社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2005－AL－103 (6)
2005／11／11

根付き木ネットワーク上において局所情報のみで負荷分散を実現する自己安定アルゴリズムについて

片山 喜章† 増澤 利光†† 和田 幸一†

† 名古屋工業大学大学院工学研究科情報工学専攻
〒 466-8555 名古屋市昭和区御器所町
†† 大阪大学大学院情報科学研究科
〒 560-8531 豊中市待兼山町 1-3
E-mail: †katayama@elcom.nitech.ac.jp

あらまし 自己安定アルゴリズムとは，任意のネットワーク状況からアルゴリズムの実行を開始しても，やがてあらかじめ与えられたネットワーク状況に到達する（解を求める）アルゴリズムである．本稿では，木ネットワーク中の各プロセスに任意の負荷（整数）が与えられると，やがて任意のプロセス間の負荷の差が高々1（1-balancing）になる自己安定アルゴリズムを提案する．また，提案アルゴリズムは負荷を表す情報のみで1-balancingを実現した．

キーワード 分散アルゴリズム，自己安定，木ネットワーク，負荷分散，1-balancing

On a self-stabilizing algorithm with local information on rooted tree networks

Yoshiaki KATAYAMA†, Toshimitsu MASUZAWA††, and Koichi WADA†

† Department of Computer Science and Engineering, Graduate School of Engineering, Nagoya Institute of Technology
Gokiso-cho, Nagoya, Aichi, 466-8555
†† Graduate School of Information Science and Technology, Osaka University
1-3, Machikaneyama, Toyonaka, Osaka, 560-8531
E-mail: †katayama@elcom.nitech.ac.jp

Abstract A elf-stabilizing algorithm is one of the distributed algorithms that can reach pre-defined legal network configuration starting from arbitrary initial network configuration. We propose a self-stabilizing algorithm for load balancing on rooted tree networks. It can achive that the difference of load values between any two processes is at most 1 (1-balancing) with only local information.

Key words distributed algorithm, self-stabilizing, rooted tree network, load balancing, 1-balancing

1. はじめに

複数のプロセスが相互に通信リンクで接続されたネットワークを分散システムという．分散システム上で，プロセスが互いに情報を交換することで与えられた問題を解くアルゴリズムを，分散アルゴリズムという．通常の分散アルゴリズムでは，実行開始時のネットワーク状況（大域状況）を仮定する．一方，Dijkstra が [1] で提案した自己安定アルゴリズム（self-stabilizing algorithm）は，任意の初期大域状況から実行を開始しても，やがて問題を解く（解状況に到達する）分散アルゴリズムである．この性質により自己安定アルゴリズムは，プロセスでの変数やプログラムカウンタの破壊，通信メッセージの改変などの一時的な故障（一時故障:transient fault）が生じても，やがて解状況に到達可能（再安定）であり，任意の一時故障に対して故障耐性を持つ．そのため，長期間に渡りネットワーク状況を安定に保ち，一時故障に対して対応する必要がある分散システムの運用に適したアルゴリズムである．なお，自己安定

アルゴリズムについては文献[2]に詳しい.

負荷分散(load balancing)問題とは,あるタスクが複数のプロセス上で並行に行なわれる際,その負荷(タスクの量)がすべてのプロセスに対して均等になるよう分散させる問題である.例えば,WWWや電子メールサービスなど,多くの要求が集中するサービスに対して複数のサーバを用意し,要求を各サーバに適切に分散させる場合などに負荷分散技術が用いられている.この例の場合,一旦特定のプロセス(ホスト)がすべての要求を受け付け,それを適切に配分する「集中制御型」で実現することが多い.一方,ネットワーク中に存在するプロセスで個別にサービスに対する要求が生じる場合もある.例えばP2Pサービスにおける資源の検索要求などがこれにあたる.この例では,各プロセスはサービス要求を出すクライアントであると同時に,サービスを提供するサーバでもあるため,前述の例のような集中制御型は難しく,分散的解決が望まれる.負荷分散問題の分散的解決手法はいくつか知られているが,そのうち自己安定アルゴリズムによる解決は,文献[3],[4]などがあげられる.文献[3]では,任意のネットワーク上で各プロセスの持つ負荷の最大差がネットワークの直径(Δ-balancing:*Delta*はネットワークの直径)となる負荷分散アルゴリズムが提案されている.また文献[4]では,根付き木ネットワークにおいて任意のプロセス間の負荷の差が高々1になる(1-balancing)負荷分散アルゴリズムが提案されている.

