数理解析研究所講究録
1489巻 2006年 8-14

時間オートマトンによるValue-Density スケジューリングアルゴリズムの性能解析手法

金沢大学・自然科学研究科　坂倉賢昭 (Masaaki Sakakura)　山根智 (Satoshi Yamane)
Graduate School of Natural Science & Technology,
Kanazawa University

1 Introduction

現在，リアルタイムシステムは高性能かつ高信頼性を要求されるシステムに多く利用されているが，これらのシステムは複雑で開発や検証を行うのは大変難しいことである．デッドライン（時間制約）を守ることが絶対とされるハードリアルタイムシステムにおいては，近年，一般的な解析方法として時間オートマトンが用いられている．時間オートマトンを用いてタスクの到着パターンを表すことにより，周期的なタスクだけでなく非周期的なタスクも扱うことができるようになり，これによってタスクの周期性にかかわらず時間オートマトンの到達可能性解析を行うことによってシステムの性質をモデル解析できるようになった．例として，Wang Yiらによってタスク付き時間オートマトンによるハードリアルタイムシステムのスケジューラビリティ解析手法が提案されている [1][2]．しかし現在では，デッドラインを守ることが最善ではあるが，多少のデッドラインミスを許容する，ソフトリアルタイムシステムも増加しており，重要視されてきている．典型的な例としては，画像や音声通信などのマルチメディアデータ処理，または分散データベースなどがある．ソフトリアルタイムシステムの解析分野では主に2種類の方法が提案されている．1つは，ソフトリアルタイムシステムのタスクの時間的性質を関数で現す方法であり，その価値関数（value function）を用いてスケジューリングアルゴリズムの性能を統計的に解析している．[4] もう1つの方法はタスクがデッドラインをどれだけの確率で満たせるかを統計的に解析する方法が提案されている．また価値関数を使った方法ではその価値関数を用いたスケジューリングアルゴリズムも提案されている．そこで本研究では，Wang Yiらによって提案されたタスク付き時間オートマトンを価値関数によって拡張し，ソフトリアルタイムシステムに対するスケジューリングアルゴリズムの性能をモデル解析する手法を提案する．

本文の構成は，2章で本研究の対象となるリアルタイムシステムの概要について説明し，3章で今回用いる解析モデルについて，4章でその解析方法について説明し，簡単な例を用いて実際に解析を行う．

2 ソフトリアルタイムシステム

本章では本研究の対象となるソフトリアルタイムシステムについて説明する．

2.1 リアルタイムシステム

リアルタイムシステムとはシステムの品質が，論理的正確さと同様に，結果を出す時間に依存しているシステムである．よって，リアルタイムのモデルはタスクがサービスを完了する完了時刻が厳格であるかどうかによって大きく次の2つに分けることができる．(図1)

ソフトリアルタイムシステム 処理の完了が時間要求を満たさなくても結果は有意義であるが，時間とともに価値が下がる．

ハードリアルタイムシステム 処理の完了が時間要求を満たさなければシステム全体に致命的な影響を与える．

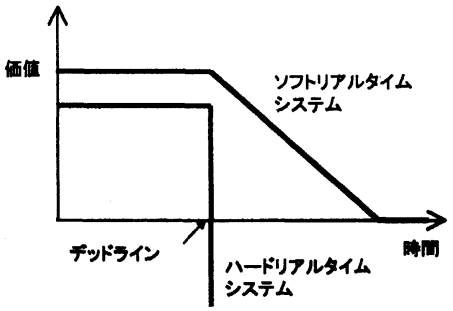

Fig. 1: リアルタイムシステムの種類

2.2 価値関数の導入

リアルタイムシステムとその他のシステムの最も大きな違いはプロセスの処理の価値が時間によって変化するということである．リアルタイムシステムでは，システムの応答時間はシステムの正確さのなかで最も重要な部分である．よって，それを表現するためにタスクごとに価値関数（value function）を導入する．価値関数というのはタスクの実行価値の時間的変化を関数で表したものである．価値関数の例をここで挙げておく（図２）．

