IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 10, 2018, accepted July 19, 2018, date of publication July 24, 2018, date of current version September 5, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2859351

An Agreement Under Early Stopping and Fault Diagnosis Protocol in a Cloud Computing Environment

MAO-LUN CHIANG[1], CHIN-LING CHEN[2,3], AND HUI-CHING HSIEH[4]

[1]Department of Information and Communication Engineering, Chaoyang University of Technology, Taichung 41349, Taiwan
[2]Department of Computer Science and Information Engineering, Chaoyang University of Technology, Taichung 41349, Taiwan
[3]School of Information Engineering, Changchun Sci-Tech University, Changchun 130600, China
[4]Department of Information Communication, Hsing Wu University, New Taipei 24452, Taiwan

Corresponding author: Chin-Ling Chen (e-mail: clc@mail.cyut.edu.tw)

ABSTRACT With its explosive development, the Internet has come to offer an increasing multitude of online service applications. In order to better facilitate such services, a cloud computing environment, composed of a large number of processors and memories, high-speed networks, and various applications, has been developed, and continues to grow, providing convenient and quick network services. In this cloud computing environment, each server processor in the environment must cooperate with other processors to satisfy various user demands. As a result, the issue of fault-tolerance needs to be revised in order to ensure the reliability of cloud computing environments. One of the most important issue of fault-tolerance is the Byzantine agreement (BA), which requires that, even if some components are damaged, a set of fault-free service processors are able to agree on a common value. Furthermore, the faulty service processors must be detected and eliminated. Therefore, a fault diagnosis agreement (FDA) issue for such environments must also be revised simultaneously. The goal of FDA is to enable each fault-free service processor to detect/locate a common set of faulty service processors. However, a lot of messages need to be collected to solve the BA and FDA problem in the previous works. Thus, this paper also uses the concept of an early stopping protocol (ESP) to allow its participants to obtain common values early, during different rounds. Furthermore, the result of ESP can then be used to detect/locate the maximum number of faulty service processors with dual failure modes efficiently, using the minimum number of rounds. As a result, the early diagnosis cloud agreement protocol can be proposed to solve the BA, ESP, and FDA problems simultaneously to provide greater computing abilities by enhancing the reliability of a cloud computing environment.

INDEX TERMS Diagnosis, early stopping, eventual byzantine agreement, fault tolerance.

I. INTRODUCTION

Cloud computing [5], [13], [15] evolved from grid computing [28] and distributed systems [10], [11]. The goal of cloud computing is to enhance the next generation of data centers and to offer users leased services based on quality of service (QoS) [24], [39]. Cloud services fall into three categories: Infrastructure as a Service (IaaS), Platform as a Service (PaaS)[14], [26] and Software as a Service (SaaS) [27].

IaaS services include servers, storage spaces, virtual devices and other core infrastructures, enabling users to store, execute and compute on hardware resources provided by the vender via virtual technologies. PaaS services include databases, network servers and development tools, and allow users to develop a number of services via this platform. SaaS services involve software developed by the cloud provider which allows users to utilize services without installing any programs. However, if components of the cloud computing environment fail, then cloud computing services will be terminated. Cloud services thus need to ensure that corresponding activities continue to function, and enhance the reliability of the cloud computing environment, even if some component breaks down. In addition, faulty service processors must be identified and removed to enhance the overall performance of the entire system. Therefore, the traditional Byzantine Agreement (BA) [18], [21], [22], [31]–[38] and Fault Diagnosis Agreement (FDA) [17], [30], [35] must be revised to ensure the reliability of cloud computing environments.

The BA problem was presented by Lamport in 1982 [21], and was solved to make n processors agree on a common value, even if t faulty processors exist (t represents the

number of faulty processors and $t \leq \lfloor (n-1)/3 \rfloor$) in a distributed system.

In addition, the goal of the FDA problem is to have each fault-free processor detect/locate a faulty component by using the minimal number of message exchanges. However, previous protocols require a large number of messages to achieve an agreement and detect/locate faulty service components in a cloud computing environment. Thus, this paper revises the related Early Stopping Problem (ESP) [1], [9], [20] in order to reduce the number of message exchanges. The result is an efficient and suitable protocol, EDCA (Early Diagnosis Cloud Agreement) for improving the reliability of a cloud computing environment by detecting/locating a maximal number of faulty components under a dual failure mode using minimal message exchange rounds.

The remainder of this article is arranged as follows. Section 2 describes previous work and underlying assumptions made in this study. Section 3 gives a detailed description of the proposed protocol. Section 4 provides examples of the proposed protocol. Section 5 demonstrates the correctness and complexity of the proposed protocol, and conclusions are offered in Section 6.

II. PREVIOUS WORK AND UNDERLYING ASSUMPTIONS

In general, cloud computing is a novel concept of a decentralized system, and is an environment with large-scale computing and scalable IT-related capabilities. It can provide a large number of applications to multiple users using network communication. The cloud providers serve various applications and general purpose computing infrastructures by using virtualizations of those infrastructures for different customers. The more popular examples are Google [14], IBM Blue Cloud [19], and Amazon [3].

The Google application, offers free storage and powerful computing capacity, as in Gmail and YouTube [2]. In addition, users can use SDKs and APIs to build web applications using the Google App Engine in a developed platform.

In order to provide virtual servers, computing and storage services, current cloud providers set up data centers in different locations, as in the Cloud Exchange (CEx) shown in Figure 1 [5]. In other words, the cloud service providers integrate available resources to the cloud, the cloud coordinators allocate the resources on the CEx through categories and arrangements, and the user negotiates with cloud coordinators to acquire various services on demand.

However, the cloud manager/coordinator dispatches user-required resources to processor clusters, and each service processor must be able to cooperate with others in its cluster, even if some components are faulty. Therefore, reliability [6], [23] is an important issue of cloud computing environment.

