M-025

Android ART の GC の性能に関する一考察

A Study on Garbage Collection Performance of ART in Android

濱中 真太郎† 坂本 寛和† 山口 実靖†
Shintaro Hamanaka Hirokazu Sakamoto Saneyasu Yamaguchi

1. はじめに

近年，スマートフォンやタブレット PC が普及し，それらの携帯端末で動作するソフトウェアプラットフォームとして Android OS が注目されている．Android OS バージョン 5.0 (Lollipop)ではアプリケーション実行環境として Android Runtime (ART)が採用されている．すべてのアプリケーションは ART 上で動作し，メモリ管理等は ART が行う．ART のメモリ管理機能のひとつとして GC (Garbage Collection)機能がある．GC は使わなくなったメモリ領域を見つけ開放を行う機能であり，GC の処理はアプリケーションの性能に影響を与える[1]．ART には複数の GC アルゴリズムが実装されており，アプリケーションの種類や状況により適した GC を選択し良い性能を得ることが好ましいと言える．本稿では ART に実装されている CMS (Concurrent Mark Sweep) GC と SS (Semi Space) GC の性能比較し，適した GC の選択手法について考察をする．

2. ART の GC

ART には複数の GC アルゴリズムが実装されている．標準で採用されているアルゴリズムは CMS GC であり，それ以外の GC アルゴリズムに変更することも可能である．

CMS GC は参照されているオブジェクト全てにマークをつけ，マークが付けられなかったオブジェクトを回収し，それらの領域を再度利用可能な状態にする．ART における CMS GC の動作は主に以下のフェイズに分けられる．カウンタなどに使用する変数などの初期化を行う InitializePhase，ルートオブジェクトをスキャンしてルートオブジェクトが参照しているオブジェクトをスキャンし，さらにそのオブジェクトが参照しているオブジェクトを再帰的にスキャンする MarkingPhase，アプリケーションを停止させマーキングフェイズ中に生じた参照の変化の差分を調査し整合性を確保するリマーク処理を行う PausePhase，ゴミオブジェクト(マークされていないオブジェクト)を回収し開放を行う ReclaimPhase，マークスタックやオブジェクトに付けたマーク等をクリアする FinishPhase に分けられる．

SS GC はヒープ領域内の参照されている全てのオブジェクトにマークを付け，マークされたオブジェクトを別領域にコピーし，元の領域を領域ごと破棄する．SS GC では InitializePhase，MarkingPhase，ReclimePhase，FinishPhase が順に行われる．InitializePhase，FinishPhase の内容は初期化，クリアを行う点で CMS GC と同等である．MarkingPhase では CMS GC 同様にオブジェクトを再帰的にマークする．その後，非ゴミオブジェクト(参照されておりマークされたオブジェクト)を別領域へコピーする．ReclaimePhase で元の領域の破棄を行う．

† 工学院大学大学院　電気電子工学専攻
Electrical Engineering and Electronics,
Kogakuin University Graduate School

