尚美学園大学芸術情報研究　第22号　論文

Web上で動作する高速･ハイブリッドトラヒックシミュレータ

四方義昭

High-speed Hybrid Traffic Simulator working on the Web

Yoshiaki　Shikata

Abstract

Traffic simulation programs have been used to evaluate the performance of communication networks such as the mean waiting time, the loss probability, or the mean number of requests. In this paper, simulation programs for the waiting system（M/M/S）and the loss system（M/M/S(0)）that work on the Web server are studied. Speedup techniques for these simulation programs are discussed in order to shorten the simulation time. Then, the hybrid simulator, which is applicable to both waiting and loss systems, is proposed. The high-speed hybrid simulation technique is applied to the performance evaluation of the limited prioritized processor sharing system.

Key Word

traffic, simulation, performance, waiting time,
hybrid, speedup technique

［抄録］

通信システム内や顧客サービス窓口において発生する待ち行列システムや即時系システムの特性評価を行う場合にコンピュータシミュレーションが用いられる。シミュレーション評価により、待ち行列システムにおける平均待ち時間、最大待ち数、平均系内数、即時系システムにおける呼損率、平均系内数等を求めることができる。本稿ではこのような待ち行列システム（M/M/S）や即時式システム（M/M/S(0)）の品質を評価するためのトラヒックシミュレータの構成法について述べるとともに､その高速化とハイブリッド化（即時･待時式共用化）のためのシミュレータ構成法について論じる。また、高速・ハイブリッドシミュレーション手法を応用した｢処理数制限のある優先度付きプロセッサシェアリングシステム｣のトラヒック評価結果を明らかにする。

［キーワード］
通信システム、待ち行列、特性評価、コンピュータシミュレーション、
高速化、ハイブリッド化、プロセッサーシェアリング

1．はじめに

通信システム内の諸装置や様々な顧客サービス窓口において発生する待ち行列システムや即時系システムの特性評価を行う場合にコンピュータシミュレーションが用いられる場合がある。特に、通信システムにおける「呼び」や各種サービス窓口における顧客の到着形態、システム構成、サービス方式が複雑で数学的な解析が困難な場合や、数学的近似解析結果の近似精度の検証を行う場合にはシミュレーションが不可欠である。シミュレーション評価により、待ち行列システムにおける平均待ち時間、最大待ち数、平均系内数、即時系システムにおける呼損率、平均系内数等を求めることができる。本稿ではこのような待ち行列システムや即時式システムの品質を評価するためのトラヒックシミュレータ構成法について考察する。特に Web サーバ上で動作するシミュレータを考える場合には、ブラウザとのインターフェース構成法とともに、その高速化のためのプログラム構成法について検討する必要がある。本稿では、最も基本的なポアソン入力・指数保留時間待時式モデル（M/M/S）用トラヒックシミュレーションプログラムの基本的な構成法とその高速化手法について述べるとともに、即時式モデル（M/M/S(0)）用シミュレータと共用化する方法（ハイブリッド化）について述べる。また、ハイブリッドシミュレータを応用した「処理数制限のある優先度付きプロセッサシェアリングシステム」のトラヒック評価結果を明らかにする。プログラムは Web サーバ上で動作することを前提とし、Web サーバで一般的に使用される PHP 言語を適用して作成する。また、シミュレーション条件（平均サービス時間、平均到着間隔、シミュレーション終了条件等）はブラウザから入力し、シミュレーション結果（呼損率、平均系内数、平均処理待ち時間等）もブラウザに出力する。

2．トラヒックシミュレーション

通信トラヒックとは、通信ネットワークにおいて接続される単位や通信ネットワーク内を転送される情報の単位、すなわち電話ネットワークにおける通話やインターネット内を転送される信号（パケットと呼ばれ、Web ページやメール情報を運ぶ）などを意味する。通信ネットワークの特性を評価する場合には、このような通信トラヒックの量がネットワークの品質に与える影響を明らかにする必要がある。例えば、インターネットでパケット信号がルータ * 等のパケット中継機器に到着した際、送信回線が使用中（別の信号を送信中）の場合には中継器内部に設けられた「送信バッファ」と呼ばれる「送信待ち合わせ室」で行列を作って順番が回ってくるまで送信を待ち合わせる。バッファ内で送信を待ち合わせている信号は一定のルール（例えば到着した順番）にしたがって送信回線へ送信される（図 1）。送信待ち合わせ室の構成法としては、一定長以上の行列（ キューとも呼ぶ ）を許容しないモデル（有限長の行列）と無限の行列を許容するモデルが考えられる。このような送信待ち合わせ行列モ

