LightDB Administration Guide

发行版本 *23.1*

LightDB

2023 年 07 月 06 日

目录:

1 前言

本文档为恒生电子企业级数据库 LightDB 日常运维手册，主要介绍日常运维常用操作的指南。

2 LightDB 单机

2.1 GUI 安装界面为什么弹不出来？是否支持命令行安装模式？

GUI 安装界面弹不出来，一般来说有两种原因：

- Linux 系统未安装 GUI 程序所需的依赖包
- Linux 系统未正确设置 DISPLAY 环境变量，或者 Windows 未正确运行 Xmanager - Passive

如果无法满足上述条件，可以使用命令行安装模式，LightDB 支持命令行安装模式，且与 GUI 安装相比仅在安装向导上有所差异，其余并无不同。

2.2 查看 LightDB 安装目录、实例目录、归档目录

```
ls $LTHOME          # 查看安装目录
ls $LTDATA          # 查看实例目录
ls $LTHOME/archive  # 查看归档目录
```

2.3 LightDB 包含哪些日志？

数据库日志，位于 $LTDATA/log 目录中。

ltcluster 日志，位于 `$LTDATA/../etc/ltcluster/` 下，仅高可用版本有。

keepalived 日志，位于 `/var/log/` 下，并且在 `$LTHOME/etc/keepalived/keepalived_lightdb.log` 有 keepalived 检测 lightdb 的心跳日志，仅高可用版本需启用 keepalived。

2.4 查看数据库最新日志

LightDB 数据库日志路径为 `$LTDATA/log/` ，日志文件命名格式为 `lightdb-yyyy-mm-dd_hhmmss.log` ，可以此找到最新的日志文件，然后用 `tail` 命令循环查看指定行数的最新日志内容，如下图所示。

```
tail -fn 10 lightdb-yyyy-mm-dd_hhmmss.log
```

```
[lightdb@localhost ~]$ ls -lt $LTDATA/log
total 136
-rw-------. 1 lightdb lightdb 111827 Jan  3 18:59 lightdb-2023-01-03_185713.log
-rw-------. 1 lightdb lightdb      0 Jan  3 18:57 lightdb-2023-01-03_185713.slow
-rw-------. 1 lightdb lightdb   7344 Jan  3 18:57 lightdb-2023-01-03_185656.log
-rw-------. 1 lightdb lightdb      0 Jan  3 18:56 lightdb-2023-01-03_185656.slow
[lightdb@localhost ~]$ tail -fn 10 $LTDATA/log/lightdb-2023-01-03_185713.log
               before or while processing the request.
       could not send SSL negotiation packet: Success
2023-01-03 18:59:34.196702T,,,,,Canopy Maintenance Daemon: 14203/10,,XX000,2023-01-03 18:57:13 PST,0,87756,CONTEXT:  while executing command on 192.168.237.136:5436
        Canopy maintenance daemon for database 14203 user 10
2023-01-03 18:59:34.196712T,,,,,Canopy Maintenance Daemon: 14203/10,,XX000,2023-01-03 18:57:13 PST,0,87756,WARNING:  server closed the connection unexpectedly
               This probably means the server terminated abnormally
```

2.5 查看数据库日志中的错误信息

LightDB 日志中的错误信息包含 `ERROR` 或 `FATAL` 标签，可以此为关键词从日志文件中过滤错误行。

```
# 单次查看当前错误日志
cat lightdb-yyyy-mm-dd_hhmmss.log | grep -E 'ERROR|FATAL'

# 实时监控最新错误日志
tail -fn 10 lightdb-yyyy-mm-dd_hhmmss.log | grep -E 'ERROR|FATAL'
```

2.6 查看是否开启了慢日志，开启与关闭慢日志

在 LightDB 中慢日志配置参数有两处：数据库自身和 auto_explain 插件，使用 `show` 可以查看这两个参数。

```
show log_min_duration_statement;     -- 数据库慢日志，默认值-1
show auto_explain.log_min_duration; -- auto_explain慢日志，默认值100ms
```

数据库慢日志仅记录 SQL，auto_explain 慢日志同时记录 SQL 和执行计划，二者参数值的含义完全相同：

- -1 表示关闭慢日志
- 0 表示启用慢日志，且记录所有 SQL
- 大于 0（如 100ms、1s）表示启用慢日志，且仅记录 `elapsed time` 大于等于该时间的 SQL

在 LightDB 中，`log_min_duration_statement` 默认值为-1，`auto_explain.log_min_duration` 默认值为 100ms（前提是 auto_explain 已启用，默认不启用），若在 `lightdb.conf` 中修改了这两个参数，不用重启数据库，仅需 `reload` 重新加载即可生效。

```
lt_ctl -D $LTDATA reload
```

