数理解析研究所講究録
第1554巻 2007年 269-275

k-Set Agreementを解く故障検知器

坂田 敦 (Atsushi Sakata) *　　小野 廣隆 (Hirotaka Ono) †
定兼 邦彦 (Kunihiko Sadakane) †　　山下 雅史 (Masafumi Yamashita) †

* 九州大学大学院システム情報科学府　† 九州大学大学院システム情報科学研究院
*†Depatment of Computer Science and Communication Engineering,
Graduate School of Information Science and Electrical Engineering, Kyushu University

1 はじめに

分散システムにおいて，プロセスの故障をいかに扱うかは，高信頼システム・アルゴリズム設計における重要な問題であり，実際分散システムの耐故障性を向上させるための研究が盛んに行われている．

分散システムの耐故障性を議論する際に注目されるのが非同期分散システムである．非同期分散システムにおいてはプロセスの故障を考慮すると多くの分散問題が決定性アルゴリズムでは解けないことが知られている [9]．

ここで重要なのは，[9] で定義されている非同期分散システムではプロセス間で全く同期をとれず，そのため各プロセスは他のプロセスの故障状況を知ることができないと仮定されていることである．しかし，実際のシステムではプロセスの処理速度や通信遅延に関する制約を用いることにより，故障したプロセスをある程度検知できる．これは上記の仮定では非可解とされた問題も，事実上可解となることがあることを意味する．このような故障プロセスに関する検知情報を抽象化し議論するために提案されたのが故障検知器である [4]．

故障検知器は各プロセスごとに備わったオラクルであり，各プロセスに対し他のプロセスの故障状況を知らせる．ただし，上述のような動機より，故障検知器は必ずしも正確な故障状況を知らせるわけではない．例えば，[4] では故障検知器をその性能によってクラス分けしている．どのクラスの故障検知器を用いれば問題が解けるかが分かれば，そのような故障検知器を実装することにより問題が解けることになる．そのため，対象となる分散問題を解くために必要かつ十分な「最弱」の故障検知器に関する考察が理論的，また実装的興味から盛んになされている ([1,3,7,8] など)．

分散システムにおける基本的な問題の1つに k-Set Agreement [5] がある．k-Set Agreement は次のような問題である：各プロセスは初期値として値を1つずつ持っており，それを他のプロセスに合意値として提案する．すべての正常プロセスは提案された値のうち1つだけ値を選んでそれを合意値として決定するが，各プロセスに決定される値は k 種類を超えてはならない．本研究ではこれを解く故障検知器の考察を行う．故障検知器を用いて k-Set Agreement を解く研究は多くなされている ([13], [10] など)．

本文の構成は以下の通りである．まず，2節において扱う分散システムのモデルを定義し，故障検知器の定義を述べる．次に3節では故障数が任意である環境において k-Set Agreement を解く故障検知器とそれを用いたアルゴリズムを提案する．また，関連研究で提案されている故障検知器を紹介し，それとの比較を行う．最後に4節でまとめと今後の課題について述べる．

2 準備

以下では取り扱う非同期分散システムのモデル，故障検知器の定義について述べる．

2.1 モデル

本節の定義の多くは [4] に基づく．

2.1.1 非同期分散システム

本研究で扱う非同期分散システムのモデルは以下の通りである．システムは n 個のプロセスの集合 $\Pi = \{p_1, p_2, \cdots, p_n\}$ で構成される．プロセスは計算を行うプロ

グラムをモデル化したものであり，その実行速度には下限が無いものとする．任意の2プロセス間に通信路が存在し，プロセス同士はこの通信路上でメッセージをやり取りし通信を行う．送られたメッセージはかならず届くが遅延時間に上限は無いとする．

本研究では1つのプロセスが全プロセスへメッセージを送信する動作を*broadcast*と表現し，これは以下の2つの性質を満たすものとする．(1) すべての正常プロセスは*broadcast*されたメッセージを必ず受け取る．ただし，正常プロセスに関しては2.1.2節で定義する．(2)*broadcast*されたメッセージを受け取るならばちょうど1つだけ受け取る．