デル（待時式モデルと呼ばれる）では、信号が到着する頻度（到着密度と呼ぶ）、信号を送信するために必要となる時間（サービス時間と呼ぶ）によって行列に並ぶ信号の数や送信待ち合わせ時間が変化する。有限長行列モデルの場合には、行列に並ぶことが許されないためサービスを受けないまま系から退去する確率（溢れ率と呼ぶ）が変化する。

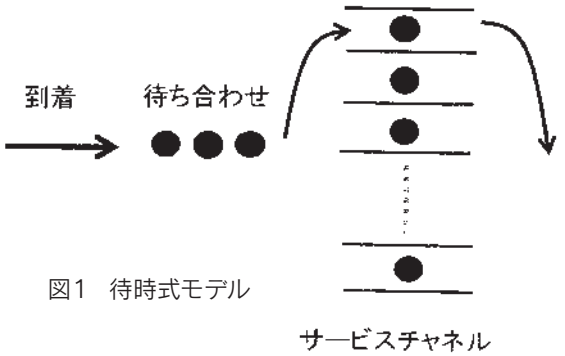

図1　待時式モデル

このような待時式のトラヒックモデルはスーパーマーケットのレジで精算を待つ客、食堂のカウンタで配膳を待つ客、コンピュータの内部で処理を待つタスク等、多くのシステムにおける処理の待ち合わせモデルに応用することができる。

また、電話ネットワークにおいては、呼びが到着する頻度（到着密度と呼ぶ）、通話が継続する時間（サービス時間と呼ぶ）およびネットワーク内で割り当て可能な通信チャネルの数によって接続できない確率（呼損率と呼ぶ）が変化する。すなわち、到着密度が高く、通信チャネル数が少なく、サービス時間が長いほど呼損率が大きくなる（即時式モデルと呼ばれる、図 2)。このようなトラヒックモデルのうち、呼びの到着がポアソン到着であり、サービス時間分布が指数分布に従い、処理窓口 (図１、２ではサービスチャネル) が１といった比較的単純なモデルについては理論的に解析ができ、呼損率、平均系内客数等を計算することが可能である [1]。しかし、到着分布が異なる複数の呼びが到着するモデル、1 つのサービスチャネルを複数の呼びが共有する場合（プロセッサシェアリング [2]）のように厳密解を求めるのが困難なモデルの場合には近似解を求め、この近似計算結果の精度をシミュレーションによって評価する必要がある。さらに、近似解を得ることさえ困難な複雑なシステムも存在し、その場合にはシミュレーションのみによってトラヒック特性を明らかにする場合もある。

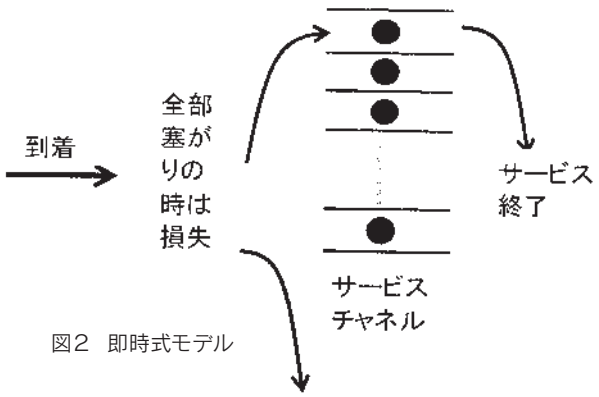

図2　即時式モデル

＊ルータ：インターネット等のＩＰネットワークの中で、ローカルエリアネットワーク（LAN）相互間を接続してパケットを中継転送する装置

３．経過時間の管理方法

コンピュータプログラムを用いてトラヒックシミュレーションを行う場合には、実際に時間を進めながら所定の時間がくると呼の発生、サービスの開始、サービスの終了、呼びの退去等のようなシステムの状態に変化を起こす事象を発生させてシステムの動きを追跡する必要がある。したがって、効率的にシミュレーションを行うにはこの時間の進め方が重要とな

る。一般的にはこのシミュレーション時間の進め方として可変時間増分法が用いられる [3]。可変時間増分法とは、呼びや信号の発生とかサービス（通話）時間の終了のようなシステムの状態に変化を起こす事象が発生するまで時間をスキップしながらシミュレーションを進める方法である（図 3)。可変時間増分法では一定時間単位で時間を進める方法（固定時間増分法）に比べてプログラム構成が複雑となるが、システムの状態に変化を起こす事象の発生が無い場合の時間経過をスキップするためシミュレーション時間を短くすることができる。

