JPUGしくみ分科会　第4回勉強会

PITR Proposalの解説

2004年4月19日

NTTコムウェア㈱
吉田 敏和

アジェンダ

- 現行のPostgreSQLのリカバリについて（WAL）
- 次期バージョンで実現しようとしているリカバリ機能（PITR）
- 現状ログとPITR

～PITRの機能紹介～

1. Xlog archival(xlogの保管)
1.1 XLogArchive API
1.1.1 XLogArchive APIの仕様
1.1.2 XLoxArchive APIの実装に関して
1.1.3 関数の呼び出し関係について
1.2 pg_archについて
2. RPIT
2.1 ログの最後までのリカバリ
2.2 利用可能なオンラインログの最後までのリカバリ
2.3 チェックポイント時間に関係するリカバリ
3 将来の拡張
4 PITRの現在の状況

現行のPostgreSQLのリカバリについて（WAL）

- **WAL(Write Ahead Logging)とは**
  - –通常トランザクションと呼ばれているものに相当する機能。
  - –PostgreSQL7.1から採用された仕組み。

- **WALの基本機能**
  - –トランザクションログがコミットしたときに、テーブル本体やインデックスにデータを書き込むだけでなくWALログファイル（xlog）にも、どのような操作を行ったかを逐一書き込む。
  - –テーブルやインデックスへの書き込みは「非同期書き込み」
  - –WALログへの書き込みは「同期書き込み」
    - →（テーブルやインデックスには）同期書き込みを行わないので、障害時には変更分が失われる可能性があるが、更新情報が保存されているWALログから復元可能。

- **WALの欠点**
  - –テーブルやインデックスが完全に失われるような障害はリカバリできない。
    - →チェックポイントを行うと古いWALログを消去するため。
  - –特定の時間にデータベースを戻せない。
    - →オペレーションミスが起こった時等を想定。

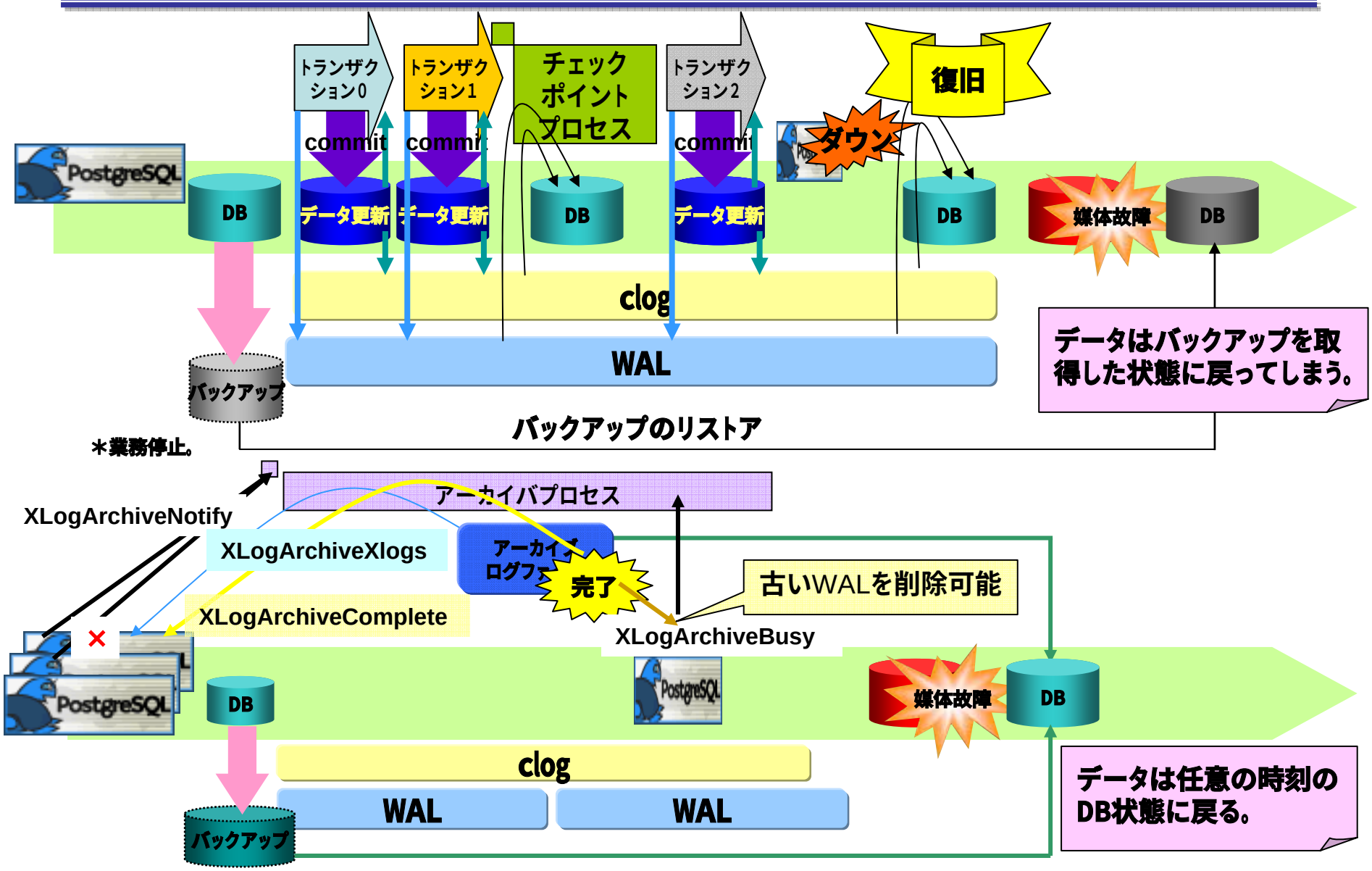
次期バージョンで実現しようとしている リカバリ機能 (PITR)

