2018年PostgreSQL中国技术大会


# 庖丁解牛-平安vacuum优化之路

/>:平安云 石勇虎

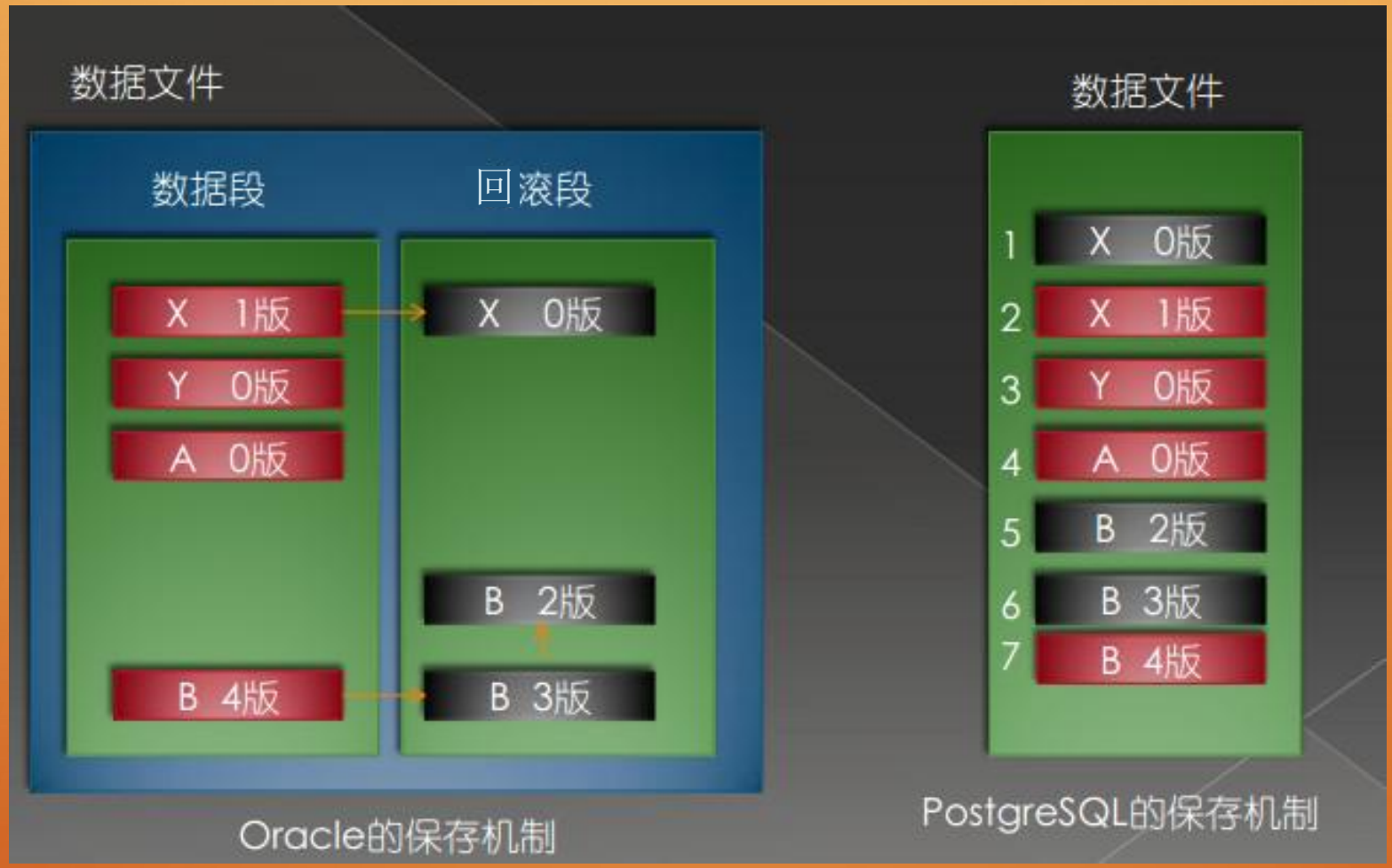
shiyonghu651@pingan.com.cn

平安科技（深圳）有限公司

# 目录

# 庖丁解牛-初识vacuum

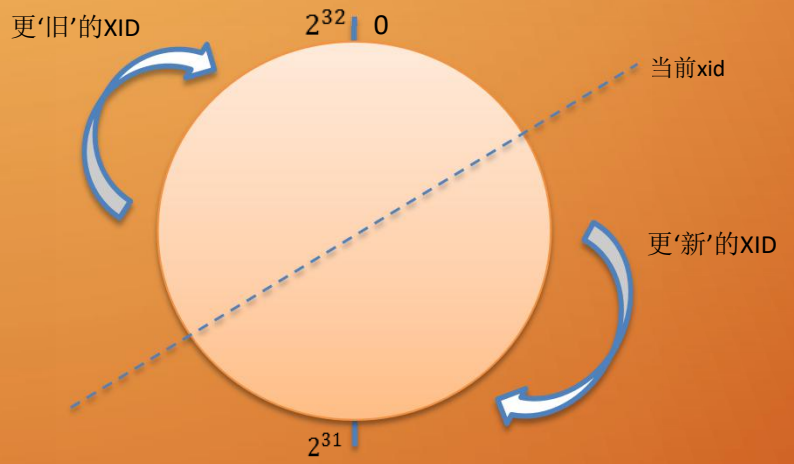
# MVCC


数据文件
数据段
回滚段
X 1版
Y 0版
A 0版
B 4版
X 0版
B 2版
B 3版
Oracle的保存机制
数据文件
1 X 0版
2 X 1版
3 Y 0版
4 A 0版
5 B 2版
6 B 3版
7 B 4版
PostgreSQL的保存机制

