A-009

不揮発メモリ搭載システムにおける Python のデータ永続化
Supporting data persistence in Python for non-volatile memory

神庭 弘樹† 山崎 憲一†
Hiroki Kaniwa Kenichi Yamazaki

1 はじめに

不揮発メモリはメインメモリとして扱うことが可能な不揮発性を有するメモリである．このメモリによってデータの保存と再利用開始の時間を大幅に短縮できる．このような特性を容易に活用するためには，プログラミング言語のレベルでのサポートが必須である．そこで，本研究では Python において，組み込み型のオブジェクトの永続化，復元を容易に実現可能にする機能を提案する．

2 関連研究

不揮発メモリを利用したアプリケーション開発のためのプログラミング言語やライブラリの研究が活発に行われている．永続データのための Scala ライブラリの研究 [1] では，永続データの書き換えが可能な形での永続化が実現された．しかし，永続データを再利用できるプログラムが限られるという問題がある．

スクリプト言語のデータ永続化の研究 [2] では，API による容易な永続化，復元が実現された．また，永続データを保存するファイル内ではオフセットを用いてデータの位置を特定することによって，高い移植性が実現された．しかし，永続化できるデータ型が限られるという問題がある．

本研究では組み込み型のオブジェクトのファイルへの永続化を可能とすることで関連研究にある制限を緩和し，容易にデータの永続化，復元が可能な機能の実現を目指す．

3 提案

本研究では DRAM をメインメモリとする従来の計算機にバイトアクセス可能な不揮発メモリをメインメモリとして追加することを考える．その場合，ポインタの扱いに注意する必要がある．DRAM 上のデータを指すポインタを永続化すると，プログラム停止時にデータのみが失われ当該ポインタが無効なポインタとなるためである．この他にも，不揮発メモリ上のファイルのマッピング位置が変わることによるアドレス変化によって無効なポインタが発生する．そのため，先行研究 [2] と同様に不揮発メモリ上ではオフセットによってデータの位置を特定する．

永続化と復元を実現するために，永続ファイルと永続データ型の 2 つを導入する．これらによってオブジェクトの状態やデータを管理し，永続化や復元，永続化されたオブジェクトの更新，障害発生による矛盾した状態からの回復を実現する．これらの操作は関数として提供される．

3.1 永続ファイル

永続ファイルは不揮発メモリ上に作成されるファイルである．永続データを利用する際には 1 つの永続ファイルを指定し，これをメモリの連続した空間にマッピングする．

† 芝浦工業大学大学院理工学研究科 Graduate School of Engineering and Science, Shibaura Institute of Technology

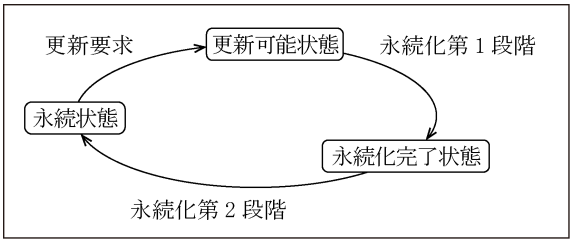

図 1 永続データ型の状態遷移図

このファイルではオブジェクト管理構造と更新管理構造の 2 つの管理構造によって永続オブジェクトを管理する．オブジェクト管理構造は有効なオブジェクトを把握するための構造であり，探索時に基準となるオブジェクトであるルートオブジェクトとその識別子をもつ．識別子とオブジェクトの組はアクセスの起点としてプログラムに公開される．更新管理構造はオブジェクトの状態を把握するための構造であり，オブジェクトと当該オブジェクトの状態を表す 3 種類のフラグによって状態を管理する．

3.2 永続データ型

Python の組み込み型のうち，内部表現にポインタを含まないものはそのまま永続化可能であるが，含むものはそのままでは永続化できない．前者のデータ型は int 型，float 型である．後者のデータ型については，本研究にて各データ型に対応する永続データ型を提供し，これに変換することで永続化を可能とする．このデータ型として dict 型，list 型がある．

