分散アルゴリズム

――多数のコンピュータが自律的に協調するアルゴリズムの理論――

角　川　裕　次
Hirotsugu KAKUGAWA

理工学部数理情報学科　教授
Professor, Department of Applied Mathematics and Informatics

1. はじめに

2019 年 4 月に数理情報学科に着任いたしました．計算機科学の 1 分野である，分散アルゴリズム（distributed algorithm）の基礎理論を専門としています．本稿ではその中でも特に，長年に渡って研究を進めてきている自己安定分散アルゴリズムを簡単に紹介します．

2. 分散システムとは

分散システムとは，複数の構成要素からなるシステム（系）であり，システムを集中制御するものが特になく，構成要素が互いに相互作用をしながら変化してゆく，あるいは変化しないような恒常性を実現するものです．そのようなものには様々なものがあり，多細胞の生物，蟻や鳥などの群れ，社会なども分散システムの例ですが，本稿では多数のコンピュータで構成されるネットワークを対象にして説明を進めます．具体例として，インターネット，無線センサネットワーク（WSN），IoT（Internet of Things），モバイルアドホックネットワークなどが挙げられます．また，オンラインショッピング，電子メイル，SNS，動画サイト，銀行のオンライントランザクションシステム，オンラインストレージなどは特に，日々の生活で欠かせない分散システム上のアプリケーションです．

今日では Raspberry Pi や Arduino など，高性能でソフトウエア開発が容易なワンボードマイコンや，それらに搭載可能な小型の無線通信ボードなどが低価格で入手できるようになり，分散システムは自分で構成できる身近なものになりつつあります．（私がこの分野の研究を開始した 30 年前とは随分と状況は変化してきました．）

3. 分散システムが持つ 3 つの性質

大規模な分散システムには，以下に説明する非同期性，局所性，情報伝達遅延性の 3 つの性質のために，後述の分散アルゴリズムの設計が困難となっています．

非同期性：動作の速度とタイミングがバラバラであることをいいます．全てのコンピュータは同じ処理速度とは限りませんし，通信速度もまちまちなのが普通です．特にネットワークが混んでいるときとそうでないときでは，通信速度が違うことが往々にして起きます．このようにネットワークの構成要素は，それぞれ随分と違った速度で動作するのが普通です．そして分散システムの規模が大きくなるほどこの傾向は顕著です．従って，大規模な分散システ

ムでは，全ての構成要素が足並み揃えて一定の処理速度で動作することは期待できません．

局所性：得られる情報は，ネットワークの近接に限られること，すなわち局所的であることをいいます．どのコンピュータも，通信路で直接的に接続されているコンピュータとは通信できます．しかしそれ以外のコンピュータとの通信は，いくつものコンピュータを経由して通信する必要があります．

情報伝達遅延性：情報の通信には時間がかかり，最新情報をすぐには得られないことをいいます．ネットワークの高速化が進んでいるとはいえ，通信にはいくらかの時間がかかります．上述の局所性より，離れたコンピュータの情報を入手しようとすれば情報はいくつものコンピュータを経由して伝達する必要があり，情報を得た瞬間でも，それはもう古い情報になっています．例えば，夜に星空を見上げて星を観察したとします．でもそれは，何光年もかかって地球に届いた光をその瞬間に観察しているだけであり，その瞬間の宇宙全体の状態を観察してはいないのと似ています．

4. 分散アルゴリズム

分散アルゴリズムとは，分散システムを動かすためのアルゴリズムのことをいいます．アルゴリズム（algorithm）とは，ある決まった計算や制御を行うための（抽象的な）処理手順のことをいいます．実際に動作するソフトウエアを作る際には，プログラムコードとして具体的に処理手順を詳細化します．ソフトウエアの基本設計書がアルゴリズムですから，アルゴリズムに間違いがあったり処理に手間がかかるものだと，その後をどう頑張っても正確かつ高速動作するソフトウエアは得られません．

