無改造な計算機資源を用いた One-Copy Serializable ミドルウェアの検討

三島　健†

† 日本電信電話株式会社　〒 180–8585 東京都武蔵野市緑町 3–9–11
E-mail: †mishima.takeshi@lab.ntt.co.jp

あらまし　複数の単体データベース（レプリカ）で構成されるクラスタに対し，低コストで高性能かつレプリカ間の一貫性維持を実現するため，One-Copy Serializability（1-Copy-SR）を提供するデータベースレプリケーションミドルウェアの検討がなされている．このデータベースクラスタにおいて，レプリカでスナップショット分離を行うことが提案されているが，データ破壊を起こすケースがあることが知られており，これを回避するには破壊検出のため，ミドルウェアとレプリカ間に専用のインタフェース，ならびに機能追加を行わなければならない．本論文では標準である SQL インターフェースを用いて 1-Copy-SR を実現するミドルウェア上の SQL 転送ルールを提案する．また，このルールを用いることで 1-Copy-SR を実現するミドルウェアである Libera を提案する．評価の結果，既存の提案がほとんど性能向上できないのに対して，Libera は参照が多い負荷の場合にスケーラブルな性能向上を実現できた．

キーワード　One-Copy Serializability，シリアライザブルスナップショット分離，ミドルウェア

1. はじめに

従来より，高性能化するためのデータベースクラスタの検討がさかんである．その中でも市中製品を無改造で使うことで低コスト化を実現するレプリケーションミドルウェア（ピュアミドルウェアと呼ぶ）に注目が集まっている．従って，ミドルウェアは高性能化と低コスト化の他にレプリカ間でコンシステンシを保つことが課題である．単一のレプリカでコンシステンシを保つために最も理想的なトランザクション分離レベルはシリアライザブルである [1]．その理由は，シリアライザブルでは *ANSI SQL phenomena* で知られている様々なデータ一貫性破壊が起きないからである [2]．複数のレプリカからなるデータベースクラスタもクライアントにシリアライザブルを提供することが望ましい（One-Copy Serializability: 1-Copy-SR と呼ぶ）．例えば，Amza らは *distributed versioning (DV)* と呼ぶ二相ロックをベースにした 1-Copy-SR を提供するミドルウェアを提案している [3]．トランザクションの開始時にテーブルにアクセスする順序を示すバージョン番号をトランザクションに与え，バージョン番号の順序でロックをかけながらアクセスする．ロックはテーブルにアクセスする必要がなくなった時点で解放される．DV はグローバルに 1-Copy-SR を提供しローカルにはシリアライザブルを提供する．残念なことにシリアライザブルは性能が低いという問題点があり，これが 1-Copy-SR も性能が低いという欠点を生み出す．

1995 年には，新たな分離レベルとして Gray らがスナップショット分離 (SI) を提案した [4]．参照命令は更新命令をブロックしないため高い並列性を実現でき，それゆえに高性能化も期待できる [2]．従って，SI のレプリカを複数使ったクラスタ構成でクライアントに SI を提供する One-Copy Snapshot Isolation (1-Copy-SI) [5] が注目された．例えば，我々はローカルには SI を提供しグローバルには 1-Copy-SI を提供する Pangea と呼ぶミドルウェアを提案した [6]．1-Copy-SI では SI は高性能化の代わりに anomalies が発生しコンシステンシを崩してしまう可能性があるという欠点を持つ. [2].

Bornea らは ローカルには SI を提供しグローバルには 1-copy-SR を提供する *Serializable Generalized Snapshot Isolation (SGSI)* アルゴリズム [7] を提案している．しかし，コンシステンシを維持するために，大きなオーバヘッドを生じるグループコミュニケーション（高信頼化マルチキャストの一種）が必要となる．また，SI anomaly を見つけ出すためにミドルウェアに検証機能が必要になる．検証機能には検証専用のデータベースエンジンが必要で，本来のデータベースエンジンのデータベースと同じスキーマを用意しておく必要がある．彼らはデータベースエンジンとして Microsoft SQL Server 2008, 検証専用のデータベースエンジンとして SQLite.NET を使っている．

2008 年には， 新しいシリアライザブルのアルゴリズムとして，シリアライザブルの性能改善を狙ったシリアライザブルスナップショット分離 (SSI) が提案された [8]. SSI では，トランザクションは SI で実行するのとともに実行中に SI anomalies を検出・回避し，結果としてシリアライザブルを提供する．これは高性能化を実現するとともに anomalies を発生しない．

本論文では，SSI を提供するレプリカからなるクラスタが 1-Copy-SR を保証するための SQL 転送ルールを提案する．そのルールを使ってクライアントに 1-Copy-SR を提供する Libera と呼ぶミドルウェアを提案する．本論文の貢献を以下に示す．

- 我々は SSI を提供するデータベースレプリカを使って 1-Copy-SR を保証するための SQL 転送ルールを提案する．このルールはシンプルであるため，ピュアミドルウェアを作るためには有益である．
- 我々は上記ルールを使って Libera と呼ぶ 1-Copy-SR を提供するミドルウェアのプロトタイプを提案する．我々の知る限りでは SSI を使って 1-Copy-SR を提供する最初のミドルウェアである．

- 我々は TPC-W ベンチマークを使って評価を行った．その結果，参照が多い環境では 1-Copy-SR を従来技術でミドルウェアを作った場合に比べて Libera は高い性能を示すことが分かった．

2. 予備知識

本章では，1-Copy-SR を論理的に議論するために必要な formal apparatus を導入する．また，関連する知識として SI と SSI, Pangea についても概説する．

2.1 データベースモデル

database はデータアイテムの集合を構成する．各データアイテムは value を持つ．value はある点においてデータベースの state を構成する．データアイテムは x または y，z と小文字で表記する．DBMS はデータベースにアクセスするコマンドをサポートするハードウェアとソフトウェアのかたまりである．このコマンドはクエリと呼ぶ．トランザクション T_i は参照クエリ，更新クエリ，終了クエリ（コミットまたはアボート）である．下付き文字 i は i 番目のトランザクションを表し，他のトランザクションと区別するために使用する．従って，x_i はトランザクション T_i によって更新されたデータアイテム x を表す．また，$w_{i,p}(x_i)$ はデータアイテム x_i を更新したトランザクション T_i の p 番目の更新クエリを表す．同様に，$r_{i,q}(x_j)$ はデータアイテム x_j に対する q 番目の参照クエリを表す．c_i と a_i は T_i のそれぞれコミットクエリとアボートクエリを表す．

