オブジェクトの生存率の理論的解析に基づいた世代別ガーベッジコレクション

田中詠子[†] 前田敦司[†]
田中良夫[††] 中西正和[†††]

世代別ガーベッジコレクション（世代別GC）においては，殿堂入りが早過ぎると，長寿命領域を無駄にしてしまうオブジェクト（tenured garbage, TG）が発生する．従来の手法では，TGを削減することに重きがおかれ，いくつかの実験，経験に基づいて殿堂入りのためのしきい値が設定されている．また，世代別GCはオブジェクトの寿命の概念に基づいたGCであるが，実際にオブジェクトの生存率について解析された例はない．本論文では，オブジェクトの生存率に関する数理モデルを示し，アプリケーション実行時のオブジェクトの生存率によって，しきい値の最適値を動的に設定するAdaptive Garbage Collection（AGC）を提案する．様々なアプリケーションを実行して求めたオブジェクトの生存率の実験結果に対する統計処理より，我々の仮説の数理モデルは正しいことを示す．しきい値の最適値に関して，世代別GCにおけるコストは，GCによる処理の中断時間（「通常GC時の複写によるコスト」）と「長寿命GCによるコスト」であることを示し，AGCでは，このコストを最小にするようなしきい値を設定する．実際に実行したいくつかのアプリケーションが従来の手法と比べ，より効率良く実行されることを示し有効性を示す．

Generational Garbage Collection Based on Theoretical Analysis of Lifetime of Objects

EIKO TANAKA,[†] ATUSI MAEDA,[†] YOSHIO TANAKA[††]
and MASAKAZU NAKANISHI[†††]

We propose a new, efficient GC algorithm based on *adaptive garbage collection* (AGC). This new approach solves tenuring problem in generational garbage collection. In this method, advancement threshold is dynamically set based on theoretical analysis of survival probability of objects. The optimal advancement threshold, in AGC, can be set taking account of the trade-off between the penalty paid for the old object space GC and the copying cost, while existing schemes only tried to reduce tenured garbage. And the optimal advancement threshold is dynamically set based on survival probability of objects on executing application. GCs are executed with adjusting advancement threshold dynamically at runtime according to the cell consumption pace. Cell consumption pace of running application is measured on the fly and advancement threshold is adjusted dynamically to minimize the total garbage collection cost. We describe a mathematical model which approximates survival probability of objects. Detailed explanation of our algorithm is also given. The paper closes with the result showing that every application we tested executed more efficiently than other schemes.

1. はじめに

リスト処理言語における，ガーベッジコレクション（GC）によるリスト処理の中断時間を短くするための効率の良いGCアルゴリズムの1つに「ほとんどのセル（オブジェクト）は生成後間もなく死んでしまい，ある程度長い期間生き抜いたセルは半永久的に生き続ける」[3]という性質に基づいた**世代別ガーベッジコレクション（世代別GC）**[3),4),12),17)]がある．世代別GCはオブジェクトをその寿命に応じていくつかの世代に分ける．世代数は処理系によって異なるが，Lispの場合には世代数を2としているものが最も多い．本論文においても，世代数は2として議論をする．メモリ領域は，若い世代のための領域と，1つ上の世代のため

† 慶應義塾大学大学院理工学研究科
Graduate School of Science and Technology, Keio University
†† 新情報処理開発機構
Real World Computing Partnership
††† 慶應義塾大学理工学部数理科学科
Faculty of Science and Technology, Keio University

の領域に分けられる．世代別GCの代表的な手法を例にあげて説明する．メモリ領域は1つの**生成領域**と2つの**短寿命領域**，1つの**長寿命領域**に分けられる[12]．また，通常のGCと長寿命領域のGCの2種類のGCがある．

(1) 通常のGC（以下**通常GC**）
世代別GCの通常のGCは短寿命領域中での複写法である．通常の複写法と異なる点は以下の3つである．
(a) 新しいオブジェクトは，生成領域に生成される．
(b) 各オブジェクトは通常GCを生き抜いた回数，つまり複写された回数を記憶する．
(c) GCを生き抜いた回数がある一定の回数（**Advancement Threshold, AT**）を超えた場合に，短寿命領域ではなく長寿命領域に複写する（この複写を**殿堂入り**という）．
つまり，通常GCは，生成領域および短寿命領域をGCの対象とする．すなわち，生成領域から短寿命領域あるいは長寿命領域へ，および，短寿命領域から長寿命領域への生存オブジェクトの複写である．

(2) 長寿命GC
長寿命領域がすべて使用されると長寿命領域も含めたGCを行う必要がある．長寿命領域内部でのGCのことを**長寿命GC**と呼ぶ．

若い世代のオブジェクトはほとんどがゴミであるため，通常GCはきわめて短い時間で終了する．このように，世代別GCは，通常GCの対象を生成後間もないオブジェクトに絞ることによりGCにかかる時間をきわめて短くする効率の良い手法である．世代別GCでは，ATの値をいくつに設定するかがGCの効率に大きな影響を与える．最適なATの値を求める問題は**殿堂入り問題（tenuring problem）**[6),12)]と呼ばれ，いくつかの解決策が提案されている[1),12)~14),19)]．

長寿命領域は一般的に短寿命領域に比べて大きな領域が割り当てられるので，長寿命GCには長い時間がかかってしまう．したがって，長寿命GCにはなるべく入らない方がよい．つまり，長寿命領域でゴミが発生するのは極力抑える必要がある．そのためにはATの値を大きくすればよいが，あまり大き過ぎると短寿命領域間で長寿命オブジェクトが繰り返し複写されることになり，通常GCにかかる時間が長くなってしまう．またATの値が小さ過ぎると，通常GCの対象から外された後にゴミとなってしまい，長寿命領域を無駄にしてしまうオブジェクト（**tenured garbage, TG**）が発生する．

