日本オペレーションズ・リサーチ学会和文論文誌
Transactions of the Operations Research
Society of Japan
2006 年 49 巻 46-61

ニューロ・ダイナミックプログラミングによる負荷分散システムの離散時間負荷分散政策

井家 敦　　大野 勝久
神奈川工科大学　　愛知工業大学

(受理 2003 年 3 月 31 日; 再受理 2005 年 7 月 28 日)

和文概要　負荷分散とは，与えられたシステム構成のもとで性能を最大限に発揮できるように，各要素に負荷を割り当てることである．本論文では，システム情報の収集を一定周期ごとに行なう動的な負荷分散システムを考え，その最適負荷分散政策を導く問題を時間平均マルコフ決定過程として定式化する．さらに，実用的な時間内に準最適な負荷分散政策を求める方法である，ニューロ・ダイナミックプログラミングを用いて計算された準最適な負荷分散政策を従来の負荷分散方式と比較し，その費用低減効果を明らかにする．

キーワード: 計算機, ニューロ・ダイナミックプログラミング, マルコフ決定過程, シミュレーション, 負荷分散システム

1. はじめに

負荷分散とは，与えられたシステム構成のもとで性能を最大限に発揮できるように，各要素に負荷を割り当てることであり，現在，コンピュータ，通信あるいは生産システムにおいて重要視されている問題の 1 つである．負荷分散は以下に述べる静的負荷分散と動的負荷分散に大別される．

静的負荷分散はシステムに固有な情報（例えば，サーバの平均処理能力やジョブの平均到着率など）のみを用いた負荷分散方式である．Tantawi and Towsley[19]，Kameda et al.[9]，Li and Kameda[10] では，負荷分散システムにおける各サーバを待ち行列モデルと考え，システムのレスポンスタイムを最小化する非線形計画問題として定式化し，その最適解を効率よく求めるアルゴリズムを提案している．

一方，動的負荷分散ではこれらに加え，動的に変化するシステムの情報も用いる負荷分散方式である．負荷分散に用いる情報としてさまざまなものがあるが，Shivaratri et al.[17] はその中でも特に各ノードのジョブ数を利用することが最も適していると述べている．Eager et al.[5] では，M/M 型の並列待ち行列システムに対して，閾値方式，最小負荷サーバ方式におけるレスポンスタイムを解析的に導いている．また Zhou[20] は，さらに多くの負荷分散方式をシミュレーションにより評価している．

Eager et al.[5] は，動的負荷分散は静的負荷分散よりよい性能を与えることを示している．しかしながら，実際的な問題として動的負荷分散は，静的負荷分散より多くのオーバーヘッド（例えば，情報収集，負荷分散処理など）を必要とする．[5] では，任意のジョブがシステムに到着した時点でその時のシステムの情報が得られる場合は，効果的な負荷分散が行なえるが，少しでも情報が古い場合はその保証がないことを指摘している．さらに，実際の大規模なシステムで，ジョブが到着するたびに情報の収集を行なうことは非常に困難であることも述べている．したがって，情報を収集する時点を適切に定め，その時点ごとに得られる情

報を有効利用した負荷分散が求められる．それらに関する研究は，例えば，Mirchandaney et al.[11, 12]，Mitzenmacher[13]，Dahlin[3] などで見ることができる．

本論文では，一定周期ごとにシステム情報の収集を行なう動的な負荷分散システムを考え，その最適負荷分散政策を導く問題を時間平均マルコフ決定過程 (Undiscounted Markov Decision Processes, UMDP) により定式化する．さらに，実用的な時間内で準最適な負荷分散政策を求める方法として，近年注目されているニューロ・ダイナミックプログラミング (Neuro-Dynamic Programming, NDP) アルゴリズム [2] を適用し，従来の負荷分散方式とのシミュレーション比較によりその有効性を評価する．ここで，NDP は人工知能の分野で強化学習 (Reinforcement Learning, RL)[18] と呼ばれている．

以下 2 節では，対象となる負荷分散システムについて述べ，UMDP として定式化を行なう．3 節では，2 節で定式化した UMDP に対し，最適負荷分散政策を求める方法の 1 つである修正政策反復法（Modified Policy Iteration Method, MPIM）[14, 16] を述べる．4 節では，NDP について述べ，そのアルゴリズムとして Semi-Markov Average Reward Technique (SMART)[4]，Simulation-Based Policy Iteration Algorithm (SBPI)[7] ，Simulation-Based Modified Policy Iteration Method (SBMPIM)[15] を挙げる．5 節では，負荷分散問題に対して前節の NDP アルゴリズムを適用し，平均費用，政策および計算時間の比較により，各アルゴリズムの性能を評価する．また，従来の負荷分散方式として [20] などで述べられている確率的割り当て方式，ラウンドロビン割り当て方式，最小負荷サーバ割り当て方式の 3 方式とのシミュレーション比較により，NDP アルゴリズムによる負荷分散方式の有効性を示す．最後に 6 節でまとめと今後の課題を述べる．

2. 負荷分散システム

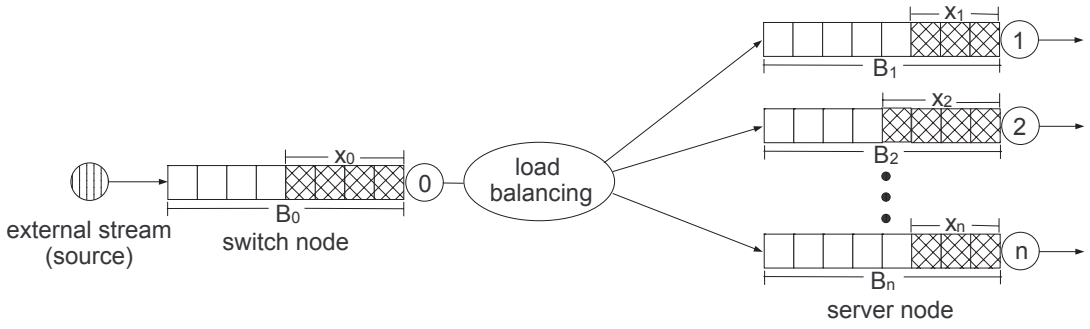

図 1: 負荷分散システム

図1に示すような，1個のスイッチノードとネットワークリンクにより接続されたn個のサーバノードで構成されるシステムを考える．スイッチノードの番号を0とし，各サーバノードの番号を$1, \ldots, n$とする．また，サーバノード，ノード全体の集合をそれぞれ$\mathcal{M}_s = \{1, \ldots, n\}$，$\mathcal{M} = \{0, \ldots, n\}$とする．

単一種類のジョブは外部からスイッチに到着し，負荷分散処理により各サーバに割り当てられ，処理を受ける．処理を完了したジョブは直ちにシステムを退去する．サーバ間のジョブ転送はないものとし，スイッチ，各サーバともバッファを持ち，その容量をB_i $(i \in \mathcal{M})$とする．スイッチにおいて，容量を超えるジョブが到着した場合，そのジョブはサービスを受けることなく，直ちにシステムを退去する．

システムの観測時点を t $(t=0,1,\ldots)$ とする．時間区間 $[t,t+1)$ でのスイッチへのジョブの到着数，サーバ i でのジョブの処理可能数をそれぞれ $A(t)$，$S_i(t)$ で表す．$A(t)$，$S_i(t)$ はそれぞれ互いに独立で，時点 t に依存しない離散分布 $\Pr\{A(t)=l\}=\alpha(l),\ \Pr\{S_i(t)=l\}=\sigma_i(l),\ i\in\mathcal{M}_s,\ (l=0,1,\ldots)$ に従うものとする．また，時点 t でのノード i における処理中のジョブを含むジョブ数を $X_i(t)$ で表せば，システムの状態は $\boldsymbol{X}(t)=(X_i(t);i\in\mathcal{M})$ で表すことができる．

