版本	1.0

瀚高软件

瀚高企业版数据库系统V7
安装手册(x86_64-centos7平台)

2021.12

瀚高基础软件股份有限公司

服务热线：400-708-8006

www.highgo.com

版本历史

版本	修改描述	修改日期	备注
1.0	手册编写完毕	2021.12	

目录

1 前言 1
2 安装前检查 1
2.1 推荐配置 1
2.2 内存和参数检查 1
3 安装前准备 2
3.1 关闭防火墙 2
3.2 关闭 NetworkManager 2
3.3 关闭 SELINUX 2
3.4 检查主机名 3
3.5 检查时间和时区 3
3.6 安装系统软件包 3
3.7 配置 hosts 4
3.8 创建 highgo 用户并修改密码 4
3.9 配置操作系统内核参数 4
3.10 创建数据库安装目录 5
3.11 配置操作系统 limits 5
4 安装及配置 5
4.1 RPM 包安装 5
4.1.1 安装数据库单机 5
4.1.2 安装数据库集群 7
4.2 License 说明 8
5 安装后操作 8
5.1 测试 8
5.2 修改数据库参数 8
5.3 修改网络访问控制 9
6 高可用 DB_HA 9
6.1 db_ha 简介 9
6.2 部署高可用 10
6.2.1 数据库相关配置 10
6.2.2 agent 11
6.2.3 ha_isready 14
6.2.4 primary_monitor 14
6.2.5 secondary_monitor 17
6.2.6 db_ha 18
6.2.7 ha_ctl 19
6.3 db_ha 的使用 19
6.3.1 查询集群信息 20
6.3.2 增加节点 21
6.3.3 踢除节点 21
6.3.4 主备切换 22
6.3.5 停止监控 22
6.3.6 重启数据库 23
6.3.7 更改数据库同异步模式 23
6.3.8 VIP 操作 24
6.3.9 重载配置文件 25

6.4 ha_ctl 使用 26
6.4.1 ha_ctl 对 agent 的操作 26
6.4.2 ha_ctl 对 primary_monitor 的操作 27
6.4.3 ha_ctl 对 secondary_monitor 的操作 27
6.4.4 ha_ctl pg_basebackup 的操作 27
6.4.5 ha_ctl pg_rewind 的操作 28
6.5 手动安装示例 28
6.5.1 在 192.168.31.21 上进行安装部署 29
6.5.2 在 192.168.31.22 上进行安装部署 32
6.5.3 在 192.168.31.20 上进行安装部署 35
6.5.4 启动 38
6.5.5 验证安装部署 38
6.6 使用一键部署安装示例 39
6.6.1 在 192.168.31.21（主）上进行安装部署 40
6.6.2 在 192.168.31.22（备）上进行安装部署 40
6.6.3 在 192.168.31.20（monitor）上进行安装部署 41
6.6.4 一键安装部署 41
6.6.5 一键回滚 42
6.7 在现有流复制集群环境搭建高可用 42
6.8 卸载高可用产品 45
6.9 已知限制条件 44
7 备份恢复 DB_BACKUP 52
7.1 db_backup 简介 52
7.2 安装部署 52
7.2.1 安装 RPM 包 52
7.2.2 创建备份目录和归档目录 52
7.2.3 开启归档 53
7.2.4 配置用户环境变量(默认$PGDATA 的路径是/usr/pgsql-12) 53
7.2.5 初始化备份目录 53
7.3 主要功能使用说明 54
7.3.1 全量备份 54
7.3.2 增量备份(默认 0 级增量备份) 54
7.3.3 备份查看 55
7.3.4 备份验证 55
7.3.5 备份恢复（先关闭数据库） 56
7.3.6 备份删除 57
7.3.7 备份物理删除 58
7.4 关于备份恢复的一些其他参数 58
7.5 db_backup 已知限制条件 59
8 双向数据复制 DBREP 59
8.1 dbrep 简介 59
8.2 准备工作 59
8.3 配置相关参数 60
8.4 双向复制部署 60
8.4.1 发布相应的表 60
8.4.2 订阅相应的表 61
8.4.3 更新发布的表 62
8.4.4 关闭双向数据复制 62

9 集群功能 63
10 分布式 OLTP 解决方案 70
11 定时任务 DB_CRON 83
11.1 db_cron 简介 83
11.2 安装 db_cron 83
11.3 配置 postgresql.conf 83
11.4 使用 db_cron 83
11.5 取消定时任务 84
12 批量导入 DB_BULKLOAD 84
12.1 db_bulkload 简介 84
12.2 安装 db_bulkload 85
12.3 使用 db_bulkload 85
12.4 使用命令行参数进行数据加载 85
12.5 使用控制文件进行数据加载 86
13 孤儿事务清理工具 88
13.1 孤儿事务清理工具简介 88
13.2 使用孤儿事务清理工具 88
14 程序卸载 90
14.1 RPM 包卸载 90

1 前言

本文档主要介绍瀚高企业版数据库系统 V7 在 x86_64-rhel7 平台的安装过程及注意事项。本文主要演示数据库安装步骤。

使用普通用户以 HGDB V7.0.1 为例进行说明。操作系统不同，对应的配置命令也会有所不同。本文主要介绍 RedHat 7.3 中的配置过程。红帽系列其他的操作系统配置步骤大致相同，其他系列的操作系统差异较大，请根据具体的操作系统灵活调整配置命令，或参考瀚高企业版数据库系统 V7 最佳实践相关手册。

本文档只针对安装过程进行说明，数据库完整的功能请参照管理手册、开发手册或其他相关手册内容。

2 安装前检查

2.1 推荐配置

配置参数	最低配置	推荐配置
CPU	4 核	16 核
内存	4GB	64GB
存储	800MB，机械硬盘	5GB 以上，SSD 或 NvMe
网络	千兆网络	万兆网络

2.2 内存和参数检查

以 root 用户身份登录并运行以下命令。

1. 要查看可用 RAM 和交换空间大小，运行以下命令：

```
df -h
free -h
grep MemTotal /proc/meminfo
grep SwapTotal /proc/meminfo
# grep MemTotal /proc/meminfo
MemTotal:512236 kB
# grep SwapTotal /proc/meminfo
SwapTotal:1574360 kB
```

表 1 内存及对应 swap 建议值参照表

MemTotal	SwapTotal
8G	2~4G
8~16G	4~8G
16~64G	8~32G

>=64G	32G

2. 检查内核参数

```
cat /proc/sys/kernel/shmmax
cat /proc/sys/kernel/shmall
cat /proc/sys/kernel/shmmni
#该参数（系统共享内存段的最大数量）数据库自动修改
```

表 2 内存大小及对应内核参数建议值参照表

MemTotal	shmall	shmax
8G	7~8G	4G
8~64G	M*50%	M*90~95%
>=64G	32G	M*90%~95%

3 安装前准备

3.1 关闭防火墙

```
systemctl stop firewalld.service
systemctl disable firewalld.service
systemctl status firewalld.service
systemctl stop  NetworkManager.service
systemctl disable NetworkManager.service
systemctl status  NetworkManager.service
```

3.2 关闭 NetworkManager

```
[root@hgdb ~]# systemctl stop  NetworkManager.service
[root@hgdb ~]# systemctl disable NetworkManager.service
Removed symlink /etc/systemd/system/multi-user.target.wants/NetworkManager.service.
Removed symlink /etc/systemd/system/dbus-org.freedesktop.NetworkManager.service.
Removed symlink /etc/systemd/system/dbus-org.freedesktop.nm-dispatcher.service.
```

3.3 关闭 SELINUX

```
[root@hgdb ~]# sed -i "s/SELINUX=enforcing/SELINUX=disabled/g" /etc/selinux/config
[root@hgdb ~]# setenforce 0
[root@hgdb ~]# cat /etc/selinux/config | grep SELINUX=disabled
[root@hgdb ~]# getenforce
```

3.4 检查主机名

```
[root@hgdb ~]# hostnamectl status
   Static hostname: hgdb
         Icon name: computer-vm
           Chassis: vm
        Machine ID: 5c7b0775b96044209a3858d9773a5b83
           Boot ID: a379f124abc746c190aca4932ee34fe1
    Virtualization: vmware
  Operating System: Red Hat Enterprise Linux Server 7.3 (Maipo)
       CPE OS Name: cpe:/o:redhat:enterprise_linux:7.3:GA:server
            Kernel: Linux 3.10.0-514.el7.x86_64
      Architecture: x86-64
```

3.5 检查时间和时区

```
[root@hgdb ~]$ timedatectl
      Local time: Fri 2018-10-19 18:52:17 CST
  Universal time: Fri 2018-10-19 10:52:17 UTC
        RTC time: Fri 2018-10-19 10:52:18
       Time zone: Asia/Shanghai (CST, +0800)
     NTP enabled: no
NTP synchronized: no
 RTC in local TZ: no
      DST active: n/a
```

修改时区：

```
[root@hgdb ~]# timedatectl list-timezones
[root@hgdb ~]# timedatectl set-timezone Asia/Shanghai
修改时间
[root@hgdb ~]# date -s "20170622 10:26:00"
```

3.6 安装系统软件包

配置 yum 源：

```
[root@hgdb ~]# vi /etc/yum.repos.d/highgo.repo
[rhel]
name=rhel
baseurl=file:///media/cdrom
enabled=1
gpgcheck=0
[root@hgdb ~]# mkdir /media/cdrom
[root@hgdb ~]# mount /dev/sr0 /media/cdrom
mount: /dev/sr0 is write-protected, mounting read-only
```

安装依赖包：

```
[root@hgdb ~]# yum clean all
```

```
[root@hgdb ~]# yum list
[root@hgdb ~]# yum install vim wget readline readline-devel zlib zlib-devel openssl openssl-devel pam-devel libxml2-devel libxslt-devel python-devel tcl-devel gcc gcc-c++ rsync -y
```

3.7 配置 hosts

```
[root@hgdb ~]# vi /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6

192.168.80.10 hgdb
```

3.8 创建 highgo 用户并修改密码

```
[root@hgdb ~]# groupadd -g 5866 highgo
[root@hgdb ~]# useradd -u 5866 -g highgo highgo
[root@hgdb ~]# passwd highgo
```

使用如下命令设置密码：

```
[root@ hgdb ~]# passwd highgo
Changing password for user highgo.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

3.9 配置操作系统内核参数

以下参数值仅用于示例，测试或生产环境请根据系统资源情况对相关参数进行调整。

```
[root@hgdb ~]# vi /etc/sysctl.conf
#for highgo db 6.0.0
net.core.wmem_default=262144
fs.file-max=76724600
vm.mmap_min_addr=65536
net.core.somaxconn=4096
net.core.wmem_max=4194304
net.core.netdev_max_backlog=10000
kernel.sem=4096 2147483647 2147483646 512000
net.core.rmem_max=4194304
vm.overcommit_ratio=90
net.ipv4.tcp_tw_reuse=1
net.core.rmem_default=262144
net.ipv4.ip_local_port_range=40000 65535
net.netfilter.nf_conntrack_max=1200000
net.ipv4.tcp_rmem=8192 87380 16777216
net.ipv4.tcp_max_syn_backlog=4096
fs.nr_open=20480000
net.ipv4.tcp_wmem=8192 87380 16777216
```

```
vm.dirty_writeback_centisecs=100
vm.dirty_ratio=95
kernel.shmmni=819200
vm.swappiness=0
net.ipv4.tcp_mem=8388608 12582912 16777216
vm.dirty_background_bytes=409600000
net.nf_conntrack_max=1200000
net.ipv4.tcp_max_tw_buckets=262144
fs.aio-max-nr = 1048576
```

root 用户运行如下命令使参数立即生效：

```
[root@hgdb ~]# sysctl -p
```

3.10 创建数据库安装目录

数据库安装目录建议使用单独的磁盘或者 lv 卷组。

```
[root@hgdb ~]# mkdir -p /data/highgo/6.0.0
[root@hgdb ~]# chown -R highgo:highgo /data
```

3.11 配置操作系统 limits

以下参数值仅用于示例，测试或生产环境请根据系统资源情况对相关参数进行调整。

```
[root@hgdb ~]# vi /etc/security/limits.conf
#for highgo db 6.0.0
highgo soft  core unlimited
highgo hard  nproc unlimited
highgo soft  nproc unlimited
highgo hard  memlock unlimited
highgo hard  nofile 1024000
highgo soft  memlock unlimited
highgo soft  nofile 1024000
```

4 安装及配置

4.1 RPM 包安装

rpm 包安装步骤以企业版 v7 为例，安装 rpm 包会默认创建 highgo 用户，该用户默认没有密码。安装文件的属主均为 highgo 用户。安装完毕后，需使用 highgo 用户执行 initdb 及后续操作。

4.1.1 安装数据库单机

(1) 使用 root 用户安装 rpm 包：

```
[root@x86-0001 opt]# rpm -ivh hgdb7.0.1-enterprise-centos7-x86-64-20211230.rpm
Preparing...                          ################################# [100%]
```

```
Updating / installing...
   1:hgdb-7.0.1-1.el7              ################################## [100%]

#安装完成后，会在/opt 目录下生成安装目录：
[root@x86-0001 hgdb]# pwd
/opt/hgdb
[root@x86-0001 hgdb]# ls
bin  data  include  lib  share
```

（2） 安装完毕后会自动配置 highgo 用户环境变量，需要用户 source 使之生效：

```
#切换 highgo 用户
[root@x86-0001 hgdb]$ su - highgo

[highgo@x86-0001 hgdb]$ cat ~/.bash_profile

PATH=$PATH:$HOME/.local/bin:$HOME/bin

export PATH
export PATH=$PATH:/opt/hgdb/bin
export LD_LIBRARY_PATH=/opt/hgdb/lib:$LD_LIBRARY_PATH
export PGDATA=/opt/hgdb/data

#source 使之生效：
[highgo@x86-0001 hgdb]$ source ~/.bash_profile

#检查环境变量是否生效
[highgo@x86-0001 hgdb]$ echo $PGDATA
/opt/hgdb/data
```

（3）手动初始化数据库

```
[highgo@x86-0001 bin]$ initdb -D ../data
The files belonging to this database system will be owned by user "highgo".
This user must also own the server process.

The database cluster will be initialized with locale "en_US.UTF-8".
The default database encoding has accordingly been set to "UTF8".
The default text search configuration will be set to "english".

Data page checksums are disabled.

fixing permissions on existing directory ../data ... ok
creating subdirectories ... ok
selecting dynamic shared memory implementation ... posix
selecting default max_connections ... 100
```

```
selecting default shared_buffers ... 128MB
selecting default time zone ... Asia/Shanghai
creating configuration files ... ok
running bootstrap script ... ok
performing post-bootstrap initialization ... ok
syncing data to disk ... ok

initdb: warning: enabling "trust" authentication for local connections
You can change this by editing pg_hba.conf or using the option -A, or
--auth-local and --auth-host, the next time you run initdb.

Success. You can now start the database server using:

    pg_ctl -D ../data -l logfile start
```

（4）启动数据库

```
#启动数据库
[highgo@x86-0001 bin]$ ./pg_ctl start
...
server started

#登录数据库
[highgo@x86-0001 bin]$ psql -U highgo -d highgo
psql (HighGo Database 7 Release 7.0.1-64-bit Production)
Type "help" for help.

highgo=#
```

4.1.2 安装数据库集群

这里以搭建一主一备集群为例：

1. 所有节点安装数据库

详细步骤参见4.1.1 安装数据库单机 章节。

注意： 主库需要 initdb，备库不需要执行这一步。

2. 主节点配置 postgresql.conf 文件

```
listen_addresses = '*'
max_connections = 100
wal_level = replica
max_wal_senders = 5
hot_standby = on
wal_log_hints=on
full_page_writes=on
```

3. 主节点配置 pg_hba.conf 文件

```
host    all             all             0.0.0.0/0               trust
host    replication     all                     0.0.0/0         trust
```

4. 主节点重启数据库

```
pg_ctl  restart
```

5. 在备节点做基础备份，搭建主备流复制

```
pg_basebackup -F p -P -X fetch -R -h 192.168.31.101 -p 5432 -U highgo -D /opt/hgdb/data
//修改 data 目录权限
chmod 0700 /opt/hgdb/data
```

6. 启动备节点数据库

```
pg_ctl  start
```

4.2 License 说明

瀚高企业版数据库系统V7及之前的版本自带365天的试用license，过期后数据库将无法使用。如需购买正式license，请拨打400-708-8006联系瀚高工作人员。

5 安装后操作

5.1 测试

```
[highgo@hgdb ~]$ psql

highgo=# \dt
No relations found.
highgo=# create table tb1 (id int);
CREATE TABLE
highgo=# \dt
        List of relations
 Schema | Name | Type  | Owner
--------+------+-------+--------
 public | tb1  | table | highgo
(1 row)

highgo=#
```

5.2 修改数据库参数

```
alter system set max_connections = 2000;        #连接数
alter system set shared_buffers = '512MB';      #total_memory*25%
```

```
alter system set checkpoint_completion_target = 0.8;
#日志参数
alter system set log_destination = 'csvlog';
alter system set logging_collector = on;
alter system set log_directory = 'hgdb_log';
alter system set log_filename = 'highgodb-%a.log';
alter system set log_rotation_age = '1d'; #每天生成一个新的日志文件
alter system set log_rotation_size = 0;    #不限制单个日志文件大小
alter system set log_truncate_on_rotation = on; #覆盖同名文件
#只保留 7 天日志，循环覆盖
alter system set log_hostname = on;
alter system set log_line_prefix = '%m';
alter system set log_statement = 'ddl';
```

参数修改完成后重新启动数据库：

```
[highgo@hgdb ~]$ pg_ctl -m fast stop
[highgo@hgdb ~]$ pg_ctl start
```

5.3 修改网络访问控制

默认外部主机不能访问该数据库，编辑$PGDATA/pg_hba.conf 文件,在位置
IPv4 local connections:
下面添加如下一行，可允许所有外部主机访问数据库。

```
# "local" is for Unix domain socket connections only
local   all             all                                     trust
# IPv4 local connections:
host    all             all             127.0.0.1/32            trust
host    all             all             0.0.0.0/0               trust
```

说明：
在 pg_hba.conf 中的条目从上到下有效性递减。如：第一条同意某主机访问，第二条禁止某主机访问，则该主机可以访问。

同时需要修改数据库参数，编辑$PGDATA/postgresql.conf 文件中 listen_addresses='*'参数或直接在数据库中执行修改命令：

```
alter system set listen_addresses = '*'
```

执行命令重新加载，使配置文件生效：

```
[highgo@hgdb data]$ pg_ctl reload
```

6 高可用 db_ha

6.1 db_ha 简介

瀚高高可用软件（db_ha）是基于 WAL 日志同步方案的高可用架构，支持瀚高系列数据库和 PostgreSQL（9.x-14 版本）。db_ha 针对的是流复制集群（主备/一写多读/一主多备/对等服务等），运行的模式可以是同步或者异步。

提供主备节点 VIP 功能，并对所有 VIP 进行管理监控；具有防脑裂检测机制，杜绝脑裂的产生；支持自动同步转异步功能，避免主库操作被挂起；提供统一的集群管理命令，操作简单方便；提供可靠的异常告警机制，最大限度保证重要信息及时传递；具有高效的业务恢复能力，保障 RTO（恢复时间目标）小于 8 秒，即从灾难状态恢复到可运行状态时间 8 秒；保证 RPO 等于 0，即切换时向前恢复的数据的时间长度为 0，实现切换前后数据 0 丢失。

6.2 部署高可用

6.2.1 数据库相关配置

本高可用系统使用了一些额外的数据库设置，故您需要在主节点使用特定参数初始化一个新的数据目录。

6.2.1.1 主节点数据库

(1) 以数据库管理员账号，创建数据目录

```
su - highgo
```

(2) 以数据库管理员账号，初始化数据目录。

用 initdb 命令初始化数据库时加上 --data-checksums，如：

```
[highgo@hgdb bin]$ initdb -D /usr/local/hgdb/data/ --data-checksums
```

(3) 修改配置文件 pg_hba.conf

修改数据目录中的配置文件 pg_hba.conf，以使流复制不需要密码。例如：

```
# IPv4 local connections 项目下添加
host    all             all             192.168.31.0/24            trust
最后添加：
host    replication     highgo        192.168.31.0/24（配置对应网段即可）       trust
```

(4) 修改配置文件 postgresql.conf 中的如下配置项

```
wal_log_hints = on;
full_page_writes = on;
wal_keep_size=100
listen_addresses = '*'
```

(5) 流复制增强与 db_ha 的同步降级

在同步流复制集群中，常常会遇到服务器软硬件故障、网络抖动、服务器资源紧张等导致备节点无法和主节点保持同步模式，主节点挂起，不能继续提供服务，严重影响到业务的连续性；瀚高企业版数据库系统 V7 增加了流复制增强功能，即在 postgresql.conf 配置文件中提供参数 hg_use_adaptive_standby_sync，如果设置为 on，节点出现流复制异常时，会自动将节点由同步降为异步，高可用自身的同步降级功能是不生效的；若数据库无流复制增强功能或未开启，在同步流复制异常时，db_ha 的同步降级功能会被触发。

(6) 以数据库管理员账号运行主节点数据库

由于在后续步骤中要为备节点创建基础备份，故应当保持主库运行。例如：

```
配置好了环境变量直接：pg_ctl start
或者 cd /usr/local/hgdb/bin 下：pg_ctl -D /usr/local/hgdb/data/ -l logfile start
```

6.2.1.2 备节点数据库

（1）以数据库管理员账号，获取一个基础备份。

```
su - highgo
```

请修改具体参数以适应您的情形，例如:

```
cd 到/usr/local/hgdb/bin 目录下
pg_basebackup -h 192.168.31.21 -U highgo -F p -P -X fetch -R -D /usr/local/hgdb/data -l
postgresbackup
```

上述命令为固定格式，其中

```
192.168.31.21                          主节点的 IP 地址
highgo                                 数据库管理员账号
/usr/local/hgdb/data                   上一步中创建的数据目录
postgresbackup                         为此备份的标签
```

（2）修改 data 目录权限

```
chmod -R 0750 data
```

6.2.2 agent

6.2.2.1 配置 agent 的配置文件

配置文件中的配置项没有默认值，每一项都需要填写。一个 agent.conf 文件的配置示例如下。

```
listen_port = 6666                          # agent 的监听端口
local_pg_path = /usr/local/hgdb/data        #本机数据库的数据目录
```

```
pg_connect_timeout = 60                          # 连接数据库时的超时时间
network_card_for_vip = ens33                     # 服务器上用于绑定 vip 的网卡
log_path = /home/highgo/db_ha_log                # agent 用于存放日志的目录
#the value of log level is following in ascending order : LOG_TRACE, LOG_DEBUG, LOG_INFO,
LOG_WARN, LOG_ERROR, LOG_FATAL
log_level_into_file = LOG_INFO             #日志等级达到 log_level_into_file 的会写入日志文
件。
log_level_on_screen = LOG_FATAL            #日志等级达到 log_level_into_file 的会显示在命令
行。
primary_connect_timeout = 60              #判断与主监控失联的时间（默认值 60 秒）
```

其中，listen_port 是 agent 的监听端口，每套配置文件的值不可重复。建议每套配置文件使用专门的日志目录以对日志加以区别。

日志等级可以设置为：LOG_TRACE, LOG_DEBUG, LOG_INFO, LOG_WARN, LOG_ERROR, LOG_FATAL

痕迹、调试、信息、警告、错误、严重。等级从低到高，日志等级设置越高打印的东西就越少，例如设置成为 LOG_FATAL 就只会打印严重的错误信息。

6.2.2.2 修改 IP 命令权限

使用 Linux 系统管理员权限（root），为 IP 命令添加 S 权限