我々は，世代別GCの基本であるオブジェクトの生存率に関する数理モデルを示し，アプリケーション実行時のオブジェクトの生存率によって，ATの最適値を動的に設定する**Adaptive Garbage Collection（AGC）**を設計，実装した．AGCは，従来のTGの削減のみに重きを置いた手法に対して，世代別GCにおけるコストの主因と考えられる「通常GC時の複写によるコスト」と「長寿命GCによるコスト」のトレードオフを考慮に入れて，ATの最適値を理論的に求めて設定することが可能である．

2. 従来の問題点

殿堂入り問題解決策の中の1つに，Ungarらが提案した**Demographic Feedback-Mediated Tenuring（DFMT）**[13),14)]がある．DFMTでは，短寿命領域間を複写されるセルの数がある一定の数を超えるまでは殿堂入りせず，超えたときに古いセルから順に，（超えた数以上になるまで）殿堂入りする．DFMTは通常GCにかかる時間をほぼ一定時間以内に抑えつつ，殿堂入りするセルの数を最小限に抑える方法である．一方，Wilsonが提案した**Opportunistic Garbage Collector（OGC）**[15),16),18),19)]では，ATの値は1から2の間がよいとしている．OGCは，メモリ領域は1つの生成領域，2つの短寿命領域および1つの長寿命領域により構成される．セルは生成領域中に作られ，生成領域が使い尽くされると通常GCに入る．生成領域には**watermark**と呼ばれる境界が設けられており，生きているセルの中でアドレスがwatermarkよりも小さい（watermarkより前に生成された）セルは，生成後ゴミになるための十分な時間が経過しているため，長寿命セルと見なされて殿堂入りされる．これらのセルのATは1となる．watermarkよりもアドレスが大きいセルは生成後まだ十分時間が経過していないため，まだゴミになる可能性があると見なされて短寿命領域に複写される．短寿命領域中で生きているセルは無条件に殿堂入りされる．これらのセルのATは2となる．watermarkの位置を調節することにより，ATの値を平均的に1から2の間に自由に設定できるようになっている．

DFMTとOGCとの間には見解の違いがある．それは，DFMTが「ATの最適値は2以上である」としているのに対し，OGCは「ATの最適値は1から2の間」としているという点である．もしも「ATの最適値が1から2の間」であれば，DFMTで行われる

短寿命領域間のセルの複写は無駄である．それらのセルは殿堂入りされるべきであり，殿堂入りすることによって複写の数も減り，通常GCにかかる時間をさらに短縮することができる．一方「ATの最適値は2以上」なのであれば，OGCのような方法ではTGが大量に発生してしまうことになる．どちらの主張が正しいかは，生成領域や短寿命領域の大きさなど，様々な要素に依存するので一概にはいえない．それらの中で最も重要な要素が「死ぬべきセルは生成されてからどの程度の時間が経てば死ぬのか」という問題である．

世代別GCでは長寿命GCには非常に時間がかかる．従来の方法では，この長寿命GCをなるべく起こさないよう，TGの発生をできるだけ抑えることを目的としてきた．短寿命領域中で何度も複写を行い，長寿命領域へ殿堂入りするセルの数を減らせば，TGの発生率は低くなる．しかし，TGの発生率が低いということは生きているセルの割合が多いということである．生きているセルの割合が大きいと，長寿命GCにかかる時間は多くなってしまう．したがって，長寿命GCの方式によって多少の差はあるが，1回の長寿命GCでの中断時間は大きい．また，短寿命領域中でのセルの複写によるコストも大きい．

逆に，短寿命領域中での複写を減らし殿堂入りを早めれば，TGの割合は高くなる．長寿命GC時にTGの割合が高く，生きているセルの割合が低くなる．よって，1回の長寿命GCによる中断時間は前者の場合に比べて少なくて済む．この場合，短寿命領域中での複写によるコストは小さく，1回の長寿命GCでの中断時間は小さい．しかし，殿堂入りしたセルは短寿命領域で死ぬまでに十分な時間を与えられず，長寿命領域を無駄に消費するTGとなってしまう．

このように，従来の手法のように，単純にTGの数を削減するだけでは長寿命GCによるコストを減らすことにはならないと考えられる．また，長寿命GCによるコストが明確に定義できたところでそのコストを削減するようにATを設定するだけでは不十分である．通常GCによる停止時間がきわめて短いという利点を持つ世代別GCでは，通常GCにおける中断時間がセルの複写によって大きくなってしまうことは避けなければならない．つまり，ATの設定には，長寿命GCによるコストと通常GC時の複写によるコストのトレードオフを考える必要がある．複写コストと長寿命GCによるコストの重みがクリアでなければこれらの議論は収束しないであろう．両者のコストを明確に定義し，両者の関係の理論的な解析を行う必要がある．我々は，オブジェクトの生存率の定式化を行った[8),9)]．そして，オブジェクトの生存率に基づいて，通常GC時の複写コストと長寿命GCによるコストのトレードオフを計算し，ATの最適値を設定するAdaptive Garbage Collection（AGC）[7)~9)]の実装を行った．

3. オブジェクトの生存率の定式化

3.1 モデルの提案

オブジェクト（セル）が生成されて t 単位時間後に生存している（ゴミとなっていない）確率を**オブジェクト（セル）の生存率**と定義し，$\boldsymbol{P(t)}$ で表す．ほとんどのアプリケーションにおいて「ほとんどのセルは短寿命である」という性質が満たされることを考えれば，セルの生存率はアプリケーションに依存することなく同じ数理モデルで表すことができると考えられる．

まず，どのような数理モデルで近似できるかを考えるために，実験を行いセルの生存率曲線をグラフに表した．時間の刻みとして，実時間ではなく関数 *cons* の呼ばれた回数（consカウンタ）を採用した．関数 *cons* は，リスト処理において新しいセルを生成する関数である．

