5L-5

“若返り”を用いた世代別 GC の実装と評価

深井　優一　　安井　浩之　　吉野　邦生
東京都市大学

1. まえがき

現在使われている多くのプログラミング言語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）が搭載されている[1]．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．

そこで世代別 GC において長い処理時間を要するメジャーGC の発生を抑制する手法として，“若返り”を用いた世代別 GC を提案した[2]．本報告では提案手法を軽量スクリプト言語である Lua[3]に実装し，その有用性を検証する．

2. 提案手法の概要

世代別 GC では，通常時は旧世代領域を対象外とするマイナーGC のみを行うことで GC 時間を短縮している．しかし，旧世代領域内のオブジェクトが増え，肥大化した場合は旧世代領域を対象に含むメジャーGC を行う必要がある．これはマイナーGC に比べ長い時間を要するため，サーバプログラムや GUI アプリケーションといったリアルタイム性を必要とするプログラムにおけるメジャーGC の発生は好ましくない．

提案手法ではメジャーGC 発生要因である旧世代領域の肥大化を抑えるため，マイナーGC 毎に旧世代領域から少量のオブジェクトを新世代領域に移動する“若返り”処理を行う（図 1）．これにより，旧世代領域の不要オブジェクトをマイナーGC 中に解放できるようになり，マイナーGC の負荷を大きく高めることなく，メジャーGC の発生を抑制できる．

Implementation and Evaluation of Generational Garbage Collection using “Reinstatement.”
Yuichi Fukai, Hiroyuki Yasui, Kunio Yoshino
Tokyo City University

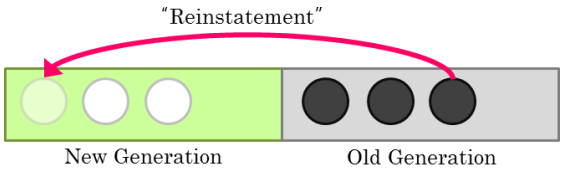

図 1 旧世代オブジェクトの“若返り”

3. 提案手法の実装

Lua はインクリメンタル GC と世代別 GC を搭載している．本実験では Lua の世代別 GC に“若返り”機能を追加し，従来手法と比較する．

Lua ではオブジェクトを単方向リストで管理しており，オブジェクトの色（属性）を塗り替えることで世代別の領域分けを行っている．色は White（未探索），Gray（探索中），Black（探索済み），Old（旧世代）に分類される．

提案手法を実装した Lua のマイナーGC は，

① root からの参照を確認して Gray / Black に塗り替える
② Gray に塗られたオブジェクトからの参照を辿り Black に塗り替える
③ White のオブジェクトを回収，解放する
④ 全てのオブジェクトを White に塗り替える
⑤ リスト末尾から少量のオブジェクトを先頭に移動（若返り）処理を行う
⑥ 若返りさせなかった旧世代領域のオブジェクトからの参照を確認して Gray / Black に塗り替える

の順に行う．なお，メジャーGC については従来手法同様に行う．また，Lua の世代別 GC では処理量が少ないときに自動的にメジャーGC を実行してしまうため，発生条件を，前回のメジャーGC 終了時のメモリ消費量と現在のメモリ消費量を比較して判断するように変更した．

4. 実験・評価

実験では提案手法を実装したLuaを用いてPC上でベンチマーク実験を行い評価した．ベンチマークスクリプトにはAO Bench[4]を用い，ベンチマーク終了までの時間（All Time），合計GC時間(Total GC Time)，メジャーGC回数(Major GC)を5回測定した平均値とした．各時間の計測のため，getrusage関数から取得されるユーザ時間を用いた計測用の処理をLuaに追加した．また，世代別GCのパラメータ設定は，メモリ消費量が前回メジャーGC完了時のメモリ消費量の2倍になったらメジャーGCを行うようにし，若返りさせるオブジェクト数は旧世代領域のオブジェクト数に対する割合(1%刻み)で設定するようにした．実験環境にはIntel Core i3 CPU 550 (3.20GHz)，メモリ8192MB，OSにはUbuntu 12.04 (x64)を使用した．

実験結果から若返り量15%のデータまでを抜粋したものが表1である．メジャーGC回数（Major GC）の結果より，若返りによってメジャーGCの発生回数を抑制できていることが確認できる．また，若返り量9%以降，メジャーGC回数が3回から減少していないが，メジャーGCの発生タイミングを調べると，プログラム開始の序盤に発生していることから，AO Benchを実行するのに最低限確保する必要があるヒープ量に達するまでに3回のメジャーGCを要したためと考えられる．

一方で，合計GC時間（Total GC Time）が既存手法に比べ，増加している．要因としては，若返りによりマイナーGCで処理するオブジェクトの数が増えたことによる処理量の増加と若返りを実装するために増加したrootや旧世代領域の再マーク処理のオーバヘッドが考えられるが，若返り量を0%に設定した場合においてもGC時間が大きく増えていることから後者のオーバヘッドが大きいといえる．なお，若返り量15%以降のデータに大きな変化は見られなかった．

表 1 実験結果

	Total Time (sec.)	Total GC Time (sec.)	Major GC
Generational	92.04	6.73	46.60
Reinst. 0%	100.64	14.47	47.00
Reinst. 1%	96.81	11.60	10.00
Reinst. 2%	93.98	9.71	8.00
Reinst. 3%	93.73	9.04	7.00
Reinst. 4%	92.85	8.76	5.00
Reinst. 5%	92.51	8.59	4.00
Reinst. 6%	92.78	8.76	4.00
Reinst. 7%	93.05	8.28	4.00
Reinst. 8%	91.33	8.24	4.00
Reinst. 9%	91.63	8.11	3.00
Reinst. 10%	92.32	8.38	3.00
Reinst. 11%	92.18	8.38	3.00
Reinst. 12%	92.24	8.43	3.00
Reinst. 13%	91.57	8.33	3.00
Reinst. 14%	91.57	7.98	3.00
Reinst. 15%	92.05	8.55	3.00

5. まとめ

本稿では，若返りを用いた世代別GCをLuaに実装し，ベンチマーク実験によりメジャーGCの抑制に効果があることを示した．

今後の課題としては，GC時間の増加解消が挙げられる．旧世代領域から新世代領域への参照情報をうまく保持できていないために，旧世代領域の再マーク処理を行っていることが主な原因であると考えられるため，再マーク処理に代わる方法を用いる必要がある．

また，今回の実験ではGCのリアルタイム性を示す最大停止時間についての検証が不十分であるため，今後実験を行う予定である．

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] 深井 優一，安井 浩之，吉野 邦生，““若返り”を用いた世代別GCの改良”，第11回情報科学技術フォーラム，pp.145-146 (2012)

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

[4] AO Bench,http://code.google.com/p/aobench