2Q-04 自己安定アルゴリズムの可視化システムの開発

山下 悟史　　角川 裕次　　阿江 忠

広島大学工学部

1. はじめに

ネットワークの発展に伴い，今日ではシステムの分散化が進んでいるが，その大規模複雑化につれて，システム全体が常に正常であることは困難になってきている．このことから，分散環境における故障耐性に対する重要性が高まり，システムの一部に一時故障が生じても，自動的に元の正しいシステム状態に戻るアルゴリズム，つまり自己安定アルゴリズムに関する研究が盛んに行われている．

自己安定アルゴリズムを理解する上で重要なことは，分散システム内に属するノード (プロセッサ) の局所状態などの推移を把握しておくことである．しかし，ノード数が増えるにつれて，それらを全て把握しておくことは容易ではなくなる．このような場合には，分散システム内に属するノードとその局所状態を可視化することが有効な手段であると考えられる．

本研究では，代表的な自己安定アルゴリズムを可視化するシステムの開発を目的とする．また，これに初期状態の設定，動作ノードの選択，シミュレーションの再実行等の機能を付け加えることで，様々な角度からアルゴリズムの解析が行えるシステムを目指す．

2. 自己安定アルゴリズム

自己安定アルゴリズムとは，任意の種類の，そして任意の有限数の一時故障より有限時間内に自動回復できるアルゴリズムである．システムの初期化の必要はなく，自動的に正常なシステム状態へ遷移することができる．

本研究で扱う自己安定アルゴリズムでは，計算モデルとして以下のものを仮定する．

- 状態通信モデル
  各プロセスは，すべての隣接プロセスの局所状態を読むことができる．(遅延は生じない．)
- C デーモン
  システムには複数のプロセスが存在するが，それらの内の一つだけを選んで，1 ステップの動作を不可分で実行する．

自己安定アルゴリズムにおいて，あるプロセスが**特権** (privilege) を持つとは，局所変数と隣接プロセスの変数に関する述語が真となることを言い，C デーモンの下では特権を持つプロセスの集合の内の一つが選ばれ実行される．

3. 可視化システムが備えるべき機能

可視化システム作成の目的には，アルゴリズムの解析，デバッグという側面がある．そのため，一般的に可視化システムにはシミュレーションを操作する機能，つまり任意のステップでのシミュレーションの停止や再開，過去のステップに戻っての再実行といった機能が必要とされる．また，アルゴリズムに間違いがないかを検証するには様々なノード数のパターンでシミュレーションを実行する必要があり，ノード数の変更機能も不可欠である．

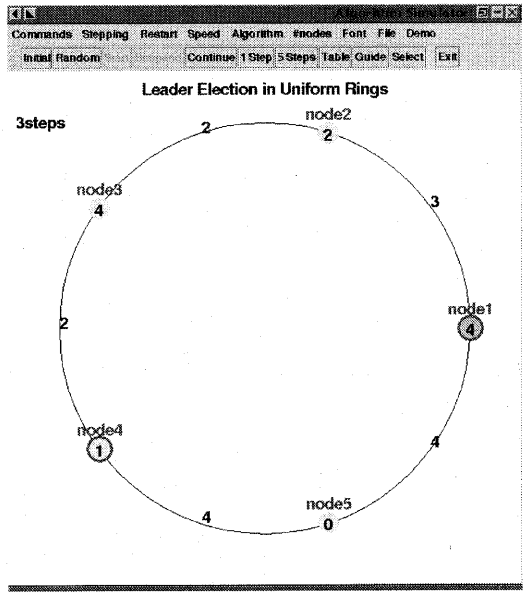

図 1: 動作中の画面例

次に，可視化の対象を分散アルゴリズムとした場合について考える．分散システムには多数のノードが存在し，これらが非同期に動作することが分散アルゴリズムの設計を困難にしている．そこで可視化システムには，分散システム内に属する全てのノードが現在どういう状況にあり，その後どのノードが動作したことでどういった状況に推移したのかを把握できる機能が要求される．

最後に対象を自己安定アルゴリズムとした場合について考える．自己安定アルゴリズムの特徴は任意の初期状態から出発しても有限数ステップで正常な状態に到達することにある．よって，可視化システムにはユーザの望む任意の初期状態を作り出し，各ステップで動作するノードを任意に選択できる機能が必要となる．また，シミュレーションの再現性がデバッグには有効であると考えられるため，同じ初期状態から出発して各ステップで同じノードを動作させる機能があるのが望ましい．さらに，アルゴリズムによっては，仮定するネットワーク形状や識別子の有無，許されるプロセス数が異なっており，これらも考慮する必要がある．

4. 可視化システムの実現

第 3 章での議論に基づき，本研究では自己安定アルゴリズムの可視化システムを実現した．本可視化システムは，Linux 上で Java (JDK 1.1.7) と Swing 1.0.3 を用いて作成した．よって，本システムを起動するには同様の環境のインストールを必要とする．以下では，本可視化システムの機能の詳細を順に説明する．

The Development of a Visualization System for Self-Stabilizing Algorithm
Satoshi Yamashita
Faculty of Engineering, Hiroshima University