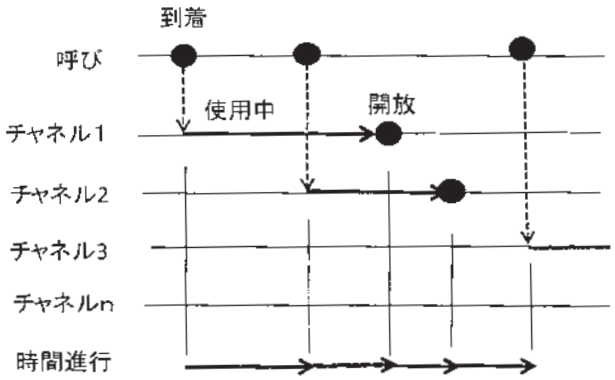

図3　可変時間増分法

4. シミュレーションプログラム

4.1 待時式トラヒックモデルのシミュレーション

4.1.1　概要

Web 上で動作する待時式トラヒックモデルのシミュレーションプログラムは以下の機能を実装する必要がある。

（1）シミュレーション実行条件を入力するための HTML フォームをブラウザに出力する。
（2）ブラウザから入力されたシミュレーション実行条件データをプログラム変数に設定し、必要なデータが全て入力されていることをチェックする。
（3）可変増分法によって時間を進め、所定の時間になるとシステム状態に変化を生じさせる事象を発生させながら、一定の条件（指定した呼数が発生した）を満足させるまでシミュレーションを実行し、必要な評価データを収集する。
（4）シミュレーション結果をブラウザに出力する。

主要なプログラムである（3）において、システム状態に変化を生じさせる事象とは「呼びが到着したことを検出し、チャネルに空きがあればサービスを開始し、無ければ待ち行列に登録する」と「使用中のチャネルのサービス時間 (保留時間とも呼ぶ) が終了したことを検出して空き状態に戻すとともに、待ち行列から呼びを取り出してサービスを開始する」の２つである。したがって、（3）ではこれらの２つの処理を忠実に記述するとともに、これらの事象が発生するまでの時間を進める処理を記述する必要がある。

以下のプログラムでは、平均が 1/A(トラヒック密度が A) の指数分布に従った時間間隔で呼が到着し、それぞれ呼びが平均 ser の指数分布に従った時間のサービスを受けることとする。また、到着した呼びは処理待ち行列の最後尾に登録し、処理待ち行列の先頭に登録された呼びからサービスを実行する方式 (ファーストイン･ファーストアウト方式と呼ぶ) とする。サービスチャネル（回線）数は N、発生させる呼びの総数（シミュレーション終了条件）は call と表す。

4.1.2　基本プログラム

本項で示すプログラムは、実際のネットワークの処理方式に基づいた接続処理を忠実にシミュレーションしたものであり、わかりやすさを優先させたプログラムとなっている [1] 。

```
1 <?php
2  define(INF, 1.0e100);
3  define(EPS, 1.0e-100);
4  $A=$_GET["A"];$N=$_GET["N"];$ser=$_GET["ser"];$call=$_GET["call"];
5  if (isset($A) && isset($N) && isset($ser) && isset($call)){
6    $A=1.0/$A; $nn=0; $nn1=0; $q=0;
7    srand((float)microtime() * 1000000) ;
8    $aa=(double)rand()/(1.0 + getrandmax());
9    $tm = - $A * log(1.0 - $aa);
10   for ($k = 1;$k <= $N; $k++) {
11      $sn[$k] = 0;  $tn[$k] = INF;
12    }
13  // 開始
14   while ($nn <= $call) {
15     // 到着
16     if (abs($tm) < EPS) {
17      $aa=(double)rand()/(1.0 + getrandmax());
18      $tm = - $A * log(1.0 - $aa);
19      $nn++;
20      $q++; $tq[$q]=$tt;
21     }
22     // サービス
23     for ($k = 1; $k <= $N; $k++) {
24      if (abs($tn[$k]) < EPS || $sn[$k]== 0) {
25        if ($q > 0) {
26          $sn[$k] = 1;
27          $aa=(double)rand()/(1.0 + getrandmax());
28          $tn[$k] = - $ser * log(1.0 - $aa);
29          $tw = $tt - $tq[1];
30          $x = $x + $tw;
31          $x2 = $x2 + $tw * $tw;
32          $nn1++;
33          $q--;
34          if ($q > 0) {
35            for ($i = 1; $i <= $q; $i ++) {
```

```
36              $tq[$i] = $tq[$i + 1];
37            }
38           }
39        } else {
40          $sn[$k] = 0;    $tn[$k] = INF;
41        }
42      }
43    }
44  // 次に状態変化の起る時刻を求める
45    $at = INF;
46    if ($tm < $at) $at = $tm;
47    for ($k = 1; $k <= $N; $k++) {
48      if ($tn[$k] < $at)  $at = $tn[$k];
49    }
50    // 時間を進める
51    $tm = $tm - $at;
52    for ($k = 1; $k <= $N; $k++) {
53      $tn[$k] = $tn[$k] - $at;
54    }
55    $tt =$tt + $at;
56  }
```