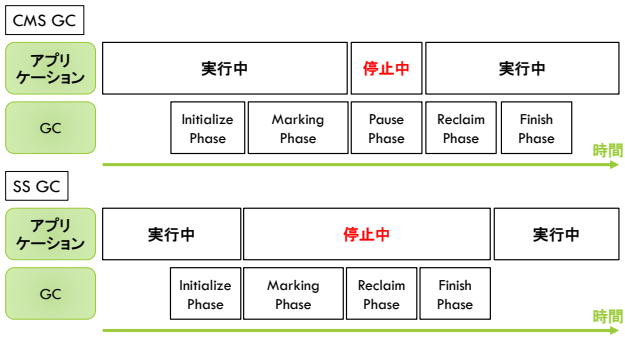

図 1. GC フェイズとアプリケーション停止時間

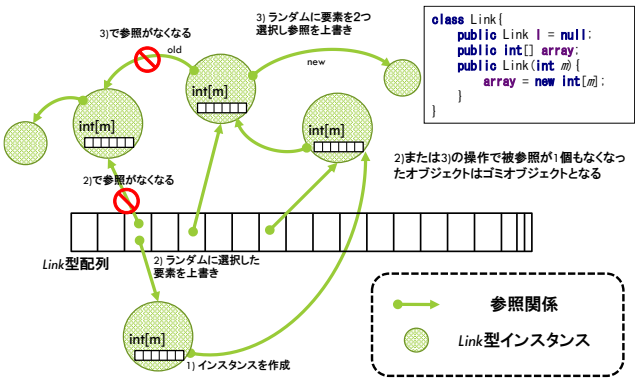

図 2. ベンチマーク概要

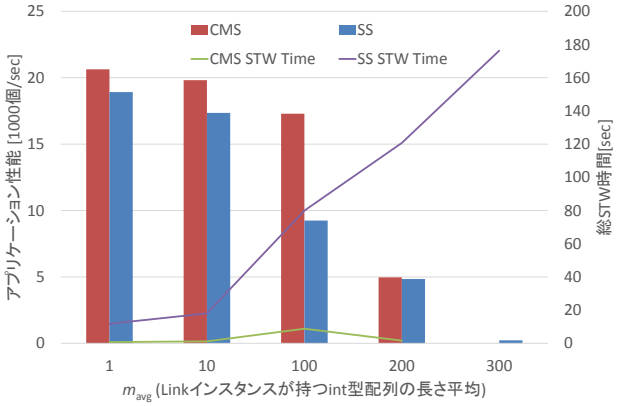

図 3. オブジェクトの大きさとアプリケーション性能

GC の処理によりアプリケーションが停止することを STW(Stop The World)と呼ぶ．図 1 の様に，CMS GC の InitializePhase，MarkingPhase，ReclimePhase，FinishPhase はアプリケーションと並行して動作するため，これらはアプリケーション性能を下げる原因とはなるがその影響は STW 処理と比較すると小さい．PausePhase では STW が発生しアプリケーションの動作は停止する．よって，アプリケーション性能への影響は大きい．SS GC においては，InitializePhase 以外では STW が発生するため，長く連続した STW が発生する．

3．性能評価

本章で CMS GC と SS GC の性能評価を行う．

3.1　測定環境

測定に使用した端末は表 1 の通りである．

表 1. 使用端末

端末名	Nexus 7　(2013)
OS	Android5.1.1_r1
CPU	Qualcomm Snapdragon S4 Pro,1.5 GHz
Memory	2GB

3.2　アプリケーション性能と STW 時間

CMS GC と SS GC を用いて自作のベンチマークアプリケーションを実行した．図 2 にベンチマークの概要を示す．ベンチマークはまず，*Link* 型オブジェクトを参照する長さ 100,000 の *Link* 型配列を作成する．*Link* 型はベンチマーク用に作成したクラスであり，*Link* 型のインスタンスは長さ *m* の int 型配列と他の *Link* 型インスタンスへのポインタを 1 個持つ．*m* はインスタンスを作成する際に決定され，平均 m_{avg} の指数分布に従う乱数で決定される．インスタンスを 100,000 個作成し，配列の要素がそれぞれを参照した状態から計測を始める．

ベンチマークは以下の 3 つの処理を繰り返す．1) *Link* 型インスタンスを 1 個作成する． 2) *Link* 型の配列の中からランダムに選択した要素を新たに作成したインスタンスで上書きする．3) *Link* 型配列からランダムに 2 つ要素を選択し，片方のインスタンスの中のポインタをもう片方へのポインタに上書きする．2)の結果，上書きされた要素から参照されていたインスタンスは配列からの参照を失う．この失われた参照が唯一の参照であった場合は，このインスタンスはゴミオブジェクトとなる．3)の書き換えは *n* 回行い，本稿ではこの *n* をリンク書き換え頻度と呼ぶ．2)および 3)の選択は，一様分布乱数により行う．測定時間は 3 分間行った．

