数理解析研究所講究録
第1148巻 2000年 212-218

木ネットワークでヒープ順序を実現する自己安定プロトコル

奈良先端大　長谷川学 (Manabu HASEGAWA)
奈良先端大　浮穴学慈 (Satoshige UKENA)
奈良先端大　片山喜章 (Yoshiaki KATAYAMA)
奈良先端大　増澤利光 (Toshimitsu MASUZAWA)
奈良先端大　藤原秀雄 (Hideo FUJIWARA)

奈良先端科学技術大学院大学 情報科学研究科
〒 630-0101 奈良県生駒市高山町 8916-5
e-mail:{manabu-h@is, satosi-u@is, katayama@itc,
masuzawa@is, fujiwara@is}.aist-nara.ac.jp

あらまし ネットワークで相互接続されたプロセスから構成される分散システムにおいて，故障耐性のあるプロトコルが重要である．故障耐性を実現する有力な手法の一つに，自己安定プロトコルがある．自己安定プロトコルとは，任意のネットワーク状況から実行を開始しても，解を求めて安定するプロトコルである．この性質から，自己安定プロトコルは任意の一時故障に耐性がある．本稿では，木ネットワークにおいてプロセス間の同期を実現する自己安定プロトコルを利用して，ヒープ順序づき木を構成する自己安定プロトコルを提案する．提案するヒープ順序づき木を構成するプロトコルは，安定時間 $O(h)$，各プロセスの領域計算量 $O(K)$ であり，既知の結果と比べ安定時間，領域計算量ともに改善されている．ここで，h は木の高さ，K は入力のサイズを表す．

キーワード
分散システム，木ネットワーク，自己安定プロトコル，隣接間同期化，ヒープ

1 序論

ネットワークで相互接続されたプロセスからなる分散システムにおいて，プロセスが協調して問題を解くプロトコルの研究が盛んに行われている．特に，一部のプロセスの故障に関わらず，問題を解くことのできる故障耐性を有するプロトコルが重要視されている．

通常のプロトコルでは，プロトコル実行開始時の分散システムの大域状況が，あらかじめ決められた初期状況であると仮定する．つまり，各プロセスは，あらかじめ決められた初期状態から実行を開始する．これに対し，**自己安定プロトコル (self-stabilizing protocol)** は，プロトコル実行開始時の分散システムの大域状況について何も定めない．つまり，自己安定プロトコルは，任意の大域状況から実行を開始しても，問題の解を求めた状況に安定するプロトコルである．この性質から，自己安定プロトコルでは，プロセスの一時的な故障 (プロセスの変数の値，プログラムカウンタの値の破壊など) により分散システムがどのような大域状況に陥っても，故障したプロセスが復旧すれば，自動的に再び解を求めた状況で安定する．したがって自己安定プロトコルは，長期に渡って分散システムの状況を安定に保ち，プロセスの一時的な故障に柔軟に対応することが求められる分散システムの実現に適している．自己安定プロトコルは 1974 年に Dijkstra[3] によってはじめて導入された概念であるが，一時故障に対して優れた故障耐性を持つことから，故障耐性のあるプロトコルとして注目され，特に近年，多くの研究が行われている．

ヒープは，逐次アルゴリズムにおいて重要なデータ構造であり，ソートや優先順序キューなど，多くの応用を持つ．このような重要なデー

タ構造をネットワーク上に分散型データ構造として実現し，分散システムの設計・開発に利用することは有用である．そのため，様々なデータ構造に対し，それらをネットワーク上に実現するためのプロトコル（分散アルゴリズム）に関する研究が，数多く行われている．

本稿では，木ネットワークで**ヒープ順序づき木**を構成する自己安定プロトコルを提案する．ここで，ヒープ順序づき木の構成とは，各プロセスが持つ値（入力値）を，ヒープ順序を満たす（つまり，各プロセスがその子プロセスよりも大きい値を持つ）ように並べ替えることである．なお，本稿では木ネットワークを対象とするが，生成木を構成する自己安定プロトコルを併用することにより，提案するプロトコルは一般のネットワークに対しても適用できる．