実験で実行したアプリケーションは，性質の異なる3種類のベンチマークプログラム（boyer[2)]，二進木の生成を行うbit[5)]，推論規則に従ってプログラムの自動合成を行うsynthe[20)]）である．これらのアプリケーションは，それぞれ以下のような特徴を持つ．

- boyer
  ある規則に従って項書き換えを行い，定理の自動証明を行うアプリケーション．多くのセルを消費するとともに，副作用によるポインタの書き換えが頻繁に行われる．大域変数への書き込みが頻繁に起こるため，長寿命領域から短寿命領域へのポインタが数多く出現する☆．
- bit
  リストを引数として与えると，リストの各要素をこの順で葉に持つすべての二進木のリストを返すアプリケーション．生成されたセルのうち多くの

☆ 一般に，このようなポインタを管理するためのテーブル（Remembered Set）を用意し，長寿命領域のセルの内容を変更するとき，それが短寿命セルへのポインタに変わるのであれば，そのセルの場所をテーブルに記憶させる．通常GCが起こったときは，このテーブルに記憶されているセルをルートの一部と見なす．長寿命領域から短寿命領域へのポインタが数多く出現する場合は，このテーブルの管理，通常GC時のマーキングにコストがかかると考えられるが，本システムでは，このテーブルを2つ用意し交互に利用することによって，管理およびGC時のコストがかからないように実現してある．

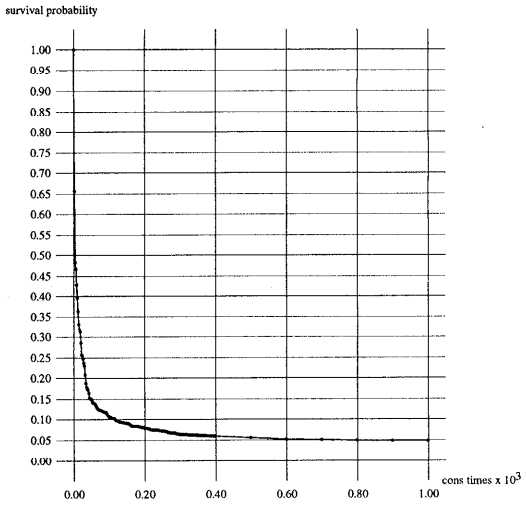


図1 boyer のセルの平均生存率（実験による）
Fig. 1 Survival probability of objects (by experiments).

セルが長い間生き続け，最終的に返される二進木のリストに使われるセルの数は非常に多い．実験では要素数は 12 であるが，その場合には返される二進木のリストに使われるセルの数は 266797 個になる．

- synthe
  関数名，引数，引数と返り値の論理的な関係を与えると，ある規則に従って推論を行い，Lisp プログラムの自動生成を行うアプリケーション．つねに持ち続ける規則が存在するが，そういった長寿命のセルの割合は少ない．他のセルは中途半端に長い間生き続けるため，世代別 GC の基本的な前提にあまりそぐわない．

これらのアプリケーションにおいて，各セルごとにその寿命を調べセルの平均生存率を求めた．図 1 に boyer を実行したときのセルの平均生存率のグラフを示す．

次に，どのようなモデルで表すことができるかを考える．長寿命セルの生存率は，$P_{long}(t)=1$ で表され，短寿命セルの寿命時間は指数分布すると考えることができると仮定し，$q(t)=\lambda e^{-\lambda t}$ で表す．「セルが時刻 t までに死ぬ確率」を考えると，$q(t)$ の累積分布関数 $Q(t)=1-e^{-\lambda t}$ と表される．したがって，短寿命セルの生存率は以下の式で表される．

$$P_{short}(t)=1-Q(t)=e^{-\lambda t}$$

全セルに対する長寿命セルの割合を r で表すと，セルの生存率は以下のように表すことができる．

表1 推定されたパラメータ
Table 1 Estimated parameters.

アプリケーション	平均寿命	λ	r
boyer	16.64	0.060	0.05
bit	13.55	0.074	0.30
synthe	28.34	0.035	0.016

$$\begin{aligned}P(t)&=(1-r)P_{short}(t)+rP_{long}(t)\\&=(1-r)e^{-\lambda t}+r\end{aligned}\qquad(1)$$

ここで，λ および r はアプリケーションによって異なるパラメータであり，λ は「セルがどの程度の速さで死んでいくか」を，r は「どの程度のセルが長寿命であるか」を表す．つまり，そのアプリケーションにおいて「寿命が長い」セルの割合が r となる．

3.2 パラメータの推定

指数分布 $e(\lambda)$ の平均は $1/\lambda$ と表される．したがって，λ は短寿命セルの平均寿命の逆数をとることによって求めることができる．また，r はセルの生存曲線より読みとることができる．3 種類のアプリケーションを実行して求めた平均寿命，λ，および前述のグラフより読みとった r の値を**表 1** に示す．

様々なアプリケーションを実行してセルの生存率に関するデータをとり，そのデータの解析から求めたパラメータを式 (1) に代入した関数のグラフを書いた．その結果，実験結果によるセルの生存率は，我々の立てた仮定とほぼ一致することが確かめられた．また，実験結果より，AT の最適値は 1 と 2 の間にあるということも確かめられている[8),9)]．

4. Adaptive Garbage Collection

Adaptive Garbage Collection (AGC)[7)~11)]は，アプリケーションの実行に応じて動的に AT を設定する有効なアルゴリズムである．AGC は，従来の，TG の削減のみに重きを置いた手法に対して，世代別 GC におけるコストの主因と考えられる「通常 GC 時の複写によるコスト」と「長寿命 GC によるコスト」のトレードオフを考慮に入れて，AT の最適値を理論的に求めて設定することが可能である[7)~9)]．