- **Point In Time Recoveryの事。人為的ミスや障害が発生した時、任意の時間までデータベースの内容を戻すことが出来る。**

- **既存のWALの仕様では障害復旧が不可能であるような状況がある。そのような状況でも復旧できるようにする機能。**
  - 障害復旧が不可能な状況とは？
    - データベースオブジェクトが削除された
    - ロールフォアードリカバリに必要な情報がxlogにない (又は不足)
    - データベースファイル自体が不完全

- **基本スタンス：今までの障害復旧機能を拡張する形で設計。**
- **PITRの近況： 最後に記載。**

ー現状ログとPITRー



PITRの機能紹介

1. xlog archival (xlogの保管)

- **xlogの保管についての考え方**
  - –市場には、バックアップとリカバリ機能を提供する製品群が存在する。それらも併用して使用できるようにarchival のAPIを考える必要がある。
  - –archival APIはPostgreSQLから直接実行される必要がある。そしてそれが流用され、広く利用されるようにAPIのリファレンスも作成する必要がある。

1.1. XLogArchive API

•XLogArchive APIの考え方

– PostgreSQLが提供している全てのxlogをアーカイブする必要があると思われる。

> 一部でもカバーできないものがあると、正常にリストアできない為

–PostgreSQLが起動した時、wal_archive_policyのパラメータ値をチェックして、アーカイブの有効･無効を決める。このパラメータ値はPostgreSQL起動時にのみ変更可能とする。

> ･XLogArchiveはバックエンドで実行すると、個々のプロセスが勝手にwal_archive_policyの値を上書きしてしまう可能性がある為。
> ･運用者が必要に応じてアーカイブの有無を指定出来た方がよい為。

–外部アーカイバがPostgreSQLと一緒に起動したほうがいいと思われる。

> 自動起動の場合、OSによって起動の順番が異なる可能性があるので、あまり厳密にはまだ決めていない。
> それと、アーカイバの処理時間もまだ把握していない為。

1.1. XLogArchive API -2

- **XLogArchive APIの考え方**
  - XLogArchive API**は**PostgreSQL**よりも先に終了しない。**postgreSQL**が終了すれば**
    XLogArchive API**も終了可能。**

