PostgreSQL中文社区

2021 PostgreSQL China Conference

主办：PostgreSQL 中文社区

# 第 11 届 PostgreSQL 中国技术大会

开源论道 × 数据驱动 × 共建数字化未来

# 《Checkpoint原理浅析》

杨向博　　DBA

- Checkpoint 定义
- Checkpoint 触发条件
- Checkpoint 会做什么
- Checkpoint skipped机制
- Checkpoint 过程记录

- Checkpoint 定义

*Checkpoints* are points in the sequence of transactions at which it is guaranteed that the heap and index data files have been updated with all information written before that checkpoint.

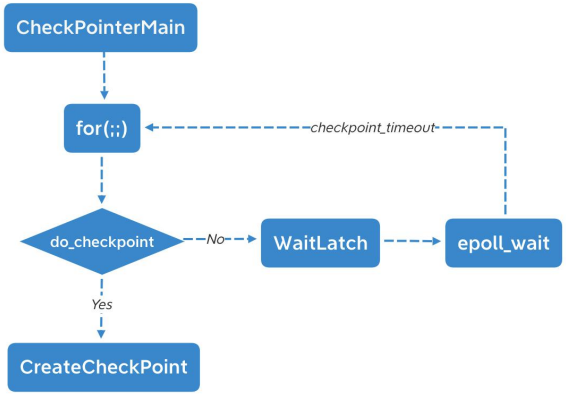
At checkpoint time, all dirty data pages are flushed to disk and a special checkpoint record is written to the log file. (The change records were previously flushed to the WAL files.)

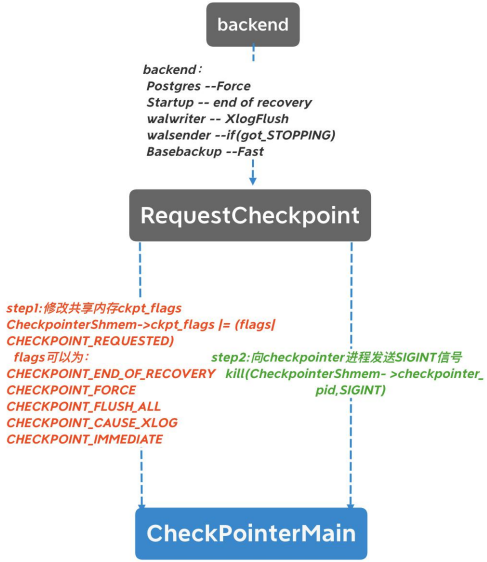
In the event of a crash, the crash recovery procedure looks at the latest checkpoint record to determine the point in the log (known as the redo record) from which it should start the REDO operation.

Any changes made to data files before that point are guaranteed to be already on disk.
Hence, after a checkpoint, log segments preceding the one containing the redo record are no longer needed and can be recycled or removed. (When WAL archiving is being done, the log segments must be archived before being recycled or removed.)

简单来说，checkpoint就是一个事务顺序的记录点。checkpoint主要是进行刷脏页，redo时会参考checkpoint进行日志回放。除了刷脏之外还会更新一些位点信息，清理一些不再需要的wal。

- Checkpoint 触发条件




在PostgreSQL中Checkpoint是由checkpointer进程执行的。

Checkpointer进程的主流程是一个无条件的for循环，在未触发checkpoint时一直在WaitLatch中sleep，也就是在epoll_wait中观察list链表，查看是否有事件句柄已经就绪（某个条件在触发checkpoint）；