This study focuses on the Byzantine Agreement (BA) [21], [22], [29], which is the most important element of reliability. It allows each fault-free processor to achieve an agreement for reliable computing [10], [11], [21], [31], [32] in an environment where some processors may be faulty.

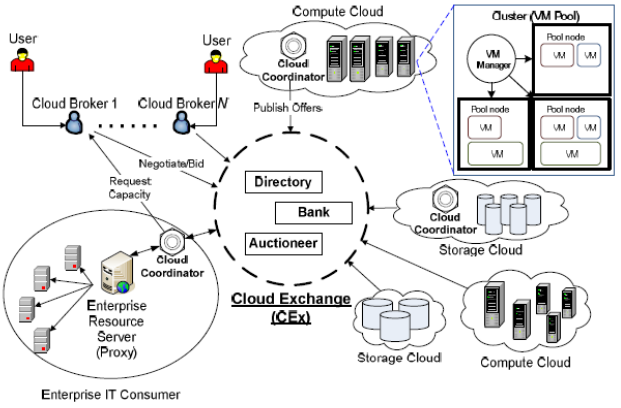

FIGURE 1. Cloud service architecture.

Examples of applications that emphasize this fact include commitment problems in distributed database systems [8], [10]–[12], and clock synchronization problems [8], [12]. Therefore, the BA problem is most applicable for discussion and revision for application in cloud computing environments to enhance system reliability. Various proposed BA protocols toned to meet the following requirements [6], [10], [11], [18], [21], [31], [32], [37], [38]:

Agreement: All fault-free processors agree on a common value v.

Validity: If the initial value of the source is v_s and the source is fault-free, then all fault-free processors shall agree on the value v_s; i.e., $v = v_s$.

There are two types of processor fault symptom: dormant faults, and arbitrary faults [12]. Dormant faults include missed and broken messages, and are easily detected. Malicious faults, however, are more difficult to address due to their unpredictable and damaging behavior. Fischer and Lynch [12] found that even if only one dormant faulty processor exists, agreement cannot be achieved in an asynchronous network. Therefore, the Byzantine Agreement (BA) problem must be considered under a synchronous network in which the processing boundary and the communication delays of fault-free components are finite [12].

In this study, once BA is achieved, the exchanged messages can be collected and used to detect/locate faulty components. For this reason, the Fault Diagnosis Agreement (FDA) problem [7], [17], [35] must also be reviewed. The FDA issue is closely related to BA problem. The main approach of FDA is to collect messages that have accumulated in a round of message exchange [22] as evidence for detecting/locating faulty processors. This is because faulty processors may exhibit the symptoms of a modified message, and those symptoms can be used to detect/locate faulty processors. Once the FDA is implemented, the performance and integrity of a distributed network can be guaranteed. Besides, the proposed FDA protocol must meet the following conditions:

**Agreement:* All fault-free processors can identify the common set of faulty processors.

Fairness: No fault-free component is incorrectly detected as faulty by any fault-free processor.

In previous FDA studies, Hsiao *et al.* [17] proposed an evidence-based FDA protocol, FDAMIX, to solve the FDA problem under dual failure mode (including arbitrary and dormant faults). FDAMIX uses the messages received in the BA protocol GPBA [31] designed for dual faults as evidence for detecting/locating faulty components.

However, the FDAMIX protocol still requires $f_a + 2$ ($f_a \leq \lfloor(n-1)/3\rfloor$, where f_a is the number of processors with malicious faults) rounds of message exchange, because the GPBA requires $f_a + 1$ rounds of message exchange to solve the BA problem, even if the number of faulty processors is less than f_a. Since message passing is a time-consuming phase, the number of messages increases the load of the protocol. Therefore, it is unreasonable and inefficient in a distributed system. Based the above, the previous EFDA [7], [9] is proposed to detect/locate faulty components early in a distributed network with dual failure of processors. However, EFDA still requires a large number of messages in a cloud computing environment. This paper therefore combines the concept of early stopping and fault diagnosis to achieve the goal of BA and FDA efficiently and quickly in a cloud computing environment.

In general, BA protocols [1], [9] require each fault-free processor to achieve agreement and stop the message exchange in the same round when the number of tolerable arbitrary processor faults is (f_a) in a distributed system. This kind of agreement is called Immediate Byzantine Agreement (IBA) [9]. Dolev [4] stated that IBA protocols cannot be achieved for $n$processors with at most f_a faulty processors within f_a or fewer rounds. It is unreasonable to require $f_a + 1$ rounds of message exchange to reach a common value when no faulty processor exists in the system, or the number of faulty processors is less than f_a. Therefore, an improved protocol, the Eventual Byzantine Agreement (EBA) protocol [9], [20] is invoked to solve this ESP (Early Stopping Problem). This protocol allows each processor to stop during rounds when a sufficient number of messages have been collected, or $f_r < f_m$ (f_r: the real number of processors with arbitrary faults) to achieve agreement early. The EBA protocol is more efficient and reasonable than the IBA protocol. Based on the EBA protocol, the proposed lower bound of rounds is $min\{f_r + 2, f_a + 1\}$ in this paper.

Based on the above, this study proposes a novel protocol, Early Diagnosis Cloud Agreement (EDCA), to solve the BA, ESP, and FDA problems simultaneously. In the EDCA protocol, each fault-free service processor can reach a common value using less message exchange rounds than previous methods [4], [20]. In addition, the EDCA protocol can tolerate the maximum number of faulty processors and detect/locate the maximum number of faulty processors under dual failure modes using the minimum number of message exchanges in a cloud computing environment.

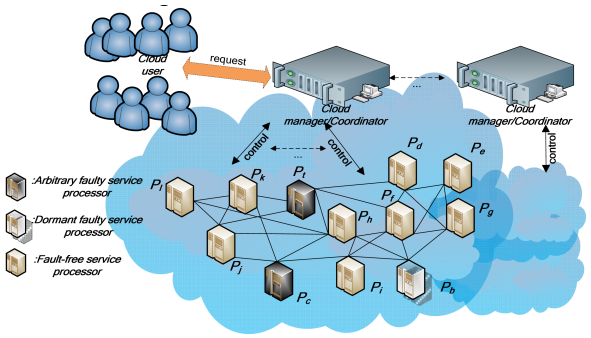

FIGURE 2. A cloud computing architecture with faulty service processors.