> XLogArchive API**と**postmaster**は**1**対多の関係があるため、先に終了することはできない。**

1.1.1 XLogArchive API仕様

- **XLogArchive APIは主に4つの関数から構成されている。**
  - –XLogArchiveNotify(xlog)
  - –XLogArchiveXlogs()
  - –XLogArchiveComplete(xlog)
  - –XLogArchiveBusy(xlog)

  - – (XLogArchiveFree)

存在はするらしいが詳細は不明。。

1.1.1 XLogArchive API仕様 -2

- **XLogArchiveNotify(xlog)の仕様**

  PostgreSQLから呼ばれる関数

  – xlogへの書き込みが次のファイルへ移るときに、PostgreSQLのバックエンドプロセスから呼ばれる。
  – 返り値は、TRUE or FALSE
    - TRUE：処理の成功を示す。
    - FALSE：処理の異常を示す。それは、この処理が異常であるにも関わらず、アドミニストレータがアーカイブプロセスを実行した場合。

  – 個々のバックエンドプロセスによって呼び出される仕様になっているので、なるべく少ない時間で起動できるようにしなければならない。
  – 呼び出し最中にさらに呼び出すことは可能だが、まったく同時に2つ呼び出すことは不可能である。
  – 基本的にバックエンドは同時にこの関数を呼び出さないと考えられているが、1つの関数が処理中にもう1つの関数が呼ばれる事があるので、非同期で処理する仕様としている。

    このような時は、異なるxlogがアーカイブ可能である状態なので、xlogに対して論理ロックをかける必要はない。

1.1.1 XLogArchive API仕様 -3

- **XLogArchiveLogs()の仕様**

  xlogのファイル名を取得する関数。

  –アーカイバから定期的に呼び出され、xlogのファイル名を取得する。
  –xlogをアーカイブするときにこの関数を使用してファイル名を知ることが出来る。

1.1.1 XLogArchive API仕様 -4

•XLogArchiveComplete(xlog)の仕様

アーカイブをコピーするための関数

–**返り値は**SUCCESS、ALREADY_NOTTIFIED、SERVER_FAILURE**がある。**
- SUCCESS：**処理の成功を示す。**
- ALREADY_NOTTIFIED：**既に呼び出し済み。または実行中なので呼び出しロックをかけている。**
- SERVER_FAILURE：**何回か実行したが、全て失敗であった場合。**

–XLogArchiveComplete**はアーカイブ** (xlog) **のコピーを行うにあたり、リードオンリのアクセス権のみを持っている。**

> xlog**は絶対に変更や削除されてはならない為**

–**実行時間を定めることはしないが、出来るだけ高速で実行される必要がある。そのため、ネットワーク越しやテープデバイスへのコピーは遅いので基本的に行わない方針であるが、より現実的な関数とする為、コピープロセスを**2**段階に分けて速度低下を避ける。**

①ローカルディスクにコピー

②ネットワーク越し等の外部へコピー

> **非同期モードで関数が動く**

1.1.1 XLogArchive API仕様 -5

- **XLogArchiveBusy(xlog)の仕様**

  xlogが勝手に削除されないようにロックをかける関数

  –返り値はARCHIVE_OK、BUSY、SERVER_ERRORである。
    - ARCHIVE_OK：処理の成功を示す。アーカイブが完了したのでxlogを削除されてもOK
    - BUSY：archivercomplete（アーカイブが完了）というメッセージをまだ受け取っていないので、xlogはまだ削除することが出来ない状態。
    - SERVER_ERROR：その他のエラー。

  –xlogは今までの仕様では、チェックポイントが行われたりすると削除されるかクリアされ、再利用可能になっていたので、チェックポイントプロセスからXLogArchiveBusy(xlog)をコールして、xlogの削除を制御するようにする。

