



PostgreSQL におけるリカバリ処理

平成16年 7月26日

岡田 敏

日本電信電話株式会社 サイバースペース研究所

➤ 本日の内容

- リカバリ処理の概要
- リカバリ処理のプロセス構成
 - いつリカバリ処理が行われるか
- StartupXLOG() 関数の処理概要
 - リカバリ開始点の特定と取得
 - REDO によるリカバリ処理
 - 後処理

リカバリ処理の概要(1)

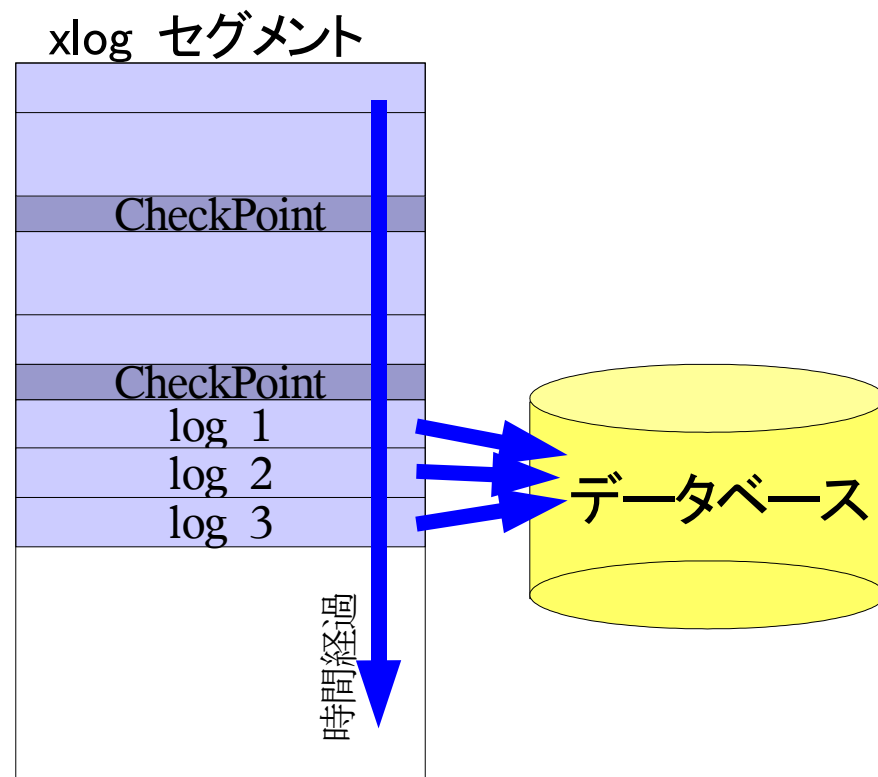
- リカバリできる障害は？
 - 現状(v.7.4.2) では xlog が残っているような障害が対象
→ xlog が無くなる障害(例: ディスクの破損など)からのリカバリはできない(これは PITR の範囲)。
- リカバリ処理の概略(方針)は？
 - xlog を使用し、REDO 処理(障害発生前に行われた処理をやり直す)ことによって可能な限り最新状態に戻す。

リカバリ処理の概要(2) 処理の流れ

- postmaster 起動
- 起動プロセス（リカバリを行う子プロセス）起動
 - REDO 処理の開始場所を特定する。
今回の起動でリカバリ処理の対象となる xlog の開始位置（チェックポイント）を特定する。
 - 順次ログを読み込み REDO 処理を行う。
特定された xlog の開始位置から、順に xlog を読み込み、REDO 処理を行い、データベースを最新状態にする。
- 起動プロセス終了