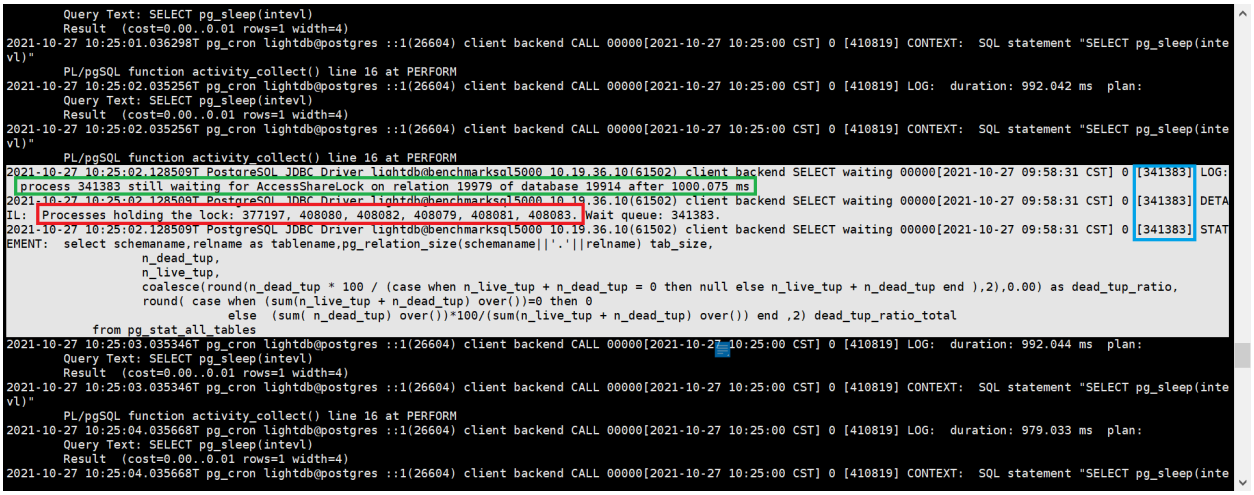
如果希望启用 auto_explain，则需要修改 `lightdb.conf` 中的 `shared_preload_libraries`，在其中添加 auto_explain，然后重启数据库。**对于 LightDB 单机版，可以直接执行 “lt_ctl -D $LTDATA restart“ 即可，但如果安装的是 LightDB 高可用或分布式，则务必按高可用和分布式的停止与启动步骤进行操作。**

2.7 查看锁表、阻塞者、阻塞者正在执行的 SQL

该语句可以查出当前数据库中的所有锁，注意是当前数据库，不是整个实例。虽然 pg_locks 本身是实例级的，但是 pg_class 是数据库级的，所以关联之后，其他数据库的锁会查询不到。

```
-- 查询当前数据库中的所有锁
SELECT d.datname, c.relname, c.reltype, a.*
FROM pg_catalog.pg_locks a, pg_catalog.pg_database d, pg_catalog.pg_class c
WHERE d.oid = a.database AND c.oid = a.relation;
```

对于长时间的锁监控，可以查看 LightDB 数据库日志，里面记录了阻塞者的 PID，如图中红圈所示，顺着 PID 向前查找蓝圈位置值（这个值代表当前日志行对应的进程 ID）等于 PID 的日志行，就可以找到阻塞者正在执行的 SQL。



2.8 查看当前正在执行的 SQL 是否被阻塞了

可以查看 LightDB 数据库日志，看是否有 "`process pid still waiting for xxxLock ......`" 的字样，如果有的话，顺着 pid 在上下文查找，就可以找到 process pid 对应的 SQL。

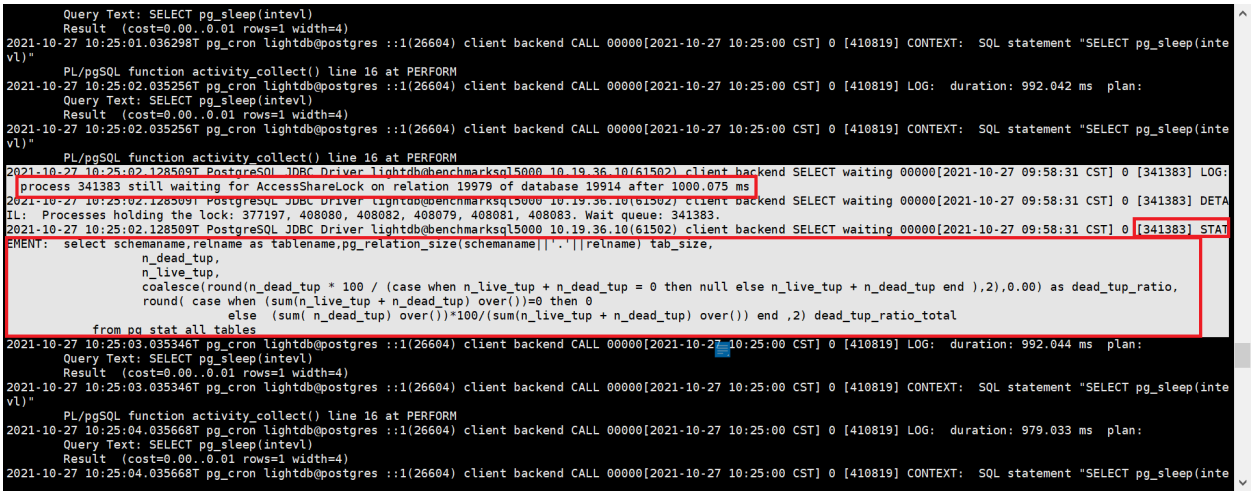
2.9 查看安装了哪些 extension

- 查看所有可用的 extension

```
select * from pg_available_extensions;
```

- 查看当前启用的 extension

```
select * from pg_extension;
```



2.10 查看按大小排序的前 20 张表

```sql
-- 查出按大小（table_size + index_size）排序的前20张表，并分离table_size和index_size
SELECT
    table_name,
    pg_size_pretty(table_size) AS table_size,
    pg_size_pretty(index_size) AS index_size,
    pg_size_pretty(total_size) AS total_size
FROM (
    SELECT
        table_name,
        pg_table_size(table_name) AS table_size,
        pg_indexes_size(table_name) AS index_size,
        pg_total_relation_size(table_name) AS total_size
    FROM (
        SELECT table_schema || '.' || table_name AS table_name
        FROM information_schema.tables
    ) AS all_tables
    ORDER BY total_size DESC
) AS pretty_sizes
LIMIT 20;
```

