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

2003－EVA－6 (7)
2003／5／30

効果的な性能改善策の策定に必要な

シミュレーションプログラムの構成と開発の留意点

株式会社 アイ・アイ・エム 河野 知行

各種のパフォーマンスツールを利用することにより、システムの稼動状況を示すパフォーマンスデータを取得は容易になった。それらのパフォーマンスデータを解析し、ボトルネック箇所を指摘するのは容易であるが、改善効果を判定するには待ち行列などの知識を必要とする。最適な性能改善策を検証する際、シミュレーションプログラムを活用する方法も有効である。本研究では、性能改善の方策を検証する際に必要となるであろうシミュレーションプログラムの構造と、その開発の留意点などを明確にする。

Understand a structure of Simulation Program, and how to develop it.

Tomoyuki Kawano, IIM Corporation

We can easily collect performance data, when we use performance measurement tool on any platforms. Those performance data may lead us to know which part of system resource may become a performance bottleneck and what kind of tuning actions we should take. However do so, it is very difficult to figure out which tuning action may be most effective among these. If we have enough knowledge of Queuing Theory, it may help us choose most effective one. In some case, simulation program also helps us to choose best tuning action. This paper describes our experience to develop simulation program which helps us to understand a theory of simulation program.

はじめに

シミュレーションとは、実システムで発生した問題を擬似環境で再現しようとするものである。擬似環境で問題を再現することができれば、問題解決に必要な対応策を立案するのに役立つ。また、それら対応策の効果も容易に判定できる。

性能評価の分野では、シミュレーションを利用した問題分析手法が活用されている。しかし、専用のソフトウェアパッケージは高価であり、簡単には使用できないのも現状である。本研究では、性能評価の分野で使用されるシミュレーション技法を整理し、シミュレーションプログラムを自作し性能評価に活用することを考える。

シミュレーションプログラムの構造を考える前に、シミュレーション用語について考察する。性能評価の分野でシミュレーションを使用するには、「資源」、「処理」、「要求」、「事象」の４つの要素を明確に定義する必要がある。

「資源（Resource）」には、ＣＰＵやディスク装置などのハードウェア資源と、バッファやサブプログラムなどのソフトウェア資源がある。何れも、性能を阻害する要因（ボトルネック）になる可能性がある。

シミュレーションを行うレベルに応じて、資源を定義する必要がある。前述のＣＰＵやディスク装置などは、一つのシステムを構成するハードウェア資源である。このレベルのシミュレーションでは、単一システム内の動作を考察するものであろう。クライアント、Ｗｅｂサーバ、ＡＰサーバ、ＤＢサーバなどを資源として定義するならば、ネットワークで接続されたコンピュータ群のシミュレーションを行うものであろう。またＣＰＵチップ、システムバス、ＳＣＳＩ制御回路などのレベルでのシミュレーションも可能である。このようにシミュレーションを行う対象範囲に応じて、資源は決定される。

「処理（Process）」は、定義された資源を如何なる順序でアクセスするか定義するものである。資源をＣＰＵやディスク装置などとした単一システム内のシミュレーションでは、この処理はアプリケーションプログラムの動作を示したものであろう。システムで一つのアプリケーションプログラムだけが実行されている訳ではない。この処理の定義においては、複数のアプリケーションが連携して動作する様子を定義することになる。また、複数システム群の連携をシミュレーションする場合は、システム間の通信状況なども処理の一部として定義する。

「要求（Request）」は、処理を起動するタイミング（頻度）を定義する。Ｗｅｂサーバのシミュレーションを行う場合、サーバ構成を資源で、アプリケーション動作を処理で定義する。また、そのＷｅｂサーバの負荷を決定するクライアントからのＨＴＴＰ要求の発生頻度などを定義するのが、ここで言うところの要求である。

シミュレーションプログラムは、「要求」により起動される「処理」により生じる「資源」の状態変遷の様子を記録する。この資源の状態変化を生じさせるタイミングのことを「事象（Event）」と呼ぶ。