負荷分散システムの負荷分散処理の手順を以下に示す．システムの状態が各観測時点ごとにスイッチに転送される．スイッチは，システムの状態をもとに負荷分散政策 $\boldsymbol{\pi}\equiv(\boldsymbol{f}(\boldsymbol{x});\boldsymbol{x}\in\mathcal{X})$ に従い各サーバへジョブを割り当てる．ここで，$\boldsymbol{f}(\boldsymbol{x})\equiv(f_i(\boldsymbol{x});i\in\mathcal{M}_s)$ は各サーバへのジョブの転送数のベクトルであり，状態 $\boldsymbol{x}$ での決定となる．また，$\mathcal{X}$ はシステムの状態空間であり，各ノードの容量はそれぞれ B_i $(i\in\mathcal{M})$ であることから

$$\mathcal{X}=\left\{(x_i;i\in\mathcal{M});\ 0\le x_i\le B_i,\ i\in\mathcal{M}\right\} \tag{2.1}$$

で与えられる．以上で述べた負荷分散システムに対し，単位時間当たりの平均費用を最小化する負荷分散政策を導く問題を考える．

スイッチ内のジョブのみが各サーバへ転送されることを考慮し，各サーバ内のジョブを含めて容量を超える転送は行なわないものと仮定すると，状態 $\boldsymbol{x}$ でとりうる決定の集合は

$$\mathcal{K}(\boldsymbol{x})=\left\{(f_i;i\in\mathcal{M}_s);\ \sum_{j\in\mathcal{M}_s} f_j\le x_0,\ f_i+x_i\le B_i,\ i\in\mathcal{M}_s\right\},\ \boldsymbol{x}\in\mathcal{X} \tag{2.2}$$

で与えられる．また，システムの政策空間は (2.2) 式を用いて $\boldsymbol{\Pi}=\{\boldsymbol{\pi};\boldsymbol{f}(\boldsymbol{x})\in\mathcal{K}(\boldsymbol{x}),\ \boldsymbol{x}\in\mathcal{X}\}$ で与えられる．

状態 $\boldsymbol{X}(t)$ から $\boldsymbol{X}(t+1)$ への状態遷移は，時点 t での決定 $\boldsymbol{f}\in\mathcal{K}(\boldsymbol{X}(t))$，スイッチへの到着数 $A(t)$，サーバ i の処理可能数 $S_i(t)$ $(i\in\mathcal{M}_s)$ を用いて，各ノードについて以下のように与えられる．

$$X_0(t+1) \;=\; \min\{X_0(t)-\sum_{i\in\mathcal{M}_s} f_i+A(t),B_0\} \tag{2.3}$$

$$X_i(t+1) \;=\; \max\{X_i(t)+f_i-S_i(t),0\},\ i\in\mathcal{M}_s \tag{2.4}$$

時点 t での状態 $\boldsymbol{x}$ で決定 $\boldsymbol{f}$ をとったとき，時点 $t+1$ で状態 $\boldsymbol{x}'$ へ遷移する確率 $p(\boldsymbol{x},\boldsymbol{x}',\boldsymbol{f})=\Pr(\boldsymbol{X}(t+1)=\boldsymbol{x}'|\boldsymbol{X}(t)=\boldsymbol{x},\boldsymbol{f})$ は，各ノードの遷移確率 $q_i(x_i,x_i',\boldsymbol{f})$ $(i\in\mathcal{M})$ の積で与えられる．すなわち，

$$p(\boldsymbol{x},\boldsymbol{x}',\boldsymbol{f})=\prod_{i\in\mathcal{M}} q_i(x_i,x_i',\boldsymbol{f}) \tag{2.5}$$

である．

スイッチでの状態遷移確率 $q_0(x_0,x_0',\boldsymbol{f})$ は，(2.3) 式より各サーバへのジョブ転送数と外部からの到着数によって決まる．決定後のスイッチ内のジョブ数は $(x_0-\sum_{i\in\mathcal{M}_s} f_i)$ 個であり，次の時点までに $(x_0'-x_0+\sum_{i\in\mathcal{M}_s} f_i)$ 個のジョブが到着し，スイッチの容量が B_0 であることと $x_0'-x_0+\sum_{i\in\mathcal{M}_s} f_i\ge 0$ を考慮すれば，

$$q_0(x_0,x_0',\boldsymbol{f})=\begin{cases}\alpha(x_0'-(x_0-\sum_{i\in\mathcal{M}_s} f_i)) & ,\ \text{for } 0\le x_0'<B_0\\ \sum_{l=B_0-(x_0-\sum_{i\in\mathcal{M}_s} f_i)}^{\infty}\alpha(l) & ,\ \text{for } x_0'=B_0\\ 0 & ,\ \text{otherwise}\end{cases} \tag{2.6}$$

で与えられる．

サーバ i $(i \in \mathcal{M}_s)$ での状態遷移確率 $q_i(x_i, x_i', \boldsymbol{f})$ は，(2.4) 式よりスイッチからのジョブ転送数とジョブ処理数によって決まる．すなわち，決定後のサーバ内のジョブ数は $(x_i + f_i)$ 個であり，次の時点までに $(x_i + f_i - x_i')$ 個のジョブが処理され，$x_i + f_i - x_i' \geq 0$ を考慮すれば

$$q_i(x_i, x_i', \boldsymbol{f}) = \begin{cases} \sigma_i((x_i + f_i) - x_i') & , \text{ for } 0 < x_i' \leq B_i \\ \sum_{l=x_i+f_i}^{\infty} \sigma_i(l) & , \text{ for } x_i' = 0 \\ 0 & , \text{ otherwise,} \end{cases} \qquad i \in \mathcal{M}_s \tag{2.7}$$

で与えられる．

システムの費用として以下を考える:

C_i^H: 単位時間当たりの 1 つのジョブに対するノード i $(i \in \mathcal{M})$ での保持費用
C^R: 単位時間当たりのスイッチでの 1 つのジョブに対する故損費用
C_i^P: 単位時間当たりのサーバ i $(i \in \mathcal{M}_s)$ で 1 つのジョブが処理を受けたときの効用

これらを用いて，状態 $\boldsymbol{x}$ で決定 $\boldsymbol{f}$ をとり，次の状態 $\boldsymbol{x}'$ へ遷移したときの費用 $r(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{f})$ は

$$\begin{aligned} r(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{f}) &= C_0^H(x_0 - \sum_{i \in \mathcal{M}_s} f_i) + \sum_{i \in \mathcal{M}_s} C_i^H(x_i + f_i) \\ &+ I_{\{x_0' = B_0\}} C^R \sum_{l=0}^{\infty} l \times \alpha(B_0 - x_0 + \sum_{i \in \mathcal{M}_s} f_i + l) \\ &- \sum_{i \in \mathcal{M}_s} C_i^P(x_i + f_i - x_i') \end{aligned} \tag{2.8}$$

で与えられる．ここで，I_X は X が真のとき 1，偽のとき 0 を与える指示関数である．したがって，状態 $\boldsymbol{x}$ で決定 $\boldsymbol{f}$ をとったときの即時費用 $r(\boldsymbol{x}, \boldsymbol{f})$ は

$$r(\boldsymbol{x}, \boldsymbol{f}) = \sum_{\boldsymbol{x}' \in \mathcal{X}} p(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{f}) r(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{f}) \tag{2.9}$$

