計算機アーキテクチャ 99-11
(1993. 3. 12)

分散リアルタイムシステム解析ツールESCORT

神余 浩夫

三菱電機(株)中央研究所

(661) 兵庫県尼崎市塚口本町 8-1-1

あらまし

ESCORT は分散リアルタイムシステム (DRTS) の設計および性能解析を目的とする並行イベントシミュレータである。処理手続きはタイムアウトハンドラなどとともにトランザクションオブジェクトとしてカプセル化され、コルーチンを用いた周期/非周期タスクによって並行処理される。プロセッサ、リソースおよびタスクなどのDRTS構成要素はオブジェクトクラスライブラリとして提供されていて、モジュール性と拡張性を特徴とする。これらの機能により ESCORT はタスク挙動の解析と構成要素の利用率を分析することができ、Rate-Monotonic 解析と組み合わせることによって、DRTS 設計評価をサポートする。

和文キーワード　リアルタイムシステム、分散システム、スケジューリング、並行処理、イベントシミュレーション

The Distributed Real-Time System Analyzer :ESCORT

Hiroo Kanamaru

Central Research Lab. Mitsubishi Electric Corp.

8-1-1 Tsukaguchi-Honmachi Amagasaki, Hyogo, 661 JAPAN

Abstract

ESCORT (Event Simulator for COncurrent Real-time Transaction) is the concurrent event simulator to design and analyze disributed real-time systems (DRTS). The processing sequences and timer handlers are encapsulated in the transaction objects, and periodic/aperiodic tasks implemented with the co-routine execute the transactions concurrently. Elements of DRTS ; processors, resources and tasks et.al., are provided as object class libraries. By these features of ESCORT, we can simulate behaviors of tasks and analyze utilization of resources. With using Rate-Monotonic Analysis, ESCORT is able to assist to design and verify DRTS.

英文 key words　real-time systems, distributed system, scheduling, cocurrent processing, event simulation

1 諸言

分散リアルタイムシステム（Distributed Real-Time Systems: DRTS）とは、バスまたはネットワークで接続された分散計算機システムにおいて、明示的な処理デッドラインを有する強リアルタイム（hard real-time）タスクを主に処理するシステムを意味する。例えば、発電/化学プラント監視制御システムや、自動生産システム/製造ロボット制御などは、アクチュエータやモータ毎の制御計算機群とそれらを統括する制御計算機が連動し、プラント動特性や製造速度に適切なタイミングでタスク処理および制御信号送出を行なう典型的なDRTSである。また、これらのシステムにおける冗長構成によるフォールトトレラントシステム設計もDRTSの一例である。DRTSの設計では、平均性能の向上より最悪の場合でのリアルタイム制約充足性；スケジューラビリティが重要である。すなわち、各タスクの最悪実行時間を測定あるいは見積り、その組合せがいずれのデッドラインも満足することを確認する必要がある。単一プロセッサでタスク数が少なくタスク間の相互影響もない小規模システムではこのような手法も有効であったが、複雑なDRTSではタスクのプロセッサ割り当てや同期の問題なども考慮しなければならず、新しい設計解析手法が求められている [1]。

リアルタイムシステムの設計解析手法として注目されるのが、Rate-Monotonic 解析法である [2]。RM 解析法はアプリケーション特性に依存せずタスク実行制御方策を与えることができ、タスク挙動も予測的である。しかし、システム性能を最大限に発揮させるためには、高優先度タスクのブロック時間が小さくなるように資源のロックタイミングやタスク周期を解析結果から調整する必要がある。また、リアルタイムシステムの性能解析を目的としてタスク挙動シミュレーションツールもいくつか提案されている [3, 4]。これらは、周期/非周期リアルタイムタスクについてRM法に必要なパラメータを設定し、タスクスケジューリング状況を表示する。複雑なDRTS設計を効率的に進めるには、RM解析法に基づくシステム性能分析とそれを支援するタスクスケジューリングのシミュレータが必要不可欠である。しかし、RM解析法が分散システムの対象としていないため、DRTSに関する有効な解析法およびツールはほとんど見当たらない。

