D-015

マルチバージョン制御方式における半順序トランザクション処理の提案
Proposal of Multi Version Concurrency Control for Partial Order Transaction

礒田 有哉† 牛嶋 一智† 田中 剛† 花井 知広† 茂木 和彦†
Yuya Isoda Kazutomo Ushijima Tsuyoshi Tanaka Tomohiro Hanai Kazuhiko Mogi

1. はじめに

近年，ハードウェア技術の進歩により，CPU のコア数増加やメモリの大規模化が進んでいる。DBMS（Database Management System）においては，CPU のマルチコア化に適応したスケーラビリティ技術，メモリの大規模化や永続化に適応したインメモリ技術の開発が急速に進んでいる[1]。

DBMS では，整合性を維持するために，原子性，一貫性，分離性，永続性から構成される ACID 特性を保証することが求められる[2]。しかし，厳密な ACID 特性は，トランザクション（Tx）処理を逐次実行する必要があり性能向上が困難である。このため，一般に ACID 特性を段階的に緩和した分離レベルを用い，スケーラビリティの向上を実現する。

近年，DBMS の整合性制御では，MVCC（Multi Version Concurrency Control）[3]による参照と更新の同時実行が可能な高スケーラブルな制御が用いられる。この制御により，インメモリ DBMS では 100 万 tps（Transaction Per Second）の性能に到達しつつある[4]。

一方，MVCC では，Tx 処理の順序制御にタイムスタンプ（Ts）を用いており，1Tx に 1Ts を採番する。100 万 tps では，1us ごとに Ts を採番する必要があり，分散環境では数 us の通信時間が発生するため実現困難である。このため，高性能な DBMS を実現するためには Ts 制御の効率化が必要不可欠となる。

本稿では，Tx 処理を半順序で制御する POMVCC（Partial Order MVCC）を提案する。POMVCC は，DB の大規模化に伴う Tx 処理の競合率低下に着目した技術であり，同時実行可能な Tx 処理の条件を緩和することで，複数の Tx 処理を 1Ts で実行する。これにより，Tx 処理のスケーラビリティ向上と Ts の採番回数を削減することができ，高スケーラブルな DBMS を実現できる見通しを得た。

2. MVCC におけるスケーラビリティの課題

この章では，MVCC における分離レベルと不整合現象の関係を整理し，Tx 処理の順序関係からスケーラビリティの課題を明確にする。

2.1 MVCC における分離レベルと不整合現象の関係

MVCC による分離レベルと不整合現象の関係を表 1 に記載する。分離レベルは，4 種類の不整合現象である Write Skew（WS），Fuzzy Read（FR），Read Skew（RS），Lost Update（LU）を許容する範囲によって定義されており，Serializable Snapshot Isolation （SSI），Snapshot Isolation（SI），Read Committed（RC），Read Uncommitted（RU）がある。これらの分離レベルは，SSI，SI，RC，RU の順に整合性要件が高く，SSI では如何なる不整合現象も発生してはならない。

† （株）日立製作所 研究開発グループ 情報通信イノベーションセンタ
Hitachi, Ltd., Research & Development Group, Center for Technology Innovation-Information and Telecommunications

表 1 ：MVCC における分離レベルと不整合現象

Isolation Level	Phenomena
Serializable Snapshot Isolation	-
Snapshot Isolation	WS
Read Committed	WS, FR
Read Uncommitted	WS, FR, RS, LU

表 2：不整合現象の定義

Phenomena	Formula
Lost Update	W2[X→X’] W1[X→X”]
Read Skew	R1[X] W2[X→X’] W2[Y→Y’] R1[Y’]
Fuzzy Read	R1[X] W2[X→X’] R1[X’]
Write Skew	R1[X] R2[Y] W1[Y→Y’] W2[X→X’]

不整合現象に関する定義を表 2 に記載する。記号として，Tx 開始（B），コミット（C），参照（R），更新（W），Tx 識別子（Tx1, Tx2），データ（X, Y）を用いる。更新競合として LU が発生し，参照と更新の競合によって RS，FR，WS が発生する。

2.2 MVCC における整合性制御

MVCC は，Ts とデータの更新履歴を用いて Tx 処理を制御する。Ts は，整合性を維持するためにデータや Tx で用いられる。本稿では説明を容易にするために，Tx 処理が利用する Ts 未満のデータは整合性正しく参照できるものとする。

