PostgreSQL 9.5 WAL format

Heikki Linnakangas / VMware

Jan 31st, 2015

WAL-logging basics

- The log is a sequence of log records
- One log record for every change
- Write Ahead Log
- Each WAL record is assigned an LSN (Log Sequence Number)

PostgreSQL's WAL log

- REDO only, no UNDO actions.
- Instantaneous rollbacks
- No limit on transaction size
- Physical log

Example: Insert a row to table with one index

```
rmgr: Heap        len (rec/tot):      3/    59,
  tx:       1133, lsn: 0/6909A748, prev 0/6909A718,
  desc: INSERT off 3,
  blkref #0: rel 1663/12726/50058 blk 0
rmgr: Btree       len (rec/tot):      2/    64,
  tx:       1133, lsn: 0/6909A788, prev 0/6909A748,
  desc: INSERT_LEAF off 1,
  blkref #0: rel 1663/12726/50064 blk 1
rmgr: Transaction len (rec/tot):     12/    38,
  tx:       1133, lsn: 0/6909A7C8, prev 0/6909A788,
  desc: COMMIT_COMPACT 2015-01-31 07:59:23.344845 CET
```

Format overview

- WAL records are written in WAL pages.
- Each page has a page header
- pages are stored in 16 MB segments (= files). Segment has a header too.

No changes here (since 9.3).

Full-page writes

- First time a page is modified after a checkpoint, a copy of the whole page is put to the log
- Subsequent changes to the same page only log the changes.

Old format (PostgreSQL 9.4 and below)

```
/*
 * The overall layout of an XLOG record is:
 *      Fixed-size header (XLogRecord struct)
 *      rmgr-specific data
 *      BkpBlock
 *      backup block data
 *      BkpBlock
 *      backup block data
 *      ...
 *
 * where there can be zero to four backup blocks (as signal
 * bits).  XLogRecord structs always start on MAXALIGN boun
 * files, and we round up SizeOfXLogRecord so that the rmgr
 * guaranteed to begin on a MAXALIGN boundary.  However, no
 * to align BkpBlock structs or backup block data.
 *
 * NOTE: xl_len counts only the rmgr data, not the XLogReco
 * and also not any backup blocks.  xl_tot_len counts every
```

Old format (PostgreSQL 9.4 and below)

```
typedef struct XLogRecord
{
    uint32      xl_tot_len;     /* total len of entire reco
    TransactionId xl_xid;       /* xact id */
    uint32      xl_len;         /* total len of rmgr data *
    uint8       xl_info;        /* flag bits, see below */
    RmgrId      xl_rmid;        /* resource manager for thi
    /* 2 bytes of padding here, initialize to zero */
    XLogRecPtr  xl_prev;        /* ptr to previous record i
    pg_crc32    xl_crc;         /* CRC for this record */

    /* If MAXALIGN==8, there are 4 wasted bytes here */

    /* ACTUAL LOG DATA FOLLOWS AT END OF STRUCT */

} XLogRecord;
```

32 bytes in total (28 on 32-bit systems)

Old format problems

pg_rewind

- A tool to resynchronize PostgreSQL clusters e.g. after failover
- rsync on steroids

Other tools

- Read-ahead of pages at WAL replay
  - *pg_readahead*, by Koichi Suzuki.
- Differential or incremental backups.

Old format Problems

The format left a lot as resource manager's responsibility

- No common format for recording which block the record applies to. (Except for full-page images).
- Bulky

Code issues

- lots of boilerplate code in WAL generation / replay
- Complex record types needed careful bookkeeping of which parts of the data were included, and which was left out due to full-page writes.

New format (PostgreSQL 9.5)

```
/*
 * The overall layout of an XLOG record is:
 *      Fixed-size header (XLogRecord struct)
 *      XLogRecordBlockHeader struct
 *      XLogRecordBlockHeader struct
 *      ...
 *      XLogRecordDataHeader[Short|Long] struct
 *      block data
 *      block data
 *      ...
 *      main data
 *
 * There can be zero or more XLogRecordBlockHeaders, and 0
 * rmgr-specific data not associated with a block.  XLogRec
 * always start on MAXALIGN boundaries in the WAL files, bu
 * the fields are not aligned.
 *
 * The XLogRecordBlockHeader, XLogRecordDataHeaderShort an
```

New format (PostgreSQL 9.5)

```
typedef struct XLogRecord
{
    uint32      xl_tot_len;     /* total len of entire reco
    TransactionId xl_xid;       /* xact id */
    XLogRecPtr  xl_prev;        /* ptr to previous record
    uint8       xl_info;        /* flag bits, see below */
    RmgrId      xl_rmid;        /* resource manager for thi
    /* 2 bytes of padding here, initialize to zero */
    pg_crc32    xl_crc;         /* CRC for this record */

    /* XLogRecordBlockHeaders and XLogRecordDataHeader foll

} XLogRecord;
```

24 bytes in total

New format (PostgreSQL 9.5)

```
/*
 * Header info for block data appended to an XLOG record.
 ...
 */
typedef struct XLogRecordBlockHeader
{
    uint8       id;             /* block reference ID */
    uint8       fork_flags;     /* fork within the relatio
    uint16      data_length;    /* number of payload bytes
                                 * image) */

    /* If BKPBLOCK_HAS_IMAGE, an XLogRecordBlockImageHeade
    /* If !BKPBLOCK_SAME_REL is not set, a RelFileNode fol
    /* BlockNumber follows */
} XLogRecordBlockHeader;
```

New format (PostgreSQL 9.5)

