Agents for Autonomous Distributed Secret Sharing Storage Systrem

Daisuke Hayashi, Toshiyuki Miyamoto, Shinji Doi, and Sadatoshi Kumagai
Department of Electrical Engineering, Faculty of Engineering
Osaka University, Japan
Phone: +81-6-6879-7694, Fax: +81-6-6879-7263

Abstract: For mission-critical and safe-critical operations of medical information, financial, or administrative systems, a reliable and robust storage system is indispensable. The main purpose of our research is to develop a high-confidential, reliable, and survivable storage system.

1. Introduction

For mission-critical and safe-critical operations of medical information, financial, or administrative systems, a reliable and robust storage system is indispensable. A conventional storage system, such as a RAID system, can guarantee its survivability only against for disk crashes of several disk drives. For security improvement, a file system, TCFS (Transparent Cryptographic File System)[1], has been proposed, in which the security is guaranteed by means of the DES (data encryption standard) algorithm. But this may pose a risk of being stolen data because a encrypted file can be decrypted if the encryption key is known. Moreover, TCFS does not distribute an encrypted data over servers. It isn't suitable for safe-critical systems, because it does not guarantee its survivability against for failures such as disk crashes.

A simple way to increase the survivability is storing data on several computers connected with a network. But in this way the confidentiality would be decrease. An encoding technology, called (k, n) threshold scheme [2], would be useful to increase both of the survivability and the confidentiality. The (k, n) threshold scheme generates n cryptographs, each of them is called a share, from an original data, and restores the original data from k shares. We are developing an autonomous distributed storage system by using the (k, n) threshold scheme[3].[1] In the system, n shares encoded from an original data are stored on distributed storage nodes (computers) connected with ultra high-speed network. If the value k and n are chosen appropriately, the storage system is sufficiently confidential, since no one can restore the original data from $k-1$ or less shares. And it is sufficiently survivable, since even if $n-k$ shares have been lost, we can restore the original data from remaining k shares.

We are developing a multi agent system for realizing the autonomous distributed storage system. Each agent offers basic storage services, such as storing, retrieving, updating, and so on, on a distributed storage node, and is an interface of the storage system to users. The (k, n) threshold scheme is survivable and confidential by itself. But if the multi agent system is brittle, the autonomous distributed storage system could not be a survivable and confidential storage system. In this paper, we discuss a Petri net model[4] of the storage system, which will be useful to analyse the behavior of the system.

[1]This research project has been carrying out with Kochi University of Technology and Iseikai Hospital under a support of Telecommunications Advancement Organization of Japan.

2. Threshold Scheme

2.1 Algorithm of Threshold Scheme

Let S be the secret information, p be a prime number greater than S, and $GF(p)$ be a Galois Field of p. Users must determine numbers k and n $(0 \leq k \leq n)$, then we can calculate n shares, $w_1, \cdots, w_n$, from S by following expression.

$$\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} \equiv \begin{bmatrix} 1 & \alpha_1 & \alpha_1^2 & \cdots & \alpha_1^{k-1} \\ 1 & \alpha_2 & \alpha_2^2 & \cdots & \alpha_2^{k-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_n & \alpha_n^2 & \cdots & \alpha_n^{k-1} \end{bmatrix} \begin{bmatrix} S \\ r_1 \\ \vdots \\ r_{k-1} \end{bmatrix} \pmod{p} \quad (1)$$

where $r_j (j = 1, 2, \cdots, k-1)$ are random numbers on $GF(p)$, and α_i, that is called an ID of share w_i, has following relations.

$$\alpha_1 = \alpha,\ \alpha_2 = \alpha^2, \cdots, \alpha_n = \alpha^n \quad (2)$$

When decrypting the original data, we must collect k shares, $w_{j_1}, w_{j_2}, \cdots, w_{j_k}$. The original data can be decrypted by following expression.

$$\begin{bmatrix} S \\ r_{j_1} \\ \vdots \\ r_{j_{k-1}} \end{bmatrix} \equiv \begin{bmatrix} 1 & \alpha^{j_1} & \cdots & \alpha^{(k-1)j_1} \\ 1 & \alpha^{j_2} & \cdots & \alpha^{(k-1)j_2} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{j_k} & \cdots & \alpha^{(k-1)j_k} \end{bmatrix}^{-1} \begin{bmatrix} w_{j_1} \\ w_{j_2} \\ \vdots \\ w_{j_k} \end{bmatrix} \pmod{p} \quad (3)$$

where α^{j_k} is the ID of share w_{j_k}.

