異種分散環境における分散トランザクションに関する一考察

6R−4

長谷川 亨　野村 真吾

国際電信電話株式会社 研究所

1. はじめに

複数の種類のデータベースやトランザクション・システムの普及に伴い、異種のシステムを結合して、データアクセスの原子性や直列性を保証する分散トランザクションを実現するニーズが高まっている。異種データベースを結合するマルチデータベース (MDB)[4]は、その一例である。MDBでは、個々のシステムは独立に開発されており、一切変更を加えないことを前提とする。従って、各システムのトランザクション機能をそのまま使って、分散トランザクションを実現する必要がある。しかし、個々のシステムは2相施錠、時刻印、楽観的制御等の異なる並列制御を行ない、また自律的にコミットや棄却を行なうために、分散トランザクションの原子性や直列性を保証することは容易でない。本稿では、異種分散環境における分散トランザクションの実現法を明らかにするとともに、補償トランザクションと時刻印を用いた分散トランザクション・プロトコルを検討する。

2. MDBにおける分散トランザクションの実現法

2.1 システム構成

MDBにおいて分散トランザクションを実現するために、図1に示すシステム構成を仮定する。

(1) ローカルなシステムは、単一サイトのデータだけを参照するローカル・トランザクション (LT) を管理するマネージャ(LTM) と、データを格納するローカル・データベース (LDB) からなる。

(2) 複数のローカル・システムのデータを参照するグローバル・トランザクション (GT) を管理するマネージャ (GTM) が、各サイトに実装される。図1に示すように、GTMに発行されたGTはサブトランザクションSTに分解される。他のサイトのデータを参照する場合は、そのサイトのGTMにSTの処理を依頼する。STはGTMからLTMに発行されることにより実際のデータを参照できる。LTとSTの処理は全てLTMが行ない、GTMはSTの発行、コミット、棄却操作を発行できるだけである。

(3) LTMはstrict(厳密)な並列制御を行い、コミット／棄却操作を他の操作とは独立に発行できる。

2.2 課題および解決法

LTMが自律的に動作することに起因する問題を以下の方針に従って解決する。

(1) 補償による意味的原子性の実現

LTMは発行された操作の実行に関する権利を全て持つため(実行自律性)[4]、コミットが決定したグローバル・トランザクション GT_iのサブトランザクションのコミット操作があるサイトで拒否されて棄却されること (内部棄却) がある。この時、GT_iのサブトランザクションが

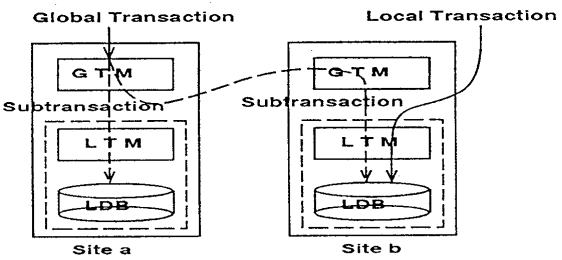

図1 システム構成

コミット(ローカル・コミット)されるサイトと棄却(内部棄却)されるサイトができ、GT_iの原子性が保証できない。従って、GT_iのサブトランザクションがコミットされた全てのサイトでその結果を打ち消す(補償する)ことにより、意味的にGT_iが存在しなかったのと同じ効果(意味的原子性)を得るのが現実的な解法である。具体的には、ローカル・コミットされたサイトで補償を行なうサブトランザクションからなる補償トランザクション CT_iにより打ち消す。

(2) チケットによる直列性の実現

GTMを介さずに直接LTMにLTを発行できる(制御自律性)[4]ため、GT間にデータ競合が無いのにLTの影響で生じる暗黙的なデータ競合を、GTMは検出できない[2]。例えば、共通なデータを参照しないGTのGT_1,GT_2がコミットされた時、LTのLT_1, LT_2の影響により、それぞれのサブトランザクションが実行されたサイトで、GT_1 -> LT_1 -> GT_2 および GT_2 -> LT_2 -> GT_1 の直列順になることがある。そこで、暗黙的な競合をGTMが検出できるように、各LDBにチケットと呼ぶデータ項目をLDB内のデータとして持たせて、全てのGTに読み書きさせることにより強制的に全てのGTに競合を起こさせる[2]。