説明のために大域時計 $\mathcal{T} = \{0, 1, 2, \cdots\}$ を考える．各プロセスはこれを参照できない．

2.1.2 故障

プロセスは停止故障を起こし，修復はされないものとする．故障パタン F は $\mathcal{T}$ から 2^{Π} への関数であり，$F(t)$ は時刻 t において故障しているプロセスの集合を表す．$crashed(F) = \bigcup_{t \in \mathcal{T}} F(t)$ で F において故障を起こすプロセスを表す．また，$correct(F) = \Pi - crashed(F)$ で F において故障しないプロセスを表し，$correct(F)$ に属すプロセスを正常プロセスという．ただし，$correct(F) \neq \emptyset$ とする．

2.2 故障検知器

2.2.1 定義

まず，[4] での故障検知器の定義を紹介する．

故障検知器履歴 H_D は $\Pi \times \mathcal{T}$ から 2^{Π} への関数で，$H_D(p,t)$ はプロセス p が時刻 t に疑っているプロセスの集合を表す．$p \neq q$ であるならば，$H_D(p,t) \neq H_D(q,t)$ であってもよい．

故障検知器 $\mathcal{D}$ は各故障パタン F から $\mathcal{D}(F)$ への関数である．$\mathcal{D}(F)$ は故障パタン F において故障検知器から出力され得るすべての故障検知器履歴の集合を表す．

故障検知器はその故障検知能力の度合いによりさまざまなクラス分けがされている．本稿では下の2つのクラスに属する故障検知器を扱う．どちらのクラスの故障検知器も [4] での定義とは異なり $H_D(p,t)$ は p が時刻 t に故障していないと思っている (以下,「信用している」と表現する) プロセスを表す．

さて，次の性質を満たす故障検知器 k-Ω を定義する．ただし，$H_{k-\Omega}(p,t)$ は1つのプロセスである．

$\exists t \in \mathcal{T}$, $\exists P \subseteq correct(F)$ $s.t.$ $|P| \leq k$, $\forall p \in correct(F)$, $\forall t' \geq t$, : $H_{k-\Omega}(p,t') \in P$.
(正常プロセスから成るサイズ k 以下の集合 P が存在し，いつかはすべての正常プロセスが P の中のある1つの正常プロセスをずっと信じるようになる．)

これは [3] で定義されたクラス Ω を拡張したものであり，1-Ω が Ω に相当する．

また，次の2つの性質を満たす故障検知器はクラス Σ [6] に属す．ただし，$H_\Sigma(p,t)$ は p が時刻 t に信用しているプロセスの集合を表す．

一様交差性 (Uniform Intersection): $\forall p, q \in \Pi$, $\forall t, t' \in \mathcal{T}$: $H_\Sigma(p,t) \cap H_\Sigma(q,t') \neq \phi$.
(任意の時刻，任意のプロセスのどの2つの検知器履歴においても，その積は空でない．)

完全性 (Completeness): $\exists t \in \mathcal{T}$, $\forall p \in correct(F)$, $\forall t' \geq t$: $H_\Sigma(p,t') \subseteq correct(F)$.
(いつかはすべての検知器履歴が永久に故障プロセスを含まなくなる．)

2.2.2 最弱の故障検知器

故障検知器の各クラス間には次のように，帰着による強弱関係が定義される．

あるクラス $\mathcal{D}$ の故障検知器をクラス $\mathcal{D}'$ の故障検知器に変換するアルゴリズムが存在するなら，かつその時に限り，$\mathcal{D}'$ は $\mathcal{D}$ より弱いと言い $\mathcal{D} \succeq \mathcal{D}'$ で表す．$\mathcal{D} \succeq \mathcal{D}'$ かつ $\mathcal{D}' \succeq \mathcal{D}$ である時，$\mathcal{D}$ と $\mathcal{D}'$ は同等であると言い，$\mathcal{D} \cong \mathcal{D}'$ で表す．に限り，$\mathcal{D}'$ は $\mathcal{D}$ より弱いと言い $\mathcal{D} \succeq \mathcal{D}'$ で表す．$\mathcal{D} \succeq \mathcal{D}'$ かつ $\mathcal{D}' \succeq \mathcal{D}$ である時，$\mathcal{D}$ と $\mathcal{D}'$ は同等であると言い，$\mathcal{D} \cong \mathcal{D}'$ で表す．言いかえれば，故障に関する情報をより多く与えてくれる故障検知器がより強いということになる．

この強弱関係の下では故障検知器は弱ければ弱いほど実装しやすいと考えられ，対象とする問題を解くために必要な最弱の故障検知器を知りたいという要求がある．また，最弱の故障検知器自体がその問題を解くのに必要な情報を規定するため，問題の複雑度解明の観点からも興味をもって研究されている．