3. Storage System Using Threshold Scheme

Fig.1 shows the image of the autonomous distributed storage system. This storage system is a multi agent system that consists of client agents and server agents. These exist on storage nodes bestrewing over network. We aim to realize the high-confidential, high-reliable and high-survivable system by coordinations among agents.

Any client agent can connect with any server agent, and can ask a request for the server agent.

A server agent, that received a store request from a client agent, encrypts a data, and sends shares to the

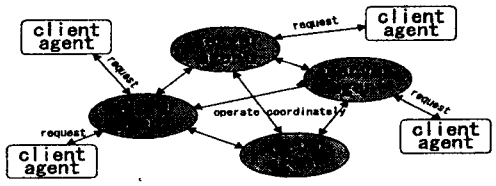

Figure 1. Storage System

appropriate server agents according to a storing policy. The storing policy is examined in the next chapter. The server agent, that received the share, stores it into its local disk, and prepares for a next request.

A server agent, that received the restore request from a client agent, collects more than k shares from other servers, and decrypts the original data.

Agents always operate based on their autonomous judgments. For example, in case where a client agent loses communication link with a server agent because of some reasons such as a breakdown of the network or a crash of the server, the client chooses another communicable server. In this sense, we can construct very robust storage system.

Moreover it is possible by use of agent technologies that agent can regenerate lost share automatically or can find altered share and repair it. In this sense, we can construct very reliable storage system.

On the other hand, agent technologies may be also useful for load-balancing.

In the system that stores data as the need arises, such as medical information system and document database, its scalability is very important factor. In our system, we can make a server agent "plug-inable". Since it enables us to change the scale, easily our system is also useful for these kinds of applications.

4. Evaluation of Storing Policy

We may say that the confidentiality and the reliability are affected by how to store n shares generated by threshold scheme.

Consequentially, we consider some storing policies and we experiment them on survival rate and disk's usage rate.

4.1 Storing Policy

We considered the following four storing policies.

(1) Choose n servers at random

In this policy, n shares are put into n servers at random. Let x be size of a share file, and n be the number of share files. Then the traffic to store this file over the netwowrk is nx under an assumption that all servers are completely connected and the length between any two nodes is 1. We call this polity by "Random".

(2) Put a share into requested server and choose $n-1$ servers at random

A share is put into the server that receive store request, and other shares are selected at random. In this policy, the traffic is described as $(n-1)x$. We call this policy by "Self-Random".

(3) Choose servers by weighting probability

Probability of choosing the server is weighted by depending on its free space. Since this policy can balance each server's free space, it has a beneficial effect in case where server's capacity isn't even. But this policy needs to check capacity of each server, so its traffic costs more. Denoting the traffic of communication for checking capacity by y, the traffic is $nx + 2y$. We call this policy by "Capacity-Weighted".

(4) Choose same n servers

This policy always chooses same n servers. We can store efficiently in a sense since we can choose servers depending on network topology. We call this policy by "Fixed".

4.2 Experiment

We store 1000 data, and its size is randomly selected between 1 and 100 (size of share is same as original data). We experiment about case of losing shares by crashing hard disk of server, and calculate survival rate for four policies. Let $n = 5$, $k = 3$, $s = 10$. Let x be the number of servers that breaks by some reason such as a disk crash. We compare survival rate of worst case by generating $0.1 \times {}_{10}C_x$ test patterns at random. The worst survival rate is defined by the following expression.

$$\min_{p \in P} \frac{N_p}{N} \times 100 \quad (\%) \tag{4}$$

where N is the number of data, N_p is the number of data that there exist more than k shares under a pattern p, and P is the set of patterns. In our storage system, there may exit multiple clients, but we paid attention to one client, and we assumed that a client always sends request to the same server.

CASE1) Capacities of servers are even

We experiment in case where the capacity of each servers is 100000. The results are presented in Fig.2. In this

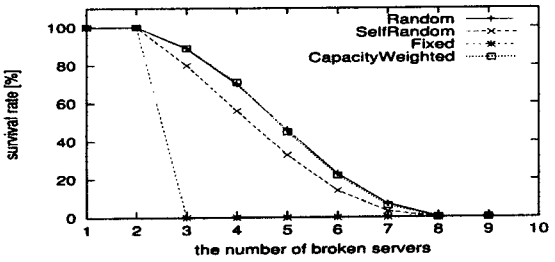

Figure 2. Worst Survival Rate (Capacity 100000)

case, the results of Random and Capacity-Weighted almost equal. But the result of Self-Random and Fixed is worse in survival rate.

