分散アルゴリズムの実験的評価について

——分散 k-相互排除アルゴリズムを例として——

角川裕次[†] 藤田聡[†]
山下雅史[†] 阿江忠[†]

強力な処理能力を持つ計算機が普及し，それを通信網によって有機的に結合した計算機ネットワークが発達したので，その上での分散処理が一般的になりつつある．計算機ネットワークの上に分散処理を目的として構成されたシステムを分散システムと呼ぶが，分散システムで問題を解くアルゴリズムは特に分散アルゴリズムと呼ばれている．分散処理の普及と共に分散アルゴリズムの活発な研究がなされてきている．本稿では，分散アルゴリズムを実験的に評価するための各種の技法について議論するとともに，分散 k-相互排除アルゴリズムを例として，実際に評価のためのプログラムを作成してアルゴリズムのシミュレーション実験を行う．シミュレーションはイーサネットで結合されたワークステーション群で行い，1台のワークステーションにつきアルゴリズムを実行するプロセスを一つ走らせる．プロセスは，他の計算機で実行されているプロセスとメッセージ通信を行いながらアルゴリズムの実行を行ってゆく．シミュレーションは実時間で行われ，実時間で実行させるための注意点についての議論がされる．また，アルゴリズムはC言語で記述して実行されるが，シミュレートすべきアルゴリズムをC言語に変換する際の技法を示す．

A Study on Experimental Evaluation of Distributed Algorithms

—Case Study : Distributed k-Mutual Exclusion Algorithms—

Hirotsugu Kakugawa,[†] Satoshi Fujita,[†]
Masafumi Yamashita[†] and Tadashi Ae[†]

The distributed processing becomes more and more common as computer networks which combine remote computers with powerful CPU power getting popular. Systems developed on such networks for distributed processing are called distributed systems and algorithms for distributed systems are called distributed algorithms. In this paper, we discuss several techniques for simulating distributed algorithms, in order to evaluate them experimentally, and as a case study, we evaluate distributed k-mutual exclusion algorithms by simulation. To simulate a distributed system, workstations connected by the Ethernet are used. An algorithm is written in C. On each workstation, one process simulating the algorithm is executed. By exchanging messages with other processes simulated on other workstations, processes simulate the algorithm. We discuss techniques for simulating distributed systems, as well as those for implementing algorithms in C.

1. はじめに

強力な計算機の小型化と計算機ネットワークの発達に伴い，これまでは1台の計算機で処理されていたものが複数の計算機を用いて処理を行う分散処理が一般的になりつつある．分散処理の特徴の一つとして，耐故障性が良いことがあげられる．複数の計算機を用いているため，もし幾つかの計算機が故障しても正常な計算機が処理を続行することが可能となるからである．

分散処理を“計算”としてとらえると，計算を行うアルゴリズムが必要となる．分散処理の場合は，1台の計算機で実行される逐次アルゴリズムと異なり，他の計算機と交信をして情報を交換しながら問題を解かなくてはならない．このようなアルゴリズムが分散アルゴリズム（distributed algorithm）である．分散アルゴリズム（以下，単にアルゴリズムと呼ぶ）の評価基準として次のものが挙げられる[9]．

- メッセージ複雑度（message complexity）—アルゴリズムの起動から停止までにどれだけのメッセージが送信されたかを数えたものである．局所的な計算に比べると，メッセージ送信はオーバーヘッドが大きいので送信するメッセージの数が実行時間を支配するであろうという考えに基づいている．
- 時間複雑度（time complexity）—アルゴリズムが

† 広島大学工学部第二類
Department of Electrical Engineering, Hiroshima University

終了するのに要する時間を評価したものである．メッセージ配送と局所的な計算に要する時間を計量するが，局所的な計算に要する時間は無視されることが多い．

著者らは，文献2),6),7)において分散 k-相互排除に関する研究を行っており，文献7)で分散 k-相互排除アルゴリズムを提案した．また，Raymond もまた文献13)において分散 k-相互排除アルゴリズムを提案している．これらのアルゴリズムのメッセージ複雑度の最良時および最悪時の評価は比較的容易だが，平均時の解析は困難である．また，アルゴリズムの実行に要する時間の評価も同様である．そこで，これら二つのアルゴリズムを実験的な評価で比較検討するという動機で本論文で報告する研究を開始した．