  –XLogArchiveBusy(xlog)はpostmasterと一対一で起動される関数である。

1.1.2 XLogArchive APIの実装に関して

- **XLogArchive APIの実装**
  - XLogArchive APIで使用するファイルはpg_xlogやpg_clogと同じ階層に、pg_rlogというディレクトリを作成して、そこに格納する。

もし何か予期せぬ事態が発生しても、既存のログファイル（xlog等）を削除することの無いように、格納先ディレクトリを新たに設けた。

```
/usr/local/pgsql/data
                    |-----base/
                    |-----global/
                    |-----pg_clog/
                    |-----pg_rlog/
                    |-----pg_xlog/
```

  - また、格納するファイルは、拡張子の名前によって処理の進み具合を表す仕様としている。

1.1.2 XLogArchive APIの実装に関して-2

- **XLogArchiveNotify(xlog)の実装**
  - –返り値は、TRUE or FALSE
    - ・TRUE：処理の成功を示す。
    - ・FALSE：通常ありえないが、アドミニストレータがアーカイブプロセスを実行した場合。
  - –<XLOG>.full というファイルを pg_rlogディレクトリ配下に書き出す。処理が成功すれば TRUEを返し、失敗すればFALSEを返す。

> <XLOG>とは？
> PostgreSQLがxlogのファイル名に使用している命名規則に
> 基づいて付与されるファイル名を指す。
> ファイルには、日付情報も付加されている。

  - –また、格納するファイルは、拡張子の名前によって処理の進み具合を表す仕様としている。

1.1.2 XLogArchive APIの実装に関して-3

- **XLogArchiveComplete(xlog)の実装**
  - –返り値はSUCCESS、ALREADY_NOTTIFIED、SERVER_FAILUREがある。
    - SUCCESS：処理の成功を示す。
    - ALREADY_NOTTIFIED：既に呼び出し済み。または実行中なのでロックをかけている。
    - SERVER_FAILURE：何回か実行されたが、全て失敗であった場合。
  - –xlogのコピー処理が完了したとき、pg_rlogの中にある<XLOG>の拡張子を .done に変更して、SUCCESSを返す。
  - –もしもその際、すでに<XLOG>.done が存在している場合はALREADY_NOTTIFIEDを返して処理を終了する。

1.1.2 XLogArchive APIの実装に関して-4

- **XLogArchiveBusy(xlog)の実装**
  - –返り値はARCHIVE_OK、BUSY、SERVER_ERRORである。
    - ・ARCHIVE_OK：処理の成功を示す。アーカイブが完了したのでxlogを削除されてもOK
    - ・ BUSY：archivercomplete（アーカイブが完了）というメッセージをまだ受け取っていないので、xlogはまだ削除することが出来ない状態。
    - ・ SERVER_ERROR：その他のエラー。
  - –pg_rlogの中に<XLOG>.doneが存在した場合は、 ARCHIVE_OKを返す。
  - –pg_rlogの中に<XLOG>.busyが存在した場合は、BUSYを返す。
  - –pg_rlogの中に<XLOG>.fullが存在した場合は、????。
    - →恐らくSERVER_ERRORを返し、処理をはじめからやり直すのでは？

