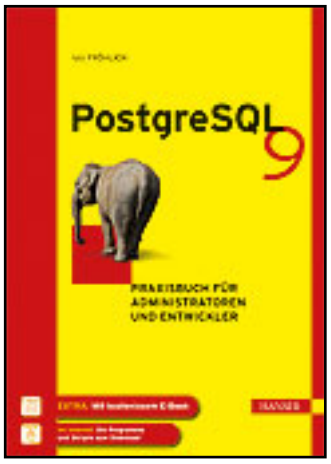
HANSER



Leseprobe

Lutz Fröhlich

PostgreSQL 9

Praxisbuch für Administratoren und Entwickler

ISBN (Buch): 978-3-446-42239-1

ISBN (E-Book): 978-3-446-42932-1

Weitere Informationen oder Bestellungen unter

http://www.hanser-fachbuch.de/978-3-446-42239-1

sowie im Buchhandel.

7 Hot Standby und Streaming Replication

„Standby-Datenbank“ und „Streaming Replication“ sind neue Features der Version 9, die PostgreSQL in der Beurteilung als Datenbank für Enterprise-Lösungen einen entscheidenden Schritt vorangebracht hat. Zusätzlich wurde mit dem Release 9.1 die synchrone Replikation bereitgestellt.

Zum Thema Hochverfügbarkeit von Datenbanken gibt es unterschiedliche Ansprüche und Lösungen. Aktiv-Passiv-Lösungen basieren auf der Spiegelung der zur Datenbank gehörenden Dateien auf Dateisystem- oder Storage-Ebene. So besteht im Open Source-Umfeld die Möglichkeit der Spiegelung auf Block-Device-Ebene mit DRBD, die in vielen Bereichen erfolgreich eingesetzt wird. Das Spiegeln auf Storage Ebene birgt die Gefahr des Spiegelns von Fehlern. Für beide Spiegelungen gilt, dass ein Performance-Impact für die Datenbank vorhanden sein kann. Während viele Standard-Datenbanken in dieser Architektur problemlos laufen, kann es bei Performance-kritischen Datenbanken zu Problemen kommen. Verursacher ist dabei häufig die erhöhte Latenzzeit für das Schreiben auf den Spiegel. Aus diesem Grund sollten die Spiegel des Storage nicht zu weit auseinander liegen.

Auf Trigger basierende Replikationslösungen werden immer seltener eingesetzt, da sie einen direkten Einfluss auf die Performance von Transaktionen haben und sehr schlecht skalieren.

Replikations-Methoden auf der Basis von Transaktionslog-Shipping besitzen die wenigsten Nachteile und bieten gleichzeitig eine hohe Sicherheit. Einerseits verursachen die asynchronen Prozesse den geringsten Impact auf die Performance der Primär-Datenbank, andererseits gestattet die Technologie eine sichere Übertragung auch bei hohem Transaktionsaufkommen. Außerdem kann die Replikation problemlos über größere Entfernungen durchgeführt werden.

Seit der Version 9 ist eine solche Replikation in PostgreSQL integriert und lässt sich durch Konfiguration der Standby-Datenbank aufsetzen. Darüber hinaus ist es möglich, die Standby-Datenbank im Read-only-Modus zu nutzen. So kann zum Beispiel das Reporting auf der Standby-Datenbank durchgeführt und somit die Primär-Datenbank entlastet werden. Dieser Modus wird „Hot Standby“ genannt. Damit bleiben die Ressourcen auf der Standby-Seite nicht ungenutzt.

Mit dem Release 9.1 ist ein weiteres Feature hinzugekommen: die synchrone Replikation. Diese kann durch einfache Konfigurationsänderung aktiviert werden. Dabei gilt eine Transaktion als abgeschlossen, wenn sie auf der Standby-Datenbank erfolgreich angewandt wurde. Diese Architektur liefert ein hohes Maß an Transaktionssicherheit.

Damit bieten sich die folgenden Einsatzmöglichkeiten für Standby-Datenbanken:

- Als Desaster-Recovery(DR)-Lösung mit asynchroner Replikation und ggf. minimalem Datenverlust.
- Als DR-Lösung mit synchroner Replikation ohne Datenverlust. Alle abgeschlossenen Transaktionen befinden sich in der Standby-Datenbank.
- Geplanter Rollentausch zum Patchen oder für Wartungsarbeiten.
- Zusätzliche Nutzung der Standby-Datenbank für Reporting und Abfragen zur Entlastung des Primärsystems (Hot Standby).