分散アルゴリズムとは図 1 に示すように，それぞれのコンピュータで動作するアルゴリズムの集合体であり，互いに通信をしながら処理を進めてゆく点に特徴があります．しかし分散システムには上述の非同期性，局所性，情報伝達遅延性があるために，その設計は容易ではありません．さらに加えて，コンピュータの故障や通信障害も発生するとなれば，その難しさは更に増します．

このような困難さのもとで，どのようにして分散アルゴリズムを設計すれば良いのでしょうか．これが私の取り組んでいる研究テーマです．

難しさを具体的にイメージできるよう，以下の問題を考えてみて下さい．あるアイドルグループのファンクラブの会員の総数を調べたいとします．問題の詳細な設定は以下の通りです．ファンクラブは各地に地域事務局があり，それぞれの地域事務局はその地域の会員のリストを持っていて，その地域での会員の数を知っています．地域事務局はボランティアで運営されているため，担当者がすぐに対応できるとは限りません．さらに電話やインターネットは一切使えず，情報のやり取りに使える方法は郵便葉書に限定されています．配達が翌日の場合もあれば，もっとかかる場合もあり，何日かかるか事前に予想できません（非同期性）．地域事務局を統括する総事務局はなく，地域事務局の住所の一覧表すらありません．あなたが今知っているのは，居住している場所の地域事務局だけです．各地域事務局は隣の地域の地域事務局の住所しか知りません（局所

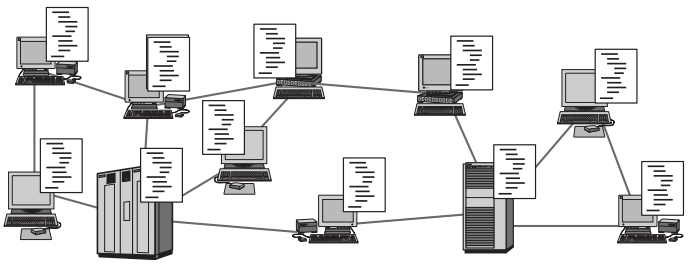


図 1　分散システムと分散アルゴリズム

性）．総会員数の調査にあたっては，他の地域事務局にお願いして，隣接する地域事務局に調査の葉書を出してもらわないといけないので結構面倒です．さらに，会員の総数を調べている最中にも入会や脱会があちこちの地域事務局で発生することに加えて，会員が引越しをすると所属する地域事務所の変更が発生します（情報伝達遅延性）．一体どのような方法で会員の総数を求めれば良いでしょうか．また，得られた数値をどう解釈すれば良いのでしょうか．（4 月 1 日午前零時の人数を全ての地域事務局で一斉に記録して報告するよう，葉書で予め調査依頼を伝えておく方法ではうまくゆきません．3 月末までに葉書がすべての地域事務局に届くとは限らないからです．従って，地域事務局ごとにまちまちの日時に記録を取るしか方法がありません．また，例えば総会員数 4096 名という数値が得られたとしても，それはいったいどの時点での数値と解釈すれば良いのでしょうか．）

このような問題でも十分に難しそうなのに，さらにコンピュータの故障や通信障害などが起きても正しく動作する仕組みを考えるのはとても大変そうです．

5. 自己安定分散アルゴリズム

大規模な分散システムで重要なのは故障耐性（fault-tolerance）や自律性（autonomy）です．なぜなら，機器の数が膨大になるとそのうちの一部が故障したり通信障害が発生するのは日常茶飯事であり，ひとつひとつ人手で復旧作業することは事実上不可能だからです．例えばインターネットでは，通信機器の更新・障害・撤去などがいつも世界中のどこかで行われています．そのたびにインターネット全体の機器，つまり世界中の機器を停止して再設定や再起動をするのは全く不可能であることは容易に想像できます．自己安定分散アルゴリズム（self-stabilizing distributed algorithm）とは，分散システムに故障耐性や自律性に優れた性質を実現して，上記の問題の解決を目指した分散アルゴリズムです．