時間の記録

シミュレーションプログラムは、「要求」が発生した時に「資源」が「処理」で定義された順にアクセスされる過程で生じる「事象」を「追跡（Trace）」するものである。要求の処理経過を追跡するにあたり、一番大切なのが、事象の発生時刻を正確に記録することである。

一回のアクセスで１０ミリ秒の動作時間が必要なディスク装置を、１秒間に２０回アクセスすることを考えてみよう。１秒間に２０回アクセスされると言うことは、この

ディスク装置は５０ミリ秒ごとに１回アクセスされることになる。

最初のディスク装置アクセスが０時０分０秒に行われたとすると、次にシミュレーションすべき事象は、ディスク装置アクセスの完了か、次の新たなディスク装置アクセスの生成かの何れかである。ディスク装置のアクセスは１０ミリ秒で完了し、ディスク装置のアクセス時間間隔は５０ミリ秒であるため、この場合、次にシミュレーションすべき事象はディスク装置アクセスの完了の事象となる。

ディスク装置アクセスの完了事象をシミュレーションする際には、シミュレーションの時刻は０時０分０．０１秒に更新される。同時に、ディスク装置の動作時間とアクセス回数の累積値が更新される。その４０ミリ秒後に、ディスク装置をアクセスする新たな要求が生成される。この時、シミュレーション時刻は０時０分０．０５秒に更新される。

シミュレーション動作の概要は以上のようなものである。ここでは、ディスク装置の動作時間を１０ミリ秒、アクセス時間間隔を５０ミリ秒とした。しかし、これらの値は平均値であり、シミュレーションを行っている過程において、これらの値は乱数で決定される。つまり、常に上記の順序でシミュレーションが進行する訳ではない。

ポアソン分布に従った制御

前述したようにディスク装置の動作時間（１０ミリ秒）やディスク装置アクセスの生成時間間隔（５０ミリ秒）は平均値である。個々の事象制御を行う際、これらの値は乱数で決定される。つまりランダムである。シミュレーションが進行するに連れ、それらの値の平均値は指定された平均値となる。

シミュレーションで使用する乱数は、ポワソン分布に従う。通常のプログラム開発環境で提供されている乱数生成機能では、０から１の範囲で乱数を得ることができる。その乱数生成機能で得られた値の自然対数を求め、利用者が指示した平均値を掛け算すればポイソン分布に従った乱数を得ることができる。ＶＢ（Visual Basic）言語で書くならば、次のような計算式となる。

時間間隔=ABS(平均時間間隔*LOG(RND))

ＶＢの乱数関数であるＲＮＤでは０（ゼロ）が生成されることがあるが、低が０の自然対数は求められない。このため、低が０の場合には、特別な処理が必要となる。

シミュレーションでは、この乱数を用いたロジックが多用されるため、シミュレーション結果が安定化（各乱数が平均値に収束する）するまでに時間を要する。

シミュレーションプログラムでの記憶域

シミュレーションプログラムでは、次のような記憶域を準備する必要がある。

時計：事象のシミュレーション時刻を記憶する。ここに記憶された時刻を基に、次にシュミレーションすべき事象を決定する。

要求生成時刻：シミュレーションを実行する過程で、次の新たな要求を生成すべき時刻を記録する。シミュレーションは０時０分０秒から開始されるが、この領域の初期値はゼロであり、シミュレーションの起動と同時に要求が一つ生成される。

要求テーブル：要求が発生してから終了するまでの状態変化を記憶する領域である。シミュレーション途中の要求は要求テーブルで管理され、処理が完了した要求はこのテーブルから削除される。

資源テーブル：資源の種類や動作条件を記録するテーブルである。前述のディスク装置であればサーバは１つであり、動作時間は１０ミリ秒／アクセスであることが記録されている。

