社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2008－AL－121（2）
2008 / 12 / 3

局所発見不可能な故障のゲーム理論的分析

木庭　淳†　菊田　健作††

† 兵庫県立大学経済学部応用経済学科
†† 兵庫県立大学経営学部組織経営学科
〒 651-2197 神戸市西区学園西町 8-2-1
E-mail: †kiniwa@econ.u-hyogo.ac.jp, ††kikuta@biz.u-hyogo.ac.jp

あらまし　局所発見不可能な複数個の故障を発生させる悪意のある敵を想定する。そのような故障の可能性はかつて指摘されていたが、その影響と分析に関する詳細な研究は知られていなかった。ここでは互いに補完しあう2つの自己安定的相互排除プロトコルを組み合わせて悪意のある故障を仮定する。我々はマイナートークンを送るか否かの2つの選択的戦略を用いることによりこの故障に対処する。ここでマイナートークンは特権を与えるためのメジャートークンとペアで用いられ、故障の拡散を防ぐ役割を担う。特権をもつプロセス群対悪意をもつ敵の利得行列を構成し、多段階2人ゼロ和ゲームを考える。ここでは2種類のゲームの解釈—Dijkstra 的故障回復動作が起こったときゲームを終了するか否か—を行い、各場合において悪意のある敵の能力分析を混合戦略を用いて行う。我々の方法は悪意のある敵に対してアルゴリズムを強化する一般的な枠組みとして考えることができる。
キーワード　自己安定、相互排除、収束下の安全性、悪意のある故障モデル、ゲーム理論、多段階2人ゼロ和ゲーム

Game Theoretic Analysis of Malicious Faults Which are Undetectable by Local Checks

Jun KINIWA† and Kensaku KIKUTA††

† Department of Applied Economics
†† Department of Strategic Management, University of Hyogo,
8-2-1 Gakuen nishi-machi, Nishi, Kobe, 651-2197 Japan
E-mail: †kiniwa@econ.u-hyogo.ac.jp, ††kikuta@biz.u-hyogo.ac.jp

Abstract　We consider a malicious adversary which generates multiple undetectable faults by local checks. Though the possibility of such faults has ever been suggested, details of its influence and handling are unknown. We assume the malicious faults in a self-stabilizing mutual exclusion protocol, a hybrid of previously proposed ones that complement each other. In the hybrid protocol, we can cope with the faults by using optional strategies, sending a minor token or not, where the minor token plays a role of preventing the contamination from spreading. We construct a payoff matrix between a group of privileged processes and an adversary, and consider a multistage two-person zero sum game. We interpret the game in two ways: whether or not it terminates when Dijkstra-like repair, i.e., moves against malicious faults, occurs. For each case, we evaluate the ability of malicious adversary by using a mixed strategy. Our idea is also considered as a general framework for strengthening an algorithm against malicious faults.
Key words　self-stabilization, mutual exclusion, safety under convergence, malicious fault model, game theory, multistage two-person zero sum game

1. Introduction

Motivation. Studies on self-stabilization have been extended to vast areas in recent years. One of the areas, which we focus on here, is how to keep a system safe under convergence in self-stabilizing mutual exclusion. The requirement of mutual exclusion is to allow at most one privileged process at any time, called legitimate configurations. Starting from an arbitrary configuration, it is difficult to keep it fully legitimate under convergence. For example, it is known that

Dijkstra's 3-state protocol [3] guarantees recovery from arbitrary initial configurations in an n-process ring as illustrated in Fig. 1.

Dijkstra's 3-state protocol

process p_0:
if $(s_0 + 1)$ **mod** $3 = s_1$ **then** $s_0 := (s_0 - 1)$ **mod** 3
process p_{n-1}:
if $s_{n-2} = s_0$ **and** $s_{n-1} \neq (s_{n-2} + 1)$ **mod** 3
then $s_{n-1} := (s_{n-2} + 1)$ **mod** 3
process p_i $(0 < i < n - 1)$:
if $(s_i + 1)$ **mod** $3 = s_{i-1}$ **then** $s_i := s_{i-1}$
if $(s_i + 1)$ **mod** $3 = s_{i+1}$ **then** $s_i := s_{i+1}$

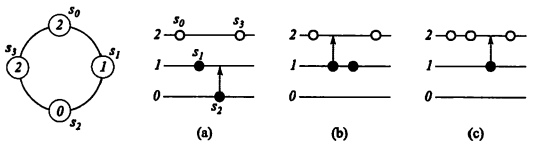

図 1 Dijkstra's 3-State Protocol

In the left hand of the figure, the four processes p_0, p_1, p_2 and p_3 forming a ring, have states $(s_0, s_1, s_2, s_3) = (2, 1, 0, 2)$ and move one by one according to the protocol. An example of state transition is depicted from (a) through (c), where each state is represented by the location of nodes; and privileged processes by black nodes. We call the transition as in Fig. 1 "*Dijkstra-like repair*", where more than one privileges are unexpectedly given to processes under repairing. Hence, the requirement of mutual exclusion is violated temporarily. Such violation of mutual exclusion occurs because each process acquires a privilege by only the difference of states between neighboring processes in a small domain $\{0, 1, 2\}$.