4.1 シミュレーションの開始，中断，再開，停止

シミュレーションを開始すると，自動的に実行を繰り返し，アルゴリズムが正常な状態に収束すれば，そこで実行を停止する．また，アルゴリズムの解析を助けるために任意のステップでの中断，再開ができる．

4.2 シミュレーションのスピードの調節

シミュレーションのスピードを5段階の中から選択し，正常な状態に到達するまでのおおまかな流れを掴むことや，各ステップにおいてどのノードの局所状態がどのように推移したのかを詳しく見ることなどができる．

4.3 アルゴリズムとノード数の変更

これまでに文献 [1] から [5] で提案されている6つのアルゴリズムに対応しており，その中から選択し実行できる．またアルゴリズムによってはノード数に制約があるため，各アルゴリズムについてノード数のパターンをいくつか用意している．

4.4 フォントの変更

フォントのポイントサイズを5段階の中から選択でき，分散システム内に多数のノードが存在する場合にも画面内に表示できる．

4.5 実行ステップ数の選択

ある一定ステップ数分だけシミュレーションを実行して中断させることができ，選択したステップ分だけ後の状態を確認できる．

4.6 各ノードの局所変数の初期状態の設定

自己安定アルゴリズムの特徴の一つに，任意の初期状態から実行を開始しても必ず正常な状態に到達できるというものがあるが，可視化システム上で初期状態をランダムに生成するだけでは，ユーザの望む初期状態から実行しても同様の結果に到達するのかどうかという確認が行えない．そこで，各ノードの局所変数の初期状態については，ランダムに設定する方法と設定ファイルから読み込ませる方法のいずれかを選択できる．

4.7 動作するノードの選択

自己安定アルゴリズムでは，正常な状態に到達するまでは，複数のノードが特権を持つことが多く，C デーモンの下で動作するアルゴリズムでは，このような場合には任意のノードが一つ選ばれる．しかし，可視化システム上で動作ノードの選択を全てランダムで行ってしまうと，どのような動作ノードの系列でも最終的に有限数ステップで安定状態に到達することができるのかどうかという確認が難しい．そこで，各ステップで特権を持ったノードの中からどのノードが動作するかについては，ランダムに決定させる方法とユーザが自分で決定する方法のいずれかを選択できる．

4.8 シミュレーションの再実行

シミュレーション実行中，一定間隔ごとにそのステップの状態を保存しておくことで，そこからのシミュレーションの再実行ができ，過去の状態からの実行の流れを再確認できる．

4.9 実行履歴表の作成

シミュレーションと同時実行で，別のウィンドウに表形式で各ステップの情報として、ステップ数、そのステップで特権を持ったノードの集合、特権ノードの中から選ばれて動作したノード、動作ノードの局所変数の変更前、変更後の値を随時追加していくことで，現在のステップまでの実行の履歴表を作成することができる．

4.10 アルゴリズムの表示

実行中のアルゴリズムの補足説明のために，各ノードが特権を持つ条件，また特権を持った時の動作内容を表示させることができる．

4.11 シミュレーションの自動デモ表示

あらかじめ設定しておいたアルゴリズム，ノード数のパターンについて，自動でデモを表示することができる．

5. おわりに

本稿では，自己安定アルゴリズムの理解を助けるための可視化システムを提案した．また，アルゴリズムの解析，デバッグに有効だと思われる機能をシステムに盛り込むことで，様々な角度からシミュレーションを行えるようにした．

現状での問題点としては，

- ユーザが新たにアルゴリズムをシステムに組み込むことが困難である
- 有限数ステップでの収束や、無限ループに入ったかどうかの判定が機械的に行われず、ユーザの判断に委ねられている

といったことがある．これらの解決策として，分散システム等の検証，シミュレーションツールである Spin の利用が考えられる．Spin を用いれば，アルゴリズムの正当性の検証ができるだけでなく，アルゴリズムが不具合を含んでいて振動状態や無限ループに陥った場合に、そこに至るまでの過程を出力できるので、これを可視化できればどの部分のノードが問題を起こしているのかが分かりやすく、デバッグに役立つと考えられる。このように，本システムに Spin の出力結果を可視化できる機能を付け加えるなどして，本システムと Spin の連動を図ることが今後の課題である．

謝辞

本研究の一部は，文部省科学研究費補助金 (課題番号 11780229) の援助を受けている．

参考文献

[1] Edsger W. Dijkstra, "Self-Stabilizing Systems in Spite of Distributed Control", *Communications of the ACM*, Vol.17, No.11, pp.643–644 (1974).

[2] Shing-Tsaan Huang, "Leader Election in Uniform Rings", *ACM Transactions on Programming Languages and Systems*, Vol.15, No.3, pp.563–573 (1993).

[3] Sukumar Ghosh, "Binary Self-Stabilization in Distributed Systems", *Information Processing Letters*, Vol.40, No.3, pp.153–159 (1991).

[4] Su-Chu Hsu, Shing-Tsaan Huang, "A Self-Stabilizing Algorithm for Maximal Matching", *Information Processing Letters*, Vol.43, No.2, pp.77–81 (1992).

[5] Sumit Sur, Pradip K. Srimani, "A Self-Stabilizing Algorithm for Coloring Bipartite Graphs", *Information Sciences*, Vol.69, pp.219–227 (1993).