7.1 Eine Standby-Datenbank aufsetzen

In diesem Abschnitt liefern wir ein Beispiel zum Aufsetzen einer Standby-Datenbank. Die Replikation erfolgt zwischen zwei Linux-Servern, lässt sich jedoch analog auf andere Betriebssysteme übertragen.

7.1.1 Vorbereitung und Planung

Primär- und Standby-Server sollten so gleichartig wie möglich aufgesetzt werden. Insbesondere müssen die Pfade zu den Tablespaces identisch sein. Obwohl die Hardware nicht identisch sein muss, wird die Administration umso einfacher, je weniger Unterschiede bestehen. Insbesondere müssen folgende Übereinstimmungen vorliegen:

- Die Pfade zu den Tablespaces müssen existieren. Legt man auf der Primärdatenbank einen Tablespace an, wird dies durch die Replikation übertragen und muss auf dem Standby-Server nachvollziehbar sein.
- Wegen der notwendigen Übereinstimmung der Pfade für die Tablespaces ist eine Replikation zwischen Windows- und Unix-Betriebssystemen nicht möglich.
- Die Hardware-Architektur muss identisch sein, ein Mischen von 32-Bit- und 64-Bit-Systemen ist nicht möglich.
- Primär- und Standby-Datenbank müssen dasselbe Major Release besitzen. Da es Policy der Development Group ist, zwischen Minor Releases keine Änderungen in den Disk-Formaten vorzunehmen, besteht keine Gefahr, wenn es Abweichungen beim Minor Release gibt. Wann immer es möglich ist, sollten jedoch die Versionen identisch sein, um Probleme auszuschließen.

Neben der Planung der Server ist es wichtig, hinreichend Netzwerk-Kapazität zur Verfügung zu stellen. Planen Sie dabei nicht nur die durchschnittliche Übertragungsrate, sondern beachten Sie den notwendigen Durchsatz in der Spitze. Kann dieser nicht garantiert werden, kommt es zu einer Übertragungslücke und im Bedarfsfall zu Datenverlust.

7.1.2 Konfiguration und Aktivierung

Die Standby-Datenbank wendet permanent die WAL Records, die sie vom Primärsystem erhält, an und befindet sich damit im Recovery-Modus. Der Standby-Server ist in der Lage, WAL Records aus dem lokalen WAL-Archiv oder direkt vom Primär-Server über die TCP/IP-Verbindung zu lesen. Die letzte Methode wird *Streaming Replication* genannt. Darüber hinaus versucht der Standby-Server natürlich die WAL-Records im lokalen pg_xlog-Verzeichnis zu verarbeiten. Damit ist gewährleistet, dass auch nach einer Verzögerung der Recovery-Prozess an den aktuellen Status herangeführt werden kann.

Bevor wir mit dem Aufsetzen der Replikation beginnen, werfen wir einen Blick auf die Arbeitsweise des Anwendungs-Prozesses auf dem Standby-Server. Nach dem Start des Standby-Servers beginnt PostgreSQL unter Verwendung des Befehls im Parameter „restore_command“ alle WAL Records aus dem Archivlog-Verzeichnis zurückzuspeichern. Sind alle WAL-Dateien im Archivelog-Verzeichnis abgearbeitet, versucht PostgreSQL die WAL Records im pg_xlog-Verzeichnis zu verarbeiten. Ist „Streaming Replication“ konfiguriert, dann versucht der Server im nächsten Schritt, Verbindung mit dem Primärserver aufzunehmen und mit der direkten Übertragung zu beginnen. Andernfalls liest er das Archivelog-Verzeichnis wieder aus. Diese Schleife wird durchgeführt, solange der Standby-Server läuft. Es ist ein einfaches, aber wirkungsvolles Prinzip, mit dem in jeder Situation wieder aufgesetzt werden kann.

Voraussetzung für den Betrieb von Standby-Datenbanken ist, dass die WAL-Archivierung konfiguriert ist. Eine detaillierte Beschreibung zur Aktivierung finden Sie in Abschnitt 3.2. Für das Beispiel wiederholen wir diesen Vorgang auf der Primär- und der Standby-Datenbank. Die Schritte können analog für ein Windows-Betriebssystem angewandt werden.