本稿では,根付き木ネットワーク上の各プロセスに与えられる負荷が整数で表現されるとき,任意の初期大域状況からやがて各プロセスの持つ負荷の差が高々1になる自己安定アルゴリズムを提案する.提案アルゴリズムは分散的手法で負荷分散を実現しており,すべての要求を一旦集める集中制御プロセスは必要ない.また文献[4]と比べた場合,[4]の手法では,あるプロセスの先祖(根に近いプロセス)に負荷の差が1の親子組が存在するかどうかという,ある種の大域情報の伝搬を必要としている.一方,提案プロトコルは大域情報を利用することなく隣接プロセス間の負荷の値(出力変数)だけ,つまりより純粋な局所情報のみから動作を決定しており,[4]と比べて局所的な性質を持つといえる(注1).つまり負荷分散問題という,各プロセスが任意の値を持つ状態からネットワークの全プロセスに対して決められたある大域状況を実現するような問題を,局所的な情報のみで解決している点で提案アルゴリズムは非常に興味深いと考える.

本稿の以降の構成は以下の通りである.2.節で提案アルゴリズムのモデルについて述べる.続く3.節で提案アルゴリズムについて説明し,4.節でアルゴリズムの正当性の証明を行なう.最後に5.節でまとめと今後の課題について述べる.

(注1):アルゴリズム中で親子関係を利用しているが,これは根付き木が自然に持つ情報であり,大域情報とみなしていない.

2. 諸定義

本節では,本稿で扱うネットワーク,プロトコル,問題などを定義する.

2.1 ネットワークとプロセス

本稿では,n個のプロセスが通信リンクで互いに接続された,根付き木ネットワークNを扱う.一般にプロセスと通信リンクからなるネットワークは,プロセスを頂点,通信リンクを辺と見ることで自然にグラフで表現できるので,グラフに対する用語や記法をネットワークに対しても用いる.

ネットワークNは$N=(P,L)$で表され,Pをプロセス集合,Lをプロセス間の通信リンクの集合とする.各プロセスは相異なる識別子を持つものとし,それぞれ$P=\{r,1,2,\cdots,n-1\}$と表し,rのプロセスを根プロセスとする.以下ではプロセスとプロセスの持つ識別子を区別せず,単に識別子で表現する.プロセスi,j間の通信リンクは$(i,j)(\in L)$と表し,i,jは互いに隣接しているという.

各プロセスは自分の識別子,隣接プロセスの識別子の集合N_i,およびN_i中のどれが親プロセス,子プロセス(集合)かを知っているものとし,それぞれparent_i, $\mathrm{Child}_i(=\{\mathrm{Child}_i(0),\mathrm{Child}_i(1),\cdots\})$で表す.また,$N_i$の要素は固定された全順序を持つとする.各プロセス$i$は(整数)変数$\mathrm{load}_i$を持ち,これを負荷(load)とよぶ.

隣接プロセス間の通信は,互いの内部状態(識別子や内部変数)を直接参照することができる状態通信モデルを仮定する.

2.2 スケジュールと実行

各プロセスiの状態(変数などの内部状態)をq_iとするとき,$c=(q_r,q_1,\cdots,q_{n-1})$を$N$のネットワーク状況とよぶ.また,$N$の取り得るすべてのネットワーク状況の集合を$C$と表す.つまり,プロセス$i$の取り得るすべての状態を$Q_i$とするとき,$C=Q_r\times Q_1\times\cdots\times Q_{n-1}$となる.

