分散アルゴリズム輪講資料
14 章 : Fault Tolerance in Asynchronous Systems

米澤研究室修士 2 年 小林 義徳

2002 年 5 月 15 日

1 復習

Theorem 2.19 γ を非同期通信分散システムにおける configuration ,e_p と e_q を互いに異なるプロセス p, q に対する event で、いずれも γ に適用可能であるとする。このとき、 e_p は $e_q(\gamma)$ に適用可能、 e_q は $e_p(\gamma)$ に適用可能で、 $e_p(e_q(\gamma)) = e_q(e_p(\gamma))$ が成立する。

2 Impossibility of Consensus

2.1 Notation,Definitions,Elementary Results

イベント列 $\sigma = (e_1, e_2, \cdots, e_k)$ が configuration γ に適用可能であり、その結果の configuration が δ のとき、 $\gamma \rightsquigarrow^{\sigma} \delta$ または、 $\sigma(\gamma) = \delta)$ と書く。また、 $S \subseteq P$ であり、 σ が S に含まれるプロセスのみからなるとき、 $\gamma \rightsquigarrow {}_S\delta$ とも書く。

Proposition 2.1 *configuration* γ の下で、 *sequences* σ_1,σ_2 が適用可能であるとし、 σ_1, σ_2 の両方に参加するプロセスは無いとする。このとき、以下のことが成り立つ。

- σ_2 は $\sigma_1(\gamma)$ に適用可能
- σ_1 は $\sigma_2(\gamma)$ に適用可能
- $\sigma_2(\sigma_1(\gamma)) = \sigma_1(\sigma_2(\gamma))$

(証明) Theorem 2.19 (p.55) を繰り返し適用することによって得られる。 □

以下、各プロセス p は、読み取り専用の input variable x_p と、 write-once output register y_p(初期値 b) を持つものとする。プロセス p は、最終的に 0 か 1 かに「決定」し、決定 (decision value) を y_p に書き込む。 y_p の初期値 b は、 decision value ではないとする。

Definition 2.2 *t-crash fair execution* とは、以下の条件を満たす *execution* のことである。

- 少なくとも $N - t$ のプロセスが、無限個の *event* を実行する。
- 正しいプロセス *(a correct process)* に送られたメッセージは、必ず受け取られる。

ここで、正しいプロセスとは、無限個の *event* を実行するプロセスのことである。

以降、あるアルゴリズムによって取り扱える faulty process の数を t と書き、そのアルゴリズムの resilience と呼ぶ。

Definition 2.3 *1-crash-robust consensus algorith* とは、以下の 3 つを満たすアルゴリズムである。

1. ***Termination.*** どの *1-crash fair execution* も、すべての正しいプロセスは決定する。
2. ***Agreement.*** 到達可能な *configuration* で、 $y_p \neq b$ かつ $y_q \neq b$ で、 p, q が正しいプロセスの時、 $y_p = y_q$ である。
3. ***Non-triviality.*** $v = 0$ のとき、 $v = 1$ のとき、ある p に対して y_p であるような、到達可能な *configuration* が存在する。

用語 (p.438 下の方)

- **configuration が v-decided.** ある p に対し、 $y_p = v$
- **configuration が decided.** configuration が 0-decided または 1-decided
- **configuration が v-valent.** その configuration から到達可能な decided configuration が v-decided
- **configuration が bivalent.** 0-decided と 1-decided の両方がその configuration から到達可能
- **configuration が univalent.** 1-valent または 0-valent

Proposition 2.4 *t-robust* なアルゴリズムのどのような到達可能な *configuration* および プロセスのどのような部分集合 S *(少なくとも $N - t$)* に対しても、 $\gamma \rightsquigarrow_S \delta$ となるような *decided configuration* δ が存在する。

(証明) γ, S を上のように仮定。 γ に到達する execution で、 S 内のプロセスからなる無限に多くの event の列 を含むものを考える。この execution は t-crash fair であり、 S に含まれるプロセスは正しいプロセスである。したがって、決定に到達する。 □

Lemma 2.5 到達可能な *fork* は存在しない。

(fork とは?)