- V1　：カーナビの位置更新などのタスク．
- V2　：センサーの入力などのタスク．
- V3　：ソフトリアルタイムのように価値が落ちていき負になるタスク．
- V4　：衛星の発射などのタスク．

2.3 価値関数を用いたスケジューリングアルゴリズム

価値関数を用いたスケジューリングアルゴリズムを説明する．価値関数を用いたスケジューリングアルゴリズムに以下の２つのものが提案されている．[4][5]

HVF (Highest Vlue First)
タスクの処理終了後の価値が待ち行列で最も高いタスクから実行していく。

HVDF(Highest Value Density First)
タスクの value density（価値／計算時間）を求め，待ち行列で最も高い value density をもつタスクから実行していく．

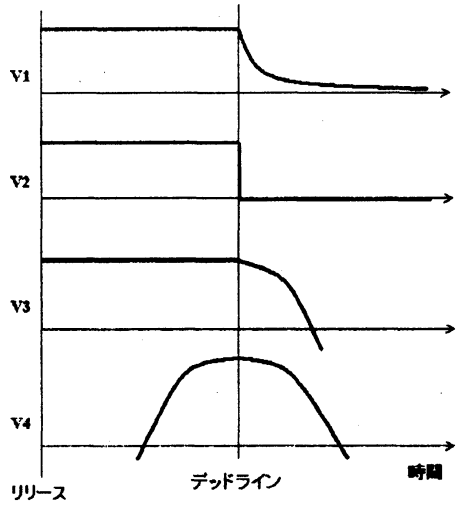

Fig. 2: 価値関数の例

3 拡張時間オートマトン

Wang Yi の手法ではタスクの到着パターンを表すのに拡張された時間オートマトンが用いられている [2]．今回はそのタスク付き時間オートマトンを価値関数を用いて拡張したものを解析モデルとして用いる．

3.1 syntax

まずタスクについて定義する．ソフトリアルタイムシステムを扱うためタスクの要素として価値関数を持つ．価値関数はタスクがリリースされてからの時間からタスクの価値を計算する関数である．

Definition 3.1.1　（タスク）
タスク $P_i \in \mathbb{P}$ は $(c_i, d_i, V_i(t_i))$ の要素からなるとする．

- c_i　タスク P_i の残り計算時間
- d_i　タスク P_i の相対デッドライン
- $V_i(t_i)$　タスク P_i の現在の価値（t_i はタスク P_i がリリースされてから経過した時間，V_i は t_i を受けて t_i の時点の P_i の価値を返す価値関数）

また，タスクの情報として以下のものが与えられているとする．

C_i　タスク P_i の計算時間

D_i　タスク P_i の相対デッドライン
（リリースしてからデッドラインまでの時間）

つまりタスクの要素 c_i は $c_i = C_i$ から始まってだんだん減っていき，$c_i = 0$ になったら処理が終了したことを表す．□

次に拡張時間オートマトンの定義について述べる．正式に定義するため，action と呼ばれるアルファベットの有限集合 *Act*，clock と呼ばれる実数値の変数の有限集合 C を定義する．a,b,…は *Act* の要素であり，x_1,x_2,…は C の要素である．$B(C)$ は g などで表され，形式的には clock の制約式の連言で示す．$B(C)$ の要素は clock constraints と呼ばれる．また，m,n は自然数である．

Definition 3.1.2　（拡張時間オートマトン）
action の集合 *Act* とクロックの集合 C，タスクの集合 $\mathbf{P}$ 上のオートマトンは $\langle N, l_0, E, I, M\rangle$ の組である．

- N はロケーションの有限集合
- $l_0 \in N$ は最初のロケーション
- $E \subseteq N \times B(C) \times Act \times 2^C \times N$ はエッジの集合
- I ： $N \hookrightarrow B(C)$ はロケーションを *clock constraint* に割り当てる関数
- M ： $N \hookrightarrow \mathbb{P}$ はロケーションをタスクに割り当てる部分関数

$\langle l, g, a, r, l'\rangle \in E$ のとき $l \xrightarrow{g,a,r} l'$ と書く．また，通常，C から非負の実数への関数（*clock assignment* と呼ばれる）によって *clock* の値を示し，*clock assignment* の集合を ν で示す．□

