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

2007－MPS－63 (10)
2007／3／3

類似実行に基づく耐故障分散アルゴリズム理解支援システムの提案

古川正広，鈴木朋子，大下福仁，角川裕次，増澤利光
大阪大学大学院情報科学研究科

f-耐故障分散アルゴリズムとは，高々f個の計算機が故障しても正しく問題を解くアルゴリズムである．本研究では，f-耐故障分散アルゴリズムの理解を支援するために，$f+1$個の故障のもとで正しい解を出力しない実行と，その実行に類似したf個または$f+1$個の故障のもとで正しい解を出力する実行を生成するシステムを提案する．これらの実行を比較することで，故障が問題を解けなくした原因を利用者は考察でき，f-耐故障分散アルゴリズムの理解に繋がる．さらに，本研究では，２つの実行の生成に必要な計算時間を削減する仕組みについて考察と実験を行なう．その結果，類似度の高い２つの実行を実用的な時間で生成できることを示す．

A tool for understanding f-resilient distributed algorithms based on similar executions

Masahiro Furukawa , Tomoko Suzuki , Fukuhito Ooshita ,
Hirotsugu Kakugawa , Toshimitsu Masuzawa
Graduate School of Information Science and Technology, Osaka University

An f-resilient distributed algorithm solves a problem correctly if there are at most f faulty processes in a system. In this paper, we propose a tool to support understanding of f-resilient distributed algorithms. Our tool presents two similar executions of an algorithm: the one is an execution with $f+1$ faults in which the algorithm cannot solve the problem, and the other is an execution with f or $f+1$ faults in which the algorithm solves the problem. Users can easily understand the strategy for tolerating faults by comparing the executions. In addition, we present a method to reduce constructing time for the executions, and show by experiments that they are constructed in practical time.

1 はじめに

分散システムとは，複数のプロセスがネットワークで接続されたシステムである．近年，分散システムの利用が増えており，分散システム上で問題を解く分散アルゴリズムの重要性が高まっている．そのため多くの大学で分散アルゴリズムに関する講義が行われている．しかし多数のプロセスが自律的に動作する分散システムにおける，分散アルゴリズムの大域的な動作の理解は困難である．そのため，シミュレータを用いて各プロセスの動作を可視化することにより，分散アルゴリズムの動作の理解を支援する研究が行われている．

DisASTer [5]，LYDIAN [2] は，分散アルゴリズムの可視化実行環境であり，利用者は分散アルゴリズムの動作をコントロールしながら観察することができる．しかし，既存のシミュレータの多くは，利用者の指示通りにアルゴリズムを実行し，その動作を可視化するのみである．つまり，利用者は理解に役立つ分散アルゴリズムの動作パターンを自ら考えてシミュレータを利用する必要がある．

また，既存のシミュレータの多くは故障を考慮していない．実際の分散システムではプロセスの故障が起こりうるため，耐故障性は分散システムに対する重要な要求の１つであり，多くの耐故障分散アルゴリズムが提案されている．耐故障分散アルゴリズムの理解には，故障を含んだ動作の理解が必要不可欠であると考えられる．しかし，故障が起こることで非常に多くの動作パターンが考えられ，理解に役立つ動作パターンを利用者が考えることは非常に困難である．

そこで本研究では，利用者の理解に役立つ分散アルゴリズムの動作パターンを自動生成することで，耐故障分散アルゴリズムの理解を支援するシステムの開発を目指す．本研究では，特にf-耐故障分散アルゴリズムを対象とする．f-耐故障分散アルゴリズムとは，高々f個のプロセスが故障しても正しく問題を解く分散アルゴリズムである．

f-耐故障分散アルゴリズムの理解には，分散アルゴリズムが耐故障性を実現している仕組みを理解する事が重要である．そのためには，分散アルゴリズムが故障に耐えて正しい解を出力する動作と，故障により正しい解を出力しない動作を比較することで，正しい解を出力するかしないかの違いを与えている動作の差について考察することができればよいと考えられる．その際，類似度の高い２つの動作を比較することができれば，故障が動作に与えた影響に注目しやすいと考えられる．