Kiniwa et al. [13] ~ [15] solved this problem by enlarging the state space: Every non-faulty state must take a value in $B_3 = \{0, L, 2L\}$, called *bases*, for a large integer $L \gg n$. Furthermore, a non-faulty process acquires a privilege when it has a state less than a neighboring state by L (**mod** $3L$). Thus, the state other than B_3, which is easily detected by local checks, is a faulty state. If we consider the states $(s_0, s_1, s_2, s_3) = (2, 1, 0, 2)$ in the large state space, only $s_2 \in B_3$ has a non-faulty state and other processes are not given privileges. Since the three values are defined in a large domain, the mutual exclusion is almost safe if we assume that faulty states are uniformly distributed over the range. Such an assumption is called a random fault model. As far as we know, every conventional argument concerning safety under convergence has used this model [10], [13], [21]. We can, however, only know the average safety of a system by the random fault model. In contrast, we can evaluate how little weakness the system has by a malicious fault model. Yen [21] said that

> "In the random failure model, a transient failure can bring the system into any illegitimate state with equal probability",

and that

> "In the malicious failure model, some faulty processors may maliciously try to violate the system legitimacy without being detected by local checks and subsequently cause critical damages".

In fact, the malicious fault may be possible if some intruder intends to violate mutual exclusion or if some biased fault occurs. In the Kiniwa et al.'s enlarged domain system [13], [15], if faulty states are maliciously limited to the three predefined values B_3, they cannot be detected by local checks. Thus, the malicious fault in the enlarged domain system is the attack on the three predefined values B_3, called a *base fault*. We should evaluate the previously proposed methods by the malicious fault model, or improve them assuming the presence of malicious faults.

So we need an argument based on the malicious fault model. To this end, we believe that a game theoretic analysis, two-person zero sum game, is useful. Since many processes obtain a privilege in the long run, they are grouped together, called player A. On the other hand, the malicious adversary is called player B. The advantage of player A, corresponding to the disadvantage of player B, is shared by the processes. In our system, the player A has two strategies, whether or not sending a *minor token*, while the player B has three strategies, not causing a fault, causing a base fault and a *non-base fault*. The minor token [9], [12], introduced as a coupled use with a *major token*, plays a role of checking every process and repairing the base fault which is undetectable by local checks. In contrast, the non-base fault is detectable by local checks. With the help of game theory, we derive optimal mixed strategies of player A and player B.

Related Work. As Dasgupta et al. [2] pointed out, only little work mixing game theory with self-stabilization has been done. We just know a technique, called a scheduler-luck game, for analyzing the performance of randomized self-stabilization [5]. In the context of distributed non-stabilizing algorithms, however, the behavior of selfish agents has been extensively studied [17], triggered by Koutsoupias and Papadimitriou [16]. They used a term "price of anarchy" to represent a ratio of the largest social welfare achievable to the least social welfare achieved at any Nash equilibrium. A similar framework was preserved in the study of selfish stabilization [2]. Our work, however, owes its technical base to the conventional game theory [11], [18].

The mutual exclusion problem has been the main topic in self-stabilization since Dijkstra's work [3], [4]. Several new ideas originated from the problem. One of them is the safety under convergence, e.g., superstabilization [9], [12], cryptography [21], fault containment [6] $\sim$ [8] and enlargement of state space [13] $\sim$ [15], [21]. However, as stated above, the safety of each protocol was measured by only the random fault model.

Notice that the term "malicious failure" is sometimes used in the sense of Byzantine failure [19] or intrusion detection [20]. However, our malicious fault is different from Byzantine failure because our adversary does not generate a Byzantine node whose behavior is arbitrary, but attacks the states in such a way that no local checks detect the fault. So we call our model a "malicious fault which is undetectable by local checks" to distinguish our topic from others.

Contributions. Our goal is not to propose a new protocol, but to develop a game theoretic analysis of the malicious fault model. Our contribution includes

(1) the construction of a malicious fault model: We do not assume that the faulty state takes any value over the domain with equal probability. The malicious fault may give any value to some states. The study on such a malicious fault model is new.

(2) a game theoretic analysis: Grouping a set of privileged processes enables us to consider a multistage two-person zero sum game. The formulation of this game in the self-stabilization is new, and

(3) the suggestion of a framework: We propose a framework for strengthening an algorithm against a malicious fault which is undetectable by local checks. The method of strengthening an algorithm is new.

The rest of this paper is organized as follows. Section 2. states our model, and then presents an example of our protocol. Section 3. provides an analysis of the malicious fault model. First, Section 3. 1 considers that a game continues after Dijsktra-like repair. Second, Section 3. 2 considers that the minor tokens can be sent only once. Third, Section 3. 3 considers that the game terminates when Dijkstra-like repair occurs. Finally, Section 4. concludes the paper.

2. Our Method

2. 1 Self-stabilizing Model

In this section we describe our method for the mutual exclusion problem on bidirectional rings. A ring consists of n processes $P = \{0, 1, \ldots, n-1\}$ of finite state machines, where process i is connected with its neighboring process $i-1 \bmod n$, called a *predecessor*, and $i+1 \bmod n$, called a *successor*. In particular, $\{i-k \bmod n, \ldots, i-1 \bmod n\} \subset P$ are called k *predecessors* of $i \in P$. Each process i has a state $\sigma_i \in \Sigma_i$ consisting of a shared set of states $\sigma_i = (major_i, minor_i, dtoken_i, atoken_i, wait_i)$ with its predecessor and successor, where Σ_i is the finite state space of process i. Let $R_K = [0, \ldots, K]$ be a set of real values, over which a primary variable $major_i$ ranges, and let a function ψ map $major_i \in R_K$ into h digits. That is, $\psi(major_i)$ is represented by h integers which are stored in an array. In the domain R_K, we define K specified integer values $B_K = \{0, 1, \ldots, K-1\}$, called *bases*, such that every correct $major_i$ must take these values (as a necessary condition). A secondary integer variable $minor_i$ ranges over $I_k = \{-k-1, \ldots, k+1\} \cup \perp$, where $\perp$ represents a normal state of $minor_i$. The states of $minor_i$ are classified as follows.