事象駆動 (event driven) による分散アルゴリズムシミュレーションは，例えば，文献11)がある．文献11)で用いられているシミュレータは，1台の計算機の上でシミュレーションを行うものであり，文献5)ではこのシミュレータを用いてリーダー選挙問題を解く各種アルゴリズムの実験的な評価を行っている．また，文献1)では，分散システムを設計するときにシステムの構成要素すべてシミュレーションで行う段階から始め，順次構成要素を実際の構成要素に置き換えていく混合型のシミュレーションについて議論されている．文献15)では，1台のパーソナルコンピュータ上で分散アルゴリズムを実行するシミュレータについて述べられているが，これはアルゴリズムの評価よりも，むしろデバッグを目的としたものであり，アルゴリズムを実行するプロセスは逐次的に動作している．一方，我々が行ったシミュレーションでは，分散アルゴリズムを実行するプロセスがすべて完全に並列に実行される．すなわち，イーサネットで結合されたワークステーション群を用い，各ワークステーションには，アルゴリズムを実行するプロセス一つを実行させる．そして，アルゴリズムを実行するプロセスは，ネットワークを通じて他のワークステーション上で実行されているプロセスとメッセージ通信をしながらアルゴリズムのシミュレートを行っていく．これは，時間駆動 (time driven) に分類されるものである．シミュレーションを行うことは一般的に複雑であるが，対象を分散アルゴリズムに限定することでシミュレーションモデルが単純になる．なぜなら，すべてのプロセスが同一のアルゴリズムを実行することと，局所計算に要する時間は無視できるほど小さいと仮定されることが多いからである．しかし，対象が実際の分散システムであったならば，きわどいタイミングの制約が正しく満たされているかを評価できないといけないので，シミュレーションはより複雑なものになる．

本研究の目的は，(1)分散 k-相互排除アルゴリズムを実験的に評価すること，(2)評価を行う際，容易にシミュレートができる基盤を作成すること，の二つである．本稿では，主に後者に的を絞り，分散アルゴリズムの評価方法，製作した評価ツールの設計方針，与えられた評価すべきアルゴリズムを記述言語であるC言語に変換する際の技法を，分散 k-相互排除アルゴリズムを例にして議論する．

2章では，評価の対象である二つの分散 k-相互排除アルゴリズムについて説明し，3章では，アルゴリズムを評価する際に必要となるプロセスの振舞いをモデル化し，これを説明する．4章では，C言語の形で提供される評価ツールの設計について述べ，5章では，評価すべきアルゴリズムを評価ツールを用いてC言語で実現する技法について議論する．6章では実験結果を示し，7章では，本研究で得られた結果を述べ，今後の課題について議論する．

2. 分散 k-相互排除アルゴリズム

本章では，評価の対象である二つの分散 k-相互排除アルゴリズムを簡単に説明する．詳細は文献13),7)を参照されたい．

二つのアルゴリズムは，次のような仮定に基づいた分散システム上で動作する．N個の一意な識別子（プロセス ID）を持つプロセスと，任意の二つのプロセス間で通信ができる通信路よりシステムは構成され，プロセス間の情報のやりとりはメッセージ通信のみによる．プロセスおよび通信路は故障しないとする．また，メッセージが配送されるのに要する時間は有限であるが，その遅延時間をプロセスがあらかじめ知ることはできない．また，プロセス間の相対的な実行速度も，一般に異なる．

2.1 Raymond のアルゴリズムの概要

このアルゴリズムは，Ricart と Agrawala によるアルゴリズム[14]を拡張したものである[13]．(以降，このアルゴリズムをアルゴリズムRと呼ぶ．)

プロセス X が臨界領域に進む時，X は自分以外の $N-1$ 個のプロセスに REQUEST メッセージを送る．そして REQUEST メッセージを受けとったプロセスのうち，臨界領域に入っていないプロセスはすぐ

にREPLYメッセージを返す．一方，臨界領域に入っているプロセスは，REPLYメッセージを送ることをそのプロセスが臨界領域から抜けるまで延期する．また，相互排除要求中のプロセス Y は，Y の要求に対するタイムスタンプ*と REQUEST を送ってきた X のタイムスタンプとプロセスIDの組を比べ，X の方の値が小さければ REPLY を返し，そうでなければ Y が臨界領域から出るまで REPLY を送ることを延期する．$N-k$ 個以上のプロセスより REPLY を受ければ，X は臨界領域に進む．

2.2 角川らのアルゴリズムの概要

このアルゴリズムは，k-コータリー（k-coterie）を利用したアルゴリズムである（以降，このアルゴリズムをアルゴリズムKと呼ぶ）．k-コータリーはコータリー[3]を一般化した概念である[2),6)]．k コータリー S に対し，$q \in S$ はコーラム（quorum）と呼ばれるプロセスの集合である．コーラルは k 個までは互いに交わらないようなものがあるが，互いに交わらない $k+1$ 個のコーラムは存在しない．

アルゴリズムKの概略は次のとおりである．相互排除要求が生じたプロセス X は，どれか一つコーラムを選び，そのコーラムに属するすべてのプロセスにREQUESTメッセージを送る．REQUESTメッセージを受けとったプロセス Y が，もし既に他のプロセスに許可メッセージ OK を送っていたら（各プロセスは各時点において高々一つのプロセスにしか許可を与えない），X に WAIT メッセージを送ると共に X の要求をキューに入れる．もしどのプロセスにも許可を与えていなければ X に OK メッセージを送る．X は，要求を送ったプロセスすべてから返事が帰ってくるのを待つ．もし一つでも WAIT メッセージが返されれば，他のコーラムを選んで今度はそのコーラムより許可を得るために上で述べた動作を繰り返す**．あるコーラムに属するすべてのプロセスより許可が得ることができれば，臨界領域に進むことができる．臨界領域を出る時は，OK を送ってきたプロセスに RELEASE メッセージを送って得た許可の解放を行う．