そこで本研究では，f-耐故障分散アルゴリズムに対して，$f+1$ 個の故障のもとで正しい解を出力しない実行と，その実行に類似した f 個または $f+1$ 個の故障のもとで正しい解を出力する実行を生成するシステムを提案する．その際，f-耐故障分散アルゴリズムは故障が $f+1$ 個でも正しい解を出力して終了する場合があるため，故障を $f+1$ 個に設定してアルゴリズムを動作をさせるだけでは，正しい解を出力しない動作を生成できない．正しい解を出力しない動作の生成は，どのプロセスが，いつ，いくつ同時に故障するかという，膨大な故障パターンの中から，正しい解を出力しない限られたパターンを選出する必要があるため多くの計算時間が必要となる．そのため本研究では，いくつかの条件を入力して，計算時間を削減できる仕組みを考察する．実装実験により，これらの条件入力によって類似度の高い2つの動作を実用的な時間で生成できることを示す．

2 諸定義

分散システム 分散システム G はプロセス集合 V，リンク集合 E の組 $G=(V,E)$ で定義される．リンク (P_i,P_j) が存在する場合，P_i は P_j へメッセージを送信できる．各プロセスは状態機械としてモデル化される．各プロセスは，局所変数と受信メッセージから，次の局所状態または送信メッセージを決定する．分散アルゴリズムは，各プロセスに対する動作の規定として定義される．

分散システム全体の状態を大域状況（以下，状況）と呼ぶ．状況は，全プロセスの局所変数と送信中のメッセージの組で表される．状況は，イベントによって変化する．イベントとして，以下の2つを考える．

- $send(P_i,P_j)$:P_i が内部計算により局所変数を更新し，P_j へメッセージを送信．
- $recv(P_i,P_j)$:P_i が P_j からメッセージを受信．

分散アルゴリズムの実行は，初期状況とそれに続くイベント列 $Q=C_0,e_1,e_2,\ldots$ と定義される．以降，とくに必要がない場合，実行を単にイベント列として記述する．

問題は状況に対する述語 $spec(C)$ で与えられるものとする．$spec(C)$ は，問題に対してシステムが最終的に達すべき状況を表している．すなわち，ある状況 C において $spec(C)$ が成立すれば，状況 C において問題が解けていることを意味する．本研究では，ある状況以降，システムが $spec(C)$ を満たしたうえで状況変化を起こさなくなる静的問題のみを扱う．

本研究では，ラウンド同期モデルを対象とする．つまり，スケジュールをラウンドに分割でき，各ラウンドの前半を $send$ イベントで構成し，後半を前半の $send$ イベントに対応する（後述する故障プロセス以外への）$recv$ イベントで構成できる．

本研究では，停止故障を考慮するため，さらに以下の故障イベントを考える．

- $crash(P_i)$:P_i が停止故障を起こす．

故障を起こしたプロセスは，それ以降一切の動作を行なわない．すなわち，任意の実行において，$crash(P_i)$ が発生した場合，その発生以降には P_i に関連するイベントは発生しない．

f-耐故障分散アルゴリズムとは，実行中に高々 f 個のプロセス故障があっても必ず $spec(C)$ を満たし続ける状況に到るアルゴリズムである．

状況を変化させるプロセスが存在しない状況を終了状況と呼ぶ．実行 Q において，その終了状況 C が $spec(C)$ を満たし，かつ状況 C までに q 個のプロセスが故障している場合，Q を q-可解実行と呼ぶ．実行 Q において，その終了状況 C が $spec(C)$ を満たさず，かつ状況 C までに q 個のプロセスが故障している場合，Q を q-非可解実行と呼ぶ．

因果関係 分散計算では2つのイベントの間に次のような因果関係 ($\rightarrow$) が定義される．

- 同一プロセス内でイベント a がイベント b より先に起こるとき $a \rightarrow b$
- イベント a とイベント b が同一メッセージの送信および受信イベントであるとき $a \rightarrow b$
- $a \rightarrow b$ かつ $b \rightarrow c$ なら $a \rightarrow c$