- `Inactive` : Every process i has $minor_i = \perp$.
- `Testing` : At least one process i has $minor_i > 0$ or $minor_i = 0 \wedge Privilege_i$, and other process $j \in P \backslash i$ has $minor_j \not\leq 0$.
- `Responding` : At least one process i has $minor_i < 0$, and other process $j \in P \backslash i$ with $\neg Privilege_j$ has $minor_j \neq 0$.
- `Resetting` : At least one process i with $\neg Privilege_i$ has $minor_i = 0$.

In addition, σ_i contains auxiliary boolean variables $dtoken_i$, $atoken_i$ and $wait_i$, where $dtoken_i$ and $atoken_i$ are used to prevent deadlocks independently, and $wait_i$ is used to let a privielged process wait until admittance to a critical section. A *configuration* $\boldsymbol{c}$ is an n-dimensional vector of states $\boldsymbol{c} = (\sigma_0, \sigma_1, \ldots, \sigma_{n-1})$. The set of all configurations, a global state space, is denoted by $\Omega = \Sigma_0 \times \Sigma_1 \times \cdots \times \Sigma_{n-1}$. Notice that the computation for $major_i$ and $minor_i$ uses **mod** K and that for process i itself uses **mod** n. For simplicity, we drop the notation of **mod** K and **mod** n in the sequel. We assume a *state-reading model* as in [3], that is, process i can directly read (but not write) the states from σ_{i-1} and σ_{i+1}. We assume a *C-daemon* (central daemon) scheduler.

We consider a *critical section* in which only a process stays for access to a single resource. We say that a process has a *privilege* if it gains admittance to the critical section. Unlike other papers [3], [12], we distinguish the terms between *enabled* and *privileged* because not all enabled processes can get into the critical section in our protocol. The C-daemon is assumed to be *fair*, that is, every enabled process is selected infinitely often. Let $\Lambda \subset \Omega$ be a set of *legitimate* configurations as given in the following definition.

[Definition 1] A configuration $\boldsymbol{c}$ is *legitimate* ($\boldsymbol{c} \in \Lambda$) if

- every process i has $major_i \in B_K$, $dtoken_i = false$ and $atoken_i = 0$,
- there exists at most one $major_i$ in $\boldsymbol{c}$ such that $major_i + 1 = major_{i-1}$, and
- the state of $minor_i \in I_k$ is one of `Inactive`, `Testing` or `Responding`. □

2.2 Underlying Protocols

Our protocol is constructed based on two protocols. The first underlying protocol was proposed by Beauquier et al. [1]. Their method, called k-STAB, guarantees that the system stabilizes in time that depends only on k from any *k-faulty configuration*, in which the states of at most k processes differ from a legitimate configuration. The feature of their method is that the process with a major token sends an additional token, called a minor token, to at most $k+1$ predecessors before acquiring a privilege as illustrated in Fig. 2. If there is no faulty process in the predecessors, the minor token is returned to the privileged process. On the other hand, if there is some other privileged process in the predecessors, the minor token is deleted by the process. When the minor token is returned to the process with a major token, the critical section is excuted.

k-STAB for process $i \neq 0$

if (i has a major token) **then**
  if (i has just received the major token) **then**
    `send minor token to a predecessor and wait it`
  if (i has been waiting for the minor token) **then**
    `perform critical section ;`
    `send major token to a successor`
else if $\neg$(i has a major token and a minor token) **then**
  `eliminate spurious major token`
fi
`send minor token to a predecessor` (if any)

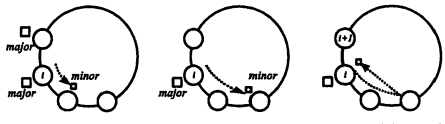

図 2 k-STAB

The second underlying protocol, called an enlarged domain protocol (EDP), was proposed by Kiniwa et al. [14], [15]. Their method guarantees that faulty processes can be detected and rapidly corrected with high probability. The feature is that the enlargement of a state space of Dijkstra's K-state protocol (注1). Every non-faulty state has to take on one of K bases. Due to the large state space, most faulty states take on some non-base value, called a non-base fault, with high probability in the random fault model. Such a process with a non-base state is enabled and reset to its neighboring base state without acquiring a privilege. Since a base fault violates the requirement of mutual exclusion and it is undetectable by local checks, we consider the base fault as a malicious fault.

