Journal of Computers Vol. 31 No. 1, 2020, pp. 57-70
doi:10.3966/199115992020023101005

Reliability of Cluster-based Mobile Cloud Computing Enhancement Using Optimal Byzantine Agreement

Shu-Ching Wang[1], Ya-Jung Lin[1], Kuo-Qin Yan[2], Yao-Te Tsai[3*]

[1] Department of Information Management, Chaoyang University of Technology, Taichung 41349, Taiwan
{scwang; s10614903}@cyut.edu.tw

[2] Department of Business Administration, Chaoyang University of Technology, Taichung 41349, Taiwan
kqyan@cyut.edu.tw

[3] Department of International Business, Feng Chia University, Taichung 40724, Taiwan
yaottsai@fcu.edu.tw

Received 30 April 2018; Revised 15 August 2018; Accepted 5 October 2018

Abstract. Mobile devices are rapidly becoming the major service tools. However, traditional client-server-based mobile service models are unable to meet the increasing demands from mobile users in terms of diversity of services. Cloud computing can enable mobile devices to offload complex operations of mobile applications, which are infeasible on mobile devices alone. Mobile Cloud Computing (MCC) is widely accepted as a concept that can significantly improve the user experience when accessing mobile services. In order to enhance the capability of fault tolerance and ensure network security, it is necessary to provide a stable mobile cloud service environment. Due to the Byzantine Agreement (BA) problem is a fundamental issue in fault-tolerant distributed systems. To enhance the reliability of a cluster-based MCC, it is the first time a protocol OACM (Optimal Agreement protocol for Cluster-based MCC) is proposed herein to solve the BA problem. The protocol makes all fault-free nodes communicate with each other and collect the exchanged messages to decide a common value. Based on the common value, the protocol ensures all fault-free nodes reach agreement without the influence of faulty components. Meanwhile, OACM uses the minimum number of rounds for message exchanges to make all fault-free nodes agree on a common value. The study has also proved the protocol can tolerate the maximum number of faulty components.

Keywords: Byzantine agreement, cluster cloud, distributed system, fault tolerant, mobile cloud computing

1 Introduction

Cloud computing is widely adopted by commercial and military environment to support data storage, on demand computing and dynamic provisioning [1]. Internet web service is currently the main way we access information from fixed or mobile terminals. Some of the information is stored in Internet clouds, where computing, communication and storage are common services provided for Internet users [2]. With the Internet environment applications, current mobile devices have many advanced features such as mobility, communication and sensing capabilities, and can serve as the personal information gateway for mobile users. However, when running complex data mining and storing operations, the integrated solution for the computation, energy and storage limitations of mobile devices relies on cloud-based computing.

The MCC is defined by Mobile Cloud Computing Forum as follows: "Mobile Cloud computing at its simplest refers to an infrastructure where both the data storage and the data processing happen outside of the mobile device. Mobile cloud applications move the computing power and data storage away from

* Corresponding Author

mobile phones and into the cloud, bringing applications and mobile computing not just to smart phone users but a much broader range of mobile subscribers" [3]. MCC would also be based on the basic cloud computing concepts. As discussed by Mei et al. in [4]: there are certain requirements that need to be met in a cloud such as adaptability, scalability, availability and self-awareness. These are also valid requirements for MCC. For example, a mobile computing cloud also needs cognizance of its availability and quality of service, as well as the ability to enable diverse mobile computing entities to dynamically plug themselves in [5].

Therefore, in MCC, a mobile entity can be considered either as a physical mobile device or a mobile computing/storage software agent within a virtualized cloud resource provisioning system [2]. On one hand, the MCC is a development of mobile computing, and an extension to cloud computing. In MCC, the previous mobile device-based intensive computing, data storage and mass information processing have been transferred to clouds; thus, since the requirements of mobile devices in computing capability and resources have been reduced, the developing, running, deploying and mode uses of mobile applications have totally changed [6]. The terminals which people use to access and acquire cloud services are suitable for mobile devices like Smartphone, Tablet and iPad, but are not restricted to fixed devices (such as PC), which reflects the advantages and original intention of cloud computing.

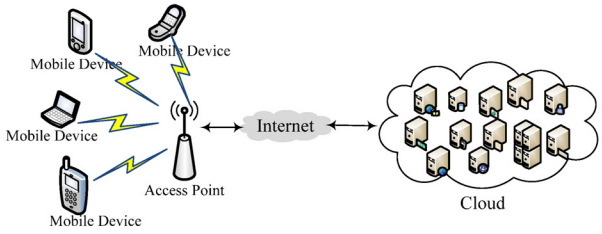

Fig. 1. Mobile cloud architecture [7]

Fig. 1 shows a topology of MCC [7]. Furthermore, under the concept of MCC, the advantages of cloud computing will not only become available for users, but will provide additional functionality to the cloud as well. MCC will help to overcome the limitations of mobile devices, in particular those of processing power and data storage [8-9]. Moreover, one of the fundamental issues with MCC is reliability; through a highly reliable MCC, many cloud computing applications can be provided.

As MCC has become increasingly popular, network topology has trended toward wireless connectivity, thus providing enhanced support for MCC. This technological trend has greatly encouraged distributed system design and support for cloud nodes [10]. The 'cluster' has attracted significant attention recently because it requires less infrastructure, it can be deployed quickly, and it can automatically adapt to changes in topology. Therefore, the structure of a cluster can suit military communication, emergency disaster rescue operations, and law enforcement [10], thanks to the cloud-computing technology of MCC [2, 7]. Therefore, in this study, cluster-based MCC will be the network topology explored.