2.11 查看 LightDB 当前的整体负载

查看 LightDB 当前整体负载，可以简单地使用 top 命令查看 CPU 利用率、内存使用情况、IO 等指标信息，也可以使用 LightDB EM 来实时监控 LightDB 与服务器主机的各项指标。

2.12 查看 LightDB 的生效配置，修改会话配置、全局配置

可以用 show 语句查看 LightDB 当前的生效配置，show 语句有以下几种用法：

```
SHOW name;    -- 查看指定的para配置参数
SHOW ALL;     -- 查看所有配置参数
SHOW name%;   -- 查看前缀为name的配置参数
SHOW %name;   -- 查看后缀为name的配置参数
SHOW %name%; -- 查看名字中间包含name的配置参数
```

修改配置参数有两种级别：会话级和全局级。

```
-- 会话级修改，并非所有参数都支持会话级修改
SET [ SESSION | LOCAL ] configuration_parameter { TO | = } { value | 'value' |␣
↪DEFAULT };

-- 全局修改有两种方法：
-- 一是修改lightdb.conf，
-- 二是使用下面的SQL语句，然后按要求reload或restart生效
ALTER SYSTEM SET configuration_parameter { TO | = } { value | 'value' | DEFAULT };
```

2.13 什么是 vacuum？为什么要执行 vacuum？怎么确定 vacuum 是否成功？

vacuum 用于清理数据库表中的 dead tuples，因为 LightDB MVCC 不使用 undo 日志，而是将 update、delete 修改或删除前的记录保留在表中，并打上一个标记，对于 update 还会插入一条更新后的新纪录，带有这种标记的 tuple 叫做 dead tuple，也就是死元组。

当执行过 checkpoint 之后，之前的死元组就没有用了，vacuum 就是用来清除这些无用的死元组的，如果长时间不进行 vacuum，表中的死元组就会堆积的越来越多，导致表膨胀。

vacuum 语句基本用法有两种，一种是直接执行 vacuum，另一种是 vcuum tablename，前者对当前 database 中的所有表进行清理，后者仅对指定的表进行清理，执行成功时，客户端会返回一行 VACUUM 信息。

2.14 查看最近的检查点执行时间

```
lt_controldata $LTDATA | grep "Time of latest checkpoint:"
```

2.15 怎么查看 checkpoint 执行频率？怎么查看 auto vacuum 频率？

```
show checkpoint_timeout;   -- 查看checkpoint频率
show autovacuum_naptime;   -- 查看autovacuum频率
```

2.16 lt_wal 目录过大，怎么确定是否可以删除？如何删除?

先使用 lt_controldata 获得 Latest checkpoint's REDO WAL file，如下所示。

```
lt_controldata $LTDATA | grep "Latest checkpoint's REDO WAL file:"
```

```
[lightdb@localhost ~]$ lt_controldata $LTDATA | grep "Latest checkpoint's REDO WAL file:"
Latest checkpoint's REDO WAL file:    000000010000000200000007
[lightdb@localhost ~]$
```

Latest checkpoint's REDO WAL file 之前的 WAL 文件（包括已归档和未归档）都可以删除.

```
lt_archivecleanup -d $LTDATA/lt_wal last_checkpoint_redo_wal_file    #␣
↪删除未归档的WAL文件
lt_archivecleanup -d $LTHOME/archive last_checkpoint_redo_wal_file   #␣
↪删除已归档的WAL文件
```

```
[lightdb@localhost ~]$ lt_archivecleanup -d $LTDATA/lt_wal 000000010000000200000007
lt_archivecleanup: keeping WAL file "/home/lightdb/data/lt_wal/000000010000000200000007" and later
[lightdb@localhost ~]$
```

2.17 查看 LightDB 启动时间

```
select * from pg_postmaster_start_time();
```

2.18 查看当前事务号

```
select * from pg_current_xact_id();
```

2.19 查看 LightDB 实例概要信息

https://www.hs.net/lightdb/docs/html/functions-info.html

```
pg_control_checkpoint(), pg_control_init(), pg_control_system(), pg_control_recovery()
```

2.20 复制管理功能

https://www.hs.net/lightdb/docs/html/functions-admin.html#FUNCTIONS-ADMIN-BACKUP

2.21 其他管理功能函数

https://www.hs.net/lightdb/docs/html/functions-admin.html

2.22 高可用归档清理

高可用归档清理通过配置 `lightdb_archive_dir`（归档目录）和 `lightdb_archive_retention_size`（归档目录中 Latest checkpoint’s REDO WAL file 之前的文件保留数，建议配置为 10 以上，具体根据磁盘空间和主备间延迟配置，尽可能大）使用。

如：Latest checkpoint’s REDO WAL file 为 000000010000000100000049，lightdb_archive_retention_size 配为 10，则清理小于 000000010000000100000039 的 wal 文件。

2.23 日志清理

日志清理通过配置 `lightdb_log_retention_age` 来清理，单位为分钟（可配置为 3d，内部会转为分钟）。

如：配置 `lightdb_log_retention_age=7d`，则只保留 7 天的日志，在切换新文件时清理旧文件，根据文件的最新更新时间来清理。

2.24 WAL 文件缺失或被误删

如果不小心删除了 wal 文件，可通过 `lt_resetwal -f $LTDATA` 重新初始化 wal 文件，但是会丢失事务日志以及数据不一致，因为可能有 full checkpoint 之前的数据丢失，极端情况下某些数据块丢失。

