RESEARCH ARTICLE
doi: 10.2306/scienceasia1513-1874.2013.39S.085

ScienceAsia 39S (2013): **85–94**

Virtual subnet agreement protocol in a cloud storage environment

Kuo-Qin Yan[a], **Yu-Fu Yang**[a], **Shu-Ching Wang**[b,*], **Guang-Yan Zheng**[c]

[a] Department of Business Administration, Chaoyang University of Technology, Taiwan
[b] Department of Information Management, Chaoyang University of Technology, Taiwan
[c] Department of Cloud System Software Institute, Institute for Information Industry, Taiwan

*Corresponding author, e-mail: scwang@cyut.edu.tw

Received 7 Jan 2013
Accepted 5 Apr 2013

ABSTRACT: Fault-tolerance is an important research topic in the study of distributed systems. In a distributed system, cooperating tasks must achieve an agreement. This is to cope with the influence of faulty components. Reaching a common agreement in the presence of faults before performing certain tasks is essential. Nowadays, network bandwidth and hardware technology are developing rapidly, resulting in the vigorous development of the internet. However, cloud computing, an internet-based development in which dynamically scalable and often virtualized resources are provided as a service over the internet has become a significant issue. In addition, the Byzantine Agreement (BA) problem is a fundamental problem in fault-tolerant distribute systems. In previous studies, the BA algorithm was used in traditional network topology. However, this algorithm does not perform well in dynamically changing networks. To enhance fault-tolerance, a new protocol VSACS (virtual subnet agreement of cloud storage) is proposed to solve the BA problem in this study. VSACS uses the minimum number of message exchange rounds to make all correct processors agree on a common value and can tolerate the maximum number of allowable faulty components.

KEYWORDS: Byzantine agreement, fault tolerant, distributed system

INTRODUCTION

Cloud computing has become a significant technology trend, many applications of cloud computing increase convenience for users, e.g., Google Storage, Dropbox or Microsoft SkyDrive, and so on. Furthermore, cloud storage (or data centre, data storage as a service) is the concept of storage behind an interface where the storage can be administered on demand [1,2]. However, one of the fundamental cloud storage issues is replication, which combined with cloud computing based infrastructure; the target mobile processors connected to cloud service provider, listen to some tasks from server and application recovery is in needed [2].

As wireless and cloud computing have become increasingly popular, network topology has shown a trend towards wireless connectivity, thus providing enhances support for cloud computing. Shortly, this technological trend has greatly encouraged distributed system design and lent practical support to mobile processors. The virtual subnet has attracted significant attention recently because they require less infrastructure, they can be deployed quickly, and they can automatically adapt to changes in topology. Hence virtual subnets suit military communication, emergency disaster rescue operations, and law enforcement [3].

The reliability of the mobile processor is one of the most important aspects in virtual subnet. In order to provide a reliable cloud storage application within replication management of cloud in a virtual subnet, a mechanism to allow a set of mobile processors to agree on an agreement value is required. The Byzantine Agreement (BA) problem [4,5] is one of the most fundamental problems to reach an agreement value in a distributed system, like cloud computing [1]. The original BA problem defined by Lamport et al [6]., is assumed as follows:

(1) There are $n \geqslant 4$ processors in a synchronous distributed system.

(2) Each processor can communicate with each other through reliable fully connected network.

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

(4) After message exchange, all correct processors should reach a common agreement, if and only if the number of faulty processors t is less than

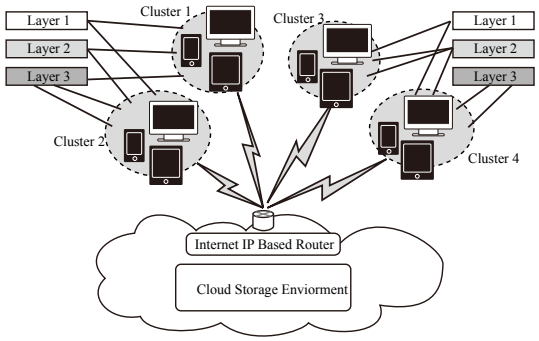


Fig. 1 The topology of virtual subnet within cloud storage.

one-third of the total number of processors in the network ($t \leqslant (n-1)/3$).

The traditional BA problem focused on fixed and well-defined networks[4,5]. However, the network structure of a virtual subnet is not fixed, and its topology can change as a result of this mobility. Hence, the traditional solutions to the BA problem are not suited to the topology of virtual subnet within a cloud storage environment.

In this study, the BA problem is revisited in the topology of virtual subnet within cloud storage environment. The proposed protocol is called the virtual subnet agreement of cloud storage (VSACS). VSACS allows each correct processor in the topology of virtual subnet within cloud storage environment reach an agreement value.

