An Advanced Performance Analysis of Self-stabilizing Protocols : Stabilization Time with Transient Faults during Convergence

Yoshihiro Nakaminami[1], Hirotsugu Kakugawa[1], and Toshimitsu Masuzawa[1]

[1]Osaka University
Dept. of Information Science and Technology
Toyonaka, Osaka 560-8531 Japan
{nakaminm, kakugawa, masuzawa}@ist.osaka-u.ac.jp

Abstract

A self-stabilizing protocol is a brilliant framework for fault tolerance. It can recover from any number and any type of transient faults and eventually converge to its intended behavior. Performance of a self-stabilizing protocol is usually measured by stabilization time: the time required to complete the convergence to its intended behavior under the assumption that no new fault occurs during the convergence. But a self-stabilizing protocol has no guarantee to complete the convergence if faults are frequently occurred.

This paper brings new light to efficiency analysis of stabilization. The efficiency is evaluated with consideration for faults occurring during the convergence. To show the feasibility and effectiveness of the approach, this paper applies the approach to the maximal matching protocol.

1 Introduction

A self-stabilizing protocol[2] is a protocol that achieves its intended behavior regardless of the initial network configuration (i.e., global state). Thus, a self-stabilizing protocol is resilient to any number and any type of transient faults and is adaptive to any number and any type of topological changes of networks: after the last fault or the last topological change occurs, the protocol starts to converge to its intended behavior. These advantages make self-stabilizing protocols extremely attractive for designing highly dependable distributed systems. The self-stabilization has attracted a great deal of attention of researchers and practitioners working in the field of distributed systems.

This work is supported in part by JSPS, Grant-in-Aid for Scientific Research ((B)15300017), JSPS, Grant-in-Aid for Encouragement of Young Scientisit (15700017), MIC, Strategic Information and Communications R&D Promotion Programme, and MEXT, "The 21st Century Center of Excellence Program".

A self-stabilizing protocol can converge to its intended behavior from any configuration, but *the convergence is guaranteed only when no new fault or topological change occurs during the convergence.* Performance of a self-stabilizing protocol is usually measured by stabilization time : the time required to complete the convergence under the assumption that no new fault occurs during the convergence. In other words, the stabilization time guarantees that the self-stabilizing protocol can complete the convergence if no new fault or topological change occurs during the period of the stabilization time. In distributed systems where faults or topological changes repeatedly occur with interval shorter than the stabilization time, a self-stabilizing protocol cannot guarantee to converge to its intended behavior.

On the other hand, because of a rapid increase in the size of distributed systems and development of dynamic distributed systems such as P2P systems and mobile systems, self-stabilizing protocols are highly desired to tolerate frequent faults or topological changes. However, to the best of our knowledge, the problem has been tackled by few work and waits further investigation.

1.1 Contribution of This Paper

It is generally said that self-stabilizing protocols are inefficient in distributed systems with frequent faults or topological changes. Its main reason is that a new fault or topological change brings the system into an unexpected configuration, and thus, the system restarts convergence to its intended behavior *from scratch*. But the reasoning seems too pessimistic. Should a single new fault or topological change spoil all the efforts a self-stabilizing protocol made before the disturbance? This is the question that motivated us to start this research.

In this paper, we propose a new measure, **stabilization time with $\sharp F$ faults**, of efficiency in convergence: it is the worst case time a self-stabilizing protocol requires to converge to its intended behavior from any configuration despite occurrence of $\sharp F$ faults during the convergence. The stabilization time with $\sharp F$ faults is usually represented as a function of network parameters (e.g., the numbers of processes and links, diameter, degree etc.) and the number $\sharp F$ of faults. Another contribution we make in this paper is to propose an analysis method to evaluate the stabilization time with $\sharp F$ faults. Roughly speaking, we regard each action of a protocol as a forward step to the convergence, and occurrence of fault as a backward step. By quantifying influence of the forward step and the backward step, we can estimate the stabilization time with $\sharp F$ faults.

To show the feasibility and effectiveness of the proposed method, we evaluate the stabilization time with $\sharp F$ faults of a maximal matching protocol proposed by Hsu and Huang [7] and show that it is $2m+n+4\Delta\cdot\sharp F$ where n and m are the numbers of processes and links respectively, and Δ is the maximal degree of all processes. Notice that the stabilization time with $\sharp F$ faults can be considered as a generalization of the already known (usual) stabilization time, $2m+n$ [5].