で与えられる．

以上より，最適負荷分散政策を導く問題は UMDP として定式化され，その最適性方程式は次式で与えられる [1, 8, 16].

$$g + h(\boldsymbol{x}) = \min_{\boldsymbol{f} \in \mathcal{K}(\boldsymbol{x})} \{r(\boldsymbol{x}, \boldsymbol{f}) + \sum_{\boldsymbol{x}' \in \mathcal{X}} p(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{f}) h(\boldsymbol{x}')\}, \quad \boldsymbol{x} \in \mathcal{X} \tag{2.10}$$

ここで，g，$h(\boldsymbol{x})$ はそれぞれ，平均費用，状態 $\boldsymbol{x}$ での相対費用であり，各 $\boldsymbol{x}$ について最適性方程式 (2.10) の右辺を最小化する定常な政策 $\boldsymbol{\pi}^* = (\boldsymbol{f}^*(\boldsymbol{x});\ \boldsymbol{x} \in \mathcal{X})$ が最適負荷分散政策となり，そのときの g^* が最小平均費用となる．ただし厳密には，(2.10) は最適政策のもとでの各連鎖内の状態で成立し，過渡状態に対しては別の最適性方程式が必要である．しかし，本論文の NDP アルゴリズムに対してはすべてシミュレーションを用いており，初期状態 $\boldsymbol{x}_0$ から訪問した状態だけを対象とするため，(2.10) だけを考慮すればよい．なお，最適負荷分散問題は，ジョブの到着数分布および各サーバの処理可能数分布に関する弱い条件のもとで単一連鎖になることを示すこともできる．

3. 修正政策反復法

(2.10) 式を解くアルゴリズムとして，ここでは修正政策反復法 (Modified Policy Iteration Method : MPIM) を挙げる．MPIM は Howard[8] による政策反復法 (Policy Iteration Method : PIM) の政策改良と政策評価（値決定）ルーチンのうち，政策評価ルーチンを逐次近似法に置き換えた方法である．また，MPIM は値反復法 (Value Iteration Method: VIM) を特別な場合として含み，政策評価ルーチンの反復回数を 0 としたとき，VIM のアルゴリズムと一致する．MPIM のアルゴリズムを以下に示す.

修正政策反復法 (MPIM)

Step 1(初期設定): 適当な $\boldsymbol{x}_r \in \mathcal{X}$ に対して，$h^0(\boldsymbol{x}_r) = 0$ を満たすベクトル $\boldsymbol{h}^0 \equiv (h^0(\boldsymbol{x}); \boldsymbol{x} \in \mathcal{X})$ を与える．初期政策 $\boldsymbol{\pi}^0 \in \boldsymbol{\Pi}$, 非負整数値 L，停止条件 $\varepsilon > 0$ を与え, 反復回数 $m = 0$ とおく.

Step 2(政策の改良): $\boldsymbol{x} \in \mathcal{X}$ に対して,

$$g^{m+1}(\boldsymbol{x}) = \min_{\boldsymbol{f} \in \mathcal{K}(\boldsymbol{x})} \{r(\boldsymbol{x}, \boldsymbol{f}) + \sum_{\boldsymbol{x}' \in \mathcal{X}} p(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{f}) h^m(\boldsymbol{x}') - h^m(\boldsymbol{x})\}$$

を計算し，$\boldsymbol{f}^m(\boldsymbol{x})$ が $g^{m+1}(\boldsymbol{x})$ を与えれば，$\boldsymbol{f}^{m+1}(\boldsymbol{x}) = \boldsymbol{f}^m(\boldsymbol{x})$ とおき，さもなければ, $\boldsymbol{f}^{m+1}(\boldsymbol{x})$ を $g^{m+1}(\boldsymbol{x})$ を与える任意の決定 $\boldsymbol{f}$ ととる.

Step 3(政策評価): $w^0(\boldsymbol{x}) = h^m(\boldsymbol{x}) + g^{m+1}(\boldsymbol{x}), \boldsymbol{x} \in \mathcal{X}$ とおき $l = 0, 1, \ldots, L-1$ に対して順次,

$$w^{l+1}(\boldsymbol{x}) = r(\boldsymbol{x}, \boldsymbol{f}^{m+1}(\boldsymbol{x})) + \sum_{\boldsymbol{x}' \in \mathcal{X}} p(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{f}^{m+1}(\boldsymbol{x})) w^l(\boldsymbol{x}')$$

を計算し，$h^{m+1}(\boldsymbol{x}) = w^L(\boldsymbol{x}) - w^L(\boldsymbol{x}_r), \boldsymbol{x} \in \mathcal{X}$ とおく．$\max_{\boldsymbol{x} \in \mathcal{X}} |h^{m+1}(\boldsymbol{x}) - h^m(\boldsymbol{x})| < \varepsilon$ であれば停止．さもなければ，$m = m + 1$ として，**step 2** へ.

Puterman[16] では，単一連鎖条件のもとでの，MPIM の収束性が示されている.

4. ニューロ・ダイナミックプログラミング

NDP は，大規模 MDP を実用的な時間内で解く方法として近年，注目を集めている．NDP では，探索やシミュレーション，ニューラルネットワークなどを用いた近似を利用する．現在，NDP を用いたアルゴリズムとして SMART[4]，SBPI[7]，SBMPIM[15] などが提案されている．以下でそれらのアルゴリズムを簡単に紹介する.

4.1. SMART[4]

SMART はセミマルコフ決定過程 (Semi-Markov decision process, SMDP) を解く NDP アルゴリズムである．その一般的な考え方は Q-学習 (Q-Learning) に基づいており，政策改良の際に状態遷移確率を計算する必要がない.

Semi-Markov Average Reward Technique(SMART)

Step 1: $m = 0$，相対値 $R_{old}(\boldsymbol{x}, \boldsymbol{f}) = R_{new}(\boldsymbol{x}, \boldsymbol{f}) = 0$，任意の初期政策 $\boldsymbol{\pi} = (\boldsymbol{f}(\boldsymbol{x}); \boldsymbol{x} \in \mathcal{X}) \in \boldsymbol{\Pi}$ とし，任意の初期状態 $\boldsymbol{x}_0 \in \mathcal{X}$ を与える．総費用を $TC = 0$，総時間 $T = 0$, 平均費用を $g = 0$ とする．また，適切な相対値学習パラメータベクトル $\boldsymbol{\gamma} = (\gamma_0, \gamma_\tau)$, 探索パラメータベクトル $\boldsymbol{\eta} = (\eta_0, \eta_\tau)$ を与える.

Step 2: $m < MAX\_STEPS$ であるならば，**Step 3** へ．さもなければ，アルゴリズム停止.