多くとも t 個のプロセスからなる集合 T が存在し、 configuration γ_0, γ_1 に対し、 $\gamma \rightsquigarrow {}_T\gamma_0$,$\gamma \rightsquigarrow {}_T\gamma_1$ である。ここに、 γ_v は v-valent であるとする。

t-robust protocol の、以上のような configuration γ を fork と呼ぶ。

簡単に言うと、 t 個のプロセスの部分集合が 0-decision も 1-decision も実行しうるとき、 γ を fork と呼ぶ。

(証明)

γ を到達可能な configuration, T を多くとも t プロセスの部分集合とする。 S を、 T の補集合、すなわち、 $S = P \backslash T$ とする。

S は少なくとも $N - t$ のプロセスをもち、したがって、 $\gamma \rightsquigarrow {}_S\delta$ となるような decided configuration δ が存在する (Proposition 2.4)。 configuration δ は、 0- または 1-decided であり、一般性を失わずに、 0-decided の場合を考える。

ここで、 1-valent な γ' に対し、 $\gamma \rightsquigarrow {}_T\gamma'$ が成立しないことが示される。

まず、 γ' を $\gamma \rightsquigarrow {}_T\gamma'$ なる 任意の configuration とする。 T と S のステップを交換することにより、 δ と γ' の両方から到達可能な configuration δ' が得られる。 δ が 0-decided なので、 δ' も 0-decided である。したがって、 γ' は 1-valent ではない。

2.2 The Impossibility Proof

この章では、 1-crash-robust な consensus algorithm が不可能であることを示す。 bivalent initial configuration の存在 (Lemma 2.6) → すべてのステップは univalent な configuration に移らずに実行できる (Lemma 2.7)→ 1-crash-robust な consensus algorithm が不可能 (Throrem **??**) という流れである。

以下、 1-crash-robust consensus algorithm を A とする。

Lemma 2.6 A に対し、 *bivalent* な *initial configuration* が存在する。

(証明)

A は non-trivial であるから (Definition 2.3)、到達可能な 0-decided および 1-decided configuration が存在する。 δ_0, δ_1 を initial configurations とし、 δ_v から v-decided configuration へ到達可能であるとする。

$\delta_0 = \delta_1$ であるとき、 initial configuration は bivalent である。そうでないとき、 $gamma_0, \gamma_1$ を initial configurations とし、 v-decided configuration が γ_v から到達可能であるとする。また、 γ_0 と γ_1 は一つのプロセスの入力のみが異なるとする。

ここで、 δ_0 から始まり、 δ_1 で終わる initial configuration の列を考える。

$$\delta_0 \cdots \gamma_0\gamma_1 \cdots \delta_1$$

この列の中では、次の initial configuration は、前の initial configuration と 1 プロセスの入力のみが異なる。 γ_0 と γ_1 は、プロセス p のみが異なるとする。

ここで、 p が 1 ステップも動かない、 fair な execution を考える。 A は 1-crash-robust なので、 γ_0 は decided configurationβ に到達する。

1. β が 1-decided のとき、 γ_0 は bivalent
2. β が 0-decided のとき、 p が動かないので、 γ_1 から β へ到達可能。従って、 γ_1 が bivalent

□

Lemma 2.7 γ を到達可能で *bivalent* な *configuration* とし、 s を、 γ 内の プロセス p に適用可能なステップとする。このとき、あるイベント列 σ が存在し、 s が $\sigma(\gamma)$ に適用可能で、 $s(\sigma(\gamma))$ が *bivalent* である。

(証明)

C を、 γ から s を適用せずに到達可能な configuration の集合とする。すなわち、 $C = \{\sigma(\gamma) : s \text{ does not occur in } \sigma\}$。 s は、 C 内のどの configuration にも適用可能である。

このとき、 v-decided configuration が $s(\alpha_v)$ から到達可能であるような configurations $\alpha_1, alpha_2 \in C$ が存在する。このことは、以下によって示される。 $v = 0, 1$ に対し、 γ から到達可能な v-decided configuration を β_v とする。 $\beta_v \in C$ のとき、すなわち、 decided configuration にたどり着くのに s が適用されない場合、 $s(\beta_v)$ も v-decided であるので、 $\alpha_v = \beta_v$ と選べば良い。 $\beta_v \notin C$ のとき、すなわち、 decided configuration にたどり着くのに s が適用される場合、 α_v として、 s が適用された configuration を選ぶ。