ある故障検知器のクラス $\mathcal{D}$ がある問題 P を解く最弱の故障検知器であることを示すには次の**十分性** (sufficiency) と**必要性** (necessity) を証明すればよい．

十分性 クラス $\mathcal{D}$ の故障検知器を用いて問題 P を解ける．

必要性 問題 P を解くすべての故障検知器のクラスがクラス D 以上に強い．

2.3 k-Set Agreement

k-Set Agreement [5] は次の3つの性質により定義される．各プロセス p は初期値 v_p を持っているとする．

停止性 すべての正常プロセス p がいつかは値 d_p に決定する．

合意性 決定値は k 種類以下．つまり，$|\bigcup_{p\in\Pi} d_p| \leq k$

妥当性 プロセス p が値 d_p に決定したならば d_p はいずれかのプロセスの初期値である．

定理 1 [2, 11, 15] 故障数が k 個以上の環境下では k-Set Agreement は解けない． □

3 k-Set Agreement を解く故障検知器

ここではプロセスの故障数が任意という環境においてクラス k-Ω とクラス Σ の両方の出力を持つようなクラス $(k$-$\Omega, \Sigma)$ の故障検知器を用いて k-Set Agreement を解くアルゴリズムを提案する．

この環境下においてはクラス $(1$-$\Omega, \Sigma)$ が 1-Set Agreement を解く最弱の故障検知器であることが知られている [6]．

3.1 アルゴリズム

図1に示すアルゴリズム1を用いる．概要は以下に記す．

アルゴリズムは [12] のアイデアを用いる．各プロセスは次のようなラウンドを繰り返す．各ラウンドでは k 個のプロセスが調整者 (coordinator) と呼ばれる役割を与えられる．各プロセスは各ラウンド r における調整者の集合 C_r を事前に知っているとする．$|C_r| = k$ である．まず，各調整者が全プロセスへ自分の保持値 v_c を提案する．各プロセス $p \in \Pi$ はいずれかの調整者から値 v を受け取るのを待つ．この時，(1) もし値を受け取ればその値を放送する．(2) 値を1つも受け取っていないうちに $H_{k-\Omega}(p,t)$ として信じているプロセスが C_r に含まれないプロセスになったら $\perp$ を放送する．次に各プロセス p は $H_\Sigma(p,t)$ に属す全てのプロセスから v か $\perp$ を受け取るまで待つ．$H_\Sigma(p,t)$ の値は時刻により変化するので，各プロセスが返事を待つ相手も時刻により変化する．この時，集めた値の集合を Rec_p として，(1)Rec_p が $\perp$ 以外の値で満たされていればそのうちの一番小さい値に決定して終了．(2)Rec_p に $\perp$ と値が混ざっていれば一番小さい値を自分の保持値 v_p にセットし次のラウンドへ．(3)Rec_p が $\perp$ で満たされていれば何もせずに次のラウンドへ．

次のラウンドでは前のラウンドとは少なくとも1つは違うプロセスが調整者となる．${}_nC_k$ 通りの全ての組み合わせのプロセスが調整者を務めたら最初の組み合わせに戻る．ラウンドを進めているうちに他のプロセスから $DECISION(v)$ が届くことがあるが，このときには v に決定して停止する．

3.2 正当性

定理 2 故障数任意の環境下でクラス $(k$-$\Omega, \Sigma)$ の故障検知器を用いればアルゴリズム *1* で *k-Set Agreement* が解ける．

証明.

・妥当性

明らかに満たす．

・合意性

case において一度 $broadcast(DECISION(v))$ が起こるとそれ以降決定候補値となる値は k 種類以下となる．理由は以下の通り．$broadcast(DECISION(v))$ が起こったということはそのラウンドを r とすると，ラウンド r の **case** において (1) を満たしたプロセスが存在することになる．この時，Σ の一様交差性より，ラウンド r の **case** で (3) を満たしたプロセスは存在しないことになる．このラウンドにおいて各プロセスの Rec_p の中には値は k 種類しか無く，決定せずに次のラウンドへ進んだプロセスはラウンド r の **case** では (1) を満たし，自身の値をこのラウンド r でやりとりされた k 種類の値のうちの1つに書き換えているはずである．よって，これ以降のラウンドでやりとりされる値はラウンド r でやりとりされたせいぜい k 種類の値となる．以上より，合意性は満たされる．