もしトランザクション T_j がスタートする前にトランザクション T_i がコミットしたならば T_i と T_j はシリアルである，と言う．もし，トランザクション T_j がスタートする前であって，トランザクション T_i はスタートしているがコミットしていない状態の時，トランザクション T_i とトランザクション T_j は平行であると言う．もし R^m が R^n と同じデータアイテム x^m と x^n とを持っていて x^m と x^n が同じデータを保持するならば，R^m と R^n は同値であると言う．もし，複数のトランザクション集合が平行に実行されていたら，クエリは交互に入り組むかもしれない．DBMS がクエリを実行しようとする順序をスケジュールという．ヒストリは実際にクエリを実行した順序を表す．スケジュールはこれからトライする順序であってヒストリは実際に実行した順序である．

データアイテムを参照する前に更新するいわゆるブラインド更新はないものと仮定する [9], [10]．T_i のスナップショットは $r_{i,1}(x_k)$ が実行される直前である．基本的には $w_i(x)$ クエリは全てのレプリカで実行し，参照クエリ $r_i(x)$ は一つのレプリカから読むといういわゆる ROWA モデルをベースとする．代表レプリカとはアクセスする毎に全レプリカの中からランダムに一つ決定する．リーダレプリカは，代表レプリカと同様一つのレプリカが選定されたものだが，一度決めたら変わらない．フォロワレプリカとは全てのレプリカからリーダレプリカを除いたレプリカの集合である．グローバルトランザクションとはクライアントが送信したトランザクションを言い，ローカルトランザクションはレプリカが実行するトランザクションを言う．

2.2 依存関係

トランザクション T_i と T_j が依存関係を持つとは，(1) ある $DBMS_i$ が他の DBMS とは異なる順序でトランザクション T_i と T_j を実行し，(2) かつ，両 DBMS は異なる結果を出力する場合を言う．この二つの DBMS を同期させるには両 DBMS で全てのディペンデンシを同じ順序で実行する必要がある．データアイテム x_j がデータアイテム $x_i (i < j)$ の「直後の更新者」と呼ぶ場合は，(1) データアイテム x_i が参照または更新されたこと，(2) データアイテム x_j は更新によって生成されたこと，(3) x_i と i_j との間に他の更新が無いこと，を指す．クエリ o_i と o_j がディペンデンシを持つとは，最低限一つのクエリが更新であり，実行順序が異なると実行結果が異なる場合を意味する．以下のように，三つのディペンデンシが存在する．

[**定義** 1]（dependencies） 二つのクエリ間で生じる三つの dependency は次の通りである．

- (1) クエリ o_i はデータアイテム x を更新し，(2) クエリ o_j はこのバージョンのデータアイテム x_i を読む，場合，クエリ o_i と o_j は wr-dependency があると言う．
- (1) クエリ o_i は T_k によって更新されたデータアイテム x を参照し，クエリ o_j が直後の更新者である場合，クエリ o_i と o_j は rw-dependency があると言う．
- (1) クエリ o_i がデータアイテム x に対して更新し x_i とした後，クエリ o_j が直後の更新者である場合，クエリ o_i と o_j は ww-dependency があると言う．

2.3 スナップショット分離

SI は強い分離レベルであると同時に高性能である [4]．SI の下ではトランザクション T_i はコミットされたデータベースの状態，すなわちスナップショットからデータを読み込む．データアイテムを更新する場合はそのスナップショットに対して行われる．トランザクション T_i のスナップショットは最初の参照クエリ $r_{i,1}(x_p)$ を実行する直前に暗に作成される．SI では参照クエリが更新クエリをブロックすることは無い，逆に更新クエリが参照クエリをブロックすることもない．この特徴が平行実行を増やし性能が向上する理由である．

2.4 シリアライザブルスナップショット分離

2008 年に Cahill らはシリアライザブルスナップショット分離 (SSI) と呼ぶ新しくて強い分離レベルを提案した [8]．SSI は実行中に SI anomaly が起きる可能性がある実行パターンを探し，見つけ次第アボートする．このアルゴリズムは楽観的平行制御に似ているが SI アノマリを見つけ次第即座にアボートする点が異なる．この提案手法は全てのシリアライザブルでないトランザクションの実行をアボートするが，アボートする必要の無い実行までアボートする可能性がある．

- アプリケーションが動作中であってもこのアルゴリズムはシリアライザブルを保証する．
- このアルゴリズムによって参照クエリが遅延することはない．また，参照クエリは更新クエリの実行を妨げない．
- 様々な環境において，スループットは SI とほぼ同じであり二相ロックよりも良い．

- SSI のアルゴリズムは SI を提供するシステムを少し改造するだけで実装できる．

2.5 Pangea [6]

2.5.1 Same Snapshot Creation プロトコル

もし複数のレプリカがそれぞれ異なるスナップショットからデータアイテムを参照したり更新したりすると，レプリカ間のコンシステンシは崩れてしまう．従って，(1) 最初の参照クエリを実行する直前の全てのコミットされたデータをスナップショットは含むこと，(2) スナップショットへの変更がデータベースに反映されるのはコミットクエリを実行した時であること，を考えると，最初の参照クエリとコミットクエリの相対順序を全てのレプリカで同じにしなければならない．より具体的にはコミットクエリを実行する時は最初の参照クエリが一つも実行されないようにしなければならない．また，逆に，最初の参照クエリを実行する時はコミットクエリが一つも実行されないようにしなければならない．

2.5.2 Leader/follower プロトコル

あらかじめ，Pangea はリーダのレプリカを選出し残りのレプリカをフォロワとする．Pangea は一旦リーダとフォロワを決めると，障害が起きない限りその役割分担を変えることは無い．あるトランザクションのスナップショットを全てのレプリカで同じにするためには Sect.2.5.1 節で述べたようにする．

