A Survey of Fault Tolerance Architecture in Cloud Computing

Article *in* Journal of Network and Computer Applications · October 2015

DOI: 10.1016/j.jnca.2015.10.004

Mehdi Nazari Cheraghlou
Islamic Azad University, South Tehran Branch
10 PUBLICATIONS **196** CITATIONS

SEE PROFILE

Majid Haghparast
University of Jyväskylä
126 PUBLICATIONS **2,066** CITATIONS

SEE PROFILE

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Review

A survey of fault tolerance architecture in cloud computing

Mehdi Nazari Cheraghlou[a,*], Ahmad Khadem-Zadeh[b], Majid Haghparast[c]

[a] Department of Computer Engineering, South Tehran Branch, Islamic Azad University, Tehran, Iran
[b] Iran Telecommunication Research Center (ITRC), Tehran, Iran
[c] Department of Computer Engineering, Yadegar -e- Imam Khomeini (RAH) Branch, Islamic Azad University, Tehran, Iran

ARTICLE INFO

Article history:
Received 10 July 2015
Received in revised form
21 September 2015
Accepted 18 October 2015
Available online 28 October 2015

Keywords:
Cloud computing
Fault tolerance architecture
Proactive
Reactive

ABSTRACT

Utilizing cloud computing services has had numerous advantages such as the reduction of costs, development of efficiency, central promotion of soft wares, compatibility of various formats, unlimited storage capacity, easy access to services at any time and from any location and, most importantly, the independency of these services from the hardware. It should be mentioned that the provision of various cloud computing services is faced with problems and challenges that the fault tolerance can be mentioned as the main restrictions.

In this paper, first, methods of creating the capacity of Fault Tolerance in Cloud computing are pointed out. Subsequently, policies of the implementation of these methods are stated. Next, the offered architectures for the production of such capacity in Cloud computing is delineated and, finally, they are compared in terms of the type of policy or policies employed in the architecture and the method of fault detection and fault recovery.

Contents

* Corresponding author.
E-mail addresses: ir.m.n@ieee.org (M. Nazari Cheraghlou), zadeh@itrc.ac.ir (A. Khadem-Zadeh), haghparast@iausr.ac.ir (M. Haghparast).

http://dx.doi.org/10.1016/j.jnca.2015.10.004
1084-8045/© 2015 Elsevier Ltd. All rights reserved.

1. Introduction

The emergence of Cloud is the biggest change in the world of IT, leading to the stimulation of all individuals and all companies. Complete definition of cloud computing which is considered a standard for cloud computing is the golden definition that is presented by NIST institute (Mell and Grance, 2011). In this definition, it is expressed that cloud computing is a demand-based and easy access model under a network to a sharing set of configurable computing resources (including servers, networks, storage devices, applications, and services). The resources are provided and used quickly and they are released with minimal effort and cost.

1.1. Cloud computing models

Cloud computing can be implemented in the forms of public, private, community and hybrid cloud (Mell and Grance, 2011). Using the public cloud is possible for all but private cloud is dedicated to a collection and only members can take advantage of cloud services. Community cloud is exploited in sharing form for individuals or organizations that have similar missions and needs. Hybrid cloud is a combination of two or more different clouds that each of them should be able to provide more combined services together while preserving their separate identities.

1.2. Cloud computing challenges

Cloud is a virtual and abstract image of a large network that neither its volume nor its size and also processing and storage resources are specified and limited. Location and time are also unknown and unlimited in the cloud. This means that resources' location is hidden from the perspective of users and applications and presentation time and completion of the services and they seem to be unlimited. Desired services can be accessed from any place and at any time. These characteristics cause removing the restrictions in using systems and traditional networks in providing service to users, but they may bring some new problems, restrictions, and challenges for users and applications. On top of these problems, the fault tolerance challenge of cloud has a special status and importance. Because if we have the best clouds with the best services but they do not have fault tolerance, they are not reliable and expectation of appropriate and desired service are futile. Therefore, fault tolerance means that if a fault occurred in a cloud, it should be able to detect and identify that fault and recovery and improve it without any damage to the final output of cloud computing. This capability causes that the cloud be able to have an optimum and acceptable performance in the presence of the faults.

1.3. Fault tolerance techniques: overview

The techniques that are used to create the fault tolerance capability in cloud computing can be divided into three general categories. The first is redundancy techniques, the second is tolerance policies against and finally, the third is load balancing fault tolerance. Redundancy techniques include hardware redundancy and software redundancy and time redundancy. Hardware redundancy is a technique of structural redundancy which masks the fault using a complete series of modules. The method is in such a way that an identical number of hardware modules perform identical operations. Hardware modules' inputs are the same and the modules' outputs are aggregated and then, a majority vote is taken. Thus, the fault effect will be removed in the output. It is worth noting that voting will be done by TMR: Triple Modular Redundancy. In the case of software redundancy, our agenda is running a program with the same input but with different implementation algorithms. In other words, the data processing method is identical but it is performed using different algorithms on the same input data. Obviously, the same outputs are expected but if different outputs be achieved, the correct output can be achieved by the majority vote of outputs. In the case of time redundancy, identical hardware and software are as the constant parameters of the problem. We pursue multiple running of a program in an identical hardware. In this case, we can completely cover the fault in the output with the majority vote of the performance results of running a program.

Fault types that have been mentioned in (Kumar et al., 2015; Saikia and Devi, 2014; Amin et al., 2015) and occur in cloud computing are different based on computing resources. Among them are Network Fault, Physical Faults, Process Faults, Processor Faults and Service Expiry Fault.

Techniques for creating and increasing the fault tolerance based on the load balancing also can be performed based on hardware and software and also based on the network (Singh and Kinger, 2013). In the case of hardware, requests from a client are sent to the hosts of a cluster. In the case of software, we have a dispatcher server that is performed on all incoming requests. The disadvantage of this method is that this dispatcher server has a high potential for bottleneck. The third method which is a software solution based on the network, does not require any additional hardware. There is also no need to have a central dispatcher to be a bottleneck because, all host receives incoming packets and redundancy occurs with respect to the number of clusters. The packet filtering algorithm is very effective in handling the packets.

Among other applications of Load Balancing Mechanism in cloud computing, we can refer to Resource Management which has been discussed in (Manvi and Krishna Shyam, 2014). Resource Management in cloud computing is accompanied by advantages including scalability, QOS, reduced overhead and increased throughput. Resources are generally divided into physical and logical groups. Logical resources provide temporary control over physical resources. Also, logical resources support the development of applications and effective communication protocols. Load Balancing mechanisms, in addition to being considered as one of the methods to increase the fault tolerance of cloud computing, provide Logical Resource Management in cloud computing. Given that physical facilities in cloud computing are placed in a distributed manner, Network Resource Management develops and improves using load balancing techniques. Further, fault tolerance increases at the same time.

The last way to create fault tolerance in the cloud computing services is creating this capability based on using a series of policies. These policies are divided into two categories of proactive and Reactive which will be studied. The method in proactive is such that the fault in cloud computing is estimated and necessary precautions should be considered. But the working style in the reactive group is different and there will not be any prediction and prevention of fault because it wastes resources and increases the response time of the system especially in the case of real time cloud computing. For this reason, in this architecture, the system tries to reduce the effects of occurred fault in the system (Vacca, 2013; Bala and Chana, 2012; Ganesh et al., 2014; Egwutuoha et al., 2012; Malik and Huet, 2011; Kaur and Kinger, 2013). Two policies are used in the proactive method. The first policy is self-heading. In this policy, improvement is achieved without any replica and redundancy. With this policy, the failed versions of an application will be handled in multiple virtual machines. The second applied policy in the proactive method is preemptive migration. In this policy, continuous monitoring and analysis are done on the output according to the feedback from the output to the input so that in the case of any bug in the produced output, the required processing operations be repeated to obtain the final output any defects and failures. The first policy which is used in the reactive method is checkpoint/restart. When an application starts to run some checkpoints are made in different parts. When a fault occurs, the application restarts at the last checkpoint onwards. The second policy is replication. Production of other data and objects is a policy which is pursued in it. The third type of policy which is used in reactive method, when a fault occurs the work will be sent to the other similar machine. The policy also called job migration.

