**关于postgresql-XL集群的入门配置**

2018年01月03日 14:42:48

阅读数：329

**一 Postgres-XL简介**

https://www.postgres-xl.org/documentation/tutorial-createcluster.html 这是官网的配置，只是一台机器的1gtm + 2 coord + 2 node 都在一台机器上

Postgres的-XL是一个基于PostgreSQL数据库的横向扩展开源SQL数据库集群，具有足够的灵活性来处理不同的数据库工作负载:

* **完全ACID，保持事务一致性**
* **OLTP 写频繁的业务**
* **需要MPP并行性商业智能/大数据分析**
* **操作数据存储**
* **Key-value 存储**
* **GIS的地理空间**
* **混合业务工作环境**
* **多租户服务提供商托管环境**
* **Web 2.0**

Postgres-XL架构

**二 组件简介**

* **Global Transaction Monitor (GTM)** 全局事务管理器，确保群集范围内的事务一致性。 GTM负责发放事务ID和快照作为其多版本并发控制的一部分。 集群可选地配置一个备用GTM，以改进可用性。此外，可以在协调器间配置代理GTM， 可用于改善可扩展性，减少GTM的通信量。
* **GTM Standby** GTM的备节点，在pgxc,pgxl中，GTM控制所有的全局事务分配，如果出现问题，就会导致整个集群不可用，为了增加可用性，增加该备用节点。当GTM出现问题时，GTM Standby可以升级为GTM，保证集群正常工作。
* **GTM-Proxy** GTM需要与所有的Coordinators通信，为了降低压力，可以在每个Coordinator机器上部署一个GTM-Proxy。
* **Coordinator** 协调员管理用户会话，并与GTM和数据节点进行交互。协调员解析，并计划查询，并给语句中的每一个组件发送下一个序列化的全局性计划。 为节省机器，通常此服务和数据节点部署在一起。
* **Data Node** 数据节点是数据实际存储的地方。数据的分布可以由DBA来配置。为了提高可用性，可以配置数据节点的热备以便进行故障转移准备。

总结：gtm是负责ACID的，保证分布式数据库全局事务一致性。得益于此，就算数据节点是分布的，但是你在主节点操作增删改查事务时，就如同只操作一个数据库一样简单。Coordinator是调度的，将操作指令发送到各个数据节点。datanodes是数据节点，分布式存储数据。

更多介绍参考：[《Postgres-XL：基于PostgreSQL的开源分布式实现》](https://link.jianshu.com/?t=https://www.biaodianfu.com/postgres-xl.html)

**三 Postgres-XL环境配置与安装**

1. **3.1 集群规划**

准备三台Centos7服务器（或者虚拟机），集群规划如下：

| **主机名** | **IP** | **角色** | **端口** | **nodename** | **数据目录** |
| --- | --- | --- | --- | --- | --- |
| gtm | 192.168.0.125 | GTM | 6666 | gtm | /nodes/gtm |
|  |  | GTM Slave | 20001 | gtmSlave | /nodes/gtmSlave |
| datanode1 | 192.168.0.127 | Coordinator | 5432 | coord1 | /nodes/coord |
|  |  | Datanode | 5433 | node1 | /nodes/dn\_master |
|  |  | Datanode Slave | 15433 | node1\_slave | /nodes/dn\_slave |
|  |  | GTM Proxy | 6666 | gtm\_pxy1 | /nodes/gtm\_pxy |
| datanode2 | 192.168.0.128 | Coordinator | 5432 | coord2 | /nodes/coord |
|  |  | Datanode | 5433 | node2 | nodes/dn\_master |
|  |  | Datanode Slave | 15433 | node2\_slave | /nodes/dn\_slave |
|  |  | GTM Proxy | 6666 | gtm\_pxy2 | /nodes/gtm\_pxy |

在每台机器的 /etc/hosts中加入以下内容：

192.168.0.125 gtm

192.168.0.126 datanode1

192.168.0.127 datanode2

gtm上部署gtm，gtm\_sandby测试环境暂未部署。  
Coordinator与Datanode节点一般部署在同一台机器上。实际上，GTM-proxy,Coordinator与Datanode节点一般都在同一个机器上，使用时避免端口号与连接池端口号重叠！规划datanode1,datanode2作为协调节点与数据节点。