Leader/Follower プロトコルは次に示す通りである．

(1) Pangea クライアントが送信した更新クエリを受信する．
(2) Pangea は leader にだけ更新クエリを送信する．
(3) leader の ww-dependency の勝者を決定する．
(4) Pangea は正解応答を受けとったら，
(5) Pangea 正解応答が返ってきた更新クエリを follower へ送信する．
(6) Pangea は follower から正解応答を受信する．

Pangea は参照クエリを一つのレプリカのみに送信する．最初のクエリ（参照クエリも含む）とコミットクエリは全てのレプリカに送信する．

3. シリアライザブルスナップショット分離を使った One Copy Serializability の実現方法

本章では 1-Copy-SR を実現するとともにレプリカ間でコンシステンシ維持を実現する新しいルールを提案する．

3.1 dependency とクエリの関係

レプリカがクエリを送信し応答を受信した後に新たなクエリを送信する．レプリカが新しいクエリを送信するのは応答を受信した後であることに注意されたい．SSI では参照クエリと更新クエリは論理的にはそれらが送信された時刻に実行されず以下のタイミングで実行される．

[**補題** 1]（論理的な参照クエリ） トランザクション T_i が参照クエリ $r_{i,1}(x_k)$, $r_{i,2}(x_k)$, ... , $r_{i,n}(x_k)$ を実行する時，最初のクエリ $r_{i,1}(x_k)$ が実行された時に全ての参照クエリを実行した場合に等価である．

証明. SSI では全ての参照クエリは同一のスナップショットから読み込む．スナップショットは最初のクエリ $r_{i,1}(x_k)$ が実行された直前に生成される．従って，参照クエリがいつ実行されようともその実行は最初の参照クエリ $r_{i,1}(x_k)$ が実行された場合行われたのと等価である． □

[**補題** 2]（論理的な更新クエリ） トランザクション T_i が更新クエリ $w_{i,1}(x_i)$, $w_{i,2}(x_i)$, ... , $w_{i,n}(x_i)$ を実行する時，コミットクエリ $c_i(x_i)$ が実行された時に全ての更新クエリを実行した場合に等価である．

証明. SSI では全ての更新クエリは同一のスナップショットに対して行われ更新内容はコミット時に確定される．従って，更新クエリがいつ実行されようともその実行はコミットクエリ $c_i(x_i)$ が実行された場合に行われたのと等価である． □

ミドルウェアレベルでクエリからトランザクションの依存関係を取得するためには dependency とクエリ間で次のような関係がある．

[**補題** 3]（wr-dependency） トランザクション T_i と T_j において，もし T_i が更新トランザクションであり，かつ $c_i < r_{j,1}(x_i)$ の関係が成り立つならばトランザクション T_i と T_j は wr-dependency である．

証明. T_i は更新トランザクションであるから $w_{i,p}(x_i) < c_i$ が成り立つ．従って，$w_{i,p}(x_i) < c_i < r_{j,1}(x_i)$ である．定義 1 よりこの関係は wr-dependency である． □

[**補題** 4]（ww-dependency） もし T_i と T_j が更新トランザクションであって，更新クエリに $w_{i,p}(x_i) < w_{j,q}(x_j)$ の関係が成り立つならば，トランザクション T_i と T_j は ww-dependency である．

証明. もし $w_{i,p}(x_i) < w_{j,q}(x_j)$ であるならば，定義 1 よりトランザクション T_i と T_j は ww-dependency であることは明らかである． □

[**補題** 5]（rw-dependency） T_i は更新トランザクションであり $r_{j,1}(x_k) < c_i$ であるならば，トランザクション T_i と T_j は rw-dependency である．

証明. T_i は更新トランザクションであるから，定義 1 と 2 より $r_{j,1}(x_k) < w_{i,q}(x_i) < c_i$ である．これは rw-dependency に相当する． □

[**定義** 2]（commit abort） もし T_i と T_j，T_k が dangerous structure [11] を持っているならば，c_i または c_j，c_k のうちどれかはアボートされる．

3.2 SSI-Schedule

SSI では補題 1 から 5，さらに定義 2 より，次のスケジュールを定義する．

[**定義** 3]（SSI-schedule） $\mathcal{T}$ をトランザクションの集合とする．$\mathcal{T}$ に対する SSI-schedule S はクエリ $o \in \{r_{i,1}(x_k), r_{i,2}(x_k), ..., r_{i,m}(x_k), w_{i,1}(x_i), w_{i,2}(x_i), ..., w_{i,n}(x_i), c_i\}$ のシーケンスである．この時，S は次のような特徴を持つ．

(1) 各トランザクション $T_i \in \mathcal{T}$ は $r_{i,1}(x_k) < c_i$ である．
(2) もしトランザクション T_i と T_j，T_k の集合が dangerous structure を持っていたら，どれか一つのトランザクションはアボートさせる．

SI-schedules と呼ぶ同じような定義が過去に二つあった [5] [6]. SSI-schedule と異なる点は対象が SI であることである．また, [5] の schedule は SQL ではなく writeset を対象としている．さらに, [6] の schedule は ww-dependency しか対象としていない．

3.3 SSI-Equivalent

レプリカ間のコンシステンシを議論するためには，equivalent の概念は重要である．

［**定義** 4］(SSI-equvalent) もし SSI-schedule S^m と S^n が次の特徴を満足するならば，S^m と S^n は SSI-equvalent であるという．

(1) T_i が更新トランザクションであり $c_i^m < r_{j,1}^m \in S^m \Leftrightarrow c_i^n < r_{j,1}^n \in S^n$ である．

(2) T_i が更新トランザクションであり $r_{j,1}^m < c_i^m \in S^m \Leftrightarrow r_{j,1}^n < c_i^n \in S^n$ である．

(3) T_i と T_j が更新トランザクションであり， $w_{i,p}^m(x_i) < w_{j,q}^m(x_j) \in S^m \Leftrightarrow w_{i,p}^n(x_i) < w_{j,q}^n(x_i) \in S^n$ である．