拡張時間オートマトンの状態は (l, σ, q) で表す．

l　現在操作しているノード

σ　clock の現在の値を示す clock assignment

q　現在の task queue

task queue は OS の待ち行列のようなものでここにタスクが加えられ，先頭から実行され処理が終了するとそのタスクは task queue から取り除かれる．

3.2 semantics

次に拡張時間オートマトンの semantics について説明していく．まず task queue の変化を計算するために次の関数を定義する．

Definition 3.2.1　（関数 Sch，関数 Run）

$Sch(q)$　*task queue* のあるスケジューリングアルゴリズムでソーティングする関数

$Run(q,t)$　実数 t が与えられたときに，時間 t 後の *task queue* を返す関数

□

関数 $Run(q,t)$ は次のような計算を行う．

全てのタスクを実行した場合

$$
\begin{aligned}
&(t \geq c_{P1} + c_{P2} + \cdots + c_{Pn})\\
&Run(q,t) =\\
&[P^1(0, d_{P1} - c_{P1}, V_{P1}(t_{P1} + c_{P1})),\\
&P^2(0, d_{P2} - (c_{P1} + c_{P2}), V_{P2}(t_{P2} + c_{P1} + c_{P2})),\\
&\vdots\\
&P^n(0, d_{Pn} - (c_{P1} + c_{P2} + \cdots + c_{Pn}), V_{Pn}(t_{Pn} +\\
&\quad c_{P1} + c_{P2} + \cdots + c_{Pn}))]
\end{aligned}
$$

タスク P^i まで実行した場合

$$
\begin{aligned}
&(c_{P1} + \cdots + c_{Pi} \leq t \leq c_{P1} + \cdots + c_{Pi} + c_{P_{i+1}})\\
&Run(q,t) =\\
&[P^1(0, d_{P1} - c_{P1}, V_{P1}(t_{P1} + c_{P1})),\\
&\vdots\\
&P^i(0, d_{Pi} - (c_{P1} + \cdots + c_{Pi}), V_{Pi}(t_{Pi} + c_{P1} +\\
&\quad \cdots + c_{Pi})),\\
&P^{i+1}(c_{Pi+1} - (t - (c_{P1} + \cdots + c_{Pi})), d_{Pi+1} -\\
&\quad t, V_{Pi+1}(t_{Pi+1} + t)),\\
&\vdots\\
&p^n(c_{Pn} - (t - (c_{P1} + \cdots + c_{Pi})), d_{Pn} -\\
&\quad t, V_{Pn}(t_{Pn} + t))]
\end{aligned}
$$

ここではタスクを task queue の先頭から順に $P^1, P^2, \cdots, P^n$ で表し，$1 \leq i \leq n$ とし，時間経過前のタスク P^i は $P^i(c_{Pi}, d_{Pi}, V_{Pi}(t_{Pi}))$ で表している．

関数 Sch は次のような処理を行う．従来の（価値関数を用いない）スケジューリングアルゴリズ

ムでは次のような計算をしてタスク P_i をタスク P_k より task queue の前に並べ替える.

- **Highestpriority first(FPS):**
  $P_i \in q, \forall P_k \in q\ Pr(P_i) \geq Pr(P_k)$
  ここで $Pr(P)$ は P の固定優先度を示す
- **Earliest deadline first(EDF):**
  $P_i \in q, \forall P_k \in q\ d_i \leq d_k$
- **First come first served(FCFS):**
  $P_i \in q, \forall P_k \in q\ D_i - d_i \geq D_k - d_k$
- **Least laxity first(LLF):**
  $P_i \in q, \forall P_k \in q\ d_i - c_i \leq d_k - c_k$

