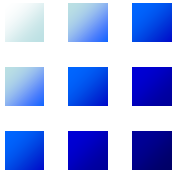


データベース
【13:障害回復】

石川　佳治

障害の分類

障害の分類

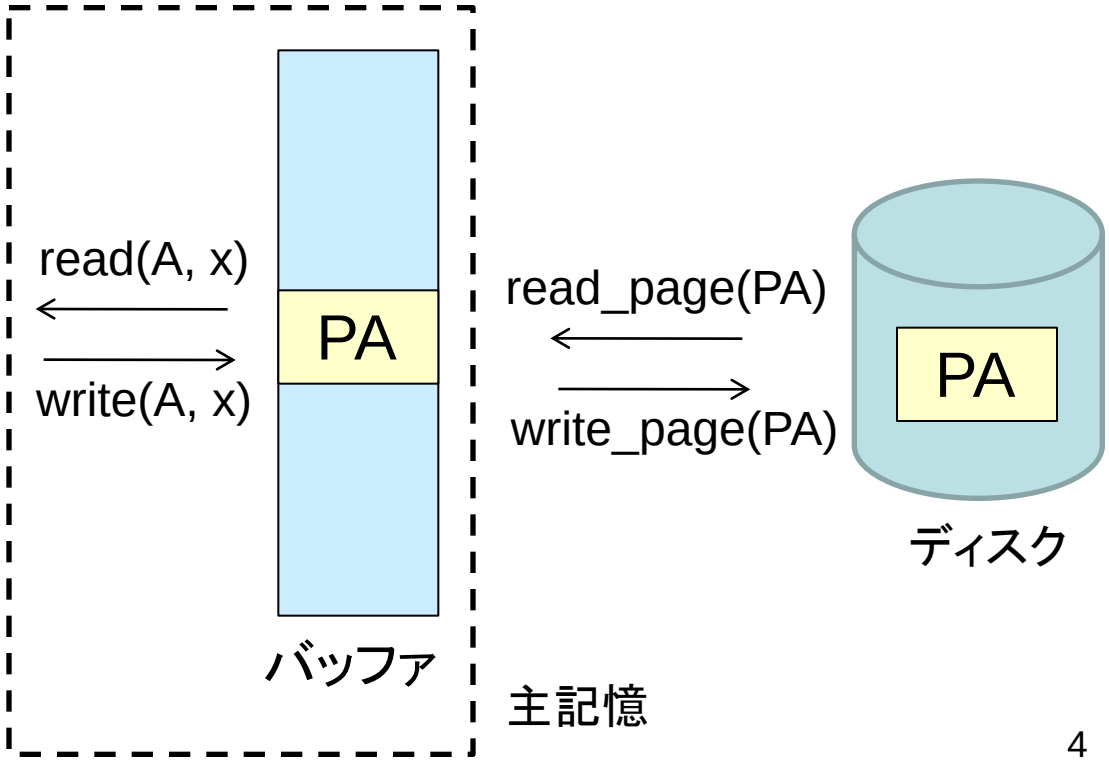
- トランザクション障害（transaction failure）
  - コミット前にトランザクションが異常終了
  - 理由
    - トランザクションが自主的にアボート
    - アプリケーションの実行時エラーによる強制終了
    - デッドロックの発生
- システム障害（system failure）
  - システムダウンが発生し，その時点のすべてのトランザクションの実行が異常終了
- メディア障害（media failure）
  - ディスク障害によりデータの読出しができない

障害への対応

- トランザクション障害の場合
  - トランザクションはアボートされる：ACID特性の原子性の要求により
  - そのトランザクションが障害前にデータ書込みをしていた場合，DBMSはそれらを取り消し元に戻す
- システム障害の場合
  - システム再起動にあわせてトランザクションをリスタート
    - 障害前の処理の後始末
    - コミット済トランザクションによるデータ書込みは確実にデータベースに反映

バッファリングの影響（1）

- ディスク上のページは，まず主記憶上のバッファに読み込まれる
- 書込みはバッファ上のページに対して実行
- バッファ管理機構がディスクに書込みを反映
  - 必ずしもバッファへの書込みとは同期しない
  - ダーティページ（dirty page：ディスク反映前のページ）が存在

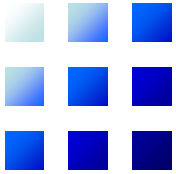


バッファリングの影響（2）

- 障害のケース例
- ケース1：①でシステム障害発生
  - write_page(PA)が障害前に実行済 ⇒ 整合がとれない結果に
  - write_page(PA)が未実行 ⇒ 整合がとれた状態
- ケース2：②で障害発生
  - バッファ上のA, Bの値はまだディスクに反映されていないかも ⇒ しかしコミットは済んでいる!

```
begin
read(A, x)
read(B, y)
x := x – 10000
y := y + 10000
write(A, x)   ← ①
write(B, y)
commit        ← ②
```

