3K-8

データの更新を漸次的に行う動的ソフトウェア更新機構

荻山 温夫† 小宮 常康† 佐藤 喬† 多田 好克†
電気通信大学 大学院情報システム学研究科‡

あらまし

動的ソフトウェア更新機構は，実行中のプログラムを新しいバージョンに更新することを可能にする．データを生成する関数(コンストラクタ)が更新によって変更された場合，古いコンストラクタの生成したデータ(旧データ)を，新しいバージョンのソフトウェアが扱えるようにする必要がある．すべての旧データを新しいバージョンで扱うデータ構造（新データ）に一括で置き換えると，プログラムの動作が長時間停止することがある．そこで，この置き換えを漸次的に行う方式を提案する．提案方式では，データの参照には必ず専用の関数(アクセサ)を用いるプログラムを前提とし，アクセサがデータの置き換えを行う．また，プログラムの扱う全ての旧データを置き換えるために，ごみ集めの実行中に旧データを見つける処理を追加した．見つかった旧データは，関数の呼び出しが行われるタイミングで一定数ずつ置き換えが行われる．

1 背景

動的ソフトウェア更新(Dynamic Software Updating, DSU)機構は，実行中のプログラムを終了させることなく，新しいバージョンのプログラムに更新することを可能にする．

これまでに開発された DSU 機構の典型的な利用形態を図 1 に示す．まず，更新対象プログラムの開発者は，更新プログラム生成ツールを用いて，更新プログラムを作成する．更新プログラムは，更新対象プログラムの更新を行うために必要な処理および情報を含んでいる．更新対象プログラムの利用者は，更新プログラムを更新対象プログラムに適用し，更新を行う．更新対象プログラムは，更新プログラムを読み込み，更新プログラムで定義された，更新対象プログラムで定義された変数を更新する処理と，データ構造を更新する処理を必要なときに呼び出す．

プログラムの更新によって，コンストラクタやデータ構造の定義が変更される場合，更新前のバージョンで生成されたデータ(旧データ)を，新しいバージョンのプログラムが扱えるようにしなければならない．これまでの DSU についての研究で用いられてきた，旧データの置き換えのための主な方式 2 つを紹介する．1 つは Ginseng[1]などで用いられている，旧データを参照する処理が行われる際に，旧データを新データに更新する方式である．この方式では，データの更新をすべて完了できない場合がある．そのため，データの内部を参照する処理は，データの更新処理を行う複雑なものを常に用いなければならず，プログラムの実行時間の増大(オーバヘッド)がもたらされる．もう 1 つは JVOLVE[2]などで用いられている，更新対象プログラムの各定義を新しいバージョンのものに置き換える際に，すべての旧データを一括で新データに更新する方式である．この方式では，データの更新処理により，プログラムの実行が長い間停止され続ける場合があり，この場合，プログラムの応答性が損なわれる．

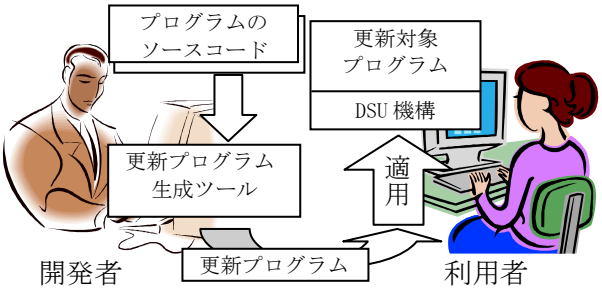

図 1 DSU 機構の利用形態

2 提案手法

本研究では，プログラムの一回の停止時間を縮減するため，データの更新を行う処理を一括ではなく，複数回に分割して行う方式を提案する．

この方式では，旧データを一定数ずつ更新する，漸次的なデータの更新を行う．漸次的なデータの更新では，一定の時間間隔ごとにプログラムの扱うヒープ領域から旧データを探し，見つかった旧データを更新する処理を一定数の旧データごとに分割して行う．分割したデータの更新処理の 1 回で更新する旧データの数の上限を設けることで，データの更新によるプログラムの連続停止時間を小さくできる．

旧データが残っている状態で更新対象プログラムが実行されるため，プログラムが旧データを参照する際に，実行時エラーが起こる恐れがある．これを避けるため，アクセサにアクセスバリアを追加し，アクセサの引数のデータ構造が旧データである場合，アクセサが本来の処理を行う前に，旧データを更新するようにする．

漸次的なデータの更新では，ごみ集め(GC)を用いることで確実に全ての旧データを更新し終えること

Dynamic Software Updating for Supporting Incremental Data Replacement.
†Atsuo OGIYAMA, Tsuneyasu KOMIYA, Takashi SATOU, Yoshikatsu TADA.
‡Graduate School of Information Systems, The University of Electro-Communications.

