IPA

2008 年度下期未踏 IT 人材発掘・育成事業　採択案件評価書

1. 担当PM

竹内　郁雄 PM （東京大学大学院　情報理工学系研究科　創造情報学専攻　教授）

2. 採択者氏名

チーフクリエータ:　郷原　浩之（東京大学　工学部システム創成学科
数理社会デザインコース）

コクリエータ:　なし

3. プロジェクト管理組織

株式会社　創夢

4. 委託金支払額

2,521,600 円

5. テーマ名

オープンかつポータブルなデータベースガーベジコレクション

6. 関連Webサイト

なし

7. テーマ概要

本提案では Java で標準的なオブジェクトリレーショナルマッピング（以下、ORM）である Hibernate 上でポータブルなデータベースガベージコレクション（DBGC）を実現する。

近年、メモリデータ構造をデータベースに保存するべく ORM と呼ばれるデータベースマッピングツールが普及し始め、DB 内部のリンク構造は複雑になる一方である。

しかし、データ構造の管理に目を向けた場合、共有循環構造の削除は到達可能性予測が困難であるため課題となっている。削除を示すフラグを用いてデータを消さずに取り置く選択肢では、データが小さくことはないため、長期運用を前提とした情報システムでは不必要に高性能なデータベースを導入せざるを得なかった。また、データ流出等のリスクを開発ベンダーが取らなければならず、セキュリティー上にも問題を生じていた。

メモリ上ではグラフ削除の最も有効な方法は GC と実証されつつある.したがって分散 DB 等、今後さらに複雑なリンク構造をもつ DB システムが登場することを考えればアクセスの遅い DB 上でも高速に動作するように設計された DBGC は必須となる。

DBGC によって本来のサービスを長時間停止させてはいけない。本システムではユーザープログラムとの並列処理時に性能が良い Dijkstra の On-the-fly リアルタイム GC アルゴリズムを用いることでこの問題を回避する。GC では計算時間のほとんどがマーキングに費やされる。DB の能力を最大限まで引き出せるよう、メモリ上ではなく SQL を用いた集合演算によるマーキングアルゴリズムを提案する。またテーブル間参照グラフマトリックスの解析からマーキングパスを枝と環に分解することで SQL 自体の発行回数を最小化する。

開発物は Apache License を適用して公開し、必要十分なドキュメントを整備し、広い普及を狙う。

8. 採択理由

竹内は 10 年前、記号処理言語の並行ガーベジコレクション（GC）のプログラムを書いていた。GC は 1959 年にマッカーシーらの Lisp の開発で初めて出てきた概念であるが、その後 50 年近くも火が消えずに（ボーボーと燃えているわけではないが）研究が続けられている基幹技術である。なぜこんなに長く研究が続いているかというと、GC を行なうべきシステムやそこに含まれる性能パラメータが本当に多様で、万能 GC というものが存在しないからである。また、GC が必要な言語も増えてきた。

さて、この提案を聞いてまず驚いたのが、世の中の巨大データベースの中身の大半がゴミということだ。ゴミを保存しておくために巨大の計算機資源が必要だし、ゴミからの情報漏洩もあるという。人海戦術でなにかやろうにも、巨大なうえに間違うとゴミでないものも消してしまうことがあるらしい。じゃあ、GC というわけだが、教科書で習うような GC では通用しないことは明らかだ。郷原君の挑戦はそこにある。内容の不透明な、商用データベース GC はすでにあるようだが、郷原君は Apache ライセンスで公開するという。

これは実に芯のあるシステムプログラミング分野の提案である。とはいえ、この問題は聞くほどやさしくはない。データベースを止めないで並列的に GC をするという技術アイデアは大丈夫そうだが、実際に一般のデータベースに適用するとなると、データベース運用者の注意深い取扱いが必要になる。このプロジェクトの最終的な成否はそこをどこまでやさしく安全にできるにかかっている。郷原君の若さのもつ力に賭けよう。