```
执行命令 which ip 找到 ip 命令的路径，如
/usr/sbin/ip
sudo chmod +s /usr/sbin/ip
```

6.2.2.3 启动 agent

以 highgo 账号运行 agent

```
su - highgo
cd 到/usr/local/db_ha/bin 下执行./agent  /usr/local/db_ha/conf/agent.conf
```

其中

```
/usr/local/db_ha/conf/agent.conf                配置文件
```

正是由于 agent 程序在运行时支持选择配置文件，才使得可以在同一台机器上部署多套高可用程序。后续小节中涉及的 primary_monitor、secondary_monitor、db_ha 也是如此。

6.2.2.4 设置 agent 开机自启动和服务故障自动重启功能

db_ha 从 v4.1.0 开始支持通过配置 systemd 的方式来实现 agent 开机自启动和服务故障自动重启功能。
用户需要配置两个服务文件：

- db_ha-config：配置 lib 路径
- db_ha-agent.service：配置 agent 所在路径，并通过设置 Restart 参数为 no/always 来控制是否开启服务故障自动重启功能。

rpm 包安装完毕会在/usr/local/db_ha 路径下包含一个新的文件夹 self_start_service，

需要配置的两个服务文件就在这个目录下。

```
[root@host1 self_start_service]# pwd
/usr/local/db_ha/self_start_service
[root@host1 self_start_service]# ls
db_ha-agent.service  db_ha-config
```

详细配置方法如下。

6.2.2.4.1 配置服务文件

（1）db_ha-config 文件

```
LD_LIBRARY_PATH=/usr/local/hgdb/lib
```

（2）db_ha-agent.service 文件

用户仅需配置以下蓝色部分，其他默认即可：

```
[Unit]
Description=db_ha-agent
After=network.target
[Service]
Type=forking
#choose user and group
User=highgo                          //用户和组
Group=highgo
EnvironmentFile=/etc/sysconfig/db_ha-config
# Start the db_ha-agent process
ExecStart=/usr/local/db_ha/bin/agent /usr/local/db_ha/conf/agent.conf       //agent 所在路径
#Stop the db_ha-agent process,the main process is killed by default
#ExecStop=
Restart=no                           //no 表示服务故障不自动重启，可以选择 always 表示总会重启
[Install]
WantedBy=multi-user.target
```

6.2.2.4.2 设置开机自启动和服务故障自动重启功能

服务文件配置完成后，执行以下命令设置开机自启动和服务故障自动重启功能：

```
[root@node-0002 agent.autostart]# cp db_ha-config /etc/sysconfig/

[root@node-0002 agent.autostart]# cp db_ha-agent.service /lib/systemd/system

[root@node-0002 agent.autostart]# systemctl enable db_ha-agent.service

Created symlink from /etc/systemd/system/multi-user.target.wants/db_ha-agent.service to /usr/lib/systemd/system/db_ha-agent.service.
```

注意：

如果 db_ha-agent.service 配置文件里的 Restart=no,则以上命令仅开启开机自启动功能；如果 db_ha-agent.service 配置文件里的 Restart=always,则以上命令将开启开机自启动功能和服务故障自启动功能。

6.2.2.4.3 取消开机自启动和服务故障自动重启功

可执行以下命令取消开机自启动和服务故障自动重启功能：

```
systemctl disable db_ha-agent.service
```

6.2.2.4.4 修改服务文件

如需修改服务文件，则修改完毕必须执行以下命令才能使其生效：

```
systemctl daemon-reload
```

6.2.3 ha_isready

高可用 4.0.1 新增检查集群配置工具 ha_isready，用户可以视情况选择是否执行，该工具用于检查用户基本配置是否合理。

以 highgo 账号运行 ha_isready

```
su - highgo
cd 到/usr/local/db_ha/bin 下执行 ./ha_isready /usr/local/db_ha/conf/primary_monitor.conf
```

在 agent 与数据库搭建完成，monitor 启动前，运行该工具，可以检查主库的状态，所有 agent 的状态，数据库密码配置，LIB 库配置。

6.2.4 primary_monitor

6.2.4.1 配置 primary_monitor 的配置文件

配置文件中必须配置主节点信息。可选择手工填写其他数据节点或使用后续章节添加节点。举例如下。

```
#number of cluster nodes
block_repair_switch = on  #块修复开关
block_repair_path_file = /usr/local/db_ha/log/special_block_repair.tmp #块修复所需的文件名及其全路径
command_repeat_count = 5 #执行失败的操作重复执行数
daemon_server_port = 8000 #监听端口号
monitor_cycle_time = 3 #primary_monitor 循环一周期的时间
secondary_max_delay = 3#secondary_montior 最大延迟周期数
max_async_delay = 5 #异步流复制延迟最大值（超过此值报警）单位 M
log_path = /home/highgosys/db_ha_log #日志文件路径 1.6 开始会自动生成日志文件
replication_failure_standby_delete = on#流复制失败是否剔除集群
failure_node_monitored = on
failure_node_polling_interval = 5
node_restart_times = 3
#user_input_switch can choose sql/shell/all/off
user_input_switch = off
user_input_sql = none
user_input_shell = none
new_node_streaming_replication_waiting_time = 60
method_sync = FIRST  #控制 synchronous_standby_names 的同步方法
num_sync = 2         #控制 synchronous_standby_names 的同步个数
log_level_info_file = LOG_INFO   #日志等级达到 log_level_into_file 的会写入日志文件。
log_level_on_screen = LOG_ERROR  #日志等级达到 log_level_into_file 的会显示在命令行。
db_restart_timeout = 10          #控制重启数据库的超时时间
db_rewind_timeout = 20           #数据库 rewind 超时时间
db_rewind_timeout = 20           #数据库 rewind 超时时间
arbitration_judge = on           #控制外部 ping 点开关
arbitration_ip = 192.168.31.40   #外 ping 的 ip，集群服务器一定要能 ping 通的
auto_start_new_secondary = on    #是否开启 secondary monitor 自动拉起功能

#Node information
nodeX_nodetype = PRIMARY#节点类型：PRIMARY（主节点）、STANDBY（备节点）,X 表示数字
nodeX_nodeip = 192.168.31.67 #节点 ip
```

```
nodeX_dbname = postgres #连接数据库用的 dbname
nodeX_applicationname = node1 #节点的复制名（用于流复制）
nodeX_username = postgres #连接节点用的数据库用户名
nodeX_streaming = NONE #节点的流复制类型（主节点填 NONE，同步填 SYNC,异步填 ASYNC）
nodeX_nodeport = 5432 #节点的数据库监听端口号
nodeX_agentport = 6666 #节点的 agent 端口号
nodeX_healty = t #节点状态 t 健康 f 不健康
nodeX_vip_num = 1 #节点 VIP 个数
nodeX_vipX = 192.168.90.10 #节点 X 的第 X 个 VIP
```

部分参数解释如下：

1、failure_node_monitored = on/off 此选项控制 monitor 对于程序自动剔除的节点是否继续监控，on 为继续监控，off 为不在监控（通过 db_ha 手动删除的节点不会被监控）

2、failure_node_polling_interval = 5 此选项控制 monitor 每多少个循环周期检查一次失效节点，在 1 配置打开后生效

3、node_restart_times = 3 此选项控制主节点发生故障后，monitor 通知 agent 重新拉起主节点的重复次数，超过此次数再进行切换

4、user_input_switch = sql/sh/all/off

user_input_sql = "select * from test01"

user_input_shell = /home/wangb/pghig/ha/pgmanager/pg_monitor/test.sh

user_input_switch 此选项控制在发生 failover 或 switchover 新主提升成功绑定 VIP 前，是否需要执行一些指令，all 表示需要执行 shell 和 sql 指令，sh 表示只执行 shell 指令，sql 表示只执行 sql 指令，off 表示不执行指令。

user_input_sql 此选项可配置一条 sql 指令，需注意 sql 指令要在双引号内。

user_input_shell 此选项可配置一个 shell 的路径及文件。

5、 new_node_streaming_replication_waiting_time = 60 此选项表示新加入集群的节点或新回归集群节点的流复制即使超出设定值也不会立即剔除节点，在等待该配置值（单位秒）后，才会剔除。

6、 method_sync = FIRST 此选项可选择填写 FIRST、ANY、EMPTY 用来控制 synchronous_standby_names 参数的同步方法

7、 num_sync = 2 此选项用来控制 synchronous_standby_names 的个数

（关于 synchronous_standby_names 规则详情请参考 PG 手册）

用户修改 method_sync、num_sync 时，会对应修改 synchronous_standby_names。如设置 method_sync = FIRST 和 num_sync = 2，对应的参数为 synchronous_standby_names = 'FIRST 2 (XX,XX)'。

8、log_level_info_file = LOG_INFO 以及 log_level_on_screen = LOG_ERROR，日志等级可以设置为：LOG_TRACE, LOG_DEBUG, LOG_INFO, LOG_WARN, LOG_ERROR, LOG_FATAL

痕迹、调试、信息、警告、错误、严重。等级从低到高，日志等级设置越高打印的东西就越少，例如设置成为 LOG_FATAL 就只会打印严重的错误信息。

9、db_restart_timeout（单位秒）配置项，控制重启数据库的超时时间。这一项为 1.5 新添加功能

10、block_repair_switch = on 块修复开关，只在块修复的情况下使用。

block_repair_path_file = /usr/local/db_ha/log/special_block_repair.tmp

块修复所需的文件名及其全路径。这两个参数是配合瀚高数据库块修复功能使用。

11、db_rewind_timeout = 20 数据库 rewind 超时时间（单位秒）。让 primary_monitor 等待指定的时间（参数设定为 20 秒），若在该时间内完成了指定的 rewind 操作，则会在原地进行后续操作。否则超时后，节点未完成指定的操作，其状态会被判定为不健康，等待该节点 rewind 成功后，高可用会自动判定为健康。

12、arbitration_judge = on ，arbitration_ip = 192.168.31.40：为高可用 1.8 新增功能，当集群只有两个数据库节点，并且主监控在其中一个节点上，当两个节点中有一个节点离网，剩下的节点无法判断是否是自身问题，如果配置外部 ping 点，可以依据是否 ping 通外部 ping 点再做相应处理。（ping 通则表示当前节点无问题，继续正常流程，ping 不通，打印报警日志，退出主监控）。**注意：ping 点的稳定性必须需要有严格的保障（建议设置成网关的 IP），且 monitor 必须能够 ping 通。**

13、auto_start_new_secondary = on，在自动拉起服务开启的情况下，如果 secondary monitor 失联，primary monitor 会自动拉起一个新的 secondary monitor 进程，规则如下：

- 优先选择除了 primary monitor 所在节点以外的备节点启动 secondary monitor；
- 如果除了 primary monitor 所在的备节点以外没有其他的备节点，则选择主节点启动 secondary monitor；

- 如果集群中只有一个节点并且 primary monitor 在此节点，则提示没有适合的 secondary monitor 宿主；
- 如果集群初始化时只启动了 primary monitor，没有启动 secondary monitor，会等待 5min 时间管理员手动启动，5min 过后如果仍未启动 secondary monitor，primary monitor 会认为 secondary monitor 失联，自动启动 secondary monitor。

【注意】db_ha 工具查询指令会显示 agent 状态、secondary monitor 状态、和流复制状态。

其中，agent 和 seconadry，只显示“正常”/“异常”；流复制状态主节点显示“none”，备节点可能是“startup/catchup/streaming/backup/stopping/stopped”的一种。

6.2.4.2 启动 primary_monitor

企业版以 highgo 账号运行 primary_monitor

```
su - highgo
cd 到 /usr/local/db_ha/bin/ 下 执 行 ./primary_monitor /usr/local/db_ha/conf/primary_monitor.conf
```

其中

```
/usr/local/db_ha/conf/primary_monitor.conf 配置文件
```

6.2.5 secondary_monitor

6.2.5.1 配置 secondary_monitor 的配置文件

配置文件中的配置项没有默认值，每一项都需要填写。一个 secondary_monitor.conf 文件的配置示例如下。

```
/usr/local/db_ha/bin/secondary_monitor /usr/local/db_ha/conf/secondary_monitor.conf

log_path = /usr/local/db_ha/log       #用于存放日志的目录
primary_monitor_ip = 192.168.31.20    #primary_monitor 的 IP 地址
primary_monitor_port = 8000           #primary_monitor 的监听端口号
primary_monitor_fail_count_max = 3    #探测 primary_monitor 失败多少次后启动在本地启动监控
primary_monitor_path = /usr/local/db_ha/bin/primary_monitor  #本地 primary_monitor 程序的路径
primary_monitor_conf_path =/usr/local/db_ha/conf/primary_monitor.conf
#本地 primary_monitor                 #配置文件的路径
primary_monitor_log_path = /home/highgodba/db_ha_log/monitor.log #本地 primary_monitor 的日志
check_interval = 10
#the value of log level is following in ascending order : LOG_TRACE, LOG_DEBUG, LOG_INFO, LOG_WARN, LOG_ERROR, LOG_FATAL
log_level_into_file = LOG_INFO       #日志等级达到 log_level_into_file 的会写入日志文件。
log_level_on_screen = LOG_FATAL      #日志等级达到 log_level_into_file 的会显示在命令行。
```

其中，primary_monitor_port 是 primary_monitor 的监听端口，每套配置文件的值不可重

复。建议每套配置文件使用专门日志目录以对日志加以区别。

日志等级可以设置为：LOG_TRACE, LOG_DEBUG, LOG_INFO, LOG_WARN, LOG_ERROR, LOG_FATAL

痕迹、调试、信息、警告、错误、严重。等级从低到高，日志等级设置越高打印的东西就越少，例如设置成为 LOG_FATAL 就只会打印严重的错误信息。

6.2.5.2 启动 secondary_monitor

企业版以 highgo 账号运行 secondary_monitor

```
cd 到 /usr/local/db_ha/bin 下 执 行 ./secondary_monitor /usr/local/db_ha/conf/secondary_monitor.conf
```

其中

```
/usr/local/db_ha/conf/secondary_monitor.conf 配置文件
```

6.2.6 db_ha

6.2.6.1 配置 db_ha 的配置文件

配置文件中的配置项没有默认值，每一项都需要填写。一个 db_ha.conf 文件的配置示例如下。

```
primary_monitor_ip = 192.168.31.20   #primary_onitor 程序的 IP 地址
primary_monitor_port = 8000          #primary_monitor 程序守护的端口
secondary_monitor_ip = 192.168.31.22  #secondary_monitor 程序的 IP 地址
```

其中，primary_monitor_port 是 primary_monitor 的监听端口，每套配置文件的值不可重复。建议每套配置文件使用专门日志目录以对日志加以区别。

当 secondary_monitor 提升 primary_monitor 时，db_ha 会自动使用该参数的 IP 地址进行连接

6.2.6.2 运行 db_ha

在此仅使用“查询集群信息”功能，以查看配置生否成功。运行 db_ha 程序时，需要 primary_monitor 程序已经运行。

企业版以 highgo 账号运行 db_ha。

例如

```
cd 到/usr/local/db_ha 下执行./bin/db_ha select -f /usr/local/db_ha/conf/db_ha.conf
```

其中

```
/usr/local/db_ha/conf/db_ha.conf                配置文件
```

若配置成功，则显示当前的配置信息。

6.2.6.3 注意事项

- 参数关联性

primary_monitor.conf 中的配置项 monitor_cycle_time、secondary_max_delay 与 secondary_monitor.conf 中的配置项 check_interval 必须满足以下关系：

```
monitor_cycle_time × secondary_max_delay > check_interval
```

6.2.7 ha_ctl

6.2.7.1 配置 ha_ctl

ha_ctl 工具为 1.8 新添加工具在解压缩 rpm 包后在 bin 目录就自带。

```
[highgo@qybl bin]$ ls
agent  db_ha  ha_ctl  primary_monitor  secondary_monitor
```

6.2.7.2 运行 ha_ctl

企业版以 highgo 账号运行 ha_ctl,并且对 highgo 用户添加环境变量

```
export PATH=/usr/local/db_ha/bin/:$PATH
```

ha_ctl 工具提供了本节点的 agent、primary_monitor、secondary_monitor 的启动、停止、重启功能
例如：

```
ha_ctl -a -f /usr/local/db_ha/conf/agent.conf restart
```

其中

```
/usr/local/db_ha/conf/agent.conf                配置文件
```

若配置成功，则重启当前节点的 agnet。

6.3 db_ha 的使用

agent、primary_monitor、secondary_monitor 是守护进程，只有启动操作，已在上一章节中介绍。应当使用 db_ha 提供的停止命令关闭上述守护进程。不建议使用 kill 命令杀死上述程序。db_ha 为用户提供了各种管理高可用系统的命令。应当以“3 配置 Linux 系统账号”中创建的账号运行 db_ha 程序。

db_ha 帮助命令：./db_ha --help 或者./db_ha -h

```
        Usage: db_ha [options] -f <db_ha.conf>
        select                  Querying cluster Status
```

```
          add [ip] [agent port]   Add a standby node
          -a,                     Example Add a standby asynchronization node
          -s,                     Example Add a standby synchronization node
          -n [application name],  Specifies the application name
          -p [database port],     Specifies the database port

          delete [ip]             Delete a specified IP node

          switchover [ip]         The standby with the specified IP is promoted to the new
master,the old master was demoted to standby
          -F,                     Forced to switch,maybe lose data

          stop                    Stop monitor(primary_monitor/secondary_monitor/agent)
          -a,                     Stop primary_monitor,secondary_monitor and all agent
          -m,                     Just stop primary_monitor and secondary_monitor

          restart [ip]            Restart the specify IP database

          syncmode [ip]           Change the database synchronous mode
          -a,                     Change the database to asynchronous mode
          -s,                     Change the database to synchronous mode

          showvip                 Viewing virtual IP Addresses
          -a,                     View all virtual IP addresses in the cluster
          -v [ip],                View the specified virtual IP address

          addvip [ip] -v [vip]     Example Add a virtual IP address to the specified IP
address

          rmvip  [ip] -v [vip]    The virtual IP address is deleted from the specified IP
address

          reload                  reload the primary_monitor.conf
```

本节描述命令时，“/”表示前后参数二选一，“[]”表示可选参数，为了阅读方便、同时因为部分命令过长，在描述时展示成了多行。

6.3.1 查询集群信息

- 命令描述

```
db_ha select -f configuration_file
```

其中

```
select                                  查询集群信息
-f                                      使用配置文件
configuration_file                      配置文件
```

- 举例

下述命令表示对 db_ha.conf 配置文件代表的集群进行集群信息的查询：

```
到/usr/local/db_ha/bin 下执行./db_ha select -f /usr/local/db_ha/conf/db_ha.conf
```

6.3.2 增加节点

- 命令描述

```
db_ha add ip_address
          agent_port
          -f configuration_file
          -n app_name
          -p database_port
          -s/-a
```

其中

add	增加节点
ip_address	数据节点的实际 IP 地址
agent_port	数据节点上配置的 agent 所监听的端口号
-f	使用配置文件
configuration_file	配置文件
-n	使用 pg_stat_replication 的 application_name 列
app_name	连接到 WAL 发送进程的应用名称，与 pg_stat_replication 中的 application_name 列作用相同。在同一集群中，app_name 不可重复。
-p	使用数据库的监听端口
database_port	数据库的监听端口
-s/-a	必须选择其中的一项。-s 表示以同步方式加入集群，-a 表示以异步方式加入集群

- 举例

下述命令表示，为 db_ha.conf 代表的集群，添加一个地址为 192.168.31.68、agent 的端口为 6666、数据库的端口为 5432 的数据节点，其应用名为 node2,以同步的方式加入。

```
到 /usr/local/db_ha/bin 下 执 行 ./db_ha add 192.168.31.68 6666 -f /usr/local/db_ha/conf/db_ha.conf -n node2 -p 5866 -s
```

6.3.3 踢除节点

- 命令描述

```
db_ha delete  ip_address
              -f configuration_file
```

其中

delete	踢除节点
ip_address	数据节点的实际 IP 地址，此数据节点应在配置文件所代表的集群中
-f	使用配置文件
configuration_file	配置文件

- 举例

下述命令表示，为 db_ha.conf 代表的集群，踢除地址为 192.168.31.68 的数据节点。

```
到 /usr/local/db_ha/bin 下 执 行 ./db_ha delete 192.168.31.68 -f /usr/local/db_ha/conf/db_ha.conf
```

6.3.4 主备切换

- 命令描述

```
db_ha switchover  ip_address
                  [-F]
                  -f configuration_file
```

其中

switchover	主备切换
ip_address	数据节点的实际 IP 地址，此数据节点应在配置文件所代表的集群中
-F	可选参数。不使用本参数时，若新主的 LSN 没有跟上旧主的 LSN，switchover 会中止。使用本参数时，会强制切换。
-f	使用配置文件
configuration_file	配置文件

- 举例

下述命令表示，为 db_ha.conf 代表的集群，将地址为 192.168.31.68 的数据节点切换为主节点。

```
到/usr/local/db_ha/bin 下执行./db_ha switchover 192.168.31.68 -f /usr/local/db_ha/conf/db_ha.conf
```

6.3.5 停止监控

- 命令描述

```
db_ha stop  -a/-m
            -f configuration_file
```

其中

stop	停止监控
-a/-m	必须选择其中的一项。-a 表示停止集群中 primary_monitor、secondary_monitor 和所有的 agent。-m 表示只停止 primary_monitor 和 secondary_monitor
-f	使用配置文件
configuration_file	配置文件

【注意】primary_monitor 分为两个进程，在执行完上面的命令后，primary_monitor 的其中一个进程会等待 secondary_monitor 的打卡，并通知 secondary_monitor 退出，然后自己也会退出，如果超过 30s，secondary_monitor 没来打卡，primary_monitor 的这个进程不在继续等待并退出。（打卡：以指定时间间隔进行连通性检测。）

- 举例

下述命令表示，为 db_ha.conf 代表的集群，停止集群中 primary_monitor、secondary_monitor 和所有的 agent。

```
cd 到/usr/local/db_ha/bin 下执行./db_ha stop -a -f /usr/local/db_ha/conf/db_ha.conf
```

6.3.6 重启数据库

- 命令描述

```
db_ha restart ip_address
              -f configuration_file
```

其中

restart	重启数据库
ip_address	数据节点的实际 IP 地址，此数据节点应在配置文件所代表的集群中
-f	使用配置文件
configuration_file	配置文件

- 举例

下述命令表示，为 db_ha.conf 代表的集群，重启地址为 192.168.31.68 的数据节点。

