A Secure Service Agreement Underlying Cloud Computing Environment

Mao-Lun Chiang
Department of Information and Communication Engineering, Chaoyang University of Technology, Taiwan
mlchiang@cyut.edu.tw

Abstract

Reliability is an important research topic of distributed systems. To achieve fault-tolerance in the distributed systems, healthy processors need to reach an agreement before performing certain special tasks, even if faults exist in many circumstances. In order to achieve fault-tolerance in distributed systems, one must deal with the Byzantine Agreement (BA) problem. However, the traditional BA problem is solved in well-defined networks, but the cloud computing environment is increasing in popularity. It can provide a large number of applications in the Internet. Therefore, the BA problem is re-examined to enhance the reliability of cloud computing environment in this paper. The proposed protocol can reach a secure service agreement while tolerating the maximum number of faulty processor in a minimal number of message exchanges under dual failure mode by using type of service requirement (*service chip*) and key ($Sign_{sk}$). Furthermore, our protocol is more adapting to the changeability of network, such as the mobility of processors than other works.

Keywords: Byzantine agreement, Fault-tolerant, Reliability, Cloud computing

1 Introduction

A distributed computing system consists of a set of processors, which can communicate with each other by exchanging messages. In order to enhance the reliability of a computer system, a mechanism allowing a set of processors to agree on a common value is needed [10, 13]. Some examples of such applications are: a commitment problem in a distributed database system [5, 13], a clock synchronization problem [6], and a landing task controlled by a flight path finding system [2]. Such a unanimity problem was first studied by Lamport et al. [10], and called a Byzantine Agreement (BA) [2-3, 7, 17]. This problem requires a number of independent processors to reach an agreement in cases where some of those processors might be faulty.

Subsequently, the symptom of processor failure needs to be mentioned, it can be classified into two categories, the dormant fault (f_d) and malicious fault (f_m) [7, 10, 16, 19, 20-22]. Dormant processor faults include broken processors (crash faults) and message misses (omission faults), and are easy to detect and solve. However, malicious faults are unpredictable and damaging. An malicious processor can withhold messages or collude with other faulty processors to send irregular message to others, and thus is a more serious problem than dormant faults. However, some malicious-resilient BA protocols treat all faults as malicious faults, even though some faults may be subjected to dormant faults. Thus, malicious-resilient BA protocols cannot tolerate the maximum number of faults if the dormant faults exist. These observations motivate this study to maximize the number of allowable faulty processors in dual fault mode (malicious and dormant faults exist simultaneously).

As the network technology continues to grow at a high rate of speed, traditional network topology is improved with huge computation ability, such as cloud computing environment [4, 8, 15-16, 18, 23, 25-27]. This kind of environment is extended from the grid computing and distributed computing, thus there exist several challenges to this new concept of cloud computing environment, such as the reliability and fluency. The service requirements need to be dispatched and executed completed even if some components are faulty. Besides, the user may emigrate away from its region, the service processors may forward the workload to other service processors. For reducing workload of service processors, the service requirements of user need to be relocated and dispatched to appropriate service processor. Therefore, the proposed protocol needs to enhance the reliability of cloud computing environment and tolerate a maximal number of faulty processors under dual failure mode by using minimal number of message exchange.

The rest of this paper is organized as follows: Section 2 illustrates the basic assumption and previous work underlying our protocol. The detail of the proposed protocol is shown in Section 3. Section 4 illustrates examples of execution procedure in detail. Subsequently, correctness and complexity are

*Corresponding Author: Mao-Lun Chiang; E-mail: mlchiang@cyut.edu.tw
DOI: 10.3966/160792642018111906012

illustrated in Section 5. Finally, the conclusion is presented in Section 6.

2 The Basic Assumption and Previous Work

In general, the cloud computing is a new concept of distributed system to provide multiple external customers "as a service" using Internet technologies and is showing in Figure 1. The providers of cloud provide various applications and computing infrastructure by using virtualization of infrastructure for different customers. The popular examples are Google [25-26], IBM Blue Cloud [27], Microsoft Azure [28], and Amazon [24]. In the Google application, the free storage capacity and powerful computing capacity are provided, such as the Gmail and YouTube. Besides, the Google App Engine provides users to make web applications with its SDKs and APIs in the develop platform.