ESCORT(Event Simulator for COncurrent Real-time Transaction) は、DRTSの効率的設計を支援することを目的としており、分散システムにおける並行トランザクション[1] の挙動タイミングに関するイベントシミュレータである。ESCORTはデッドラインを持つリアルタイムトランザクションをオブジェクトとしてモデル化する。すなわち、トランザクション処理、タイムアウト処理をまとめてカプセル化することにより、実行制御に関する多様な情報および動作を隠蔽化していることが特徴である。同様に、ESCORTが提供するシミュレーション機構、タスク、プロセッサやリソースは抽象クラスライブラリとして提供されているため、ユーザーは抽象度の高い定義だけでシミュレーションが可能であり、機能拡張もクラス定義として容易に実現できる。DRTSのイベントシミュレータであるESCORTにRM解析法に基づく支援ツールを組み合わることによって、効果的なDRTS設計解析環境を得ることができる。

以下、はじめに現状のDRTS設計構築手法について簡単に紹介し、ESCORTの目的と構成およびDRTSのモデル化について説明する。次に、ESCORTを利用したシミュレーション例と解析例を示し、本ツールの有効性について検討する。

2 DRTS構築方法

2.1 現状と要求

従来、リアルタイムシステムは特定のタスク群を専門に処理する専用システムとして設計することがほとんどであった。タスクスケジューリングやリソースの競合問題も全て設計者が直接制御できる規模であったからである。しかし、計算機の高性能化に従い処理するタスクが多数かつ複雑となってきた。また、最近のオープンシステム指向はリアルタイム性を保証できないデバイス、タスクの

[1] ここでは、トランザクションは一連の処理の流れという広義の意味で用いている

混在が避けられなくなってきている。

このような新しい環境下のDRTS設計の問題点として、まずタスクが複雑になったためそのタイミング挙動が従来ほど予測的でなくなったことが挙げられる。DRTSのタスクはディスク入出力やネットワーク間通信などを多く行ない、タスク間の排他制御や同期も頻繁である。このため、優先度逆転(priority inversion)や相互デッドロック(mutual deadlock)などの現象[5]による不則的な遅延が起こるようになり、タスクのタイミング挙動を把握することが困難になってきた。これまでのタスク実行時間と優先度に基づくシステム解析手法でなく、通信やリソースの操作も考慮したトランザクションに基づくミクロなタスク挙動解析が必要である。

DRTS設計のもうひとつの問題は、初期設計における構成見積りの問題である。上で述べたように、処理タスクが複雑になるとそのタイミング挙動が不則的となるが、これは分散システム上においてさらに悪化する。通信するタスクを同じプロセッサに割り当てた場合と異なるノードにおいた場合とで通信時間が異なるように、分散システムの構成にタスク応答性とスケジューラビリティは大きく左右される。特にプロセッサ処理能力以外のボトルネックの存在は、従来のタスク処理時間積み上げ方式によるDRTS性能予測を困難にしている[9]。そこで、要求されるタスクの概要からスケジューラビリティを確保できるシステム構成を推定するためのマクロな性能推定ツールも重要となる。

ESCORTはミクロな動作解析を目的としたイベントシミュレータであるが、システム構成要素およびトランザクションをオブジェクトとしてモジュール化しているので、抽象モジュールの利用によりマクロ性能予測も可能である。

2.2 Rate-Monnotoninc法

Rate-Monotonic Scheduling(RMS)法とは、「周期タスクにおいて周期の短いものに高優先度を与える」固定優先度方式の最適静的スケジューリング方策である[6]。