2、3 行は十分に大きな値と小さな値の定義であり、シミュレーションを進める際の時間管理のために使用する。4､5 行では､連想配列 $_GET[] から Web ブラウザ上で入力されたシミュレーション条件データを取り出してプログラム変数に設定し、全て入力されているかをチェックする。6 行ではプログラム変数を初期化する。この中で､$nn は到着した呼びの数を、$nn1 はサービスが終了した呼びの数を、$q は待ち行列の長さ (処理を待っている呼びの数) を表す。また、入力したトラヒック密度 A（プログラム内では $A）を逆数に変換することにより平均到着間隔値を得ている。8 行では、rand ( ) 関数によって 0 から getrandmax() 関数により求まる最大値までの乱数を発生させ､それを（1 ＋getrandmax()）で割ることによって 1 より小さい乱数を生成する。7 行では実行する度に異なる乱数を生成するための「種」を発生する。ここで生成された乱数値を使って 9 行目で最初の呼びが発生するまでの時間間隔を求める。時間間隔は平均 1/$A(トラヒック密度が $A) の指数分布に従うことを仮定しており、指数分布の逆関数を用いて計算している。11 行目に示した配列のうち、$sn[k] は番号 k のチャネルの空・塞状態を、$tn[k] はチャネル k の残りサービス時間を表す。10-12 行で、全ての通信チャネルについて $sn[] を空き状態に、$tn[] を INF（十分大きな時間）に初期設定する。残りのサービス時間を「十分大きな時間」にするのは、シミュレーションが進行してサービス時間が減っていっても「0」（サービス終了事象が発生する条件）にならないくらい大きな値とするためである。後で解説するように 35 行で $tn[] を参照してサービス時間が

終了したことを検出して次の変化点とする。このため未使用チャネルについては十分大きな値とすることによって、このような変化事象が発生しないようにしなくてはならない。

14 行から 56 行までが可変増分法で時間を進めながらシミュレーションを実行するプログラムである。14 行ではシミュレーションの終了条件（到着した呼の数がシミュレーション条件として入力した値、すなわち $call となるまで）をチェックし、その条件が満たされるまで 55 行までの処理を繰り返し実行させる。16 行では呼びが発生する条件（発生間隔が 0 に限りなく近くなったこと）が整ったことを検出して到着処理を開始する。到着した呼びは全て一旦待ち行列に登録されてサービスが開始されるのを待つ。すなわち、19 行で到着数を加算し、20 行で待ち行列への登録数（処理を待つ呼びの数）を加算するとともに、到着時刻を $tq[$q] に設定する。ここで $q は待ち行列の先頭から $q 番目に並んでいることを示している。ほかに待っている呼びが存在せず、チャネルに空きが存在する場合には、その後のサービス処理の中で即時的に処理される。23―28 行でチャネル 1 から $ N(シミュレーション条件として指定したチャネル数) まで順次サービスが終了したかまたは元々空きであったかをチェックし、該当チャネル（番号は $k）が見つかれば処理待ち行列から呼びを取り出してそのチャネルを割付けて「使用中」（$sn[$k]=1）にしたうえで、平均値 $ser の指数分布に従った乱数を発生させ、その値をサービス時間として $tn[$k] に設定する。