```
cd 到/usr/local/db_ha/bin 下执行./db_ha restart 192.168.31.68 -f /usr/local/db_ha/conf/db_ha.conf
```

6.3.7 更改数据库同异步模式

- 命令描述

```
db_ha syncmode  ip_address
                -s sync/async
                -f configuration_file
```

其中

syncmode	更改数据库同异步模式
ip_address	数据节点的实际 IP 地址，此数据节点应在配置文件所代表的集群中
-s	使用同异步参数
sync/async	必须选择其中的一项。sync 表示同步模式，async 表示异步模式
-f	使用配置文件
configuration_file	配置文件

- 举例

下述命令表示，为 db_ha.conf 代表的集群，将地址为 192.168.31.68 的数据节点修改为异步模式。

```
cd 到/usr/local/db_ha/bin 下执行./db_ha syncmode 192.168.31.68 -s async -f /usr/local/db_ha/conf/db_ha.conf
```

6.3.8 VIP 操作

6.3.8.1 查看所有 VIP

- 命令描述

```
db_ha showvip -a
              -f configuration_file
```

其中

showvip	查看 VIP
-a	所有数据节点
-f	使用配置文件
configuration_file	配置文件

- 举例

下述命令表示，为 db_ha.conf 代表的集群，查看所有数据节点的 VIP。

```
cd 到/usr/local/db_ha/bin 下执行./db_ha showvip -a -f /usr/local/db_ha/conf/db_ha.conf
```

6.3.8.2 查看单个节点的 VIP

- 命令描述

```
db_ha showvip -v ip_address
              -f configuration_file
```

其中

showvip	查看 VIP
-v	单个数据节点
ip_address	数据节点的实际 IP 地址，此数据节点应在配置文件所代表的集群中
-f	使用配置文件
configuration_file	配置文件

- 举例

下述命令表示，为 db_ha.conf 代表的集群，查看地址为 192.168.31.68 的数据节点的 VIP。

```
cd 到/usr/local/db_ha/bin 下执行/db_ha showvip -v 192.168.31.68 -f /usr/local/db_ha/conf/db_ha.conf
```

6.3.8.3 添加单个 VIP

- 命令描述

```
db_ha addvip  ip_address
              -v vip_address
              -f configuration_file
```

其中

addvip	添加 VIP
ip_address	数据节点的实际 IP 地址，此数据节点应在配置文件所代表的集群中
-v	指定 VIP
vip_address	VIP 地址
-f	使用配置文件
configuration_file	配置文件

【注意】db_ha 工具仅支持一次添加单个 VIP 的操作，不支持一次添加多个 VIP 的操作

- 举例

下述命令表示，为 db_ha.conf 代表的集群，给地址为 192.168.31.68 的数据节点的添加一个值为 192.168.31.231 的 VIP。

```
cd 到/usr/local/db_ha/bin 下执行./db_ha addvip 192.168.31.68 -v 192.168.31.231 -f
/usr/local/db_ha/conf/db_ha.conf
```

6.3.8.4 删除单个 VIP

- 命令描述

```
db_ha rmvip ip_address
            -v vip_address
            -f configuration_file
```

其中

rmvip	删除 VIP
ip_address	数据节点的实际 IP 地址，此数据节点应在配置文件所代表的集群中
-v	指定 VIP
vip_address	VIP 地址
-f	使用配置文件
configuration_file	配置文件

【注意】db_ha 工具仅支持一次删除单个 VIP 的操作，不支持一次删除多个 VIP 的操作

- 举例

下述命令表示，为 db_ha.conf 代表的集群，给地址为 192.168.31.68 的数据节点的删除一个值为 192.168.31.231 的 VIP。

```
cd 到/usr/local/db_ha/bin 下执行/db_ha rmvip 192.168.31.68 -v 192.168.31.231 -f
/usr/local/db_ha/conf/db_ha.conf
```

6.3.9 重载配置文件

- 命令描述

```
db_ha reload -f configuration_file
```

其中

reload	重载配置文件
-f	使用配置文件
configuration_file	配置文件

- 举例

下述命令表示对配置文件重载：该命令重载配置文件部分参数（配置文件与数据库节点相关的配置、守护端口和日志路径 reload 不生效，需重启生效）

```
cd 到/usr/local/db_ha/bin 下执行./db_ha reload -f /usr/local/db_ha/conf/db_ha.conf
```

6.4 ha_ctl 使用

db_ctl 帮助命令：./db_ctl --help 或者./db_ctl -h

```
$ ha_ctl -h
Options:
 -a,      choose agent process
 -m,      choose primary monitor process
 -s,      choose secondary process
 -f,      this option is followed by the configuration file full path
          example: -f /home/conf/agent.conf
Usage:
 start          start database
 stop           stop database
 restart        restart database
 pg_basebackup  clone data from primary
 pg_rewind      resynchronizes cluster
```

本节描述命令时，“/”表示前后参数二选一。

6.4.1 ha_ctl 对 agent 的操作

- 命令描述

```
ha_ctl -a -f  configuration_file start/stop/restart
```

其中

-a	对 agnet 进行操作
-f	使用配置文件
configuration_file	配置文件
start	启动
stop	停止
restart	重启

- 举例

下述命令表示，为重启本节点的 agent。

```
ha_ctl -a -f /usr/local/db_ha/conf/agent.conf restart
```

6.4.2 ha_ctl 对 primary_monitor 的操作

- 命令描述

```
ha_ctl -m -f  configuration_file start/stop/restart
```

其中

-m	对 primary_monitor 进行操作
-f	使用配置文件
configuration_file	配置文件
start	启动
stop	停止
restart	重启

- 举例

下述命令表示，为重启本节点的 primary_monitor。

```
ha_ctl -m -f /usr/local/db_ha/conf/primary_monitor.conf restart
```

6.4.3 ha_ctl 对 secondary_monitor 的操作

- 命令描述

```
ha_ctl -s -f  configuration_file start/stop/restart
```

其中

-s	对 secondary_monitor 进行操作
-f	使用配置文件
configuration_file	配置文件
start	启动
stop	停止
restart	重启

- 举例

下述命令表示，为重启本节点的 secondary_monitor。

```
ha_ctl -s -f /usr/local/db_ha/conf/secondary_monitor.conf restart
```

6.4.4 ha_ctl pg_basebackup 的操作

可以用 ha_ctl 来进行基础备份

```
ha_ctl pg_basebackup -h 192.168.31.21 -U highgo -F p -P -X fetch -R -D /usr/local/hgdb/data
-l postgresbackup
```

上述命令为固定格式，其中

192.168.31.21	主节点的 IP 地址
highgo	数据库管理员账号
/usr/local/hgdb/data	创建的数据目录
postgresbackup	为此备份的标签

6.4.5 ha_ctl pg_rewind 的操作

ha_ctl pg_rewind 可以在宕机后自动恢复，并且可以通过--write-recover-conf 选项来配置备库。 pg_rewind 也可以在目标实例上使用 restore_command 来获取所需的 WAL 日志。
使用方法
首先去备库 postgresql.conf 参数文件里修改"wal_log_hints = on"，然后执行

```
ha_ctl pg_rewind --target-pgdata /usr/local/hgdb/data --source-server='host=192.168.31.21
port=5432  user=highgo dbname=highgo' -P
```

上述命令为固定格式，其中

--target-pgdata	备库的数据目录
--source-server='host	主端的 IP 地址
user=highgo	数据库用户
dbname=highgo	数据库名称

以上示例就是是备库追上主库的 wal 日志。
【注意】rewind 操作：当旧的主节点以备节点的身份加入集群时，搭建流复制关系时可能失败，可以通过 rewind 的方式使主备的数据一致以加入进群。

6.5 手动安装示例

由于宽松模式的部署最为简单，只需单独设置即可，故此处演示最为复杂的、各种设置操作都涉及到的一般部署。

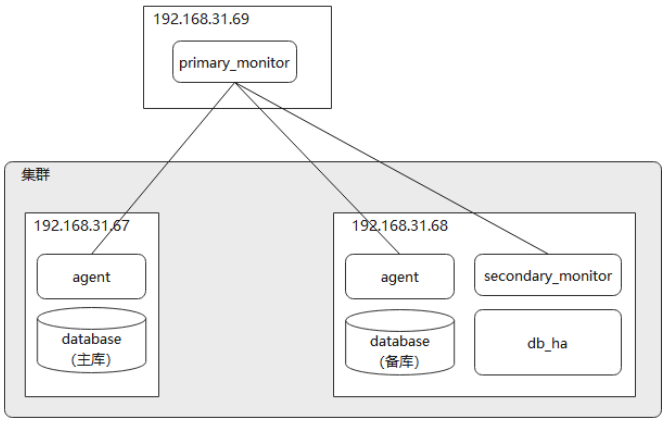
本示例共使用 3 台 Linux 机器，采用一般部署。假设数据库程序已经安装。2 台 Linux 作为数据节点，信息如下：

IP 地址	application_name	运行程序
192.168.31.21	node1	主数据库 agent
192.168.31.22	node2	备数据库 agent secondary_monitor db_ha

1 台用于运行 primary_monitor，信息如下：

IP 地址	运行程序
192.168.31.20	primary_monitor

拓扑结构图如下所示：



下面进行步骤讲解。

6.5.1 在 192.168.31.21 上进行安装部署

(1) 将 rmp 安装包复制到/usr/local/test 目录下，其结果如下

```
[root@thbytwo-centos2009 ~]# cd /usr/local/test
[root@thbytwo-centos2009 test]# ll
总用量 128
-rwxrwxrwx. 1 root root 129384 5 月  17 15:57 db_ha4.0.2-ee-centos7-x86_64.rpm
```

(2) 检测是否可安装

```
[highgo@thbytwo-centos2009 ~]$ pg_config | grep LIBDIR | awk '{print $3}'
/usr/local/hgdb/lib
/usr/local/hgdb/lib
```

这里显示的值与实际安装的数据库程序的 lib 目录一致，可以进行后续步骤的安装。

(3) 安装 rpm 包

```
[root@thbytwo-centos2009 test]# rpm -ivh db_ha4.0.2-ee-centos7-x86_64.rpm
准备中...                          ################################# [100%]
正在升级/安装...
   1:db_ha4.0.2-ee-centos7-x86_64.rpm                  ################################
[100%]
```

(4) 配置账号

创建隐藏文件，修改权限，编辑内容

```
[root@thbytwo-centos2009 test]# su - highgo
[highgo@thbytwo-centos2009 ~]$ touch ~/.pgpass
[highgo@thbytwo-centos2009 ~]$ chmod 0600 ~/.pgpass
[highgo@thbytwo-centos2009 ~]$ vim ~/.pgpass
```

其中 vim 编辑的内容为

```
*:*:*:*:Highgo@123（初始化数据库的密码）
```

Hgdb 企业版 6.0.4 在安装完毕后已配置了环境变量

```
[root@xqyb1 ~]# su - highgo
Last login: Mon Aug 23 10:03:24 CST 2021 on pts/0
[highgo@xqyb1 ~]$ vim .bash_profile

# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/.local/bin:$HOME/bin

export PATH
export PATH=$PATH:/usr/local/hgdb/bin
export LD_LIBRARY_PATH=/usr/local/hgdb/lib:$LD_LIBRARY_PATH
export PGDATA=/usr/local/hgdb/data
export PATH=/usr/local/db_ha/bin/:$PATH
```

运行以下命令使配置文件生效

```
[highgo@hgdb etc]# source .bash_profile
```

检查环境变量是否生效

```
[highgo@hgdb etc]# echo $PGDATA
/usr/local/hgdb/data
```

修改目录的所有者和权限

```
[highgo@thbytwo-centos2009 ~]$ exit
登出
[root@thbytwo-centos2009 test]# chmod -R 750 /usr/local/db_ha
[root@thbytwo-centos2009 test]# chown -R highgo:highgo /usr/local/db_ha
```

(5) 初始化主数据库

```
[root@thbytwo-centos2009 test]# su - highgo
上一次登录：五 7月 16 10:15:59 CST 2021pts/0 上
[highgo@thbytwo-centos2009 ~ bin]$ initdb -D ../data >/usr/local/hgdb/bin/initdb.log

[highgo@thbytwo-centos2009 ~]$ chmod -R 750 /usr/local/hgdb/data
```

修改主库的配置文件

```
[postgres@thbytwo-centos2009 data]$ vim pg_hba.conf
```

添加内容

```
# "local" is for Unix domain socket connections only
local   all             all                                     trust
# IPv4 local connections:
host    all             all             127.0.0.1/32            trust
host    all             all             0.0.0.0/0               trust
# IPv6 local connections:
host    all             all             ::1/128                 trust
# Allow replication connections from localhost, by a user with the
# replication privilege.
local   replication     all                                     trust
host    replication     all             127.0.0.1/32            trust
host    replication     all             ::1/128                 trust
host    replication     all             0.0.0.0/0               trust
```

编辑配置文件 postgresql.conf

```
[postgres@thbytwo-centos2009 data]$ vim postgresql.conf
```

在 vim 中修改如下内容

```
wal_log_hints = on;
full_page_writes = on;
wal_keep_size=100
listen_addresses = '*'
```

(6) 运行主库

```
[highgo@thbytwo-centos2009 data]$ pg_ctl start

waiting for server to start.... done
server started
```

(7) 创建一个目录用于存放 log

```
[highgo@thbytwo-centos2009 data]$ cd ../
[highgo@thbytwo-centos2009 ~]$ pwd
/home/highgo
[highgo@thbytwo-centos2009 ~]$ mkdir db_ha_log
[highgo@thbytwo-centos2009 ~]$ ls
db_ha_log
```

(8) 配置 agent 的配置文件

```
[highgo@thbytwo-centos2009 ~]$ cd /usr/local/db_ha/conf/
[highgo@thbytwo-centos2009 conf]$ ls
agent.conf  db_ha.conf  primary_monitor.conf  secondary_monitor.conf
[highgo@thbytwo-centos2009 conf]$ vim agent.conf
```

在 vim 中修改如下内容

```
listen_port = 6666
local_pg_path = /usr/local/hgdb/data
pg_connect_timeout = 60
network_card_for_vip = ens33
log_path = /home/highgo/db_ha_log
```

```
#the value of log level is following in ascending order : LOG_TRACE, LOG_DEBUG, LOG_INFO,
LOG_WARN, LOG_ERROR, LOG_FATAL
log_level_into_file = LOG_INFO
log_level_on_screen = LOG_FATAL
primary_connect_timeout = 60
```

(9) 修改 IP 命令权限

使用管理员权限，查找 IP 命令程序所在位置

```
[root@thbytwo-centos2009 ~]# whereis ip
ip: /usr/sbin/ip /usr/share/man/man7/ip.7.gz /usr/share/man/man8/ip.8.gz
```

使用管理员权限，为 IP 命令添加 S 权限

```
[root@thbytwo-centos2009 ~]# chmod +s /usr/sbin/ip
```

6.5.2 在 192.168.31.22 上进行安装部署

(1) 将 rmp 安装包复制到/usr/local/test 目录下，其结果如下

```
[root@thbytwo-centos2009 ~]# cd /usr/local/test
[root@thbytwo-centos2009 test]# ll
总用量 324
-rw-r--r--. 1 root root 129384 7 月  16 13:00 db_ha4.0.2-ee-centos7-x86_64.rpm
```

(2) 检测是否可安装

```
[highgo@thbytwo-centos2009 ~]$ pg_config | grep LIBDIR | awk '{print $3}'
/usr/local/hgdb/lib
/usr/local/hgdb/lib
```

这里显示的值与实际安装的数据库程序的 lib 目录一致，可以进行后续步骤的安装。

(3) 安装 rpm 包

```
[root@thbytwo-centos2009 test]# rpm -ivh db_ha4.0.2-ee-centos7-x86_64.rpm
准备中...                          ################################# [100%]
正在升级/安装...
   1:db_ha4.0.2-ee-centos7-x86_64.rpm                  ################################
[100%]
```

(4) 修改目录权限

由于后续 agent、secodary_monitor 和 db_ha 都会使用 db_ha 目录，所以先设置权限。（若您有安全需求，请酌情赋权。）

```
[root@thbytwo-centos2009 ~]# chmod -R 777 /usr/local/db_ha/
[root@thbytwo-centos2009 ~]# chown -R highgo:highgo /usr/local/db_ha/
```

(5) 配置 agent 账号

创建隐藏文件，修改权限，编辑内容

```
[root@thbytwo-centos2009 test]# su - highgo
[highgo@thbytwo-centos2009 ~]$ touch ~/.pgpass
[highgo@thbytwo-centos2009 ~]$ chmod 0600 ~/.pgpass
[highgo@thbytwo-centos2009 ~]$ vim ~/.pgpass
```

其中 vim 编辑的内容为

```
*:*:*:*:Highgo@123（初始化数据库的密码）
```

继续修改配置文件.bash_profile

```
[highgo@thbytwo-centos2009 ~]$ vim .bash_profile
```

其中 vim 中的最终内容为

```
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/.local/bin:$HOME/bin

export PATH
export PATH=$PATH:/usr/local/hgdb/bin
export LD_LIBRARY_PATH=/usr/local/hgdb/lib:$LD_LIBRARY_PATH
export PGDATA=/usr/local/hgdb/data
export PATH=/usr/local/db_ha/bin/:$PATH
```

使配置文件生效

```
[highgo@thbytwo-centos2009 ~]$ source .bash_profile
```

修改目录的所有者和权限

```
[highgos@thbytwo-centos2009 ~]$ exit
登出
[root@thbytwo-centos2009 test]# chmod -R 750 /usr/local/db_ha/bin/agent
[root@thbytwo-centos2009 test]# chown -R highgo:highgo /usr/local/db_ha/bin/agent
[root@thbytwo-centos2009 test]# chmod -R 750 /usr/local/db_ha/conf/agent.conf
[root@thbytwo-centos2009 test]# chown -R highgo:highgo /usr/local/db_ha/conf/agent.conf
```

修改目录的所有者和权限

```
[highgo@thbytwo-centos2009 ~]$ exit
登出
[root@thbytwo-centos2009 test]# chmod -R 750 /usr/local/db_ha/bin/secondary_monitor
[root@thbytwo-centos2009 test]#
chown -R highgosys:highgosys /usr/local/db_ha/bin/secondary_monitor
[root@thbytwo-centos2009 test]# chmod -R 750 /usr/local/db_ha/conf/secondary_monitor.conf
[root@thbytwo-centos2009 test]#
chown -R highgosys:highgosys /usr/local/db_ha/conf/secondary_monitor.conf
```

(6) 初始化备库

```
[root@thbytwo-centos2009 test]# su - highgo
上一次登录：五 7月 16 13:08:09 CST 2021pts/0 上
[highgo@thbytwo-centos2009 ~]$ cd /usr/local/hgdb/bin
[highgo@thbytwo-centos2009 ~]$ chmod -R 0750 data/
[highgo@thbytwo-centos2009 ~]$ pg_basebackup -h 192.168.31.21 -U highgo -F p -P -X fetch -
R -D /usr/local/hgdb/data -l postgresbackup
40975/40975 kB (100%), 1/1 tablespace
```

(7) 为 agent 创建一个目录用于存放 log

```
[highgo@thbytwo-centos2009 ~]$ pwd
/home/highgo
[highgo@thbytwo-centos2009 ~]$ mkdir db_ha_log
```

(8) 为 secondary_monitor 创建一个目录用于存放 log

```
[highgo@thbytwo-centos2009 ~]$ exit
登出
[root@thbytwo-centos2009 test]# su - highgo
上一次登录：六 7月 17 13:02:09 CST 2021pts/0 上
[highgo@thbytwo-centos2009 ~]$ pwd
/home/highgo
[highgo@thbytwo-centos2009 ~]$ mkdir db_ha_log
```

(9) 配置 agent 的配置文件

```
[root@thbytwo-centos2009 ~]# su - highgo
上一次登录：六 7月 17 13:44:50 CST 2021pts/0 上
[highgo@thbytwo-centos2009 ~]$ cd /usr/local/db_ha/conf
[highgo'@thbytwo-centos2009 conf]$ ls
agent.conf  db_ha.conf  primary_monitor.conf  secondary_monitor.conf
[highgo@thbytwo-centos2009 conf]$ vim agent.conf
```

在 vim 中修改如下内容

```
listen_port = 6666
local_pg_path = /usr/local/hgdb/data
pg_connect_timeout = 60
network_card_for_vip = ens33
log_path = /home/postgres/db_ha_log
#the value of log level is following in ascending order : LOG_TRACE, LOG_DEBUG, LOG_INFO,
LOG_WARN, LOG_ERROR, LOG_FATAL
log_level_into_file = LOG_INFO
log_level_on_screen = LOG_FATAL
primary_connect_timeout = 60
```

(10)配置 secondary_monitor 的配置文件

```
[root@thbytwo-centos2009 ~]# su - highgo
上一次登录：六 7月 17 13:46:01 CST 2021pts/0 上
[highgo@thbytwo-centos2009 ~]$ cd /usr/local/db_ha/conf
[highgo@thbytwo-centos2009 conf]$ ls
agent.conf  db_ha.conf  primary_monitor.conf  secondary_monitor.conf
[highgo@thbytwo-centos2009 conf]$ vim secondary_monitor.conf
```

在 vim 中修改如下内容

```
log_path = /home/highgo/db_ha_log
primary_monitor_ip = 192.168.31.20
primary_monitor_port = 8000
primary_monitor_fail_count_max = 3
primary_monitor_path = /usr/local/db_ha/bin/primary_monitor
primary_monitor_conf_path = /usr/local/db_ha/conf/primary_monitor.conf
primary_monitor_log_path = /home/highgo/db_ha_log/monitor.log
check_interval = 5
#the value of log level is following in ascending order : LOG_TRACE, LOG_DEBUG, LOG_INFO,
LOG_WARN, LOG_ERROR, LOG_FATAL
log_level_into_file = LOG_INFO
log_level_on_screen = LOG_FATAL
```

(11)配置 db_ha 相关文件的权限和所有者

```
[highgo@thbytwo-centos2009 conf]$ exit
登出
[root@thbytwo-centos2009 ~]# chmod -R 750 /usr/local/db_ha/bin/db_ha
[root@thbytwo-centos2009 ~]# chown -R highgo:highgo /usr/local/db_ha/bin/db_ha
[root@thbytwo-centos2009 ~]# chmod -R 750 /usr/local/db_ha/conf/db_ha.conf
[root@thbytwo-centos2009 ~]# chown -R highgo:highgo /usr/local/db_ha/conf/db_ha.conf
```

(12)配置 db_ha 的配置文件

```
[root@thbytwo-centos2009 ~]# su - highgo
上一次登录：六 7 月 17 13:55:12 CST 2021pts/0 上
[highgo@thbytwo-centos2009 ~]$ cd /usr/local/db_ha/conf
[highgo@thbytwo-centos2009 conf]$ ls
agent.conf  db_ha.conf  primary_monitor.conf  secondary_monitor.conf
[highgo@thbytwo-centos2009 conf]$ vim db_ha.conf
```

在 vim 中修改如下内容

```
primary_monitor_ip = 192.168.31.20
primary_monitor_port = 8000
secondary_monitor_ip = 192.168.31.22
```

(13)修改 IP 命令权限

使用管理员权限，为 IP 命令添加 S 权限

