中启乘数科技 @ PostgreSQL 14版本新特性介绍

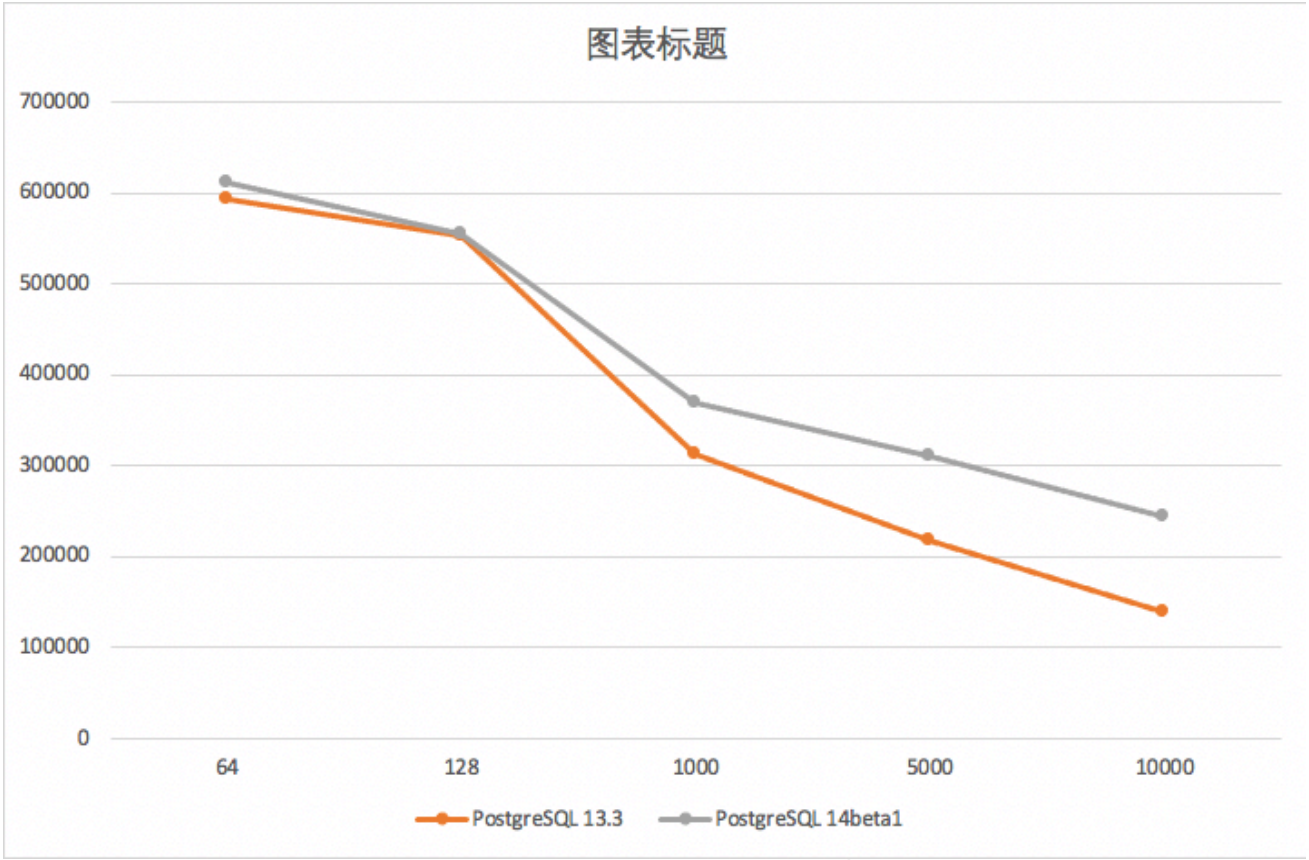
PostgreSQL中文社区

性能

- 这个版本对有大量数据库连接的 PostgreSQL 系统的事务吞吐量有明显的改进，不管它们是处于活动状态还是空闲状态：
  - http://www.pgsql.tech/article_101_10000104

2路服务器：Intel(R) Xeon(R) Silver 4210R CPU @ 2.40GHz
造2千万数据：
pgbench -i -s 200
测试语句：
pgbench -S -c 64 -j 96 -M prepared -T30 -P 2

场景 / 连接数	PostgreSQL 13.3	PostgreSQL 14 beta1
64	593654	610892
128	553454	554202
1000	312592	369856
5000	217679	310764
10000	138422	244271



性能

- PostgreSQL 14 B-tree频繁更新时减少了膨胀
  - https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=d168b666823b6e0bcf60ed19ce24fb5fb91b8ccf
  - 让nbtree和heapam更好的配合，以便更积极的移除因MVCC产生的重复行
  - 避免因为多版本的重复行导致的索引块分裂
  - bottom-up index deletion
  - Deleting older versions in unique indexes to avoid page splits
  - Pass down "logically unchanged index" hint
    - https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=9dc718bd
  - the "extra tuples" simple deletion enhancement increases the number of index tuples deleted with almost any workload that has LP_DEAD bits set in leaf pages.

