3G-01

Promote 条件の制御による Android 世代別 GC の性能向上に関する一考察

森竜佑[†1]　濱中真太郎[†2]　小口正人[‡]　山口実靖[†2]
工学院大学　工学部　情報通信工学科[†1]　工学院大学 工学研究科 電気・電子工学専攻[†2]
お茶の水女子大学 理学部 情報科学科[‡]

1. はじめに

仮想マシンAndroid Runtime(ART)には，ガベージコレクション(GC)機能があり．GCは空きメモリ量が減少すると不要となったメモリ領域を見つけて開放する．ARTに実装されているGCの一つに世代別GCがある．世代別GCではオブジェクトに年齢という概念を取り入れ，年齢ごとにオブジェクトを分類しGCを行う．ARTにおける世代別GCでは，メモリ領域を新世代領域と旧世代領域に分類し，新世代領域内GCで一定回数回収されなかったオブジェクトは旧世代領域へとPromote(昇進)する．また過去の研究にて，オブジェクトの分類の改善によりGCの性能を向上できる可能性が示されている[1]．本研究では，オブジェクトのPromote条件の制御による世代別GCの性能向上に関する考察を行う．

2. ARTにおける世代別GC

世代別GCとは，「生成されたばかりのオブジェクトはすぐにGCによって回収され，回収されずに長く生き残るオブジェクトは少ない」という経験に基づいた仮説に基づいたアルゴリズムである．ARTにはGenerational Semi Space(GSS)という世代別GCが実装されている．GSSではオブジェクトを2つの領域に分けて管理し，2つの異なる範囲のGCを行う．生成されたばかりのオブジェクトはyoung領域に置かれ，bpsGC(bump pointer space GC)によって高速にかつ積極的に回収される．bpsGCを2回生き残った(年齢:2)オブジェクトはold領域へとPromoteされる．そしてPromoteした全オブジェクトの累積バイト数がPromotedThreshold(初期設定にて4MB)を超えたらwholeGCが行われる．WholeGCの範囲は全領域でありGC時間が長いため，頻繁にwholeGCを行うことは好ましくない．ただし，old領域にPromoteしてから参照されなくなったオブジェクトはbpsGCの調査範囲外となってしまい回収できなくなってしまうため，wholeGCの回数の削減はヒープ拡大の原因になる可能性が考えられる．

A Study on Performance Improvement of Android Generational GC by Optimizing its Promote Condition
†[1]Ryusuke Mori †[2]Shintaro Hamanaka, Saneyasu Yamaguchi
‡Masato Oguchi
†[1]Department of Information and Communications Engineering, Kogakuin University
†[2]Electrical Engineering and Electronics, Kogakuin University
‡Department of Information Sciences, Ochanomizu University

3. Promote抑制によるヒープサイズの縮小

3.1 Promote抑制(時代考慮なし)手法

本節にて，Promote条件の制御によってヒープサイズを縮小させる手法を提案する．

本手法では，Promote対象のオブジェクトに対してn個に1個しかPromoteさせないようにし，old領域の拡大を防ぐとともに，寿命の短いオブジェクトが不必要にold領域に移動されてしまいbpsGCの対象から外れてしまうことを防ぐ．本手法は，Promote条件が最適な条件より緩い状況において，有効であると考えられる．

3.2 性能評価

上記手法をARTに実装し，性能評価を行った．1回のGCでPromoteするオブジェクトが少なくなることを考慮しPromotedThresholdは1MBと変更した．また，nは15とした．性能評価には以下の自作のアプリケーションを使用した．

自作アプリケーションでは，最初に長さ100のint型配列を保持するObj型クラスのインスタンスを100,000個発生させる．そして，Objクラスのインスタンスを発生して，その新規インスタンスで既存Obj型インスタンスへのポインタを1つ上書きすることを，60秒間繰り返す．ポインタが上書きされた既存インスタンスは，配列からの参照を失いゴミの状態となり，GCによって回収される対象となる．上書き対象のインスタンスの選択は，平均100の指数分布に従う．オブジェクトの発生箇所を偏らせることですぐにゴミとなるオブジェクトを多く発生させ，前述の仮説が有効に働くアプリケーションとなっている．

使用した端末はNexus7(2013)，OSのバージョンはAndroid 6.0.1，CPUはSnapdragon S4 Pro 1.5GHz，メモリは2GBである．

図1に各GC後のヒープサイズを示す．図1の横軸はGC回数を示している．測定結果より，最大で約6%のヒープサイズの縮小を実現できることがわかる．

また，図2のDefaultとPromote抑制(時代考慮無し)手法に，それぞれのアプリケーション性能を示す．図2の縦軸は自作アプリケーションにおけるアプリケーション性能であり，1秒間に作成されたオブジェクト数を示している．図2より，アプリケーション性能はPromote抑制(時代考慮なし)手法により劣化することが確認できる．