具体丢多少数据，可以通过 lt_controldata 输出中的 latest checkpoint：

```
[lightdb@hs-10-20-30-199 bin]$ lt_controldata | grep -i checkpoint
Latest checkpoint location:           D4/78EFF8E8
Latest checkpoint's REDO location:    D4/78DE1390
Latest checkpoint's REDO WAL file:    000000010000000D400000003
Latest checkpoint's TimeLineID:       1
Latest checkpoint's PrevTimeLineID:   1
Latest checkpoint's full_page_writes: on
Latest checkpoint's NextXID:          0:50116967
Latest checkpoint's NextOID:          57309
Latest checkpoint's NextMultiXactId:  783
Latest checkpoint's NextMultiOffset:  1565
Latest checkpoint's oldestXID:        482
Latest checkpoint's oldestXID's DB:   1
Latest checkpoint's oldestActiveXID:  50116967
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:            Mon 04 Jul 2022 08:35:03 PM CST
```

2.25 lt_cron 重装

插件 lt_cron 重装之后，需要手动创建以下定时任务：

```
SELECT cron.schedule('lt_show_plans', '*/5 * * * *', 'SELECT lt_catalog.pg_show_
↪plans()');
SELECT cron.schedule('collect_activity_history', '*/1 * * * * *', 'SELECT collect_
↪activity_history()');
SELECT cron.schedule('collect_activity_profile', '*/1 * * * *', 'SELECT collect_
↪activity_profile()');;
SELECT cron.schedule('clean_activity_profile', '0 0 * * *', 'SELECT clean_activity_
```

(续下页)

(接上页)

```
↪profile()');
SELECT cron.schedule('take_sample', '*/10 * * * *', 'SELECT lt_catalog.take_sample()
↪');
```

3 LightDB 高可用

3.1 查看 LightDB 是否高可用、集群信息、主从节点

如果是单机版，则没有 ltcluster 库，可使用命令 `ltsql ltcluster` 尝试连接 `ltcluster` 库来确认，预期提示数据库不存在。单机版也不会有 `$LTDATA/../etc/ltcluster/ltcluster.conf` 这个配置文件。

如果是高可用部署，使用主节点或从节点运行下面的命令查看集群节点信息：

```
ltcluster -f $LTDATA/../etc/ltcluster/ltcluster.conf cluster show
```

示例结果：

```
 ID | Name    | Role    | Status    | Upstream | Location | Priority | Timeline |␣
↪Connection string
----+---------+---------+-----------+----------+----------+----------+----------+-----
↪-------------------------------
 1  | node199 | primary | * running |          | default  | 100      | 1        | host=node199␣
↪port=5432 user=ltcluster dbname=ltcluster connect_timeout=2
 2  | node193 | standby |   running | node199  | default  | 100      | 1        | host=node199␣
↪port=5432 user=ltcluster dbname=ltcluster connect_timeout=2
```

也可以使用 LightDB-EM 查看是单机部署还是高可用部署。

3.2 判断集群健康状态

在主节点或从节点运行命令 `ltcluster -f $LTDATA/../etc/ltcluster/ltcluster.conf cluster show` 展示的信息中没有 `WARNING`； `Status` 和 `Upstream` 字段没有出现 ? 和 ! 符号。

```
ID | Name    | Role    | Status    | Upstream| Location | Priority | Timeline |␣
↪Connection string
----+---------+---------+-----------+---------+----------+----------+----------+------
↪--------------------------
 1  | node199 | primary | * running |         | default  | 100      | 1        |
 2  | node193 | standby |   running | node199 | default  | 100      | 1        |
```

在各个节点运行命令 `ltcluster -f $LTDATA/../etc/ltcluster/ltcluster.conf node check` 展示的各个检查项的均为 `OK` 。

示例结果：

```
Node "node193":
Server role: OK (node is standby)
Replication lag: OK (0 seconds)
WAL archiving: OK (0 pending archive ready files)
Upstream connection: OK (node "node193" (ID: 2) is attached to expected upstream node
↪"node199" (ID: 1))
Downstream servers: OK (this node has no downstream nodes)
```

(续下页)

(接上页)

```
Replication slots: OK (node has no physical replication slots)
Missing physical replication slots: OK (node has no missing physical replication␣
↪slots)
Configured data directory: OK (configured "data_directory" is "/data1/data5432")
```

3.3 查看集群事件

在排查集群问题，或监控集群事件时，除了查看 $LTDATA/../etc/ltcluster/ltcluster.log，ltcluster 在 events 表中记录了更清晰有效的信息。

可运行 `ltcluster -f $LTDATA/../etc/ltcluster/ltcluster.conf cluster events` 查看集群事件，最新的事件排在最上面，示例结果如下：

```
 Node ID | Name    | Event                | OK | Timestamp           | Details
---------+---------+----------------------+----+---------------------+---------------------
↪---------------------
 1       | node199 | child_node_reconnect | t  | 2021-11-22 21:06:58 | standby node
↪"node193" (ID: 2) has reconnected after 1303 seconds
 1       | node199 | child_node_reconnect | t  | 2021-11-22 21:06:58 | standby node
↪"node193" (ID: 2) has reconnected after 1303 seconds
 2       | node193 | standby_register     | t  | 2021-11-22 21:06:55 | standby␣
↪registration succeeded; upstream node ID is 1
 2       | node193 | standby_recovery     | t  | 2021-11-22 21:06:42 | reconnected to␣
↪local node "node193" (ID: 2), marking active
 2       | node193 | standby_clone        | t  | 2021-11-22 21:05:44 | cloned from host
↪"node199", port 5432; backup method: lt_basebackup; --force: N
```