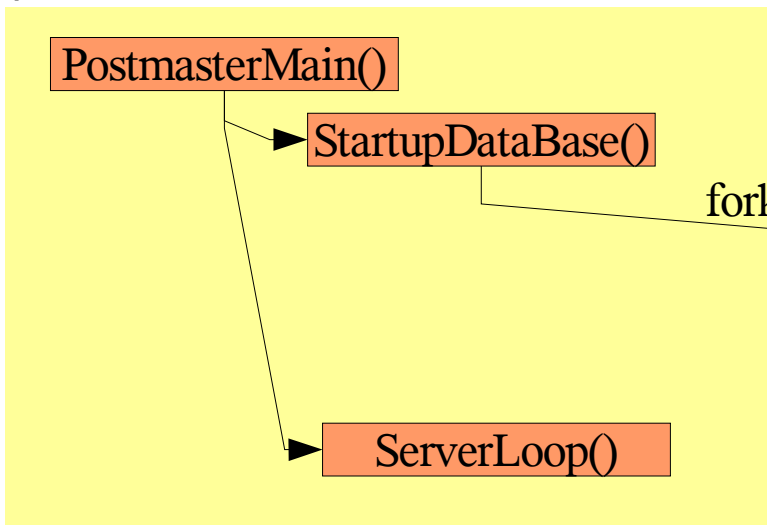
リカバリ処理の概要(2) 処理の流れ

- チェックポイント以後のログでなぜ良いのか？
 - チェックポイントの処理により、最新のチェックポイント以前の更新状況はデータベースに反映されている。
 - チェックポイント以後のログの内容はデータベースに反映されていない可能性があるため、チェックポイント以後のログを使って REDO 処理を行う。