1.2 Related Work

Many work considered occurrence of transient faults after convergence and designed self-stabilizing protocols that can recover from the transient faults efficiently. The concepts of time-adaptability [8], fault-containment [3, 4] and continuous containment [11] are proposed to approach the subject. While all of them consider the impact of transient faults at a legitimate configuration (i.e., after convergence), our desired goal is to consider the impact of transient faults during convergence and to design self-stabilizing protocols that can achieve efficient convergence despite transient faults during the convergence.

The paper [9] is most related to this paper. It considers self-stabilizing mutual exclusion on a dynamic ring network. It evaluates efficiency of convergence with considering process joins and leaves during convergence. The paper insists that the analysis method can be applied to convergence analysis with consideration for transient faults during convergence. However, the protocol considered in the paper is vulnerable to transient faults during convergence, and thus the analysis with transient faults is left as a future work. In this paper, we extend the proposed method to the analysis with consideration for transient faults during convergence and prove that the method is applicable.

1.3 Organization of This Paper

The rest of this paper is organized as follows. Section 2 introduces the computation model, faults and the stabilization time with $\sharp F$ faults. Section 3 proposes the method to evaluate a stabilization time of $\sharp F$ faults. Section 4 analyzes the stabilization time of $\sharp F$ faults for the maximal matching protocol proposed by Hsu and Huang [7]. We conclude this paper in Section 5.

2 Model

A **distributed system** $\mathcal{S} = (P, L)$ consists of a set $P = \{v_0, v_1, \ldots, v_{n-1}\}$ of processes and a set L of (communication) links. A link connects two distinct processes. When a link connects processes v and w, this link is denoted by (v, w). We say w is a **neighbor** of v if $(v, w) \in L$. The set of all neighbors of v is denoted by N_v. The number of neighbors of v is denoted by $|N_v|$ and $|N_v|$ is called the degree of v. The degree of $\mathcal{S}$ is the maximum degree of all processes and is denoted by Δ ($\Delta = \max_{v \in V}\{|N_v|\}$).

Each process v is a state machine. Each process can directly read its neighbors' states and can change its own state according to its current state and its neighbors' states. The **normal action** of each process v is defined by a set of *guarded actions* in the following form:

$$\langle guard_v \rangle \rightarrow \langle statement_v \rangle$$

The **guard** $\langle guard_v \rangle$ of process v is a boolean expression on its own state and its neighbors' states. When the guard is evaluated to be true, $\langle statement_v \rangle$ is executed and changes the state of v. In this paper, we assume that the guarded action can be atomically executed: evaluation of the guard and execution of the statement are executed in one atomic action. This model is called the shared-state model.

In this paper, a transient fault of v is modeled by a **faulty action** of v. A faulty action of v can change v's state arbitrarily. A faulty action models undesirable reset of the process, memory corruption and so on in real distributed systems.

A **configuration** (i.e., a global state) of a distributed system is specified by an n-tuple $\sigma = (s_0, s_1, \ldots, s_{n-1})$ where s_i stands for the state of process v_i. A process v is said to be **enabled** at a configuration σ when v has a normal action whose guard is true at σ. A process v is said to be **disabled** at σ when it is not enabled at σ.

As the execution model, we adopt the **central daemon** (sometimes called the *sequential execution model*) where only a single process can execute an action between two consecutive configurations.

Now we consider transition of the consecutive configurations σ and σ'. The transitions are classified into the following two cases:

1. Transition by a normal action of a process v_i: Let v_i be an enabled process at $\sigma = (s_0, s_1, \ldots, s_{n-1})$ and $\sigma' = (s'_0, s'_1, \ldots, s'_{n-1})$ be the configuration resulting from σ by v_i's normal action. A normal action of v_i changes its state from s_i to s'_i depending on s_i and its neighbors' states at σ. Notice that $s_j = s'_j$ holds for each $j(\neq i)$.

2. Transition by a faulty action of a process v_i: Let $\sigma = (s_0, s_1, \ldots, s_{n-1})$ and $\sigma' = (s'_0, s'_1, \ldots, s'_{n-1})$ be consecutive configurations where σ' is resulted from σ by v_i's faulty action. A faulty action of v_i changes its state from s_i to s'_i, where s'_i can be an arbitrary state. Notice that $s_j = s'_j$ holds for each $j(\neq i)$. The faulty action can occur at any configuration.

When configurations are changed from σ to σ' by a normal action or a faulty action, the transition is denoted $\sigma \mapsto \sigma'$.