木ネットワークでヒープ順序づき木を構成する自己安定プロトコルは，Bourgon ら [2] によって提案されている．このプロトコルの安定時間は $O(nh)$ である．ここで，n はシステムのプロセス数，h は木の高さである．また，各プロセスの領域計算量は $O(\delta K)$ である．ここで，δ はプロセスの次数，K は入力値のサイズである．

Alima[1] は，木ネットワークにおいて安定時間 $O(h)$，各プロセスの領域計算量 $O(\delta + K)$ のヒープ順序づき木を構成する自己安定プロトコルを提案している．しかしこのプロトコルでは，隣接プロセス間で 1 度しか値の比較・交換を行わず，正しくヒープ順序づき木を構成できない．このプロトコルを修正したものを繰返し適用すれば，ヒープ順序づき木の構成も可能となるが，そのときの安定時間は $O(h^2)$ になってしまう．

本稿では，安定時間 $O(h)$，各プロセスの領域計算量 $O(K)$ のヒープ順序づき木を構成する自己安定プロトコルを提案する．これは文献 [2]，[1] のプロトコルと比べ，安定時間，領域計算量を共に改善している．文献 [2]，[1] のプロトコルの安定時間が大きいのは，大域的同期化プロトコル（根がシステム全体の調停者となってプロセスを同期させる）を使用しているためである．そこで，本稿では，隣接プロセス間でのみ同期を実現する**隣接間同期化プロトコル** [4] を利用して，ヒープ順序づき木を構成する自己安定プロトコルを設計する．

同期化プロトコルは，分散システムにおいて，非同期式システムで同期式システムを模倣することを可能にするプロトコルであり，幅広く研究されている [5, 6, 7]．Johnen らは，木ネットワークにおいて自己安定隣接間同期化プロトコルを提案している [4]．これは，木ネットワークで隣接プロセス間の同期を実現するものである．つまり，任意の隣接プロセス間で，同時に両方が動作することなく，いずれか一方ずつ，交互に動作させる仕組みを提供する．このプロトコルの安定時間は 0 であり，1 プロセッサにつき 1 ビットの領域 (領域計算量 $O(1)$) を必要とする．

本稿の構成は，以下の通りである．第 2 節では，分散システム，自己安定プロトコル，隣接間同期化問題，ヒープ順序づき木構成問題について定義を行う．第 3 節では，隣接間同期化プロトコルを示す．第 4 節では，ヒープ順序づき木を構成する自己安定プロトコルを提案する．最後に，第 5 節で本稿での結論について述べる．

2 諸定義

2.1 分散システム

分散システムは，無向連結グラフ $D = (V, E)$ で定義される．ここで，V は頂点の集合 ($V = \{0, 1, \cdots, n-1\}$ とする)，E は辺の集合とする．それぞれの頂点はプロセスを表し，辺は双方向通信リンクを表す．本稿では，木構造のネットワークを扱う．ネットワーク中で根プロセスを r，葉プロセスの集合を L，そして内部プロセス (根プロセスは内部プロセスに含めない) の集合を I で表す．

各プロセス i の隣接プロセスの集合を N_i とする．各プロセス i は，隣接するプロセスを区別できるものとする．また，隣接するプロセス数 $|N_i|$ をプロセス i の次数という．各プロセス i は，木ネットワーク上での親と子のプロセスを認識できるものとし，親プロセスを P_i，子プロセスの集合を Cld_i で表す．従って，P_i，Cld_i から，プロセス i は根，葉，内部プロセスのいずれであるかを認識できる．また，木の