3. 分散トランザクション・プロトコル

本章では、2章で述べた方法に基づいて、MDBにおいて分散トランザクションを実現するプロトコルについて述べる。

3.1 概要

(1) サイトの2相施錠による意味的原子性の実現

意味的原子性としては、文献[3]が提案する"補償に関して直列" (SRC特性) を採用する。

SRC特性：Sを全てのグローバル・トランザクションがグローバルに終了したスケジュールとする。Sは以下の全ての条件が満たされる場合、補償に対して直列 (SRC) である。

(a) 内部棄却が発生することにより非原子的な全てのトランザクション GT_i に対して、補償トランザクション CT_iがコミットされている。

(b) Sは直列化可能 (serializable) である。

(c) GT_jをS内の任意のトランザクションとする。全てのGT_jより直列順が前の非原子的なトランザクション GT_iに関して、GT_jがコミットされたサイトでは、GT_iは棄却 (内部棄

"A Study on Distributed Transaction Management under Heterogeneous Distributing Environments"
Toru Hasegawa and Shingo Nomura
KDD R & D Laboratories

却) されていない。

(c) の条件を満たすためには、GT_iのサブトランザクションが実行される全てのサイトが、"これら全てのサイトで、GT_iより直列順が前のトランザクションが内部棄却されておらず、将来もされない" ことを合意すれば良い。そこで、プロトコルの開始時にこの分散合意を取ってからコミット処理を行う。各サイトに錠を持たせて、全てのサイトの錠を取れたトランザクションだけが、プロトコルの処理を進めることができ、それ以外の場合は棄却させる。GT_iのサブトランザクションは、実行されるサイトでGT_iより直列順が前の全てのGTのサブトランザクションが以下の条件を満たす場合に施錠できる。

・ローカルにコミットされている
・グローバルにコミット／棄却されている

また、GT_iがローカルにコミットされた時点および、内部棄却された場合は全てのサブトランザクションが補償された時点で解錠する。

(2) 時刻印による直列性の実現

直列性は時刻印方式により実現する。具体的には、全てのGTおよびCTに、時刻印をチケットに書き込ませ、GTMはこの値をトランザクションのスケジューリングに使用する[1]。GTMが一意な時刻印を割り当てるが、GTだけでなく内部棄却が発生した時には補償トランザクションCTにも時刻印を与える必要がある。内部棄却の発生時に割り当てるには、GTが実行されている全てのサイトでコミットされたトランザクションの中で最大の時刻印を調べて、それより大きい値を割り当てる必要があり、効率が悪い。そこで、CTに関しても、GTが発行された時点で割り当てることにする。具体的には、元のGTの時刻印に一定の定数W(ウィンドウ)を加えた時刻印を割り当てる。従って、ローカルにコミットされたがグローバルにコミットされていないGTが読み書きしたデータに、W分の大きさまでの時刻印を持つGTが参照できる。しかし、それ以上の時刻印を持つGTはそのGTがグローバルにコミットされるまでブロックされる。

3.2 コミット・プロトコル

以下では、コーディネータ (CO) とサブオーディネート (SUB) から構成されるコミット・プロトコルを検討する。プロトコルはトランザクションGT_iを対象として説明する。まず、各サイト$Site_l$は、以下の不揮発性の変数を持つ。

・TIK_l: LDB内のチケット。最も最近にローカル・コミットされたトランザクションの時刻印を保持する。
・$CTList_l$: グローバル・コミット／棄却されておらず、実行中のGTのリスト。二次記憶に置かれ、GTの時刻印、状態を保持する。