1. Erstellen Sie ein Verzeichnis, das als Speicherort für die Archiv-Dateien dienen soll.

```
# cd /usr/local/pgsql
# mkdir archive
# chown postgres:postgres archive
```

2. Setzen Sie die notwendigen Parameter zur Aktivierung der Archivierung:

```
archive_mode = on
archive_command = 'cp -i %p /usr/local/pgsql/archive/%f </dev/null'
wal_level = archive
```

3. Führen Sie einen Neustart des PostgreSQL-Servers durch.

Auf dem Standby-Server muss nun der Recovery-Befehl verfügbar gemacht werden. Legen Sie dazu eine Datei recovery.conf im Verzeichnis $PGDATA an. Diese Datei muss den Parameter *restore_command* enthalten:

```
restore_command = 'cp /usr/local/pgsql/archive/%f %p'
```

Weitere Details zum Restore-Befehl finden Sie in Kapitel 4 („Sicherung und Wiederherstellung“).

Hilfreich ist in diesem Zusammenhang noch der Parameter *archive_cleanup_command*. Mit seiner Hilfe können ältere, nicht länger vom Standby-Server benötigte archivierte WAL-Dateien automatisch gelöscht werden. Der Parameter „%r“ steht für den Dateinamen mit

dem letzten gültigen Zeitpunkt für einen erfolgreichen Neustart. Der Parameter wird ebenfalls in der Datei „recovery.conf“ gesetzt.

```
archive_cleanup_command = 'pg_archivecleanup /usr/local/pgsql/archive %r'
```

Praxistipp

Wenn Sie planen, die Konfiguration als Desaster-Recovery-Umgebung einzusetzen oder wenn ein Rollentausch zur Durchführung von Wartungsarbeiten durchgeführt werden soll, dann fungiert der Primär-Server auch als Standby-Server und umgekehrt. Führen Sie deshalb die Vorbereitungsmaßnahmen unter diesem Blickwinkel durch. Alle Konfigurationen, die Sie auf dem Standby-Server vornehmen, sollten deshalb auch auf dem Primär-Server erfolgen (und umgekehrt), um einen Rollentausch erfolgreich durchführen zu können.

Schließlich müssen Sie dem Standby-Server noch mitteilen, dass er im Recovery-Modus arbeiten soll. Dies geschieht mit Hilfe des Parameters *standby_mode*. Die Datei „recovery.conf“ würde damit für unser Beispiel wie folgt aussehen:

Listing 7.1 Beispiel für die Konfigurationsdatei recovery.conf

```
standby_mode=on
restore_command = 'cp /usr/local/pgsql/archive/%f %p'
archive_cleanup_command = 'pg_archivecleanup /usr/local/pgsql/archive %r'
```

Schließlich muss noch das WAL Shipping aktiviert werden. Dies übernimmt der PostgreSQL-Server. Setzen Sie dazu den Parameter „archive_command“ wie folgt:

```
archive_command = 'cp -i %p /usr/local/pgsql/archive/%f </dev/null; scp %p
192.168.178.212:/usr/local/pgsql/archive/%f </dev/null'
```

Damit sind alle Vorbereitungen für den Betrieb der Standby-Datenbank getroffen. Für das initiale Setup muss nun der Standby-Server auf den Stand des Primär-Servers gebracht werden. Die sicherste Methode ist, ein Backup auf den Standby-Server zu übertragen. Führen Sie dazu die folgenden Schritte durch:

1. Versetzen Sie den Primär-Server in den Backup-Modus.

```
(postgres@[local]:5432) [postgres] > SELECT pg_start_backup('standby');
 pg_start_backup
-----------------
 0/7000020
```

2. Kopieren Sie den Inhalt des Verzeichnisses $PGDATA auf den Standby-Server. Sichern Sie vorher auf dem Standby-Server die Konfigurationsdateien „postgresql.conf“ und „recovery.conf“.

```
$ cd $PGDATA
$ tar -cvf prim.tar *
$ scp prim.tar 192.168.178.212:$PGDATA
. . .
tar -xvf prim.tar
```

```
cp postgresql.conf.ori postgresql.conf
$ cp recovery.conf.ori recovery.conf
```

3. Beenden Sie den Backup-Modus auf dem Primär-Server.

```
(postgres@[local]:5432) [postgres] > SELECT pg_stop_backup();
NOTICE:  pg_stop_backup complete, all required WAL segments have been archived
 pg_stop_backup
----------------
 0/7000094
```

4. Starten Sie den Standby-Server.

```
$ pg_ctl start
pg_ctl: another server might be running; trying to start server anyway
server starting
$ LOG:  database system was interrupted; last known up at 2011-10-17
 16:22:35 CEST
LOG:  entering standby mode
LOG:  restored log file "000000010000000000000007" from archive
LOG:  redo starts at 0/7000070
LOG:  consistent recovery state reached at 0/8000000
cp: cannot stat `/usr/local/pgsql/archive/000000010000000000000008': No
such file or directory
LOG:  unexpected pageaddr 0/4000000 in log file 0, segment 8, offset 0
```

Damit befindet sich die Standby-Datenbank im permanenten Recovery-Modus. Die archivierten WAL-Dateien werden zum Standby-Server übertragen. Mit dem folgenden Test lässt sich feststellen, ob die Replikation funktioniert.