また、サービスを開始した呼について、待ち行列内での処理待ち時間とその 2 乗値を計算し、累計値に加算する（29－31 行）。シミュレーション終了後にこの累計値を処理された全ての呼数で割り算することにより平均値と分散値を計算する。33 行から 38 行では、待ち行列から先頭の呼を削除するとともに、待ち行列の順序にしたがって到着時間を管理する配列を 1 データ毎に配列の先頭に向かって移動させる。40 行ではサービスが終了したチャネルを空き状態とし、残りのサービス時間を INF とする。これは 11 行目の処理と同じ理由による。44 行－55 行で次の変化事象（呼びの発生またはサービスの終了）が発生する時刻までの時間間隔（時間増分）を求め、その時刻まで実行時間を進める。まず、45 行で時間増分を INF に初期設定する。次に、46 行から 49 行で次の呼が発生するまでの時間（$tm）、各々のチャネルのサービスが終了するまでの時間（$tn[]）のうち最小の時間を求め、その時間を増分（$at）とする。すなわち、51 行から 55 行で $tm、$tn[] を $at 時間分短縮するとともに、シミュレーションの経過時間 $tt を加算する。

4.1.3　高速シミュレーションプログラム

（1）処理概要

ここで解説するプログラムは、4.1.2 項で説明した待時式の基本プログラムを高速に実行できるように改善したものであり、約 1/4 から 1/10 程度（シミュレーション条件によって異なる）に実行時間を短縮することができる。Web サーバ内の PHP プログラムはインタープリタ方式で動作する。このため、コンパイル形式のプログラミング言語で開発した場合よりも動作速度が遅くなる可能性があり、高速化対策が必須である。

実行時間を短縮するために基本プログラムから以下の処理を変更している。

・基本プログラムでは「どのチャネル」が使用中であり、「どのチャネル」の残りサービス時間がいくらであるか、というデータをそれぞれ専用の配列（インデクス値はチャネル番号）を用いて管理している。しかし、シミュレーションを行う場合にはこのように実際にサービスを行っているチャネル毎の状態を管理しなくとも、システム状態の変化を検出するために必要なデータのみを管理すればよい。したがって高速プログラムでは「サービス中チャネルと空きチャネルの数」および「サービス中チャネルの残り時間」のみを管理することとする。管理するデータの中から「どのチャネルが（の）」というトラヒックシミュレーションには不必要な情報を削除することにより処理負荷を軽減することができる。なお、空きチャネル数は「全チャネル数 - サービス中チャネル数」で求めることができるが、直接空きチャネル数を管理して配列のインデックス値として使用することにより処理速度が改善される。
・待ち行列から呼びを取り出してサービスを開始する際には、空きチャネル数が 0 でない場合にサービスが可能と判断する（ 全てのチャネルの空き塞がり状態をチェックしない ）。これにより、サービス開始時の空きチャネルハント処理が不要となる。
・サービス中のチャネルについては、残りのサービス時間が少ない順に行列を形成して管理する。これにより全てのチャネルについて残りのサービス時間の最小値を求める処理（基本プログラム中の 47 行から 49 行）が不要となる。また、基本プログラム中の 52 行から 54 行で全てのチャネルについて実行している時間減算処理を、サービス中のチャネルについての時間（行列で管理している時間）のみを減算すればいいことになり、処理負荷が軽減される。
・時間の進行に伴って次に実行すべき変化事象を示す「イベントフラグ」を導入する。
・サービスが終了した場合には、空きチャネル数と使用中チャネル数をカウントするのみであり、サービスが終了したチャネル番号を特定する必要が無いため、基本プログラム中の 24 行から 26 行の処理負荷は軽減される。

（2）プログラム構成

高速シミュレーションプログラムの構成（ ブラウザとのインターフェース部を除く）を以下に示す。

```
1 <?php
2  define(INF, 1.0e100);
3  $A=$_GET["A"];$N=$_GET["N"];$ser=$_GET["ser"];$call=$_GET["call"];
4  if (isset($A) && isset($N) && isset($ser) && isset($call)) {
5   $A=1.0/$A; $nn=0;$nn1=0;$tw=0;$x=0;$x2=0;$q=$N;$qw=0;$qu=0; $flg=INF; $timer
= array();
6   srand((float)microtime() * 1000000) ;
7   $aa=(double)rand()/(1.0 + getrandmax());
8   $tm = - $A * log(1.0 - $aa);
9   $timer[0] = INF;
10   // 開始
11   while ($nn <= $call) {
```

```
12    if ( $flg == 0 ) {
13      $aa=(double)rand()/(1.0 + getrandmax());
14      $tm = - $A * log(1.0 - $aa);
15      $nn++;
16      $qw ++; $tqw[$qw]=$tt;
17    }
18    if ($flg == 1) {
19      $q++;
20      $qu--;
21      $aki =  array_shift($timer);
22      if ( $qu == 0 ) $timer[0]=INF;
23    }
24    // サービス開始
25      if ($qw>=1 && $q >= 1){
26        $aa=(double)rand()/(1.0 + getrandmax());
27        $timer[$qu] = - $ser * log(1.0 - $aa);
28        sort($timer, SORT_NUMERIC);
29        $qu++;
30        $tw = $tt - $tqw[1];
31        $x = $x + $tw;
32        $x2 = $x2 + $tw * $tw;
33        $nn1++;
34        $q--;
35        $qw--;
36        if ($qw > 0) {
37          for ($i = 1; $i <= $qw; $i ++) {
38            $tqw[$i] = $tqw[$i + 1];
39          }
40        }
41     }
42  // 次に状態変化の起る時刻を求める
43   if ($timer[0] < $tm)  {
44     $at = $timer[0]; $flg=1;
45   } else {
46     $at = $tm; $flg=0;
47    }
48   // 時間を進める
49   $tm = $tm - $at;
```

```
50    for ($k = 0; $k < $qu; $k++) {
51      $timer[$k] -= $at;
52    }
53    $tt += $at;
54  }
55 ?>
```