RELATED WORK

Nowadays, the virtual subnet is practical more and more due to it can provide processors join to the network or leave anytime with non-infrastructure. A group of multiple processors in virtual subnet is cooperating to achieve some objectives; each processor communicates with other processors by using the broadcast in virtual subnet, but also leads to a severe problem, such as broadcast storm[7]. Many researchers proposed cluster schemes, broadcast limited and virtual subnet[3,8] to improve the broadcast storm[9]. However, virtual subnet has been a more important topic than the other topics recently[10]. The virtual subnet is composed of several groups by overlap network approach[3,7]. Fig. 1 shows a topology of virtual subnet within cloud storage environment. There are three situations that the processors communicate underlying virtual subnet:

(1) Processors in the same group communicate to each other directly by virtual backbone.

(2) Processors in different groups exchange messages with each other via virtual subnet or physical communication media (internet IP based), e.g., host agent communication.

(3) Host agent processors can communicate with the cloud service via physical communication media in different groups and exchange messages with each other via virtual subnet or physical communication media (internet IP based).

In addition, virtual backbone can be used to (1) collect topology information for routing; (2) provide a backup route; (3) multicast or broadcast messages[7] and so on. Hence, virtual subnet can improve the broadcast storm efficiently.

However, the symptoms of processor failure in the topology of virtual subnet within cloud storage environment can be classified into dormant fault (crash or omission) and malicious fault (also called Byzantine fault)[11,12]. The dormant fault means the processor does not work correctly (i.e., fails to send or receive a required message on time). In the malicious fault, the processor can do anything arbitrary at any time. Thus the behaviour of a mobile processor with malicious fault is unpredictable. According to above description about malicious fault, the malicious fault generates any kind of failures randomly, it is the most severe failure type, and causes the worst problem in distributed system. The dual failure mode on processor is one where both dormant fault and malicious fault exist simultaneously with the faulty processor in the distributed system.

The topology of virtual subnet within cloud storage environment is different with the traditional network topology, so the pervious protocols of BA are not suit for the topology of virtual subnet within cloud storage environment. As the result, the new protocol that we proposed can use a minimum number of message exchanges and can tolerate a maximum number of allowable faulty components to make each correct processor reach a common agreement in the topology of virtual subnet within cloud storage environment with dual failure processors.

THE PROPOSED PROTOCOL

In the distributed system, the purpose of the BA protocol is to make all correct processors to reach a common agreement. For the reason, processors should exchange messages to all other processors. Each correct processor receives messages from other processors by a number of rounds of message exchange. Afterwards, all correct processors can get enough messages to make a decision value that is

called agreement value or common value. Thus all correct processors agree on the same value.

The notation and parameters of the proposed protocol VSACS in the topology of virtual subnet within cloud storage environment are shown in following:

(1) Let N be the set of all processors in the network and $|N| = n$, where n is the number of processors in the underlying network.

(2) Let G be the set of all groups in the network and $|G| = g$, where g is the number of groups in the underlying network and $g \geqslant 4$.

(3) Let x, y be the group identifier where $1 \leqslant x$, $y \leqslant g$ and $g \geqslant 4$.

(4) Let η_x be the number of processors in group Gp_x, $0 \leqslant x \leqslant g$. If there are at least $\lceil \eta_x/2 \rceil$ malicious faulty processors in Gp_x, then Gp_x will be the malicious faulty group. If there are at least $\lceil \eta_x/2 \rceil$ dormant faulty processors in Gp_x, then Gp_x will be the dormant faulty group.

(5) Let f_{m} be the total number of malicious faulty processors.

(6) Let f_{d} be the total number of dormant faulty processors.

(7) Let f_{Gm} be the maximum number of malicious faulty groups allowed.

(8) Let f_{Gd} be the maximum number of dormant faulty groups allowed.

(9) Let T_{FP} be the total number of allowable faulty processors, $T_{\mathrm{FP}} = f_{\mathrm{m}} + f_{\mathrm{d}}$.

(10) Let T_{FG} be the total number of allowable faulty groups, $T_{FG} = F_{\mathrm{Gm}} + F_{\mathrm{Gd}}$.

(11) Let η_x be the number of processors in Gp_x, $0 \leqslant x \leqslant g$.

(12) Let c be the connectivity of the virtual subnet, where c is $g - 1$.

In BA protocol, the first step is to count the number of required rounds of messages exchange, which is determined by the total number of processors at the beginning of protocol execution. Hence if the variety of faulty processors can be discovered, then the number of rounds of messages exchange can be reduced, and then the fault tolerance capability is moved up.

