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

2003－MPS－46　(9)
2003／9／19

強化学習による UNIX ワークステーションのメモリエージングに対する保全アルゴリズム

岡村 寛之[1]　　中後 智[2]　　土肥 正[1]

概要： UNIX ワークステーションにおいてプロセスのリスタートはメモリリークなどのエージング現象に対して，高い効果があることが知られている．そこで本論文では，メモリエージングを回避するために強化学習を用いたリスタート制御アルゴリズムを提案する．また，提案したアルゴリズムを実システムを想定したシミュレーション環境上で実行することでその有効性を検証する．

A Maintenance Algorithm for the Aging of Memory in UNIX Workstations Based on the Reinforcement Learning

Hiroyuki Okamura,[1]　Satoru Nakago[2]　and　Tadashi Dohi[1]

Abstract: Restarting processes take advantage of the aging of memory which is caused by the memory leaking in UNIX workstations. In this paper, we propose an algorithm based on the reinforcement learning to prevent the aging of memory, and investigate the effectiveness of the algorithm by the simulation which is assumed to be a real UNIX workstation.

1　はじめに

コンピュータシステム上でアプリケーションを稼働する場合，正常動作しているように見えても，稼働時間が長期にわたると突然アプリケーションがハングアップすることがある．そのような問題が発生する原因の１つとしてメモリリークなどのメモリエージング現象が考えられる．メモリエージングとはソフトウェアにおける経年劣化現象の一つである．より大きな枠組みではソフトウェアエージングと呼ばれ，オペレーティングシステム, ミドルウェア, 通信アプリケーションの動作時において観測されることが報告されている [1, 2, 3].

一般的にソフトウェアエージングによる障害は一過性の障害 (transient failure) であることが多い [4]. すなわち，障害が発生した後，若干異なる内容（データ，環境）でシステムをリトライすることにより，あたかも障害が発生していなかったかのような操作可能な状況に復帰する．しかしながら，このような一過性の障害は，ソフトウェアのソースコード上で障害の原因を特定することが極めて困難である．そこで，ソフトウェアエージングによる障害を回避するため，周期的にシステムの稼働を一時的に停止・再稼働行うという一連の予防保全手続きが経験的に行われてきた．

特に，エージングの原因が主にメモリリークなどのメモリエージングによって生じる場合，定期的にシステムあるいはプロセスを停止し，メモリを解放およびシステムの動作環境を多様化（environment diversity）による若化が有効である．Castelli ら [3] は，UNIX 上の Apache[3]が長時間にわたって稼動することに伴ってリソース消費の増大が発生する現象に着目して，Apache に関するプロセスをリスタート（再起動）するソフトウェア若化方策がリソースの消費増大に対して高い効果があることを報告している．このような，若化と呼ばれるソフトウェアシステムの予防保全は，N バージョンプログラミングやリカバリブロックのような設計多様化（design diversity）思想に基づいたソフトウェア耐故障技術と比較して非常に安価であり，最も一般的かつ有効なソフトウェア保全技術のひとつである．

ここで問題は，どのようなタイミングでシステム

[1]広島大学大学院工学研究科，Graduate School of Engineering, Hiroshima University
[2]日進ソフトウェア株式会社，Nissin Software Corp.

[3]「The Apache Software Foundation」http://www.apache.org/

を予防的に若化するかにある．Huang ら [5] は，電話料金勘定システムに対するソフトウェア若化を行うための指針を与える目的で，定期的なソフトウェア若化スケジュールを決定するための確率モデルを提案している．すなわち，システムの時間的挙動を，正常稼働状態，障害が発生可能な状態，障害の発生状態，ソフトウェア若化状態の 4 状態をもつ連続時間マルコフ連鎖で記述し，ランダムな若化スケジュールの下でシステムのアンアベイラビィティや定常状態における運用期待費用を評価している．Dohi ら [6] は Huang ら [5] のモデルをセミマルコフモデルに拡張し，さらに障害発生時間データから直接最適な若化スケジュールを推定する統計的手法を開発している．しかしながら，いずれの方法においても最適なタイミングを導出するためにあらかじめ多くの障害データを準備する必要があり，実用に際しては試験的な運用による障害データの採取作業が必要となる．