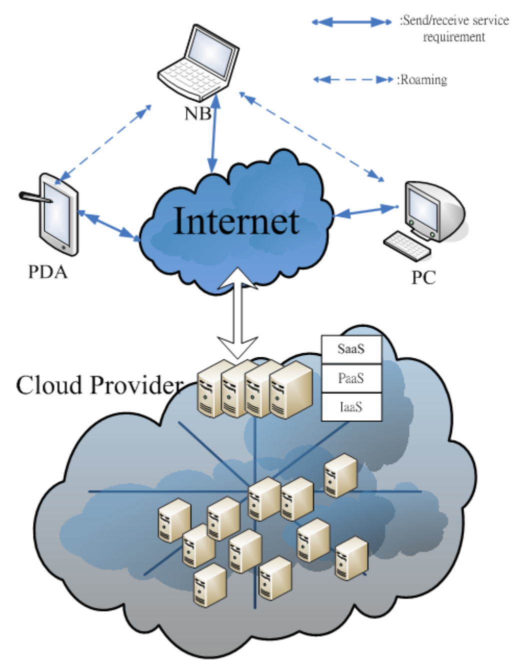


Figure 1. The cloud computing environment

Besides, the related classification of services including Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS) [15]. The SaaS allows the users to use the applications over Internet on demand, such as Google Docs [26]. And, the popular instance of PaaS is Google App Engine [26] that enables to deploy the Web platform. In IaaS, the basic infrastructure is provided, such as ability of CPU, memory, storage, and related resources. The Amazon's Elastic Compute Cloud (EC2) [24] is an example for IaaS.

However, the services of cloud computing must be terminated while the components are damaged or failure. Therefore, a safe agreement protocol is necessary to reach reliability and integrity. The proposed agreement protocol needs to obtain the common message to recognize the type of service by service chip [14] and forward the job to other service processors safety. Subsequently, the basic assumptions agreement protocol is described as follows.

In general, the traditional agreement problem cannot achieve agreement in a cloud computing environment. It is because that the cloud computing environment is consisting of a large number of processors and applications, each service needs to be cooperated with service processor to get the agreement. As a result, the BA problem needs to be revised in cloud computing environment. Besides, the result of Fischer and Lynch [7], showing agreement, is impossible in an asynchronous environment with even one processor failure. Therefore, the BA problem is considered in a synchronous network in this paper. In addition, Lamport argues for the agreement problem under the assumption of synchronous behavior BA, showing that $3f_m$+1 processors are allowed f_m failures where f_m is the number of malicious faulty processors in the network [10].

Traditionally, the BA problem was defined by Lamport et al. [10], as follows:

- There are n (n>3) processors, of which at most one-third of the total number of processors could fail without breaking down a workable network;
- The message's sender is always identifiable by the receiver;
- An malicious faulty processor is chosen as a source, and its initial value is broadcasted to other processors and to itself to execute the protocol.

In general, a healthy source processor sends the same inital value to all processors and malicious processors cannot affect this inital value sent from the healthy source processor. Therefore, the faulty influence of this situation is easy to be solved. However, the source processor, which has malicious faults, may transmit different values to different processors. This situation is the worst case of the BA problem and is worth discussing. Therefore, we assume the source processor is an malicious processor in (4). Furthermore, the agreement is reached if the following requirements are satisfied [1, 5, 16, 21]:

- (BA_1) Agreement: All healthy service processors shall agree on a common value v.
- (BA_2) Validity: If the initial value of the source is v_s, and the source is fault-free, then all healthy service processors shall agree on the value v_s; i.e., $v = v_s$.

Under these assumptions and requirements, several protocols [1-2, 5-6, 10-13, 21] have been proposed for solving such problems. The protocol in Lamport et al. [10] indicates that f_m+1 ($f_m \leq \lfloor (n-1)/3 \rfloor$) rounds (a round denotes the interval of message exchange) of message exchange are required to reach a common agreement in a synchronous fully connected network where n is the number of processors. Further, Fischer and Lynch [7] point out that f_m+1 rounds are the minimum number of rounds needed for sufficient messages to achieve BA in synchronous network. Therefore, we assume that the bounds on the processing and communication delays of healthy components are finite [6, 9].

Based on the reason above, this paper proposes an efficient and suitable protocol to enhance the reliability of cloud computing environment by using the minimum round of message exchange. Besides, the proposed protocol can still tolerate a maximal number of faulty processors in a minimal number of message exchanges under dual failure mode. The detail of protocol is shown in next Section.

3 Protocol CSA

The Cloud Service Agreement (CSA) we proposed to solve the BA problem under dual failure mode by using minimal number of message exchange in cloud computing environment is shown in Figure 2. The CSA requires σ rounds of message exchange to reach an agreement where $n > \lfloor (n-1)/3 \rfloor + 2f_m + f_d$ and $c > 2f_m + f_d$ and includes three phases: *message collect phase*, *decision-making phase*, and *secure relay phase*. The descriptions of workflow diagram and parameters are shown in Figure 3 and Table 1. Besides, the related explanation can be proven in Section 5.

