JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 34, 205-223 (2018)
DOI: 10.6688/JISE.2018.34.1.13

Reaching Optimal Interactive Consistency in a Fallible Cloud Computing Environment

SHU-CHING WANG[1], SHUN-SHENG WANG[2] AND KUO-QIN YAN[3]
[1]*Department of Information Management*
[2]*Department of Industrial Engineering and Management*
[3]*Department of Business Administration*
Chaoyang University of Technology
Taichung, 413 Taiwan
E-mail: {scwang; sswang; kqyan}@cyut.edu.tw

Nowadays, network bandwidth and hardware technology are developing rapidly and resulting in the vigorous development of the Internet. However, cloud computing, an Internet-based development in which dynamically scalable and often virtualized resources are provided as a service over the Internet has become a significant issue. In a cloud-computing environment, the fault-tolerance is an important research topic. To cope with the influence from faulty components, reaching a common consistency at the presence of faults before performing some special tasks is essential. However, the previous protocols for the interactive consistency problem of cloud computing are not enough for a cloud-computing environment with hybrid dual fallible components that nodes and communication media maybe in dormant or malicious fault simultaneously. In this study, the interactive consistency problem with a hybrid dual fallible cloud computing topology is revisited. The new proposed protocol can make all fault-free nodes reach consistency with minimal rounds of message exchanges and tolerate the maximal number of allowable dormant and malicious faulty nodes and communication media in a cloud computing environment.

Keywords: interactive consistency problem, fault tolerance, reliability, cloud computing, distributed computing

1. INTRODUCTION

As network bandwidth and quality outstrip computer performance, various communication and computing technologies previously regarded as being of different domains can now be integrated, such as telecommunication, multimedia, information technology, and construction simulation [9]. Thus, applications associated with network integration have gradually attracted considerable attention. Similarly, cloud computing facilitated through distributed application over networks has also gained more recognition [3]. Cloud computing has greatly encouraged distributed systems design and practiced to support user-oriented service applications [9]. However, distributed systems have grown rapidly in both size and number. In a distributed computing system, nodes allocated to different places or in separate units are connected together so that they collectively may be used to greater advantage. In many cases, reaching a common consistency in the presence of faulty components is the central issue of fault-tolerant distributed computing, because many applications require such consistency [3].

Received June 29, 2016; revised September 25, 2016; accepted October 4, 2016.
Communicated by Ren-Hung Hwang.

Currently, cloud computing can ensure increased ability to use the low-power nodes to achieve high reliability [5]. Furthermore, many applications of cloud computing provide the convenience of users. For users, the system must provide better reliability and fluency [9]. Therefore, reliability is one of the most important aspects of cloud computing. To ensure that a cloud-computing environment is reliable, a mechanism to allow a set of nodes to reach an agreed value is necessary.

The Byzantine Agreement (BA) problem first studied by Lamport *et al*. is a well-known paradigm for the problem of achieving reliability in a distributed network of nodes. The definitions of the BA problem are [4]: (1) There are n nodes ($n \geq 4$), of which at most one-third of the total number of nodes could fail without breaking down a workable network; (2) The nodes communicate with each other through message exchange in a fully connected network; (3) The message's sender is always identifiable by the receiver; (4) A node is chosen as a source, and its initial value v_s is transmitted to other nodes for executing the protocol; (5) The faulty component considered is node only.

A closely related sub-problem, the *interactive consistency* problem (IC problem) has been studied extensively [2]. The definition of IC problem is to make the fault-free nodes in an n-node distributed system reach interactive consistency. Each node chooses an initial value and communications with the others by exchanging messages. There is interactive consistency in that each node i has its initial value v_i and agrees on a set of common values. Therefore, interactive consistency has been achieved if the following conditions are met [2]:

Consistency: Each fault-free node agrees on a set of common values $V = [v_1, v_2, \ldots, v_n]$.
Validity: If the initial value of fault-free node i is v_i, then the ith value in the common vector V should be v_i.

However, cloud computing is an Internet-based development. It is a style of computing in which dynamically scalable and often virtualized resources are provided as a service over the Internet. Nevertheless, in a cloud-computing environment, the connected topology is not very significant. In previous protocol proposed by Wang *et al*., the faulty component is assumed node only [12, 13]. However, the communication medium fault was assumed as the node fault in [12, 13]. And, the fault-free nodes might be treated as faulty nodes due to their failed communication media [12, 13]. This assumption is unreasonable, for it violates the definition of interactive consistency that all fault-free nodes should agree on a common value. In additional, the topology of the previous research results is a fully connected network [4]. However, the topology does not meet the current network topology that cloud computing environment used.

In this study, the IC problem is to be solved on a cloud-computing environment. In addition, the types of faulty components are assumed to be in the hybrid where faults can come from both the nodes and the communication media, and the types of faults are assumed to be in the dual where faults can be dormant or malicious. The proposed protocol, is named Optimal Cloud Consistency Protocol (in short OCCP) that can use a minimum number of message exchanges and can tolerate a maximum number of allowable faulty components to make each fault-free node reach a common interactive consistency in the cases of node failure, communication medium failure, and both node and communication medium failure.

The rest of this paper is organized as follows. Section 2 will serve to introduce the basic assumption of the interactive consistency. Then, the proposed protocol OCCP will be brought up and illustrated in detail in Section 3. In Section 4, gives an example of executing the proposed protocol. Section 5 is responsible for proving the correctness and complexity of our new protocol. Finally, Section 6 gives conclusions of this research.

2. BASIC ASSUMPTIONS OF INTERACTIVE CONSISTENCY

Before the IC problem can be solved, two basic assumptions must be made and clearly defined in advance. They are the network structures and the failure types of faulty components.

2.1 The Network Structure

With the advancement and development of various technologies, computing problems become more complicated and larger [9]. A cloud-computing environment allows a user faster operation of Internet applications. The majority of cloud-computing infrastructure consists of reliable services delivered through data centers and built on servers with different levels of virtualization technologies [5]. The services are accessible anywhere that has access to networking infrastructure. Commercial offerings must meet the quality of service requirements of customers, and typically offer service-level agreements [9]. Therefore, a distributed system must be having high stability to handle instances where many users utilize a given environment. In this section, the topology of cloud computing is discussed.

According to the researches of [1, 10], a two-layer cloud topology is used in this study. The topology of two-layer cloud environment is shown in Fig. 1. The topology is composed of two layers, as follows:

(1) The nodes in Layer-A receive the service requests from users of different types of applications.

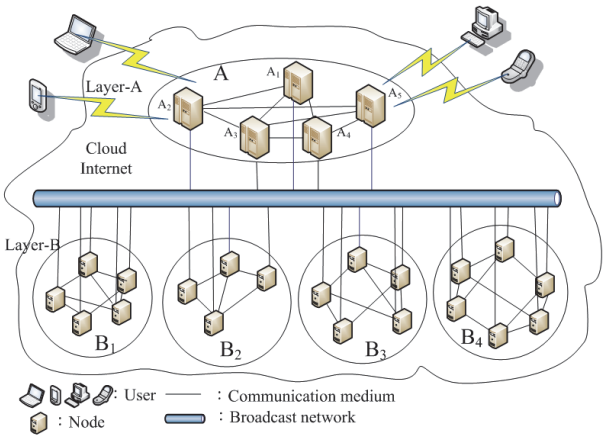


Fig. 1. The topology of two-layer cloud computing environment.