$\alpha_0 = \alpha_1$ であるとき、 $s(\alpha_0)$ は 求める bivalent configuration である。 $\alpha_0 \neq \alpha_1$ であるとき、 γ から α_0 および α_1 への経路を考える。それらの経路の中で、ある configuration が、別の configuration から 1 step で得られるとき、それらの configurations を neighbors と呼ぶ。 0-decided configuration が $s(\alpha_0)$ から得られ、 1-decided configuration が $s(\alpha_1)$ から得られるので、以下の二つのケースが考えられる。

1. $s(\gamma')$ が bivalent であるような γ' が、経路上に存在する。
2. $s(\gamma_0)$ が 0-valent で、 $s(\gamma_1)$ が 1-valent であるような neighbors γ_0, γ_1 が存在する。

最初のケースでは、 $s(\gamma')$ が求める bivalent configuration なので、 OK. 二番目のケースでは、 γ_0, γ_1 のどちらかが fork となり、矛盾。一般性を失わずに、 $\gamma_1 = e(\gamma_0)$ とできる。すると、 $s(e(\gamma_0)) = s(\gamma_1)$ なので、 1-valent。しかし、すでに $s(\gamma_0)$ は 0-valent なので、 $e(s(\gamma_0))$ は 1-valent ではない。 e と s は可換ではないので、 Theorem 2.19(p.55) より、 $p = q$。しかし、このとき、 reachable configuration γ_0 について、 $\gamma_0 \rightsquigarrow {}_p s(\gamma_0)$ と $\gamma_0 \rightsquigarrow {}_p s(e(\gamma_0))$ が成立してしまう。前者は 0-valent で、後者は 1-valent なので、 γ_0 が到達可能な fork になってしまい、矛盾。 □

Theorem 2.8 *asynchronous,deterministic,1-crash-robust consensus algorithm* は存在しない。

(証明)

このようなアルゴリズムが存在するとすると、 bivalent な initial configuration γ_0 から始まる fair な execution が構築できてしまう。 configuration γ_i まで構築ができたとき、 s_i を、一番長い間適用可能なステップに選ぶ。

Lemma2.7より、 s_i が実行出来るまでに execution が拡張され、 bivalent configuration γ_{i+1} にたどり着く。このような作り方をすると、全てのプロセスが正しいが、永遠に結論が出ない fair execution が作れる。 □

2.3 Discussion

以上、 asynchronous,deterministic,1-crash-robust consensus algorithm は存在しないことが示された。

しかし、少し条件を弱めれば、問題が解けるようになる。

1. **Weaker fault model** Section 14.2 fault を、 initially dead process のみに制限。すると、 consensus , election が可能になる。
2. **Weaker coordination** Section 14.3 よりプロセス間の連携が弱い問題で、 deterministic に解ける問題がある (例： Renaming)。
3. **Randomization** Section 14.4 Byzantine failure があっても OK.
4. **Weak Termination** Section 14.5 Byzantine failure があっても OK.
5. **Synchrony** 15 章

3 Initially Dead Processes

Initially Dead Model では、各プロセスは、無限個のイベントを実行するか、一つも実行しないかのどちらかである。

Definition 3.1 *t-initially-dead fair execution* とは、以下の条件を満たす *execution* である。

- 少なくとも $N - t$ のプロセスが active
- active なプロセスは無限に多くのイベントを実行。
- 正しいプロセスに送られたメッセージは必ず受け取られる。

t-initially-dead-rebust consensus algorithm では、全ての正しいプロセスは、 t-initially-dead-fair execution で決定する。 Agreement と non-triviality については、 crash model の場合と同じ。

プロセスは、メッセージを送った後に落ちることは無いので、プロセス p がすでにメッセージを発信したことがわかった場合、 p からのメッセージの受信を待つのは安全。以下に、 consensus と election が、 $t < \frac{N}{2}$ の場合に解けることが示される。

アルゴリズム 14.1 を参照のこと。

4 Deterministically Achievable Cases

中には、 crash model であっても、 deterministic に解ける問題もある。

例: renaming 各プロセスの ID を、限られた範囲の ID につけ変えるアルゴリズム。

decision vectorタスク T は、ありうる input X と、 output D によって、次のような Mapping として表される。

$$T : X^N \to \mathcal{P}(D^N)$$

ここに、 vector $\vec{x} = (x_1, x_2, \cdots, x_N)(\in X^N)$ は input を表し、 $T(\vec{x})$ は、アルゴリズムの正しい output を表す。これは、 decision vector $\vec{d} = (d_1, d_2, \cdots, d_N)(\in D^N)$ として記述される。

