最小生成木構成問題を解く自己安定分散アルゴリズム

1P－3　有永 憲一　若林 真一　小出 哲士　吉田 典可
広島大学 工学部

1 まえがき

近年, 分散アルゴリズムの研究において, 自己安定 (self-stabilizing) 分散アルゴリズムが注目されている. 自己安定アルゴリズムとは, ネットワークの初期状態として何も仮定しなくても有限時間内に問題を解くアルゴリズムである.

Dijkstra[3] によって自己安定アルゴリズムの概念が導入されて以来, 様々な自己安定アルゴリズムが提案されている. ネットワークのトポロジに何も仮定しない一般ネットワークに対して, 生成木構成問題 [1, 4], 最小生成木構成問題 [5] などを解く自己安定アルゴリズムが提案されている. しかし, 文献 [2] や [5] のアルゴリズムでは, 根から各プロセッサまでのパスをメッセージとして交換しているので, 大規模なネットワークではメッセージ長が長くなってしまう. 本稿では, 一般ネットワーク上で最小生成木構成問題を解くメッセージ効率の良い自己安定アルゴリズムを提案する.

2 モデルと問題

2.1 ネットワークモデル

- ネットワークは $N = (P, L)$ で表される. 但し, P をプロセッサの集合, L を通信リンクの集合とする. $(p, q) \in L$ のとき, p, q 間に全 2 重リンクが存在する. プロセッサ数を n とすると, 各プロセッサを $p_0, p_1, \cdots, p_{n-1}$ と表す.
- 各プロセッサは相異なる識別子を持ち, その識別子をパラメタとする状態機械である. プロセッサ p_i は 2 項組 S_i, T_i で定義される. S_i を p_i の状態集合, S'_j を p_i の隣接プロセッサ $p_i[j]$ の状態集合, T_i を p_i の状態遷移関数とすると, $T_i : S_i \times S'_1 \times S'_2 \times \cdots \times S'_d \to S_i$ である. 但し, d はプロセッサ p_i の次数.
- 通信方式はレジスタ通信モデルを仮定する. リンク $(p_i, p_j) \in L$ は 2 つのレジスタ r_{ij}, r_{ji} からなっており, p_i から p_j への通信はレジスタ r_{ij} を, p_j から p_i への通信はレジスタ r_{ji} を用いる.
- ネットワーク状況は $c = (s_0, s_1, \cdots, s_{n-1})$ で表される. $s_i \in S_i$ を任意のネットワーク状況 c でのプロセッサ p_i の状態, $\mathcal{A}$ を任意のアルゴリズム, Q をプロセッサ集合の任意の部分集合とする. Q に属するすべてのプロセッサが状態遷移関数によって状態を変化させ, ネットワーク状況が c' になるとする. これを $c \to c'(Q)$ と表す. プロセッサの部分集合の無限系列をスケジュールと呼ぶ. c_0 をネットワーク状況, $T = Q_0, Q_1, \cdots$ を任意のスケジュールとする. この時, ネットワーク状況の無限系列 $E = c_0, c_1, \cdots$ が各 $i (0 \le i)$ について $c_i \to c_{i+1}(Q_i)$ を満たすならば, E を 初期状態 c_0, スケジュール T に対するアルゴリズム $\mathcal{A}$ の実行という. スケジュール T にネットワークのすべてのプロセッサが無限回現れる時, T は公平であるという. また, 実行 E のスケジュール T が公平ならば, 実行 E は公平であるという.
- デーモンは R/W デーモン (Read/Write Demon) を仮定する. 同時に 1 つのプロセッサしか動作せず ($|Q_i| = 1$), 1 原子動作では, 一つの隣接レジスタからの読み込みと状態変化, または一つの隣接レジスタへの書き込みと状態変化が行える.

2.2 自己安定アルゴリズム

LS を問題によって定まる正しい実行の集合とする. 任意のネットワーク状況 c_0 で始まるアルゴリズム $\mathcal{A}$ の任意の公平な実行 $E = c_0, c_1, \cdots$ に対して, ある j が存在し, $E_j = c_j, c_{j+1}, \cdots$ が LSに属する時, $\mathcal{A}$ は LSに関して自己安定であるという. このとき, 正しい実行 E_j 内のネットワーク状況 $c_j, c_{j+1}, \cdots$ を正当な状況と呼ぶ.

2.3 最小生成木構成問題

任意のネットワークにおいて, どのような初期状態からアルゴリズムを実行しても, いずれは各プロセッサが最小生成木 (以下 MST) に属するリンクを正しく求める自己安定分散アルゴリズムを構成せよ.

3 自己安定最小生成木構成アルゴリズム

3.1 アルゴリズムの概要

まず, アルゴリズムの概要を述べる. 以下では, アルゴリズムの実行中に生成されるネットワーク中の各連結成分のことをフラグメントと呼ぶ. MST の性質より, MST に含まれないあるリンク e を MST に追加すると, ループができる. その時, そのループの中で最大の重みを持つリンクは e である.

このアルゴリズムは, フラグメントに新しくリンクを加えて生成木 (MST でなくてもよい) を構成する部分と, その生成木に含まれないリンク e を生成木に加えた時にできるループ内で, e よりも大きな重みのリンクと e を置き換えることで MST を構成する部分からなっている. リンクを追加して生成木を構成する部分では, 各プロセッサは自分を根として生成木を構成しようとす

"A Self-Stabilizing Algorithm for Constructing a Minimum Spanning Tree"
Ken'ichi ARINAGA, Shin'ichi WAKABAYASHI, Tetsushi KOIDE, and Noriyoshi YOSHIDA
Faculty of Engineering, Hiroshima University.