性能

- GiST 索引现在可以在其构建过程中对数据进行预排序，从而可以更快地创建索引并缩小索引。
  - https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=16fa9b2b30a357b4aea982bd878ec2e5e002dbcc
  - https://www.postgresql.org/message-id/1A36620E-CAD8-4267-9067-FB31385E7C0D%40yandex-team.ru
  - As the first user of this facility, add 'sortsupport' function to the point_ops opclass.
  - create table x as select point (random(),random()) from generate_series(1,3000000,1);
  - PostreSQL 14中：
    - postgres=# create index ON x using gist (point);
    - CREATE INDEX
    - Time: 2551.954 ms (00:02.552)
    - 大小为：148955136
  - PostgreSQL 13.3中
    - postgres=# create index ON x using gist (point);
    - CREATE INDEX
    - Time: 49804.780 ms (00:49.805)
    - 大小为：223264768

性能

- SP-GiST 索引现在支持覆盖索引，该索引使用户可以通过 INCLUDE 子句向索引添加其他不可搜索的列
  - https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=09c1c6ab4bc5764dd69c53ccfd43b2060b1fd090
  - create table students(p point, addr text, student text);
  - insert into students select point (random(),random()), seq, seq from generate_series(1,1000000,1) as seq;
  - create index on students using spgist (p) include(addr,student);
    - postgres=# explain analyze select p,addr,student from students where p >^ '(10,10)'::point;
    - QUERY PLAN
    -
    - ---
      Index Only Scan using students_p_addr_student_idx on students (cost=0.29..4966.28 rows=100000 width=28) (actual time=0.613..0.613 rows=0 loops=1)
    - Index Cond: (p >^ '(10,10)'::point)
    - Heap Fetches: 0
    - Planning Time: 0.057 ms
    - Execution Time: 0.630 ms
    - (5 rows)

性能

- BRIN 索引支持布隆过滤器
  - 目前BRIN索引的变种出现, 可以支持bloom filter, 也就是说它存储的是占位bits. 每个连续heap blocks, 存储一个占位bits, 被索引字段的hash value经过bloom hash填充占位bit
  - https://www.postgresql.org/message-id/5d78b774-7e9c-c94e-12cf-fef51cc89b1a%402ndquadrant.com
  - CREATE TABLE t (a int);
  - CREATE INDEX ON t USING brin (a int4_bloom_ops(false_positive_rate = 0.05, n_distinct_per_range = 100));
  - create index test_brin_idx on bloom_test using brin (id);
  - create index test_bloom_idx on bloom_test using brin (id uuid_bloom_ops);
  - create index test_btree_idx on bloom_test (id);
  - List of relations
    Schema | Name | Type | Owner | Table | Size
    --------+----------------+-------+-------+------------+---------
    public | test_bloom_idx | index | tomas | bloom_test | 12 MB
    public | test_brin_idx | index | tomas | bloom_test | 832 kB
    public | test_btree_idx | index | tomas | bloom_test | 6016 MB
    (3 rows)

性能

- BRIN 索引支持多区间
  - BRIN索引特别适合边界清晰的堆存储数据, 例如BLOCK 1到8 存储的id范围是100-10000, 9到16 存储的id范围是100001到200000, 检索id=1000时, 只需要扫描1到8号数据块.
  - 然而如果堆存储里面的边界不清晰(或者说存储顺序和value相关性不高), 那么被索引字段的值分布很零散,或者范围跨度很大时, 检索一个ID值时可能要扫描很多很多数据块.
  - 为了解决这个问题, PG 14 支持multi range brin, 1到8号块存储的ID范围可能是1-199, 10000-10019, 20000-20000, 占用5个value(1,199,10000,10019,20000), 一个blocks区间存储多少个value取决于values_per_range参数(8到256).
  - 当不断插入数据时, 这些范围还可以被合并
  - https://www.postgresql.org/message-id/c1138ead-7668-f0e1-0638-c3be3237e812@2ndquadrant.com
  - https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=ab596105b55f1d7fbd5a66b66f65227d210b047d
  - CREATE TABLE t (a int);
  - CREATE INDEX ON t USING brin (a int4_minmax_multi_ops(values_per_range=16));

性能

- 并行查询中的每个worker分配连续的多个BLOCK，提升性能。
  - Allocate consecutive blocks during parallel seqscans
  - https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=56788d2156fc32bd5737e7ac716d70e6a269b7bc
- PL/pgSQL 中的 RETURN QUERY 指令现在可以执行具有并行性的查询。
- REFRESH MATERIALIZED VIEW 现在也可以使用查询并行性。
  - REFRESH MATERIALIZED VIEW： Pass CURSOR_OPT_PARALLEL_OK to pg_plan_query() so that parallel plans are considered when running the underlying SELECT query