しかし，HVF や HVDF ような価値関数を用いた場合，時間経過中にタスクの優先度の高低が入れ替わることがある．後述するが，従来のアルゴリズムを用いた解析と同じく，今回もタスクが新しく加えられたときに関数 Sch で task queue を並べ替えるようにしている．そこで，まず実行中のタスクが新しく加わったタスク以外にプリエンプトされる事のないように価値関数を単調減少の関数のみとする．単調減少でないタスクの例とは図２．２のＶ４のようなデッドライン付近で最も価値が高くなるような価値関数である．このようなタスクは特別な例であるため今回は対象外とする．また，タスク実行中（時間経過中）に次に優先度の高いタスクが変わることを考えて HVF と HVDF の関数 Sch は次のようにする．n は task queue q の中のタスクの数とする．

HVF

$$Sch(q) = \begin{cases} \cdot n = 0,1 \text{ の場合：何もしない} \\ \cdot n \geq 2 \text{ の場合：次のタスク } P_i \\ \quad \text{を } q \text{ の先頭にもってきて，残} \\ \quad \text{りのタスクを次のようにする} \\ P_i \in q, \forall P_k \in q \\ V_i(t_i + c_i) \geq V_k(t_k + c_k) \\ Sch([P_j(c_j - c_i, d_j - c_i, \\ \qquad V_j(t_j + c_i)), \\ \quad P_k(c_k - c_i, d_k - c_i, \\ \qquad V_k(t_k + c_i)), \\ \quad \cdots]) \end{cases}$$

HVDF

$$Sch(q) = \begin{cases} \cdot n = 0,1 \text{ の場合：何もしない} \\ \cdot n \geq 2 \text{ の場合：次のタスク } P_i \\ \quad \text{を } q \text{ の先頭にもってきて，残} \\ \quad \text{りのタスクを次のようにする} \\ P_i \in q, \forall P_k \in q \\ V_i(t_i + c_i)/c_i \geq V_k(t_k + c_k)/c_k \\ Sch([P_j(c_j - c_i, d_j - c_i, \\ \qquad V_j(t_j + c_i)), \\ \quad P_k(c_k - c_i, d_k - c_i, \\ \qquad V_k(t_k + c_i)), \\ \quad \cdots]) \end{cases}$$

このように，実行中のタスクの処理後の状態のタスクを考えて，タスクを再帰的に並べ替えていくことにより，タスクが新しく加えられたときのみ関数 Sch で task queue を並べ替えればよいようにしている．

この関数を用いて拡張時間オートマトンの semantics 定義する．

Definition 3.2.2　(semantics)

スケジューリング関数 Sch が与えられたときの，初期状態 (l_0, σ_0, q_0) のタスク付き時間オートマトン $\langle N, l_0, E, I, M \rangle$ の遷移体系を次のように定義する．

- $(l, \sigma, q) \xrightarrow{a} (m, \sigma[r \mapsto 0], Sch(M(m) :: q))$ *if* $l \xrightarrow{g,a,r} m$ *and* $\sigma \models g$　(遅延遷移)
- $(l, \sigma, q) \xrightarrow{t} (l, \sigma+t, Run(q,t))$ *if* $(\sigma+t) \models I(l)$　(離散遷移)

d は非負の実数であり，$\sigma + d$ は変数 C の各 *clockx* を値 $\sigma(x) + d$ に写像する *clock assignment* である．$r \subseteq C$ であり，$[r \mapsto 0]\sigma$ は r の各 *clock* を値０に写像する C の *assignment* であり，r を除く C 上の σ と一致する．$\sigma \in g$ であるなら *clock assignment* σ は制限 g を満たすことになる．ここで $M(m) :: q$ とは $M(m)$ を q に挿入した *task queue*. □

3.3　動作例

図３の動作例の１つを示す.