```
[root@thbytwo-centos2009 ~]# chmod +s /usr/sbin/ip
```

6.5.3 在 192.168.31.20 上进行安装部署

(1) 将 rmp 安装包复制到/usr/local/test 目录下，其结果如下

```
[root@localhost ~]# cd /usr/local/test
[root@localhost test]# ll
总用量 1928
-rw-r--r--. 1 root root  129384 7 月  16 14:55 db_ha4.0.2-ee-centos7-x86_64.rpm
```

(2) 安装 rpm 包

```
[root@thbytwo-centos2009 test]# rpm -ivh db_ha4.0.2-ee-centos7-x86_64.rpm
准备中...                          ################################# [100%]
```

```
正在升级/安装...
   1:db_ha4.0.2-ee-centos7-x86_64                 ################################# [100%]
```

(3) 配置账号

添加系统账号，修改密码，创建隐藏文件，修改权限，编辑内容

```
[root@thbytwo-centos2009 test]# su - highgo
[highgo@thbytwo-centos2009 ~]$ touch ~/.pgpass
[highgo@thbytwo-centos2009 ~]$ chmod 0600 ~/.pgpass
[highgo@thbytwo-centos2009 ~]$ vim ~/.pgpass
```

其中 vim 编辑的内容为

```
*:*:*:*:Highgo@123（初始化数据库的密码）
```

复制 lib 库，将 192.168.31.21 上/usr/local/hgdb/lib 目录复制到/usr/local/hgdb/lib 目录，并为其赋权限和修改所有者

```
[highgo@localhost ~]$ ll
总用量 1800
-rw-r--r--. 1 root root 1841116 7月  16 15:08 lib.tar.gz
[highgo@localhost ~]$ tar -zxvf lib.tar.gz
lib/euc_jp_and_sjis.so
lib/libpq.so.5.12
...略...
lib/plpgsql.so
[highgo@localhost ~]$ ls
lib  lib.tar.gz
[highgo@localhost ~]$ exit
登出
[root@localhost test]# cd /usr/local/hgdb-ee-6.0.3/lib
[root@localhost highgosys]# ls
lib  lib.tar.gz
[root@localhost highgosys]# chmod -R 750 lib
[root@localhost highgosys]# chown -R highgo:highgo lib
```

(4) 继续修改配置文件~/.bash_profile

```
[root@localhost highgosys]# su - highgo
上一次登录：五 7月 16 13:55:24 CST 2021pts/0 上
[highgo@localhost ~]$ vim ~/.bash_profile
```

其中 vim 中的最终内容为

```
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/.local/bin:$HOME/bin
export PATH=$PATH:/usr/local/hgdb/bin
export LD_LIBRARY_PATH=/usr/local/hgdb/lib:$LD_LIBRARY_PATH
export PGDATA=/usr/local/hgdb/data
export PATH=/usr/local/db_ha/bin/:$PATH
```

使配置文件生效

```
[highgo@thbytwo-centos2009 ~]$ source ~/.bash_profile
```

修改目录的所有者和权限

```
[highgo@thbytwo-centos2009 ~]$ exit
登出
[root@thbytwo-centos2009 test]# chmod -R 750 /usr/local/db_ha
[root@thbytwo-centos2009 test]# chown -R highgo:highgo /usr/local/db_ha
```

(5) 创建一个目录用于存放 log

```
[root@localhost db_ha]# su - highgo
上一次登录：五 7 月 16 14:08:51 CST 2021pts/0 上
[highgo@localhost ~]$ pwd
/home/highgosys
[highgo@localhost ~]$ mkdir db_ha_log
[highgo@localhost ~]$ ls
db_ha_log  lib  lib.tar.gz
```

(6) 配置 primary_monitor 的配置文件

```
[highgo@localhost db_ha]$ cd /usr/local/db_ha/conf/
[highgo@localhost conf]$ ls
agent.conf  db_ha.conf  primary_monitor.conf  secondary_monitor.conf
[highgo@localhost conf]$ vim primary_monitor.conf
```

在 vim 中修改如下内容

```
#number of cluster nodes
block_repair_switch = on
block_repair_path_file = /usr/local/db_ha/log/special_block_repair.tmp
command_repeat_count = 5
daemon_server_port = 8000
monitor_cycle_time = 3
secondary_max_delay = 3
max_async_delay = 5
log_path = /usr/local/db_ha/log/monitor.log
replication_failure_standby_delete = on
failure_node_monitored = on
failure_node_polling_interval = 5
node_restart_times = 3
#user_input_switch can choose sql/shell/all/off
user_input_switch = off
user_input_sql = "none"
user_input_shell = "none"
new_node_streaming_replication_waiting_time = 60
method_sync = FIRST
num_sync = 2
log_level_info_file = LOG_INFO
log_level_on_screen = LOG_ERROR
db_restart_timeout = 10
db_rewind_timeout = 20
arbitration_judge = on
arbitration_ip = 192.168.31.40
#Node information
```

```
node1_nodetype = PRIMARY
node1_nodeip = 192.168.31.21
node1_dbname = highgo
node1_applicationname = node1
node1_username = highgo
node1_streaming = NONE
node1_healthy = t
node1_nodeport = 5432
node1_agentport = 6666
node1_vip1 = 192.168.31.24
node1_vip_num = 1
```

6.5.4 启动

(1) 启动 192.168.31.21 的 agent

```
[highgo@thbytwo-centos2009 ~]$ /usr/local/db_ha/bin/agent /usr/local/db_ha/conf/agent.conf
agent started successfully
```

(2) 启动 192.168.31.22 的 agent

```
[highgo@thbytwo-centos2009 ~]$ /usr/local/db_ha/bin/agent /usr/local/db_ha/conf/agent.conf
agent started successfully
```

(3) 启动 192.168.31.20 的 primary_monitor

```
[highgo@thbytwo-centos2009 conf]$
/usr/local/db_ha/bin/primary_monitor /usr/local/db_ha/conf/primary_monitor.conf
monitor process start success
[highgo@thbytwo-centos2009 conf]$ daemon process start success
```

(4) 启动 secondary_monitor

```
[highgo@thbytwo-centos2009 ~]$
/usr/local/db_ha/bin/secondary_monitor /usr/local/db_ha/conf/secondary_monitor.conf
[highgosys@thbytwo-centos2009 ~]$ secondary start success
```

6.5.5 验证安装部署

6.5.5.1 使用 db_ha 查看集群状态

```
[highgo@thbytwo-centos2009 ~]$
/usr/local/db_ha/bin/db_ha select -f /usr/local/db_ha/conf/db_ha.conf
connect monitor success
cluster num = 2        seconadary monitor is normal
nodeip=192.168.31.21,nodetype=RPIMARY,replicationName=node1            streamingType=NONE
streamingState=none healthy=t agentState=NORMAL
nodeip=192.168.31.22,nodetype=STANDBY,replicationName=node2           streamingType=ASYNC
streamingState=streaming healthy=t agentState=NORMAL
[highgosys@thbytwo-centos2009 ~]$
```

6.5.5.2 使用 db_ha 查看 VIP 绑定信息

```
[highgo@thbytwo-centos2009 ~]$ /usr/local/db_ha/bin/db_ha add 192.168.31.22 6666 -a -n
node3 -p 5432 -f /usr/local/db_ha/conf/db_ha.conf
connect monitor success
```

```
再次查询状态
[highgo@thbytwo-centos2009 ~]$
/usr/local/db_ha/bin/db_ha select -f /usr/local/db_ha/conf/db_ha.conf
connect monitor success
cluster num = 2        seconadary monitor is normal
nodeip=192.168.31.21,nodetype=RPIMARY,replicationName=node1            streamingType=NONE
streamingState=none healthy=t agentState=NORMAL
nodeip=192.168.31.22,nodetype=STANDBY,replicationName=node2           streamingType=ASYNC
streamingState=streaming healthy=t agentState=NORMAL
```

6.5.5.3 使用 db_ha 查看同异步信息

```
[highgo@thbytwo-centos2009 ~]$
/usr/local/db_ha/bin/db_ha select -f /usr/local/db_ha/conf/db_ha.conf
connect monitor success
cluster num = 2        seconadary monitor is normal
nodeip=192.168.31.21,nodetype=RPIMARY,replicationName=node1            streamingType=NONE
streamingState=none healthy=t agentState=NORMAL
nodeip=192.168.31.22,nodetype=STANDBY,replicationName=node2           streamingType=ASYNC
streamingState=streaming healthy=t agentState=NORMAL
[highgo@thbytwo-centos2009 ~]$ /usr/local/db_ha/bin/db_ha syncmode 192.168.31.22 -s sync -
f /usr/local/db_ha/conf/db_ha.conf
connect monitor success
[highgo@thbytwo-centos2009 ~]$
/usr/local/db_ha/bin/db_ha select -f /usr/local/db_ha/conf/db_ha.conf
connect monitor success
cluster num = 2        seconadary monitor is normal
nodeip=192.168.31.21,nodetype=RPIMARY,replicationName=node1            streamingType=NONE
streamingState=none healthy=t agentState=NORMAL
nodeip=192.168.31.22,nodetype=STANDBY,replicationName=node2            streamingType=SYNC
streamingState=streaming healthy=t agentState=NORMAL
```

6.6 使用一键部署安装示例

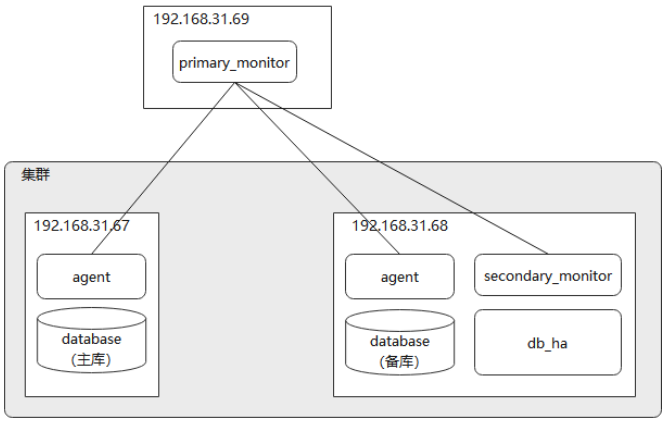
本示例共使用 3 台 Linux 机器，采用一般部署。假设数据库程序已经安装。2 台 Linux 作为数据节点，信息如下：

IP 地址	application_name	运行程序
192.168.31.21	node1	主数据库 agent
192.168.31.22	node2	备数据库 agent secondary_monitor

1 台用于运行 primary_monitor，信息如下：

IP 地址	运行程序
192.168.31.20	primary_monitor db_ha

拓扑结构图如下所示：



- **注意：**

1、所有节点均已经安装数据库软件，主节点需要 init 并启动。

2、一键安装是指自动完成流复制和高可用的配置。

下面进行步骤讲解。

6.6.1 在 192.168.31.21（主）上进行安装部署

（1）将此台服务器上的数据库进行初始化

```
[highgo@thbytwo-centos2009 ~]$ cd /usr/local/hgdb/bin
[highgo@thbytwo-centos2009 bin]$ /initdb -D /usr/local/hgdb/data/
```

（2）修改配置文件 pg_hba.conf

修改数据目录中的配置文件 pg_hba.conf，以使流复制不需要密码。例如：

```
# IPv4 local connections 项目下添加
host    all             all             192.168.31.0/24            trust
最后添加：
host    replication     highgo        192.168.31.0/24（配置对应网段即可）       trust
```

（3）安装 expect

```
[root@thbytwo-centos2009 ~]# yum install expect
```

（5）启动数据库

```
[highgo@thbytwo-centos2009 ~]# pg_ctl start
```

6.6.2 在 192.168.31.22（备）上进行安装部署

安装 expect

```
[root@thbytwo-centos2009 ~]# yum install expect
```

6.6.3 在 192.168.31.20（monitor）上进行安装部署

安装 expect

```
[root@thbytwo-centos2009 ~]# yum install expect
```

6.6.4 一键安装部署

（1）在执行一键安装的服务器上安装 sshpass

```
[root@thbytwo-centos2009 ~]# yum install sshpass
```

（2）上传一键部署包

```
[root@thbytwo-centos2009 ~ DbhaQuickInstall]#  pwd
/usr/local/DbhaQuickInstall
[root@thbytwo-centos2009 ~ DbhaQuickInstall]#  ls
agent.sh    db_ha1.8-se-centos7-x86_64.rpm    dbhaQuickInstall.sh    dbhaQuickInstall.conf
primary_monitor.sh  secondary_monitor.sh  set_ssh_trust.sh
```

（3）编辑参数配置文件

```
[root@thbytwo-centos2009 ~ DbhaQuickInstall]#  vi dbhaQuickInstall.conf
install_script_path = /usr/local/DbhaQuickInstall                          #安装脚本所在路
径
PGPASSWORD = Highgo@123                                                   #主节点数据库密码
DB_VERSION = ee                                                            #标准版 se，企业版 ee，安全
版 see
node1_password = 1324555                                                  #数据库节点 root 密码
node2_password = 1324555
pm_server_passwd = 1324555                                                #primary_monitor root 密码
sm_server_passwd = 1324555                                               #secondary_monitor root 密码
node1_network_card = ens33                                                #服务器 1 上用于绑定 vip 的网
卡
node2_network_card = ens33                                                #服务器 2 上用于绑定 vip 的网
卡
######################agent 配置#####################
listen_port = 6666                                                         #agent 的监听端口
local_pg_path = /usr/local/hgdb/data                                       #本机数据库的数据目录
#################primary_monitor 配置################
node_num = 3                                                               #集群节点个数
daemon_server_port = 8000                                                  #监听端口号
arbitration_judge = on                                                     #控制外部 ping 点开关
arbitration_ip = 192.168.31.40                                             #外 ping 的 ip，集群服务器
一定要能 ping 通的

#节点信息
node1_nodetype = PRIMARY              #节点类型：PRIMARY（主节点）、STANDBY（备节点）,X 表
示数字
node1_nodeip = 192.168.31.21                                               #节点 ip
node1_dbname = highgo                                                      #连接数据库用的 dbname
node1_applicationname = node1                                             #节点的复制名（用于流复制）
node1_username = highgo                                                    #连接节点用的数据库用户名
node1_streaming = NONE                                                     #节点的流复制类型（主节点
填 NONE，同步填 SYNC,异步填 ASYNC）
node1_nodeport = 5333                                                      #节点的数据库监听端口号
node1_agentport = 6666                                                     #节点的 agent 端口号
```

```
node1_healthy = t                                                    #节点状态 t 健康 f
不健康
node1_vip_num = 1                                                    #节点 VIP 个数
node1_vip1 = 192.168.31.23                                          #节点 1 的第 1 个 VIP

node2_nodetype = STANDBY                                             #节点类型：PRIMARY
（主节点）、STANDBY（备节点）,X 表示数字
node2_nodeip = 192.168.31.22                                        #节点 ip
node2_dbname = highgo                                               #连接数据库用的 dbname
node2_applicationname = node2                                       #节点的复制名（用于流复制）
node2_username = highgo                                            #连接节点用的数据库用户名
node2_streaming = SYNC                                               #节点的流复制类型（主节点
填 NONE，同步填 SYNC,异步填 ASYNC）
node2_nodeport = 5333                                               #节点的数据库监听端口号
node2_agentport = 6666                                              #节点的 agent 端口号
node2_healthy = t                                                    #节点状态 t 健康 f 不健康
node2_vip_num = 0
###############secondary_monitor 配置################
primary_monitor_ip = 192.168.31.20                                  #primary_monitor 的 IP 地址
primary_monitor_port = 8000                                          #primary_monitor 的监听端
口号
#####################db_ha 配置####################
secondary_monitor_ip = 192.168.31.22                          #secondary_monitor 程序的 IP 地址
```

（4）启动一键安装脚本：

```
[root@thbytwo-centos2009 ~ DbhaQuickInstall]#
./dbhaQuickInstall.sh    dbhaQuickInstall.conf    /usr/local/DbhaQuickInstall/db_ha1.8-ee-
centos7-x86_64.rpm
Begin to read config file paramForDbha, please wait......
Begin to set auth with db server
Host root not found in /root/.ssh/known_hosts
/root/.ssh/id_rsa already exists.
Overwrite (y/n)?          ------遇到这种情况直接回车跳过
.........
```

（5）安装完自动显示节点信息：

```
connect monitor success
cluster num = 2        seconadary monitor is normal
nodeip=192.168.31.21,nodetype=PRIMARY,replicationName=node1            streamingType=NONE
streamingState=none healthy=t agentState=NORMAL
nodeip=192.168.31.22,nodetype=STANDBY,replicationName=node2            streamingType=SYNC
streamingState=streaming healthy=t agentState=NORMAL
```

6.6.5 一键回滚

```
[root@thbytwo-centos2009 ~ DbhaQuickInstall]#
./rollback.sh dbhaQuickInstall.conf
Begin to read config file paramForDbha, please wait......
Begin to set auth with db server
Host root not found in /root/.ssh/known_hosts
/root/.ssh/id_rsa already exists.
Overwrite (y/n)?          ------遇到这种情况直接回车跳过
.........
```

6.7 在现有流复制集群环境搭建高可用

注意：不能使用一键安装部署

用户除了可以用 db_ha 添加节点搭建高可用外也可以把当前现有的流复制环境加入到高可用集群中。

假如用户已搭建了一主两备的流复制集群，并想把他们同时初始化进高可用集群 primary_monitor.conf 文件里必须把节点信息写全举例如下：

```
#number of cluster nodes
node_num = 3 #集群节点个数
block_repair_switch = on  #块修复开关
block_repair_path_file = /usr/local/db_ha/log/special_block_repair.tmp #块修复所需的文件名及其全路径
command_repeat_count = 5 #执行失败的操作重复执行数
daemon_server_port = 8000 #监听端口号
monitor_cycle_time = 3 #primary_monitor 循环一周期的时间
secondary_max_delay = 3#secondary_montior 最大延迟周期数
max_async_delay = 5 #异步流复制延迟最大值（超过此值报警）单位 M
log_path = /usr/local/db_ha/log          #日志文件路径 1.6 开始会自动生成日志文件
replication_failure_standby_delete = on  #流复制失败是否剔除集群
failure_node_monitored = on
failure_node_polling_interval = 5
node_restart_times = 3
#user_input_switch can choose sql/shell/all/off
user_input_switch = off
user_input_sql = empty
user_input_shell = empty
new_node_streaming_replication_waiting_time = 60
method_sync = FIRST  #控制 synchronous_standby_names 的同步方法
num_sync = 2         #控制 synchronous_standby_names 的同步个数
log_level_info_file = LOG_INFO   #日志等级达到 log_level_into_file 的会写入日志文件。
log_level_on_screen = LOG_ERROR  #日志等级达到 log_level_into_file 的会显示在命令行。
db_restart_timeout = 10          #控制重启数据库的超时时间
db_rewind_timeout = 20           #数据库 rewind 超时时间

#Node information
node1_nodetype = PRIMARY      #节点类型：PRIMARY（主节点）、STANDBY（备节点）,X 表示数字
node1_nodeip = 192.168.31.21  #节点 ip
node1_dbname = highgo         #连接数据库用的 dbname
node1_applicationname = node1 #节点的复制名（用于流复制）
node1_username = highgo       #连接节点用的数据库用户名
node1_streaming = NONE        #节点的流复制类型（主节点填 NONE，同步填 SYNC,异步填 ASYNC）
node1_nodeport = 5432         #节点的数据库监听端口号
node1_agentport = 6666        #节点的 agent 端口号
node1_healty = t              #节点状态 t 健康 f 不健康
node1_vip_num = 1             #节点 VIP 个数
node1_vip1 = 192.168.31.10    #节点 1 的第 1 个 VIP
node2_nodetype = STANDBY
node2_nodeip = 192.168.31.22
node2_nodeport = 5432
node2_dbname = highgo
node2_applicationname = node2
node2_username = highgo
node2_streaming = SYNC
node2_agentport = 6666
node2_vip_num = 0
node2_healthy = t
node3_nodetype = STANDBY
node3_nodeip = 192.168.31.23
node3_nodeport = 5432
node3_dbname = highgo
```

```
node3_applicationname = node3
node3_username = highgo
node3_streaming = SYNC
node3_agentport = 6666
node3_vip_num = 0
node3_healthy = t
```

在备节点上按照如下修改：

（1）执行 pg_basebackup 时指定了”-R“参数：需要关闭备节点上的数据库，在数据库 data 目录的 postgresql.auto.conf 文件中，修改字符串 primiary_conninfo，添加 application_name=XXX（其中：XXX 为该节 application_name。）

（2）执行 pg_basebackup 时没有指定”-R“参数：需要关闭备节点上数据库，在数据库 data 目录的 postgresql.auto.conf 文件中，添加 primary_conninfo 字符串。例如：primary_conninfo = 'user=postgres passfile=''/home/postgres/.pgpass'' host=192.168.31.67 port=5432 sslmode=disable sslcompression=0 gssencmode=disable krbsrvname=postgres target_session_attrs=any application_name= XXX'（其中：XXX 为该节点的 application_name。）

配置成功后启动各节点的 agent，然后再启动 primary_monitor、secondary_monitor 即可。

6.8 卸载高可用产品

卸载命令如下：

```
[root@host2 HighGosee]# rpm -qa | grep db_ha
 rpm -e db_ha4.0.2-ee-1.el7.x86_64

[root@host2 HighGosee]# rpm -e db_ha4.0.2-ee-1.el7.x86_64
```

6.9 已知限制条件

1. 不建议使用 kill -9 命令杀死数据库：kill -9 杀死的数据库重回集群需管理员手动将数据库启动，再通过 db_ha 命令加入集群；kill -9 方式还会导致服务恢复时间过长以及其他不可预知的问题。

2. V1.3 及以前版本的 synchronous_standby_names 不接受用户配置同步节点名称，由高可用软件 db_ha 统一自动管理，该版本设置的同步级别为“严格同步”，使用“synchronous_standby_names = First N (name1,name2,...nameN)”方式，确保数据的完整性和强一致性；一旦某个同步节点的流复制出现异常，在同步节点降为异步

节点之前，集群会被短暂 hang 住。

3. V1.4 及以后版本的 synchronous_standby_names 参数，接受用户手工配置，但只能通过 db_ha 工具进行修改。

4. 仅有两个节点的情况下，需配置 ping 点保障高可用正常工作。

5. V4.1.0 及以后版本不再需要区分 4 种部署模式。

6.10 数据块修复 db_blockguard

6.10.1 db_blockguard 简介

瀚高块修复工具（db_blockguard）是基于流复制环境建立的文件坏块扫描与修复软件，检测能力强大，对 checksums 正确的坏块也有一定的检测能力，能够有效的解决坏块修复问题。本工具支持瀚高系列数据库和 PostgreSQL（9.x-14 版本）。

瀚高块修复（db_blockguard）工具包含两大模块：

1. 核心功能模块
   以动态库形式提供坏块扫描、修复功能。
2. 工具模块
   以可执行程序形式为用户提供启动扫描修复、停止扫描修复、查看扫描修复当前进度信息等功能。

前提条件：

1. 保证客户机已经成功安装部署 PostgreSQL 数据库（不支持实时修复功能）或者瀚高数据库（支持实时修复功能）。
2. 本工具的使用需要流复制环境。
3. 本工具的使用要求 pg_checksum 为打开状态。

6.10.2 安装部署

6.10.2.1 安装块修复工具 RPM 包

注意：
所有开启块修复的节点（包括本地节点和远程节点）均需要安装块修复工具 rpm 包。