性能

- postgres_fdw 支持批量插入,
  - 为了加速FDW的写入, 避免写入时本地数据库和远端数据源的交互round-trip造成的延迟, PostgreSQL 在FDW接口(ExecForeignBatchInsert , GetForeignModifyBatchSize)中支持bulk操作, 减少round-trip交互次数

性能

- postgres_fdw并可以使用 IMPORT FOREIGN SCHEMA 导入表分区，
  - IMPORT FOREIGN SCHEMA import_source LIMIT TO (t1, nonesuch, t4_part) FROM SERVER loopback INTO import_dest4;
- postgres_fdw现在可以在外部表中执行 TRUNCATE。
- postgres_fdw 长连接
  - server option : keep_connections , 保持foreign server连接.
  - keep_connections off表示foreign 事务结束后 立即释放foreign连接.
  - keep_connections on表示保持foreign连接. 直到退出当前会话. 好处是高并发的foreign oltp业务, 减少大量建连接到开销. 坏处是总连接数可能增多, 访问过foreign server的连接都会被保持, 可以通过管理函数主动释放.
  - ALTER SERVER loopback OPTIONS (keep_connections 'off');
  - SELECT server_name FROM postgres_fdw_get_connections() ORDER BY 1;
  - ALTER SERVER loopback OPTIONS (SET keep_connections 'on');

性能

- 分区表
  - Avoid creating duplicate cached plans for inherited FK constraints cache， 节省内存
  - 更新或删除操作时可以定位到更精细的分区， 这种情况下性能有很大提高
    - set plan_cache_mode to 'force_generic_plan'
    - https://www.postgresql.org/message-id/CA+HiwqG7ZruBmmih3wPsBZ4s0H2EhywrnXEduckY5Hr3fWzPWA@mail.gmail.com
  - ALTER TABLE ... DETACH PARTITION .. CONCURRENTLY.
    - https://www.postgresql.org/message-id/20200803234854.GA24158@alvherre.pgsql

- alter table 支持两阶段rewrite
  - 只会持有短暂的锁
  - 例如vacuum full目前需要rewrite table, 全程排他锁, 所以需要pg_repack, 而有了这个2阶段特性支持, 可以在最后切换的时候加一个ddl锁即可

性能

- 在上一版本中引入的增量排序，现在可以被 PostgreSQL 14 中的窗口函数所使用。

```
EXPLAIN (COSTS OFF)
SELECT * FROM
  (SELECT depname,
          empno,
          salary,
          enroll_date,
          row_number() OVER (PARTITION BY depname ORDER BY enroll_date) AS first_emp,
          row_number() OVER (PARTITION BY depname ORDER BY enroll_date DESC) AS last_emp
    FROM empsalary) emp
WHERE first_emp = 1 OR last_emp = 1;
QUERY PLAN
------------------------------------------------------------------------------
 Subquery Scan on emp
   Filter: ((emp.first_emp = 1) OR (emp.last_emp = 1))
   ->  WindowAgg
         ->  Incremental Sort
               Sort Key: empsalary.depname, empsalary.enroll_date
               Presorted Key: empsalary.depname
```

性能

- 扩展的统计数据增加了更多的功能，现在可以应用于表达式。
  - https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=a4d75c86bf15220df22de0a92c819ecef9db3849
  - CREATE TABLE t (a int);
  - CREATE STATISTICS s ON mod(a,10), mod(a,20) FROM t;
  - ANALYZE t;
  - SELECT * FROM t WHERE mod(a,10) = 0 AND mod(a,20) = 0;
  - SELECT 1 FROM t GROUP BY mod(a,10), mod(a,20);
- Improve estimation of OR AND clauses using multiple extended statistics.
  - SELECT * FROM mcv_lists_multi WHERE a = 0 OR b = 0 OR c = 0 OR d = 0');

性能

- 大表search IN (consts) - linear search TO hash table probe (consts 个数>= MIN_ARRAY_SIZE_FOR_HASHED_SAOP)
  - https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=50e17ad281b8d1c1b410c9833955bc80fbad4078
  - select * from tbl where id in (1,2,3,4,5,6,7,8,9);
  - tbl表很大时linear search会成为巨大瓶颈
  - 当hash内存足够大：
    - select x from tbl where id = any (array[1,2,3,4,5,6,7,8,9]);
  - 或
    - select x from tbl where id = any(array(select xxx from t));
  - 当IN里面的常数大于等于MIN_ARRAY_SIZE_FOR_HASHED_SAOP (9)时, 会对IN里面的常数构造一个hash结构, 采用hash匹配, 而不是linear search.
  - select count(*) from generate_series(1,100000) n(i) where i in (<1000 random integers in the series>)
    - 640ms降低到30ms

