オブジェクト参照の単一アトミック更新による不揮発性メモリのための Java ArrayList の整合性保証

田 井 直 也　【 ソフトウェア検証・解析学研究室 】

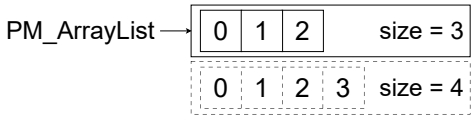
1　はじめに

不揮発性メモリ (Non-Volatile Memory, NVM) は，通常の RAM と同様に主記憶として使用でき，かつ電源を喪失するなどしても内容が失われない記憶装置である．NVM を利用することで，プログラム実行途中で電源喪失等しても停止直前の状態から実行を再開できるシステムを構築できる．しかし，キャッシュメモリが介在することによる不整合などを防ぎながら正しくプログラムすることは，開発者の大きな負担になる．これに対し，プログラム処理系のメモリ管理機構に NVM 用の機能を持たせて開発者の負担を軽減する手法が研究されており，その一つに，松本らによる NVM 用 Java 仮想機械 [1] (以下，NVM–JVM) がある．この仮想機械では，指定された変数から到達可能なオブジェクト群が自動的に NVM に，アクシデント後の復元の際に不整合（参照先のないポインタ等）を生じないように保存される．しかし，この機能は 1 フィールドを更新する際の不正なメモリ状態を防ぐためのものであり，不可分な複数の更新 (トランザクション) は考慮されない．そのため，ArrayList 等の標準ライブラリで提供されるデータ構造では更新途中の停止により一部のみ更新された不整合なデータが復元されてしまう．

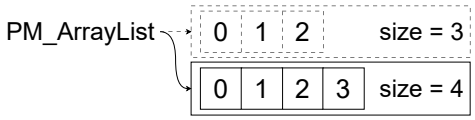
本研究では，プログラマがトランザクションを意識せずに使用でき，予期せぬ停止時にも不整合を生じない ArrayList の代替実装（PM_ArrayList）を開発する．本手法は，Haria らによる MOD[2] で提案された，既存データを直接書き換えずに新しいコピーを作成するシャドウページングを参考にしている．具体的には，配列の実体と要素数 (size) を集約したオブジェクトの参照をアトミックに更新することで整合性を保証する．NVM–JVM の単一フィールド更新保証を活用し，参照の付け替えのみで常に整合した状態を保つことが可能である．

2　実現方法

PM_ArrayList は，配列の実体と要素数 (size) を保持するコンテナオブジェクトを定義し，参照の付け替えによって不整合を防止する．更新手順を図 1 に示す．まず，既存データを書き換えずに，新しい内容を持つ別のコンテナオブジェクトを別途作成する (図 1 (1))．次に，PM_ArrayList が保持するコンテナへの参照を，作成したオブジェクトへ一括して付け替える (図 1 (2))．NVM–JVM の単一フィールド更新保証により，この付け替えのみで「更新前」か「更新後」の整合した状態を常に保つことが可能となる．



(1) 新たなコンテナオブジェクトの作成



(2) アトミックな参照の付け替え

図 1 PM_ArrayList の更新手順

3　実験結果

NVM–JVM を用いて，プログラムの強制終了に対して整合性が維持されることを確認するテストを行なった．

標準の ArrayList と PM_ArrayList それぞれを使用し，要素の追加と削除を繰り返すループプログラムを実行した．実行中に `kill −9` コマンドを用いてプロセスを強制終了させ，再起動後にデータ構造の整合性 (配列の要素数と size 変数の値が一致しているか) を確認する作業を 100 回行い，これを 1 セットとした．

このテストを各 1,000 セット (合計 100,000 回の強制終了) 実施した結果，標準の ArrayList では全てのセットで不一致が発生した．これに対し，PM_ArrayList では全てのセットにおいて不整合が発生せず，正常にデータを復元できることを確認した．

一方，実行性能については，配列の再生成のため $O(1)$ の操作が $O(n)$ となる．`−Xint` オプションを用いた評価では，`add(E e)` を 10,000 回実行した場合に標準実装に対し約 30 倍の実行時間を要することを確認した．

4　まとめ

本研究では，NVM 環境での停止時にも整合性を維持できる PM_ArrayList を開発した．参照のアトミック更新により複雑な整合性保証を簡潔に実現し，100,000 回のテストを通じてその論理的妥当性を実証した．今後は他データ構造への適用と性能最適化が課題である．

参考文献

[1] 松本 康太郎, “不揮発性メモリを用いた複製に基づく Java オブジェクトの永続化”, 高知工科大学 修士学位論文, 2023.

[2] S.Haria, et al., MOD: Minimally Ordered Durable Datastructures for Persistent Memory, ASPLOS ’20, pp.775–788, 2020.