The security assurance of cloud data management and transfer arises as a key issue. Cloud auditing can only be effective if all operations on the data can be tracked reliably [1]. Therefore, the reliability of cloud nodes is one of the most important aspects in MCC. In order to provide a reliable cluster-based MCC, a mechanism to allow a set of cloud nodes to agree on an agreement value is required. The Byzantine Agreement (BA) problem is one of the most fundamental problems in seeking to reach an agreement value in a distributed system [8-9]. The original BA problem defined by Lamport *et al.* is assumed as follows [9]:

(1) There are n nodes in a synchronous distributed system, where n is a constant and $n \geq 4$.

(2) Each node can communicate with all other nodes through a reliable fully connected network.

(3) One or more of the nodes might fail, so a faulty node may transmit incorrect message(s) to other nodes.

(4) After message exchanges, all fault-free nodes should reach a common agreement, if and only if the number of faulty nodes t is less than one-third of the total number of nodes in the network, or $t \leq (n\text{-}1)/3$.

The solutions define a protocol, which can reach agreement by using the minimum number of rounds for message exchanges to obtain the maximum number of nodes with allowable faulty capability. The problem in the paper is to enable all fault-free cloud nodes to reach agreement underlying an n-nodes cluster-based MCC. The cloud main server (source node) chooses an initial value to start with, and

communicates to all other nodes by exchanging messages. The cloud nodes can reach an agreement if following conditions are satisfied [9]:

(Agreement): All fault-free nodes agree on a common value.

(Validity): If the source node is fault-free, then all fault-free nodes shall agree on the initial value the source node sent.

In previous studies, the BA algorithms were designed in traditional network topology [8-9, 11-15]. A comparison of the studies among the most previous relative researches is given in Table 1. Those works reach Byzantine agreement underlying different topologies respectively, including Broadcasting Network (BCN), Fully Connected Network (FCN), Multicasting Network (MCN), Wireless Sensor Network (WSN), and Cloud Computing environment (CC). All those previous protocols are not suitable for MCC due to the difference of network topology. To enhance fault-tolerance of MCC, it is the first time a protocol OACM (Optimal Agreement protocol for Cluster-based MCC) is proposed in this study to solve the BA problem. The proposed protocol ensures that all fault-free nodes in MCC can reach an agreement value to cope with the influences of the faulty components by using the minimum number of message exchanges, while tolerating the maximum number of faulty components at any time.

Table 1. The comparisons among previous approaches and the proposed protocol OACM

Topology / Results	BCN	FCN	MCN	WSN	CC	MCC
Babaoglu et al. [11]	V					
Lamport et al. [9]		V				
Fischer [8]		V				
Siu et al. [12]		V				
Wang et al. [13]			V			
Yan et al. [15]				V		
Wang et al. [14]					V	
OACM						V

The rest of this paper is organized as follows. Section 2 discusses related works. Then, the proposed protocol OACM is introduced and illustrated in detail in Section 3. For easy understanding, an example of the execution of the proposed protocol is given in Section 4. Section 5 demonstrates the correctness and complexity of the protocol proposed. Finally, Section 6 concludes the study and the future works.

2 Related Work

Currently, the cluster cloud is a more practical kind of cloud computing. A cluster of multiple cloud nodes in a cluster cooperates to achieve some objectives [2]. Fig. 2 shows an example of Amazon's cluster-based cloud.

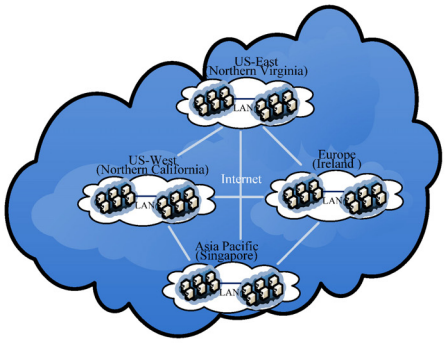

Fig. 2 An example of Amazon's cluster-based cloud

Cluster-based cloud computing consists of a set of loosely or tightly connected cloud nodes that work together so that, in many respects, they can be viewed as a single system. The components of a cloud cluster are usually connected to each other through fast LANs (local area networks) with each cloud node. All cloud nodes of cluster-based cloud computing are usually deployed to offer better performance and

availability compared to a single computer.

Mobile cloud computing is a combination of both technologies from the aspects of mobile computing and cloud computing [16]. The traditional algorithms of distributed, grid, or centralized are still able to continue to develop on this paradigm. Various mobile applications have taken the advantages of MCC, such as mobile commerce, mobile learning, mobile healthcare, and mobile gaming [17]. The typical cluster-based MCC is shown in Fig. 3.

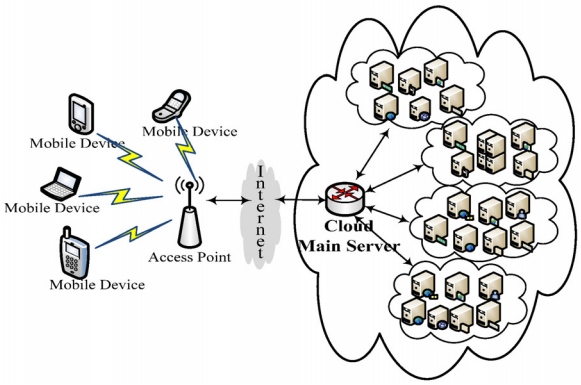

Fig. 3. Cluster-based MCC

In conventional BA problem, many cases are solved on the assumption of node failure in a fail-safe network [18]. The optimal algorithm of solving BA problem requires using the minimum number of rounds for message exchanges to achieve agreement. A protocol to reach agreement in a reliable communication environment of traditional fully connected network was proposed firstly by Lamport et al. [9]. The typical protocol proposed by Fischer [8] can tolerate $f \leq \lfloor (n-1)/3 \rfloor$ faulty nodes in malicious attacks and require σ ($\sigma = f + 1$) round(s) to get enough messages to achieve agreement where n is the total number of nodes in the network.