3. 相互排除要求プロセスのモデル

本章では，シミュレーションを行うに当たり，相互排除要求プロセスの振舞いをモデル化する．各プロセスは時計を持っており，この時間の単位を単位時間と呼ぶ．プロセスは四つの状態（通常状態，要求状態，臨界領域状態，脱出状態）をとり，一定の規則にしたがって状態遷移を行う．

通常状態は，プロセスが能動的な動作を何もしていない状態である．しかし，プロセスがこの状態にある時，単位時間当たり確率 p $(0 \leq p \leq 1)$ で相互排除要求が生じる．相互排除要求が生じると，プロセスはその状態を要求状態に遷移する．要求状態は，プロセスが臨界領域に進むために何らかの要求動作を行っている状態である．臨界領域に進むための条件が成立すれば，プロセスは臨界領域状態に遷移する．ある一定時間（T_{cs} 単位時間）経た後に，プロセスは臨界領域より出る．この時，プロセスは脱出状態に遷移する．この脱出状態に入ると，プロセスは臨界領域から出た時の動作を行う．

4. 評価ツールの設計

本章では，評価しようとしているアルゴリズムを実験的に評価するためのツールの設計について述べる．

4.1 プロセスの配置方法と実行方法

分散アルゴリズムを実行して評価するのが目的であるので，何らかの形で複数のプロセスが並列に実行される必要がある．これを実現するには，次の4通りのアプローチが考えられる．

1. 1台の計算機上で，複数のプロセスを論理時間で動作させる．
2. 1台の計算機上で，複数のプロセスを実時間で動作させる．
3. 複数の計算機を用い，各計算機上で一つのプロセスを論理時間で動作させる．
4. 複数の計算機を用い，各計算機上で一つのプロセスを実時間で動作させる．

すべてのシミュレーション方式がこれらの四つに分類できる訳ではなく，中間に位置するものもある．文献12),4)では，複数台の計算機を用い，複数のプロセスを1台の計算機で実行するシミュレータについて述べられている．また文献1)では，論理時間によるシミュレーションと実際のシステムの構成要素の混合されたモデルでのシミュレーション方式について述べてある．

上に示した4通りの方法の各々について，長短を以下に述べる．なお，上で述べた論理時間とは，例えばメッセージ送信を行ってから次の送信を行うまでの間

---

* タイムスタンプについては，文献8)を参照されたい．
** デッドロック回避のため，タイムスタンプによる優先順位に従い，必要に応じて許可の横取りが行われる．

を1単位時間として時計を進めていくものである．また，実時間とは，1単位時間を例えば1秒としアルゴリズムの実行の流れとは独立としたものである．

方法1.は，本来ならば並列に実行されないといけないプロセスが逐次的に実行される．（これは2.にもいえる．）従って，正しく分散システムがシミュレートされていない．より正確なシミュレーションを行うためには，論理時間の粒度を細かくする必要がある．例えば，仮想計算機を定義して，それに対する機械語でアルゴリズムを書き，機械語単位でプロセスの実行のスケジューリングを行ってシミュレーションを進めていく方法が考えられる．この方法は有効と考えられるが，シミュレーションのためのシステムが大きくなるという欠点がある．

方法2.では，プロセスが疑似並列に実行されるので，後に述べる方法4.と同様な注意が必要である．実行するプロセスの数が多くなると，1台の計算機でシミュレートするよりも複数の計算機を用いた方が負荷分散の点からも好ましい．

方法3.は，各プロセスを均等に実行させるよう並列に動作できるプロセスは並列に動作できる．しかし，すべてのプロセスが論理時間に従って同期をとりながら実行される必要がある．従って，この方法は1の方法を高速化したものと考えるのが妥当である．

本研究では，方法4.を採用した．この方法の良いところは，各プロセスは並列に動作できるところにある．すなわち，実際の分散システムに近いと考えられる．しかし，単純にこの考えを導入して実現すると不都合が生じることがある．もし，用いる計算機の構成に結果が依存するような実験手法を用いると，実験条件が特殊なものとなり，実験結果の一般性に欠ける．用いる計算機群をすべて同一のものである環境を用いることは，コストの点から考えて，実現が困難である．たとえ同一の計算機を必要な数だけ用意した場合でも，バックグラウンドで他のプログラムが実行されていれば，同一の実験状況の再現が不可能である．従って，処理能力が異なる計算機群から成る環境でも一般性のある実験結果が得られる実験方法を考えないといけない．

4.2 時間の取り扱い

シミュレーションでの時間の取り扱いについて述べる．各プロセスは対応した計算機上で実行されるが，全節で述べた理由により，時間の流れ具合が各計算機の処理能力や負荷に依存しない実現方法を採用しないといけない．そこで，各プロセスの時間の流れを各計算機がハードウェアで持っているリアルタイムクロックの流れに合わせることで解決した．