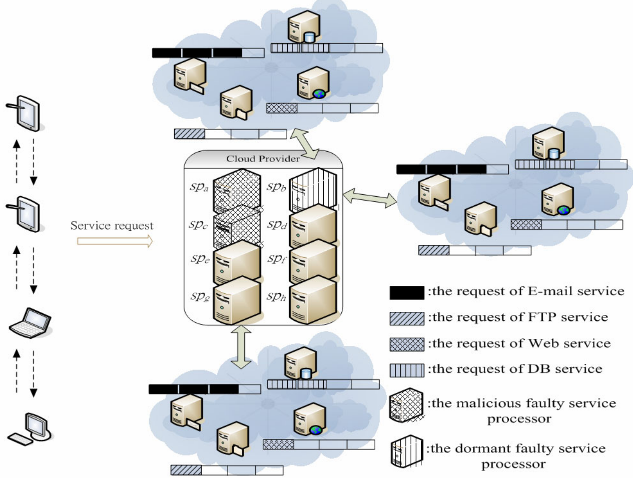

Service request
Cloud Provider
sp_a
sp_b
sp_c
sp_d
sp_e
sp_f
sp_g
sp_h
:the request of E-mail service
:the request of FTP service
:the request of Web service
:the request of DB service
:the malicious faulty service processor
:the dormant faulty service processor

Figure 2. The cloud computing environment with faulty components

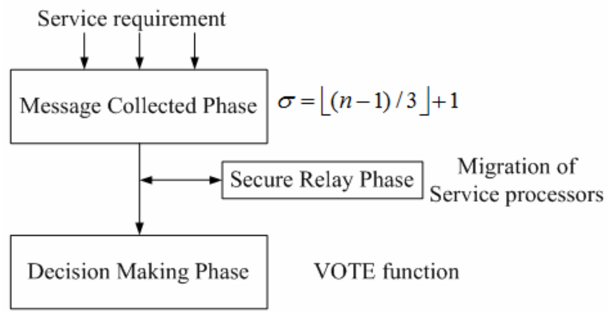


Figure 3. The workflow diagram of CSA protocol

Table 1. The description of parameter in our protocol

parameter	description
n	The total number of service precessors in the cloud computing environment.
$v(s)$	The initial value of service precessor s broadcasting to all other service precessors.
$v(sc)$	The value $v(s)$ is sent from the service processor c.
f_m	The number of service processors with malicious faults.
f_d	The number of service processors with dormant faults.
λ	When a service processor is detected to be a dormant service processor, the value is sent from is replanced by λ.
c	The connectivity of the cloud computing environment. Based on Menger's theorem [3], at least c disjoint paths must exist between any pairs of service processors x and y when the connectivity of the network is c.
T_i	An information collecting tree(*ic-tree*) of service processor sp_i.
d_i	The decision value of service process or i.
$sign_{sk}(d_i)$	The service processor i signs its decision value d_i by using its signing key sk.
Φ	The default value, and $\Phi \in \{0,1\}$.
σ	The required rounds of message exchange ($\sigma \le \lfloor (n-1)3 \rfloor + 1$).

The main work of the *message collect phase* is collecting the service requirement and random key from users. Subsequently, the received values are exchanging and storing into corresponding *ic-tree* (an information collecting; T_i) [2, 16, 20-22] for each round. With regard to the *ic-tree*, it is a convenient tree structure and is constructed from the accumulated the messages. The vertex of an *ic-tree* is labeled with a list of service processor names, and the value received from the source processor is denoted as $v(s)$ at the root of the *ic-tree*. The service processor name list contains the names of the service processors through which the stored message has been transferred.

For example, the statement $v(sbc)$ represents the service processor having received the value *sb* from service processor c which was sent from source service processor s to service processor b. The vertices having repeated service processor names of *ic-trees* are removed in order to avoid cyclical influences from the faulty service processors. Therefore, the *ic-tree* (T_i) can be used to store the received messages and to eliminate the influence of faulty components, as shown as Figure 4.

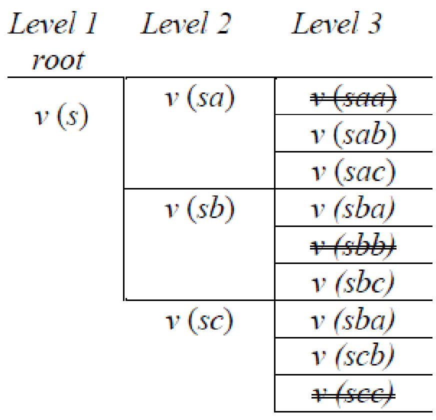


Figure 4. The *ic-tree* (T_i)

After the *message collected phase*, the function VOTE [16, 22] is used to eliminate the influence of faulty processors by taking a decision vector value VOTE(*s*) of the root *s* of each healthy service processor's *ic-tree*. The decision vector value (d_j) includes the type of service requirement (*service chip*) and key (*Sign*$_{sk}$). The service chip is constructed by *Walsh code* [9, 14] to represent the type of service. In general, each bit time is subdivided into *m* short intervals called chips and each type of service is assigned a unique *m*-bit code called a chip sequence. The normalized inner product of any two distinct chip sequences, S and T (written as S * T), is 0 due to all chip sequence are pairwise. Based on binary chip sequence, the represent of bipolar notation is used to compute the normalized inner product of the received chip sequence by equation (1). For example, the service chip (1, 0, 0, 1) can be switched to (+1, -1, -1, +1). As a result, a user can transmit a 1 bit for specific type of service.

$$S*S=\frac{1}{m}\sum_{i=1}^{m}S_iS_i \qquad \textbf{(1)}$$

However, the low-power service processor may be exhausted in a cloud computing environment, the tasks need to be dispatched to other service processor. Besides, the user may keep to send the request to server when it migrates to other region. For providing efficient service, the following requirements need to be served in local server processors. Therefore, we need the extra phase, *secure relay phase*, to transfer the unfinished works to appropriate service processors. The service processor relay the package *Sign*$_{sk}$(*task message*) to the service processor which is near to user in this phase. After receiving *Sign*$_{sk}$(*task message*) and *task message*, the receiver will verify the signature by using the corresponding verification key. Eventually, the neighboring service processor can take over the unfinished works. The detail of protocol CSA is shown in Figure 5.