(2) Some nodes form service blocks in Layer-B, where each service block provides a specific application service. According to the properties of nodes, the nodes are clustered to service block B_i where $1 \leq i \leq g_n$ and g_n is the total number of service blocks in Layer-B.
(3) For the reliable communication, the broadcast network is used to connect the nodes between Layer-A and Layer-B. In Layer-A, each node must forward the message to all nodes in the corresponding service block of Layer-B.

2.2 Failure Types

In previous works, researchers used to assume that the faulty components in the IC problem were nodes only [11]. In reality, it is also possible for any communication medium to be failed. From this viewpoint, to treat a communication medium fault as a node fault violates the definition of the IC problem because the innocent nodes will be excluded from the common consistency when communication medium faults are treated as node faults. To deal with this problem, a communication medium fault should not be taken for a node fault.

However, network components may not always work well. A node is said to be fault-free if it follows protocol specifications during the execution of a protocol; otherwise, the node is said to be faulty. The symptoms of node failure can be classified into two categories. There are dormant fault and malicious fault (also called as the Byzantine fault) [4]. The dormant faults of nodes include crashes and omission. A crash fault happens when a node is broken. An omission fault takes place when a node fails to transmit or receive a message on time or at all. On the occasion of a malicious fault, the behavior of a faulty node is unpredictable and arbitrary. The message transmitted by a malicious faulty node is random or arbitrary. It is the most damaging failure type and causes the worst problem. That is, if the IC problem can be solved in a malicious fault case, then the IC problem can also be solved in other failure mode.

The symptoms of communication medium failure can also be classified into two categories: dormant fault and malicious fault [11]. The dormant faults of communication media are crash and stuck-at. A crash fault happens when a communication medium is broken. A stuck-at fault takes place when the message received from a certain communication medium is always a constant value. However, a communication medium with the malicious fault is one whose behavior is unrestricted and arbitrary. It is also the most damaging failure type of all and causes the worst problem. If a common consistency can be reached at the presence of a malicious fault, then a common consistency in the other failure modes can also be reached. That is, a fault-free communication medium can transmit messages on time and correctly, but the message which is transmitted by a faulty communication medium may be changed or delayed.

The hybrid failure mode is one where both faulty nodes and faulty communication media exist [6]. In this mode, there are totally nine types of failure combinations because there are three types of node faults and three types of communication medium faults. The nine types of failure combinations are crash-crash, crash-stuck (stuck-at), crash-malicious, omission-crash, omission-stuck, omission-malicious, malicious-crash, malicious-stuck, and malicious-malicious. The first part in the pair indicates the type of faulty node, and the second one denotes the type of faulty communication medium. The worst case of all those above is the case of malicious-malicious [6]. If the IC problem in the case of

malicious-malicious failure can be solved, then the IC problem in all the other eight types of failure modes can also be solved.

In the synchronous system, a fault-free node can detect the dormant faults by encodes a transmitted message by Manchester code before transmission [7, 8]. Fischer *et al.* also indicate that BA in an asynchronous network is impossible even if only one crash faulty node [2]. Hence, the assumption of underlying cloud computing environment is synchronous.

3. THE DEFINITIONS AND CONDITIONS FOR THE IC PROBLEM

In this study, the IC problem is discussed in a synchronous cloud-computing environment, so no delay of nodes or communication media is included in our discussion. Therefore, the nodes executing our new protocol should receive the messages from other nodes within a predictable period of time. If the message is not received on time, the message must have been influenced by faulty components.

3.1 Constraints

In the IC problem, the number of faulty components can be allowed is determined by the total number of nodes and the connectivity of the network. In Lamport *et al*.'s protocol [4], the fallible component is node only, the failure type of the fallible node is malicious and network topology is fully connected. So that, the constraints of Lamport *et al.* is $n > 3f_m$ and $c=n-1$ where n is the number of nodes, f_m is the total number of allowable malicious faulty nodes and c is the connectivity in the distributed system. In Meyer *et al.*, the assumption of failure types of the fallible node are dual failure mode (both dormant fault and malicious fault), and the underlying network topology may not be fully connected [6]. Therefore, the constraint of Meyer *et al.* is $n > 3f_m+f_d$ and $c > 2f_m+f_d$ where f_d is the total number of allowable dormant faulty nodes in the distributed system. However, Siu *et al.* found that the correct constraint on number of nodes required should be $n > \lfloor(n-1)/3\rfloor+2f_m+f_d$ [11].

In this paper, OCCP is used to solve the IC problem in a cloud-computing environment ***with fallible nodes and communication media***, and ***dual failure mode*** assumed. With consideration for efficient consistency, the interactive consistency is applied to each node in Layer-A, and the majority function is applied to the nodes in Layer-B.

Therefore, the constraint of the OCCP in Layer-A is $n_A > \lfloor(n_A-1)/3\rfloor+2f_{mA}+f_{dA}$ where n_A is the number of nodes, f_{mA} is the total number of allowable malicious faulty nodes, and f_{dA} is the total number of allowable dormant faulty nodes in Layer-A. This constraint specifies the number of nodes required in Layer-A.

Constraint 1: $n_A > \lfloor(n_A-1)/3\rfloor+2f_{mA}+f_{dA,}$ where n_A is the number of nodes, f_{mA} is the total number of allowable malicious faulty nodes, and f_{dA} is the total number of allowable dormant faulty nodes in Layer-A.

Constraint 2: $c_A > 2C_{mA}+C_{dA,}$ where c_A is the connectivity of Layer-A, C_{mA} is the maximum number of malicious faulty communication media and C_{dA} is the maximum number of dormant faulty communication media in Layer-A.

Constraint 1 specifies the number of nodes in Layer-A required; due to the unit of the cloud computing environment is node, so that a consistency can be achieved if $n_A > \lfloor (n_A - 1)/3 \rfloor + 2f_{mA} + f_{dA}$. On the other hand, **Constraint 2** specifies the required connectivity of Layer-A. The constraint with the connectivity of Layer-A is based on the number of malicious faulty nodes f_{mA}, the number of dormant faulty nodes f_{dA}, the number of malicious faulty communication media C_{mA}, and the number of dormant faulty communication media C_{dA} in Layer-A. In addition, the total number of malicious faulty nodes and the total number of malicious faulty communication media must be smaller than half of $c_A - C_{dA}$. Hence, the constraint as to the connectivity of Layer-A in cloud computing environment is $c_A > 2C_{mA} + C_{dA}$.