プロセスの部分集合を$S\subseteq P$とする.あるネットワーク状況$c_i\in C$で,Sに属するプロセスが同時にアルゴリズム$\mathcal{A}$の1原子動作を実行することによってc_{i+1}になったとき,$c_{i+1}=c_i(S,\mathcal{A})$と表す.

[定義1] (スケジュールと実行)空でないプロセス集合の無限系列をスケジュールと呼ぶ.アルゴリズム$\mathcal{A}$,スケジュール$T=S(0),S(1),\cdots$について,ネットワーク状況の無限系列$E=c_0,c_1,\cdots$が$c_{i+1}=c_i(S(i),\mathcal{A})(0\leq i)$を満たすとき,$E$を「初期状況$c_0$,スケジュール$T$に対するアルゴリズム$\mathcal{A}$の実行」とよび,$E(\mathcal{A},T,c_o)$と表す. □

[定義2] (公平なスケジュール)スケジュールTにすべてのプロセス$i\in P$が無限回現れるとき,Tは公平であるという. □

本稿では公平なスケジュールのみを対象とし,以降,単

にスケジュールと呼ぶ.

スケジュールによって選ばれたプロセスは，1 原子動作のみを行うことができる．スケジュール T が選び出すプロセス数と 1 原子動作の違いによりいくつかのモデルが考えられる．本稿では，以下のモデルを扱う（D デーモンと呼ぶ）.

- プロセス数:任意の $t(t \geq 0)$ について $|S(t)| \geq 1$
- 原子動作:全隣接プロセスから内部状態を読み込み，自分の内部状態を変更

2.3 自己安定

$\mathcal{LE} \subseteq C$ を，N のネットワーク状況の任意の集合とする．次の (1)，(2) の条件を満たすとき「アルゴリズム $\mathcal{A}$ は $\mathcal{LE}$ に関して自己安定である」といい，$SS(\mathcal{A}, \mathcal{LE})$ と書く．また，$SS(\mathcal{A}, \mathcal{LE})$ が成立するとき，$\mathcal{LE}$ を「アルゴリズム $\mathcal{A}$ に関して正当な状況」という．ただし，$\mathcal{A}$ が明らかな場合，単に正当な状況という.

（1） **到達可能性**: 任意のネットワーク状況 $c \in C$ と任意のスケジュール T に対し，cから始まるスケジュール T によるアルゴリズム $\mathcal{A}$ の実行 $E(\mathcal{A}, T, c)$ に，$\mathcal{LE}$ に含まれるネットワーク状況が現れる．つまり，$E(\mathcal{A}, T, c) = c_0, c_1, \cdots$ とするとき，$c_i \in \mathcal{LE}$ なる $i \geq 0$ が存在する.

（2） **閉包性**: $\mathcal{LE}$ 中の任意のネットワーク状況を c，プロセスの任意の集合を S とする．このとき，$c' = c(S, \mathcal{A})$ が $\mathcal{LE}$ に属する.

すなわち，任意の初期状況から開始しても，任意の公平なスケジュールによるアルゴリズムの実行は，有限時間内に正当な状況に到達し，一度正当な状況に達すると，それ以降の状況は正当な状況である（正当な状況で安定する）.

2.4 負荷分散問題

本稿で扱う負荷分散問題（LBP）を定義する.

[定義 3]（負荷分散問題（LBP））ネットワーク中の各プロセス i に，任意の（整数の）負荷 ($load_i$) が与えられている時に，任意の二つのプロセス間の負荷の差をある値 k 以下にする問題を，負荷分散問題（LBP）という．またその時の解状況を k-balancing 状況と呼び，以下のように定義する.

$$\forall i, j : |load_i - load_j| \leq k. \qquad \square$$

ここで，LBP を解く自己安定アルゴリズム（$SS\mathcal{LB}$）が満たすべき条件として，以下の 2 つがあげられる [4] (注2).