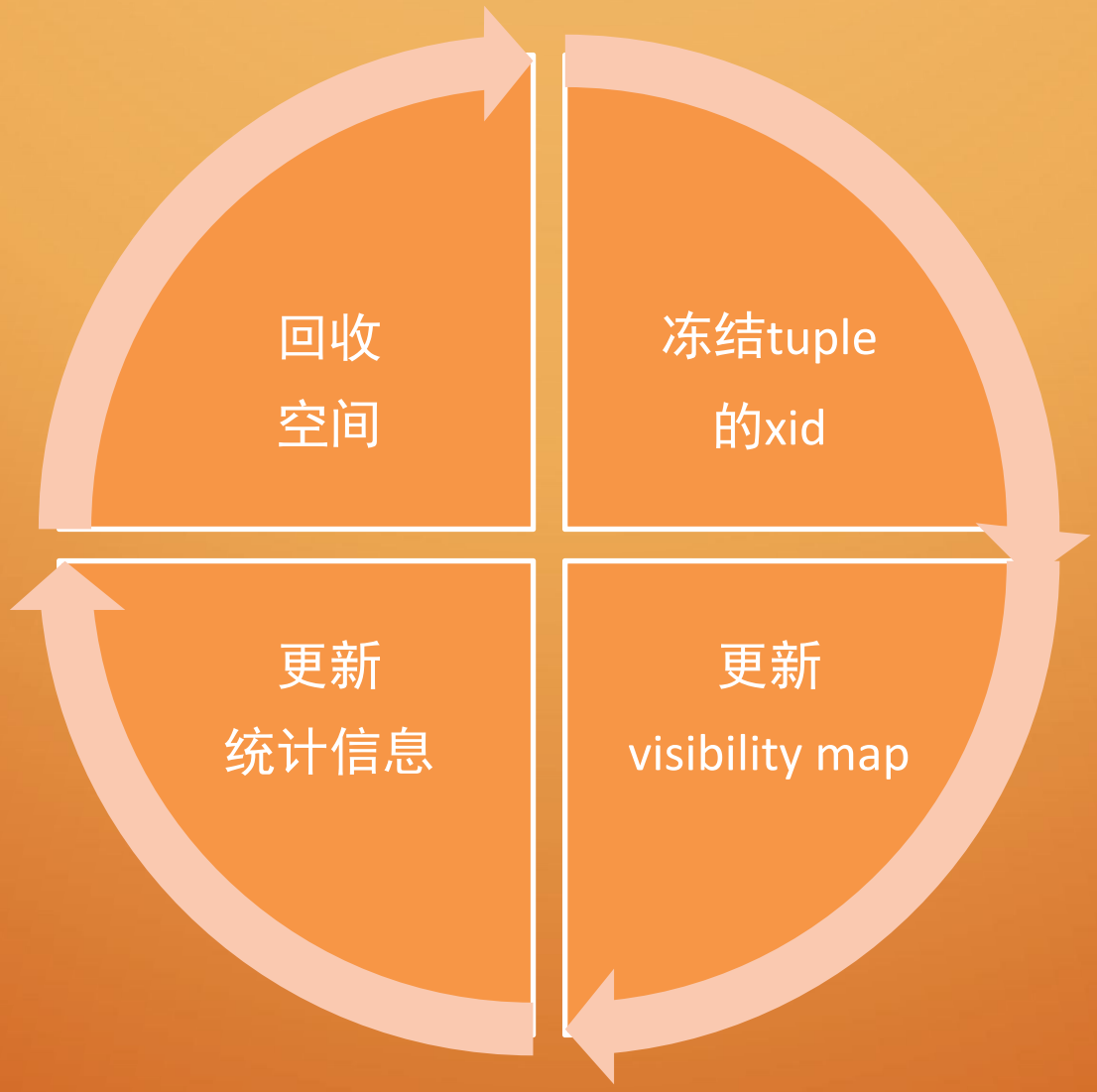
# 事物ID

- xid是32位的整数，在3到最大值之间循环使用，0到2是特殊值
- 每产生一个事务ID，pg_clog下的commit log都会占用2bit（事务状态）
- 没有做任何实际变更的事务，只会产生vxid（虚拟事务ID）

- InvalidTransactionId = 0
- BootstrapTransactionId = 1
- FrozenTransactionId = 2