一単位時間Qを何秒とするかであるが，原理的には長い方が良い．なぜなら，アルゴリズムを処理するのに要する時間は零ではないため，Qが大きいほど一単位時間当たりの処理時間が小さくなって，計算機の違いによる処理時間の差が小さくなるからである．しかし，計算機の処理速度を考えれば，数秒程度で十分と思われる（6章で行う実験では $Q=1$ 秒とした．）

4.3 評価ツールの設計方針

ここでは，作成した評価ツールの設計方針を説明する．本評価ツールは，C言語の関数群で構成される．すなわち，提供された関数を組み合わせてアルゴリズムを記述し，実行と測定を行う．アルゴリズムの評価は Unix の稼働するワークステーション群上で行い，プログラミング言語はC言語を用いることにした．C言語のネットワークプログラミング用のライブラリ関数は細かな記述が可能であるが，その欠点としてプログラマが細かな点にまで気を使ってプログラミングを行わないといけないことが挙げられる．アルゴリズムを評価する立場からいえば，本質的でない細かな点を気にせずアルゴリズムの記述を行いたい．このことは，単に細かなことを書く必要から解放されるばかりでなく，アルゴリズム記述の見通しが良くなり，コーディング時のバグが入り込む余地が少なくなる．

4.4 評価ツールの構成

シミュレーションは1台のワークステーションにつき一つのプロセスが実行されることにしたので，シミュレーションシステムは簡潔なものとなる．しかし，逆に，利用できるワークステーションの数でシミュレーションできる分散システムの大きさが制限されてしまう欠点がある．我々の作成したツールは，主に通信のための関数群と時刻を取り扱う関数群より成る．

まずは，通信を行う機構について述べる．なお，各計算機には一つのアルゴリズム実行プロセスのみが実行されているので，計算機とアルゴリズム実行プロセスを同一視する．Unix の用語であるプロセスと区別するために，以降ではアルゴリズム実行プロセスのことをサイトと呼ぶことにする．プロセス間でメッセージの送受信を行うには，サイト識別子 (site ID) と呼ばれる，0から $N-1$ の整数で通信相手を指定する．ここでNはアルゴリズム実行に参加するサイトの総数である．サイト識別子からホスト名への対応づけは，

```
##
## Configuration file
##
pluto
charon
mercury
saturn
jupiter
```

図 1 構成ファイルの例
Fig. 1 An example of configuration file.

構成ファイル (configuration file) と呼ばれる表に記述する．構成ファイルの例を**図1**に示す．構成ファイルにはホスト名のリストを書き，上からサイト識別子が 0, 1, … の順で付けられる．この例では，五つのホストが指定されてあり，$N=5$ となる．pluto のサイト識別子は 0 であり，jupiter のそれは 4 である．

以下に，本ライブラリが提供する関数群を示す．なお，初期化は InitNetlib(f) で行われる．ここで f は用いる構成ファイルのファイル名である．

A. メッセージオブジェクト

- NewMessage(m)—文字列 m を本体とするメッセージオブジェクトを生成し，それを返す．
- GetMessageString(M)—メッセージオブジェクト M のメッセージ本体を文字列にして返す．
- SenderID(M)—メッセージオブジェクト M を送ってきたサイトの識別子を返す．
- DisposeMessage(M)—不要となったメッセージオブジェクト M を解放する．

B. メッセージの送受信

- SendMessageTo(s, M)—メッセージオブジェクト M をサイト s に送信する．
- ReceiveMessage()—届いているメッセージオブジェクトの中で一番古いものを返す．なお，メッセージが何も届いていなければ，メッセージが届くまでブロックされる．
- ReceiveMessageFrom(s)—サイト s から届けられたメッセージの中で一番古いものを返す．もしサイト s からメッセージが届いていなければサイト s よりメッセージが届くまでブロックされる．
- PendingMessage()—任意のサイトから到着したメッセージがあれば真を返す述語．
- PendingMessageFrom(s)—サイト s から到着したメッセージがあれば真を返す述語．

C. 時計

- InitClock(q)—時計の時刻を 0 にすると共に，1単位時間を q 秒とする．
- CurrentClock()—時計が初期化されてから何単位時間経たかを実数値で返す．

D. その他

- GetTotalSites()—全サイト数返す．
- GetMyID()—自分のサイト識別子を返す．

上で説明した関数群を用いることで，アルゴリズム実行に参加するホストおよび参加サイト数をアルゴリズム記述より分離できるので，プログラムの再コンパイルなしに参加ホストや参加ホスト数の変更を，ライブラリ初期化の時指定する構成ファイルを変えることで柔軟に対応できる．

4.5 評価ツールの実現

本ツールの実現について簡単に述べる．サイト間の通信は，インターネットドメインのデータグラムによる非同期通信を用いている．ポート番号は，全サイト共通のあらかじめ固定したものを用いている．各サイトは FIFO のメッセージの待ち行列を持つ．メッセージを受信する時は待ち行列を調べ，メッセージがあればそれを返す．そうでなければ，メッセージ受信のシステムコール recvfrom() を発行する．特定のサイト s からのメッセージを受信する時は，まず待ち行列を調べ，s からのメッセージがあればそれを返す．そうでなければ，s からメッセージが届くまで受信を繰り返す．このとき，s 以外からのメッセージは待ち行列に入れられる．メッセージの送信は，システムコール sendto() を発行する．

