Moreover, the output of this fuzzy evaluation engine has been designed on five levels. The labels of the linguistic variables in FIS (1) are very low, low, medium, high, and very high. The results of the assessments that were conducted by each architecture by FIS (1) engine have been reported in Table 9. An important point is that the VFT architecture and the proposed PRP-FRC architecture, which are considered to be hybrid architectures, are not valuable to this engine, because the first input of this engine is designed as a three-level input. The first input of this engine is related to the situation of the policies that are used in the architecture.

The trapezoid membership functions have been used for designing the FIS (2) engine like the FIS (1) engine. The main difference between the two engines is in terms of the numbers of the first input linguistic variables of these engines. The first input is dedicated to the policies that are used in

each of the architectures. This input has been considered as three-level on FIS (1). On FIS (2), the policy input has been designed as five-level, with the labels being very low, low, medium, high, and very high. Table 10 shows the results reported on the assessments that were conducted on the measurement of the fault tolerance of each architecture.

Table 9. Assessment results of the architectures in our proposed FIS (1) engine.

Fault Tolerance Architectures	Policy Level	Fault Detection Level	Fault Recovery Level	Fault Tolerance Present	Rule No.
Map—Reduce	Medium	Medium	High	66.67	15
Haproxy	Medium	Medium	High	66.67	15
BFT-Cloud	Low	High	High	66.67	9
Gossip	Low	High	High	66.67	9
MPI	Medium	Medium	High	66.67	15
FTM	High	Medium	High	83.33	24
FTM-2	Medium	Medium	High	66.67	15
LLFT	Low	High	Medium	50.00	8
FTWS	Medium	Medium	High	66.67	15
Candy	Low	Medium	High	50.00	6
FT-Cloud	Low	Medium	High	50.00	6
Magi Cube	Medium	Medium	High	66.67	15
Vega Warden	Medium	Medium	Medium	50.00	14
AFTRC	High	Medium	High	83.33	24
PLR	High	Low	High	66.67	21

Table 10. Comparing FT capability of our proposed architecture with other existing architectures in the FIS (2) engine.

Fault Tolerance Architectures	Policy Level	Fault Detection Level	Fault Recovery Level	Fault Tolerance Present	Rule No.
Map—Reduce	Low	Medium	High	50%	15
Haproxy	Low	Medium	High	50%	15
BFT-Cloud	Very Low	High	High	50%	18
Gossip	Very Low	High	High	50%	18
MPI	Low	Medium	High	50%	15
FTM	Medium	Medium	High	50%	24
FTM-2	Low	Medium	High	50%	15
LLFT	Very Low	High	Medium	50%	8
FTWS	Low	Medium	High	50%	15
Candy	Very Low	Medium	High	50%	6
FT-Cloud	Very Low	Medium	High	50%	6
Magi Cube	Low	Medium	High	50%	15
Vega Warden	Low	Medium	Medium	50%	14
AFTRC	Medium	Medium	High	50%	24
PLR	Medium	Low	High	50%	21
VFT	Medium	Low	Low	50%	19
PRP-FRC (Proposed)	Very High	Medium	High	75%	42

It is observed that architectures' VFT and PRP-FRC, in which the FIS (1) engine was not evaluated, have been measured by the FIS (2) engine. The weakness of the FIS (2) engine on the conducted assessments could be due to the numbers of the membership functions that are considered on the output of this engine. The FIS (2) output had been produced on a five-level with labels, such as very low, low, medium, high, and very high. The FIS (3) engine has been designed in order to improve the performance of this engine on the assessments.

FIS (3) was designed in order to render the assessment results better and more accurate. The main difference between FIS (2) and FIS (3) can be expressed in two cases. The first one is that FIS (3), unlike FIS (2), which uses trapezoid membership functions, has used triangle membership functions. The second, FIS (3) output is a linguistic variable with seven levels. The FIS (3) output labels include very low, low, low medium, medium, high medium, high, and very high. Thus, on the basis of the obtained results of the evaluating architectures, the accuracy on the measurement of the FT ability of the architectures has remarkably increased. Table 11 presents the results of this assessment.

Table 11. Comparing FT capability of our proposed architecture with other existing architectures in the FIS (3) engine.