1. **3.2 系统环境设置**

以下操作，对每个服务器节点都适用。  
关闭防火墙：

[root@localhost ~]# systemctl stop firewalld.service

[root@localhost ~]# systemctl disable firewalld.service

selinux设置:

[root@localhost ~]#vim /etc/selinux/config

设置SELINUX=disabled，保存退出。

# This file controls the state of SELinux on the system.

# SELINUX= can take one of these three values:

# enforcing - SELinux security policy is enforced.

# permissive - SELinux prints warnings instead of enforcing.

# disabled - No SELinux policy is loaded.

SELINUX=disabled

# SELINUXTYPE= can take one of three two values:

# targeted - Targeted processes are protected,

# minimum - Modification of targeted policy. Only selected processes are protected.

# mls - Multi Level Security protection.

安装依赖包：

[root@localhost ~]# yum install -y flex bison readline-devel zlib-devel openjade docbook-style-dsssl

重启服务器！一定要重启！

1. **3.3 新建用户**

每个节点都建立用户postgres，并且建立.ssh目录，并配置相应的权限：

[root@localhost ~]# useradd postgres

[root@localhost ~]# passwd postgres

[root@localhost ~]# su - postgres

[root@localhost ~]# mkdir ~/.ssh

[root@localhost ~]# chmod 700 ~/.ssh

1. **3.4 ssh免密码登录**

仅仅在gtm节点配置如下操作：

[root@localhost ~]# su - postgres

[postgres@localhost ~]# ssh-keygen -t rsa

[postgres@localhost ~]# cat ~/.ssh/id\_rsa.pub >> ~/.ssh/authorized\_keys

[postgres@localhost ~]# chmod 600 ~/.ssh/authorized\_keys

将刚生成的认证文件拷贝到datanode1到datanode2中，使得gtm节点可以免密码登录datanode1~datanode2的任意一个节点：

[postgres@localhost ~]# scp ~/.ssh/authorized\_keys postgres@datanode1:~/.ssh/

[postgres@localhost ~]# scp ~/.ssh/authorized\_keys postgres@datanode2:~/.ssh/

对所有提示都不要输入，直接enter下一步。直到最后，因为第一次要求输入目标机器的用户密码，输入即可。

1. **3.5 Postgres-XL安装**

pg1-pg3每个节点都需安装配置。切换回root用户下，执行如下步骤安装

[root@localhost ~]# cd /opt

[root@localhost ~]# git clone git://git.postgresql.org/git/postgres-xl.git

[root@localhost ~]# cd postgres-xl

[root@localhost ~postgres-xl]# ./configure --prefix=/home/postgres/pgxl/

[root@localhost ~postgres-xl]# make

[root@localhost ~postgres-xl]# make install

[root@localhost ~postgres-xl]# cd contrib/

[root@localhost ~contrib]# make

[root@localhost ~contrib]# make install

[root@localhost ~contrib]#chown -R postgres.postgres /home/postgres/pgxl

cortrib中有很多postgres很牛的工具，一般要装上。如ltree,uuid,postgres\_fdw等等。

1. **3.6 配置环境变量**

进入postgres用户，修改其环境变量，开始编辑

[root@localhost ~]#su - postgres

[postgres@localhost ~]#vi .bashrc

在打开的文件末尾，新增如下变量配置：

export PGHOME=/home/postgres/pgxl

export LD\_LIBRARY\_PATH=$PGHOME/lib:$LD\_LIBRARY\_PATH

export PATH=$PGHOME/bin:$PATH

按住esc，然后输入:wq!保存退出。输入以下命令对更改重启生效。

[root@localhost ~]# source .bashrc

#输入以下语句，如果输出变量结果，代表生效

[root@localhost ~]# echo $PGHOME

#应该输出/home/postgres/pgxl代表生效

如上操作，除特别强调以外，是datanode1-datanode2节点都要配置安装的。

**四 集群配置**

1. **4.1 生成pgxc\_ctl配置文件**

[postgres@localhost ~]# pgxc\_ctl （注意，这里只有GTM配置即可）

PGXC prepare config empty

---执行该命令将会生成一份配置文件模板

PGXC ---按ctrl c退出。

1. **4.2 配置pgxc\_ctl.conf （也是只用配置GTM）**

在pgxc\_ctl文件夹中存在一个pgxc\_ctl.conf文件，编辑如下：

pgxcInstallDir=$PGHOME

pgxlDATA=$PGHOME/data

pgxcOwner=postgres

#---- GTM Master -----------------------------------------

gtmName=gtm

gtmMasterServer=gtm

gtmMasterPort=6666

gtmMasterDir=$pgxlDATA/nodes/gtm

gtmSlave=y # Specify y if you configure GTM Slave. Otherwise, GTM slave will not be configured and