上述命令实际读取了 ltcluster.events 这张表，所以也可以通过 SQL 直接查询：

```
$ ltsql ltcluster # 连接ltcluster库
ltsql (13.3-21.2)
Type "help" for help.

ltcluster=# select * from ltcluster.events ;
 node_id |           event            | successful |        event_timestamp        | ␣
↪                                          details
---------+----------------------------+------------+-------------------------------+--
↪-------------------------
       1 | cluster_created            | t          | 2021-11-21 22:17:20.421939+08 |
       1 | primary_register           | t          | 2021-11-21 22:17:20.423033+08 |
       2 | standby_clone              | t          | 2021-11-21 22:27:39.853675+08 |␣
↪cloned from host "node199", port 5432; backup method: lt_basebackup; --force: N
       2 | standby_register           | t          | 2021-11-21 22:31:49.270459+08 |␣
↪standby registration succeeded; upstream node ID is 1
       1 | child_node_reconnect       | t          | 2021-11-21 22:31:55.155461+08 |␣
↪standby node "node193" (ID: 2) has reconnected after 440552 seconds
       1 | child_node_disconnect      | t          | 2021-11-21 22:35:49.769979+08 |␣
↪standby node "node192" (ID: 2) has disconnected
```

3.4 查看主从同步模式与延时

可在主节点执行 `select * from pg_stat_replication` 得到各个节点的实时同步状态信息。

```
lightdb@postgres=# select * from pg_stat_replication;
-[ RECORD 1 ]----+------------------------------
pid              | 1135698
usesysid         | 21741
usename          | ltcluster
application_name | lightdbCluster1019691835468
client_addr      | 10.19.69.183
client_hostname  |
client_port      | 38320
backend_start    | 2022-04-25 20:26:56.4993+08
backend_xmin     |
state            | streaming
sent_lsn         | 23/2A3ED548
write_lsn        | 23/2A3ED548
flush_lsn        | 23/2A3ED548
replay_lsn       | 23/2A3ED548
write_lag        | 00:00:00.000144
flush_lag        | 00:00:00.000915
replay_lag       | 00:00:00.00093
sync_priority    | 1
sync_state       | sync
reply_time       | 2022-05-06 17:28:27.523848+08
```

如果是多备机的情况下，每个备机都有一条记录。通过 `write_lag`, `flush_lag`, `replay_lag` 可以查看当前主从同步延迟信息。

可以从表 ltcluster.monitoring_history 中获取各个时间段的延时：

```
ltcluster=# select * from ltcluster.monitoring_history order by last_monitor_time␣
↪limit 10 ;
 primary_node_id | standby_node_id |       last_monitor_time       |        last_
↪apply_time        | last_wal_primary_location | last_wal_standby_location |␣
↪replication_lag | apply_lag
-----------------+-----------------+-------------------------------+------------------
↪-------------+---------------------------+---------------------------+--------------
↪---+-----------
               1 |               2 | 2021-12-21 16:57:48.537956+08 | 2021-12-21␣
↪16:57:48.52187+08  | 0/60012308                | 0/60012308                |       ␣
↪       0 |         0
               1 |               2 | 2021-12-21 16:57:50.561467+08 | 2021-12-21␣
↪16:57:50.294248+08 | 0/6001C540                | 0/6001C540                |       ␣
↪       0 |         0
               1 |               2 | 2021-12-21 16:57:52.577251+08 | 2021-12-21␣
↪16:57:52.55301+08  | 0/6001F1B0                | 0/6001F1B0                |       ␣
↪       0 |         0
               1 |               2 | 2021-12-21 16:57:54.590478+08 | 2021-12-21␣
↪16:57:53.66048+08  | 0/60020878                | 0/60020878                |       ␣
↪       0 |         0
               1 |               2 | 2021-12-21 16:57:56.6056+08   | 2021-12-21␣
↪16:57:55.944149+08 | 0/60023598                | 0/60023598                |       ␣
↪       0 |         0
               1 |               2 | 2021-12-21 16:57:58.618428+08 | 2021-12-21␣
↪16:57:58.19143+08  | 0/600278E0                | 0/600278E0                |       ␣
↪       0 |         0
```

(续下页)

(接上页)

```
              1 |               2 | 2021-12-21 16:58:00.638982+08 | 2021-12-21␣
↪16:58:00.615274+08 | 0/600C3150                | 0/600C3150                |       ␣
↪       0 |          0
              1 |               2 | 2021-12-21 16:58:02.686736+08 | 2021-12-21␣
↪16:58:01.813462+08 | 0/6023B0A8                | 0/6023B0A8                |       ␣
↪       0 |          0
              1 |               2 | 2021-12-21 16:58:04.712443+08 | 2021-12-21␣
↪16:58:04.117613+08 | 0/6023FA10                | 0/6023FA10                |       ␣
↪       0 |          0
              1 |               2 | 2021-12-21 16:58:06.730236+08 | 2021-12-21␣
↪16:58:06.310637+08 | 0/60242C48                | 0/60242C48                |       ␣
↪       0 |          0
```

也可以从 LightDB-EM 监控页面查看延时。

3.5 集群复制级别

不同的业务场景对数据库主备一致性有不同的要求。一致性越高对性能影响越大。用户可通过配置 `synchronous_commit` 来达到不同级别的一致性。

```
# 同步模式，在主节点修改
synchronous_commit = 'on'
synchronous_standby_names = '*'

# 异步模式，在主节点修改
synchronous_commit = 'local'
synchronous_standby_names = ''

# 修改后，主节点调用reload生效
lt_ctl -D $LTDATA reload
```

下表概括了 `synchronous_commit` 不同设置对应不同的一致性级别:

synchrono us_commit 设置	本地提交持久化	备库提交持久化 (数据库崩溃)	备库提交持久化 (OS 崩溃)	备库查询一致
remote_apply	是	是	是	是
on	是	是	是	
remote_write	是	是		
local	是			
off				