（1） **制約**: ネットワーク中の負荷の総和は変化しない.

（2） **収束**: 任意のネットワーク状況からアルゴリズムの実行を始めても，有限時間内に前もって与えられた任意の整数 k に関して k-balancing 状況に到達する.

本論文では，分散的手法によって 1-balancing を実現する負荷分散自己安定アルゴリズム $\mathcal{LB}$ を提案する.

(注2)：[4] では，もうひとつ "分散性" が条件として与えられているが，一般的な $SS\mathcal{LB}$ の条件としては本稿では含めていない.

3. 自己安定負荷分散アルゴリズム $\mathcal{LB}$

本節では，負荷が整数の場合に 1-balancing を実現するアルゴリズム $\mathcal{LB}$ について述べる．まずアルゴリズムの実行モデルについて説明し，続いてアルゴリズムのアイデアとその詳細について説明する.

3.1 $\mathcal{LB}$ の実行モデル

$\mathcal{LB}$ においては集中制御を行なうプロセスを仮定せず隣接プロセス間での局所的な負荷調整のみによって，1-balancing を実現している．そのためには負荷調整（隣接プロセス間での負荷演算）を行なう際に，一対一で実行させる必要がある．例えば，3 つの連続する隣接プロセス i, j, k を考えた時，負荷調整は i, j 間，あるいは j, k 間のいずれか一方で行なう必要があり，もし同時に行なわれると負荷の総和が変化してしまう，つまり制約条件を満たさなくなる可能性がある．したがって，各プロセスで隣接プロセスをひとつ選び，かつそのプロセスでも自分が選ばれた状態でのみ，負荷調整を行なわなければならない.

本稿で仮定するスケジュールである D デーモンの下では，ネットワーク中の任意の複数のプロセスが動作を行なうため，上述のような実行を可能にするためには，各プロセス間での実行の制御を行なう必要がある．しかし，この制御を考慮してアルゴリズムを記述すると大変繁雑になり，本質の理解が困難になる．したがって本稿では，具体的な制御方法は別の機会に議論することとし，アルゴリズムに関する説明は，以下のような実行モデル上で行なうこととする(注3).

D デーモンに対して，以下のような付加条件を与える.

- 任意の $t(t \geq 0)$ について，$S(t)$ は独立点集合である.
- $i(\in S(t))$ は，アルゴリズムにしたがいその子プロセスのうちのひとつ $j(\in \mathrm{Child}_i)$ との間で，演算を行なう．このとき j では，i によるアルゴリズムの実行にしたがって適切な演算が行なわれ，かつ i，j での演算は，1 原子動作で実行される.

つまり，スケジューラによって選ばれたプロセス i が，演算相手として子プロセス j を選択すると，i，j は互いに 1 原子動作としてアルゴリズムにしたがった演算を行なう．また，スケジューラに選ばれるプロセスは互いに隣接しないため，ひとつのプロセスが複数のプロセス相手に演算を行なうことはない．なお，プロセス i, j で実行される演算（負荷計算）については，アルゴリズムの詳細（3.3 節）で説明する.

3.2 アルゴリズム $\mathcal{LB}$ のアイデア

負荷分散を実現するために，3.1 節での実行モデルにしたがい，各プロセス（i とする）は，隣接プロセス中に自プロセスの負荷との差が 2 以上のものが存在すればそのプ

(注3)：具体的な制御方法は，フラグを用いたハンドシェークを基本とした制御（例えば文献 [5] を利用）で実現可能である.

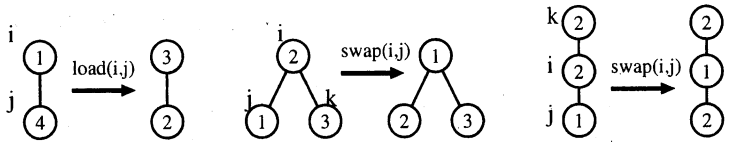

図 2　ルール R1,R2,R3 の適用状況例

Fig. 2　Examples of configurations adopting R1,R2,R3.