An **execution** of a protocol is represented by an infinite sequence of configurations in the order they appear in the execution: an infinite sequence of configurations $E = \sigma_0, \sigma_1, \ldots$ is an execution if and only if $\sigma_i \mapsto \sigma_{i+1}$ holds for every $i (i \geq 0)$. A partial execution is denoted by $E[\sigma_i, \sigma_j] = \sigma_i, \sigma_{i+1}, \ldots, \sigma_j (i \leq j)$. An execution is called **fault-free** if it contains no transition by a faulty action.

The execution is not uniquely determined only from the initial configuration, since there may exist several actions that can be executed at each configuration: normal actions of enabled processes and faulty actions of all processes. One of them is arbitrarily chosen to be executed. In this paper, we assume that every execution is **weakly fair**: every enabled process eventually executes its normal action unless it becomes disabled.

A **problem** is specified by legitimate predicates. We say a configuration is **legitimate** when it satisfies the legitimate predicates. Protocols are required to reach a legitimate configuration specified by a problem.

A **self-stabilizing protocol** is resilient to any number and any type of transient faults. A self-stabilizing protocol satisfies two properties, **convergence** and **closure**, under the assumption that no fault occurs. The convergence means that a system eventually reaches a legitimate configuration from any initial configuration. The closure means that a system never goes out from the legitimate configurations once it reaches a from any legitimate configuration. **Stabilization time** is a performance measure for self-stabilizing protocols. The stabilization time is the worst case time complexity required to reach a legitimate configuration from any initial configuration under the assumption that no faulty action occurs.

Definition 1 *The stabilization time of a protocol is $f(\mathcal{S})$ if any fault-free execution E from any initial configuration σ_0 satisfies the followings:*

> *Let σ be any configuration in E and $\sharp N$ be the number of normal actions in $E[\sigma_0, \sigma]$. If $\sharp N \geq f(\mathcal{S})$ holds, then there is a legitimate configuration in $E[\sigma_0, \sigma]$.* □

The stabilization time $f(\mathcal{S})$ is usually a function with parameters of the distributed system $\mathcal{S}$ such as the numbers of processes and links, diameter, degree and so on.

In this paper, we try to evaluate the worst case time complexity required to converge to a legitimate configuration with considering transient faults during the convergence.So we define the **stabilization time with $\sharp F$ faults** as a new performance measure of self-stabilizing protocols.

Definition 2 *For a self-stabilizing protocol, the stabilization time with $\sharp F$ faults is $g(\mathcal{S}, \sharp F)$ if any execution E from any initial configuration σ_0 satisfies the followings:*

> *Let σ be any configuration in E and $\sharp N$ and $\sharp F$ be the numbers of normal actions and faulty actions in $E[\sigma_0, \sigma]$ respectively. If $\sharp N \geq g(\mathcal{S}, \sharp F)$ holds, then there is a legitimate configuration in $E[\sigma_0, \sigma]$.* □

The stabilization time with $\sharp F$ faults $g(\mathcal{S}, \sharp F)$ is a function that has parameter $\sharp F$ in addition to these of network characteristics.

Remark: Under the assumption that no faulty action occurs, the closure property guarantees that all configurations after a legitimate configuration are also legitimate. Thus, we can rephrase the last sentence of Definition 1 as follows: If $\sharp N \geq f(\mathcal{S})$ holds, then σ is a legitimate configuration. However we cannot rephrase

Definition 2 in a similar way since the closure property does not hold when a faulty action occurs: for example, configuration σ cannot be legitimate when a faulty action occurs immediately before σ.

3 Proposed Method

In this section, we propose a method to evaluate the stabilization time with $\sharp F$ faults. Roughly speaking, we regard each normal action as a forward step to the convergence, and each faulty action as a backward step. We will define a **variant function** to represent the distance (in some sense) of a configuration from legitimate configurations. The value of the variant function decreases by each normal action, and may increase by faulty actions. A system is legitimate if a variant function is equal to zero.