処理テーブル：要求を処理する際の手順を記録したテーブルである。この処理テーブルでは、資源をアクセスする順序や処理の分岐条件などを指定する。

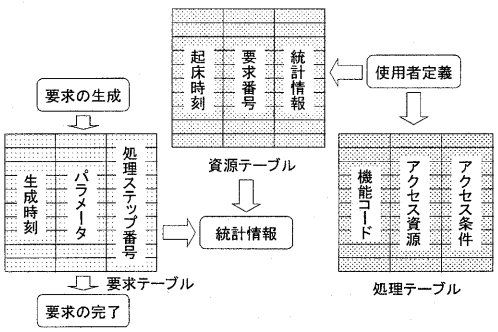


図１　シミュレーションプログラムの記憶域

処理テーブル

前述したように、処理テーブルは「要求」が発生した際のシミュレーション手順を定義した「処理」の内容が記録されている。利用者は処理の過程をプログラム形式で指定（コーディング）するが、その一つの操作（オペレーション）が処理テーブルの一つのエントリーに対応している。

高度な言語体系を採用すれば、利用者が定義する一つの操作に対して、複数の処理テーブルエントリーを対応させることも可能であろう。何れにしても、この処理テーブルの一つのエントリーがシミュレーションプログラムが提供する一つの操作に相当する。

処理テーブルのエントリーは機能コードが記録されている。この機能コードがそのエントリーで実行すべき操作を明示する。機能コードとして定義されるものは、次のようなものがある。

- 資源アクセス制御
- 分岐処理制御
- サブルーチン定義
- 繰り返し制御
- 要求の分割と結合

資源アクセスとは、そのステップで特定の資源をアクセスすることを意味する。この場合、そのエントリーにはアクセスすべき資源の名称とアクセス方法に関する情報が格納されている。利用者が資源アクセスを定義する際、次のような文法でコーディングする。ここではＡＣＣＥＳＳ文と呼ぼう。

［ラベル］　ACCESS　資源名

ラベルが準備されているのは、分岐処理が必要となるためである。単純な分岐処理はＧＯＴＯやＪＵＭＰなどに代表されるものである。ここではＧＯＴＯ文を準備したものとしよう。

［ラベル］ GOTO 分岐先ラベル

このＧＯＴＯ文の処理は単純明快であり、ただ単に次に実行すべき処理テーブルの位置を変更するだけである。

分岐処理にはＩＦなどの条件分岐も考えられる。シミュレーションでのＩＦ文では確立分岐を実現する。確立分岐とは、指定された確立に従い分岐の成立・不成立を制御するものである。例えば、７０％の確立で分岐は成立すると言った条件分岐機能を提供する。

通常のプログラム言語と同様に、分岐制御の一部としてサブルーチン機能が提供される。しかし、シミュレーションにおいてはサブルーチンも一つの資源のごとくデータ収集の対象となる。この機能を活用することにより、要求の処理に必要であった時間（レスポンス時間）の内訳を知ることが可能になる。

処理を定義する際、同じ処理を繰り返し実行することがある。この繰り返し処理の定義を簡素化するために、シミュレーションプログラムにも、このループ制御機能が実装されるべきである。通常のプログラム言語ではＦＯＲやＮＥＸＴ、ＷＨＩＬＥなどで実現されている機能である。このループ制御において、繰り返し回数などは要求ごとに管理する必要があるため、要求テーブルのパラメータ域に記録される。

パラレルＳＱＬ処理などの場合、一つの要求（ＳＱＬ）が複数の要求に分割され、分割された全ての処理が完了した時点で、ＳＱＬの処理が完了したことになる。バッチ処理などの場合にも、同様な並列処理技法が活用される。そのような特殊処理を行うのが、要求の分割（ＳＰＬＩＴ）と結合（ＪＯＩＮ）である。この場合、分割された処理は、親子関係で管理される。

資源テーブル

資源テーブルは、利用者が定義した「資源」の定義を記録している。利用者が定義した一つの資源が、資源テーブルの一つのエントリーを占有する。

利用者が一つの資源に複数サーバを定義すると、サーバごとに資源テーブルのエントリーが準備されることになる。複数サーバ構成の資源の代表格は、複数ＣＰＵチップを搭載したプロセッサである。この場合、プロセッサ（ＣＰＵチップ群）が一つの資源であり、ＣＰＵチップの一つ一つがサーバである。この場合、ＣＰＵチップごとに一つの資源テーブルエントリーが準備され、それらのエントリーは一つの資源を構成していることを明示するために、親子関係を示す関連付けが行われる。