Rate-Monotonic解析は、このタスクスケジューリング方策に基づき周期/非周期タスクの性能解析を行なうものであり、以下が挙げられる[7]。

1. 利用限界テスト

   利用限界(utilization bound)テストは、独立な周期タスクがRMSにより常にデッドラインを満足できるかどうかを調べる。タスクiの実行時間をC_i、周期をT_iとすると、プロセッサ利用率$U(n)$は次式で得られる。

   $$C_1/T_1 + \cdots + C_n/T_n \leq U(n) = n(2^{1/n} - 1)$$

   こののテストの結果は、成功$(0 \leq U \leq U(n))$、不完全$(U(n) < U < 1)$または過負荷$(1 < U)$のにずれかである。

2. 完遂時間テスト

   完遂時間(completion time)テストはスケジューラビリティを調べるもので、最悪のタスク位相におけるデッドライン充足を調べる。タスクiの完遂時間をW_iとすると、次式の繰り返しにより全タスクのW_iを得ることができる。

   $$W_i(n+1) = C_i + \Sigma_{j<i}\lceil W_i(n)/T_j \rceil C_j$$

   W_iがそのタスクのデッドライン以前であれば、iはスケジューラブルである。ただし、$W_i(0) = 0$である。

これらのテストは周期タスクを前提としていたが、適用範囲を拡大すべく研究が続けられている。

このほか、解析手法ではないが、優先度逆転現象を解決する優先度継承/セイリングプロトコル(priority inheritance / ceiling protocol)、非周期リアルタイムタスクを扱う非周期タスクサーバ(sporadic server)などの機構が提案されている[11, 8]。これらの機構と上記のRM解析法を組み合わすことによって、複雑なリアルタイムシステムの解析が可能となる。しかし、本来タスク周期と実行時間だけに注目すればよかったRMSに比べると、セマ

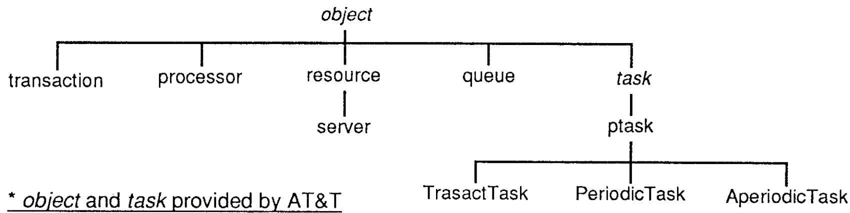


図 1: ESCORT クラス階層

フォや同期などタスク処理内容にまで踏み込んだ分析が必要となりこと、分散システムにおける RM 解析法が研究途上であることから、現在のところ DRTS の性能解析は動作解析シミュレータを主に活用せざるを得ない。

3 ESCORT の概要

3.1 概要と構成

ESCORT はユーザーが定義したリアルタイムトランザクションをコルーチンを用いて並行に動作させるイベントシミュレータである。筆者らが以前開発したリアルタイムタスクの動作解析シミュレータ:TSA[4] では、周期 / 非周期タスクについて周期や実行時間などのパラメータを設定するだけであった。ESCORT は処理手続きとしてのトランザクションをユーザーが定義する必要があるので、TSA のような閉じた実行環境ではなく、シミュレーション機能を C++ のクラスライブラリとして提供している。すなわち、ユーザーはシミュレーションに必要なトランザクション、タスクやリソースを定義し、そのインスタンスを生成するプログラムを書くことを要求される。

ESCORT が提供するクラスライブラリの階層構成を図 1 に示す。コルーチンによる並行動作制御は AT&T C++ the task class library パッケージ [10] に基づいている。提供するクラスは大別してトランザクション、タスク、プロセッサクおよび リソースの各クラスである。トランザクションは処理手続きでありタイムアウト処理などとともにカプセル化されている。タスクは時間軸上の制御をうける主体であり、指定されたトランザクションを実行する。プロセッサは優先度に基づくタスクの実行制御を司る。リソースはセマフォの役目を果たす。以下に各クラスについて具体的に説明する。

3.2 トランザクションクラス

リアルタイム処理では本来の処理とともにデッドラインや例外処理などを扱う必要があり、最近はこれらをまとめてオブジェクトとしてモデル化する手法が提案されている [12]。ESCORT のトランザクションも実行制御に必要な情報のカプセルっ化を目的として、以下のように定義されている。

```
class transacion:public object{
pubic:
  time deadline;
  transaction(time deadline=unlimit);
  virtual void start();
  virtual void timeout();
    ....
};
```