高さを h で表す．

各プロセスを状態遷移機械としてモデル化する．状態を変数で表し，状態遷移をガード付アクション (以下，アクションという) で表す．ここで，各プロセスは自分のもつ変数のみに書き込み可能であるとし，また，隣接プロセスの変数の値を直接参照できる (状態通信モデル) ものとする．各プロセスの各アクションはラベルづけされており，以下のように表す．

$< label > :: < guard > \rightarrow < statement >$

i の各アクションのガード$< guard >$は，i および i の隣接プロセスの変数からなる論理式で表される．プロセス i はガードが真の場合のみ命令文$< statement >$を実行する．命令文では，i の持つ変数を更新する．本稿では，ガードが評価され，命令文が実行されるまでを 1 原子動作とする．また，この 1 原子動作を 1 ステップと呼び，ガードが真の命令文の実行をアクションの実行と呼ぶ．

各プロセス i の状態集合を S_i で表す．分散システム全体の取り得る状況の集合を$\mathcal{C}$ とすると，$\mathcal{C} = S_0 \times S_1 \times \cdots \times S_{n-1}$ である．つまり，分散システムの各大域状況 (以下単に状況と呼ぶ) c は，$(s_0, s_1, \ldots s_{n-1})$ で表される．

プロトコル $\mathcal{P}$ は，各プロセスの動作を規定する．従って，プロトコル $\mathcal{P}$ を，すべての状況の集合 $\mathcal{C}$ 上の 2 項関係 $\mapsto$ とみなすことができる．$c \in \mathcal{C}$ を任意の状況，$Q \subseteq V$ をプロセスの任意の部分集合とする．Q に属するすべてのプロセスが，同時に 1 ステップを行うことにより状況が c から c' となる (つまり $c \mapsto c'$) とき，$c' = \triangle(c, Q)$ と表す．$c = (s_0, s_1, \ldots, s_{n-1})$, $c' = (s'_0, s'_1, \ldots, s'_{n-1})$ とするとき，$i \notin Q$ のとき，あるいは，$i \in Q$ でもそのアクションのガードが偽のとき，$s_i = s'_i$ である．

プロセスの空でない部分集合の無限系列をスケジュールと呼び，$\mathcal{Q} = Q_1, Q_2, \ldots$ と表す．このとき，状況の無限系列 $E = c_0, c_1, c_2, \cdots$ が $c_{i+1} = \triangle(c_i, Q_{i+1})$ $(i \geq 0)$ を満たすとき，E を初期状況 c_0，スケジュール $\mathcal{Q}$ に対する実行と呼ぶ．つまり，E は $Q_1, Q_2, \ldots$ に属するプロセスが順に動作するときの状況変化を表す系列である．本稿では，弱公平な実行のみを考える．これは，各プロセス i に対し，ある状況 c_j 以降常にガードが真でありながら，c_j 以降 命令文 が実行されないようなアクションは存在しないことを保証する．つまり，連続してガードが真のアクションを持つとき，いつかはその命令文が実行されることを保証する．

本稿では，非同期式システムを想定しているが，時間計算量の評価は，同期式システムで用いられるラウンドを用いて行う．つまり，時間計算量の評価では，各 $i (i \geq 1)$ に対し，$Q_i = V$ となるスケジュール $\mathcal{Q} = Q_1, Q_2, \cdots$ に対する実行 $E = c_0, c_1, \cdots$ を考え，状況 c_i を第 i ラウンド終了時の状況とする．このような実行 E を同期式実行と呼ぶ．

2.2 自己安定プロトコル

自己安定プロトコルは，任意の状況から実行を開始しても，問題の解を求めた状況に安定するプロトコルである．この性質から，自己安定プロトコルは，プロセスの一時的な故障 (プロセスの変数の値，プログラムカウンタの値の破壊) により，分散システムがどのような状況に陥っても，故障したプロセスが復旧すれば，やがて再び解を求めた状況で安定する．つまり，自己安定プロトコルは，一時的な故障に対する高度な故障耐性を有する．