・停止性

ブロックしてしまう可能性があるのは 06 と 09 だけであるので，どのラウンドにおいても 06 と 09 で待ち続けるプロセスは存在しないことを示す．まず，06 において待ち続けるプロセス p が存在するとする．このとき，p が 06 に到達した時刻を t とすると，任意の時刻 $t' \geq t$，において $H_{k-\Omega}(p,t') \in C_r$ となっている．もしそうでなければ p は 06 を抜ける．今，C_r に含まれるいずれかのプロセス c が $broadcast(P1[r_c, v_c])$ を行ったならばいつかは p にそれが届き 06 を抜けるはずなので，k 個のどの調整者も $broadcast(P1[r_c, v_c])$ を行う前に故障したことになる．このとき，k-Ω の性質より，いつかは $H_{k-\Omega}(p,t^*) \notin C_r$ となる

全プロセス p は次の 2 つのタスクを並行して実行.
01.$r_p \leftarrow 0$;
02.$v_p \leftarrow p$ の初期値;

03.**parallel task 1**
——————Phase 1——————
04. **while** *true* **do**
05. **if** (p は調整者) **then** $broadcast(P1[r_p, v_p])$; **end-if**
06. **wait until** (いずれかの調整者から $P1[r, v]$ が届く) $\vee(H_{k-\Omega}(p,t) \notin C_r$ になる);
07. **if** (06 で $P1[r,v]$ を受信) **0then** $d_p \leftarrow v$; **else** $d_p \leftarrow \perp 0$; **end-if**
——————Phase 2——————
08. $broadcast(P2[r_p, d_p])$;
09. **wait until** (すべての $q \in H_\Sigma(p,t)$ から $P2[r_q, d_q]$ を受信. 待っている途中で $H_\Sigma(p,t)$ が $H_\Sigma(p,t') \neq H_\Sigma(p,t)$ に変化したら $H_\Sigma(p,t')$ に含まれるプロセスからの $P2[r_q, d_q]$ を受信するまで待つ;)
10. $Rec_p \leftarrow$09 で受け取った d の集合;
11. **case** (1)$Rec_p \not\ni \perp$ **then** Rec_p に含まれる最も小さい値を d とし, $broadcast(DECISION(d))$; d に決定して停止;
12. (2)$Rec_p = \{d_1, d_2, \ldots, d_x (x \leq k), \perp\}$ **then** $v_p \leftarrow d_1$;
13. (3)$Rec_p = \{\perp\}$ **then skip**
14. **endcase**
15. **end-while**

16.**parallel task 2**
17. **upon** ($DECISION(d)$ を受信) **do**
18. $broadcast(DECISION(d))$; d に決定して停止;
19. **end-upon**

図 1: アルゴリズム 1

はずなので, 矛盾. よって 06 で待ち続けるプロセスは存在しない.

09 においては Σ の完全性より, いつかは $H_\Sigma(p,t)$ が正常プロセスのみを含み続けるようになるので, その $H_\Sigma(p,t)$ に含まれるすべてのプロセス q から必ず $P2[r_q, d_q]$ が届く. よって, 09 で待ち続けることも無い. 以上より, どのプロセスも決定が起きない限りラウンド数を増やし続ける.

次に, k-Ω と Σ の出力がどちらも定義中「いつかは」で保証されている性質を満たし始める時刻 t を考える. k-Ω の定義中の P を用いると, t 以降に $P \subseteq C$ であるような C が調整者となるようなラウンド r がある. k-Ω の性質より, P に含まれるプロセスはすべて正常プロセスであるのでラウンド r の phase1 の 06 においてはどのプロセスも必ずいずれかの調整者から値を受け取ることになる. よって, ラウンド r の Phase2 の 08 において $\perp$ を投じるプロセスは存在しない. このときどのプロセスにおいても 10 では $\perp$ ではない値のみが揃い, **case** の (1) が満たされ決定が起きて全プロセスが停止する. □

故障数が半数未満の 2.1.1 のシステムでは, Σ を実装可能 [6] なことから下の系が導かれる.

系 1 故障数半数未満の環境下でクラス k-Ω の故障検知器を用いれば *k-Set Agreement* が解ける.