それぞれのエントリーには、利用者が定義した資源の特性（平均動作時間など）が定数として記録される。またエントリーには、パフォーマンス統計情報などが格納される記憶域も準備される。

シミュレーションの本質は資源の状態変化のタイミング（事象）をトレースすることにあり、資源ごとに事象発生タイミングを記録しておく必要がある。そのため、資源テーブルに起床時刻が準備されている。資源アクセスが行われた場合、そのアクセス時間により決定されたアクセス完了時刻が資源テーブルの起床時刻に記録される。

また、資源をアクセスしている要求を明示するために、資源テーブルには要求番号が記録されている。この要求番号は要求テーブルのエントリー番号を示しており、資源アクセス完了時にシミュレーションを続行すべき要求を決定するために使用される。

要求テーブル

要求テーブルは、要求の処理過程を記録している。新たな要求が生成されると新しい要求テーブルのエントリーが準備され、要求が完了するとその要求の状態を記録していた要求テーブルのエントリーが削除される。

要求テーブルのエントリーには、要求の生成時刻が記録されている。この情報は、要求の処理が完了した時点でレスポンス時間（処理経過時間）を算出するために使用される。

要求の処理過程を管理するために、処理テーブルのエントリー位置を示すためのステップ番号が記録される。要求が生成された時点で、このステップ番号は処理テーブルの先頭を指している。処理が進むにつれ、対応する処理テーブルのステップ番号へと更新される。

利用者が繰り返し処理を定義した場合、その繰り返し回数などの情報は要求ごとに管理する必要がある。要求テーブルには、これら要求ごとに記憶すべきパラメータ域が準備されている。

処理の分割が行われる場合、分割された数だけの要求テーブルのエントリーが確保される。それらのエントリーは親子関係を示す関連付けが行われる。結合を行う場合は、この親子関係の情報を基に、親の処理を再開するか否かを判定する。

シミュレーションの開始

シミュレーションプログラムの実行制御を行うのがディスパッチャである。ディスパッチャはシミュレーション中に生じる事象を管理し、最初に実行すべき事象に応じて各種テーブルを更新する機能を提供する。

ディスパッチャが最初の要求をシミュレーションする様子を見てみよう。シミュレーションが開始されると同時に最初の要求が生成される。この時、要求テーブルに一つのエントリーが確保され、生成時刻が記録されると同時に処理テーブルの先頭エントリーのステップ番号がセットされる。

ディスパッチャは処理テーブルを参照し、最初に行うべき処理の内容を知る。処理テーブルに記録される操作には、前述したように資源アクセス処理や分岐処理などがある。ディスパッチャは資源アクセス処理を見つけるまで、処理テーブルで指定された操作を実行し続ける。分岐処理を検出すると、ステップ番号を更新し処理テーブルの検索を継続する。

処理テーブル内に資源アクセスの操作を検出すると、アクセスすべき資源を資源テーブルから探し出す。資源テーブルのエントリーが特定されると、その資源が動作していないかを判定する。資源がアクセスされていなければ（当然最初は資源はアクセスされていない）、その資源ア

クセスの条件に応じて動作完了時刻を算出し起床時刻として資源テーブルに記録する。また、誰がアクセスしているかを示すために資源テーブルのエントリーに現在シミュレーションを実行している要求の要求番号（要求テーブルのエントリー位置）を記録する。

ディスパッチャは、以上のような一連の制御を行うことにより、最初の要求の起動処理を完了したことになる。ディスパッチャは要求を生成した時、次の要求が生成されるべき時刻を乱数で決定し、ディスパッチャが管理する事象生成時刻に記録しておく。