ここでは文献 [1, 2] と同様に，*attractor* という概念を用いて，自己安定プロトコルを定義する．

定義 2.1 Closed Attractor

X, Y を，分散システムの状況の集合 $\mathcal{C}$ に対して定義される述語とする．以下の条件を満たすとき Y は X に対する closed attractor であるといい，$X \triangleright Y$ と表す．

1. 述語 X を満たす任意の状況を c_0 とする．c_0 を初期状況とする任意の実行 $E = c_0, c_1, \ldots$ において，述語 Y を満たす状況 c_i $(i \geq 0)$ が存在する．
2. 述語 Y をみたす任意の状況を c'_0 とする．c'_0 を初期状況とする，任意の実行 $E' = c'_0, c'_1, \ldots$ において，すべての状況 c'_i が述語 Y を満たす． □

$X \triangleright Y$ は，述語 X を満たす任意の状況から始まるすべての実行に対して，システムが述

語 Y を満たすような状況に到達し，いったんそのような状況に到達すると，その後のすべての状況も述語 Y を満たすことを意味する．次に，この概念を用いて自己安定プロトコルを定義する．

定義 2.2 自己安定プロトコル

$\mathcal{P}$ をプロトコルとし，$\mathcal{P}$ のすべての実行の集合を $\mathcal{P}$ とする．また，$\mathcal{SP}$ を実行の集合 $\mathcal{P}$ に対して定義される述語とする．すべての状況の集合 $\mathcal{C}$ に対して定義される述語 $\mathcal{L}$ が存在し，以下の条件を満たすとき，プロトコル $\mathcal{P}$ は $\mathcal{SP}$ に対して自己安定であるという．

1. 正当性：述語 $\mathcal{L}$ を満たす任意の状況を α とする．α を初期状況とする，任意の実行 E が述語 $\mathcal{SP}$ を満たす．
2. 閉包性・収束性：$true \triangleright \mathcal{L}$．つまり，述語 $\mathcal{L}$ が，システムのすべての状況の集合 $\mathcal{C}$ に対する closed attractor である． □

定義 2.2 において，述語 $\mathcal{SP}$ は，分散システムに要求される動作を規定するものである．そこで，述語 $\mathcal{SP}$ のことをプロトコル要求と呼ぶ．また述語 $\mathcal{L}$ は，プロトコル要求を満たす実行の初期状況を規定するので，$\mathcal{L}$ を満たす状況を正当な状況と呼ぶ．

プロトコル要求 $\mathcal{SP}$ に対する自己安定プロトコルは，いずれ正当な状況に到達し，それ以降の実行は $\mathcal{SP}$ を満たす．自己安定プロトコルの安定時間を以下のように定義する．

定義 2.3 安定時間

$\mathcal{P}$ をプロトコル要求 $\mathcal{SP}$ に対する自己安定プロトコルとする．$\mathcal{P}$ の任意の実行 E において，第 t ラウンド終了時の状況が正当な状況のとき，$\mathcal{P}$ の安定時間は t であるという． □

2.3 隣接間同期化問題

本稿では，木ネットワークにおいて隣接プロセス間の同期を実現する自己安定プロトコルを利用する．この自己安定プロトコルのプロトコル要求 NS は次のように定義される．

定義 2.4 プロトコル要求 NS

任意の実行を $E = c_0, c_1, \cdots$ とする．任意のプロセスを i とし，i が状況 c_s, c_t $(s < t)$ でアクションを実行した (アクションを実行する直前の状況が c_s, c_t) とする．i が $c_s, \cdots, c_t$ の間で正確に一度だけアクションを実行するなら，i の任意の隣接プロセス j も $c_s, \cdots, c_t$ の間で正確に一度だけアクションを実行する． □

これは，プロトコル要求 NS を満たす任意の実行 E において，あるプロセス p が実行する連続する 2 つのアクションの間に，p に隣接するすべてのプロセスが正確に一度だけアクションを実行することを意味する．

