The Construction and Analysis of Agent Fault-Tolerance Model Based on π-Calculus

Yichuan Jiang[1], Zhengyou Xia[2], Yiping Zhong[1], and Shiyong Zhang[1]

[1] Department of Computing & Information Technology, Fudan University
Shanghai 200433, P.R.China
http://www.cit.fudan.edu.cn
[2] Department of computer, NanJing University of Aeronautics and Astronautics
Nanjing 210043, P.R.China
http://ice.nuaa.edu.cn/academic/4.php
{jiangyichuan,zhengyou_xia}@yahoo.com.cn,
{ypzhong,szhang}@fudan.edu.cn

Abstract. Agent replication and majority voting is a typical method to realize agent fault-tolerance. However, with such method, many agent replicas are produced in agent execution, which may cost much network and time resource. The paper constructs a novel agent migration fault-tolerance model based on integrity verification (AMFIV), which can reduce the complexity degree of agent communication and agent replicas amount so that network and time resource can be much saved. At last, the paper makes analysis for AMFIV based on π-calculus. The π-calculus analysis result proves that the novel model provided by the paper is correct and valid.

1 Introduction

Mobile agent technology can support agent migration among hosts and make network application more flexible and effective. However, mobile agent system may also bring out such problem: when there are malicious hosts, how to protect mobile agents against them? This is the **Problem of Malicious Host** [1].

To solve the Problem of Malicious Host, there have been some works, such as Time Limited Blackbox [1], Reference States [2], Cryptographic Traces [3], Authentication and State Appraisal [4], and some other solutions which adopted the measures of cryptography, digital signature and trusted environment [5], etc.

The above researches have made very effect for solving the Problem of Malicious Host. However, they often focus on the prevention and detection of the problem, and not cope with how to keep the mobile agent system uninterruptedly operating well when the Problem of Malicious takes place. Aiming at such situation, the concept of **agent fault-tolerance** was presented. Among the relative researches, [6] is the representative one in which the measure of agent replication and majority voting was adopted.

Now we introduce the typical relative work on agent fault-tolerance-**replicated agent migration computation with majority voting (RAMMV)** in [6], as shown in Fig.1.

M. Bubak et al. (Eds.): ICCS 2004, LNCS 3038, pp. 591–598, 2004.

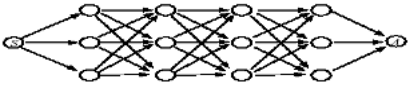

Fig. 1. Replicated agent migration computation with majority voting

In RAMMV, a node p in stage i takes as its input the majority of the inputs it receives from the nodes comprising stage i-1. And, p sends its output to all of the nodes that it determines comprising stage i+1 [6]. The voting at each stage makes it possible for the computation to heal by limiting the impact of the faulty host in one stage on hosts in subsequent stages. More precisely, it is possible to tolerate faulty values from a minority of the replicas in each stage.

However, such model is not feasible in practice, mainly as the model requests that all agent replicas should keep alive until the end of agent migration and assures that replicated hosts fail independently [5], and the large numbers of agent replicas may cost much network and host resource. Otherwise, the result voting among the replicas of agent can also cost much resource and time.

To resolve the deficiency of the method in [6] and other relative works, we suggest a novel agent migration fault-tolerance model based on integrity verification (AMFIV). The new model suggested by us can reduce the cost of resource and time very much.

The rest of the paper is organized as follows. Section 2 presented the novel agent migration fault-tolerance model-AMFIV. Section 3 makes analysis for the model based on π-calculus. Then the paper concludes in Section 4.

2 A Novel Agent Migration Fault-Tolerance Model (AMFIV)

2.1 Illustration of AMFIV

To solve the Problem of Malicious Host, we presented a novel agent migration fault-tolerance model based on integrity verification called AMFIV. We can see the trace example of AMFIV in Fig 2.

Fig 2 can be explained as following: agent at stage i runs on $host_i$ and selects a node with the highest priority as the next host to migrate which can be denoted as $host_{i+1}$ (0); then the agent spawns a replica which migrates to $host_{i+1}$ (0); agent replica runs on $host_{i+1}$ (0), after running its integrity is verified by $host_i$; if the integrity verification result is ok, then the agent on $host_{i+1}(0)$ spawns a replica to migrate to $host_{i+2}(0)$, and the agent on $host_i$ is terminated; otherwise $host_{i+1}(0)$ is a malicious one, then the agent on $host_i$ re-selects another host with the second priority as the next one to migrate which can be denoted as $host_{i+1}(1)$, and the model will execute the operations as the same as above operations. If $host_{i+1}(1)$ is also malicious, then the agent on $host_i$ will re-select another host $host_{i+1}(2)$ with the third priority as the next one to migrate…until there exists a normal host to migrate or there don't exist any other adjacent nodes to select. If $host_i$ hasn't any other adjacent nodes, then the agent on $host_i$ returns to $host_{i-1}$, and selects another node as $host_i$ (1).