# all the following variables will be reset.

gtmSlaveName=gtmSlave

gtmSlaveServer=gtm # value none means GTM slave is not available. Give none if you don't configure GTM Slave.

gtmSlavePort=20001 # Not used if you don't configure GTM slave.

gtmSlaveDir=$pgxlDATA/nodes/gtmSlave # Not used if you don't configure GTM slave.

#---- GTM-Proxy Master -------

gtmProxyDir=$pgxlDATA/nodes/gtm\_proxy

gtmProxy=y

gtmProxyNames=(gtm\_pxy1 gtm\_pxy2)

gtmProxyServers=(datanode1 datanode2)

gtmProxyPorts=(6666 6666)

gtmProxyDirs=($gtmProxyDir $gtmProxyDir)

#---- Coordinators ---------

coordMasterDir=$pgxlDATA/nodes/coord

coordNames=(coord1 coord2)

coordPorts=(5432 5432)

poolerPorts=(6667 6667)

coordPgHbaEntries=(0.0.0.0/0)

coordMasterServers=(datanode1 datanode2)

coordMasterDirs=($coordMasterDir $coordMasterDir)

coordMaxWALsender=0 #没设置备份节点，设置为0

coordMaxWALSenders=($coordMaxWALsender $coordMaxWALsender) #数量保持和coordMasterServers一致

coordSlave=n

#---- Datanodes ----------

datanodeMasterDir=$pgxlDATA/nodes/dn\_master

primaryDatanode=node1 # 主数据节点

datanodeNames=(node1 node2)

datanodePorts=(5433 5433)

datanodePoolerPorts=(6668 6668)

datanodePgHbaEntries=(0.0.0.0/0)

datanodeMasterServers=(datanode1 datanode2)

datanodeMasterDirs=($datanodeMasterDir $datanodeMasterDir)

datanodeMaxWalSender=4

datanodeMaxWALSenders=($datanodeMaxWalSender $datanodeMaxWalSender)

datanodeSlave=n

#将datanode1节点的slave做到了datanode2服务器上，交叉做了备份

datanodeSlaveServers=(datanode2 datanode1) # value none means this slave is not available

datanodeSlavePorts=(15433 15433) # value none means this slave is not available

datanodeSlavePoolerPorts=(20012 20012) # value none means this slave is not available

datanodeSlaveSync=y # If datanode slave is connected in synchronized mode

datanodeSlaveDirs=($datanodeSlaveDir $datanodeSlaveDir)

datanodeArchLogDirs=( $datanodeArchLogDir $datanodeArchLogDir)

如上配置，都没有配置slave，具体生产环境，请阅读配置文件，自行配置。

1. **4.3 集群初始化，启动，停止 （在gtm机器中启动即可）**

第一次启动集群，需要初始化，初始化如下：

[postgres@pg1 pgxc\_ctl]$ pgxc\_ctl -c /home/postgres/pgxc\_ctl/pgxc\_ctl.conf init all

初始化后会直接启动集群。

ALTER NODE coord1 WITH (HOST='datanode1', PORT=5432);

ALTER NODE

CREATE NODE coord2 WITH (TYPE='coordinator', HOST='datanode2', PORT=5432);

CREATE NODE

CREATE NODE datanode1 WITH (TYPE='datanode', HOST='datanode1', PORT=5433, PRIMARY, PREFERRED);

CREATE NODE

CREATE NODE datanode2 WITH (TYPE='datanode', HOST='datanode2', PORT=5433);

CREATE NODE

SELECT pgxc\_pool\_reload();

pgxc\_pool\_reload

------------------

t

(1 row)

CREATE NODE coord1 WITH (TYPE='coordinator', HOST='datanode1', PORT=5432);

CREATE NODE

ALTER NODE coord2 WITH (HOST='datanode2', PORT=5432);

ALTER NODE

CREATE NODE datanode1 WITH (TYPE='datanode', HOST='datanode1', PORT=5433, PRIMARY);

CREATE NODE

CREATE NODE datanode2 WITH (TYPE='datanode', HOST='datanode2', PORT=5433, PREFERRED);

CREATE NODE

SELECT pgxc\_pool\_reload();

pgxc\_pool\_reload

------------------

t

(1 row)

Done.