EDP for process i ($i \neq 0$)

if (i has a non-base fault) **then**
  `reset to a neighboring base` (if any) ;
  `send dtoken if` i `receives it`
else if (i has a privilege) **then**
  `perform critical section` ;
  $major_i := major_{i-1}$ and `send major token to` $i+1$
fi

We briefly summarize the characteristics of the underlying protocols as follows.

	advantage	disadvantage
k-STAB	tolerant to a base fault	costly (minor token circulation)
EDP	tolerant to non-base faults, cheap (no minor token)	weak against base faults (Dijkstra-like repair)

表 1 Characteristics of Underlying Protocols

2.3 Description of Our Protocol

Here we exemplify our protocol that combines the previous two protocols. Our method guarantees both advantages in the underlying protocols. The feature of our method is that a privileged process has the option of whether or not sending a minor token. If the minor token is sent, called a strategy ST, the spreading of base fault contamination can be prevented in any k-faulty configuration. However, ST is costly because it requires to wait the minor token coming back. If the minor token is not sent, called a strategy NS, the base fault contamination may spread through the system. Notice that a non-base fault can be corrected without any aid of minor tokens. Our aim is to know optimal mixed strategies, the probabilities of ST and NS.

Our protocol for process i ($i \neq 0$)

if (i has a non-base fault) **then**
  `reset to a neighboring base` (if any) ;
  `send dtoken if` i `receives it`
else if (i has a major token) **then**
  if (i has just received the major token) **then**
    `choose ST with` p `and NS with` $1-p$
  if (i has been waiting for the minor token)
    or (i chooses NS) **then**
    `perform critical section` ;
    $major_i := major_{i-1}$ (`send major token`)
else if $\neg$(i has a major token and a minor token) **then**

(注1) : It was originally an extension of Dijkstra's 3-state protocol.

```
        eliminate spurious major token
fi
send minor token to a predecessor (if any)
```

3. Analysis

In this section, we analyze the malicious fault which is undetectable by local checks in our protocol. We summarize our setting as follows.

- We consider a multistage two-person zero sum game played by a set of privileged processes, called player A, and an adversary, called player B. Since a privilege is passed from one process to another, the process playing the role of player A also changes from one to another. We characterize a protocol as the multistage game because the conflict between processes and an adversary continues repeatedly.
- We call the time interval during which a process holds a true privilege a *stage*. The unit stage begins when a process obtains a true privilege and terminates when any fault is removed and the new process obtains a true privilege. The player B can generate faults at most once in each stage. Though the faults may pertub values in any variable, we analyze only the primary variable $major_i$. Both the player A and the player B cannot know other player's strategy in advance. A game is represented by the number of remaining stages out of m stages in total.
- A privileged process makes a choice whether or not sending a minor token. The minor token plays a role of preventing contamination from spreading. If it meets a faulty privilege, the process that has sent the minor token receives no response. Otherwise, the process receives a response and it confirms that there is no fault in the k predecessors. The reward of detecting a base fault is α and the damage of a base fault is β relative to the cost 1 of not sending a minor token when no fault occurs. On the other hand, the cost of sending a minor token (ST) takes about $2(k+1)$ times larger than not sending a minor token (NS). We assume the minor token can correct the base fault with probability r because some fault may occur after having passed a minor token.
- The goal of player A is to maximize the reward of player A, shared by a set of privileged processes, through the m stages. The mixed strategy of player A consists of sending a minor token (ST) with probability p and not sending a minor token (NS) with probability $1-p$. On the other hand, the mixed strategy of player B consists of causing no faults, base faults and non-base faults with some probabilities.

Since it is difficult to evaluate the cost of Dijkstra-like repair, we consider it from two different points of view, (Sections 3.1 and 3.2) and (Section 3.3). First, in Sections 3.1 and 3.2, we assume that the cost of Dijkstra-like repair can be constantly evaluated as β and the game continues after the repair. Next, in Section 3.3, we assume that the cost cannot be easily evaluated, thus the game terminates when the Dijkstra-like repair begins because it means player A's defeat. Thus, player B wants to increase the workload of the algorithm in Sections 3.1 and 3.2, and wants to drive the system to an unsafe state in Section 3.3.

3.1 Game Continuation after Dijkstra-like Repair

To begin with, we consider only one stage, i.e., $m = 1$. In the payoff matrix below, each row means player A's strategy and each column means player B's strategy.

	non-fault	base fault	non-base fault
ST	$-2(k+1)$	$r\alpha+(1-r)(-\beta)$	$-2(k+1)$
NS	1	$-\beta$	0

表 2 Payoff Matrix

We can exclude the case of non-fault because it is dominated by the non-base fault. Let $a = (p, 1-p)$ be the mixed strategy of player A, that is, player A takes the strategy ST with probability p and the strategy NS with probability $1-p$. Then the expected payoffs $E(a, \text{base fault})$ and $E(a, \text{non-base fault})$ of player A against the pure strategies {base fault, non-base fault} of player B are

$$E(a, \text{base fault}) = \{r\alpha-(1-r)\beta\}p-\beta(1-p), \quad (1)$$

$$E(a, \text{non-base fault}) = -2(k+1)p+0 \quad (2)$$

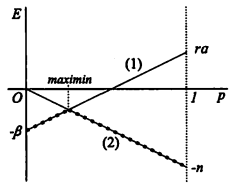

図 3 Maximin Strategy (player B's optimal counterstrategy : dotted line)

Since player A is a maximizer, the maximin value (see Fig. 3) derived from the intersection of (1) and (2) is

$$a^* = \left(\frac{\beta}{r(\alpha+\beta)+2(k+1)},\ 1-\frac{\beta}{r(\alpha+\beta)+2(k+1)}\right).$$

The game value is $-\beta 2(k+1)/(r(\alpha+\beta)+2(k+1))$. Since player B does not take the non-fault strategy, let $b = (0, q, 1-q)$ be the mixed strategy of player B, that is, player B takes the base fault strategy with probability q and the non-base fault strategy with probability $1-q$. Then, the expected payoffs of player B against the pure strategies of player A are

$$E(b, \text{ST}) = \{r\alpha-(1-r)\beta\}q-2(k+1)(1-q) \quad \text{and}$$

$$E(b, \text{NS}) = -\beta q+0.$$

Thus, the optimal strategy of player B is

$$b^* = \left(0,\ \frac{2(k+1)}{r(\alpha+\beta)+2(k+1)},\ \frac{r(\alpha+\beta)}{r(\alpha+\beta)+2(k+1)}\right).$$

The minimax value is $-2\beta(k+1)/(r(\alpha+\beta)+2(k+1))$.