III. EARLY DIAGNOSIS CLOUD AGREEMENT (EDCA) PROTOCOL

As demonstrated by the Cloud Service Architecture shown in Figure 1, each service processor in a cloud computing environment is allocated to different places, and must be able to cooperate with other processors to provide greater computing abilities, even if some components are faulty. As a result, a corresponding cloud computing architecture is proposed in Figure 2. Subsequently, some parameters are set as follows:

(1). n: The total number of service processors in the cloud computing environment.

(2). $v(s)$: The value of the initial service processor.

(3). V_k: The vector in service processor P_k.

(4). MAJv_i: The majority value of service processor P_i.

(5). If a service processor does not receive any value, the value "λ" will be stored.

(6). c: The connectivity of a synchronous network. Based on the Menger theorem [3], there are at least c disjoint paths between processor P_a and P_b when the connectivity of the network is c.

(7). Φ : The default value, and $\Phi \in \{0, 1\}$.

(8). f_a: The number of service processors with arbitrary faults.

(9). f_d: The number of service processors with dormant faults.

(10). f_r: The real number of service processors with arbitrary faults.

(11). T_i: An information collecting tree of service processor P_i.

(12). $v(T_j^k)$: The value of T_j in level k.

(13). r: The required rounds of message exchange, $r = min\{f_r + 2, f_a + 1\}$.

(14). r_c: The current round of message exchange.

(15). Num_f_k: Some service processors suspect that P_k is a faulty processor and the Num_f_k is used to accumulate it.

Δv_i^k: This value determines whether the number of received messages is sufficient to reach a common value in service processor i during round k.

This study proposes the Early Diagnosis Cloud Agreement (EDCA) protocol to solve the BA, ESP and FDA problems to provide greater computing abilities by enhancing the reliability of a cloud computing environment. There are three parts

of the EDCA: The message exchange phase, the decision-making phase, and the fault diagnosis phase. Additionally, the *early stopping function* is invoked during the message exchange phase to determine whether the number of received messages is sufficient to achieve agreement. The details of EDCA are shown in Figure 3.

At the beginning of our protocol, the required rounds (r) of message exchange are $min\{f_r + 2, f_a + 1\}$ based on the results of [9] and [20]. In general, the user first sends the service requirements to the cloud managers/coordinators, and then the managers/coordinators divide those service requirements and dispatch them to service processors using a control message. The corresponding service processors must be able to cooperate with each other to provide greater computing abilities in the message exchange phase. As a result, each service processor needs to broadcast its initial values to the other processors until round r. During this phase, an *ic-tree* (information collecting tree; T_i) [31], [34], [37] is constructed to store the received messages. This structure allows convenient collection of a majority value from the vertices for all service processors. The vertices of the *ic-tree* are marked with a list of service processor names. The list of service processor names contains the name of the processors from which stored messages are transmitted.

Furthermore, each service processor can execute the early stopping function to determine whether the protocol can be stopped when $r > 2$ during the message exchange phase to reduce the number of messages exchanged. This is because the number of messages in one round is insufficient to detect/locate faulty service processors in the fault diagnosis phase. Therefore, the early stopping function is only to be invoked when $r > 2$ is satisfied.

In addition, the concept of [18] and [23] is used to solve dual failures with service processors in the early stopping function under dual failure modes. However, in this paper, r_c is redefined as the current message exchange round. In the beginning of this phase, the MAJ function is applied to the *ic-tree* of each processor P_i $(1 \le i \le n)$ to obtain the MAJ(v_i). Subsequently, the Δv_i^k are accumulated when the vertices of the *ic-tree* of each processor are equal to the MAJ(v_i). Similarly, the range of Δv_i^k differs from that in previous works [6], [18], [23], and is revised to $1 \le k \le f_a + 1$ under dual failure modes. The protocol can be allowed to stop early if the following improvement constraint can be satisfied:

$$\Delta v_i^k > (f_a - (r_c - 1)) + \frac{(n - (r_c - 1) - \lambda)}{2}$$

In the following phase, the decision-making phase, each fault-free service processor can retrieve a common value by applying the voting function VOTE, shown in Figure 3, to the root of an *ic-tree*. Namely, the collected messages are enough to make each fault-free service processor reach a common value. Subsequently, the collected messages can be used to the last phase, the fault diagnosis phase. This phase differs from that in [7], and is invoked to detect/locate faulty components. This is because rules *fdr2* and *fdr3* can reduce the

EDCA

Preprocessing:
Calculate the total required number of rounds $r = min\{f_t+2, f_a+1\}$;

Message Exchange Phase:

$r = 1$, do:
1) The initial service processor broadcasts its initial value v_s to all service processors (including itself) to cooperate with each other to provide greater computing abilities.
2) Each service processor P_i stores the received message to its vertex.

$r = 2$, do:
1) Each service processor broadcasts its vertex, and then receives a column vertex broadcasted by other processors, and constructs an *ic-tree* (T_i) for each round.
2) Run the early stopping function.

For $r > 2$, do:
1) Each service processor broadcasts its T_i to other service processors, and stores this value to level r of its T_i.
2) Run the early stopping function.

Decision-making Phase:
1) The VOTE function is applied to each service processor's T_i. Subsequently, the common value can be retrieved from the root of T_i.

Fault Diagnosis Phase: Let P_h be a fault-free processor, $(1 \le h \le n)$, which can detect processor P_k $(1 \le k \le n)$ as a faulty processor if:

Fault diagnosis rule 1 (*fdr1*):
P_h receives λ from P_k (or no message is received from P_k) and the number of copies from P_k is greater than $\lfloor (n\text{-}1\text{-}f_a)/3 \rfloor$; then processor P_k is a dormant faulty service processor.