5.1 定義

分散システム全体の状態を「状況」と呼ぶことにします．状況には，各コンピュータの状態（変数値など）と各通信路の状態（伝送途中のメッセージなど）すべてがひとまとめに表現されています．変数値が取りうる値は様々あり，またメッセージ内容も様々ですから，取りうる状況の組み合わせの数は膨大になります．分散システムが正しく動作している望ましい状況のことを「正当な状況」と呼ぶことにします．状況が正当かどうかは，各コンピュータの状態がネットワーク全体で相互に整合が取れているかどうかで決まります．1 台のコンピュータの状態だけで分散システムが正しく動作しているかは判断できないからです．

自己安定性を持つ分散システムは，一時故障（transient fault）と呼ばれる障害への極めて強靭な耐性を持ちます．一時故障には，例えば，伝送途中のメッセージの損失（通信障害），メッセージ内容の変化（ノイズ），コンピュータの再起動によるメモリ内容の消失（リブート），宇宙線の影響によるメモリ内容の変化（ソフトエラー）などが含まれます．すなわち，ハードウエアやソフトウエアのプログラムコード自体は正常であっても，情報が意図しない内容に変化してしまう類の障害のことを言います．

分散アルゴリズムが自己安定であるとは，以下に説明する閉包性と収束性のふたつの条件を満たすときをいいます．

閉包性：ひとたび分散システムが正当な状況になると，その後の実行で現れる状況は正当である．（新たな一時故障が生じない限りは，システムは安定して正しく動作し続ける性質を表します．）

収束性：必ずしも正当とは限らない状況から分散システムが動作を開始しても，その後のいかなる実行に対しても，正当な状況に到達する．（正当な状況に到達しないまま無限に続く実行は一切存在しないということです．つまり一時故障が同時多発的に分散システムの各所で生じても，やがては正しい動

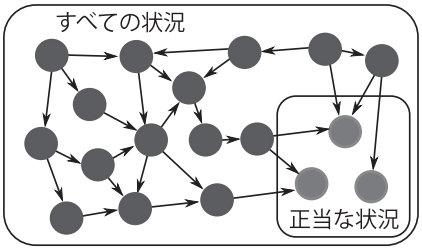


図 2　自己安定分散システムの状況の遷移図

作に復帰することを保証しています.）

図 2 に自己安定分散システムの状況の遷移図を示します．丸（●）はシステムの状況を，矢印（→）はシステムの動作による状況間の遷移を，それぞれ表します．この図において，閉包性と収束性がともに成り立っていることを確認してみて下さい．正当ではない状況からどのような矢印のたどり方をしても，やがて必ず正当な状況へ到達します.

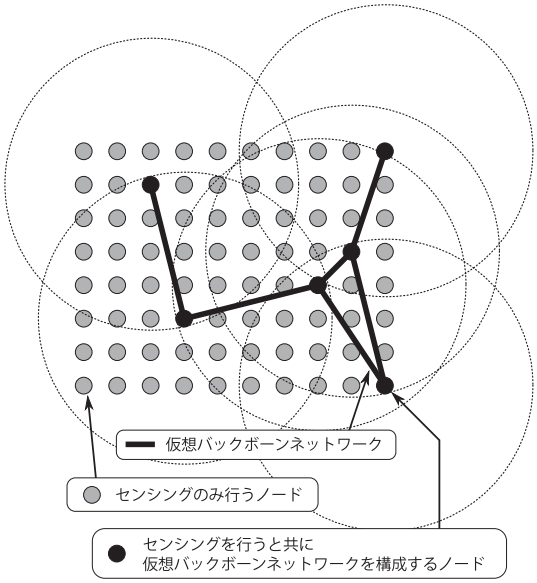


図 3　極小連結支配集合（黒丸のノード群）による無線ワイヤレスセンサネットワークにおける仮想バックボーンネットワーク

5.2　例：連結支配集合を求める問題

無線センサネットワークでのデータ収集に有用なネットワーク構造のひとつとして，連結支配集合が挙げられます．例えば図 3 に示すように，センシングだけを行うノードと，センシングに加えてデータ送信のための仮想バックボーンネットワークを構成するノードの 2 つに役割を分けて，データ収集を行うのに有用です．仮想バックボーンネットワークとは分散システムの背骨となる仮想的なネットワークであり，分散システム全体での通信を可能にする役目を持ちます．従って仮想バックボーンネットワークは分散システム全体をカバーしていることが重要です.