(4) T_i が更新トランザクションであり, $c_i^m < c_j^m \in S^m \Leftrightarrow c_i^n < c_j^n \in S$ である．

［**定理** 1］(SSI-equvalent) R^m と R^n は同じデータアイテム x を持っており，かつ，x^m と x^n は両方とも同じ値である．もし，S^m と S^n が SSI-equivalent である時 R^m と R^n はコンシステンシを保つ．

証明. S^m と S^n が SSI-equivalent であるならば，補題 3 と (1) より S^m and S^n には同じ wr-dependency を持つ．また，補題 4 と (3) から S^m に S^n 同じ ww-dependency を持つ．さらに，補題 5 と (2) から S^m と S^n に同じ rw-dependency を持つ．(4) は SI anomaly を排除する．従って，三つ全ての dependency が S^m と S^n で成り立つため，anomaly は R^m and R^n に存在しせず consistent である． □

3.4 Mapping Function

グローバルトランザクション（ミドルウェアが受けとるトランザクション）からローカルトランザクション（ミドルウェアから送信するトランザクション）を生成する mapping function を説明する．mapping function は以下に定義する．

［**定義** 5］(Mapping Function) $\mathcal{T}$ をグローバルトランザクションとし，各トランザクションはクエリ $o_i \in \{r_{i,1}, r_{i,2}, ..., r_{i,p}, w_{i,1}, w_{i,2}, ..., w_{i,q}, c_i, a_i\}$ を持つ．mapping function $\mathcal{F}$ はグローバルトランザクションからローカルトランザクション $\mathcal{T}^l, \mathcal{T}^m, \mathcal{T}^n$ を以下のようにして出力する．

(1) もし，グローバルトランザクション T_i が参照のみのトランザクションであるならば，mapping function $\mathcal{F}$ は一つの代表レプリカ R_i^l のために全てのクエリを持ったローカルトランザクション T_i^l を出力する．

(2) もし，グローバルトランザクション T_i が更新を含むトランザクションであるならば，mapping function $\mathcal{F}$ はリーダレプリカ R_i^m のために全てのクエリを持ったローカルトランザクション T_i^m を出力する．また，mapping function $\mathcal{F}$ はフォロワレプリカ R_i^n のために最初の参照クエリ $r_{i,1}$ とコミットクエリ c_i を持ったローカルトランザクションを出力する．

(3) もし，グローバルトランザクション T_i が更新を含むトランザクションであるならば，mapping function $\mathcal{F}$ はフォロワレプリカ R_i^n のために更新クエリを持ったローカルトランザクションを出力する．

3.5 1CS Rule

レプリカ間のコンシステンシ維持を実現すると同時に 1-Copy-SR を実現するルールを提案する．

［**定義** 6］(1CS rule) $\mathcal{R}$ を ROWA アプローチに従うレプリカの集合とする．$\mathcal{T}$ をグローバルトランザクションの集合とする．S をグローバルトランザクション $\mathcal{T}$ の集合に対する SSI-schedule とする．レプリカ $R^m \in \mathcal{R}$ のローカルトランザクションの集合 $\mathcal{T}^m$ に対する SSI-schedule を S^m とする．以下の特徴が成り立つ時 S は 1-Copy-SR を提供する．

(1) $\cup_k = rmap(\mathcal{T}, \mathcal{R})$ である ROWA mapping function rmap が存在する．

(2) $T_i, T_j \in \mathcal{T}$ から変換した $T_i^m, T_j^m \in \mathcal{T}^m$ に対する各 S^m を含む $\mathcal{T}$ に対する SSI-schedule S が存在する．

(a) T_i は更新トランザクションであり, $c_i^m < r_{j,1}^m \in S^m \Leftrightarrow c_i < r_{j,1} \in S$

(b) T_i は更新トランザクションであり, $r_{j,1}^m < c_i^m \in S^m \Leftrightarrow r_{j,1} < c_i \in S$.

(c) T_i と T_j は更新トランザクションであり, $w_{i,p}^m(x_i) < w_{j,q}^m(x_j) \in S^m \Leftrightarrow w_{i,p}^n(x_i) < w_{j,q}^n(x_i) \in S$

(d) T_i と T_j は更新トランザクションであり, $c_i^m < c_j^m \in S^m \Leftrightarrow c_i < c_j \in S$.

1CS の特徴は以下の通りである．

［**定理** 2］(1CS rule) もし，仮想的な一つのノード R のスケジュール S が 1CS ルールに従うならば，S は 1-Copy-SR を提供する．

証明. 最初に更新を含むトランザクションに注目する．(1) よりリーダ R^m のためのローカルトランザクション T_i が作成される．もし，リーダトランザクション T_i が 1CR ルールの (2) を満たすならリーダ R^m と仮想ノード R は equivalent である．従って，S^m が 1-Copy-SR を提供するならば S も 1-Copy-SR を提供する．次に参照のみのトランザクションに注目する．(1) よりフォロワのためのローカルトランザクション T_j が作成される．もし，フォロワスケジュール S_i が 1CR ルールの (2) を満たすならば，フォロワ R^n と仮想ノード R は equivalent である．S^n は 1-Copy-SR を保証するので S は 1-Copy-SR を提供する．以上より，S は 1-Copy-SR を提供する． □

4. Libera

本章では，1-Copy-SR を実現する Libera と呼ぶピュアミドルウェアを提案する．我々のアプローチは現実的なミドルウェアを実現するために市中技術を使って高性能化を実現する．このために，same snapshot creation プロトコル, Leader/Follower プロトコル, SSI の certifier を利用する．既存のレプリケーションミドルウェア [3], [7] を拡張して 1-Copy-SR を提供可能ではあるが，たくさんの高価な実装に悩まされる．例えば，グループコミュニケーションの実現，readset の作成方法，writeset の

T1	T2	T3
BEGIN; UPDATE table1 SET a=1 WHERE a=0; COMMIT;	BEGIN; UPDATE table1 SET a=0 WHERE a=1; COMMIT;	BEGIN; UPDATE table2 SET a=1 WHERE a=0; COMMIT;