1.1.2 XLogArchive APIの実装に関して-5

•ソースファイルについて

XLogArchiveの機能を実現する関数は以下のファイル名に追加する。

–src/backend/utils/misc/guc.c

- ⟨XLogArchiveの？⟩コンフィグパラメータを追加。

–src/backend/access/transam/xlog.c

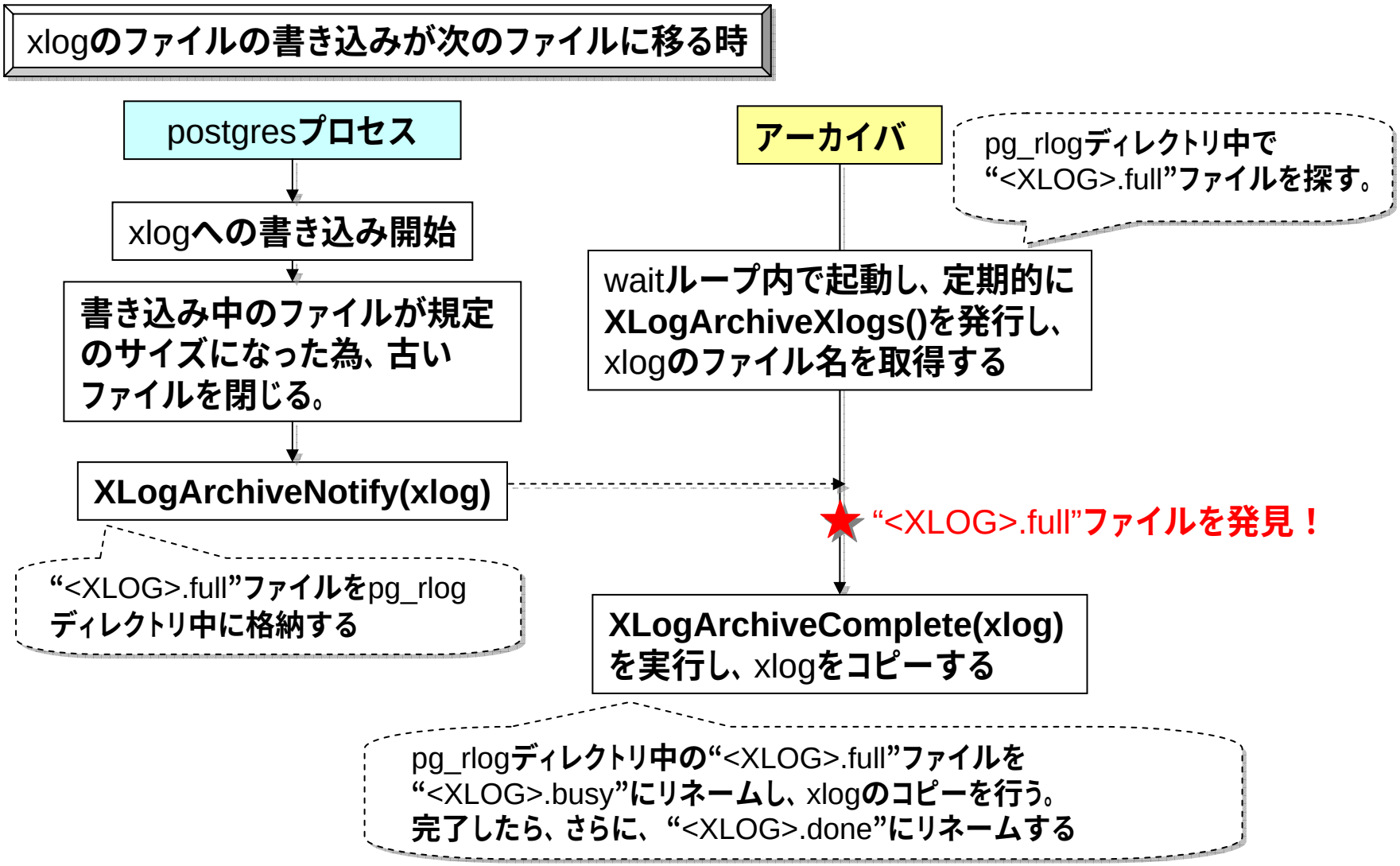
- XLogArchiveNotify(xlog) と XLogArchiveBusy(xlog)を追加。