Fault diagnosis rule 2 (*fdr2*):
For each T_i
If $v(\alpha k) \neq$ MAJ(v_i), then $Num\_f_k + 1$
Compute the number of $|Num\_f_k|$ and compare with f_a.
If $|Num\_f_k| > f_a$, then service processor P_k is an arbitrary faulty service processor.

Fault diagnosis rule 3 (*fdr3*):
If $|Num\_f_k| < f_a$, then scan T_i to compute $|Num\_f_\alpha|$
If $|Num\_f_\alpha| > f_a$, then processor P_α is an arbitrary faulty service processor.

Fault diagnosis rule 4 (*fdr4*):
Compute T_i^{k-1} by taking the majority value on each T_i^k $(i \not\subset f_a)$ and compare with other T_j^{k-1}. If $v(T_i^{k-1}) \neq v(T_j^{k-1})$ $(i \neq j)$ and $|v(T_i^{k-1}) \neq v(T_j^{k-1})| > f_a$, then the initial service processor is an arbitrary faulty service processor.

Early Stopping Function:
Step 1: Use the MAJ function on T_i of each processor and obtain the majority value MAJ(v_i).
Step 2: Compute the number of values Δv_i^k that is equal to MAJ(v_i).
Step 3: If the value $\Delta v_i^k > (f_a - (r_c - 1)) + \frac{(n - (r_c - 1) - \lambda)}{2}$
then the message exchange is stopped.
else go to next round of *message exchange phase*.

MAJ(α) Function:
MAJ(α)=
1. The majority value in the set of $\{v(\alpha i) | 1 \le i \le n\}$, if such a majority value exists.
2. The complement value of $v(\alpha)$, denoted as $\neg v(\alpha)$, is chosen, otherwise.

VOTE Function:
VOTE(α)=
1. $v(\alpha)$, if α is a leaf and $1 \le \alpha \le n$.
2. The majority value in the set of $\{\text{VOTE}(\alpha i) | 1 \le i \le n$, and vertex αi is a child of vertex $\alpha\}$, if such majority values exists.
3. A default value ϕ is chosen.

FIGURE 3. The proposed edca protocol.

comparison times by comparing $|Num\_f_k| > f_a$ only to find arbitrary faulty service processors. Rule *fdr4* is also corrected to efficiently distinguish an initial service processor from all service processors. As a result, the dormant faulty components can be detected/located by diagnosis rule *fdr1* since the transmitted message was encoded by the Non-Return-to-Zero code or the Manchester code [10], [16]. Therefore, messages sent from dormant faulty components can be replaced by λ and can be detected in each message exchange round.

The diagnosis rules *fdr2*, *fdr3* and *fdr4* can be applied to detect/locate arbitrary faulty service processors in the fault diagnosis phase. In general, processor P_k can be identified as an arbitrary faulty processor (Num_f_k) when $v(\alpha k) \neq \text{MAJ}(v_i)$ and $|Num\_f_k| > f_a$. Rule *fdr4* is also used to identify whether the initial service processor is fault-free by taking a majority value on T_i^k. Based on the $n > \lfloor (n-1)/3 \rfloor + 2f_a + f_d$ constraint, the initial service processor can be identified as an arbitrary faulty initial service processor when $|v(T_i^{k-1}) \neq v(T_j^{k-1})| > f_a$. Finally, only $min\{f_r + 2, f_a + 1\}$ message exchange rounds are necessary to ensure all fault-free service processors achieve a common value. As a result, the proposed protocol, EDCA, is more suitable and efficient than previous methods like the FDAMIX protocol, which still requires $f_a + 2$ ($f_a \leq \lfloor (n-1)/3 \rfloor$) rounds to solve the BA problem. An example and the proof of the EDCA protocol are shown in Sections 4 and 5.

IV. EXAMPLE OF EXECUTING EDCA

This section gives an example in Figure 2 to illustrate how the EDCA protocol is implemented. It is assumed that the service processors P_t and P_c are arbitrary service processors, and that processor P_t is also an arbitrary initial service processor in a 12-processor cluster of a cloud computing environment. Therefore, P_t is invoked to cooperate with other processors to actively provide greater computing abilities. Furthermore, service processor P_b is assumed to be a dormant faulty service processor. However, the results of the fault-free service processors are the focus of this protocol, so this example only shows the results of fault-free service processors to meet the requirements of BA and FDA.

The first round of the message exchange phase in the protocol begins with the arbitrary initial service processor P_t sending different values 0, 0, 0, 0, 0, 1, 1, 1, 1, 1 and 1 to service processors P_b, P_c, P_d, P_e, P_f, P_g, P_h, P_i, P_j, P_k and P_l, respectively, as shown in Table 1. Next, each service processor exchanges its received value with other service processors in round 2. Unfortunately, the behaviors of arbitrary faulty service processors are assumed to be smart, thus the arbitrary service processor P_c may send differing values to other service processors. The worst case here differs from that in previous works [34], [36], [37], [39], as shown in Table 2.

After the second round of the message exchange phase, each processor can construct an *ic-tree* (T_i) level by level by the received value from P_t and a column vector broadcasted by other processors. As shown in Figure 4, P_d is used to explain the example, and records the received values from P_t in round 1 and other service processors in round 2 into Level 1 and Level 2 of *ic-trees*. In the third round of the message exchange phase, each service processor exchanges the received values of round 2 to the other processors, and stores the value in Level 3 of their *ic-trees*. The results of P_d in round 3 are shown in Figures 5(A)∼(K).

Level 1	Level 2	Level 3
v(t)	v(tb) λ	v(tbb) λ
0		v(tbc) λ
		v(tbd) λ
		v(tbe) λ
		v(tbf) λ
		v(tbg) λ
		v(tbh) λ
		v(tbi) λ
		v(tbj) λ
		v(tbk) λ
		v(tbl) λ

FIGURE 4. The received value of P_d in round 2.