また，$a \rightarrow b$ も $b \rightarrow a$ も共に成立しないとき，a と b は並行であるという．

類似度 まず，実行内の各イベントを絶対的に起こった順序で並べ，各イベントを文字とみなす事で，実行を表す文字列として捉える．しかし，並行なイベントの文字列上での順序はアルゴリズムの動作に影響を与えない．そこで，並行なイベントについてはその文字列上での順序が類似度に影響を与えないようにする．実行間の類似度はそれぞれの実行を表す文字列間の Levenshtein 距離 [3] を用いて算出する．Levenshtein 距離は，一方の文字列をもう一方の文字列と一致させるために必要な編集操作 (文字の挿入，削除，置換) の回数の最小値として定義される．

以下，$A(\alpha)$ を，実行 α を表す文字列内のイベントを因果関係を保ったまま任意の順で並び替えた，実行を表す文字列の集合とする．また，実行を表す文字列 α',β' 間の Levenshtein 距離を $LD(\alpha',\beta')$ と表し，実行を表す文字列 α',β' 間の Levenshtein 距離を計算する際に行われた，編集操作後の実行を表す文字列のうち長さが最長であるものの文字列長を $el(\alpha',\beta')$ と表す．ここで，2つの実行 α,β の類似度 $simil(\alpha,\beta)$ を，2つの実行内で起こるイベントが，どの程度の割合で同じかを表すために次のように定義する．

$$simil(\alpha,\beta)=\max\{\frac{el(\alpha',\beta')-LD(\alpha',\beta')}{el(\alpha',\beta')} \mid \alpha'\in A(\alpha),\beta'\in A(\beta)\}$$

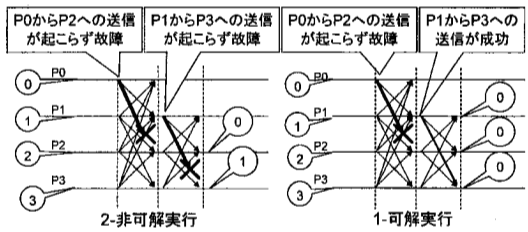

図 1: 類似度の高い 2-非可解実行と 1-可解実行

3 提案するシステムの概要

3.1 入出力

システムへの入力は以下の 4 つである：

- 分散システム $G = (V, E)$.
- f-耐故障分散アルゴリズム.
- 問題に対する述語 $spec(C)$.
- 探索条件 (詳細は 4.1 節で述べる).

システムは与えられた f-耐故障分散アルゴリズムに対して，次の二つの実行の組のうちどちらかを出力する．どちらを出力するかは利用者が指定する．

- $(f+1)$-非可解実行とそれに類似した f-可解実行
- $(f+1)$-非可解実行とそれに類似した $(f+1)$-可解実行

ただし，上記の実行を発見できない場合はその旨を出力する．

3.2 システムの利用例

本システムで出力する実行の組は，類似度が高いため，プロセスの動作が少し異なるだけで可解実行と非可解実行に分かれている．つまり，その少しの動作の違いが，実行を可解実行とするか非可解実行とするかを決めていると考えられる．利用者を動作の違いに注目させ，その違いによる実行の違いを考察させることで，故障が実行に与える影響を考察させることができると考えられる．例えば図 1 は FloodSet コンセンサスアルゴリズム [4] の 2-非可解実行と 1-可解実行の組 (類似度 0.87) である．このアルゴリズムは既知の故障数 f に対し，全プロセスが $f+1$ 回自プロセスの ID と既知の ID を送信し，$f+1$ 回の受信が終わった後に受信した ID の最小値に合意するアルゴリズムである．図の例は既知の故障数を 1 とした 1-耐故障アルゴリズムである．この 2 つを見比べると，最後の 1 つの故障の部分が異なっており，故障のために受信されなかったメッセージの影響でプロセスの出力が互いに異なっている事が分かる．故障数が 2 個未満であれば，必ずあるメッセージの受信で最小の ID を受信するため，この部分がアルゴリズムが故障に耐える仕組みである事が分かる．