更详细的 `synchronous_commit` 及 `synchronous_standby_names` 请参考 LightDB 官方文档。

3.6 主备切换

在需要维护 primary 节点时，可做 switchover，互换主从角色。switchover 操作的内部执行比较复杂，非必要尽量不要执行。

具体操作时，请严格按照下面步骤执行:

1. 主备之间需要有 SSH 免密访问 (LightDB 安装时有要求)
2. 尽量减少应用程序的访问
3. 检查主备间的网络状况是否良好，确保有良好的网络
4. 确保当前主备之间没有明显的复制延迟，尤其在集群复制级别较低的情况下 (参考查看主从同步模式与延时 ， 集群复制级别)
5. 检查等待归档的文件是否有积压, 可通过下面的命令来检查

   ```
   ltcluster -f $LTDATA/../etc/ltcluster/ltcluster.conf node check --archive-ready
   ```

   确保输出是：`OK (0 pending archive ready files)` 。

   如果是其他输出，则应检查归档进程是否正常。如果归档正常，则可以等待一会儿再试下。
6. 使用 dry-run 试运行 switchover 命令，查看输出是否有警告和错误

   ```
   ltcluster -f $LTDATA/../etc/ltcluster/ltcluster.conf standby switchover \
       --siblings-follow --dry-run
   ```

   如果最后一行信息为：prererequisites for executing STANDBY SWITCHOVER are met，则表示成功
7. **在备机上正式执行 switchover** (打开最详细的日志级别)

   ```
   ltcluster -f $LTDATA/../etc/ltcluster/ltcluster.conf standby switchover \
         --log-level=DEBUG --verbose --siblings-follow
   ```
8. 在各节点上查看集群状态，确认各节点执行结果中 primary 和 standby 角色确实已互换

   ```
   ltcluster -f $LTDATA/../etc/ltcluster/ltcluster.conf service status
   ```

   输出要确保没有警告和错误信息
9. 查看 paused 状态是否为 no

   确认 Paused 列为 no (如果 switchover 过程出现异常，经过处理后，switchover 成功, 此时在这一步可能处于 yes)

   如果为 yes，则执行

   ```
   ltcluster -f $LTDATA/../etc/ltcluster/ltcluster.conf service unpause
   ```
10. 如果使用同步模式，则需要把新主改成同步模式 (和旧主一样)，新备改成 local 模式 (参考集群复制级别)
11. 确认 VIP 是否切换到新的主机上 (参考如何查看 *VIP* 当前在哪个节点)
12. 确认应用程序是否可以正常访问数据库

3.7 故障恢复，主节点重新加入作为从节点

当主库发生故障（如宕机）failover 后，备库会自动提升为新主库，以确保集群继续可用。在原主库修复后，可以以备机的方式加入集群，使得整个集群仍然保持高可用正常状态。

在原主库故障修复后，数据库本身会仍然运行主模式，我们需要执行 rejoin 命令，rejoin 命令会把数据库改为备模式，并且从新主把最新的 WAL 日志同步过来，确保数据一致。

在本节后续描述中，**主库**指的是新主库 (即原备机提升后的主库), **备库**指的是原主库 (发生宕机的节点)。

以下是具体操作步骤:

1. 确认主库是正常运行状态，而备库是停止状态
2. 同步归档日志，把主库的归档日志同步到备库中，归档目录为: `$LTHOME/archive` 备库中的原有归档日志应备份到其他地方或者删除。
3. 在备库上执行以下命令，检查当前是否满足 rejoin 条件 (注意把命令中的 `<primary_host>` 替换为主库的 ip 地址)

```
ltcluster -f $LTDATA/../etc/ltcluster/ltcluster.conf node rejoin \
    -d 'host=<new_primary_host> port=new_primary_port dbname=ltcluster␣
↪user=ltcluster' \
    --verbose --force-rewind --dry-run
```

   确认输出有 `INFO: prerequisites for executing NODE JOIN are met` 并且无警告或者错误信息。

4. 正式执行 rejoin (注意把命令中的 `<primary_host>` 替换为主库的 ip 地址)

```
ltcluster -f $LTDATA/../etc/ltcluster/ltcluster.conf node rejoin \
    -d 'host=<new_primary_host> port=new_primary_port dbname=ltcluster␣
↪user=ltcluster' \
    --verbose --force-rewind
```

3.8 什么时候会 rejoin 失败、如何确定肯定无法 rejoin 了？无法 rejoin 的节点如何重新加入？

如果备机离线时间较久，必要的 WAL 日志在主上已经被移除，则 rejoin 会失败。此时需要使用 standby clone 操作来恢复集群，standby clone 的原理是从主上把整个库拷贝过来，在数据库较大的情况下耗时会比较久。

standby clone 的步骤如下：

1. 确认备库 LightDB 已停止
2. 清空备库归档目录 (`$LTHOME/archive`) 下的内容（若有需要，清空前可先备份）
3. clone 试运行，将 new_primary_host 替换为原备，也就是新主的 host

```
ltcluster -h new_primary_host -p new_primary_port -U ltcluster \
    -d ltcluster -f $LTDATA/../etc/ltcluster/ltcluster.conf standby clone \
    --dry-run
```

4. 确认试运行结果显示 `all prerequisites for "standby clone" are met`
5. clone 实例目录，new_primary_host 同上，如果库比较大，这里执行时间会很长，具体执行时间取决于网络情况和数据量大小在我们的测试中 800G 左右的库大概需要一个小时我们建议采用异步的方式执行这个命令，以避免执行过程中终端意外关闭的影响。另外我们开启了最详细的日志级别，以便协助定位问题

```
nohup ltcluster -h new_primary_host -p new_primary_port -U ltcluster \
   -d ltcluster -f $LTDATA/../etc/ltcluster/ltcluster.conf standby clone \
   -F --log-level=DEBUG --verbose            >standby_clone.log  2>&1 &
```