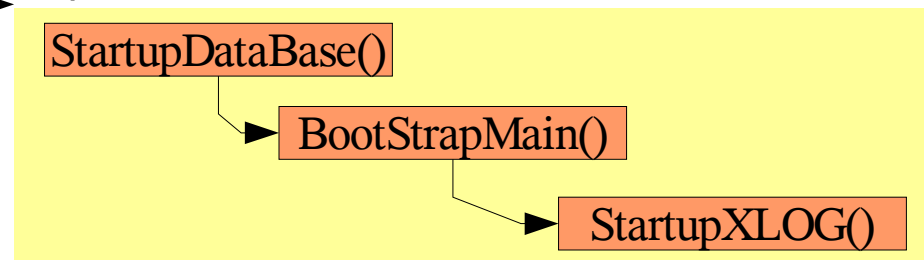


リカバリ処理のプロセス構成

postmaster プロセス

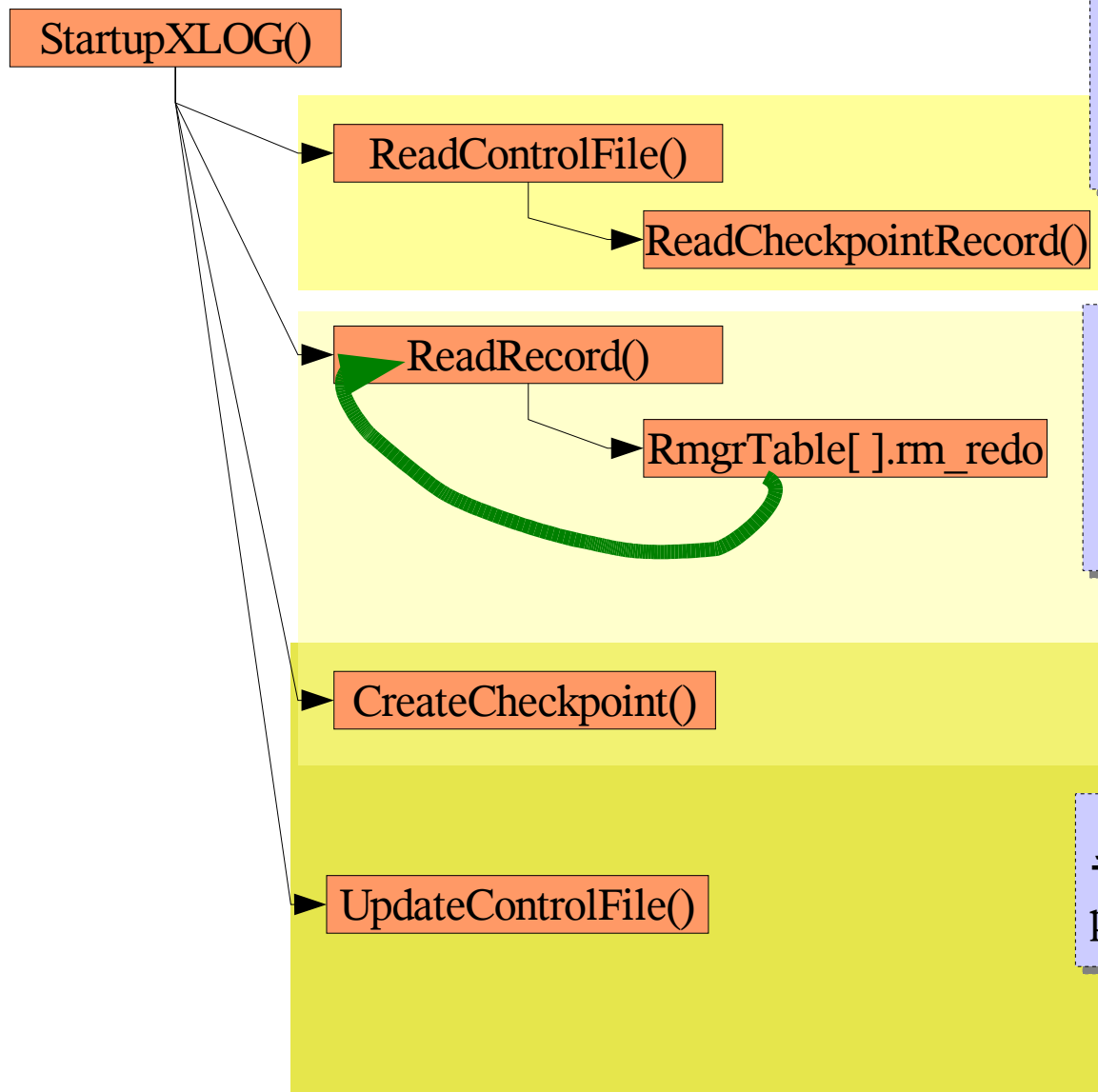


postmaster 子プロセス



- リカバリ処理は postmaster から fork された子プロセス中で実施される。
- リカバリ処理の中心は `StartupXLOG()` である。

StartupXLOG() の処理概要