トランザクション処理本体である start() と、タイムアウト処理の timeout() を仮想関数として持つ。ユーザーがトランザクション処理を定義したい場合は、派生クラスを定義してその start() に処理手続きを記述する。この処理記述はトランザクションクラスのメンバ関数を用い、以下のものが提供されている。

```
gain(time)          // プロセッサの消費
lock(resource*)     // リソースロック
unlock(resource*)   // リソース解放
```

また、start の中から他のトランザクションオブジェクトの start を呼び出す、入れ子トランザクション (nested transaction) も可能である。ただし、異常処理はユーザーに任されている。

3.3 タスククラス

AT&T のタスククラスは優先度を持たないので、まず優先度スケジューリングが可能なタスククラス (ptask) を定義した。優先度およびプロセッサを内部変数とする。他のタスクは全て ptask クラスの派生クラスである。

```
class ptask:public task{
public:
  int prirority;
  ptask* T_next;     // スケジューリング用
  processor* cpu;    // 実行プロセッサ
  ptask(......);
    ....
};
```

TransactTask はトランザクションを引数とし、トランザクションのメンバ関数 start() を実行する。start() 実行前にプロセッサ待ち行列に入り割当を受けている間だけトランザクション処理を進める。トランザクション処理が終了するとプロセッサ待ち行列から離れ終了状態となる。デストラクト関数はトランザクションの異常処理を行なう、すなわち timeout() を実行する。

```
class TransactTask:public ptask{
private:
  transaction* trns;
public:
  TransactTask(......);
  ~TransactTask();
};
```

PeriodicTask, AperiodicTask は周期 / 非周期タスクである。いずれも周期 / 非周期で trans_task を生成した後、タイマからの起動により TransactTask の状態を調べ未終了であれば異常終了させる。PeriodicTask の動作モデルを図 2 に示す。

```
class PeriodicTask:public ptask{
private:
  time period;
  TransactTask* tr_tk;
  transaction* trs;
public:
  PeriodicTask(......);
  ~PeriodicTask();
};

class AperiodicTask:public ptask{
  ........  // Periodic 同様
};
```

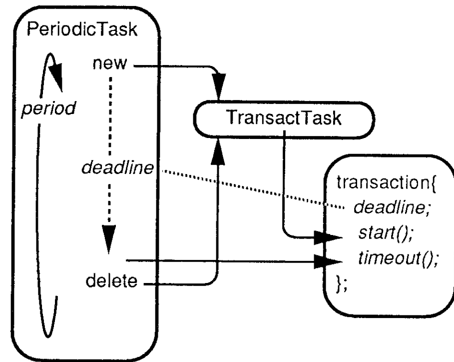


図 2: PeriodicTask の動作モデル

いずれもデッドラインは与えられた transaction の値を優先するが、周期の方が短い場合は周期を優先する。また、非周期タスクの場合、タスク生起間隔の最小値が明示的でなければ最悪状態を決定的にできないので、ここでは上下限を指定した一様乱数としてシミュレーションしている。

3.4 プロセッサクラス

プロセッサクラスは固定優先度のプリエンプション方式に従いタスクのスケジューリングを行なう。並行動作するタスククラスの同期機構としての役割も担っている。プロセッサスケジューリングに関する関数を持つが、トランザクションの記述には用いない。

```
class processor:public object{
public:
  processor(char* n);
  void request(ptask*);
  void suspend(ptask*);
  void release(ptask*);
  ptask* reschedule();
    ....
};
```

分散システムを指向しているので ESCORT では複数プロセッサを生成することができる。プロセッサ間での同期機構は明示的に用意されていないので、次のリソースクラスで同期機構を定義しなければならない。

3.5 リソースクラス

リソースクラスはセマフォの役割を持ち、ロック中であれば要求タスクをブロックし、解放時にはブロックされていたタスクをプロセッサ待ち行列に戻す。複数プロセッサにおいて共有することができる。リソースの priority は優先度継承プロトコルをサポートするためであり、現在ブロック中のタスクの最高優先度が記録される。

```
class resource:public object{
private:
  int priority;
public:
  resource(char* n);
  void lock();
  void unlock();
};
```