2.4 ヒープ順序づき木構成問題

本稿では，木ネットワークの各プロセスが，初期状況で持つ値 (初期値) を並べ換えることにより，ヒープ順序 (各プロセスがその子プロセスよりも大きい値を持つ) を構成する自己安定プロトコルを提案する．

ただし，異なるプロセスの初期値は異なるものとする．以下に，ヒープ順序づき木構成問題のプロトコル要求 HP を定義する．

定義 2.5 プロトコル要求 HP

各プロセス i は，読出し専用の入力変数 in_i と書込専用の出力変数 out_i を持つ．
実行を E とする．E において，in_i の値が変化しなければ，out_i の値は変化せず，以下の条件を満たす．

1. $\{out_i |\ i \in V\} = \{in_i |\ i \in V\}$
2. $\forall i \in V - \{r\}:\ out_i < out_{P_i}$ □

3 隣接間同期化プロトコル

木ネットワークにおける隣接間同期化プロトコル $\mathcal{NSP}$[4] を図 3.1 に示す．各プロセス i は，論理型変数 col_i を持つ．各プロセス i は，親プロセスと子プロセスの 変数 col を読む．そして，col_i と親プロセスの col_{P_i} が異なる値であり，かつ，すべての子プロセスの col が col_i と同じ場合のみ動作し，col_i の値を反転させる．これより，プロセスとその隣接プロセスが交互に動作することを実現している．なお，根プロセスは親プロセスとの比較をせず，葉プロセスは子プロセスとの比較をしない点以外は，他のプロセスと同一である．

(定数) Cld_i : 子の集合, P_i : 親のプロセス
(変数) col_i : 論理型変数

(アクション):
For the root
S_1 :: $\forall j \in Cld_i$: $col_j = col_i$ $\rightarrow$ $col_i := \neg col_i$

For the internal processes
S_2 :: $col_{P_i} \neq col_i \wedge (\forall j \in Cld_i : col_j = col_i)$
$\rightarrow$ $col_i := col_{P_i}$

For the leaf processes
S_3 :: $col_{P_i} \neq col_i$ $\rightarrow$ $col_i := col_{P_i}$

図 **3.1** 隣接間同期化自己安定プロトコル $\mathcal{NSP}$ (プロセス i)

定理 3.1 [6] プロトコル $\mathcal{NSP}$ は, 安定時間 0, 各プロセスの領域計算量 $O(1)$ の隣接間同期化自己安定プロトコルである. □

さらに, プロトコル $\mathcal{NSP}$ に対して, 次の定理が成り立つ.

定理 3.2 プロトコル $\mathcal{NSP}$ の任意の同期式実行を Eとする. E の第 $2h$ ラウンド終了時以降, 各プロセスは 2 ラウンドに 1 度, 正確にアクションを実行する. □

4 ヒープ順序つき木を構成する自己安定プロトコル

本節では, 木ネットワークにおいてヒープ順序づき木を構成する安定時間 $O(h)$, 各プロセスの領域計算量 $O(K)$ の自己安定プロトコル $\mathcal{HPP}$を提案する. ここで, h は木の高さを, K は入力値のサイズを表す.このプロトコルは, 文献 [2] のプロトコルの安定時間 $O(nh)$, 領域計算量 $O(\delta K)$ を改良している.

4.1 プロトコル $\mathcal{HPP}$ の概略

木ネットワーク上の各プロセスが, それぞれ 1 つずつデータを持つとする. ヒープ順序を実現するためには, より大きな値を根に向かって移動させ, 小さな値は葉に向かって移動させればよい. つまり, 各プロセスと, その子プロセスの間で値を比較し, 子プロセスの持つ値の最大値が親プロセスの持つ値より大きな場合, その親と子プロセス間で値を交換する. これを繰り返せば, やがて木ネットワーク全体でヒープ順序を実現できる.