Fault Tolerance Architectures	Policy Level	Fault Detection Level	Fault Recovery Level	Fault Tolerance Present	Rule No.
Map—Reduce	Low	Medium	High	50%	15
Haproxy	Low	Medium	High	50%	15
BFT-Cloud	Very Low	High	High	50.44%	9
Gossip	Very Low	High	High	50.44%	9
MPI	Low	Medium	High	50%	15
FTM	Medium	Medium	High	66.67%	24
FTM-2	Low	Medium	High	50%	15
LLFT	Very Low	High	Medium	33.83%	8
FTWS	Low	Medium	High	50%	15
Candy	Very Low	Medium	High	34.3%	6
FT-Cloud	Very Low	Medium	High	34.3%	6
Magi Cube	Low	Medium	High	50%	15
Vega Warden	Low	Medium	Medium	33.33%	14
AFTRC	Medium	Medium	High	66.67%	24
PLR	Medium	Low	High	50.5%	21
VFT	Medium	Low	Low	33.33%	19
PRP-FRC (Proposed)	Very High	Medium	High	82.39%	42

FIS (2) and FIS (3) are able to evaluate hybrid architectures, but separating the proactive and reactive policies that have been applied in the architectures represents their weakness. FIS (4) has been presented in order to remove the weaknesses of FIS (2) and FIS (3). On FIS (4), the input related to the situation of the policies used in the architectures, has been divided into two separate inputs.

Four inputs have been used for designing FIS (4). The first input is related to the proactive policies and the second one is related to the reactive policies. Additionally, the third and fourth inputs are related to the situation of fault detection and fault recovery on each architecture. Similar to FIS (1) and FIS (2), the trapezoid membership functions have also been used to design FIS (4). The FIS (4) output, like FIS (3), is a linguistic variable with seven different levels to achieve more accurate measurement of the FT ability of the architectures. The significant difference of FIS (4) with three previous FIS, is that, in the previous engines, each of the architectures on one of the engine database rules was fire, but, on FIS (4), each of the architectures on three of the engine database rules of FIS (4) was simultaneously fire. The assessment reports of each of the architectures have been presented as a separately fired rule of each of them in Table 12.

Table 12. Evaluation report of the FT capabilities of various architectures, compared to the proposed architecture of PrP_FRC by using the FIS (4) engine.

Fault Tolerance Architectures	Pro Policy Level	Re Policy Level	Fault Detection Level	Fault Recovery Level	Fault Tolerance Present	Fired Rule No.
Map—Reduce	High	-	Medium	High	66.66%	33,69,78
Haproxy	-	Medium	Medium	High	58.33%	15,60,87
BFT-Cloud	-	Low	High	High	58.33%	9,63,81
Gossip	-	Low	High	High	58.33%	9,63,81
MPI	-	Medium	Medium	High	58.33%	15,60,87
FTM	-	Low	Medium	High	66.66%	24,60,96
FTM-2	-	Medium	Medium	High	58.33%	15,60,87
LLFT	-	Low	High	Medium	41.67%	8,62,80
FTWS	-	Medium	Medium	High	58.33%	15,60,87
Candy	-	Low	Medium	High	41.67%	6,60,78
FT-Cloud	Low	-	Medium	High	41.67%	6,60,78
Magi Cube	-	Medium	Medium	High	58.33%	15,60,87
Vega Warden	-	Medium	Medium	Medium	50%	14,59,86
AFTRC	-	High	Medium	High	66.66%	24,60,96
PLR	-	High	Low	High	50%	21,57,93
VFT	High	Low	Low	Low	25%	28,64,73
PRP-FRC (Proposed)	High	High	Medium	High	83.33%	51,69,96

7. Discussion

Inside the DMTF, whose focus is on the FCAPS capabilities in management, there is another group, which is known as the Cloud commands. The main task of this group is the development of service measurement index technologies, which is briefly called SMI. The goal of the SMI is to evaluate and measure the aspects of cloud performance in the standard form, and some methods were established in this field. The purpose of this article was to present a new hybrid architecture of fault tolerance. The simulation results in CloudSim, Pegasus-WMS, as well as fuzzy modeling and evaluation, confirmed the increasing fault tolerance and reliability in the implementation of the proposed architecture.

VFT architecture has been a hybrid architecture, but the biggest weakness of this architecture has happened when all of the nodes failed. In this case, on the VFT architecture, there was a feedback toward the fault handler that an appropriate decision should be taken. In this condition, the architecture should be designed and implemented, such that the job is migrated toward another host; but, since there is no policy of job migration in this architecture, the internal feedback of the same host occurred, which reduced the recovery ability and, finally, the architecture fault tolerance. Additionally, when the policy of checkpoint/restart has not been implemented in this architecture, in the case of any task being encountered to the lowest fault, again, the task was entirely implemented so that the general effect was significantly affected. If this policy was used, then, implementing that task began from the last checkpoint and, consequently, the architecture efficiency remarkably increased.