図 1　Sample transactions

挿入方法，SSI anomaly を検出する certifier の実現，などである．本章では，市中技術を無改造で用いて 1-Copy-SR を実現する新しいミドルウェアを提案する．

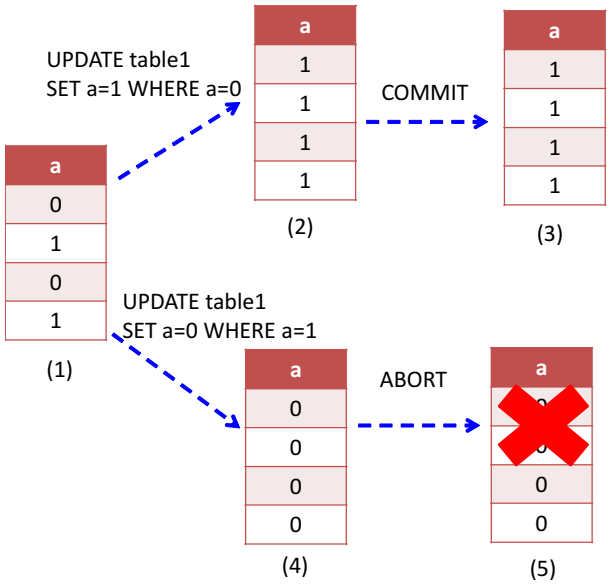

図 2　Problem of commit order

4.1　問題の提起

SSI ではコミットの順序が変わると最終的な結果が異なるケースがある．例として，図 1 に示す三つのトランザクション T_1 と T_2, T_3 と図 2(1) に示すテーブル 1 を考える．また，三つのトランザクションは平行して実行されているとする．図 2(2) に示すように T_1 は UPDATE クエリを実行する．また，図 2(4) に示すように T_2 は UPDATE クエリを実行する．コミットの順序に注意されたい．もし，T_1 のコミットクエリが T_2 よりも早かったとすると，テーブルは図 2(3) で確定し，T_2 は図 2(5) に示すようにアボートされる．反対に T_2 のコミットクエリが T_1 よりも早かったとすると，テーブルは図 2(4) で確定し，T_1 はアボートされる．このアボートは write skew anomaly によるものである．図 1 に示すように三番目のトランザクション T_3 を考える．この場合何の anomaly も発生しないので T_1 と T_3 のコミットクエリの順序は重要ではない．従って，複数のレプリカでコンシステンシを保つためには anomalies を生じる COMMIT クエリはシリアルに実行しなければならない．また，anomalies を生じないコミットクエリは性能を上げるために同時並行に実行すべきである．

4.2　キーアイデア

我々のキーアイデアは (a) same snapshot creation プロトコルと (b) Leader/Follower プロトコルを (c) 各レプリカが持っている certifier の機能をミドルウェアで組み合わせることである．つまり，

(a)　Libera は全てのレプリカで同じスナップショットを作成し，

(b+c)　もし，リーダレプリカの certifier が anomaly を見つけたとすると，Libera は全てのレプリカでその原因のトランザクションをアボートする，もし，なんら anomaly が見つからなかったとすると，全てのフォロワレプリカはコミットクエリを平行に実行する．

Algorithm 1 Read-only transaction

```
1: receive a query from a client
2: send the query to the representative replica
3: receive an answer from the replica
4: send the answer to the client
```

4.3　Libera アーキテクチャ

DBMS へ送信されたクエリを受信するため，Libera はクライアントと DBMS の間に位置する．Libera は市中製品を無改造で使うだけでなく，certifier, group communication, readset の抽出機能，writeset の作成機能，など複雑な機能は必要としない．

4.4　アルゴリズム

本節では Libera の動作を詳細に説明する．もしあるレプリカからアボート応答を受け取ったとすると，Libera は全てのレプリカにロールバッククエリを送信する．あらかじめ，Libera は全てのレプリカからリーダレプリカを選出し，残りをフォロワと決めておかなければならない．一旦、役割を決めたら障害が起きない限り役割を変えない．Libera は参照のみのトランザクションのために全てのトランザクションの中からランダムに代表レプリカを決める．代表レプリカは参照トランザクション毎に決める．

アルゴリズム 1 は参照のみのトランザクションを受信した時の Libera の動作である．Libera はクライアントと代表レプリカの間にあり，クエリと応答を中継するだけである．

アルゴリズム 2 は更新を含むトランザクションを Libera が受信した時の動作である．もし，Libera が更新クエリを受信したら (line 1)，Libera は Leader/Follower プロトコルを実行する；つまり，Libera はクエリをリーダにのみ送信する (line 3)．Libera はリーダから応答を受信する (line 4)，もし，応答がアボートである場合 (line 5)，Libera はロールバッククエリを全てのレプリカに送信する (line 6)．そして，全てのレプリカから応答を受信する (line 7)．もし，応答が成功であった場合，Libera は全てのフォロワにクエリを送信する (line 9)．そして全てのフォロワから応答を受信したら (line 10)，一つをクライアントへ返す (line 12)．

もし，Libera が最初の参照クエリを受信したら (line 13)，クリティカルリージョンに入る (line 14)．このリージョンではコミットクエリが一つも実行されていないことを保証する．Libera が最初の参照クエリを全てのレプリカへ送信したら (line 16)，Libera は全てのレプリカから応答を受信する (line 17)．クリ

Algorithm 2 Update transaction

```
 1: receive a query from a client
 2: if the query is a write query then
 3:     send the query to the leader;
 4:     receive a response from the leader;
 5:     if the response is an abort then
 6:         send a rollback query to all replicas;
 7:         receive a response from the all replicas;
 8:     else
 9:         send the query to all the followers;
10:         receive a response from all the followers;
11:     end if
12:     send the response to the client
13: else if the query is the first read query then
14:     enter critical region;
15:     /* there is no commit operation executed */
16:     send the operation to all replicas;
17:     receive responses from all the replicas;
18:     leave critical region;
19:     send the response to the client
20: else if the query is a read query (excluded the first read query) then
21:     send the query to the leader
22:     receive a response from the leader;
23:     send the response to the client;
24: else if the query is a commit operation then
25:     enter critical region;
26:     /* there is no first read operation executed */
27:     send the query to the leader;
28:     receive a response from the leader;
29:     if the response is an abort then
30:         send a rollback query to all replicas;
31:         receive a response from the all replicas;
32:     else
33:         send the query to all the followers;
34:         receive responses from all the followers;
35:     end if
36:     leave critical region;
37:     send the response to the client
38: end if
```