Procedure iTRANSMISSION
Defmition: 1. For the virtual subnet, each processor has the common knowledge of entire graphic information $\hat{G}$ = (E, Gp), where Gp is the set of groups in the network and E is a set of group pairs (Gp_x , Gp_y) indicating a physical communication medium (the sensing is covered) between group Gp_x and group Gp_y. 2. Each processor communicates with all other processors via virtual subnet, virtual backbone or physical communication media [2,11]. 3. The processor plays sender, receiver or agent, depends on the behavior of which kinds of iTRANSMISSION [2]. 4. The host agent processor communicates with cloud service via physical communication media (Internet based). 5. The host agent processor cannot garble the message between the sender processor and receiver processor; this assumption has achieved by the technology of encryption (such as RSA [8]).
Step 1. The sender processor P_i ($1 \leq i \leq n$) transmits the value v_i to the receiver group. Step 2. If the group-disjoint path from sender processor to destination group passes through any dormant faulty processor or if the sender processor has dormant faults, then stores λ^0 itself. Step 3. The processors in the receiver group take the local majority value from the same group paths and then construct the vector $V_i = [\, v_{path\,1}, v_{path\,2}, \ldots, v_{path\,c-1}, v_{path\,c}]$. Step 4. The processors in the destination group apply VMAJ on vector V_i.
Function VMAJ (for each vector V_i) 1. Count the received value: Take the majority 2. If the majority value is λ^0 and the number of value λ^0 is greater than or equal to $c - \lfloor (g-1)/3 \rfloor$, then output the value λ^0. 3. Else set majority value m, where m$\in\{0, 1\}$. If the majority value does not exist, then Output the maiority value λ^0. Otherwise, output the majority m, where m$\in\{0, 1\}$.

Fig. 2 The procedure iTRANSMISSION.

The proposed protocol VSACS can solve the BA problem due to faulty processor(s), which may send wrong messages to influence the system to reach agreement in the topology of virtual subnet within cloud storage environment. By using the proposed protocol VSACS and procedure iTRANSMISSION, all correct processors in the topology of virtual subnet

VSACS (source processor with initial value vs)

Pre-Execute. Computes the number of rounds required c←⌊(g-1)/3⌋

Message Exchange Phase:
Case $\theta = 1$, run
A. The source processor transmits its initial value vs to each group's processors by using iTRANSMISSION.
B. Each receiver processor obtains the value and stores it in the root of its mg-tree.
C. If the source processor is dormant fault, and then the value "λ^0" has replaced the initial value received from source processor.
Case $\theta > 1$, run
A. Each processor without the source processor uses ITRANSMISSION to transmit the values at level θ-1 in its mg-tree to each group's processors. If the value at level θ-1 is "λ^i", and the value λ^i will be replaced by λ^{i+1}, where $0 \leq i \leq TFG-1$.
B. Each receiver processor applies RMAJ on its received messages and stores RMAJ value in the corresponding vertices at level θ of its mg-tree.

Decision Making Phase:
Step 1. Reorganizing the mg-tree into a corresponding ic-tree. (The vertices with repeated group names are deleted).
Step 2. Using function VOTE on the root s of each processor's ic-tree, then the common value VOTE(s) has obtained.
Function RMAJ(V)
1. The majority value in the vector $Vi = [v_1, \cdots, v_{\eta x-1}, v_{\eta x}]$, if it exists.
2. Otherwise, choosing a default value (Φ).

Function VOTE(μ)
1. If the μ is a leaf or the number of value λ^0 is $3 * (TFG - \theta + 1) + (g-1) \% 3$, then output μ. //* Rule 1 *//
2. Else if the majority value is not existed, then output Φ. //* Rule 2 *//
3. Else if the majority value is λ^i, where $1 \leq i \leq TFG$, then output λ^{i+1}. //* Rule 3 *//
4. Otherwise, output m, where $m \in \{0, 1\}$ //* Rule 4 *//

Fig. 3 The proposed protocol VSACS.

within cloud storage environment can reach a common agreement only requires θ rounds of messages exchange, where $\theta = \lfloor (g-1)/3 \rfloor + 1$. The procedure iTRANSMISSION used to transmit messages in VSACS. Based on the distinctions of the topology of virtual subnet within cloud storage environment, iTRANSMISSION can provide a virtual channel for each processor to transmit messages to each other without influence from faulty inter-communication medium. The description of iTRANSMISSION is shown in Fig. 2.

The proposed protocol VSACS is organized as two phases with the procedure iTRANSMISSION. One is message exchange phase and the other is decision making phase. In the first round of the message exchange phase, the source processor sends its initial value to all processors by using iTRANSMISSION, and then receiver processor 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[13]. If the source processor is dormant fault, then the value λ^0 has replaced the initial value from source processor. After the first round of message exchange phase ($\theta > 1$), each processor without source processor uses iTRANSMISSION to transmit the value at level $\theta - 1$ in its mg-tree to all processors. If the value at level $\theta - 1$ is λ^i (the value λ^i is used to report absent value), then the value λ^i will be replaced by λ^{i+1}, where $0 \leqslant i \leqslant T_{FG} - 1$. At the end of each round, the receiver processor uses the function RMAJ on it received VMAJ values, which are from the same group by iTRANSMISSION, to get a single value. Moreover, each receiver processor stores the received messages (VMAJ values) and function RMAJ value in its mg-tree.

