4G−3

データ構造上の並列度*

金山二郎, 三浦孝夫†
産能大学‡

1 前書き

データオブジェクト上で複数の操作要求が起こった場合、各処理を構成する操作の時間的実行順序が問題になる。本稿では、一貫性次数を元に、主なデータ構造について抽象データ型 (Abstract Data Type :ADT) を定義し、その性質を調査する。この調査により、データ構造上の並列操作における並列性が、データ構造自身の性質にある程度依存することを明らかにする。また、データ構造の内部構成や要素数によっても並列性が著しく異なることを示す。さらに、並列性の規則を加減した場合、データ構造を物理的に破壊する可能性がある ADT しか作れないデータ構造があることを示す。

2 トランザクション

データオブジェクトに対する意味的にまとまった処理をトランザクションという。また、データオブジェクトに対する基本的な手続きの単位をアクションという。

トランザクションは1つ以上のアクションの列から構成される。アクションは、処理系の命令や手続き、関数などの組み合わせによって作られる。アクションは時間的に分割して実行できない最小の単位とする。

本稿では、トランザクションは、データオブジェクトを正しい状態から別の正しい状態に遷移させる機能を果たすと仮定する。綴じ込まれて (インターリーブ) 実行されたトランザクションが、1つづつ逐次に実行されるものと同じデータオブジェクトを残すことを一貫性という。一貫性を満たすため、実行を開始したトランザクションは最終的にそのすべてが実行されたか、なにも実行されなかったかのどちらかでなければならない。この規則をアトミック性という。

アトミック性において、トランザクション中のすべてのアクションと、値のディスクへの書き込みが終了した状態をコミット状態という。また、データオブジェクトが、トランザクションが実行される直前の値に戻された状態をアボート状態という。トランザクションが実行中の、コミット状態でもアボート状態でもない状態を実行状態という。各トランザクションが障害時にアボート状態への遷移を保証されている時、これを回復可能という。

これらのトランザクションの管理はすべてOSもしくはアプリケーションシステムが行なうものとする。

トランザクションを並列動作させるための施錠に関する規則として、一貫性次数の概念 [Gray 1976] を用いる。

トランザクションがデータオブジェクトに値を書き込んでからコミットするまでの間、そのデータオブジェクトのことをダーティデータという。このとき、一度読んだデータが何度読んでも同じ内容であるとき、これを再現可能という。また、無矛盾読み込みとは、ダーティデータを決して読まないことを言う。

トランザクションの一貫性次数が3であるとは、一旦読んだデータは再現可能で、しかもダーティデータへの書き込みをトランザクションがコミットするまで許さないときとする。このクラスは、一般に直列可能であると言われる。同時に2とは、ダーティデータへの書き込みは許さず、無矛盾読み込みではあるが、再現可能ではないときとする。次数1とは、再現可能であるが、ダーティデータへの書き込みは許されないときを言う。このときダーティデータが読み込まれる可能性がある。

3 データ構造

3.1 スタック構造

スタック構造は、L に対する要素の追加、削除、参照が片一方の端からしか行えないものである。スタックには、読み込みアクションとして Top,Empty、書き込みアクションとして Push,Pop が考えられる。

これを配列を用いて考える場合、配列 $STACK[\,]$ とスタックの先頭 (データオブジェクトの入っている最後の位置) を指す変数 $TAIL$ により表現する。

次数3の場合には、書き込みのアクションは長施錠の共有施錠を発行する。また、読み込みのアクションは長施錠の共有施錠を発行する。長施錠を有効にさせるためには、すべてのアクションが参照する $TAIL$ に施錠を発行する必要がある。次数3においては、すべての施錠は長施錠であるため、Top,Empty アクションの列にしか並列性は適用されない。

次数2の場合、共有施錠を短施錠に変更する。次数3と比べると、Top,Empty が行なわれた後のデータオブジェクトに対する Push,Pop が可能になった分だけ並列性は向上する。

次数1の場合だと、共有施錠をなくす。次数2と比べると、ダーティデータに対する Top,Empty を可能にしただけ、並列性は向上する。

1方向ポインタの場合には、1方向ポインタ $STACK(val, next)$ とスタックの先頭を表わすポインタ $TAIL$ により表現する。

1方向ポインタにおいても，施錠は先頭を指すポインタ $TAIL$ になされる。したがって，各次数の並列性は配列の場合と全く変わらないといってよい。

このように、配列によるスタックにおいては、スタックの先頭を指す変数に対して集中的に施錠が行なわれるので、高い並列性は得られない。1方向ポインタについてもほぼ同じ並列性であると考えられる。

3.2 キュー構造

キュー構造は L に対する操作として、一方の端から要素の追加を、そしてもう一方の端から削除と参照を許すものである。キューには読み込みアクションとして Front,Empty、書き込みアクションとして EnQ,DeQ が考えられる。配列の場合には、配列 $QUEUE[\,]$ と、

*Concurrency level on Data Structures
†Jiro KANAYAMA,Takao MIURA
‡SANNO College

キューの先頭と最後の位置を指す変数 $FRONT, LAST$により表す。1方向ポインタの場合には、1方向ポインタ $QUEUE(val, next)$ と、ポインタの最初と最後を指すポインタ $FRONT, LAST$により表す。