3.2 Approach

In the hybrid case, both nodes and communication media may be failed simultaneously. If a common consistency value from fault-free nodes in the hybrid case needs to be reached, the faulty influences from both the nodes and the communication media must to be removed. In this section, OCCP is introduced to solve IC problem in dual failure mode with both fallible nodes and fallible communication media in a cloud-computing environment. In OCCP, Virtual Channel (VC) is used to transmit the message(s), so the VC is introduced at first.

3.2.1 The transmission protocol VC

The concept of Virtual Channel (VC) is a way to make a reliable un-fully connected network (without faulty communication media) work just like a fully connected network. In this paper, the proposed protocol VC cannot only make an un-fully connected network work just like a fully connected network but also remove the influence from malicious/dormant faulty communication media and malicious/dormant faulty nodes between the sender node and receiver node. The definition of our protocol VC is shown in Fig. 2.

In VC, the receiver can always detect the message(s) through dormant faulty components if the protocol VC appropriately encodes a transmitted message by Manchester code [7, 8]. Therefore, the message(s) through dormant faulty nodes and dormant faulty communication media can be detected and the value λ^0 is replaced as the message received. The value λ^0 is used to represent the absence message.

By using our VC, the influences by the dormant faulty components between any pairs of nodes can be gotten rid of in each round of message exchange, and the influences by the malicious faulty communication media between any pairs of nodes can be ruled out in each round of message exchange if $c_A > 2C_{mA} + C_{dA}$. Due to the fault-free sender node can send c_A copies of a message to fault-free receiver nodes. In the worst case, a fault-free receiver node can receive $c_A - C_{dA}$ messages transmitted by the fault-free sender node. So that, a fault-free destination node can decide which the correct messages are by taking the majority value.

The function VMAJ is shown in Fig. 2. There are four cases in the function VMAJ. The case 1 is used to detect the sender node is a dormant faulty node or not. The case 2 and case 3 are the usual cases to output the correct messages from the sender node that is not a dormant faulty node and also not an asymmetric malicious faulty node. The case 4 is used to detect the sender node is an asymmetric malicious faulty node or not.

An example of VC is shown in Fig. 3. Fig. 3 (a) illustrates a 4-connectivity network model with six nodes, and Fig. 3 (b) illustrates the sender node A_1 using VC to transmit a message to destination node A_4. There are four adjacent paths to node A_4, so node A_4 can receive four values from the sender node. In Fig. 3 (b), these four values can be acquired by node A_1 via VC to transmit messages to node A_4 directly, through node A_2 to node A_4, through node A_5 to node A_4, and through node A_6 to node A_4. Node A_4 can receive the vector $V_1 = [v_1, v_2, v_5, v_6]$ and use function VMAJ on vector V_1.

3.2.2 The proposed protocol OCCP

In this section, OCCP is introduced to solve IC problems in dual failure mode with fallible nodes and communication media underlying a cloud-computing environment. The proposed protocol OCCP is organized as two parts, the *Interactive Consistency Process* and the *Agreement Process*. However, the nodes of Layer-A execute *Interactive Consistency Process* first, and then the nodes in service block of Layer-B execute *Agreement Process.*

Protocol VC (Virtual Channel)
Definition:

- Each node has the common knowledge of graphic information $G = (E, N)$, where N is the set of nodes in the Layer-A and E is a set of node pairs, (A_i, A_j), indicating a communication medium between node A_i and node A_j, where $1 \le i, j \le n_A$.
- There are c_A ($c_A > 2C_{mA}+C_{dA}$) paths from sender node to destination nodes.
- The c_A disjoint paths between the sender nodes to destination nodes can be predefined.
- These c_A paths from sender node to destination nodes are node-disjoint paths.
- Each intermediate node on these c_A paths should not be passed through more than once.

Steps:

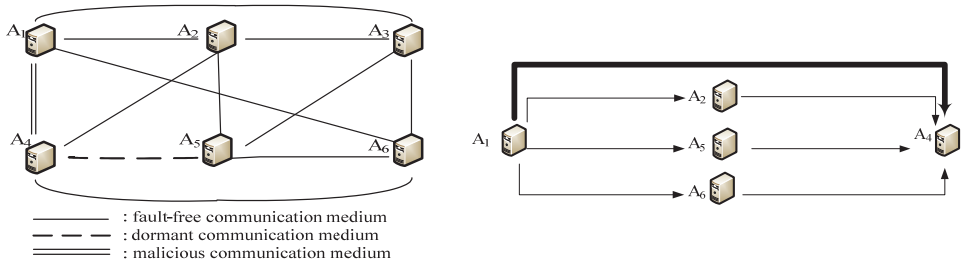
1. The sender node A_i ($1 \le i \le n_A$) transmits initial v_i to the destination node through c_A node-disjoint paths.
2. If the node-disjoint path from sender node to destination node goes through any dormant communication media, then the transmits initial v_i is replaced by λ^0.
3. If the node-disjoint path from sender node to destination node goes through any dormant faulty node or if the sender node suffers from dormant faults, then the transmits initial v_i is replaced by λ^0.
4. The nodes in the destination node take the majority value from the same node-disjoint paths and construct the vector $V_i = [v_{path\ 1}, v_{path\ 2}, \ldots, v_{path\ cA\text{-}1}, \ldots, v_{path\ cA}]$ for $c_A > 2C_{mA}+C_{dA}$.
5. The nodes in the destination node apply the function VMAJ on the values of vector V_i.

The function VMAJ(V)

Begin
if the majority value is λ^0 and the number of value λ^0 is greater than or equal to $c_A - \lfloor (n_A-1)/3 \rfloor$
  then output the value λ^0 /* case 1*/
  else begin
    count the non-λ^0 value
    if the majority value is λ^k, where $1 \le k \le \lfloor (n_A-1)/3 \rfloor$
      output the value λ^k /*case 2*/
    if the majority value is the non-λ^k value, where $0 \le k \le \lfloor (n_A-1)/3 \rfloor$, $m \in \{0,1\}$

```
      output the value m                                    /*case 3*/
    if the majority value is not existed
      output the value λ^0                                  /*case 4*/
  end
end.
```

Fig. 2. The proposed protocol VC.



(a) An example of c_A-connected network (n_A = 6, c_A = 4).

(b) The sender node A_1 uses VC to transmit message to destination node A_4.

Fig. 3. An example of VC.

In the *Interactive Consistency Process*, the primary work of Layer-A's nodes is to collect the user's service requests, and then the request vector of the interactive consistency can be obtained to trigger the nodes in Layer-B to perform the service request. In a cloud-computing environment, each node in Layer-A receives the various requests from users, and the nodes in Layer-B's service block provide services to the users. Hence, each node of Layer-A may receive different service requests from users. Each node in Layer-A uses the service request as the initial value to execute OCCP to obtain the common vector DEC_A. Then, each node of Layer-A forwards the element of vector DEC_A to the nodes in the service block of Layer-B. However, the specific service request is to be confirmed by the nodes of same service block in the *Agreement Process*. Each node in the service block of Layer-B receives the element from the nodes of Layer-A, and then the majority value of the received element values is taken. Finally, the consistency value is obtained by each fault-free node. The proposed protocol OCCP is presented in Fig. 4.