9. 開発目標

ORMを使用したDBには決して参照されることがないことが保証されているデータが存在する。ORMはメモリ上のオブジェクトをそのままDBに格納するため、メモリ上ではGCによって削除されるようなオブジェクトもDBに保存される。そしてそのようなオブジェクトはデータベースの容量を無駄に占拠するだけの存在である。

本プロジェクトでは、メモリ上で広く実現されている GC を DB 上で実現することを目標とする。また特定の DB ソフトに依存せず動作するソフトウェアとする。

10. 進捗概要

当初予定していた「On-the-fly GC の実装」については、「湯淺式の実装」に変更した。湯淺式GC の方がOn-the-fly GC よりも発行すべきSQL の回数が少ないために本システムのパフォーマンス的に有利であった。

11. 成果

本プロジェクトの開発項目と到達度は以下の通りである。

(1) 完全な追跡可能経路情報収集

ガーベジコレクション（GC）ではルートと呼ばれる基幹の情報から到達可能なデータを同定する必要がある。ORM はその動作の原理上、内部に完全な追跡可能経路情報を各クラスのメタ情報として保持する。よって、Hibernate が提供する API を組み合わせることでそのメタ情報を取り出し、解析することで、完全な追跡可能経路情報を獲得できる。

これについては実装を完了した。Hibernate がサポートする多対一、一対多、多対多、継承関係を把握でき、コレクションについては Set、List、Map などの基本的なコレクションはサポートできた。

(2) マーキング SQL の生成

Hibernate は多対一をはじめ、一対一、一対多、多対多、継承、さらには遅延ローディングもサポートする。これら Hibernate がサポートするすべての関連について適切にマーキングがなされる SQL を生成する。これは Hibernate が提供する HQL と呼ばれる統合化 SQL を用いる。HQL で記述されたクエリは各データベースで実行される際に各データベースごとの SQL 方言に翻訳されるため、データベース毎に SQL を記述し直す必要がなくなる。

これについては、サポート範囲を限定し、終了した。ただし、Hibernate が HQL をデータベースに合わせた各 SQL に変換する HQL コンパイラにバグが存在するために、Map は非サポートとした。

図 2.2.1 にその概要を示した。ここで注意すべきは、既存のテーブルに GC のマーキング結果を格納するためのカラムを用意していることである。このようにしてマーキングが通常のデータベースアクセスになっているところが本システムの大きな特徴である。

なお、図ではマーキングコラムの値は true か false であるが、実際には何回目のマーキングであるかを示す「マーキング世代番号」が格納されている。

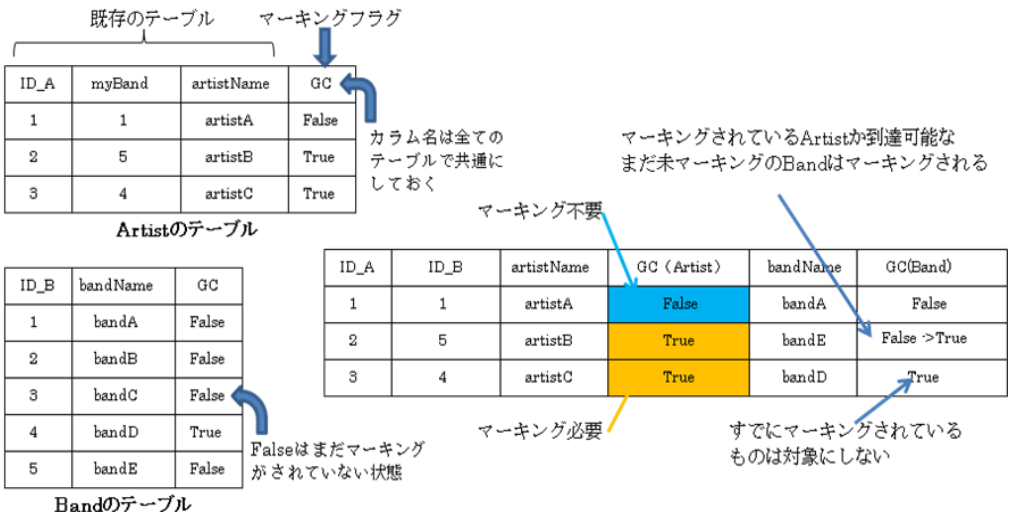

図 2.2.1 データベースにおける SQL による GC のマーキング方法とその過程

(3) SQL 発行回数最小化のためのグラフ解析

SQL はディスクアクセスを伴うため、可能な限りその発行回数を少なくしたい。ここでは循環構造の同定と、グラフ理論で用いられるトロポジカルソートというアルゴリズムを用いて SQL の発行回数が最も少なくて済むような発行順序を決定する。