初期の AGC (oldAGC) においてはもともと，TG を最小とするような AT の値のうち，複写によるコストも最小となるときの AT の値を，AT の最適値としていた[10),11)]が，本稿においては，**GC コストと AT の最適値**を以下のように定義する．

GC コスト = 1 回の通常 GC 時の複写によるコスト + 長寿命 GC のコストの m 分の 1 (ただし，m 回の通常 GC に対して 1 回の長寿命 GC が発生するとする)．

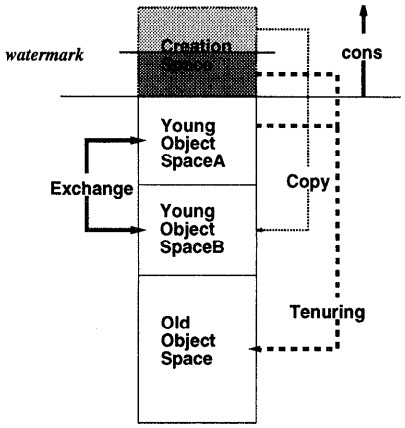


図2 AGCにおけるメモリ領域
Fig. 2 Memory usage of AGC.

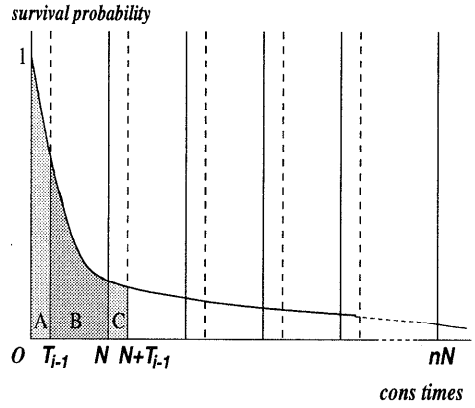


図3 セルの生存曲線1
Fig. 3 Survival probability of objects.

ATの最適値：GCコストが最小となるようなATの値.

セル消費のペースやセルの生存率は1つのアプリケーション実行中でも一定ではない．したがって，通常GCが起こるたびにセルの平均生存率を推定してGCコストを計算し，ATの最適値を求める必要がある．これによって，セルの消費のペースの変化に柔軟に対応することができる．

4.1 AGCのアルゴリズム

AGCでは，メモリ領域は**図2**のようになっている．AGCでは，通常GCのたびにwatermarkをずらすことによって，ATを動的に設定する．oldAGCでは，生成領域から直接長寿命領域へ生き残るセルの割合のみをパラメータとして，通常GCのたびにATを設定していた[10),11)]．本稿における新しいAGC（以下，単にAGCと記す）においては，セルの生存率を考慮に入れた新たなアルゴリズムを提案する[7)~9)]．以下にAGCのアルゴリズムを示す．

- セルの生存率を表す式を推定する．
- GCコストを最小にするようなATの値を求める．
- 求まったATの値を次の通常GCでのATとして採用する．

セルの生存率を表す式の推定方法，その推定された式に基づいた通常GC時の複写によるコストと長寿命GCによるコストの和であるGCコストの計算方法，ATの最適値の決定方法について順に述べる．

4.2 セルの生存曲線の推定

アプリケーションごとのセルの生存率を表す式 $(1-r)e^{-\lambda t}+r$ は，アプリケーション実行中に λ と r の値が変化するので，通常GCの起きるたびに λ，r の値が求まれば，そのときの生存率を表す式が求められる．

図3は，時刻0で生成されたセルの生存率を表すグラフである．今，単位時間に生成されるセルの数は一定であるものとする．生成領域のセルの数を N とすれば，セルが N 個生成されたときに1回目の通常GCが起こる．これを時刻 N で表すことにする．通常GCの回数を n で表せば，セルが nN 個生成されたときに n 回目の通常GCが起こることになる．図の面積A，Bは，i 回目の通常GC時に生成領域中で生き残っているセルの数を表す．いま，n 回目の通常GCで領域 a から領域 b へ生き残るセルの数を $S(a,b,n)$ と表すことにする．したがって，i 回目の通常GCで，生成領域から短寿命領域および長寿命領域へ生き残るセルの数はそれぞれ $S(C,Y,i)$（面積A），$S(C,O,i)$（面積B），短寿命領域から長寿命領域へ生き残るセルの数は $S(Y,O,i)$（面積C）と表される．図の点線の T_{i-1} の部分はwatermarkの位置である．i 回目の通常GCにおける生存曲線の推定は前回のGCにおいて設定されたwatermarkの値を使う．したがって図の点線の T_{i-1} の部分は前回のGCにおいて設定されたwatermarkの位置である．今，$S(Y,O,i)$ から，r を求めることを考える．$S(Y,O,i)$ は，以下の式で表される．

$$\begin{aligned} S(Y,O,i) &= \int_{N}^{N+T_{i-1}} \{(1-r)e^{-\lambda t}+r\}dt \\ &= \frac{1-r}{\lambda e^{\lambda N}} + \frac{-1+r}{\lambda e^{\lambda(N+T_{i-1})}} + rT_{i-1} \\ &\cong rT_{i-1} \end{aligned} \tag{2}$$

したがって，r は，

$$r = \frac{S(Y,O,i)}{T_{i-1}} \tag{3}$$

と求めることができる．

次に，r と，$S(C,Y,i)$，$S(C,O,i)$ より，λ を求め

ることを考える．

$$\begin{aligned} & S(C,Y,i)+S(C,O,i) \\ &= \int_0^N \left\{(1-r)e^{-\lambda t}+r\right\} dt \\ &= \frac{1-r}{\lambda}+\frac{-1+r}{\lambda e^{\lambda N}}+rN \\ &\cong \frac{1-r}{\lambda}+rN \end{aligned} \tag{4}$$