As an example of cloud computing applications, we can refer to Cloud-Based Mobile Application which has been discussed in (Ahmed et al., 2015). In this reference, Mobile Applications use two sets of components including Transferable Components and non-Transferable Components. Transferable components are sensitive to memory and computer, but this is not the case with other components such as Hardware Access and Security Related Tasks. Increasing the fault tolerance of the first set of components is provided by an increase in redundancy while regarding the second set of components, the techniques such as Proactive and Reactive should be used, depending on their application and function.

In this paper, we will first describe the proposed architectures using the policies of proactive and reactive methods. Then, we will compare them. At the end of the paper, we will describe the results of the comparison and evaluation.

2. Fault tolerance architecture models

In this section, all the existing architectures that provide fault tolerance in cloud computing are described. It should be noted that these architectures are divided into Proactive and Reactive groups, according to the type of policies from which they have benefited. Moreover, in the structure of some architectures, more than one technique has been used. This will strengthen the architecture regarding Fault Detection and Fault Recovery.

2.1. Proactive architecture

2.1.1. Map-reduce architecture

The structure that uses the policies of proactive is the famous architecture of Map-Reduce. Map-Reduce divides works into smaller pieces. These small parts process the work on different machines in parallel form. The final output is achieved by combining the partial outputs together. Obviously, if running the small parts in a machine gets into trouble, it sends it to another machine to produce output without fault and defect because this architecture takes advantage of feedback in its structure. This architecture is used to process big data in cloud environments. Because the mentioned architecture provides the possibility to store of large volumes of data and parallel processing of them in the form of some clusters of machines (Hashem et al., 2015; Inukollu et al., 2014; Abadi, 2009; Purcell, 2014; Ahuja and Moore, 2013).

It is necessary to mention two points before addressing the fault tolerance architectures that use the policies of reactive. First, some parameters are expressed to measure the capability of fault tolerance in cloud computing (Ganesh et al., 2014) which are RPO and RTO. RPO is the amount of data which is disappeared during a fault. RTO is determining the minimum down time for recovery. Various models of fault tolerance are stated and compared in (Ganesh et al., 2014). These evaluations show that the models which are used in combination form of proactive and reactive provide a higher level of reliability but, they increase response time. Instead, they significantly increase the network performance. Second point is the comparison between reactive and proactive methods.

In proactive method, the occurrence of a fault is estimated. Then, its occurrence it prevented. This method is more efficient than reactive method. But we should always keep in mind that some of the predictions may not be accurate.

In reactive method, after the occurrence of a fault, it is tried to reduce its impact and recovery. Reactive methods are most used because they have higher reliability. But, since availability techniques significantly reduce the system reactive, thus these techniques are not appropriate for the systems that need more availability (Ganesh et al., 2014).

2.1.2. FT-Cloud architecture

Cloud Computing Applications are usually provided in large and wide scales and with their own complexity. However, unfortunately their reliability is still far from the ideal state. In (Zheng et al., 2010), a component ranking based framework has been introduced which is called FT-Cloud. This framework comprises two phases of operating algorithms including ranking and fault tolerance. FT-Cloud enables Cloud Computing Applications to confront the faults. This architecture provides fault tolerance for cloud computing in the face of Value and Crash faults. The structure of this architecture has been illustrated in Fig. 1.

2.2. Reactive architecture

2.2.1. Haproxy architecture

HAProxy architecture is the first architecture of reactive architecture that will be surveyed. In this architecture job migration and replication policies are used. Operation of this architecture is in such a way that there are two server machines and one Haproxy in it. One of the server machines has the redundancy role for the other server machine. In other words, if the first server failed, the second server will backup the operation of the entire system without any interruption and conversely, if the second server failed, the first server will follow the overall performance of the entire system without any interruption. Haproxy server is responsible for managing these tasks. Always one of the servers in the system will be activated and there is a moment in which both servers are disabled. It is even possible that in some cases the both servers be activated in the system, but there is no possibility that they both be disabled. Server one is as the replica of server two and server two is as the replica of server one. At the time of the failure of a server, job migration technique is used and continuing the work is transferred to another server (Bala and Chana, 2012).

2.2.2. BFT-Cloud architecture

The second architecture is BFT-Cloud structure which is proposed in (Zhang et al., 2011). Replication policy is used in this structure. This architecture can be classified as reactive architectures. When a request comes in a cloud computing system, the request must be implemented on different nodes. One of the nodes is selected as the primary node and the other are selected as backup nodes. In the request execution, all applications are executed locally on the machine. If the results of running on the backup nodes and the primary node be the same, the output is correct and properly responds the requesting module. If one of the backup versions or primary versions has failure, they provide a different answer compared to the other outputs. In this case, that node is known as the faulty node and in the updating stage, recovery operations should be performed. If a faulty node be the primary node, it will be changed with the new primary version in the primary updating stage and if the faulty node be one of the backup nodes, it should be replaced with the appropriate nodes in replica updating stage. The operational structure of this architecture is shown in Fig. 2.

In (Zhang et al., 2011; Lim et al., 2013; Tanenbaum and Steen, 2001), Byzantine fault tolerance architecture is introduced. This architecture is also one of the reactive methods. Replication policy is used in this architecture which is considered as the fundamental architecture of the BFT-Cloud. The method in this architecture includes three different phases. In the first phase, the same input is distributed between nodes and the output should be sent to all the reviewed node in the set. The reviewed set includes the primary and backup nodes. Obviously, we expect that the same input in different nodes produces the same output because the process is same on all nodes. Different output will be produced only in the case of a faulty node. The first stage is continued by broadcasting the output of each node to the neighboring nodes and the other members of the reviewed nodes. At the end of the first phase, each node should form the output vector of the neighbors. In the second phase, each of the nodes broadcasts the made vector in the previous step between neighboring nodes. This output vector considers the neighborhood of one node. In the third stage which the last stage, the comparison operation is done and the conclusion about the safety of faulty nodes is performed. The nodes in this stage take the majority vote between the received borders from the neighbors and with this operation it concludes that whether the node is healthy or faulty. If a node be healthy its output in the vector of all neighboring nodes will be the same and they cannot be voted. Therefore all neighbors and members of the reviewed set conclude that the node is faulty. The fault type was arbitrary and produces a different value in each time of running.

In Byzantine architecture, the fault detection capability is approximately 33% because $3k+1$ node in the network is needed for detection of k faulty node. In other words, maximum k faulty node in the network can be identified in $3k+1$ node. For example, if we have 100 nodes in our cloud infrastructure, this architecture will identify up to 33 faulty nodes in the cloud.