TABLE 1. The transfer values of arbitrary source processor P_T round 1.

	P_b	P_c	P_d	P_e	P_f	P_g	P_h	P_i	P_j	P_k	P_l
P_t	0	0	0	0	0	1	1	1	1	1	1

TABLE 2. The transfer values of arbitrary processor P_c round 2.

	P_b	P_d	P_e	P_f	P_g	P_h	P_i	P_j	P_k	P_l
P_c	1	0	1	0	1	0	1	0	1	0

After round 2 of message exchange, the early stopping function is used to determine whether a sufficient number of messages has been collected. However, Δv_d^2 is not greater than the limited bound in Formula (1), thus the following process message exchange must be executed. In Formula (2), all service processors can stop the message exchange process in round 3, rather than round 4 ($\sigma = \lfloor (12-1-1)/3 \rfloor + 1 = 4$ [9], [20]), because the early stopping condition in EDCA is satisfied.

$$\begin{aligned} R_2 : \Delta v_d^2 &= \{(f_a - (r-1)) + \frac{(n-(r-1)-\lambda)}{2} \\ &= (3-(2-1)) + \frac{(12-(2-1)-1)}{2} = 7\} \quad (1) \\ R_3 : \Delta v_{d\sim f}^3 &> \{(f_a - (r-1)) + \frac{(n-(r-1)-\lambda)}{2} \\ &= (3-(3-1)) + \frac{(12-(3-1)-1)}{2} = 5.5\} \\ \Delta v_{g\sim l}^3 &> \{(f_a - (r-1)) + \frac{(n-(r-1)-\lambda)}{2} \\ &= (3-(3-1)) + \frac{(12-(3-1)-1)}{2} = 5.5\} \quad (2) \end{aligned}$$

For example:

Δv_d^2 of $val(tc)$'s sub-tree in level $2 = 6 < 7$ (The EAFD protocol cannot stop in R_2).

Δv_d^3 of $val(tc)$'s sub-tree in level $3 = 6 > 5.5$ (The EAFD protocol can be stopped in R_3).

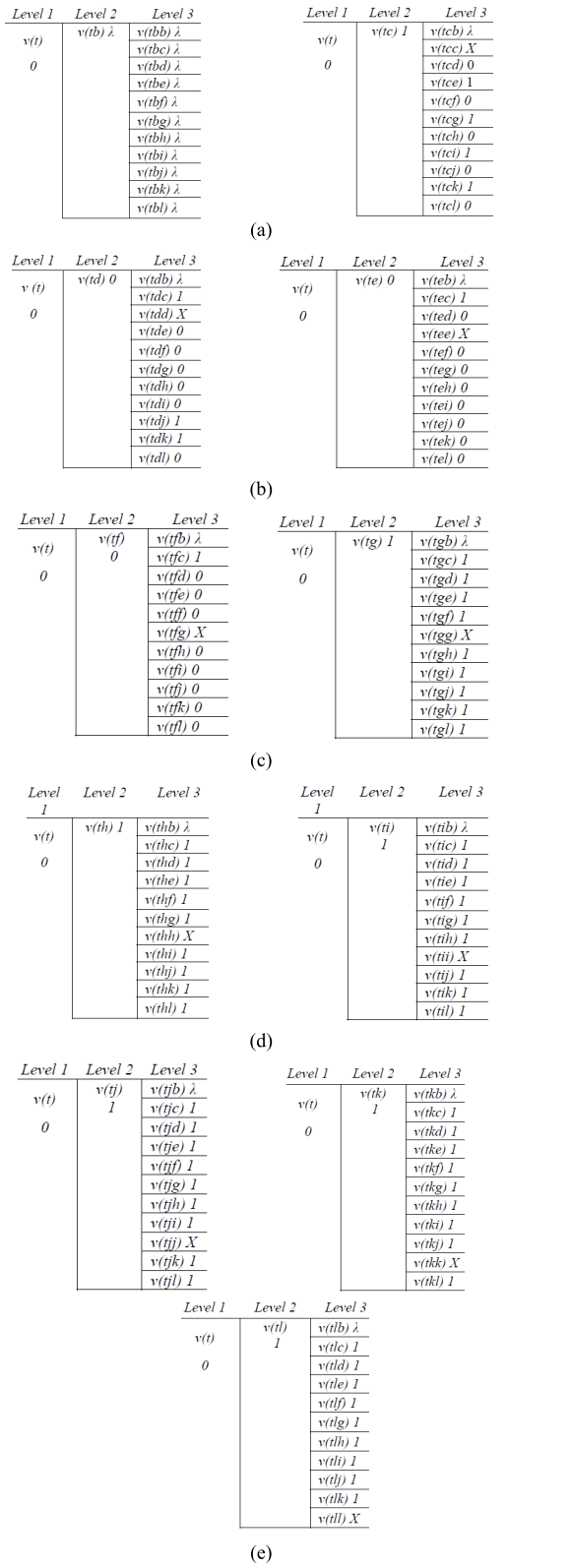

FIGURE 5. The received value of P_d in round 3. (A) The received values of vertex B and vertex C in P_d. (B) The received values of vertex D and vertex E in P_d. (C) The received values of vertex F and vertex G in P_d. (D) The received values of vertex H and vertex I in P_d. (E) The received values of vertexs J ∼L in P_d.