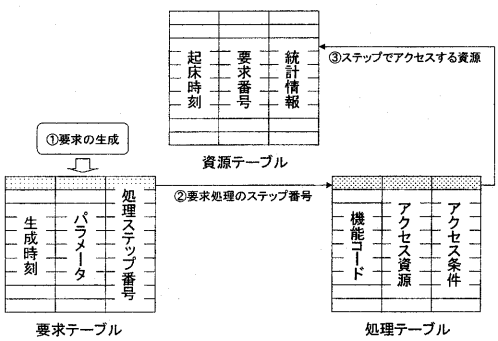


図2　要求の生成処理

シミュレーション事象の選択

資源アクセスを行っている間、シミュレーション環境は変化しない。つまり、次の事象が発生するまでの時間は、シミュレーションプログラムにとって興味ない部分である。次にシミュレーションすべき事象は、資源アクセスの完了時刻、もしくは新たな要求生成時刻の何れか若い事象である。ディスパッチャは、要求生成時刻と資源テーブルに記録されている起床時刻の内、最も若い時刻を探し出す。そして、その事象（新しい要求の生成もしくは資源アクセスの完了）の処理を実行する。

ここでは、ある資源テーブルに記録された起床時刻が最も若い時刻であったとしよう。この場合、その資源アクセスの完了処理が行われる。資源アクセスが完了した際に行わなければならないのは、現在の時刻の更新とシミュレーションしている要求の切り替えである。この処理を行うため、資源テーブルに記録された起床時刻が現在時刻に、また要求番号から要求テーブルを検索しシミュレーション対象の要求を決定する。

該当要求テーブルには、処理ステップ番号が記録されている。資源アクセスが完了した時点では、この処理ステップ番号は資源アクセスを要求した処理ステップが指されている。その処理は完了したので、処理ステップ番号をプラス１し、次の処理を実行する。ここでは要求の流れ全体を紹介するために、次に実行すべき処理ステップが要求の完了操作（ＥＮＤ文）であったと想定しよう。

要求が完了すると、その要求の処理経過時間を求める必要がある。要求が生成された時刻は要求テーブルに生成時刻として記録されている。また要求の完了時刻は、このＥＮＤ処理を行った時刻となる。今回の例では、資源アクセスが完了した時刻と同じである。この様にして処理経過時間を求め、統計情報として記録すると同時に、完了した要求数もプラス１しておく。

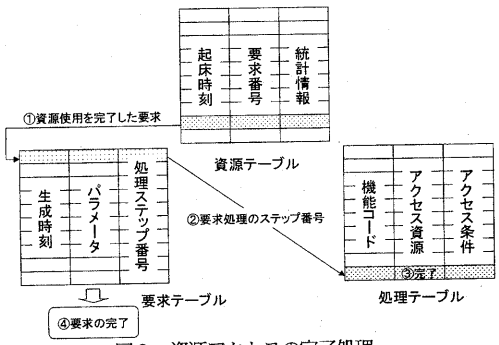


図3　資源アクセスの完了処理

競合による資源アクセスの待ち

今まで考察したシミュレーションの流れでは、一つの資源を同時に複数の要求がアクセスすることはなかった。しかし、要求の生成時間間隔や資源のサービス時間を乱数によって決定されるのであれば、先行した要求による資源アクセスが完了する前に、同一資源を後続の要求がアクセスしようとすることがある。

この場合、シミュレーションプログラムは、後続の要求の実行を一旦中断（保留）させ、先行した資源アクセスの完了を待たせる必要がある。この際、資源テーブルにアクセス待ち状態になった要求群の待ち行列を作成し、それらを管理する。

この待ち行列を作成するために、資源テーブルにアクセス待ちとなった要求の要求番号が記録されるフィールドが準備されている。このフィールドのことを待ち要求番号と呼ぶ。この待ち要求番号がゼロの場合、この資源にはアクセス待ちとなった要求はない。もし、待ち要求番号にゼロ以外の値が格納されていれば、その値はアクセス待ちとなった要求の要求番号である。