Definition 3 *Let Σ be a set of possible configurations of a distributed system. For a self-stabilizing protocol, a variant function $\mathcal{F}\colon \Sigma \to \mathbb{Z}$ is a function from a configuration to a non-negative integer that satisfies the following properties:*

- $\exists M\colon \forall \sigma \in \Sigma\colon \mathcal{F}(\sigma) \leq M$
- *A configuration σ is legitimate if $\mathcal{F}(\sigma) = 0$*
- *There is an integer x satisfies the followings for any transition $\sigma \mapsto \sigma'$ caused by a normal action:*
$$\max\{\mathcal{F}(\sigma) - x, 0\} \geq \mathcal{F}(\sigma')$$
- *There is an integer y satisfies the followings for any transition $\sigma \mapsto \sigma'$ caused by a faulty action:*
$$\mathcal{F}(\sigma) + y \geq \mathcal{F}(\sigma')$$

Using the variant function, we can estimate the stabilization time with $\sharp F$ faults.

Theorem 1 *Let $\mathcal{F}$ be a variant function. Let M, x, and y be the constants used in Definition 3. The stabilization time with $\sharp F$ faults is at most $\lceil \frac{M + \sharp F \cdot y}{x} \rceil$.*

Proof: Proof by contradiction. Assume that there is no legitimate configuration before the time when $\lceil \frac{M + \sharp F \cdot y}{x} \rceil$-th normal action is executed. Then each normal action makes $\mathcal{F}$ decrease so the total sum which all normal actions make $\mathcal{F}$ decrease is at least $M + \sharp F \cdot y$. On the other hand, the total sum which all faulty actions make $\mathcal{F}$ increase is at most $\sharp F \cdot y$. The maximum of $\mathcal{F}$ is M. Therefore $\mathcal{F}$ becomes zero before the time when the $\lceil \frac{M + \sharp F \cdot y}{x} \rceil$-th normal action is executed and the configuration is legitimate at that time because of the definition of $\mathcal{F}$. □

Using $\mathcal{F}$, we present a sufficient condition that a configuration is legitimate.

Theorem 2 *A configuration σ is legitimate if there is a preceding configuration δ which satisfies the following property:*

Let $E[\delta, \sigma] = \sigma_0, \sigma_1, \ldots, \sigma_k$ be a partial execution where $\sigma_0 = \delta$ and $\sigma_k = \sigma$. Then the value of h computed by the following procedure is equal to zero.

```
$h = M$
for $i := 0$ to $k - 1$
    if ($\sigma_i \mapsto \sigma_{i+1}$ is caused by a normal action)
    then
        $h := \max\{f - x, 0\}$
    else % $\sigma_i \mapsto \sigma_{i+1}$ is caused by a faulty action
        $h := \min\{f + y, M\}$
    endif
end
```

Proof: Let m_i be a value of function $\mathcal{F}$ at σ_i and h_i be a value of h when the i-th iteration of the "for" statement is executed. We show that $m_i \leq h_i$ holds for any i ($0 \leq i \leq k$). Clearly, $m_0 \leq h_0$ holds because of $h_0 = M$ and $\mathcal{F} \leq M$. Assume that $m_i \leq h_i$ holds. It is sufficient to consider the following three cases for the transition $\sigma_i \mapsto \sigma_{i+1}$.

1. Case that $\sigma_i \mapsto \sigma_{i+1}$ is caused by a normal action and $m_{i+1} = 0$ holds: From the procedure, $h_{i+1} \geq 0$ holds so $m_{i+1} \leq h_{i+1}$ holds.

2. Case that $\sigma_i \mapsto \sigma_{i+1}$ is caused by a normal action and $m_{i+1} > 0$ holds: From the definition of $\mathcal{F}$, $0 < m_{i+1} \leq m_i - x$ holds. On the other hand, $h_{i+1} = \max\{h_i - x, 0\}$ holds. From the induction hypothesis, $m_{i+1} \leq h_{i+1}$ holds.

3. Case that $\sigma_i \mapsto \sigma_{i+1}$ is caused by a faulty action: From the definition of $\mathcal{F}$, $m_i + y < m_{i+1} \leq M$ holds. On the other hand, $h_{i+1} = \min\{h_i + y, M\}$ holds. From the induction hypothesis, $m_{i+1} \leq h_{i+1}$ holds. □

4 Self-stabilizing Maximal Matching

In this section, we analyze the stabilization time with $\sharp F$ faults of the self-stabilizing maximal matching protocol proposed by Hsu and Huang [7]. The protocol is called Hsu-Huang protocol in the followings. A *matching* of $\mathcal{S} = (P, L)$ is a subset of links $L'(L' \subseteq L)$ in which no two links connect to a common process. A matching L' is *maximal* if it is not properly contained in any other matching.