本論文では，強化学習 [7] を取り入れることで運用環境に対して適応的に最適な若化のタイミングを学習するモデルを提案する．特に UNIX ワークステーションにおけるメモリエージング現象に着目し，従来のソフトウェア若化モデルの修正および強化学習に基づいたリスタート制御アルゴリズムの構築を行う．さらに，提案アルゴリズムの有効性をシミュレーション環境上で検証する．

2 強化学習に基づいた若化方策

2.1 若化モデル

本論文では，UNIX 上のプロセスに対してリスタート制御をするときのダイナミクスをモデル化することを考える．Dohi ら [6] はシステムを正常状態，劣化状態，故障状態，保全状態の 4 つに分類したセミマルコフモデルを取り扱っている．しかしながら，いま対象とする slapd プロセスのメモリエージングに対する若化を考えた場合，観測可能な量（CPU 占有率やメモリの消費量）から slapd プロセスが正常状態なのか劣化状態なのかを判断することは容易ではない．つまり，プロセス稼動状況を正常状態，劣化状態の 2 つに分類することは非現実的である．それよりも，プロセスの CPU 占有率やメモリの消費量などの計測可能な情報によってプロセスの状態を分類することが重要であると考えられる．また障害の概念においても再考が必要である．一般的に UNIX におけるプロセスの障害はリスタートによって復旧する．つまり，障害が発生したとしても通常のリソース消費を回避するためのリスタート制御と同じ操作で復旧が可能であるため，保全状態と故障状態を区別することは不可能である．

そこで本論文では，プロセスの状態を多数の稼動状態と一つの非稼動状態に分類し，多数の稼動状態それぞれにおいてプロセスリスタートの実施制御をマルコフ決定過程によってモデル化する．(図 1 参照)

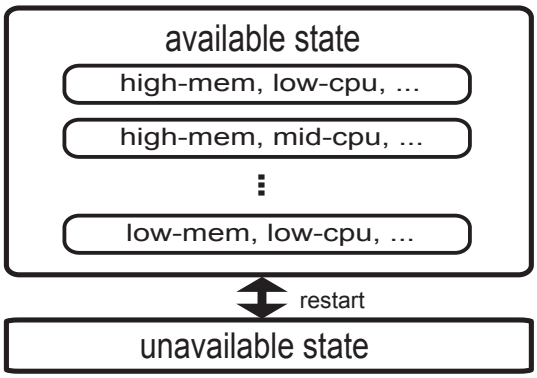

図 1: プロセスのリスタート制御の概念図.

2.2 強化学習によるリスタート制御アルゴリズム

強化学習とは，エージェント（学習と意思決定を行う者）が未知な環境との相互作用から学習して目標を達成する学習過程の枠組みである．エージェントは行動を選択し，環境はその行動に応答し，エージェントに新しい状況を提示する．また環境は報酬の発生源でもあり，エージェントの目標は最終的に受け取る報酬の総量を最大化することである．ここで報酬とはエージェントが時間の経過の中で最大化しようとする目標パラメータである．具体的に本論文では前述の環境，報酬，行動を以下のように定める．

環境： slapd プロセスのメモリ消費量を Low=〜20000byte, Middle=20000〜25000byte, High=25000byte〜の 3 つに分割し，その時系列中の二系列でプロセスの状態を記述したときの 9 つの状態を稼動状態とする．そのときのプロセスの 9 つの状態を環境とする．