これには JgraphT というグラフ解析ライブラリを用いた。また循環構造を同定したあとの SQL の発行順序決定で用いるトポロジカルソートも JgraphT を用いた。本ソフトウェアで実装した SQL の発行順序は大域的最適解ではないが、一般的なグラフ構造に対応したものとしては SQL の発行回数が十分に少ない。

ここでは詳細を説明しないが、図 2.2.2 の左のような参照構造があった場合、A からマーキングを始めるのが最もマーキングのためのアクセス重複が少ない（つまり、すでにマークしたものをさらにマークしにいく可能性が少ない —— 対象がディスクの中にあるので、これは少なければ少ないほどよい。）循環構造の扱いには注意をしないといけないが、図のように一旦かたまりとして扱い、その内部は別途方法を考えるという方策を採った。

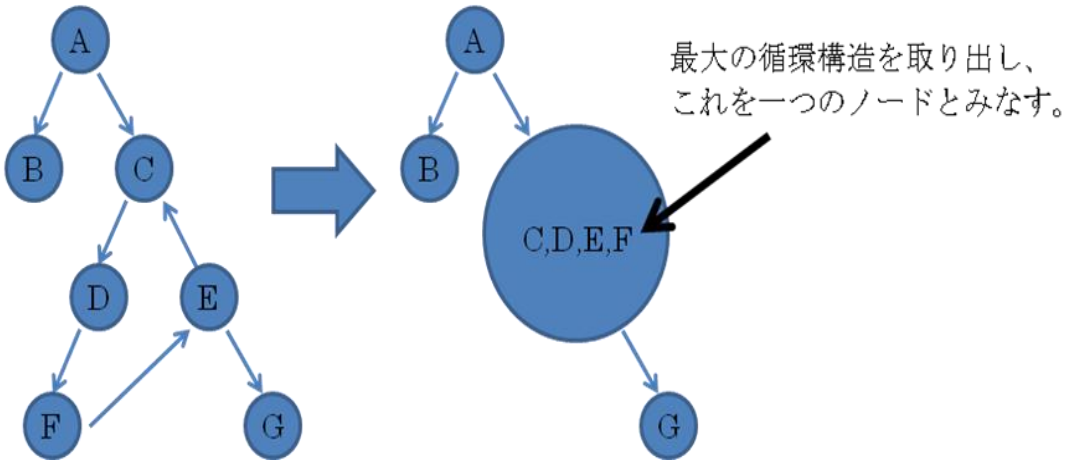

図　2.2.2 (循環構造を含む)エンティティの参照関係における SQL 発行回数最小化戦略図の概念

(4) 湯淺式 GC の実装

実際のシステムの運用を考慮したとき、データベースのサービスを停止せずに GC を行なうことが求められる。それ故に本プロジェクトではユーザプログラムと並行に動作するシステムを湯淺の Snapshot at the beginning GC を用いて実現した。なお、当初の Dijkstra の On-the-fly はアクセスが多すぎるため採用しなかった。

湯淺式 GC の実現において必要なのは GC の最中にユーザプログラムが行なった。エンティティの新規追加とエンティティのポインタの付け替えに関する情報である。これらは Hibernate のダーティチェックと呼ばれるエンティティの状態監視機構のコードを改変することで、獲得することができた。