性能

- libpq驱动 支持pipeline 模式， SQL请求支持异步化通信
  - https://www.postgresql.org/docs/devel/libpq-pipeline-mode.html
  - 在延迟大的网络中，大大提升处理速度
  - pgbench中使用pipeline:
    - \startpipeline -- 把多条SQL放这, 不需要等SQL结果返回即可连续发射
    - -- sql1
    - -- sql2
    - -- sql3 ...
    - \endpipeline
  - 性能提升多少倍取决于每一次交互的延迟在整个请求中的时间占比
  - 本地环境, 网络延迟只有0.1毫秒左右. 50条SQL, 采用pipeline模式比普通事务性能提升4倍. 如果网络延迟达到1毫秒, 性能将提升50倍.

性能

- PostgreSQL 一直支持对其 "超大数据" 列的压缩（即 TOAST 系统），但这个版本增加了可以选择使用 LZ4 压缩列的功能。
  - ./configure --prefix=/home/pg14/pgsql-14beta1 --with-lz4
    - create table t1(a text);
    - create table t2(a text compression lz4);
    - postgres=# insert into t1(a) select lpad('a',1000000,'a') from generate_series(1,1000);
    - INSERT 0 1000
    - Time: 5131.500 ms (00:05.131)
    - postgres=# insert into t2(a) select lpad('a',1000000,'a') from generate_series(1,1000);
    - INSERT 0 1000
    - Time: 335.604 ms
    - postgres=# truncate table t1,t2;
    - TRUNCATE TABLE
    - Time: 7.410 ms
    - postgres=# insert into t1(a) select lpad('a',1000000,'a') from generate_series(1,1000);
    - INSERT 0 1000
    - Time: 5155.426 ms (00:05.155)
    - postgres=# insert into t2(a) select lpad('a',1000000,'a') from generate_series(1,1000);
    - INSERT 0 1000
    - Time: 327.484 ms

数据类型+SQL

- 在现有的范围类型支持的基础上，PostgreSQL 14 增加了新的多范围类型，让你指定一个非连续范围的有序列表，例如：
  - SELECT datemultirange(daterange('2021-07-01', '2021-07-31'), daterange('2021-09-01', '2021-09-30'), daterange('2021-11-01', '2021-11-30'), daterange('2022-01-01', '2022-01-31'), daterange('2022-03-01', '2022-04-07'))
  - SELECT '{}'::int4multirange;
  - SELECT '{[3,7)}'::int4multirange;
  - SELECT '{[3,7), [8,9)}'::int4multirange;
- PostgreSQL 14 现在增加了一个通用的下标框架，用于检索嵌套对象中的信息。例如，你现在可以使用下标语法检索 JSONB 数据类型中的嵌套信息，例如。
  - SELECT ('{ "this": { "now": { "works": "in postgres 14!" }}}':jsonb) ['this']['now']['works'] ;

数据类型+SQL

- PostgreSQL 14 允许 GROUP BY 子句使用 DISTINCT 关键字来删除重复的 GROUPING SET 组合。
  - SQL 标准兼容: GROUP BY DISTINCT
  - https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=be45be9c33a85e72cdaeb9967e9f6d2d00199e09
  - select a, b, c
  - from (values (1, 2, 3), (4, null, 6), (7, 8, 9)) as t (a, b, c)
  - group by distinct rollup(a, b), rollup(a, c)
  - order by a, b, c;

数据类型+SQL

- 对于递归的普通表表达式（WITH 查询），PostgreSQL 14 增加了 SEARCH 和 CYCLE 的语法便利
  - SEARCH选择是广度优先还是深度优先
  - CYCLE循环检测。CYCLE默认已经使用了深度优先，所以如果使用了cycle就不要重复使用深度优先语法了
  - https://www.postgresql.org/docs/devel/queries-with.html#QUERIES-WITH-RECURSIVE
  - WITH RECURSIVE search_tree(id, link, data) AS (
  - SELECT t.id, t.link, t.data
  - FROM tree t
  - UNION ALL
  - SELECT t.id, t.link, t.data
  - FROM tree t, search_tree st
  - WHERE t.id = st.link
  -) SEARCH DEPTH FIRST BY id SET ordercol
  - SELECT * FROM search_tree ORDER BY ordercol;
  - WITH RECURSIVE search_tree(id, link, data) AS (
  - SELECT t.id, t.link, t.data
  - FROM tree t
  - UNION ALL
  - SELECT t.id, t.link, t.data
  - FROM tree t, search_tree st
  - WHERE t.id = st.link
  -) SEARCH BREADTH FIRST BY id SET ordercol
  - SELECT * FROM search_tree ORDER BY ordercol;