5. Führen Sie auf der Primär-Datenbank eine DML-Anweisung durch.

```
(postgres@[local]:5432) [postgres] > INSERT INTO test VALUES(1,'FROM PRIMARY');
INSERT 0 1
```

6. Da eine Replikation erst mit der Übertragung der nächsten WAL-Datei stattfindet, forcieren wir an dieser Stelle einen Log Switch.

```
(postgres@[local]:5432) [postgres] > SELECT pg_switch_xlog();
 pg_switch_xlog
----------------
 0/8000110
```

7. Simulieren Sie eine Failover-Situation, und aktivieren Sie die Standby-Datenbank. Starten Sie dazu die Datenbank mit dem Parameter „standby_mode=off“.

```
$ pg_ctl stop
$ pg_ctl start
. . .
$ psql
(postgres@[local]:5432) [postgres] > SELECT * FROM test;
 id |     text
----+--------------
  1 | FROM PRIMARYpostgres=# SELECT * FROM test;
 id |     text
----+--------------
  1 | FROM PRIMARY
```

Damit ist der Nachweis erbracht, dass die Replikation funktioniert. Ein Nachteil dieser Methode ist, dass eine Anwendung der WAL Records auf dem Standby-Server erst nach Übertragung einer archivierten WAL-Datei erfolgt. Damit ist im Havarie-Fall ein Datenverlust denkbar. Dieser Nachteil wird durch das Feature „Streaming Replication“ ausgeglichen.

7.1.3 Streaming Replication einsetzen

Beim Einsatz von „Streaming Replication“ erfolgt die Übertragung der WAL Records nicht erst beim Log Switch, wenn eine neue WAL-Datei geöffnet wird, sondern direkt, mit einer minimalen Verzögerung beim Schreiben des WAL Records. Es handelt sich in der Standard-Konfiguration immer noch um einen asynchronen Prozess, so dass der Performance-Impact auf die Primär-Datenbank sehr klein ist. Andererseits erfolgt bei einer gut geplanten Infrastruktur die Übertragung zum Standby-Server durchschnittlich in weniger als einer Sekunde.

Als Aufsatzpunkt für die Einrichtung der „Streaming Replication“ dient die Konfiguration mit Datei-basierender WAL-Übertragung aus dem vorhergehenden Abschnitt. Für die Verwendung von „Streaming Replication“ ist eine Verbindung vom Standby- zum Primär-Server erforderlich. Die Verbindung sollte über einen so genannten „technischen User“ erfolgen. Das Passwort eines technischen Users darf niemals ablaufen. Für eine funktionierende Verbindung sind zwei Schritte erforderlich:

- Den technischen User im Primär-Server anlegen:

```
(postgres@[local]:5432) [postgres] > CREATE ROLE rep_admin
> LOGIN REPLICATION PASSWORD 'rep1234';
CREATE ROLE
```

- Die Verbindung in der Datei „pg_hba.conf“ gestatten:

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host    replication     rep_admin       192.168.178.212/32      md5
```

Zusätzlich muss der Parameter *max_wal_senders*, der die maximale Anzahl von gleichzeitigen Standby-Verbindungen festlegt, auf einen Wert größer als „Null“ gesetzt werden. Führen Sie anschließend einen Neustart des Servers durch, damit die Änderungen wirksam werden.

Der Prozess wird aktiviert durch Setzen des Parameters *primary_conninfo* in der Datei „recovery.conf“ auf dem Standby-Server. Im Beispiel sieht die Datei wie folgt aus:

```
standby_mode=on
restore_command = 'cp /usr/local/pgsql/archive/%f %p </dev/null'
archive_cleanup_command = 'pg_archivecleanup /usr/local/pgsql/archive %r'
primary_conninfo = 'host=192.168.178.100 port=5432 user=rep_admin
password=rep1234'
```

Starten Sie jetzt den Standby-Server. Er wird sich zum Primär-Server verbinden, wenn alle archivierten WAL-Dateien abgearbeitet sind und in den Modus „Streaming Replication“ übergehen. Sie sollten als Bestätigung die folgende Meldung im Server-Logfile oder auf dem Bildschirm finden:

```
LOG:  streaming replication successfully connected to primary
```

Weiterhin sollte auf dem Primär-Server ein Sender- und auf der Standby-Seite ein Receiver-Prozess laufen:

```
$ ps -ef|grep wal
postgres  6083  5937  0 22:05 ?        00:00:00 postgres: wal sender process
rep_admin 192.168.178.212(24389) streaming 0/16000070
. . .
$ ps -ef|grep wal
postgres  4118  4101  0 22:38 ?        00:00:00 postgres: wal receiver
process   streaming 0/16000070
```

7.1.4 Die Replikation überwachen

Eine einfache, aber zuverlässige Methode für die Überwachung ist die Anzahl von WAL Records, die auf dem Primär-Server generiert und noch nicht auf dem Standby-Server eingearbeitet wurden. Während die Funktion *pg_current_xlog_location* den aktuellen WAL Record anzeigt, erhält man mit der View *pg_stat_replication* den Status des Receiver-Prozesses. Größere Differenzen zwischen dem aktuellen und dem übertragenen WAL Record (Spalte „sent_location“) bedeuten, dass es bei der Übertragung zu Verzögerungen im Streaming Replication-Prozess gekommen ist. Im Beispiel in Listing 7.2 ist die Replikation synchronisiert.

Listing 7.2 Überwachung des Streaming-Replication-Prozesses

```
(postgres@[local]:5432) [postgres] > SELECT * FROM pg_current_xlog_location();
 pg_current_xlog_location
--------------------------
 0/16000070
(postgres@[local]:5432) [postgres] > SELECT
application_name,state,sync_state,sent_location,write_location
> FROM pg_stat_replication;
 application_name |   state   | sync_state | sent_location | write_location
------------------+-----------+------------+---------------+---------------
 walreceiver      | streaming | async      | 0/16000070    | 0/16000070
```

7.1.5 Synchrone Replikation

Bisher haben wir das Streaming-Replication-Feature als asynchronen Prozess kennengelernt. Dabei sind wir uns bewusst, dass im Falle eines Crashs des Primär-Servers möglicherweise wenige abgeschlossene Transaktionen, die noch nicht auf den Standby-Server übertragen wurden, verloren gehen können. Wir haben dies in Kauf genommen und einen sinnvollen Kompromiss zwischen Ausfallsicherheit und Performance-Impact erzielt.

Falls für eine Datenbank die Ausfallsicherheit eine höhere Priorität als die Performance hat, empfiehlt sich die synchrone Replikation. Dieses Feature steht mit der Version 9.1 zur Verfügung. Die Bestätigung eines COMMIT-Befehls wird erst erfolgreich erteilt, wenn die Transaktion erfolgreich in das Transaktions-Logfile auf dem Primär- und dem Standby-Server geschrieben wurde. Damit ist ein Datenverlust für abgeschlossene Transaktionen bei Ausfall des Primär-Servers ausgeschlossen.

Eingeschaltet wird die synchrone Replikation durch Setzen des Parameters *synchronous_standby_names*. Der Parameter enthält durch Komma getrennte Namen des Applikationsnamens der Standby-Server. Der Applikationsname lässt sich durch folgende SQL-Abfrage herausfinden:

```
(postgres@[local]:5432) [postgres] > SELECT application_name FROM pg_stat_
replication;
 application_name
------------------
 walreceiver
```

Alternativ ist eine Wildcard zulässig in der Form `synchronous_standby_names= '*'`. Außerdem muss der Parameter *synchronous_commit* auf „on“ gesetzt sein, allerdings ist dies Standard.

Eine Bestätigung, dass die synchrone Replikation eingeschaltet ist, liefert ebenfalls die View „pg_stat_replication“ (siehe Listing 7.3).

Listing 7.3 Den Status der Replikation abfragen