複数の要求がアクセス待ちとなった場合、それら全ての要求をアクセス待ち行列で管理する。このために、要求テーブルにも待ち要求番号フィールドが準備されている。この待ち要求番号がゼロの場合、この要求テーブルがアクセス待ち行列の終端であることを意味する。待ち要求番号にゼロ以外の値が格納されていれば、更にアクセス待ち行列が続いていることを示している。

資源アクセス時にアクセス待ちが生じる際、シミュレーションプログラムは資源テーブルのアクセス待ちの待ち行列を検査し、その終端に新たにアクセス待ちとなる要求を連結する。

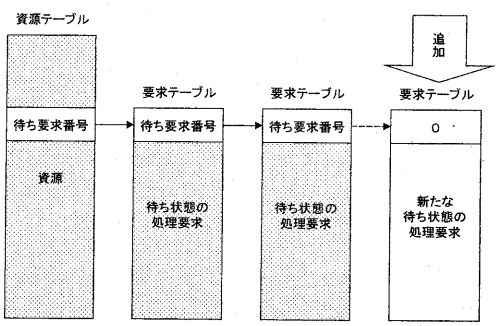
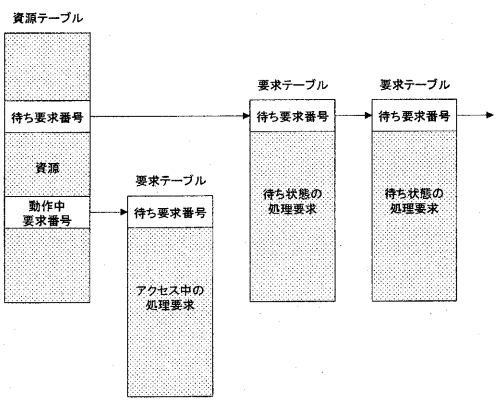


図4　待ち行列への追加



待ちとなったアクセスの再開

資源アクセスが競合することにより要求のシミュレーションが一旦中断されると、ディスパッチャは次にシミュレーションすべき事象を探す。具体的には、要求生成時刻と資源テーブルが持つ起床時刻の内、最も若い時刻を持つ事象を探し出し、そのシミュレーションを行う。

ここでは、先ほどアクセスの競合が発生した資源の起床時刻が最も若かったと仮定しよう。資源アクセスを行っている要求の要求番号は、資源テーブルの動作中要求番号のフィールドに格納されている。ディスパッチャは、このフィールドからシミュレーションを再開すべき要求を知る。

ディスパッチャは、資源アクセスが完了した要求のシミュレーションを開始する前に、アクセス待ちとなっている要求があれば、その処理を行っておく必要がある。現在、その資源にアクセス待ちの要求があるか否かは、待ち要求番号のフィールドがゼロであるか否かで判定する。

もし、待ち要求番号のフィールドがゼロでなければ、アクセス待ちの要求があることになる。その場合は、待ち行列の先頭にある要求を待ち行列から取り出し、その処理を行う。待ち行列から先頭の要求を取り出すのは簡単であり、要求テーブルの待ち要求番号のフィールドを資源テーブルの待ち要求番号のフィールドにコピーするだけである。

待ち行列の先頭にあった要求を取り出し、その資源アクセスが正常に行われた処理を施す。具体的には、その要求番号を動作中要求番号のフィールドに格納すると同時に、今回のアクセスに必要なアクセス時間を乱数で算出し、予定のアクセス完了時刻を計算し起床時間のフィールドに格納する。これでアクセス待ちとなっていた要求の再開処理が完了したことになる。これらの処理を行った後、この資源のアクセスを完了した要求のシミュレーションを実行する。

以上の説明では、新たにアクセス待ちとなった要求は、待ち行列の最後尾に付け加えられた。また、先行する資源アクセスが完了した際、待ち行列の先頭にある要求が資源アクセス権を得る。このような制御を行う方式をＦＩＦＯ（First In First Out）と呼ぶ。これ以外にも、優先順位制御など幾つかの制御方式がある。しかし、ここではその説明を割愛する。