EXECUTE DIRECT ON (datanode1) 'CREATE NODE coord1 WITH (TYPE=''coordinator'', HOST=''datanode1'', PORT=5432)';

EXECUTE DIRECT

EXECUTE DIRECT ON (datanode1) 'CREATE NODE coord2 WITH (TYPE=''coordinator'', HOST=''datanode2'', PORT=5432)';

EXECUTE DIRECT

EXECUTE DIRECT ON (datanode1) 'ALTER NODE datanode1 WITH (TYPE=''datanode'', HOST=''datanode1'', PORT=5433, PRIMARY, PREFERRED)';

EXECUTE DIRECT

EXECUTE DIRECT ON (datanode1) 'CREATE NODE datanode2 WITH (TYPE=''datanode'', HOST=''datanode2'', PORT=5433, PREFERRED)';

EXECUTE DIRECT

EXECUTE DIRECT ON (datanode1) 'SELECT pgxc\_pool\_reload()';

pgxc\_pool\_reload

------------------

t

(1 row)

EXECUTE DIRECT ON (datanode2) 'CREATE NODE coord1 WITH (TYPE=''coordinator'', HOST=''datanode1'', PORT=5432)';

EXECUTE DIRECT

EXECUTE DIRECT ON (datanode2) 'CREATE NODE coord2 WITH (TYPE=''coordinator'', HOST=''datanode2'', PORT=5432)';

EXECUTE DIRECT

EXECUTE DIRECT ON (datanode2) 'CREATE NODE datanode1 WITH (TYPE=''datanode'', HOST=''datanode1'', PORT=5433, PRIMARY, PREFERRED)';

EXECUTE DIRECT

EXECUTE DIRECT ON (datanode2) 'ALTER NODE datanode2 WITH (TYPE=''datanode'', HOST=''datanode2'', PORT=5433, PREFERRED)';

EXECUTE DIRECT

EXECUTE DIRECT ON (datanode2) 'SELECT pgxc\_pool\_reload()';

pgxc\_pool\_reload

------------------

t

(1 row)

Done.

以后启动，直接执行如下命令：

[postgres@pg1 pgxc\_ctl]$ pgxc\_ctl -c /home/postgres/pgxc\_ctl/pgxc\_ctl.conf start all

停止集群如下：

[postgres@pg1 pgxc\_ctl]$ pgxc\_ctl -c /home/postgres/pgxc\_ctl/pgxc\_ctl.conf stop all

这几个主要命令暂时这么多，更多请从pgxc\_ctl --help中获取更多信息。

**五 Postgres-XL集群测试**

1. **5.1 插入数据**

在datanode1节点，执行psql -p 5432进入数据库操作。

[postgres@localhost]$ psql -p 5432

psql (PGXL 10alpha1, based on PG 10beta1 (Postgres-XL 10alpha1))

Type "help" for help.

postgres=# select \* from pgxc\_node;

node\_name | node\_type | node\_port | node\_host | nodeis\_primary | nodeis\_preferred | node\_id

-----------+-----------+-----------+-----------+----------------+------------------+-------------

coord1 | C | 5432 | datanode1 | f | f | 1885696643

coord2 | C | 5432 | datanode2 | f | f | -1197102633

datanode1 | D | 5433 | datanode1 | t | t | 888802358

datanode2 | D | 5433 | datanode2 | f | f | -905831925

(4 rows)

postgres=# create table test1(id it,name text);

postgres=# insert into test1(id,name) select generate\_series(1,8),'测试';

1. **5.2 查看数据分布**

在datanode1节点上查看数据

[postgres@bogon ~]$ psql -p 5433

psql (PGXL 10alpha1, based on PG 10beta1 (Postgres-XL 10alpha1))

Type "help" for help.

postgres=# select \* from test1;

id | name

----+------

1 | 测试

2 | 测试

5 | 测试

6 | 测试

8 | 测试

(5 rows)

在datanode2节点上查看数据

postgres=# select \* from test1;

id | name

----+------

3 | 测试

4 | 测试

7 | 测试

(3 rows)