Step 3: 探索と相対値のステップサイズ η_m, γ_m を次のように求める．

$$\eta_m = \frac{\eta_0}{1+\frac{m^2}{\eta_\tau + m}},\ \gamma_m = \frac{\gamma_0}{1+\frac{m^2}{\gamma_\tau + m}}$$

Step 4: 区間 $[0,1]$ 上の一様乱数 U を発生させる．$U < 1-\eta_m$ ならば

$$\boldsymbol{f}^* = \arg\min_{\boldsymbol{f}\in\mathcal{K}(\boldsymbol{x}_m)} R_{old}(\boldsymbol{x}_m, \boldsymbol{f})$$

とし，$\tilde{\boldsymbol{f}} = \boldsymbol{f}^*$ とする．さもなければ $\mathcal{K}(\boldsymbol{x}_m)$ の中から $\boldsymbol{f}^*$ 以外の決定をランダムに選択し，それを $\tilde{\boldsymbol{f}}$ とする．

Step 5: システムのシミュレーションを行い，状態が $\boldsymbol{x}_{m+1}$ に遷移したならば

$$r_{imm} = r(\boldsymbol{x}_m, \boldsymbol{x}_{m+1}, \tilde{\boldsymbol{f}})$$

を即時費用とする．

Step 6: 次式を用いて R_{new} を求める．

$$R_{new}(\boldsymbol{x}_m, \tilde{\boldsymbol{f}}) = (1-\gamma_m)R_{old}(\boldsymbol{x}_m, \tilde{\boldsymbol{f}}) + \gamma_m\{r_{imm} - g + \min_{\boldsymbol{f}\in\mathcal{K}(\boldsymbol{x}_{m+1})} R_{old}(\boldsymbol{x}_{m+1}, \boldsymbol{f})\}$$

Step 7: :$\tilde{\boldsymbol{f}} = \boldsymbol{f}^*$ ならば，**Step8** へ．さもなければ **Step 9** へ．

Step 8: 次式を用いて TC，T，g を更新する．

$$TC = TC + r(\boldsymbol{x}_m, \boldsymbol{x}_{m+1}, \tilde{\boldsymbol{f}}),\ T = T+1,\ g = \frac{TC}{T}$$

Step 9: $R_{old}(\boldsymbol{x}_m, \tilde{\boldsymbol{f}}) = R_{new}(\boldsymbol{x}_m, \tilde{\boldsymbol{f}}), m = m+1$ として **Step 2** へ．

4.2. SBPI[7]

SBPI は PIM の政策評価ルーチンをシミュレーションに置き換えた方法である．

Simulation-Based Policy Iteration Algorithm(SBPI)

Step 1(初期設定): 初期政策 $\boldsymbol{\pi}^0 \in \boldsymbol{\Pi}$ を与える．状態 $\boldsymbol{x}_r \in \mathcal{X}$，シミュレーション長 M を与え，$m = 0$ とする．

Step 2(シミュレーション): g^m と $\boldsymbol{h}^m$ の推定を行なう．

(a) 次の手順にしたがって g^m を推定する．

(i) 任意の初期状態 $\boldsymbol{x}_0 \in \mathcal{X}$ からシミュレーションにより，トラジェクトリー $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_M$ を生成する．

(ii) $g^m = 0$ として，$t = 0, \ldots, M-1$ にたいして $(\boldsymbol{x}_t, \boldsymbol{x}_{t+1})$ の推移に伴う g^m を次式で更新する．

$$g^m = (1-\frac{1}{t+1})g^m + \frac{1}{t+1}r(\boldsymbol{x}_t, \boldsymbol{x}_{t+1}, \boldsymbol{f}^m(\boldsymbol{x}_t))$$

(b) 次の手順にしたがって $\boldsymbol{h}^m$ を推定する．

(i) 上記 **Step 2 (a)-(i)** で訪問回数が最大の状態を $\boldsymbol{x}^*$ とする．

(ii) 任意の状態 $\boldsymbol{x}_0$ から始まり，$\boldsymbol{x}^*$ へ至るトラジェクトリーを L 本生成する．

(iii) 各トラジェクトリー $(\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}^*)$ にたいして，$(\boldsymbol{x}_t, \boldsymbol{x}_{t+1})$ の推移に伴う $w(\boldsymbol{x}_i)$ $(i = 1, 2, \ldots, t)$ を次式で更新する．

$$w(\boldsymbol{x}_i) = w(\boldsymbol{x}_i) + \gamma_i \eta^{t-i} d_t$$

ここで，γ_i はそのトラジェクトリー内で $\boldsymbol{x}_i$ を訪問した回数の逆数，$0 \leq \eta \leq 1$，$d_t = r(\boldsymbol{x}_t, \boldsymbol{x}_{t+1}, \boldsymbol{f}(\boldsymbol{x}_t)) - g^m + w(\boldsymbol{x}_{t+1}) - w(\boldsymbol{x}_t)$ である．

(iv) $\boldsymbol{h}^m$ を次のように与える．

$$h^m(\boldsymbol{x}) = w(\boldsymbol{x}) - w(\boldsymbol{x}_r),\ \boldsymbol{x} \in \mathcal{X}$$

Step 3(政策の改良): すべての $\boldsymbol{x} \in \mathcal{X}$ に対して，

$$\boldsymbol{f}^{m+1}(\boldsymbol{x}) = \arg\min_{\boldsymbol{f} \in \mathcal{K}(\boldsymbol{x})} \{r(\boldsymbol{x}, \boldsymbol{f}) + \sum_{\boldsymbol{x}' \in \mathcal{X}} p(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{f}) h^{m+1}(\boldsymbol{x}')\}$$

を計算する．$\boldsymbol{f}^{m+1}(\boldsymbol{x})$ の候補が複数ある場合はその中の任意の値を $\boldsymbol{f}^{m+1}(\boldsymbol{x})$ とする．$\boldsymbol{\pi}^m = \boldsymbol{\pi}^{m+1}$ ならばアルゴリズム終了．さもなければ，$m = m + 1$ とし，**Step 2** へ．

以上のアルゴリズムにおいて，[7] では **Step 2 (a)** で L_A 本のトラジェクトリーを生成するとしているが，ここでは $L_A = 1$ とした．また，**(b)** では状態分類アルゴリズム（Fox-Landi algorithm，例えば [16] を参照）を用いることになっているが，**(a)** での結果を用いることにした．

4.3. SBMPIM[15]

SBMPIM は，MPIM の政策評価ルーチンをシミュレーションに置き換えることにより計算時間の短縮を図ったものである．[15] では，生産ラインの最適制御問題に対してこの手法を開発し，その有効性を示している．

Simulation-Based Modified Policy Iteration Method(SBMPIM)

Step 1(初期設定): 初期状態 $\boldsymbol{x}_0 \in \mathcal{X}$ と望ましい状態 $\boldsymbol{x}^*$ を定め，シミュレーションステップ数 k および η，$(0 \leq \eta \leq 1)$ を定めて，アルゴリズム終了までに訪問した状態の集合 $\boldsymbol{S}_v = \phi$（空集合），一回の反復で訪問した状態の集合 $\boldsymbol{S}_T = \phi$，累積費用 $TC = 0$，$\boldsymbol{x} = \boldsymbol{x}_0$，$m = l = 1$ とおく．

Step 2(シミュレーション): $\boldsymbol{x} \notin \boldsymbol{S}_v$ ならば，$\boldsymbol{S}_v = \boldsymbol{S}_v \cup \{\boldsymbol{x}\}$，$\boldsymbol{S}_T = \boldsymbol{S}_T \cup \{\boldsymbol{x}\}$，$\boldsymbol{x}$ の訪問回数 $v(\boldsymbol{x}) = 1$ とおき，$f(\boldsymbol{x})$ を状態 $\boldsymbol{x}^*$ へ向かう実行可能な決定（複数存在する場合は適当なものが選ばれる）と定め，$u(\boldsymbol{x}) = r(\boldsymbol{x}, \boldsymbol{f}(\boldsymbol{x}))$ とおく．$\boldsymbol{x} \in \boldsymbol{S}_v$ ならば，$\boldsymbol{x} \notin \boldsymbol{S}_T$ のとき，$\boldsymbol{S}_T = \boldsymbol{S}_T \cup \{\boldsymbol{x}\}$，$v(\boldsymbol{x}) = 1$，$u(\boldsymbol{x}) = r(\boldsymbol{x}, \boldsymbol{f}(\boldsymbol{x}))$ とおき，$\boldsymbol{x} \in \boldsymbol{S}_T$ ならば，

$$v(\boldsymbol{x}) = v(\boldsymbol{x}) + 1,\ u(\boldsymbol{x}) = u(\boldsymbol{x}) + r(\boldsymbol{x}, \boldsymbol{f}(\boldsymbol{x}))$$