より，λ は，

$$\lambda = \frac{1-r}{S(C,Y,i)+S(C,O,i)-rN} \tag{5}$$

と求めることができる．

$S(C,Y,i)$，$S(C,O,i)$，$S(Y,O,i)$ は，通常 GC のたびに容易に数えることができる．したがって，式 (3)，式 (5) より連立方程式を解くことによって，λ，r の値を求めることができる．これで，セルの生存率を表す式を求めることができる．

4.3 通常 GC 時の複写によるコストと長寿命 GC によるコスト

GC による停止時間がきわめて短いという利点を持つ世代別 GC では，通常 GC における中断時間がセルの複写によって大きくなってしまうことは避けなければならない．このことを考慮にいれると，AT を決定する際の「通常 GC 時の複写によるコスト」とは，1 回の通常 GC での複写の回数と考えられる．これは，生成領域から短寿命領域，長寿命領域への複写の数，および短寿命領域から長寿命領域への複写の数の和である．これは，図 3 における面積 A，B，C の和になる．したがって，通常 GC1 回の複写によるコストを V^i_{copy} で表すと，次の式で表される．

$$V^i_{copy} = \int_0^{N+T_i} \{(1-r)e^{-\lambda t}+r\} dt \tag{6}$$

長寿命 GC による中断時間は，長寿命 GC の手法によって多少異なるが，生きているセルの数に影響されると考えられる．つまり，長寿命 GC が起きると，そのコストは長寿命領域で生きているセルの数が多いほど大きくなる．この，長寿命 GC が起きたときに長寿命領域中で生きているセルの数を $N_{longlive}$ とする．長寿命 GC によるコストは，長寿命 GC が起こったときの生きているセルの数に比例すると考えれば，その比例定数はシステムによって異なると考えられる．この比例定数を k とする．k はシステムによって異なる．

次に，通常 GC 1 回あたりの長寿命 GC によるコストを考える．長寿命 GC は，殿堂入りするセルによって長寿命領域が使い尽くされたときに起きる．ここで，殿堂入りするセルの数は通常 GC において毎回一定とし，長寿命 GC が起こるまでの通常 GC の回数を m とする．m の値が大きければ，長寿命 GC はなかなか起きない．逆に m の値が小さければ，長寿命 GC が早く起きる．したがって，通常 GC 1 回あたりの長寿命 GC によるコストは，長寿命 GC が起きる時点での生きているセルの数が同じであれば m の回数に反比例すると考えられる．通常 GC 1 回あたりの長寿命 GC によるコストを V^i_o とすれば，以下の式が導かれる．

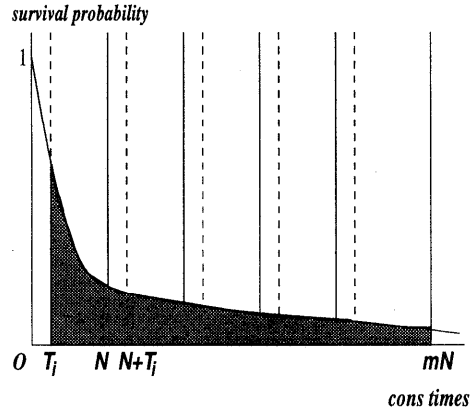


図 4 セルの生存曲線 2
Fig. 4 Survival probability of objects.

$$V^i_o = \frac{1}{m} k N_{longlive} \tag{7}$$

$N_{longlive}$ は，図 4 の斜線部分である．したがって，以下の式で表される．

$$N_{longlive} = \int_{T_i}^{mN} \{(1-r)e^{-\lambda t}+r\} dt \tag{8}$$

実際には長寿命領域へ殿堂入りするセルの数は通常 GC の度に変わるので，m の値も通常 GC のたびに変わると考えられる．m は，

$$m = \frac{\text{長寿命領域の全セルの数}}{\text{通常 GC 1 回で長寿命領域へ移されたセルの数}} \tag{9}$$

と表される．長寿命領域の全セルの数，1 回の通常 GC で長寿命領域へ移されたセルの数は容易に求めることができるので，m の値も容易に計算することができる[7)~9)]．

4.4 AT の最適値の決定

i 回目の通常 GC における GC コストを V^i_{gc} と表すことにすると，GC コストは以下の式のように表される．

$$V^i_{gc} = V^i_{copy} + V^i_o \tag{10}$$

式 (10) は T_i に関する式になるので，V^i_{gc} を最小にするような T_i の値を求めることができる．V^i_{gc} を最

小にするような T_i の値は，

$$\frac{d}{dT_i}V_{gc}^i = 0 \tag{11}$$

を解くことによって求められる．式 (11) は，簡単にすると以下の式で与えられる．この場合，N_{long} は長寿命領域の全セルの数を表す．

$$\begin{aligned}&\frac{2k(1-r)^2}{\lambda r}e^{-2\lambda T_i}\\&-k(1-r)\left(T_i-N-\frac{1}{\lambda}\right)e^{-\lambda T_i}\\&+krN-N_{long}=0\end{aligned} \tag{12}$$

N は生成領域のセルの数であり，この式を満たす T_i は，watermark までに含まれるセルの数の最適値ということになる．この値を T_{opt} で表すと，$0 < T_{opt} < N$ ならば，AT の最適値は，1 と 2 の間に設定される．λ，r はアプリケーションに固有な数値であるから，λ，r によって AT の最適値が決まることが分かる．したがって，アプリケーション実行中に λ，r を求め，セルの生存曲線を求めることによって，式 (12) から，そのときの AT の最適値を求めることができる．

5. 評　　価

本章では，AGC 実装における評価方法と評価結果について述べる．4.3 節で定めた GC コストは，1 回の通常 GC におけるコストを示す．したがって，アプリケーションを実行した後の評価方法にこの GC コストはそのまま使えない．評価には，アプリケーション実行全体にかかる GC コストを考える．