ができ，データの更新が全て完了したことを確認することができる．データの更新が完了した際には，アクセスバリアを挿入されたアクセサからアクセスバリアを取り外し，新データのみを扱うアクセサに置き換えることができる．これによって，データの更新が完了した後では，アクセスバリアによるオーバヘッドを無くすことができる．

3 DSU 機構の実装

本研究では，Scheme プログラミング言語処理系 TUTScheme[3]に変更を加え，DSU 機構を実装した．TUTScheme は，バイトコードに変換されたプログラムを逐次解釈することで，プログラムを実行する．

DSU を行うためには，更新プログラムを読み込む処理と，変更される定義を更新する処理を適切なタイミングで行う必要がある．これらの処理を，TUTScheme のバイトコードの命令 xcall を解釈する処理に追加した．xcall は関数の呼び出し命令であり，どのようなプログラムでも頻繁に実行される．そのため，xcall 命令の解釈処理に追加された処理は，プログラムの実行中に，必要な時に確実に実行することができる．

漸次的なデータの更新の実装のため，TUTScheme のコピーGC の処理に，コピーされるデータが旧データであるかを判別する処理を追加した．この判別は，データ構造の，生成された時点のプログラムのバージョンを表すデータスロットが，旧データのものであるかを調べることで行う．GC を用いることで，全ての旧データを発見できる．発見された旧データは，旧データへの参照を保存するための専用のリストに格納する．その後の xcall 命令の実行時に，このリストに格納された旧データを更新する．xcall 命令が一定回数実行されるごとにデータの更新処理を行うようにすることで，データの更新を行う時間間隔を設定できるようにした．

データの更新が完了した際にアクセスバリアを取り外す機能は，以下のように実装した．コピーGC の実行時に旧データが 1 つも発見されなかった場合，その次に xcall 命令の実行時にアクセスバリアを取り外す処理が呼び出されるよう，コピーGC の処理と，xcall 命令の解釈を行う処理に変更を加えた．

4 性能評価

漸次的なデータの更新を行う際の，1 回で更新するデータの数の上限を変更した場合の，プログラムの連続停止時間と合計処理時間の変化を調べるため，実験を行った．

この実験では，旧データとなる 10^5 個のデータ構造をあらかじめ生成しておき，プログラムの更新を適用した．プログラムの各定義が更新された後，あらかじめ生成されたデータ構造の参照を 10^6 回行う処理を実行した．この時，処理と並行してデータの更新が行われる．この処理の実行時間の合計と，

表 1 比較実験の結果

	D_{max}=100,000	D_{max}＝1,000
合計処理時間比	104.7%	105.2%
連続停止時間	203[ms]	1.58[ms]
更新処理間隔	-	70.4[ms]

データの更新による連続停止時間を測定した．データの更新は，xcall 命令が 10^5 回実行されるごとに行うよう設定した．

この測定を，1 回で更新するデータの数の上限 D_{max} を 100,000 個および 1,000 個に設定した場合について行った結果を表 1 に示す．表 1 では，合計処理時間を，プログラムの更新を行わずに処理を行った時間と比較した割合で示した．一度に更新するデータの数の上限を 1,000 個に設定した場合，連続停止時間を 1.6 ミリ秒程に減らすことができた．この時，データの更新処理を行う間隔は 70 ミリ秒程であった．一方，合計処理時間は，プログラムの更新を行わなかった場合に比べて 5%程長くなった．

5 関連研究

Subramanian らは Java 仮想マシン Jikes RVM に変更を加え，コピーGC を行う際に新しいバージョンで変更されるクラスのインスタンスを更新する機能を追加し，DSU 機構 JVOLVE[2]を実装した．この機構では通常実行時の性能にオーバヘッドをほとんどもたらさずに DSU を実現する．JVOLVE はソフトウェアの更新時にソフトウェアの扱うオブジェクトを一括で更新する．そのため，オブジェクトの更新処理によってソフトウェアの実行は一時停止される．ソフトウェアが巨大なヒープ領域を扱う場合，停止時間は数秒に及ぶことがあると報告されている．

6 まとめ

本研究では，DSU におけるデータ構造の更新処理を漸次的に行う方式を提案し，この方式を用いる DSU 機構を実装した．性能評価では，1 回で更新するデータの数の上限を設定することで，データの更新によるプログラムの連続停止時間を小さくできることを示した．

謝辞 本研究の一部は，科学研究費補助金(20500028)の補助を受けている．

参考文献

[1] I. Neamtiu et al., Practical Dynamic Software Updating for C, in Proc. 2006 ACM SIGPLAN Conf. PLDI, pp.72-83, June 2006.
[2] S. Subramanian et al., Dynamic Software Updates: A VM-centric Approach, in ACM SIGPLAN Notices Vol.44, Issue 6, pp.1-12, June 2009.
[3] 京都大学 湯淺研究室, TUTScheme, http://www.yuasa.kuis.kyoto-u.ac.jp/~komiya/tus_intro.html.