6. 把主库的归档目录下的所有文件复制到备库的归档目录中 (`$LTHOME/archive`)
7. 启动数据库

```
lt_ctl -D $LTDATA start
```

8. 重新注册为 standby

```
ltcluster -f $LTDATA/../etc/ltcluster/ltcluster.conf standby register -F
```

9. 确认 ltclusterd 是否启动，若不存在则启动它

```
ps aux | grep ltclusterd

ltclusterd -d -f `realpath $LTDATA/../etc/ltcluster/ltcluster.conf` \
   -p $LTDATA/../etc/ltcluster/ltclusterd.pid
```

10. 查看集群状态，确认集群运行正常

```
ltcluster -f $LTDATA/../etc/ltcluster/ltcluster.conf service status
```

输出要确保没有警告和错误信息

3.9 什么是 timeline，timeline 什么时候变化？如何查看当前的 timeline id？

timeline 可以认为是数据库 wal 的分支（类比版本管理系统，比如 svn）。

当进行一次恢复，或发生主备切换，会生成一个 timeline。每个 timeline 有一个 id，从 1 开始编号。当生成一个新的 timeline 时，它的 wal 是独立的，不会覆盖其它 timeline 的 wal，这就保证了可以多次来回恢复。如果没有 timeline，即恢复后 wal 覆盖写，则只能一直往“以前”恢复。

可以查看 lt_wal 中的 history 文件，来确定当前有几个 timeline、各自创建时的 LSN、创建的原因，如

```
$ cat $LTDATA/lt_wal/00000004.history
1   16/F20000A0 no recovery target specified

2   16/F50000A0 no recovery target specified

3   16/F60000A0 no recovery target specified
```

序号最大的 history 文件即是当前 timeline id。

可以通过 sql 查看当前 timeline id: `ltsql "dbname=postgres replication=database" -c "IDENTIFY_SYSTEM";` 或在主库执行 `select substring(pg_walfile_name(pg_current_wal_lsn()), 1, 8);`

高可用命令 `ltcluster -f $LTDATA/../etc/ltcluster/ltcluster.conf cluster show` 获取的 timeline 是当前最近做 checkpoint 的 timeline，可能不是最新的 timeline。

3.10 当出现双主时如何处理

如果出现双主，把老主停掉，重新加入集群作为 standby。参考 `node rejoin` 章节，如果 rejoin 失败，老主通过 `standby clone` 重新加入集群。

3.11 如何查看 VIP 当前在哪个节点

使用命令 `ip a` 可看到 vip 是否在当前节点，比如

```
$ ip a | grep 251
    inet 10.19.36.251/32 scope global enp2s0f0
```

如果 grep 没有匹配行，则 vip 不在当前节点。

可以在 keepalived.conf 中查看 vip 配置，比如

```
$ cat $LTHOME/etc/keepalived/keepalived.conf
...
    interface enp2s0f0
...
    virtual_ipaddress {
        10.19.36.251
    }
...
```

3.12 如何触发 VIP 漂移

在以下场景会触发 VIP 漂移:

- 主库崩溃、意外停止，导致自动主从切换 (failover)
- 手动进行主从切换 (switchover)

3.13 为什么会出现 VIP 同时在两个节点？

如果主从之间网络出现问题，从节点可能误判主节点故障，把自己提升，这时会出现两个 VIP。

建议集群中加入 witness 节点，避免网络问题引起主从切换或从节点自动切主。

3.14 重启主库

主库因修改数据库参数或其他原因需要重启，可以按以下步骤操作。**(注意: 重启期间数据库不提供服务)**

1. **先停止从库的 "keepalived" (重要)**，在 `root` 用户下执行以下命令

```
# 1. 获得备库keepalived进程pid
cat /var/run/keepalived.pid

# 2. 杀死keepalived进程
kill keepalived_pid

# 3. 确认keepalived进程确实已不存在
ps aux | grep keepalived
```

2. 主库重启，需要在 lightdb 用户下执行

```
# 1. 暂停ltclusterd，防止自动failover
ltcluster -f $LTDATA/../etc/ltcluster/ltcluster.conf service pause

# 2. 查看集群状态，确认primary的Paused?状态为yes
ltcluster -f $LTDATA/../etc/ltcluster/ltcluster.conf service status

# 3. 先断开所有连接到数据库的客户端和应用程序（否则数据库将stop failed），然后停止主库
lt_ctl -D $LTDATA stop  # 默认会回滚所有未断开的连接

# 如果有连接存在导致stop failed，则可以尝试使用
lt_ctl -D $LTDATA stop -m smart

# 如果仍然stop failed，且因条件限制无法或不希望断开所有客户端连接，则可以使用-m␣
↪immediate强制停止数据库，
↪此方式下没有回滚连接，即强制断开、强制停止，没有完全shutdown，会导致在启动时recovery
lt_ctl -D $LTDATA stop -m immediate

# 4. 等待数据库停止成功，确认步骤3执行结果中出现server stopped信息

# 5. 修改数据库参数，或做其他事情

# 6. 启动主库
lt_ctl -D $LTDATA start

# 7. 等待数据库启动成功，确认步骤6执行结果中出现server started的信息

# 8. 恢复ltclusterd
ltcluster -f $LTDATA/../etc/ltcluster/ltcluster.conf service unpause

# 9. 查看集群状态，确认primary的Paused?状态为no
ltcluster -f $LTDATA/../etc/ltcluster/ltcluster.conf service status
```