CSA protocol

Preprocessing: The service requirement including type of service chip and random key is sent to any service processors. The total number of rounds is σ ($\sigma=\lfloor(n-1)/3\rfloor+1$);

Message Collected Phase:

σ = 1, do:

1) Each service processor *i* broadcasts the received service requirement *v*(*s, s, s, s, s*) to other service processors and itself. Subsequently, the corresponding *ic-tree* is constructed by each service processor.

σ = 2, do:

2) Broadcast the *ic-tree* to all service processors and construct its corresponding *ic-tree* for each round.

For σ> 2, do:

3) Each service processor broadcasts the its *ic-tree* to others, and receives the other *ic-trees* sent by other service processors to construct the corresponding *ic-tree* for each round.

4) The vertices having repeated processor names of *ic-trees* are removed in order to avoid cyclical influences in this round.

Decision Making Phase:

Use the VOTE function to the root of each processor's *ic-tree* T_j and the common decision vector value d_j(=*service chip* + *Sign*$_{sk}$) is obtained. Subsequently, the type of service can be computed by the normalized inner product of itself by equation (1).

Secure Relay Phase:

The service processor forwards its signature *Sign*$_{sk}$(*task message*) on the *task message* to newly service processors. After receiving *Sign*$_{sk}$(*task message*) and *task message*, the newly service processor verifies the signature by using the corresponding verification key *Sign*$_{sk}$.

VOTE(α)

1. *val*(α), if α is a leaf.
2. The majority value in the set of {VOTE(α*i*)|1 ≤ *i* ≤ g_p, and vertex α*i* is a child of vertex α}, if such a majority value exists.

Otherwise a default value ϕ is chosen.

Figure 5. Cloud Service Agreement (CSA) protocol

According to description above, the f_m+1 ($f_m\le\lfloor(n-1)/3\rfloor$) rounds of message exchange is necessary to reach an agreement. Namely, all healthy service processors can reach an agreement under a cloud computing environment where $n>\lfloor(n-1)/3\rfloor+2f_m+f_d$ and $c>2f_m+f_d$.

4 Examples of Execution Procedure

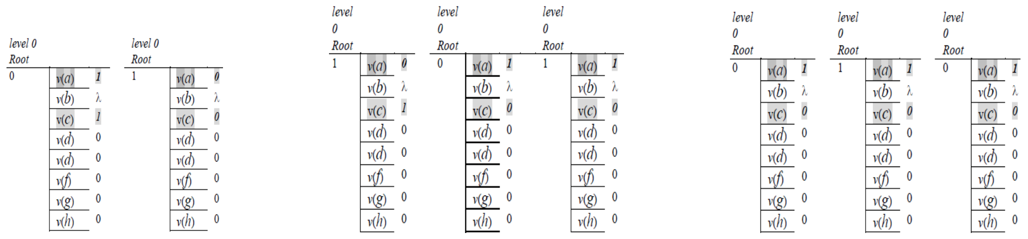
In this section, an example of 8-service processors is shown to illustrate the CSA protocol in Figure 6. Each service processors can receive four kinds of service requirements, the FTP, Email, Web, and DB Services. Those services have its only chip number, FTP (01000010), Email (00101110), Web (00011011), and DB Services (01011100) respectively. Besides, the service processors sp_a and sp_c are assumed as malicious service processors and service processor sp_b as the dormant service processor shown in Figure 6(a). Basically, the user will send the requirement messages including the service chip and random key to service processors. The service chip can be used to represent the type of service and then the secure transmission can be achieved by this random key. Furthermore, the results of healthy service processors are discussed in this example to satisfy the requirement of A (BA_1). Besides, the results of faulty service processors are ignored due to those are trivial thing [13, 16, 21].

	Service chip	Key
User's requirement	0110 (Email & Web services)	1010

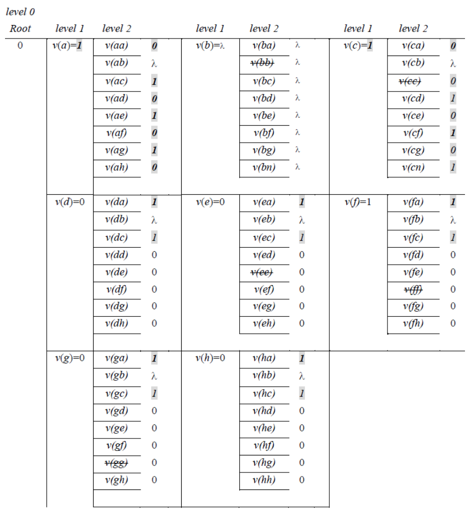
(a) The user's service requirement

	Service chip	Key
sp_a	1001	1111
sp_b	λλλλ	λλ
sp_c	1010	0000
sp_d	0110	1010
sp_e	0110	1010
sp_f	0110	1010
sp_g	0110	1010
sp_h	0110	1010