2.2.3. *Gossip architecture*

Gossip architecture is introduced to improve the performance of the Byzantine architecture (Lim et al., 2013). This structure is a part of reactive method which exploits replication policy to detect a fault and creating fault tolerance capability in the cloud computing system. In this architecture, each of the cloud machines has a decision vector. At every stage of the network operation, each node selects a neighbor node and the two nodes are required to update their decisions vector together. The size of the decision vector depends on the number of neighboring nodes. Since, the

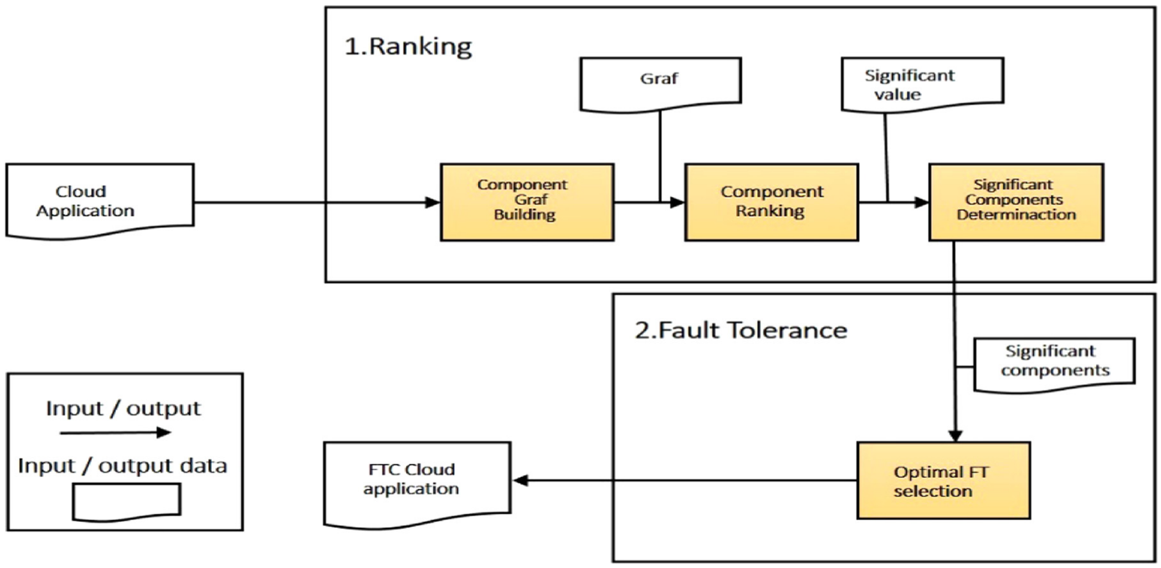

Fig. 1. The FT_Cloud fault tolerance architecture introduced in (Zheng et al., 2010).

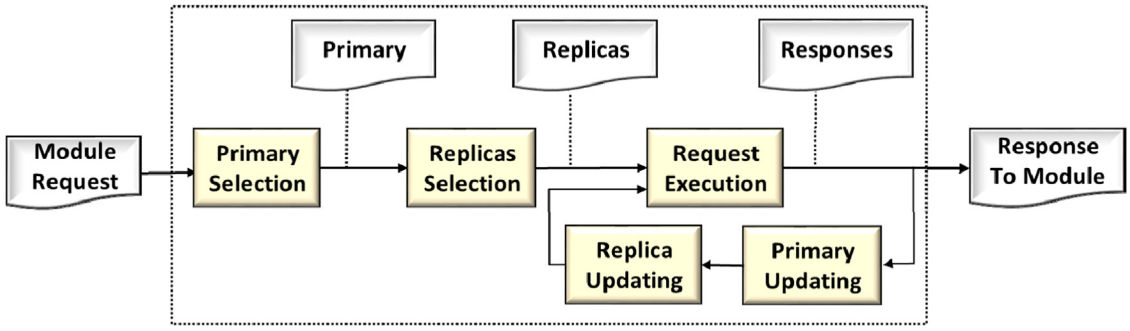

Fig. 2. The BFT-Cloud fault tolerance architecture described in (Zhang et al., 2011).

faulty node produces different outputs in each run, thus, its values have conflict in each stage of updating in decision vector and all the nodes realize this difference in values. Nodes will consider the producer machines with different output values as a faulty node.

An example of the stages of function and operations of this architecture has been provided in Figs. 3. and 4.

Obviously, this architecture improves the reliability of fault detection to 50%. Because in the Gossip architecture, if $2k+1$ node be available in the network, k faulty node can be identified. For example, if we have 200 machines under the cloud computing infrastructure, almost 100 faulty machines can be identified by using the Gossip architecture.

2.2.4. MPI (Message Passing Interface) architecture

The next structure is MPI architecture which is introduced in (Kaur and Kinger, 2013). This architecture is also one of the reactive methods and uses checkpoint/restart and job migration techniques. The structure of this architecture is shown in Fig. 5.

This architecture is a standard for parallel programming. Implementation of high-level protocols makes possible to use the relationships between processes and process management. It is also possible to create a new implementation by rewriting only low-level functions. MPI has two layers. The top layer is independent of the infrastructure communication and the bottom layer is called SSI and specifies that whether the backup of the

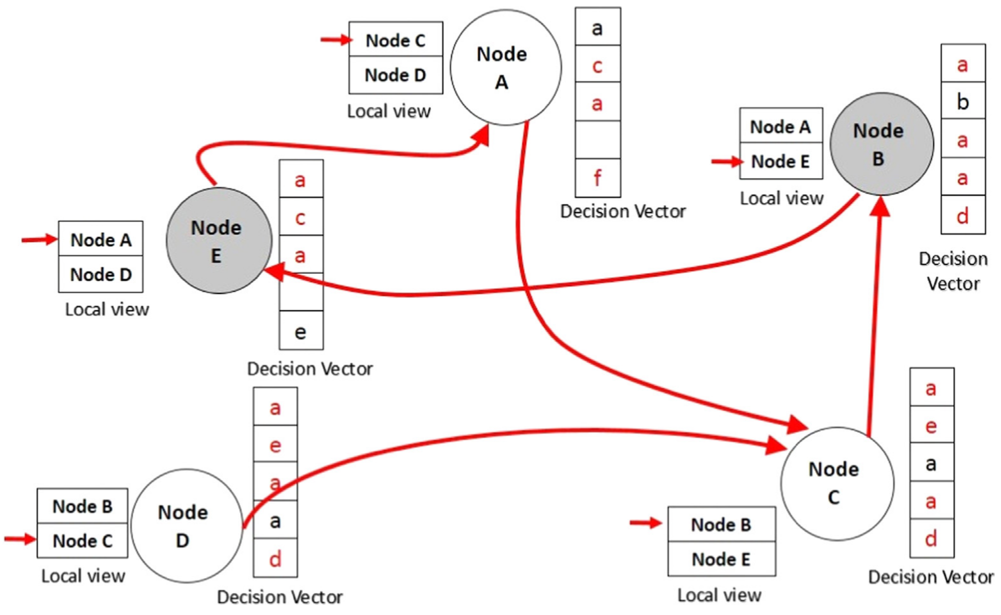

Fig. 3. Stage of node calculation in Gossip architecture (Lim et al., 2013).