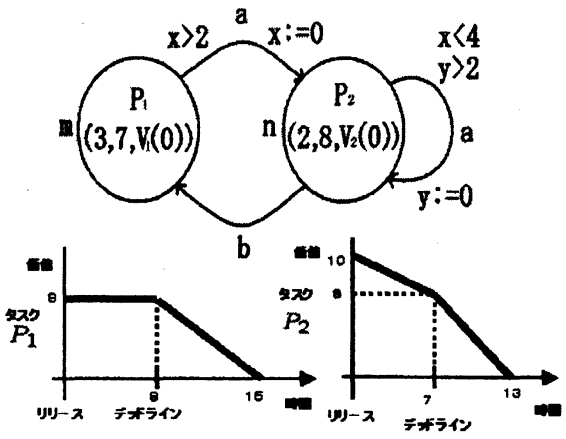

Fig. 3: 拡張時間オートマトンの例 1

$$
\begin{aligned}
&(m, [x=0, y=0][]) \\
&\xrightarrow{3} (m, [x=3, y=3], []) \\
&\xrightarrow{a} (n, [x=0, y=3], [P_2(2,8,V_2(0))]) \\
&\xrightarrow{a} (n, [x=0, y=0], [P_2(2,8,V_2(0)), \\
&\qquad\qquad P_2(2,8,V_2(0))]) \\
&\xrightarrow{3} (n, [x=3, y=3], [P_2(1,5,V_2(3))]) \\
&\xrightarrow{a} (n, [x=3, y=0], [P_2(2,8,V_2(0)), \\
&\qquad\qquad P_2(1,5,V_2(3))]) \\
&\xrightarrow{1} (n, [x=4, y=1], [P_2(1,7,V_2(1)), \\
&\qquad\qquad P_2(1,4,V_2(4))]) \\
&\xrightarrow{b} (m, [x=4, y=1], [P_2(1,7,V_2(1)), \\
&\qquad P_2(1,4,V_2(4)), P_1(3,7,V_1(0))]) \\
&\xrightarrow{1} (m, [x=5, y=2], [P_2(1,4,V_2(5)), \\
&\qquad\qquad P_1(3,6,V_1(1))]) \\
&\xrightarrow{a} (n, [x=0, y=2], [P_2(1,4,V_2(5)), \\
&\qquad P_1(3,6,V_1(1)), P_1(3,7,V_1(0))]) \\
&\vdots
\end{aligned}
$$

まず最初にノード m で 3 の遅延遷移が起こり時間が経過する．時間が経過するとリリースされてからの経過時間 t_i が加算されていく．次にノード n への離散遷移が起こり task queue にタスク P_2 が加えられ，続いて同様に離散遷移が起こり task queue にタスク P_2 が加えられる．そして 3 の遅延遷移が起こると時間が経過するので task queue の先頭のタスクが実行され終了し，task queue から取り除かれる．また，5行目では HVDF の方針に従って value density の高い新しく来たタスク P_2 が task queue の先頭に並べ替えられている．このように拡張時間オートマトンは動作していく．

4 解析手法

本章では性能解析の手法を説明していく．

4.1 解析方針

本解析では，あるスケジューリングアルゴリズムに対し，拡張時間オートマトンの初期状態から到達可能な全状態を調べ，それぞれのタスクについて処理を完了した時の価値の値の最大値と最小値を求め，それをシステムに対するスケジューリングアルゴリズムの評価とする．また，本解析ではシステムに対して次の仮定を設ける．

- 価値関数は単調減少
- 扱うスケジューリングアルゴリズムはプリエンプト
- システムは正常に動作し続ける（タスクが処理されずに task queue に溜まっていき，延々伸びていく可能性がない）

本解析では拡張時間オートマトンの解析に時間オートマトンの解析で一般的に用いられる zone automaton を用いる．zone automaton の successor（ある状態から到達可能な状態）は次のように求めることができる．詳しくは [3] に示されている．

Definition 4.1.1 (Zone Successor)
時間オートマトン A の *clock zone* φ，遷移 e において，*clock interpretation* の集合 $succ(\varphi, e)$ は *clock zone* である．

zone automaton は $zone(l, \varphi)$ と $(s', succ(\varphi, e))$ の間に遷移を加えると得ることができる．オートマトン A において，*zone automaton* $Z(A)$ は次のような遷移体系である．

- $Z(A)$ の状態は A の *zone* である．
- A の初期ロケーション s 毎において，$(l, [X := 0])$ は $Z(A)$ の初期ロケーションである．

- A の遷移 $e=(l,a,\psi,\lambda,l')$ 毎と *clock zone* φ 毎に，遷移 $((l,\varphi),a,(s',succ(\varphi,e)))$ がある．