2-9 行では基本プログラムの場合と同様に、シミュレーション条件のチェックと設定・プログラム変数の初期化等を行っている。この中で、$q は空きチャネル数を、$qu は使用中のチャネル数を表す。9 行では残りサービス時間を管理する配列の先頭要素を初期化（十分大きな値）する。これはチャネルが使用されていない場合にシミュレーション時間が正常に進行しなくなるのを防止するためである（44 行)。

11 行から 54 行までが可変増分法で時間を進めながらシミュレーションを実行するプログラムである。11 行ではシミュレーションの終了条件をチェックし、その条件が満たされるまで 54 行までの処理を繰り返し実行させる。12 行で呼びが発生する事象（イベントフラグが 0）が発生したことを検出して到着処理を開始する。15 行目で到着呼数を加算したあと、16 行で処理待ち行列に入れる。すなわち、待ち数 $qw を加算するとともに、待ち数をインデックス値とする配列 $tqw[] に到着時間を設定する。18 行から 20 行ではサービスが終了したこと（イベントフラグが 1）を検出してサービス中チャネルの数 $qu を減算し、空きチャネル数 $q を加算する。

21 行でサービス中チャネルのサービス時間を管理する配列の先頭要素（$timer[0]: サービスが終了したチャネルのデータ）を削除し、$aki に入れるが、このデータは以後使用されない。22 行目では使用中のチャネル数が 0 となった場合に $timer[0] を INF に設定する (9 行目の処理と同様)。25 行目で空きチャネル数（$q）が 1 以上であり、かつ処理待ちの信号が待ち行列の中に存在すること（$qw>=1）を確認し、26,27 行で平均値 $ser の指数分布に従ったサービス時間を計算して $timer[$qu] に設定する。ここで、$timer[] は「残りサービス時間をサービス中の呼びの数分だけ管理する配列」であり、$qu はサービス中のチャネルの数を示している。この配列の最後に新しくサービス中となった呼びのサービス時間が設定される。28 行では $timer[] を小さい順にソートする。すなわち、「サービス中の呼びに関して、サービス時間が終了するまでの最短時間が常に $timer[0] に表示されている」こととなる。29 行でサービス中のチャネル数を加算する。30 行ー 32 行で処理待ち時間とその 2 乗値の合計を計算すし、シミュレーションが終了した際に平均待ち時間とその分散値を計算する。また、サービス呼数の加算（33 行）、空きチャネル数の減算（34 行）、処理待ち呼び数の減算（35 行）を行ったあと、36 行から 40 行では処理待ち呼びの到着時間を管理する配列のデータを前に進める。

43 行－47 行で次の事象が発生するまでの時間を計算するが、「サービス中の呼びに関して、サービスが終了するまでの最短時間が常に $timer[0] にある」ため、この値と次の呼びが生起するまでの時間のうち小さいほうを進める時間とすればよい。

49-53 行では次の呼びが発生するまでの時間間隔とサービス中の呼びについてのサービス

時間を時間増分減算する。

4.2 即時・待時式ハイブリッドシミュレータ

4.2.1 概要

呼びの到着とサービス処理方式が同一のトラヒックモデルにおいて、即時式と待時式それぞれについてトラヒック評価を行う場合、即時・待時式ハイブリッドシミュレータを準備しておくと、モデルの処理方式の変更に迅速に対応することができる。