注意：由于所有的数据节点组成了完整的数据视图，所以一个数据节点down机，整个pgxl都启动不了了，所以实际生产中，为了提高可用性，一定要配置数据节点的热备以便进行故障转移准备。

**六 集群应用与管理**

1. **6.1 建表说明**

* REPLICATION表：各个datanode节点中，表的数据完全相同，也就是说，插入数据时，会分别在每个datanode节点插入相同数据。读数据时，只需要读任意一个datanode节点上的数据。  
  建表语法：

postgres=# CREATE TABLE repltab (col1 int, col2 int) DISTRIBUTE BY REPLICATION;

* DISTRIBUTE ：会将插入的数据，按照拆分规则，分配到不同的datanode节点中存储，也就是sharding技术。每个datanode节点只保存了部分数据，通过coordinate节点可以查询完整的数据视图。

postgres=# CREATE TABLE disttab(col1 int, col2 int, col3 text) DISTRIBUTE BY HASH(col1);

模拟部分数据，插入测试数据：

#任意登录一个coordinate节点进行建表操作

[postgres@gtm ~]$ psql -h datanode1 -p 5432 -U postgres

postgres=# INSERT INTO disttab SELECT generate\_series(1,100), generate\_series(101, 200), 'foo';

INSERT 0 100

postgres=# INSERT INTO repltab SELECT generate\_series(1,100), generate\_series(101, 200);

INSERT 0 100

查看数据分布结果：

#DISTRIBUTE表分布结果

postgres=# SELECT xc\_node\_id, count(\*) FROM disttab GROUP BY xc\_node\_id;

xc\_node\_id | count

------------+-------

1148549230 | 42

-927910690 | 58

(2 rows)

#REPLICATION表分布结果

postgres=# SELECT count(\*) FROM repltab;

count

-------

100

(1 row)

查看另一个datanode2中repltab表结果

[postgres@datanode2 pgxl9.5]$ psql -p 5433

psql (PGXL 9.5r1.3, based on PG 9.5.4 (Postgres-XL 9.5r1.3))

Type "help" for help.

postgres=# SELECT count(\*) FROM repltab;

count

-------

100

(1 row)

结论：REPLICATION表中，datanode1,datanode2中表是全部数据，一模一样。而DISTRIBUTE表，数据散落近乎平均分配到了datanode1,datanode2节点中。

1. **6.2新增datanode节点与数据重分布**
2. **6.2.1 新增datanode节点**

在gtm集群管理节点上执行pgxc\_ctl命令

[postgres@gtm ~]$ pgxc\_ctl

/bin/bash

Installing pgxc\_ctl\_bash script as /home/postgres/pgxc\_ctl/pgxc\_ctl\_bash.

Installing pgxc\_ctl\_bash script as /home/postgres/pgxc\_ctl/pgxc\_ctl\_bash.

Reading configuration using /home/postgres/pgxc\_ctl/pgxc\_ctl\_bash --home /home/postgres/pgxc\_ctl --configuration /home/postgres/pgxc\_ctl/pgxc\_ctl.conf

Finished reading configuration.

\*\*\*\*\*\*\*\* PGXC\_CTL START \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Current directory: /home/postgres/pgxc\_ctl

PGXC

在PGXC后面执行新增数据节点命令:

Current directory: /home/postgres/pgxc\_ctl

# 在服务器datanode1上，新增一个master角色的datanode节点，名称是dn3

# 端口号暂定5430，pool master暂定6669 ，指定好数据目录位置，从两个节点升级到3个节点，之后要写3个none

# none应该是datanodeSpecificExtraConfig或者datanodeSpecificExtraPgHba配置

**PGXC add datanode slave datanode2 datanode1 15432 6670 /home/postgres/pgxc/nodes none none none**

等待新增完成后，查询集群节点状态：

[postgres@gtm ~]$ psql -h datanode1 -p 5432 -U postgres

psql (PGXL 9.5r1.3, based on PG 9.5.4 (Postgres-XL 9.5r1.3))

Type "help" for help.

postgres=# select \* from pgxc\_node;

node\_name | node\_type | node\_port | node\_host | nodeis\_primary | nodeis\_preferred | node\_id