如果已经存在就绪事件，则wake up（通过SetLatch中write pipe的方式wake up），执行checkpoint。

- # Checkpoint 触发条件

触发checkpoint的Flags:

```
/* These directly affect the behavior of CreateCheckPoint and subsidiaries */
#define CHECKPOINT_IS_SHUTDOWN          0x0001  /* Checkpoint is for shutdown */
#define CHECKPOINT_END_OF_RECOVERY.     0x0002  /* Like shutdown checkpoint, but
                                                 * issued at end of WAL recovery */
#define CHECKPOINT_IMMEDIATE            0x0004  /* Do it without delays */
#define CHECKPOINT_FORCE                0x0008  /* Force even if no activity */
#define CHECKPOINT_FLUSH_ALL            0x0010  /* Flush all pages, including those
                                                 * belonging to unlogged tables */

/* These are important to RequestCheckpoint */
#define CHECKPOINT_WAIT                 0x0020  /* Wait for completion */
#define CHECKPOINT_REQUESTED            0x0040  /* Checkpoint request has been made */
/* These indicate the cause of a checkpoint request */
#define CHECKPOINT_CAUSE_XLOG           0x0080  /* XLOG consumption */
#define CHECKPOINT_CAUSE_TIME           0x0100  /* Elapsed time */
```

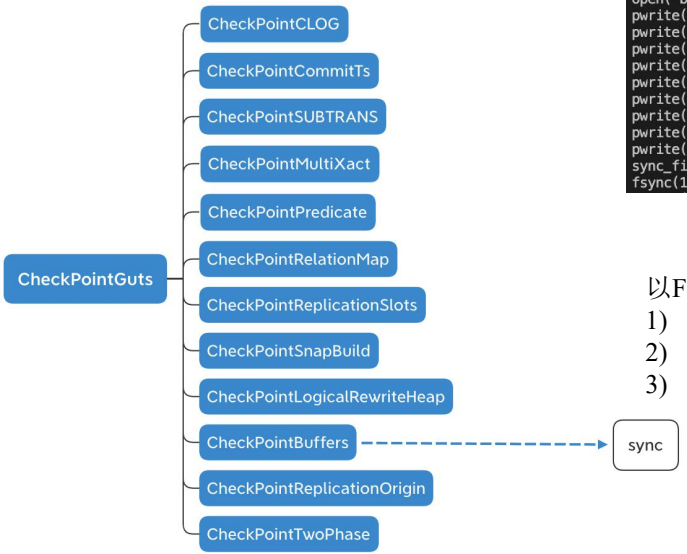
以手动触发checkpoint为例：

```
case T_CheckPointStmt:
    if (!superuser())
        ereport(ERROR,
                (errcode(ERRCODE_INSUFFICIENT_PRIVILEGE),
                 errmsg("must be superuser to do CHECKPOINT")));

    RequestCheckpoint(CHECKPOINT_IMMEDIATE | CHECKPOINT_WAIT | (RecoveryInProgress() ? 0 : CHECKPOINT_FORCE));
    break;
```

- Checkpoint 会做什么

## 1、Flush dirty page




```
open("base/13578/16428.6", O_RDWR) = 15 <0.000016>
pwrite(15, "\20\0\0\0h-\10\4\0\0\0\0\370\1\0\2\0 \4 \0\0\0\0\300\237z\0\200\237z\0"..., 8192, 384622592) = 8192 <0.000024>
pwrite(15, "\20\0\0\0\270Z\10\4\0\0\0\0\370\1\0\2\0 \4 \0\0\0\0\300\237z\0\200\237z\0"..., 8192, 384630784) = 8192 <0.000020>
pwrite(15, "\20\0\0\0\350\207\10\4\0\0\0\0\370\1\0\2\0 \4 \0\0\0\0\300\237z\0\200\237z\0"..., 8192, 384638976) = 8192 <0.000019>
pwrite(15, "\20\0\0\0\0\265\10\4\0\0\0\0\370\1\0\2\0 \4 \0\0\0\0\300\237z\0\200\237z\0"..., 8192, 384647168) = 8192 <0.000018>
pwrite(15, "\20\0\0\0\0000\342\10\4\0\0\0\0\370\1\0\2\0 \4 \0\0\0\0\300\237z\0\200\237z\0"..., 8192, 384655360) = 8192 <0.000018>
pwrite(15, "\20\0\0\0H\17\t\4\0\0\0\0\370\1\0\2\0 \4 \0\0\0\0\300\237z\0\200\237z\0"..., 8192, 384663552) = 8192 <0.000018>
pwrite(15, "\20\0\0\0`<\t\4\0\0\0\0\370\1\0\2\0 \4 \0\0\0\0\300\237z\0\200\237z\0"..., 8192, 384671744) = 8192 <0.000019>
pwrite(15, "\20\0\0\0220i\t\4\0\0\0\0\370\1\0\2\0 \4 \0\0\0\0\300\237z\0\200\237z\0"..., 8192, 384679936) = 8192 <0.000023>
pwrite(15, "\20\0\0\0\250\207\t\4\0\0\0\0X\1\0\f\0 \4 \0\0\0\0\300\237z\0\200\237z\0"..., 8192, 384688128) = 8192 <0.000019>
sync_file_range(0xf, 0x16ece000, 0x12000, 0x2) = 0 <0.000045>
fsync(15)               = 0 <0.002481>
```