Most parts of a distributed computing system may not be fully connected. The cluster-based MCC is one of them and differs from the traditional network. So, the pervious protocols of reaching BA are not suitable for the cluster-based MCC. As the result, a new protocol is proposed that can be used to solve the BA problem with malicious faulty nodes in a cluster-based MCC. When all fault-free nodes reach agreement, the fault tolerance capability has been enhanced, even if the communication media may have failed between nodes; the backbone of the system can be used to provide a backup route [19].

3 The Proposed Protocol

In this section, the scenario of the protocol OACM (Optimal Agreement protocol for Cluster-based MCC) is illustrated to solve the BA problem in a cluster-based MCC. Basically, the messages received from non-faulty components will be the same as from other nodes. Based on the same messages, every fault-free node can reach the same agreement value. Thus, the protocol OACM should help the fault-free nodes to remove all influences of faulty components in the messages received from all other nodes.

The assumptions, notations and parameters of the proposed protocol OACM are shown as follows:

- Each node in the network can be identified uniquely.
- Let n be the total number of nodes in the cluster-based MCC.
- Let C be the total number of clusters in the cluster-based MCC, and $C \geq 4$.
- Let x be the cluster identifier, where $1 \leq x \leq C$ and $C \geq 4$.
- Let n_x be the total number of nodes in cluster C_x, $0 \leq x \leq C$. If there are more than $\lceil n_x/2 \rceil$ malicious faulty cloud nodes in C_x, then C_x will be named as a malicious faulty cluster.
- Let T_{FC} be the total number of malicious faulty clusters.
- Let $val(i)$ be the value stored in the vertex i.
- Let σ be the number of rounds executed.
- Let θ be the total number of rounds required.

The purpose of the BA protocol is to enable all fault-free nodes to reach common agreement in the

cluster-based MCC. For this reason, nodes should exchange messages with all other nodes. Each fault-free node receives messages from other nodes by a number of rounds of message exchanges. Afterwards, all fault-free nodes can get enough messages to make a decision value that is called an agreement value or a common value. Thus, all fault-free nodes agree on the same value.

In order to reach agreement, each node should exchange messages with all other nodes. Then, each fault-free node collects enough messages to determine the decision value (the agreement value); all fault-free nodes' agreement values should be identical. Therefore, the nodes executing the proposed protocol should receive messages from other nodes within a predictable period. If the message is not received on time, the message must have been influenced by faulty components.

The first step of the proposed protocol is to determine the number of rounds of messages exchanged so that the number of message exchanges required can be minimized. After determining the required rounds of message exchanges, a BA protocol must execute two phases of work: the message exchange phase and the decision-making phase. The task of the message exchange phase is to collect enough messages from other nodes through the cluster-based MCC. Furthermore, in the decision-making phase, each fault-free node uses the messages received during the phase of message exchanges to determine the common agreement value.

In a cluster-based MCC, the proposed protocol OACM can solve the BA problem of which faulty nodes may send wrong messages to influence the system to reach agreement. Based on the protocol proposed by Fischer [8], OACM can reach common agreement by θ rounds of message exchanges, where $\theta = \lfloor (C-1)/3 \rfloor + 1$ and C is the total number of clusters in the cluster-based MCC, once we treat the clusters of MCC as the nodes of traditional network.

The proposed protocol OACM is organized into two phases: message exchange phase and decision making phase. In the first round of the message exchange phase, the cloud main server sends its initial value to all clusters, and then receiver node stores the received value in the root of its mg-tree. The mg-tree is a tree structure that is used to store the received message from the cloud main server in the message exchange phase [15]. A detail description of mg-tree is illustrated in Appendix A. After the first round of the message exchange phase ($\sigma>1$), each node transmits the value at level $\sigma-1$ in its own mg-tree to all other nodes. At the end of each round, the receiver node applies the function $RMAJ(V)$ to the values received from the same cluster to get a single value. Moreover, each receiver node stores the received messages and the value of function $RMAJ(V)$ in its mg-tree.

After finishing $\lfloor (C-1)/3 \rfloor + 1$ rounds of the message exchange, each node will execute the decision making phase. In order to mitigate the influence of malicious faulty nodes and avoid the repetition of faulty clusters, no cluster name is repeated in any vertex, and each fault-free node must reorganize the mg-tree into a corresponding ic-tree by using the following reorganization rules (A detailed description is presented in Appendix B.):

- The leaves in level $\lfloor (C-1)/3 \rfloor + 1$ of the mg-tree are deleted.
- The vertices with repeated cluster's names are deleted.

Subsequently, all fault-free nodes can use function $VOTE(\mu)$ to remove the influence of malicious faulty clusters and to obtain a common agreement value. It is noteworthy that the function $VOTE$ only counts the non-value "μ" (excluding the last level of the ic-tree) for all vertices at the μ-th level of an ic-tree, where $1 \leq \mu \leq \lfloor (C-1)/3 \rfloor + 1$. Since $VOTE(s)$ is a common value, each fault-free node can mitigate the influence of malicious faulty clusters and agree on the same value, thus reaching agreement. In addition, a detailed definition of the OACM is shown in Fig. 4.

4 An Example of OACM Executed

An example of the execution of OACM is given for easy understanding. The example topology of cluster-based MCC is shown in Fig. 5. There are 21 nodes falling into 7 clusters. C_1 contains n_1 and n_2. C_2 includes n_3, n_4, n_5 and n_6. C_3 has n_7, n_8, n_9 and n_{10}. C_4 is organized by n_{11} and n_{12}. Nodes n_{13} and n_{14} compose C_5. C_6 is comprised by n_{15} and n_{16}. Nodes n_{17}, n_{18}, n_{19}, n_{20} and n_{21} form C_7.