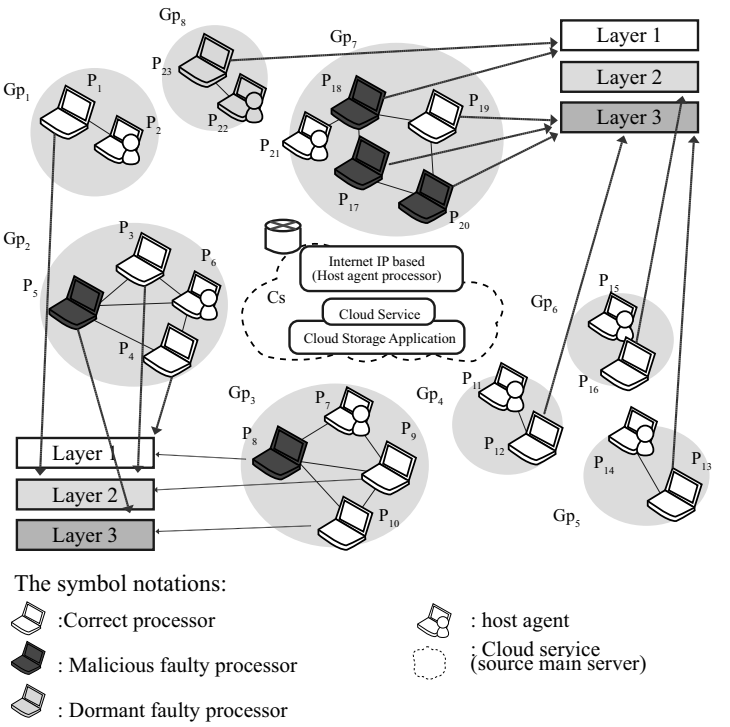
Subsequently, in the decision making phase, each processor without the source processor reorganizes its mg-tree into a corresponding ic-tree[13]. The ic-tree is a tree structure that is used to store a received message without repeated group names[14]. Hence the common value VOTE(s) has obtained by using function VOTE on the root s of each processor's ic-tree. The function VOTE counts the non-value λ^0 (expert the last level of the ic-tree) for all vertexes at the θth level of an ic-tree, where $1 \leqslant \theta \leqslant T_{FG} + 1$. The conditions in the function VOTE are similar to conventional majority vote[15,16]. The detail steps of the proposed protocol VSACS is presented in Fig. 2.

AN EXAMPLE OF VSACS EXECUTED

An example is given to execute our protocol VSACS, the topology of virtual subnet within cloud storage environment is shown in Fig. 4. There are 23 processors falling into eight groups. Gp_1 includes source processor P_1 and P_2. Gp_2 includes P_3, P_4, P_5 and P_6. Gp_3 includes P_7, P_8, P_9 and P_{10}. Gp_4 includes P_{11} and P_{12}. Gp_5 includes P_{13} and P_{14}. Gp_6 includes P_{15} and P_{16}. P_{17}, P_{18}, P_{19}, P_{20} and P_{21} belong to Gp_7, P_{22} and P_{23} belong to Gp_8.

In BA problem with fallible processors, the worst situation is that the source processor is no longer honest[17]. Simply, here is the worst case example. Suppose the application server of cloud storage is the source processor C_s, which has a malicious fault. C_s may send arbitrarily different replication commands to different groups. Hence in order to solve the BA problem among correct processors of the example, VSACS requires θ ($\lfloor (g-1)/3 \rfloor + 1$) rounds of message exchange phase. Pre-Execute counts the number of rounds required before the message exchange phase in VSACS. There needs to be three ($\lfloor (8-1)/3 \rfloor + 1 = 3$) rounds to message exchange for this example.

The source processor C_s uses iTRANSMISSION to transmit replication commands to all other processors in the first round of the message exchange phase. The replication command obtained of each correct processor is listed in Fig. 5. In the σth ($1 < \sigma \leqslant \theta$) round of message exchange, except for the C_s, each processor uses iTRANSMISSION to transmit RMAJ



The messages sent from the source processor by iTRANSMISSION and then start to execute VSACS.

- The source processor Cs (Cloud-Storage Application Source) is a malicious faulty processor.
- Cs sends the replication processing command (let this command is 1) to the processors of Gp2, Gp4, Gp5, Gp6, Gp7 and Gp8.
- Cs sends replication process c command (let this command is 0) to the processors of Gp1 and Gp3.

Fig. 4 The initial status of executing VSACS.

	Level 1 Root s
Cp_1's Correct Processors	0
Cp_2's Correct Processors	1
Cp_3's Correct Processors	0
Cp_4's Correct Processors	1
Cp_5's Correct Processors	1
Cp_6's Correct Processors	1
Cp_7's Correct Processors	1
Cp_8's Correct Processors	1

Fig. 5 The mg-tree of each processor at the first round.