Per block flags:

```
#define BKPBLOCK_HAS_IMAGE  0x10    /* block data is an XLo
#define BKPBLOCK_HAS_DATA   0x20
#define BKPBLOCK_WILL_INIT  0x40    /* redo will re-init th
#define BKPBLOCK_SAME_REL   0x80    /* RelFileNode omitted,
```

Code changes

New format required changes to

- every function that generates a WAL record,
- and every REDO routine.

```
src/backend/access/brin/brin.c | 11 +-
src/backend/access/brin/brin_pageops.c | 97 +–
src/backend/access/brin/brin_revmap.c | 23 +-
src/backend/access/brin/brin_xlog.c | 111 ++-
src/backend/access/gin/ginbtree.c | 111 +–
src/backend/access/gin/gindatapage.c | 162 ++–
src/backend/access/gin/ginentrypage.c | 64 +-
src/backend/access/gin/ginfast.c | 92 +–
93 files changed, 3945 insertions(+), 4366 deletions(-)
```

Code changes / writing a WAL record

Before:

```
xl_heap_lock xlrec;
XLogRecData rdata[2];

xlrec.target.node = relation->rd_node;
xlrec.target.tid = tuple->t_self;
xlrec.locking_xid = xid;
xlrec.infobits_set = compute_infobits(new_infomask,
                                      tuple->t_data->t_info
rdata[0].data = (char *) &xlrec;
rdata[0].len = SizeOfHeapLock;
rdata[0].buffer = InvalidBuffer;
rdata[0].next = &(rdata[1]);

rdata[1].data = NULL;
rdata[1].len = 0;
rdata[1].buffer = *buffer;
```

Code Changes / Writing a WAL record

After:

```
xl_heap_lock xlrec;

XLogBeginInsert();
XLogRegisterBuffer(0, *buffer, REGBUF_STANDARD);

xlrec.offnum = ItemPointerGetOffsetNumber(&tuple->t_self);
xlrec.locking_xid = xid;
xlrec.infobits_set = compute_infobits(new_infomask,
                                      tuple->t_data->t_info
XLogRegisterData((char *) &xlrec, SizeOfHeapLock);

recptr = XLogInsert(RM_HEAP_ID, XLOG_HEAP_LOCK);

PageSetLSN(page, recptr);
```

Code Changes / Writing a WAL record

```
/* flags for XLogRegisterBuffer */
#define REGBUF_FORCE_IMAGE  0x01    /* force a full-page in
#define REGBUF_NO_IMAGE     0x02    /* don't take a full-pa
#define REGBUF_WILL_INIT    (0x04 | 0x02)   /* page will be
                                     * replay (implies NO_I
#define REGBUF_STANDARD     0x08    /* page follows "standa
                                     * (data between pd_low
                                     * will be skipped) */
#define REGBUF_KEEP_DATA    0x10    /* include data even if
                                     * is taken */
```

Code changes / redo routine

```
/* If we have a full-page image, restore it and we’re done
if (record->xl_info & XLR_BKP_BLOCK(0))
{
    (void) RestoreBackupBlock(lsn, record, 0, false, false)
    return;
}

buffer = XLogReadBuffer(xlrec->target.node,
                        ItemPointerGetBlockNumber(&(xlrec->
                        false);
if (!BufferIsValid(buffer))
    return;
page = (Page) BufferGetPage(buffer);

if (lsn <= PageGetLSN(page))    /* changes are applied */
{
    UnlockReleaseBuffer(buffer);
    return;
```

Code changes / redo routine / after

```
if (XLogReadBufferForRedo(record, 0, &buffer) ==
            BLK_NEEDS_REDO)
{
    ... apply the changes from the record ...
}
if (BufferIsValid(buffer))
    UnlockReleaseBuffer(buffer);
```

Code changes / xlogreader.c

xlogreader is an API for reading WAL records

- Used by WAL replay functions
- Can be used by external tools
  - pg_xlogdump
  - pg_rewind

XLogRecGetData XLogRecGetDataLen

XLogRecGetBlockData XLogRecGetBlockTag

Testing

Lots of changes -> Lots of bugs

- Need for automated testing
- block comparison tool

Block comparison tool

- Every time a page is locked, stash an image of the block as it was
- Every time a page lock is released, compare the image with the before-image
- If it differs, dump it to a file along with the LSN

Testing with the block comparison tool

- Set up a master-standby system
- run “make installcheck”
  - produces about 11 GB of dumped pages
  - in both master and standby
- run a little tool to compare the dumped pages between master and standby
  - masks out hint bits etc.

Found existing bugs

Found three existing bugs in obscure corner cases:

- bit in visibility map might not be set correctly (9.3-)
- concurrent scan of GiST index might miss records in hot standby (9.0-)
- Insertion to GIN internal pages didn't take a full-page image (9.0-)

Comparison

- How does the new WAL format perform?

Comparison: WAL size

WAL size of various UPDATE commands.

testname	9.4	9.5	difference
two short fields, no change	367	329	-10 %
two short fields, one changed	405	331	-18 %
two short fields, both changed	405	370	-9 %
one short and one long field, no change	73	54	-26 %
ten tiny fields, all changed	445	369	-17 %
hundred tiny fields, all changed	162	156	-4 %
hundred tiny fields, half changed	174	162	-7 %
hundred tiny fields, half nulled	93	77	-17 %
9 short and 1 long, short changed	91	89	-3 %

BTW

- Full Page Compression patch by Fujii Masao, Michael Paquier, et al

BTW 2

The checksum algorithm changed in 9.5. It's now CRC-C.

- Allows hardware computation on some platforms, like modern Intel (patch pending)
- Slicing-by-8 on other platforms (patch pending)

The end

- WAL generation and replay code is cleaner now.
- You can now write tools that read WAL and make some sense of it.
    - See contrib/pg_xlogdump for an example.