サーバークラスはリソースクラスの派生クラスで、内部に待ち行列を持ち FIFO あるいは優先度順でタスクにリソースを割り当てる。

```
class server:public resource{
public:
  server(char* n);
  void request();
  void release();
};
```

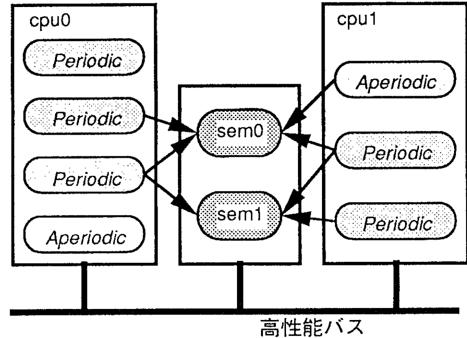
3.6 RM 解析支援ツール

現在、利用限界テストを行なうツールだけが提供されている。トランザクションの start 関数についてプロセッサを消費する gain() の時間合計を求め、トランザクションの実行時間を得る。次に、各プロセッサについて PeriodicTask の周期とトランザクション実行時間から、利用限界テストを行ない判定を下す。

ただし、gain の引数が確定的でなければいけない、非周期タスクを無視するなど課題は多い。また、同ツールを使うより ESCORT の結果を見た方が効果的であることがほとんどであるなど、改善の余地は多い。

4 使用例

4.1 問題とプログラム例



task	実行時間	セマフォ	deadline	周期
cpu0				
pt0	20ms	-	100ms	100ms
pt1	50ms	sem0	260ms	260ms
pt2	60ms	sem0,sem1	260ms	260ms
at0	20ms	-	100ms	180-220
cpu1				
pt3	50ms	sem1	180ms	180ms
pt4	60ms	sem0,sem1	190ms	190ms
at1	50ms	sem0	260ms	320-400

図 3: DRTS の例題

ESCORT を用いた DRTS シミュレーションの例題を示す。図 3 のようにバス接続された 2 台のプロセッサと共用メモリがあり、それぞれ周期 / 非周期タスクを実行する。簡単のためバス競合は起こらないものと考える。すなわち、タスク実行が待たされるのは、プロセッサと共用メモリ上のふたつのセマフォだけである。それぞれのタスクのパラメータは図 3 の表に示している。ここで、タスク pt1,at1 はふたつのリソース sem0,sesm1 を必要とし、そのトランザクションは次の順番で行なわれる。

...,P(sem0),...,P(sem1),...,V(sem1),V(sem0),...

これらのタスクについて RM 解析法の利用限界テストを行なうと以下の結果を得る。

$U(cpu0) = 0.20 + 0.192 + 0.231 + 0.111 = 0.734$

$U(cpu1) = 0.278 + 0.315 + 0.156 = 0.749$

$U(4) = 0.756$ であるから、いずれも成功と判断される。ただし、今回はリソース待ちによる遅延があるため、RM 解析法だけではスケジューラビリティを議論することはできない。そこで、RMS 法の短周期高優先度方針に従いタスクに優先度を与え、ESCORT によりシミュレーション

```
#include <stdio.h>
#include "escort.h"

processor* prc0=new processor("cpu0");
processor* prc1=new processor("cpu1");
resource*  sem0=new resource("sem0");
resource*  sem1=new resource("sem1");

class trans1:public transaction{
private:
  resource* ss;
public:
  trans1(time d, resource* s){ deadline=d; ss=s;}
  void transaction()
    {gain(1);lock(ss);gain(3);unlock(ss);gain(1);}
  void timeout(){ unlock(s0); }
};
  ........
main(){
  trans0 *trs0=new trans0(10);
  trans1 *trs1=new trans1(40,sem0);
  trans2 *trs2=new trans2(22);
  trans1 *trs3=new trans1(34,sem1);

  PeriodicTask  *pt0=
        new PeriodicTask("pt0",prc0,trs0,30,10);
        // 30 is priority, 10 is period
  PeriodicTask  *pt1=new PeriodicTask("pt1",...);
  PeriodicTask  *pt2=new PeriodicTask("pt2",...);
  AperiodicTask *at0=new AperiodicTask("at0",...);
  PeriodicTask  *pt3=new PeriodicTask("pt3",...);
  PeriodicTask  *pt4=new PeriodicTask("pt4",...);
  AperiodicTask *at1=new AperiodicTask("at1",...);

  thistask->delay(90);
  delete pt0; ........
  thistask->resultis(0);
}
```