5.1 評価方法

1 回の実行における GC コストを V_{gc}^{total} と表す．V_{gc}^{total} は，アプリケーション実行終了後のセルの全複写数（V_{copy}^{total}）と 1 つのアプリケーション実行において長寿命 GC を引き起こす総コスト（V_o^{total}）の和で表され，次の式で与えられる．

$$V_{gc}^{total} = V_{copy}^{total} + V_o^{total} \tag{13}$$

まず，V_o^{total} について考える．式 (7) より，

$$V_o^i = \frac{1}{m}kN_{longlive}$$

が，1 回の通常 GC あたりの長寿命 GC のコストであったので，V_o^{total} は，総コストとして，V_o^i を通常 GC の総回数（x）倍する．したがって，式 (7) より，以下の式が導かれる．ただし，この場合の $N_{longlive}$ は，$\int_T^{mN}\{(1-r)e^{-\lambda t}+r\}dt$ で与えられる．式 (8) における T_i の値は 1 回の通常 GC における値であるので，アプリケーション実行全体にかかる GC コストを計算するときには，平均値 T を使う．

$$\begin{aligned}V_o^{total} &= xV_o^i\\&=\frac{x}{m}kN_{longlive}\\&=\frac{x}{m}k\int_T^{mN}\{(1-r)e^{-\lambda t}+r\}dt\end{aligned} \tag{14}$$

次に，V_{copy}^{total} は，アプリケーション実行中のセルの全複写数であるから，

$$\begin{aligned}&V_{copy}^{total}\\&=\sum_{i=1}^{x}\{S(C,Y,i)+S(C,O,i)+S(Y,O,i)\}\end{aligned} \tag{15}$$

と表される．

m は，長寿命領域が使い尽くされるまでの通常 GC の回数である．アプリケーション実行全体にかかる GC コストを計算するときには，そのアプリケーション実行時の 1 回の通常 GC で長寿命領域に殿堂入りされるセルの数の平均値（N_{tenure}^{ave}）を使って次の式で与えられる．

$$m = \frac{N_{long}}{N_{tenure}^{ave}} \tag{16}$$

X をアプリケーション実行後の長寿命領域の消費セル数，通常 GC の回数を x とすれば，長寿命領域に殿堂入りされるセルの数の平均値（N_{tenure}^{ave}）は

$$N_{tenure}^{ave} = \frac{X}{x} \tag{17}$$

である．

したがって，アプリケーション実行後の総 GC コストは，式 (13)〜(17) より，以下のように示される．

$$\begin{aligned}V_{gc}^{total} &= V_{copy}^{total} + V_o^{total}\\&\cong\sum_{i=1}^{n}\{S(C,Y,i)+S(C,O,i)+S(Y,O,i)\}\\&\quad+\frac{kX}{N_{long}}\left\{r(mN-T)+\frac{(1-r)e^{-\lambda T}}{\lambda}\right\}\end{aligned} \tag{18}$$

V_{copy}^{total} は，各通常 GC のたびに複写の数を数えることによってその数のデータをとることができる．V_o^{total} は，アプリケーション実行終了後に，m を計算することによって求めることができる．X はアプリケーション実行後の長寿命領域の消費セル数であり，N は生成領域の全セル数，N_{long} は長寿命領域の全セル数，λ と r は 3 章で示した，各アプリケーションごとに求めた実験値である．T は，OGC においては，実行

中にその値は変わらないので，設定されている T の値を計算に使う．AGC，DFMT においては，実行中にその値が変わるので，平均値を使った．この方法によって，AGC を実装したシステムと DFMT，OGC を実装したシステムでアプリケーションを実行し，評価を行った．

5.2 評価結果

評価を行ったアプリケーションは，boyer，bit，synthe である．AGC では，AT の初期値を 1.0 から 2.0 の間に 0.1 刻みで設定して実験を行ったが，初期値にかかわらず AT の値は次第に 1.0 に近づく．OGC は，生成領域の watermark を上下させて，AT を 1.0 と 2.0 の間に 0.1 刻みで設定し，各場合における総 GC コストを求めた．DFMT では，短寿命領域中のしきい値を等間隔で 10 通りに設定し，それぞれの場合における総 GC コストを求めた．OGC，DFMT，AGC ともに，k の値を変えて，実験，計算を行った．

OGC，DFMT，AGC を実装したシステムにおける boyer，synthe の総 GC コストを図 **5**，図 **6** に示す．横軸は k の値を示す．縦軸は総 GC コストを表す．OGC，DMFT に関しては，T の値によって結果が異なるので，総 GC コストの最大値，最小値を示す．AGC に関しては，初期値にかかわらず T の値はほぼ同じようになるので，初期値が変わっても総 GC コストにはほとんど差がない．boyer においては k の値にかかわらず，AGC の総 GC コストは，OGC，DFMT における総 GC コストの最小値とほぼ同等か，それ以下になることが分かる．DFMT における総 GC コストの最大値が他のものに比べて非常に大きいのは，短寿命領域中で何度も長寿命のセルが複写されることによるコストである．bit でも同様の結果を得た．synthe において，k の値が 40 を超えると，DFMT の総 GC コストの最小値が AGC の総 GC コストより小さくなる．これは，synthe が世代別 GC の性質に反し，短寿命のセルが中途半端に長く生き続けることから，k が大きい場合，長寿命 GC によるコストにかかる重みが大きいので，この場合の AT の最適値は 2.0 を超えるためと考えられる．つまり，中途半端に長い間生き続ける中寿命のセルが多いようなアプリケーションにおいて，k が大きい場合は，AT の最適値は 2.0 を超えてしまうため，DFMT のように長寿命のセルを繰り返し複写して長寿命領域へ殿堂入りするセルの数を抑える必要があるということである．AGC は，短寿命のセルが多く，中途半端な寿命を持つセルが少ないアプリケーションにおいては，総 GC コストを最小に抑えることができる．