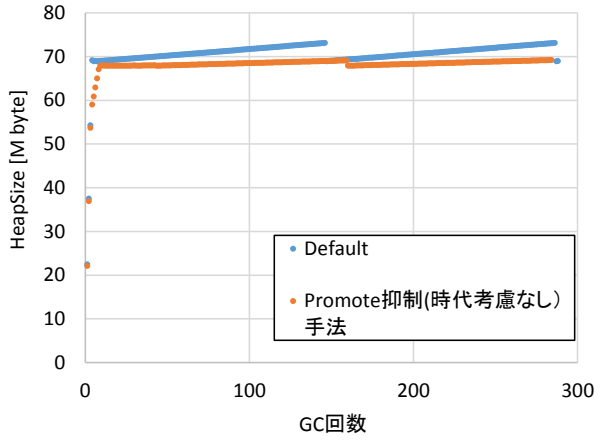

図1 ヒープサイズ

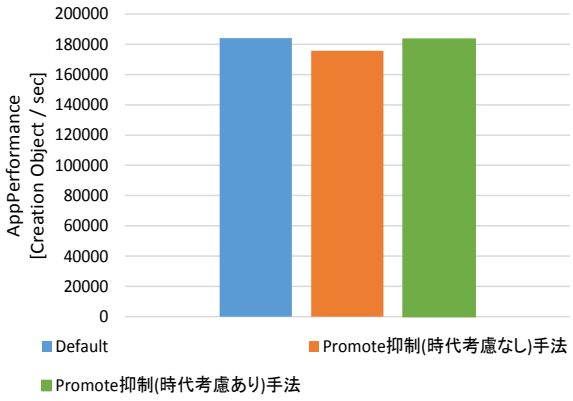

図2 アプリケーション性能

次にStop The World(STW)時間の評価を行う．図3にPromote抑制(時代考慮なし)手法における各GCのSTW時間の推移を示す．図3より，GC回数50回未満においては，本手法によりSTW時間が増加していることが確認できる．

4. 時代を考慮したPromote抑制

4.1 Promote抑制(時代考慮あり)手法

前章の手法の時代(アプリケーションが開始されてから累積GC回数)による性能の差を考慮してPromote条件を制御することにより，アプリケーション性能を低下させずにヒープサイズを縮小させる手法を提案する．

本手法では，時代(累積GC回数)が50回以下ではDefaultのPromote条件を用い，それ以降では前章で提案したPromote条件を用いてGCを動作させる．

4.2 性能評価

時代を考慮した手法の性能を図2のPromote抑制(時代考慮あり)に示す．

測定結果より，Promote抑制(時代考慮あり)手法はDefaultと同等の性能を実現していることがわかる．また，図4に本手法における各GC後のヒープサイズを示す．

Promote抑制(時代考慮なし)手法と同様に，最大で約6%

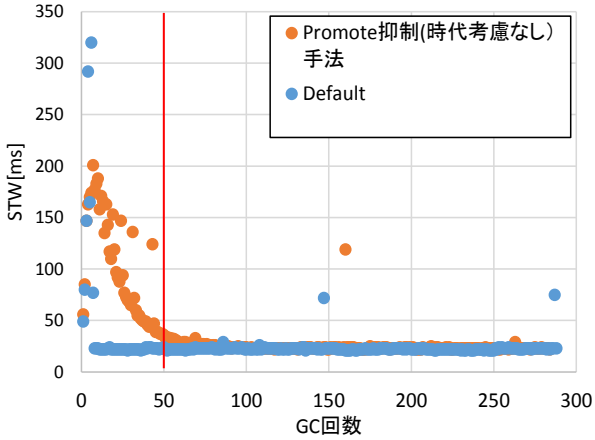

図3 提案手法1のSTWの推移

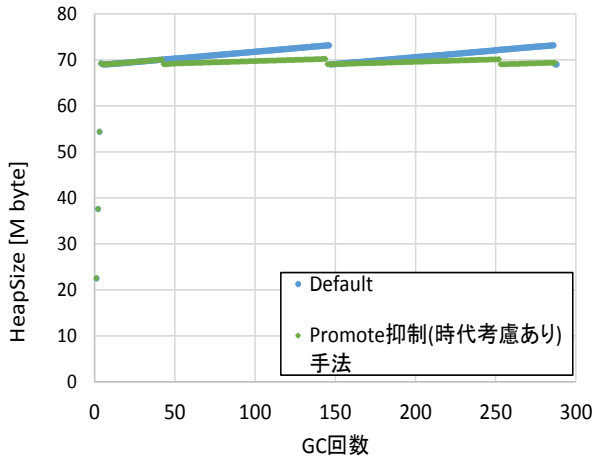

図4 ヒープサイズ

のヒープサイズの縮小を実現していることがわかる．以上より，アプリケーション性能を低下させずにヒープサイズの縮小を実現でき，時代考慮あり手法が有効であることが確認できる．

5. おわりに

本稿では，Promote条件を制御する手法を2つ提案し，性能評価を行った．その結果，時代を考慮する手法によりアプリの性能を低下させずにヒープサイズを縮小できたことが確認された．今後は，実アプリケーションにおける性能向上に取り組んでいく予定である．

謝辞

本研究はJSPS科研費 25280022, 26730040, 15H02696の助成を受けたものである．

本研究は，JST，CRESTの支援を受けたものである．

参考文献

[1]Shintaro Hamanaka, Shun Kurihara, Shoki Fukuda, Ryusuke Mori, Masato Oguchi, Saneyasu Yamaguchi “Object Lifetime Trend of Modern Android Applications for GC Performance Improvement”, ACM IMCOM 2017, 2017/1