A-036

“若返り”を用いた世代別 GC の改良
Improvement of Generational Garbage Collection by "Reinstatement."

深井 優一† 安井 浩之† 吉野 邦生†
Yuichi Fukai Hiroyuki Yasui Kunio Yoshino

1. まえがき

現在使われているプログラミング言語の多くにはメモリ管理機構としてガベージコレクション（GC：Garbage Collection）[1]が搭載されている．しかし，GC はプログラム本来の処理を停止して行うため，リアルタイム性の高いプログラムでは影響が大きい．

世代別 GC[1]は，生まれたばかりのオブジェクトだけを対象とするマイナーGC を行うことで GC 処理時間を短縮する手法である．しかし，メモリ領域が不足した場合に実行される全オブジェクトを対象としたメジャーGC が発生した場合，処理に長い時間を要する．

そこで，長寿命オブジェクトの“若返り”を行い，メジャーGC の発生の抑制を行う．本研究ではメモリの物理的移動を伴わない保守的 GC を採用する処理系に実装が可能な Treadmill GC[2]上で世代別 GC を行い，マイナーGC 毎に長寿命オブジェクトを新世代領域に移動することで，旧世代領域の肥大化，およびメジャーGC の発生を抑制する．本稿ではシミュレータ実験により本手法の有効性を示す．

2. 関連研究

Treadmill GC は，オブジェクトを双方向環状リストで管理する GC 手法である．リンクの付け替えを行うことで，メモリの移動を行うことなく Copying GC の動作を行うことができるメリットがある．Treadmill GC に世代別 GC の考えを導入した Opportunistic Treadmill GC[3]では，世代ごとにマイナーループ，メジャーループそれぞれ 2 種類の Treadmill に分けることで世代別 GC を実現している．メジャーGC を行う場合はメジャーループをマイナーループに繋ぐことで，Treadmill GC と同様の状態にして行う．ただし，マイナーGC を行う場合は処理時間が短くなることが保証されているが，メジャーGC を行った場合は Treadmill GC と同じ処理時間がかかる問題がある．

その問題を解決するため，Caterpillar GC[4]は，Opportunistic Treadmill GC のメジャーループを複数に分割し，1 度のメジャーGC で対象とするオブジェクトを減らすことで，メジャーGC にかかる処理時間を抑えるようにした手法である．

3. 提案手法

世代別 GC におけるメジャーGC の処理時間の問題を解決するため，提案手法では Treadmill GC をベースとした長寿命オブジェクトの“若返り”を行う世代別 GC を提案する．メジャーGC は，マイナーGC の対象外である旧世代領域に不要オブジェクトが残留していくことによるメモリ不足が原因で発生する．提案手法ではマイナーGC の際に旧世代領域のオブジェクトを少量ずつ新世代領域に移動させることで，旧世代領域の不要オブジェクトをマイナーGC 中にも解放できるようにする．

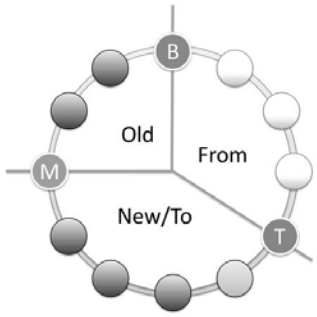

図 1　提案手法の Treadmill

提案手法では世代別 GC を実現するために，旧世代領域（Old 領域）を Treadmill 内に定義する（図 1）．

マイナーGC は以下の手順に従って行う．

1. Root および Old 領域からの参照確認
   システムの Root および Old 領域から参照されている From 領域のオブジェクトを調べ，対象が一定回数の GC を行われたことがある場合は Old 領域，そうでない場合は To 領域に移動する．
2. To 領域の走査
   To 内のオブジェクトから参照されている From 領域のオブジェクトを調べ，1.同様に移動を行う．
3. From 領域の解放
   2.までの処理を完了した時点で From 領域のオブジェクトは不要オブジェクトであることが確定するので，From 領域内のオブジェクトを解放する．
4. 長寿命オブジェクトの“若返り”
   Old 領域の先頭から少数のオブジェクトを New/To 領域に移動し，次回のマイナーGC の対象とする．
5. New/To 領域を From 領域へ移動
   次回のマイナーGC の準備として，New/To 領域のオブジェクトを From 領域とする．