以Flush Datafile为例：

1) open 打开文件；
2) pwrite写入修改内容，DB BLCLKSZ为8k，OS为4k，非原子写，FPW特性兜底；
3) fsync将缓冲块立即落盘；

- Checkpoint 会做什么

  2、Update some points

```
/*
 * Update the control file.
 */
LWLockAcquire(ControlFileLock, LW_EXCLUSIVE);
if (shutdown)
    ControlFile->state = DB_SHUTDOWNED;
ControlFile->checkPoint = ProcLastRecPtr;
ControlFile->checkPointCopy = checkPoint;
ControlFile->time = (pg_time_t) time(NULL);
/* crash recovery should always recover to the end of WAL */
ControlFile->minRecoveryPoint = InvalidXLogRecPtr;
ControlFile->minRecoveryPointTLI = 0;
/*
 * Persist unloggedLSN value. It's reset on crash recovery, so this goes
 * unused on non-shutdown checkpoints, but seems useful to store it always for debugging purposes.
 */
SpinLockAcquire(&XLogCtl->ulsn_lck);
ControlFile->unloggedLSN = XLogCtl->unloggedLSN;
SpinLockRelease(&XLogCtl->ulsn_lck);

UpdateControlFile();
LWLockRelease(ControlFileLock);

/* Update shared-memory copy of checkpoint XID/epoch */
SpinLockAcquire(&XLogCtl->info_lck);
XLogCtl->ckptFullXid = checkPoint.nextFullXid;
SpinLockRelease(&XLogCtl->info_lck);
```

- Checkpoint 会做什么

  3、Remove old wal

```
/*
* Update the average distance between checkpoints if the prior checkpoint
* exists.
*/
if (PriorRedoPtr != InvalidXLogRecPtr)
/*根据ptr偏移量，预估出两次checkpoint间产生的wal量
CheckPointDistanceEstimate*/
    UpdateCheckPointDistanceEstimate(RedoRecPtr - PriorRedoPtr);
/*
* Delete old log files, those no longer needed for last checkpoint to
* prevent the disk holding the xlog from growing full.
*/
XLByteToSeg(RedoRecPtr, _logSegNo, wal_segment_size);
/*根据min{wal_keep_segments, min(replication_slot.restart_lsn)}计算出_logSegNo，
比_logSegNo早的日志后续将会被清理掉*/
KeepLogSeg(recptr, &_logSegNo);
_logSegNo--;
/*首先根据CheckPointDistanceEstimate 结合一套公式，计算出开始回收重用的
recycleSegNo，从这个日志开始回收重用
（wal_recycle默认开启，主要是保留日志并rename为新的序列号，回收一个序列
号加一）*/
/*然后将_logSegNo之前并已经归档（如果开启归档）的wal都清理掉*/
RemoveOldXlogFiles(_logSegNo, RedoRecPtr, recptr);
```

- Checkpoint skipped机制

```
/*
 * If this isn't a shutdown or forced checkpoint, and if there has been no
 * WAL activity requiring a checkpoint, skip it.  The idea here is to
 * avoid inserting duplicate checkpoints when the system is idle.
 */
if ((flags & (CHECKPOINT_IS_SHUTDOWN | CHECKPOINT_END_OF_RECOVERY |
              CHECKPOINT_FORCE)) == 0)
{
    if (last_important_lsn == ControlFile->checkPoint)
    {
        WALInsertLockRelease();
        LWLockRelease(CheckpointLock);
        END_CRIT_SECTION();
        ereport(DEBUG1,
                (errmsg("checkpoint skipped because system is idle")));
        return;
    }
}
```