次に時計について説明する．システムコール ftime() により，リアルタイムクロックの値を得ることができるので，これを利用する．時計の初期化をする関数 InitClock() では，サイトの時刻を 0 にすると共に，リアルタイムクロックの値を記録する．現在時刻を得る関数 CurrentClock() は，現在のリアルタイムクロックの値から時計の初期化が行われた時のリアルタイムクロックの値を引き，単位時間に変換して返す．

5. 評価ツールを用いてのアルゴリズムの実現法

本章では，4章で述べた関数群を使って，シミュレートすべきアルゴリズムの実現方法を述べる．まずシミュレータの起動方法について述べる．サイトは (1) マスタサイト，(2) アルゴリズムサイトの二種類に分類される．まず最初にマスタサイトがユーザによって起動され，構成ファイルを読んでサイト識別子

が1以上のサイトを遠隔手続き呼び出しで起動する．なお，マスタサイトは構成ファイルの最初に現れるサイト，すなわちサイト識別子が0のものである．サイト識別子が1以上のサイトがアルゴリズムサイト（以降特に断らない限りは単にサイトと呼ぶ）であり，これがアルゴリズムの実行を行う．各サイトはそれらの実行過程を逐次マスタサイトに知らせ，実行が完了すると集計した各種の実行結果をマスタサイトに送る．

各サイトが実行する k-相互排除アルゴリズムの実現は，**図2**に示される構造を持つ．

1. 変数の初期化—アルゴリズムが使用する変数の初期化．
2. サイトの振舞いの決定—能動的動作を行う．相互排除要求を出す，とか臨界領域から出るかなどを関数 SiteBehavior() で決定し，決定に応じた動作を（もしあれば）行う．
3. メッセージの受理—受動的動作を行う．他のサイトから送られたメッセージがあれば受け取り，内容に応じた処理を行う．
4. 2，3をあらかじめ指定された時間繰り返す．

しかし，以上のような動作を行うと計算機は絶えずプログラムの実行（時計を読んで一単位時間経たかどうかのチェックとメッセージの有無のチェックの繰り返しが大部分を占める）を行い，一見 CPU 時間の無駄使いのように見える．これを行う理由を以下に述べる．メッセージ受信でブロックされると，プロセスのモデルで述べた振舞いに正しく従うことができない．なぜなら，モデルでは1単位時間当たりある一定の確率 p で相互排除要求が生じるとしたが，もしサイトが通常状態にある時にメッセージ受理のためにブロックされると正しい確率で相互排除要求の発生が不可能となるからである．また，全サイトがメッセージ受信でブロックされたら，どのサイトも他のプロセスにメッセージを送ることはできず，デッドロックとなる．他の解決法として，Unix の alarm システムコールを用いて一定時間後にシグナルを送る方法が考えられるが，指定した時間経った後にシグナルが送られてくることは保証されておらず，一般には指定した時間より遅れてシグナルが送られてくる[10]．alarm システムコールとシグナルの受理を繰り返している内に時間の誤差は蓄積されてゆく．従って，他の機械と同一の時間の流れを得ることができない．sleep システムコールも同様に，指定した期間の正確なスリープは保証されていない．以上の理由により，我々の方式な有効な方法であることがいえる．しかし，この手法は確かに CPU 時間の浪費である．改良法については，7章で述べる．

```
Algorithm()
{
    初期化
    while(true) {
        SiteBehavior();
        能動的動作記述;
        if（メッセージが届いていない）
            continue;
        メッセージの受信 ;
        受動的動作記述;
    }
}
```

図2 アルゴリズム実現の雛型
Fig. 2 The template of implementation of an algorithm.

5.1 アルゴリズム実現の技法

ここでは，与えられたアルゴリズムの記述より本評価ツールが提供する関数群を用いてC言語でアルゴリズムを実現する際の技法について述べる．アルゴリズムにおいて，「サイト S はサイト T にメッセージ M を送り，M に対する T の応答が N ならば A を行う．」という記述がしばしばなされる．このような形の記述には2通りの意味，(1)メッセージ M を送った後，M に対する応答が T から返ってくるまで S は他のメッセージは処理しない，(2)メッセージ M を送った後，M に対する応答が T から返ってくるまで S は他のメッセージに対する処理を行う，がある．すなわち，その動作がアトミックに行われるのかそうでないのかによって，アルゴリズムの実現法が異なる．明らかに，その記述をそのままの形でC言語で表すことはできないので，変換が必要となる．

(1)アトミックな動作の場合—メッセージオブジェクトの FIFO の待ち行列を用意する．サイト S はメッセージ M をサイト T に送る．これ以降，S にメッセージが到着したらその送り手およびメッセージの種類を調べて期待しているもの以外であれば，待ち行列にそのメッセージを入れる．期待しているメッセージであれば，それに対する処理を行う．そしてその後は待ち行列に入っているメッセージを順次取り出して，それらに対応した処理を行う．