3. **从库重新启动 “keepalived“ （需 root 用户）**。

3.15 重启从库

备库因修改数据库参数或其他原因需要重启，可以在 lightdb 用户下按以下步骤操作。

```
# 1. 暂停ltclusterd,防止自动failover
ltcluster -f $LTDATA/../etc/ltcluster/ltcluster.conf service pause

# 2. 查看集群状态，确认standby的Paused?字段为yes
ltcluster -f $LTDATA/../etc/ltcluster/ltcluster.conf service status

# 3. 先断开所有连接到数据库的客户端和应用程序（否则数据库将stop failed），然后停止备库
lt_ctl -D $LTDATA stop  # 默认会回滚所有未断开的连接

# 如果有连接存在导致stop failed，则可以尝试使用
lt_ctl -D $LTDATA stop -m smart

# 如果仍然stop failed，且因条件限制无法或不希望断开所有客户端连接，则可以使用-m␣
↪immediate强制停止数据库，
↪此方式下没有回滚连接，即强制断开、强制停止，没有完全shutdown，会导致在启动时recovery
lt_ctl -D $LTDATA stop -m immediate
```

(续下页)

(接上页)

```
# 4. 等待数据库停止成功，确认步骤3执行结果中出现server stopped信息

# 5. 修改数据库参数，或做其他事情

# 6. 启动备库
lt_ctl -D $LTDATA start

# 7. 等待数据库启动成功，确认步骤6执行结果中出现server started的信息

# 8. 恢复ltclusterd
ltcluster -f $LTDATA/../etc/ltcluster/ltcluster.conf service unpause

# 9. 确认standby的Paused?字段为no
ltcluster -f $LTDATA/../etc/ltcluster/ltcluster.conf service status
```

3.16 高可用归档清理与 lt_probackup 备份归档清理

当同时使用高可用归档与 lt_probackup 备份归档时建议建立两个归档目录，归档两份，分别给高可用和备份使用，不然如果使用同一个，然后只开启备份的清理，有可能出现误删高可用所需的 wal 文件；只开启高可用的归档清理，可能导致误删备份所需的 wal 文件。

高可用归档清理参见章节高可用归档清理 。

3.17 集群启停管理脚本

可以通过使用 `lightdb_service.py` 来进行集群的启停及简单的状态检测。脚本在 `$LTHOME/bin/` 下，依赖于 uninstall 目录下的 uninstallFile.json 来获取集群信息。此脚本在 lightdb 用户下执行，lightdb 需要支持 sudo 免密。

python 版本需为 python3。使用方式如下：

```
usage: lightdb_service.py [-h] [-F <install_info>]
                          [-c {start,stop,restart,status}] [-C | -D] [-P | -S]
                          [-n <node_info>] [--dry-run] [-f]
                          [-B <parallel_processes>] [-v] [-q] [-l <directory>]

use for start/stop/restart LightDB service

optional arguments:
  -h, --help            show this help message and exit
  -F <install_info>, --filename <install_info>
                        specifies the path to a json file containing
                        installation information(uninstall/uninstallFile.json)
  -c {start,stop,restart,status}, --command {start,stop,restart,status}
                        start/stop/restart/status lightdb service
  -C, --cn_only         only start/stop/restart coordinator node
  -D, --dn_only         only start/stop/restart data node
  -P, --primary_only    only start/stop/restart primary node for primary
                        restart
  -S, --standby_only    only start/stop/restart standby node for standby
                        restart
  -n <node_info>, --node <node_info>
                        only start/stop/restart specified node, node_info
                        formart: ip:port
```

(续下页)

(接上页)

```
  --dry-run             show what would happen for action, but don't execute
                        it
  -f, --force           force stop of cluster even if some nodes are in
                        incorrect state, skip incorrect nodes
  -B <parallel_processes>, --parallel <parallel_processes>
                        number of segment hosts to run in parallel. Default is
                        1
  -v, --verbose         debug print
  -q, --quiet           suppress status messages for stdout logging
  -l <directory>, --log_dir <directory>
                        Logfile directory, default is /tmp/ltAdminLogs
```

1. 启动集群

```
python3 lightdb_service.py -c start
```

2. 停止集群

```
python3 lightdb_service.py -c stop
```

3. 重启集群

```
python3 lightdb_service.py -c restart
```

4. 查看集群状态

```
python3 lightdb_service.py -c status
```

5. 只启停主

   启停时会保证主备状态不变

```
python3 lightdb_service.py -c start/stop/restart --primary_only
```

6. 只启停备

```
python3 lightdb_service.py -c start/stop/restart --standby_only
```

7. 只启停 CN 节点

```
python3 lightdb_service.py -c start/stop/restart --cn_only
```

8. 只启停 DN 节点

```
python3 lightdb_service.py -c start/stop/restart --dn_only
```

9. 只启停某个节点

```
python3 lightdb_service.py -c start/stop/restart -n 10.20.148.122:54333
```

10. 强制停止

    当集群状态不对时，stop 会失败，此时如果仍需停止集群可以使用 `-f --force`

```
python3 lightdb_service.py -c stop -f
```

11. 试运行

```
python3 lightdb_service.py -c xxx --dry-run
```

4 LightDB 分布式

4.1 查看分布式节点信息

```
select * from pg_dist_node;
```

4.2 设置分布式 CN 节点

```
select canopy_set_coordinator_host('CN_NODE_IP', CN_NODE_PORT);
```

4.3 添加分布式 DN 节点

```
select master_add_node('DN_NODE_IP', DN_NODE_PORT);
```