# Vacuum作用

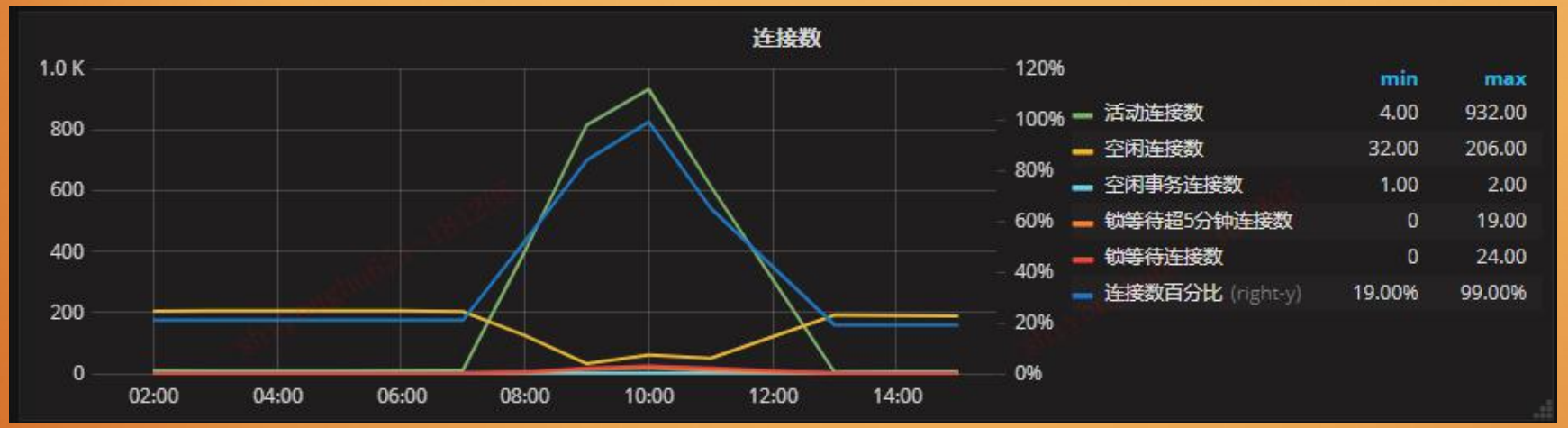


# 常见问题

- Blocking DDL
- Autovacuum runs,performance sucks.
- Autovacuum running for days.
- Autovacuum is constantly running on table A,B,C.
- Autovacuum finished， but not work. --long transaction.

# 庖丁解牛-解决vacuum

# 案例分享




从pgawr中可以看到阻塞的SQL性能是跟这个表的dead row versions成正比关系的，目前看性能从凌晨1点开始就下降明显了。从单次耗时3ms增长到目前的90ms。

| snap_time | executes | queryid | toal_time | elapse_time | physical_reads | total_gets | buffer_gets |
|---|---|---|---|---|---|---|---|
| 2018-04-04 09:15:03 | 1202 | 4098489151 | 108558.295000028 | 90.3147212978601 | 97 | 14320474 | 11913 |
| 2018-04-04 09:00:02 | 1533 | 4098489151 | 141425.898999978 | 92.2543372472132 | 142 | 17728015 | 11564 |
| 2018-04-04 08:45:02 | 1090 | 4098489151 | 96497.4660000131 | 88.5297853211129 | 95 | 12243614 | 11232 |
| 2018-04-04 08:30:02 | 873 | 4098489151 | 72480.7689999826 | 83.0249358533593 | 69 | 9473094 | 10851 |
| 2018-04-04 08:15:02 | 534 | 4098489151 | 42066.5430000052 | 78.7762977528187 | 71 | 5637067 | 10556 |
| 2018-04-04 08:00:03 | 476 | 4098489151 | 35730.4419999905 | 75.0639537814926 | 39 | 4820976 | 10128 |
| 2018-04-04 07:00:02 | 324 | 4098489151 | 20217.7309999913 | 62.4004043209608 | 37 | 2806237 | 8661 |
| ..... | | | | | | | |
| 2018-04-04 01:45:02 | 6 | 4098489151 | 88.4390000030398 | 14.73983333384 | 1 | 11528 | 1921 |
| 2018-04-04 01:30:02 | 30 | 4098489151 | 366.309000000358 | 12.2103000000119 | 4 | 43253 | 1441 |
| 2018-04-04 01:15:03 | 24 | 4098489151 | 225.026000000536 | 9.37608333335569 | 2 | 23597 | 983 |
| 2018-04-04 01:00:02 | 36 | 4098489151 | 271.227000001818 | 7.53408333338383 | 3 | 22796 | 633 |
| 2018-04-04 00:45:02 | 45 | 4098489151 | 215.191999997944 | 4.78204444439875 | 3 | 15834 | 351 |
| 2018-04-04 00:30:03 | 58 | 4098489151 | 217.315999995917 | 3.7468275861365 | 3 | 17420 | 300 |

# 案例分享

- 异常分析

1）PG数据库的表在正常情况下会自动回收空间（dead row versions）。如果表被长事务（idle in transaction）访问的情况下，就不能自动回收表的dead row versions，造成表空间持续增长。

2）该数据库存在一个JOB（collect_customer_job），PG库一个function只能包含一个事务，在job执行完之前，该事务无法提交，导致PG库的AUTOVACUUM进程无法正常回收表elifeassist_product的空间（dead row versions），造成该表空间持续增大，进而引发访问该表的SQL效率持续下降，因为效率降低，PG库的连接数被耗光。