Based on the early stopping function, the collected messages of each service processor in R_3 are sufficient to complete this round and enter the next phase, the decision-making phase. Subsequently, the VOTE function is applied to this phase, and the common value (''1'') of fault-free service processors can be obtained if $n > \lfloor (n-1)/3 \rfloor + 2f_a + f_d$ and $c > 2f_a + f_d$. Next, the fault diagnosis phase is invoked to detect/locate faulty service processors. Following the example above, rule *fdr1* can be used by P_d to detect P_b as a dormant faulty processor. However, the faulty messages are not sufficient to find arbitrary faulty processors by *fdr2*, and thus some service processors may be marked as fault-like processors ($*f_k$) when $v(\alpha k) \neq \text{MAJ}(v_i)$, as with P_c, P_d, P_e and P_f. In rule *fdr3*, the precursor service processors k are marked and $Num\_f_k = Num\_f_k + 1$ when $|Num\_f_k| < \lfloor (n-1-f_d)/3 \rfloor$, as with the precursor service processors of f_d, f_e and f_f. However, P_c can be detected as an arbitrary service processor by accumulating the number of $|Num\_f_c|$ by *fdr2* and *fdr3*. Furthermore, the initial service processor P_t can be detected as an arbitrary faulty service processor by rule *fdr4*. Finally, the diagnosis result of service processor P_d is shown in Table 3, and is achieved using 3 rounds of message exchange in a cloud computing environment. In contrast to EDCA, the FDAMIX requires 5 rounds of message exchange to detect/locate the faulty service processors in the same example. Therefore, the proposed EDCA can detect/locate the maximum number of faulty service processors using the minimum number of rounds and messages under dual failure mode. The next section presents a proof that EDCA can obtain better results than previous methods [17], [25], [30], [31], [36].

V. CORRECTNESS AND COMPLEXITY OF EDCA

This section proves the correctness and complexity of EDCA by lemmas and theorems. The correctness of EDCA can ensure that the requirements of BA, ESP and FDA problems can be achieved simultaneously under $n > (\lfloor (n-1)/3 \rfloor) + 2f_a + f_d$ and $c > 2f_a + f_d$. In addition, only $min\{f_r + 2, f_a + 1\}$ rounds of message exchange are necessary to achieve agreement for all fault-free service processors.

Lemma 1: The fault-free service processor can detect the messages sent from dormant faulty processors.

Proof: Due to the feature of coding, transmitted messages encoded using either the Non-Return-to-Zero code or the Manchester code [16] can be detected by fault-free service processors.

Lemma 2: Each fault-free service processor can communicate with other processors, if $c > 2f_a + f_d$.

Proof: In general, a fault-free service processor can receive at least $c - f_d$ messages sent in each round of the message exchange. This is because the maximum number of dormant faulty service processors is (f_d). If $c - f_d > 2f_a$, a fault-free service processor can determine the correct messages by majority function.

TABLE 3. Diagnosis results of processors P_d.

Faulty type	Faulty Processor	Diagnosis rule	Explanation
Dormant faulty service processor	P_b	*fdr1*	$\lambda > \lfloor (n\text{-}1\text{-}f_d)/3 \rfloor$
Arbitrary faulty service processor	P_c	*fdr2*	$\lvert Num\_f_c \rvert = 3$
		fdr3	$v(tcd, tce, tcf) \neq$ MAJ(v_i) $\lvert Num\_f_c \rvert + 3 = 6 > f_a$
An initial arbitrary faulty service processor	P_t	*fdr4*	$v(T_i^{k-1}) \neq v(T_j^{k-1})$ and $\lvert v(T_i^{k-1}) \neq v(T_j^{k-1}) \rvert > f_a$,

Lemma 3: A dormant faulty service processor can be detected by a fault-free service processor by the forwarding technique.

Proof: If the number of received λ values is greater than or equal to $c - \lfloor (n-1)/3 \rfloor$, then the transmitting processor has a dormant fault. This is because there are at most $\lfloor (n-1)/3 \rfloor$ arbitrarily faulty service processors in the system; thus, there are at most $\lfloor (n-1)/3 \rfloor$ non-λ values in the vector V_i.

Theorem 1: A fault-free service processor can remove and detect the faulty influences from dormant faulty processors, if $c > 2f_a + f_d$.

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

Theorem 2: The BA problem can be solved by EDCA.

Proof: Due to previous results in [31], [32], [33], [34], and [36], the VOTE(s) of each correct vertex in the *ic-tree* are common and equal to v. Therefore, the constraints of BA [21], [34], [36], [37] can be satisfied.

Theorem 3: The EDCA protocol can achieve agreement in $min\{f_r + 2, f_a + 1\}$ rounds.

Proof: Based on Theorem 2, the constraints of BA can be met in the EDCA protocol when $f_r = f_a$. However, the values of a descendant path are fixed and common in round $min\{f_r + 2, f_a + 1\}$ when the message exchange process is stopped early. Therefore, the constraints of BA are also satisfied in round $min\{f_r + 2, f_a + 1\}$.

Lemma 4: The arbitrary faulty service processor can be detected/located if $n > \lfloor (n-1)/3 \rfloor + 2f_a + f_d$, $c > 2f_a + f_d$, *and* $r > f_a + 1$.

Proof: Due to the constraint $n > \lfloor (n-1)/3 \rfloor + 2f_a + f_d$ and $c\text{-}1 > 2f_a + f_d$, there are at most f_d dormant faulty service processor. By Lemma 3, all the dormant faulty service processor can be detected by each fault-free service processor, so $f_d = \lvert$ number of $P_k \rvert$ in rule *fdr*1. By the same constraint, there are at most f_a arbitrary faulty service processor, so there are at most f_a values (except λ) at the same labeled vertex in the *ic-tree* different from the most common value, that is $n - \lfloor (n-1)/3 \rfloor$-number of $P_k > 2f_a, f_a < \lfloor (2n+1)/6 \rfloor - \lvert$ number of $P_k \rvert$. So, if the most common value does not appear at the same labeled vertex in the *ic-tree* more than c-($\lvert$number of $P_k \rvert + \lfloor (2n+1)/6 \rfloor) - 1(n - (f_a + f_d) - 1)$ times, then the component is in arbitrary fault.

Theorem 4: Protocol EDCA satisfies the fairness requirement of FDA.

Proof: By Lemma 3 and Lemma 4, no fault-free service processor is falsely detected as faulty by any fault-free service processor if $n > \lfloor (n-1)/3 \rfloor + 2f_a + f_d$ and $c - 1 > 2f_a + f_d$.