なお、本システムでは、マーキング世代番号の概念を用いることにより、マークフェーズからスイープフェーズへの移行に関する微妙なタイミングの問題を回避し

ている。詳しくは説明しないが、図 2.2.3 異なるマーキング世代番号が見掛け上オーバーラップしているようにする巧妙な方法を採用した。

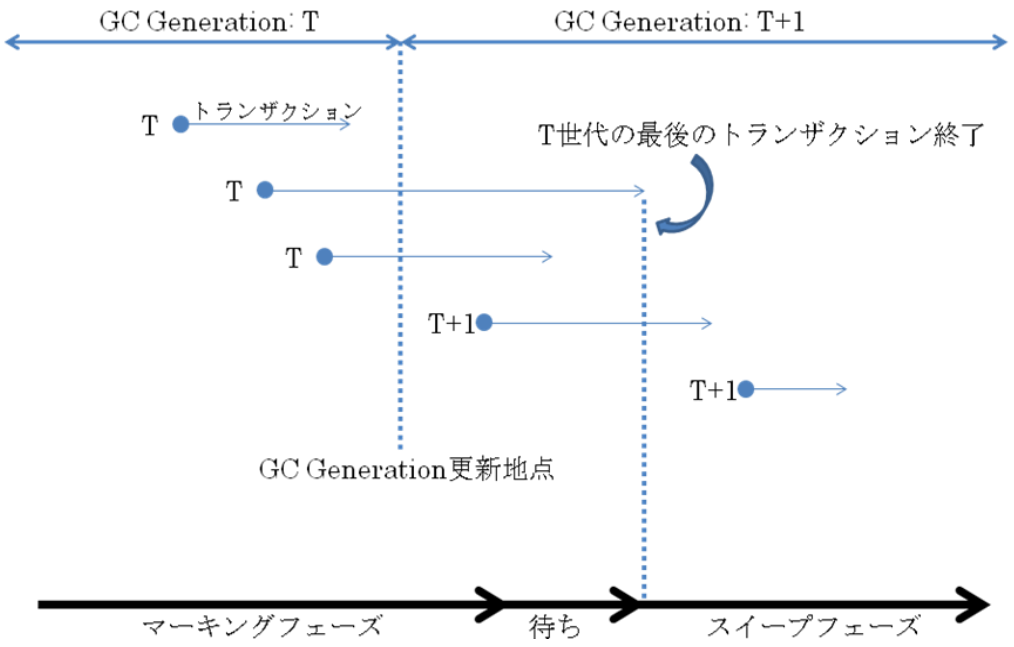

図 2.2.3 リアルタイム GC 実現のためのトランザクション管理

(5) ドキュメント作成

データベース GC システムにおいて設定の誤りはデータベースの破壊を招きかねず、またインデックスの付け方によりシステム速度が大きく異なるため、詳細なドキュメントを整備して公開する。

これについては簡易版が完成した。使用方法をグラフィカルに説明するのみならず、設定方法の詳細な記述や本ソフトウェアの動作原理を記述した。複数環境下でのテストが不十分なので、完成版には至っていないが、テストが終了次第、各データベースでの基本テストの結果やデータベース別のパフォーマンスチューニングガイドを追加し、完成版とする。

(6) 複数環境下でのテスト

ソフトウェアの性質上、正確に動作することが極めて強く要求される。広く普及しフィードバックを受けていくことが本プロジェクトを成功に導く。それ故に知識の集約を意図して、本システムは環境に非依存なものとする。環境に非依存である以上、複数環境下でのテストが必要不可欠である。HSQLDB ほか、ユーザ数の多い PostgreSQL や MySQL についてテストを行なった。

HSQLDB は基本開発環境で用いたデータベースであるため、すべてのテストにパスする。PostgreSQL はリアルタイム性の実現、MySQL は多対多関連のマーキング SQL にてそれぞれテストをまだパスしていない。MySQL については、HQL コンパイラの力不足が原因と思われる。