Definition 4.1 アルゴリズムがタスク T についての *t-crash-robust solution* であるとは、次の条件を満たすことをいう。

1. **Termination** すべての正しいプロセスは決定する。
2. **Consistency** もし全てのプロセスが正しいならば、 decision vector $\vec{d}$ は、 $T(\vec{x})$ に含まれる。

consistency の条件は、もし、プロセスの部分集合が決定するとき、その vector を $T(\vec{x})$ に必ず拡張されることも表している。

(例)

- $D_{cons} = \{(0, 0, \cdots, 0), (1, 1, \cdots, 1)\}$ (consensus problem)
- $D_{elec} = \{(1, 0, \cdots, 0), (0, 1, \cdots, 0), \cdots, (0, 0, \cdots, 1)\}$ (election problem)
- $D_{rename} = \{(d_1, \cdots, d_N) : i \neq j \Rightarrow d_i \neq d_j\}$

4.1 A Solvable Problem : Renaming

Theorem 4.2 $t \geq \frac{N}{2}$ に対して、 *renaming algorithm* は存在しない。

(証明の概略)

$t \geq \frac{N}{2}$ の場合、サイズ $N-t$ の、二つの別のグループ S と T が形成される。プロセスは、 t 個まで落ちる可能性があるため、一つの execution で、各グループはそれぞれ独立に決定にたどり着かなければならない。これらが互いに矛盾した決定であることを示せば良い。

(証明) Proposition 2.4より、全ての initial condition γ に対し、 S 内のプロセスは全て決定済で、 $\gamma \rightsquigarrow_S \delta_S$、となるような configuration δ_S が存在する。 T についても同様。アルゴリズムは、 N-t のプロセスからなるグループについて、 $N-t$ 個の初期 ID からなるベクトルから、 $N-t$ の新しい ID からなるベクトルの関係を定義するものである。

初期 ID の namespace は unbounded で、新しい ID は限られた範囲より選ぶので、重複した出力にマップされる、 disjoint な input が存在する。すなわち、次のような input vector($N-t$ 個の成分) を $\vec{u}, \vec{v}$ が存在する。

- 全ての i, j について、 $u_i \neq u_j$ である。
- 対応する output vector を $\vec{d}, \vec{e}$ としたとき、ある i, j について、 $d_i = e_i$ となる。

そして、つぎのようにして、正しくない execution をつくることができる。 initial configuration γ において、プロセスグループ S, T の input がそれぞれ $\vec{u}, \vec{v}$ であるとする。 δ_T を、 output が $\vec{e}$ である決定したプロセスで、 sequence σ_T によって γ から到達するものとする。 Proposition **??**により、 σ_T は γ_S(S から到達する、 $\vec{d}$ に決定する configuration) に適用可能。従って、プロセス全体として、最終的に、 $\sigma_T(\gamma_S)$ に決定する。ところが、 $d_i = e_j$ なので、二つのプロセスが同じ名前に決定したことになり、アルゴリズムは inconsistent である。 □

以下、 $t < \frac{N}{2}$ であるとする。

4.2 The renaming algorithm

Algorithm 14.2 を参照のこと。入力 x_p は、プロセス p の、最初の名前。出力 y_p は、プロセス p の、後の名前。決定は、集合 V_p と、集合 V_p 中での x_p の rank のペア。

Stable set. p が、同じ $V(=V_p)$ を $N-t$ 回受け取ったとき、そのプロセス p は stable set V に到達した、という。

Lemma 4.3 *Stable sets* は *totally ordered* である。すなわち、 q が *stable set*V_1 に到達し、 r が *stable set*V_2 に到達するとき、 $V_1 \subseteq V_2$ または $V_2 \subseteq V_1$ である。

Lemma 4.4 どの *fair t-crash execution* でも、正しいプロセスはそれぞれ少なくとも一度は *stable set* に到達する。

Theorem 4.5 *Algorithm 14.2* は、出力の *namespace* の大きさが $K = (N - \frac{t}{2})(t+1)$ の場合の、 *renaming problem* の解になっている。

ちなみに、 $K = \sum_{s=N-t}^{N} s = (N - \frac{t}{2})(t+1)$ である。 (証明)

どの t-crash fail execution でも、正しいプロセスは stable set に到達するので、全ての正しいプロセスは、新しい名前に決定する。次に、新しい名前が全て違うことを示すために、 stable set V_1, V_2 を、それぞれ q, r から到達するものとする。