と更新する．状態 $\boldsymbol{x}$ で決定 $f(\boldsymbol{x})$ をとったときの状態遷移をシミュレーションし，次期の状態 $\boldsymbol{x}'$ を定める．

$$TC = TC + r(\boldsymbol{x}, f(\boldsymbol{x})),\ \boldsymbol{x} = \boldsymbol{x}'$$

と更新し，$l = k$ ならば **Step 3** へ．さもなければ，$l = l + 1$ として **Step 2** へ

Step 3(g の推定) :平均費用 g を次式により推定する.

$$g = \frac{TC}{k}$$

Step 4(h の推定) :$\boldsymbol{S}_v$ の中で $\boldsymbol{x}_r$ を定め

$$h(\boldsymbol{x}_r) = (1 - \frac{\eta v(\boldsymbol{x}_r)}{k})(w(\boldsymbol{x}_r) - g) + \frac{\eta v(\boldsymbol{x_r})}{k}(\frac{u(\boldsymbol{x}_r)}{v(\boldsymbol{x}_r)} - g)$$

を計算し，$\boldsymbol{x}(\neq \boldsymbol{x}_r) \in \boldsymbol{S}_v$ にたいして

$$h(\boldsymbol{x}) = (1 - \frac{\eta v(\boldsymbol{x})}{k})(w(\boldsymbol{x}) - g) + \frac{\eta v(\boldsymbol{x})}{k}(\frac{u(\boldsymbol{x})}{v(\boldsymbol{x})} - g) - h(\boldsymbol{x}_r)$$

を計算し，$h(\boldsymbol{x}_r) = 0$ とおく．ただし，$m = 1$ のときには

$$h(\boldsymbol{x}_r) = \frac{u(\boldsymbol{x}_r)}{v(\boldsymbol{x}_r)} - g,\ h(\boldsymbol{x}) = \frac{u(\boldsymbol{x})}{v(\boldsymbol{x})} - g - h(\boldsymbol{x}_r)$$

である.

Step 5(政策の改良): すべての $\boldsymbol{x} \in \boldsymbol{S}_v$ に対して

$$w(\boldsymbol{x}) = \min_{\boldsymbol{f} \in \mathcal{K}^{\mathrm{N}}(\boldsymbol{x}, \boldsymbol{f}(\boldsymbol{x}))} \{r(\boldsymbol{x}, f) + \sum_{\boldsymbol{x}' \in \mathcal{X}} p(\boldsymbol{x}, \boldsymbol{x}', f) h(\boldsymbol{x}')\}$$

を計算し，$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{f}$，$v(\boldsymbol{x}) = 1$ とおく．ここで，$\mathcal{K}^{\mathrm{N}}(\boldsymbol{x}, \boldsymbol{f}(\boldsymbol{x}))$ $(\in \mathcal{K}(\boldsymbol{x}))$ は $\boldsymbol{f}(\boldsymbol{x})$ の近傍（解かれる問題によって適当なものが定義される）であり，$p(\boldsymbol{x}, \boldsymbol{x}', f) > 0$ となる $\boldsymbol{x}' \notin \boldsymbol{S}_v$ にたいしては，$\boldsymbol{S}_v = \boldsymbol{S}_v \cup \{\boldsymbol{x}'\}$，$v(\boldsymbol{x}') = 1$ とおき，$\boldsymbol{f}(\boldsymbol{x}')$ を $\boldsymbol{x}^*$ へ向かう実行可能な決定（複数存在する場合は適当なものが選ばれる）とする．$w(\boldsymbol{x}') = r(\boldsymbol{x}', \boldsymbol{f}(\boldsymbol{x}'))$ とおき，$h(\boldsymbol{x}') = h(\boldsymbol{x})$ として，$w(\boldsymbol{x})$ を計算する．$f(\boldsymbol{x})$ が $w(\boldsymbol{x})$ を与えなければ，$w(\boldsymbol{x})$ を与える任意の決定として，$f(\boldsymbol{x})$ を改良する．m が停止回数に達すればアルゴリズム終了．さもなければ $\boldsymbol{S}_T = \phi$，$TC = 0$，$l = 1$，$m = m + 1$ とおき，**Step 2** へ.

5. 数値例

5.1. NDP アルゴリズムの比較

前節で，既存の NDP アルゴリズムを述べたが，まだ適用例が少なく，最適負荷分散問題に対する有効性は知られていない．したがって，まず比較的小規模な負荷分散問題に対して MPIM と比較し，どの NDP アルゴリズムが適しているかを評価する．以下すべての数値例において，スイッチへの到着数の分布を修正された幾何分布，すなわち

$$\alpha(l) = \lambda(1 - \lambda)^l,\ l = 0, 1, \ldots \tag{5.1}$$

で与え，サーバ i の処理可能数の分布も同様に

$$\sigma_i(l) = \mu_i(1 - \mu_i)^l,\ l = 0, 1, \ldots,\ i \in \mathcal{M}_s \tag{5.2}$$

で与える．システムのパラメータとして，$n = 2$，$\lambda = 0.4$，$\mu_1 = 0.2$，$\mu_2 = 0.3$，$C_i^H = 2$ $(i \in \mathcal{M})$，$C^R = 15$，$C_i^P = 0$ $(i \in \mathcal{M}_s)$ を固定し，各ノードの容量の違いによる次の 3 つの問題:

1) $B_i = 5$ $(i \in \mathcal{M})$，2) $B_i = 8$ $(i \in \mathcal{M})$，3) $B_i = 10$ $(i \in \mathcal{M})$

について，比較を行なう．各問題における状態数はそれぞれ 216, 729, 1331 である．また，計算機として，DOS/V 機 (CPU: AMD Athlon XP 2200+, Memory: 512MB，OS: Windows 2000) を使用した．

各アルゴリズムによる，各問題に対する平均費用を表 1 に示す．ここで各アルゴリズムにおけるパラメータとして，MPIM については $\varepsilon = 10^{-5}$，$m = 20$，SMART については $(\eta_0,\ \eta_\tau) = (0.4,\ 5 \times 10^5)$，$(\gamma_0,\ \gamma_\tau) = (0.6,\ 5 \times 10^5)$，$MAX\_STEPS = 1.0 \times 10^6$ (2), 3) については 1.0×10^7)，SBPI については $\eta = 0.9$，$m = 50000$，$L = 1000$，SBMPIM については $\eta = 0.8$，$k = 5000$，停止回数 $= 40$ とした．また，SBMPIM における近傍 $\mathcal{K}^{\mathrm{N}}(\boldsymbol{x}, \boldsymbol{f}')$ は，$\boldsymbol{f}' = (f'_1, f'_2, \ldots, f'_n) \in \mathcal{K}$ に対して，次の 2 つの集合

$$\mathcal{K}^{\mathrm{A}}(\boldsymbol{f}') = \left\{ (f_i;\ i \in \mathcal{M}_s);\ \sum_{j \in \mathcal{M}_s} f'_j - 1 \leq \sum_{j \in \mathcal{M}_s} f_j \leq \sum_{j \in \mathcal{M}_s} f'_j + 1 \right\},\ \boldsymbol{x} \in \mathcal{X}$$

$$\mathcal{K}^{\mathrm{B}}(\boldsymbol{f}') = \left\{ (f_i;\ i \in \mathcal{M}_s);\ f'_i - 1 \leq f_i \leq f'_i + 1, i \in \mathcal{M}_s \right\},\ \boldsymbol{x} \in \mathcal{X}$$

