

PostgreSQL 未来技术展望

以 PostgreSQL 18 为起点，展望社区未来技术挑战与突破

目录



01

PostgreSQL 发展历史



02

更新时的索引写放大



03

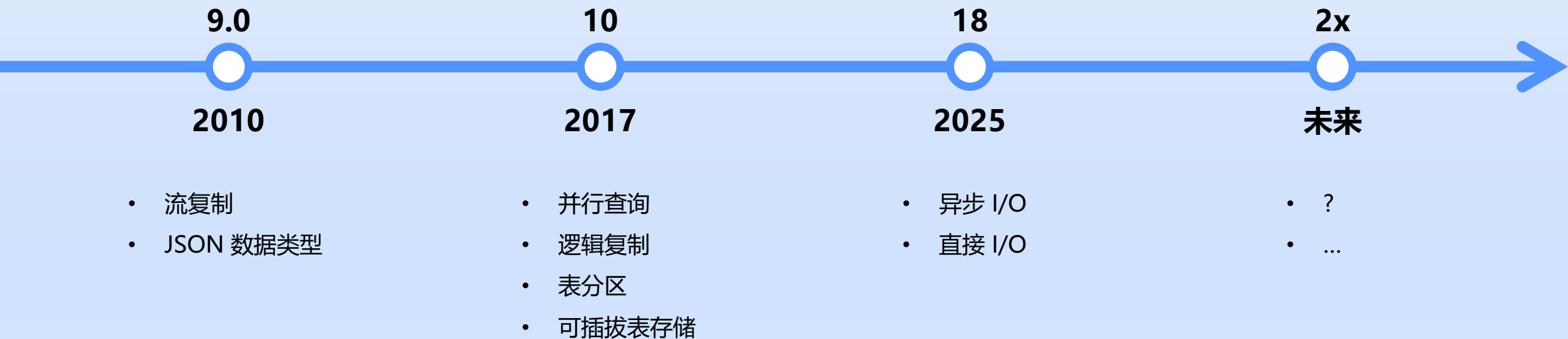
旧行版本清理的高成本和随机性



04

表行冻结与可能的停机

PostgreSQL 发展历史



- PostgreSQL 已经持续迭代三十年，近几年来用户基数不断扩大；
- 四次获得 DB-Engines 年度数据库，连续三年获得 Stack Overflow 排行榜首；
- 未来的 PostgreSQL 版本将会更加强大，你最期待的特性能力是什么？

更新时的索引写放大

```
CREATE TABLE boat (  
  id integer,  
  name text,  
  reg double precision  
);  
  
INSERT INTO boat(id, name, reg)  
  SELECT i, md5(i::text), log(i)  
  FROM generate_series(1, 500000) as i;  
  
CREATE INDEX boat_idx1 ON boat(id);  
CREATE INDEX boat_idx2 ON boat(reg);  
  
VACUUM ANALYZE boat;  
CHECKPOINT;
```

我们在 PostgreSQL 的数据库中，创建一个表，插入一些数据，并对表数据进行更新操作，查看更新操作修改的页面数目。

我们进行了检查点操作，刷写了缓冲区中的所有脏页面。这样，就可以通过 EXPLAIN ANALYZE 来查看 UPDATE 修改的页面情况。

更新时的索引写放大

```
EXPLAIN (analyze, buffers, wal, costs false)
UPDATE boat SET name = 'trade winds'
WHERE mod(id, 1000) = 1;
```

QUERY PLAN

```
-----
Update on boat (actual time=119.992..119.993 rows=0 loops=1)
  Buffers: shared hit=8680 read=1012 dirtied=1509 written=4
  WAL: records=2000 fpi=1501 bytes=11616847
  -> Seq Scan on boat (actual time=0.021..95.015 rows=500 loops=1)
        Filter: (mod(id, 1000) = 1)
        Rows Removed by Filter: 499500
        Buffers: shared hit=4673
Planning:
  Buffers: shared hit=73 read=1
Planning Time: 0.625 ms
Execution Time: 120.126 ms
(11 rows)
```