3.3 関連クラス Ω^k との比較

[14] において以下の性質を満たすクラス Ω^k の故障検知器で k-Set Agreement を解くということが考えられている.

1.$\forall t \in \mathcal{T}, \forall p \in \Pi$: $|H_{\Omega^k}(p,t)| \leq k$.

2.$\exists t \in \mathcal{T}$, $\exists Q \subseteq \Pi$(*s.t.* $Q \cap correct(F) \neq \emptyset$), $\forall p \in correct(F)$, $\forall t' \geq t$, : $H_{\Omega^k}(p,t') = Q$.

(どのプロセスも常に k 個以下のプロセスを信用している. かつ, 正常プロセスを少なくとも 1 つは含む集合 Q が存在し, いつかはすべてのプロセスの検知器履歴がずっと Q になる.)

[14] において用いられている分散システムはアトミックレジスタを備えたメモリ共有システムであり, このシステムはクラス Σ を備えた 2.1.1 のシステムと同等であることが示されている. [6]

定理 3 [14] アトミックレジスタを備えたメモリ共有システムにおいてクラス Ω^k の故障検知器を用いれば故障数任意の環境下で *k-Set Agreement* が解ける.

[14] の著者らはこのクラス Ω^k が k-Set Agreement を解く最弱の故障検知器ではないかと予想しており，本研究で提案するクラス k-Ω との強弱関係を比較することにより，どちらかが，もしくはそのどちらもが k-Set Agreement を解く最弱の故障検知器となる可能性を有するのかを検討する．

3.3.1 k-$\Omega \preceq \Omega^k$

図 2 に示すアルゴリズム 2 を用いれば，$f < n$ の環境下においてクラス Ω^k の故障検知器を k-Ω の故障検知器に変換することができる．アルゴリズムの概要は以下の通り．

各プロセス $p \in \Pi$ は自分が生存を信じているプロセス全てにメッセージを送る．メッセージを受け取ったプロセスはその送り主 p に返事を出す．p は最後に届いた返事を送ってきたプロセス q を出力する．これを無限回繰り返す．

アルゴリズム 2 は入力として故障検知器 $\mathcal{D}$ を取る．入力として Ω^k を与えると下の 2 つの補題が成り立つ．

補題 1 ある時刻 t_1 が存在し，任意の $t \geq t_1$ において $|\cup_{p \in correct(F)} output(p,t)| \leq k$ となる．

証明. Ω^k の性質より，ある時刻 t' が存在し，t' 以降すべてのプロセスの検知器履歴はサイズ k 以下の集合 Q に一致している．t' 以前に送られた全ての *message* に対する *reply* がすべて届き終わった時刻を t^* とする．この時，t^* 以降に送られるすべての *message* は Q に含まれるプロセスに対してのみ送られる．Q には正常プロセスが含まれるので，*reply* を返し続けるプロセスが存在する．$|Q| \geq k$ であることから，その *reply* により出力されるプロセスはせいぜい k 種類． □

補題 2 ある時刻 t_2 が存在し，任意の $t \geq t_2$，任意の $p \in correct(F)$ において $output(p,t) \in correct(F)$ となる．

証明. 故障するプロセスがすべて故障し終わる時刻を t' とする．t' 以降に送られた *message* に対する *reply* を返すプロセスはすべて正常プロセスである．よっていつかはどのプロセスも永久に正常プロセスだけを出力し続けるようになる． □

定理 4 故障数任意の環境下において，k-$\Omega \preceq \Omega^k$．

証明. 補題 1 と補題 2 における t_1 と t_2 をそのまま用いて $t = \max\{t_1, t_2\}$ とし，
$\cup_{p \in correct(F), t' \geq t} output(p,t') = S$ とする．補題 1 と補題 2 より，この S は k-Ω の定義中の P の性質を満たす．よって，アルゴリズム 2 の *output* は k-Ω の性質を満たす． □

各プロセス p は次の 3 つのタスクを並行して実行.

```
01.parallel task1
02.   periodically do
03.     すべての q ∈ H_D(p,t) に message を送信;
04.   end-do

05.parallel task2
06.   upon (q から message を受信) do
07.     reply を q に送信;
08.   end-upon

09.parallel task3
10.   upon (q から reply を受信) do
11.     output(p,t) ← q;
12.   end-upon
```