ネットワーク（グラフ）の極小連結支配集合（Minimal Connected Dominating Set；以下 MCDS）とは，以下の条件を満たすノード（頂点）の集合です.

支配性：MCDS に含まれないノードは，MCDS に含まれるノードと直接通信できる．すなわちセンシングだけを行うノードは，仮想バックボーンネットワークを構成するノードに直接隣接している，ということです.

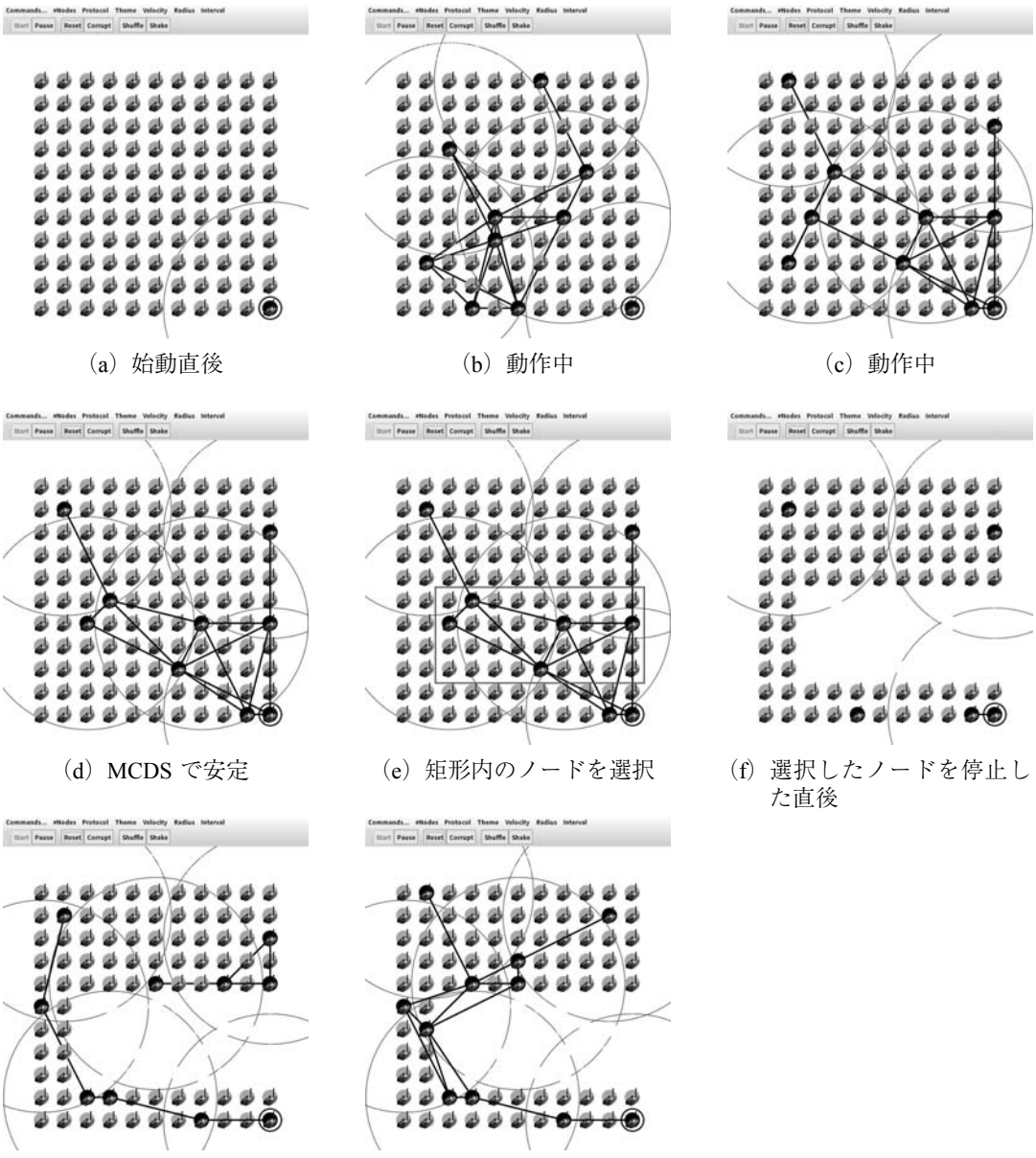
連結性：MCDS に含まれているノードだけで構成するネットワークは 1 つの連結成分だけである．すなわち，仮想バックボーンネットワークの数は 1 つだけであり，相互にマルチホップ通信が可能である，ということです.