```
(postgres@[local]:5432) [postgres] > SELECT application_name, sync_state
> FROM pg_stat_replication;
 application_name | sync_state
------------------+------------
 walreceiver      | sync
```

Es stellt sich die Frage, wie groß der Impact auf die Performance ist. In einem Vergleich zwischen synchroner und asynchroner Replikation werden eine Million Sätze in eine Tabelle eingefügt. Die Laufzeit mit asynchroner Replikation war ca. 12 Sekunden, mit synchroner ca. 15 Sekunden. Der Unterschied ist eher klein, insbesondere da der Standby-Server weniger Ressourcen zur Verfügung hat. Allerdings stehen die Server nebeneinander, so dass die Netzwerk-Roundtrip Zeiten sehr klein sind. Im Falle einer Replikation über größere Entfernungen werden die Unterschiede größer sein. Störfaktor ist dabei weniger die Durchsatzrate, sondern die Latenzzeit. Die kleinstmögliche Verzögerung pro Transaktion ist die Zeit für einen Netzwerk-Roundtrip.

7.2 Eine Hot-Standby-Datenbank betreiben

Eine Hot-Standby-Datenbank verfügt auch über die Fähigkeit, SQL-Abfragen ausführen zu können, während die Datenbank im Recovery-Modus läuft. Damit ist also ein paralleler Read-only-Betrieb möglich. Die Nutzungsmöglichkeiten sind zahlreich. Häufig werden solche Datenbanken für das Reporting und Ad-hoc-Abfragen eingesetzt und entlasten somit den Primär-Server. Transaktionen, die auf der Hot-Standby-Datenbank gestartet werden, erhalten aufgrund des Read-only-Modus keine Transaktions-ID. Demzufolge können nur Transaktionen ausgeführt werden, die keine Veränderungen vornehmen.

Hinweis

Bei der Datenübertragung zwischen Primär- und Standby-Datenbank kann es zu Verzögerungen kommen. Aus diesem Grund können Abfragen, die auf beiden Systemen gleichzeitig abgesetzt werden, unterschiedliche Ergebnisse liefern. Deshalb sollte man davon ausgehen, dass der Datenbestand auf dem Standby-Server nur als nahezu identisch zu betrachten ist. Die ist bei der Planung des Einsatzes der Standby-Datenbank als Quelle für das Reporting zu berücksichtigen. In vielen Fällen ist jedoch diese kurzfristige Differenz unerheblich.

Der Hot-Standby-Modus wird eingeschaltet, wenn der Parameter *hot_standby* auf den Wert „on“ gesetzt wird und eine Datei „recovery.conf“ existiert. Darüber hinaus muss der Parameter *wal_level* in der Primär-Datenbank auf den Wert „hot_standby“ gesetzt werden. Nach Aktivierung der Hot-Standby-Datenbank finden Sie im Logfile die folgende Meldung:

```
LOG:  database system is ready to accept read only connections
LOG:  standby "walreceiver" is now the synchronous standby with priority 1
```

Auf der Host Standby-Datenbank ist der Parameter *transaction_read_only* stets auf „on“ gesetzt und lässt sich nicht verändern:

```
(postgres@[local]:5432) [postgres] > show transaction_read_only;
 transaction_read_only
-----------------------
 on
```

Schauen wir uns an dieser Stelle weitere Parameter an, die Einfluss auf die Konfiguration einer Hot-Standby-Umgebung haben. In der Primär-Datenbank sind folgende Parameter einzustellen:

- *wal_level:* Dieser Parameter legt fest, wie viele Informationen in die WAL-Dateien geschrieben werden sollen. Bei „minimal“ werden gerade so viele Daten geschrieben, wie notwendig sind, um das Crash-Recovery der Primärdatenbank zu gewährleisten. Um eine Hot-Standby-Datenbank betreiben zu können, muss der Parameter auf den Wert „hot_standby“ gesetzt werden.
- *vacuum_defer_cleanup_age:* Der Parameter legt die Anzahl von Transaktionen fest, um die die Beseitigung von Dead-Row-Versionen verzögert wird. Eine Vergrößerung des Wertes ermöglicht dem Standby-Server, mehr SQL-Abfragen zu bedienen, ohne dass es zu Konflikten mit zu früh entfernten Row-Versionen kommt. Der Standardwert ist „0“ und sollte beim Betrieb einer Hot-Standby-Umgebung in Abhängigkeit vom Transaktions-Aufkommen nach oben gesetzt werden.

Auf dem Standby-Server haben folgende Parameter Einfluss auf den Betrieb:

- *max_standby_archive_delay:* Ist die Hot-Standby-Datenbank eingeschaltet, legt dieser Parameter fest, wie lange der Standby-Server warten soll, bis er die Abfrage wegen eines Konflikts (About-to-be-applied) abbricht. Dieser Parameter bezieht sich auf eine Situation, wenn der Standby-Server aus einer archivierten WAL-Datei liest.
- *max_standby_streaming_delay:* Dieser Parameter besitzt eine analoge Funktion wie der Parameter „max_standby_archive_delay“, in einer Situation, in der die Standby-Datenbank WAL-Daten über „Streaming Replication“ empfängt.

Praxistipp

Beachten Sie beim Festlegen der Parameter, dass die Datenbank-Server beide Rollen, Primary und Standby übernehmen können.

In Zusammenhang mit SQL-Abfrage auf einer Hot-Standby-Datenbank kann es zu Konflikten kommen. Ein typisches Beispiel ist das Absetzen eines DROP TABLE-Befehls auf der Primär-Datenbank, während eine SELECT-Anweisung auf dem Standby-Server läuft. Die Abfrage auf dem Standby-Server könnte nicht zu Ende laufen, wenn der DROP TABLE-Befehl abgesetzt würde. Da der Primär-Server keine Informationen darüber hat, welche SQL-Abfragen auf dem Standby-Server laufen, wird er den DROP TABLE-Befehl direkt ausführen. Andererseits würde der Standby-Server in seinem APPLY-Prozess zurückfallen, wenn er auf das Ende der SELECT-Anweisung warten würde.

Um einen Kompromiss herbeizuführen, können Sie die beiden soeben erwähnten Parameter „ max_standby_streaming_delay“ und „ max_standby_archive_delay“ setzen.

Eine Zusammenfassung von aufgetretenen Konflikten liefert die Tabelle *pg_stat_database_conflicts*.

Listing 7.4 Konflikte auf der Hot Standby-Datenbank abfragen