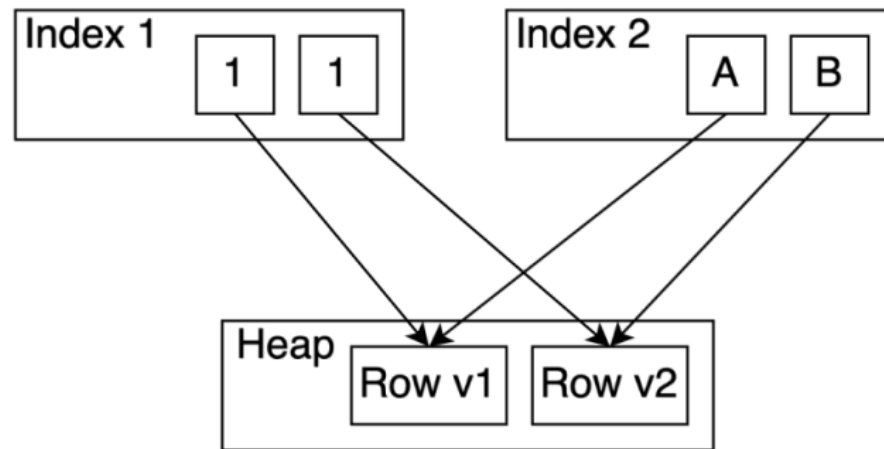
这里的 UPDATE 操作更新了 500 条记录，**不涉及修改索引相关的列**。总共修改了 1509 个页面，里面涉及约 500 个堆表页面和 **1000 个索引页面**。

由于刚刚进行过检查点操作，在 WAL 日志中产生了 1501 个页面镜像记录，WAL 日志总量为 **11.08 MB**。

注意：
PostgreSQL 服务器在检查点后对每个磁盘页面进行首次修改期间，会将该页面的全部内容写入 WAL 日志。

更新时的索引写放大

```
CREATE TABLE test ( id integer, name text );  
  
INSERT INTO test(id, name) VALUES(1, 'A');  
  
UPDATE test SET name = 'B' WHERE id = 1;
```



更新的原理与现象

PostgreSQL 采用多版本并发控制（MVCC）机制，当数据被更新或删除时，原始记录不会立即移除，而是标记为不可见，以支持事务回滚和快照读。更新操作在将旧行标记为不可见后，会在堆表中创建新的行版本（新元组）。

当新元组与旧元组不位于同一数据页，或者虽然处于同一数据页，但更新修改了索引相关的列时，就需要创建新的索引条目。例如，在一个订单状态频繁更新的订单表中，每次更新都需要创建新的索引条目，如果该表有 3 个索引，每次更新需要修改 4 个页面。

旧行版本清理的高成本和非确定性

会话一

```
CREATE TABLE t_large (id integer, name text);
INSERT INTO t_large (id, name)
  SELECT i, repeat('Pg', 64)
    FROM generate_series(1, 1000000) AS s(i);
```

```
CREATE INDEX large_idx ON t_large (id);
VACUUM ANALYZE t_large;
```

让我们对表中的一部分数据进行一些更新，当然，也可以删除一部分数据，让表中产生一定量的旧行版本。

```
UPDATE t_large SET name = 'dummy'
  WHERE mod(id, 100) = 1;
```

```
DELETE FROM t_large
  WHERE mod(id, 100) = 50;
```

会话二

在准备好基础表和初始数据后，让我们启动一个会话，在该会话中开始一个事务，并保持在空闲状态。

```
BEGIN ISOLATION LEVEL REPEATABLE READ;
SELECT name FROM t_large WHERE id = 202510;
...
```

旧行版本清理的高成本和非确定性

会话一

```
VACUUM (verbose) t_large;  
INFO: vacuuming "postgres.public.t_large"  
INFO: finished vacuuming "postgres.public.t_large": index scans: 0  
pages: 0 removed, 20409 remain, 20409 scanned (100.00% of total)  
tuples: 0 removed, 1010000 remain, 20000 are dead but not yet removable  
removable cutoff: 753, which was 2 XIDs old when operation ended  
frozen: 0 pages from table (0.00% of total) had 0 tuples frozen  
index scan not needed: 0 pages from table (0.00% of total) had 0 dead item identifiers removed  
avg read rate: 0.000 MB/s, avg write rate: 0.000 MB/s  
buffer usage: 40833 hits, 0 misses, 0 dirtied  
WAL usage: 1 records, 0 full page images, 188 bytes  
system usage: CPU: user: 0.12 s, system: 0.00 s, elapsed: 0.12 s  
...
```

上面的 VACUUM 操作扫描了全表 20409 个页面，没有清理任何的旧行版本。由于会话二中的事务还在进行中，系统需要保留旧行版本以对该事务保持可见性，会话一的 VACUUM 操作只会做无用功，不会清理任何的垃圾数据。

