並行処理待ち行列網シミュレータ D-SSQ

佐藤 圭・中西 暉・真田 英彦・手塚 慶一

1. まえがき

通信網や交通網などの優れた解析モデルである待ち行列網の解析手法には，理論的解析手法の相補的手段として，シミュレーションによる解析手法がある．待ち行列網シミュレーションは，従来，GPSS等のシミュレーション言語を用いて大型計算機上で実行されてきたが，多量のメモリや計算時間を必要とするため，網規模の縮小等のモデルの変更を余儀なくされたり，少量のサンプル数で我慢させられたりすることが多い．

待ち行列網は複数の待ち行列システムにより構成されており，個々の待ち行列システムは，相互に関連をもちつつも高い独立性をもって並列動作を行なっている．それゆえ，待ち行列網シミュレーションにおいても，このきわめて高い並列性を利用し，複数のプロセッサを用いて並行処理することにより，シミュレーションが高速化できると考えられる．

そこで，近年半導体技術の進歩により低価格かつ高性能で提供されているマイクロプロセッサを用いて並行処理を行なえば，高スループットかつ安価な専用待ち行列網シミュレータが実現できるのではないかとの期待が生まれ，その開発および研究が行なわれており，NTTのNEWTS[1]，慶応義塾大学のKDSS-I[2]，大阪大学基礎工のHASS-QN[3]などが発表されている．本稿で紹介するD-SSQも，大阪大学工学部における新しいタイプのそのような試みの1つである．

さとう　けい，なかにし　ひかる，さなだ　ひでひこ，てづか　よしかず　大阪大学　工学部

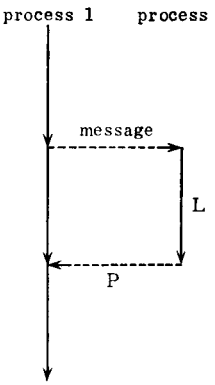

図1　メッセージ・パッシング

2. 待ち行列網シミュレーションの性質

分散処理システムは，分散して存在する複数プロセスが並列動作し，全体としてあるまとまりのある仕事を遂行する．この時のメッセージのやりとりの型をメッセージ・パッシングの概念[4]，[7]を用いて分類してみる．

図1は，メッセージパッシングの概念図を表わしている．図において縦線は論理時刻の進行を表わし，横線はメッセージの伝送を表わしている．論理時刻とは，ある処理と他の処理との順序関係を表わすものである．すなわち，処理Aを行なった後に処理Bを行なう場合に，処理Aのもつ論理時刻が処理Bのもつ論理時刻より早い．

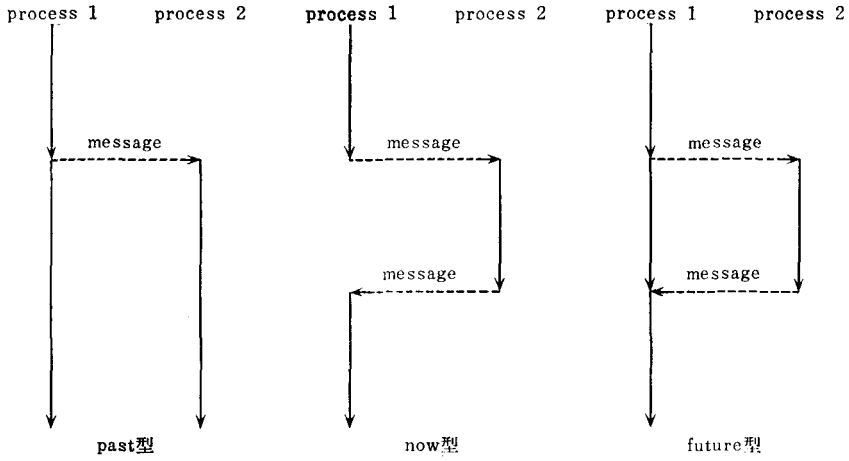

図 2　known 型メッセージ・パッシング

図1ではプロセス1がプロセス2に対しメッセージを送ると，プロセス1が何らかの処理を行なう．同図において，Lはその処理に要する論理時間を意味する．プロセス2が処理を行なった後メッセージをプロセッサ1に対し送る確率がPである．このLがあらかじめ確定しており，かつPが0または1であるものを known 型メッセージ・パッシング，Lが不確定であるか，あるいは $0<P<1$ であるものを unknown 型メッセージ・パッシングと呼ぶことにすると，known 型メッセージ・パッシングには，図2に示すように past 型，now 型，future 型の3種が考えられる．