ティカルリージョンを抜ける (line 18)．Libera は一つの応答をクライアントへ返す (line 19)．

もし，Libera が参照クエリ（最初の参照クエリは除く）を受信したら (line 1)，Libera はリーダに送信し (line 21)，応答をリーダから受信する (line 22)．

もし，Libera がコミットクエリを受信したら (line 1)，クリティカルリージョンに入る (line 25)．このリージョンでは最初の参照クエリが実行されていないことを保証する．Libera はリーダにコミットクエリを送信すると (line 27)，リーダは応答を受信する (line 28)．もし，応答がアボートならば (line 29)，Libera は全てのレプリカへロールバッククエリを送り (line 30)，応答を受けとる (line 31)．もし，応答が成功ならば，Libera は全てのフォロワへクエリを送信する (line 33)．そして，Libera は全てのフォロワから応答を受けとる (line 34)．もし，Libera がアボート応答を受けとったら，全てのトランザクションをロールバックする．Libera はクリティカルリージョンを抜ける (line 36)．Libera はクライアントへ応答を返す (line 37)．

1.00.7

5. 評　　価

Libera の有効性を示すために，プロトタイプを実装し TPC-W ベンチマーク [12] を使った性能評価を行った；スループット、応答時間、拡張性について Libera と 既存技術を使って 1-Copy-SSI 提供するミドルウェアとを比較した．

5.1 性能比較のために基準としたミドルウェア

Libera の有効性を示すためには，Libera の性能と 1-Copy-SI を実現する他のミドルウェアの性能を比較したい．そこで，*SGSI* [7] と *Distributed Versioning (DV)* [3] の二つの候補がある．Libera と違って SGSI は非標準なインターフェースである多くの機能を必要とする．具体的には，readset の抽出，writeset の挿入，高信頼マルチキャストプロトコルの実装，certifier の実装，である．特に，高信頼化マルチキャストプロトコルは JGroups [13] や Horus [14]，Ensemble [15] など多数の研究成果が存在するが，どれも複雑で高価である．特に SGSI については certifier 専用の *CertDB* と呼ぶ DBMS（彼らの研究では SQLite.NET）が必要であり本末転倒である．対して DV は SQL インターフェースを前提としており certifier のような高価な機能を必要としない．従って，DV を選択し，1-Copy-SR を SSI を提供するレプリカを使って実現するように改造した（DV-SSI と呼ぶ）．

5.2 TPC-W ベンチマーク

TPC-W ベンチマーク [12] はオンライン書店にアクセスするお客をモデル化している．データベースは item と country, author, customer, orders, order_line, cc_xacts, address を管理している．14 種類のアクセスパターンがあり，そのうち 6 個は参照のみのトランザクションから構成されている．このベンチマークはブラウジング，ショッピング，オーダリングの三つの異なる負荷を試験できる．テーブル 1 は三つの負荷におけるアクセスパターンの比率を表している．

表 1　Three workloads in TPC-W benchmark

	read-only	update
browsing mix	95%	5%
shopping mix	80%	20%
ordering mix	50%	50%

ブラウジング負荷は参照が多く，オーダリング負荷は更新が多いという特徴がある．評価基準は スループット（WIPS: web interactions per second）と応答時間 (ms) である．

5.3 実 験 環 境

測定には最大で 8 ノードを使い，一つはリーダ，残りがフォロワである．さらに，ミドルウェアを動作させる一つのノード，負荷を発生させるノードを四つ使った．全てのノードは同じスペックである (一つの 3.1 GHz Xeon E3-1220 CPU, 16 GBytes RAM, and one 250 GB SCSI HDD)．全てのノードは一つのギガビットイーサスイッチで相互接続されている．ノードで動作させる DBMS として，バージョン 9.4.9 の PostgreSQL を使った．設定ファイルを変更した箇所は `default_transaction_isolation` を `SERIALIZABLE` (注1) とした．TPC-W ベンチマークに与えたパラメータは 100 万 item と 288 万 customers であり，これは約 6 GBytes の大きさを占める．