- 改进项

1）长事务（long transaction）的监控。

2）对频繁更新的表设置定期vacuum（如每小时一次），作为autovacuum的补充。同时在开发规范中对频繁更新表增加风险提示。

3）JOB（collect_customer_job）的分拆、改造工作，减少job运行时长。

4）继续跟进分析session kill 后进程没正常结束的问题。

# 案例分享

结合问题重现时的处理过程，现场抓取的 pstack， strace，perf等信息，以及对PG源码的分析，结论如下：
从4月4日的处理过程，我们可以确认使用pg_terminate_backend函数结束进程是有效的，只是存在大量LWLock（类似oracle latch）争用的情况下，进程结束的过程比较慢。

#0 0x0000003ae0eeb0d7 in semop () from /lib64/libc.so.6
#1 0x0000000000646fff in PGSemaphoreLock ()
#2 0x00000000006a2dd3 in LWLockAcquire ()
#3 0x00000000004c0d6d in SimpleLruWaitIO ()

```
/*
 * LWLockAcquire - acquire a lightweight lock in the specified mode
 *
 * If the lock is not available, sleep until it is.  Returns true if the lock
 * was available immediately, false if we had to sleep.
 *
 * Side effect: cancel/die interrupts are held off until lock release.
 */
bool
LWLockAcquire(LWLock *l, LWLockMode mode)
{
    return LWLockAcquireCommon(l, mode, NULL, 0);
}
```

PG和Oracle都是多进程架构，在进程或者会话被kill后，为了保护共享内存结构，保证数据库的ACID，均需要对进程进行清理，在这点上两个数据库是相似的。但在进程清理的具体机制实现上略有不同。在Oracle中，PMON进程会监控其他进程，当其他进程被kill或有其他异常时，PMON进程负责清理、恢复、回滚以及资源的释放。而PG需要用户进程自己完成事务状态的更改，并且释放所有的资源。在清理公共资源的时候，进程间需要通过信号量（semaphore）来进行通讯确保资源清理和释放有效，并且是安全的，因此高并发下需要较多的时间结束进程 。

# pg_squeeze

```
[postgres:5566@postgres] [11-13.10:43:39]=# \d+ test
                                        Table "public.test"
 Column |          Type          | Collation | Nullable |          Default          | Storage  | Stats target | Description
--------+------------------------+-----------+----------+---------------------------+----------+--------------+-------------
 id     | integer                |           | not null |                           | plain    |              |
 info   | text                   |           |          | 'xxx'::text               | extended |              |
 info2  | character varying(100) |           |          | 'xxx'::character varying  | extended |              |
Indexes:
    "test_pkey" PRIMARY KEY, btree (id)

[postgres:5566@postgres] [11-13.10:43:41]=# select squeeze.squeeze_table('public','test','test_pkey','pg_default','{{test_pkey,pg_default}}');
 squeeze_table
---------------


(1 row)

Time: 15.861 ms
```

优势：
1. 相比vacuum full，pg_repack或pg_reorg，pg_squeeze不需要建触发器，所以在重组时对原表的DML几乎没有性能影响。
2. pg_squeeze支持自动的重组，即通过设置阈值、比较用户表与阈值，自动启动WORKER进程，将数据复制到重组表，最后加锁，切换FILENODE。

使用限制：
1. 9.6及以上版本。
2. 表上必须有PK或者UK。
3. 由于pg_squeeze需要使用logical replication，所以必须设置足够多的slots，而且必须注意可能与STANDBY争抢SLOTS。
4. 不建议在负载较高的库启用自动收缩，否则影响性能。

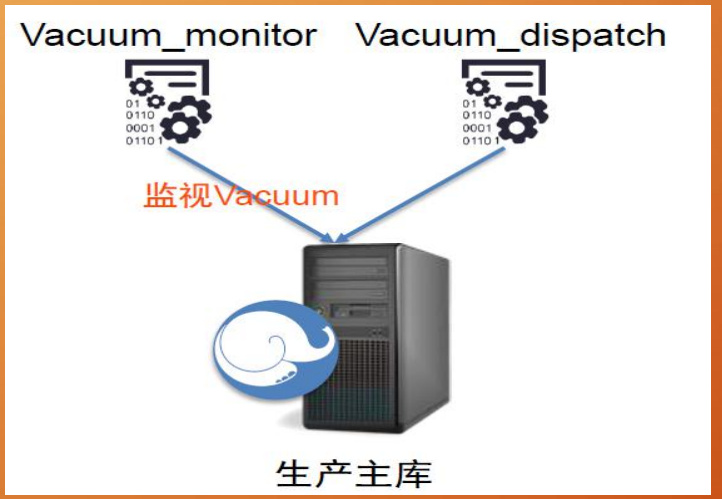
# 监控

- Xid监控，分级告警
- 长事物监控
- 添加大表监控告警
- 达到vacuum触发的条件，但是7天内未完成vacuum生成报表
- 增加定时vacuum  cron。