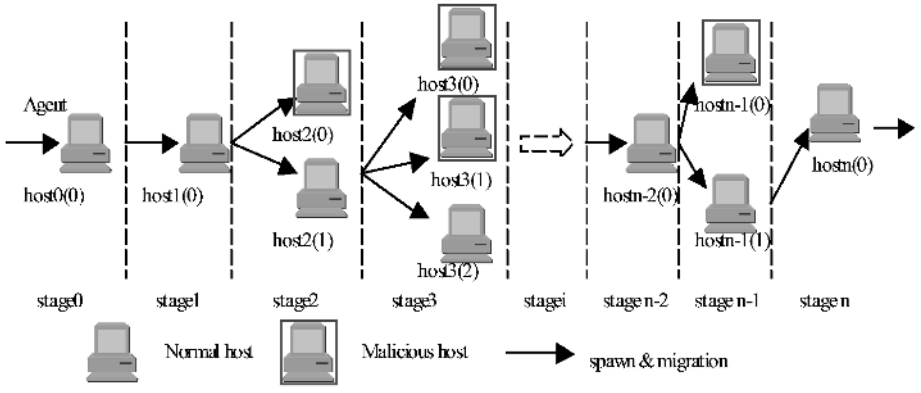

Fig. 2. The agent migration trace example of AMFIV

From Fig 2, we can see that agent needn't to produce replica at every migration step. In the AMFIV model, firstly agent migrates according to linear trace, only when the agent integrity is damaged by a malicious host then a new path is re-selected. But RAMMV model requires that at every migration step the replicas should be produced. Otherwise, AMFIV model limits the fault-tolerance problem to be solved in single hop, which avoid the multi steps accumulative problem.

Let the number of agent migration steps is n, and the number of standby nodes in every step is m, obviously the complexity of agent migration communication degrees in RAMMV is $O(n*m^2)$, and the one in AMFIV is $O(n*m)$. So AMFIV reduces the complexity degrees of agent migration communication from cube level to square level. Therefore, in AMFIV the network load can be reduced much accordingly.

On the amount of replicas produced, the average complexity in RAMMV is $O(n*m)$, but in AMFIV only under the worst situation, i.e. in every step the first m-1 nodes are all malicious, the complexity can reach $O(n*m)$. Obviously, the worst situation seldom takes place in practice, so AMFIV can also reduce the amount of agent replicas.

2.2 The Verification of Agent Integrity in AMFIV

The agent integrity includes the integrity of agent code, data and state. Here we discuss how to make agent code and data integrity verification in AMFIV model. In our verification protocol, we suppose the hosts have a shared key.

▸ The Sub-module of Agent Code Integrity Verification

After the agent runs on $host_{i+1}$, we make code integrity verification to detect that whether the agent code is damaged by $host_{i+1}$.

The agent code integrity verification protocol is explained as follows: ($K_{i,i+1}(x)$ denotes that encrypting x with the key shared by $host_i$ and $host_{i+1}$).

- A). $host_i \rightarrow host_{i+1} : i, R_i, K_{i,i+1}(t_i);$
- B). $host_{i+1} \rightarrow host_i: R_{i+1}, K_{i,i+1}(R_i , t_{i+1});$ /*R_i denotes the request message sent by $host_i$, and R_{i+1} denotes the request message sent by $host_{i+1}$ */

– C). $host_i \rightarrow host_{i+1}$: $K_{i,i+1}(R_{i+1})$; /*A), B), C) denote the identification authentication between $host_i$ and $host_{i+1}$ */
– D). $host_{i+1} \rightarrow host_i$: $K_{i,i+1}(hash(Code_{i+1}||t_{i+1}))$; /* $host_{i+1}$ sends the hash value of the agent code on $host_{i+1}$ with time stamp to $host_i$ */
– E). $host_i$: *Check:*
compute $hash(Code_i||t_{i+1})$;
if $hash(Code_i||t_{i+1}) == hash(Code_{i+1}||t_{i+1})$
then Agent code integrity is ok;
else Agent code integrity isn't ok.
/* $host_i$ computes the hash value of the agent code on itself, then compares it with the hash value returned by $host_{i+1}$ and judge if the agent code integrity is ok.*/

Analysis for the protocol: since the agent code shouldn't be changed in migration, so if some malicious hosts change the agent code, then the hash value of code should be different and can be detected. Since $host_{i+1}$ don't know the hash(Code$\|t_{i+1}$) computed by $host_i$, so it can't forge hash(Code$\|t_{i+1}$). If $host_{i+1}$ makes any change to the agent code, the hash value returned is different from the one computed by $host_i$, so the change can be detected. Therefore, the protocol is secure and correct.