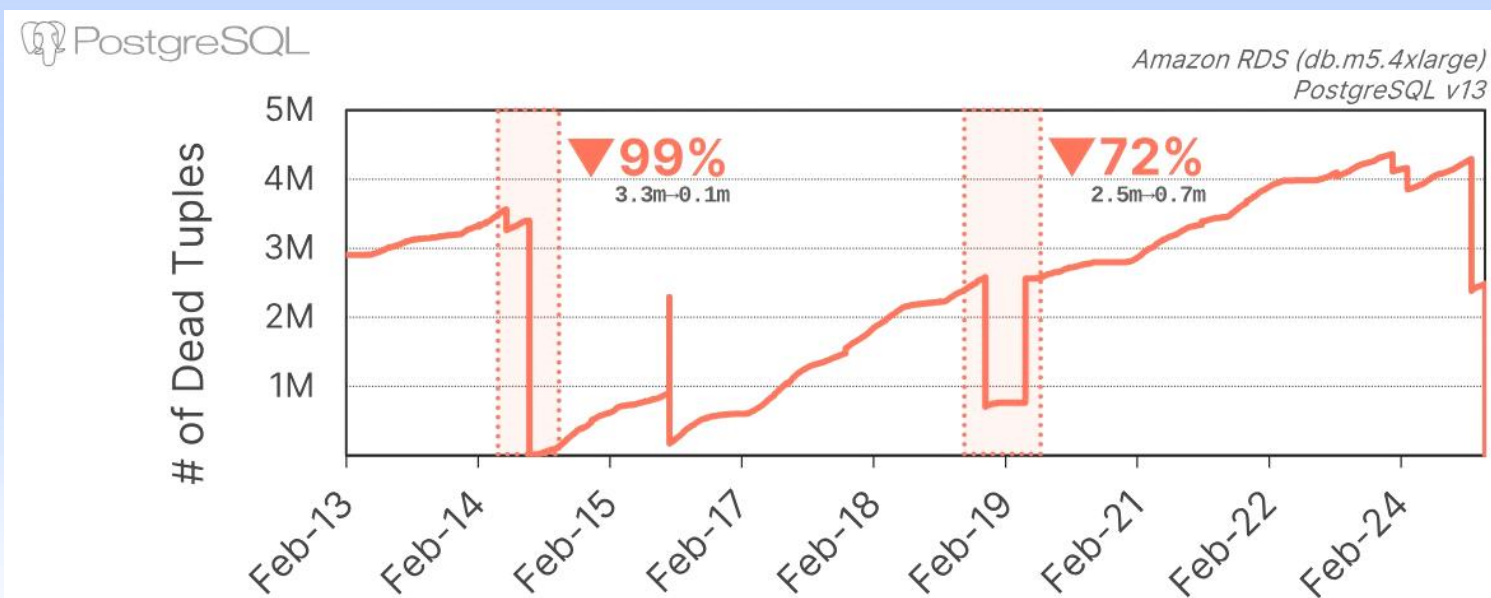
对于后台 autovacuum 进程定期调度执行的 VACUUM 动作，同样也会出现上面的问题，它会频繁消耗存储的 IO 资源，实际上没有清理掉任何的旧行版本。而且，随着后续不断地进行更新和删除操作，系统产生的旧行版本会持续积累，最终会给系统造成如同雪崩的效应。

旧行版本清理的高成本和非确定性

清理成本高的场景分析

PostgreSQL 的自动清理默认配置并非对所有表都适用，尤其是大型表。例如，配置项 `autovacuum_vacuum_scale_factor`（控制“表更新比例达到多少时触发自动清理”）的默认值为 20%。这意味着：若一个表包含 1 亿个元组，只有当至少 2000 万个元组被更新时，数据库才会触发自动清理。这种阈值设置会导致表中长时间保留大量死元组，进而增加 I/O 和内存成本。

PostgreSQL 自动清理的另一个问题是：它可能被“长时间运行的事务”阻塞，导致死元组堆积和统计信息过期。未能及时清理过期版本会引发一系列性能问题，而性能下降又会导致更多长时间运行的事务，进一步阻塞自动清理，形成“恶性循环”。这种情况往往需要人工干预，手动终止长时间运行的事务。下面图表展示了某数据库在两周内的死元组数量变化：



注：图表数据来源于
ottertune

表行冻结与可能的停机

问题现象：数据库出现错误：“数据库不接受命令”，应用程序陷入停机状态。

ERROR: *database is not accepting commands* that assign new XIDs to avoid wraparound data loss in database "postgres"

HINT: Execute a database-wide VACUUM in that database.

You might also need to commit or roll back old prepared transactions, or drop stale replication slots.