The Hsu-Huang protocol was proposed in [7] and was analyzed in [10, 5]. In the protocol, each node v maintains a single variable which is either null, denoted by $v \succ null$, or points to one of its neighbors $w(\in N_v)$, denoted by $v \succ w$. For $(v, w) \in L$, if $v \succ w$ and $w \succ v$ hold, (v, w) is regarded as a *matched link*. The protocol consists of three guarded actions R1, R2 and R3 (see Fig. 1).

The following theorem is shown in [7, 5]:

Theorem 3 *The Hsu-Huang protocol is a self-stabilizing protocol and its stabilization time is $2m + n$ where m is the number of links.* □

Now we analyze the stabilization time with $\sharp F$ faults of the Hsu-Huang protocol. From Fig. 1, each action needs a neighbor denoted by w in Fig. 1 to be executed. So we say that a normal action is executed for (v, w) where v is a process and w is the neighbor of v. More precisely, a normal action of R1 or R2 is said to be executed for link (v, w), when it change v's variable from $v \succ null$ into $v \succ w$. Also, a normal action of R3 is said to be executed for link (v, w), when it change v's variable from $v \succ w$ into $v \succ null$. We use (v, w, Rk) to denote that a normal action $Rk(1 \leq k \leq 3)$ is executed for (v, w). Let $c(v, w)$ be the number of normal actions that can be executed for (v, w) or (w, v) under the assumption that no faulty action occurs. Note that $c(v, w) = c(w, v)$.

Lemma 1 [5] *Let v and w be processes such that $(v, w) \in L$. If no faulty action occurs at any process, then the followings hold:*

1. *When $(v, w, R1)$ is executed, no more normal action can be executed for (v, w) (i.e., $c(v, w) = 0$).*
2. *When $(v, w, R2)$ is executed, exactly one additional normal action can be executed for (v, w) (i.e., $c(v, w) = 1$), and it is $(v, w, R3)$ or $(w, v, R1)$.*
3. *When $(v, w, R3)$ is executed, at most two additional normal actions can be executed for (v, w) (i.e., $c(v, w) \leq 2$), and the next normal action is $(v, w, R2)$ or $(w, v, R2)$.* □

```
R1:   $(v \succ null) \land (\exists w \in N_v :: w \succ v)$
             $\rightarrow (v \succ w)$
R2:   $(v \succ null) \land (\forall w \in N_v :: \neg(w \succ v))$
          $\land(\exists w \in N_v :: w \succ null)$
             $\rightarrow (v \succ w)$
R3:   $(v \succ w) \land (w \succ x) \land (x \neq v)$
             $\rightarrow (v \succ null)$
```

Figure 1. Actions in the Hsu-Huang protocol of v

Let $f_{mm}(\sigma)$ be the function that returns the worst case (i.e., maximum) number of normal actions executed to reach a legitimate configuration under the assumption that no faulty action occurs at any process. From Theorem 3, $f_{mm}(\sigma)$ is at most $2m + n$ for any configuration σ. Now we analyze impact of executing a normal action and a faulty action on the value of f_{mm}.

Lemma 2 *Let σ and σ' be configurations such that $\sigma \mapsto \sigma'$. If its transition is caused by a normal action, $f_{mm}(\sigma') \leq f_{mm}(\sigma) - 1$ holds.*

Proof: It is obvious from the definition of f_{mm}. □

Lemma 3 *Let σ and σ' be configurations such that $\sigma \mapsto \sigma'$. If its transition is caused by a faulty action, then $f_{mm}(\sigma') \leq f_{mm}(\sigma) + 4\Delta$ holds.*

Proof: Let v be a process that executes a faulty action and w be any neighbor of v (i.e., $w \in N_v$). Focusing the link (v, w), there are three possible states of v: (α) $v \succ null$ (β) $v \succ w$ (γ) $v \succ x \land x \neq w$. The case (γ) is further classified into three sub-cases depending on x's state: $(\gamma 1)$ $x \succ null$ $(\gamma 2)$ $x \succ v$ $(\gamma 3)$ $x \succ x' \land x' \neq v$ (See Fig. 2. Note that each arrow means a pointer of v or x. The self-loop arrows mean null.) In the same way, possible states of w are classified to the five states (instead of x and x', we use y and y' for w). So the state of the link (v, w) is denoted by $s(v, w) = (\delta, \varepsilon)$ where where $\delta, \varepsilon \in \{\alpha, \beta, \gamma 1, \gamma 2, \gamma 3\}$. Now we consider $c(v, w)$ of all twenty five cases depending on $s(v, w)$.