Next, the game $\Gamma(m)$ for $m > 1$ can be expressed as follows.

$$\Gamma(m) = \begin{bmatrix} r\alpha-(1-r)\beta+\Gamma(m-1) & -2(k+1)+\Gamma(m-1) \\ -\beta+\Gamma(m-1) & \Gamma(m-1) \end{bmatrix}$$

For simplicity, let $\kappa = 2(k+1)$. Then, the game value v_m for $\Gamma(m)$ can be easily solved as follows.

$$\begin{aligned} v_m &= val\begin{bmatrix} r\alpha-(1-r)\beta+v_{m-1} & -\kappa+v_{m-1} \\ -\beta+v_{m-1} & v_{m-1} \end{bmatrix} \\ &= v_{m-1} + val\begin{bmatrix} r\alpha-(1-r)\beta & -\kappa \\ -\beta & 0 \end{bmatrix} \\ &= v_{m-1} + \frac{-\beta\kappa}{r(\alpha+\beta)+\kappa} \\ &= \frac{-\beta m\kappa}{r(\alpha+\beta)+\kappa}. \end{aligned}$$

Similar to the argument for $m = 1$, the optimal strategies of players A and B are

$$a^* = \left(\frac{\beta}{r(\alpha+\beta)+\kappa},\ 1-\frac{\beta}{r(\alpha+\beta)+\kappa}\right) \text{ and}$$

$$b^* = \left(0,\ \frac{\kappa}{r(\alpha+\beta)+\kappa},\ \frac{r(\alpha+\beta)}{r(\alpha+\beta)+\kappa}\right),$$

respectively.

3.2 Finite Sending of Minor Tokens

Here, we assume that we can send minor tokens at most k times, represented by $\Gamma(m,k)$, by timing constraints. By excluding the non-fault case as in Section 3.1, we have

$$\Gamma(m,k) = \begin{bmatrix} r\alpha-(1-r)\beta \\ +\Gamma(m-1,k-1) & -\kappa+\Gamma(m-1,k-1) \\ -\beta+\Gamma(m-1,k) & \Gamma(m-1,k) \end{bmatrix}.$$

The game value $v_{m,k}$ is

$$v_{m,k} = val\begin{bmatrix} r\alpha-(1-r)\beta+v_{m-1,k-1} & -\kappa+v_{m-1,k-1} \\ -\beta+v_{m-1,k} & v_{m-1,k} \end{bmatrix},$$

where $v_{h,0} = -h\beta$ and $v_{0,*} = 0$ (h is an arbitrary positive integer and $*$ is an arbitrary non-negative integer).

3.2.1 Case for $k = 1$

Suppose that we can send a minor token at most $k = 1$. Then, the game value is

$$\begin{aligned} v_{m,1} &= val\begin{bmatrix} r\alpha-(1-r)\beta+v_{m-1,0} & -\kappa+v_{m-1,0} \\ -\beta+v_{m-1,1} & v_{m-1,1} \end{bmatrix} \\ &= val\begin{bmatrix} r(\alpha+\beta)-m\beta & -\kappa-(m-1)\beta \\ -\beta+v_{m-1,1} & v_{m-1,1} \end{bmatrix}. \end{aligned}$$

For simplicity, we denote $v_{m,1}$ by v_m. The expected payoffs of player A's mixed strategy $a = (p, 1-p)$ against player B's pure strategies are

$$\begin{aligned} E(a, \text{base fault}) &= p\cdot\{r(\alpha+\beta)-m\beta\} \\ &\quad +(1-p)\cdot(-\beta+v_{m-1}) \quad \text{and} \\ E(a, \text{non-base fault}) &= p\cdot(-\kappa-(m-1)\beta)+(1-p)\cdot v_{m-1}. \end{aligned}$$

Since the intersection of them is $p = \beta/(r(\alpha+\beta)+\kappa)$, we have

$$v_m = -p\{(m-1)\beta+\kappa\}+(1-p)v_{m-1}.$$

Hence,

$$\begin{aligned} v_m+(m-1)\beta+\kappa &= (1-p)\{v_{m-1}+(m-1)\beta+\kappa\} \\ &\vdots \\ &= (1-p)^m\{v_0+(m-1)\beta+\kappa\} \end{aligned}$$

Since $v_0 = 0$ and $p = \beta/(r(\alpha+\beta)+\kappa)$, we obtain

$$v_m = ((m-1)\beta+\kappa)\left\{\left(1-\frac{\beta}{r(\alpha+\beta)+\kappa}\right)^m - 1\right\}.$$