4 類似度の高い実行の組の生成方法

本章では $(f+1)$-非可解実行と，その実行に類似した f-可解実行，または $(f+1)$-可解実行を生成する方法を述べる．

4.1 $(f+1)$-非可解実行と f-可解実行の生成

$(f+1)$-非可解実行と f-可解実行を生成する際の大まかな手順は次の通りである．

1. $(f+1)$-非可解実行を 1 つ探索し，その実行に類似した f-可解実行を生成.
2. 1 を全ての $(f+1)$-非可解実行に対して行う.
3. 求めた $(f+1)$-非可解実行と f-可解実行の組のうち最も類似度の高いものを出力する.

$(f+1)$-非可解実行の探索は，実行木の深さ優先探索で行なう．実行木とは，起きうる状況すべてを表すものである．各頂点がラウンド開始時の状況を表すもので，以下のように定義する．

- 根は初期状況を表す.
- 頂点 v が状況 C を表しているとき，1)C が終了状況であれば，v は子をもたない，2)C が終了状況でないとき，v の子 $w_1, w_2, \ldots$ は C から 1 ラウンド後の状況 C' を表す.

状況 C から 1 ラウンド後の状況 C' は，以下のように作成される．状況 C で故障プロセス数が $f+1$ 未満のとき，そのラウンド中に故障が発生する場合を考える．その場合，新たに故障を起こすプロセスの組合せを，状況内の合計故障プロセス数が $f+1$ 以下となるように全て列挙し，それぞれの組合せに対して 1 ラウンド後の状況を作成する．その際，故障プロセスが送信したメッセージは送信先プロセスで受信される場合と受信されない場合があるため，全ての受信パターンについて C' を作成する．

実行木を探索し，終了状況 C に対応する葉 v に至ったとする．このとき，C において，$spec(C)$ が満たされていなければ，そのような状況 C における故障数は $f+1$ であるため，根から v へのパスに対応する実行は $(f+1)$-非可解実行である．

この探索方法では，1 ラウンドに故障するプロセスの組合せ，メッセージの受信パターンがプロセス数に対して指数的に増大するため，探索すべき実行木が非常に大きくなる．そのため，その探索には膨大な時間を要する．そこで，対象とする分散アルゴリズムに対して，システムの利用者が断片的な知識を持っていれば探索時間を削減できるように，探索条件を入力することで探索時間を削減する仕組みを導入する．本システムで用いている探索時間削減の方法は次の通りである．

- 1 ラウンド中の最大故障数を指定
- 求める $(f+1)$-非可解実行の数を指定
- アルゴリズム特有の条件の入力
- 部分的な実行パターンの入力

1 ラウンド中の最大故障数を指定することで 1 ラウンド中に故障を起こすプロセスの組み合わせの数を減らすことができる．求める $(f+1)$-非可解実行の

数を制限することで，実行木の探索を途中で打ち切ることができる．アルゴリズム特有の条件とは，「ある状況が特定の条件を満たしている場合，その状況を含む全ての実行が可解実行となる」という条件である．この条件を満たす実行木の頂点では，その子頂点を計算しないことで実行木を小さくできる．部分的な動作パターンとは，あるラウンドで起きるイベントが予め分かっている場合に入力する文字列であり，入力通りの実行にならないような実行木のパスを計算しないことで，実行木を小さくできる．これらの方法を導入することで，計算すべき実行木を小さくし，探索時間を削減できると考えられる．

f-可解実行は，$(f+1)$-非可解実行で起きる $f+1$ 個の故障から1つ故障を減らして計算する．つまり，$f+1$ 個のそれぞれの故障に対して，その故障以外の故障が元の実行と同じラウンドで起こる実行を計算し，その実行を f-可解実行とする．

4.2 $(f+1)$-非可解実行と $(f+1)$-可解実行の生成

$(f+1)$-非可解実行と $(f+1)$-可解実行を出力する際の大まかな手順は次の通りである．

1. $(f+1)$-非可解実行を全て探索し，探索の途中で $(f+1)$-可解実行があればそれを記録する．
2. 1で求めた $(f+1)$-非可解実行と $(f+1)$-可解実行の全ての組み合わせについて類似度を計算し，最も類似度の高い組を出力する．