数据类型+SQL

- 对于递归的普通表表达式（WITH 查询），PostgreSQL 14 增加了 SEARCH 和 CYCLE 的语法便利
  - CYCLE循环检测，防止无限循环
  - https://www.postgresql.org/docs/devel/queries-with.html#QUERIES-WITH-RECURSIVE
  - WITH RECURSIVE search_graph(id, link, data, depth) AS (
  - SELECT g.id, g.link, g.data, 1
  - FROM graph g
  - UNION ALL
  - SELECT g.id, g.link, g.data, sg.depth + 1
  - FROM graph g, search_graph sg
  - WHERE g.id = sg.link
  -) CYCLE id SET is_cycle USING path
  - SELECT * FROM search_graph;

数据类型+SQL

- tid range scan
  - https://www.postgresql.org/docs/devel/queries-with.html#QUERIES-WITH-RECURSIVE
  - 原先：select ctid,* from test01 where ctid='(10,1)';
  - select * from test01 where ctid >= '(10,0)' and ctid < '(11,0)';
  - 并行全表数据更新
    - https://github.com/digoal/blog/blob/master/202102/20210228_01.md
- ECPG支持DECLARE语句以兼容oracle的PROC*C
  - EXEC SQL [AT ***connection_name***] DECLARE ***statement_name*** STATEMENT

数据类型+SQL

- PostgreSQL 14 还增加了对存储过程中 OUT 参数的支持，
- 在 PostgreSQL 14 中还有一个新的 date_bin 函数，可以将时间戳与指定的间隔对齐，这种技术被称为 "binning":
  - 常用于BI系统, 从指定时间点开始, 按指定interval分割bucket, 输入一个ts返回它对应的bucket(这个bucket的开始时间), 通常用于group聚合统计
  - postgres=# select date_bin('15 minutes' , timestamp '2021-06-03 23:54:55', '2021-06-01 00:00:00');
  - date_bin
  - ---------------------
  - 2021-06-03 23:45:00
  - (1 row)
  - 类似timescaledb中的time_bucket()
  - SELECT time_bucket('5 minutes', datetime) AS bucket, COUNT(*) AS nb_datas FROM measures WHERE id_sensor = 123456 GROUP BY bucket HAVING COUNT(*) = 0 ORDER BY bucket DESC;

数据类型+SQL

- Create or Replace trigger support
- REINDEX concurrently on Partitioned table
- split_part function support negative index
  - select split_part('','@',-1) AS "empty string";
- bit_count 计算比特位1的个数
- 新增 bit_xor 聚合函数
- unistr 函数, 支持Unicode escapes字符串
  - unistr('d\0061t\+000061') → data
-

DBA管理

- PostgreSQL 14 继续对 VACUUM 进行改进，并针对索引进行了优化。Autovacuum 现在可以分析分区表，并可以将行数信息传播到父表。在 ANALYZE 中也有性能提升，可以用 maintain_io_concurrency 参数来控制。
- PostgreSQL 14 在可以监控的信息方面有很多改进，包括使用 pg_stat_progress_copy 视图跟踪 COPY 的进展。这个版本允许你从 pg_stat_wal 视图跟踪 WAL 活动，并从 pg_stat_replication_slots 视图检查复制槽的统计数据。
- 在 PostgreSQL 14中，有几个新的参数可以帮助管理连接。这些参数包括 idle_session_timeout，它可以在指定时间后关闭空闲的连接，以及 client_connection_check_interval 参数，它可以让 PostgreSQL 在客户端断开连接时取消长期运行的查询。
- REINDEX 命令现在可以处理一个分区表的所有子索引，PostgreSQL 14 增加了 pg_amcheck 工具来帮助检查数据损坏。

DBA管理

- 紧急情况下切换到全速VACUUM模式
  - vacuum_failsafe_age = 1600000000
  - vacuum_multixact_failsafe_age = 1600000000
  - 当年龄到达上面的值时，切换到全速VACUUM模式，忽略vacuum_cost_delay
  - 针对的是紧急情况, 即防止xid wraparound
- 防止长时间创建索引导致VACUUM不能回收垃圾
  - 当create index concurrently时, 只要不是表达式索引, partial index, 不是rc或ssi隔离级别, 那么这个操作的snapshot xmin就不会用做计算oldestxmin. 从而它运行多长时间都不会导致vacuum无法回收某些垃圾而导致膨胀.
  - https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=d9d076222f5b94a85e0e318339cfc44b8f26022d

