状態制約に基づく並行処理の非直列可能制御方式

4G−4

徐 海燕* 古川 哲也** 史 一華***

* 福岡工業大学 ** 九州大学 *** 福岡工業短期大学

1 まえがき

設計作業に固有の特徴により，従来の事務ＤＢ(データベース)に用いられる並行処理制御方式を，設計ＤＢに直接適用することは困難である[1]．本稿では，設計作業の特徴を分析することにより，設計ＤＢにおける一貫性制約を，設計規則や設計手順によって定義される状態の変化に対する状態制約とすべきであることを指摘する．さらに，このように定義された状態制約が処理単位と独立である特徴を利用することにより，直列可能性を要求しなくても並行処理制御の役割を果たせる制御方式を提案する．

2 状態制約

ＤＢで大量のデータを統一的に管理するために，一貫性制約という実世界の事物の値を正しく表現していることを保証するための条件を設けることができる．しかし，設計ＤＢにおける一貫性制約は，設計作業の次のような特徴を考慮したものでなければならない．

(1) 異なる設計段階で生成される中間結果．
(2) 設計作業の従っている設計規則や手順．

本稿では，設計データを実体とその状態で記述することにより特徴(1)，設計規則や設計手順を設計ＤＢにおける一貫性制約とすることにより特徴(2)に対処する．

定義 1 設計ＤＢは次のような2元組 $< U, M >$ である．
$U = \{o_i \mid o_i$ は実体である $\}$：実体の集合という．

$$M = \left\{ test_k(\alpha) \;\middle|\; \begin{array}{l} (\alpha \in U) \vee (\alpha \subseteq U),\ k \in \{1,2,\cdots,n\} \\ \alpha \text{ が } k \text{ 番目のテストを通過している} \end{array} \right\}:$$

状態の集合という．□

議論を簡単にするため，テストの種類を n 以内としている．また，閉世界仮説を仮定する．即ち，$test_k(\alpha) \notin M$ ならば，α は k 番目のテストを通過していない．本稿では，状態の明記により，設計規則や手順はルールの集合によって表現できる．例えば，実体 x が単体テストを通過しなければ，x を含む実体集合 z が同種類の複合テストを通過できない制約や，1番目のテストを通過しなければ，2,3番目のテストを通過できない制約は，$test_k(x) \leftarrow test_k(z), in(x,z)(k = 1,2,\cdots,n)$, $test_1(x) \leftarrow test_2(x)$, $test_1(x) \leftarrow test_3(x)$ のようなルールの集合で表現できる．

定義 2 設計ＤＢ $=< U, M >$ における状態制約は，次のような状態制約(1)と，状態制約(2)から構成される．

- 状態制約(1)：各 $test_k(\alpha) \in M$ が現時点での $\alpha(\alpha \in U \vee \alpha \subseteq U)$ の状態を記録しているものであることを要求する．
- 状態制約(2)：M 内の状態間の満たすべき性質を要求する．ルールの集合を用いて定義される．□

状態の変化に対する制約は，一般的に，ある関連を持つ実体の状態間の満たすべき性質と，同一実体の異なる種類のテスト結果間の満たすべき性質を要求するものに分類できる．このような制約は，必ず上記で例示したようなルールで表現できる．ただし，ルールの集合で表現される状態制約(2)が，既存の事実から新しい事実を推論するのではなく，既存の事実間の満たすべき性質を要求している．設計ＤＢ $< U, M >$ が状態制約(2)P を満たすとは，M の下で P が成り立つ時である．これを $M \models P$ で表す．

即ち，設計作業に必要な設計規則や手順を状態制約として定義すれば，設計ＤＢ $< U, M >$ が一貫しているのは，状態制約を満たした時かつその時に限る．

3 状態制約と並行処理制御との関わり

処理単位は，設計ＤＢに対する1つの意味のある一連の基本操作の系列から構成される．

定義 3 設計ＤＢ $< U, M >$ に対する基本操作を，次のものとする．

- $read_p(o_i)$：実体 $o_i \in U$ を読み出す．
- $write_p(o_i)$：実体 $o_i \in U$ を書き換える．
- $read_p(l_k)$：事実 $l_k \in M$ を読み出す．
- $insert_p(l_k)$：事実 l_k を M に追加する．
- $delete_p(l_k)$：事実 l_k を M から削除する．□