–⟨src/interfaces/の下かな？⟩libpgarch.c

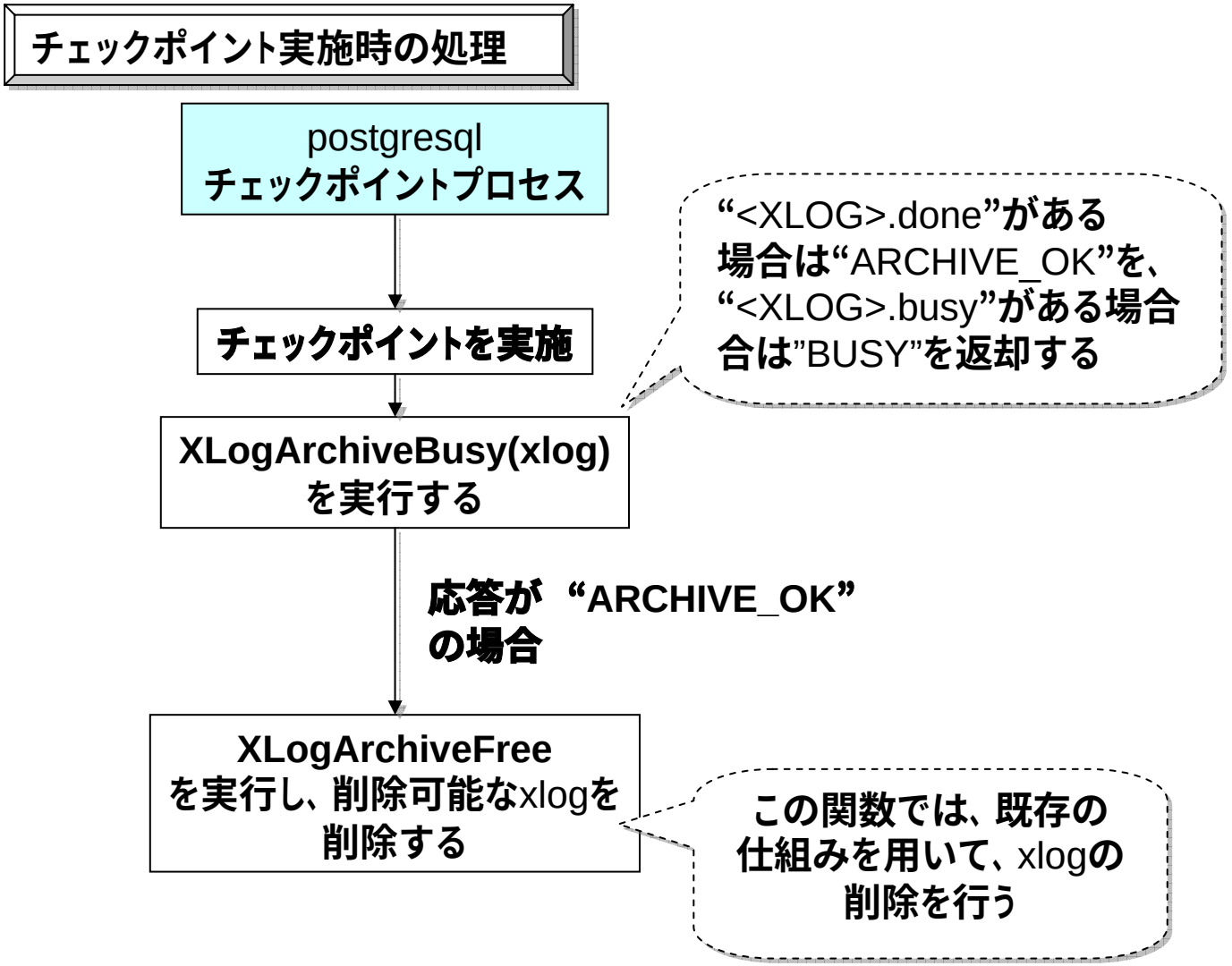
- XLogArchiveXlogs() と XlogArchiveComplete() を追加

–src/tools/の下に pg_arch.c を追加する。

1.1.3 関数の呼び出し関係について



1.1.3 関数の呼び出し関係について-2



1.2 pg_arch

- **pg_arch (簡単なxlogアーカイブツール)～の仕様**
  - –libpgarch.cを使用できるシングルスレッドプログラムである。
  - –pg_xlogのファイルを別ディレクトリへコピーする機能を持っている。
  - –foreground、backgroundどちらでも動作可能なプロセスにする。

> foregroundはテストなどの目的使用されることを想定して。
> backgroundはpostmasterが起動したときに一緒に起動される事を想定して。