□

4.2 拡張時間オートマトンのsuccessor

通常 zone automaton の successor はその zone (l,φ) の入力シンボルに対して最大でも 1 つしか持たない．しかし，拡張時間オートマトンを計算するためには task queue の状態が時間とともに変化するため経過した時間の値が重要になってくる．そこで次のものを定義する．そして拡張時間オートマトンでは通常の clock に clock t_l を追加し，通常の successor を t_l の値で場合分けしたものを拡張時間オートマトンの successor とする．

拡張時間オートマトンの successor を計算するため新しく clock と変数を定義する．

Definition 4.2.1　時間経過を測る clock

t_l ロケーション l で経過した時間を表す *clock*

□

まず，時間経過 t_l を，いくつタスクを実行したかについて successor を以下のように場合分けする．オートマトンの動作は zone で計算していくので t_l の値は zone で与えられ，t_l の値は定数ではわからない．よってクロック t_l だけでは task queue の状態を厳密に場合分けできない．しかし t_l を定めなければ task queue を計算できないので，今回は次の二つに successor を場合分けする．

- invariant を満たす限り長く状態に留まる場合 (invariant がない場合 task queue のタスクが全て処理されるまで)
- guard を満たし次第すぐに遷移する場合 (guard がない場合即時遷移)

この長くとどまる場合の clock zone を ρ_1 とし，早く遷移する場合の clock zone を ρ_0 とする．

Definition 4.2.2 (Extend Zone Successor)

拡張時間オートマトン A の *clock zone* φ，遷移 e，時間経過 ρ_0,ρ_1 における，*clock interpretation* の集合 $succ(\varphi,e,\rho_i)$ は *clock zone* である．拡張時間オートマトンの *zone automaton* は $zone(l,\varphi,q)$ と $(l',succ(\varphi,e,\rho_i),Sch(M(l')::Run(q,t_l)))$ の間に遷移を加えることで得ることができる．

拡張時間オートマトン A の *zone automaton* $Z(A)$ は次のような遷移体系である．

- $Z(A)$ の状態は A の *zone* である．
- A の初期ロケーション l 毎において，$(l,[X:=0],q)$ は $Z(A)$ の初期ロケーションである．
- A の遷移 $e=(l,a,\psi,\lambda,l')$ と *clock zone* φ，そのときの時間経過 $i=0,1$ 毎に，遷移

$$((l,\varphi,q),a,(l',succ(\varphi,e,\rho_i),\ Sch(M(l')::Run(q,t_l)))$$

がある．

□

4.3 解析アルゴリズム

拡張時間オートマトンの zone automaton の初期状態を (l_0,φ_0,q) とする．n は現在 task queue にあるタスクの数とする．

1. (l_0,φ_0,q) から遷移可能な全ての e と時間経過 ρ_0,ρ_1 について $(l,\varphi = succ(\varphi_0,e,\rho_i),Sch(M(l) :: Run(q,t_l)))$ を求める
2. 求めた全ての successor に対して，そこから遷移可能な全ての e と時間経過 ρ_0,ρ_1 について successor $(l',succ(\varphi,e,\rho_i),Sch(M(l')::Run(q,t_l)))$ を求める
3. 2 を新しい状態が出力されなくなるまで行う
4. 各タスクに対して処理が完了したときの価値関数の値の最大値，最小値を求める

4.4 Example

例として図 4 のシステムを考える．初期状態 $(m_1,x=0,[])$ から上記の successor で到達可能な状態を求めると図 5 のようになる．矢印の先のないものは，既に求めた状態と同じ状態になる．各タスクの価値の最大，最小値は次の表のようになる．