並行処理制御の目的は，並行実行されている各処理単位 T_i に対して，次のような3つの性質を保証することである．

(1) 論理性：T_i 内の各操作の結果が他の処理単位により変更されないまま，T_i 内のその後の操作に利用できる．さらに，T_i によって $test_k(\alpha)$ が操作された後，実体 $(o_i = \alpha) \vee (o_i \in \alpha)$ も他の処理単位により変更されないまま，T_i 内のその後の操作に利用できる．
(2) 処理一貫性：T_i の実行がＤＢの一貫性を保持する．即ち，T_i の操作されるＤＢが一貫しており，かつ終了されるＤＢも一貫しなければならない．
(3) 回復性：T_i 内の各操作は，全て実行されたかもしくは全く実行されなかったかでなければならない．

従来のＤＢにおいては，直列に実行される処理単位が処理一貫性を満たすものとするため，直列可能性を満たす(実行結果がある順序での直列実行と等価となる)並行実行のみを許可することによって，並行実行における処理単位の処理一貫性を保証している．しかし，設計ＤＢにおいては，処理一貫性を満足しているかどうかは，設計規則や手順を表した状態制約により検査できるので，直列可能性を要求する必要がなくなる．例えば，T_i が処理一貫性を満たす複数個の操作から構成される場合に，T_i の操作される実体や状態が他の処理単位により変更されなければ，T_i は処理一貫性を満たす．

A Concurrency Control Protocol Without Serializablity based on State Constraints

Haiyan XU*, Tetsuya FURUKAWA**, Yihua SHI ***

*Fukuoka Institute of Technology, **Kyushu University, *** Fukuoka Junior College of Technology

例 1 次の並行実行においては，T_i と T_j とも処理一貫性を満たすことができる (T_i と T_j は M の異なる部分を変更とする) が，互いに相手の操作結果を読み出しているため，直列可能ではない．

	T_i	T_j
t_1	$write_p(o_i)$	$write_p(o_j)$
	制約保持のために M を変更	制約保持のために M を変更
t_2	$read_p(o_j)$	$read_p(o_i)$
t_3	$read_p(test_1(o_i))$	$read_p(test_1(o_j))$

4 直列実行における状態制約の保持

設計ＤＢが変更される時に，状態制約 (1) の保持に対しては，次のような管理方法しか用いられない．

定義 4 状態制約 (1) の１つの保持方法：

(1) $write_p(o_i)$ 操作の後，必ず $delete_p(test_k(o_i))$ と $delete_p(test_k(\{o_i,\cdots\}))$ $(k=1,\cdots,n)$ 操作を行う．

(2) $insert_p(test_k(\alpha))$ 操作は，必ず α が k 番目のテストを通過できたテスト操作の後に行う． □

補題 1 定義 4の方法に従えば，任意の直列実行において状態制約 (1) は保持される．

証明：状態制約 (1) を破壊する可能性があるのは，変更操作中の $write_p(o_i)$ と $insert_p(test_k(\alpha))$ 操作のみである．その２つの操作が以上のように実行されれば，明らかに直列実行において状態制約 (1) が保持される． □

状態制約 (2) の保持方法として，真理保全と検証という２つの方法がある．

定義 5 $< U, M >$ を設計ＤＢとし，ルールの集合 P をその状態制約 (2) とする．ある変更操作によって，設計ＤＢが $< U, M' >$ に変更される時に，真理保全とは，$M' \cup M'' \models P$ が恒真になるような極小な M'' を求めて，設計ＤＢを $< U, M' \cup M'' >$ にすることである．検証とは，$M' \models P$ を検査することである．式が成り立つなら，検証が通過できたといい，そうでなければ，検証が通過できていないという． □