以上の手順がマイナーGC の 1 サイクルとなる．メジャーGC を行う場合は 4.の“若返り”を行う対象を Old 領域全てのオブジェクトにすることで，通常の Treadmill GC 同様の全体を対象とするメジャーGC を実現できる．

“若返り”を行うことにより，旧世代領域の不要オブジェクトをマイナーGC 中に回収できるため，全体的なメモリ消費量の軽減が期待でき，メジャーGC の発生を考えられる．しかし，本来処理対象ではない旧世代領域をマイナーGC の対象とすることにより，世代別 GC に比べマイナーGC の処理負荷が大きくなってしまうことが考えられる．これについては，どの程度の量のオブジェクトを“若返り”させるか，適切に設定する必要がある．

† 東京都市大学，Tokyo City University.

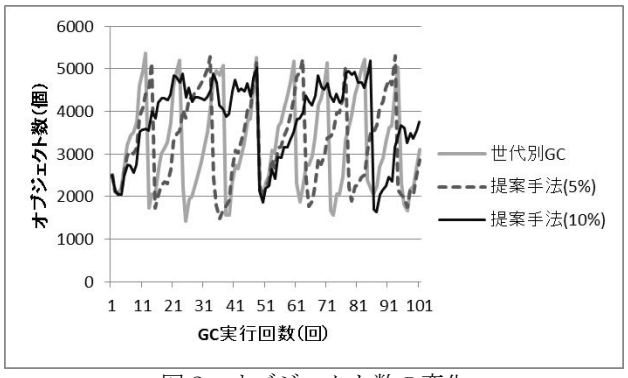

図 2　オブジェクト数の変化

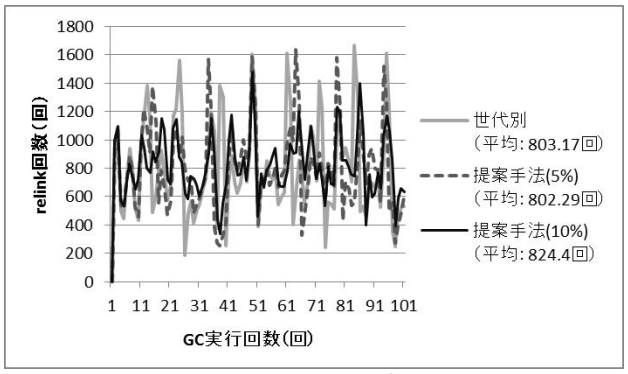

図 3　relink 回数

4. 予備実験

提案手法の有用性を調べるため， GC 動作シミュレータを作成し，世代別 GC と提案手法を比較する予備実験を行った．シミュレータの実装には CoffeeScript[5]を用いた．

4.1 実験方法

本実験では，初期オブジェクト数 2500 個，GC サイクル毎に平均 500 個の新オブジェクトを発生させるよう設定し，GC サイクル 100 回分のデータ収集を行った．オブジェクト間の参照関係はオブジェクトが新しく追加される時に変更されるものとし，既存のオブジェクトから新オブジェクトに対してランダムに設定した．

世代別 GC のパラメータは，旧世代領域に昇格させるまでの GC 実行回数（Advancement Threshold）を 1 回，メジャーGC の発生はヒープ領域にオブジェクトが 5000 個を超えると発生するように設定した．また，本手法の“若返り”させる長寿命オブジェクトは Old 領域の 5%および 10%を対象とし，それぞれ計測を行った．