リスタート時に整合をとる必要性あり



ログを用いた障害回復

ログ（log）とは

- ジャーナル（journal）とも呼ばれる
- すべてのトランザクションがどのような基本操作を行ったかを逐次ディスクに記録したもの
  - ログ情報を用いてデータベースの値の復元やコミット済の更新内容のディスクへの反映が可能
- ログは，ログレコードの列からなる
  - 各ログレコードには一意的識別子であるLSN（log sequence number）を付与
- ログ先書きプロトコル（WALプロトコル; write ahead log protocol）
  - 更新トランザクションにより作成されたバッファ上のダーティページをコミット前にディスクに書き込む際は，それ以前にその更新に関するログレコードをディスクに書き込む

ログレコードの種類

① トランザクションTの開始（begin）： [B : T]

② Tによる書込み（write(A, x)）：
[W : T, A, Old, New]

- Oldは更新前の値（ビフォアイメージ）
- Newは更新後の値（アフタイメージ）

③ Tによる読出し（read(A, x)）: [R : T, A]

④ Tのコミット（commit）: [C : T]

⑤ Tのアボート（abort）: [A : T]

※ 読出しに関するログ（③）の記録は不要:「現在アクティブな状態にあるトランザクションがwriteを行った項目に対する他のトランザクションのreadまたはwriteは許さない」という厳格性の条件が成り立っている場合

ログを用いた障害回復

- ノーアンドゥ・リドゥ（no-undo/redo）方式
- アンドゥ・ノーリドゥ（undo/no-redo）方式
- アンドゥ・リドゥ（undo/redo）方式

ノーアンドゥ・リドゥ方式：アイデア

- トランザクション中でのwrite操作はログにのみ記録
  - トランザクションがコミット処理に入るまでディスク中のデータの更新は行わない
  - アボートされる可能性がある時点では，write前のビフォアイメージが必ずディスク内に保持
    - ログレコードにビフォアイメージを記録する必要なし

ノーアンドゥ・リドゥ方式：基本操作

A) トランザクションの開始：ログレコード [B：T] をログ用バッファに書き込む

B) トランザクションによる書込み：ログレコード [W：T, A, New] を書き込む

C) トランザクションのコミット：

- ログレコード [C：T] を書き，ログ用バッファをフラッシュ（強制書込み）
- ログの内容に基づき，トランザクション中でのwrite操作をデータベースに反映（遅延更新）
- フラッシュ処理が完了した時点でトランザクションがコミット済の状態に達する

D) トランザクションのアボート：ログレコード [A：T] を書きこむ

ノーアンドゥ・リドゥ方式：リスタート処理

- ログを調べ，ログレコード [C : T] で示されるコミット済トランザクション T を検出
- ログレコード [W : T, A, New] として記録されたそれらのトランザクションのwrite操作をデータベースに反映：リドゥ（redo）処理

アンドゥ・ノーリドゥ方式：アイデア

- トランザクションがコミット済の状態に入る前に，そのwrite操作をすべてディスク中のデータベースに反映
  - トランザクションがコミット済になった時点で，write後のアフタイメージがすでにディスク中のデータに反映されている
  - ログレコードへのアフタイメージの記録が不要

アンドゥ・ノーリドゥ方式：基本操作

A) 開始：ログレコード [B : T] を書き込む

B) トランザクションによる書込み：

- ログレコード [W : T, A, Old] を書き込む
- データ用バッファ上でAを含むページを更新

C) トランザクションのコミット：

- 本トランザクションに関するデータ用バッファをフラッシュ
- ログレコード [C : T] を書きこむ
- ログ用バッファをフラッシュ：完了時点でコミット済の状態に

D) トランザクションのアボート

- ログレコード [A : T] を書きこむ
- ログに基づき，writeした項目をビフォアイメージに戻す：アンドゥ（undo）処理

アンドゥ・ノーリドゥ方式：リスタート処理

- ログを調べ，ログレコード [C : T] のないアボート済あるいはアボートすべきトランザクションを検出
- ログレコード [W : T, A, Old] として記録されたそれらのトランザクションのwrite操作のアンドゥ処理を行う

アンドゥ・リドゥ方式

- 先の二つの手法の欠点への対応
  - ノーアンドゥ・リドゥ方式：コミット済となるまでディスク中のデータに書込みできない
  - アンドゥ・ノーリドゥ方式：コミットする前に必ずディスクへの書込みが必要で，ディスクアクセス増加
- ダーティページのディスクへの書込みをコミット処理と独立にする
  - バッファ ⇔ ディスク間のデータ転送の制約が少ない：効率化が可能
  - 処理はやや面倒
- 手法については省略

チェックポイント（checkpoint）

- アンドゥ・リドゥ方式でのリスタート処理は面倒
  - ログを解析して，必要とされるアンドゥ，リドゥ操作を判断して適用する必要あり
  - 一般にシステム起動時からのログは膨大：リスタート時の解析処理のコストが大
- 一定の周期で，バッファの内容を明示的にディスクにフラッシュ
  - リスタート時には，基本的には最新のチェックポイントから現在までのログレコードが解析対象
- 現在のDBMSで広く利用

シャドーページング（shadow paging）

- ログを用いない障害回復の方式
- アイデア
  - トランザクション処理において，あるページへの書込みが発生した場合，そのページのコピーを作成し，そこに書き込む
  - トランザクションがコミットした時点で元のページとすり替える
  - 障害が発生しても元のページがそのまま残っている
- 詳細は省略