定義 6 直列実行におけるＤＢ操作とは，次の２つの条件を満たした基本操作の系列から構成されるものである．

(1) $write_p(o_i)$ と $insert_p(l_k)$ は，定義 4に従って行われなければならない．

(2) $insert_p(l_k)$ と $delete_p(l_k)$ に対しては，事前の検証通過もしくは事後の真理保全が要求される． □

例１の時点 t_1 における操作が定義 6に従って行われたとすると，T_i と T_j はＤＢ操作から構成されることが分かる．

補題 2 定義 6の任意のＤＢ操作の直列実行の結果は，状態制約を満たす．

証明：状態制約 (1) は補題１により，状態制約 (2) は真理保全あるいは通過できた検証により保証される． □

5 並行処理の非直列可能制御方式

本節では，３節で示した考え方に基づいて，ＤＢ操作を利用することにより，並行処理の非直列可能制御方式を示す．

定義 7 並行実行におけるＤＢ操作は，定義 6に次のように施錠し，最後に一括解錠するものである．

(a) $read_p(o_i)/write_p(o_i)(o_i \in U)$ 操作の前に，$rl(o_i)$ / $wl(o_i)$ を行う．

(b) $read_p(l_k)/insert_p(l_k)/delete_p(l_k)(l_k = test_m(\alpha) \in M)$ 操作の前に，$rl(l_k)/il(l_k)/dl(l_k)$ を行い，さらに，実体 $(o_i = \alpha) \vee (o_i \in \alpha)$ に対しても $rl(o_i)$ を行う．

ただし，異なるＤＢ操作内の $rl(o_i)$ と $wl(o_i)$，$wl(o_i)$ と $wl(o_i)$，$rl(l_k)$ と $il(l_k) \vee dl(l_k)$，$il(l_k)$ と $dl(l_k)$ は競合する． □

定理 1 任意の並列実行において，ＤＢ操作の結果が状態制約を満たす．

証明： 定義 7は，狭義二相ロック方式に従って施錠・解錠しているため，並行実行におけるＤＢ操作が必ず直列可能である．従って，補題 2によると，定理が成り立つ． □

次に，処理単位をＤＢ操作から構成させる．

定義 8 処理単位 T_i は，次にような半順序 $<_i$ を持つＤＢ操作の集合である．

(1) $T_i \subseteq \{p \mid p$ はＤＢ操作 $\} \cup \{a_i, c_i\}$,

(2) $a_i \in T_i$ の必要十分条件は，$c_i \notin T_i$ である，

(3) t が a_i または c_i ならば，任意の $p \in T_i - \{a_i, c_i\}$ に対して，$p <_i t$ が成り立つ，

(4) 任意の $p, q \in T_i - \{a_i, c_i\}$(p と q に競合するロックを含む) に対して，$p <_i q$ または $q <_i p$ が成り立つ．

ここで，a_i/c_i は T_i に対する abort/commit 操作である． □

(2) は，a_i と c_i が二者択一であることを表す．(3) は，a_i と c_i が T_i 内の他の全ての操作より後でなければならないことを表す (従って，T_i が回復性を満たす)．(4) は，半順序 $<_i$ の条件である．

定義 9 並行処理制御方式とは，各 T_i に対して次のように施錠・解錠するものである．ただし，$trl_i(o_j)$ と $wl(o_j) \notin T_i$，$trl_i(l_k)$ と $il(l_k) \vee dl(l_k) \notin T_i$ は競合する．

(1) $trl_i(o_j)$ を任意の $rl(o_j) \vee wl(o_j) \in T_i$ より前に，$tru_i(o_j)$ を任意の $ru(o_j) \vee wu(o_j) \in T_i$ より後に行う．

(2) $trl_i(l_k)$ を任意の $rl(l_k) \vee il(l_k) \vee dl(l_k) \in T_i$ より前に，$tru_i(l_k)$ を任意の $ru(l_k) \vee iu(l_k) \vee du(l_k) \in T_i$ より後に行う． □

処理単位は状態制約を満たすＤＢ操作から構成される性質と，$trl_i(o_j)$ と $trl_i(l_j)$ により T_i が操作中の実体 o_j や状態 l_j は他の処理単位により変更できないことから，次の結果が得られる．

定理 2 定義 9に従って実行された処理単位は，任意の並列実行において，論理性と処理一貫性を満たす．

即ち，本稿で提案している並行実行制御方式は，処理単位の３つの性質を保証している．ただし，例１のような並行実行は本方式より許可されることが確認できるので，本方式は直列可能性を要求していないことが分かる．

6 むすび

本稿では，設計ＤＢに対して，文献 [1] に指摘された高水準ＤＢにおける並行処理制御の新たな課題を取り組み，従来の直列可能性を要求する方式から脱出する切り口を見つけた．

参考文献

[1] Barghouti, N. S. and G. E. Kaiser, G. E.: Concurrency Control in Advanced Database Applications, *ACM Computing Surveys*, Vol.23, No.3, Sept. pp.269-317 (1991).