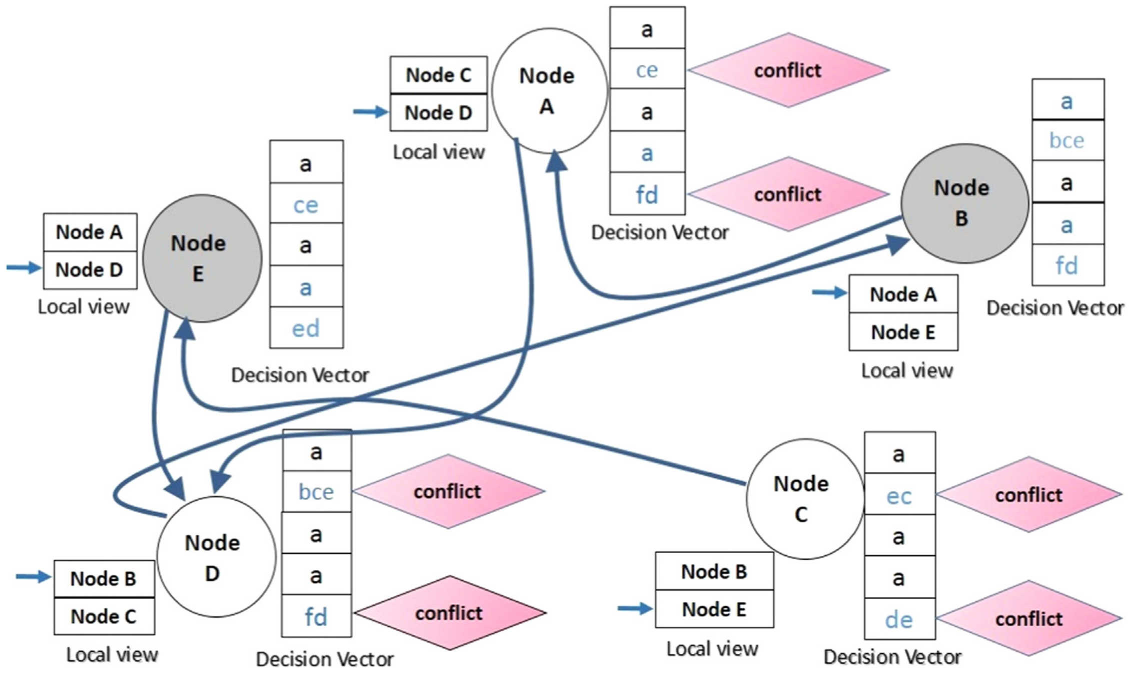

Fig. 4. Stage of node results in Gossip architecture (Lim et al., 2013).

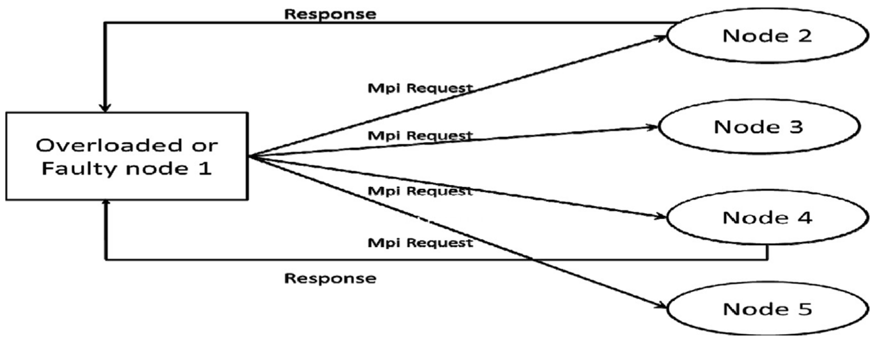

Fig. 5. The MPI fault tolerance architecture introduced in (Kaur and Kinger, 2013).

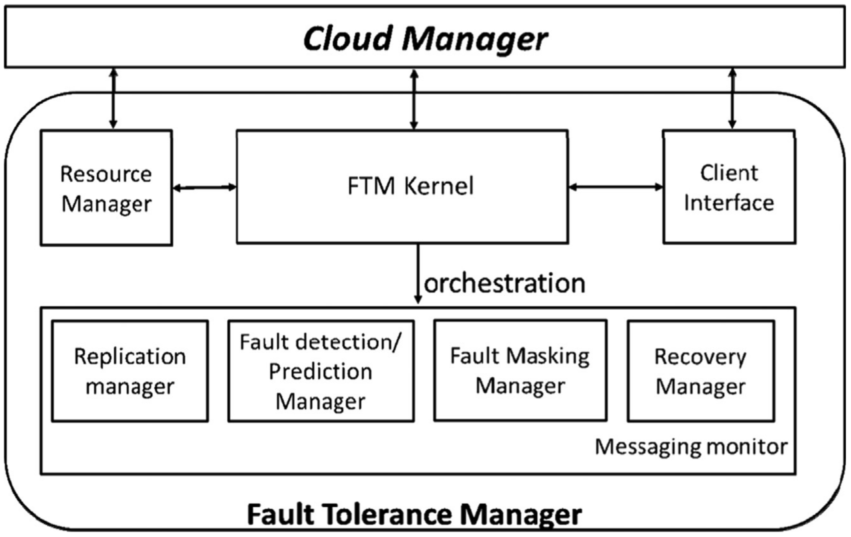

Fig. 6. The FTM fault tolerance architecture proposed in (Kaur and Kinger, 2013).

checkpoint is needed or not. In this architecture, the faulty node sends MPI Request to the other nodes. The number of nodes that have received the faulty node request sends the positive response of the request to the faulty node and the faulty node migrates the program running from the last checkpoint onward to the healthy node using checkpoint/restart and job migration techniques.

2.2.5. FTM architecture

The other architecture that will be investigated in the field of nodes' fault tolerance in cloud computing is FTM which stands for fault tolerance manager. The mentioned architecture which is introduced in (Kaur and Kinger, 2013) is one of the reactive techniques and uses every three policies related to this method i.e. replication, checkpoint/restart, and job migration. The structure of this architecture is shown in Fig. 6. In the following, we will refer the task of each module in this architecture.

The first module of this architecture is FTM kernel that is the manager and decide maker about the type of fault recovery. The next module is messaging monitor. This module is managed by the FTM Manager and has two tasks. First, it monitors the messages among the replica and second, it monitors the exchange messages among the different modules of this structure. This module has four sub modules in itself. One of them is replication manager module that its duty is to produce and replica from the desired node. The next module is called fault detection/prediction manager and its duty is to manage and predict fault. According to results, FTM kernel module will make decisions about the recovery method. Nodes' recovery can be in the form of fault effect coverage in output. This work will be done by fault masking manager module. Or fault recovery policy can be pursued and this is the responsibility of recovery management module. There are two other communication modules which are responsible for linking the fault tolerance management layer and cloud management. The duty of client interface module is a communication between the end user and FTM. Resource manager module also provides a database of login information and resources under the authority of virtual machines in the cloud.

In (Jhawar et al., 2012), another similar architecture with the same name FTM has been suggested. A view of this architecture is shown in Fig. (7). As can be observed, fault detection task in this architecture has been assigned to Fault Detector component and recovery operations begin after fault detection. In this phase, two policies called Checkpoint/Restart and Replication have been exploited. These policies are managed respectively by Checkpoint Manager and Replication Manager.

2.2.6. Magi-Cube architecture

In (Feng et al., 2012), Magi-Cube architecture has been introduced that has suggested the increased reliability for storage architecture in cloud computing, using low redundancy. In this architecture, only one replica version has been used in HDFS structure and one algorithm has been exploited to create fault tolerance. In cloud storage systems, three important parameters that conflict with each other include high reliability, high performance and low cost. On one hand, current cloud storage systems which are widely used benefit from the systems such as GFS, Hadoop, HDFS and S3. These systems provide two factors of high reliability and high efficiency, but they tolerate high overhead cost. In Magi-Cube architecture, due to the fact that unlike the aforementioned systems, Multi Replication Policy has not been used, low redundancy and increased fault tolerance have been obtained. The structure of this architecture has been presented in Fig. 8.

2.2.7. LLFT architecture

The architecture introduced in (Zhao et al., 2010) which has been called LLFT provides fault tolerance with low latency. This architecture has provided F.T capacity for developing Distributed Application or Data Centers. LLFT has benefited from Leader/Follower Replication approach. There is minimum end to end latency and replica transparency is very robust. When a fault occurs, LLFT is reconfigured and recovery mechanisms make sure that a backup version exists. Then, the state is transferred from the existing replica to new replica. One of the most important tasks of LLFT is to robustly maintain the compatibility of different duplicate copies. On the other hand, the operations should be synchronized so that minimum end to end latency will be achieved (Fig. 9).