図 4: ESCORT プログラム例

を行なう。

ESCORT のプログラミング例を図 4 に示す。はじめにトランザクション定義に必要なプロセッサやリソースなどを定義し、処理手続きを記述したトランザクション派生クラスを定義する。次にトランザクションを実行する周期 / 非周期タスクのインスタンスを生成する。このとき、優先度と周期（非周期タスクは上下限値）を指定する。主ルーチンはシミュレーション終了時間まで遅延したあと、生成したコルーチンを停止 / 消滅させて終了する。ここではシミュレーション時間単位を 10msec としている。

4.2 解析とチューニング

図 3 に基づくシミュレーション結果のガントチャートを図 5 に示す。図において、各行はプロセッサまたはリソースにおける各タスクの状態を表していて、'#' は使用中、'-' は待機またはブロック中、'!' はタイムアウトの発生を意味する。

非周期タスク at1 が 260msec においてタイムアウトを発生している。いずれのプロセッサも利用限界テストで示したように処理余裕を残しており、遅延の原因はリソース sem0 であることが判る。sem0 は 4 つのタスクがアクセスする最も渋滞するリソースであり、現時点の性能ボトルネックとなっている。sem0 をアクセスするタスクで最高優先度であるのは pt4 であるが、cpu1 にはそれより高い優先度の pt3 が存在し pt4,at1 を遅延させることができる。このことが sem0 のロック時間延長の主因となっている。ところが、優先度セイリングプロトコルを適用しても sem0 を要求するタスク優先度は pt3 を上回らないから、ロック状態で preempt されてしまう現象は解消されない。結局、sem0 を要求するタスクがロック状態でアイドルしないように sem0 に依存しないタスクの優先度を下げてスケジューラビリティを確保できるようにチューニングを行なうことになる。この場合、pt3 の優先度を cpu1 において最低となるように設定し、シミュレーションを行なう。

図 6 の調整後のシミュレーション結果が示すように、デッドライン超過は解消されている。RMS 方策から外れてタスク優先度を与えることはスケジューラビリティの予測性を低下させることになる。しかし、仮想的にデッドラインを短く設定して RMS によるタスク優先度を高くした上で完遂時間テストを行なうなどの解析手段により、限定的なスケジューラビリティ解析は可能である。

5 結論

DRTS 設計構築支援のための動作解析ツール ESCORT の概要について述べてきた。ESCORT 自体はイベントシミュレータであり、その使い勝手の向上のためにはクラスライブラリの拡充、対話的な設定および図表化ツールの充実などを行なっていくべきであろう。しかし、DRTS の設計支援の目的のためには、システムのスケジューラビリティを解析的に検討できるサポートツールが不可欠である。RM 解析法を中心とした DRTS 解析手法はまだ研究

```
cpu0    ::=========10========20========30========40========50========60========70========80=======:
 pt0     :##........##........##........##........##........##........##........##........##......:
 pt1     :------------#####.........------------------####....---#------------##--##....----------:
 pt2     :----######--..............#--#--####................###-------###.............#-------##:
 at0     :--##..............##--.................#--#...............##--.................#--#.....:
cpu1    ::=========10========20========30========40========50========60========70========80=======:
 pt3     :#####.............#####.............#####.............#####.............#####...........:
 pt4     :-----#---#####.....----#-#####........---#-----#####.....--#----#####.......-######.....:
 at1     :------#---------##------#!......#--#------###........................#-------------####.:
sem0    ::=========10========20========30========40========50========60========70========80=======:
 pt2     :.....####..................--######..................###########...................---##:
 pt4     :......---####...........-####.............-----####.........----####..........####......:
 at1     :.......---------#########........--#########..........................-------------###..:
 pt1     :.............###...........-----------------###.........------------#####..........-----:
sem1    ::=========10========20========30========40========50========60========70========80=======:
 pt3     :.###...............###...............###...............###...............###............:
 pt2     :.......##........................##....................-------##........................:
 pt4     :...........##..............##....................##...............##............##......:
```