(2)非アトミックな動作の場合—ブール変数 v（初期値は false）とサイト識別子変数 s を用意する．サイト S はメッセージ M をサイト T に送り，v の値を true，s の値を T にそれぞれ設定する．そして普通の動作（サイトの振舞いの決定や他サイトからのメッ

セージ受信）を行う．メッセージを受信して，それが期待しているメッセージNと同じ種類のものであり，vが真かつそのメッセージの送信者がsであれば，あらかじめ定められた動作を行う．

本来なら，与えられたシミュレートすべきアルゴリズムの記述の形を保つべきだが，セマンティックギャップのためにそれが不可能であり，ユーザが手作業で変換しなくてはならない．このことはコーディングミスが入り込む可能性が大きいことを意味しており，より複雑なアルゴリズムの場合では問題となる．

5.2 測定のための機構

ここでは，アルゴリズムを実行させた時の各種データの測定法について述べる．メッセージ数の計数は，計数用のカウンタ変数を用意しておき，メッセージ送信が行われる度にその値を1増やせば良い．次に，時間に関する測定法について述べる．例として，相互排除要求が生じてから臨界領域に進むまでの時間を測定する方法を説明する．サイトで相互排除要求が生じた時に，現在時刻を CurrentClock() で取得し，その値を変数 T_r に記録する．そして，サイトが臨界領域に進む時に，現在時刻と T_r の差を計算すれば，所望の時間が得られる．

6. 実験と考察

本章では，これまでに述べた方法を用いて実現された二つのアルゴリズムの実験を行い，その結果を示す．

6.1 実　　験

実験は次に示す条件で行った．与えるパラメタは，1単位時間の長さ Q（秒），単位時間当たりの相互排除発生確率 p，サイトが臨界領域にいる時間 T_{CS}（単位時間），同時に臨界領域に進むことのできるサイト数 k，用いる k-コータリー S（アルゴリズム K のみ），サイト数 N，シミュレートを行う時間 c（単位時間），である．我々の行った実験では，$Q=1$，$T_{CS}=1$，S は多数決コータリー☆，$c=500$ と固定した．$p=0.001$〜1.0 とし，k，N を次の値の場合に対して実験した：$k=2$ に対し $N=5, 8, 11$，$k=3$ に対し $N=7$，$k=4$ に対し $N=9$．

測定した項目は，各サイトでの臨界領域に進んだ回数，メッセージ送信を行った総数，相互排除要求が生じてから臨界領域に進むまでに要した時間の総和の三つである．そしてこれらより相互排除1回当たりに送信されるメッセージ数の平均を求めた．なお，実験に用いたワークステーションは，日本データジェネラル社 AV-300（オペレーティングシステムは DG/UX 4.32）および同社 DS-7400（オペレーティングシステムは DG/UX 4.02）である．本論文はアルゴリズムの評価自体が目的ではないので，結果の一部のみを示す．$(k, N)=(2, 5), (4, 9)$ のときのメッセージ数についての結果を各々**図3**，**図4**に示す．

6.2 本手法に対する考察

本シミュレーション方式で得られた結果の信頼性に

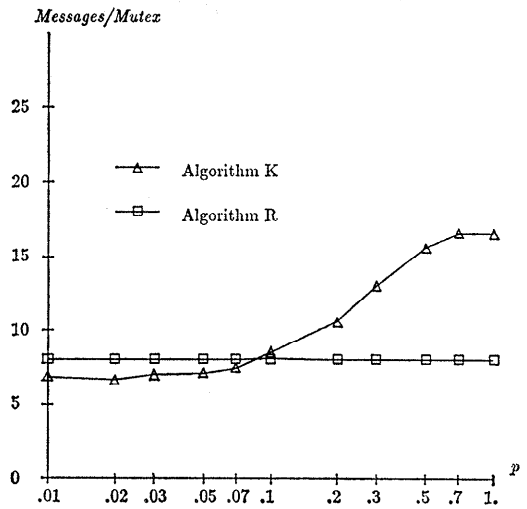


図3 メッセージ数 ($k=2, N=5$)
Fig. 3 The number of messages ($k=2, N=5$).

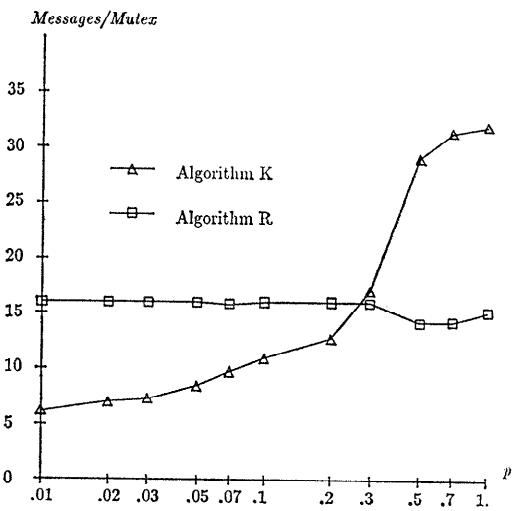


図4 メッセージ数 ($k=4, N=9$)
Fig. 4 The number of messages ($k=4, N=9$).