2.2.8. Vega Warden architecture

One of the most important issues in cloud computing is virtualization. In a cloud computing environment, due to cluster virtualization and infrastructure sharing, two management problems arise for the users, including usability and security. In (Xiaoyi Lu et al., 2010), a unified user management system has been proposed to solve these problems. Since Vega Warden architecture has adopted similar mechanisms and uses decentralized and distributed rules, it also possesses fault tolerance in its structure due to the existence of redundancy. Vega Warden supports both Virtual Infrastructure provider and Application Service Provider models. Additionally, when more than one application service is installed, it is expected that Load Balancing mechanisms are automatically used. This increases the network fault tolerance. Different parts of this architecture have been provided in Fig. 10.

2.2.9. FTWS architecture

FTWS architecture that is shown in (Jayadivya et al.) has provided fault tolerance using replication and restart techniques while considering the priority of tasks. This architecture schedules the workflows with a deadline in the presence of faults. It should be noted that repeating things depends on an exploratory and metric measurement that should be determined in the form of trade off from among the factors of the techniques used in

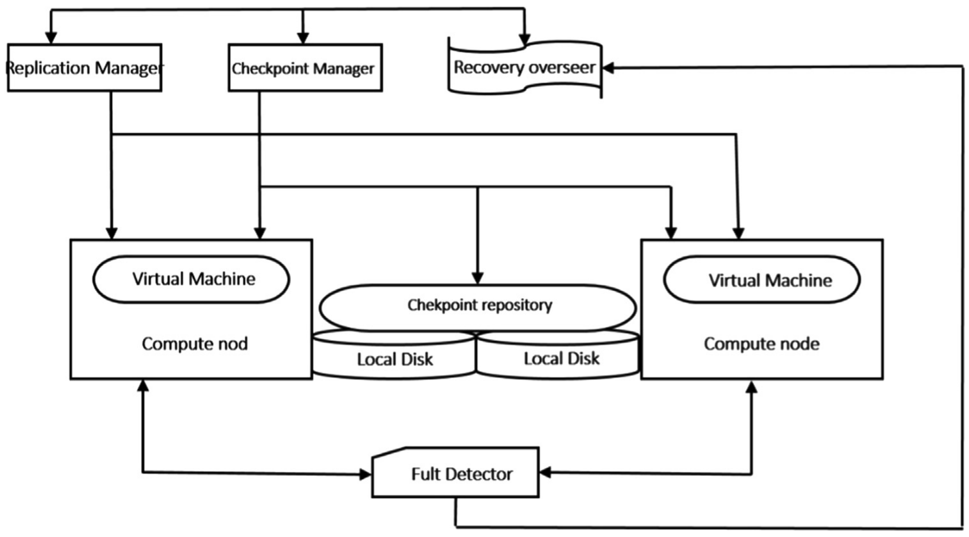

Fig. 7. The FTM-2 fault tolerance architecture proposed in (Jhawar et al., 2012).

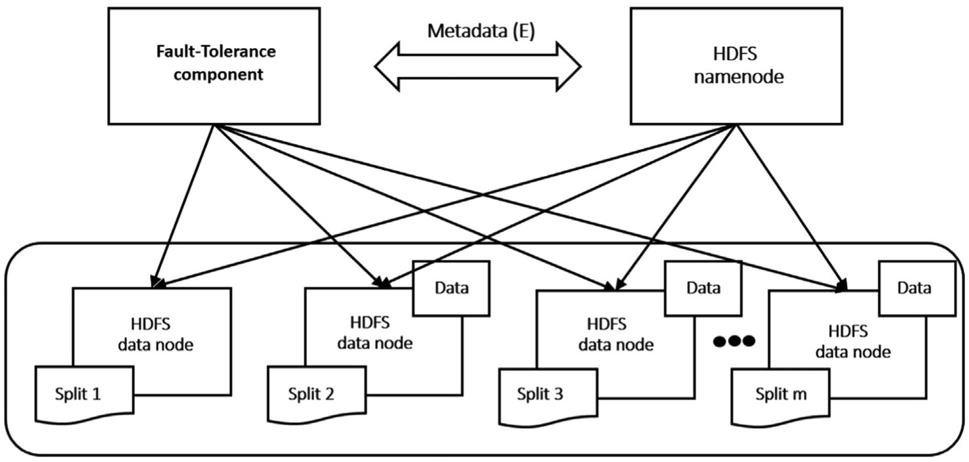

Fig. 8. The Magi-Cube fault tolerance architecture proposed in (Feng et al., 2012).

replication and restart. View of this architecture has been provided in Fig. 11.

2.2.10. *Candy architecture*

In (Machida et al., 2011), a component-based availability modeling framework has been introduced, which is a useful and comprehensive semi-automatic model and has been identified and described by system model language. This model indicates that cloud services and Cloud Computing Provider must be ensured regarding accessibility which is one of the most striking features of cloud services. Components and structure of this architecture have been illustrated in Fig. 12.

2.2.11. *AFTRC architecture*

Many of the applications that are hosted by cloud systems are RTHPC: Real Time High Performance Cloud Computing. Therefore, on the one hand, these systems are extremely sensitive to time and system response in them is considered at a specific time. So that if the system responds after this time period, its output will not have value. On the other hand, fault tolerance techniques increase the system's response time and delay the output production.

In this section, we will investigate two architectures of creating the capability of fault tolerance in real time cloud computing systems. The first architecture is AFTRC. The name of this architecture is an acronym for Application Fault Tolerance in Real Time Cloud Computing. This structure is also a part of reactive methods and uses all the policies (Sudha Lakshmi, 2013).

By input buffer from input work, different replica are produced to run on the virtual machines. The next stage is done by applying job migration policy, transfer and migration of works to different virtual machines in the system. Works are done according to different real time algorithms in the different virtual nodes. There is a module with the name of AT: Acceptance Test exactly after the output of each algorithm in all virtual machines that its duty is surveying the output validity of each VMs (VM: Virtual Machine). In

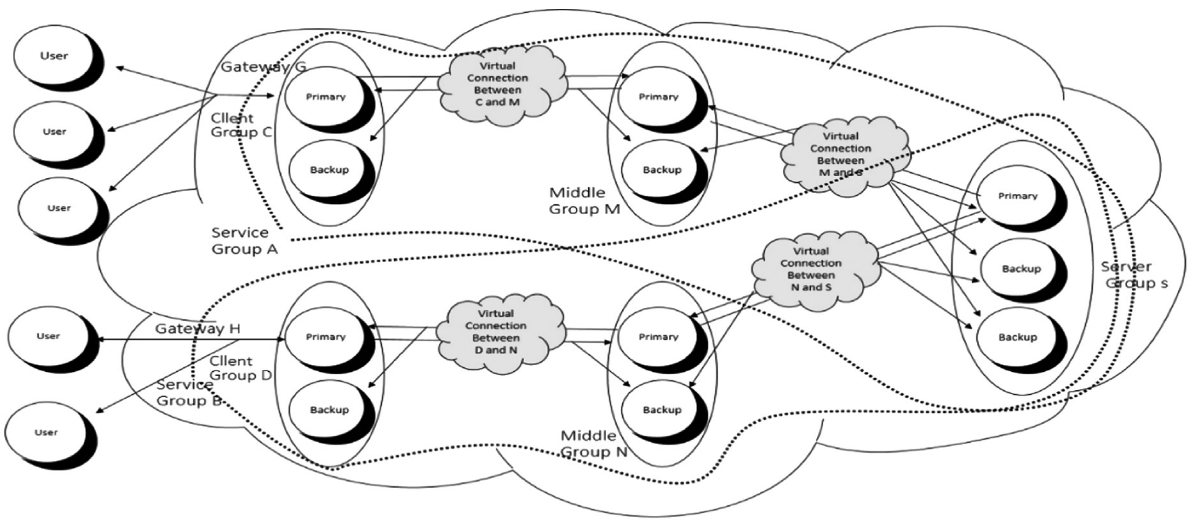

Fig. 9. The LLFT fault tolerance architecture proposed in (Zhao et al., 2010).