```
(postgres@[local]:5432) [postgres] > SELECT * FROM pg_stat_database_conflicts;
 datid |  datname  | confl_tablespace | confl_lock | confl_snapshot | confl_
bufferpin | confl_deadlock
-------+-----------+------------------+------------+----------------+--------
---------+----------------
     1 | template1 |                0 |          0 |              0 |
0 |              0
 12690 | template0 |                0 |          0 |              0 |
0 |              0
 12698 | postgres  |                0 |          0 |              0 |
0 |              0
```

Da wir nun in der Lage sind, Abfragen an den Standby-Server zu schicken, ist ein direkter Vergleich der angewandten WAL Records möglich. Es lässt sich also feststellen, ob eine Anwendungslücke im Recovery-Prozess vorliegt (siehe Listing 7.5).

Listing 7.5 Den Status von Primär- und Standby-Server vergleichen

```
(postgres@[local]:5432) [postgres] > SELECT * FROM pg_current_xlog_location();
 pg_current_xlog_location
--------------------------
 0/500000A0
postgres=# SELECT * FROM pg_last_xlog_receive_location();
 pg_last_xlog_receive_location
-------------------------------
 0/500000A0
```

7.3 Failover und Switchover

Während „Switchover“ einen geplanten Rollentausch zwischen Primär- und Standby-Datenbank bezeichnet, ist ein „Failover“ die Übernahme der Primär-Rolle durch den bisherigen Standby-Server bei Ausfall des Primärservers.

PostgreSQL stellt keine Standard-Software für die Verwaltung von Switchover und Failover zur Verfügung. Wird die Standby-Datenbank für Desaster-Recovery-Zwecke eingesetzt, dann ist das Failover in der Regel ein manueller Prozess. Nach dem Crash des Primär-Servers wird die Standby-Datenbank als Primär-Datenbank aktiviert, und der Client verweist auf den neuen Server durch Änderung des Servernamens oder der IP-Adresse. Führen Sie die folgenden Schritte für ein manuelles Failover aus:

1. Führen Sie einen INSERT-Befehl durch, um die Replikation zu überprüfen.

```
(postgres@192.168.178.100:5432) [postgres] > INSERT INTO test VALUES
(1000001,'Before Crash');
INSERT 0 1
```

2. Simulieren Sie einen Crash des Primär-Servers mit dem folgenden Kommando:

```
$ pg_ctl stop -m immediate
$ LOG: received immediate shutdown request
```

3. Die Verbindung vom Client zum Server ist abgebrochen.

```
(postgres@192.168.178.100:5432) [postgres] > SELECT * FROM test WHERE id =
1000001;
WARNING:  terminating connection because of crash of another server process
```

4. Aktivieren Sie den bisherigen Standby-Server.

```
$ pg_ctl promote
LOG:  received promote request
LOG:  redo done at 0/5700130C
LOG:  last completed transaction was at log time 2011-11-12 15:28:32.53834+01
LOG:  archive recovery complete
. . .
LOG:  autovacuum launcher started
LOG:  database system is ready to accept connections
```

5. Verbinden Sie sich mit dem Client zum soeben aktivierten Server. Überprüfen Sie, ob die letzte Transaktion gespeichert ist.