MVCC では，データ X の更新履歴を X (Ts=1) → X’ (Ts=2)とした場合，R1 (Ts=2) [X]は X，R2 (Ts=3) [X]は X’を参照する。このような Ts 制御により，任意の時刻の状態を常に参照できる。更新処理では，楽観的な整合性制御による First Committer Wins（FCW）制御に基づく場合，最新データに対して最も早くコミットした Tx 処理は成功し，競合する後続の Tx 処理は失敗する。また，コミット成功時に Ts を採番する。

これらの制御により，不整合現象を防ぐことができる。LU は FCW 制御によって防げ，RS はクエリ開始時，FR は Tx 開始時に Ts を取得することで防ぐことができる。WS は，ある Tx 処理の参照したデータが他 Tx 処理によって更新されていないこと，更新したデータが他 Tx 処理によって参照されていないことを検出することで防げる[5]。

2.3 MVCC のスケーラビリティ

ACID 特性を厳密に保証するためには，Tx 処理を狭義の全順序関係で実行する必要があるためスケーラビリティが低い。このため MVCC では，広義の全順序関係を用いた並行実行によってスケーラビリティの向上を実現している。表 3 に，狭義の全順序関係を D1，広義の全順序関係を D2，MVCC における Tx 処理の順序関係を D3 に示す。記号として，整数（i, j），コミット時の時間（CT），コミット時の Ts（CTs），Tx 処理の実行内容（Type）を用いる。Type は，参照のみの Read，更新を含む Write で表現する。

表 3：順序関係の定義

D1. Definition of Strict Total Order	
i < j ＜＝＝＞ i ≦ j AND i ≠ j	
D2. Definition of Total Order	
i ≦ j ＜＝＝＞ i < j OR i = j	
D3. Definition of Committed Tx. Order for MVCC	
CTs (Tx i) ⊴ CTs (Tx j) ＜＝＝＞ Ⅰ OR Ⅱ	
Ⅰ	CT (Tx i) < CT (Tx j)
Ⅱ	CT (Tx i) = CT (Tx j) AND Type (Tx i) = Read

MVCC では，D3.Ⅱに示すように，同一 Ts で複数の更新 Tx 処理の実行を許容しない。このため，MVCC における更新 Tx 処理は，必ず D3.Ⅰを満たす必要があり，狭義の全順序関係で実行する必要がある。一方，参照 Tx 処理は，D3.Ⅱに示すように広義の全順序関係で実行可能であるためスケーラビリティが高い。

このように MVCC では，D1 と比べ D3.Ⅱを許容することでスケーラビリティの向上を実現している。このことから，D3 以上のスケーラビリティを実現するためには，D3.Ⅱにおける更新 Tx 処理の順序関係を緩和することが課題となる。

3. POMVCC の提案

この章では，POMVCC の基本アイデアと定義について提案する。また，POMVCC の整合性制御で重要となる Ts の制御について述べる。

3.1 POMVCC の基本アイデア

データベースの一貫性に基づけば，Tx 処理は半順序に制御可能である。たとえば，2 つの更新 Tx 処理で実行順序を入れ替えても実行結果が等しければ，これらの Tx 処理を非順序に実行しても整合性は保たれる。このことを鑑みると，更新 Tx 処理ごとに Ts を採番する必要はなく，同一 Ts で複数の更新 Tx 処理を実行可能である。

一方，DBMS における Tx 制御は，実行順序に依存しない整合性を提供可能だが，アプリケーション（AP）によっては Tx の実行順序を規定することが想定される。このため，DBMS における Tx の実行順序を AP に基づき規定できれば，AP から見た DBMS は整合性正しく動作していることとなる。このような Tx の実行順序を Application View Serializability（AVSR）とする。AVSR では，AP が Tx の実行順序を規定する場合に，先行 Tx の実行結果を後続 Tx が参照できることを保証する。

POMVCC では，Tx 処理の半順序性と AVSR に着目した整合性制御方式である。POMVCC の定義を表 4 に示す。記号として，Tx 実行後の結果（DB），Tx の実行順序（→）を用いる。ただし，SI における→は同時実行を許容し，SSI における→は同時実行を許容しない。