もし、 V_1, V_2 のサイズが異なれば、決定に $\#V_p(V_p$ のサイズ) が含まれているので、決定は異なる。もし、 $\#V_1 = \#V_2$ ならば、 Lemma **??** より $V_1 = V_2$ である。その場合、 q, r の rank は異なるので、決定は異なる。

4.3 Discussion

Algorithm 14.2 では、各プロセスは、自分の名前を決定した後も止まらず、他のプロセスを助ける。プロセスの中に非常に遅い奴がいる場合にも対処するために必要。

4.4 Extended Impossibility Results

この節では、 Theorem 4.7 を証明するのが目的。

Definition 4.6 以下のようなグラフ $G_T = (V, E)$ を、タスク T の *decision graph* と定義する。

- $V = D_T$
- $E = \{(\vec{d_1}, \vec{d_2}) : \vec{d_1}$ と $\vec{d_2}$ は、ただ一つの成分のみ異なる。 $\}$

また、 G_T が連結なとき、タスク T は連結であるという。

Impossibility results は、 Non-trivial なアルゴリズム、つまり次の条件を満たすアルゴリズムについて示される。

- termination
- consistency
- **Non-trivaiality.** 全ての $\vec{d} \in D_T$ に対し、 reachable configuration が存在し、その中では、 $\vec{d}$ に決定している。

Theorem 4.7 連結でないタスク T に対し、 *Non-trivial,1-crash-robust decision algorithm* は存在しない。

(証明の方針) 上のようなアルゴリズムがあったとする。そのとき、G_T が二つの異なる連結成分をもつとき、決定がどちらの連結成分に属するか、ということを出力する新しいアルゴリズムを作ることができる。ところが、そのようにして作ったアルゴリズムは asynchronous,deterministic,1-crash-robust consensus algorithm になってしまうので、矛盾となることを示す。

(証明)

上のようなアルゴリズムがあったとし、これを A とする。このとき、A から consensus algorithm A' が導かれるかもしれないが、このときに矛盾が生じることを示す。簡単のため、G_T は二つの連結成分のみ持つとする ('0' と '1')。

アルゴリズム A' は、最初は、A を simulate し、d に決定する際に、プロセスは ¡**vote**,d¿ を shout し、$N-1$ 個のメッセージを待つとする。この際、正しいプロセスは すでに A で決定しているので、デッドロックは起こらない。

メッセージを受信した後、各プロセス p は、D^N 中の vector の、$N-1$ 個の成分を持っている。この vector は vote を受け取ってないプロセス用の値で拡張され、このようにして、全部のベクトルは D_T に入る。

違うプロセスはベクトルを拡張するべく、違う extension を計算してしまう可能性があると思うかもしれないが、しかし、これらの extension は、G_T の、同じ連結成分に属する。$N-1$ 個の vote を受け取ったそれぞれのプロセスは、extension が属する連結成分の名前に決定することにすれば、A' は consensus algorithm のままである。

Termination. すでに、全ての正しいプロセスは 少なくとも $N-1$ 個の vote を受け取ったことは示した。

Agreement. まず、vector $\vec{d_0} \in D_T$ の存在を議論する。全ての正しいプロセスは、$\vec{d_0}$ の成分のうち、$N-1$ 個を得るものとする。

1. A 内の全てのプロセスが決定する場合。各々のプロセスは、$\vec{d_0}$ 中の $N-1$ 個の成分を得る。そのうち足りない成分がどのプロセスの成分かは、一般には各プロセスで異なる。

2. プロセス r 以外が、決定する場合。全ての正しいプロセスは、同じ $N-1$ 個の成分をもつ (プロセス r の成分)。プロセス r は 故障した可能性があるが、r がただ単に遅いだけの場合もあり、r が決定に到達できる可能性も残されていなければならない。すなわち、vector $\vec{d_0}$ が存在しなければならない。

従って、vector $\vec{d_0}$ が存在するので、全てのプロセスは、d_0 の同じ連結成分に決定する。

Non-triviality. A の Non-triviality により、成分が 0 または 1 の、両方の成分に到達可能である。A' を作るにさいし、どちらの決定も可能である。

ここに、asynchronous,deterministic,1-crash-robust consensus algorithm A' が A より構築できた。しかし、そのようなアルゴリズムは Theorem **??** により禁止されているので、A は存在しない。 □