When OCCP is initiated by the nodes of Layer-A, each node of Layer-A requires $\sigma = \lfloor (n_A-1)/3 \rfloor + 1$ rounds to receive enough values to reach interactive consistency in the *Interactive Consistency Process*. In the first round of *Message Exchange Phase* in *Interactive Consistency Process*, each node of Layer-A parallel transmits its initial value to other nodes in Layer-A by using VC, and then receives the value by using VC and stores it in the root of its mg-tree. The mg-tree is a tree structure which is used to store the received messages. Subsequently, each node of Layer-A parallel transmits the values at level $(r-1)$ by using VC in the corresponding mg-tree to other nodes in Layer-A. In the *Decision Making Phase*, each node of Layer-A reorganizes its mg-tree into a corresponding ic-tree. The ic-tree is a tree structure that is used to store a received message without repeated node names.

Finally, all fault-free nodes use function VOTE to remove the faulty influence from

faulty nodes to obtain the common value. The function VOTE only counts the non-value λ^0 (excluding the last level of the ic-tree) for all vertexes at the rth level of an ic-tree, where $1 \le r \le (n_A-1)/3\rfloor+1$. The condition 1, condition 4, and condition 5 in the function VOTE are similar to conventional majority vote. The condition 2 is used to remove the influence by a malicious faulty node. The condition 3 is used to solve the case of dual failure mode and describes the existence of an absentee. When the function VOTE is applied to the root of each corresponding ic-tree, and then a vector DEC_A with n_A elements is obtained. Each element of DEC_A is mapped to a specific application that will be executed in the corresponding service block of Layer-B.

In Layer-B, each service block provides a specific application, and some service blocks maybe provide the same applications. All nodes in the same service block provide the same application. However, each element of the vector DEC_A is mapped to a specific service in a service block of Layer-B. Therefore, nodes in the same service block will receive the mapped element from the nodes of Layer-A. The node of Layer-B that receives the element of DEC_A represents the same application for all nodes in the same service block. In the *Agreement Process* of OCCP, each node in the same service block of Layer-B receives the element from each node of Layer-A with the vector DEC_A. However, the maximal number of allowable faulty nodes in Layer-A cannot exceed $\lfloor(n_A-1)/3\rfloor$. Hence, each node in same service block of Layer-B can receive at least $n_A-\lfloor(n_A-1)/3\rfloor$ correct values. The amount of correct values is greater than the amount of faulty values. Therefore, when each node takes the majority value from the received values, each node in the same service block of Layer-B can obtain the common correct value.

OCCP (Optimal Cloud Consistency Protocol)

The nodes of Layer-A execute ***Interactive Consistency Process***.
The nodes of Layer-B's service block execute ***Agreement Process***.

Interactive Consistency Process (for the node A_i in the Layer-A with initial value v_i; $1 \le i \le n_A$, where n_A is the number of nodes in Layer-A, for $n_A > 3$)

Pre-Execute.

Compute the number of rounds required $\sigma = \lfloor(n_A-1)/3\rfloor+1$

Message Exchange Phase:

If $r = 1$ then:
1) Each node A_i parallel broadcasts its initial value v_i to other nodes in Layer-A by using VC.
2) Each node receives and stores the n_A values sent from n_A nodes of Layer-A in the corresponding root of its mg-tree by using VC.
3) If the initial value v_i received from node A_i is "λ^0", then using the value "0" to replace the value "λ^0".

For $1 < r \le \sigma$, do:
1) Each node uses VC to parallel transmit the function VMAJ values at level $r-1$ in the corresponding mg-tree to other nodes in Layer-A, if the VMAJ value at level $r-1$ is λ^k, then replace the value λ^{k+1} as the transmitted value, where $0 \le k \le \lfloor(n_A-1)/3\rfloor$.
2) Each receiver node stores the VMAJ values in the corresponding vertices at level r of its mg-tree.

Decision Making Phase:

Step 1: Reorganize each mg-tree into a corresponding ic-tree by deleting the vertices with repeated node names.

Step 2: Using function VOTE with the root i of each node's ic-tree and obtaining the common value VOTE(i).

```
VOTE(α) = begin
            if the α is a leaf
            then output the value α                                  /* condition 1*/
            else begin
               if the number of value λ^0 is 3*(⌊(n_A–1)/3⌋–r+1)+[(n_A–1) mod 3]
                  output the value α                                 /* condition 2*/
               if the majority value is λ^k, where 1≤k≤⌊(n_A-1)/3⌋
                  output the value λ^(k-1)                           /* condition 3*/
               if the majority value is the non-λ^k value, where 0≤k≤⌊(n_A–1)/3⌋,
                  output the majority value m, m∈{0,1}               /* condition 4*/
               if the majority value is not existed
                  output the default value φ                         /* condition 5*/
             end
          end.
```

Agreement Process (for the node B_{ii} in the service block B_i of Layer-B, $1 \le i \le n_{Bi}$ where n_{Bi} is the number of nodes in service block B_j of Layer-B)

Step 1: All nodes in the same service block B_j of Layer-B receive the element value of DEC_A that transmits from the nodes in Layer-A for the specific application needs.

Step 2: Each node of same service block B_j in Layer-B takes a majority value of the received element values and the consistency value v is obtained.

Fig. 4. Protocol OCCP.

4. AN EXAMPLE OF EXECUTING OCCP AND VC

The example of executing the OCCP based on a hybrid dual fallible cloud-computing environment is discussed in this section. An example of Layer-A of a cloud-computing environment is shown in Fig. 5 (a).

The nodes in Layer-A receive the service request. The protocol, for this example, two rounds ($\sigma = \lfloor (n_A-1)/3 \rfloor + 1 = \lfloor (5-1)/3 \rfloor + 1 = 2$, where n_A is the number of nodes in Layer-A) are required to exchange the messages. In this example, there are five nodes in Layer-A, where each node receives the user's service request. Fig. 5 (b) is the initial value of each node in Layer-A. Each node receives different service requests from different users, *e.g.*, A_1 receives the mail service request and A_2 receives the video service request, etc. However, some nodes in Layer-A may receive the same requests. During the first round of *Message Exchange Phase*, each node of Layer-A parallel transmits the initial value by using VC to all nodes of Layer-A and stores the received $n_A(=5)$ values in the corresponding root of each mg-tree, as shown in Fig. 5 (c). In the second round, each node parallel transmits the values in the root of the corresponding mg-tree by using VC to other nodes in Layer-A and stores the received values in level 1 of the $n_A(=5)$ corresponding mg-trees. The progression of nodes A_1 and A_4 during *Message Exchange Phase i*s shown in Figs. 5 (d) and (f).