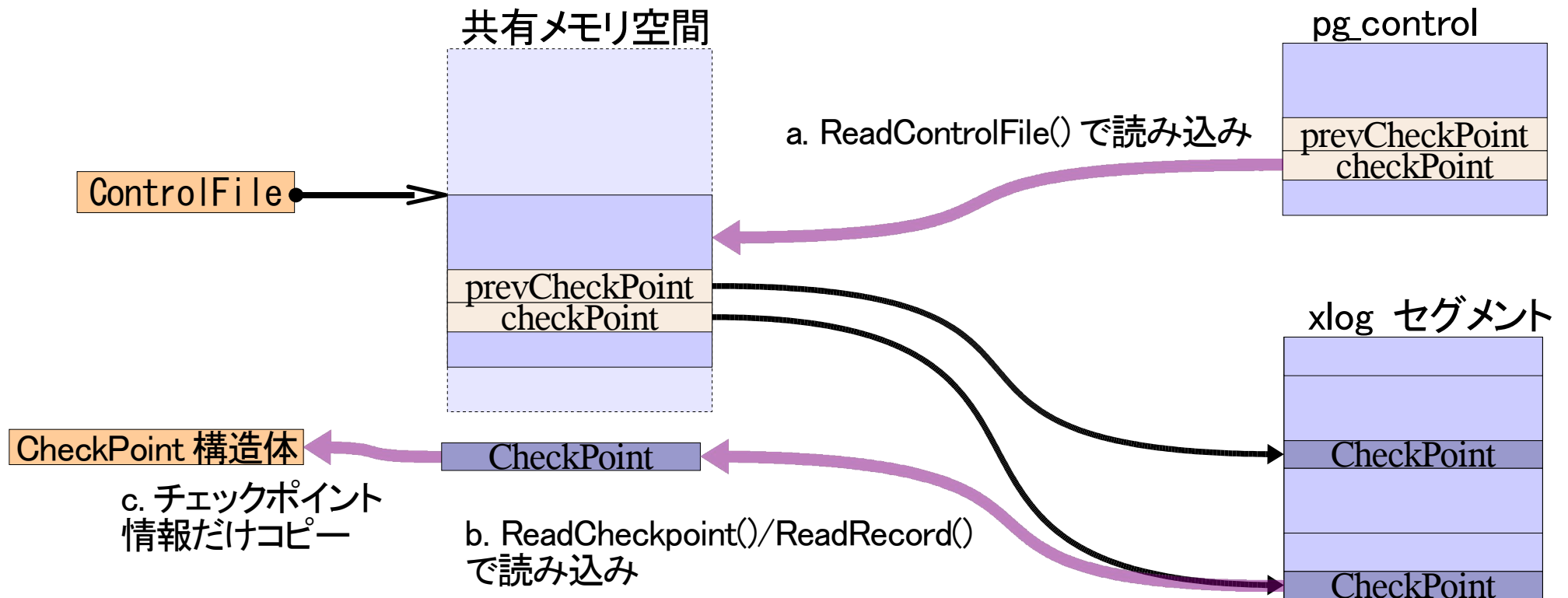


(1) リカバリ開始点の特定と取得
pg_control ファイルを読み込み、
pg_control が指定しているチェックポイント
のログを読み込む

(2) REDO によるリカバリ処理
ログレコードを xlog セグメントから順次
読み込み、REDO 処理を行う
※ ログがすべてデータへ反映されていれば
リカバリは必要なく、この処理は行われない