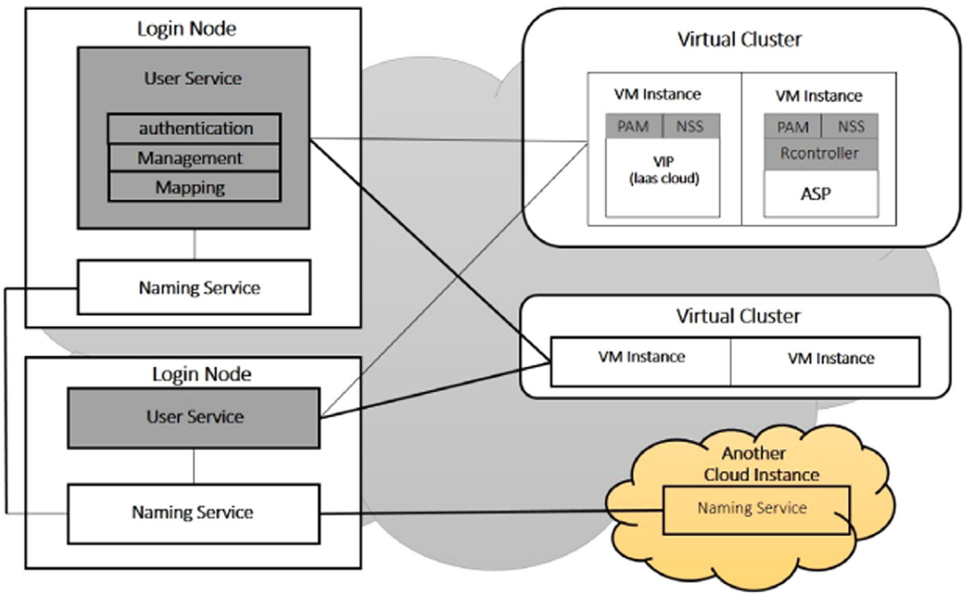

Fig. 10. The LLFT fault tolerance architecture proposed in (Xiaoyi Lu et al., 2010).

the case of the validity of the output, it will pass it to the TC module and in the case of invalidity of the output, an exception signal will be sent to TC. In the next stage, TC module (TC: Time Checker) will survey the timing validity of the valid produced output in each of VMs and it will report the results to the module. RA module (RA: Reliability assessor) will assess the reliability of VMs' output based on the two previous parameters i.e. TC and AC. The second task of the RA module is that it defines a min and max limit for reliability and if the reliability of a virtual machine be less than the minimum, it will remove that node. In continue, DM module (DM: Decision Mechanism) which is the decision maker of the final output selection mechanism based on the reliability of a computing cycle, will produce the final output based on this mechanism. RC module (RC: Recovery Cache) is a store for maintaining check points so that if needed recovery operation be performed based on the last checkpoint. It is worth noting that the three modules of TC and RA and DM in this architecture is called adjudicator node. The structure of this architecture is shown in Fig. 13.

2.2.12. PLR architecture

The last architecture that we will investigate in this paper is named PLR architecture which is introduced in (Egwutuoha et al., 2012). In this structure, all the three policies of the reactive method are used. This architecture is for real time high performance cloud systems. PLR stands for Process Level Redundant and the reason for this name is creating its redundancy in the process. The remarkable thing in this architecture is using MPI architecture within it and it is embedded in an interesting form.

This architecture uses five modules in itself. The first module is a fault predictor that its duty is predicting fault. The next module is PLR Controller Daemon which is responsible for MPI monitoring and application. Replica redundancy management is one of the other duties of this module. The third module is Fault Tolerance Policy which is responsible for selecting the type of policy in dealing with fault in this architecture. The fourth module is Fault Tolerance Dream on protocol which initially sends a monitoring-based message to PLR. Then, live migration is done and check points are initialized according to checkpoint library which is called BLCR. In the end, Check point/restart module stores check points on the neighboring nodes to eliminate failure points and release the resources used to check points. This release of resources to checkpoint is also called incomplete clean.

As mentioned above, the policies for creating and increasing the fault tolerance capability increase the response time of the

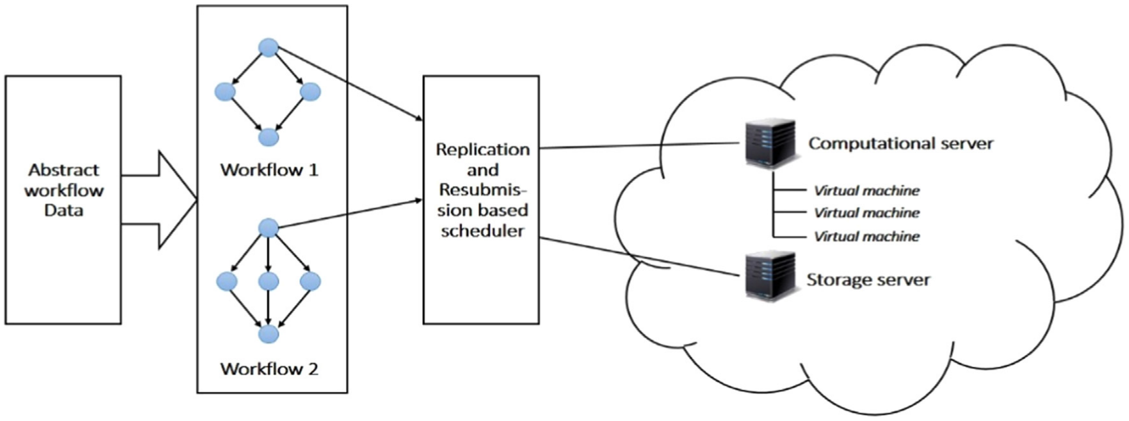

Fig. 11. The FTWS fault tolerance architecture proposed in (Jayadivya et al.).