Subsequently, in the *Decision Making Phase*, the mg-tree is reorganized into the ic-tree by deleting those vertices with repeated node names. The corresponding ic-tree of

nodes A_1 and A_4 is shown in Figs. 5 (e) and (g). Then, function VOTE is applied on the ic-tree root of each node to take the majority value, as shown in Figs. 5 (h) and (i). Eventually, the common consistency value DEC_A of nodes A_1 and A_4 is obtained.

All nodes in the service block of Layer-B receive the element of DEC_A from the nodes of Layer-A by broadcast network, as shown in Fig. 5 (j). However, the nodes of the service block in Layer-B execute *Agreement Process*. For example, all nodes in B_3 receive the element value of DEC_A that is transmitted from the nodes in Layer-A for the specific application needs. If the nodes in Layer-A send the E-mail service request with elements of DEC_A to all nodes in B_3, then all nodes in B_3 receive the element of DEC_A that is transmitted from Layer-A's nodes, as shown in Fig. 5 (k). Subsequently, all nodes in the B_3 must take a majority value from the received element values, as shown in Fig. 5 (l). Finally, the common values through the *Agreement Process* with each fault-free node can be obtained.

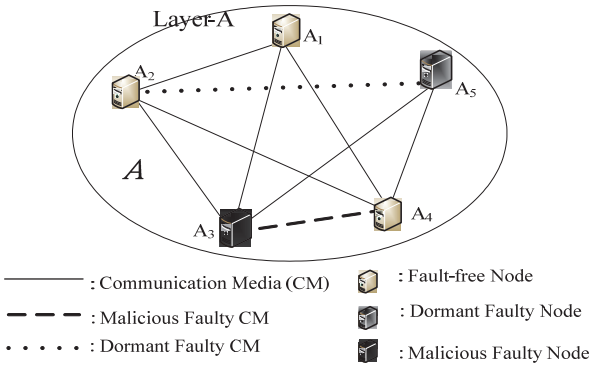


Fig. 5. (a) Example of Layer-A.

A_1	A_2	A_3	A_4	A_5
1	1	0	1	0

Fig. 5. (b) The initial value of each node in Layer-A.

	level 0 Root		*Using VC*
A_1	1	1	←(1,1,*0*,1,*0*)
	2	1	←(1,1,*0*,1,*0*)
	3	***0***	←(0,0,*1*,0,*1*)
	4	1	←(1,1,*0*,1,*0*)
	5	λ^0	←(λ^0,λ^0,*0*,λ^0,*0*)

	level 0 Root		*Using VC*
A_2	1	1	←(1,1,*0*,1,*0*)
	2	1	←(1,1,*0*,1,*0*)
	3	***1***	←(1,1,*1*,1,*1*)
	4	1	←(1,1,*0*,1,*0*)
	5	λ^0	←(λ^0,λ^0,*0*,λ^0,*0*)

	level 0 Root		*Using VC*
A_3	1	1	←(1,1,*0*,1,*0*)
	2	1	←(1,1,*0*,1,*0*)
	3	***1***	←(1,1,*0*,1,*1*)
	4	1	←(1,1,*0*,1,*0*)
	5	λ^0	←(λ^0,λ^0,*0*,λ^0,*0*)

	level 0 Root		*Using VC*
A_4	1	1	←(1,1,*0*,1,*0*)
	2	1	←(1,1,*0*,1,*0*)
	3	***0***	←(0,0,*1*,0,*1*)
	4	1	←(1,1,*0*,1,*0*)
	5	λ^0	←(λ^0,λ^0,*0*,λ^0,*0*)

	level 0 Root		*Using VC*
A_5	1	1	←(1,1,*0*,1,*0*)
	2	1	←(1,1,*0*,1,*0*)
	3	***1***	←(1,1,*1*,1,*1*)
	4	1	←(1,1,*0*,1,*0*)
	5	λ^0	←(λ^0,λ^0,*0*,λ^0,*0*)

Fig. 5. (c) The mg-tree of each node in Layer-A at the first round of *Message Exchange Phase*.

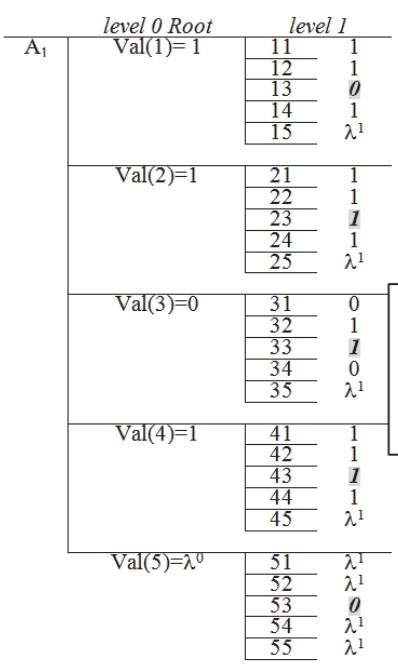


Fig. 5. (d) The final mg-tree of A_1.

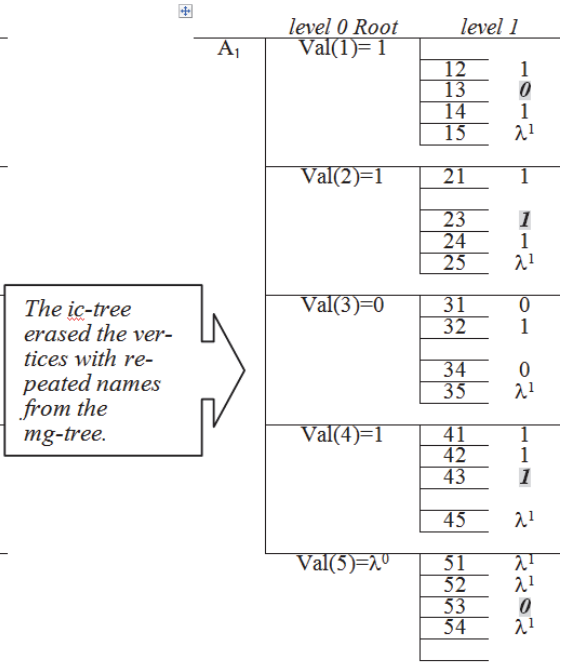


Fig. 5. (e) The ic-tree of A_1 by *Decision Making Phase*.

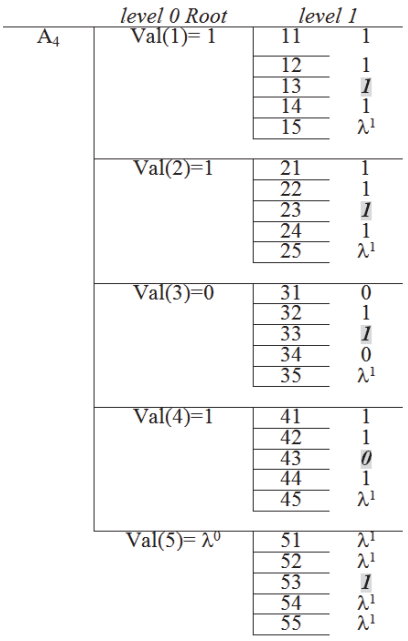


Fig. 5. (f) The final mg-tree of A_4.