The player A's optimal strategy a^* until sending a minor token once is

$$a^* = \left(\frac{\beta}{r(\alpha+\beta)+\kappa},\ 1-\frac{\beta}{r(\alpha+\beta)+\kappa}\right).$$

The expected payoffs of player B's mixed strategy $b = (0, q, 1-q)$ against player A's pure strategies are

$$\begin{aligned} E(b, \text{ST}) &= q(r(\alpha+\beta)-m\beta)+(1-q)(-\kappa-(m-1)\beta) \text{ and} \\ E(b, \text{NS}) &= q(-\beta+v_{m-1})+(1-q)v_{m-1}. \end{aligned}$$

The intersection of them is

$$q = \frac{(m-1)\beta+\kappa+v_{m-1}}{r(\alpha+\beta)+\kappa}.$$

Thus, the player B's optimal strategy b^* for remaining m stages is

$$b^* = \left(0,\ \frac{(m-1)\beta+\kappa+v_{m-1}}{r(\alpha+\beta)+\kappa},\ 1-\frac{(m-1)\beta+\kappa+v_{m-1}}{r(\alpha+\beta)+\kappa}\right),$$

where $\kappa = 2(k+1)$ and $v_{m-1} = ((m-2)\beta+\kappa)\{(1-\beta/(r(\alpha+\beta)+\kappa)^{m-1}-1\}$.

3.3 Game Termination when Dijkstra-like Repair

If Dijkstra-like repair begins, we consider here that our game cannot be continued because the cost of Dijkstra-like repair cannot be easily evaluated. Since the non-fault case can be excluded as the previous section, the game $\Gamma(m)$ is

$$\Gamma(m) = \begin{bmatrix} r(\alpha+\Gamma(m-1)) \\ +(1-r)(-\beta) & -2(k+1)+\Gamma(m-1) \\ -\beta & \Gamma(m-1) \end{bmatrix}.$$

Let $\kappa = 2(k+1)$. Then, the game value is represented by

$$v_m = val\begin{bmatrix} r(\alpha+v_{m-1})+(1-r)(-\beta) & -\kappa+v_{m-1} \\ -\beta & v_{m-1} \end{bmatrix}.$$

The expected payoffs of player A's mixed strategy $a = (p, 1-p)$ against player B's pure strategies are

$$\begin{aligned} E(a, \text{base fault}) &= p\{r(\alpha+v_{m-1})-\beta(1-r)\} \\ &\quad +(1-p)(-\beta) \text{ and} \\ E(a, \text{non-base fault}) &= p(-\kappa+v_{m-1})+(1-p)v_{m-1}. \end{aligned}$$

Then, the intersection of them is

$$p = \frac{v_{m-1}+\beta}{r(v_{m-1}+\alpha+\beta)+\kappa}.$$

Since the game value is $v_m = v_{m-1} - p\kappa$,

$$\begin{aligned} v_m &= v_{m-1} - \frac{\kappa(v_{m-1}+\beta)}{r(v_{m-1}+\alpha+\beta)+\kappa} \qquad (3) \\ &= v_{m-1} + \frac{\kappa}{r}\cdot\frac{\alpha+\kappa/r}{v_{m-1}+\alpha+\beta+\kappa/r} - \frac{\kappa}{r}. \end{aligned}$$

By (3), notice that $v_m \to -\beta$ holds when $m \to \infty$. For simplicity, we use $(\kappa/r)(\alpha+\kappa/r) = \delta$. By adding $\alpha+\beta+\kappa/r$, we have

$$\begin{aligned} v_m+\alpha+\beta+\frac{\kappa}{r} &= v_{m-1}+\alpha+\beta+\frac{\kappa}{r} \\ &\quad +\frac{\delta}{v_{m-1}+\alpha+\beta+\kappa/r} - \frac{\kappa}{r}. \end{aligned}$$

Furthermore, if we use $v_m+\alpha+\beta+\kappa/r = u_m$,

$$u_m = u_{m-1} + \frac{\delta}{u_{m-1}} - \frac{\kappa}{r} = u_{m-1}\left(1+\frac{\delta}{u_{m-1}^2} - \frac{\kappa}{ru_{m-1}}\right).$$

By taking the logarithm,

$$\begin{aligned} \ln u_m &= \ln u_{m-1} + \ln\left(1+\frac{\delta}{u_{m-1}^2} - \frac{\kappa}{ru_{m-1}}\right) \\ &\vdots \\ &= \ln u_0 + \sum_{j=0}^{m-1} \ln\left(1+\frac{\delta}{u_j^2} - \frac{\kappa}{ru_j}\right). \end{aligned}$$

Since $-\beta < v_m < 0$, we have $\alpha+\kappa/r < u_m < \alpha+\beta+\kappa/r$. Thus

$$\frac{-\beta}{r(\alpha+\beta+\kappa/r)^2} < \frac{\delta}{u_j^2} - \frac{\kappa}{ru_j} < 0$$

Letting $\delta/u_j^2 - \kappa/ru_j = -\epsilon_j$ and $u_m \simeq \alpha+\beta e^{-m\epsilon_m}+\kappa/r$, we obtain $v_m \simeq \beta(e^{-m\epsilon_m}-1)$, where $\epsilon_m = \kappa\beta e^{-m\epsilon_m}/r(\alpha+\beta e^{-m\epsilon_m}+\kappa/r)^2$. Then, the player A's optimal strategy a_m^* for remaining m stages is

$$a_m^* \simeq \left(\frac{\beta e^{-(m-1)\epsilon_{m-1}}}{r(\beta e^{-(m-1)\epsilon_{m-1}}+\alpha)+2(k+1)}, \right. \\ \left. 1 - \frac{\beta e^{-(m-1)\epsilon_{m-1}}}{r(\beta e^{-(m-1)\epsilon_{m-1}}+\alpha)+2(k+1)}\right).$$