In BA problem with fallible nodes, the worst situation occurs when the source node is no longer honest [8]. Simply, to illustrate the worst case of the example, the source node (let it be n_{cs}) of the cloud main server is supposed in malicious. It means n_{cs} may arbitrarily send different message values (e.g., replicate command [16]) to different clusters. The scenario of the example is described as follows:

OACM (cloud main server n_{cs} with initial value v_s)

Pre-Execute. Computes the number of rounds required $\theta = \lfloor (C-1)/3 \rfloor + 1$, where C is the total number of clusters in the cluster-based MCC.

Message Exchange Phase:

Case $\sigma = 1$, *run*

1. The n_{cs} transmits its initial value v_s to each cluster's nodes.
2. Each receiver node obtains the value and stores it in the root of its mg-tree.

Case $1 < \sigma \leq \theta$, *run*

1. Each node without the n_{cs} transmits the values at level $\sigma - 1$ in its mg-tree to each cluster's nodes.
2. Each receiver node applies *RMAJ* to its received messages and stores the *RMAJ* value in the corresponding vertices at level θ of its mg-tree.

Decision Making Phase:

Step 1. The mg-tree is reorganized into a corresponding ic-tree.

Step 2. After using function *VOTE* on the root s of each node's ic-tree, the common value *VOTE*(s) is obtained.

Function *RMAJ*(*V*)

The majority value of the vector $V_i = [v_1, \ldots, v_{rpx-1}, v_{rpx}]$, if the majority exists.
Otherwise, a default value ϕ is chosen.

Function *VOTE*(μ)

If the μ is a leaf, then outputs *val*(μ).
If the majority value in the set of $\{VOTE(\mu j) | \ 1 \leq j \leq C$ and vertex μj is a child of vertex $\mu\}$ exists
then outputs the majority value
else a default value ϕ is outputted.

Fig. 4. The proposed protocol OACM

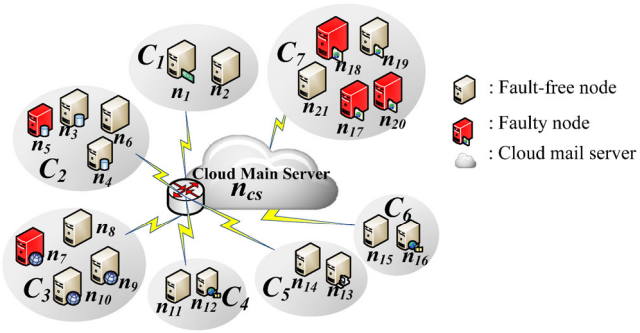

Fig. 5. The initial status of executing OACM

- The messages are sent from the cloud main server (source node) n_{cs} and then start to execute OACM.
- The cloud main server n_{cs} is assumed to be a malicious faulty server.
- n_{cs} sends 1 to all nodes of C_2, C_4, C_5, C_6 and C_7, and sends 0 to all nodes of C_1 and C_3.

Therefore, in order to solve the BA problem among fault-free nodes of this example, OACM requires $\theta = \lfloor (C-1)/3 \rfloor + 1 = \lfloor (7-1)/3 \rfloor + 1 = 3$ rounds of message exchanges, where C = 7. The cloud main server n_{cs} transmits messages to all other nodes in the first round of the message exchange phase. The message obtained by each fault-free node is listed in Fig. 6.

	Level 1(Root *s*)
C_1's fault-free nodes	0
C_2's fault-free nodes	1
C_3's fault-free nodes	0
C_4's fault-free nodes	1
C_5's fault-free nodes	1
C_6's fault-free nodes	1
C_7's fault-free nodes	1

Fig. 6. The mg-tree of each node in the first round

In the second round of message exchange phase, each node transmits the values at the first level in its mg-tree to all other nodes and itself. Subsequently, each receiver node applies *RMAJ*() to its received messages; and stores the received messages and *RMAJ*() values at the corresponding vertices at the second level of its mg-tree. The mg-tree of fault-free node n_1 at the second round in the message exchange phase is shown in Fig. 7. For easy understanding, the message affected by the faulty nodes is denoted in bold and italic.

Level 1	*Level 2*	*Take RMAJ*	
val(*s*)=1	*s1*	0	(0,0)
	s2	1	(1,1,***0***,1)
	s3	0	(0,***0***,0,0)
	s4	1	(1,1)
	s5	1	(1,1)
	s6	1	(1,1)
	s7	0	(***0***,***0***,1,***0***,1)

Fig. 7. The mg-tree of fault-free node n_1 at second round

In the third round of message exchange phase, each node transmits the values at the second level of its mg-tree to all other nodes and itself. Subsequently, each receiver node applies *RMAJ*() to the messages received; and stores the received messages and *RMAJ*() values at the corresponding vertices at the third level of its mg-tree. The mg-tree of fault-free node n_1 at the second round of the message exchange is shown in Fig. 8.