Level 1 Root s	Level 2	Function VMAJ	
s	s1	0	(0, 0)
1	s2	1	(1, 1, 0, 1)
RMAJ ⟵	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)
	s8	λ^0	(λ^0, λ^0)

Fig. 6 The mg-tree of correct processor P_1 at the second round.

values at the $(\sigma - 1)$-th level in its mg-tree to all other processors and itself. Subsequently, each receiver processor applies RMAJ to its received messages

Level 1	Level 2	Level 3		Function VMAJ
s	s1	s11	0	(0,0)
0	0(0,0)	s12	0	(0,0,0,0)
	RMAJ →	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)
		s18	λ^0	(λ^0,λ^0)
	s2	s21	1	(1,1)
	1(1,1,1,1)	s22	1	(1,1,1,1)
	RMAJ →	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)
		s28	λ^0	(λ^0,λ^0)
	s3	s31	0	(0,0)
	0(0,0,0,0)	s32	0	(0,0,1,0)
	RMAJ →	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)
		s38	λ^0	(λ^0,λ^0)
	s4	s41	1	(1,1)
	1(1,1)	s42	1	(1,1,0,1)
	RMAJ →	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)
		s48	λ^0	(λ^0,λ^0)
	s5	s51	1	(1,1)
	1(1,1)	s52	1	(1,1,1,1)
	RMAJ →	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)
		s58	λ^0	(λ^0,λ^0)
	s6	s61	1	(1,1)
	1(1,1)	s62	1	(1,1,1,1)
	RMAJ →	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)
		s68	λ^0	(λ^0,λ^0)
	s7	s71	0	(0,0)
	0(0,0,1,0,1)	s72	1	(1,1,1,1)
	RMAJ →	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)
		s78	λ^0	(λ^0, λ^0)
	s8	s81	λ^1	(λ^1,λ^1)
	λ^0(λ^0, λ^0)	s82	λ^1	(λ^1,λ^1,λ^1,λ^1)
	RMAJ →	s83	λ^1	(λ^1,λ^1,λ^1,λ^1)
		s84	λ^1	(λ^1,λ^1)
		s85	λ^1	(λ^1,λ^1)
		s86	λ^1	(λ^1,λ^1)
		s87	0	(0,0, λ^0,0,λ^0)
		s88	λ^0	(λ^0,λ^0)

Fig. 7 The final mg-tree of processor P_1 after the message exchange phase.

Level 1	Level 2	Level 3		
s	s1			
0	0(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)
		s18	λ^0	(λ^0,λ^0)
	s2	s21	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)
		s28	λ^0	(λ^0,λ^0)
	s3	s31	0	(0,0)
	0(0,0,0,0)	s32	0	(0,0,1,0)
		s34	0	(0,0)
		s35	0	(0,0)
		s36	0	(0,0)
		s37	0	(0,0,1,0,1)
		s38	λ^0	(λ^0,λ^0)
	s4	s41	1	(1,1)
	1(1,1)	s42	1	(1,1,0,1)
		s43	1	(1,1,1,1)
		s45	1	(1,1)
		s46	1	(1,1)
		s47	1	(1,1,1,0,1)
		s48	λ^0	(λ^0,λ^0)
	s5	s51	1	(1,1)
	1(1,1)	s52	1	(1,1,1,1)
		s53	1	(1,0,1,1)
		s54	1	(1,1)
		s56	1	(1,1)
		s57	0	(0,0,1,0,1)
		s58	λ^0	(λ^0,λ^0)
	s6	s61	1	(1,1)
	1(1,1)	s62	1	(1,1,1,1)
		s63	1	(1,1,1,1)
		s64	1	(1,1)
		s65	1	(1,1)
		s67	1	(1,1,1,1,1)
		s68	λ^0	(λ^0,λ^0)
	s7	s71	0	(0,0)
	0(0,0,1,0,1)	s72	1	(1,1,1,1)
		s73	0	(0,0,0,0)
		s74	1	(1,1)
		s75	0	(0,0)
		s76	1	(1,1)
		s78	λ^0	(λ^0, λ^0)
	s8	s81	λ^1	(λ^1,λ^1)
	λ^0(λ^0, λ^0)	s82	λ^1	(λ^1,λ^1,λ^1,λ^1)
		s83	λ^1	(λ^1,λ^1,λ^1,λ^1)
		s84	λ^1	(λ^1,λ^1)
		s85	λ^1	(λ^1,λ^1)
		s86	λ^1	(λ^1,λ^1)
		s87	0	(0,0, λ^0,0,λ^0)

Fig. 8 The ic-tree of processor P_1.

and stores the received messages (VMAJ values) and RMAJ values at the corresponding vertices at level σ of its mg-tree. The mg-tree of correct processor P_1 at the second and final round in the message exchange phase are shown in Fig. 6 and Fig. 7.