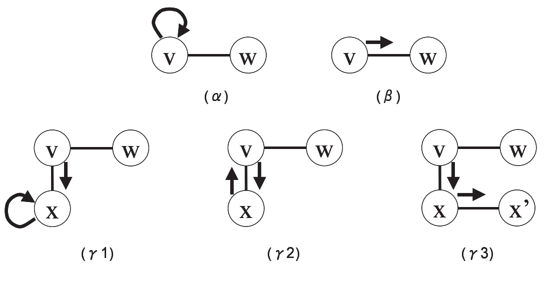

Figure 2. The states of v focusing on the link (v, w)

In the state of $s(v,w) = (\alpha,\alpha)$, $(v,w,R2)$ or $(w,v,R2)$ may occur. From Lemma 1, $c(v,w)$ is at most two. In the case of $s(v,w) = (\alpha,\beta)$, the next action of v is `R1`. So the normal action on (v,w) is $(v,w,R1)$ or $(w,v,R3)$, so $c(v,w)$ is at most one. In the same way, $c(v,w)$ is calculated for all the cases (in Table 1).

Table 1. $c(v,w)$ **for each state**

$w\backslash v$	(α)	(β)	$(\gamma 1)$	$(\gamma 2)$	$(\gamma 3)$
(α)	2	1	2	0	2
(β)	1	0	3	1	3
$(\gamma 1)$	2	3	2	0	2
$(\gamma 2)$	0	1	0	0	0
$(\gamma 3)$	2	3	2	0	2

When v executes a faulty action at σ, the state of v can change. The state change of v may increase the value of $c(v,w)$ for all the links connecting to v. Moreover, it may increase the value of $c(x,y)$ for all the links where x is a process pointing v at σ and y is a neighboring process of x. We observe these two case.

First, we observe the links connecting to v. Since the increase of each $c(v,w)$ is at most 3, the increase of each $c(v,w)$ is at most 3. However, the cases where $c(v,w)$ increases by 3 is only the following four cases (where σ and σ' be configurations before and after the faulty action respectively) (See Fig. 3. Note that process v and w are colored processes in the figure.):

(1) $s(v,w) = (\beta,\beta)$ at σ then $s(v,w) = (\gamma 1,\beta)$ at σ'.

(2) $s(v,w) = (\beta,\beta)$ at σ then $s(v,w) = (\gamma 3,\beta)$ at σ'.

(3) $s(v,w) = (\gamma 2,\gamma 1)$ at σ then $s(v,w) = (\beta,\gamma 1)$ at σ'.

(4) $s(v,w) = (\gamma 2,\gamma 3)$ at σ then $s(v,w) = (\beta,\gamma 3)$ at σ'.

The cases (1) and (2) require that $v \succ w$ holds at σ, so only one of the cases can occur for only one link (v,w). Similarly, the cases (3) and (4) require that $v \succ w$ holds at σ', so only one of the cases can occur for only one link (v,w). Thus the influence of links connecting to v is at most $2(\Delta-2)+2*3 = 2(\Delta+1)$.

Second, we observe links connecting to processes that point v at σ. Let x be a process pointing v at σ and y be a process adjacent to x. (See Fig. 4) The faulty action of v can change the state of x from γi to γj where $1 \le i \le 3$, $1 \le j \le 3$ and $i \neq j$. Table 1 shows that the increase of each $c(x,y)$ is at most 2 and it can occur at each link connecting to the matched link

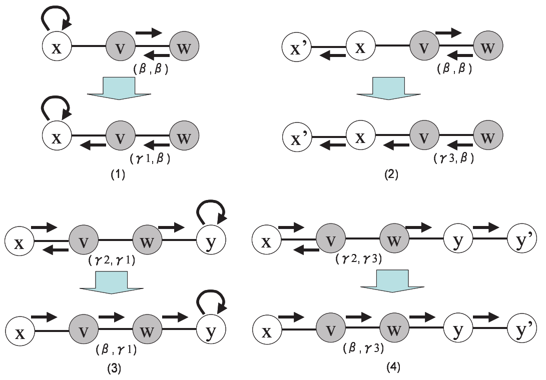

Figure 3. All 4 **cases** $c(v,w)$ **increases by** 3

(v,x) at σ with v. So the influence of links connecting to x except (v,x) is at most $2(\Delta-1)$.