次数3の場合、書き込みアクションには、長施錠の排他施錠を発行する。また、読み込みのアクションには、長施錠の共有施錠を発行する。長施錠を有効にさせるためには、すべてのアクションが参照する $FRONT$または $LAST$のどちらかに対して施錠を発行しなければならない。しかし、データオブジェクトが1つしかない場合、DeQにおいては $FRONT$と $LAST$の両方に対して施錠を発行しなければならない。次数3においては、すべての施錠は長施錠であるため、$FRONT$もしくは $LAST$にいったん施錠が発行され、その施錠が共有施錠であるときのみ、その後の Front,Empty に対して並列性が適用される。

次数2の場合だと、共有施錠を短施錠に変更する。次数3と比べると、Front,Empty が行なわれた後のデータオブジェクトに対するDeQ,EnQ が可能になった分だけ並列性は向上する。

次数1の場合は、共有施錠をなくす。次数2と比べると、ダーティデータに対する Front,Empty アクションを可能にしただけ、並列性は向上する。

1方向ポインタについてはスタックの時と同様，配列の場合と本質的に変わらない。配列の場合と同じく，施錠は先頭を指すポインタ $TAIL$ もしくは最後を指すポインタ $REAR$になされる。したがって，各次数の並列性は配列の場合と全く変わらないといってよい。

3.3 リスト構造

リスト構造は、L に対して任意の位置の要素を削除、参照、変更でき、任意の位置に対して追加が可能なものである。リストには読み込みアクションとして Locate,Retrieve,Next,Previous,First,Endlist、書き込み操作として Insert,Delete,Replace が考えられる。

まず、配列の場合には、配列 $LIST[\ ]$ とリストの先頭 (データオブジェクトの入っている最後の位置) を指す変数 $TAIL$ により表現する。

次数3の場合、読み込みアクションには、長施錠の共有施錠を発行する。このうち Locate は、Insert もしくは Replace により、トランザクションが実行状態にある間に値 x の最初の位置が変更される可能性があるので、Locate で探し出された位置から前方のデータオブジェクトすべてに対して施錠を行なわなければならない。また、Next,Previous,First,Endlist はデータオブジェクトの値ではなく位置を返すが、Insert,Delete によってその位置がなくなる可能性があるため、位置施錠を発行しなければならない。Replace,Insert,Delete は書き込みアクションなので、長施錠の排他施錠を発行する。このうち Insert,Delete はリスト全体のデータオブジェクトの数を変更するので、シフト動作が起こるため、シフトされるデータオブジェクトすべてに対して施錠しなければならない。Replace は既存のデータオブジェクトの値を変更してリスト全体のデータオブジェクトの数に影響を与えないので、1つのデータオブジェクトを施錠するだけでよい。次数3においては、すべての施錠が長施錠であり、シフト動作を含む場合もあるため、並列性は著しく低い。

次数2の場合、共有施錠を短施錠に変更する。次数3と比べると、Locate,Retrieve,Next,Previous,First,Endlist が行なわれた後のデータオブジェクトに対する Insert,Delete,Replace が可能になった分だけ並列性は向上する。

次数 1 の場合、共有施錠をなくす。次数 2 と比べると、Insert,Delete,Replace が行なわれた後のデータオブジェクトに対する Locate,Retrieve,Next,Previous,First,Endlist を可能にしただけ、並列性は向上する。

2方向ポインタの場合には、2方向ポインタ $LIST(next, val, last)$ とリスト先頭と最後を指すポインタ $FRONT$と $REAR$により表現する。

2方向ポインタの場合、次数3で配列と違うのは、シフト動作の起こる可能性がなく、施錠の範囲が常に小さいことだけである。

次数2以下の場合だと、単施錠の共有施錠を用いることにより，リスト構造自体が物理的に破壊される可能性がある。例として Next,Previous を行なった Tが実行状態の時を考えると，そのアクションにより得られた位置には施錠がなされていない。したがって，他のトランザクションの Delete により位置自体が消失してしまうことが許される。共有施錠を用いない場合にも同様のことがいえる。したがって，次数2以下については，ADT を定義することができない。

4 並行操作の性能評価

シミュレーションによって、これまでに定義した平行操作の性能評価を行う。シミュレーション言語には、GPSS を用いる。

時間の尺度として、1アクションあたりの I/O 時間を 30、CPU 時間を 1 とする。トランザクションは任意の間隔で発生させるが、10 から 20 を一般的な場合とする。また、読み込みアクションと書き込みアクションの比は 1:9 とする。

現在，性能値をシミュレーションで分析中である。

5 結び

本稿では、一貫性次数を用いた並列操作に、データ構造上での並列性の獲得について考察した。

結果として、データ構造上の並列性はデータ構造自身やその構成要素の持つ性質に左右されることがわかった。また、次数によっては操作を表現できないデータ構造があることがわかった。

参考文献

[1] Aho.A.V, Hopcroft.J.E, Ullman.J.D, データ構造とアルゴリズム. 培風館 (1987)

[2] Gray.J.N, Lorie.R.A, Putzolu.G.R, Traiger.I.L, *Granularity of Lock and Degrees of Consistency in a Shared Data Base.*IHP Working Conf on Modeling of Data Base Management Systems(1976),1-29

[3] Agrawal.R, Caray.M.J, Livny.M, *Concurrency Control Performance Modeling: Alternatives And Implications.*ACM-TODS 12-4(1987),609-654