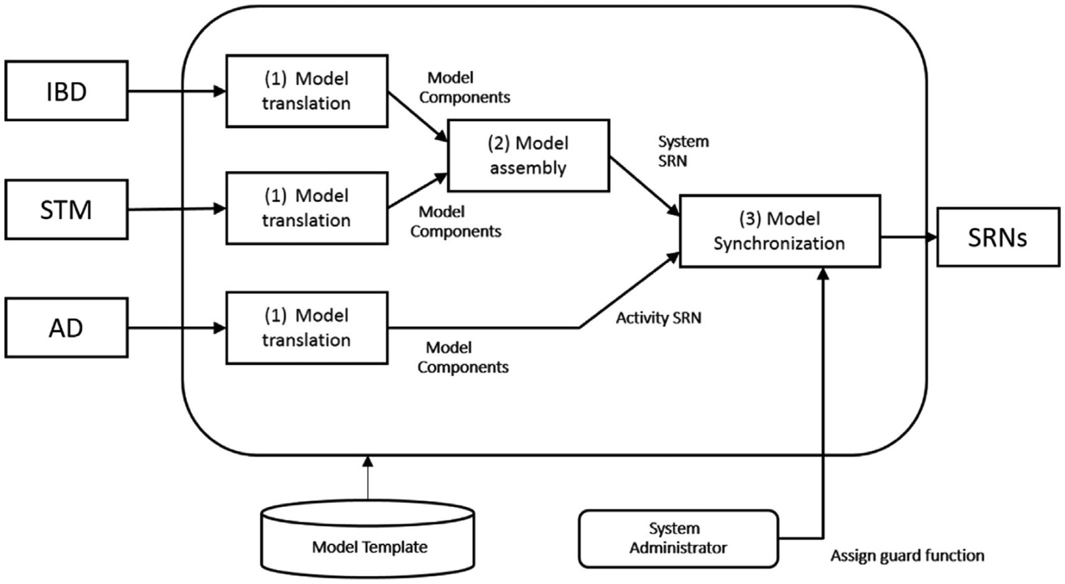

Fig. 12. The FTWS fault tolerance architecture proposed in (Machida et al., 2011).

system and output production. In PLR architecture, this issue is interpreted in such a way that increasing the fault tolerance capability increases wall clock time in RTHPC systems. However, providers of this architecture claim that PLR architecture reduces wall clock time and shows a better performance compared to the other architectures in term of response delay. The structure of this architecture is shown in Fig. 14.

3. Discussion and evaluation

Map-reduce architecture has used proactive techniques. This architecture is useful for Big Data processing and indeed, is considered as an engine for Hadoop structure and has been successfully implemented in the companies such as Google, Yahoo, IBM and Amazon.

FT-Cloud architecture, like the previous model, has used proactive techniques. The difference between FT-Cloud and Map-Reduce is that FT-Cloud benefits only from self-healing policy whereas Map-Reduce uses both policies of the proactive method. Another different and obvious characteristic of FT-Cloud is that it is a component ranking based architecture.

From among the category of reactive architectures, Haproxy model was first introduced that uninterruptedness is one of the important features of this architecture. The strength of this architecture can be seen in the simultaneous use of Job Migration and Replication policies.

Byzantine and Gossip architectures are almost similar, but Gossip model is a more efficient model compared to Byzantine model in terms of efficiency in output. Moreover, Gossip architecture, due to applying two methods out of the three methods of fault detection, enjoys higher fault detection power.

MPI architecture is specific to parallel programming systems. Fault detection in this model as in most of the other models is done outside the node and by another module. FTM and FTM-2 architectures, unlike having the same name, have significant differences. FTM model is a full-policy architecture that uses all three policies of the reactive method. But FTM-2 architecture has not benefited from Job Migration policy in its structure. In fault recovery phase, FTM architecture also has the ability to mask the faults while FTM-2 model lacks this ability. Another significant difference is the separateness of fault detection module in this architecture.

The main characteristic of LLFT architecture is its low latency. However, this model may be considered among the weakest architectural models of fault tolerance in cloud computing since it has used only the replication policy.

Magi-Cube and Vega Warden architectures are similar in terms of the type and number of the policies applied in them. Regarding fault detection, these two architectures act in a similar manner. But in terms of fault recovery, Magi-Cube architecture recovers (repairs) the entire system while Vega Warden model merely repairs (recovers) the node. The main feature of Magi-Cube architectural model is its low redundancy whereas the main

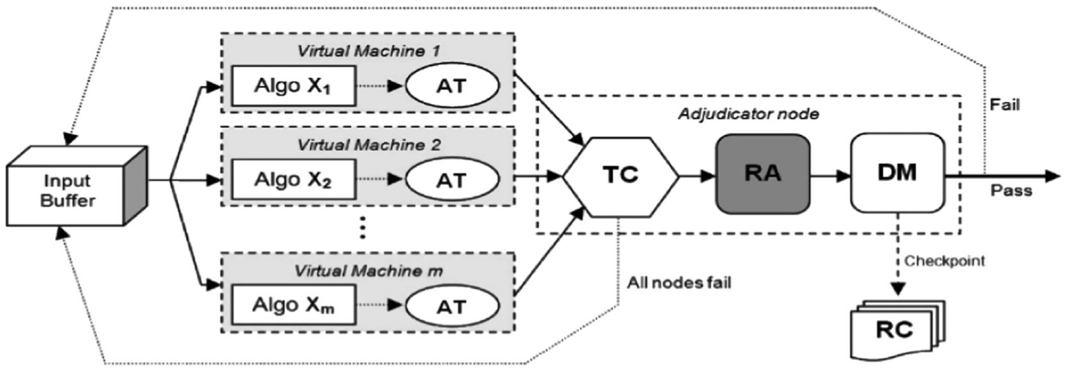

Fig. 13. The AFTRC fault tolerance architecture introduced in (Sudha Lakshmi, 2013).

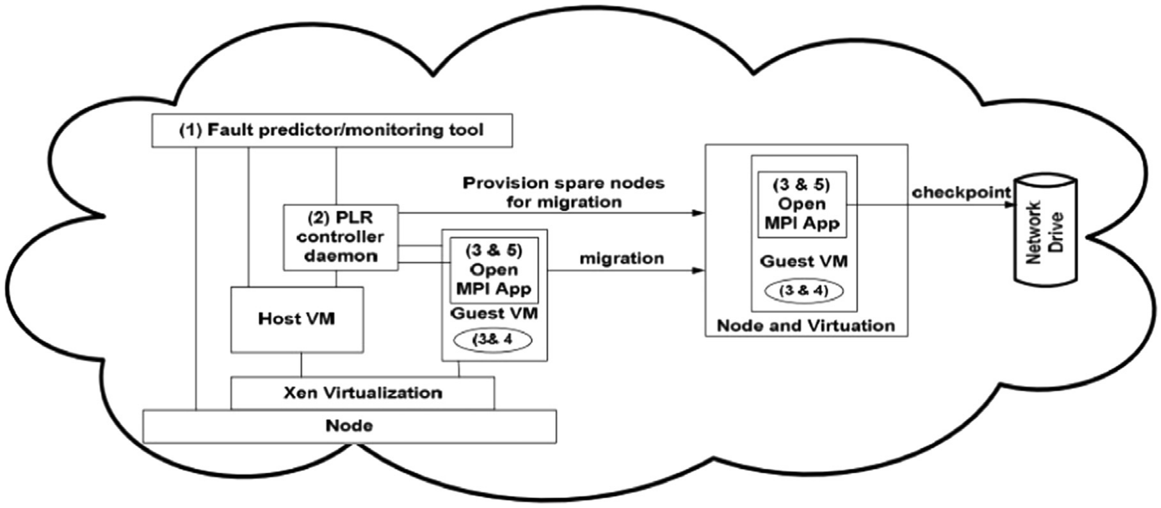

Fig. 14. The PLR fault tolerance architecture proposed in (Egwutuoha et al., 2012).

Table 1
Comparison of the applied policies in each of the architectures.