4.2 実験結果

シミュレータにより，オブジェクト数の変化と GC 処理にかかる負荷量を測定した．

オブジェクト数の変化（図 2）をみると，世代別 GC はオブジェクト数が頻繁に 5000 を超え，メジャーGC の実行が必要になるのに対し，提案手法ではオブジェクト数の増加が世代別 GC に比べ，“若返り”量を 5%に設定した場合 3/4 程度，10%に設定した場合は 1/3 から 1/4 程度になり，メジャーGC の発生が抑制されていることが確認できた．オブジェクト数が増えるに従い増加量がゆるやかになるのは，“若返り”を行うオブジェクトの数を旧世代領域の割合で設定したため，全体に対して旧世代領域の割合が大きくなるに従い，“若返り”対象となるオブジェクトが多くなり，より多くの不要オブジェクトを回収できていると考えられる．

GC 処理にかかる負荷については，Treadmill GC は双方向リストの繋ぎ替え（relink）を行う GC であるため，GC 処理にかかる負荷量は relink 回数に比例するといえる．世代別 GC と提案手法の relink 回数を比較（図 3）すると，提案手法では，メジャーGC の発生が少ない分，振れ幅が小さくなっている．平均的な relink 回数を見ると，5%の場合では大きな変化は無いが，10%の場合では，平均的な relink 回数が世代別 GC より多くなっていることがわかる．これは前述の通り，“若返り”により，本来マイナーGC で対象となっていない旧世代領域のオブジェクトがマイナーGC の対象となることで，処理対象が増えてしまったことが原因であると考えられる．

5. 考察

予備実験の結果より，世代別 GC に対し，長寿命オブジェクトの“若返り”が，メジャーGC の発生抑制に有用であることがわかった．しかし，通常の世代別 GC に比べ，マイナーGC の平均的な処理負荷が増加しているため，なんらかの対策を行う必要がある．今回の予備実験では“若返り”を行う量を一律で設定していたが，空きヒープ量に応じて動的に設定することで，メジャーGC の発生をより抑制でき，またヒープ量に余裕がある場合は“若返り”させる量を少なくすることで，処理負荷を軽減できるのではないかと考えられる．

6. まとめ

本稿では，世代別 GC において長寿命オブジェクトを新世代領域に“若返り”させ，メジャーGC の発生を抑制させる手法を提案した．また，シミュレータ実験により，“若返り”手法の有用性を示した．

今回のシミュレータ実験においては，双方向リンクの繋ぎ替え以外の GC 処理にかかるオーバヘッドを全て無いものとして考え計測を行っている．また実際のプログラムにおける動作に対して有用性があるとは言い切れない．そのため，今後は提案手法をスクリプト言語である Lua[6]に実装を行い，ベンチマークスクリプトを用いて，世代別 GC との比較実験を行う．

参考文献

[1] 中村 成洋，相川 光 著，竹内 郁雄 監修．“ガベージコレクションのアルゴリズムと実装”，秀和システム，2010 年，第 1 版第 2 刷，ISBN978-4-7980-2562-9

[2] Henry G. Baker，”The Treadmill: Real-Time Garbage Collection Without Motion Sickness”，ACM SIGPLAN Notices，Vol.27，No.3，pp.66-70 (1992)

[3] 小池 龍信，岩井 輝男，中西 正和，“オブジェクトの世代を考慮に入れたインクリメンタルなごみ集め処理”，情報処理学会論文誌，Vol.40，No.SIG7，PRO4，pp.1-8 (1999)

[4] 尾沢 崇，矢崎 俊志，阿部 公輝，“Caterpillar GC: 旧世代領域の分割を行うインクリメンタルな世代別実時間ごみ集め”，第 10 回情報科学技術フォーラム，Vol.1，pp.169-172 (2011)

[5] CoffeeScript，http://coffeescript.org/

[6] The Programming Language Lua，http://www.lua.org/