-----------+-----------+-----------+-----------+----------------+------------------+-------------

coord1 | C | 5432 | datanode1 | f | f | 1885696643

coord2 | C | 5432 | datanode2 | f | f | -1197102633

node1 | D | 5433 | datanode1 | f | t | 1148549230

node2 | D | 5433 | datanode2 | f | f | -927910690

dn3 | D | 5430 | datanode1 | f | f | -700122826

(5 rows)

可以发现节点新增完毕。

1. **6.2.2 数据重分布**

之前我们的DISTRIBUTE表分布在了node1,node2节点上，如下：

postgres=# SELECT xc\_node\_id, count(\*) FROM disttab GROUP BY xc\_node\_id;

xc\_node\_id | count

------------+-------

1148549230 | 42

-927910690 | 58

(2 rows)

新增一个节点后，将sharding表数据重新分配到三个节点上，将repl表复制到新节点：

# 重分布sharding表

postgres=# ALTER TABLE disttab ADD NODE (dn3);

ALTER TABLE

# 复制数据到新节点

postgres=# ALTER TABLE repltab ADD NODE (dn3);

ALTER TABLE

查看新的数据分布：

postgres=# SELECT xc\_node\_id, count(\*) FROM disttab GROUP BY xc\_node\_id;

xc\_node\_id | count

------------+-------

-700122826 | 36

-927910690 | 32

1148549230 | 32

(3 rows)

登录dn3(新增的时候，放在了datanode1服务器上，端口5430)节点查看数据：

[postgres@gtm ~]$ psql -h datanode1 -p 5430 -U postgres

psql (PGXL 9.5r1.3, based on PG 9.5.4 (Postgres-XL 9.5r1.3))

Type "help" for help.

postgres=# select count(\*) from repltab;

count

-------

100

(1 row)

很明显,通过 ALTER TABLE tt ADD NODE (dn)命令，可以将DISTRIBUTE表数据重新分布到新节点，重分布过程中会中断所有事务。可以将REPLICATION表数据复制到新节点。

1. **6.2.3 从datanode节点中回收数据**

postgres=# ALTER TABLE disttab DELETE NODE (dn3);

ALTER TABLE

postgres=# ALTER TABLE repltab DELETE NODE (dn3);

ALTER TABLE

1. **6.3 删除数据节点**

Postgresql-XL并没有检查将被删除的datanode节点是否有replicated/distributed表的数据，为了数据安全，在删除之前需要检查下被删除节点上的数据，有数据的话，要回收掉分配到其他节点，然后才能安全删除。删除数据节点分为四步骤：

* 查询要删除节点dn3的oid

postgres=# SELECT oid, \* FROM pgxc\_node;

oid | node\_name | node\_type | node\_port | node\_host | nodeis\_primary | nodeis\_preferred | node\_id

-------+-----------+-----------+-----------+-----------+----------------+------------------+-------------

11819 | coord1 | C | 5432 | datanode1 | f | f | 1885696643

16384 | coord2 | C | 5432 | datanode2 | f | f | -1197102633

16385 | node1 | D | 5433 | datanode1 | f | t | 1148549230

16386 | node2 | D | 5433 | datanode2 | f | f | -927910690

16397 | dn3 | D | 5430 | datanode1 | f | f | -700122826

(5 rows)

* 查询dn3对应的oid中是否有数据

testdb=# SELECT \* FROM pgxc\_class WHERE nodeoids::integer[] @> ARRAY[16397];

pcrelid | pclocatortype | pcattnum | pchashalgorithm | pchashbuckets | nodeoids

---------+---------------+----------+-----------------+---------------+-------------------

16388 | H | 1 | 1 | 4096 | 16397 16385 16386

16394 | R | 0 | 0 | 0 | 16397 16385 16386

(2 rows)

* 有数据的先回收数据

postgres=# ALTER TABLE disttab DELETE NODE (dn3);

ALTER TABLE

postgres=# ALTER TABLE repltab DELETE NODE (dn3);

ALTER TABLE

postgres=# SELECT \* FROM pgxc\_class WHERE nodeoids::integer[] @> ARRAY[16397];

pcrelid | pclocatortype | pcattnum | pchashalgorithm | pchashbuckets | nodeoids

---------+---------------+----------+-----------------+---------------+----------

(0 rows)

* 安全删除dn3

PGXC$ remove datanode master dn3 clean

1. **6.4 coordinate节点管理**

同datanode节点相似，列出语句不做测试了：

# 新增coordinate

PGXC$ add coordinator master coord3 localhost 30003 30013 $dataDirRoot/coord\_master.3 none none none