```
rpm -ivh db_blockguard-1.4-1.el7.x86_64.rpm
```

安装完成后会在/usr/local 路径下生成 db_blockguard 文件夹，

```
[root@host1 db_blockguard]# pwd
/usr/local/db_blockguard
[root@host1 db_blockguard]# ls
bin  cfg  dat
```

其中:

- /usr/local/db_blockguard/bin

  该目录用来存放工具程序 db_blockguard_tool 和动态库 db_blockguard.so，无需改动。

- /usr/local/db_blockguard/cfg

  该目录用来存放配置文件 db_blockguard.conf，需用户配置。

- /usr/local/db_blockguard/dat

  该目录用来存放产生的统计数据文件 db_blockguard.dat，安装 rpm 包不会生成此文件，执行 init 命令初始化后才会自动产生。

6.10.2.2 配置 db_blockguard.conf 文件

1. /usr/local/db_blockguard/cfg 路径下的 db_blockguard.conf 文件用来存储本地节点和远程节点的信息，包括:

- **本地节点**：ip/port/dbname/user
- **远程节点**：只需要配置 ip/port，不需要配置 dbname/username（因为流复制环境中与本地节点一致）。

2. 主节点和备节点的 db_blockguard.conf 文件配置略有不同：

- 如果当前节点是主节点，则远程节点为备节点，这时可以设置一个或多个远程节点：

```
local_host=localhost port=5438 user=postgres dbname=postgres
remote_host=192.168.31.142 port=5444
remote_host=192.168.31.143 port=5445
```

- 如果当前节点是备节点，则远程节点只能是主节点，所以只能设置一个远程节点：

```
local_host=localhost port=5444 user=postgres dbname=postgres
remote_host=192.168.31.141 port=5438
```

6.10.2.3 配置~/.pgpass 文件

用户需要配置~/.pgpass 文件（其权限应为 0600）来保存密码（以下以上述主节点为例）：

```
#hostname:port:database:username:password
localhost:5438:postgres:postgres:postgres
```

6.10.2.4 打开 pg_checksum

本工具的使用以 pg_checksum 打开为前提，否则执行 db_blockguard_tool 命令时会报错：

```
[root@host2 bin]# ./db_blockguard_tool init
Please turn on data checksums for this cluster first!
Initialize failed
```

1. 用以下命令查看 pg_checksum 是否处于打开状态：

```
pg_controldata -D ../data | grep checksum
```

说明：

这里的 ../data 是 PG 的数据目录，请按实际情况输入。

Data page checksum version 为 1，pg_checksum 为打开状态；

```
[postgres@host2 bin]$ ./pg_controldata -D ../data | grep checksum
Data page checksum version:           1
```

Data page checksum version 为 0，pg_checksum 为关闭状态；

```
[postgres@host2 bin]$ ./pg_controldata -D ../data | grep checksum
Data page checksum version:           0
```

2. 用以下命令打开 pg_checksum：

```
pg_ctl stop
pg_checksums -e -D ../data
```

```
[postgres@host2 bin]$ ./pg_checksums -e -D ../data
Checksum operation completed
Files scanned:  916
Blocks scanned: 2967
pg_checksums: syncing data directory
pg_checksums: updating control file
Checksums enabled in cluster
```

也可以在执行 initdb 时指定 -k 或 --data-checksums 参数开启 checksums。

需要注意的是，打开 PG 的 checksums 功能需要一定的时间，取决于数据库文件的多少及大小。

6.10.3 主要功能使用说明

本工具提供的所有功能（实时修复功能除外）都在/usr/local/db_blockguard/bin 目录下通过执行 db_blockguard_tool 命令来实现：

```
[root@x86-0001 bin]# ./db_blockguard_tool
An option is needed!
        init            Initialize the block guard
        start           Start scanning and repair bad blocks
        stop            Stop the runing block guard
        list            List block guard statistics info
        destroy         Delete all functions created when init,only for primary node
```

6.10.3.1 块修复工具初始化（init）

```
db_blockguard_tool init
```

使用本工具进行块扫描修复之前，**所有相关节点都必须初始化**，包括本地节点和远程节点。特别的，**主节点必须初始化**。

```
[root@host1 bin]# ./db_blockguard_tool  init
Initialized successfully
```

主节点和备节点初始化成功都会如上图所示，但主备节点初始化的结果略有不同：

- 备节点初始化成功之后，会在/usr/local/db_blockguard/dat 目录下生成 db_blockguard.dat 文件：

```
[root@host1 dat]# pwd
/usr/local/db_blockguard/dat
[root@host1 dat]# ls
db_blockguard.dat
```

- 主节点初始化成功之后，除了会在/usr/local/db_blockguard/dat 目录下生成 db_blockguard.dat 文件之外，还会在数据库中创建以下三个函数：

```
postgres=# \df
 Schema |       Name       | Result data type |
--------+------------------+------------------+
 public | blockguard_fetch | cstring          |
 public | blockguard_lsn   | pg_lsn           |
 public | blockguard_start | cstring          |
(3 rows)
```

6.10.3.2 启动单次块扫描修复任务（start）

```
db_blockguard_tool start
```

也可以在 start 后面加上&符号使扫描修复在后台执行。

6.10.3.2.1 扫描修复成功的例子

假设有一坏块 tb2，扫描修复前查询结果是这样的：

```
postgres=# select * from tb2;
WARNING:  page verification failed, calculated checksum 17258 but expected 19508
ERROR:  invalid page in block 0 of relation base/13578/24579
```

扫描修复成功是这样的：

```
[root@host2 bin]# ./db_blockguard_tool start
WARNING:  page verification failed, calculated checksum 17258 but expected 19508
WARNING:  invalid page in block 0 of relation base/13578/24579; zeroing out page
block scan completed.
```

使用 db_blockguard_tool list 命令（1.2.2）查询可以看到刚刚扫描修复的结果：

```
status=inactive
pid=346340
start_time=Mon Oct 11 16:20:30 2021
blk_found=1
blk_fixed=1
end_time=Mon Oct 11 16:20:30 2021
```

再次查询可以发现，坏块确实已被修复：

```
postgres=# select * from tb2;
 name
------
 6666
 5555
(2 rows)
```

6.10.3.2.2 扫描修复失败的例子

1.执行 start 之前本地节点没有 init：

```
[root@host2 bin]# ./db_blockguard_tool start
open source /usr/local/db_blockguard/dat/db_blockguard.dat failed!Has the init function been
performed?
Start failed
```

2.执行 start 之前主节点没有 init：

```
[root@host2 bin]# ./db_blockguard_tool start
SELECT failed: ERROR:  function public.blockguard_start() does not exist
LINE 1: select public.blockguard_start();
               ^
HINT:  No function matches the given name and argument types. You might need to add explicit
type casts.

Has the init function been performed on the primary?
Start failed
```

3. 已经有进行中的扫描修复任务：

```
[root@host2 bin]# ./db_blockguard_tool start
another blockguard is in progress!Start is aborted!
Start failed
```

6.10.3.3 停止块扫描修复（stop）

```
db_blockguard_tool stop
```

1. 停止进行中的块扫描任务：

```
[root@host2 bin]# ./db_blockguard_tool stop
send SIGUSR1 to pid:376358!
Stopped successfully
```

2. 没有进行中的块扫描任务：

```
[root@host2 bin]# ./db_blockguard_tool stop
blockguard is not in progress!
Stop failed
```

6.10.3.4 查看扫描统计信息（list）

```
db_blockguard_tool list
```

此命令可以查看所有扫描的统计信息，包括当前任务信息和历史任务信息，每条任务信息包括 6 行：

（1）第一行是状态信息，inactive 表示已完成的历史任务，active 则表示正在进行的任务。

（2）第二行是扫描任务的进程 ID。

（3）第三行是任务启动时间。

（4）第四行是发现的坏块数量。

（5）第五行是修复的坏块数量。

（6）第六行是任务结束时间。

1. 下图例子中，一共有 3 条扫描记录信息：

✓ 记录 1 为历史任务，发现 1 个坏块，修复 1 个坏块；

✓ 记录 2 为历史任务，发现 1 个坏块，修复 0 个坏块；

✓ 记录 3 为正在进行的任务，目前发现 0 个坏块，修复 0 个坏块。

```
[root@host2 bin]# ./db_blockguard_tool list
status=inactive
pid=371933
start_time=Tue Oct 12 10:27:14 2021
blk_found=1
blk_fixed=1
end_time=Tue Oct 12 10:27:14 2021

status=inactive
pid=372310
start_time=Tue Oct 12 10:42:28 2021
blk_found=1
blk_fixed=0
end_time=Tue Oct 12 10:42:28 2021

status=active
pid=376473
start_time=Tue Oct 12 12:53:05 2021
blk_found=0
blk_fixed=0
end_time=0

All information listed
```

1

2

3

2. 如果 init 之后还没有进行扫描修复：

```
[root@host2 bin]# ./db_blockguard_tool list
No blockguard info!
List failed
```

3. 如果没有 init:

```
[root@host2 bin]# ./db_blockguard_tool list
open source /usr/local/db_blockguard/dat/db_blockguard.dat failed!
List failed
```

6.10.3.5 删除块修复工具函数（destroy）

当块修复工具版本升级时，需要先卸载旧的 rpm 包，然后再安装新版本的 rpm 包。

但是卸载时使用 rpm -e 命令无法删除块修复工具初始化时创建的三个函数，所以，需要使用 destroy 命令将它们删除，然后再用 rpm -e 命令卸载块修复 rpm 包。

```
db_blockguard_tool destroy
```

注意：

执行 destroy 之前，应先将所有节点上正在进行的扫描修复任务停止或等待其结束，否则可能会有不可预知的问题。

```
[root@x86-0001 bin]# ./db_blockguard_tool destroy
All functions have been destroyed successfully!
Destroyed successfully
```

6.10.3.6 定时扫描功能

本工具支持通过 crontab 工具来创建定时运行的扫描任务。

例如，可通过以下命令来实现每周六的 1:10 启动扫描：

```
10  1  *  *  6  /usr/local/db_blockguard/db_blockguard_tool start
```

6.10.3.7 实时修复功能

如果客户机安装的是瀚高数据库，则可以通过设置 postgresql.conf 配置文件中的 real_time_blockguard 参数来启用/停止实时修复功能。

real_time_blockguard 参数默认值为 off，即默认实时修复功能是关闭的。

6.10.3.7.1 启用实时修复功能

```
real_time_blockguard = on
```

✓ 实时修复成功：

```
postgres=# select * from tb2;
WARNING:  page verification failed, calculated checksum 26768 but expected 30002
WARNING:  invalid page in block 0 of relation base/13578/24579; already fixed
 name
------
 6666
 5555
(2 rows)
```

✓ 实时修复失败：

```
postgres=# select * from tb1;
WARNING:  page verification failed, calculated checksum 16557 but expected 62768
WARNING:  repair_block: could not receive block from remote node: ERROR:  invalid
page in block 0 of relation base/13578/16390
ERROR:  invalid page in block 0 of relation base/13578/16390
```

6.10.3.7.2 关闭实时修复功能

```
real_time_blockguard = off
```

这种情况下坏块是不能实时修复的：

```
postgres=# select * from tb1;
WARNING:  page verification failed, calculated checksum 56616 but expected 2600
ERROR:  invalid page in block 0 of relation base/13578/16390
```

可以通过手动方式（1.2.2）来修复。

6.10.4 db_blockguard 工具已知限制条件

1. 如果节点断电重启，或者扫描任务进程被 kill，则可能导致统计数据文件中的最新任务信息无法得到更新，重新启动新的任务时工具程序认为当前任务未结束而拒绝启动新任务。这种情况需要将统计数据文件删除或者更名，然后重新执行初始化操作。

2. 由于安全版的加密限制，目前暂不支持瀚高数据库安全版。

6.11 同步流复制增强

7 备份恢复 db_backup

7.1 db_backup 简介

db_backup 备份恢复工具主要针对于 PostgreSQL 数据库以及瀚高数据库等数据库的备份和使用。

前提条件：
保证客户机已经成功安装部署 PostgreSQL 数据库或者瀚高数据库。

术语解释：
数据库服务器: 主要包括 瀚高数据库和 PostgreSQL 数据库

7.2 安装部署

前提保证客户机已经成功安装部署 PostgreSQL9 以上数据库或者瀚高数据库。

7.2.1 安装 RPM 包

```
rpm -ivh db_backup-1.1-1.el7.x86_64.rpm  --nodeps --force
```

会在/usr/local/db_backup 目录下生成 db_backup 备份恢复工具。
（此安装包仅适用于 centos7 及以下版本的 linux）

7.2.2 创建备份目录和归档目录

备份目录创建：

```
    mkdir -p /data/db_backup
```

对文件夹赋权：

```
chown -R postgres:postgres  /data
```

归档目录创建：

```
mkdir  -p  /archive_log
```

进行文件夹赋权：

```
chown postgres:postgres /archive_log
```

7.2.3 开启归档

以 postgres 用户身份打开$PGDATA 目录下的 postgresql.conf 配置文件,配置完成后重启数据库，生效。

```
# - Archiving -

archive_mode = on               # enables archiving; off, on, or always
                                # (change requires restart)
archive_command = ' cp %p /archive_log/%f'              # command to use to archive a logfile segment
                                # placeholders: %p = path of file to archive
                                #               %f = file name only
                                # e.g. 'test ! -f /mnt/server/archivedir/%f && cp %p /mnt/server/archivedir/%f'
#archive_timeout = 0            # force a logfile segment switch after this
                                # number of seconds; 0 disables
```

配置文件说明：

开启归档模式；

将归档的 wal 日志拷贝到指定的的归档目录下。

7.2.4 配置用户环境变量(默认$PGDATA 的路径是/usr/pgsql-12)

```
vim  ~/.bash_profile
export BACKUP_PATH=/data/db_backup
export PATH=/usr/pgsql-12/bin:$PATH:/usr/local/db_backup
export LD_LIBRARY_PATH=/usr/pgsql-12/lib
export PGDATA=/usr/pgsql-12/data
配置完环境变量后
source ~/.bash_profile
```

7.2.5 初始化备份目录

在使用工具之前需要指定备份目录并初始化。

```
db_backup init -B /data/db_backup  -D /usr/pgsql-12/data
```

```
[postgres@localhost ~]$ db_backup init -B /data/db_backup  -D /usr/pgsql-12/data
INFO: ARCLOG_PATH is set to '/archive_log'
INFO: SRVLOG_PATH is set to '/usr/pgsql-12/data/log'
```

初始化备份目录主要生成一些相关文件：

```
[postgres@localhost db_backup]$ ls
20210323  backup  db_backup.ini  system_identifier  timeline_history
[postgres@localhost db_backup]$
```

20210323 文件夹：该目录下存储着备份文件的相关信息，以备份开始的时间作为目录结构。

backup 文件夹：保存服务器日志和 wal 日志。

db_backup.ini：文件中保存了归档日志的路径，以及可以实行的备份策略。

system_identifier：数据库系统标识符，在连接数据库的时候需要进行校验，防止数据混乱。

timeline_history：里面存储着时间线的历史文件。

7.3 主要功能使用说明

7.3.1 全量备份

db_backup backup -b full

```
[postgres@localhost local]$ db_backup  backup -b full
INFO: copying database files
INFO: copying archived WAL files
INFO: The current timeline does not have a full backup
INFO: backup complete
INFO: Please execute 'db_backup validate' to verify the files are correctly copied.
[postgres@localhost local]$ db_backup show
========================================================================
 StartTime           EndTime             Mode    Size   TLI  Status
========================================================================
2021-10-19 16:41:44  2021-10-19 16:41:50  FULL    56MB     1  DONE
```

7.3.2 增量备份(默认 0 级增量备份)

0 级增量备份

db_backup backup -b incremental --level=0

```
[postgres@localhost local]$ db_backup backup -b incremental --level=0
INFO: copying database files
INFO: copying archived WAL files
INFO: backup complete
INFO: Please execute 'db_backup validate' to verify the files are correctly copied.
[postgres@localhost local]$ db_backup show
========================================================================
 StartTime           EndTime             Mode    Size   TLI  Status
========================================================================
2021-10-19 16:43:02  2021-10-19 16:43:05  0INC    33MB     1  DONE
2021-10-19 16:41:44  2021-10-19 16:41:50  FULL    56MB     1  OK
```

1 级增量备份

db_backup backup -b incremental --level=1（一定在前一个增量备份验证后在操作，不然是错误的）

错误情况（上次增量备份未验证）：如果之前有两次增量备份，最近的一次增量没有验证，那么它会根据第一次增量备份进行增量备份。

```
[postgres@localhost local]$ db_backup backup -b incremental --level=1
INFO: copying database files
ERROR: cannot take an incremental backup
DETAIL: There is no validated Incremental backup with current timeline.
HINT: Please take a Incremental backup and validate it before doing an incremental backup.
[postgres@localhost local]$ db_backup show
========================================================================
 StartTime           EndTime             Mode    Size   TLI  Status
========================================================================
2021-10-19 16:43:42  2021-10-19 16:43:42  1INC      0B     1  ERROR
2021-10-19 16:43:02  2021-10-19 16:43:05  0INC    33MB     1  DONE
2021-10-19 16:41:44  2021-10-19 16:41:50  FULL    56MB     1  OK
```

正确情况：

```
[postgres@localhost local]$ db_backup backup -b incremental --level=1
INFO: copying database files
INFO: copying archived WAL files
INFO: backup complete
INFO: Please execute 'db_backup validate' to verify the files are correctly copied.
[postgres@localhost local]$ db_backup show
========================================================================
 StartTime           EndTime             Mode    Size   TLI  Status
========================================================================
2021-10-19 16:45:05  2021-10-19 16:45:07  1INC    50MB     1  DONE
2021-10-19 16:43:42  2021-10-19 16:43:42  1INC      0B     1  ERROR
2021-10-19 16:43:02  2021-10-19 16:43:05  0INC    33MB     1  OK
2021-10-19 16:41:44  2021-10-19 16:41:50  FULL    56MB     1  OK
```

7.3.3 备份查看

db_backup show

```
[postgres@localhost local]$ db_backup backup -b incremental --level=1
INFO: copying database files
INFO: copying archived WAL files
INFO: backup complete
INFO: Please execute 'db_backup validate' to verify the files are correctly copied.
[postgres@localhost local]$ db_backup show
========================================================================
 StartTime           EndTime             Mode    Size   TLI  Status
========================================================================
2021-10-19 16:45:05  2021-10-19 16:45:07  1INC    50MB     1  DONE
2021-10-19 16:43:42  2021-10-19 16:43:42  1INC      0B     1  ERROR
2021-10-19 16:43:02  2021-10-19 16:43:05  0INC    33MB     1  OK
2021-10-19 16:41:44  2021-10-19 16:41:50  FULL    56MB     1  OK
```

7.3.4 备份验证

db_backup validate

```
[postgres@localhost local]$ db_backup show
========================================================================
 StartTime           EndTime             Mode    Size   TLI  Status
========================================================================
2021-10-19 16:45:05  2021-10-19 16:45:07  1INC    50MB     1  DONE
2021-10-19 16:43:42  2021-10-19 16:43:42  1INC      0B     1  ERROR
2021-10-19 16:43:02  2021-10-19 16:43:05  0INC    33MB     1  OK
2021-10-19 16:41:44  2021-10-19 16:41:50  FULL    56MB     1  OK
[postgres@localhost local]$ db_backup validate
INFO: validate: "2021-10-19 16:45:05" backup and archive log files by CRC
INFO: backup "2021-10-19 16:45:05" is valid
[postgres@localhost local]$ db_backup show
========================================================================
 StartTime           EndTime             Mode    Size   TLI  Status
========================================================================
2021-10-19 16:45:05  2021-10-19 16:45:07  1INC    50MB     1  OK
2021-10-19 16:43:42  2021-10-19 16:43:42  1INC      0B     1  ERROR
2021-10-19 16:43:02  2021-10-19 16:43:05  0INC    33MB     1  OK
2021-10-19 16:41:44  2021-10-19 16:41:50  FULL    56MB     1  OK
[postgres@localhost local]$
```

7.3.5 备份恢复（先关闭数据库）

db_backup restore
如果不指定-D 指定恢复目录，默认恢复到原来的$PGDATA 目录下。
恢复也可以指定参数，例如时间，时间线，事务 id 等。
默认恢复为例：

```
[postgres@localhost local]$ pg_ctl stop
waiting for server to shut down.... done
server stopped
[postgres@localhost local]$ db_backup restore
INFO: the recovery target timeline ID is not given
INFO: use timeline ID of current database cluster as recovery target: 1
INFO: calculating timeline branches to be used to recovery target point
INFO: searching latest full backup which can be used as restore start point
INFO: found the full backup can be used as base in recovery: "2021-10-19 16:41:44"
INFO: copying online WAL files and server log files
INFO: clearing restore destination
INFO: validate: "2021-10-19 16:41:44" backup and archive log files by SIZE
INFO: backup "2021-10-19 16:41:44" is valid
INFO: restoring database files from the full mode backup "2021-10-19 16:41:44"
INFO: Start thread 1
INFO: theads 1- Backup files are restored
INFO: Backup files are restored.
INFO: searching incremental backup to be restored
INFO: validate: "2021-10-19 16:43:02" backup and archive log files by SIZE
INFO: backup "2021-10-19 16:43:02" is valid
INFO: restoring database files from the incremental mode backup "2021-10-19 16:43:02"
INFO: Start thread 1
INFO: theads 1- Backup files are restored
INFO: Backup files are restored.
INFO: validate: "2021-10-19 16:45:05" backup and archive log files by SIZE
INFO: backup "2021-10-19 16:45:05" is valid
INFO: restoring database files from the incremental mode backup "2021-10-19 16:45:05"
INFO: Start thread 1
INFO: theads 1- Backup files are restored
INFO: Backup files are restored.
INFO: searching backup which contained archived WAL files to be restored
INFO: backup "2021-10-19 16:45:05" is valid
INFO: restoring WAL files from backup "2021-10-19 16:45:05"
INFO: restoring online WAL files and server log files
INFO: add recovery related options to postgresql.conf
INFO: generating recovery.signal
INFO: removing standby.signal if exists to restore as primary
INFO: restore complete
HINT: Recovery will start automatically when the PostgreSQL server is started.
[postgres@localhost local]$
```

以选定时间点恢复为例：
目前备份集中包含以下时间点的备份：

```
[postgres@localhost local]$ db_backup show
========================================================================
 StartTime           EndTime             Mode    Size   TLI  Status
========================================================================
2021-10-19 16:45:05  2021-10-19 16:45:07  1INC    50MB     1  OK
2021-10-19 16:43:02  2021-10-19 16:43:05  0INC    33MB     1  OK
2021-10-19 16:41:44  2021-10-19 16:41:50  FULL    56MB     1  OK
```

然后数据库宕机，想恢复到 2021-10-19 16:43:02 时间点：
那么我们需要关闭数据库后，执行
db_backup restore --recovery-target-time “2021-10-19 16:43:02”