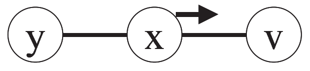

Figure 4. A topology of v, x, y

Consequently, $f_{mm}(\sigma') \le f_{mm}(\sigma)+2(\Delta+1)+2(\Delta-1) = f_{mm}(\sigma)+4\Delta$ holds. □

Lemmas 2 and 3 induce the following theorem.

Theorem 4 *The stabilization time with $\sharp F$ faults of the Hsu-Huang protocol is $2m+n+4\Delta\cdot\sharp F$.* □

From Theorem 4, readers may think of the **deferred penalty** [9] of transient faults: when huge number of transient faults are executed in the beginning part of an execution, the huge number of normal actions seem to be needed to reach a legitimate configuration. However, in the context of self-stabilization, we can escape the deferred penalty of transient faults by considering some configuration after the period of frequent occurrence of transient faults as an initial configuration. Consequently, the execution reaches a legitimate configuration if it has a part of the execution satisfying the condition of Theorem 4.

In what follows, we show an example of an execution such that the stabilization time with $\sharp F$ faults is $2m+n-5+2(\Delta+1)\cdot\sharp F$. This example implies that the stabilization time with $\sharp F$ faults is not so overestimated in Theorem 4: the estimation can be improved by at most $2\Delta\cdot\sharp F$.

A topology of the example is the same as the one presented in [5], the tree T_m of Fig. 5 having $m+1$ processes. Note that $\Delta = m-2$. The pointers in the initial configuration σ_0 are shown by the arrows in Fig 5. The self-loop arrow means null.

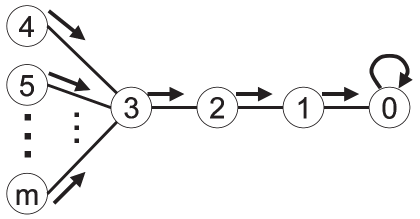

Figure 5. Tree T_m

In [5], they present the following schedule $\mathcal{Q}_s$, and show that the (usual) stabilization time is $2m+n-5$. Fig. 6 represent an execution from σ_0 by $\mathcal{Q}_s$.

$$\begin{aligned}&(4,3,R3),(5,3,R3),\ldots,\\&\quad(m,3,R3),(3,2,R3),(2,1,R3),\\&(4,3,R2),(5,3,R2),\ldots,\\&\quad(m,3,R2),(2,3,R2),(3,2,R1),\\&(4,3,R3),(5,3,R3),\ldots,\\&\quad(m,3,R3),(0,1,R1)\end{aligned}$$

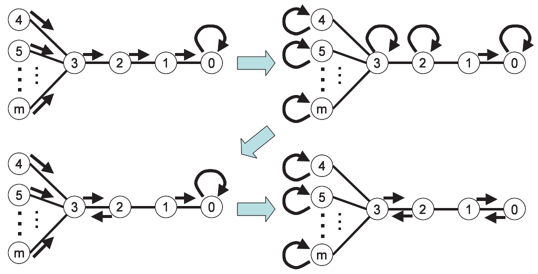

Figure 6. An execution from σ_0 by $\mathcal{Q}_s$

Now we consider the following partial schedule $\mathcal{Q}_f$. Note that $(2,*,F)$ denotes a faulty action that changes the pointer of process 2 from any to process 1. Fig. 7 represent an execution from σ_0 by $\mathcal{Q}_f$.

$$\begin{aligned}&(4,3,R3),(5,3,R3),\ldots,\\&\quad(m,3,R3),(3,2,R3),(2,1,R3),\\&(4,3,R2),(5,3,R2),\ldots,\\&\quad(m,3,R2),(2,3,R2),(3,2,R1),(2,*,F)\end{aligned}$$

Let $E[\sigma_0,\sigma_f]$ be the partial execution starting from σ_0 by schedule $\mathcal{Q}_f$. The last configuration σ_f is the same as σ_0. The number of normal actions included in $\mathcal{Q}_f$ is $2(m-1)=2(\Delta+1)$.

