PostgreSQL upgrade project

Zdeněk Kotala
Revenue Product Engineer
Sun Microsystems

Agenda

- Overview
- Catalog upgrade
- Storage upgrade
- Others

Overview

Goals

- Minimal downtime
- No extra disk space
- No old version
- Easy to use

Possible design

- Standalone product
  - Separate binaries which converts database cluster from one version to another.
- PostgreSQL offline upgrade mode
  - Special mode like bootstrap only on already created cluster.
  - Data are binary converted.
- PostgreSQL online data conversion
  - PostgreSQL converts data structure on the fly. Data are converted on background.
  - PostgreSQL will be able to read old structure.

Possible design

Standalone product

- Advantages
  - No or minimal impact on core
- Disadvantages
  - Difficult maintenance – synchronization with core generates a lot of double work
  - Two code could generate inconsistence
  - Database is offline during upgrade – downtime depends on database size
  - Does not fit with PostgreSQL release cycle
  - No responsibility to implement changes from core

Possible design

PostgreSQL offline upgrade mode

- Advantages
  - > Integrated into core – can reuse server code
  - > No extra application
  - > Middle impact on core (mostly new functions)
  - > Faster then data export/import
- Disadvantage
  - > Database is offline during upgrade – downtime depends on database size

Possible design

PostgreSQL online data conversion

- Advantage
  - > Minimal downtime
  - > Downtime doesn't depend on database size
- Disadvantage
  - > How to convert catalog content a structure
  - > Online data conversion has performance impact depends on implementation

...and the winner is

?

Catalog upgrade

List of affected objects

- Control file
- Flat files
- Directory structure
- Catalog tables
- Configuration

Current solutions

- Pg_migrator or pg_upgrade.sh
  - > Only works for 8.1->8.2
  - > Does not support data layout changes (inet/cidr)
  - > Fast (short downtime)
  - > Problem with tablespaces (keep data on one mount point)
  - > Problem with TOAST tables (TOAST pointer)
  - > Depends on private interfaces

How pg_upgrade.sh works*

1) Dump metadata
2) Save relation map (relfilenode<->name)
3) Export control file data
4) Initdb new database cluster
5) Freeze database cluster
6) Copy CLOG
7) Set control data (XID,OID,XLOG ...)
8) Create databases, users ...

*Simplified version without tablespaces

How pg_upgrade.sh works (cont.)

9) Protect TOAST tables (need to have same relfilenode)
10) Create tables, views ...
11) Adjust relfilenode for TOAST tables,idx
12) Copying and renaming data files
13) Done

How upgrade should work

```
pg_ctl -D /var/postgres upgrade
```

How upgrade should work II.

```
check directory /var/postgres ... ok (version 822)
check subdirectories ... ok
creating template1 database in /tmp/pokus/base/1 ... ok
initializing pg_authid ... ok
initializing dependencies ... ok
creating system views ... ok
loading system objects' descriptions ... ok
creating conversions ... ok
creating dictionaries ... ok
setting privileges on built-in objects ... ok
creating information schema ... ok
vacuuming database template1 ... ok
upgrading pg_global database ... ok
upgrading template0 ... ok
upgrading postgres ... ok
upgrading super_db ... ok
```

Control file

- Compatibility verification (BLCKSZ, MAXALIGN, FP format...)
- BLCKSZ, RELSEGSIZE, TOAST MAX CHUNK SIZE could be modified during upgrade
- Translate XID, OID, LC_COLLATE, LSN...

Catalogs

- Structure
  - > Use postgres.bki to initialize catalog
  - > Keep old data files for data transfer
- Contents
  - > User metadata will be transferred and converted to the new structure
  - > Strict rules for systems OID modification
  - > Some kind of changes is not allowed (e.g. binary format change must invoke new data type – new OID)

Configuration files

- postgresql.conf
  - New GUC variable will contain default value
  - Obsolete GUC variable will be ignored – warning in log file
  - Out of range values will be set to default
  - Problem is with different meaning of values
- pg_hba.conf, pg_ident.conf
  - Depends on kind of change ...