```
[postgres@localhost local]$ db_backup restore  --recovery-target-time  "2021-10-19 16:43:02"
INFO: the recovery target timeline ID is not given
INFO: use timeline ID of current database cluster as recovery target: 1
INFO: calculating timeline branches to be used to recovery target point
INFO: searching latest full backup which can be used as restore start point
INFO: found the full backup can be used as base in recovery: "2021-10-19 16:41:44"
INFO: copying online WAL files and server log files
INFO: clearing restore destination
INFO: validate: "2021-10-19 16:41:44" backup and archive log files by SIZE
INFO: backup "2021-10-19 16:41:44" is valid
INFO: restoring database files from the full mode backup "2021-10-19 16:41:44"
INFO: Start thread 1
INFO: theads 1- Backup files are restored
INFO: Backup files are restored.
INFO: searching incremental backup to be restored
INFO: searching backup which contained archived WAL files to be restored
INFO: backup "2021-10-19 16:41:44" is valid
INFO: restoring WAL files from backup "2021-10-19 16:41:44"
INFO: backup "2021-10-19 16:43:02" is valid
INFO: restoring WAL files from backup "2021-10-19 16:43:02"
INFO: backup "2021-10-19 16:45:05" is valid
INFO: restoring WAL files from backup "2021-10-19 16:45:05"
INFO: restoring online WAL files and server log files
INFO: add recovery related options to postgresql.conf
INFO: generating recovery.signal
INFO: removing standby.signal if exists to restore as primary
INFO: restore complete
HINT: Recovery will start automatically when the PostgreSQL server is started.
```

然后启动数据库,就完成了相应的数据恢复

```
[postgres@localhost local]$ pg_ctl start
waiting for server to start....2021-10-19 16:55:02.985 CST [2812] LOG:  starting PostgreSQL 13.4 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 4.8.5 20150623 (Red Hat 4
.8.5-44), 64-bit
2021-10-19 16:55:03.079 CST [2812] LOG:  listening on IPv6 address "::1", port 5432
2021-10-19 16:55:03.079 CST [2812] LOG:  listening on IPv4 address "127.0.0.1", port 5432
2021-10-19 16:55:03.081 CST [2812] LOG:  listening on Unix socket "/tmp/.s.PGSQL.5432"
2021-10-19 16:55:03.085 CST [2813] LOG:  database system was interrupted; last known up at 2021-10-19 16:41:45 CST
2021-10-19 16:55:03.098 CST [2813] LOG:  starting point-in-time recovery to 2021-10-19 16:43:02+08
2021-10-19 16:55:03.116 CST [2813] LOG:  restored log file "00000001000000000000000F" from archive
2021-10-19 16:55:03.314 CST [2813] LOG:  redo starts at 0/F000060
2021-10-19 16:55:03.315 CST [2813] LOG:  consistent recovery state reached at 0/F000138
2021-10-19 16:55:03.315 CST [2812] LOG:  database system is ready to accept read only connections
2021-10-19 16:55:03.336 CST [2813] LOG:  restored log file "000000010000000000000010" from archive
 done
server started
[postgres@localhost local]$ 2021-10-19 16:55:03.395 CST [2813] LOG:  restored log file "000000010000000000000011" from archive
2021-10-19 16:55:03.572 CST [2813] LOG:  restored log file "000000010000000000000012" from archive
2021-10-19 16:55:03.666 CST [2813] LOG:  recovery stopping before commit of transaction 22144, time 2021-10-19 16:43:05.031499+08
2021-10-19 16:55:03.666 CST [2813] LOG:  pausing at the end of recovery
2021-10-19 16:55:03.666 CST [2813] HINT:  Execute pg_wal_replay_resume() to promote.
```

数据库重启成功后
psql 进入到命令行中

```
test=# select pg_wal_replay_resume();
 pg_wal_replay_resume
----------------------

(1 行记录)

test=#
```

在 sql 命令行输入 select pg_wal_replay_resume();命令。使数据库从只读模式中恢复。
有时候启动后会弹出一些信息，可以忽略，并不影响我们的数据。

7.3.6 备份删除

db_backup delete 2021-03-19 09:00:06
删除此时间点之前的全量备份之前的所有的备份。
会保留能够进行一次完整恢复的所有备份。

```
[postgres@localhost local]$ db_backup show
========================================================================
 StartTime           EndTime             Mode    Size   TLI  Status
========================================================================
2021-10-19 16:56:03  2021-10-19 16:56:05  FULL   190MB     2  OK
2021-10-19 16:55:44  2021-10-19 16:55:44  FULL     0B      0  ERROR
2021-10-19 16:45:05  2021-10-19 16:45:07  1INC    50MB     1  OK
2021-10-19 16:43:02  2021-10-19 16:43:05  0INC    33MB     1  OK
2021-10-19 16:41:44  2021-10-19 16:41:50  FULL    56MB     1  OK
[postgres@localhost local]$ db_backup delete 2021-10-19 16:56:05
WARNING: cannot delete backup with start time "2021-10-19 16:56:03"
DETAIL: This is the latest full backup necessary for successful recovery.
INFO: delete the backup with start time: "2021-10-19 16:55:44"
INFO: delete the backup with start time: "2021-10-19 16:45:05"
INFO: delete the backup with start time: "2021-10-19 16:43:02"
INFO: delete the backup with start time: "2021-10-19 16:41:44"
[postgres@localhost local]$ db_backup show
========================================================================
 StartTime           EndTime             Mode    Size   TLI  Status
========================================================================
2021-10-19 16:56:03  2021-10-19 16:56:05  FULL   190MB     2  OK
```

7.3.7 备份物理删除

db_backup purge
删除已经删除的备份的备份目录。

```
[postgres@localhost local]$ db_backup purge
INFO: DELETED backup "2021-10-19 16:55:44" is purged
INFO: DELETED backup "2021-10-19 16:45:05" is purged
INFO: DELETED backup "2021-10-19 16:43:42" is purged
INFO: DELETED backup "2021-10-19 16:43:02" is purged
INFO: DELETED backup "2021-10-19 16:41:44" is purged
[postgres@localhost local]$
```

7.4 关于备份恢复的一些其他参数

可用 db_backup --help 查看相关命令参数。

backup 命令支持的 OPTION 参数如下：

-B, --backup-path=PATH	备份文件将会保存到此目录
-d, --dbname=DBNAME	备份连接的数据库名称
-b, --backup-mode=MODE	备份模式 full, incremental, or archive
-Z, --compress-data	备份数据是否压缩
-C, --smooth-checkpoint	是否执行 smooth checkpoint
-F, --full-backup-on-error	增备模式时，当前时间线没有找到合适的全备备份，将此次备份转为全备备份(只适用于不开启 BCT 备份的情况)
--standby-host=HOSTNAME	从备机备份时，备机的 IP
--standby-port=PORT	从备机备份时，备机的端口
--level	增量备份的 0 级或者 1 级

restore 命令支持的 OPTION 参数如下：

--recovery-target-time 恢复到的时间
--recovery-target-xid 恢复到的 xid
--recovery-target-inclusive 是否包含条件项
--recovery-target-timeline 目标时间线

show 命令支持的 OPTION 参数如下：

-a, --show-all deleted 的备份集也可显示

delete 命令支持的 OPTION 参数如下：

--keep-data-generations=NUM (保留策略)保留全备的数量,仅 delete backup 使用
--keep-data-days=NUM (保留策略)备份保留的天数,仅 delete backup 使用
--keep-arclog-files=NUM (保留策略)保留归档文件的数量,仅 delete arclog 使用
--keep-arclog-days=DAY (保留策略)归档文件保存的天数,仅 delete arclog 使用

7.5 db_backup 已知限制条件

1）需要读取数据库集群目录和写入备份目录目录。
2）db_backup 和数据库的块大小应该匹配。BLCKSZ 和 XLOG_BLCKSZ 也应匹配。（8KB）
3）如果数据库集群目录，WAL 目录或已归档的 WAL 目录中有一些不可读的文件/目录，则备份或还原将失败。
4）您不能同时在主数据库和备用数据库上获得备份。
5）您也无法同时在多个备用数据库上获得备份。

8 双向数据复制 dbrep

8.1 dbrep 简介

双向数据复制 dbrep 解决数据库间数据共享，实现数据库表级别的数据一致；基于数据库 WAL(Write Ahead Logging)日志，提取结构化数据，解析出数据库中数据的增、删、改等变更，再将这些变更应用到对端数据库，使两端的数据库实现双向数据同步，以达到数据库站点间数据一致，并防止数据重复发送。

双向数据复制是基于数据库内核开发的功能，部署双向数据复制产品需要使用瀚高企业版提供的数据库安装包。

以下演示的安装过程基于两台服务器：服务器 A 和服务器 B，服务器上安装 Linux 操作系统(CentOS7)，服务器上安装的瀚高企业版数据库分别称为数据库 A 和数据库 B。

8.2 准备工作

1、关闭防火墙

```
systemctl stop firewalld.service
systemctl disable firewalld.service
systemctl status firewalld.service
```

2、关闭 SELINUX

```
# sed -i "s/SELINUX=enforcing/SELINUX=disabled/g"  /etc/selinux/config
# setenforce 0
# cat /etc/selinux/config | grep SELINUX=disabled
# getenforce
```

8.3 配置相关参数

修改数据库配置文件 postgresql.conf

```
vi postgresql.conf
listen_addresses = '*'
wal_level = logical
hg_enable_duplex_copy= on
```

8.4 双向复制部署

Note: 在部署数据复制产品时，请确保两端数据库上已经创建需要复制的表，且表在相同的 schema 下。

8.4.1 发布相应的表

8.4.1.1 数据库 A 发布

登录数据库 A

Note：若没有表，请先建立相应的表

```
psql
highgo=# create publication pub_name_A for table table_name;
```

Note: pub_name_A 为 数据库 A 上发布的名字
table_name 表名，多个表时用分号隔开

例子：

```
highgo=#create publication pub_83 for table hhh;
```

查看相应的发布

```
highgo=#select * from pg_publication;
 oid | pubname | pubowner | puballtables | pubinsert | pubupdate | pubdelete | pubtruncate
-----+---------+----------+--------------+-----------+-----------+-----------+-------------+-------------
24600 pub_83            10               f           t               t               t           t
(1 rows)
```

记住 publication 的名字 pub_name_A, 用于对方订阅时使用

8.4.1.2 数据库 B 发布

登录 数据库 B

Note：若没有表，请先建立相应的表（和 数据库 A 一致）

```
psql
highgo=# create publication pub_name_B for table table_name;
```

Note: pub_name_B 为 数据库 B 上发布的名字
table_name 表名，多个表时用分号隔开，和数据库 A 保持一致

例子：

```
highgo=#create publication pub_84 for table hhh;
```

查看相应的发布

```
highgo=#select * from pg_publication;
 oid | pubname | pubowner | puballtables | pubinsert | pubupdate | pubdelete | pubtruncate
-----+---------+----------+--------------+-----------+-----------+-----------+-------------+-------------
24600 pub_84          10          f          t          t          t          t
(1 rows)
```

✧ 记住 publication 的名字 pub_name_B, 用于对方订阅时使用。

8.4.2 订阅相应的表

8.4.2.1 数据库 A 订阅数据库 B

登录 数据库 A

创建订阅

```
highgo=#create subscription sub_name_A connection 'host=ip port=port_number user=user_name password=password dbname=database_name' publication pub_name_B with(copy_data=false);
```

Note: sub_name_A 为本地创建的订阅名
ip 为对端 数据库 B 的地址
port_number 为对端 数据库 B 数据库的连接端口
user_name 为对端 数据库 B 数据库的登录名，要有 Replication 权限。
password 如果需要, 请添加此内容，为对端 数据库 B 的密码
database_name 为对端 数据库 B 的数据库名
pub_name_B 为对端 数据库 B 创建的发布名

例子：

```
highgo=#create subscription sub_83 connection 'host=192.168.31.84 port=5432 user=highgo password=highgo dbname=test' publication pub_84 with(copy_data=false);
```

查看订阅

```
highgo=#select * from pg_subscription;
 oid | subdbid | subname | subowner | subenabled |        subconninfo
| subslotname | subsynccommit | subpublications
-----+----------+------------+-------------+----------------+-----------------------------------
+-------------+---------------+-----------------
24598 16384    sub_83              10              t      host=192.168.31.84.....
  sub_83          off                    (pub_84)
(1 rows)
```

查看具体信息

```
highgo=#select * from pg_stat_subscription;
```

8.4.2.2 数据库 B 订阅数据库 A

登录 数据库 B

创建订阅

```
highgo=#create subscription sub_name_B connection 'host=ip port=port_number user=user_name password=password dbname=database_name' publication pub_name_B with(copy_data=false);
```

Note: sub_name_B 为本地创建的订阅名
ip 为对端 数据库 B 的地址

port_number 为对端 数据库 B 数据库的连接端口
user_name 为对端 数据库 B 数据库的登录名，要有 Replication 权限。
password 如果需要, 请添加此内容，为对端 数据库 B 的密码
database_name 为对端 数据库 B 的数据库名
pub_name_B 为对端 数据库 B 创建的发布名

例子：

```
highgo=#create subscription sub_84 connection 'host=192.168.31.83 port=5432 user=highgo password=highgo dbname=test' publication pub_83 with(copy_data=false);
```

查看订阅

```
highgo=#select * from pg_subscription;
 oid | subdbid | subname | subowner | subenabled |        subconninfo
| subslotname | subsynccommit | subpublications
-----+----------+------------+-------------+----------------+-----------------------------------
+-------------+---------------+-----------------
24598 16384    sub_84              10              t      host=192.168.31.83.....
  sub_83          off                    (pub_83)
(1 rows)
```

查看具体信息

```
highgo=#select * from pg_stat_subscription;
```

至此两端相同的表就可以进行双向复制了

8.4.3 更新发布的表

8.4.3.1 添加发布的表

如需增加复制的表（两端数据库已创建相同的表），使用下列语句在两端数据库上执行

相同操作：

1）在发布端执行添加

```
highgo=#alter publication pub_name add table table_name;
```

参数解释：

pub_name:已经存在的 publication 的名称，需要和配置文件中的名称一致

table_name:需要添加到 publication 中的表名

2）在订阅端刷新

```
highgo=#alter subscription sub_name refresh publication;
```

8.4.3.2 删除发布的表

如果表不再需要复制，在两端数据库上执行相同的下述操作，只是去除发布信息，该表本身并不会删除。在发布端执行：

```
highgo=#alter publication pub_name add table table_name;
```

8.4.4 关闭双向数据复制

如下操作需要在两端服务器上都进行操作。

订阅端取消订阅

```
highgo=#drop subscription subscription_name;
```

发布端取消发布

```
highgo=#drop publication  publication_name;
```

双向数据复制开关关闭

```
#hg_enable_duplex_copy = off          ( postgresql.conf )
```

NOTE:

dbrep 除了支持瀚高数据库企业版至企业版的数据复制。还支持瀚高企业版数据库至 mysql、企业版至 oracle、mysql 至瀚高企业版数据库。这里就不细展开描述了。详细参考《dbrep 数据复制 v1.0_用户手册》、《dbrep_mysql 数据复制 v1.0_用户手册》。

9 集群功能

9.1 对等服务集群简介

传统的主备集群对外提供数据服务时，通常只有主端提供读写服务，备端只提供读服务，难以实现负载均衡，通常需要通过中间件，或者在应用端实现读写分离，保证备节点只处理读操作，主节点处理读写操作。

对等服务集群，在数据库内核中通过语法、语义分析来区分读写操作，将写操作转发到主端执行，读操作在本地执行。通过事务内可见性映射、同步点确认等技术保证数据的一致性和可见性。在数据库内核端实现读写分离，所有节点对应用端提供统一的读写能力。

- ✓ 相对于传统集群，具有以下优势：
- ✓ 每个节点对外提供读写能力，对应用端没有入侵
- ✓ 读写分离分布于各数据节点，没有集中化读写分离模块的性能瓶颈
- ✓ 所有节点对外提供服务，充分利用集群硬件资源
- ✓ 各节点功能对等，更利于简化负载均衡实现

9.1.1 配置

- ✓ 前提条件：
  已搭建好同步或异步流复制集群。
- ✓ 对等服务器集群配置：
  在备上打开对等服务功能开关 enable_standby_dispatch = true

```
[highgo@highgo data]# vi postgresql.conf
enable_standby_dispatch = true
```

重启数据库使配置文件生效

9.1.2 使用示例

备上可支持所有 DML 和 DDL 操作

- ✓ DDL 转发示例：

1. 在备节点执行以下 DDL 语句：

```
CREATE database fruitstore;
```

```
highgo=# CREATE database fruitstore;
CREATE DATABASE
highgo=#
```

2. 查看备节点 LOG，有如下字样可确认已转发；

```
statement dispatched:
```

3. 查看主节点 LOG，有如下字样可确认主节点已接收到转发信息：

```
we got a connection from high dispatch backend
```

4. 分别在主和备上都能查看到该库已成功创建

```
highgo=# \l
                                  List of databases
    Name    | Owner  | Encoding |   Collate   |    Ctype    | Access privileges
------------+--------+----------+-------------+-------------+-------------------
 fruitstore | highgo | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 highgo     | highgo | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0  | highgo | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/highgo        +
            |        |          |             |             | highgo=CTc/highgo
 template1  | highgo | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/highgo        +
            |        |          |             |             | highgo=CTc/highgo
(4 rows)
```

✓ DML 转发示例

1. 在备节点执行以下语句。

```
 CREATE table fruits
(Id smallserial,
 unit varchar(10),
 name varchar(20),
 price money,
 stock int,
 country_orig varchar(20),
 purchase_date date
);

  INSERT INTO fruits (name,unit,price,stock,country_orig,purchase_date)
values
    ('苹果','公斤','10.00','50','中国','2020-12-01'),
    ('香蕉','公斤','8.80','90','中国','2020-12-30'),
    ('青柠檬','个','5.00','200','中国','2020-2-28');
```

2. 查看备节点 LOG，有如下字样可确认已转发；

```
statement dispatched:
```

3. 查看主节点 LOG，有如下字样可确认主节点已接收到转发信息：

```
we got a connection from high dispatch backend
```

4. 用 SELECT 主备查询以确认 DML 语句执行成功。

```
  select * from fruits;
```

```
highgo=# select * from fruits;
 id | unit |  name  | price  | stock | country_orig | purchase_date
----+------+--------+--------+-------+--------------+---------------
  1 | 公斤 | 苹果   | $10.00 |    50 | 中国         | 2020-12-01
  2 | 公斤 | 香蕉   |  $8.80 |    90 | 中国         | 2020-12-30
  3 | 个   | 青柠檬 |  $5.00 |   200 | 中国         | 2020-02-28
(3 rows)
```

✓ 事务外 select 不转发示例：

1. 不在事务里的简单 SELECT 语句：
（1） 备 1 上更新，备 1 上查询；
（2） 备 1 上更新，备 2 上查询。
2. 查看备节点 LOG，没有以下转发 log 字样；

```
statement dispatched:
```

3. 查看主节点 LOG，没有有如下转发 log 字样可确认 select 未转发到主：

```
we got a connection from high dispatch backend
```

9.2 负载均衡

9.2.1 负载均衡简介

瀚高数据库负载均衡选件是由瀚高基础软件公司公司自主研发的，用于“瀚高对等服务数据库集群 ”之上的、实现数据库连接均衡负载的软件。该选件属于“瀚高对等服务数据库集群版”的一部分。

在客户应用端发起数据库连接请求（在使用 JDBC 驱动连接数据库）时，该选件会根据数据库集群中各服务器的连接负载情况结合用户事先设定的各服务器的负载权重，自动地分配客户端连接到最佳的数据库服务器上，以实现群集中各数据库服务器的负载均衡。

该选件提供瀚高数据库集群的会话（连接）级别的负载均衡能力，能有效地提升数据库集群的整体设备利用率。

9.2.2 安装部署

9.2.2.1 前提条件

使用该选件前需安装部署瀚高对等服务集群，该集群提供多个对等的 HGDB 数据库服务器。在完成瀚高数据库集群的部署后，请记录下集群中各数据库服务器的网址、端口；并给各数据库服务器起容易记忆的名字。如果集群中的各服务器有主备之分的话，也请记下来。

另外，根据业务的需要，结合各服务器所在设备的处理能力等多方面的考虑，给出集群中各数据库服务器的负载权重。注意：每个服务器在整个集群中负载占比等于：本服务器的权重/各服务器权重之和。因此，权重越大、所承担负载越大；相同的权重，可获得均衡的负载。

例如：假设我们有 4 个数据库服务器构成的集群（我们起名字叫 Server1 到 4），其中第 1 个是主节点，我们记下了它们的网址（从 192.168.8.142 开始）和端口（假设都用 5866 端口）。同时，我们想以 1:3:3:3 的配比设定它们的负载权重。这样，我们就

能得到以下的列表：

Server1: 192.168.8.142, 5866, 主节点, 10

Server2: 192.168.8.143, 5866, 从节点, 30

Server3: 192.168.8.144, 5866, 从节点, 30

Server4: 192.168.8.145, 5866, 从节点, 30

有了以上的信息，我们就可以着手安装、配置该负载均衡选件了。

9.2.2.2 安装步骤

1. 将 hgloadbalance 文件夹中的 SQL 文件上传至集群主节点服务器中。该文件夹中有五个文件：
   - 1-create_user&shema.sql
   - 2-create_table&func.sql
   - 3-grant_public.sql
   - 4-insert_db_ha.sql
   - 5-remove_all.sql
2. 登录到群集中的数据库服务器进行配置工作：理论上讲，我们可以登录到集群中的任意一个数据库服务器做配置工作。但为了稳妥起见，我们建议用户登录到集群中的主服务器节点上做负载均衡选件的配置。同时请注意：如果我们的应用需要用到多个数据库，我们就需要以每一个数据库的名义多次登录服务器去做以下的操作：
3. 登录到数据库集群中的节点（建议主节点），运行：1-create_ user&schema.sql，如下图所示：

```
[highgo@node3 hgload]$ pwd
/home/highgo/hgload
[highgo@node3 hgload]$ ll
total 20
-rw-r--r-- 1 highgo highgo  409 Jul  6 14:53 1-create_user&schema.sql
-rw-r--r-- 1 highgo highgo 5468 Jul  6 14:53 2-create_table&func.sql
-rw-r--r-- 1 highgo highgo  178 Jul  6 14:53 3-remove_all.sql
-rw-r--r-- 1 highgo highgo  198 Jul  6 14:53 4-insert_hg_load_node_infos.sql
[highgo@node3 hgload]$ psql
psql (5.6.5)

PSQL: Release 5.6.5
Connected to:
HighGo Database V5.6 Enterprise Edition Release 5.6.5 - 64-bit Production

Type "help" for help.