図2に時刻印、不揮発性変数の関係を示す。ここで、GT_aはグローバルにコミットされ、GT_b, GT_cはローカルにコミットされ、GT_dは開始されたばかりの状態である。また、$CTList_l$の先頭と最後のGTの時刻印をそれぞれ、$Head_l$, $Tail_l$とする。

GT_iは開始に先だって時刻印 $TS(GT_i)$ を、CT_iは$TS(GT_i)$+Wが与えられる。個々のLDBへ全ての操作を発行してから、プロトコルを開始する。プロトコルは3相からなり、COは1相目で全てのSUBにコミット可能かどうかを質問し、各サイトを施錠する。2相目で

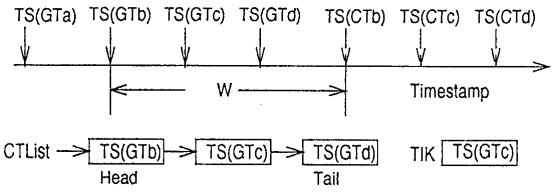

図2 時刻印と不揮発性変数

は、全てのSUBからコミット可能 (yes) を受信すると、SUBにローカル・コミットを依頼する。それ以外の場合、SUBに棄却を依頼する。3相目では、全てのSUBからコミット確認を受信した場合は、SUBにグローバル・コミットを指示し、また、あるSUBから補償を要求された場合は、全てのSUBに補償を要求して、プロトコルを終了する。

$Site_l$のサブオーディネートでは、条件P1が満たされる場合のみ、GT_iを開始し、以降の読み、書き等の操作を受け付ける。それ以外の場合は、一方的にGT_iを棄却する。条件の記述で、>は左辺が右辺より直列順が前ということである。

全ての操作が終了すると、以下の手順でコミット・プロトコルを実行する。

第1相
  if P2 then サイトを施錠、施錠の完了まで待つ
    $CTList_l$にGT_iを追加してから、yes投票
  else サブトランザクションを棄却し、noを投票
第2相
  if コミット要求受信 then TIK_lに TS(GT_i) を書き込んでから、LTMにコミット操作を発行
    if コミット成功 then サイトを解錠、
      確認をCOに返す
    else COに補償を要求 /*コミット失敗(内部棄却)*/
  if 棄却要求受信 then $CTList_l$からGT_iを削除、棄却し、サイトを解錠してから終了
第3相
  if グローバル・コミット要求受信 then
    $CTList_l$から$TS(GT_i)$を削除し、終了
  if ローカル・コミット状態で補償が要求される then
    if P3 then $TS(GT_i)$ + W をTIK_lに書き込む
      補償トランザクションを発行し、
      完了後、$CTList_l$から$TS(GT_i)$を削除し、
      サイトを解錠してから、終了。
    else P3が満たされるまでブロック。

P1 : GTの開始条件 $TS(GT_i) > TIK_l$
P2 : プロトコルの開始条件 $TS(GT_i) < Head_l$ + W
P3 : CTの開始条件 $TS(CT_i) = Head_l$ + W

4. おわりに

本稿では、異種の独立したデータベースを結合する分散環境において、分散トランザクション実現する方法と、補償トランザクションと時刻印を用いたコミット・プロトコルについて述べた。最後に日頃御指導頂くKDD研究所小野所長、浦野次長に感謝する。

参考文献

[1] R. K. Batra, "A Decentralized Deadlock-free Concurrency Control Method for Mutidatabase transactions," The 12 th ICDCS, June 1992
[2] D. Georgakopoulos, et al., "On Serializability of Multidatabase Transactions Through Forced Local Conflicts," 7th IEEE Data Engeneering, 1991
[3] S. Mehrotra, et. al, "A Trasaction Model for Multidatabase Systems," The 12 th ICDCS, June 1992
[4] N. Soparker, et. al, "Failure-Resilient Transaction Management in Multidatabases," IEEE COMPUTER, Dec. 1991