```
C:\Users\Lutz>psql -h 192.168.178.212 -U postgres -W
(postgres@192.168.178.212:5432) [postgres] > SELECT * FROM test WHERE id =
1000001;
   id    |     text
---------+--------------
 1000001 | Before Crash
```

Damit ist das Failover abgeschlossen, und der Standby-Server hat die Funktion des Primär-Servers übernommen.

Der Prozess lässt sich vereinfachen, indem man einen sogenannten „Logischen Host“ definiert und dabei eine virtuelle IP-Adresse verwendet, die mit dem Netzwerk-Interface verbunden wird. Diese legt man mit dem Failover auf das Interface des bisherigen Standby-Servers. Damit muss die Konfiguration des Clients nicht geändert werden, es genügt, die Verbindung wiederherzustellen. Die folgenden Schritte liefern ein Beispiel für ein Failover mit einem Logischen Host.

6. Konfigurieren Sie den Primärserver für den Betrieb mit einem logischen Hostnamen (Virtuelle IP-Adresse). Legen Sie die virtuelle IP-Adresse auf das Netzwerk-Interface und starten Sie den PostgreSQL-Server neu, wobei der Parameter listen_addresses auf die virtuelle IP-Adresse verweist. Die Änderungen des Netzwerk-Interfaces müssen unter dem Benutzer „root“ ausgeführt werden.

```
# ifconfig eth0:0 192.168.178.222
# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:24:81:B2:F9:73
          inet addr:192.168.178.100  Bcast:192.168.178.255
. . .
eth0:0    Link encap:Ethernet  HWaddr 00:24:81:B2:F9:73
          inet addr:192.168.178.222  Bcast:192.168.178.255
# su - postgres
$ cd $PGDATA
$ vi postgresql.conf → listen_addresses = '192.168.178.222'
```

7. Führen Sie einen Neustart des Servers durch.

```
$ pg_ctl restart
```

8. Verbinden Sie sich von einem Client unter Verwendung der virtuellen IP-Adresse, und fügen Sie einen Testsatz ein.

```
C:\Users\Lutz>psql -h 192.168.178.222 -U postgres -W
(postgres@192.168.178.222:5432) [postgres] > INSERT INTO test
VALUES(2000000,'Before Crash 2');
INSERT 0 1
```

9. Simulieren Sie den Crash des Datenbank-Servers.

```
$ pg_ctl stop -m immediate
```

10. Entfernen Sie die virtuelle IP-Adresse vom Primär-Server.

```
# ifconfig eth0:0 down
```

11. Legen Sie die virtuelle IP-Adresse auf das Netzwerk-Interface des Standby-Servers.

```
# ifconfig eth0:0 192.168.178.222
```

12. Aktivieren Sie den Standby-Server, und starten Sie diesen mit der virtuellen IP-Adresse.

```
$ pg_ctl promote
$ pg_ctl restart
```

13. Verbinden Sie den Client mit der unveränderten Konfiguration.

```
(postgres@192.168.178.222:5432) [postgres] > \c
You are now connected to database "postgres" as user "postgres".
(postgres@192.168.178.222:5432) [postgres] > SELECT * FROM test WHERE id =
2000000;
```

```
   id    |     text
---------+--------------
 2000000 | Before Crash 2
```

Der Vorteil dieser Methode besteht darin dass die Konfiguration der Clients nicht geändert werden muss. In der Regel liegen die Zeiten für das Failover bei wenigen Sekunden. Dieses Verfahren kann in Skripten hinterlegt werden. Es genügt also ein einfaches „Re-Connect“, um die Verbindung zur Datenbank wiederherzustellen.

Ein Switchover ist ein geplanter Rollentausch zwischen Primär- und Standby-Server. Damit kann man die Verfügbarkeit des Datenbank-Servers im Falle von Wartungsarbeiten am Betriebssystem oder der Hardware erhöhen. Ein einfaches Switchover wird von PostgreSQL noch nicht unterstützt. Dennoch lässt sich die Konfiguration so aufsetzen, dass ein Rollentausch mit wenig Konfigurationsaufwand möglich ist.

Die Idee basiert auch hier auf dem Prinzip des logischen Hosts (virtuelle IP-Adressen), das Sie vom Failover bereits kennen. Allerdings verwendet man hier zwei virtuelle IP-Adressen – eine für den Primär-Server und eine für den Standby-Server. Mit dem Rollentausch werden die virtuellen IP-Adressen sowie die Konfigurationsdateien „postgresql.conf“ zwischen beiden Servern getauscht. Die Vorgehensweise ist folgende:

1. Herunterfahren der Primär-Datenbank (Option „fast“).
2. Austausch der virtuellen IP-Adressen und Konfigurationsdateien.
3. Neustart und Aktivierung des bisherigen Standby-Servers.
4. Neuaufbau des bisherigen Primär-Servers als Standby-Server.