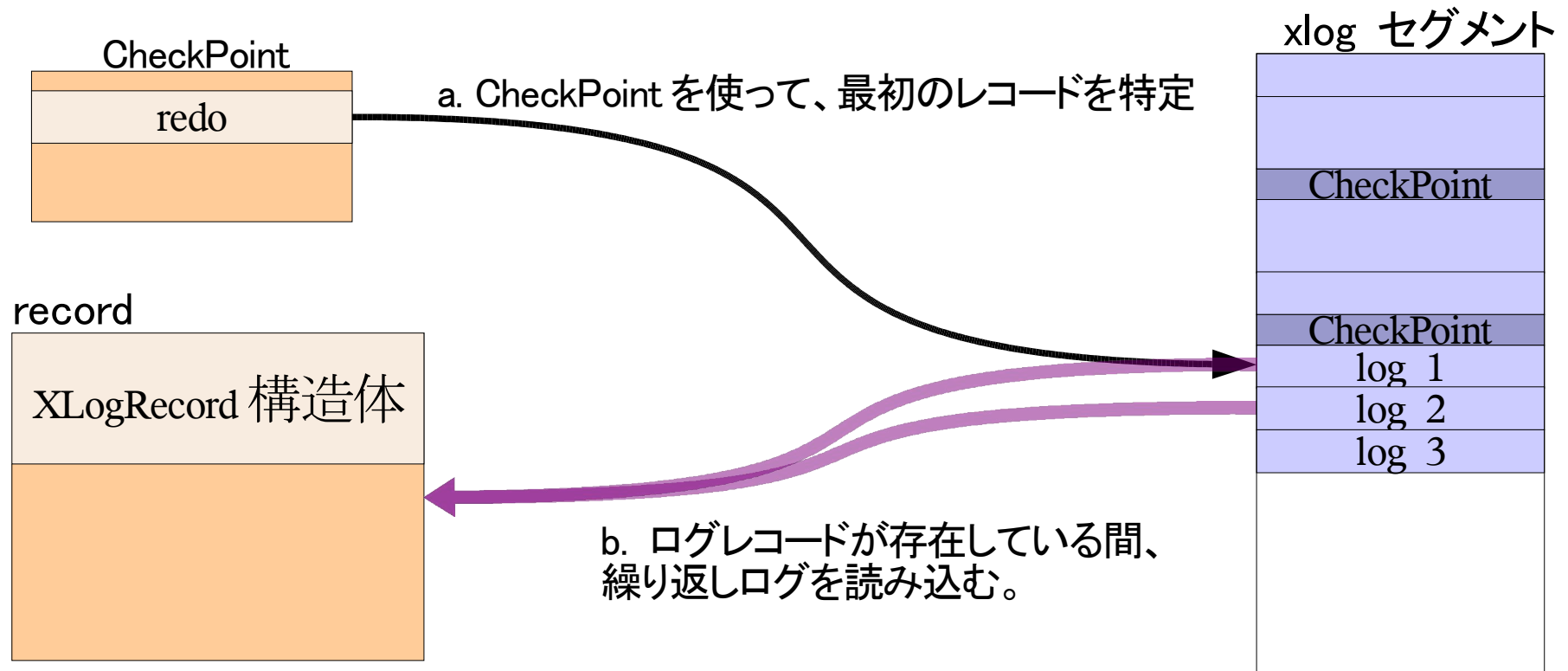
(3) 後処理
チェックポイントの取得
pg_control の更新

➤ (1) リカバリ開始点 (チェックポイント) の特定と取得



```
typedef struct CheckPoint
{
    XLogRecPtr redo; /* REDO を開始点を示す RecPtr */
    XLogRecPtr undo; /* ( 現在未使用 ) */
    StartUpID ThisStartUpID; /* current SUI */
    TransactionId nextXid; /* 次のフリーのトランザクション ID */
    Oid nextOid; /* 次のフリーのオブジェクト ID */
    time_t time; /* チェックポイントのタイムスタンプ */
} CheckPoint;
```

➤ (2) REDO によるリカバリ処理 ログレコードの読み込み方法



➤ (2) REDO によるリカバリ処理 取得したログレコードの内容

record

XLogRecord 構造体

```
typedef struct XLogRecord
{
    crc64          xl_crc;                /* このレコードのためのCRC */
    XLogRecPtr      xl_prev;              /* 一つ前のログの位置 */
                                           /* 一つ前のログの位置(同一トランザクション) */
    XLogRecPtr      xl_xact_prev;
    TransactionId   xl_xid;               /* トランザクション ID */
    uint16          xl_len;               /* rmgr データの全長 */
    uint8           xl_info;              /* フラグビット */
    RmgrId          xl_rmid;              /* リソースマネージャ */
    /* この構造体が続いて、実際のログデータが続く */
} XLogRecord;
```

XLogRecord 構造体を除いた部分は ...

- 各リソースマネージャが REDO 処理を行うために必要となる情報
- 物理ログ(REDO の対象となるデータページ)が含まれる場合もあり

➤ (2) REDO によるリカバリ処理 REDO を行うリソースマネージャ

- 取得したログから REDO を行うリソースマネージャを特定し、リソースマネージャに対応した REDO 用の関数を実行する。

リソースマネージャ名	redo を行う関数
XLOG	xlog_redo
Transaction	xact_redo
Storage	smgr_redo
CLOG	clog_redo
Heap	heap_redo
Btree	btree_redo
Btree	hash_redo
Rtree	rtree_redo
Gist	gist_redo
Sequence	seq_redo

➤ (3) 後処理

- チェックポイントの取得
 - ログをすべてデータベースに反映した時点で、チェックポイントを取得
- コントロールファイルの更新
 - 新規にチェックポイントを取得したので、その情報を反映するようにコントロールファイルを更新。

以上で、リカバリ処理終了。
クライアントからの処理が受け付けられるようになる。