当非停库、redo完成、强制触发checkpoint时，如果数据库没有写入操作，则直接return不进行Flush dirty page等操作

- Checkpoint skipped机制

wal_keep_segments配置256，不存在复制槽，归档无报错的情况下保留17000+个wal

```
postgres=# select * from pg_replication_slots ;
 slot_name | plugin | slot_type | datoid | database | temporary | active | active_pid | xmin | catalog_xmin | restart_lsn | confirmed_flush_lsn
-----------+--------+-----------+--------+----------+-----------+--------+------------+------+--------------+-------------+---------------------
(0 rows)

postgres=# show wal_keep_segments ;
 wal_keep_segments
-------------------
 256
(1 row)

postgres=# \q
[redacted]$ cd pg_wal
[redacted]pg_wal]$ ll | wc -l
17631
```

打开debug日志后，checkpoint_timeout触发checkpoint后发现是进入了checkpoint skipped机制

```
[redacted]$ grep checkpoint postgresql-29-*.csv
[redacted]29 13:01:16.676 CST,,,38142,,6082d451.94fe,2644,,2021-04-23 22:06:09 CST,,0,DEBUG,00000,"checkpoint skipped because system is idle",,,,,,,,,""
```

手动强制触发checkpoint后，wal日志清理成功

```
Type "help" for help.

postgres=# checkpoint;
CHECKPOINT
postgres=# \q
[redacted]$ cd pg_wal
[redacted]/pg_wal]$ ll | wc -l
322
```

- Checkpoint 过程记录

1) 当log_checkpoints=on时，pglog会记录checkpoint过程；

```
2022-01-02 16:57:34.249 CST [19090] LOG:  checkpoint starting: immediate force wait
2022-01-02 16:57:34.269 CST [19090] LOG:  checkpoint complete: wrote 9 buffers (0.0%); 0 WAL file(s) added, 0 removed, 0 recycled; write=0.003 s, sync=0.002 s, tot
al=0.020 s; sync files=1, longest=0.002 s, average=0.002 s; distance=97 kB, estimate=99 kB
```

2) 使用命令行工具pg_controldata解析pg_control文件；

```
~]$pg_controldata -D $PGDATA | grep checkpoint
Latest checkpoint location:           10/4098810
Latest checkpoint's REDO location:    10/40987D8
Latest checkpoint's REDO WAL file:    000000010000001000000004
Latest checkpoint's TimeLineID:       1
Latest checkpoint's PrevTimeLineID:   1
Latest checkpoint's full_page_writes: on
Latest checkpoint's NextXID:          0:611
Latest checkpoint's NextOID:          32858
Latest checkpoint's NextMultiXactId:  1
Latest checkpoint's NextMultiOffset:  0
Latest checkpoint's oldestXID:        478
Latest checkpoint's oldestXID's DB:   1
Latest checkpoint's oldestActiveXID:  611
Latest checkpoint's oldestMultiXid:   1
Latest checkpoint's oldestMulti's DB: 1
Latest checkpoint's oldestCommitTsXid:0
Latest checkpoint's newestCommitTsXid:0
Time of latest checkpoint:            Sun Jan  2 16:57:34 2022
~]$
```

3) 登陆数据库查询系统函数pg_control_checkpoint()；

```
postgres=# select * from pg_control_checkpoint();
-[ RECORD 1 ]---------+--------------------------
checkpoint_lsn        | 10/4098810
redo_lsn              | 10/40987D8
redo_wal_file         | 000000010000001000000004
timeline_id           | 1
prev_timeline_id      | 1
full_page_writes      | t
next_xid              | 0:611
next_oid              | 32858
next_multixact_id     | 1
next_multi_offset     | 0
oldest_xid            | 478
oldest_xid_dbid       | 1
oldest_active_xid     | 611
oldest_multi_xid      | 1
oldest_multi_dbid     | 1
oldest_commit_ts_xid  | 0
newest_commit_ts_xid  | 0
checkpoint_time       | 2022-01-02 16:57:34+08
```

# THANKS

谢谢观看