past型：プロセス2からプロセス1へのメッセージの送信はない．

now型：プロセス1の論理時刻はプロセス2からメッセージが返ってくるまで進まない．

future型：プロセス1がプロセス2からのメッセージを受信したときの時刻は，プロセスに対してメッセージを送信した時刻よりも進んでいる．

unknown 型メッセージ・パッシングに対しては図3に示すように type Ⅰ，type Ⅱ，type Ⅲの3種がある．

type Ⅰ：プロセス2からプロセス1に対し，いつメッセージが送られるかわからない

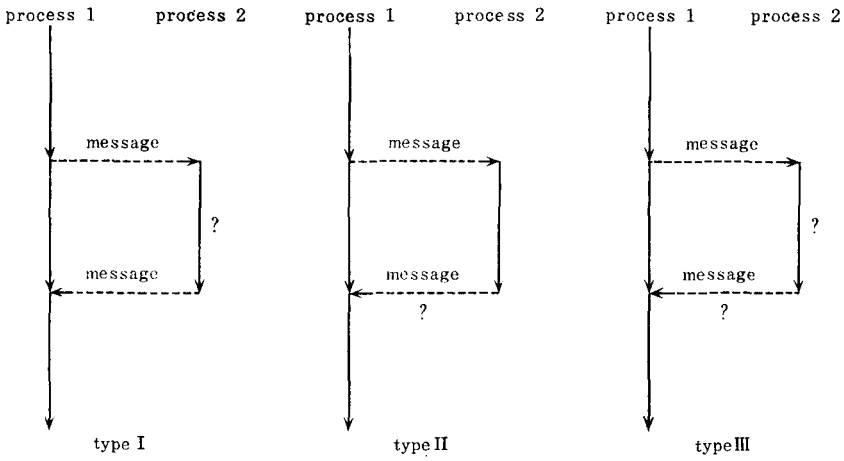

図 3　unknown 型メッセージ・パッシング

表 1　メッセージ・パッシングの分類

P : probability　　L : length

P	L	message passing
0	—	past
1	known	now, future
	?	unknown I
?	known	unknown II
	?	unknown III

typeII：プロセス2からプロセス1にメッセージが送られるかどうかわからない．ただし，送られる可能性のある時刻は確定している．

typeIII：プロセス2からプロセス1にメッセージが送られるかどうかわからない．また，送られるとしても，いつ送られるかわからない．

メッセージ・パッシングの分類を表にまとめると表1となる．

ところで待ち行列網シミュレータはtype IIIのunknown型である．すなわち隣接ノードへ送ったメッセージの結果としてのメッセージが返されてくるかどうかもわからないし，また返されるとしても，それがいつであるかは前もってはわからない．

このようなunknown型のメッセージ・パッシングを含むモデルを並行処理しようとすると，なんらかの制御信号をプロセス2からプロセス1に送り返すことにより，known型に変換する方法が考えられる．しかしながらtype IIのみがfuture型に変換できるものの，一般に，ほとんどがnow型になるため並行動作はおこらず，モデルのもつ並列性をほとんど利用できない場合が多い．そこでunknown型のままで並行処理することが，どうしても必要だということになる．

3. 並行処理待ち行列網シミュレータの分類

並行処理待ち行列網シミュレータとして，実際に実験システムを作成し，実験を通じて論じられた報告が国内では，D-SSQの他に3つある．NTTのNEWTS，阪大基礎工のHASS-QN，慶応大のKDSS-Iがそれである．これらの概略とその特徴を紹介しておこう．