POMVCC では，D3.Ⅱの代わりに D4.Ⅱを用いることで，Tx 処理の実行順序を入れ替えても実行結果が等しければ，Tx 処理の同時実行を許容する。これにより，Tx 処理の並行実行条件を MVCC の D3 より緩和することができる。D4.Ⅱおよび，D3.Ⅱの Tx 処理の並行実行許容範囲を表 5 に示す。表 5 に示すように，D4.Ⅱが#4 を許容することで，Tx 処理の並行実行条件が D3.Ⅱより緩和される。また，D4.Ⅱでは，同一 Ts を複数の更新 Tx 処理で利用できるため，Ts の採番回数を削減できる。

表 4：POMVCC の定義

D4. Definition of Committed Tx. Order for POMVCC	
CTs (Tx i) ≤ CTs (Tx j) ＜＝＝＞ Ⅰ OR Ⅱ	
Ⅰ	CT (Tx i) < CT (Tx j)
Ⅱ	CT (Tx i) = CT (Tx j) AND (DB(Tx i → Tx j) = DB(Tx j → Tx i))

表 5：D3.Ⅱおよび，D4.Ⅱの成立条件範囲

#	D3.Ⅱ	D4.Ⅱ	Formula
1	**Success**	**Success**	R1[X] R2[X]
2	**Success**	**Success**	R1[X] W2[X→X’]
3	**Success**	**Success**	W1[X→X”] R2[X]
4	Failure	**Success**	W1[X→X”] W2[Y→Y’]
5	Failure	Failure	W1[X→X”] W2[X→X’]

3.2 POMVCC における Ts 制御

POMVCC における Ts の採番は，MVCC のように更新 Tx 処理ごとに実施せず，FCW 制御の失敗または，先行 Tx 処理依存を契機に実施する。

FCW 制御の失敗は，Tx 処理における不整合現象の検出によって発生する。たとえば，同一 Ts において複数 Tx が同じデータを更新する LU が発生する場合，D4.Ⅱでは，LU を許容しないことから，Tx 処理のアボート時に Ts をインクリメントする。他不整合現象でも同様である。

先行 Tx 処理依存とは，同一 AP において後続 Tx が先行 Tx の実行結果を参照できない不整合現象（HR：Historical Read）である。具体的には，W1 [X→X’] C1 B2 R2 [X]において，R2 が X’を参照できない現象である。MVCC では，更新 Tx 処理ごとに Ts を採番するため，後続 Tx は必ず先行 Tx の実行結果を参照できる。しかし，POMVCC では，更新 Tx 処理ごとに Ts を採番しないため，先行 Tx の実行結果を参照できない可能性がある。このため，HR を未然に防ぐために，同一 AP による連続した Tx 処理では，後続 Tx 開始時に Ts をインクリメントするか判断する必要がある。ただし，先行 Tx の実行結果を考慮しない Tx 処理であれば，HR は発生しないため Ts を考慮する必要はない。

4. 結論・今後の方針

本報では，MVCC のスケーラビリティを分析し，より高スケーラブルな POMVCC を提案した。POMVCC は，Tx 処理の半順序性と AVSR に着目した整合性制御であり，同一 Ts における複数の更新 Tx 処理を実行可能とする。これにより，更新 Tx 処理の並行実行条件の緩和と Ts の採番回数削減を実現でき，更新 Tx 処理を同時実行する場合においても，スケーラビリティを向上できる見通しを得た。今後は，開発中のインメモリ DB エンジン[6]に POMVCC を実装し，性能およびスケーラビリティの評価を実施する。

参考文献

[1] Stephen Tu, Wenting Zheng, et al., “Speedy Transactions in Multicore In-memory Databases,” SOSP'13, Pages 18-32, (2013).
[2] Hal Berenson, Phil Bernstein, et al., “A critique of ANSI SQL isolation levels,” SIGMOD’95, Pages 1-10, (1995).
[3] Gerhard Weikum, Gottfried Vossen, “Transaction Information System,” Morgan Kaufmann Publishers.
[4] Hideaki Kimura, “FOEDUS: OLTP Engine for a Thousand Cores and NVRAM,” SIGMOD’15, Pages 691-706, (2015)
[5] Alan Fekete, Dimitrios Liarokapis, et al., “Making snapshot isolation serializable,” ACM TODS, Vol. 30, Issue 2, Pages 492-528, (2005).
[6] 礒田有哉, 友田敦, 他, “スケールアップ指向のインメモリデータベースエンジン開発,” FIT’15, D-035, (2015).