Storage upgrade

Page Layout Structures

BLCKSZ

TOAST_MAX_CHUNK_SIZE

PageHeaderData

*MaxItemSize

ItemIdData

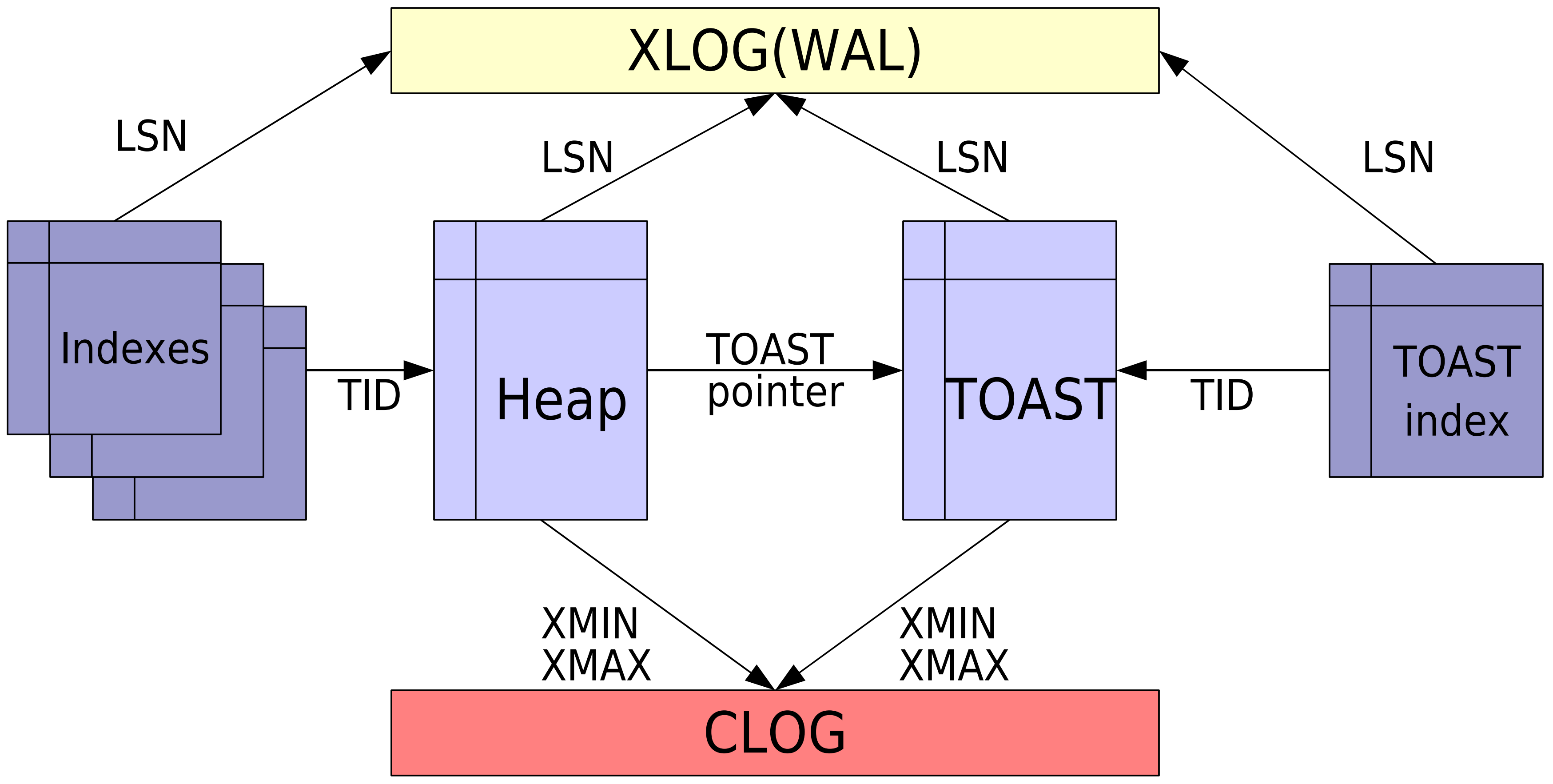
*OpaqueData

IndexTupleData

varatt*

HeapTupleHeaderData

Storage dependency graph



Storage upgrade methods

- On line
  - Read only mode
  - Read Old, Write New
  - On fly page layout conversion
- Off line
  - Inside heap tuple reorganization
  - Heap translation
  - Retoasting
  - Reindexing

Storage upgrade methods

Read Only Mode

- Need to learn PostgreSQL to work with old data structures
- Add extra code which could slow down general performance
- Easy return back to prior version
- Problem with catalog