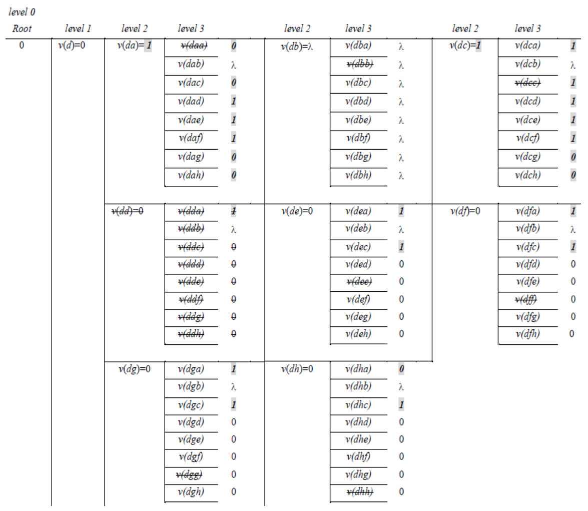
(b) The received service requirements in *preprocessing*



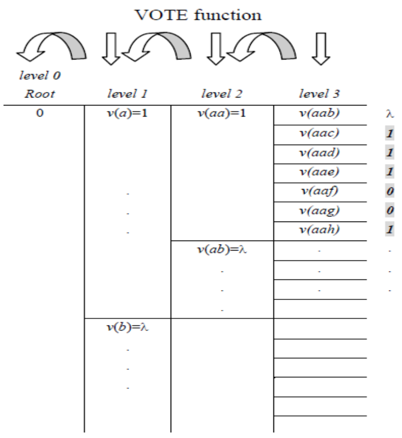
(c) The sp_d's ic-tree of in first round of message exchange phase



(d) The first vector value of sp_d's *ic-tree* in second round of *message exchange phase*



(e) The value $v(d)$ of first vector value in sp_d's *ic-tree* in third round of *message exchange phase*



(f) A VOTE function of the *decision-making phase* is applied to first vector value in sp_d

Figure 6. The procedure of CSA in an example of 8-service processors

At the beginning of the protocol, the user sends the message to service processor to acquire the service in preprocessing and shows in Figure 6(a). For example, the user sends the service requirements (01101010) including service chip (0110) and key (1010) to service processors. Subsequently, the received service processors broadcast this requirements to all service processors. However, the malicious service processors sp_a and sp_c send the different vector value to others, the details of procedure are shown in Figure 6(b).

After preprocessing procedure, each service processor broadcasts its a set of initial values (vector) to each other. For clarity of this example, the result of healthy service processor sp_d is illustrated to explain the exchange process in *message collect phase*. It is because that the same steps are required for the other healthy service processors and the results of the faulty service processors are not necessary to be discussed. Subsequently, the sp_d stores the received vector values into the root of its *ic-tree* respectively in the first round of *message collect phase*, as shown in Figure 6(c). Due to the malicious faulty service processor wants to break-down the agreement, thus the transmitted vector is changeable for each round. The faulty values are marked by the shadow and italic.

Subsequently, each service processors exchanges the received *ic-trees* with all service processors during the second round of *message collected phase*. Similarly, the first vector value of sp_d's *ic-tree* in second round of *message collected phase* is only shown in Figure 6(d) to make it clear. The procedures of other vector values are the same. In the third round of *message collected phase*, each service processor exchanges the received *ic-trees* and stored into the third level of their *ic-trees*. Besides, the vertices having repeated processor names of *ic-trees* are removed in order to avoid cyclical influences. Due to the number of message is huge and complicated, the value $v(d)$ of first vector value in sp_d's *ic-tree* is also used to explain the result of third round of message exchange in Figure 6(e).

Finally, the function VOTE is applied to root of first vector value in sp_d's *ic-tree* to obtain a common value in the *decision-making phase* in Figure 6(f). Subsequently, the other vector values can be required during this phase, the common results of service requirements in sp_d are (01101010). Subsequently, the normalized inner product of the received chip sequence can be computed by equation (1) as follows.

(0110) $\rightarrow$ (0, 0, +2, 0, +2, +4, -2, -2)$\rightarrow$Based on binpolar notation.