(a) NEWTS

NEWTSは，2段の階層型共通バスにより，19台のノードプロセッサ（NP）と1台のマスタプロセッセ（MP）が接続された構成となっている．NPは交換点の模擬を行ない，MPはNPの実行管理を行なう．

NEWTSの特徴を以下に示す．

(1) 時刻更新方式として，単位時間ごとに同期して時刻を更新する時刻駆動方式を採用している．

(2) 階層型共通バスによりプロセッサを結合し，パラレルDMA伝送で通信を行なう．

(3) 個別メモリ管理方式

(4) マイクロコンピュータ用シミュレーション言語COSMICをもつ．

(5) シミュレーション実行途中にコマンド投入が可能である．

(b) HASS-QN

HASS-QNは12台のNP (Node Processor) と1台のCP (Control Processor) およびCM (Common Memory) が共通バスにより結合され，プロセッサはCMを通して情報交換を行なう構成となっている．

(1) 時刻駆動方式により時刻管理を行なう．

(2) パラメータを与えるのみでシミュレーションモデルが簡単に構築できる．

(c) KDSS-I

KDSS-Iは，マルチリードパケットメモリ (MRPM) と呼ばれる同時読み出し可能な共有メモリを用いて最大64台のプロセッサが結合された構成となっている．

(1) 時刻更新方式として，同期フラグを用いた事

象駆動方式を採用している．

(2) Queue や Server 機能単位としＱＳＶプロセッサと呼ばれる論理的なプロセッサに１対１に割り当てる方式であり，モデルとの対応がつきやすい．

NEWTS および HASS-QN が時刻制御方式として採用している時刻駆動方式は，単位時間内に発生する事象はすべて同一の時刻の事象であると考えて処理する方式である．すなわち単位時間内にはメッセージ・パッシングはおこらないとして，図１のメッセージ・パッシング概念図における論理時刻Ｌを強制的に単位時間とすることによって typeⅡ unknown 型に変換し，さらに単位時間ごとの時刻更新記号という制御信号を送ることにより，future型に変換したうえで，その並列性を引き出そうとする方式である．

しかしながら，事象の発生時刻が単位時刻にまるめられるため精度が限られ，精度を上げるために時刻単位を小さくすると，単位時間内に並行動作するプロセス数が少なくなるという難点がある．しかし，実行制御が簡単であり，同時刻の事象が多数発生するとみなしうる大規模網のシミュレーションには有効な制御方式である．

一方，KDSS-I が採用している事象駆動方式は，事象の変化時点をとらえて時刻を更新する方式であり，プロセス相互で時刻制御を行なうために信号の送受が必要である．フィードバックのない網に対象をかぎれば，比較的容易に past 型または future 型に変換するための有効な制御ができる．

しかし，通信網や計算機の待ち行列網モデルをとり扱うためには，フィードバックのある網を主な対象とせねばならない．

阪大工学部で開発された並行処理型待ち行列網シミュレータ D-SSQ (Distributed System Simulator for Queueing-Network) は，フィードバックのある網を対象として，事象駆動方式による，非同期並行処理システムの，unknown 型のままでの実行制御方式が果たして可能かどうかについての実験的検討を目的としている．

3. D-SSQ の実行制御方式[5], [6]

複数のプロセッサを用いて unknown 型のジョブを並行処理する場合，個々のプロセッサがプロセッサ間の論理時刻の依存関係を無視して独立に処理を行なうと処理結果に論理矛盾が発生する可能性がある．そこで，処理結果の正しさを保証するために実行制御が必要になる．実行制御方式としては，future型に変換して論理矛盾が発生しないように制御する方式[1],[2],[3]と unknown 型のままで論理矛盾の発生の可能性を許して先行実行を行ない，論理矛盾の発生を検出すると直ちに矛盾解消の操作を行なう方式が考えられる．