図5　アクセス中と待ち状態の要求

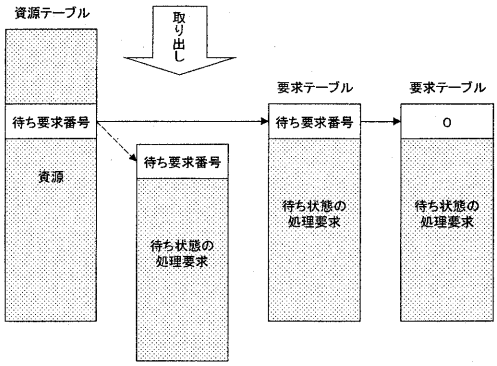


図6　待ち行列からの取り出し

複数サーバ構成の資源

もう一つ、資源について考慮しなければならないことがある。それが複数サーバ構成の資源である。複数サーバ構成とは何か。通常、ディスク装置などは、同時に複数のアクセスを行うことが出来ない。同時に複数のアクセス要求が来ると、先行したものがアクセス権を得て後続のものは待ち状態となる。

複数サーバ構成の資源の代表格は、複数ＣＰＵチップを搭載したプロセッサである。この場合、プロセッサ（ＣＰＵチップ群）が一つの資源であり、ＣＰＵチップの一つ一つがサーバである。このシステム内で実行されるプログラムは、ＣＰＵチップを特定することなくプロセッサを使用する。つまり、その時点で空いているＣＰＵチップがあれば、そのＣＰＵチップを使用したプログラム実行を行う。

この複数サーバ構成の資源をシミュレーションする場合、サーバ単位に資源テーブルを準備する。また、管理上、一つの資源を構成するサーバの資源テーブルは、一つのグループとしておく必要もある。

シミュレーションプログラムでは、複数サーバ構成の資源にアクセス要求があった場合、その資源の全サーバの状態を検査し、空いたサーバがあればそのサーバでアクセスを実行させる。もし空いたサーバがなければ、その要求は待ち行列で待たされることになる。

複数サーバ構成の資源には複数の資源テーブルが準備されるが、待ち行列は一つしか準備してはならない。何故ならば、それぞれのアクセスのサービス時間は乱数で決定されるため、どのサーバが最も早くアクセスを完了するかが予測できないためである。サーバのアクセスが完了した時点で、共通の待ち行列の先頭に位置する要求を再開するロジックが必要となる。

アクセス要求が到着した時点で、複数の空きサーバがあった場合、何れのサーバでアクセスを実行するかを決定するには、幾つかの制御方式がある。「最初に見つけた空きサーバで実行する」と言うのが、最もシンプルな方法である。この場合、資源テーブルの定義順にサーバ検索が行われているとすると、定義されたサーバ順に使用率が高くなる結果が予想される。それぞれのサーバの使用率をバランス化させるのであれば、最も使用率の低い空きサーバでアクセスを実行する方式を採用する。

「ＣＰＵチップが６つ搭載されたプロセッサを使用し

ているが、ある業務グループでは、ＣＰＵ３番と４番しか使用しない」。このような条件を持ったシステムが稀にある。このようなシステムのシミュレーションを行うには、サーバ選択の条件を工夫する必要がある。

分岐制御によるバッファ評価の実現

性能向上の技法として、バッファメモリやキャッシュメモリ（ここではバッファと総称する）が多用される。このバッファ技法は、頻繁にアクセスされるオブジェクト（データやプログラムなど）を一定の大きさのメモリ域に読み込んでおき、要求されるたびにディスク装置をアクセスすることを回避しようとするものである。

このバッファ技法を評価するために、シミュレーションを使用できるのか。例えば、バッファで使用するメモリ容量を大きくすれば、バッファヒット率（要求されたオブジェクトがバッファ内で見つかる確率）を予測することができるのかと言うことである。この疑問に答えるにはバッファの管理技法、バッファ操作の対象となるオブジェクトの大きさなどを明確にする必要がある。

具体的に考察すればするほど、バッファ評価をシミュレーションで行うことが困難であると考えられる。このため、本研究ではバッファ評価を分岐制御で行うこととした。

通常、バッファが準備されている機構では、バッファヒットした際には、バッファミスした時の処理が不要になる。この特性を利用して、条件分岐で対応する。例えば、７０％のバッファヒット率が保証されている場合、３０％の確率でバッファミス時の処理を実行する。