を考え，$\mathcal{K}^{\mathrm{N}}(\boldsymbol{x}, \boldsymbol{f}') = \mathcal{K}(\boldsymbol{x}) \cap \mathcal{K}^{\mathrm{A}}(\boldsymbol{f}') \cap \mathcal{K}^{\mathrm{B}}(\boldsymbol{f}')$ とした．

表 1 より，すべての問題について SBPI と SBMPIM における平均費用は MPIM による最小平均費用に近い値を示した．ここで，SBMPIM の最小平均費用はバッチ平均法を適用して，95%信頼区間を求めている．一方，SMART はあまりよい値には達していない．これは，[15] で述べられている結果と同様の現象であり，最適解に収束していないことを示している．SMART については各パラメータを様々に変化させて実行を行なったが，どの場合についてもよい値を与えることはなかった．このように SMART が良い解を与えない原因として，負荷分散問題のように各状態における決定数が比較的多い問題では，政策改良がうまく行われないことが考えられる．また，計算時間については，問題の状態数の増加に伴い SBMPIM の低減効果が現れた．状態数と計算時間の関係を図 2 に示す．図 2 から，SBMPIM における状態数の計算時間への影響は他のアルゴリズムと比較して非常に小さいことが示される．

さらに，各アルゴリズムの最適解への収束性を政策の比較により検討する．表 2 は 1) で得られた各アルゴリズムの政策の一部を示し，$\surd$ は MPIM と政策が一致したことを示している．結果から明らかなように，SBMPIM はすべての状態について政策が一致し，SBPI も比較的よい政策を与えることがわかる．紙面の都合上，政策の大部分を省略したが，この問題において SBMPIM はすべての状態で MPIM と一致した．したがって，本研究の負荷分散問題に対して，SBMPIM がすべての点で有効であることが示された．

表 1: 各アルゴリズムによる各問題に対する平均費用

アルゴリズム	1)	2)	3)
MPIM	5.906	4.987	4.844
SMART	6.363	5.641	5.283
SBPI	5.878	4.991	4.848
SBMPIM	5.894±0.066	4.968±0.087	4.834±0.101

5.2. 従来の負荷分散方式との比較

SBMPIM が既存の NDP アルゴリズムの中では最もよいことを示した上で，従来の負荷分散方式との比較により，提案法の有効性を示す．[5, 17, 20] では，複数の負荷分散の比較によ

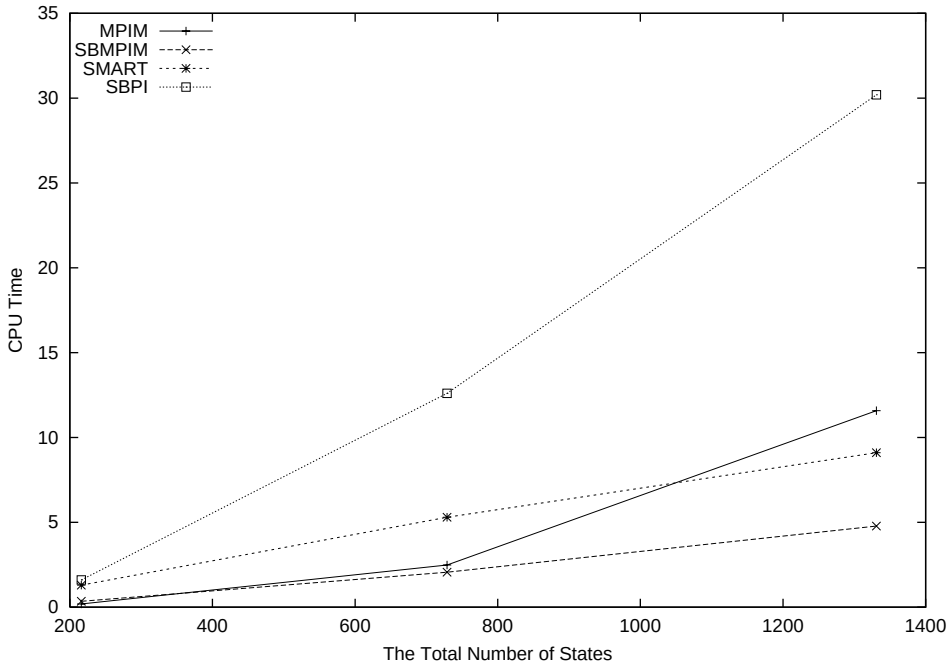

図 2: 各 NDP アルゴリズムにおける状態数と計算時間の関係

り，最小負荷サーバ割り当て方式がシステム全体の平均レスポンスタイムを最小化することが示されている．以下のシミュレーション比較では，単位時間あたりの平均費用について，確率的割り当て方式，ラウンドロビン割り当て方式，最小負荷サーバ割り当て方式の 3 方式と SBMPIM による負荷分散政策との比較を行なう．ただし，本論文の負荷分散システムは各サーバが有限容量であることを仮定しているため，各サーバのバッファ容量を超えるジョブ転送は一切行なわないものとする．

以下各方式に対し，時点 t でスイッチ内に $X_0(t)$ のジョブがあった場合における，各サーバへのジョブ転送数 $K_i(t)$ $(i \in \mathcal{M}_s)$ を与える手順を示す．

5.2.1. 確率的割り当て方式

確率的割り当て方式は，スイッチ内の各ジョブを確率的に各サーバへ割り当てる．各サーバの処理能力 $E[S_i(t)]$ を考慮し，任意のジョブがサーバ i へ転送される確率 P_i を

$$P_i = \frac{E[S_i(t)]}{\sum_{i \in \mathcal{M}_s} E[S_i(t)]} \tag{5.3}$$

で与える．確率的割り当て方式の手順を以下に示す．

確率的割り当て方式

Step 1: $H = X_0(t)$，$K_i(t) = 0$ $(i \in \mathcal{M}_s)$ とする．

Step 2: もし，$H = 0$ あるいは $X_i(t) + K_i(t) = B_i$ $(i \in \mathcal{M}_s)$ ならば，**Step 4** へ．

Step 3: 確率 P_j で j への送信を決定する．もし，$X_j(t) + K_j(t) < B_j$ ならば，$K_j(t) = K_j(t) + 1$，$H = H - 1$ とおき，**Step 2** へ戻る．

Step 4: 以上で得られた $\boldsymbol{K}(t) = (K_i(t);\ i \in \mathcal{M}_s)$ が時点 t での決定となる．