DBA管理

- 当DEAD TUPLE覆盖的page较少时, 跳过index vacuum
  - 并且, LP是可能被索引引用的, 如果清理垃圾时lp也被清除, 那么表的index也要做一次vacuum, 把引用这个LP的index item也清除掉.
    为了避免每次vacuum都要清理index, PostgreSQL 14进行了优化, 当vacuum一个table时, 如果低于2%的PAGE有dead LP(例如一个表占用了100个page, 如果只有2个page里面有dead LP), 那么将跳过index vacuum, 并保留这些lp_dead.
    当table中的dead lp积累到超过2% page时, 才需要执行index vacuum.
  - 因为LP 只占用4字节, 所以不清理也影响不大, 但是大幅降低了index vacuum带来的vacuum负担.
  - 为什么是2%, 代码写死的, 未来也许会支持索引级别配置, 或者支持GUC配置
  - https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=5100010ee4d5c8ef46619dbd1d17090c627e6d0a

DBA管理

- 当DEAD TUPLE覆盖的page较少时, 跳过index vacuum
  - 并且, LP是可能被索引引用的, 如果清理垃圾时lp也被清除, 那么表的index也要做一次vacuum, 把引用这个LP的index item也清除掉.
    为了避免每次vacuum都要清理index, PostgreSQL 14进行了优化, 当vacuum一个table时, 如果低于2%的PAGE有dead LP(例如一个表占用了100个page, 如果只有2个page里面有dead LP), 那么将跳过index vacuum, 并保留这些lp_dead.
    当table中的dead lp积累到超过2% page时, 才需要执行index vacuum.
  - 因为LP 只占用4字节, 所以不清理也影响不大, 但是大幅降低了index vacuum带来的vacuum负担.
  - 为什么是2%, 代码写死的, 未来也许会支持索引级别配置, 或者支持GUC配置
  - https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=5100010ee4d5c8ef46619dbd1d17090c627e6d0a
- ANALYZE中使用预读以加快速度
  - 预读参数：maintenance_io_concurrency
  - 原理： posix_fadvise()

DBA管理

- COPY 导入数据时进度监控
  - 视图pg_stat_progress_copy
  - 导入多少行, 排除多少行(where filter)

```
postgres=# \d pg_stat_progress_copy
            View "pg_catalog.pg_stat_progress_copy"
      Column       |  Type   | Collation | Nullable | Default
-------------------+---------+-----------+----------+---------
 pid               | integer |           |          |
 datid             | oid     |           |          |
 datname           | name    |           |          |
 relid             | oid     |           |          |
 command           | text    |           |          |
 type              | text    |           |          |
 bytes_processed   | bigint  |           |          |
 bytes_total       | bigint  |           |          |
 tuples_processed  | bigint  |           |          |
 tuples_excluded   | bigint  |           |          |
```

DBA管理

- 新增pg_stat_wal统计信息视图, 跟踪wal日志统计信息
  - 增加 track_wal_io_timing GUC参数, 支持wal日志buffer write, fsync IO等待时长统计
  - postgres=# \d pg_stat_wal
  - View "pg_catalog.pg_stat_wal"
  - Column | Type | Collation | Nullable | Default
  - ---+---+---+---+---
  - wal_records | bigint | | |
  - wal_fpi | bigint | | |
  - wal_bytes | numeric | | |
  - wal_buffers_full | bigint | | |
  - wal_write | bigint | | |
  - wal_sync | bigint | | |
  - wal_write_time | double precision | | |
  - wal_sync_time | double precision | | |
  - stats_reset | timestamp with time zone | | |

DBA管理

- 新增 replication slot 统计信息视图 - pg_stat_replication_slots

```
postgres=# \d pg_stat_replication_slots
                View "pg_catalog.pg_stat_replication_slots"
    Column     |           Type           | Collation | Nullable | Default
---------------+--------------------------+-----------+----------+---------
 slot_name     | text                     |           |          |
 spill_txns    | bigint                   |           |          |
 spill_count   | bigint                   |           |          |
 spill_bytes   | bigint                   |           |          |
 stream_txns   | bigint                   |           |          |
 stream_count  | bigint                   |           |          |
 stream_bytes  | bigint                   |           |          |
 total_txns    | bigint                   |           |          |
 total_bytes   | bigint                   |           |          |
 stats_reset   | timestamp with time zone |           |          |
```

DBA管理

- pg_locks 增加 wait_start 字段, 跟踪锁等待开始时间

```
postgres=# \d pg_locks
                            View "pg_catalog.pg_locks"
       Column        |           Type           | Collation | Nullable |
Default
--------------------+--------------------------+-----------+----------+-------
--
 locktype            | text                     |           |          |
 database            | oid                      |           |          |
 relation            | oid                      |           |          |
...
...
 virtualtransaction  | text                     |           |          |
 pid                 | integer                  |           |          |
 mode                | text                     |           |          |
 granted             | boolean                  |           |          |
 fastpath            | boolean                  |           |          |
 waitstart           | timestamp with time zone |           |          |
```