Level 1	*Level 2*	*Level 3*	*Take RMAJ*( )	
s	*s1*	*s11*	0	(0)
0	0	*s12*	0	(0,0,***0***,0)
	(0)	*s13*	0	(0,***1***,0,0)
		s14	0	(0,0)
		s15	0	(0,0)
		s16	0	(0,0)
		s17	1	(***1***,***1***,1,***0***,1)
	s2	*s21*	1	(1)
	1	*s22*	1	(1,1,***1***,1)
	(1,1,1,1)	*s23*	1	(1,***1***,1,1)
		s24	1	(1,1)
		s25	1	(1,1)
		s26	1	(1,1)
		s27	0	(***0***,***0***,1,***0***,1)
	s3	*s31*	0	(0)
	0	*s32*	0	(0,0,***1***,0)
	(0,0,0,0)	*s33*	0	(0,***1***,0,0)
		s34	0	(0,0)
		s35	0	(0,0)
		s36	0	(0,0)
		s37	0	(***0***,***0***,1,***0***,1)
	s4	*s41*	1	(1)
	1	*s42*	1	(1,1,***0***,1)
	(1,1)	*s43*	1	(1,***1***,1,1)
		s44	1	(1,1)
		s45	1	(1,1)
		s46	1	(1,1)
		s47	1	(***1***,***1***,1,***0***,1)
	s5	*s51*	1	(1)
	1	*s52*	1	(1,1,***1***,1)
	(1,1)	*s53*	1	(1,***0***,1,1)
		s54	1	(1,1)
		s55	1	(1,1)
		s56	1	(1,1)
		s57	0	(***0***,***0***,1,***0***,1)
	s6	*s61*	1	(1)
	1	*s62*	1	(1,1,***1***,1)
	(1,1)	*s63*	1	(1,***1***,1,1)
		s64	1	(1,1)
		s65	1	(1,1)
		s66	1	(1,1)
		s67	1	(***1***,***1***,1,***1***,1)
	s7	*s71*	0	(0)
	0	*s72*	1	(1,1,***1***,1)
	(0,0,1,0,1)	*s73*	0	(0,***0***,0,0)
		s74	1	(1,1)
		s75	0	(0,0)
		s76	1	(1,1)
		s77	0	(***0***,***0***,1,***0***,1)

Fig. 8. The final mg-tree of node n_1 after the message exchange phase

After the message exchange phase, the mg-tree of each fault-free node is reorganized to the ic-tree by deleting the vertices with duplicated names. The example ic-tree is shown in Fig. 9.

Level 1	*Level 2*	*Level 3*	*Take RMAJ*	
s	*s1*	~~*s11*~~		
0	0	*s12*	0	(0,0,***0***,0)
		s13	0	(0,***1***,0,0)
		s14	0	(0,0)
		s15	0	(0,0)
		s16	0	(0,0)
		s17	1	(***1***,***1***,1,***0***,1)
	s2	*s21*	1	(1)
	1	~~*s22*~~		
		s23	1	(1,***1***,1,1)
		s24	1	(1,1)
		s25	1	(1,1)
		s26	1	(1,1)
		s27	0	(***0***,***0***,1,***0***,1)
	s3	*s31*	0	(0)
	0	*s32*	0	(0,0,***1***,0)
		~~*s33*~~		
		s34	0	(0,0)
		s35	0	(0,0)
		s36	0	(0,0)
		s37	0	(***0***,***0***,1,***0***,1)
	s4	*s41*	1	(1)
	1	*s42*	1	(1,1,***0***,1)
		s43	1	(1,***1***,1,1)
		~~*s44*~~		
		s45	1	(1,1)
		s46	1	(1,1)
		s47	1	(***1***,***1***,1,***0***,1)
	s5	*s51*	1	(1)
	1	*s52*	1	(1,1,***1***,1)
		s53	1	(1,***0***,1,1)
		s54	1	(1,1)
		~~*s55*~~		
		s56	1	(1,1)
		s57	0	(***0***,***0***,1,***0***,1)
	s6	*s61*	1	(1)
	1	*s62*	1	(1,1,***1***,1)
		s63	1	(1,***1***,1,1)
		s64	1	(1,1)
		s65	1	(1,1)
		~~*s66*~~		
		s67	1	(***1***,***1***,1,***1***,1)
	s7	*s71*	0	(0)
	0	*s72*	1	(1,1,***1***,1)
		s73	0	(0,***0***,0,0)
		s74	1	(1,1)
		s75	0	(0,0)
		s76	1	(1,1)
		~~*s77*~~		

Fig. 9. The ic-tree of node n_1

Finally, the function *VOTE* is applied to the root (the vertex *s*) of each fault-free node's ic-tree, an agreement value 1 can be obtained, as shown in Fig. 10. The decision making phase has been completed. Every fault-free node shall agree on the value *VOTE*(*s*) = 1.

Level 1	*Level 2*	*Level 3*	
s	*s1*	~~*s11*~~	
	VOTE(*s1*)	*s12*	0
	= the majority of	*s13*	0
	(*val*(*s12*), *val*(*s13*), *val*(*s14*), *val*(*s15*), *val*(*s16*), *val*(*s17*))	*s14*	0
VOTE(*s*)	=the majority of (0,0,0,0,0,1)	*s15*	0
= the majority of	=0	*s16*	0
(*VOTE*(*s1*), *VOTE*(*s2*), *VOTE*(*s3*),		*s17*	1
VOTE(*s4*), *VOTE*(*s5*), *VOTE*(*s6*),	*s2*	*s21*	1
VOTE(*s7*))	*VOTE*(*s2*)	~~*s22*~~	
=the majority of (0,1,0,1,1,1, ϕ)	= the majority of	*s23*	1
= 1	(*val*(*s21*), *val*(*s23*), *val*(*s24*), *val*(*s25*), *val*(*s26*), *val*(*s27*))	*s24*	1
	=the majority of (1,1,1,1,1,0)	*s25*	1
	=1	*s26*	1
		s27	0
	s3	*s31*	0
	VOTE(*s3*)	*s32*	0
	= the majority of	~~*s33*~~	
	(*val*(*s31*), *val*(*s32*), *val*(*s34*), *val*(*s35*), *val*(*s36*), *val*(*s37*))	*s34*	0
	=the majority of (0,0,0,0,0,0)	*s35*	0
	=0	*s36*	0
		s37	0
	s4	*s41*	1
	VOTE(*s4*)	*s42*	1
	= the majority of	*s43*	1
	(*val*(*s41*), *val*(*s42*), *val*(*s43*), *val*(*s45*), *val*(*s46*), *val*(*s47*))	~~*s44*~~	
	=the majority of (1,1,1,1,1,1)	*s45*	1
	=1	*s46*	1
		s47	1
	s5	*s51*	1
	VOTE(*s5*)	*s52*	1
	= the majority of	*s53*	1
	(*val*(*s51*), *val*(*s52*), *val*(*s53*), *val*(*s54*), *val*(*s56*), *val*(*s57*))	*s54*	1
	=the majority of (1,1,1,1,1,0)	~~*s55*~~	
	=1	*s56*	1
		s57	0
	s6	*s61*	1
	VOTE(*s6*)	*s62*	1
	= the majority of	*s63*	1
	(*val*(*s61*), *val*(*s62*), *val*(*s63*), *val*(*s64*), *val*(*s65*), *val*(*s67*))	*s64*	1
	=the majority of (1,1,1,1,1,1)	*s65*	1
	=1	~~*s66*~~	
		s67	1
	s7	*s71*	0
	VOTE(*s7*)	*s72*	1
	= the majority of	*s73*	0
	(*val*(*s71*), *val*(*s72*), *val*(*s73*), *val*(*s74*), *val*(*s75*), *val*(*s76*))	*s74*	1
	=the majority of (0,1,0,1,0,1)	*s75*	0
	=ϕ	*s76*	1
		~~*s77*~~	