☆ Uの下のk-多数決コータリーCとは，任意の $\lceil(|U|+1)/(k+1)\rceil$ 個のサイトよりなるコーラムの集合であり，$N \geq k^2$ のときに定義され，$C=\{Q \mid Q \subseteq U, |Q|=\lceil(|U|+1)/(k+1)\rceil\}$ である．

ついて考える．分散アルゴリズムのシミュレーションを行う際には，時間の取り扱いが大きなポイントとなる．サイトの持つ時計はリアルタイムクロックに同期しているので，すべてのサイトの時計は同じである*．ワークステーションでは一般に，複数のプロセスが実行されているため，各サイトのアルゴリズムの実行速度が異なる．しかし，一単位時間を数秒にすることでその差は無視できる**．メッセージの通信速度も一単位時間と比較して無視でき，通信時間のふらつきも無視できる．以上に述べた時計の実現方法により，シミュレーションは次のような性格を持つ：(1)各プロセスが並列に実行され，(2)実時間シミュレーションが可能となる．上で述べたライブラリでは，局所計算時間および，通信遅延が無視できる分散システムを想定したものとなっている．通信遅延が大きな分散システムを想定したシミュレーションを行う場合でも，本手法は容易に対応できる．通信遅延を T_D とするには，送信サイトは送信命令が実行されるとすぐにそのメッセージを受信サイトに送り，受信サイトはそのメッセージが到着した時刻から T_D 経た後にそのメッセージが受信可能であるとすれば良い．局所計算時間が無視できないような分散システムのシミュレーションを行うことも可能である．ある処理に要する時間が T_P であるとすれば，単に T_P だけ時間が経過するのを待てば良い．このため，本シミュレーション技法は，より現実に近い分散システムのシミュレーションが可能となっている．

本手法の欠点は，実験結果の完全な再現性がないことである．これは，アルゴリズム実行の際，バックグラウンドプロセスの影響によるアルゴリズムの実行速度や，メッセージ通信の遅延時間に不確定な要素があるからである．しかし，これらの不確定性は一単位時間と比較すれば無視できるので，実験結果の統計的な再現性はある．完全な再現性を保証するには，1台のワークステーション上で仮想的な分散システムを実現する必要があると考えられる．

7. 結　　論

本稿では，分散アルゴリズムを実験的に評価するための技法について，分散 k-相互排除アルゴリズムを例にして議論した．アルゴリズムを実現するためのツールとして，通信のための手続きなどをライブラリの形で提供し，アルゴリズムを見通し良く記述することを可能とした．また，与えられたアルゴリズムをC言語で実現するための技法についても議論した．そして，二つの分散相互排除アルゴリズムを実現し，それらの評価を行った．

作成したシミュレーションプログラムはライブラリも含めてC言語で記述され，全体で約 4000 行の小さなシステムである．内訳は，約1000行がライブラリ，約400行がアルゴリズムR，約600行がアルゴリズムK，そして残りがマスターサイトの記述や計測部分等である．プログラミングは10日程度で完成した．このように，我々の行ったシミュレーション方式は実現も単純であり，各サイトが完全に並列に動作するという点でも有効な方式であると考えられる．

我々の実現方法では，メッセージ受信および相互排除要求発生等に必要な一単位時間経過の検出を，いわゆる busy-waiting で行っている．このため計算機に与える負荷はかなり大きなものとなった．他のユーザも計算機を利用している場合は，負荷軽減のために以下の改良を加えれば良いと考えられる．まず，メッセージ受信について述べる．Unix には，メッセージが届いた時にシグナルを送る機能があるのでこれを利用する．すなわち，絶えずメッセージが届いているか否かをチェックせずに，シグナルハンドラ中でのみメッセージを受信し，プログラム側であらかじめ用意しておいた待ち行列に入れる．次に，一単位時間が経過するのを検出する方法について述べる．Unix の sleep，alarm システムコールでは不十分なことは前に述べた．そこで，usleep システムコールを用いる．usleep(n) を発行すると，n ミリ秒間プロセスがスリープ状態になる．実現法であるが，usleep(1)を発行した後に時間を調べ，一単位時間経過するまでに十分な時間があれば同様なことを繰り返す．そして，一単位時間経過するのが近付けば，busy-waiting で待てば良い．

与えられたアルゴリズムをそのままの形でC言語で表現できないことがアルゴリズムの実現の際に分かった．本稿では，ある構文についての一変換方法を示した．しかし，手作業での変換では変換ミスをする可能性が高く，得られたプログラムの可読性は大変低い．複雑な分散アルゴリズムの場合に，何らかの問題が見つかった時には，元のアルゴリズムにバグがあるのか