12. プロジェクト評価

採択理由でも述べたが、竹内は、10 年ほど前に、実時間性能の観点ではしばらく世界トップレベルの性能を誇れるような並列ガーベジコレクション（GC）のマイクロプログラムをゼロから書いた経験をもつ。なので、一応 GC のプロと自負していたものである。しかし、SQL でマーキングを行なうという郷原君の提案を知って驚いた。私の知るかぎり、前代未聞である。それと同時にこれはうまいと思った。SQL を使うかぎり、最初から通常のデータベース操作と並行して GC を行なうことがいとも容易だからである（ただし、実際の消去・メモリ回収を行なうためには、データベースの完全性制約に関する注意が必要）。データベースを止めないサービスをしてこそデータベースなのだから、並行あるいは並列に GC が行なえることは極めて重要な条件である。

関係データベースのままでは、実は上のようが芸当はできないが、オブジェクト指向データベースと関係データベースの写像を定める ORM を利用すると、通常の GC でいう到達可能性がうまく検出できる。これもアイデアである。

と、まぁ、大枠ではこれはうまく行きそうだと思ったのであるが、案の定いろいろな伏兵が潜んでいた。プロジェクト期間中、郷原君はこれらや思わぬシステムバグに悩まされながらも、予想以上に早く実用的な形になるところまで仕上げてくれた。竹内も昔取った杵柄で、ときどき一緒に考えさせてもらい、（多分、有用だったはずの）アドバイスもした。そういう意味で、竹内の血も少し湧き上がった。昔を思い出させてくれた郷原君に感謝したい。

それはともかく、このデータベース GC（DBGC）はいろいろな意味で面白い。基本的にディスクの中のデータの GC なので、GC のためのディスクアクセスを減らすことが重要である。そうしないと、通常のデータベースアクセスの性能を大幅に落してしまう可能性がある。当初、郷原君はそのためにエンティティ関係の構造分析にえらく力を入れていた。ところが計画段階で採用予定だった Dijkstra の On-the-fly は、その努力を帳消しにしかねないものであった。こっちのほうで無駄な SQL が大量に発行されてしまう。早い段階で湯淺の方式に変更したのは大正解である。

プロジェクトが終盤に差しかかったころ、マーキング世代番号のアイデアが打ち出されてきた。一瞬、世代別 GC のことかと思ったが違った。これも面白いアイデアだ。データベースなのでマーキングのために、十分な大きさのコラムを使ってもいい。だから、こういうアイデアも十分にありなのだ。私も郷原君と一緒にこのアイデアの正当性に

ついて考えさせてもらった。うーむ、血が湧く。この方法だと、回収（スイープ）はマーキングと、なんと並列に実行できるのだ。これまた前代未聞である。空いているときに、いつでもゆっくりと回収することができる。また、ゴミはゴミと明示して残しておくことも可能である。これは意外に実用的だ。それでいいというユーザも多そうだ。また、2 年以上前にゴミになったものだけを回収するといったことも可能だ。

上にも述べたように、このプロジェクトは当初実はかなり心配していたのであるが、プロジェクト期間の終わりまで 2 ヵ月ほど余して、急に仕上りが見えてきた。素晴らしいことである。その余裕もあって、お約束通り、わかりやすいドキュメントも出来上がってきた。

このソフトを実際に本物のデータベースで使えるようになるためには、慎重なうえにも慎重な検証を重ねる必要がある。ついでにHibernate の虫を発見してしまったので、それを訴える必要もある。しかし、基本ができたので、あとは時間と労力の問題であろう。この DBGC のアイデアは、郷原君がアルバイトをしている(株)情報基盤開発の中で、いたずらに脹れ上がるデータベースをどうしようかという問題からニーズが出てきたものである。つまり、机上の議論から出てきたソフトではない。ちなみに、(株)情報基盤開発は大昔に竹内が未踏ユースに採択した学生たちがつくって、最近結構ビジネス的に成功しているベンチャー企業である。まさに因果は巡る、である。

郷原君は、このソフトを近日中に Apache ライセンスで公開する。それも重要だが、この成果にしっかりとした性能評価を加え、さらに若干の理論的考察を行なって、国際的な場で発表すべきだと思う。GC 好きの竹内としては太鼓判を押したい。

13. 今後の課題

現在は、本ソフトウェアの機能の一部を Hibernate 本体に取り込むための活動を行っている。個人的なつながりがあるシステムで運用実績を稼ぐための準備をしている。