Fault tolerance architectures for cloud computing	**Techniques & Policies**					**Fault Detection**			**Fault Recovery**			**Main features/usage**
	Proactive		Reactive									
	Self-Healing	Preemptive migration	Checkpoint/ Restart	Replication	Job migration	Self-detection	Other detection	Group detection	Fault mask	Node recovery	System recovery	
Map-Reduce	✓	✓	✗	✗	✗	✗	✓	✗	✗	✗	✓	**Big Data Processing**
Haproxy	✗	✗	✗	✓	✓	✗	✓	✗	✗	✗	✓	**uninterrupted**
BFT-Cloud	✗	✗	✗	✓	✗	✗	✗	✓	✗	✗	✓	**Arbitrary Fault Detection**
Gossip	✗	✗	✗	✓	✗	✓	✗	✓	✗	✗	✓	**Optimize than Byzantine**
MPI	✗	✗	✓	✗	✓	✗	✓	✗	✗	✗	✓	**For Parallel Programing**
FTM	✗	✗	✓	✓	✓	✓	✓	✗	✓	✗	✓	**Full policy**
FTM-2	✗	✗	✓	✓	✗	✗	✓	✗	✗	✗	✓	**Separated Fault Detector**
LLFT	✗	✗	✗	✓	✗	✗	✗	✓	✗	✓	✗	**Low Latency**
FTWS	✗	✗	✓	✓	✗	✗	✓	✗	✗	✗	✓	**Workflow**
Candy	✗	✗	✗	✓	✗	✗	✓	✗	✗	✗	✓	**Component-Based**
FT-Cloud	✓	✗	✗	✗	✗	✗	✓	✗	✗	✗	✓	**Component Ranking Based**
Magi-Cube	✗	✗	✗	✓	✓	✗	✓	✗	✗	✗	✓	**Low Redundancy**
Vega Warden	✗	✗	✗	✓	✓	✗	✓	✗	✗	✓	✗	**Virtualization**
AFTRC	✗	✗	✓	✓	✓	✗	✓	✗	✓	✗	✓	**Real Time HPC**
PLR	✗	✗	✓	✓	✓	✓	✗	✗	✗	✗	✓	**Real Time HPC**

application of Vega Warden model is in the systems that have used virtualization.

AFTRC and PLR architectures are applied in Real Time HPC systems. Both of these models have fully benefited from all the techniques of reactive method. But they are different in fault detection phase. AFTRC has used other detection method while PLR has applied self-detection method. In fault recovery phase, they both act as system recovery. Further, AFTRC architectural model has the ability of fault masking. Summary of the status and policies used in each of the studied architectures has been provided in Table 1.

4. Conclusion

In this article, techniques for creating fault tolerance in cloud computing were mentioned. Afterwards, architectures proposed in this area were introduced. The focus was on the type and number of policies that had been exploited in the mentioned architectures. Considering the main feature, performance and application of each architecture, it was observed that the discussed models benefited from fault tolerance techniques in different ways. The investigators are recommended to provide new architectures with higher fault tolerance in future. Also, they can offer new and optimum models with regard to the type of application that is expected from cloud computing services.

References

Mell P, Grance T. NIST Special Publication 800–145. The NIST Definition of Cloud Computing, Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg, MD 20899-8930, U.S. Department of Commerce; September 2011.

Kumar S, Rana DS, Dimri SC. Fault tolerance and load balancing algorithm in cloud computing: a Survey. IJARCCE Int J Adv Res Comput Commun Eng 2015;4(7).

Saikia LP, Devi YL. Fault tolerance techniques and algorithms in cloud computing. Int J Comput Sci Commun Netw 2014;4(1):01–8.

Amin Z, Sethi N, Singh H. Review on fault tolerance techniques in cloud computing. Int J Comput Appl 2015;116:11–7.

Singh G, Kinger S. A survey on fault tolerance techniques and methods in cloud computing. IJERT 2013;2(6).

Manvi SS, Krishna Shyam G. Resource management for Infrastructure as a Service (IaaS) in cloud computing: a survey. J Netw Comput Appl 2014;41(0):424–40.

Vacca JR. Cyber security and IT infrastructure protection. Syngress; 2013.

Bala A, Chana I. Fault tolerance- challenges, techniques and implementation in cloud computing. ISSN (Online): 16940814. IJCSI Int J Comput Sci 2012;9(1) www.IJCSI.org..

Ganesh A, Sandhya M, Shankar S. A Study on Fault Tolerance methods in cloud computing. IEEE Int Adv Comput Conf (IACC) 2014:844–9. http://dx.doi.org/10.1109/IAdCC.2014.6779432.

Egwutuoha IP, Chen S, Levy D, Selic B. A fault tolerance framework for high performance computing in cloud, Cluster, Cloud and Grid Computing (CCGrid). In: Proceedings of the 12th IEEE/ACM international symposium; DOI:10.1109/CCGrid.2012.80, 13–16 May 2012, pp. 709–710.

Malik S, Huet F. Adaptive fault tolerance in real time cloud computing, Services (SERVICES), IEEE World Congress pp. 280–287 :4–9 July 2011.

Kaur J, Kinger S. Analysis of different techniques used for fault tolerance. IJCSIT: Int J Comput Technol 2013;4(2):737–41.

Ahmed E, Gani A, Sookhak M, Hafizah Ab Hamid S, Xia F. Application optimization in mobile cloud computing: motivation, taxonomies, and open challenges, J Netw Comput Appl 2015;52(0):52–68.

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 Elsivier 2015;47:98–115.

Inukollu V, Arsi S, Ravuri S. Security issues associated with big data in cloud computing. Int J Netw Secur Appl (IJNSA) 2014;6(3).

Abadi DJ. Data management in the cloud: limitations and opportunities. Bull IEEE Comput Soc Tech Comm Data Eng 2009;32(1):3–12.

Purcell BM. Big data using cloud computing. J Technol Res 2014;5(1):1–8.

Ahuja SP, Moore B. State of big data analysis in the cloud. Netw Commun Technol 2013;2(1).

Zheng Z, Zhou TC, Lyu MR, King I. FT-Cloud: a component ranking framework for fault-tolerant cloud applications. In: Proceedings of the 2010 IEEE 21st international symposium on software reliability engineering; 1–4 Nov 2010; pp. 398–407; doi: 10.1109/ISSRE.2010.28.

Zhang Y, Zheng Z., Lyu MR. BFTCloud: a byzantine fault tolerance framework for voluntary-resource cloud computing cloud computing (CLOUD). In: Proceedings of the IEEE international conference on 4–9 July (2011); pp. 444–451.

Lim J, Suh T, Gil J, Yu H. Information systems frontiers. Springer; 2013.

Tanenbaum AS, Steen MV. Distributed systems, Pearson Prentice Hall∼ is a trademark of Pearson Education, Inc, 2001.

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), DOI: 10.1109/SysCon.2012.6189503: 19–22 March 2012, pp. 1–5.

Feng Q, Han J, Gao Y, Meng D. Magi-cube: high reliability and low redundancy storage architecture for cloud computing. In: Proceedings of the 2012 IEEE seventh international conference on networking, architecture, and storage, doi: 10.1109/NAS.2012.15: 28–30 June 2012; pp. 89–93.

Zhao W, Melliar PM, Mose, LE. Fault tolerance middleware for cloud computing. In: Proceedings of the 2010 IEEE 3rd international conference on cloud computing. 10.1109/CLOUD.2010.26; 5–10 July 2010; pp. 67–74.

Xiaoyi Lu, J, Yu L, Zou Y, and 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. doi: 10.1109/NAS.2010.34: 15–17 July 2010; pp. 457–464.

Jayadivya SK, JayaNirmala S, SairaBhanus M. Fault tolerance workflow scheduling based on replication and resubmission of tasks in cloud computing. In: Proceedings of the international journal on computer science and engineering (IJCSE).

Machida F, Andrade E, Seong Kim D, Trivedi KS. Candy: component-based availability modeling framework for cloud service management using Sys-ML. In: Proceedings of the 2011 30th IEEE international symposium on reliable distributed systems. doi: 10.1109/SRDS.2011.33.4-7 Oct: 2011; pp. 209–218.

Sudha Lakshmi S. Fault tolerance in cloud computing. ACICE 2013;04(Special Issue 01).