After the message exchange phase, the tree structure of each correct processor is converted from mg-tree to ic-tree by deleting the vertices with duplicated names. The example ic-tree h (Fig. 8). Eventually, using the function VOTE to root the value s for each correct processor's ic-tree VOTE(s) = VOTE($s1$), ...,

VOTE(s1) = (VOTE(s12),VOTE(s13),VOTE(s14),VOTE(s15),VOTE(s16),VOTE(s17),VOTE(s18))
= (0, 0, 0, 0, 0, 1, λ^0) = 0
VOTE(s2) = (VOTE(s21),VOTE(s23),VOTE(s24),VOTE(s25),VOTE(s26),VOTE(s27),VOTE(s28))
= (1, 1, 1, 1, 1, 0, λ^0) = 1
VOTE(s3) = (VOTE(s31),VOTE(s32),VOTE(s34),VOTE(s35),VOTE(s36),VOTE(s37),VOTE(s38))
= (0, 0, 0, 0, 0, 0, λ^0) = 0
VOTE(s4) = (VOTE(s41),VOTE(s42),VOTE(s43),VOTE(s45),VOTE(s46),VOTE(s47),VOTE(s48))
= (1, 1, 1, 1, 1, 1, λ^0) = 1
VOTE(s5) = (VOTE(s51),VOTE(s52),VOTE(s53),VOTE(s54),VOTE(s56),VOTE(s57),VOTE(s58))
= (1, 1, 1, 1, 1, 0, λ^0) = 1
VOTE(s6) = (VOTE(s61),VOTE(s62),VOTE(s63),VOTE(s64),VOTE(s65),VOTE(s67),VOTE(s68))
= (1, 1, 1, 1, 1, 1, λ^0) = 1
VOTE(s7) = (VOTE(s71),VOTE(s72),VOTE(s73),VOTE(s74),VOTE(s75),VOTE(s76),VOTE(s78))
= (0, 1, 0, 1, 0, 1, λ^0) = Φ
VOTE(s8) = (VOTE(s81),VOTE(s82),VOTE(s83),VOTE(s84),VOTE(s85),VOTE(s86),VOTE(s87))
= (0, 1, 0, 1, 0, 1, λ^0) = Φ

VOTE(s)=(VOTE(s1),VOTE(s2),VOTE(s3),VOTE(s4),VOTE(s5),VOTE(s6),VOTE(s7),VOTE(s8))
=(0, 1, 0, 1, 1, 1, Φ, Φ) = 1

Fig. 9 The common value VOTE(s) by correct processor P_1.

VOTE($s8$) = 1, an agreement value 1 can be obtained, (Fig. 9), and the decision making phase has completed.

VSACS CORRECTNESS AND COMPLEXITY

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

Correctness of VSACS

In order to prove the correctness of the proposed protocol, a vertex μ is called common[3] if each correct processor has the same value for μ. That is to say, if vertex μ is common, then the value stored in vertex μ of each correct processor mg-tree or ic-tree is identical. When each correct processor has a common initial value of the source processor in the root of an ic-tree, if root s of an ic-tree in a correct processor is common and the initial value received from the source processor is stored in the root of the tree structure, then an agreement can be reached because the root is common. Thus the constraints, Agreement and Validity can be said as

Agreement: Root s is common, and

Validity: VOTE(s) = v_s for each correct processor, if the source processor is fault-free.

To prove that a vertex is common, the term common frontier[3] is defined as follows: When every root-to-leaf path of the mg-tree contains a common vertex, the collection of the common vertices forms a common frontier. In other words, every correct processor has the same messages collected in the common frontier if a common frontier does exist in a correct processor's mg-tree. Subsequently, using the same function VOTE to compute the root value of the tree structure, every correct processor can compute the same root value because the same input (the same collected messages in the common frontier) and the same computing function will produce the same output (the root value).

Lemma 1 *Correct destination processor can detect the influence of the messages through dormant faulty components.*

Proof: If the protocol appropriately encodes a transmitted message by using either the NRZ Code[18] (Non-Return-to-Zero Code) or the Manchester code[18] before iTRANSMISSION, then the correct destination processor can detect the messages from dormant faulty components. □

Theorem 1 *Each processor can receive messages without influences of faulty components between the sender processor via iTRANSMISSION in each round and* $g > 2F_{\mathrm{Gm}} + F_{\mathrm{Gd}}$

Proof: Inasmuch as Lemma 1, we can eliminate the influences of the dormant faulty components between any pair of processors in each round of message exchange. Hence we can ignore the influences of the malicious faulty components between any pairs of processors in each round of message exchange and $g > 2F_{\mathrm{Gm}}+F_{\mathrm{Gd}}$. The reason is that the correct sender processor sends g copies of a message to correct destination processors. In the worst case, a correct destination processor can receive $g - F_{\mathrm{Gd}}$ messages transmitted via the correct sender processor. A correct destination processor can decide which the correct messages are by taking the majority value since NRZ-Code or Manchester code[18] can detect the dormant faulty components and $g > 2F_{\mathrm{Gm}} + F_{\mathrm{Gd}}$. □

Lemma 2 *By iTRANSMISSION, the correct destination processor can detect the dormant faulty sender processor.*