Fig. 10. The common value *VOTE*(*s*) for fault-free node n_1

5 Correctness and Complexity

In this section, the correctness and complexity of the protocol will be proved by some lemmas and theorems.

5.1 Correctness of OACM

To prove our protocol's correctness, the value reached by all fault-free nodes should be the same whether the source node is fault-free or not. Because the values of exchanged are all stored in the tree structure, the vertex of the tree is called common [8] if each fault-free node has the same value for the vertex. That

means the vertex is common, then the value stored in the vertex of every fault-free node's mg-tree or ic-tree is identical. In other words, if OACM can make the root *s* be common, then all fault-free nodes can reach an agreement by the value of the root VOTE(*s*). Next, we should prove the value of VOTE(*s*) to be the initial value if the source node is fault-free. Thus, the constraints (Agreement) and (Validity) can be rewritten as:

(Agreement'): Root *s* is common, and

(Validity'): VOTE(*s*) = v_s for each fault-free node, if the source node 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 common vertices forms a common frontier [14]. In other words, every fault-free node has identical messages collected in the common frontier if a common frontier exists in a fault-free node's tree structure (mg-tree or ic-tree). Subsequently, using the same majority function to compute the root value of the tree structure, every fault-free node can obtain the same root value because they all have the same collected messages in the common frontier.

To prove OACM can satisfy the constraints (Agreement') and (Validity'), the following two terms need to be defined:

(1) Fault-free vertex: A fault-free vertex is a place to store the values received from the fault-free node or the fault-free cluster.

(2) True value: The stored value of a fault-free vertex is called the true value.

By definition, a correct vertex is one that contains a stored value that is received from the nodes in a fault-free cluster, and a fault-free cluster always transmits the same value to all nodes. Therefore, the fault-free vertices of such an mg-tree are in common. After reorganizing the mg-tree into its corresponding ic-tree by deleting the vertices with repeated cluster names, the values stored in the fault-free vertices of an ic-tree are the same. As a result, all the fault-free vertices of an ic-tree are also in common. Again, by the definition of a fault-free vertex, a common frontier does exist in the ic-tree inasmuch as Agreement' and Validity' are true, regardless of whether the source node is fault-free or faulty if the BA problem has been solved.

Lemma 1: All fault-free vertices of an ic-tree are common.

Proof: After reorganization, no repeatable vertices are in an ic-tree. At the level T_{FC} + 1 or above, the fault-free vertex α has at least $2T_{FC}$ + 1 children among which at least T_{FC} + 1 children are fault-free, where T_{FC} is the total number of malicious faulty clusters. The true value of these T_{FC} + 1 fault-free vertices is in common, and the majority value of vertex α is common. The fault-free vertex α is common in the ic-tree if the level of α is less than T_{FC} + 1. As a result, all fault-free vertices of the ic-tree are common.

Lemma 2: The common frontier exists in the ic-tree.

Proof: There are T_{FC} + 1 vertices along each root-to-leaf path of an ic-tree in which the root is labeled by the source name, and the others are labeled by a sequence of cluster names. Since at most T_{FC} clusters can fail, at least one vertex is fault-free along each root-to-leaf path of the ic-tree. By Lemma 1, the fault-free vertex is common, and the common frontier exists in each fault-free node's ic-tree.

Lemma 3: 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, then α is common. If the height of α is σ, the children of α are all in common by using induction hypothesis with the height of the children at σ-1; then the vertex α is common.

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

Theorem 1: The root of a fault-free node's ic-tree is common.

Proof: By Lemmas 1, 2 and 3 and Corollary 1, the theorem is proved.

Theorem 2: Protocol OACM solves the BA problem in a cluster-based MCC.

Proof: To prove the theorem, it has to show that OACM meets the constraints (Agreement') and (Validity')

(**Agreement**'): Root *s* is common. By Theorem 1, (Agreement') is satisfied.

(**Validity**'): *VOTE*(*s*)=*v* for all fault-free nodes if the initial value of the source is v_s, then $v = v_s$.

Since most of the nodes are correct, they transmit the messages to all the others. The value of correct vertices for all fault-free nodes' mg-tree is *v*. When the mg-tree is reorganized

into an ic-tree, the correct vertices still exist. As a result, each correct vertices of the ic-tree is common (Lemma 1), and its true value is *v*. By Theorem 1, this root is common. The computed value $VOTE(s) = v$ is stored in the root for all fault-free nodes. (Validity') is satisfied.

5.2 Complexity of OACM

The complexity of OACM is evaluated in terms of: (1) the minimum number of rounds for message exchanges and (2) the maximum number of allowable faulty components. Theorems 3, 4 and 5 below will show that the optimal solution is reached.