Ftp's service chip = (-1, -1, -1, +1, +1, -1, +1, +1)$\bullet$(0, 0, +2, 0, +2, +4, -2, -2)/8=-1

Email's service chip = (-1, -1, +1, -1, +1, +1, +1, -1)$\bullet$(0, 0, +2, 0, +2, +4, -2, -2)/8=1

Web's service chip = (-1, +1, -1, +1, +1, +1, -1, -1)$\bullet$(0, 0, +2, 0, +2, +4, -2, -2)/8=1

DB's service chip = (-1, +1, -1, -1, -1, -1, +1, -1)$\bullet$(0, 0, +2, 0, +2, +4, -2, -2)/8=-1

Based on the computation above, the requirements of user are the Email and Web services. Subsequently, the requirements can be transmitted to the corresponding processors. Since all healthy service processors will execute the same procedures, a set of common vector values are also reached in sp_d, sp_e, sp_f, sp_g, and sp_h. Therefore, the CSA protocol can make each healthy service processor to enhance the reliability of system even if a large number of service processors in cloud computing environment.

5 The Correctness and Complexity of CSA

For clarity of this paper, the agreement and validity property of BA can be proven to show the CSA is optimal solution in this section.

5.1 Correctness of CSA

At first, this paper defined a vertex α as common [2, 20-22] if each healthy service processor computes the same value for α. In other words, the value stored in vertex α of each healthy service processor's *ic-tree* is common to all. Once each healthy service processor has a common initial value from the source service processor or user in the root of its *ic-tree*, an agreement is reached since the root is common to all. Thus, the agreement (BA_1) and (BA_2), can be rewritten as:

(BA_1'): Root *s* is common, and

(BA_2'): VOTE(*s*) = v_s for each healthy service processor, if the source is healthy.

The term common frontier [2, 20-22] is defined as follows: "If every root-to-leaf path of the *ic-tree* contains a common vertex, the collection of the common vertices forms a common frontier." In other words, every healthy service processor collects the same messages within a common frontier if a common frontier exists in that healthy service processor's *ic-tree*. Subsequently, using the same voting function VOTE to compute the root value of the *ic-tree* to eliminate the influence of faulty service processor. Subsequently, every healthy service processor can obtain the same root value because they utilize the same input and the same computing function. Due to the above concepts can be used to prove the correctness of interactive/BA problem, thus the CSA will follow this way to prove the correctness.

Before proving the correctness of CSA, the term *correct vertex* is defined as:

✧ Correct vertex: Vertex α*i* of a tree is a correct vertex if service processor sp_i is healthy. In other words, a correct vertex is a place to store values received from healthy service processors.
✧ True value: For a *correct vertex* α*i* in the tree of a healthy service processor sp_i, val(α*i*) is the true value of vertex α*i*. Namely, the stored value is called the

true value.

By the definition of a correct vertex, the stored value in the *ic-tree* is received from healthy service processor, and a healthy service processor always transmits the same value to other service processors. Namely, the root can be proven to be a *common vertex* (BA_1') due to the existence of a common frontier, regardless of the correctness of a source processor. Based on reasoning above, an agreement among the root values is reached.

Subsequently, we will check the condition of (BA_2'). Based on (BA_2'), we know that when the source processor fails, the (BA_2') is true. This is because the propositional logic P→Q indicates (NOT(P) OR Q), then (NOT(P) OR Q) or (P→Q) is true when P is false; where P implies "the source processor is healthy" and (P→Q) implies BA_2'. Conversely, root *s* is a correct vertex by the definition of a correct vertex if the source processor is healthy. If all correct vertices' true values can be computed by CSA, then the true value of the root on *ic-tree* can be computed because the root is a correct vertex. By definition, the true value of the root is the initial value of the source processor if the source processor is healthy. Namely, each healthy service processor's root value is the initial value of the source processor; if the source processor is healthy, then BA_2' is true when the source processor is healthy. In short, the BA_1' and BA_2' are both true whether the source processor is healthy or fails, the interactive/BA problem is solved.

Lemma 1. *A dormant faulty service requirement can detected by means of forwarding technique used in a cloud computing.*

Proof. The healthy destination service processor can detect the message(s) from dormant faulty components if the protocol appropriately encodes a transmitted message by using the Manchester code [9] before transmission. Besides, there are at most $\lfloor(n-1)/3\rfloor$ non-λ value in a vector due to the faulty components are less than $\lfloor(n-1)/3\rfloor$ in a cloud computing environment.

Theorem 1. A healthy service processor can detect the dormant faulty processor in a cloud computing environment.

Proof. In the protocol CSA, there are σ rounds of message exchange in *message collected phase*, where $f_m \leq \lfloor(n-1)/3\rfloor$ and n>3, so there are at least two rounds of message exchange in the *message collected phase*. therefore, each processor can receive all other processors' messages in the network after two rounds of message exchange. According to the Lemma 1, each healthy service processor can detect the dormant faulty processor in a cloud computing environment.

Lemma 2. *All correct vertices of an ic-tree are common.*

Proof. After reorganization, no repeated vertices remain in an *ic-tree*. At the level of f_m +1 or above, the correct vertex α have at least $2f_m$ +1 children, out of which at least f_m +1 children are correct. The true value of these f_m +1 correct vertices is 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 than f_m+1. As a result, all correct vertices of the *ic-tree* are common.

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

Theorem 2. *The root of a healthy processor's ic-tree is common.*

Proof. By Lemma 2 and Corollary 1, the theorem is proven.