* 各計算機の持つリアルタイムクロックは厳密に同じ時刻を指していないので，各サイトの時計の値は厳密には同一でない．

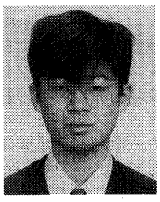
** サイトの実行するアルゴリズムが複雑で時間を要する場合はその限りではない．

コーディングにバグがあるのかを見極めるのは難しい．このため，分散アルゴリズムを記述するための言語とその処理系が是非とも必要であるといえよう．

参考文献

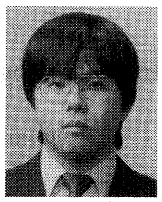
1) Bagrodia, R. L. and Shen, C.-C.: MIDAS: Integrated Design and Simulation of Distributed Systems, *IEEE Trans. Softw. Eng.*, Vol. 17, No. 18, pp. 1042-1058 (1991).
2) Fujita, S., Yamashita, M. and Ae, T.: Distributed *k*-Mutual Exclusion Problem and *k*-Coterie, *Proc. 2nd International Symposium on Algorithms*, Lecture Notes in Computer Science 557, pp. 22-31 (1991).
3) Garcia-Molina, H. and Barbara, D.: How to Assign Votes in a Distributed System, *J. ACM*, Vol. 32, No. 4, pp. 841-860 (1985).
4) 弘田暢幸，梅本秀樹，相原玲二，山下雅史，阿江 忠：分散環境で動作する分散アルゴリズムシミュレーター，日本ソフトウエア科学会ソフトウエア研究会，SW-92-10-5, pp. 31-38 (Apr. 1992).
5) 池川幸徳，山下雅史，阿江 忠：リングネットワークにおけるリーダー選挙アルゴリズムの実験的評価，信学論 (D-I), Vol. J73-D-I, No. 3, pp. 261-268 (1990).
6) Kakugawa, H., Fujita, S., Yamashita, M. and Ae, T.: Availability of *k*-Coterie, *IEEE Trans. Comput.*, (to appear).
7) 角川裕次，藤田 聡，山下雅史，阿江 忠：分散*k*-相互排除のプロトコル，信学技報 COMP 91-28, pp. 57-64 (June 1991).
8) Lamport, L.: Time, Clocks, and the Ordering of Events in a Distributed System, *Comm. ACM*, Vol. 21, No. 7, pp. 558-565 (1978).
9) Lamport, L. and Lynch, N.: Distributed Computing: Models and Methods, in *Handbook of Theoretical Computer Science*, ed. van Leeuwen, J., Vol. B, The MIT Press (1990).
10) 中村 明：オペレーティングシステム構築法—UNIX 詳説—構造編—，丸善 (1986).
11) 永松祥治：木構造ネットワークの負荷分散アルゴリズムについて（広島大学卒業論文）(Mar. 1988).
12) 小野雅弘：分散環境で動くアルゴリズムシミュレーター（広島大学卒業論文）(Mar. 1991).
13) Raymond, K.: A Distributed Algorithm for Multiple Entries to a Critical Section, *Inf. Process, Lett.*, Vol. 30, pp. 189-193 (Feb. 1989).
14) Ricart, G. and Agrawala, A. K.: An Optimal Algorithm for Mutual Exclusion in Computer Network, *Comm. ACM*, Vol. 24, No. 1, pp. 9-17 (1981).
15) 都倉信樹：分散アルゴリズム設計支援ツール，情報処理，Vol. 30, No. 4, pp. 380-386 (1989).

（平成4年8月31日受付）
（平成5年4月 8 日採録）



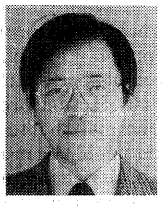
角川 裕次（正会員）

平成2年山口大学工学部電子工学科卒業．平成4年広島大学大学院工学研究科情報工学専攻博士課程前期修了．平成5年同博士課程後期中退．同年広島大学工学部第二類助手．分散アルゴリズム，分散処理の研究に従事．



藤田 聡（正会員）

昭和 60 年広島大学工学部第二類（電気系）卒業．平成2年同大大学院博士課程修了．工学博士．現在広島大学工学部第二類助手．並列アルゴリズムの設計，分散システム等に興味をもつ．電子情報通信学会会員．



山下 雅史（正会員）

昭和 49 年京都大学工学部情報卒業．昭和 52 年同大大学院修士課程修了．昭和 55 年名古屋大学大学院博士課程修了．工学博士．豊橋技術科学大学助手を経て，現在，広島大学工学部第二類助教授．この間，昭和 61 年より約1年間，カナダサイモンフレーザー大学客員助教授．マルチプロセッサの負荷分散問題，画像処理の基礎，組合せ問題の研究に従事．電子情報通信学会会員．

阿江 忠（正会員）

昭和 39 年東北大学工学部通信卒業．昭和 44 年同大大学院博士課程修了．工学博士．東北大学助手，広島大学助教授を経て，昭和 57 年広島大学教授（工学部第二類計算機工学教育科目担当）となり現在に至る．この間，昭和49年より1年間，仏グルノーブル大学客員研究員．現在，主として，分散処理システム（マルチプロセッサおよび LAN）の設計，制作ならびに性能評価の研究に従事．電子情報通信学会，ACM，IEEE 各会員．