報酬： それぞれの時間区分におけるシステム全体のメモリ消費量の平均値を 1000 分の 1 にした

ものとする．本論文では，システム全体のメモリ消費量の最小化を目標とするので報酬を負の評価とし，それを最大化することを考える．そのとき報酬は以下の式で与えられる．

$$\Delta p = -\frac{\mathrm{mem}_{t-\Delta t} + \mathrm{mem}_t}{2} * \frac{1}{1000} \qquad (1)$$

ただし，$\mathrm{mem}_{t-\Delta t}$ は時間 Δt 経過前のメモリ全体の消費量，mem_t は現在のメモリ全体の消費量とする．

行動：プロセスのリスタートを行う/行わない．

先に述べた若化モデルに対して，上記の「環境・報酬・行動」をもつ強化学習を適用することで，全体のメモリ消費量を最小にする最適なリスタート制御を学習することになる．

また，強化学習は行動に対する学習方法の違いから様々な種類がある．本論文では，行動の選択および学習に Q 値と呼ばれる推定値を用いる Q 学習 [7] と呼ばれる手法を適用する．Q 学習は強化学習の代表的な手法であり，観測状態，行動選択および学習の 3 つによって構成される．

状態観測：現在のプロセスの状態を観測する．

行動選択：現在の状態からある行動を選択し，それ以後最適な行動を選択するときの報酬 Δp の推定値 (Q 値) に基づいて現在の行動を選択する．本論文では行動選択法として以下の ε-greedy を用いる．

ε-greedy：ε の確率でランダムに行動を選択する．それ以外では Q 値が最小の行動を選択する．

学習：選択した行動により発生した報酬と現在の Q 値を用いて，Q 値を更新する．

最終的に Q 学習を適用することで，プロセスの状況に対して適応的に最適な制御を学習するアルゴリズムを次のように構築することができる．

Step1 時刻 t におけるプロセスの状態 s_t を観測する．

Step2 確率 ε でランダムなリスタート制御を行う．また確率 $1-\varepsilon$ で最大の Q 値を示す制御を行う（ここで選択した行動を a_t とする）．

Step3 一定時間 Δt 経過後，Δt 期間中における報酬 Δp とプロセスの状態 $s_{t+\Delta t}$ を計測する．

Step4 以下の更新式により Q 値を更新する．

$$\begin{aligned} Q(s_t, a_t) \leftarrow\ & (1-\alpha)Q(s_t, a_t) \\ & + \alpha[\Delta p + \gamma \max_a Q(s_{t+\Delta t}, a)]. \end{aligned} \qquad (2)$$

ただし α は学習率 $(0 < \alpha \leq 1)$，γ は割引率 $(0 \leq \gamma < 1)$ である．

Step5 $t \leftarrow t + \Delta t$ として Step2 へ戻る．

3　シミュレーションによる有効性検証

提案したリスタート制御アルゴリズムの有効性を示すためにシミュレーションによる検証を行う．特に，slapd プロセスのメモリ消費量とシステム全体のメモリ消費量を 2 変量自己回帰モデルでモデル化し，slapd プロセスに対するリスタート制御アルゴリズムを適用した時の有効性をシミュレーション上で検証する．

リスタート制御アルゴリズムのパラメータを学習率 $\alpha = 0.3$，割引率 $\gamma = 0.99$，$\varepsilon = 0.1$ として，構築したシミュレーション環境で適用した．ここに示す結果は，全て 10 回の試行の平均となっていることに注意する．

図 2 はリスタートをしないときとリスタート制御アルゴリズムを適用したときのシステム全体のメモリ消費量のシミュレーション結果を示している．この結果より，リスタートを全くしないよりもリスタート制御アルゴリズムを適用したときのほうがメモリ消費に関するパフォーマンスが向上していることが分かる．また，定期的にプロセスをリスタートをし