[R1] $\exists j \in \mathrm{Child_i}, |\mathrm{load_i} - \mathrm{load_j}| \geqq 2 \longrightarrow lb(i,j)$

[R2] $\exists j \in \mathrm{Child_i}, \exists k \in \mathrm{N_i}, |\mathrm{load_i} - \mathrm{load_k}| \leqq 1$ かつ $|\mathrm{load_j} - \mathrm{load_k}| \geqq 2 \longrightarrow swap(i,j)$

[R3] $\exists j \in \mathrm{Child_i}, \mathrm{load_i} = \mathrm{load_{parent_i}}$ かつ $|\mathrm{load_i} - \mathrm{load_j}| = 1 \longrightarrow swap(i,j)$

図 3　アルゴリズム $\mathcal{LB}$

Fig. 3　Algorithm $\mathcal{LB}$

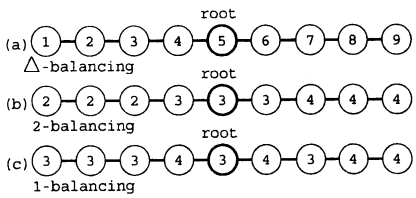

図 1　負荷分散の例

Fig. 1　Examples of the balancing configuration

ロセスとの間で負荷調整を行なう．つまり，互いの負荷を加算し 2 で割ったものを新たな負荷とする．このとき，負荷の和が偶数の場合は両者が同じ負荷を持つ．一方，奇数の場合は 2 で割り切れず新しい負荷が偶数と奇数になり負荷の差が 1 のプロセス組となる．

しかし，これだけでは 1-balancing は達成できない．例えば図 1 (a) の状態（Δ-balancing）からは負荷調整が行なわれず，1-balancing を達成できない．そこで，例えば常に奇数（偶数でも同様）を根の方へ集める戦略を取ったとすると，図 1 (b) の状態（2-balancing）になり，これも 1-balancing を達成できない．

1-balancing がうまく実現できないのは，同じ負荷がネットワーク中のある部分に固まってしまい，互いに差が 2 以上ある負荷の存在を認識できない点が問題である．そこで，我々は 1-balancing における負荷は 2 種類であり，それらを「かき混ぜる」点に着目した．つまりかき混ぜることでネットワーク中にある差が 2 以上である負荷の存在を互いに認識可能とするわけである．

提案アルゴリズムでは，各プロセスの持つ負荷が，できるだけ隣接間で異なるように配置する．例えばネットワーク中に 3 種類の負荷 x,y,z（ただし，$|x-y|=1, x \neq y \neq z$ とする）が存在したとする．このとき，根から任意の葉への経路上に同じ負荷が連続しないよう配置する（例えば $x,y,x,z,y,\cdots$）．このように配置することで，差が 2 以上の負荷の存在を認識できるプロセスが必ず存在し（図 1 (b) を排除），負荷調整を直接行なう，あるいは負荷調整が生じるよう負荷交換を行なう（図 1 (a) を排除）ことが可能となる．

結果として，1-balancing が実現されると，ネットワーク中には高々2 種類の負荷 x,y のみが存在し，かつそれらは根から任意の葉への経路上に交互に配置される．しかしネットワーク中の負荷の総和によっては，交互配置を完全に実現することが無理な場合もある．その場合は，根から任意の葉への経路について，根から順に $x,y,x,y,\cdots$ と配置していき，あるプロセス（i とする）以降，すべて x（または y）であるような配置も許す．ただし，このとき i 以降，葉プロセスまではすべて同じ負荷となるよう配置する（図 1 (c)）．