highgo=# \i 1-create user&schema.sql
CREATE ROLE
CREATE SCHEMA
ALTER SCHEMA
GRANT
CREATE EXTENSION
highgo=#
```

说明：

该 SQL 文件会在用户业务库中创建负载均衡选件所需要的数据库对象。

注意：若出现下图所示错误，则说明扩展默认没有安装，可按以下步骤手动安装解决：

```
[postgres@localhost hgloadbalance_sql]$ psql
psql (12.1)
Type "help" for help.

postgres=# \i 1-create_user&shema.sql
CREATE ROLE
CREATE SCHEMA
ALTER SCHEMA
GRANT
psql:1-create_user&shema.sql:15: ERROR:  could not open extension control file "/usr/local/pgsql/share/extension/dblink.control": No such file or directory
postgres=#
```

(1) 切换到源码安装目录中的 dblink 扩展文件夹，如：cd /usr/local/pgsql/postgresql-12.1/contrib/dblink/

(2) make

(3) make install

(4) 重新运行 1-create_ user&schema.sql

4. 继续执行 SQL 文件： 2-create_table&func.sql，如下图所示：

```
postgres=# \i 2-create_table&func.sql
You are now connected to database "postgres" as user "hgloadbalance".
CREATE TABLE
CREATE TABLE
INSERT 0 1
CREATE FUNCTION
CREATE FUNCTION
CREATE FUNCTION
postgres=#
```

说明：

该 SQL 文件会创建该选件运行所需的数据库控制表和函数。

5. 继续执行 SQL 文件：3-grant_public.sql，如下图所示：

```
postgres=# \i 3-grant_public.sql
GRANT
GRANT
GRANT
GRANT
GRANT
GRANT
postgres=#
```

说明：

该 SQL 文件会设置权限，授予普通用户做负载操作时所需要的权限。

6. 修改添加集群信息 sql 模板，配置节点负载权重

修改 4-insert_db_ha.sql 文件，初始化 db_ha 表的节点数据，配置各个节点的权重，如果不配置，所有节点权重默认为 1。

示例:

```
insert into public.db_ha (node_ip, db_port, weight)
Values ( '192.168.8.142', 5866, 10 ),
( '192.168.8.143', 5866, 30 ),
```

```
( '192.168.8.144', 5866, 30 ),
( '192.168.8.145', 5866, 30 );
```

字段说明：

- node_ip：数据库服务器的 IP 地址；
- db_port：节点数据库的端口号；
- weight：该数据库节点连接数分配权重；

在编辑完 4-insert_db_ha.sql 脚本后，运行该脚本。这样，我们就完成了负载均衡选件在服务器端的部署。

9.2.3 客户端 JDBC 配置

在客户端应用中，请采用瀚高提供的 HGDB 客户端 JDBC 驱动包。在应用中如下设置 JDBC 的连接串：

JDBC 的连接串模板:

jdbc:highgo://192.168.168.142:5866/hgdb?user=hgdb&password=hgdb&loadBalanceHosts=true&standByInfoType=chooseserver

说明：

其中，该 URL 中的 host，port，database 等参数为用户在应用中所需要用的数据库连接串。理论上讲，host 可以是集群中的任一数据库服务器，但为简便起见，host 设为集群中的主服务器。

loadBalanceHosts = true，表示开启负载均衡功能。

standByInfoType = chooseserver 设置负载均衡的实现方案为 chooseserver。

其它使用方法与普通 JDBC 没有任何区别。

9.2.4 配置更改与清除

9.2.4.1 更改负载权重

负载均衡的设置信息在系统运行过程中可以通过以下 SQL 语句获得：

```
select node_ip, db_port, weight
from public.db_ha;
```

该选件支持用户在线修改各服务器的负载权重，通过语句：

```
update public.db_ha
set weight = xxx
```

```
where node_ip= 'yyy' and db_port = xxx;
```

9.2.4.2 更改服务器负载刷新时间

选件在运行时，后台会定期刷新各个服务器的实际连接负载。缺省的刷新间隔是60 秒。用户可以在线修改这个时间间隔。例如：将刷新间隔设为 30 秒：

```
update hgloadbalance.hg_load_global set deltatime =30;
```

9.2.4.3 清除负载均衡选件

此外，如果用户决定集群不再使用负载均衡功能或有其它需要取消 hgloadbalance 相关功能的需求，该版本提供负载均衡选件的清除功能。登录数据库服务器（任意节点都可以，但建议从主节点登录，并通过所使用的各数据库），执行 sql 脚本：5-remove_all.sql

如下图所示：

```
postgres=# \i 5-remove_all.sql
psql:5-remove_all.sql:3: NOTICE:  drop cascades to 4 other objects
DETAIL:  drop cascades to table hgloadbalance.hg_load_global
drop cascades to function hgloadbalance.hg_choose_server()
drop cascades to function hgloadbalance.hg_refresh_load_node_infos(text,text,text)
drop cascades to function hgloadbalance.hg_refresh_load_node_info_handler(text,integer,text,text,text)
DROP SCHEMA
DROP TABLE
REVOKE
DROP ROLE
postgres=#
```

注意：该负载均衡选件使用到了数据库的 dblink 插件，它在运行脚本 1-create_user&shema.sql 时被引入，但并未在该清除脚本中进行删除。这是因为 dblink 属于数据库通用选件，用户的其它应用模块可能对此存在依赖。若确认不需要 dblink 插件，可以执行以下 SQL 手动删除：

```
DROP EXTENSION dblink;
```

10 分布式 OLTP 解决方案

10.1分布式数据库安装

10.1.1 安装检查

10.1.1.1 推荐配置

配置参数	最低配置	推荐配置
CPU	4 核	16 核
内存	4GB	64GB
存储	800MB，机械硬盘	5GB 以上，SSD 或 NvMe
网络	千兆网络	万兆网络

10.1.1.2 内存和参数检查

以 root 用户身份登录并运行以下命令。

1、要查看可用 RAM 和交换空间大小，运行以下命令：

```
df -h
free -h
grep MemTotal /proc/meminfo
grep SwapTotal /proc/meminfo
# grep MemTotal /proc/meminfo
MemTotal:512236 kB
# grep SwapTotal /proc/meminfo
SwapTotal:1574360 kB
```

表 2 内存及对应 swap 建议值参照表

MemTotal	SwapTotal
8G	2~4G
8~16G	4~8G
16~64G	8~32G
>=64G	32G

2、检查内核参数

```
cat /proc/sys/kernel/shmmax
cat /proc/sys/kernel/shmall
cat /proc/sys/kernel/shmmni
##该参数（系统共享内存段的最大数量）数据库自动修改
```

表 2 内存大小及对应内核参数建议值参照表

MemTotal	shmall	shmmax
8G	7~8G	4G
8~64G	M*50%	M*90~95%
>=64G	32G	M*90%~95%

10.1.2 安装前准备

10.1.2.1 关闭防火墙

在 Red Hat Enterprise Linux Server release 7.X 执行如下命令：

```
systemctl stop firewalld.service
systemctl disable firewalld.service
systemctl status firewalld.service
systemctl stop  NetworkManager.service
systemctl disable NetworkManager.service
systemctl status  NetworkManager.service
```

如果是 Red Hat Enterprise Linux Server release 6.X，执行如下命令：

```
service iptables stop
chkconfig iptables off
service iptables status
service NetworkManager stop
chkconfig NetworkManager off
service NetworkManager status
```

10.1.2.2 关闭 SELINUX

```
[root@host1 ~]# sed -i "s/SELINUX=enforcing/SELINUX=disabled/g"
```

```
/etc/selinux/config
  [root@host1 ~]# setenforce 0
  [root@host1 ~]# cat /etc/selinux/config | grep SELINUX=disabled
  [root@host1 ~]# getenforce
```

10.1.2.3 检查主机名

```
  [root@host1 ~]# hostnamectl status
    Static hostname: host1
          Icon name: computer-server
            Chassis: server
         Machine ID: 67d04193d2a14465897d3185871da992
            Boot ID: c45ce687acc849bd960f6b151cf04305
   Operating System: Red Hat Enterprise Linux Server 7.4 (Maipo)
        CPE OS Name: cpe:/o:redhat:enterprise_linux:7.4:GA:server
             Kernel: Linux 3.10.0-693.el7.x86_64
       Architecture: x86-64
```

10.1.2.4 检查时间和时区

```
  [root@host1 ~]$ timedatectl
      Local time: Sat 2020-08-08 06:51:49 CST
  Universal time: Fri 2020-08-07 22:51:49 UTC
        RTC time: Sat 2020-10-10 13:30:28
       Time zone: Asia/Shanghai (CST, +0800)
     NTP enabled: yes
NTP synchronized: no
 RTC in local TZ: yes
      DST active: n/a
```

时区不对情况下如何修改时区：

```
  [root@host1 ~]# timedatectl list-timezones
  [root@host1 ~]# timedatectl set-timezone Asia/Shanghai
  修改时间
  [root@host1 ~]# date -s "20170622 10:26:00"
```

10.1.2.5 配置操作系统 limits

```
  [root@host1 ~]# vi /etc/security/limits.conf
  #for highgo db 4.5
```

```
highgo soft  core unlimited
highgo hard  nproc unlimited
highgo soft  nproc unlimited
highgo hard  memlock unlimited
highgo hard  nofile 1024000
highgo soft  memlock unlimited
highgo soft  nofile 1024000
highgo hard  stack  65536
highgo soft  stack  65536
```

10.1.2.6 安装系统软件包

配置 yum 源：

```
[root@host1 ~]# cat /etc/yum.repos.d/highgo.repo
[rhel]
name=rhel
baseurl=file:///media/cdrom
enabled=1
gpgcheck=0
[root@host1 ~]# mkdir /media/cdrom
[root@host1 ~]# mount /dev/cdrom /media/cdrom
mount: /dev/sr0 is write-protected, mounting read-only
```

安装依赖包：

```
[root@host1 ~]# yum clean all
[root@host1 ~]# yum list
[root@host1 ~]# yum install vim wget readline readline-devel zlib zlib-devel openssl openssl-devel pam-devel libxml2-devel libxslt-devel python-devel tcl-devel gcc gcc-c++ rsync -y
```

10.1.2.7 配置 hosts

说明：安装脚本内有自动配置 hosts 功能，可以不进行该项。若安装脚本配置不成功可以手动进行该项作为补充。

```
[root@host1 ~]# cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
```

```
192.168.31.141 host1
192.168.31.142 host2
192.168.31.143 host3
192.168.31.144 host4
```

10.1.2.8 创建 highgo 用户并修改密码

说明：安装脚本内有自动创建安装用户的功能，可以不进行该项。若安装脚本创建不成功（部分国产机创建用户命令不一样有创建不成功现象）可以手动进行该项作为补充。

```
[root@host1 ~]# groupadd -g 5866 highgo
[root@host1 ~]# useradd -u 5866 -g highgo highgo
[root@host1 ~]# passwd highgo
```

10.1.2.9 建互信

说明：安装脚本内有自动建互信功能，可以不进行该项。若互信创建不成功（部分国产机因命令等原因存在创建不成功现象）可以手动进行该项作为补充。

1） 生成密钥

```
[root@host1 ~]$  ssh-keygen -t rsa
```

2）将本机的公钥复制到各个节点机器的 authorized_keys 文件中

```
$ cat id_rsa.pub >> authorized_keys
$ chmod 600 authorized_keys
$ ssh-copy-id -i id_rsa.pub root@host2
$ ssh-copy-id -i id_rsa.pub highgo@host2
```

10.1.2.10 设置 highgo 用户环境变量

说明：安装脚本内有自动创建环境变量功能，可以不进行该项。若自动创建环境变量不成功（部分国产机因命令等原因存在创建不成功现象）可以手动进行该项作为补充。

```
[highgo@host1 ~]# vi /home/highgo/.bashrc
export HGDB_HOME=/usr/local/hg_dist
export PATH=$HGDB_HOME/bin:$PATH:$HOME/bin
export LD_LIBRARY_PATH=$ HGDB_HOME/lib:$LD_LIBRARY_PATH

[highgo@host1 ~]$  source  ~/.bashrc
```

说明：

若不设置 LD_LIBRARY_PATH 可能会导致无法登录数据库。

10.1.3 安装数据库

10.1.3.1 安装脚本说明

（1）安装脚本主要分为三部分：安装包 hgeddb.tar.gz　　root 用户执行脚本、数据库安装用户执行脚本。

（2）root 用户执行脚本完成了 root 用户的互信、新建数据库安装用户、数据库安装用户的互信、配置 hosts、配置 sudoer、分发安装包 hgeddb.tar.gz，hidha service 开机自启动，rsyslog 配置等。

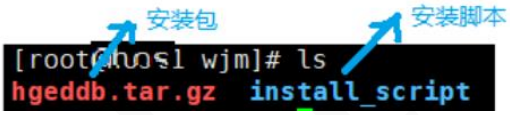
（3）数据库安装用户执行的脚本完成了环境变量配置、CN 数据的安装和启动、DN 数据的安装和启动、建表工具的环境搭建、备份安装、hidha 的安装和启动、listener 启动、backup 工具的安装配置、数据库节点配置文件的生成和分发等。

10.1.3.2 安装脚:本目录结构

（1）tar -xvzf　HGEDDB.tar.gz

（2）hgeddb.tar.gz:　是打包工具成功后的安装包

（3）install_script:自动化安装脚本



10.1.3.3 Root 用户安装脚本执行

（1）cd install_scrip 下配置 all_hosts.conf

```
[root@host1 install_script]# cat all_hosts.conf
#ip,user,password,host_name,datanode_num,slave_num,port_range,vip,nt_card,dbname,dbuser,dbpwd,partition_num,schema_name
192.168.31.151,tester7,1,host1,1,1,5100,192.168.31.162,p4p2,hgdb,hgdb,hgdb,6,benchmarksql
192.168.31.153,tester7,1,host3,1,1,5300,192.168.31.164,p4p2,hgdb,hgdb,hgdb,6,benchmarksql
192.168.31.154,tester7,1,host4,1,1,5400,192.168.31.165,p4p2,hgdb,hgdb,hgdb,6,benchmarksql
```

✓ 配置文件说明：

此配置文件一共包括 14 列：ip、user、password、host_name（/etc/hosts 里配置的内容，注意 host_name 不能重名）、datanode_num(每个 Host 机器上要安装几个数据库节点)、slave_num(每个数据库节点有几个备份)、port_range(端口号的起始值)、vip(虚拟 IP)、nt_card(网卡)、dbname(新建的数据库名称)、dbuser(新建数据库的用户名)、dbpwd(新建数据的密码)、partition_num（分片数，各个 host 机器配置的分片数填写要一致且为集群总的分片数，比如你要想集群做 6 个分片，就每行配

置为 6）、schema_name（分片 schema 名）

✓ 注意事项：

1、集群各个 Host 机器安装的 datanode_num 总数要大于等于 3；

2、partition_num 分片数需要大于等于 datanode_num 总数、最好是 datanode_num 总数的倍数关系。

（2）执行./quick_install.sh

此脚本会生成 initdb_config.conf 配置文件，查看内容是否有需要修改的部分，比如数据目录等的配置：

```
hgdb_package_name=hgdb.tar.gz

hgdbInstallDir=/usr/local/hg_dist/

coordPorts=(5500 7700 9900)

 #[Coordinators Master]
coordMasterServers=(192.168.31.125 192.168.31.126 192.168.31.127)
coordMasterDirs=(/data/hg_dist/coord1 /data/hg_dist/coord2 /data/hg_dist/coord3)

 #[Datanodes Master]
dataNodePorts=(5501 7701 9901)

 dataNodeMasterServers=(192.168.31.125 192.168.31.126 192.168.31.127)
dataNodeMasterDirs=(/data/hg_dist/dnmaster1 /data/hg_dist/dnmaster2 /data/hg_dist/dnmaster3)
dataNodeMasterVIp=(192.168.31.11 192.168.31.11 192.168.31.12)
dataNodeMasterNetCard=(ens192 ens192 ens192)

#[Datanodes Slave]
 datanodeSlave=1

 #[Datanodes Slave_0]
 dataNodeSlaveServers_0=(192.168.31.126 192.168.31.127 192.168.31.125)
 dataNodeSlaveDirs_0=(/data/hg_dist/dnslave /data/hg_dist/dnslave /data/hg_dist/dnslave)

 install_log_file=/var/log/hgdist/install.log
~
~
```

（3）执行./root_main.sh 进入 root 部分的安装

在 allhosts 配置的第一个服务器的/var/log/hgdist 目录里 install.log 可以查看安装日志。

10.1.3.4 新建普通用户安装脚本执行

切换到新建数据库安装用户：

su 新建用户，比如：su highgo

（1）完善集群相关配置：

✓ 配置集群 CN 的 postgresql.conf 内容：

```
[highgo@host1 ~]# vi  /usr/local/new_user_install_script/coordExtraConfig

  listen_addresses = '*'
```

```
###audit config#####################

logging_collector = on
log_line_prefix = ''
log_destination = 'syslog'
syslog_facility = 'LOCAL1'

shared_preload_libraries = 'pgaudit'
pgaudit.log = 'all, -misc'
pgaudit.log_catalog = on
pgaudit.log_client = on
pgaudit.log_level = log
pgaudit.log_parameter = on
pgaudit.log_relation = on
pgaudit.log_statement_once = off
#pgaudit.role='header'

log_file_mode = 0640

wal_level = replica
archive_mode = on
```

✓ 配置集群 CN 的 pg_hba.conf 内容

```
[highgo@host1 ~]# vi  /usr/local/new_user_install_script/coordExtraPgHba
  host all all 0.0.0.0/0 trust
  host replication all 0.0.0.0/0 trust
```

✓ 配置集群 DN 的 postgresql.conf 内容

```
[highgo@host1 ~]# vi  /usr/local/new_user_install_script/dataNodeExtraConfig
  listen_addresses = '*'
  shared_preload_libraries = 'repmgr,pgaudit'
```

```
wal_log_hints=on
full_page_writes=on
wal_keep_segments=100
wal_level = replica
archive_mode = on
synchronous_commit = remote_write
log_destination = 'csvlog'
logging_collector = on
log_directory = 'pg_log'
log_filename = 'postgresql-%Y-%m-%d_%H%M%S.log'
```

✓ 配置集群 DN 的 pg_hba.conf 内容

```
[highgo@host1 ~]# vi  /usr/local/new_user_install_script/dataNodeExtraPgHba
host    all     all     0.0.0.0/0     trust
host    replication     all             0.0.0.0/0            trust
```

（2）执行安装脚本

```
[highgo@host1 ~]# ./main.sh
```

当出现如下界面时视为安装成功：

```
**********coordinator state**********
-----> coordinator state
Name    IP            Port    Datadir                        dbname  dbuser  status              postmaster_start_time
cn1     192.168.30.141  4100    /ssd_dir/hg_dist/coord1 hgdb    hgdb    running                 2020-07-25 12:32:12
cn2     192.168.30.143  4300    /ssd_dir/hg_dist/coord2 hgdb    hgdb    running                 2020-09-23 03:18:32
cn3     192.168.30.144  4400    /ssd_dir/hg_dist/coord3 hgdb    hgdb    running                 2020-09-23 03:15:59

**********datanode state**********
-----> datanode state
Name            IP              Port            Datadir                 dbname  dbuser  status          repmgrd         postmaster_start_time
primary_1_1     192.168.30.141  4101    /ssd_dir/hg_dist/dnmaster1      hgdb    hgdb    running         running         2020-07-25 12:32:22
standby_1_2     192.168.30.143  4101    /ssd_dir/hg_dist/dnslave1       hgdb    hgdb    running         running         2020-09-23 15:20:22
primary_2_1     192.168.30.141  4102    /ssd_dir/hg_dist/dnmaster2      hgdb    hgdb    running         running         2020-07-25 12:32:22
standby_2_2     192.168.30.143  4102    /ssd_dir/hg_dist/dnslave2       hgdb    hgdb    running         running         2020-09-23 15:20:22
primary_3_1     192.168.30.143  4301    /ssd_dir/hg_dist/dnmaster3      hgdb    hgdb    running         running         2020-09-23 03:18:41
standby_3_2     192.168.30.144  4301    /ssd_dir/hg_dist/dnslave3       hgdb    hgdb    running         running         2020-09-23 03:17:49
primary_4_1     192.168.30.143  4302    /ssd_dir/hg_dist/dnmaster4      hgdb    hgdb    running         running         2020-09-23 03:18:41
standby_4_2     192.168.30.144  4302    /ssd_dir/hg_dist/dnslave4       hgdb    hgdb    running         running         2020-09-23 03:17:50
primary_5_1     192.168.30.144  4401    /ssd_dir/hg_dist/dnmaster5      hgdb    hgdb    running         running         2020-09-23 03:16:08
standby_5_2     192.168.30.141  4401    /ssd_dir/hg_dist/dnslave5       hgdb    hgdb    running         running         2020-07-25 00:34:13
primary_6_1     192.168.30.144  4402    /ssd_dir/hg_dist/dnmaster6      hgdb    hgdb    running         running         2020-09-23 03:16:08
standby_6_2     192.168.30.141  4402    /ssd_dir/hg_dist/dnslave6       hgdb    hgdb    running         running         2020-07-25 00:34:14
```

以下为补充内容，非安装数据库操作步骤必须：安装脚本内有自动生成 TPCC 配置文件 partitionUrls.properties 的功能，若需要用 benchmarkSQL 跑 TPCC 需要 partitionUrls.properties，可以到如下位置获得：

```
[highgo@host1 ~]# cd /usr/local/new_user_install_script
./create_partitionUrls_config.sh
```

生成 partitionUrls.properties，该文件拷贝到 benchmarksql 的 run 目录下即可使用

10.1.4 卸载数据库

说明：因卸载过程中会删除数据目录，强烈建议您先备份各 CN 及 DN 的数据目录，再执行如下操作，以防止数据丢失后无法找回。

su 新建用户，比如: su tester

（1）cd /usr/local/new_user_install_script/，新建普通用户脚本存放位置

（2）./uninstall

（3）切换到 root 用户下，cd /droot/install_script/root_install_script

（4）./hguninstall.sh

10.2 分布式分片表管理工具

分布式数据库使用使用参见《瀚高企业版分布式数据库系统 V4.5-用户使用手册》，现只对分布式分片管理工具的使用做个介绍。

10.2.1 分片表管理工具简介

HGDB Sharding 提供了统一的分片表管理工具，来进行用户数据库、模式及表等结构的创建、修改、删除的管理，用户也可以使用建表工具进行数据库用户权限的授予及收回等操作。提供近于完整的 SQL 支持，对用户来讲屏蔽掉了数据库对数据的水平分拆带来的集群规模的复杂度。

10.2.2 分片表管理工具使用

用户在服务器中使用数据库用户在 HGDB 分布式数据库安装目录中的 conf/“hgMultiDdlSql.conf”文件里自定义符合 HG 分布式数据库的 SQL 命令，并以“；“为结尾标记单条命令结束，运行命令“hgExecMultiSql”执行定义在“hgMultiDdlSql.conf”中的 SQL 列表。

对于数据库对象的创建、删除、修改必须使用本执行器进行操作，不可以直连单库进行操作。

/usr/local/hg_dist/conf/hgMultiDdlSql.conf 文件（例）：

✓ 分片表创建语法：