CASE2) Capacities of servers are not even

We experimented on two cases where a range of server's capacity differs: (30000, 80000), and (50000, 100000). The number of data and its size are equal to CASE1.

Therefore approximately it requires 50000. In the first case, some servers may overflow. When a server uses up the disk space, another server is selected from remaining servers by the same policy.

each case, and Table.1 shows usage rates of servers' disk for the first case. Let UR_i be $i(i = 0, 1, \cdots, 9)$th server's usage rate, C_i be the server's capacity and U_i be the server's usage. Then UR_i is calculated by following by expression.

$$UR_i = \frac{U_i}{C_i} \times 100 \ (\%) \tag{5}$$

As Fig.3-4 indicate, the second case is similar to CASE1). But in the first case, survival rate of CapacityWeighted is reduced. And as Table1 indicates, servers' usages rate of CapacityWeighted is balanced, but servers' usage rates of the other policies aren't balanced, and some of the servers overflow.

In terms of survival rate, Random is the best policy and SelfRandom is the second best policy in first case. But when the SelfRandom server overflows, SelfRandom don't perform as SelfRandom policy itself. Therefore its policy can't be used at the first case.

We considered the survival rate, the usage rate, and the load to the network as the comparative object of the storing policy. Since the survival rate is reduced after some of servers overflow, averaging usage rate keeps survival rate high. But the results of experiments show that Random servers keep high survival rate in spite of some overflows. Absolutely if the servers' capacities are reduced further, it is sure that the survival rate of Random is reduced. But the system where the server's capacity is severe shortage wouldn't be expected actually. So far, when using CapacityWeighted, not only the traffic to store one file is bigger than Random, but also the load to the network around the server that have bigger free space increases.

Consequently, we determine to use Random as the storing policy.

Table 1. Usage Rate(Capacity 30000-80000)

	Random	SelfRandom	Fixed	Capacity
srv0	99	99	0	66
srv1	42	34	0	51
srv2	46	49	23	55
srv3	77	56	43	51
srv4	78	45	26	63
srv5	81	44	99	63
srv6	72	53	99	60
srv7	39	48	81	61
srv8	57	47	99	61
srv9	85	75	92	53

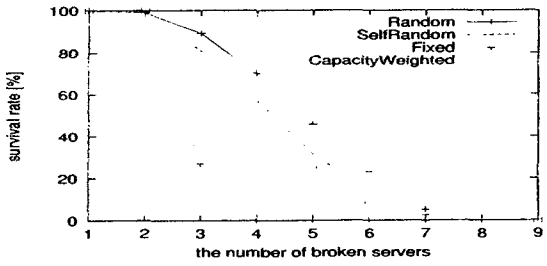

Figure 3. Worst Survival Rate (Capacity 20000-70000)

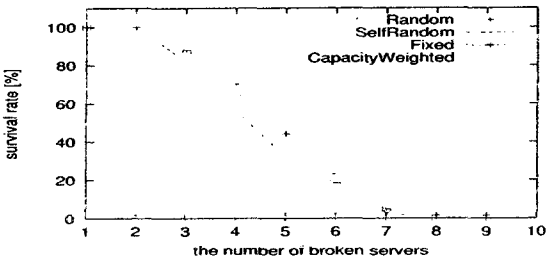

Figure 4. Worst Survival Rate (Capacity 50000-100000)

5. MAN model

5.1 The Model of the System

To describe a behavior of each agent, we have been proposed a description language for multi agent systems, called the Multi Agent Net (MAN) [5], and developed a simulation environment of MAN. With the simulation environment, we can easily construct and simulate the multi agent system.

As Fig.5-9 indicate, we modeled the five basic file-handlings (store, delete, restore, update,and refer) using MAN.