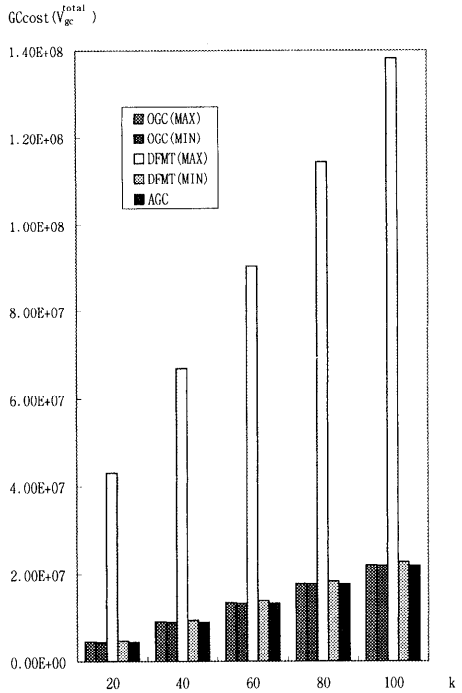


図 5 boyer における総 GC コスト

Fig. 5 Comparing total GC cost of OGC, DFMT and AGC on boyer.

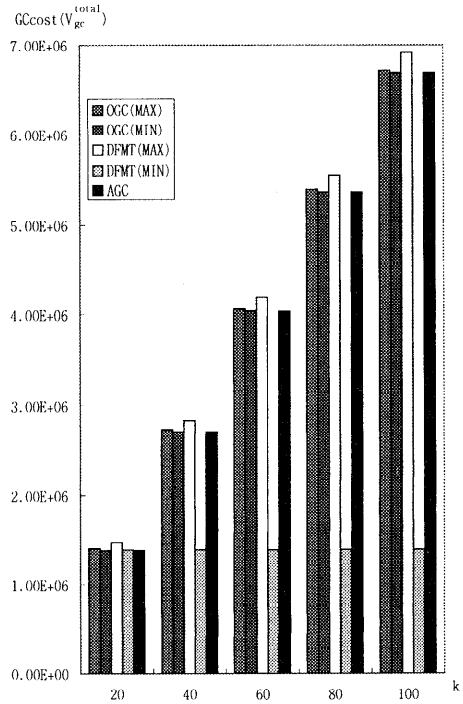


図 6 synthe における総 GC コスト

Fig. 6 Comparing total GC cost of OGC, DFMT and AGC on synthe.

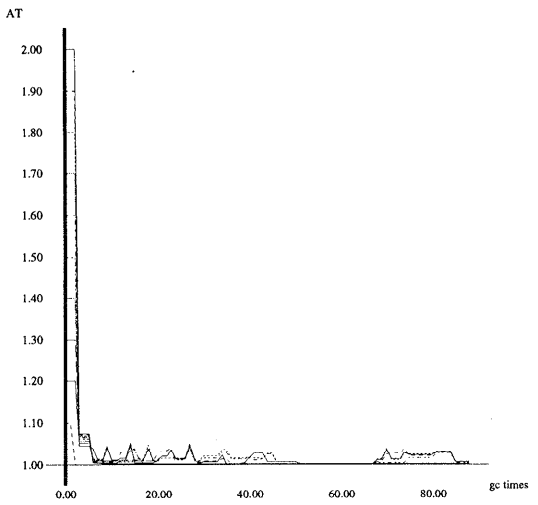


図7 boyer における AT の変動
Fig. 7 Fluctuation of AT on boyer.

AGC において，AT は動的に変化する．AT がどのように変化するかをグラフに表す．**図7** は boyer における AT の変動を表したものである．他のアプリケーションにおいても AT の値は変動する．AT の初期値を 1.0 から 2.0 の間に 0.1 刻みで設定して実験を行ったが，初期値にかかわらず，AT の値はほぼ同じ位置になることが分かる．横軸は通常 GC の回数，縦軸は AT の値である．このグラフを見ても，AT を動的に変化させる必要があることが分かる．

6. 結　　論

本論文では，世代別 GC の基本であるオブジェクトの生存率に関する数理モデルを示し，アプリケーション実行時のオブジェクトの生存率によって，しきい値の最適値を動的に設定する Adaptive Garbage Collection（AGC）の提案を行った．AGC では，従来の TG の削減のみに重きを置いた手法に対して，世代別 GC におけるコストの主因と考えられる「通常 GC 時の複写コスト」と「長寿命 GC によるコスト」のトレードオフを考慮に入れて，しきい値の最適値を理論的に求めて動的に設定することが可能である．設計した AGC を実際に実装し，他の世代別 GC（DFMT，OGC）を実装したシステムとの比較を行った結果，AGC の 1 回のアプリケーションにおける総 GC コストは，他の世代別 GC において最小になる総 GC コストと同等かそれ以下になることが示された．世代別 GC は，「オブジェクトの寿命」に基づいた GC であり，その効果は広く認められているが，オブジェクトの寿命について実際に解析を行った例はない．本論文において，オブジェクトの寿命に関する解析と，GC コストの理論的な解析ができたことによって，生成領域，短寿命領域，長寿命領域の大きさ，キャッシュメモリなど，その他のいろいろな要素を考慮に入れることによって，よりコストの少ない GC を開発することが可能になる．