In the proposed architecture of PrP-FRC, simultaneously implementing all of the proactive and reactive policies has been sought. It is clear that in this case, the significant weaknesses of the VFT architecture, due to the lack of policies of job migration and checkpoint/restart, would not be in the proposed architecture of PrP-FRC. The phase of fault detection has been carried out due to the decision marker module by the other detection method.

In the proposed architecture, each of the three fault detection methods has been implemented. The AT module that investigated the output of each of the VMs has provided a self-detection method. The TC module that had straight supervision on the time validity of the output generated by each of the VMs caused the use of other detection methods in this architecture. Finally, due to the roles played by the RA and DM modules in the PrP-FRC architecture, this architecture had not used the group detection method in the fault detection phase.

In the fault recovery phase, the VFT architecture has implemented system recovery, due to the feedback that was in its final output. In the PrP-FRC architecture, both the final architecture output and the output of each VM has been separately implemented. They designed feedback to the management system to trigger the recovery, which was simultaneously carried out on two levels, which included the node and the system.

The evaluations by FIS (1) to FIS (4) fuzzy engines also showed that the fault tolerance in the PrP-FRC proposed architecture has increased from 16 to 25 percent in comparison with the AFTRC architecture. The PrP-FRC architecture also showed a 25 to 58 percent increase in the VFT hybrid architecture among the fuzzy evaluators engines.

The fault mask, which is one of the fault recovery methods, has been implemented in the AFTRC simple architecture because the outputs of all the VMs were collected and the fault effects were disappeared. However, in the proposed PrP-FRC, due to the feedback considered in the output of each VM, the node recovery method has been implemented and the fault effects that were made on the VMs were not masked. In addition, in the AFTRC architecture, the node recovery method has not been implemented and it was just used for masking fault effects. As a new idea in the following of leading research studies, reference may be made to a method for using the strategy of masking the fault effects in the PrP-FRC architecture.

8. Conclusions

Presenting a new architecture of fault tolerance, which simultaneously uses proactive and reactive policies, was the goal of this research. The proposed PrP-FRC architecture has covered fault tolerance on the quintuple phases, which consisted of fault forecasting, fault prevention, fault detection, fault isolation, and fault recovery. The main reason was to fully cover each of the five phases of fault tolerance by the aforementioned architecture, while simultaneously using all of the proactive and reactive policies in this architecture. Given the full implementation of all the policies in the PrP-FRC proposed architecture, it was expected that the proposed architecture would provide higher fault tolerance than previous architectures. The results of the simulations that were performed in the CloudSim and the comparison and simulation of proposed architectural workflow confirmed the increasing of the aforementioned capability. The time execution of PrP-FRC proposed architecture had no significant difference with previous architectures and it had only slightly increased. This feature highlighted the prominence of the proposed architecture.

The proposed architecture in this paper had simultaneously used methods of self-detection and other detection in the fault detection phase. Additionally, this architecture had simultaneously used the triple methods of fault mask, node recovery, and system recovery in the fault recovery phase. If a group detection method has been used in fault detection phase in the PRP-FRC, then it could be considered as a complete architecture for implementing all fault detection methods. Using and implementing this method on fault detection in the aforementioned architecture can be followed as a future work.