The expected payoffs of player B's mixed strategy $b = (0, q, 1-q)$ against player A's pure strategies are

$$\begin{aligned} E(b, \text{ST}) &= q\{r(\alpha+v_{m-1})+(1-r)(-\beta)\} \\ &\quad +(1-q)(-\kappa+v_{m-1}) \text{ and} \\ E(b, \text{NS}) &= q(-\beta)+(1-q)v_{m-1}. \end{aligned}$$

Then, the player B's optimal strategy b_m^* is

$$b_m^* \simeq \left(0, \frac{2(k+1)}{r(\beta e^{-(m-1)\epsilon_{m-1}}+\alpha)+2(k+1)}, \right. \\ \left. 1 - \frac{2(k+1)}{r(\beta e^{-(m-1)\epsilon_{m-1}}+\alpha)+2(k+1)}\right).$$

4. Conclusion

In this paper, we developed not only a game theoretic analysis for a malicious fault, but also a general framework for strengthening an algorithm against the fault. The method is

(1) develop two algorithms that complement each other,

(2) combine them and specify their strategies to be executed with some probabilities,

(3) construct a payoff matrix against an adversary, and

(4) determine maximin values and an optimal strategy.

In our analysis, we assumed a general setting that includes probability r of repairing a base fault and rewards α and β. If we simplify them, e.g., $r = 1$ and $\alpha = \beta$, we obtain an intuitive result. That is, in the case the game is continued after Dijkstra-like repair, player A's optimal strategy is

$$a^* = \left(\frac{\alpha}{2(\alpha+k+1)}, \frac{\alpha+2(k+1)}{2(\alpha+k+1)}\right).$$

Thus, if α is sufficiently larger than $k+1$, the optimal strategy in our protocol is approximately $a^* \simeq (1/2, 1/2)$. Or if α is almost equal to $k+1$, the optimal strategy is $a^* \simeq (1/4, 3/4)$.

文　献

[1] J.Beauquier, C.Genolini and S.Kutten, "Optimal reactive k-stabilization: the case of mutual exclusion," In *Proceedings of the 18th Annual ACM Symposium on Principles of Distributed Computing*, (1999) 209–218.

[2] A.Dasgupta, S.Ghosh and S.Tixeuil, "Selfish stabilization," In *Proceedings of the 8th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, (2006) LNCS 4280:231–243.

[3] E.W.Dijkstra, "Self-stabilizing systems in spite of distributed control," *Communications of the ACM*, **17**, 11 (1974) 643–644.

[4] E.W.Dijkstra, "A belated proof of self-stabilization," *Distributed Computing*, **1** (1986) 5–6.

[5] S.Dolev, A.Israeli and S.Moran, "Analyzing expected time by scheduler-luck games," *IEEE Transactions on Software Engineering*, **21**, 5 (1995) 429–439.

[6] S.Ghosh and A.Gupta, "An exercise in fault-containment: Self-stabilizing leader election," *Information Processing Letters*, **59** (1996) 281–288.

[7] S.Ghosh, A.Gupta, T.Herman and S.V.Pemmaraju, "Fault-containing self-stabilizing algorithms," In *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing*, (1996) 45–54.

[8] S.Ghosh and X.He, "Fault-containing self-stabilization using priority scheduling," *Information Processing Letters*, **73** (2000) 145–151.

[9] T.Herman, "Superstabilizing mutual exclusion," *Dis-*

tributed Computing, **13**, 1 (2000) 1–17.

[10] T.Herman and S.Pemmaraju, "Error-detecting codes and fault-containing self-stabilization," *Information Processing Letters*, **73** (2000) 41–46.

[11] R.Hohzaki, "A compulsory smuggling model of inspection game taking account of fulfillment probabilities of players' aims," *Journal of the Operations Research Society of Japan*, **49**, 4 (2006) 306–318.

[12] Y.Katayama, E.Ueda, H.Fujiwara and T.Masuzawa, "A latency optimal superstabilizing mutual exclusion protocol in unidirectional rings," *Journal of Parallel and Distributed Computing*, **62**, 5 (2002) 865-884.

[13] J.Kiniwa, "Avoiding faulty privileges in fast stabilizing rings," *IEICE Transactions on Fundamentals*, **E85-A**, 5 (2002) 949–956.

[14] J.Kiniwa, "How to improve safety under convergence using stable storage," *IEEE Transactions on Parallel and Distributed Systems*, **17**, 4 (2006) 389–398.

[15] J.Kiniwa and M.Yamashita, "A randomized 1-latent, time-adaptive and safe self-stabilizing mutual exclusion protocol," *Parallel Processing Letters*, **16**, 1 (2006) 53–61.

[16] E.Koutsoupias and C.H.Papadimitriou, "Worst-Case Equilibria," In *Proceedings of the 16th Annual Symposium on the Theory of Computing (STACS)*, LNCS:1563, (1999) 404-413.

[17] N.Nisan, T.Roughgarden, E.Tardos and V.V.Vazirani (ed.), "Algorithmic game theory," *Cambridge University Press* (2007).