可能原因

某个数据库事务长期保持打开状态 (idle in transaction)

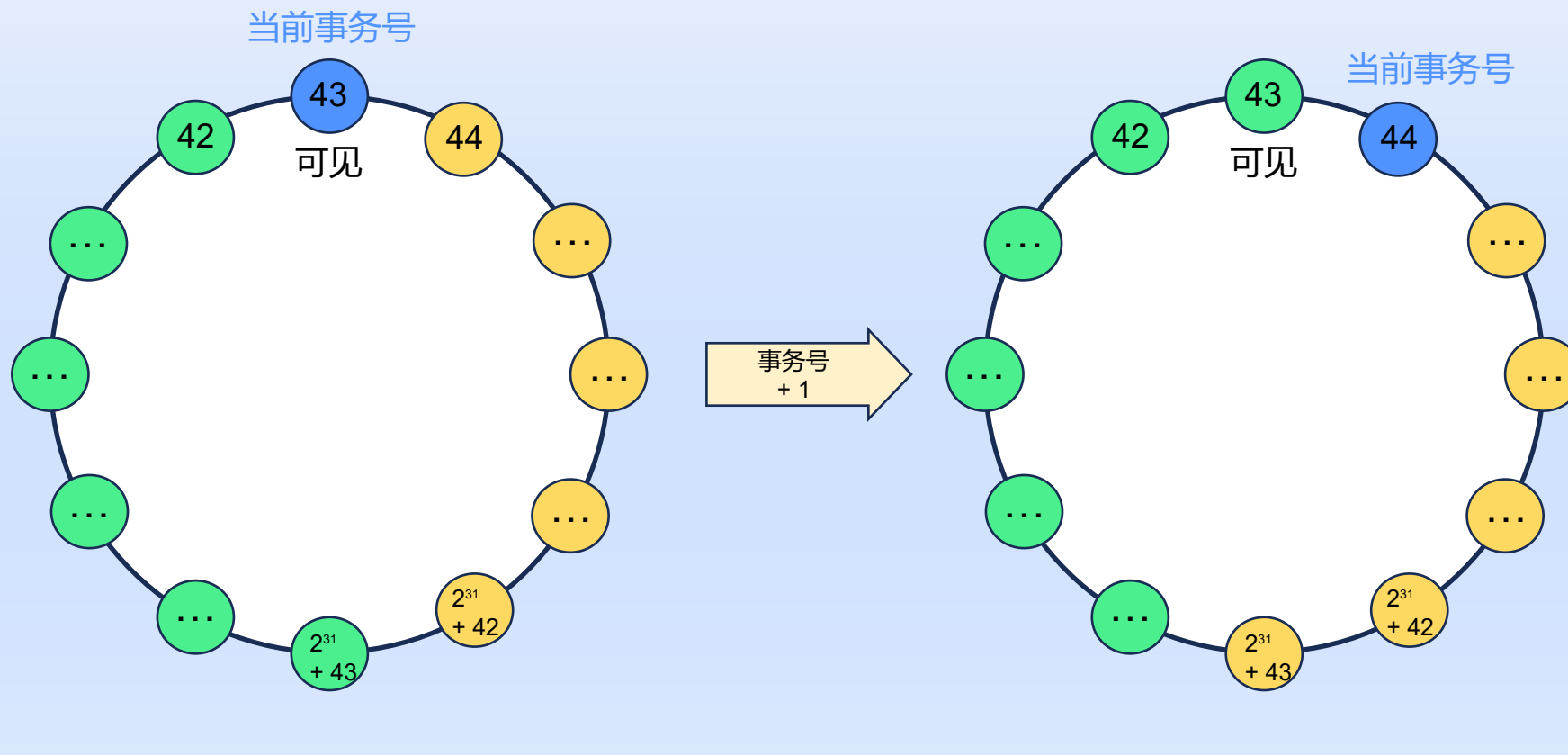
某个查询的执行时间过长，或者发生死锁被阻塞

存在未提交或未回滚的预备事务

存在遗留的复制槽，且备用服务器已启用 hot_standby_feedback (热备反馈) 功能

存在数据损坏，导致 VACUUM 操作失败

表行冻结与可能的停机



每个事务/子事务都有一个 32 位的事务号，它有点像数据库系统的运行时间

每个行版本都需要标记事务号，以记录它们的创建和删除时间

事务号最大值约为 20 亿，在到达阈值前，需要扫描所有改过的数据行，进行事务号的冻结操作；
否则，MVCC 的机制将无法再正常工作

谢谢聆听



www.ifclub.com.cn