D-SSQ の実行制御方式は後者の矛盾発生の可能性を許容する制御方式である．すなわち，プロセッサ間通信によるプロセッサ間の従属性を無視して，個々のプロセッサが独立に処理を行なうため処理の並行度は高い．しかし，プロセッサ間通信の頻度が高く矛盾の発生率が高ければキャンセル処理による無効処理量が多くなり処理効率の改善は望めないが，プロセッサ間通信がそれほど頻繁でなければ矛盾の発生回数が少なくなるので並列性の有効利用が図れ，処理効率の改善が期待できる．

このような D-SSQ の実行制御方式を先行制御方式と呼び，これについて詳細に述べる．

4.1 事象の処理履歴の保存

各プロセッサには，他のプロセッサからの影響は unknown であるから，自プロセッサ内の事象で最小の発生時刻をもつものを処理する．プロセッサ間通信の際に発生する矛盾を解消する処理は，過去のある時点のプロセスの状態を復元する操作である．D-SSQ においては，プロセスの現在状態と，さかのぼりたい時刻であるキャンセル時刻から現在までの事象の処理履歴を用いて，キャンセル時刻のプロセスの状態を復元する．

4.2 キャンセル処理

プロセッサは矛盾の発生を検出すると，矛盾解消を行なうためのキャンセル処理を開始する．プロセッサは矛盾発生時刻以後に行なわれた処理を，保存していた事象の処理履歴をたどって，未処理の状態にもどし，プロセッサの状態を矛盾発生時刻直前の状態に復元する．キャンセル対象の事象の中に，他プロセッサへのメッセージをともなう履歴を発見すると，そのメッセージを受信したプロセッサに対して，そのメッセージが無効であることを示すキャンセル要求メッセージを送信する．キャンセル要求メッセージを受信したプロセッサは，キャンセル対象の事象を見いだし，その事象が未処理であれば，その事象と，それ以後にキャンセル要求送信プロセッサから到着した事象を消去する．一方，キャンセル対象の事象が既処理である場合は，その時刻以後に実行されたすべての事象の処理をキャンセルし，キャンセル対象となった事象の直後の事象から処理を再開する．したがってキャンセル処理には以下の3つの場合が考えられる．

(1) モード1　到着メッセージの要求時刻よりもプロセッサの時刻が進んでいる場合

(2) モード2　キャンセル要求を受信した際にキャンセル対象の事象が未処理である場合

(3) モード3　キャンセル要求を受信した際にキャンセル対象の事象が既処理である場合

4.3 確定時刻の検出

キャンセル処理に備えて事象の処理履歴をそのまま保存しておくと，シミュレーションの進行にともない，必要とするメモリ量が無制限に増大する．しかしメモリ量は有限であるので，キャンセル対象となりえない処理履歴を見いだして消去を行なう作業が必要である．その際，統計量も同時に収集する．

このキャンセル対象となりえない処理履歴は，矛盾が発生しても絶対にもどりえない時刻以前の処理履歴であり，この時刻を確定時刻と呼ぶ．すべてのプロセッサの時刻の最小値（システム最小時刻）は確定時刻の上限値となる．

現在，D-SSQ では，CPの台数が少なく，プロセッサ間通信バスがブロードキャスト可能であることから，MPが集中制御型の確定時刻検出を行なっている．

4.4 先行規制 [6]

先行制御方式において処理能力を低下させる原因の1つは，先行過大によるキャンセル処理の頻発が挙げられる．そこで処理の先行に対して適当に制限を行ない，矛盾の発生を少なくし，処理能力の改善を図る．システム全体のプロセッサが先行処理できる時刻の上限を意味する先行規制時刻を定め，プロセッサ間の時刻進行のアンバランスを抑えるグローバル規制方式をとる．