Author Contributions: The authors contributed equally to the reported research and writing of the paper. All authors have read and approved the final manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Al-Fuqaha, A.; Guizani, M.; Mohammadi, M.; Aledhari, M.; Ayyash, M. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Commun. Surv. Tutor.* **2015**, *17*, 2347–2376. [CrossRef]
2. Yaqoob, I.; Hashem, I.A.T.; Ahmed, A.; Kazmi, S.M.A.; Hong, C.S. Internet of things forensics: Recent advances, taxonomy, requirements, and open challenges. *Future Gener. Comput. Syst.* **2019**, *92*, 265–275. [CrossRef]
3. Lin, J.-W.; Chelliah, P.R.; Hsu, M.C.; Hou, J.-X. Efficient Fault-Tolerant Routing in IoT Wireless Sensor Networks Based on Bipartite-Flow Graph Modeling. *IEEE Access* **2019**, *7*, 14022–14034. [CrossRef]
4. Kaiwartya, O.; Abdullah, A.H.; Cao, Y.; Lloret, J.; Kumar, S.; Shah, R.R.; Prasad, M.; Prakash, S. Virtualization in wireless sensor networks: Fault tolerant embedding for internet of things. *IEEE Internet Things J.* **2018**, *5*, 571–580. [CrossRef]
5. Cheraghloua, M.N.; Khadem-Zadeh, A.; Haghparast, M. A survey of fault tolerance architecture in cloud computing. *J. Netw. Comput. Appl.* **2016**, *61*, 81–92. [CrossRef]
6. Kumar, S.; Rana, D.S.; Dimri, S.C. Fault Tolerance and Load Balancing algorithm in Cloud Computing: A Survey. *IJARCCE Int. J. Adv. Res. Comput. Commun. Eng.* **2015**, *4*, 92–96.
7. Saikia, L.P.; Devi, Y.L. Fault Tolerance Techniques and algorithms in cloud computing. *Int. J. Comput. Sci. Commun. Netw.* **2014**, *4*, 1–8.
8. Amin, Z.; Sethi, N.; Singh, H. Review on Fault Tolerance Techniques in Cloud Computing. *Int. J. Comput. Appl.* **2015**, *116*, 11–17. [CrossRef]
9. Hashem, I.; Yaqoob, I.; Anuar, N.; Mokhtar, S.; Gani, A.; Ullah Khan, S. The rise of "big data" on cloud computing: Review and open research issues. *Inf. Syst.* **2015**, *47*, 98–115. [CrossRef]
10. Inukollu, V.; Arsi, S.; Ravuri, S. Security Issues Associated with Big Data in Cloud Computing. *Int. J. Netw. Secur. Its Appl. (IJNSA)* **2014**, *6*, 45. [CrossRef]
11. Abadi, D.J. Data Management in the Cloud: Limitations and Opportunities. *Bull. Ieee Comput. Soc. Tech. Comm. Data Eng.* **2009**, *32*, 3–12.