図 2: アルゴリズム 2

3.3.2 k-$\Omega \succeq \Omega^k$

また，逆の変換も同様に可能である事が示せる．

図 3 に示すアルゴリズム 3 を用いれば，$f < n$ の環境下においてクラス k-Ω の故障検知器を Ω^k の故障検知器に変換することができる．アルゴリズムの概要は以下の通り．

各プロセス $p \in \Pi$ は自らの局所時計が Δ 経過するごとに自分が信じているプロセスの *id* を放送する．ただし，Δ は 0 でない任意の有限時間．各プロセスは各 *id* が何度届いたかを記憶しておき，届いた回数の多い方から k 個のプロセスの集合を出力しておく．

アルゴリズム 3 は入力として故障検知器 $\mathcal{D}$ を取る．アルゴリズム 3 の入力として k-Ω を与えると，以下の補題が成り立つ．

補題 3 ある時刻 t_1 が存在し，すべての $p \in correct(F)$ において任意の時刻 $t \geq t_1$ で $output(p,t) \cap correct(F) \neq \emptyset$ となる．

証明. k-Ω の性質より，ある時刻 t' が存在し，任意の $t \geq t'$ ではすべてのプロセス $p \in correct(F)$ において $H_{k-\Omega}(p,t)$

各プロセス p は以下の 3 つのタスクを並行して実行. 局所記憶として $NUM_p[1...n]$ を持つ.

01.**parallel task 1**
02. **while** *true* **do**
03. 局所時計が Δ 経つ毎に $H_D(p,t)$ を *broadcast*.
04. **end-while**

05.**parallel task 2**
06. **while** *true* **do**
07. **if** ($H_D(q,t) = i$ を受け取る) **do**
08. $NUM_p[i] = NUM_p[i] + 1$;
09. **end-if**
10. **end-while**

11.**parallel task 3**
12. **while** *true* **do**
13. $NUM_p[i]$ の大きい方から k 個のプロセスの集合を $output(p,t)$ とする. ただし, k 番目に NUM の値が大きいプロセスが複数存在する場合は id の小さいものを選ぶ.
14. **end-while**

図 3: アルゴリズム 3

の値はずっと k-Ω の定義中の P に含まれるプロセスの id になる. よって, すべての正常プロセスがアルゴリズムの 03 において P に含まれるプロセスの id だけを放送し続けるようになる. このことから, いつかは P 中の全プロセスが NUM の値上位 k 位に入り, その後 k 位未満に落ちることは無い. P は正常プロセスだけを含む集合であるから, 補題は示された. □

補題 4 ある時刻 t_2 が存在し, すべての $p \in correct(F)$ において任意の $t \geq t_2$ で $output(p,t) = output(p,t_2)$ となる.

証明. 補題 3 より, 全ての正常プロセスの *output* が k-Ω 定義中の集合 P をずっと含むようになるような時刻 t_1 が存在する. このことから $|P| = k$ ならば補題は明らかに成り立つので, $|P| \leq k$ とする.

t_1 以前に送信された全てのメッセージが受信され終わる時刻を t' とする. t' において任意のプロセス $q \in (\Pi \setminus P)$, 任意の正常プロセス i,j について $NUM_i[q] = NUM_j[q]$. これは放送の性質より, 放送が行われればすべての正常プロセスが同じメッセージをちょうど 1 つずつ受け取るため. t' 以降は P に含まれるプロセスの NUM の値だけが増えるので, すべての正常プロセス i の $NUM_i[q]$ の値は変化しない. したがって, すべての i において $output(i,t)$ の値は変化しなくなる. このことから, 時刻 t_2 の存在が示された. □

補題 5 ある時刻 t_3 が存在し, 任意の $t \geq t_3$ ではすべての $p,q \in correct(F)$ において $output(p,t) = output(q,t)$ となる.

証明. 今, 補題 3 と補題 4 の t_1, t_2 を用いて, $t^* = \max\{t_1, t_2\}$ を考える. すると, 任意の $t \geq t^*$ ではすべての $p \in correct(F)$ において $output(p,t) = output(p,t^*)$ かつ $P \subseteq output(p,t)$. ただし, P は k-Ω の定義中の P の性質を満たす. ここで, 補題 4 の証明の中で用いた t' を用いる. $t^+ = \max\{t^*, t'\}$ とすると, t^+ がこの補題の保証する時刻 t_3 の性質を満たす. □

定理 5 故障数任意の環境下において, k-$\Omega \succeq \Omega^k$.