即時・待時式ハイブリッドシミュレータは、待時式シミュレータ（4.1 節参照）に以下の機能を追加することにより容易に実現することができる。

（1）即時モデルか待時モデルかを指定するフラグ（$sokuji）を設け、シミュレーションパラメータとしてブラウザ上で指定できるようにする。

（2）呼びが到着した時、即時処理でかつ通信チャネルに空きが無い場合には呼損として待ち行列に登録せず（または登録したあと待ち行列から削除し）、その他の場合には処理待ち行列に登録する。

4.2.2 プログラム

（1）基本待時式プログラムへの機能追加

4.1.2 項で述べた基本の待時式シミュレータに以下の処理を追加すればよい。

(a) 23 行　$busy = 1;（チャネルビジーフラグをオン (1) にする）

(b) 25 行　$busy = 0;（空きチャネルが見つかった場合にビジーフラグをオフ (0) にする）

(c) 44 行

```
if($sokuji==1 && $busy==1) {
    $koson++;
    $q--;
}
```

（即時式の場合（$sokuji==1）で、かつ空きチャネルが無い場合（$busy==1）には呼損とし（$koson++）、待ち行列内の要求を削除する（$q--））

（2）高速待時式プログラムへの機能追加

4.1.2 項で示した高速待時式高速シミュレーションプログラムの場合には、16 行目を以下のように変更することにより実現できる。

```
if ($sokuji==1 && $q == 0){
        $koson++;
}   else {
        $qw++; $tqw[$qw]=$tt;
}
```

即ち、即時式の場合（$sokuji==1）で、かつ空きチャネルが無い場合（$q==0）には呼損とする（$koson++）。

4.2.3 応用例

（1）処理数制限のある優先制御プロセッサシェアリングモデル [4]

単一サーバシステム内に2つの処理要求クラス（クラス 1 またはクラス 2）が存在し、要求が到着した際に n1 個のクラス 1 要求と、n2 個のクラス 2 要求（到着した要求も含む）が存在すると仮定する。このようなシステムにおいて、m * n1 + n2 ≦ C の場合、クラス 1 の要求はサービス容量の m /（m * n1 + n2）のサービスを、クラス 2 の要求はサービス容量の 1 /(m * n1 + n2)のサービスを受け、それ以外の場合(m * n1 + n2 ＞ C)は、到着した要求は、対応するクラスの待合室で待ち行列に入れるか（待時式）、サービスを拒否される（即時式）。ここで、m (>=1) はクラス 1 要求とクラス 2 要求のサービス比を、C (<=∞) はサービス容量 (Service-facility capacity) を表す。

（2）ハイブリッドシミュレーション

待時式・即時式の処理数制限のある優先制御プロセッサシェアリングモデルのハイブリッドシミュレーションプログラムを作成し、トラヒック特性を評価した結果を以下に示す。本評価では要求の到着分布としてアーラン二次分布を、またサービス時間分布として超指数分布を仮定し、サービス比（m）は 2 としている。得られた値は 10 回のシミュレーション結果の平均値であり、95%信頼区間はマーカの範囲に含まれる。

(a) 待時式

無限容量の待合室が準備されている待時式システムにおける平均滞在時間とサービス容量との関係の評価例を図 4 に示す。待ち行列内での平均待ち時間（クロスマーカで示す）と、サーバ内での平均滞在時間の両方を（円マーカで示す）評価している。クラス 1 要求の待ち行列内の平均待ち時間は、クラス 2 要求の値よりも大きい。これはクラス 1 要求のサービスに必要とされるプロセッサ容量を確保できる確率がクラス 2 要