る．そして，より大きな識別子の根を持ったフラグメントにプロセッサが結合していくことによって，いずれは最大の識別子の根を持ったフラグメントがネットワーク中に広がり生成木となる．リンク置換えでMSTを構成する部分では，まずリンク e を加えることによってできるループを見つけるために，リンク e の情報をフラグメント内に伝搬させる．ループを発見したならば，その中に e の重みよりも大きな重みを持つリンクがあるかどうかを調べ，もしそのようなリンクがあれば，e と置き換える．この操作をすべてのリンクに対して適用すればMSTを構成できる．

3.2 リンクを追加する部分の動作

フラグメントにリンクを追加して生成木を構成する部分の動作は，文献 [1] と同様である (図 1)．

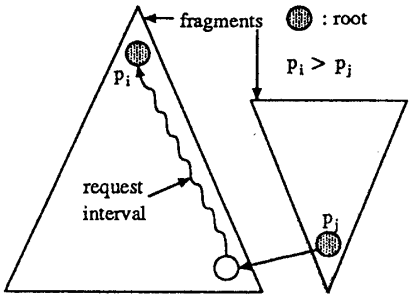

図 1 リンク追加の動作

各プロセッサは自分を根としてフラグメントを構成しようとする．各フラグメントの根は，より大きな識別子の根を持つフラグメントに結合するために，リクエストを相手のフラグメントの根に向かって伝搬させ，許可が返ってきたならばそのフラグメントに結合する．図 1 では，右のフラグメントの根である p_j が隣接しているフラグメントの根が $p_i > p_j$ であることを見つけ，そのフラグメントに対してリクエストを伝搬させている．この動作を繰り返すことによって，いずれ最大の識別子の根を持つ生成木が構成される．

3.3 リンク置換えを行う部分の動作

リンク置換えを行う部分の動作を図 2 に示す．フラグメント内のプロセッサ u は，自分の隣接リンクの中で木に含まれていないリンク e を選ぶ．そして，そのリンクの重み w，そのリンクに隣接しているプロセッサ u, v といった情報を，キュー $FQueue$ に入れる．各プロセッサは，隣接プロセッサの $FQueue$ の先頭の内容を自分の $FQueue$ にコピーすることによって，これらの情報を伝搬させる．複数の子を持つプロセッサが $FQueue$ によってメッセージを子から受けとったならば，キュー $BQueue$ を用いて子孫にも伝搬する．伝搬のさせ方は $FQueue$ の場合と同様である．このとき，キュー $BQueue$ によってメッセージを受信したプロセッサは，自分の親に繋がるリンクの重みが $BQueue$ の先頭のリンクの重みよりも大きいかどうかを調べる．もし大きければ，メッセージにリンク置き換え可能のフラグを立てて子孫に伝搬する．図 2 では，プロセッサ t は $w < w'$ であることが分かるので，メッセージにフラグを立てる．このフラグが立っているメッセージを受信したプロセッサ v は，そのリンクが自分に隣接しているので，親に繋がるリンクを e に置き換える．以上の操作を繰り返すことにより，いずれ生成木からMSTが構成される．

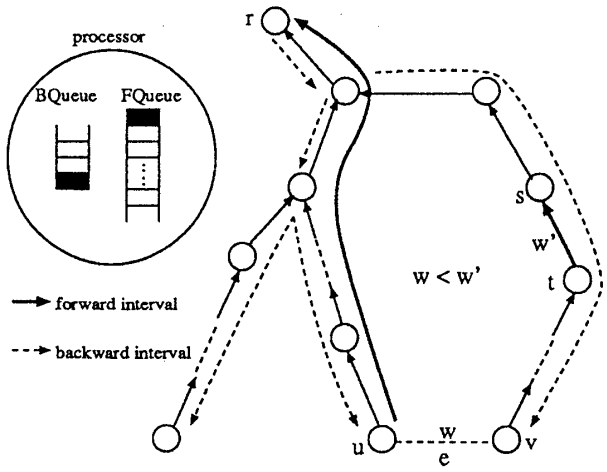

図 2 リンク置き換えの動作

4 アルゴリズムの正当性

補題 1 リンク追加の動作によって構成されたフラグメントにおいてリンクの置換えが起きたとしても，そのフラグメント内の親子関係に矛盾は生じず，いずれ生成木が構成される． □

補題 2 あるプロセッサより祖先のすべてのプロセッサが MST に属するリンクを正しく求めているとする．その時，それらのプロセッサはその後も MST に属するリンクを正しく求めた状態であり続ける． □

定理 1 いずれ生成木は MST になる． □

5 あとがき

今後の課題として，一度故障が起こった時から再び安定するまでのラウンド数の解析などが挙げられる．

参考文献

[1] Y. Afek, S. Kutten and M. Yung: "Memory-efficient self-stabilizing protocols for general networks," Proc. 4th International Workshop on Distributed Algorithms (LNCS 486), pp. 15–28 (1990).

[2] Z. Collin and S. Dolev: "Self-stabilizing depth-first search," Inf. Process. Lett., 49, pp. 297–301 (1994).

[3] E. W. Dijkstra: "Self-stabilizing systems in spite of distributed control," Commun. ACM, Vol. 17, No. 11, pp. 643–644 (1974).

[4] S. Dolev, A. Israeli and S. Moran: "Self-stabilization of dynamic systems assuming only read/write atomicity," Proc. 9th ACM Symposium on Principles of Distributed Computing, pp. 103–117 (1990).

[5] 小谷, 片山, 増澤, 都倉："ネットワークの重み最小生成木を構成する自己安定分散アルゴリズムについて," 信学技報, COMP 92-5 (1992).