Let $\mathcal{Q}=\mathcal{Q}_1\mathcal{Q}_2\cdots\mathcal{Q}_{\sharp F+1}$ where $\mathcal{Q}_i=\mathcal{Q}_f$ for $1\le i\le\sharp F$ and $\mathcal{Q}_{\sharp F+1}=\mathcal{Q}_s$. The execution from σ_0 by

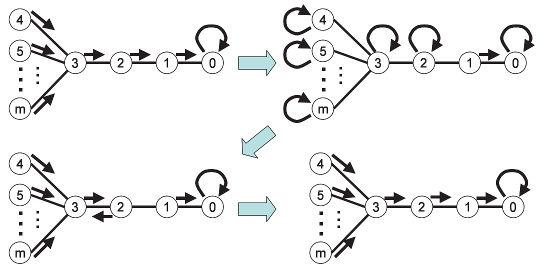

Figure 7. An execution from σ_0 by $\mathcal{Q}_s$

schedule $\mathcal{Q}$ requires $2m+n-5+2(\Delta+1)\cdot\sharp F$ normal actions.

We analyze the Hsu-Huang protocol with transient faults during convergence. We can analyze it with topological changes (i.e., process join and leave, link appearance and disappearance) during convergence, but we omit it because the topological changes affect the global parameters of systems (i.e., n, Δ, and so on) and the obtained complexity has a complicated formula.

5 Conclusion

We proposed a new performance measure, *the stabilization time with $\sharp F$ faults*, for self-stabilizing protocols. It is the worst case time a self-stabilizing protocol requires to converge from any configuration despite occurrence of $\sharp F$ faults during the convergence. We also proposed a new analysis method of the stabilization time with $\sharp F$ faults. The main idea of our method is that we regard each normal action as a forward step to the convergence, and each faulty action as a backward step. By quantifying influence of the forward step and the backward step, we can estimate the stabilization time with $\sharp F$ faults.

We applied the method to the self-stabilizing maximal matching protocol proposed by Hsu and Huang and showed that its stabilization time with $\sharp F$ faults is $2m+n+4\Delta\cdot\sharp F$. It is generalization of the (usual) stabilization time $2m+n$.

We can show that the stabilization time with $\sharp F$ faults of the Fast Coloring protocol proposed by Hedetniemi et al. [6] is $n+(\Delta+1)\cdot\sharp F$. But we omit it due to lack of space.

Both the Fast Coloring and the maximal matching are locally correctable problems: each process knows how to correct its state only from the states of its neighbors' and its own. And both are non-reactive protocols. So analysis of other protocols, especially locally in-correctable or reactive protocols, are one of our future work. It is worth noting that a self-stabilizing mutual exclusion protocol proposed in [1], one of the most investigated reactive protocols, is vulnerable to a transient fault during the convergence: a single transient fault can spoil almost all the efforts the protocol made before the fault [9].

In this paper, stabilization time with $\sharp F$ faults are estimated by the step complexity: total number of normal actions. We will extend it to other complexities (e.g., round complexity and so on).

References

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

[2] S. Dolev. *Self-stabilization*. The MIT Press, 2000.

[3] G. Ghosh, A. Gupta, T. Herman, and S. Pemmaraju. Fault-containing self-stabilizing algorithms. In *Proc. of the 15th ACM Conference on Principles of Distributed Computing (PODC)*, pages 45–54, 1996.

[4] G. Ghosh and X. He. Fault-containing self-stabilization using priority. *Information Processing Letters (IPL)*, 73(3-4):145–151, 2000.

[5] S. T. Hedetniemi, D. P. Jacobs, and P. K. Srimani. Maximal matching stabilizes in time $o(m)$. *Information Processing Letters (IPL)*, 80(5):221–223, 2001.

[6] S. T. Hedetniemi, D. P. Jacobs, and P. K. Srimani. Linear time self-stabilizing colorings. *Information Processing Letters (IPL)*, 87(5):251–255, 2003.

[7] S. C. Hsu and S. T. Huang. A self-stabilizing algorithm for maximal matching. *Information Processing Letters (IPL)*, 43(2):77–81, 1992.

[8] S. Kutten and B. Patt-Shamir. Time-adaptive self stabilization. In *Proc. of the 16th ACM Conference on Principles of Distributed Computing (PODC)*, pages 149–158, 1997.

[9] T. Masuzawa and H. Kakugawa. Self-stabilization in spite of frequent changes of networks: Case study of mutual exclusion on dynamic rings. In *Proc. of the 7th International Symposium of Self-Stabilizing Systems(SSS)*, pages 183–197, 2005.

[10] G. Tel. Maximal matching stabilizes in quadratic time. *Information Processing Letters (IPL)*, 49(6):271–272, 1994.

[11] H. Zhang and A. Arora. Continuous containment and local stabilization in path-vector routing. *Tech. Rep. of Ohio State University*, OSU-CISRC-5/04-TR027, 2004.