このとき, 複製や損失を起こすことなく, 正しくデータ交換を行うために, 本プロトコルでは, 隣接間同期化プロトコルを用いる.

各プロセス i の入力変数 in_i の値は, プロトコル実行中は変化しないものとする. ヒープ順序を実現するには, 前述したようにこの値を移動させなければならず, 各プロセス i はそのための作業用変数 w_i, r_iを持っている. 自己安定プロトコルでは, 初期状況に仮定をおかないため, 初期状況において, 作業用変数の値がどのプロセスの入力変数 val の値とも一致しないことや, あるプロセスの入力変数の値がどのプロセスの作業変数の値とも一致しない可能性がある. そこで本プロトコルでは, 作業用変数の値を並べ換えることにより, ヒープ順序を実現した後, ネットワーク全体にリセットをかけ, 各プロセス i の入力変数 in_i の値を作業用変数にコピーし, 再び作業用変数に対してヒープ順序の構成を繰り返す. ヒープ順序づき木が構成されると, 各プロセス i は作業用変数の値を出力変数 out_i にコピーする. リセット, ヒープ順序づき木構成は繰り返し実行されるが, 入力変数の値は変化しないので, 2 回目以降では同じヒープ順序づき木が構成される. 従って, 各プロセス i は out_i に同じ値を書き込むことになり, out_i の値は変化しなくなる.

なお, 異なるプロセスの入力変数 in_i の値は異なるものとする (同じ値がある場合, プロセスの識別子との二項組にすることにより, 異なる値とみなせる).

4.2 プロトコル $\mathcal{HPP}$

ヒープ順序づき木を構成する自己安定プロトコルを図 4.1, 4.2 に示す. 各プロセスのアクションにおけるガードは, 隣接間同期化プロトコルのものと同じである. プロセスの動作は隣接間同期化プロトコルの動作を含んでおり, これによりプロトコル $\mathcal{HPP}$ は同期化して動作する.

次に，アクション中に含まれる定数，変数，手続きを説明する．

(定数)

P_i : i の親
Cld_i : i の子の集合
in_i : i への入力値
(値はユニークで実行中変化しない)

---

for root processes
$H_1 :: \forall j \in Cld_i : \ col_j = col_i$
→ **HEAPIFY**;
  if ($r_i = \perp$
    $\wedge$ $(\forall j \in Cld_i : change_j = false)$)
    /* ヒープ完成 */
    $change_i := false$; $reset_i := true$;
    RESET;
  else
    $change_i := true$; $reset_i := false$
  $col_i := \neg col_i$;

for internal processes
$H_2 :: col_{P_i} \neq col_i \ \wedge \ (\forall j \in Cld_i : \ col_j = col_i)$
→ **if** ($reset_{P_i} = false$)
    $reset_i := false$; **HEAPIFY**;
    if ($r_i = \perp$
      $\wedge$ $(\forall j \in Cld_i : change_j = false)$)
      $change_i := false$;
    else
      /* i を根とする部分木のなかで値が交換されている */
      $change_i := true$;
  else
    $reset_i := true$; **RESET**;
  $col_i := col_{P_i}$;

for leaf processes
$H_3 :: col_{P_i} \neq col_i$
→ **if** ($reset_{P_i} = false$)
    $reset_i := false$; **HEAPIFY**;
    $change := false$;
  else
    $reset_i := true$; **RESET**;
  $col_i := col_{P_i}$;

図 **4.1** プロトコル $\mathcal{HPP}$ (プロセス i)

(変数)

col_i : 論理型変数
out_i : 出力変数
w_i : ヒープを構築するための作業変数
r_i: 値の交換後，子へ返す値
$change_i$: i を根とする部分木で作業変数の値の変化の有無を示す論理型変数
$reset_i$: 手続き **RESET** を行うかどうかを示す論理型変数

RESET:
  $out_i := w_i$; $w_i := in_i$;
  $r = \perp$; $change_i := true$;