Theorem 3: OACM requires $T_{FC} + 1$ rounds to solve the BA problem with malicious fault in a cluster-based MCC where $T_{FC} \leq \lfloor (C\text{-}1)/3 \rfloor$ and C is the total number of clusters in the cluster-based MCC.

Proof: Due to the message passing being required in the message exchange phase only, the message exchange phase is a time consuming phase. Fischer [8] pointed out that $t+1$ ($t \leq \lfloor (n\text{-}1)/3 \rfloor$) rounds are the minimum number of rounds to get enough messages to achieve BA. The unit of Fischer is a node [8], but the unit of the cluster-based MCC is a cluster. The number of required rounds of message exchange in the cluster within cluster-based MCC is $T_{FC} + 1 (T_{FC} \leq \lfloor (C\text{-}1)/3 \rfloor)$. Thus, OACM requires $T_{FC} + 1$ rounds, and this number is the minimum.

Theorem 4: The total number of allowable faulty components by OACM is T_{FC} malicious faulty clusters, where $T_{FC} \leq \lfloor (C\text{-}1)/3 \rfloor$ and C is the total number of clusters in the cluster-based MCC.

Proof: Based on the model of Fischer [8], the maximum number of allowable faulty nodes to reach BA underlying a network is f and $f \leq \lfloor (n\text{-}1)/3 \rfloor$. However, the unit of cluster-based MCC is a cluster, so we can treat a node in Fischer [8] as a cluster in cluster-based MCC. Therefore, $f \leq \lfloor (n\text{-}1)/3 \rfloor$ in Fischer [8] implies $T_{FC} \leq \lfloor (C\text{-}1)/3 \rfloor$ in cluster-based MCC. Then, the total number of allowable faulty components by OACM is T_{FC} malicious faulty clusters.

Theorem 5. The number of allowable faulty nodes T_{FN} is the maximum.

Proof: Every fault-free node agrees on a value, which is dominated by most of the nodes in a cluster of MCC. When the number of faulty nodes is greater than a half of all the nodes in a cluster, the cluster is a faulty cluster. For this reason, two cases of fault tolerance are discussed, the best case and the worst case. There is the maximum number of faulty nodes in a MCC, and no more faulty node can be increased, named best case; if a faulty node is increased in any non-faulty, and let the non-faulty cluster be a faulty cluster, named worst case.

[Best case]:

According to the proof of Theorem 4**,** the total number of allowable faulty components by OACM is T_{FC} malicious faulty clusters, where $T_{FC} \leq \lfloor (C\text{-}1)/3 \rfloor$ and C is the total number of clusters in cluster-based MCC. Therefore, in the best case of malicious faulty nodes, let $n_{\max(i)}$ be the number of nodes in i-th maximum cluster. The number of malicious faulty nodes is $\sum_{i=1}^{T_{FC}} n_{\max(i)}$. An additional number of malicious faulty nodes $\sum_{j=T_{FC}+1}^{C} (n_{max(j)}/2 - 1)$ cannot influence the network, and the number of malicious faulty nodes cannot be increased. If the number of malicious faulty nodes is increased, the assumption of $T_{FC} \leq \lfloor (C\text{-}1)/3 \rfloor$ is contradicted. Therefore, the number of allowable malicious faulty nodes can be written as $T_{FN} = \sum_{i=1}^{T_{FC}} n_{\max(i)} + \sum_{j=T_{FC}+1}^{C} (n_{max(j)}/2 - 1)$.

[Worst case]:

In the worst case of malicious faulty nodes, let $n_{\min(i)}$ be the number of nodes in the i-th minimum cluster. The number of nodes in the malicious faulty cluster is $\sum_{i=1}^{T_{FC}} \lceil n_{\min(i)}/2 \rceil$. An additional number of malicious faulty nodes $\lceil n_{\min(T_{FC+1})}/2 \rceil - 1$ cannot influence the

network. Nevertheless, if a malicious faulty node is increased in $\min(T_{FC+1})^{th}$ cluster, then a malicious faulty cluster is increased, and the assumption of $T_{FC} \leq \lfloor (C\text{-}1)/3 \rfloor$ is contradicted. Thus, the number of allowable malicious faulty nodes can be written as $T_{FN} = \sum_{i=1}^{T_{FC}} \lceil n_{\min(i)} / 2 \rceil + (\lceil n_{\min(T_{FC+1})} / 2 \rceil - 1)$.

As a result, OACM requires the minimum number of rounds for message exchange and tolerates a maximum number of faulty components to reach a common agreement. The optimality of the protocol is proved.

6 Conclusion

In this study, the reliable Byzantine Agreement problem was redefined by the OACM protocol in an MCC paradigm. The proposed protocol ensures that all fault-free nodes in the network can agree on a common value to cope with the influences of the faulty nodes. In the meanwhile, the protocol requires the minimum number of message exchanges and can tolerate the maximum number of faulty nodes at any time. Our protocol is the first time to visit the Byzantine Agreement problem under the paradigm MCC. To sum up, the OACM is an optimal protocol with the following features: solving the BA problem in a cluster-based MCC, allowing the join nodes to reach the same agreement value, by costing the minimum number of rounds of message exchanges, and increasing the fault tolerance capability by allowing for malicious faulty nodes.

Furthermore, in a generalized case, the fallible components are not only nodes, but also communication media. Protocol OACM may be extended in the future to reach BA in a generalized case underlying the topology of the cluster-based MCC.

Acknowledgments

This work was supported in part by the Ministry of Science and Technology MOST 107-2221-E-324-005-MY3.

References

[1] X. Liang, S. Shetty, D. Tosh, C. Kamhoua, K. Kwiat, L. Njilla, Provchain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability, in: Proc. 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2017.