以下では，負荷分散を行なう自己安定アルゴリズム $\mathcal{LB}$ の詳細について述べる．

3.3　アルゴリズム $\mathcal{LB}$

負荷分散を実現する自己安定アルゴリズム $\mathcal{LB}$ における負荷に対する操作（負荷計算）を定義する．

[定義 4]（負荷調整と負荷交換）アルゴリズム $\mathcal{LB}$ における負荷計算は，以下のいずれかである．

負荷調整： $lb(i,j)$

互いに隣接するプロセス i と j の間で負荷分散を行なう．プロセス i が j の親プロセスとするとき，手続きは以下の通り．

$$\mathrm{load_i} = \lceil \mathrm{load_i} + \mathrm{load_j} \rceil \text{(プロセス } i \text{での演算)}$$

$$\mathrm{load_j} = \lfloor \mathrm{load_i} + \mathrm{load_j} \rfloor \text{(プロセス } j \text{での演算)}$$

負荷交換： $swap(i,j)$

互いに隣接するプロセス i と j の間で負荷の交換を行なう．プロセス i が j の親プロセスとするとき，手続きは以下の通り．

$\mathrm{load_i} = \mathrm{load_j}$(プロセス i での演算)

$\mathrm{load_j} = \mathrm{load_i}$(プロセス j での演算)

□

アイデアで述べた負荷の配置を実現するために，以下の3つのルールを用いる．ただし，各ルールはプロセス i で適用されるもので，i が j と負荷計算を行なうとする．

[R1] j が子，かつ j と自分 (i) の負荷の差が2以上の場合，i と j で負荷調整 ($lb(i,j)$) を行なう．(例：図2 (1))

[R2] j が子，かつ自分 (i) と負荷の差が2以上の隣接プロセスが存在せず，かつ j との負荷の差が2以上の隣接プロセスが存在する場合，i と j で負荷交換 ($swap(i,j)$) を行なう．(例：図2 (2))

[R3] j が子，かつ自分 (i) と負荷の差が2以上の隣接プロセスが存在せず，かつ j との負荷の差が2以上の隣接プロセスが存在せず，かつ親と自分 (i) の負荷が等しく，かつ自分 (i) と j の負荷の差が1の場合，i と j の間で負荷交換 ($swap(i,j)$) を行なう．(例：図2 (3))

直観的には，R1によって負荷が調整され，R3によってネットワーク中の負荷の値を混ぜる．さらにR2によって負荷の差が2以上のものを互いに認識させる（隣合わせる）．

なお各ルールには優先順位が付けられており，R1が最も高くR3が最も低い．つまり，ある隣接プロセスとの間で複数のルールが適用可能な場合，優先順位の高いルールが適用されるものとする．

アルゴリズムを図3にあげる．アルゴリズム中の $A \longrightarrow B$ は，「述語 A が成立する場合，B を実行する」ことを意味する．

4. 正当性の証明

負荷分散問題を解く自己安定アルゴリズムの正当な状況 $\mathcal{L}_{\mathcal{LB}}$ を定義する．

[定義5] （正当な状況）以下のふたつの条件を満たすネットワーク状況を，負荷分散問題を解く自己安定アルゴリズムに関して正当な状況といい，$\mathcal{L}_{LB}$ と表す．

LE1: ネットワーク中に高々二種類の負荷 x, y が存在し，$|x-y|=1$ を満たす．

LE2: 根 (i_1 とする) から任意の葉 (i_k とする) への経路 $(i_i, i_2, \cdots, i_{k-1}, i_k)$ において，

$\exists s(1 \leqq s \leqq k-1)[\mathrm{load_1} = \mathrm{load_3} = \cdots = \mathrm{load_{s-1}} = x$ かつ $\mathrm{load_2} = \mathrm{load_4} = \cdots = \mathrm{load_s} = y$ かつ $\forall t(s \leqq t), \mathrm{load_{i_t}} = \mathrm{load_{i_{t+1}}}]$

が成立する．s が奇数の場合も同様． □

つまり，正当な状況におけるネットワーク上での負荷（二種類）の直観的な配置は，根から任意の葉への経路で根から順に負荷を比べた時，あるプロセスまで二つの負荷が交互に現れ，それ以降はすべて同じ負荷となるような状況である．