謝辞 本論文の査読におきまして有益なご助言をいただきました査読者の方々に感謝いたします．

参 考 文 献

1) Barrett, D.A. and Zorn, B.G.: Garbage Collection Using a Dynamic Threatening Boundary, *Proc. SIGPLAN '95 Conference on Programming Language Design and Implementation (PLDI)*, pp.301–314 (1995).
2) Gabriel, R.P.: *Performance and Evaluation of Lisp Systems*, Series in Computer Science, MIT Press, Cambridge, Massachusetts (1985).
3) Lieberman, H. and Hewitt, C.: A Real-time Garbage Collector Based on the Lifetimes of Objects, *Comm. ACM*, Vol.26, No.6, pp.419–429 (1983).
4) Moon, D.A.: Garbage Collection in a Large Lisp System, *Conference Record of the 1984 ACM Symposium on Lisp and Functional Programming*, pp.235–246 (1984).
5) 中西正和：Lisp 入門 — システムとプログラミング，近代科学社 (1985).
6) 中西正和，田中詠子：特集：<ごみ集めの基礎と最近の動向>世代別ごみ集め，情報処理，Vol.35, No.11, pp.1014–1019 (1994).
7) Tanaka, E., Maeda, A., Tanaka, Y. and Nakanishi, M.: Adaptive Garbage Collection Based on Theoretical Analysis of Lifetime of Objects, *Proc. Fourteenth IASTED International Conference on Applied Informatics*, pp.19–22 (1996).
8) Tanaka, E., Tanaka, Y. and Nakanishi, M.: A Theoretical Analysis of Advancement Threshold in Generational Garbage Collection, *Proc. 13th IASTED International Conference on Applied Informatics*, pp.378–381 (1995).
9) 田中詠子：世代別ガーベッジコレクションに関する研究，博士論文，慶應義塾大学 (1996).
10) 田中詠子，田中良夫，中西正和：Adaptive Garbage Collection の提案および実験，電子情報通信学会論文誌（D-I），Vol.J77-D-I, No.9, pp.611–618 (1994).
11) 田中詠子，田中良夫，中西正和：Adaptive Garbage Collection — 実装とその評価，電子情報通信学会論文誌（D-I），Vol.J79-D-I, No.5, pp.253–260 (1996).
12) Ungar, D.: Generation Scavenging: A Non-

Disruptive High Performance Storage Reclamation Algorithm, *ACM SIGPLAN Notices*, Vol.19, No.5, pp.157–167 (1984).
13) Ungar, D. and Jackson, F.: Tenuring Policies for Generation-based Storage Reclamation, *Proc. OOPSLA '88*, pp.1–17 (1988).
14) Ungar, D. and Jackson, F.: An Adaptive Tenuring Policy for Generation Scavengers, *ACM Trans. Prog. Lang. Syst.*, Vol.14, No.1, pp.1–27 (1992).
15) Wilson, P.R.: A Simple Bucket-brigade Advancement Mechanism for Generation-based Garbage Collection, *ACM SIGPLAN Notices*, Vol.24, No.5, pp.38–46 (1989).
16) Wilson, P.R.: Some Issues and Strategies in Heap Management and Memory Hierarchies, *ACM SIGPLAN Notices*, Vol.26, No.3, pp.45–52 (1991).
17) Wilson, P.R.: Uniprocessor Garbage Collection Techniques, *International Workshop on Memory Management*, Bekkers, Y. and Cohen, J.(eds.), Lecture Notes in Computer Science, Vol.637, pp.1–42, Springer-Verlag, St. Malo, France (1992).
18) Wilson, P.R. and Moher, T.G.: A Card-marking Scheme for Controlling Intergenerational References in Generation-based Garbage Collection on Stock Hardware, *ACM SIGPLAN Notices*, Vol.24, No.5, pp.87–92 (1989).
19) Wilson, P.R. and Moher, T.G.: Design of the Opportunistic Garbage Collector, *Proc. OOPSLA '89*, pp.23–35 (1989).
20) 山口文彦，中西正和：演繹的手法によるプログラムの自動合成，日本ソフトウェア科学会第10回大会論文集，No.C4-1，日本ソフトウェア科学会，pp.297–300 (1993).

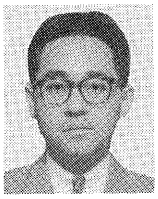
(平成8年9月13日受付)
(平成9年1月10日採録)

田中 詠子（正会員）

平成2年慶應義塾大学理工学部数理科学科卒業．平成4年同大大学院理工学研究科計算機科学専攻修士課程了．平成8年同博士課程了．工学博士．現在，同大理工学研究科計算機科学専攻中西研究室訪問研究員．Lisp，および並列処理に興味を持ち，ガーベッジコレクションを中心とするLisp処理系，および並列Lispの研究を行っている．電子情報通信学会，日本ソフトウェア科学会，ACM各会員．

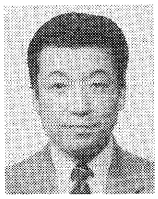
前田 敦司（正会員）

昭和61年慶應義塾大学理工学部数理科学科卒業．平成6年同大学大学院後期博士課程単位取得退学．現在，同大学数理科学科中西研究室研究生．Lisp処理系，コンパイラ，プログラミング言語理論，情報理論に関心を持つ．日本ソフトウェア科学会，ACM各会員．



田中 良夫（正会員）

昭和62年慶應義塾大学理工学部数理科学科卒業．平成元年同大大学院理工学研究科数理科学専攻修士課程了．平成7年同博士課程単位取得退学．工学博士．現在，技術研究組合新情報処理開発機構勤務．並列プログラムの性能予測や並列計算機の性能評価に関する研究に従事．並列ガーベッジコレクションに関する研究も行っている．電子情報通信学会，ACM会員．



中西 正和（正会員）

昭和41年慶應義塾大学工学部卒業．昭和44年同大工学部助手．平成元年同大理工学部教授．工学博士．昭和42年，日本初の実用Lisp処理系を作成．以後，記号処理言語，人工知能用言語などの研究に従事．昭和57年LispマシンSYNAPSEの開発など．電子情報通信学会会員，情報処理学会プログラミングシンポジウム委員会幹事長．