12. Purcell, B.M. Big data using cloud computing. *J. Technol. Res.* **2014**, *5*, 1–8.
13. Ahuja, S.P.; Moore, B. State of Big Data Analysis in the Cloud. *Netw. Commun. Technol.* **2013**, *2*, 62. [CrossRef]
14. Zheng, Z.; Zhou, T.C.; Lyu, M.R.; King, I. FTCloud: A Component Ranking Framework for Fault–Tolerant Cloud Applications. In Proceedings of the 2010 IEEE 21st International Symposium on Software Reliability Engineering, San Jose, CA, USA, 1–4 November 2010; pp. 398–407. [CrossRef]
15. Kaur, J.; Kinger, S. Analysis of Different Techniques Used for Fault Tolerance. *IJCSIT Int. J. Comput. Technol.* **2013**, *4*, 737–741.
16. Sudha Lakshmi, S. Fault Tolerance in Cloud Computing. *Int. J. Eng. Sci. Res. IJESR* **2013**, *4*, 1285–1288.
17. Egwutuoha, I.P.; Chen, S.; Levy, D.; Selic, B. A Fault Tolerance Framework for High Performance Computing in Cloud. In Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), Ottawa, ON, Canada, 13–16 May 2012; pp. 709–710. [CrossRef]
18. Bala, A.; Chana, I. Fault Tolerance-Challenges, Techniques and Implementation in Cloud Computing. *IJCSI Int. J. Comput. Sci. Issues* **2012**, *9*, 288.
19. Zhang, Y.; Zheng, Z.; Lyu, M.R. BFTCloud: A Byzantine Fault Tolerance Framework for Voluntary-Resource Cloud Computing. In Proceedings of the 2011 IEEE International Conference on Cloud Computing (CLOUD), Washington, DC, USA, 4–9 July 2011; pp. 444–451.
20. Lim, J.; Suh, T.; Gil, J.; Yu, H. *Information Systems Frontiers*; Springer: Berlin/Heidelberg, Germany, 2013.
21. Jhawar, R.; Piuri, V.; Santambrogio, M. A Comprehensive Conceptual System Level Approach to Fault Tolerance in Cloud Computing. In Proceedings of the 2012 IEEE International Systems Conference (SysCon), Vancouver, BC, Canada, 19–22 March 2012; pp. 1–5. [CrossRef]
22. Zhao, W.; Melliar, P.M.; Mose, L.E. Fault Tolerance Middleware for Cloud Computing. In Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing, Miami, FL, USA, 5–10 July 2010; pp. 67–74. [CrossRef]
23. Jayadivya, S.K.; JayaNirmala, S.; SairaBhanus, M. Fault Tolerance Workflow Scheduling Based on Replication and Resubmission of Tasks in Cloud Computing. *Int. J. Comput. Sci. Eng. (IJCSE)* **2012**, *4*, 996.
24. Machida, F.; Andrade, E.; Seong Kim, D.; Trivedi, K.S. Candy: Component-based Availability Modeling Framework for Cloud Service Management Using SysML. In Proceedings of the 2011 30th IEEE International Symposium on Reliable Distributed Systems, Madrid, Spain, 4–7 October 2011; pp. 209–218. [CrossRef]
25. Feng, Q.; Han, J.; Gao, Y.; Meng, D. Magicube: High Reliability and Low Redundancy Storage Architecture for Cloud Computing. In Proceedings of the 2012 IEEE Seventh International Conference on Networking, Architecture, and Storage, Xiamen, China, 28–30 June 2012; pp. 89–93. [CrossRef]
26. Lin, J.; Lu, X.; Yu, L.; Zou, Y.; Zha, L. Vega Warden: A Uniform User Management System for Cloud Applications. In Proceedings of the 2010 Fifth IEEE International Conference on Networking, Architecture, and Storage, Macau, China, 15–17 July 2010; pp. 457–464. [CrossRef]
27. Das, P.; Khilar, P.M. VFT: A virtualization and fault tolerance approach for cloud computing. In Proceedings of the IEEE Conference on Information & Communication Technologies (ICT), Thuckalay, Tamil Nadu, India, 11–12 April 2013.
28. Ganesh, A.; Sandhya, M.; Shankar, S. A study on fault tolerance methods in Cloud Computing. In Proceedings of the IEEE International Advance Computing Conference (IACC), Gurgaon, India, 21–22 February 2014.
29. Chudzikiewicz, J.; Furtak, J.; Zielinski, Z. Fault-tolerant techniques for the Internet of Military Things. In Proceedings of the 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT), Milan, Italy, 14–16 December 2015; pp. 496–501.
30. Misra, S.; Gupta, A.; Krishna, P.V.; Agarwal, H. An Adaptive Learning Approach for Fault-Tolerant Routing in Internet of Things. In Proceedings of the 2012 IEEE Wireless Communications and Networking Conference (WCNC), Shanghai, China, 1–4 April 2012; pp. 815–819.
31. Li, X.; Ji, H.; Li, Y. Layered Fault Management Scheme for End-to-end Transmission in Internet of Things. In Proceedings of the 2011 6th International ICST Conference on Communications and Networking in China (CHINACOM), Harbin, China, 17–19 August 2011; pp. 1021–1025.
32. Zhou, S.; Lin, K.; Na, J.; Chuang, C.; Shih, C. Supporting Service Adaptation in Fault Tolerant Internet of Things. In Proceedings of the 2015 IEEE 8th International Conference on Service-Oriented Computing and Applications (SOCA), Rome, Italy, 19–21 October 2015; pp. 65–72.

33. Su, P.H.; Shih, C.S.; Hsu, J.Y.J.; Lin, K.J.; Wang, Y.C. Decentralized fault tolerance mechanism for intelligent IoT/M2M middleware. In Proceedings of the 2014 IEEE World Forum on Internet of Things (WF-IoT), Seoul, Korea, 6–8 March 2014; pp. 45–50.
34. Taylor, I.; Deelman, E.; Gannon, D.; Shields, M. (Eds.) *Workflows in e-Science*; Springer: Berlin/Heidelberg, Germany, 2006.
35. Ahmad, W.; Hasan, O.; Pervez, U.; Qadir, J. Reliability modeling and analysis of communication networks. *J. Netw. Comput. Appl.* **2017**, *78*, 191–215. [CrossRef]
36. Cheraghloua, M.N.; Khadem-Zadeh, A.; Haghparast, M. A Framework for Optimal Fault Tolerance Protocol Selection Using Fuzzy Logic on IoT Sensor Layer. *Int. J. Inf. Commun. Technol. Res.* **2018**, *10*, 19–32.
37. Cheraghloua, M.N.; Khadem-Zadeh, A.; Haghparast, M. New Fuzzy-Based Fault Tolerance Evaluation Framework for Cloud Computing. *J. Netw. Syst. Manag.* **2019**. [CrossRef]