Theorem 5: The FDA problem can be solved by EDCA if $n > \lfloor (n-1)/3 \rfloor + 2f_a + f_d$, $c > 2f_a + f_d$, and $r > f_a + 1$.

Proof: By theorems 1, 2 and 4, the theorem is proved.

Theorem 6: The EAFD protocol requires $min\{f_r + 2, f_a + 1\}$ rounds to solve the FDA and ESP problems in dual failure mode in a cloud computing environment if $n > \lfloor (n-1)/3 \rfloor + 2f_a + f_d$ and $c > 2f_a + f_d$. Furthermore, the $min\{f_r + 2, f_a + 1\}$ rounds are the minimum number of rounds.

Proof: Based on the work of Fischer and Lynch [12], the $f_a + 1$ rounds are the lower boundary for message exchanges when the transmission medium is fault-free. As a result, at least $f_a + 1$ number of rounds are necessary to solve the BA problem. Based on the above, the real number of message exchange rounds in EBA is $min\{f_r + 2, f_a + 1\}$, which are proved in Krings and Fisher [20], are also proved when the fallible components are the only processors in the system.

VI. CONCLUSION

Due to the size, range and variety of cloud computing environments available today, cloud service processors need to provide users with a large number of services through the Internet. The reliability issue is thus of particular importance in such environments. However, in traditional BA and FDA protocols, large numbers of messages must be exchanged between processors, resulting in high protocol overhead. This study therefore combines the concepts of early stopping and fault diagnosis to achieve the goal of BA and FDA efficiently and quickly in a cloud computing environment. First, the EDCA protocol can stop the message exchange process earlier when a sufficient number of messages has been received. Then, the diagnosis rule of EDCA can be used to detect/locate the maximum number of faulty service processors using the minimum number of rounds. Only $min\{f_r + 2, f_a + 1\}$ rounds of message exchange are necessary for all fault-free service processors to achieve agreement. As a result, the proposed EDCA protocol is more suitable and efficient than previous works [9], [11], [31], [32], [34], [36], especially for cloud computing environments with a large number of service processors.

In the future, the consensus between cloud cluster needs to be considered. This is because that each service processor can be located in different clusters in cloud computing environment and needs to cooperate with others among all of clusters. As a result, our future work will discuss the

consensus between different clusters under the dual failure mode, the link fault especially.

REFERENCES

[1] I. Abraham and D. Dolev, "Byzantine agreement with optimal early stopping, optimal resilience and polynomial complexity," in *Proc. 47th Annu. ACM Symp. Theory Comput.*, New York, NY, USA, 2015, pp. 605–614.

[2] S. Alcock and R. Nelson, "Application flow control in YouTube video streams," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 2, pp. 25–30, Apr. 2011.

[3] Amazon Web Services. *Explore the AWS Platform, Cloud Products, and Capabilities*. Accessed: Jul. 26, 2009. [Online]. Available: http://aws.amazon.com/

[4] R. Baldoni, J.-M. Hélary, M. Raynal, and L. Tangui, "Consensus in Byzantine asynchronous systems," *J. Discrete Algorithms*, vol. 1, no. 2, pp. 185–210, 2003.

[5] R. Buyya, R. Ranjan, and R. N. Calheiros, "Modeling and simulation of scalable cloud computing environments and the CloudSim toolkit: Challenges and opportunities," in *Proc. Int. Conf. High Perform. Comput. Simulation*, Jun. 2009, pp. 1–11.

[6] C. Cachin, S. Schubert, and M. Vukolić. (2016). "Non-determinism in Byzantine fault-tolerant Replication." [Online]. Available: http://arxiv.org/abs/1603.07351

[7] M.-L. Chiang, S.-C. Wang, and L.-Y. Tseng, "An early fault diagnosis agreement under hybrid fault model," *Expert Syst. Appl.*, vol. 36, no. 3, pp. 5039–5050, 2009.

[8] N. Deo, *Graph Theory With Applications to Engineering and Computer Science*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1974.

[9] D. Dolev, R. Reischuk, and H. R. Strong, "Early stopping in Byzantine agreement," *J. ACM*, vol. 37, no. 4, pp. 720–741, 1990.

[10] D. Dolev and R. Reischuk, "Bounds on information exchange for Byzantine agreement," *J. ACM*, vol. 32, no. 1, pp. 191–204, 1985.

[11] D. Dolev and H. R. Strong, "Requirements for agreement in a distributed system," in *Proc. 2nd Int. Symp. Distrib. Data Bases*, Berlin, Germany, 1982, pp. 115–129.

[12] M. J. Fisher and N. A. Lynch, "A lower bound for the time to assure interactive consistency," *Inf. Process. Lett.*, vol. 14, pp. 183–186, Jun. 1982.

[13] *Gartner Says Cloud Computing Will be as Influential as E-Business*. Accessed: Jul. 26, 2009. [Online]. Available: http://www.gartner.com/it/page.jsp?id=707508

[14] *Google Cloud*. Accessed: Jun. 9, 2018. [Online]. Available: https://en.wikipedia.org/wiki/Google_Cloud_Platform

[15] T. L. Gunarathne, T.-L. Wu, J. Qiu, and G. Fox, "MapReduce in the clouds for science," in *Proc. IEEE 2nd Int. Conf. Cloud Comput. Technol. Sci. (CloudCom)*, Indianapolis, IN, USA, Nov./Dec. 2010, pp. 565–572.

[16] F. Halsall, *Data Communications, Computer Networks and Open Systems*, 4th ed. Reading, MA, USA: Addison-Wesley, 1995, pp. 112–125.

[17] H.-S. Hsiao, Y.-H. Chin, and W. P. Yang, "Reaching fault diagnosis agreement under a hybrid fault model," *IEEE Trans. Comput.*, vol. 49, no. 9, pp. 980–986, Sep. 2000.