極小性：MCDS に含まれているノードをひとつでも減らすと，支配性あるいは連結性が成り立たなくなる．すなわち，仮想バックボーンネットワークを構成するために働くノードの数は極力少なくして資源の消費を少なくしたい，ということです．極端な場合，全てのノードを選べば（極小ではない）連結支配集合になりますが，それでは全てのノードが仮想バックボーンネットワークを構成するために働くことになり，資源の浪費になるからです.

無線センサネットワークでは，バッテリー切れによるノードの停止，バッテリ充電によるノードの復活，通信障害など，様々な事象が発生しますので，自己安定性があればネットワークの管理・運営に極めて便利です．文献［7］では MCDS を求める自己

安定分散連結支配集合アルゴリズムが提案されており，その実行の経過の様子を図 4 に示します．これは 1 台のパーソナルコンピュータ上で実行する自己安定分散アルゴリズムのビジュアルシミュレータ（著者作成）による表示のスクリーンショットで，以下に実行の経過を説明します．

図 4（a）：動作開始してすぐの状況です．11×11 の計 121 のノードを格子状に配置しており，各ノードは無線により周囲のノードと通信をします．右下のノードを中心とする大きな円は，無線通信可能な範囲を表しています．他のノードも同じ通信半径を持ちますが，表示は一部のノードに対してのみ行っ



（a）始動直後　（b）動作中　（c）動作中

（d）MCDS で安定　（e）矩形内のノードを選択　（f）選択したノードを停止した直後

（g）動作中　（h）MCDS で安定

図 4　自己安定分散連結支配集合アルゴリズムの実行例

ています．全てのノードに対して描くと見にくくなってしまうからです．右下のノードに各センサの測定データを収集するというシナリオを想定すると分かりやすいと思います．

図 4（b）：動作開始後，しばらく経過した状況です．各ノードは MCDS 自己安定分散アルゴリズムを実行しています．MDCS のノードとして選ばれているものは黒い丸で描かれています．まだ途中段階ですので，MDCS は完成していません．

図 4（c）：更に時間が経過した様子です．これもまだ途中段階で，MDCS は完成していません．ノードは MDCS のノードとして選ばれれるか否かを互いに調整しながら動作しますので，いったん選ばれたノードもその後には選ばれなくなることはしばしば起きます．

図 4（d）：MCDS が求まった状況であり，分散システムに障害等が発生しない限りはこのまま変化は起きず，安定した状況が続きます．MCDS のノードとして選ばれたものの中で，互いに直接通信が可能な，つまり電波が到達する 2 ノード間に直線を描いてあります．この状況を見ると，以下の 2 点が成り立っていることが分かります．

・MCDS のノードとして選ばれているものだけで構成した部分ネットワークは，ひとつの連結した成分となっている．（バックボーンネットワークを通じて，データは右下のノードに必ず到達できる．）

・MCDS のノードとして選ばれていないものは，選ばれているノードの通信半径内に入っている．（バックボーンネットワークを構成するノードと直接的に通信ができる．）

図 4（e）：ここで意図的に分散システムに障害を発生させてみます．矩形で囲まれたノードを選択し，これらの電源の停止を行います．

図 4（f）：選択されたノードの電源を停止した直後の状況です．残ったノードだけでは MCDS が構成できていません．各ノードは不整合が起きていることを検知して，MCDS 自己安定分散アルゴリズムに従って動作します．このとき各ノードは周囲のノードと相互に通信をすることで不整合の解消を行う形のアルゴリズムになっています．あるひとつの特定のノードがネットワーク全体の状況を確認して，修復司令を一斉に出すのではありません．