Theorem 3. *Protocol CSA solves the interactive/BA problem in cloud computing environment.*

Proof. To prove the theorem, one must show that CSA meets the constraints (BA_1') and (BA_2'). (BA_1'): Root *s* is common. By Theorem 2, (Agreement') is satisfied. (BA_2'): VOTE(*s*)=*v* for all healthy service processors, if the initial value of the source is v_s, say v=v_s. Since most of service processors are healthy, they transmit the message to all others. As a result, each of the correct vertices of the *ic-tree* is common (Lemma 2), and its true value is *v*. By Theorem 2, this root is common. The computed value VOTE(*s*) = *v* is stored in the root for all healthy service processors. (Validity') is satisfied.

5.2 Complexity of CSA

The complexity of CSA is judged in terms of: (1) the minimal number of rounds, (2) the maximum number of allowable faulty components, and (3) the number of exchanged messages.

Theorem 4. *CSA requires σ rounds to solve the consensus/BA problem by dual failure mode (containing malicious and dormant faults) in cloud computing environment if* $n>\lfloor(n-1)/3\rfloor+2f_m + f_d$ *and* $c>2f_m + f_d$, *where* $f_m \leq \lfloor(n-1)/3\rfloor$ *and* f_m+1 *(σ) are the minimum number of rounds of message exchange.*

Proof. Due to the fact that message passing is required in the message exchange phase, it is very time-consuming. Fischer and Lynch [7] pointed out that f_m+1 ($f_m \leq \lfloor(k-1)/3\rfloor$) rounds are the minimum number of rounds needed to obtain enough messages to achieve interactive/BA. The unit utilized by Fischer and Lynch [7] is the processor, so the number of required rounds of message exchange in cloud computing environment is σ ($\sigma\leq \lfloor(n-1)/3\rfloor+1$). Thus, CSA requires σ rounds and this number is the minimum.

Theorem 5. The total number of allowable faulty components by CSA is f_m malicious faulty processors and f_d dormant faulty processors, where $n> \lfloor(n-1)/3\rfloor+2f_m +f_d$ and $c>2f_m +f_d$.

Proof. According to the constraints of the BA problem for processors which was proposed by Siu et al. [13]. In this study, we use service processors to substitute the unit of Siu *et al.*, thus the constraints in our protocol are $n > [(n-1)/3] + 2f_m+f_d$, $c > 2f_m+f_d$. In the worst-case scenario, the malicious faulty service processors and dormant faulty service processor exist simultaneously, thus the total number of allowable faulty components by the CSA is f_m malicious faulty service processors and f_d dormant faulty service processors.

Theorem 6. The total number of messages in the CSA *is* cn^{σ}.

Proof. The protocol CSA must use the *ic-trees* to reach a set of common value. The *ic-tree* is constructed from the *message collected phase* of each round, and thus we have cn^{σ} (c represents a constant) messages in each *ic-tree*.

As a result, the CSA utilizes the minimum number of rounds and tolerates the maximum number of faulty processors of all systems in order to ensure that all healthy service processors reach a common agreement. The superiority of the protocol is thus proven.

6 Conclusion

In this study, the service chip and random key can be used to recognize the type of service safely in a cloud computing environment with respect to dual failure mode in fallible processors. Besides, the proposed protocol CSA is more adapting to changeability of network than, such as the mobility of processors than previous works [11, 13, 16, 22]. Furthermore, the loading of service processors may overload with a large number of requirements. For reducing workload of service processors, the service requirements of user in this paper can be relocated and dispatched to appropriate service processor safely and efficiently after a service agreement is obtained.

Finally, the proposed protocol can tolerate a maximal number of faulty processors in a minimal number of message exchanges under dual failure mode by using minimal number of message exchange. The reliability of cloud computing environment is enhanced and is more efficient than the others [7, 10-11, 13, 16, 18, 20-22].

However, the requests of users are allocated to different places and are sent to processor at anytime. In the future, the type of transmission medium failure will be considered to enhance the reliability in a cloud computing environment.

References

[1] I. Abraham, D. Dolev, Byzantine Agreement with Optimal Early Stopping, Optimal Resilience and Polynomial Complexity, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, Portland, Oregon, 2015, pp. 605-614.

[2] A. Bar-Noy, D. Dolev, C. Dwork, H. R. Strong, Shifting Gears: Changing Algorithms on the Fly to Expedite Byzantine Agreement, *Information and Computation*, Vol. 97, No. 2, pp. 205-233, April, 1992.

[3] C. Cachin, S. Schubert, M. Vukolić, *Non-determinism in Byzantine Fault-tolerant Replication*, arXiv preprint arXiv: 1603.07351, 2016.

[4] N. Deo, *Graph Theory with Applications to Engineering and Computer Science*, Prentice-Hall, 1974.

[5] D. Dolev, R. Reischuk, Bounds on Information Exchange for Byzantine Agreement, *Journal of the ACM (JACM)*, Vol. 32, No. 1, pp. 191-204, January, 1985.