The Sub-module of Agent Data Integrity Verification

On the base of the work of [7], we design the sub-module of agent data integrity verification in AMFIV.

Thereinafter D_i signifies the data collected by the agent on $host_i$, and AD_i signifies the list of data collected by the agent from $host_0$ to $host_i$ accumulatively.

– Stage 0: $Host_0$ generates a secret number C_0, then computes C_1=hash(C_0), and passes C_1 to the agent, now AD_0={};
– Agent encrypts C_1, then migrates to $host_1$;
– On $host_1$: C_1 can be obtained by decryption, agent collects data D_1, AD_1= $AD_0 \cup D_1$, computes the data proof $PROOF_1$=hash(D_1,C_1), C_2=hash(C_1);
– On $host_i$ ($1 \le i \le n-1$): C_i can be obtained by decryption, Agent collects data D_i, AD_i= $AD_{i-1} \cup D_i$, computes the data proof $PROOF_i$=hash(D_i, C_i, $PROOF_{i-1}$), C_{i+1}=hash(C_i); then Agent encrypts C_{i+1} and passes it with AD_i, $PROOF_i$ together to $host_{i+1}$.
– The protocol that $host_i$ verifies the agent data integrity after running on $host_{i+1}$ is shown as follows.
– A). $host_i \rightarrow host_{i+1}$: i, R_i, $K_{i,i+1}(t_i)$;
– B). $host_{i+1} \rightarrow host_i$: R_{i+1}, $K_{i,i+1}(R_i, t_{i+1})$;
– C). $host_i \rightarrow host_{i+1}$: $K_{i,i+1}(R_{i+1})$; /*Similar to the protocol of agent code integrity verification, the A), B), C) are used for identification authentication between $host_i$ and $host_{i+1}$ */
– D). $host_{i+1} \rightarrow host_i$: C_{i+1}, AD_{i+1}, $PROOF_{i+1}$, ; /* $host_{i+1}$ passes the agent data information to $host_i$ */
– E). $host_i$: *Computes* $proof_{i+1}=hash(AD_{i+1}\text{-}AD_i, hash(C_i), PROOF_i)$;
/* Computes $proof_{i+1}$ on $host_i$ */
– F). $host_i$: *if* $(proof_{i+1} == PROOF_{i+1})$ *and* $(C_{i+1} == hash(C_i))$
then agent data integrity is ok;
else agent data integrity isn't ok.

Analysis for the protocol: since C_{i+1}= hash(C_i), so $host_{i+1}$ can't obtain C_i from C_{i+1}, and $host_{i+1}$ can't obtain C_j(j<i+1); $host_{i+1}$ doesn't know C_j(j<i+1), and can't modify D_j(j<i+1), so it can't forge PROOF. Therefore the protocol is secure. Obviously, if the original data of agent is damaged by $host_{i+1}$, then $proof_{i+1}$ isn't equal to $PROOF_{i+1}$, so the damage of data integrity can be detected, therefore the protocol is correct. Obviously, we can see that the protocol can only detect any tampering of the data collected before $host_{i+1}$, and can't detect whether $host_{i+1}$ collects dirty data. Therefore, the protocol only guarantees the integrity of validly collected data.

3 Make Analysis Based on π-Calculus

Now we will make analysis to AMFIV based on π-calculus.

In our π-calculus model, the channels used are seen in Table 1.