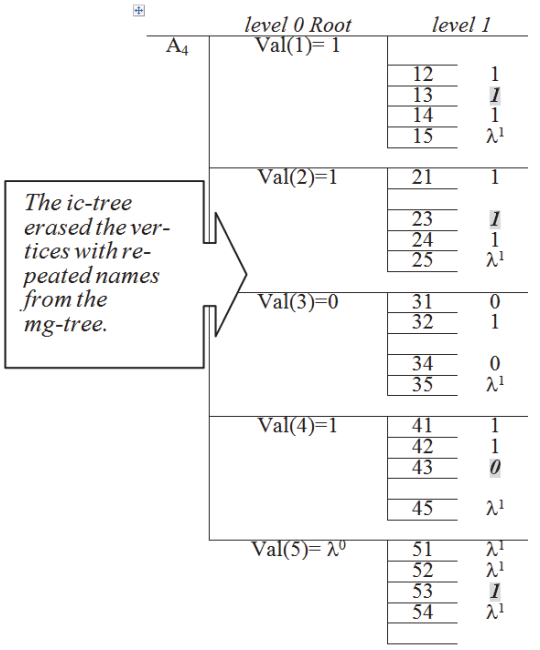


Fig. 5. (g) The ic-tree of A_4 by *Decision Making Phase*

VOTE(1)= majority(*val*(12), *val*(13),*val*(14),*val*(15))= majority(1, ***0,*** 1, λ^1)= 1
VOTE(2)= majority(*val*(21),*val*(23),*val*(24),*val*(25))= majority(1, ***1,*** 1, λ^1)= 1
VOTE(3)= majority(*val*(31),*val*(32),*val*(34),*val*(35))= majority(0, 1**,** 0, λ^1)= *0*
VOTE(4)= majority(*val*(41),*val*(42),*val*(43),*val*(45))= majority(1, 1, ***1,*** λ^1)= 1
VOTE(5)= majority(*val*(51),*val*(52),*val*(53),*val*(54))= majority(λ^1, λ^1, ***0,***, λ^1)= λ^1

DEC_A=(1,1,*0*,1, λ^1)

Fig. 5. (h) The common value VOTE(*i*) by A_1 in *Decision Making g Phase*.

VOTE(1) = majority(*val*(12),*val*(13),*val*(14),*val*(15)) = majority(1, ***1,*** 1, λ) = 1
VOTE(2) = majority(*val*(21),*val*(23),*val*(24),*val*(25)) = majority(1, ***1,*** 1, λ) = 1
VOTE(3) = majority(*val*(31),*val*(32),*val*(34),*val*(35)) = majority(0, 1**,** 0, λ) = *0*
VOTE(4) = majority(*val*(41),*val*(42),*val*(43),*val*(45)) = majority(1, 1, ***0,*** λ) = 1
VOTE(5) = majority(*val*(51),*val*(52),*val*(53),*val*(54)) = majority(λ, λ, ***1,***, λ) = λ

DEC_A=(1,1,*0*,1, λ)

Fig. 5. (i) The common value VOTE(*i*) by A_4 in *Decision Making Phase*.

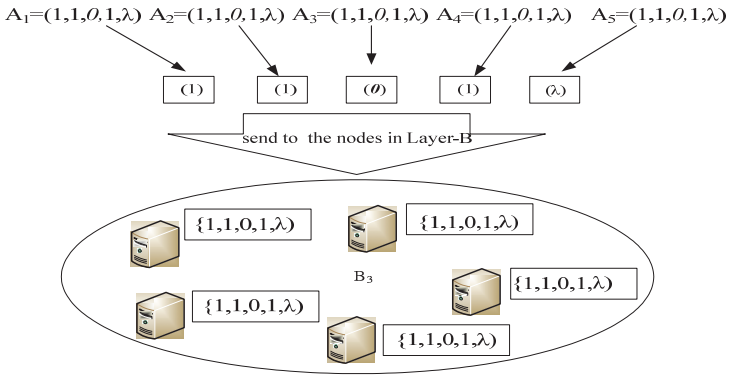


Fig. 5. (j) Example of the nodes in Layer-A forwarding the value to service block B_3 of Layer-B.

A_1	1	A_1	1	A_1	1	A_1	1	A_1	1
A_2	1	A_2	1	A_2	1	A_2	1	A_2	1
A_3	*0*	A_3	*0*	A_3	*1*	A_3	*0*	A_3	*0*
A_4	1	A_4	1	A_4	1	A_4	1	A_4	1
A_5	λ^1	A_5	λ^1	A_5	λ^1	A_5	λ^1	A_5	λ^1
B_{31}		B_{32}		B_{33}		B_{34}		B_{35}	

Fig. 5. (k) Each node of B_3 receives element of DEC_A from Layer-A's node.

	majority			majority	
B_{31}= (1,1,0,1,λ^1)	=>	MAJ_1=1	B_{34}= (1,1,0,1,λ^1)	=>	MAJ_4=1
B_{32}= (1,1,0,1,λ^1)	=>	MAJ_2=1	B_{35}= (1,1,0,1,λ^1)	=>	MAJ_5=1
B_{33}= (1,1,1,1,λ^1)	=>	MAJ_3=1			

Fig. 5. (l) The common value of each node in service block B_3.

Fig. 5. The example of executing the OCCP.

5. THE CORRECTNESS AND COMPLEXITY OF OCCP

The following lemmas and theorems are used to prove the correctness and complexity of OCCP.

5.1 Correctness of OCCP

To prove the correctness of our proposed protocol, a vertex α is called common if each fault-free node has the same value for α. That is, if vertex α is common, then the value stored in vertex α of each fault-free node's mg-tree or ic-tree is identical. When each fault-free node has a common initial value of node A_i in the root of an ic-tree, if the root A_i of an ic-tree in a fault-free node is common and the initial value received from the node A_i is stored in the root of the tree structure, then a consistency is reached because the root is common. Thus, the constraints, (Consistency) and (Validity), can be rewritten as:

(**Consistency'**): Root i is common, and
(**Validity'**): VOTE(i)=v_i for each fault-free node, if the node A_i is fault-free.

To prove that a vertex is common, the term common frontier is defined as follows: When every root-to-leaf path of the tree (an mg-tree or an ic-tree) contains a common vertex, the collection of the common vertices forms a common frontier. In other words, every fault-free node has the same messages collected in the common frontier if a common frontier does exist in a fault-free node's tree structure (mg-tree or ic-tree); subsequently, using the same majority voting function to compute the root value of the tree structure, every fault-free node can compute the same root value because the same input (the same collected messages in the common frontier) and the same computing function will cause the same output (the root value).

Since OCCP can solve the IC problem, the correctness of OCCP should be examined by the following two terms.

(1) Correct vertex: Vertex αi of a tree is a correct vertex if node A_i (the last node name in the name list of vertex αi) is fault-free. In other words, a correct vertex is a place to store the value received from a fault-free node.
(2) True value: For a correct vertex αi in the tree of a fault-free node, val(αi) is the true value of vertex αi. In other words, the stored value for a correct vertex is called the true value.