[18] H.-C. Hsieh and M.-L. Chiang, "New approach to improve the generalized Byzantine agreement problem," *Int. J. Comput. Theory Eng.*, vol. 7, no. 2, pp. 120–125, 2015.

[19] *IBM's Blue Cloud Project*. Accessed Oct. 20, 2009. [Online]. Available: http://www03.ibm.com/press/us/en/pressrelease/22613.wss/

[20] A. W. Krings and T. Fisher, "The Byzantine agreement problem: Optimal early stopping," in *Proc. 32nd Hawaii Int. Conf. Syst. Sci.*, 1999, pp. 1–12.

[21] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, Jul. 1982.

[22] L. Lamport, "Lower bounds for asynchronous consensus," in *Proc. Int. Workshop Future Directions Distrib. Comput.*, Jun. 2002, pp. 22–23.

[23] W. H. Li, Y. Yang, and D. Yuan, "A novel cost-effective dynamic data replication strategy for reliability in cloud data centres," in *Proc. IEEE 9th Int. Conf. Dependable, Auton. Secure Comput. (DASC)*, Sydney, NSW, Australia, Dec. 2011, pp. 496–502.

[24] Y. Mansouri and R. Monsefi, "Optimal number of replicas with QoS assurance in data grid environment," in *Proc. 2nd Asia Int. Conf. Modeling Simulation (AICMS)*, Kuala Lumpur, Malaysia, May 2008, pp. 168–173.

[25] J. P. Martin and L. Alvisi, "Fast Byzantine consensus," *IEEE Trans. Dependable Secure Comput.*, vol. 3, no. 3, pp. 202–215, Jul. 2006.

[26] *Platform as a Service*. Accessed: Jun. 9, 2018. [Online]. Available: https://en.wikipedia.org/wiki/Platform as a service

[27] E. J. Qaisar, "Introduction to cloud computing for developers: Key concepts, the players and their offerings," in *Proc. Inf. Technol. Prof. Conf. (TCF Pro IT)*, Ewing, NJ, USA, Mar. 2012, pp. 1–6.

[28] K. Ranganathan and I. Foster, "Identifying dynamic replication strategies for a high-performance data grid," in *Proc. 2nd Int. Workshop Grid Comput.*, 2002, pp. 75–86.

[29] P. K. Sangdeh, M. Mirmohseni, and F. Poursabzi, "Applying the Byzantine agreement in wireless sensor networks based on clustering," in *Proc. IEEE 23rd Iranian Conf. Elect. Eng.*, May 2015, pp. 619–624.

[30] K. Shin and P. Ramanathan, "Diagnosis of processors with Byzantine faults in a distributed computing system," in *Proc. Symp. Fault-Tolerant Comput.*, 1987, pp. 55–60.

[31] H.-S. Siu, Y.-H. Chin, and W.-P. Yang, "Byzantine agreement in the presence of mixed faults on processors and links," *IEEE Trans. Parallel Distrib. Syst.*, vol. 9, no. 4, pp. 335–345, Apr. 1998.

[32] S.-S. Wang, S.-C. Wang, and K.-Q. Yan, "An optimal solution for Byzantine agreement under a hierarchical cluster-oriented mobile ad hoc network," *Comput. Elect. Eng.*, vol. 36, no. 1, pp. 100–113, 2010.

[33] S.-C. Wang, K.-Q. Yan, S.-S. Wang, and G.-Y. Zheng, "Reaching agreement among virtual subnets in hybrid failure mode," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 9, pp. 1252–1262, Sep. 2008.

[34] K.-Q. Yan and S.-C. Wang, "Grouping Byzantine agreement," *Comput. Standard Interfaces*, vol. 25, no. 1, pp. 75–92, 2005.

[35] K.-Q. Yan and S.-C. Wang, "Reaching fault diagnosis agreement on an unreliable general network," *Inf. Sci.*, vol. 170, pp. 397–407, Feb. 2005.

[36] K.-Q. Yan, S.-C. Wang, and S.-S. Wang, "An optimal solution of Byzantine agreement in a scale free network," in *Proc. IEEE 22nd Int. Conf. Adv. Inf. Netw. Appl. (AINA)*, Okinawa, Japan, Mar. 2008, pp. 270–275.

[37] K.-Q. Yan, S.-S. Wang, and S.-C. Wang, "Reaching an agreement under wormhole networks within dual failure component," *Int. J. Innov. Comput. Inf. Control*, vol. 6, no. 3, pp. 1151–1164, 2010.

[38] H. Yoshino, N. Hayashibara, T. Enokido, and M. Takizawa, "Byzantine agreement protocol using hierarchical groups," in *Proc. 11th Int. Conf. Parallel Distrib. Syst.*, Jul. 2005, pp. 40–64.

[39] Y. Zhang and M. R. Lyu, "QoS-aware Byzantine fault tolerance," in *QoS Prediction in Cloud and Service Computing* (Springer Briefs in Computer Science). Singapore: Springer, 2017.

MAO-LUN CHIANG received the Ph.D. degree in computer science from National Chung Hsing University, Taiwan. He is currently an Associate Professor with the Department of Information and Communication Engineering, Chaoyang University of Technology, Taiwan. His current research interests include mobile computing, parallel computing, fault tolerance, and cloud computing.

CHIN-LING CHEN received the Ph.D. degree from National Chung Hsing University, Taiwan, in 2005. From 1979 to 2005, he was a Senior Engineer with Chunghwa Telecom Company Ltd. He is currently a Distinguished Professor. He has published over 80 articles in SCI/SSCI international journals. His research interests include cryptography, network security, and electronic commerce.

HUI-CHING HSIEH received the B.S. and M.S. degrees in information management from the Chaoyang University of Technology, Taiwan, in 2002 and 2004, respectively, and the Ph.D. degree in computer science from National Tsing Hua University, Taiwan, in 2010. She is currently an Assistant Professor with the Department of Information Communication, Hsing Wu University, Taiwan. Her research interests include distributed data processing, fault tolerant computing, and P2P network computing.