5. D-SSQ による実験とその評価

図4は実際に構成した実験システムのハードウェア構成である．実験システムはシステムの管理を行なう1台のマスタプロセッサ（MP）とノードの擬似動作を行なう5台のコンピューティングプロセッサ（CP）により構成されている．CPの処理には，優先順位の高いものから，メッセージの送受，キャンセル処理，統計処理，シミュレーションの実行がある．MPは，シミュレーション開始時にはシミュレーションパラメータの入力を行ない，シミュレーション実行時には通信バスの管理，確定時刻の検出，先行規制情報のブロードキャスト等を行ない，シミュレーション終了時には統計量の収集，出力を行なう．

D-SSQを用いて先行制御方式の実現性を示し，その有効性を評価するための実験を次の条件のもとで行なった．

(1) シミュレーションモデルはN(=2, 3, 4, 5) 局の完全網としプロセッサにそれぞれ1ノードを割り当てる．

(2) ノードの負荷は均一とする．

(3) ノード数に関係なく各プロセッサの負荷は

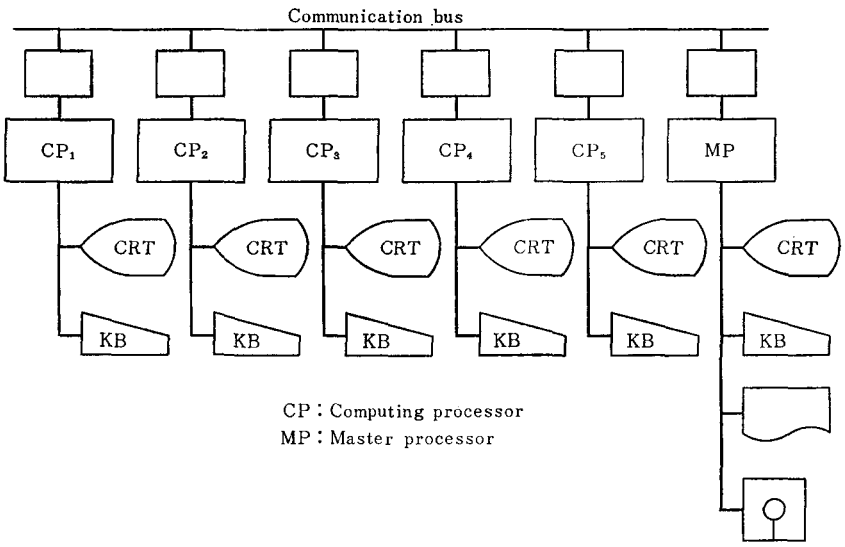

図 4　実験システムの構成

一定とする．

また，処理能力を以下のように定義する．

$$\text{処理能力比}=\frac{\text{マルチプロセッサでの処理速度}}{\text{シングルプロセッサでの処理速度}}$$

ここで，シングルプロセッサでの処理速度とは履歴保存等の分散化オーバーヘッドをまったく必要としない単一プロセッサシミュレータ（純ＳＳＱ）の処理速度である．

図５および図６に処理能力特性を示す．図５は実験システムによる実測値であり，図６は実験システムにおける分散化オーバーヘッド等の値のもとで，プロセッサ台数を実用レベルである40台程度まで増やした際の処理能力を，大型計算機を用いてD-SSQシミュレータのシミュレーションを行なうことにより求めた値である．

これらの図から，処理能力はプロセッサ台数に対して線形に増加することがわかる．しかし，実際にはプロセッサ台数が，通信システムが輻輳するまで増加すると，処理能力は飽和すると思われる．

また，図６において，プロセッサ台数が40台の場合の処理能力が４と，かなり小さいことがわかる．この原因としては，履歴保存等の分散化オーバーヘッドが大きいことが考えられる．実験システムで実測した分散化オーバーヘッドとして，事象の処理時間は約２倍にもなっており，分散化オ