$(f+1)$-非可解実行の探索は4.1節と同様に行い，その探索中に故障数が $f+1$ で，かつ $spec(C)$ を満たす終了状況 C に対応する葉 v に到れば，根から v へのパスに対応する実行が $(f+1)$-可解実行である．

5 評価

提案システムの計算時間の評価結果を以下に示す．Majority-Ack Uniform Reliable Broadcast [1](以下，放送問題) と，FloodSet コンセンサス [4](以下，合意問題) の2つのアルゴリズムに対して，計算時間を計測した．実行環境は，Windows XP Professional, Pentium D 3.00GHz，メモリ 1GB，実装言語 Java，処理系は Sun Microsystems の JDK 5.0 である．

表1は，さまざまな条件入力に対して $(f+1)$-非可解実行と f-可解実行を出力するために要した時間を表す．+,*が付いている行はそれぞれ部分的な実行パターンを入力した場合と，アルゴリズム特有の条件を入力したときの結果である．部分的な実行パターンとして，1ラウンド目に起こるある1つの故障を入力した．また，アルゴリズム特有の条件は，必ず合意問題が解ける「全プロセスが最小 ID を受信」という条件を入力した．表1から，条件入力を行うことで多少類似度は低くなるが，十分に短い実行時間で実行を求められることが分かる．合意問題では $(f+1)$-非可解実行が1つしかないため，条件入力によって類似度は変化しない．

表 1: 実行結果 (上：放送問題, 下：合意問題)

プロセス数, 故障数 f	1ラウンド中の最大故障数	求める $(f+1)$-非可解実行の数	計算時間 (s)	最大類似度
7,3	4	all	23576	0.95
7,3	1	all	26	0.9
7,3	4	1	4	0.86
7,3	1	1	4	0.9
7,3+	4	all	904	0.88
5,2	3	all	70	0.93
5,2	1	all	13	0.93
5,2	3	1	12	0.93
5,2	1	1	2	0.93
5,2*	3	all	21	0.93

6 おわりに

本研究では，類似実行に基づく f-耐故障分散アルゴリズムの理解支援システムを提案した．提案システムでは，$(f+1)$-非可解実行と，それに類似した f-可解実行または $(f+1)$-可解実行を示す．さらに $(f+1)$-非可解実行の探索に要する計算時間の短縮の為に，利用者が断片的な知識を基に探索条件を入力できる仕組みを導入した．提案システムについて実装実験を行い，条件入力により十分に短い実行時間で2つの実行を生成することができた．今後の課題として，より広いクラスの故障への対応が考えられる．

謝辞 本研究の一部は，文部科学省 21 世紀 COE プログラム（研究拠点形成費補助金）の研究助成，日本学術振興会科学研究費補助金（基盤研究 (B)15300017，基盤研究 (B)17300020），日本学術振興会特別研究員奨励費 (2005,50673)，文部科学省科学研究費補助金（若手研究 (B)18700059），総務省戦略的情報通信研究開発推進制度 (SCOPE)，および，大川情報通信基金によるものである．

参考文献

[1] R. Guerraoui and L. Rodrigues, Introduction to Reliable Distributed Programming, Springer-Verlag, 2006.

[2] B. Koldehofe, M. Papatriantafilou, and P. Tsigas, Distributed algorithms visualisation for educational purposes, Proceedings of the 4th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and technology in computer science education, pp. 103–106, 1999.

[3] V. I. Levenshtein, Binary codes capable of correcting deletions, insertions, and reversals, Soviet Physics Doklady, Vol. 10, No. 8, pp. 707–710, 1966.

[4] N. A. Lynch, Distributed Algorithms, Morgan Kaufmann, 1996.

[5] R. Oechsle and T. Gottwald, Disaster (distributed algorithms simulation terrain): a platform for the implementation of distributed algorithms, Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education, pp. 44–48, 2005.