DBA管理

- pg_stat_database 增加 active_time, idle_in_transaction_time, sessions, sessions_abandoned, sessions_fatal, sessions_killed统计 指标

```
postgres=# \d pg_stat_database
                          View "pg_catalog.pg_stat_database"
          Column           |           Type           | Collation | Nullable |
Default
---------------------------+--------------------------+-----------+----------+-
datid                      | oid                      |           |          |
datname                    | name                     |           |          |
...
...
session_time               | double precision         |           |          |
 active_time               | double precision         |           |          |
 idle_in_transaction_time  | double precision         |           |          |
 sessions                  | bigint                   |           |          |
 sessions_abandoned        | bigint                   |           |          |
 sessions_fatal            | bigint                   |           |          |
 sessions_killed           | bigint                   |           |          |
 stats_reset               | timestamp with time zone |           |          |
```

DBA管理

- pg_prepared_statements 增加硬解析、软解析次数统计
  - postgres=# \d pg_prepared_statements
  - View "pg_catalog.pg_prepared_statements"
  - Column | Type | Collation | Nullable | Default
  - ---+---+---+---+---
  - name | text | | |
  - statement | text | | |
  - prepare_time | timestamp with time zone | | |
  - parameter_types | regtype[] | | |
  - from_sql | boolean | | |
  - generic_plans | bigint | | |
  - custom_plans | bigint | | |

DBA管理

- 查看当前会话和其他会话的内存上下文, 诊断内存消耗问题
  - 查看自己的会话的内存上下文：
    - `select * from pg_backend_memory_contexts;`
  - 查看其他会话内存的分配情况：
    - select pid,query from pg_stat_activity ;
    - select pg_log_backend_memory_contexts(49554);
    - `less postgresql-2021-05-22_000000.csv`
    - `....`
    - `554,,60a71979.c192,21,,2021-05-21 10:22:49 CST,1/782,0,LOG,00000,"level: 2; hba parser context: 17408 total in 5 blocks; 8120 free (6 chunks); 9288 used",,,,,,,,,"","autovacuum launcher",,0`
    - `2021-05-22 09:51:03.097 CST,,,49554,,60a71979.c192,22,,2021-05-21 10:22:49 CST,1/782,0,LOG,00000,"level: 1; ErrorContext: 8192 total in 1 blocks; 7928 free (5 chunks); 264 used",,,,,,,,,"","autovacuum launcher",,0`

DBA管理

- 增加 log_recovery_conflict_waits 参数
  - 掌握只读standby库的查询和WAL恢复进程的冲突等待时间.
  - pg_ctl reload生效
- 增加 client_connection_check_interval 参数
  - 协议层支持心跳包, 如果客户端已离线, 可以快速中断这个客户端此前运行中的长SQL - Detect POLLHUP/POLLRDHUP while running queries
  - 用户如果发现SQL较慢, 直接退出终端, 在执行中的不接受中断信号过程中的SQL要等执行结束才会退出, 现在不需要等执行结束, 检测到客户端推出后SQL即刻推出.
- 增加会话超时参数idle_session_timeout

DBA管理

- REINDEX command 增加 tablespace 选项, 支持重建索引到指定表空间
  - REINDEX TABLESPACE new_tablespace ....
- REINDEX command 支持分区表, 自动重建所有子分区的索引.
- 新增 old_snapshot 插件, 打印快照跟踪条目(每分钟一条, OldSnapshotTimeMapping结构)的内容 , old_snapshot_threshold 相关
  - pg_old_snapshot_time_mapping(array_offset OUT int4, end_timestamp OUT timestamptz, newest_xmin OUT xid) returns setof record

DBA管理

- 新增 pg_surgery 插件, 可用于修复 corrupted tuple
  - 修复损坏的事务日志
  - test=> select * from t1 where ctid = '(0, 1)';
  - ERROR: could not access status of transaction 4007513275
  - DETAIL: Could not open file "pg_xact/0EED": No such file or directory.

  - test=# select heap_force_kill('t1'::regclass, ARRAY['(0, 1)']::tid[]);
  - heap_force_kill
  - ----------
  -
  - (1 row)

  - test=# select * from t1 where ctid = '(0, 1)';
  - (0 rows)

DBA管理

- 新增 pg_surgery 插件, 可用于修复 corrupted tuple
  - 修复错误(wrapped)的xid
  - test=> vacuum t1;
  - ERROR: found xmin 507 from before relfrozenxid 515
  - CONTEXT: while scanning block 0 of relation "public.t1"
  - test=# select ctid from t1 where xmin = 507;
  - ctid
  - -------
  - (0,3)
  - (1 row)
  - test=# select heap_force_freeze('t1'::regclass, ARRAY['(0, 3)']::tid[]);
  - heap_force_freeze
  - ------------------
  - (1 row)
  - test=# select ctid from t1 where xmin = 2;
  - ctid
  - -------
  - (0,3)
  - (1 row)

DBA管理

- pg_amcheck插件增加heap table数据页格式错误、逻辑错误检测功能
  - postgres=# create extension amcheck ;
  - CREATE EXTENSION

  - $ pg_amcheck -v postgres
  - pg_amcheck: including database "postgres"
  - pg_amcheck: in database "postgres": using amcheck version "1.3" in schema "public"
  - pg_amcheck: checking heap table "postgres"."public"."tbl"
  - pg_amcheck: checking btree index "postgres"."public"."idx_t_1"
  - pg_amcheck: checking btree index "postgres"."public"."idx_tbl_2"
  - ... ...