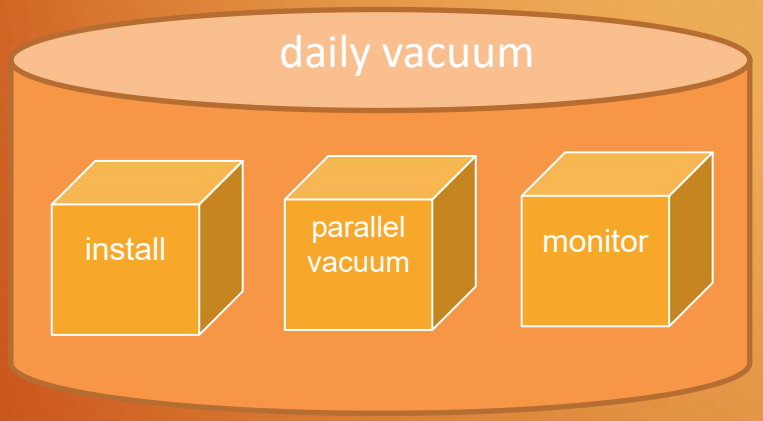
# daily vacuum

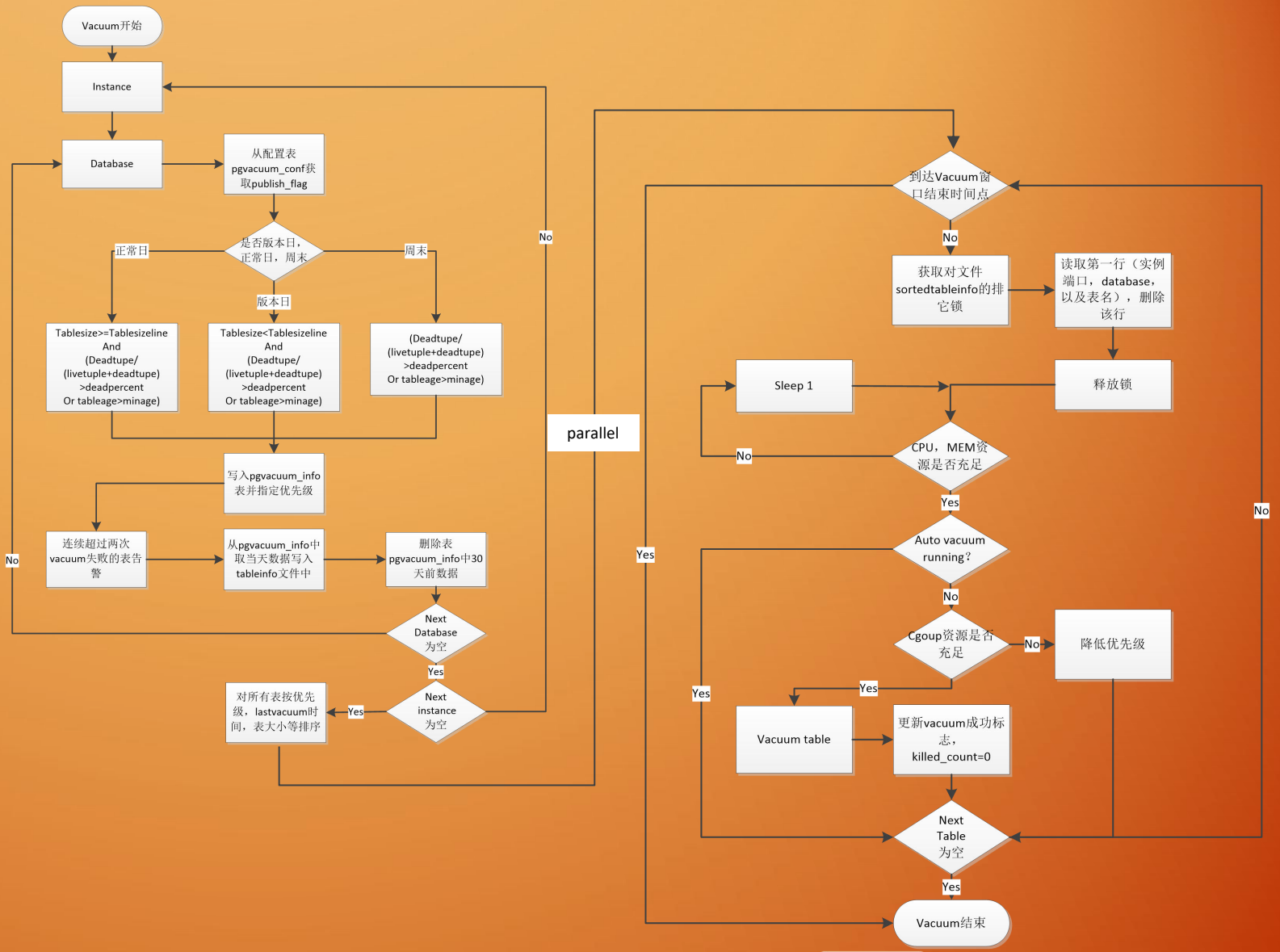
**daily Vacuum：**

1.利用业务空闲时间窗加大vacuum数量
2.并发度根据主机资源动态调整
3.vacuum内存根据表大小动态调整
4.任务队列根据表大小，age，dead tuple PCT，失败次数等条件全局排序，避开AutoVacuum表
5.记录所有Vacuum执行情况供事后分析
6.增加Vacuum Monitor进程，
- 中断阻塞DDL的Vacuum进程
- 维护时间窗口外终止任务队列，取消非auto的vacuum进程。