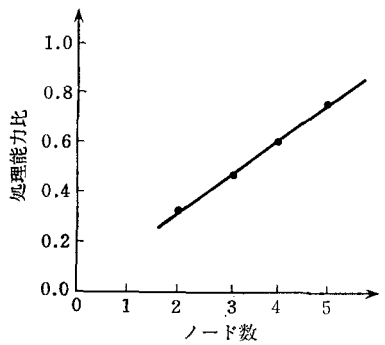

図 5　処理能力特性（実験システム）

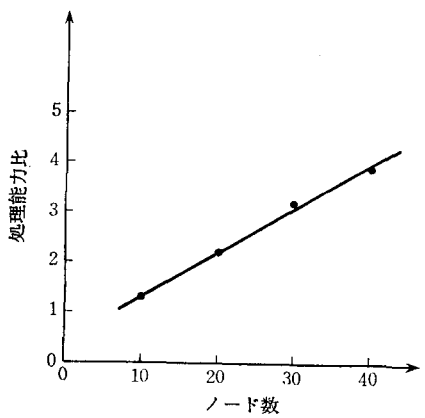

図 6　処理能力特性（シミュレーション）

ーバーヘッドにより大きく処理能力を低下させる原因となっている．

6. 展　望

待ち行列網シミュレーションを並列処理システムで実行する場合のようにunknown型メッセージ・パッシングをともなう並列処理は，その処理効率を上げることがきわめて困難な対象であることがわかる．それにもかかわらず，並列性の利用効果が線形的のびを示していることは，本質的に大きな希望を残してくれた．分散化オーバーヘッドを減らすためのうまい工夫をこらせばよいからである．

参考文献

[1] 稲森，戸田："複数マスタプロセッサを用いた並列形通信網トラヒックシミュレータの評価"，信学論（B），J68-B，1，pp.22-29（昭和60-01）

[2] 中川，小林，相磯："データ駆動型離散系シミュレータKDSS-I"，信学論（D），J65-D，3，pp.386-393（昭和57-03）

[3] 佐竹，上月，西田，宮原，高島："分散型待行列網シミュレータ"，信学技報，EC83-40（1983-12）

[4] Yonezawa, A. and Matsuda, H："Toward Object Oriented Concurrent Programming", Proc. of 6 th RIMS Symposium on Software Science and Engineering, 1984

[5] 渡辺，佐藤，中西，真田，手塚："並行処理事象型待ち行列網シミュレータ D-SSQについて"，信学技報，CS83-102（1983-8）

[6] 佐藤，渡辺，中西，真田，手塚："並行処理型待ち行列網シミュレータ D-SSQについて"，分散処理システム，23-1，1984

[7] 渡辺，佐藤，山口，中西，真田，角所，手塚："並行処理待ち行列網シミュレータ D-SSQにおけるPrologの記述性について"，情処全大，3Q-1，1985

•ミニ•ミニ•　　•O•R•

撤退のORこそ攻めの戦略

•市場の成熟化につれて，革新的な商品が生まれにくくなっている．多様化する消費者の嗜好に対応するため，あるいは競合他社の商品に対処するため，商品ラインは拡がり，多品種少量生産を余儀なくされている．しかも競争激化により，商品のライフサイクルはますます短くなり，収益を生み出す期間も短縮されてきた．つぎつぎとくり出さざるを得ない新商品もヒットするとは限らない．

•商品の芽は意外なところにある．今日のパソコン・ブームを作ったきっかけは，生産した集積回路の使用先を模索するためのマイコン・キットにあった．

•従来のように，将来の予測にもとづいた事業化計画を策定して商品を開発し，設備投資を行なうやり方では，企業の戦略的な展開は期待できない．霧に包まれた林の中のゴルフよりもっと厳しい．林の向うは崖かもしれない．

•また，競争と提携の時代．業界をこえた提携による相互補完，シナジー効果の追求も多いが，はじめるにあたって，将来を明確に展望できるわけではない．

•ここで必要なOR，それは撤退のOR．まず「やってみなはれ」．必要なのは，どの程度のリスクがあり，戦略がうまくいかなかった場合どれほどの有形無形の損失をこうむり，自社の体力でこれを吸収できるかという読みである．戦略の成否は不確実．企業戦略に成長への有効性を狙うものと効率をめざすものがある．前者のための生きたORは，撤退のためのOR，これこそ前向きの攻めのための戦略である．

（山下達哉）