永続データ型は図 1 にある 3 つの状態をもつ．永続状態は永続化時点のデータのみをもつ状態である．永続状態ではデータの更新は行えず，実行可能な操作がデータの取得などの更新を伴わない操作のみに限定される．更新可能状態は永続化時のデータと更新中のデータの 2 種類のデータを同時にもつ状態である．前者のデータは不揮発メモリ上に，後者のデータは DRAM 上に置かれる．更新中のデータが最新のデータであり，実際の参照では DRAM 側のみが用いられる．ただし，回復時には永続化時のデータを参照し 1 つ前の永続状態へと戻す．永続化完了状態は永続化処理をしている途中にとる状態である．更新可能状態と同様に永続化時のデータと更新中のデータの 2 種類をもつ．ただし，両方のデータが不揮発メモリ上に存在する．この状態では回復先を 1 つ前の永続化時の状態と永続化後の状態から選択できる．

永続化では更新可能状態が持つ更新中のデータを永続化して永続化完了状態にし (永続化第 1 段階目)，当該データで元々の永続化時のデータを上書きして永続状態にする (永続化第 2 段階目)．このように永続化を 2 つの段階に分けて実行する理由を述べる．永続化は不揮発メモリ上の更新された永続オブジェクトと，当該オブジェクトから到達可能な DRAM 上のオブジェクトを対象に行われる．つまり，1 回の永続化で複数のオブジェクトが永続化される．

そのため，永続化を 1 段階で行うとあるオブジェクトは永続化後，他のオブジェクトは永続化前という状況が起きうる．この時に障害が発生すると，その後の回復処理によって永続化後の状態になるオブジェクトと 1 つ前の永続状態に戻るオブジェクトが混在し，矛盾した状態となる．これを防ぐために永続化は 2 つの段階に分けて行われる．回復処理において，１段階目のデータは永続化前の状態，２段階目のデータは永続化後の状態にすることで矛盾が生じなくなる．

永続化がどちらの段階かは更新管理構造内の完了フラグによって判断できる．完了フラグはオブジェクトの永続化第 1 段階が完了すると真になるフラグである．したがって，更新管理構造に登録されているすべてのオブジェクトの完了フラグが真の場合は永続化第 2 段階であり，そうでない場合は永続化第 1 段階である．

3.3 永続データ型 NVdict のデータ構造

ここでは dict 型に対応する永続データ型である NVdict のデータ構造と 3 つの状態の実現方法について述べる．NVdict は図 2 に示すように dict 基部，インデックステーブル，エントリテーブルの 3 つで構成される．Python の dict 型ではインデックステーブルとエントリテーブルは分かれておらず 1 つの構造体であるが，NVdict では DRAM へのコピーを最小限にするために分割される．

dict 基部は NVdict 内の要素にアクセスする際に起点となる構造である．NVdict の状態を表す値を持ち，この値によって NVdict が永続状態，更新可能状態，永続完了状態のいずれの状態かを判別する．この他に，要素数やインデックステーブルの位置をもつ．要素数やインデックステーブルの位置には 2 種類あり，それぞれ永続化時のデータ，更新されたデータに対応している．

エントリテーブルはデータ本体を管理する構造である．Python の dict 型と同様にキー，キーのハッシュ値，値の 3 つを 1 つのエントリとして配列の先頭から順に追加していく．あるキーをもつエントリが何番目にあるかはインデックステーブルによって特定する．

インデックステーブルはエントリを管理するハッシュテーブルである．大きさや格納可能数，エントリテーブルがもつエントリ数，エントリテーブルの位置をもつ．また，キーのハッシュ値とエントリの格納位置の対応表である配列をもつ．