図 5: シミュレーション結果

```
cpu0    ::=========10========20========30========40========50========60========70========80=======:
 pt0     :##........##........##........##........##........##........##........##........##......:
 pt1     :------------#####.........----------###----#........---#------------##--##....----------:
 pt2     :----######--..............####--##..................###-------###.............#----#####:
 at0     :--##..............##--.................#--#...............##--.................#--#.....:
cpu1    ::=========10========20========30========40========50========60========70========80=======:
 pt3     :-------------###..#------####.......-#-####...........###-##............#------##---....:
 pt4     :######.............######.............#----#####.........#------#####.......###--###....:
 at1     :------#--####...................#####................................#---###---.........:
sem0    ::=========10========20========30========40========50========60========70========80=======:
 pt4     :.####...............####...............----####...........------####.........######.....:
 pt2     :.....####..................######....................###########...................####.:
 at1     :.......--###.....................###..................................---####...........:
 pt1     :.............###..................--#######.............------------#####...............:
sem1    ::=========10========20========30========40========50========60========70========80=======:
 pt4     :...##.................##.....................##...................##............--##....:
 pt2     :.......##....................####......................-------##.....................##.:
 pt3     :.......--######..........###...........###.............####..............########.......:
```

図 6: チューニング後の結果

の余地が多く残されており、これからの成果に期待したい。また、RTC++ など [13] 新しい DRTS 実行環境と ESCORT との親和性についても、これから検討を進めていく予定である。

参考文献

[1] J.A.Stankovic : Distributed Real-Time Computing : The Next Generation, 計測自動制御学会誌 , 31 - 7, (1992)

[2] J.Lehoczky, L.Sha and Y.Ding : The Rate Monotonic Scheduling Algorithm : Exact Characterization and Average Case Behavior, Proc.of IEEE Real-Time Systems Symposium, 166/171 (1989)

[3] H.Tokuda and C.W.Mercer : Arts: A Distributed Real-Time Kernel, Operating Systems Review, 23-3, 29/53 (1989)

[4] 藤田, 竹垣 : 制御計算機用タスクスケジューリングアナライザの試作, 第 43 回情報処理学会全国大会論文集 6 359/ 360 (1991)

[5] P.A.Laplante ed.:Real-Time Systems Dsesign and Analysis -an engineer's handbook-, IEEE CS Press, 199/230 (1993)

[6] C.L.Liu and J.W.Layland : Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment, J.ACM 20-1, 46/61 (1973)

[7] L.Sha : An Introduction to Rate-Monotonic Analysis, tutorial note (non-published) (1992)

[8] 藤田, 竹垣 : 分散制御システムへの非周期タスクサーバの適用性, 電子情報通信学会研究会 RTP'92 論文集, 91/ 97 (1992)

[9] 神余, 藤田, 竹垣 : 協調分散スケジューリング方式のリアルタイム保証性, 電子情報通信学会研究会 RTP'92 論文集, 99/106 (1992)

[10] B.Stroustrup and J.E.Shapiro : A Set of C++ Classes for Co-routine Style Programminng, AT&T C++ Language Syetem Release 2.1, Library Manual

[11] J.P.Lehoczky, L.Sha and J.K.Stronsnider : Enhanced Aperiodic Reponsiveness in Hard Real-Time Environments, Proc. IEEE RTSS'87, 261/270 (1987)

[12] C.W.Mercer and H.Tokuda : The Arts Real-Time Object Model, Proc. IEEE RTSS'90, 2/10 (1990)

[13] 石川裕, 徳田英幸 : 分散型実時間プログラミング言語 RTC++, コンピュータソフトウェア, 9-2 (1992)