# 删除coordinate,clean选项可以将相应的数据目录也删除

PGXC$ remove coordinator master coord2 clean

1. **6.5 故障切换**

* 查看当前数据集群

postgres=# SELECT oid, \* FROM pgxc\_node;

oid | node\_name | node\_type | node\_port | node\_host | nodeis\_primary | nodeis\_preferred | node\_id

-------+-----------+-----------+-----------+-----------+----------------+------------------+-------------

11819 | coord1 | C | 5432 | datanode1 | f | f | 1885696643

16384 | coord2 | C | 5432 | datanode2 | f | f | -1197102633

16385 | node1 | D | 5433 | datanode1 | f | t | 1148549230

16386 | node2 | D | 5433 | datanode2 | f | f | -927910690

(4 rows)

* 模拟node1节点故障

PGXC$ stop -m immediate datanode master node1

Stopping datanode master node1.

Done.

* 测试集群查询

postgres=# SELECT xc\_node\_id, count(\*) FROM disttab GROUP BY xc\_node\_id;

ERROR: Failed to get pooled connections

postgres=# SELECT xc\_node\_id, \* FROM disttab WHERE col1 = 3;

xc\_node\_id | col1 | col2 | col3

------------+------+------+------

-927910690 | 3 | 103 | foo

(1 row)

测试发现，查询范围如果涉及到故障的node1节点，会报错，而查询的数据范围不在node1上的话，仍然可以查询。

* 手动切换node1的slave

PGXC$ failover datanode node1

# 切换完成后，查询集群

postgres=# SELECT oid, \* FROM pgxc\_node;

oid | node\_name | node\_type | node\_port | node\_host | nodeis\_primary | nodeis\_preferred | node\_id

-------+-----------+-----------+-----------+-----------+----------------+------------------+-------------

11819 | coord1 | C | 5432 | datanode1 | f | f | 1885696643

16384 | coord2 | C | 5432 | datanode2 | f | f | -1197102633

16386 | node2 | D | 5433 | datanode2 | f | f | -927910690

16385 | node1 | D | 15433 | datanode2 | f | t | 1148549230

(4 rows)

发现node1节点的ip和端口都已经替换为配置的slave了。

**七 部署遇到的问题**

在配置的时候一定要细心，避免端口号之类的配置冲突等错误。  
错误一：

postgres=# create table test1(id integer,name varchar(20));

LOG: failed to connect to node, connection string (host=192.168.0.125 port=1925 dbname=postgres user=postgres application\_name=pgxc sslmode=disable options='-c remotetype=coordinator -c parentnode=coord1 -c DateStyle=iso,mdy -c timezone=prc -c geqo=on -c intervalstyle=postgres -c lc\_monetary=C'), connection error (fe\_sendauth: no password supplied

)

WARNING: can not connect to node 16384

WARNING: Health map updated to reflect DOWN node (16384)

LOG: Pooler could not open a connection to node 16384

LOG: failed to acquire connections

STATEMENT: create table test1(id integer,name varchar(20));

ERROR: Failed to get pooled connections

HINT: This may happen because one or more nodes are currently unreachable, either because of node or network failure.

Its also possible that the target node may have hit the connection limit or the pooler is configured with low connections.

Please check if all nodes are running fine and also review max\_connections and max\_pool\_size configuration parameters

STATEMENT: create table test1(id integer,name varchar(20));

原因：这个是由于某些环境或配置出了问题，我的就是pg\_hba.conf配置出了问题，Ipv4要改成 0:0:0:0/0 trust才行。  
但这仅仅是一个问题，开发者搭建环境遇到这个错误，一定要检查如下:

* \*\* 各个机器的防火墙是否关闭？\*\*
* \*\* 各个机器的SELINUX状态是否是disabled？\*\*
* \*\* 各个机器的ssh免密登录是否成功？\*\*
* \*\* 各个节点的pg\_hba.conf,postgresql.conf是否配置为信任登录？是否有IP限制？\*\*
* **超过某些节点的最大连接数？（对于我们测试环境来说，肯定不会是这个问题）**

作者搭建pgxl是为地理大数据做技术预研的，使用postgis作为空间数据，欢迎postgis开发者参与交流。