NVdict ではこれらの構造を複数組み合わせることで永続データ型の 3 つの状態を実現する．永続状態では dict 基部，インデックステーブル，エントリテーブルをそれぞれ 1 つずつもつ．更新可能状態では，図 2 に示すようにインデックステーブルとエントリテーブルの組を 2 つもつ（厳密には 2 つのインデックステーブルが不揮発メモリ上のエントリテーブルを共有する状態も存在する．）．永続化完了状態も更新可能状態と同様の組み合わせをもつが，両方とも不揮発メモリ上にある点が異なる．

なお，もう 1 つの永続データ型である NVlist も NVdict のように 3 つの状態を実現させ，正しい回復を実現しているが，これについては本稿では省略する．

3.4 永続データのプログラミング

永続データは 6 つの関数で操作する．nv_mmap は永続ファイルの初期化を行う．この関数によって永続ファイルがマッピングされ利用可能となる．その際，更新管理構造を確認し，前述の完了フラグにより永続化が完了していないと判断された場合は正常な状態へと回復させる．nv_munmap は永続ファイルの終了処理を行う．nv_add は指定したオブジェクトの永続化対象への登録を行う．当該オブジェクトはルートオブジェクトとして永続ファイルに追加され，実行時に指定した識別子によるアクセスが可能となる．nv_persist は永続化対象として登録されているすべてのオブジェクトの永続化を行う．nv_root は永続ファイルに存在するルートオブジェクトの一覧取得を行う．nv_delete は指定したルートオブジェクトの削除を行う．

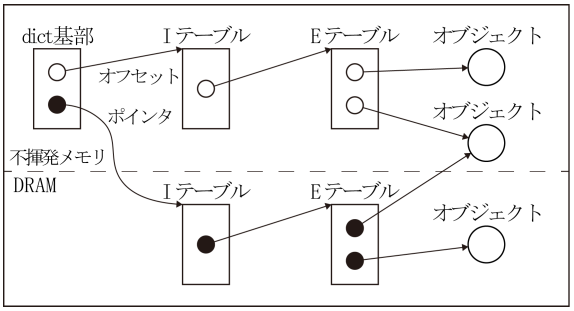

図 2　NVdict の構造 (更新可能状態の例)

4 実装方法

本研究の機能は拡張モジュールとして実装する．また，永続データ型が呼び出す Python の標準関数も若干の変更が必要となる．永続ファイルについて，本来は専用ファイルシステム上に実現すべきだが，今は不揮発メモリの特定番地を疑似的にファイルとみなしている．永続ファイルを操作する関数は拡張モジュールにて定義する．なお，現時点では同時に利用できる永続ファイルは 1 つのみだが，オブジェクトが属する永続ファイルを特定する機能を追加することで複数ファイルの同時利用は可能である．永続データ型は Python の組み込み型と同様に PyObject 型の拡張として実装する．これによって Python の組み込み型と同様に扱うことが可能となる．ただし，Python 標準の GC では永続オブジェクトを解放できないため，永続ファイル内のオブジェクトのみを対象とした GC によってルートオブジェクトから到達可能なオブジェクト以外を解放する．このような実装によって Python の実装を大きく変更することなく Python のデータ永続化と復元を実現する．

なお，本研究の機能はマルチスレッドによる実行には対応していない．

5 おわりに

本稿では永続ファイルと永続データ型によって，Python 組み込み型オブジェクトの永続化と復元を実現する方法について考察した．また，関数によって既存のコードへのわずかな変更のみで本研究の機能を利用可能とした．今後は提案手法を実際に Python に導入し，永続化や復元にかかるコストや耐故障性を評価する必要がある．

参考文献

[1]明畠 利樹, 山崎 憲一,“高速不揮発メモリ搭載システムにおける永続データのための Scala ライブラリ”, 電子情報通信学会論文誌 D, Vol.J100-D, No.11, pp.907-916 (2017) .

[2]今村 智史, 吉田 英司,“Persistent Memory を用いたスクリプト言語のデータ永続化および復元処理の実装と評価”, コンピュータシステム・シンポジウム論文集, Vol.2020, pp72-77 (2020) .