[6] M. J. Fischer, The Consensus Problem in Unreliable Distributed Systems (A Brief Survey), *Proceedings of the 1983 International FCT-Conference on Fundamentals of Computation Theory*, Berlin, Heidelberg, 1983, pp. 127-140.

[7] M. J. Fischer, N. A. Lynch, A Lower Bound for the Time to Assure Interactive Consistency, *Information Processing Letters*, Vol. 14, No. 4, pp. 183-186, June, 1982.

[8] C. Gong, J. Liu, Q. Zhang, H. Chen, Z. Gong, The Characteristics of Cloud Computing, *Proceedings of the 39th International Conference on Parallel Processing Workshops*, San Diego, CA, 2010, pp. 275-279.

[9] F. Halsall, *Data Communications, Computer Networks and Open Systems*, 4th ed., Addison-Wesley, 1995.

[10] L. Lamport, R. Shostak, M. Pease, The Byzantine Generals Problem, *ACM Transactions on Programming Languages and Systems*, Vol. 4, No. 3, pp. 382-401, July, 1982.

[11] A. Mostéfaoui, H. Moumen, M. Raynal, Signature-free Asynchronous Binary Byzantine Consensus with $t < n/3$, $O(n^2)$ Messages, and O(1) Expected Time, *Journal of the ACM (JACM)*, Vol. 62, No. 4, Article No. 31, August, 2015.

[12] P. K. Sangdeh, M. Mirmohseni, F. Poursabzi, Applying the Byzantine Agreement in Wireless Sensor Networks Based on Clustering, *2015 IEEE 23rd Iranian Conference on Electrical Engineering*, Tehran, Iran, 2015, pp. 619-624.

[13] H. S. Siu, Y. H. Chin, W. P. Yang, A Note on Consensus on Dual Failure Modes, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 7, No. 3, pp. 225-230, March, 1996.

[14] A. S. Tanenbaum, *Computer Networks*, 4th ed., Prentice-Hall, 2003.

[15] L. M. Vaquero, L. R. Merino, J. Caceres, M. Lindner, A Break in the Clouds: Towards a Cloud Definition, *ACM SIGCOMM Computer Communication Review*, Vol. 39, No. 1, pp. 50-55, January, 2009.

[16] S. S. Wang, S. C. Wang, The Consensus Problem with Dual Failure Nodes in a Cloud Computing Environment, *Information Sciences*, Vol. 279, pp. 213-228, September, 2014.

[17] S. S. Wang, K. Q. Yan, S. C. Wang, An Optimal Solution for Byzantine Agreement under a Hierarchical Cluster-oriented Mobile Ad-hoc Network, *Computers and Electrical Engineering*, Vol. 36, No. 1, pp. 100-113, January, 2010.

[18] S. C. Wang, S. S. Wang, K. Q. Yan, L. H. Chang, C. P. Huang, Reaching Fast Agreement in a Generalized Cloud Computing Environment, *Journal of Internet Technology*, Vol. 11, No. 7, pp. 975-984, December, 2010.

[19] S. C. Wang, K. Q. Yan, C. L. Ho, S. S. Wang, The Optimal Generalized Byzantine Agreement in Cluster-based Wireless Sensor Networks, *Computer Standards & Interfaces*, Vol. 36, No. 5, pp. 821-830, September, 2014.

[20] S. C. Wang, K. Q. Yan, S. S. Wang, G. Y. Zheng, Reaching Agreement among Virtual Subnets in Hybrid Failure Mode, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 19, No. 9, pp. 1252-1262, September, 2008.
[21] K. Q. Yan, S. C. Wang, Grouping Byzantine Agreement, *Computer Standards & Interfaces*, Vol. 28, No. 1, pp. 75-92, July, 2005.
[22] K. Q. Yan, S. S. Wang, S. C. Wang, Reaching an Agreement under Wormhole Networks within Dual Failure Component, *International Journal of Innovative Computing, Information and Control*, Vol. 6, No. 3(A), pp. 1151-1164, March, 2010.
[23] S. Zhang, S. Zhang, X. Chen, X. Huo, Cloud Computing Research and Development Trend, *Proceedings of Second International Conference on Future Networks*, Sanya, Haina, China, 2010, pp. 93-97.
[24] Amazon, *Amazon Elastic Compute Cloud*, http://aws.amazon.com/ec2.
[25] Gartner, *Gartner Says Cloud Computing Will Be As Influential As E-business*, http://www.gartner.com/it/page.jsp?id=707507/.
[26] Google, *Google App Engine*, http://code.google.com/appengine/.
[27] IBM, *IBM Blue Cloud Project [URL]*, http://www03.ibm.com/ press/us/en/pressrelease/22613.wss/.
[28] Microsoft, *Windows Azure*, http://www.microsoft.com/windowsazure/windowsazure/

Biography

Mao-Lun Chiang received the Ph.D. degree in Department of Computer Science from National Chung-Hsing University, Taiwan. He is an associate professor in the Department of Information and Communication Engineering at the Chaoyang University of Technology, Taiwan. His current research interests include mobile computing, fault tolerant computing, and cloud computing.