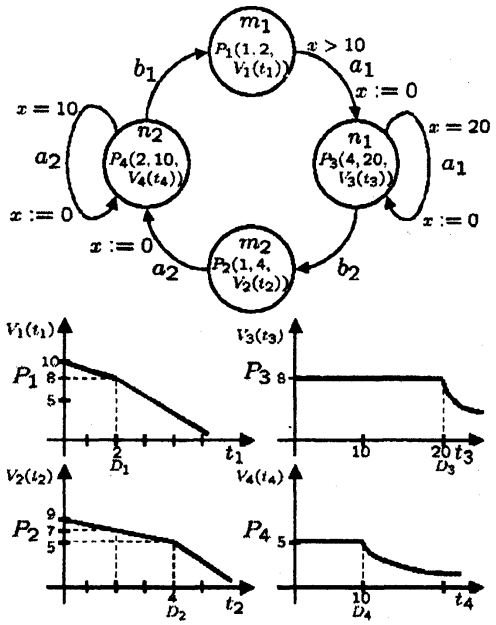

Fig. 4: 拡張時間オートマトンの例2

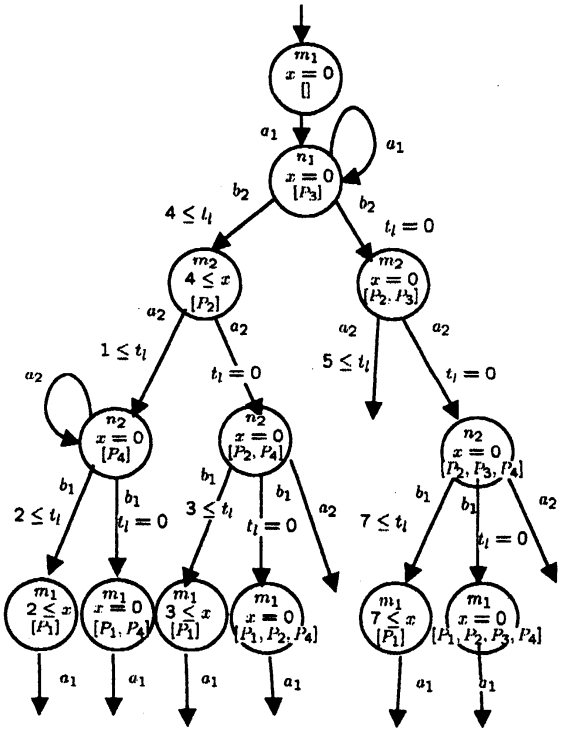

Fig. 5: 計算木

	最大値	最小値
P_1	$V_1(1)$	$V_1(1)$
P_2	$V_2(1)$	$V_2(2)$
P_3	$V_3(4)$	$V_3(6)$
P_4	$V_4(2)$	$V_4(8)$

5 結論

まとめとしてソフトリアルタイムシステムの性質を表現できるようにタスク付き時間オートマトンを拡張し，ソフトリアルタイムシステムに対するスケジューリングアルゴリズムの性能解析を行う方法を提案することができた．今後の課題としては，今回時間経過によって2つに場合分けしたが，クロックの数を増やすなどしてもっと厳密に場合分けすることによって価値の平均などを導出できるようにする．また，タスクが処理されずにtask queueに溜まっていく可能性がないとしたが，その可能性ないかを検証できるようにすることがある．そして実際のシステム，分散データベースなどを対象に解析を行えるようにしていきたい．

参考文献

[1] C.Norstrom,A.Wall,Wang Yi , Timed Automata as Task Models for Event-Driven Systems , RTCSA,pp.182-189,IEEE CS,1999.

[2] E.Fersman,Wang Yi , A Generic Approach to Schedulability Analysis of Real Time Tasks , In Nordic Journal of Computing, Volume 11, 2006(to appear).

[3] R Alur , Timed Automata.,LNCS 1633, pp. 8-22, Springer-Verlag, 1999.

[4] D.Jensen,C.D.Locke,H.Tokuda , A Time-Driven Scheduling Model for Real-Time Operating Systems,pp. 112-122,IEEE Real-Time Systems Symposium ,1985.

[5] Y.Ronen,D.Mosse,M.E.Pollack , Value-Density Algorithms for the Deliberation-Scheduling Problem. SIGART Bulletin 7(2),pp.41-49 ,ACM,1996.

[6] Mark K.Gardner,Jane W.-S.Liu , Analyzing Stochastic Fixed-Priority Real-Time Systems, LNCS 1579, pp.44-58, springer-Verlag,1999.