Table 1. List of Channel Name in the π-Calculus Model

Channel	Sender	Receiver	Message
next	$host_i$	$host_i$	cha: denotes the next host to migrate
cha	$host_i$	$host_{i+1}$	Agent replica (includes data, code, etc.)
chr	$host_{i+1}$	$host_i$	The agent running result on $host_{i+1}$
chv_1	$host_i$	$host_i$	The verification result of agent integrity
chv_2	$host_i$	$host_{i+1}$	The verification result of agent integrity

We can define the π-calculus model of $host_i$ as Formula (1)[1]:

$$
\begin{aligned}
host_i &\overset{def}{=} next(cha).\overline{cha}(RUN(agenti_i)) \mid chr(x).(let\ (code_{i+1}, data_{i+1}) \\
&= x\ in\ (\overline{chv_1}(VALIDATE(code_{i+1}, data_{i+1}). \\
&\overline{chv_2}(VALIDATE(code_{i+1}, data_{i+1})))) \\
&\mid chv_1(y).([y = true].TERMINATE(agent_i) + \\
&[y = false].(\overline{next}(SELECT(i+1)).host_i))
\end{aligned}
\tag{1}
$$

The Formula (1) is explained as following: from channel *next* $host_i$ obtains the $host_{i+1}$ which is denoted as channel *cha*, then spawns a replica of $agent_i$ after running on $host_i$, and migrates the replica through channel *cha* to $host_{i+1}$; From channel *chr* $host_i$ obtains the running result of $agent_{i+1}$ on $host_{i+1}$, and makes verification (VALIDATE) for its code and data integrity, then passes the verification result to channels chv_1, chv_2; From channel chv_1 $host_i$ obtains the verification result, if it is true the agent on $host_i$ is terminated, or else $host_i$ should re-select a new node to migrate, and passes the new node to channel *next*, then repeats all the acts of the model.

[1] A pair splitting process *let (x, y)=M in P* behaves as *P[N/x][L/y]* if term M is the pair (N,L), and otherwise it is stuck.

We can define the π-Calculus model of $host_{i+1}$ as Formula (2):

$$host_{i+1} \overset{def}{=} cha(agent_{i+1}).\overline{chr}(RUN(agent_{i+1})) \mid chv_2(z).([z = true].GETMASTER + [z = false].ISOLATED) \tag{2}$$

The Formula (2) is explained as following: from channel cha_1 $host_{i+1}$ receives the agent replica and runs it, and then passes the result (data part and code part) back to $host_i$; From channel chv_2 $host_{i+1}$ receives the verification result, if the verification result is true then $host_{i+1}$ gets the control power and agent can migrate further, or else $host_{i+1}$ is a malicious one and it should be isolated.

Therefore, we can define the π-Calculus model of AMFIV as Formula (3):

$$\begin{aligned} AMFIV &\overset{def}{=} (\nu next, cha, chr, chv_1, chv_2)(host_i \mid host_{i+1}) \\ &\equiv (\nu next, cha, chr, chv_1, chv_2)(next(cha).\overline{cha}(RUN(agent_i)) \mid \\ &chr(x).(let\ (code_{i+1}, data_{i+1}) = x\ in\ (\overline{chv_1} \\ &(VALIDATE(code_{i+1}, data_{i+1})).\overline{chv_2}(VALIDATE(code_{i+1}, data_{i+1})))) \\ &\mid chv_1(y).([y = true].TERMINATE(agent_i) + [y = false]. \\ &\overline{next}(SELECT(i+1).host_i)) \mid \\ &cha(agent_{i+1}).\overline{chr}(RUN(agent_{i+1})) \mid chv_2(z).([z = true]. \\ &GETMASTER + [z = false].ISOLATED)) \end{aligned} \tag{3}$$

Now we can use the π -calculus simulate the execution of AMFIV.

$$\begin{aligned} &AMFIV \xrightarrow{next(cha)} (\nu next, cha, chr, chv_1, chv_2)(\overline{cha}(RUN(agent_i)) \mid \\ &chr(x).(let\ (code_{i+1}, data_{i+1}) = x\ in\ (\overline{chv_1} \\ &(VALIDATE(code_{i+1}, data_{i+1})).\overline{chv_2}(VALIDATE(code_{i+1}, data_{i+1})))) \\ &\mid chv_1(y).([y = true].TERMINATE(agent_i) + [y = false]. \\ &(\overline{next}(SELECT(i+1)).host_i)) \mid \\ &cha(agent_{i+1}).\overline{chr}(RUN(agent_{i+1})) \mid \\ &chv_2(z).([z = true].GETMASTER + [z = false].ISOLATED)) \end{aligned} \tag{4}$$

$$\xrightarrow{\tau}(\nu next, chr, chv_1, chv_2(chr(x).(let\ (code_{i+1}, data_{i+1}) = x\ in$$
$$(\overline{chv_1}(VALIDATE(code_{i+1}, data_{i+1})).\overline{chv_2}$$
$$(VALIDATE(code_{i+1}, data_{i+1})))) \mid chv_1(y).([y = true]$$
$$.TERMINATE(agent_i) + [y = false].(\overline{next}(SELECT(i+1)).host_i)) \mid$$
$$\overline{chr}(RUN(agent_{i+1})) \mid chv_2(z).([z = true]$$
$$.GETMASTER + [z = false].ISOLATED))$$