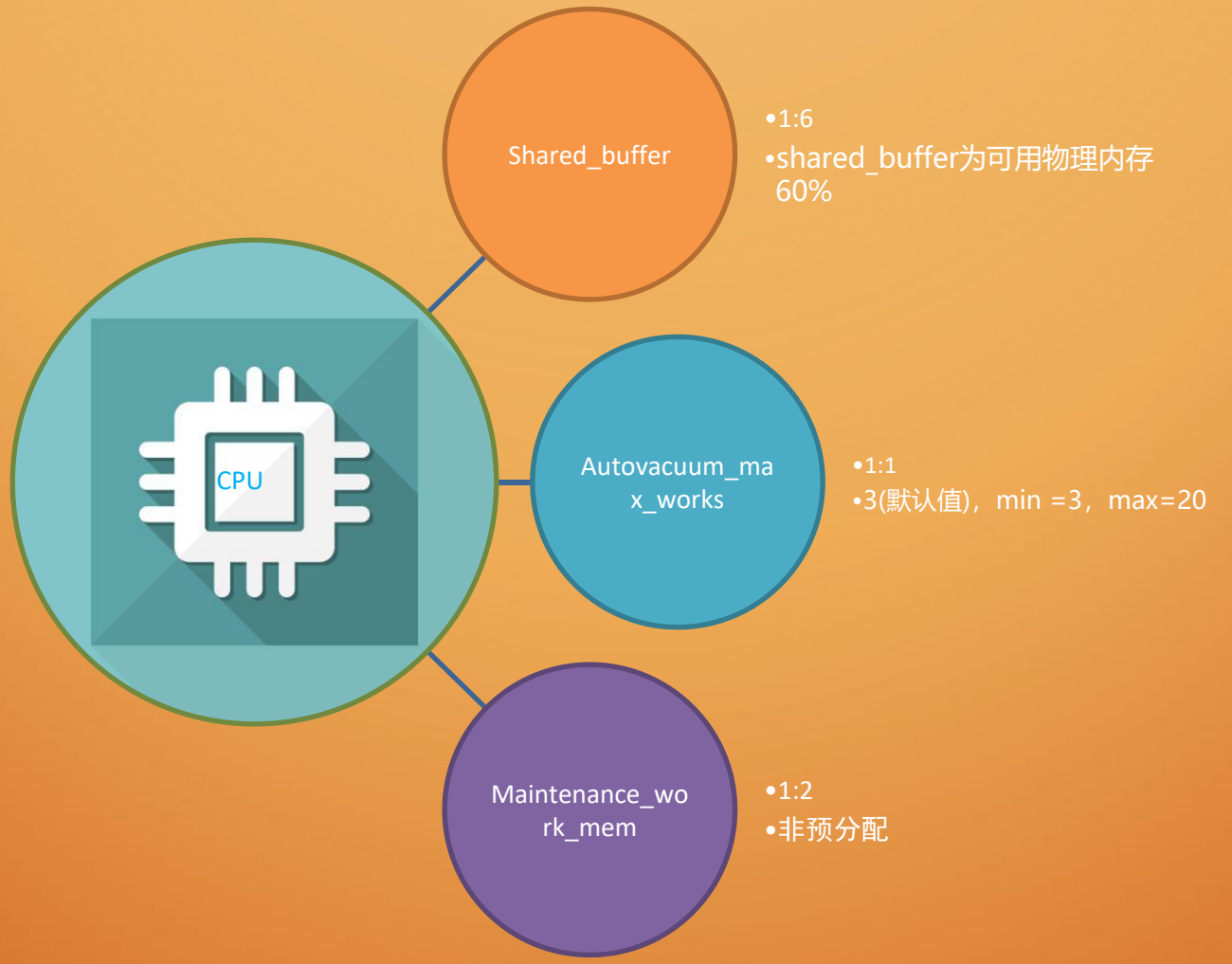
# daily vacuum



# 庖丁解牛-治理vacuum

# 参数调优




同时在调整autovacuum_max_workers时需要考虑database个数，尽量让每个database可以分配到一个autovacuum进程，避免个别database饿死。