5.4 結　　果

本節では，TPC-W ベンチマークを使った場合における Libera と DV-SSI のスループットと応答時間を比較した．

(注1)：PostgreSQL では `SERIALIZABLE` を指定すると，serializable snapshot isolation で動作する．

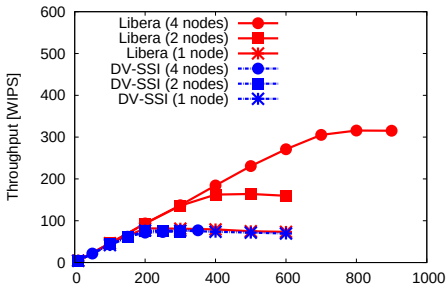

図 3　Throughput under browsing mix

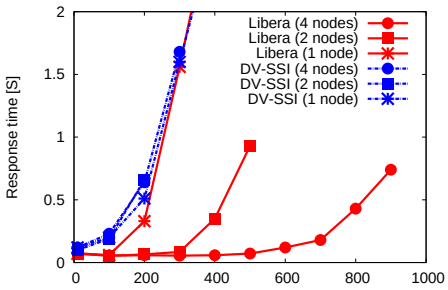

図 4　Response time under browsing mix

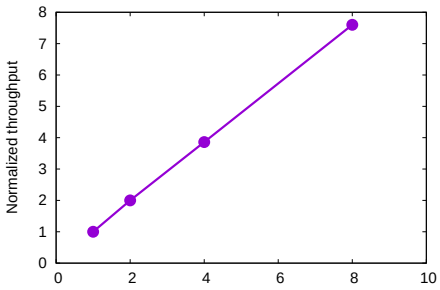

図 5　Scale out

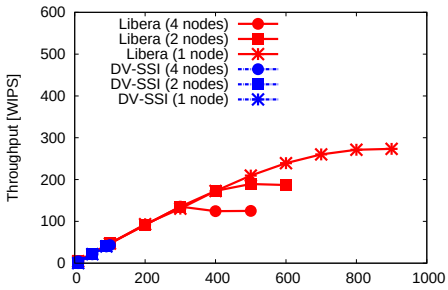

図 6　Throughput under ordering mix

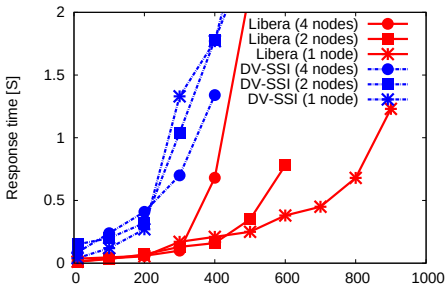

図 7　Response time under ordering mix

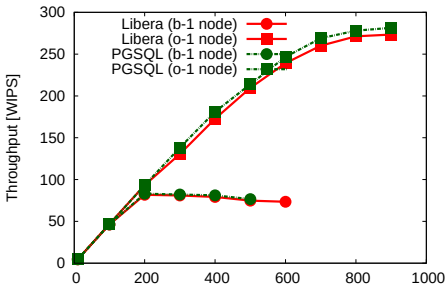

図 8　Throughput with 1 node

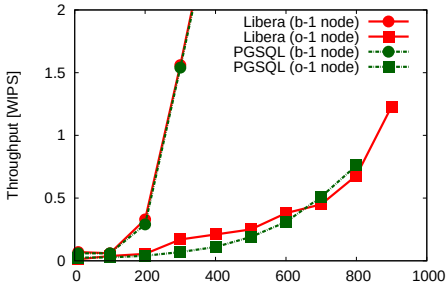

図 9　Response time with 1 node

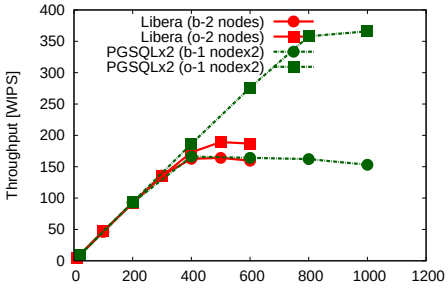

図 10　Response time under ordering mix

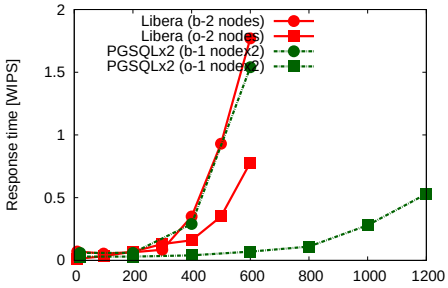

図 11　Scale out

5.4.1　ブラウジング負荷におけるスループット

最初の評価は，ブラウジング負荷における Libera と DV-SSI のスループットである．レプリカ数は 1, 2, 4 と変化させた．図 3 はブラウジング負荷のスループットである．x 軸は負荷（EB 数）であり y 軸はレプリカ数を変化させた時のスループットである．低負荷 (約 100 EBs) の場合，レプリカ数が変わったとしても Libera も DV-SSI も約 50 WIPS である．従って，低負荷の場合には Libera も DV-SSI もスループットに変化はほとんどない．高負荷の場合，Libera の最大スループットに変化があった．1 レプリカ，2 レプリカ，4 レプリカの時にそれぞれ 81.8 WIPS, 164 WIPS, 315 WIPS であった．レプリカ数を増やせば増やすほどスループットは線形に増加した．他方では，DV-SSI はレプリカ数を増やしても約 74 WIPS でありほとんど変わらない．

5.4.2　ブラウジング負荷における応答時間

第二の評価は，ブラウジング負荷における Libera と DV-SSI の応答時間である．レプリカ数は 1, 2, 4 と変化させた．図 4 はブラウジング負荷時の応答時間である．x 軸は負荷（EB 数）であり y 軸はレプリカ数を変化させた時の応答時間である．低負荷 (約 100 EBs) の場合，レプリカ数が変わったとしても Libera は約 0.06 s である．他方，DV-SSI の応答時間はレプリカ数が変わったとしても 0.20 s である．負荷が低い場合ではわずかに Libera が優位である．負荷が大きくなると Libera の最大レスポンス時間になる負荷に変化が現れる．つまり，1, 2, 4 レプリカでそれぞれ 300 EBs, 500 EBs, 900 EBs となる．他方で，DV-SSI はレプリカ数を変えてもそれほど変化はない．

5.4.3　ブラウジング負荷の拡張性

第三の評価は，Libera のスループットの拡張性である．第 5.4.1 と 第 5.4.2 ではレプリカ数を増やすとより高いスループットが実現できることが分かった．そこで，レプリカ数を 8 まで増やして最大スループットを評価した．図 5 はブラウジング負荷において，x 軸はレプリカ数，y 軸は正規化したスループットを示す．驚いたことに，スループットは 8 ノードまで線形に増加する．8 レプリカのシステムは 621.3 WIPS となりシングルレプリカシステムの 7.60 倍である．SGSI が 8 レプリカで 4 倍しか達成できてないこと，また，理想的な 8 倍に近いこと，などから考えると，Libera はオーバヘッドが少なく高性能化に長けていると言える．

5.4.4　オーダリング負荷時のスループット

第四の評価は，オーダリング負荷における Libera と DV-SSI のスループットである．レプリカ数は 1, 2, 4 と変化させた．

図 6 はオーダリング負荷のスループットである．x 軸は負荷 (EB 数) であり y 軸はレプリカ数を変化させた時のスループットである．ブラウジング負荷と同様に，低い負荷（約 100 EBs）では Libera も DV-SSI も約 50 WIPS である．負荷を高くすると，Libera では奇妙なことが起きる．レプリカの数を増や

すと，スループットが下がる．例えば，500 EBs の時は，Libera のスループットはレプリカ数 1, 2, 4 において 209 WIPS, 189 WIPS, 125 WIPS である．更新の負荷が高い場合には Libera は無視できないオーバヘッドが生じることが分かった．このオーバヘッドの詳細は 5.4.6 節で議論する．DV-SSI はどのような場合でもスループットは低い．