Storage upgrade methods

Read Old, Write New

- Based on Read Only Mode
- Modified data are written in new format

Storage upgrade methods

Read Only Mode - example

```
#define SizeOfPageHeaderData(page)  \
        (PageGetPageLayoutVersion(page) == 4 ? \
                (offsetof(PageHeaderData_04, pd_linp[0])) :\
                (offsetof(PageHeaderData_03, pd_linp[0])))

typedef struct HeapTupleData
{
        uint32      t_len;              /* length of *t_data */
        ItemPointerData t_self;         /* SelfItemPointer */
        Oid         t_tableOid;         /* table the tuple came from */
        uint16    t_version;            /* page layout version */
        HeapTupleHeader t_data; /* -> tuple header and data */
} HeapTupleData;
```

Storage upgrade methods

Online Page Layout Conversion

- Possible only when converted data fits on same page.
- Not possible between layout version 3 and 4 (8.2->8.3).
  - Pageheader has been extended to 24 bytes.
  - Index tuples does not fit on a page, different toast chunk size and heap tuples does not fit on machines with MAXALIGN=8 (e.g. SPARC)
- WAL generates a lot of full page writes.

Storage upgrade methods

Online Page Layout Conversion - example

- Convertor hook in ReadBuffer_common

```
{
        smgrread(reln->rd_smgr, blockNum, (char *) bufBlock);
        /* Page Layout Convertor hook. We assume
           that page version is on same place. */
        if( plc_hook &&  PageGetPageLayoutVersion(reln,bufBlock)
                != PG_PAGE_LAYOUT_VERSION )
        {
                plc_hook((char *)bufBlock);
                bufHdr->flags |= (BM_DIRTY | BM_JUST_DIRTIED);
                log_newpage(&reln->rd_node, blockNum ,bufBlock);
        }
}
```

Storage upgrade methods

Inner heap tuples reorganization

- Similar to page layout conversion, but tuple which does not fit on the page have to be moved to a new page
- Requires reindex (only if inter page transfer happened) or introduce inter page redirection pointer
- Requires WAL logging
- Does not need extra disk space

Storage upgrade methods

Heap translation

- Tuples are translated from old heap to the newly created
- Possible to change BLCKSZ, RELSEGSIZE
- Does not require WAL logging
- Needs space for a new table (old indexes could be dropped or continuously drop segments)
- If TOAST table is translated (has new relfilenode), TOAST pointers must be updated

Storage upgrade methods

Retoasting

- Needed when TOAST_MAX_CHUNK_SIZE has been changed
- More possible solutions:
  - Add TOAST_MAX_CHUNK_SIZE to pg_class
  - Adjust toast_fetch_datum() accept different size
  - Combine retoasting with heap translation
- Take care about TOAST pointer
- Requires full index scan on original TOAST IDX related to the TOAST table

Storage upgrade methods

Reindexing

- Reindexing is necessary every time when
  - tid of any tuple has been changed
  - index structure has been changed
  - index tuples does not fit on a new page layout
- Reindex could be performed on the running system

Write Ahead Log (WAL/XLOG)

- CHECKPOINT is last operation on shutdown. All changes are applied and WAL files can be dropped.
- Needs to keep XLOG pointer to protect correct recovery (LSN dependency on WAL)

Commit log (CLOG)

- Array of transactions status
- No changes for long time – stable
- Some upgrade methods could produce a frozen database, afterwards CLOG files could be removed

Other

Stored procedures

- Changes in PL languages
  - All changes are usually backward compatible
  - Possible to add language version into catalog and delivery more *.so
  - Problem with procedures written in C

Tsearch2

- Any change in FTS configuration or dictionary implies regeneration of affected tsvectors fields. Unfortunately, there is not relation between tsvector and original source.

Proposed upgrade devel policy

- Each submitted patch MUST handle upgrade
- All affected structures should have version number
- Binaries should work with multiple versions of database clusters (e.g. pg_controldata)
- System OIDs in catalog shouldn't be reused

References

http://pgfoundry.org/projects/pg-migrator/

http://src.opensolaris.org/source/xref/sfw/usr/src/cmd/postgres/postgresql-upgrade/

http://wiki.postgresql.org/wiki/In-place_upgrade

PostgreSQL upgrade project

Zdeněk Kotala
zdenek.kotala@sun.com