Proof: When the sender processor is in dormant fault, thus the number of value λ^0 must be greater than or equal to $(g-1) - \lfloor (g-1)/3 \rfloor$. That is, there are at most $\lfloor (g-1)/3 \rfloor$ malicious faulty components in the network. Hence there are at most $\lfloor (g-1)/3 \rfloor$ without value λ^0 in the vector V_i. □

Theorem 2 *In the network, each correct processor can detect the dormant faulty processor.*

Proof: There are at least θ rounds of message exchange in the protocol VSACS, where $\theta \leqslant \lfloor (g-1)/3 \rfloor + 1$ and $g \geqslant 4$. For the reason, there are at least two rounds of message exchange in the message exchange phase. Each correct processor can receive the message from the source processor in the first round of message exchange and receives other processors' message(s) in the second round of message exchange. Each processor can receive all other processor' message(s) in the network without source processor after at least two rounds of message exchange. In terms of Lemma 2, each correct processor can detect the dormant faulty processor in the network. □

Lemma 3 *In an ic-tree, all correct vertices are common.*

Proof: When the tree structure has conversed from mg-tree to ic-tree, there is no duplicate vertex in the ic-tree. At the level θ or upon, the correct vertex μ has at least $2\theta - 1$ children, in which at least θ children are correct. The real value of these θ correct vertices is common, and the majority value of vertex μ is common. The correct vertex μ is common in the ic-tree if the level of μ is less than θ. For the reason, all correct vertices of ic-tree are common. □

Lemma 4 *The common frontier exists in the ic-tree.*

Proof: There are θ vertices along each root-to-leaf path of an ic-tree that the root is marked by the origin name, and the others are marked by an order of group name, so that most $\theta - 1$ groups can be failed, at least one vertex is correct along each root-to-leaf path of the ic-tree. The correct vertex is common, and the common frontier exists in each correct processor's ic-tree by Lemma 3. □

Lemma 5 *Let μ be a vertex, and μ is common if there is a common frontier in the subtree rooted at μ.*

Proof: When the height of μ is 0, and the common frontier exists, thus μ is common. If the height of μ is θ, the children of μ are all in common by using induction hypothesis with the height of the children at $\theta - 1$, then the vertex μ is common. □

Corollary 1 *If the common frontier exists in the ic-tree, then the root is common.*

Theorem 3 *The root of a correct processor's ic-tree is common.*

Proof: Lemma 3, Lemma 4, Lemma 5 and Corollary 1 prove the theorem 3. □

Theorem 4 *In the topology of virtual subnet within cloud storage, the proposed protocol VSACS solves the BA problem.*

Proof: Inasmuch as the theorem has to describe that VSACS meets the constraints (Agreement' and Validity'). **Agreement:** Root s is common, and by Theorem 3, Agreement' is satisfied. **Validity:** VOTE(s) = v for each correct processor, if the initial value of the source processor is v_s, then $v = v_s$. iTRANSMISSION to communicate with all others. The value of correct vertices for all correct processors' mg-tree is v. When the tree structure has converted from mg-tree to ic-tree, the correct vertices still exist. Hence every correct vertices of the ic-tree is common by Lemma 3, and its true value is v. This root is common by Theorem 3. The value v is computed by VOTE(s), and it value v is stored in the root for all correct processors. Validity' is satisfied. □

Complexity of VSACS

The complexity of VSACS is evaluated in terms of (1) the minimal number of rounds; and (2) the maximum number of allowable faulty components. Theorems 5 and 6 are shown that the optimal solution is reached.

Theorem 5 *VSACS requires θ rounds to complete and solve the BA problem with dual fallible processors virtual subnet within cloud storage environment if $g > \lfloor (g-1)/3 \rfloor + 2F_{\mathrm{Gm}} + F_{\mathrm{Gd}}$ where $\lfloor (g-1)/3 \rfloor + 1 \geqslant \theta$, and θ are the minimum number of rounds of message exchange.*

Proof: The message passing is required in the message exchange phase; in other words, the phase is a period-consuming phase. Wang et al[13]. argued out that $t+1$ (where $t \leqslant (g-1)/3$) rounds are the minimum number of rounds to get enough messages to reach BA with processor fault. The network topology

of Wang et al[13]. is traditional network architecture, and the unit of Wang et al is group-oriented network; In the topology of virtual subnet within cloud storage environment, there are several groups by overlap network approach[3,7]. Hence, the number of required rounds of message exchange in the topology of virtual subnet within cloud storage environment is θ (where $\theta \geqslant \lfloor (g-1)/3 \rfloor + 1$). Hence, VSACS requires θ rounds and the number θ is minimum. □

Theorem 6 *The total number of allowable faulty components by VSACS is* F_{Gm} *malicious faulty groups and* F_{Gd} *dormant faulty groups, where* $g > \lfloor (g-1)/3 \rfloor + 2F_{\mathrm{Gm}} + F_{\mathrm{Gd}}$.