# Strace分析

```
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 72.32    0.024996          10      2443           select
 26.46    0.009146           1     14655           read
  1.21    0.000419           0      3694           semop
  0.00    0.000000           0         1           lseek
------ ----------- ----------- --------- --------- ----------------
100.00    0.034561                 20793           total
```

```
38802      0.000132 select(0, NULL, NULL, NULL, {0, 20000}) = 0 (Timeout) <0.020145>
38802      0.020207 semop(108364390, {{9, 1, 0}}, 1) = 0 <0.000017>
38802      0.000061 read(461, "6#\0\0\260m\210\376\0\0\4\0\364\0\20\1\0 \4 \0\0\0\0p\237\22\1\340\236\22\1"..., 8192) = 8192 <0.000020>
38802      0.000066 read(461, "6#\0\0\320\215\210\376\0\0\4\0\374\0\200\1\0 \4 \0\0\0\0x\237\20\1\360\236\20\1"..., 8192) = 8192 <0.000018>
38802      0.000061 read(461, "6#\0\0X\256\210\376\0\0\4\0\364\0\20\1\0 \4 \0\0\0\0p\237\22\1\340\236\22\1"..., 8192) = 8192 <0.000030>
38802      0.000070 read(461, "6#\0\0\340\316\210\376\0\0\4\0\364\0\20\1\0 \4 \0\0\0\0p\237\22\1\340\236\22\1"..., 8192) = 8192 <0.000017>
38802      0.000058 read(461, "6#\0\0h\357\210\376\0\0\4\0\364\0\20\1\0 \4 \0\0\0\0p\237\22\1\340\236\22\1"..., 8192) = 8192 <0.000017>
38802      0.000058 read(461, "6#\0\0\360\17\211\376\0\0\4\0\364\0\20\1\0 \4 \0\0\0\0p\237\22\1\340\236\22\1"..., 8192) = 8192 <0.000017>
38802      0.000055 select(0, NULL, NULL, NULL, {0, 20000}) = 0 (Timeout) <0.020078>
38802      0.020124 read(461, "6#\0\0H0\211\376\0\0\4\0\374\0H\1\0 \4 \0\0\0\0x\237\20\1\360\236\20\1"..., 8192) = 8192 <0.000020>
38802      0.000068 read(461, "6#\0\0\330P\211\376\0\0\4\0\0\1\20\1\0 \4 \0\0\0\0x\237\20\1\360\236\20\1"..., 8192) = 8192 <0.000070>
38802      0.000115 read(461, "6#\0\0008q\211\376\0\0\4\0\0\1@\1\0 \4 \0\0\0\0x\237\20\1\360\236\20\1"..., 8192) = 8192 <0.000018>
38802      0.000058 read(461, "6#\0\0\300\221\211\376\0\0\4\0\364\0\20\1\0 \4 \0\0\0\0p\237\22\1\340\236\22\1"..., 8192) = 8192 <0.000017>
38802      0.000073 read(461, "6#\0\0H\262\211\376\0\0\4\0\364\0\20\1\0 \4 \0\0\0\0p\237\22\1\340\236\22\1"..., 8192) = 8192 <0.000016>
38802      0.000069 semop(108397159, {{15, 1, 0}}, 1) = 0 <0.000015>
38802      0.000042 read(461, "6#\0\0\220\322\211\376\0\0\4\0\374\0X\1\0 \4 \0\0\0\0x\237\20\1\360\236\20\1"..., 8192) = 8192 <0.000018>
38802      0.000055 select(0, NULL, NULL, NULL, {0, 20000}) = 0 (Timeout) <0.020078>
```

```
DESCRIPTION
       select() and pselect() allow a program to monitor multiple file descriptors, waiting until one or more of the file descriptors become "ready" for some class of I/O
       operation (e.g., input possible).  A file descriptor is considered ready if it is possible to perform the  corresponding  I/O  operation  (e.g.,  read(2))  without
       blocking.
```

# Pstack及源码分析

#0 0x00000032b00e1433 in __select_nocancel () from /lib64/libc.so.6
#1 0x00000000007836ea in pg_usleep ()
#2 0x00000000005874ec in vacuum_delay_point ()
#3 0x00000000005899d3 in lazy_vacuum_rel ()
#4 0x0000000000588465 in vacuum_rel ()
#5 0x00000000005887bf in vacuum ()
#6 0x0000000000618fa1 in do_autovacuum ()
#7 0x0000000000619436 in AutoVacWorkerMain ()
#8 0x0000000000619506 in StartAutoVacWorker ()
#9 0x0000000000626ea8 in sigusr1_handler ()
#10 <signal handler called>
#11 0x00000032b00e1433 in __select_nocancel () from /lib64/libc.so.6
#12 0x000000000062835e in PostmasterMain ()
#13 0x00000000005c15e8 in main ()

```
1423  /*
1424   * vacuum_delay_point --- check for interrupts and cost-based delay.
1425   *
1426   * This should be called in each major loop of VACUUM processing,
1427   * typically once per page processed.
1428   */
1429  void
1430  vacuum_delay_point(void)
1431  {
1432      /* Always check for interrupts */
1433      CHECK_FOR_INTERRUPTS();
1434
1435      /* Nap if appropriate */
1436      if (VacuumCostActive && !InterruptPending &&
1437          VacuumCostBalance >= VacuumCostLimit)
1438      {
1439          int         msec;
1440
1441          msec = VacuumCostDelay * VacuumCostBalance / VacuumCostLimit;
1442          if (msec > VacuumCostDelay * 4)
1443              msec = VacuumCostDelay * 4;
1444
1445          pg_usleep(msec * 1000L);
1446
1447          VacuumCostBalance = 0;
1448
1449          /* update balance values for workers */
1450          AutoVacuumUpdateDelay();
1451
1452          /* Might have gotten an interrupt while sleeping */
1453          CHECK_FOR_INTERRUPTS();
1454      }
1455  }
1456
```

# 参数调优

调整autovacuum_vacuum_cost_limit(默认200)为1000,autovacuum_vacuum_cost_delay为10ms。
vacuum_cost_page_miss和vacuum_cost_page_dirty是否需要调小?
在VACUUM和ANALYZE命令的执行过程中，系统维持着一个内部计数器来跟踪各种被执行的I/O操作的估算开销。当累计的代价达到一个限制（由vacuum_cost_limit指定），执行这些操作的进程将按照vacuum_cost_delay所指定的休眠一小段时间。然后它将重置计数器并继续执行。

PG9.6及以上版本，调整old_snapshot_threshold（默认值为3h）避免长事物对vacuum影响， 调整前需要和开发运营评估是否有job或者其他长事物，如果有需要确认长事物运行时间，根据具体情况设置，否则会有快照过旧的报错。

对于频繁update的表fillfactor建议设置为50，另外是否调整autovacuum_vacuum_scale_factor为0.1?