  - –pg_archには以下のオプションを用意する。
    - -D <data-file>
    - -A <archive directory>

～Simon Riggs氏が後のメール(Date: Tue, 9 Mar 2004 22:38:19)でコメントしている内容～
pg_archはAPIの一部ではない。外部のディレクトリにxlogをコピーするだけの簡単なプログラム（サブツール的なもの）で、主にテスト用を考えている。
基本的にはAPIの一連の処理で必要なものをきちんとアーカイブ可能にしている。

2. RPIT (Recovery to Point-in-Time)

- **特定時刻へのリカバリ**

ーリカバリの方法には、以下の3つのいずれかを選択できる。

2.1 ログの最後までのリカバリ

2.2 利用可能なオンラインログの最後までリカバリ

2.3 チェックポイントが実行された時間、または、最後のチェックポイントの時間までリカバリ

ーpg_xlogディレクトリには、必ずxlogのアーカイブログを配置する必要がある。
（これは、手動または自動で起動される外部のアーカイバにより提供される機能。）

> xlogを取得するときにエラーがあった場合は、再度適切なxlogを選択し直せば良いので、リカバリを試行する回数に制限は設けない。

2.1 ログの最後までのリカバリ

- **ログの最後までのリカバリ**
  - –デフォルトのオプションであり、現在のPostgreSQLに変更を加える必要がない。
    アーカイブAPIは、このオプションにてテストするのが望ましい。

2.2 利用可能なオンラインログの最後までのリカバリ

•利用可能なオンラインログの最後までのリカバリ

–利用可能なログがリカバリされるまで、xlogのロールフォワードを実行した後に、postmasterをシャットダウンする方法である。

–この方法は、postmasterのコマンドラインのスイッチで、利用可能となる。

–この方法は次のような手順で、ログを分割してリカバリを行うことができる。

・xlogアーカイブが利用可能なディスク領域を超える場合：

① このモードで続けて実行し、管理者がバッチでリカバリする

② 最後のバッチまで達したら、コマンドラインのスイッチは必要ない。

2.3 チェックポイントに関係するリカバリ

- RPIT
  - リカバリパラメータを受け付けて、その時間に達したら、リカバリを中止する機能を追加する。

3. 将来的な拡張

•将来の拡張について

–一般的な通知 (notification) のインタフェースが提案されているが、それが利用可能であれば、それを利用する為のアーカイブＡＰＩを書き換えることは容易であろう。
（この問題は本開発では考慮外である。）

–このようなＡＰＩは、先進的な企業向けストレージ管理システムで容易に動作する XBSA,NDMPクライアントアプリケーションの基盤を作るのに用いることができるだろう。

4. PITRの現在の状況

- **PITRの開発状況**
  - –XLogArchive API用のPostgreSQL本体のクライアントサイドのコードは単体試験完了
  - –XLogArchive APIとpg_archを現在作成中。
    - •3月末の時点で、あと2週間ぐらいで0版が完成しそうと言っていたのだが、まだアナウンスがないので、恐らく今月中か!?

- **PITRの開発の全体スケジュール**
  - –第一段階：XLogArchive APIとpg_archの作成
    - •予想では4月中旬→4月中？
    - •xlog.c, xlog.h, guc.c のへの機能追加
      - –こりあえずこれが完成すればロールフォアードが可能となる。
  - –第二段階：RPITの作成
    - •5月中旬ぐらいに完成予定。
      - –これが完成すれば指定時間までのロールフォアードが可能となる。

4. PITRの現在の状況-2

- **PITRの開発の全体スケジュール**
  - –第三段階：その他のコード追加、マニュアル作成等。
    - •6月の上旬ぐらい完了したい。
  - –ここまでのステップが完了すると、PostgreSQL7.5X にPITR機能として、追加する予定である。

と順調に書いていますが、Simon氏のHDDがクラッシュしたとメールしていました。（もしかしてジョークか!?）

→恐らく上記のスケジュールよりはリリースが遅れるでしょう・・・