5.4.5 オーダリング負荷における応答時間

第 5 の評価は，オーダリング負荷における Libera と DV-SSI の応答時間である．レプリカ数は 1, 2, 4 と変化させた．図 7 はオーダリング負荷時の応答時間である．x 軸は負荷（EB 数）であり y 軸はレプリカ数を変化させた時の応答時間である．Libera はスループットと同様，反対の順序となった．つまり，最も小さい応答時間は 1 レプリカの時であり次に 2 レプリカの時である．この原因の解析は 5.4.6 節で議論する．DV-SSI は Libera の 4 レプリカのケースよりもさらに悪い．

5.4.6 Overhead

第六の評価は Libera のオーバヘッドの解析である．一般的にはピュアミドルウェアはデータベースエンジンを改造しないことから非効率なレプリケーションを引き起こし大きなオーバヘッドを発生させてしまう可能性がある．Libera では次の二つがオーバヘッドを発生させる原因となりうる．

(1) Same Snapshot Creation protocol
(2) Leader/Follower protocol

最初に，一つの PostgreSQL ノードを持った Libera と Libera を使わない一つの PostgreSQL を比較することによってオーバヘッド (1) を解析した．図 8 は一つの PostgreSQL ノードを持った Libera のスループットと Libera を使わない PostgreSQL のスループットを示す．オーダリング負荷では，Libera のスループット曲線は PostgreSQL の曲線のすぐ下に位置する．その結果，Libera によるスループット低下率は約 1.5% である．ブラウジング負荷では，Libera のスループット曲線は PostgreSQL の曲線にほとんどオーバラップする．図 9 は一つの PostgreSQL ノードを持った Libera の応答時間と Libera を使わない PostgreSQL の応答時間を示す．スループットのケースと同じように Libera のカーブと Libera を使わない PostgreSQL のカーブはほぼ一致する．従って，オーバヘッド (1) は無視できる．

次に，二つの PostgreSQL ノードを持った Libera のスループットと Libera を使わない PostgreSQL のスループットを 2 倍した値（PGSQLx2 と表記する）とを比較してオーバヘッド (2) を議論する．図 10 に二つの PostgreSQL ノードを持った Libera のスループットと PGSQLx2 を示す．ブラウジング負荷の場合，両カーブはほとんど一致する．従って，ブラウジング負荷の場合，Leader/Follower protocol のオーバヘッドはほとんど生じないと言える．他方でオーダリング負荷の場合を考える．低負荷の場合は Libera と PGSQLx2 は非常に近いが，高負荷の場合は Libera と PGSQLx2 は離れていて Libera の方が高い．以上より，Libera は更新が多く高負荷の場合に Leader/Follower protocol で無視できないオーバヘッドが生じることが分かった．

6. おわりに

1-Copy-SR を SQL 転送制御で提供する 1CS ルールを提案した．また，1CS を使って 1-Copy-SR を提供する Libera と呼ぶミドルウェアも提案した．Libera は SSI を使って 1-Copy-SR を実現する最初のミドルウェアである．Libera は (1) DBMS とクライアントに透過であること，(2) 無改造で市中技術のハードウェアとソフトウェアを使っていること，(3) readsets と writesets を処理したり，group communication を実装したり，専用の解析機を実装したり，という特殊でコストの高いことを必要としないピュアミドルウェアである．我々は無改造の PostgreSQL を使い Libera を実装した．規模は C 言語で 3000 ライン程度である．TPC-W ベンチマークを使って性能評価を行った．Libera は参照クエリが多い環境で従来技術よりも高いスループットを実現できた．驚くことに Libera はほとんどロスすることなく線形にスループットを向上できる．8 レプリカのシステムで単一レプリカのパフォーマンスの 7.60 倍である．Libera はシンプルであるばかりでなく，参照クエリが多い場合はレプリカ数が多いシステムを構築可能な現実的なシステムである．

文　献

[1] P. A. Bernstein, V. Hadzilacos and N. Goodman “Concurrency Control and Recovery in Database Systems”, Addison-Wesley Publishing Company, 1987.
[2] H. Berenson, P. Bernstein, J. Gray J. Melton, E.O’Neil and P. O’Neil “A Critique of ANSI SQL Isolation Levels” ACM SIGMOD, 1995.
[3] C. Amza, A. L. Cox and W. Zwaenepoel “Distributed Versioning: Consistent Replication for Scaling Back-end Databases of Dynamic Content Web Sites”
[4] J. Gray, P. Helland, P. O’Neil and D.Shasha “The Dangers of Replication and a Solution” ACM SIGMOD, 1996.
[5] Y. Lin, B. Kemme, M. Patiño-Martínez and R. Jiménez-Peris “Middleware based Data Replication providing Snapshot Isolation” ACM SIGMOD, 2005.
[6] T. Mishima and H. Nakamura “Pangea: An Eager Database Replication Middleware guaranteeing Snapshot Isolation without Modification of Database Servers” PVLDB, 2009.
[7] M. A. Bornea, O. Hodson, S. Elnikety and A. Fekete “One-Copy Serializability with Snapshot Isolation under the Hood” ICDE. 2011.
[8] M. J. Cahill, U. Röhm and A. D. Fekete “Serializable Isolation for Snapshot Databases” ACM SIGMOD, 2008.
[9] “blind write, http://en.wikipedia.org/wiki/Blind_write”
[10] H. Jung, H. Han, A. Fekete and U. Röhm “Serializable Snapshot Isolation for Replicated Databases in High-Update Scenarios” VLDB, 2011.
[11] A. Fekete, D. Liarokapis, E. O’Neil, P. O’Neil and D.Shasha “Making Snapshot Isolation Serializable”
[12] “Transaction Processing Performance Council, TPC-W”
[13] http://jgroups.org/manual/index.html
[14] http://www.cs.cornell.edu/Info/Projects/HORUS/Overview.html
[15] http://www.cs.technion.ac.il/Labs/csl/projects/Ensemble