[2] D. Huang, T. Xing, H. Wu, Mobile cloud computing service models: a user-centric approach, IEEE Journals & Magazines 27(5)(2013) 6-11.

[3] G. Mukesh, S. Sukhwinder, Mobile cloud computing, International Journal of Enhanced Research in Science Technology & Engineering 3(4)(2014) 517-521.

[4] L. Mei, W. Chan, T. Tse, A tale of clouds: paradigm comparisons and some thoughts on research issues, in: Proc. Asia-Pacific Services Computing Conference, 2008.

[5] F. Niroshinie, W.L. Seng, R. Wenny, Mobile cloud computing: a survey, Future Generation Computer Systems 29(1)(2013) 84-106.

[6] T. Taleb, S. Dutta, A. Ksentini, M. Iqbal, H. Flinck, Mobile edge computing potential in making cities smarter, IEEE Communications Magazine 55(3)(2017) 38-43.

[7] A. Khan, M. Othman, S. Madani, S. Khan, A survey of mobile cloud computing application models, IEEE Communications Surveys & Tutorials 16(1)(2014) 393-413.

[8] M. Fischer, The consensus problem in unreliable distributed systems (a brief survey), Lecture Notes Computer Science 158(2002) 127-140.

[9] L. Lamport, R. Shostak, M. Pease, The Byzantine general problem, ACM Transactions on Programming Language and Systems 4(3)(1982) 382-401.

[10] K. Gai, M. Qiu, H. Zhao, L. Tao, Z. Zong, Dynamic energy-aware cloudlet-based mobile cloud computing model for green computing, Journal of Network and Computer Applications 59(2016) 46-54.

[11] O. Babaoglu, D. Rogério, Streets of Byzantium: Network architectures for fast reliable broadcasts, IEEE Transactions on Software Engineering 9(1985) 546-554.

[12] H.S. Siu, Y.H. Chin, W.P. Yang, A note on consensus on dual failure modes, IEEE Trans. Parallel Distrib. Syst. 7(3)(1996) 225-230.

[13] S.C. Wang, K.Q. Yan, C.F. Cheng, Efficient multicasting agreement protocol, Computer Standards & Interfaces 26(2)(2004) 93-111.

[14] S.C. Wang, S.S. Wang, K.Q. Yan, Reaching optimal interactive consistency in a fallible cloud computing environment, Journal of Information Science and Engineering 34(1)(2018) 205-223.

[15] K.Q. Yan, S.C. Wang, C.S. Peng, S.S. Wang, Optimal malicious agreement protocol for cluster-based wireless sensor networks, ScienceAsia Journal 40S (2014) 8-15.

[16] C.V. Raja, K. Chitra, M. Jonafark, A survey on mobile cloud computing, International Journal of Scientific Research in Computer Science 3(3)(2018) 2096-2100.

[17] H.T. Dinh, C. Lee, D. Niyato, P. Wang, A survey of mobile cloud computing: architecture, applications, and approaches, Wireless Communications And Mobile Computing 13(18)(2013) 1587-1611.

[18] M. Sookhak, F.R. Yu, M.K. Khan, Y. Xiang, R. Buyya, Attribute-based data access control in mobile cloud computing: taxonomy and open issues, Future Generation Computer Systems 72(2017), 273-287.

[19] C.H. Yang, C.W. Wang, Routing protocols for improving survivability of scatternet routes, International Journal of Emerging Trends & Technology in Computer Science 6(3)(2017), 21-26.

Appendix A: The Message-Gathering Tree (mg-tree)

Fig. 8 shows an mg-tree for a cluster-based MCC in Fig. 5. Each fault-free node maintains such an mg-tree during the execution of OACM. In the first round, the source (node n_s) multicasts its initial value to each cluster's nodes. When fault-free node receives the value sent from the source, it stores the received value, denoted as *val*(*s*), at the root of its mg-tree as shown in Fig. 6. In the second round, each node multicasts the value stored in the root of the mg-tree to each cluster's nodes (except the source node). If the node n_1 sends message *val*(*s)* to the nodes in cluster C_4, then the nodes in cluster C_4 stores the received value from node n_1, denoted as *val*(*s4*), in vertex *s4* of its mg-tree, which is shown in Fig. 7. Similarly, if the nodes in cluster C_2 sends messages *val*(*s1*) to the nodes in cluster C_1, then the value is denoted as *val*(*s12*) and stored in vertex *s12* of node n_1's mg-tree in the third round, which is illustrated in Fig. 8. The *val*(αi) tells that the message is sent to a series of receivers, denoted as α, and the nodes in cluster C_i are the latest receivers. For instance, the message *val*(*s1…4*), stored in the vertex *s1…4* of an mg-tree, which implies that the message just received was sent through the source, the nodes in cluster C_1, the nodes in cluster C_4 (the nodes in cluster C_4 is the latest nodes to pass the message). In addition, it is denoted as *val*($\alpha 4$). When the message is transmitted through the nodes in cluster more than once, the name of the cluster will be repeated correspondingly. For example, *val*(*s11*), stored in vertex *s11* of Fig. 8, indicates that the message is sent from *s* to the nodes in cluster C_1, then to the nodes in cluster C_1 again.

In summary, the root of an mg-tree is always named *s* to denote that the stored message is sent from the source node in the first round, and the vertex of an mg-tree is labelled by a list of cluster names. The cluster name list contains the names of the clusters through which the stored message was transferred.

Appendix B: The Information-Collecting Tree (ic-tree)

An ic-tree is reorganized from a corresponding mg-tree by removing the vertices with repeated cluster names in order to reduce the influence from a faulty node repeatedly in an ic-tree. Fig. 9 is an example of reorganizing an mg-tree to an ic-tree by deleting the repeated cluster names of mg-tree.