[18] R.B.Myerson, "Game theory : analysis of conflict," *Harvard University Press* (1991).

[19] M.Nesterenko and S.Tixeuil, "Tolerance to unbounded byzantine faults," In *Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems*, (2002) 22–29.

[20] Y-S.Wu, B.Foo, Y-C.Mao, S.Bagchi and E.H.Spafford, "Automated adaptive intrusion containment in systems of interacting services," *Computer Networks*, **51**, 5 (2007) 1334–1360.

[21] I-L.Yen, "A highly safe self-stabilizing mutual exclusion algorithm," *Information Processing Letters*, **57** (1996) 301–305.

付　　録

$$
\begin{aligned}
OnBase_i &\equiv (major_i \in B)\\
LeftDtoken_i &\equiv (i \neq 0) \wedge (dtoken_{i-1} = true)\\
&\quad \wedge (dtoken_i = false)\\
ReturnDtoken_0 &\equiv (dtoken_0 = true) \wedge (dtoken_{n-1} = true)\\
FirstGet_i &\equiv (wait_i = false)\\
MinPriv_i &\equiv (minor_{i-1} = -1) \wedge (minor_i = 0)\\
Privilege_i &\equiv OnBase_i \wedge (major_i \neq major_{i-1}) \wedge (i \neq 0)\\
Privilege_0 &\equiv OnBase_0 \wedge (major_0 = major_{n-1})\\
MinInit_i &\equiv Privilege_i \wedge (minor_i = \perp)\\
MinExtend_i &\equiv ((Privilege_{i+1} \wedge minor_{i+1} = 0) \vee (0 <\\
&\quad minor_{i+1} \leqq k)) \wedge (0 \ngeq minor_i \neq minor_{i+1} + 1)\\
MinShrink_i &\equiv (minor_i = k+1) \vee (minor_{i-1} < 0 \wedge minor_i > 0)\\
MinReset_i &\equiv (minor_i \neq \perp) \wedge (minor_{i-1} = 0 \vee (minor_{i+1} = 0\\
&\quad \wedge \neg Privilege_{i+1}))\\
MinReturn_i &\equiv (minor_i \leqq 0) \wedge (minor_{i+1} < 0 \vee minor_{i+1} = \perp)\\
&\quad \wedge \neg Privilege_i
\end{aligned}
$$

Our protocol for process i:

if $\neg OnBase_i$ **then** *HandleFault*(i)
else if $Privilege_i \wedge MinPriv_i$ **then**
  if $FirstGet_i$ **then** $wait_i := true$ (with probability p)
    else $wait_i := false$ (with probability 1)
  if ($\neg wait_i$) **then**
    `perform critical section ;`
    SendMajor(i) ;
    SetVar(i) **fi**
else if $\neg(Privilege_i \wedge MinPriv_i)$ **then**
  SendMajor(i) **fi** ;
SendMinor(i) ; *AntiDlock*(i)

Function *HandleFault*(i):
if $OnBase_{i-1} \vee OnBase_{i+1}$ **then** *BaseReset*(i)
else if $(i = 0) \wedge (dtoken_0 = false)$ **then** $dtoken_0 := true$
else if $(i = 0) \wedge ReturnDtoken_0$ **then** *BaseReset*(0)
else if $LeftDtoken_i$ **then** $dtoken_i := true$

Function *BaseReset*(i):
if $(i \neq 0) \wedge OnBase_{i-1}$ **then** $major_i := major_{i-1}$
else if $(i \neq n-1) \wedge OnBase_{i+1}$ **then** $major_i := major_{i+1}$
else if $(i = 0) \wedge ReturnDtoken_0$ **then**
    $major_0 := base_j$ for some $j \in B$ **fi** ;
SetVar(i)

Function *SetVar*(i):
$dtoken_i := false$; $wait_i := false$; $minor_i := \perp$; $atoken_i := 0$

Function *SendMajor*(i):
if $(i = 0)$ **then**
  if $Privilege_0$ **then** $major_0 := major_{n-1} + 1$
else if $(0 < i \leqq n-1)$ **then**
  if $Privilege_i$ **then** $major_i := major_{i-1}$

Function *SendMinor*(i):
if $MinInit_i$ **then** $minor_i := 0$;
if $MinExtend_i$ **then**
  if $\neg Privilege_i$ **then** $minor_i := minor_{i+1} + 1$;
  if $Privilege_i$ **then** $minor_i := 0$ **fi** ;
if $MinShrink_i$ **then** $minor_i := -minor_i$;
if $MinReset_i$ **then** $minor_i := 0$;
if $MinReturn_i$ **then** $minor_i := \perp$

Function *AntiDlock*(i):
if $(minor_i > 0) \vee (minor_i = 0 \wedge Privilege_i)$ **then**
  if $(atoken_0 = 0) \wedge (atoken_{n-1} = 0)$ **then** $atoken_0 := 1$
  if $(atoken_{i+1} = 1) \wedge (atoken_i = 0)$ **then** $atoken_i := atoken_{i+1}$
  if $(atoken_0 = 1) \wedge (atoken_{n-1} = 1)$ **then** $atoken_0 := 2$
  if $(atoken_{i-1} = 2) \wedge (atoken_i = 1)$ **then** $major_i := major_{i-1}$;
    $minor_i := \perp$
else $atoken_i := 0$