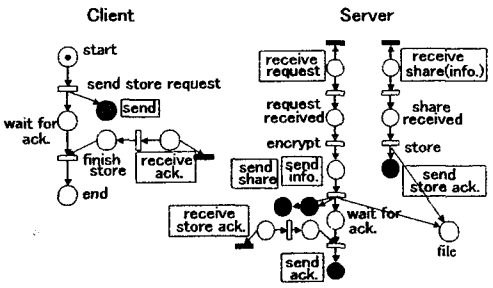

Figure 5. Model for Store Process

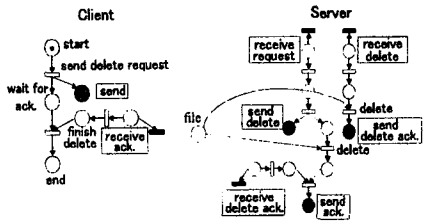

Figure 6. Model for Delete Process

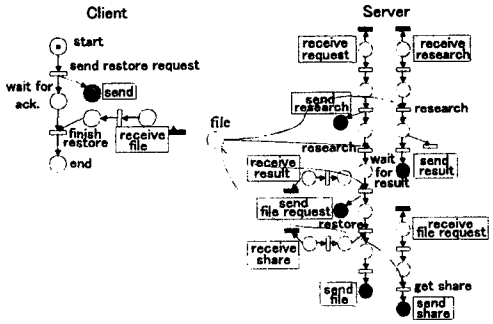

Figure 7. Model for Restore Process

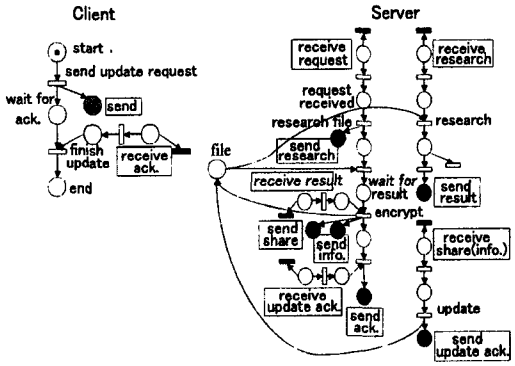

Figure 8. Model for Update Process

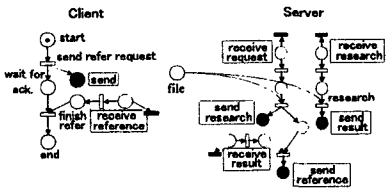

Figure 9. Model for Refer Process

6. Conclusion

In this research, the agent for the design of the autonomous distributed storage system guaranteed confidentiality, reliability and survivability was developed.

We define five file-handlings of this system, and model them with MAN.

Furthermore, we simulate this multi agent system by using the MAN simulation environment to check operation of system.

Because the present agent net model just simulates operations and the encryption of data, in future, we need to include the actual file-handling library offered from the Kochi University of Technology. Moreover, SSL/TLS[6] may improve a security of communication between client agents and server agents. As for a tolerance for alterations, message digest algorithms such as MD5[7] may be useful.

Since each agent behaves asynchronously and concurrently, the whole state space of the system we propose would be immense, and it is not easy to comprehend it. To overcome the problem, we are going to use a reachability analysis method of Petri nets[4] for a verification of our storage system.

References

[1] G. Cattaneo, L. Catuogno, A. Del Sorbo, and P. Persiano. "Design and Implementation of a Transparent Cryptographic File System for Unix", *Proceedings of the Freenix Track: 2001 USENIX Annual Technical Conference*, pp.199-212, Boston, MA, June 2001.

[2] A. Shamir. "How to Share a Secret" *communication of the ACM*, Vol. 22, No. 11, pp.612-613, 1979.

[3] S. Funahashi, M. Fukumoto and Y. Kikuchi. "Implementation of a command for data distribution using SSS", *IPSJ Distributed System and internet Management technology 2001 SIG*, pp.27-32, 2001.

[4] T. Murata. "Petri Nets: Properties, Analysis and Applications" , *Proceedings of the IEEE*, Vol. 77, No. 4 pp.554-580, April, 1989.

[5] T. Miyamoto and S. Kumagai. "A Multi Agent Net Model and the Realization of Software Environment" *Proceedings of Workshop of Petri Nets to intelligent system development with 20th International Conference on Application and Theory of Petri Nets*, pp.83-92, 1999.

[6] The TLS Protocol Version 1.0, IETF Internet-Draft, http://www.ietf.org/internet-draft/draft-ietf-tls-rfc2246-bis-00.txt

[7] R. Rivest, MIT Laboratory for Computer Science and RSA Data Security, Inc. "The MD5 Message-Digest Algorithm", April 1992. http://www.ietf.org/rfc/rfc1321.txt.