DBA管理

- 参数compute_query_id，　pg_stat_activity.query_id
- log_connections 支持打印更多内容, pg_hba第几行, 使用什么认证方法等, 方便判断客户通过什么方式在与数据库进行登陆认证
- autovacuum 打印更多信息, 每个索引的stats被打印
- huge_page_size参数指定大页是使用2MB还是1GB的大页
-

复制和恢复

- 长事务逻辑复制优化
  - 增加streaming接口，逻辑复制支持流式decoder和发送，无需等待事务结束，大幅度降低大事务、长事务的复制延迟
  - stream_start
  - stream_stop
  - stream_abort
  - stream_commit
  - stream_change
  - stream_message
  - stream_truncate
- alter subscription语法增强，支持add/drop publication
  - ALTER SUBSCRIPTION name SET PUBLICATION publication_name [, ...] [WITH (set_publication_option [= value] [, ...])]
  - ALTER SUBSCRIPTION name ADD PUBLICATION publication_name [, ...] [WITH (set_publication_option [= value] [, ...])]
  - ALTER SUBSCRIPTION name DROP PUBLICATION publication_name [, ...] [WITH (set_publication_option [= value] [, ...])]
  - ALTER SUBSCRIPTION name REFRESH PUBLICATION [WITH (refresh_option [= value] [, ...])]

复制和恢复

- PostgreSQL 14支持使用binary格式传输逻辑订阅数据. (暂时不包括用户自定义数组和composite类型)
- 在 PostgreSQL 14 中， 对 PostgreSQL 在崩溃恢复时的启动方式进行了性能改进
  - recovery_init_sync_method=syncfs | fsync
  - 解决表很多时, crash recovery 递归open所有file的性能问题 - 需Linux新内核支持

- pg_rewind可以使用standby库做为源库
- 增加 remove_temp_files_after_crash GUC参数, 在数据库crash后重启时自动清理临时文件
- standby wal receiver 接收时机优化, 无需等待startup process replay结束, 大幅度降低standby在重启后的wal接收延迟
- 参数 in_hot_standby 获取当前实例是否是standby角色
- 支持 restore_command 参数修改 reload生效, 无需重启实例

安全性

- PostgreSQL 14 增加了通过使用 pg_read_all_data 和 pg_write_all_data 预定义角色，分别给予用户在表/视图/序列上的通用 "只读" 和 "只写" 权限的能力。
  - select rolename from pg_roles ;
- 这个版本也默认在新的 PostgreSQL 实例上使用 SCRAM-SHA-256 进行密码管理。此外，pg_hba.conf 中的 clientcert 参数现在必须使用 verify-ca 或 verify-full 的值，而不是传统的值。
- PostgreSQL 14 可以在 pg_hba.conf 文件中使用证书的 "区分名称"（DN）来进行基于证书的认证，并使用 clientname=DN 参数。通常clientname = DN 可以 结合username mapping 使用。
  - CN: common name, DN: distinguished name
  - hostssl all all 0.0.0.0/0 clientcert=verify-full clientname=DN
  - hostssl all all 0.0.0.0/0 clientcert=verify-full # 如果不配置clientname则默认clientname=CN
  - hostssl all all 0.0.0.0/0 cert clientname=CN
  - hostssl all all 0.0.0.0/0 cert clientname=DN

安全性

- libpq协议层支持数据库状态判断(standby or primary)
  - 提高判断数据库角色的效率, 不需要发起SQL即可判断数据库处于 读写 还是 只读 角色
- libpq支持target_session_attrs属性配置: "any", "read-only", "read-write", "primary", "standby", and "prefer-standby".
  - 在多数据源场景中, 支持根据状态选择是否要连接该目标. 可以根据SQL分配不同的角色, 用于balance.
  - postgresql://host1:123,host2:456/somedb?target_session_attrs=any&application_name=myapp
- Add "pg_database_owner" default role. 表示数据库owner
  - ALTER TABLE datdba_only OWNER TO pg_database_owner;

4008878716

services@csudata.com

THANKS

乘数科技 CSUDATA

PostgreSQL中文社区