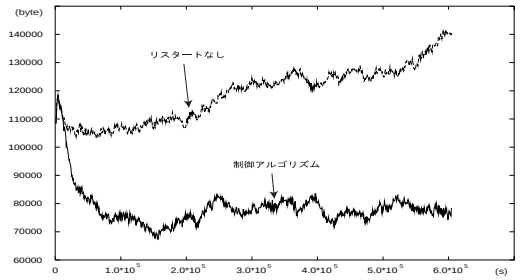

図 2: システム全体のメモリ消費量のシミュレーション結果.（リスタートなしとリスタート制御アルゴリズム）

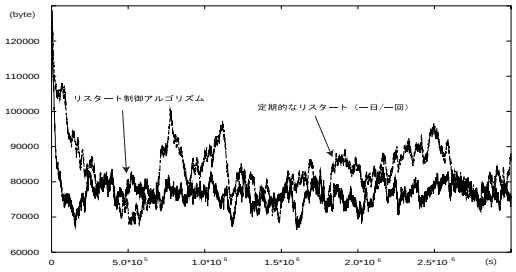

図 3: システム全体のメモリ消費量のシミュレーション結果.（リスタート制御アルゴリズムと定期的なプロセスのリスタート）

たときのメモリ消費量に関して考察する．図 3 はリスタート制御アルゴリズムを適用したときと，1 日に 1 回定期的にプロセスのリスタートをしたときのメモリ消費量のシミュレーション結果を示している．この結果から，システム全体のメモリ消費量に関して，1 日 1 回プロセスをリスタートしたときの方が明らかにメモリ消費が大きい区間が多数存在する．つまり，メモリ消費量の低減を維持するという観点から提案アルゴリズムの方が有効であると考えられる．

4 むすび

本論文では，UNIX ワークステーション上のメモリエージング現象に着目し，それを解消するためのリスタート制御アルゴリズムを構築した．提案したアルゴリズムは強化学習に基づいており，利用状況に応じて適応的に最適なリスタート制御を学習する．さらに，自己回帰モデルによるシミュレーション環境を構築し，提案アルゴリズムの有効性について検証した．その結果として他の保全方策（リスタートをしないモデル，一日に一回リスタートをするモデル）に対して提案アルゴリズムが有効であることを示した．

今後の課題としては，システムのアベイラビリティを考慮に入れたリスタート制御をすることが挙げられる．

謝辞：本研究の一部は文部省科学研究費若手研究 (B) Grant No. 15700060 (2003–2004), 萌芽研究 15651076 (2003-2005) および基盤研究 (B) Grant No. 13480109 (2001–2004) による助成の下で行われたものである．

参考文献

[1] E. Adams, “Optimizing preventive service of the software products”, *IBM Journal of Research & Development*, vol. 28, pp. 2–14 (1984).

[2] D. L. Parnas, “Software aging”, *Proc.* 16^{th} *Int’l Conf. on Software Engineering*, pp. 279–287, ACM Press, New York, NY (1994).

[3] V. Castelli, R. E. Harper, P. Heidelberger, S. W. Hunter, K. S. Trivedi, V. Vaidyanathan and W. P. Zeggert, “Proactive management of software aging”, *IBM J. Research & Development*, vol. 45, pp. 11–332 (2001).

[4] J. Gray and D. P. Siewiorek, “High-availability computer systems”, *IEEE Computer*, vol. 9, pp. 39–48 (1991).

[5] Y. Huang, C. Kintala, N. Kolettis and N. D. Funton, “Software rejuvenation: analysis, module and applications”, *Proc.* 25^{th} *IEEE Int’l Symp. on Fault Tolerant Computing*, pp. 381–390, IEEE Computer Society Press, Los Alamitos, CA (1995).

A. Avritzer and E. J. Weyuker, “Monitoring smoothly degrading systems for increased dependability”, *Empirical Software Engineering*, vol. 2, pp. 159–77 (1997).

[6] T. Dohi, K. Goševa-Popstojanova and K. S. Trivedi, “Estimating software rejuvenation schedule in high assurance systems”, *Computer Journal*, vol. 44, pp. 473–485 (2001).

[7] R. Sutton and A. Barto, *Reinforcement Learning - An Introduction*, MIT Press (1998).