# 分区

Pathman：

优点：

支持HASH和RANGE分区，后续会支持LIST分区
支持自动和手动的分区维护
为分区表生成更有效的执行计划
通过引入两个自定义的执行计划节点RuntimeAppend & RuntimeMergeAppend，实现运行时动态添加分区到执行计划中
为新插入数据自动创建分区（只对RANGE分区）
提供用户callbacks接口处理创建分区事件。
提供在线分区实施（在线重定义），父表数据迁移到子表，拆分，合并分区

不足：

不支持list分区；不支持二级分区；权限，索引，trigger等无法继承；
修改主键默认的seq需要重建分区。

PG11内置分区

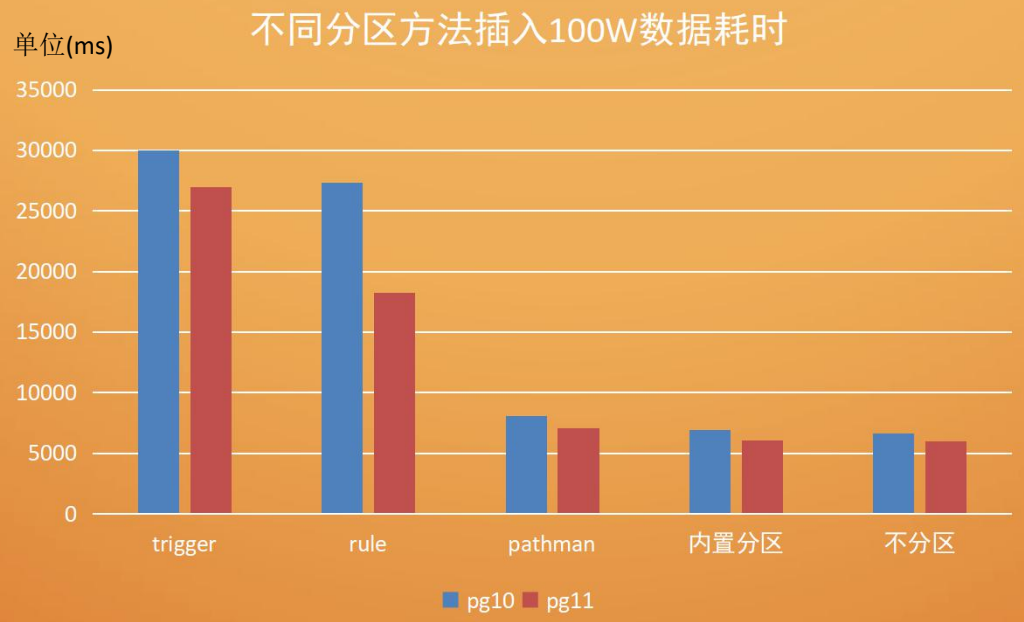
优点：

支持hash，range，list分区
支持多字段组合分区，支持表达式分区
支持创建主键，外键，索引，分区表自动继承。
支持update分区键
支持分区表DETACH，ATTACH，支持二级分区
分区自动创建
Default partition
Partition improvements

不足：

在主表添加权限，索引，trigger等无法继承
分区表不可以作为其他表的外键主表

# 分区


不同分区方法插入100W数据耗时
单位(ms)
35000
30000
25000
20000
15000
10000
5000
0
trigger
rule
pathman
内置分区
不分区
pg10
pg11

# 版本升级

## E.12. Release 9.6

E.12.3.1.6. VACUUM

Avoid re-vacuuming pages containing only frozen tuples (Masahiko Sawada, Robert Haas, Andres Freund)

Formerly, anti-wraparound vacuum had to visit every page of a table, even pages where there was nothing to do. Now, pages containing only already-frozen tuples are identified in the table's visibility map, and can be skipped by vacuum even when doing transaction wraparound prevention. This should greatly reduce the cost of maintaining large tables containing mostly-unchanging data.

If necessary, vacuum can be forced to process all-frozen pages using the new DISABLE_PAGE_SKIPPING option. Normally this should never be needed, but it might help in recovering from visibility-map corruption.

Avoid useless heap-truncation attempts during VACUUM (Jeff Janes, Tom Lane)

This change avoids taking an exclusive table lock in some cases where no truncation is possible. The main benefit comes from avoiding unnecessary query cancellations on standby servers.

## E.7. Release 10

Improve speed of VACUUM's removal of trailing empty heap pages (Claudio Freire, Álvaro Herrera).

## E.1. Release 12+

Zheap-A Storage Engine to Provide Better Control Over Bloat.

# Thanks