```
create table tab_a (
        a int,
        b char(5),
        c varchar(10),
        d char(5))
        partition by hash(a);

create table tab_b (
        a int,
        b char(5),
        c varchar(10),
        d char(5))
        partition by hash(a);

create table tab_c (
        a int,
        b char(5),
        c varchar(10),
        d char(5))
        partition by hash(a);
```

✓ 复制表创建语法：

```
create table c (
  cfg_name    varchar(30) primary key,
  cfg_value   varchar(50)
) PARTITION BY REPLICATION;
```

执行/usr/local/hg_dist/bin/hgExecMultiSql 命令成功后输出信息如下图：

```
------------------------------------------------------------
HG MultiSQL EXEC MainConn ip:[192.168.59.151] port:[1921] Succeed!
------------------------------------------------------------
------------------------------------------------------------
HG MultiSQL EXEC MainConn ip:[192.168.59.149] port:[1922] Succeed!
------------------------------------------------------------
```

10.3 Sharding -jdbc

10.3.1 HighGo 分布式 JDBC 产品前置条件

（1）数据是分片的，具体分片规则可参考技术白皮书里的第 2 章节
（2）执行 execue(key)，目前 key 的类型只支持 int 类型
（3）执行 execute 不传 key 时，目前指定了一个机器的全表扫描
（4）分片 url 的配置

```
partition_num=3
#part_0=jdbc:postgresql://192.168.95.121:5436,192.168.95.122:5436,192.168.95.123:5436/benchmarksql?user=benchmarksql&password=benchmarksql&sslmode=disable&loadBalanceHosts=true&preferQueryMode=extended
part_0=jdbc:postgresql://192.168.100.121:5436/benchmarksql?user=benchmarksql&password=benchmarksql&sslmode=disable&preferQueryMode=extended
part_1=jdbc:postgresql://192.168.100.121:5436/benchmarksql?user=benchmarksql&password=benchmarksql&sslmode=disable&currentSchema=benchmarksql0&preferQueryMode=extended
part_2=jdbc:postgresql://192.168.100.122:5436/benchmarksql?user=benchmarksql&password=benchmarksql&sslmode=disable&currentSchema=benchmarksql1&preferQueryMode=extended
part_3=jdbc:postgresql://192.168.100.123:5436/benchmarksql?user=benchmarksql&password=benchmarksql&sslmode=disable&currentSchema=benchmarksql2&preferQueryMode=extended
~
```

10.3.2 HighGo 分布式分片 url 配置规则

（1） partitionUrls.properties：这是配置文件的名字不能修改

（2） 需要设置环境变量来配置上述配置文件的路径：

export PARTITION_URL=/home/postgres/benchmarksql-5.0/run
export PARTITION_URL=文件存放目录

（3） 分片配置文件的规则：

- 第一行：partition_num(此名字不能修改)是分片数
- 第二行：key 的名字可以随便修改，配置的是协调节点的 url
- 从第三行开始，配置的是各个分片的地址 url

```
partition_num=2   //分片数
part_0=jdbc:postgresql://192.168.100.123,192.168.100.124:5436/benchmarksql?user=benchmarksql&password=benchmarksql&sslmode=disable&loadBalanceHosts=true&preferQueryMode=extended //0 分片是协调节点，当 client 端调用无参的 execute 方法时，就取这个 parititionUrl,在所有节点里随机选一个几点连接
part_1=jdbc:postgresql://192.168.100.123:5436/benchmarksql?user=benchmarksql&password=benchmarksql&sslmode=disable&currentSchema=benchmarksql1&preferQueryMode=extended//分片 1
part_2=jdbc:postgresql://192.168.100.124:5436/benchmarksql?user=benchmarksql&password=benchmarksql&sslmode=disable&currentSchema=benchmarksql2&preferQueryMode=extended //分片 2
```

（4） 分片 url 说明：

part_0=jdbc:postgresql://192.168.100.121:5436/benchmark

sql?user=benchmarksql&password=benchmarksql&sslmode=disable&preferQueryMode=extended

- part_0：key 的名字可以视具体情况而定
- ip:port/databaseName?user=&password=
- preferQueryMode=extended:扩展模式

10.3.3 应用端开发者调用接口案例

```
1.  public int updateLastName(String addr, int id, String name, int age) {
2.      String SQL = "update testliz set addr=? where id=? and name=? and age=?";
3.      int affectedrows = 0;
4.
5.      try {
6.          HgConnection conn = connect();
7.          HgPreparedStatement pstmt = conn.prepareStatement(SQL);
8.
9.          pstmt.setString(1, addr);
10.         pstmt.setInt(2, id);
11.         pstmt.setString(3, name);
12.         pstmt.setInt(4, age);
13.         System.out.println("sql is ==============" + pstmt.punit.sqlStmt);
14.
15.         affectedrows = pstmt.executeUpdate(partitionKey);
16.
17.     } catch (SQLException ex) {
18.         System.out.println(ex.getMessage());
19.     }
20.
21.     return affectedrows;
22.   }
23.
24.   /**
25.    * @param args the command line arguments
26.    */
27.   public static void main(String[] args) {
28.     Test main = new Test();
29.     main.updateLastName("aaaa", 1, "jack", 12);
```

11 定时任务 db_cron

11.1 db_cron 简介

db_cron 是瀚高企业版数据库的一个简单的基于 cron 的作业调度工具，它作为扩展在数据库中运行。db_cron 目前提供定时任务的新增，在线更改、删除、查看等功能。db_cron 与常规 cron 保持相同的语法，但它允许直接从数据库执行 SQL 命令。

db_cron 可以平行的运行多个作业，但每个时间点最多运行一个作业实例。如果第二个作业应该在第一次运行完成之前开始运行，则第二个作业的运行将排队，并在第一个作业运行完成后立即开始。

11.2 安装 db_cron

1、上传 db_cron1.0-ee-el7.x86_64.rpm 安装包

2、执行 export 命令

```
# export PGPATH = /usr/local/hgdb
```

3、安装 rpm 包

```
# rpm -ivh db_cron-1.0-el7.x86_64.rpm --nodeps
```

11.3 配置 postgresql.conf

```
[highgo@highgo data]# vi postgresql.conf
shared_preload_libraries = 'db_cron'
cron.database_name = 'highgo'
```

重启数据库使配置文件生效

11.4 使用 db_cron

1、创建拓展

```
[highgo@highgo data]# psql -U highgo -d highgo
highgo=# create extension db_cron;
CREATE EXTENSION
```

2、创建测试表

```
[highgo@highgo data]# psql -U highgo -d highgo
highgo=# create table student(id serial,name char(30),create_time timestamp(0));
CREATE TABLE
```

3、创建定时任务

1）调用函数创建定时任务

```
highgo=# SELECT cron.schedule('*/5 * * * *', 'VACUUM');
```

每 5 分钟一次 VACUUM

2）使用标准 SQL 创建定时任务

```
highgo=#SELECT cron.schedule('*/2 * * * *', $$insert into student(name,create_time) values('abc',now())$$);
```

每 2 分钟插入一条数据;

4、查询定时任务

```
highgo=#select * from cron.job;
```

```
highgo=# select * from cron.job;
 jobid |  schedule   |                          command                          | nodename  | nodeport | database | username | active | jobname
-------+-------------+-----------------------------------------------------------+-----------+----------+----------+----------+--------+---------
     1 | */5 * * * * | VACUUM                                                    | localhost |     5432 | highgo   | highgo   | t      |
     3 | */2 * * * * | insert into student(name,create_time) values('abc',now()) | localhost |     5432 | highgo   | highgo   | t      |
(2 rows)
```

5、定时任务执行结果

```
highgo=# select * from student;
 id |                name                |      create_time
----+------------------------------------+-----------------------
  1 | abc                                | 2021-11-05 10:26:00
  2 | abc                                | 2021-11-05 10:28:00
  3 | abc                                | 2021-11-05 10:30:00
  4 | abc                                | 2021-11-05 10:32:00
  5 | abc                                | 2021-11-05 10:34:00
  6 | abc                                | 2021-11-05 10:36:00
  7 | abc                                | 2021-11-05 10:38:00
  8 | abc                                | 2021-11-05 10:40:00
(8 rows)
```

11.5 取消定时任务

1、取消单个定时任务

```
highgo=#select cron.unschedule(1);
```

2、取消所有定时任务

```
highgo=#select cron.unschedule (jobid) FROM cron.job ;
```

3、删除表 cron.job 表中对应记录来取消定时任务

```
highgo=#delete from cron.job where jobid=1;
```

12 批量导入 db_bulkload

12.1 db_bulkload 简介

瀚高 db_bulkload 跳过对内存以及 WAL 日志的写入，直接写入文件，速度远超传统数据加载工具。 同时，db_bulkload 支持更多特性，输入数据源可以是本地文件，或者 SQL 函数的输出，也可以是跨越网络的标准输入，输入格式除了 CSV 也支持 binary 格式，数据加载支持多进程的 multi-process 模式，速度更快，支持用户自定义的 filter 等等。

12.2 安装 db_bulkload

安装 rpm 包
1、上传 db_bulkload-1.0-1.el7.x86_64.rpm

2、执行 export 命令

```
# export PGPATH = /usr/local/hgdb
```

3、安装 rpm 包

```
[root@highgo local]# rpm -ivh db_bulkload-1.0-1.el7.x86_64.rpm --nodeps
```

12.3 使用 db_bulkload

1、创建扩展

```
[highfo@highgo etc]$ psql -U highgo -d highgo
highgo=# create extension db_bulkload;
CREATE EXTENSION
```

2、创建测试表

```
[highfo@highgo etc]$ psql -U highgo -d highgo
highgo=# create table tbl_lottu(id int,name text);
CREATE TABLE
```

3、创建测试文件
1）批量导入创建测试文件

```
[highgo@dbpulkload ~]$ seq 100000| awk '{print $0"|lottu"}' > tbl_lottu_output.txt
```

2）手动创建测试文件

```
[highgo@dbpulkload ~]$ vi tbl_lottu_output.txt
1|2|b|5
3|4|c|6
```

Note: 每个数据之间用‘|’分割，以上例子为两行数据一行 4 个字符。

12.4 使用命令行参数进行数据加载

1、运行命令

```
[highgo@pgbulkload ~]$ db_bulkload -i /home/highgo/tbl_lottu_output.txt -O tbl_lottu -l /home/highgo/tbl_lottu_output.log -P /home/highgo/tbl_lottu_bad.txt -o "TYPE=CSV" -o "DELIMITER=|" -d highgo -U highgo
```

上述命令解释，其中

/home/highgo/tbl_lottu_output.txt	创建的测试文件路径
tbl_lottu	创建的测试表
-d highgo	数据库名
-U highgo	数据库用户名

（1）加载选项

加载数据的选项。
-i INPUT --input = INPUT --infile = INPUT 从中加载数据的源。与控制文件中的“INPUT”相同；
-O 输出 --output = OUTPUT 将数据加载到的目标。与控制文件中的“OUTPUT”相同；
-l LOGFILE --logfile = LOGFILE 写结果日志的路径。与控制文件中的“LOGFILE”相同；
-P PARSE_BADFILE --parse-badfile = PARSE_BADFILE 用于写入无法正确解析的不良记录的路径。与控制文件中的“PARSE_BADFILE”相同；
-u DUPLICATE_BADFILE --duplicate-badfile = DUPLICATE_BADFILE 在索引重建期间写入与唯一约束冲突的坏记录的路径。与控制文件中的“DUPLICATE_BADFILE”相同；
-o“key = val” --option =“key = val” 控制文件中可用的任何选项；
您可以传递多个选项。
（2）连接选项
连接到服务器的选项。
-d dbname --dbname dbname 指定要连接的数据库的名称。如果未指定，则从环境变量 PGDATABASE 读取数据库名称。如果未设置，则使用为连接指定的用户名；
-h host --host host 指定运行服务器的计算机的主机名。如果值以斜杠开头，它将用作 Unix 域套接字的目录；
-p port --port port 指定服务器在其上侦听连接的 TCP 端口或本地 Unix 域套接字文件扩展名；
-U username --username username 要连接的用户名称；
-W --password；
（3）通用选项
-e --echo Echo 命令发送到服务器。；
-E --elevel 从 DEBUG，INFO，NOTICE，WARNING，ERROR，LOG，FATAL 和 PANIC 中选择输出消息等级。默认值为 INFO；
--help 显示程序的使用；
--version 显示程序的版本号；

12.5 使用控制文件进行数据加载

1、新建控制文件 lottu.ctl

```
[highgo@pgbulkload ~]$ vi lottu.ctl
INPUT = /home/highgo/tbl_lottu_output.txt
PARSE_BADFILE = /home/highgo/tbl_lottu_bad.txt
LOGFILE = /home/highgo/tbl_lottu_output.log
LIMIT = INFINITE
PARSE_ERRORS = 0
CHECK_CONSTRAINTS = NO
TYPE = CSV
SKIP = 0
DELIMITER = |
QUOTE = "\""
ESCAPE = "\""
OUTPUT = tbl_lottu
MULTI_PROCESS = NO
WRITER = DIRECT
DUPLICATE_BADFILE = /home/highgo/tbl_lottu.dup.csv
DUPLICATE_ERRORS = 0
ON_DUPLICATE_KEEP = NEW
TRUNCATE = YES
```

2、使用控制文件加载

```
[highgo@pgbulkload ~]$ db_bulkload  /home/highgo/lottu.ctl -d highgo -U highgo
```

上述命令解释，其中

/home/highgo/lottu.ctl	控制文件路径
-d highgo	数据库名
-U highgo	数据库用户名

您可以指定以下加载选项。 控制文件可以指定绝对路径或相对路径。 如果通过相对路径指定它，它将相对于执行 pg_bulkload 命令的当前工作目录。 如果不指定控制文件，您应该通过 pg_bulkload 的命令行参数传递必需的选项。

以下参数在控制文件中可用。

TYPE = CSV | BINARY | FIXED |功能 输入数据的类型。默认值为 CSV；

INPUT | INFILE = path | stdin | [schemaname。] function_name（argvalue，...）从中加载数据的源。始终需要；

WRITER | LOADER = DIRECT | BUFFERED | BINARY |平行 加载数据的方法。默认值为 DIRECT；
OUTPUT |“ TABLE = {[schema_name。] table_name | outfile} 将数据加载到的目标。总是需要。根据 WRITER（或 LOADER）选项；
SKIP | OFFSET = n 跳过输入行的数量。默认值为 0.您不能同时指定“TYPE = FUNCTION”和 SKIP。 LIMIT | LOAD = n 要加载的行数。默认值为 INFINITE，即所有数据将被加载。即使使用 TYPE = FUNCTION，此选项也可用；
ENCODING = encoding 指定输入数据的编码。检查指定的编码是否有效，如果需要，将输入数据转换为数据库编码。默认情况下，输入数据的编码既不验证也不转换。如果可以确保输入数据在数据库编码中进行编码，则可以通过不指定此选项以及跳过编码验证和转换来减少加载时间。注意，只有当 INPUT 是 stdin 时，默认情况下才使用 client_encoding 作为输入数据的编码。您不能同时指定“TYPE = FUNCTION”和 ENCODING。 有关有效的编码名称，请参阅内置转换。这里是选项值和实际行为： DB 编码 SQL_ASCII 非 SQL_ASCII ENCODING 未指定既未选中也未转换，未选中或未转换 SQL_ASCII 既未选中也未转换仅选中非 SQL_ASCII，与仅作为 DB 选中的选项相同 非 SQL_ASCII，不同于 DB 只检查并转换 ；
FILTER = [schema_name。] function_name [（argtype，...）] 指定过滤器函数以转换输入文件中的每一行。您可以省略 argtype 的定义，只要函数名在数据库中是唯一的。如果未指定，则输入数据将直接解析为加载目标表。参见如何编写 FILTER 函数来创建 FILTER 函数。 您不能同时指定“TYPE = FUNCTION”和 FILTER。此外，CSV 选项中 FORCE_NOT_NULL 不能与 FILTER 选项一起使用；
CHECK_CONSTRAINTS = YES |没有 指定在加载期间是否检查 CHECK 约束。默认值为 NO。您不能同时指定“WRITER = BINARY”和 CHECK_CONSTRAINTS；
PARSE_ERRORS = n 在解析，编码检查，编码转换，FILTER 函数，CHECK 约束检查，NOT NULL 检查或数据类型转换期间抛出错误的导入元组数。无效的输入元组未加载并记录在 PARSE BADFILE 中。默认值为 0.如果有等于或大于该值的解析错误，则已提交已加载的数据，并且不加载剩余的元组。 0 表示不允许错误，-1 和 INFINITE 表示忽略所有错误；
DUPLICATE_ERRORS = n 违反唯一约束的导入元组数。冲突的元组从表中删除并记录在 DUPLICATE BADFILE 中。默认值为 0.如果存在等于或大于该值的唯一违例，则回滚整个负载。 0 表示不允许违反，-1 和 INFINITE 表示忽略所有违规。您不能同时指定“WRITER = BINARY”和 DUPLICATE_ERRORS；

ON_DUPLICATE_KEEP = NEW |旧 指定如何处理违反唯一约束的元组。删除的元组记录在 BAD 文件中。默认值为 NEW。如果启用该选项，还需要将 DUPLICATE_ERRORS 设置为大于 0。您不能同时指定“WRITER = BINARY”和 ON_DUPLICATE_KEEP；
LOGFILE = path 写结果日志的路径。如果由相对路径指定，则将其视为与 INPUT 相同。默认值为$ PGDATA / pg_bulkload / <timestamp> _ <dbname> _ <schema> _ <table> .log；
PARSE_BADFILE = path BAD 文件的路径记录在解析，编码检查，编码转换，FILTER 函数，CHECK 约束检查，NOT NULL 检查或数据类型转换期间导致错误的无效记录。文件的格式与输入源文件相同。如果由相对路径指定，则将其视为与 INPUT 相同。默认为$ PGDATA / pg_bulkload / <timestamp> _ <dbname> _ <schema> _ <table> .bad。<extension-of-infile>；
DUPLICATE_BADFILE = path 在索引重建期间写入与唯一约束冲突的坏记录的路径。文件的格式始终为 CSV。如果由相对路径指定，则将其视为与 INPUT 相同。默认值为$ PGDATA / pg_bulkload / <timestamp> _ <dbname> _ <schema> _ <table> .dup.csv。您不能同时指定“WRITER = BINARY”和 DUPLICATE_BADFILE；
TRUNCATE = YES |没有 如果为 YES，使用 TRUNCATE 命令从目标表中删除所有行。如果否，不做任何事。默认值为 NO。您不能同时指定“WRITER = BINARY”和 TRUNCATE；
VERBOSE = YES |没有 如果是，在服务器日志中写错误的元组。如果否，不要在 serverlog 中写它们。默认值为 NO。
MULTI_PROCESS = YES |没有 如果是，我们使用多线程并行地进行数据读取，解析和写入。如果否，我们只使用单线程，而不是做并行处理。默认值为NO。如果WRITER为PARALLEL，则忽略 MULTI_PROCESS。如果将密码认证配置为要加载的数据库，则必须设置密码文件。

13 孤儿事务清理工具

13.1 孤儿事务清理工具简介

孤儿事务处理工具是对数据库管理系统中长期未完成的 prepared transaction 进行处理的工具。孤儿事务经常发生在两阶段提交时，客户机崩溃，或者服务器崩溃导致客户机连接被终止而无法重新连接的场景中。孤儿事务往往会占用关键系统资源或使事务 ID 保持活跃状态。孤儿事务清理工具可以自定义时长，查询系统中的孤儿事务时超过此时间便会认为 prepared transaction 是孤儿事务。孤儿事务处理工具可以根据查询出的孤儿事务 id 来对这些孤儿事务进行提交或回滚以达到清理的目的。

13.2 使用孤儿事务清理工具

1、获取孤儿事务清理工具脚本 orphaned_pt.sh。

2、修改孤儿事务清理工具脚本权限。

```
chmod +x orphaned_pt.sh
```

3、获取脚本帮助信息。

```
./orphaned_pt.sh -h
```

```
--------------------------------------------------------------------------------

usage: ./orphaned_pt.sh OPTIONS

OPTIONS can be:
    -h      Print this help message
    -v      Verbose; print information; version, SQL and variables

    -c      Commit a prepared transaction identified by gid
            sets variable [commit_gid]
    -l      List all prepared transactions
    -t      Sets the timeout after which prepared transactions are
            considered orphaned.
            - default value is '30min'
            - sets variable [timeout_interval]
    -o      Options for "psql" command
            - sets variable [psql_options]
    -p      Path for a psql binary, if not given, psql in path will be used
            - sets variable [psql_path]
    -r      Rollback a prepared transaction identified by gid
            - (ignores -m)
            - sets variable [rollback_gid]

--------------------------------------------------------------------------------
```

4、查询数据库中的孤儿事务。

./orphaned_pt.sh -p "/usr/local/pgsql/bin/psql" -o "-U postgres -p 5432" -t 1 -lv

其中：

-p 选项指定 psql 程序的路径。

-o 选项指定 psql 连接数据库的选项和参数。

-t 选项指定 prepared transaction 超过多长时间未完成即被视为孤儿事务，单位：分钟，默认 30 分钟。

-l 选项为列出所有孤儿事务

-v 选项为启用调试模式

```
[root@localhost ~]# ./orphaned_pt.sh -p "/usr/local/pgsql/bin/psql" -o "-U postgres -p 5432" -t 1 -lv

================================================================================
DEBUG INFORMATION
================================================================================
[DEBUG] [variable] timeout_interval = '1'
[DEBUG] [variable] rollback_gid =
[DEBUG] [variable] commit_gid =
[DEBUG] [variable] print_info = false
[DEBUG] [variable] psql_path = /usr/local/pgsql/bin/psql
[DEBUG] [variable] psql_options = -U postgres -p 5432

================================================================================
PREPARED TRANSACTION LIST
================================================================================
[DEBUG] [SQL Query] SELECT gid, prepared FROM pg_prepared_xacts WHERE prepared + CAST('1' AS INTERVAL) < NOW();
    gid    |           prepared
-----------+-------------------------------
 insert_1m | 2021-12-30 09:19:23.567324+08
(1 row)

[STATUS] Script completed without errors.
[root@localhost ~]#
```

5、对孤儿事务进行提交或回滚

```
./orphaned_pt.sh -p "/usr/local/pgsql/bin/psql" -o "-U postgres -p 5432" -c "gid"
或
./orphaned_pt.sh -p "/usr/local/pgsql/bin/psql" -o "-U postgres -p 5432" -r "gid"
```

```
[root@localhost ~]# ./orphaned_pt.sh -p "/usr/local/pgsql/bin/psql" -o "-U postgres -p 5432" -c "insert_1m"

================================================================================
COMMIT PREPARED TRANSACTION
================================================================================
COMMIT PREPARED
[STATUS] Script completed without errors.
```

14 程序卸载

14.1 RPM 包卸载

执行以下命令卸载 rpm 包:

```
[root@x86-0001 opt]#  rpm -qa | grep 7.0.1
hgdb-7.0.1-1.el7.x86_64
[root@x86-0001 hgdb]#  rpm -e hgdb-7.0.1-1.el7.x86_64
```