以後の証明では，アルゴリズム実行中に一時故障が生じないと仮定する．

アルゴリズム $\mathcal{LB}$ より，以下の補題1がすぐにいえる．

[補題 1] （閉包性）正当な状況 $\mathcal{L}_{LB}$ では，どのルールも適用されない． □

[補題 2] どのルールも適用されない状況において，任意のプロセス i とその子孫 k について，$|\mathrm{load_i} - \mathrm{load_k}| \leqq 1$ である．

[証明] どのルールも適用されない状況において，$|\mathrm{load_i} - \mathrm{load_k}| \geqq 2$ である i の子孫 k が存在すると仮定（背理法）．i から k の経路において，R1が適用されないことより少なくとも三種類の負荷が存在する．いま，三種類の負荷 $x(=\mathrm{load_i}), y(=x+1), z(=x+2=\mathrm{load_k})$ が存在すると仮定．R1が適用されないことより，x と z は互いに隣接しない．R3が適用されないことより，x が連続した後に y，または y が連続した後に x あるいは z が出現することはない．つまり，x と z が y を挟んで出現することとなり，必ずR2が適用される．よって矛盾．負荷が四種類以上の場合も同様． □

[補題 3] どのルールも適用されない状況は，正当な状況 $\mathcal{L}_{LB}$ である．

[証明] どのルールも適用されない状況を考える．ネットワーク中に $|\mathrm{load_s} - \mathrm{load_u}| \geqq 2$ なる負荷を持つ二つのプロセス u，s が存在すると仮定（LE1に関する背理法）．補題2より，u, s は互いに祖先・子孫の関係ではない．ここで u, s の最も近い共通の祖先プロセスを t とする．補題2より，$|\mathrm{load_t} - \mathrm{load_s}| \leqq 1$ かつ $|\mathrm{load_t} - \mathrm{load_u}| \leqq 1$．したがって，$|\mathrm{load_s} - \mathrm{load_u}| \leqq 2$ がいえる．一般性を失うことなく $\mathrm{load_s} < \mathrm{load_u}$ と仮定すると，$\mathrm{load_t} = \mathrm{load_s} + 1 = \mathrm{load_u} - 1$ となる．ここで，プロセス t から u までの経路に注目すると，経路中のどのプロセスもR3が適用されないことより，経路における t の子プロセス t_u の負荷は $\mathrm{load_{t_u}} = \mathrm{load_t} + 1$ である．同様に，プロセス t から s までの経路に注目すると，経路中のどのプロセスにもR3が適用されないことより，経路における t の子プロセス t_s の負荷は $\mathrm{load_{t_s}} = \mathrm{load_t} - 1$ である．つまり，t の二つの隣接プロセス t_s, t_u の負荷の差が2となり，R2が適用される．これは矛盾．一方，LE2については，もしLE2が満たされていないと必ずR2が適用されることよりすぐにいえる． □

さらに以下の補題で，公平なスケジュールによりいつかはすべてのルールが適用できない状況（正当な状況）に到達することをいう．

[補題 4] ルールR1は有限回しか適用されない．

[証明] ネットワーク中に存在する全ての負荷を降順に並べた系列を考える．アルゴリズムにしたがいR1が適用されると，負荷調整によって系列は辞書式順序で必ず小さくなる．プロセス数および負荷は有限であることより，いつかはR1が適用されなくなる． □

[補題 5] ルール R2 が適用されると，その後または同時に必ず R1 が適用される．

[証明] R2 が適用されるプロセスを i，i での R2 適用における負荷交換の相手（子プロセス）を j とする．また j の負荷との差が 2 以上のプロセスを k とする．

i で R2 が適用され，$swap(i,j)$ が実行された直後の状況を考える．**case1:** i,k で R1 が適用される場合

仮定より，R2 適用前の $|\mathrm{load_j} - \mathrm{load_k}| \geqq 2$ より，k において i との間で R1 が適用される．