表 2: 1) の各アルゴリズムで得られた政策の比較

$\boldsymbol{x}$	MPIM	SMART		SBPI		SBMPIM	
(2,0,0)	(1,1)	(1,1)	√	(1,1)	√	(1,1)	√
(2,0,1)	(2,0)	(1,1)		(2,0)	√	(2,0)	√
(2,0,2)	(2,0)	(2,0)	√	(2,0)	√	(2,0)	√
(2,0,3)	(2,0)	(0,2)		(2,0)	√	(2,0)	√
(2,0,4)	(2,0)	(2,0)	√	(2,0)	√	(2,0)	√
(2,0,5)	(2,0)	(2,0)	√	(2,0)	√	(2,0)	√
(2,1,0)	(1,1)	(0,2)		(1,1)	√	(1,1)	√
(2,1,1)	(2,0)	(0,2)		(2,0)	√	(2,0)	√
(2,1,2)	(2,0)	(0,0)		(2,0)	√	(2,0)	√
(2,1,3)	(2,0)	(0,2)		(2,0)	√	(2,0)	√
(2,1,4)	(2,0)	(2,0)	√	(2,0)	√	(2,0)	√
(2,1,5)	(2,0)	(2,0)	√	(2,0)	√	(2,0)	√
(2,2,0)	(1,1)	(0,2)		(1,1)	√	(1,1)	√
(2,2,1)	(1,1)	(0,0)		(1,1)	√	(1,1)	√
(2,2,2)	(2,0)	(0,2)		(2,0)	√	(2,0)	√
(2,2,3)	(2,0)	(2,0)	√	(2,0)	√	(2,0)	√
(2,2,4)	(2,0)	(2,0)	√	(2,0)	√	(2,0)	√
(2,2,5)	(2,0)	(0,0)		(2,0)	√	(2,0)	√
(2,3,0)	(0,2)	(2,0)		(0,2)	√	(0,2)	√
(2,3,1)	(1,1)	(1,1)	√	(1,1)	√	(1,1)	√
(2,3,2)	(1,1)	(2,0)		(2,0)		(1,1)	√
(2,3,3)	(2,0)	(1,1)		(2,0)	√	(2,0)	√
(2,3,4)	(2,0)	(1,1)		(2,0)	√	(2,0)	√
(2,3,5)	(2,0)	(2,0)	√	(2,0)	√	(2,0)	√

5.2.2. ラウンドロビン割り当て方式

ラウンドロビン割り当て方式は，スイッチ内のジョブを各サーバへ巡回的に，すなわち，サーバ 1 から始まり $2, 3, \ldots$ と順番にジョブを各サーバへ 1 つづつ転送する．ただし，サーバ n へ割り当てた後は，再びサーバ 1 へ戻り同じ動作を繰り返す．また，時点 t で最後にジョブが割り当てられたサーバが i ならば，時点 $t+1$ での割り当てはサーバ $i+1$（$i=n$ ならばサーバ 1）から始まる．ラウンドロビン割り当て方式は，サーバが同一性能でかつバッファが無限の場合，レスポンスの点で確率的割り当て方式よりよい性能を与えることが報告されている．ラウンドロビン割り当て方式の手順を以下に示す．

ラウンドロビン割り当て方式

Step 1: $H = X_0(t)$，$K_i(t) = 0$ $(i \in \mathcal{M}_s)$ とする．もし，$t = 0$ ならば $j = 1$ とし，さもなければ j は $t-1$ でのアルゴリズム終了時点での値をとる．

Step 2: もし，$H = 0$ ならば，**Step 4** へ．

Step 3: $K_j(t) = K_j(t) + 1$，$H = H - 1$ とする．$j = j + 1$ とし，もし，$j = n + 1$ ならば $j = 1$ とおき，**Step 2** へ戻る．

Step 4: 以上で得られた $\boldsymbol{K}(t) = (K_i(t);\ i \in \mathcal{M}_s)$ が時点 t での決定となる．

5.2.3. 最小負荷サーバ割り当て方式

最小負荷サーバ割り当て方式は，システムの状態を把握し，負荷の最も小さいサーバへジョブを優先的に割り当てる．まず，時点 t でのサーバ i の負荷量 L_i を次のように定義する．

$$L_i = \frac{X_i(t)}{E[S_i(t)]},\ i \in \mathcal{M}_s \tag{5.4}$$

明らかに (5.4) 式は，$X_i(t)$ 個のジョブが処理完了までに要する平均時間である．また，転送数 $K_i(t)$ を含んだ場合の負荷量を L_i' とすると

$$L_i' = \frac{X_i(t) + K_i(t)}{E[S_i(t)]},\ i \in \mathcal{M}_s \tag{5.5}$$

で与えられる．すなわち，この方式では，スイッチ内のジョブを (5.5) 式の右辺が最小になるよう 1 つづつ割り当てる．最小負荷サーバ割り当て方式の手順を以下に示す．

最小負荷サーバ割り当て方式

Step 1(初期設定): $H = X_0(t)$，$K_i(t) = 0$ $(i \in \mathcal{M}_s)$ とする．

Step 2(アルゴリズム停止条件) :もし，$H = 0$ あるいは $X_i(t) + K_i(t) = B_i$ $(i \in \mathcal{M}_s)$ ならば，**Step 4** へ．

Step 3(負荷分散処理) :次式により j を求める．

$$j = \arg\min_{j \in \mathcal{M}_s} \{L_j';\ X_j(t) + K_j(t) < B_j\}$$

とし，$K_j(t) = K_j(t) + 1$，$H = H - 1$ とする．**Step 2** へ戻る．

Step 4(政策の決定) :以上で得られた $\boldsymbol{K}(t) = (K_i(t);\ i \in \mathcal{M}_s)$ が時点 t での決定となる．

5.2.4. シミュレーションによる比較

以上で示した従来の負荷分散方式と SBMPIM による負荷分散方式との比較をシミュレーションを用いて行なう．まず，費用は前節での実験と同様に (2.9) 式を用いる．また，スイッチへの到着数分布として (5.1) 式を用い（すなわち，平均到着数は $Z_0 = (1-\lambda)/\lambda$ で与えられる.），処理可能数分布として (5.2) 式を用いる．

すべての実験において，SBMPIM のパラメータは $\eta = 0.9$，停止回数 40，シミュレーション回数 $m = 10000$ とした．また，シミュレーション結果からの各負荷分散方式の比較は NM+MCB[6] を用いて，費用差の 95%信頼区間を求めることによって行なった．また，負荷分散システム全体におけるトラフィック率 ρ を

$$\rho = \frac{E[A]}{\sum_{i \in \mathcal{M}_s} E[S_i]} \tag{5.6}$$

と定義する．

システムのパラメータを次のように設定する：$n = 4$，$B_0 = 10$，$B_i = 2$，$(i \in \mathcal{M}_s)$，$\mu_i = 0.5$ $(i \in \mathcal{M}_s)$，$C_i^H = 3$ $(i \in \mathcal{M})$，$C^R = 5$，$C_i^P = 1$ $(i \in \mathcal{M}_s)$．このときのシステム全体のトラフィック率 ρ を 0.1 から 1.0 まで変化させたときの費用差のグラフを図 3 に示す．ここで，方式 i の費用が Y_i で与えられるとき，その費用差は，$Y_i - \min_{j \neq i} Y_j$ により与えられる．図 3 において，Random は確率的割り当て方式，RoundRobin はラウンドロビン割り当

て方式，Shortest は最小負荷サーバ割り当て方式を表している．ここで，ρ の各値に対応する到着数分布のパラメータ λ はそれぞれ，0.714，0.556，0.455，0.385，0.333，0.294，0.263，0.238，0.217，0.200 である．図 3 より，トラフィックが飽和に近づくにつれ，SBMPIM による負荷分散方式の優位性が明らかになっている．これは，SBMPIM では多少のジョブのリジェクトを許すことにより，全体での費用の低減を行なっていることが原因であると考えられる．