$$\xrightarrow{\tau}(\nu next, chv_1, chv_2)(let\ (code_{i+1}, data_{i+1}) = RUN(agent_{i+1})$$
$$in\ (\overline{chv_1}(VALIDATE(code_{i+1}, data_{i+1})).\overline{chv_2}(VALIDATE$$
$$(code_{i+1}, data_{i+1})))) \mid chv_1(y).([y = true].TERMINATE(agent_i)$$
$$+[y = false].(\overline{next}(SELECT(i+1).host_i)) \mid$$
$$chv_2(z).([z = true].GETMASTER + [z = false].ISOLATED))$$

$$\xrightarrow{\tau}(\nu next)(([VALIDATE(code_{i+1}, data_{i+1}) = true].$$
$$TERMINATE(agent_i) +$$
$$(VALIDATE(code_{i+1}, data_{i+1}) = false].(\overline{next}(SELECT(i+1).host_i))$$
$$\mid ([VALIDATE(code_{i+1}, data_{i+1}) = true].GETMASTER +$$
$$[VALIDATE(code_{i+1}, data_{i+1}) = false].ISOLATED))$$
$$\begin{cases} \xrightarrow{[VALIDATE(code_{i+1}, data_{i+1}) = true]} TERMINATE(agent_i) \mid GETMASTER \\ \xrightarrow{[VALIDATE(code_{i+1}, data_{i+1}) = false]} (\nu next_1)(\overline{next}(SELECT(i+1)).host_i \mid \\ \qquad\qquad ISLOATED) \end{cases}$$

From above we can see that: if the integrity of agent code and data is ok, the ultimate result is $TERMINATE(agent_i) \mid GETMASTER$, so the agent on $host_i$ is terminated, $host_{i+1}$ gets the control power, and agent migrates according to a linear trace; if the integrity of agent code and data is damaged by $host_{i+1}$, the ultimate result is $(\nu next_1)(\overline{next_1}(SELECT(i+1)).host_i \mid ISOLATED)$, so $host_i$ re-selects another node as the next one to migrate, and repeats the acts of the model, and $host_{i+1}$ is isolated.

Therefore, from the above simulation result of the π-Calculus model of AMFIV, we can see that AMFIV is correct.

4 Conclusion

In this paper, aiming at the deficiency of other typical agent fault-tolerance models, we suggested a novel agent migration fault-tolerance model based on integrity verification called AMFIV. Comparing to other agent fault-tolerance models, our model can reduce the complexity degree of agent communication and agent replicas amount. The π-calculus simulation validation results prove that AMFIV is correct and efficient.

References

1. Fritz Hohl: Time Limited Blackbox Security: Protecting Mobile Agents from Malicious Hosts. Mobile Agents and Security, Giovanni Vigna (ed.), Springer-Verlag, 1998, pp. 92–113
2. Fritz Hohl: A Protocol to Detect Malicious Hosts Attacks by Using Reference States. http://elib.uni-stuttgart.de/opus/volltexte/2000/583/
3. Giovanni Vigna: Cryptographic Traces for Mobile Agents. Mobile Agents and Security, Giovanni Vigna (ed.), Springer-Verlag, 1998, pp. 137–153
4. William M. Farmer, Joshua D. Guttma, and Vipin Swarup: Security for Mobile Agents: Authentication and State Appraisal. In: Proceedings of the Fourth European Symposium on Research in Computer Security, 1996
5. Bennet S. Yee: A Sanctuary for Mobile Agents. In: DARPA Workshop on Foundations for Secure Mobile Cde, 1997. http://www.cs.ucsd.edu/~bsy/pub/sanctuary.ps
6. Fred B. Schneider: Towards Fault-tolerant and Secure Agentry. Invited paper. In: Proceedings of 11th International Workshop on Distributed Algorithms, Saarbrücken, Germany, Sept 1997
7. Paolo Maggi and Riccardo Sisto: Experiments on Formal Verification of Mobile Agent Data Integrity Properties. www.labic.disco.unimib.it/woa2002/papers/15.pdf
8. R. Milner: The Polyadic π-Calculus: a Tutorial. In. F.L. Bauer, W. Braueer, and H. Schwichtenberg (eds.), Logic and Algebra for Sepcification. Berlin: Springer-Verlag, 1993, pp. 203–246.