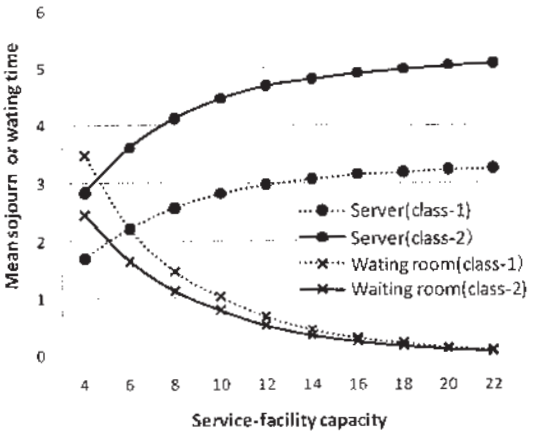

図4　平均滞在時間と待ち時間（平均サービス時間＝1、到着率＝0.4）

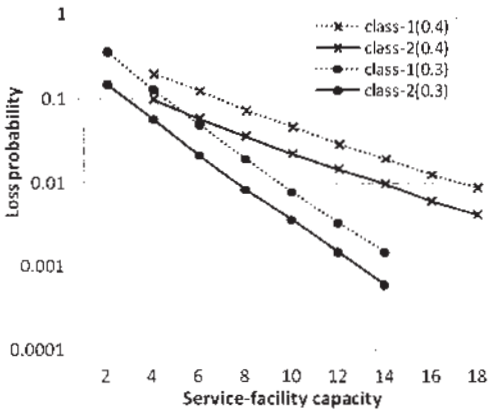

図5　呼損率（平均サービス時間＝1、到着率＝0.3or0.4）

求の場合より小さいためである。一方、クラス 1 要求のサーバ内平均滞在時間は、クラス 2 要求の値よりも小さい。これはクラス 1 要求に割り当てられるプロセッサ容量が大きいため、一旦サービスが開始されると滞在時間は短くなるためである。

(b) 即時式システム

図 5 に呼損率とサービス容量 C の関係の評価結果を示す。これらの評価結果から以下のことがわかる。

・呼損率の対数値は、サービス容量の増加とともに直線的に減少する。

・クラス 1 要求（点線で示す）とクラス 2 要求（実線）の呼損率の対数値はほぼ同じ比率で減少する。

・到着率が 0.3 の場合の呼損率は（ラウンドマーカで示す）、到着率 0.4 の場合（クロスマーカで示す）よりも、サービス容量の減少の影響を強く受ける。

図 6 は、平均滞在時間とサービス容量との関係の評価結果を示している。クラス 1 要求の平均滞在時間（点線で示す）はクラス 2 要求（実線で示す）の値よりも小さい。到着率が 0.4 の場合には、2 つのクラスの要求間の平均滞在時間の差は、サービス容量の増加とともに大きくなる。一方、到着率が 0.3 の場合には、その差はサービス容量に関係なくほぼ一定である。

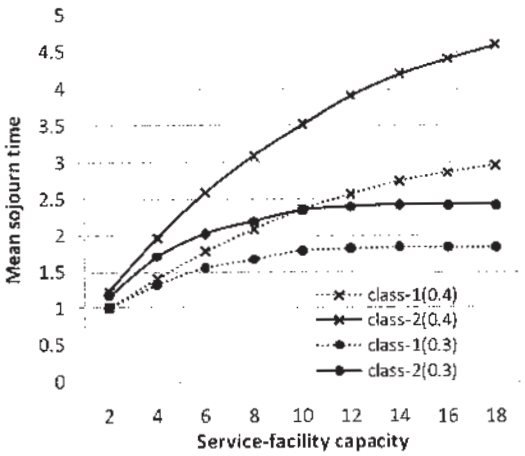

図6　平均滞在時間（平均サービス時間＝1、到着率＝0.3or0.4）

5. 結論

待ち行列システム（M/M/S）や即時式システム（M/M/S(0)）のトラヒック特性を評価するためのシミュレーションプログラム構成法について述べた。まず、通信チャネル毎の空塞（使用中か否か）と使用中チャネルの残りサービス時間を管理し、シミュレーションの経過時間に合わせてその変化をトレースする従来型のシミュレーション方式を Web 上で動作するシミュレータとして実現する方法について述べた。次に、シミュレーション速度を上げるために、チャネル毎の空塞は管理せず、空き通信チャネルの数のみを管理し、使用中チャネルの残サービス時間を常に昇順に並び変えておいてシミュレーションを進める方法を提案し、従来型のシミュレータの場合の処理時間を 1/4 ― 1/10 に短縮できることを明らかにした。さらに、待時式のシミュレータに簡単な機能追加を行うことにより、即時・待時式ハイブリッドシミュレータを作成できることを明らかにした。また、本稿で示した高速型ハイブリッドシミュレータの構成法をプロセッサシェアリングシステムの特性評価用シミュレータに実際に応用して数々の特性評価を行っていった結果を明らかにした。

参考文献

[1]「情報通信トラヒック」秋丸春夫、川島幸之助 , 1990 年 8 月 , オーム社
[2]「An Approximate Formula for a GI/G/1 Processor-Sharing System」, K.Hoshi, Y.Takahashi and Y.Shikata, International Conference on Operations Research, September 1 - 3, 2010, Munich
[3]「C によるシミュレーションプログラム」石川宏 , 1994 年 8 月 , ソフトバンク KK
[4]「Performance Evaluation of Prioritized Limited Processor-Sharing System」, Y.Shikata, W.Katagiri, and Y.Takahashi, WASET International Conference ICCEL2012, June 11-12, 2012, Copenhargen.