証明. 補題 3〜5 より, k-Ω を入力としたアルゴリズム 3 の *output* が Ω^k の定義中の Q の性質を満たすことが分かり, このことから示される. □

定理 4 と定理 5 から次の系が導かれる.

系 2 故障数任意の環境下において, k-$\Omega \cong \Omega^k$. □

このことから, k-Set Agreement を解く最弱の故障検知器は k-Ω と Ω^k の両方, もしくはどちらもそうでないかのどちらかであることが分かる.

4 まとめと今後の課題

クラス k-Ω を提案し, 故障数任意の環境下においてクラス $(k\text{-}\Omega, \Sigma)$ の故障検知器を用いれば k-Set Agreement が解けることを示した. また, 関連研究で提案されているクラス Ω^k と k-Ω の比較を行い, 両者が同等の強さを持つことを示した.

今後の課題として, Ω^k と k-Ω のどちらかの必要性を示すことにより k-Set Agreement を解く上での両故障検知器の最弱性を示したいと考えている.

参考文献

[1] M. Aguilera, S. Toueg, B. Deianov, Revisiting the weakest failure detector for uniform reliable broadcast, 13th International Symposium on Distributed Computing, September 1999.

[2] E. Borowsky and E. Gafni, Generalized FLP Impossibility Results for t-Resilient Asynchronous Computations, Proc. 25th ACM Symposium on Theory of Computation(STOC'93), San Diego(CA), pp. 91-100, 1993.

[3] T. D. Chandra, V. Hadzilacos and S. Toueg, The weakest failure detector for solving consensus, J. ACM, 43(4):685-722, 1996.

[4] T. D. Chandra and S. Toueg, Unreliable Failure Detectors for Reliable Distributed Systems, J. ACM, 43(2):225-267, 1996.

[5] S. Chaudhuri, Agreement Is Harder Than Consensus:Set Consensus Problem in Totally Asynchronous Systems, Proc.9th ACM Symposium on Principles of Distributed Computing,Quebec(Canada), 311-324, 1990.

[6] C. Delporte-Gallet, H. Fauconnier and R. Guerraoui, Shared Memory vs Message Passing, Tech. Rep. IC/2003/77, EPFL. December.

[7] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, V. Hadzilacos, P. Kouznetsov, S. Toueg, The Weakest Failure Detectors to Solve Certain Fundamental Problems in Distributed Computing, In PODC'04, 338-346, 2004.

[8] J. Eisler, V. Hadzilacos, S. Toueg, The Weakest Failure Detector to Solve Nonuniform Consensus, In PODC'05, 189-196,2005.

[9] M. J. Fischer, N. A. Lynch and M. S. Paterson, Impossibility of distributed consensus with one faulty process, J. ACM, 32(2):374-382, 1985.

[10] M. P. Herlihy and L. D. Penso, Tight bounds for k-Set Agreement with Limited Scope Accuracy Failure Detectors, Distributed Computing, 18(2): 157-166, 2005.

[11] M. Herlihy and N. Shavit, The Asynchronous Computability Theorem for t-Resilient Tasks, Proc. 25th ACM Symposium on Theory of Computation, California (USA), pp. 111-120, 1993.

[12] A. Mostefaoui and M. Raynal, Solving Consensus Using Chandra-Toueg's Unreliable Failure Detectors:a General Quorum-Based Approach, Proc. 13th Symposium on Distributed Computing (DISC'99), Springer Verlag LNCS #1693, pp. 49-63, Bratislava (Slovakia), 1999.

[13] A. Mostefaoui and S. Raynal, k-Set Agreement with Limited Accuracy Failure Detectors, Proc. 19th ACM Symposium on Principles of Distributed Computing (PODC'00), ACM Press, pp. 143-152, 2000.

[14] A. Mostefaoui, M. Raynal and C. Travers, Exploring Gafni's reduction land: from Ω^k to wait-free adaptive (2p - $\lceil \frac{p}{k} \rceil$)-renaming via k-set Agreement, Proc. 20th Symposium on Distributed Computing (DISC'06), Springer Verlag LNCS #4167, pp. 1-15, Stockholm(Sweden), 2006.

[15] M. Saks and F. Zaharoglou, Wait-free k-Set Agreement is Impossible: the Topology of Public Knowledge, SIAM Journal on Computing,29(5):1449-1483,2000.