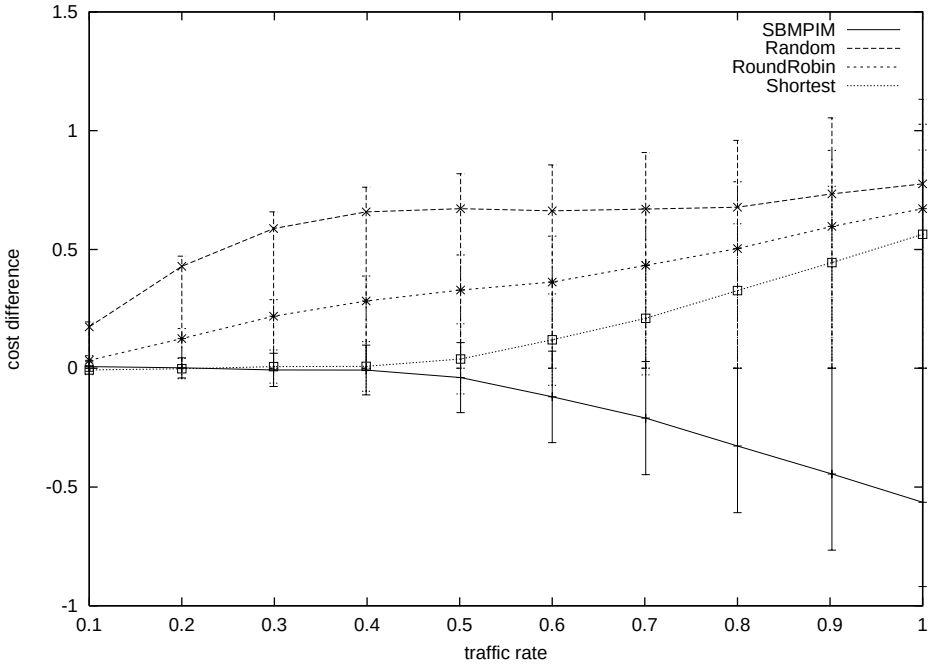

図 3: 各負荷分散方式におけるトラフィック率と費用差の関係

6. おわりに

本論文では，一定周期ごとにシステム情報の収集を行なう動的な負荷分散システムに対し，その最適負荷分散政策を求める問題を時間平均マルコフ決定過程 (UMDP) として定式化し，ニューロダイナミックプログラミング (NDP) を用いたアルゴリズムにより準最適な負荷分散政策を導き，その評価を行なった．まず，比較的小規模な問題に対して修正政策反復法 (MPIM) と，Semi-Markov Average Reward Technique (SMART)，Simulation-Based Policy Iteration Algorithm (SBPI)，Simulation-Based modified Policy Iteration Method (SBMPIM) といった 4 つの NDP アルゴリズムとの比較を行ない，最も適した NDP アルゴリズムを選定した．結果としては，NDP アルゴリズムの中では Simulation-Based Modified Policy Iteration Method(SBMPIM) がすべての面において最もよい性能を与えることが示された．さらに，各サーバが同一性能である場合，異なる性能の場合において，SBMPIM と確率的割り当て方式，ラウンドロビン割り当て方式，最小負荷サーバ割り当て方式の従来の負荷分散政策との比較を行ない，飽和に近いトラフィックにおける SBMPIM の優位性を示した．

今後の課題として，複数種類のジョブを扱う負荷分散システムや連続的に観測を行なう負荷分散システムにおける最適負荷分散問題など，より現実的な問題に有効な NDP アルゴリズムの開発が残されている．

参考文献

[1] R.E. Bellman: *Dynamic Programming* (Princeton University Press, Princeton, 1957).

[2] D.P. Bertsekas, and J.N. Tsitsiklis: *Neuro-Dynamic Programming* (Athena Scientific, Belmont, 1996).

[3] M. Dahlin: Interpreting stale load information. *IEEE Transactions on Parallel and Distributed Systems*, **11-10** (2000), 1033-1047.

[4] T.K. Das, A. Gosavi, S. Mahadevan, and N. Marchalleck: Solving semi-markov decision problem using average reward reinforcement learning. *Management Science*, **45** (1999), 560-574.

[5] D. Eager, E. Lazowska, and J. Zahorjan: Adaptive load sharing in homogeneous distributed systems. *IEEE Transactions on Software Engineering*, **12-5** (1986), 662-675.

[6] D. Goldsman, and B.L. Nelson: Computing systems via Simulation. In J. Banks (ed.): *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice* (John Wiley & Sons, New York, 1998), 273-306.

[7] Y. He, M.C. Fu, and S.I. Marcus: A simulation-based policy iteration algorithm for average cost unichain Markov decision processes. In M. Laguna and J. L. G. Velarge (eds.): *Computer Tools for Modeling, Optimization and Simulation* (Kluwer Academic, Boston, 2000), 161-182.

[8] R. Howard: *Dynamic Programming and Markov Processes* (MIT Press, Cambridge, 1960).

[9] H. Kameda, J. Li, C. Kim, and Y. Zhang: *Optimal Load Balancing in Distributed Computer Systems* (Springer, London, 1997).

[10] J. Li, H. Kameda: Optimal load balancing problems for multiclass jobs in distributed/parallel computer systems. *IEEE Transactions on Computers*, **47-3** (1998), 322-332.

[11] R. Mirchandaney, D. Towsley, and J. Stankovic: Analysis of the effects of delays on load sharing. *IEEE Transactions on Computers*, **38-11** (1989), 1513-1525.

[12] R. Mirchandaney, D. Towsley, and J. Stankovic: Adaptive load sharing in heterogeneous distributed systems. *Journal of Parallel and Distributed Computing*, **9** (1990), 331-346.

[13] M. Mitzenmacher: How useful is old information. *IEEE Transactions on Parallel and Distributed Systems*, **11-1** (2000), 1033-1047.

[14] K. Ohno: Modified policy iteration algorithm with nonoptimality tests for undiscounted Markov decision process. Working Paper, Dept. of Information System and Management Science, Konan University (1985).

[15] 大野　勝久, 八嶋　憲司, 伊藤　崇博: ニューロ・ダイナミックプログラミングによる生産ラインの最適制御に関する研究, 日本経営工学会論文誌, **54-5** (2003), 316-325.

[16] M.L. Puterman: *Markov Decision Processes* (John Wiley & Sons, New York, 1994).

[17] N.G. Shivaratri, P. Krueger, and M. Shinghal: Load distributing for locally distributed systems. Computer, (1992), 33-43.

[18] R.S. Sutton and A.G. Barto: *Reinforcement Learning* (MIT Press, Cambridge, 1998)

[19] A.N. Tantawi and D. Towsley: Optimal static load balancing in distributed computer systems. *Journal of the ACM*, **32-2** (1985), 445-465.

[20] S. Zhou: A trace-driven simulation study of dynamic load balancing. *IEEE Transactions on Software Engineering*, **14-9** (1988), 1327-1341.

井家　敦
神奈川工科大学情報学部情報ネットワーク工学科
〒 243-0292 神奈川県厚木市 2-4-16 下荻野 1030
E-mail: inoie@nw.kanagawa-it.ac.jp

ABSTRACT

A DISCRETE-TIME LOAD BALANCING PROBLEM BY NEURO-DYNAMIC PROGRAMMING ALGORITHMS

Atsushi Inoie
Kanagawa Institute of Technology

Katsuhisa Ohno
Aichi Institute of Technology

The meaning of load balancing is to dispatch jobs among resources of a system for maximizing the system performance. This paper deals with a discrete-time optimal load balancing problem that minimizes an expected total cost. This problem is formulated as an undiscounted Markov decision process, and is solved by the modified policy iteration method. Since the modified policy iteration method can not solve practical sized problems due to the curse of dimensionality, a near-optimal load balancing policy is computed by neuro-dynamic programming algorithms. We further compare this policy with heuristic policies such as the random policy, round-robin policy and shortest policy.