By the definition of a correct vertex, its stored value is received from the fault-free node, and a fault-free node always transmits the same value to all nodes; therefore, the correct vertices of such an mg-tree are common. After reorganizing the mg-tree into its corresponding ic-tree by deleting the vertices with repeated node names, the values stored on the correct vertices of an ic-tree shall be the same. As a result, all the correct vertices of an ic-tree are also in common [4]. Again, by the definition of a correct vertex, a common frontier does exist in the ic-tree inasmuch as **Consistency'** and **Validity'** are true, regardless of whether the source node is fault-free or faulty if the IC problem has been solved.

Lemma 1: The message(s) through dormant faulty nodes and dormant faulty communication media can be detected by fault-free destination node.

Proof: The message(s) from dormant faulty components can be detected if the protocol appropriately encodes a transmitted message by Manchester code before transmission [7, 8].

Theorem 1: The fault-free destination node can receive the message(s) from sender node without influence from any faulty components between the sender node and destination node if $c_A > 2C_{mA} + C_{dA}$.

***Proof*:** By Lemma 1, the influence from dormant faulty components between any pair of sender node and destination node can be removed in each round of message exchange, and the influences by the malicious faulty components between any pairs of nodes in Layer-A in each round of message exchange if $c_A > 2C_{mA}+C_{dA}$ can be ruled out. The reason is that the fault-free sender node sends c_A copies of a message to fault-free destination nodes. In the worst case, a fault-free destination node can receive $c_A - C_{dA}$ messages transmitted by the fault-free sender node. Due to the message(s) from dormant faulty components can be detected; in addition, the $c_A - C_{dA} > 2C_{mA}$ can also be obtained. Therefore, a fault-free destination node can decide which the correct messages are by taking the majority value.

Lemma 2: The fault-free destination node can detect the dormant faulty sender node by VC.

***Proof*:** If the number of value λ^0 is greater than or equal to $c_A - \lfloor (n_A-1)/3 \rfloor$ then the sender node is in dormant fault. The reason is that there are at most $\lfloor (n_A-1)/3 \rfloor$ malicious faulty components in the network, hence there are at most $\lfloor (n_A-1)/3 \rfloor$ non-λ^0 value in the vector V_i.

Theorem 2: The fault-free node can detect all dormant faulty nodes in the network.

***Proof*:** In the protocol OCCP, there are $\lfloor (n_A-1)/3 \rfloor+1$ rounds of message exchange in *Interactive Consistency Process*, where $n_A \geq 4$, so there are at least two rounds of message exchange in the *Message Exchange Phase*. Each fault-free node can receive the message from node A_i in the first round of message exchange by using VC, and receive other nodes message(s) in the second round of message exchange by using VC. Without node A_i, each node can receive all other nodes' message(s) in the network after two rounds of message exchange by using VC. According to the Lemma 2, each fault-free node can detect all dormant faulty nodes in the network.

Lemma 3: All correct vertices of an ic-tree are common.

***Proof*:** After reorganization, no repeatable vertices are in an ic-tree. At the level $\lfloor (n_A-1)/3 \rfloor+1$ or above, the correct vertex α has at least $2\lfloor (n_A-1)/3 \rfloor+1$ children in which at least $\lfloor (n_A-1)/3 \rfloor+1$ children are correct. The true value of these $\lfloor (n_A-1)/3 \rfloor+1$ correct vertices is in common, and the majority value of vertex α is common. The correct vertex α is common in the ic-tree, if the level of α is less then $\lfloor (n_A-1)/3 \rfloor+1$. As a result, all correct vertices of the ic-tree are common.

Lemma 4: A common frontier exists in the ic-tree.

***Proof*:** There are $\lfloor(n_A-1)/3\rfloor+1$ vertices along each root-to-leaf path of an ic-tree in which the root is labeled by the name of node A_i, and the others are labeled by a sequence of node names. Since at most $\lfloor(n_A-1)/3\rfloor$ nodes can be failed, there are at least one vertex is correct along each root-to-leaf path of the ic-tree. By Lemma 3, the correct vertex is common, and the common frontier exists in each fault-free node's ic-tree.

Lemma 5: Let α be a vertex; α is common if there is a common frontier in the subtree rooted at α.

***Proof*:** If the height of α is 0, and the common frontier (α itself) exists, and then α is common. If the height of α is r, the children of α are all in common by using induction hypothesis with the height of the children at $r-1$, then the vertex α is common.

Corollary 1: The root is common if a common frontier exists in the ic-tree.

Theorem 3: The root of a fault-free node's ic-tree is common.

***Proof*:** By Lemma 3, Lemma 4, Lemma 5 and Corollary 1, the theorem is proved.

Theorem 4: Protocol OCCP solves the IC problem in a cloud computing.

***Proof*:** To prove the theorem, it has to show that OCCP meets the Consistency' and Validity'.

(Consistency'): Root i is common. By Theorem 3, (Consistency') is satisfied.
(Validity'): VOTE(i) = v for all fault-free nodes, if the initial value of the node A_i is v_i, say $v = v_i$.

Since most of nodes are fault-free, they use VC to transmit the message to all others. The value of correct vertices for all fault-free nodes' mg-tree is v. When the mg-tree is reorganized to an ic-tree, the correct vertices still exist. As a result, each correct vertices of the ic-tree is common (Lemma 3), and its true value is v. By Theorem 3, this root is common. The computed value VOTE(i) = v is stored in the root for all fault-free nodes. (Validity') is satisfied.

5.2 Complexity of OCCP

The complexity of OCCP is evaluated in terms of (1) the minimal number of rounds, and (2) the maximum number of allowable faulty components. Theorems 5 and 6 below will show that the optimal solution is reached.

Theorem 5: The maximum number of allowable faulty components by OCCP is $T_F = T_{FA} + T_{FB}$. T_{FA} is the total number of allowable faulty nodes in Layer-A and $T_{FA} = f_{mA}+f_{dA}$, where f_{mA} is the total number of allowable malicious faulty nodes, f_{dA} is the total number of allowable dormant faulty nodes in Layer-A, and $n_A > \lfloor(n_A-1)/3\rfloor+2f_{mA}+f_{dA}$. T_{FB} is the to-

tal number of allowable faulty nodes in Layer-B and $T_{FB} = \sum_{j=1}^{g_n} (n_{Bj} - 1)$.