HEAPIFY:
for root processes
  $max_i := \max\{w_j | \ j \in Cld_i\}$;
  if ($w_i < max_i$)
    $r_i := w_i$; $w_i := max_i$; /* 値の交換 */
  else
    $r_i := \perp$

for internal processes
  if ($w_i = w_{P_i} \ \wedge \ r_{P_i} \neq \perp$)
    /* 親が自分と値の交換をした */
    $w_i := r_{P_i}$;
  $max_i := \max\{w_j | \ j \in Cld_i\}$;
  if ($w_i < max_i$)
    $r_i := w_i$; $w_i := max_i$; /* 値の交換 */
  else
    $r_i := \perp$

for leaf processes
  if ($w_i = w_{P_i} \ \wedge \ r_{P_i} \neq \perp$)
    $w_i := r_{P_i}$;
  $r_i := \perp$

図 **4.2** プロトコル $\mathcal{HPP}$ の手続き **RESET**, **HEAPIFY** (プロセス i)

(手続き)
RESET (図 4.2)
すべてのプロセスで作業用変数の値の交換がなくなった (ヒープ順序づき木が構成された) ときに，リセットをかけるために根プロセスが実行を開始する手続き．各プロセス i は，

作業用変数 w_i の値を出力変数 out_i に代入し，入力変数 in_i の値を作業用変数に代入する．

HEAPIFY (図 4.2)
ヒープ順序を実現する手続き．親プロセスと子プロセス間での値の交換を行う．各プロセス i は，すべての子プロセスの作業用変数を読み，自分の作業用変数の値よりも大きなものがある場合，その最大値を自分の作業用変数に代入する．また，交換前の自分の作業用変数の値を r_i に代入する．なお，値の交換が起こらなかった場合は，r_i に特別な記号 $\perp$ を代入する．子プロセスは，親プロセスの r を読み，その値が自分の作業用変数と同じ値の場合に，値の交換が起きたと認識し，親の r の値を自分の作業用変数に代入する．

定理 4.1 プロトコル $\mathcal{HPP}$ は，安定時間 $O(h)$，領域計算量 $O(K)$ の自己安定ヒープ順序づき木構成プロトコルである． □

5 結論

本稿では，木ネットワークでヒープ順序を実現する自己安定プロトコルを提案した．このプロトコルは隣接間同期化プロトコルを利用して，安定時間 $O(h)$，領域計算量 $O(K)$ でヒープ順序を実現する．ここで，h は木の高さ，K は入力値のサイズを表す．これは既知の結果と比べ，安定時間，領域計算量を共に改善している．

謝辞 有益な御討論を頂きました奈良先端科学技術大学院大学情報科学研究科情報論理学講座の皆様に感謝致します．本研究の一部は，文部省科学研究費補助金 (特定研究 (B)(2) 10205218)，および，中部電力基礎技術研究所研究助成による．

参考文献

[1] L. Alima: "Self-stabilizing max-heap," *Proc. of the 4th Workshop on Self-stabilizing Systems*, pp.94-101, 1999.

[2] B. Bourgon, A. Datta: "A self-stabilizing distributed heap maintenance protocol," *Proc. of the 2nd Workshop on Self-Stabilizing Systems*, pp.5.1-5.13, 1995.

[3] E. Dijkstra: "Self-stabilizing systems in spite of distributed control," *CACM*, 17,11, pp.643-644, 1974.

[4] C. Johnen, L. Alima, A. Datta, S. Tixeuil: "Self-stabilizing neighborhood synchronizer in tree networks," *Proc. of ICDCS*, pp.487-494, 1999.

[5] N. Lynch: "Distributed Algorithms," *Morgan Kaufmann*, 1996.

[6] M. Raynal, J. Helary: "Synchronization and Control of Distributed Systems and Programs," *John Wiley and Sons*, 1990.

[7] G. Tel: "Introduction to Distributed Algorithms," *Cambridge university press*, 1994.