Proof: In the past, Wang et al pointed out that the constraints of BA problem for processor faults is only $g > \lfloor (g-1)/3 \rfloor + 2F_{\mathrm{Gm}} + F_{\mathrm{Gd}}$. The unit of Wang et al[13]. is group-oriented network, and the topology of virtual subnet within cloud storage environment is composed of several groups, so that a processor in the topology of virtual subnet within cloud storage environment is similar to a group-oriented network[14]. Consequently, $g > \lfloor (g-1)/3 \rfloor + 2F_{\mathrm{Gm}} + F_{\mathrm{Gd}}$ in Wang et al is same to the topology of virtual subnet within cloud storage environment. □

As a result, VSACS requires a minimal number of message exchange rounds and tolerates a maximum number of faulty components to reach a common agreement with correct processors. The optimality of the protocol is proven.

CONCLUSIONS

Due to the mobility of the mobile network, these processors may immigrate into or move away from the network at any time. Furthermore, some of the processors in the network may be fallible, so the network would not be stable. The network topology developed in recent years[3,19] shows a mobile feature. The previous protocols[6,16,17] cannot adapt to solve BA problem in the topology of virtual subnet within cloud storage environment, and none of the BA protocol is designed for the topology of virtual subnet within cloud storage environment. In this paper, the proposed VSACS can ensure that all the correct processors of the topology of virtual subnet within cloud storage environment can reach a common value to cope with the influences of the dual faulty processors by using a minimum number of message exchanges and tolerate a maximum number of faulty processors at any time.

Acknowledgements: This study is conducted under the Cloud computing systems and software development projects of the Institute for Information Industry which is subsidized by the Ministry of Economic Affairs of the Republic of China.

REFERENCES

1. Julia M, Thinn TN (2011) Management of data replication for PC cluster based cloud storage system. *Int J Cloud Comput: Serv Arch* **1**, 31–41.
2. Tsai WT, Zhong P, Elston EJ, Bai X, Chen Y (2011) Service replication with MapReduce in clouds. In: *IEEE International Symposium on Autonomous Decentralized Systems*, pp 381–8.
3. Chiang TC, Tsai HM, Huang YM (2005) A partition network model for ad hoc networks. In: *IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*, vol 3, pp 467–72.
4. Wang SC, Yan KQ (1998) Revisit consensus problem on dual link failure modes. In: *International Computer Software & Applications Conference*, pp 84–9.
5. Wang SC, Yan KQ, Cheng CF (2002) Achieving high efficient Byzantine agreement with dual components failure mode on a multicasting network. In: *The 9th International Conference on Parallel and Distributed Systems*, pp 577–82.
6. Lamport L, Shostak R, Pease M (1982) The Byzantine general problem. *ACM Trans Program Lang Syst* **4**, 382–401.
7. Min M, Wang F, Du DZ, Pardalos PM (2004) A reliable virtual backbone scheme in mobile ad-hoc networks. In: *IEEE International Conference on Mobile Ad-hoc and Sensor Systems*, pp 60–9.
8. Paul K, Bandyopadhyay S, Mukherjee A, Saha D (2001) A stability-based distributed routing mechanism to support unicast and multicast routing in ad hoc wireless network. *Comput Comm* **8**, 128–35.
9. Wang SC, Yang WP, Cheng CF (2004) Byzantine agreement on mobile ad-hoc network. In: *IEEE International Conference on Networking, Sensing and Control*, pp 52–7.
10. Perkins PE, Royer EM (1999) Ad-hoc on-demand distance vector routing. In: *IEEE Workshop on Mobile Computing Systems and Applications*, Vol 3, pp 90–100.
11. Molina G (1986) Application of Byzantine agreement in database systems. *ACM Trans Database Syst* **11**, 27–47.
12. Wang SC, Yan KQ, Zheng GY (2006) Reaching consensus underlying fallible virtual subnet of mobile ad-hoc network. In: *12th Mobile Computing Workshop*, pp 257–63.
13. Wang SC, Yan KQ (2005) Group Byzantine agreement. *Comput Stand Interfac* **28**, 75–92.
14. Wang SC, Yan KQ, Cheng CF (2003) Asynchronous consensus protocol for the unreliable unfully connected network. *ACM Operating Syst Rev* **37**, 43–54.
15. Gifford DK (1979) Weighted voting for replicated data.

Tech Rep CSL-79-14, XEROX Palo Alto Research Center.
16. Siu HS, Chin YH, Yang WP (1996) A note on consensus on dual failure modes. *IEEE Trans Parallel Distr Syst* **7**, 225–30.
17. Fischer M, Lynch N (1982) A lower bound for the assure interactive consistency. *Inform Process Lett* **14**, 183–6.
18. Halsall F (1995) *Data Links, Computer Networks and Open Systems*, 4th edn, pp 112–25.
19. Meyer FJ, Pradhan DK (1991) Consensus with Dual Failure Modes. *IEEE Trans Parallel Distr Syst* **2**, 214–22.