***Proof*:** Siu *et al*. indicate the constraint of BA problem for node faults only as $n > \lfloor (n-1)/3 \rfloor + 2f_m + f_d$, and the unit they focus on is the node [11]. However, the fault status of our assumption is also that nodes are faulty. Hence, the constraint of the maximum number of allowable faulty nodes can be applied to our study. Therefore, $n > \lfloor (n-1)/3 \rfloor + 2f_m + f_d$ in Siu *et al.*, is implied to $n_A > \lfloor (n_A-1)/3 \rfloor + 2f_{mA} + f_{dA}$ in a hybrid dual fallible cloud-computing environment with n_A nodes of Layer-A. The constraint is rewritten as $n_A > \lfloor (n_A-1)/3 \rfloor + 2f_{mA} + f_{dA}$, and the total number of allowable faulty components by OCCP in Layer-A is f_{mA} malicious faulty nodes and f_{dA} dormant faulty nodes, which is maximal if $n_A > \lfloor (n_A-1)/3 \rfloor + 2f_{mA} + f_{dA}$.

In addition, there are g_n service blocks in Layer-B. Each service block has B_{ij} nodes, where $1 \le j \le g_n$. However, when a fault-free node of service block B_j in Layer-B exists, then the specific application can possibly be carried out. Therefore, the fault tolerant capability of Layer-B is $\sum_{j=1}^{g_n} (n_{Bj} - 1)$ where n_{Bj} is the number of nodes in service block B_j of Layer-B and there are g_n service blocks in Layer-B.

In conclusion, the maximum number of allowable faulty components of OCCP is $T_F = T_{FA} + T_{FB}$, where T_{FA} is the total number of allowable faulty nodes in Layer-A and T_{FB} is the total number of allowable faulty nodes in Layer-B.

Theorem 6: OCCP requires $\lfloor (n_A-1)/3 \rfloor + 2$ rounds to solve the IC problem in a hybrid dual fallible cloud-computing environment, and $\lfloor (n_A-1)/3 \rfloor + 2$ is the minimum number of rounds required to exchange messages.

***Proof*:** Because message passing is required only in the *Message Exchange Phase*, the *Message Exchange Phase* is time consuming. Wang *et al.* pointed out that $\lfloor (n-1)/3 \rfloor + 1$ rounds are the minimum number of rounds to send sufficient messages to achieve consistency in an *n*-node fallible distributed system [11]. However, in a hybrid dual fallible cloud-computing environment, the nodes and communication media maybe in dormant or malicious fault simultaneously. In addition, each node in the hybrid dual fallible cloud-computing environment must exchange messages with other nodes. Therefore, a constraint on the minimum number of rounds can be applied to the study. However, in a hybrid dual fallible cloud-computing environment, there are n_A nodes in Layer-A, OCCP needs $\lfloor (n_A-1)/3 \rfloor + 1$ rounds to exchange message. In addition, in the *Decision Making Phase*, each node in Layer-A sends the specific element of DEC_A to the nodes of a specific application executed service block in Layer-B. Therefore, an additional round of message exchange is required. In conclusion, the minimum number of rounds of OCCP is $\lfloor (n_A-1)/3 \rfloor + 2$. Moreover, the number rounds required is optimal.

6. CONCLUSIONS

The IC problem is fundamental in a distributed system, and has been extensively studied. Network topology is an important issue related to consistency. However, cloud computing is a new concept for distributed systems. It has greatly encouraged distributed

system design and practice to support user-oriented services. In this paper, the OCCP protocol is proposed to make all fault-free nodes reach consistency. This protocol can use a minimal number of rounds of message exchange and tolerate a maximal number of allowable faulty components in a hybrid dual fallible cloud-computing environment. The IC problem for dormant or malicious faulty nodes and communication media in a cloud-computing environment is revisited and the fault-tolerance capacity is enhanced by OCCP.

REFERENCES

1. S. Bera, S. Misra, and J. P. C. Rodrigues, "Cloud computing applications for smart grid: A survey," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 26, 2015, pp. 1477-1494.
2. M. Fischer, "The consensus problem in unreliable distributed systems (a brief survey)," *Lecture Notes in Computer Science*, 1983, pp. 127-140.
3. P. Kumar and S. K. Gupta, "Abstract model of fault tolerance algorithm in cloud computing communication networks," *International Journal on Computer Science and Engineering*, Vol. 3, 2011, pp. 3283-3290.
4. L. Lamport, R. Shostak, and M. Pease "The Byzantine generals problem," *ACM Transactions on Programming Languages and Systems*, Vol. 4, 1982, pp. 382-401.
5. V. Mauch, M. Kunze, and M. Hillenbrand, "High performance cloud computing," *Future Generation Computer Systems*, Vol. 29, 2012, pp. 1408-1416.
6. F. J. Meyer and D. K. Pradhan, "Consensus with dual failure modes," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 2, 1991, pp. 214-222.
7. M. Newman, *Networks: An Introduction*, Oxford University Press, Inc., NY, USA 2010.
8. N. Oliferand and V. Olifer, *Computer Network: Principles*, *Technologies and Protocols for Network Design*, John Wiley & Sons, 2006.
9. D. Puthal, B.P.S. Sahoo, S. Mishra, and S. Swain, "Cloud computing features, issues, and challenges: A big picture," in *Proceedings of International Conference on Computational Intelligence and Networks*, 2015, pp. 116-123.
10. M. N. Rajkumar and P. M. B. Mansingh, "An efficient and secure storage using delegated access control in multi-cloud environment," *International Journal of Software & Hardware Research Engineering*, Vol. 1, 2013, pp. 38-42.
11. H. S. Siu, Y. H. Chin, and W. P. Yang, "A note on consensus on dual failure modes," *IEEE Transactions on Parallel and Distributed System*, Vol. 7, 1996, pp. 224-230.
12. S. S. Wang and S. C. Wang, "The consensus problem with dual failure nodes in a cloud computing environmen" *Information Sciences*, Vol. 279, 2014, pp. 213-228.
13. S. S. Wang, K. Q. Yan, and S. C. Wang, "Achieving efficient agreement within a dual-failure cloud-computing environment," *Expert Systems with Applications*, Vol. 38, 2011, pp. 906-915.

Shu-Ching Wang (王淑卿) received the B.S. degree in Computer Science from Feng-Chia University, the M.S. degree in Electrical Engineering from National Chen-Kung University, and Ph.D. degree in Information Engineering from National Chiao-Tung University, Taiwan. Currently, she is a Professor with the Department of Information Management, Chaoyang University of Technology, Taiwan. Her current research interests include distributed computing, cloud computing, and Internet of Things. Wang is one of the corresponding authors.

Shun-Sheng Wang (王順生) received his Ph.D. degree in Engineering from Ohio State University, USA. Currently, he is an Associate Professor with the Department of Industrial Engineering and Management, Chaoyang University of Technology, Taiwan. His current research interests include distributed system, fault tolerant, and cloud computing. Wang is one of the corresponding authors.

Kuo-Qin Yan (嚴國慶) received the B.S. and M.S. degrees in Electrical Engineering from Chung Cheng Institute of Technology and the Ph.D. degree in Computer Science from National Tsing-Hua University, Taiwan. Currently, he is a Professor with the Department of Business Administration, Chaoyang University of Technology, Taiwan. His current research interests include distributed data processing, parallel processing, fault tolerant computing, mobile computing, and ubiquitous computing. Yan is one of the corresponding authors.