3.3 測定結果

測定結果を図 3，図 4 に示す．各図の左縦軸はアプリケーション性能(オブジェクトの作成個数)であり縦棒で示されている．右縦軸は 3 分間の計測中に発生した STW 時間の合計で折れ線にて示されている．図 3 の横軸は *Link* 型オブジェクトのインスタンスが持つ int 型配列の長さの平均 m_{avg} である．図 4 の横軸はリンク書き換え頻度 *n* を示している．図 3，図 4 より，CMS GC はアプリケーション性能と STW 時間の両方において，すべての場合において SS GC よりも優れていることが確認できる．また，m_{avg} が大きくなると SS GC の STW 時間が大きくなることが確認できる．これは SS GC は GC のたびに非ゴミオブジェクトをコピーするからだと考えられる．ただし，図 3 の m_{avg} が 300 の場合 CMS GC では Out of Memory Error が発生し計測が途中で終了してしまい，計測を正常に行うことができなかった．これに関しては次節で考察を行う．

3.4　メモリ確保成功率

前節において Out of Memory Error が発生して計測が最後まで行えない場合があることを示した．本節では，オブジェクトの大きさとオブジェクト作成(メモリ確保)の成功率の関係を調査する．確認方法は，オブジェクトの作成を 1,000,000 回行いアプリケーションが Out of Memory Error を出さず正常に終了したら成功とする．この操作を 10 回

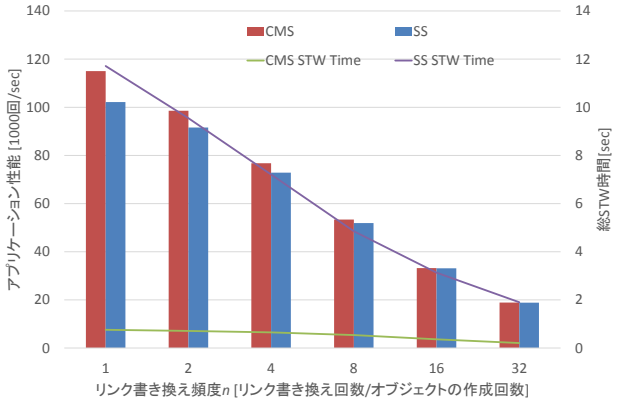

図 4. リンク書き換え頻度とアプリケーション性能

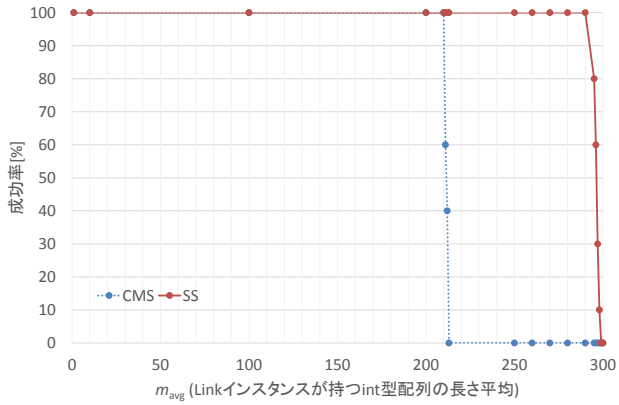

図 5. メモリ確保成功率

行う事により成功率を求めた．結果を図 5 に示す．図 5 の縦軸は成功率で，横軸が m_{avg} である．図より，SS GC のメモリ確保成功率は CMS GC より高いことが確認できる．

4. まとめ

本稿では ART に実装されている CMS GC と SS GC の性能の比較を行った．実験結果より，通常の(大量のメモリを消費しない)アプリケーションにおいては CMS GC の方がアプリケーション性能が高く適しており，大量のメモリを確保するアプリケーションにおいては SS GC が適していることが分かった．

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．

参考文献

[1]永田恭輔，中村優太，野村　駿，山口実靖, "Dalvik VM コンカレント GC の STW 時間の短縮に関する考察", 情報処理学会 第 76 回全国大会, 6J-2