この手法ではバッファヒット率は定数であり、環境条件の変化による動的なバッファ性能の予測を行うことはできない。しかし、バッファ管理手法をコーディングしたりするコストやその精度などを考慮すると、この機能でバッファ対応とする方法は充分実用に耐えると考える。但し、このようなシミュレーション手法を採用していることを利用者は理解しておく必要はあろう。

サブルーチン制御

シミュレーションでは、アプリケーションの振る舞いを処理として定義する。しかし、その中の一部の動作に着目した観察を行う必要もある。例えば、Ｗｅｂサーバの評価を行うような際、認証プロセスの時間を知りたいなどと言う要望も出てくるであろう。このような目的に応えるため、シミュレーションのサブルーチン制御を考える必要がある。

通常のプログラミング環境では、サブルーチンは処理が完了した時点で呼び出し元に復帰できる機能を提供する。しかし、シミュレーションにおいてサブルーチンは、資源と同様に統計情報収集の単位であると同時に、同時処理が可能な最大要求数を定義できるものである必要がある。

サブルーチンで提供される統計情報収集機能では、そのサブルーチンが呼び出された回数、またその処理に必要であった平均処理時間が算出できるものでなければならない。この機能を提供することにより、シミュレーションで定義された処理の一部を監視対象にすることができる。

サブルーチンで処理可能な最大要求数を制限する機能は、若干制御が難しい。この機能は、あるアプリケーションプログラムを使用できるユーザ数が制限されているようなシステムのシミュレーションを行う場合に利用されるものである。

この最大要求数の制限を行う場合には、サブルーチンに入っている要求数を管理する必要がある。また、サブルーチンの入口処理で滞留要求数を検査し、最大要求数に達しているのであれば、新たな要求は資源アクセスが生じた際の待ち行列と同じ要領で処理を一旦中断される。

サブルーチンの出口処理では、サブルーチンに入っている要求数を一つ減じると同時に、サブルーチンの入口で実行待ちとなっている要求がないかを検査する。もし、実行待ちとなっている要求があれば、その再開処理を行った後、本来の要求のサブルーチン出口処理を継続する。

ポワソン分布だけか

シミュレーションでは随所で乱数を使用する。その乱数はポワソン分布に従ったものである。しかし、場合によっては、特殊な分布の乱数を必要とすることもあろう。その場合、ランダムさを歪める機能が必要となる。

ランダムさを歪めるには、幾つかの方法がある。ここでは、最も簡単な方法を紹介しよう。図７のようにＲＮＤ関数で求められた値を計算式で補正するものである。①の式は、ＲＮＤ関数で得られた値と同じランダム値を生成する。②の式では、１に近い値の発生頻度が高くなる。

乱数を生成する必要とする際、このランダムさの歪みを定義する機能を提供しておけば、違ったシミュレーションを可能にすることもできる。

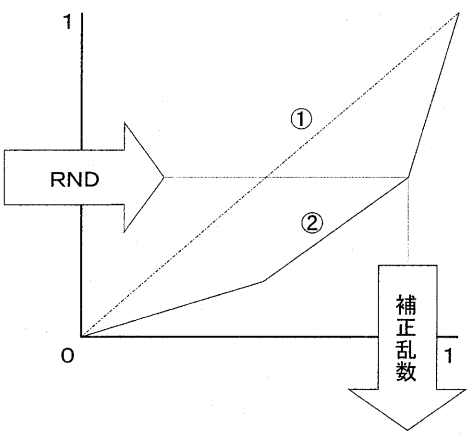


図７　生成された乱数の歪みの設定

参考文献

野瀬純郎：情報システムの性能評価技法、
ＩＩＭレポート１９９８　ＮＯ１０５～１０９

河野知行、森本舞：ボトルネック箇所の特定と改善効果の判定
第３回システム評価研究会発表会

以上