図 4（g）：MCDS 自己安定分散アルゴリズムに従って動作している途中の状況で，MCDS が再構成されつつあります．

図 4（h）：現在動作しているノード群に対する MCDS が求まり，安定した状況です．分散システムに障害等が発生しない限りはこのまま変化は起きず，安定した状況が続きます．

以上では簡単な動作シナリオを見てみましたが，分散システムのあちこちで同時多発的に障害やノードの停止などが起きたとしても，それらが収まれば自動的に安定します．

5.3 前方復旧

分散システムの変化が起きても，上述のようにやがては安定するほか，ノードのメモリをデタラメに書き換えてもやがては安定します．ここで注意していただいたいのは，システムのバックアップをとっておき，それに基づいた復旧（レストア）を行うのではないという点です．バックアップをとっておいても，上記のようにノードの電源がオフになったり，新たなノードが追加されたりすると，分散システムの構成が変化しているために，バックアップのデータそのものが役に立ちません．

自己安定の概念は，過去の正しい状況に戻すという後方復旧（backward recovery）とは異なり，今の諸条件に合う正しい状況にシステムを持ってゆくという前方復旧（forward recovery）という考え方です．参加するノードや利用可能なネットワークがしばしば変わる分散システムでは，自己安定の概念はとても有用な設計技法といえます．

6. おわりに

本稿では多数のコンピュータで構成される分散システムを動作させるための分散アルゴリズム，特に

故障耐性や自律性を実現する自己安定分散アルゴリズムを紹介しました．コンピュータの数が多数になるほど，故障耐性や自律性をアルゴリズムそのものに持たせることが重要になります．自己安定分散システムは，この課題の解決を目指した分散システムの設計手法です．なお本稿で紹介した自己安定アルゴリズムの定義では，自己安定分散アルゴリズムが完全に収束し安定するまで時間がかかることがあります．この問題点を解決する研究も行っていますが本稿では省略します．

なお本文中で紹介したアイドルグループのファンクラブの会員の総数を求める問題は，分散アルゴリズムの分野ではグローバルスナップショット（global snapshot）問題として知られており，分散システム全体のスナップショット（分散システムの状態）を適切に記録する問題です．重要なデータのバックアップをグルーバルスナップショットアルゴリズムにより取得しておき，システム障害の際にはバックアップのデータをリストアすることで障害からの復旧が行えます（後方復旧）．グローバルスナップショット問題は分散アルゴリズムの教科書では必ずと言っていいくらい取り上げられてる基本的な問題のひとつですので，興味ある方は調べてみて下さい．

参考文献

［1］ E. W. Dijkstra,“Self-Stabilizing Systems in Spite of Distributed Control”, Communications of the ACM, Vol.17, No.11, pp.643-644, 1974.

［2］ K. Altisen, S. Devismes, S. Dubois, F. Petit,“Introduction to Distributed Self-stabilizing Algorithms”, Morgan & Claypool, 2019.

［3］ S. Dolev,“Self-Stabilization”, MIT Press, 2000.

［4］ 増澤，山下，“適応的分散アルゴリズム”，共立出版，2010 年.

［5］ M. Schneider,“Self-stabilization”, ACM Computing Serveys, Vol.25, No.1, pp.45-67, 1993.

［6］ F. C. Gärtner,“Fundamentals of Fault-Tolerant Distributed Computing in Asynchronous Environments”, ACM Computing Serveys, Vol.31, No.1, pp.1-26, 1999.

［7］ S. Kamei, and H. Kakugawa,“A Self-stabilizing Distributed Approximation Algorithm for the Minimum Connected Dominating Set,” International Journal of Foundation of Computer Science, Vol.21, No.3, pp.459-476, 2010.