case2: i,k で R1 が適用されない場合

$\mathrm{load_k}$ の値が変化したため R1 が適用されなかった．これは，以下の場合に分けられる．

case2.1: k の親において，k との間で R1 が適用された．

case2.2: k の親において，k との間で R2 が適用された．

case2.3: k の親において，k との間で R3 が適用された．

ここで case2.3 の場合，k，$\mathrm{parent_k}$ において R3 が適用されたことより，新しい負荷の値は $\mathrm{load_k}+1$ または $\mathrm{load_k}-1$ のいずれか．新しい $\mathrm{load_i}$ の値との関係により，k と i の間で R1 が適用されるか，または R2 が適用されることはすぐにいえる（R3 は適用されない）．つまり，case2.2 が永遠に起こり続けないことを証明すればよい．

case2.2 の直後の状況を考える．このとき，k において，$\mathrm{load_{parent_k}}$ と $\mathrm{load_i}$ の差が 2 以上になっている．つまり，直観的には差が 2 以上の値のペアが，順に根に向かって移動していると捉えることができる．これを繰り返すと，やがて R2 における i に対する k が根プロセスである状況に到達し，それ以上 k とその親の間で R2 が適用できなくなり，R1 が適用される． □

補題 4，5 より，ルール R2 は有限回しか適用されないことがわかる．以降，ルール R1,R2 が適用されなくなったと仮定する．

[補題 6] アルゴリズムの実行において，ある時点以降，ルール R3 が適用されなくなる．

[証明] 根から任意の葉への経路を考える．経路中，負荷が同じ隣接プロセスペアの重みを 1，それ以外を 0 とし，その経路の根から葉への重みの系列を考える．経路中のプロセスペアに R3 が適用されると，重みの系列は辞書式順序において必ず減少する．プロセス数および負荷が有限であることより，R3 は有限回しか適用されない． □

補題 4，5, 6 より，以下の補題（到達可能性）がいえる．

[補題 7] （到達可能性）アルゴリズムの実行において，いずれのルールも適用されない状況に到達する． □

以上より以下の定理を得る．

[定理 1] アルゴリズム $\mathcal{LB}$ は，任意のプロセスの持つ負荷の差が高々1 になる，負荷分散問題を解く自己安定アルゴリズムである． □

5. おわりに

本論文では，負荷が整数で表される場合に，任意の二つのプロセス間の負荷の差を高々 1（1-balancing）にする，負荷分散自己安定アルゴリズムを提案した．本アルゴリズムは，1-balancing を実現するために必要な情報が隣接プロセスの負荷の値（出力変数）と根付き木ネットワークが自然に持つ親子関係の情報のみであり，他の大域情報を一切必要としない．既存の負荷分散のための分散アルゴリズムに比べより純粋に局所的な情報のみで問題を解くという意味で興味深いと考える．

今後の課題のとして，アルゴリズムの解析（安定するまでに必要な負荷計算の回数など），負荷分散を実現するために必要な情報量の下界の解明（親子関係を使わずに実現可能かどうか）などがあげられる．

謝辞 本研究の一部は，日本学術振興会科学研究費補助金・若手（B）（課題番号 16700010）基盤研究（B）(2)（課題番号 15300017）および基盤（C）（課題番号 17500036），ならびに H17 電気通信普及財団の研究助成によるものである．

文　献

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

[2] S.Dolev. "Self-Stabilization" "MIT Press", 2000.

[3] W.Aiello, B.Awerbuch, B.Maggs and S.Rao. "Approximate load balancing on dynamic and asynchronous networks" *Proc. of the 25th Annual ACM Sympo. on Theory of Computing: 632-641*, 1993.

[4] A.Arora. "A foundation of fault-tolerant computing(Chapter 7)" *Ph.D. Dissertation, University of Texas at Austin*, 1992.

[5] T.Herman, T.Masuzawa. "Self-Stabilizing Agent Traversal" *Proc. of 5th Int. Workshop on Self-Stabilizing Sys. (WSS2001) (LNCS2194): 152-166*, 2001.